Vue.js в действии

Эрик Хэнчетт,
Бенджамин Листуон

MANNING

Vue.js in Action

ERIK HANCHETT
WITH BEN LISTWON

MANNING
SHELTER ISLAND

Эрик Хэнчетт, Бенджамин Листуон

Vue.js в действии

ПИТЕР® Санкт-Петербург • Москва • Екатеринбург • Воронеж
Нижний Новгород • Ростов-на-Дону • Самара • Минск

2019

ББК 32.988-02-018
УДК 004.738.5
Х88

Хэнчетт Эрик, Листуон Бенджамин

Х88 Vue.js в действии. — СПб.: Питер, 2019. — 304 с.: ил. — (Серия «Библиотека программиста»).

ISBN 978-5-4461-1098-8

Vue.js — это популярная библиотека для создания пользовательских интерфейсов. В ней значительно переосмыслены реактивные идеи, впервые появившиеся в React.js.

Книга «Vue.js в действии» рассказывает о создании быстрых и эластичных пользовательских интерфейсов для Интернета. Освоив ее, вы напишете полноценное приложение для интернет-магазина, где будут присутствовать списки товаров, админка, а также организован полноценный процесс онлайнового заказа.

16+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 32.988-02-018
УДК 004.738.5

ISBN 978-1617294624 англ.
ISBN 978-5-4461-1098-8

Краткое содержание

Оглавление

Часть III. Моделирование данных, работа с API и тестирование

Приложения

Вступление

Клиентская веб-разработка стала на удивление сложной. Если вы никогда не имели дела с современными фреймворками для JavaScript, то можете потратить целую неделю на создание своей первой программы, которая всего лишь выводит приветствие! Это звучит нелепо, и здесь я с вами соглашусь. Проблема в том, что большинство фреймворков требуют навыков работы с терминалом, углубленного знания JavaScript, владения такими инструментами, как NPM (Node Package Manager – диспетчер пакетов Node), Babel, Webpack, – и часто это еще не полный список.

Однако к Vue все это не относится. Мы называем эту библиотеку прогрессивной, поскольку она масштабируется в *обе* стороны. Если приложение простое, используйте Vue по аналогии с JQuery, вставляя элемент `script`. Но с ростом требований и приобретением вами новых навыков работа с Vue станет все более разносторонней и продуктивной.

У Vue есть еще одна отличительная черта. Этот проект разрабатывается не только программистами, но и дизайнерами, преподавателями и специалистами других гуманитарных профессий. Благодаря этому наши документация, руководства и инструменты для разработки одни из лучших в мире. Ощущения от *использования* Vue имеют для нас такое же значение, как производительность, надежность и гибкость.

Эрику удалось сделать эту книгу ориентированной на обычных людей. Прежде всего она необыкновенно наглядная. Множество подробных иллюстраций и снимков экрана с примечаниями неразрывно связывают приведенные здесь примеры с реальным процессом разработки. В итоге, чтобы закрепить свои знания, вам придется научиться работать с браузером и инструментами для разработки Vue и, что важнее, овладеть методами отладки при возникновении проблем.

Для читателей, у которых нет существенного опыта клиентской разработки, написания программ на JavaScript или даже просто программирования, Эрик тщательно излагает фундаментальные концепции и объясняет принцип работы и назначение Vue. Добавьте к этому методологию описания новых возможностей в контексте реальных проектов – и вы получите идеальную книгу для относительно неопытных разработчиков, которые начинают знакомство с современными клиентскими фреймворками с Vue и хотят расширить свои знания в этой области.

Крис Фриц, член основной группы
разработки Vue и куратор документации

Предисловие

Эту книгу мне предложили написать в начале 2017 года, после того, как по личным причинам от этой возможности вынужден был отказаться Бенджамин Листуон (Benjamin Listwon). Я как раз недавно получил диплом магистра делового администрирования университета Невады, а с момента публикации моей последней книги *Ember.js Cookbook* (Pact Publishing, 2016) прошел целый год. Я завел на YouTube канал *Program with Erik* («Программируем с Эриком»), и бóльшая часть моего времени уходила на попытки записать как можно более качественные видеоуроки для небольшой, но растущей аудитории. Одновременно с этим я начал серию скринкастов, посвященных Vue.js, и получил положительные отзывы от зрителей. Это подтолкнуло меня к дальнейшему изучению Vue.

Я начал с прослушивания докладов Эвана Ю (Evan You), создателя Vue.js, и знакомства с планами развития этого фреймворка. Затем перешел к размещенным на YouTube бесчисленным руководствам и видеоурокам от других разработчиков. Начал заглядывать на форумы и в группы Facebook, чтобы узнать, о чем сейчас говорят. Куда бы я ни зашел, люди везде выражали свой восторг по поводу Vue.js и его будущего. Поэтому я и начал размышлять о возможности написания данной книги.

После долгих раздумий и разговора с женой я решил согласиться. К счастью, Бенджамин уже заложил отличный фундамент, поэтому я смог сразу приступить к работе. Следующие десять месяцев состояли из бесчисленных ночей и выходных, потраченных на исследования, тестирование и написание текста.

Не стану приукрашивать: книга далась мне нелегко. Скажем так, не все пошло по плану. Вдобавок к личным проблемам, пропущенным срокам и творческому кризису пришлось вносить существенные изменения после выхода новой версии Vue.js.

Но, несмотря на все это, я очень горжусь книгой. Каждая неудача заставляла меня работать с удвоенной энергией. Я решил закончить книгу и сделать ее максимально качественной. Надеюсь, это будет заметно при чтении.

Огромное спасибо читателям, которые купили эту книгу. Я очень надеюсь на то, что она поможет вам с изучением Vue.js. Пожалуйста, напишите мне о своих впечатлениях. Свяжитесь со мной через Twitter (@ErikCH), электронную почту (erik@programwitherik.com) или список рассылки по адресу goo.gl/UmemSS. Еще раз спасибо!

Благодарности

Прежде всего я хотел бы поблагодарить свою жену Сьюзен, потому что без ее помощи эта книга никогда не была бы завершена. Спасибо сыну и дочке — Уайатту и Вивиан. Это ради них я так тяжело работаю. Спасибо всем рецензентам, членам форума Vue.js in Action и тем, кто прислал отзывы, — ваша помощь сделала эту книгу намного лучше. Благодарю также Криса Фрица (Chris Fritz) за прекрасное предисловие. Наконец, я хотел бы выразить безмерную и искреннюю признательность сообществу Vue.js, Эвану Ю и всем, кто внес свой вклад в этот замечательный фреймворк.

Эрик Ханчетт

В первую очередь я хотел бы выразить искреннюю благодарность своей жене Киффен за поддержку и поощрение — не только в то время, когда я участвовал в этом проекте, но и во всех аспектах нашей совместной жизни. Звездой и сердцем нашей семьи является сын Лео: спасибо тебе за неиссякаемые улыбки, объятия и радость. За одобрение, понимание и поддержку от всего сердца хотел бы поблагодарить редакцию издательства Manning. Я по-настоящему признателен Эрику, без которого эта книга никогда не увидела бы свет: желаю тебе всего наилучшего. Наконец, спасибо Эвану Ю и всем тем, кто помогает развитию Vue.js, — вы собрали воедино прекрасный программный продукт и еще более прекрасное сообщество, частью которого я имею честь быть.

Бенджамин Листуон

Выражаем совместную признательность техническому редактору Джею Келкару (Jay Kelkar), а также всем рецензентам, давшим на разных этапах свои отзывы. Среди них Алекс Миллер (Alex Miller), Алексей Галиуллин (Alexey Galiullin), Крис Коппенбаргер (Chris Coppenbarger), Клайв Харбер (Clive Harber), Дарко Божи-

новски (Darko Bozhinovski), Ферит Топцу (Ferit Topcu), Гарро Лиссенберг (Harro Lissenberg), Ян Петер Хервейер (Jan Pieter Herweijer), Джеспер Петерсен (Jesper Petersen), Лаура Стедмэн (Laura Steadman), Марко Летич (Marko Letic), Пауло Нуин (Paulo Nuin), Филиппе Шаррье (Philippe Charriere), Рохит Шарма (Rohit Sharma), Рональд Борман (Ronald Borman), Райан Гарви (Ryan Harvey), Райан Габер (Ryan Huber), Сандер Зегвельд (Sander Zegveld), Убалдо Пескаторе (Ubaldo Pescatore) и Витторио Марино (Vittorio Marino).

Об этой книге

Позвольте мне, прежде чем приступать к программированию на Vue.js, остановиться на нескольких моментах, о которых вы должны знать.

Здесь вы найдете все, что может понадобиться для овладения Vue.js. Цель этой книги — дать вам знания, с помощью которых вы без колебаний присоединитесь к любому проекту, использующему эту библиотеку.

Проводя исследования для этой книги, я неоднократно слышал, что лучший источник информации для изучения Vue.js — официальная документация. Действительно, на официальном сайте есть отличные руководства, и я настоятельно рекомендую вам с ними ознакомиться. Однако они неидеальны и не охватывают все темы, поэтому советую использовать их как дополнительные справочные материалы. Во время написания книги я задался целью не останавливаться на темах, рассмотренных в официальной документации. Я сделал примеры более понятными и наглядными, чтобы вам проще было применять описанные здесь концепции к собственным проектам. Темы, которые показались мне недостаточно важными или просто не вошли в книгу, даны в виде ссылок на официальные руководства.

Этой книгой можно пользоваться по-разному. Прочитав ее с начала до конца, вы получите полноценное представление о Vue.js. Или работайте с ней как со справочником, просматривая информацию о тех функциях, о которых вам хочется знать больше. Оба подхода допустимы.

Позже мы перейдем к созданию приложений на Vue.js с помощью системы сборки. Не волнуйтесь, инструкции по использованию соответствующего инструмента под названием Vue-CLI приведены в приложении A. Vue-CLI помогает создавать более сложные приложения, беря на себя сборку и преобразование исходного кода.

Лейтмотивом этой книги будет разработка на основе Vue.js приложения зоомагазина. В некоторых главах этот пример рассматривается чаще, в других — реже. Это было сделано специально, чтобы вы могли легко усвоить абстрактные концепции без связи с конкретным проектом. Вместе с тем мы не забываем и о читателях, которые предпочитают учиться на реальных примерах.

Целевая аудитория

Эта книга предназначена для всех, кто заинтересован в изучении Vue.js и имеет опыт работы с JavaScript, HTML и CSS. От вас не требуются глубокие знания этой области, но понимание основ, таких как массивы, переменные, циклы и HTML-элементы, не помешает. Что касается CSS, то мы будем задействовать библиотеку Bootstrap 3. При этом, чтобы выполнять упражнения, вам не обязательно о ней что-либо знать — она применяется лишь для визуального оформления.

В начале книги представлен код на языке ECMAScript 2015, известном также как ES6. Желательно ознакомиться с его спецификацией, прежде чем приступать к чтению. По большей части я использую всего несколько возможностей ES6, таких как стрелочные функции и импорт. Я дам знать, когда мы будем переходить на этот язык.

Структура книги

Книга разбита на три части. Часть I знакомит с Vue.js. В главах 1 и 2 вы создадите свое первое приложение, а также узнаете, что такое экземпляр Vue и как он относится к нашему коду.

В части II, а именно в главах 3–9, мы более подробно рассмотрим концепции представления (View) и модели-представления (ViewModel), а также самые «сочные» аспекты Vue.js. Можно сказать, что часть I — всего лишь затравка, а основной курс начинается с части II. Вы узнаете о сложностях, связанных с созданием программ на Vue.js. Мы начнем с изучения реактивной модели, а затем создадим приложение зоомагазина, которое будет использоваться в дальнейшем.

Мы также рассмотрим формы, поля ввода и привязку информации с помощью мощных директив, встроенных в Vue.js, после чего перейдем к условным выражениям и циклам.

Чрезвычайно важны главы 6 и 7. В них вы научитесь разбивать приложение Vue.js на несколько логических частей, используя компоненты, а также познакомитесь с инструментами сборки, которые понадобятся для построения программ.

Помимо прочего, в главе 7 затрагивается тема маршрутизации. До этого для навигации по приложению будем применять простые условные выражения. Благодаря маршрутизации мы сможем правильно переходить между компонентами и передавать информацию между маршрутами.

Глава 8 познакомит вас с гибкой анимацией и переходами, которые выполняются в Vue.js. Эти замечательные возможности, встроенные в язык, заслуживают вашего внимания.

В главе 9 вы научитесь расширять Vue с помощью примесей и пользовательских директив без дублирования кода.

Часть III посвящена моделированию данных, работе с API и тестированию. В главах 10 и 11 мы подробно рассмотрим систему управления состоянием в Vue

под названием Vuex, а затем попробуем пообщаться с серверной стороной. Вы также познакомитесь с Nuxt.js — фреймворком для серверной отрисовки.

Тестирование рассматривается в главе 12. Мы обсудим базовые навыки, необходимые в любой профессиональной среде.

Исходный код

Эта книга содержит множество примеров исходного кода в виде пронумерованных листингов и элементов строк. Чтобы отличить их от обычного текста, используется моноширинный шрифт. Иногда изменения, вносимые на следующем этапе, выделяются жирным шрифтом, например, когда к уже существующей строке кода добавляется что-то новое.

Во многих случаях оригинальный исходный код был переформатирован — я добавил переносы строк и убрал отступы, чтобы сэкономить место. Иногда даже этого оказывалось недостаточно и приходилось вставлять символы продолжения строки (➥). Кроме того, при описании исходного кода часто убирал из него комментарии. Во многих листингах есть примечания, которые выделяют важные моменты.

Исходный код для книги доступен для загрузки на сайте издательства (www.manning.com/books/vue-js-in-action) и в моем репозитории на GitHub (github.com/ErikCH/VuejsInActionCode). Дополнительные инструкции по загрузке исходного кода и настройке среды программирования можно найти в приложении A.

При чтении книги вы заметите, что я часто разбиваю листинги на отдельные файлы. В архиве для каждой главы код имеется как в виде единого файла, так и в виде нескольких модулей, чтобы было легче ориентироваться. Если вы найдете ошибку в коде, не стесняйтесь отправить мне письмо. Я буду поддерживать свой репозиторий на GitHub и комментировать любые обновления в файле README.

Требования к программному обеспечению

Чтобы вам было легче, весь код в этой книге адаптирован к работе в любом современном браузере. Я лично проверил его в Firefox 58, Chrome 65 и Microsoft Edge 15. Не рекомендую запускать представленные здесь примеры в более старых браузерах, так как вы неминуемо столкнетесь с какими-нибудь проблемами. Сама библиотека Vue.js не поддерживает IE8 и более старые версии. Совместимость с ECMAScript 5 обязательна.

В нескольких начальных главах я использую возможности языка ES6. Для выполнения этих примеров вам понадобится современный браузер.

Приложение зоомагазина, которое мы будем разрабатывать в этой книге, может работать в мобильном браузере. Но, поскольку оно для этого не оптимизировалось, советую запускать его на настольном компьютере.

Вам не нужно беспокоиться о выборе операционной системы — главное, чтобы работал браузер. Это фактически единственное требование.

Интернет-ресурсы

Как уже упоминалось, в ходе работы с примерами, представленными в книге, отличным подспорьем послужит официальное руководство Vue.js. Оно находится по адресу vuejs.org/v2/guide/ и постоянно обновляется.

Ведется также список избранных проектов, относящихся к Vue.js, который вы найдете на странице GitHub github.com/vuejs/awesome-vue. Там есть ссылки на подкасты, дополнительные ресурсы, сторонние библиотеки и даже компании, которые работают с Vue.js. Настоятельно рекомендую ознакомиться с ним.

У Vue.js огромное и активно растущее сообщество пользователей. Одно из лучших мест в Сети для общения с другими разработчиками на Vue.js — официальный форум проекта forum.vuejs.org/. Там вы сможете получить помощь и обсудить все, что касается этой библиотеки.

Если вы ищете видеоуроки, загляните на страницу моего канала на YouTube erik.video, который содержит множество материала по Vue.js и JavaScript в целом.

Дополнительная информация

В этой книге я раскрыл большое количество тем. Если вы застряли в каком-то месте или нуждаетесь в помощи, то всегда можете связаться непосредственно со мной. Если я не смогу вам помочь, то по крайней мере направлю в нужное русло. Не стесняйтесь. Члены сообщества Vue.js дружелюбны к новичкам.

В процессе чтения пытайтесь самостоятельно реализовывать изученные концепции. Практическое применение — один из лучших способов усвоения материала. Вместо зоомагазина попытайтесь создать собственную торговую площадку, используя эту книгу как руководство.

И последнее: подходите к работе творчески и создайте что-нибудь интересное. Если у вас получится, обязательно напишите мне в Twitter: @ErikCH!

Об авторе

Эрик Ханчетт занимается веб-разработкой на протяжении последних десяти лет. Он автор книги *Ember.js Cookbook* (Packt Publishing, 2016), канала YouTube erik.video и блога по адресу programwitherik.com. Кроме того, Эрик ведет список рассылки goo.gl/UmemSS, в котором делится секретами разработки на JavaScript. Свободное от работы или написания кода время он проводит со своими детьми Уайаттом и Вивиан и женой Сьюзен.

Об иллюстрации на обложке

Иллюстрация на обложке называется «Костюм молодой русской торговки с Охты в 1765 году»[1] и позаимствована из четырехтомника Томаса Джефферис (Thomas Jefferys) «Коллекция платья разных народов, старинного и современного»[2], изданного в Лондоне в 1757–1772 годах. На титульном листе утверждается, что это гравюра ручной раскраски с применением гуммиарабика.

Томаса Джефферис (1719–1771) называли географом при короле Георге III. Это английский картограф и ведущий поставщик карт своего времени. Он чертил и печатал карты для правительства и других государственных органов. На его счету целый ряд коммерческих карт и атласов, в основном Северной Америки. Занимаясь картографией в разных уголках мира, он проявлял интерес к местным традиционным нарядам, которые впоследствии были блестяще переданы в данной коллекции. В конце XVIII века заморские путешествия для собственного удовольствия были относительно новым явлением, поэтому подобные коллекции пользовались популярностью, позволяя получить представление о жителях других стран как настоящим туристам, так и «диванным» путешественникам.

Разнообразие иллюстраций в книгах Джефферис — яркое свидетельство уникальности и оригинальности народов мира в то время. С тех пор тенденции в одежде сильно изменились, а региональные и национальные различия, которые были такими значимыми 200 лет назад, постепенно сошли на нет. В наши дни часто сложно отличить друг от друга жителей разных континентов. Оптимисты могут сказать, что взамен культурного и визуального многообразия мы получили более насыщенную и интересную личную жизнь (или, по крайней мере, ее интеллектуальную и техническую стороны).

В то время как большинство компьютерных изданий мало чем отличаются друг от друга, компания Manning выбирает для своих книг обложки, показывающие богатое региональное разнообразие, которое Джефферис воплотил в своих иллюстрациях два столетия назад. Это ода находчивости и инициативности современной компьютерной индустрии.

---

[1] Habit of a Young Market Woman of Octha in Russia in 1765.

[2] A Collection of the Dresses of Different Nations, Ancient and Modern.

Часть I
Знакомимся с Vue.js

Прежде чем переходить к изучению всех тех крутых возможностей, которые предлагает Vue, нужно поближе познакомиться с этой библиотекой. В первых двух главах мы рассмотрим философию, стоящую за Vue.js и шаблоном проектирования MVVM, а также сравним ее с аналогичными проектами.

Усвоив основы, на которые опирается Vue.js, мы подробно изучим корневой экземпляр Vue (сердце любого приложения) и его структуру. Затем вы узнаете, как привязать к нему программные данные.

Эти главы станут хорошим началом. Прочитав их, вы сможете создавать простые приложения и будете знать, как работает Vue.js.

1 Введение в Vue.js

В этой главе

- Шаблоны проектирования MVC и MVVM.
- Определение реактивного приложения.
- Описание жизненного цикла Vue.
- Оценка архитектуры Vue.js.

Мы уже давно привыкли к интерактивным сайтам. В середине 2000-х годов, на заре Web 2.0, интерактивность и вовлечение пользователей в процесс были в центре внимания. Именно в этот период появились компании Twitter, Facebook и YouTube. Благодаря бурному росту социальных сетей и пользовательского контента Интернет менялся в лучшую сторону.

Чтобы поспевать за этими изменениями и предоставлять бóльшую интерактивность, разработчики начали создавать библиотеки и фреймворки, которые упрощали построение интерактивных сайтов. В 2006 году Джон Резиг (John Resig) выпустил jQuery, тем самым значительно облегчив написание клиентских скриптов внутри HTML. Со временем появились и другие подобные проекты. Сначала они были большими, монолитными и навязывали разработчикам свое видение. Теперь же произошел сдвиг в сторону компактных облегченных библиотек, которые очень просто добавить в любое приложение. Вот мы и подошли к Vue.js.

Vue.js — это библиотека, позволяющая внедрять интерактивное поведение и дополнительные возможности в любой контекст, в котором выполняется JavaScript. Vue можно использовать как на отдельных страницах, решая простые задачи, так и в качестве фундамента для полноценных промышленных приложений.

ПРИМЕЧАНИЕ

В Интернете названия Vue и Vue.js практически взаимозаменяемы. В этой книге я чаще использую более простой вариант, Vue, оставляя Vue.js для тех случаев, когда речь идет непосредственно о коде или библиотеке.

Вы увидите, как Vue и вспомогательные библиотеки позволяют создавать полноценные сложные веб-приложения. Мы исследуем все, от интерфейса, с которым взаимодействуют посетители, до базы данных, снабжающей наш код информацией.

Увидим также, каким образом каждый из приводимых здесь примеров вписывается в общую картину, рассмотрим лучшие методики, которые предлагает индустрия веб-разработки, и научимся интегрировать представленный далее код в собственные проекты, как существующие, так и новые.

Эта книга рассчитана в основном на веб-разработчиков, у которых уже есть некоторый опыт работы с JavaScript и общее представление об HTML и CSS. Тем не менее благодаря универсальности своего *программного интерфейса* (application programming interface, API) Vue подходит для разработчиков всех мастей и проектов любой степени сложности. Она станет надежным подспорьем, даже если вам нужно создать лишь небольшой прототип или простую любительскую программу для личного пользования.

1.1. На плечах у гигантов

Прежде чем начинать писать код для нашего первого приложения или углубляться в недра Vue, следует обратиться к истории программного обеспечения. Лишь зная о проблемах и вызовах, с которыми сталкивался мир веб-разработки в прошлом, можно по достоинству оценить преимущества, которыми обладает эта библиотека.

1.1.1. Шаблон проектирования MVC

Клиентский шаблон проектирования *«модель — представление — контроллер»* (Model – View – Controller, MVC) лежит в основе множества современных фреймворков для разработки веб-приложений (если вы уже знакомы с MVC, можете пролистывать дальше).

Здесь следует упомянуть, что со временем концепция MVC претерпела некоторые изменения. В так называемой классической версии MVC был предусмотрен отдельный набор правил для того, каким образом должны взаимодействовать между собой представление, контроллер и модель. Мы же остановимся на упрощенной интерпретации этого клиентского шаблона проектирования, которая лучше подходит для современных реалий.

MVC применяется для разделения ответственности в приложении (рис. 1.1). Представление отвечает за вывод информации пользователю — это *графический пользовательский интерфейс* (graphical user interface, GUI). В центре находится контроллер. Он помогает передавать события из представления в модель, а данные — из

модели в представление. Внизу мы видим модель, которая содержит бизнес-логику и может предоставлять некое хранилище данных.

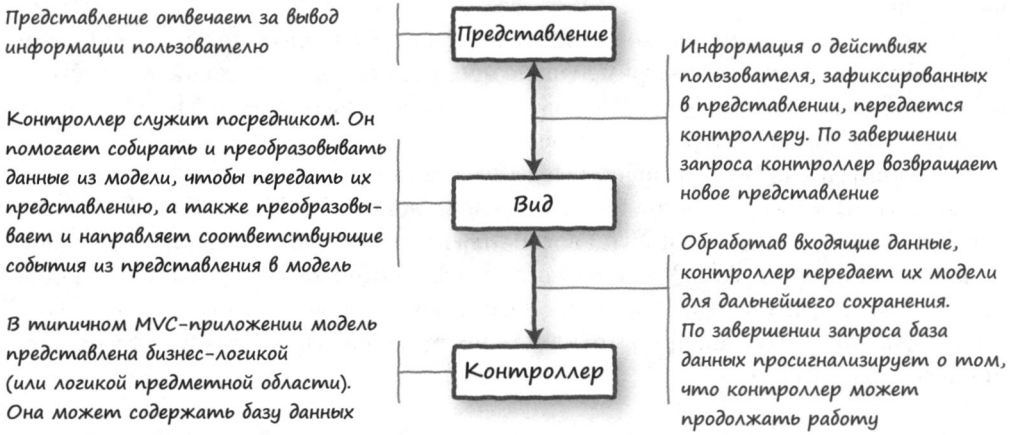


Рис. 1.1. Взаимоотношения модели, представления и контроллера согласно MVC

ДОПОЛНИТЕЛЬНО

Если хотите узнать больше о шаблоне проектирования MVC, можете начать со статьи Мартина Фаулера (Martin Fowler), посвященной эволюции MVC: martinfowler.com/eaaDev/uiArchs.html.

Авторы веб-фреймворков любят этот шаблон проектирования за его надежную, проверенную временем архитектуру. Если вас интересует устройство современных веб-фреймворков, советую почитать книгу Эммитта А. Скотта-младшего (Emmitt A. Scott Jr.) *SPA Design and Architecture* (Manning, 2015).

В современной разработке MVC часто задействуется в рамках одного приложения, предоставляя отличный механизм для разделения программного кода на разные роли. В сайтах, основанных на MVC, каждый запрос инициирует передачу потока информации от клиента к серверу, от сервера к базе данных, а затем обратно. Этот процесс отнимает много времени и ресурсов, что сказывается на отзывчивости пользовательского интерфейса.

Со временем приложения, основанные на веб-технологиях, стали более интерактивными. Теперь они применяют асинхронные запросы и клиентскую архитектуру MVC, чтобы передача данных на сервер не блокировала выполнение и не требовала ожидания ответа. Веб-страницы все больше напоминают настольные программы, поэтому задержки, связанные с клиент-серверным взаимодействием, могут замедлить или даже нарушить работу приложения. И вот тут на выручку приходит следующий шаблон.

Несколько слов о бизнес-логике

Клиентский шаблон проектирования MVC отличается большой гибкостью при выборе места реализации бизнес-логики. На рис. 1.1 логика сконцентрирована в модели. Но это было сделано лишь для простоты, можно было выбрать для этого другие уровни приложения, включая контроллер. С тех пор как в 1979 году шаблон MVC был представлен Трюгве Ринскаугом (Trygve Reenskaug) для языка Smalltalk-76, он претерпел некоторые изменения.

Представьте процесс проверки почтового индекса, введенного пользователем.

- Представление может содержать код на JavaScript, который проверяет почтовый индекс по мере ввода или перед отправкой.
- Модель способна проверить почтовый индекс при создании объекта для хранения входящих данных.
- База данных может накладывать ограничения на поле почтового индекса, то есть модель тоже применяет бизнес-логику. Некоторые разработчики считают это плохим тоном.

Иногда бывает сложно определить, что именно представляет собой бизнес-логика. Мы отдельно остановимся на том, как и где она должна реализовываться. Вы также узнаете, каким образом Vue и сопутствующие библиотеки помогают изолировать функциональность в строго заданных рамках.

1.1.2. Шаблон проектирования MVVM

Когда JavaScript-фреймворки начали поддерживать асинхронные методы программирования, исчезла необходимость в обновлении всей веб-страницы при каждом запросе. Частичное обновление представления позволяет сайтам и приложениям быстрее реагировать, но в определенной степени приводит к дублированию кода. Логика представления часто повторяет бизнес-логику.

Шаблон MVVM (Model — View — ViewModel — «модель — представление — модель представления») — это следующий этап в развитии MVC. Его отличительными чертами являются *модель представления* (ViewModel, VM) и связывание данных. MVVM обеспечивает основу для построения клиентских приложений, которые делают взаимодействие с пользователем и обратную связь более отзывчивыми, избегая при этом затратного дублирования кода в рамках всей архитектуры. Этот шаблон упрощает также модульное тестирование. Но имейте в виду, что для простых пользовательских интерфейсов он может оказаться избыточным.

MVVM позволяет создавать веб-приложения, которые мгновенно реагируют на действия пользователя и позволяют свободно переходить от одной задачи к другой. Модель представления тоже способна играть несколько ролей (рис. 1.2). Такая консолидация ответственности имеет одно фундаментальное последствие для представлений приложения: при изменении данных внутри VM автоматически обновляются все представления, связанные с этой моделью. Механизм связывания

делает данные доступными и гарантирует, что любые их изменения будут отражены в представлении.

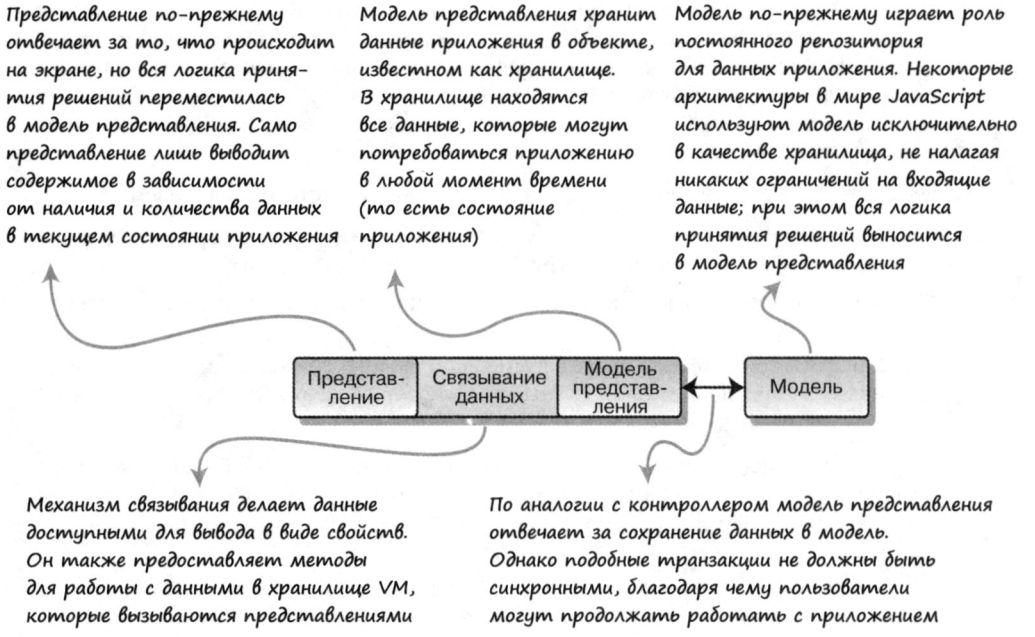


Рис. 1.2. Элементы шаблона проектирования MVVM

ДОПОЛНИТЕЛЬНО

Больше информации об MVVM можно найти в статье Мартина Фаулера (Martin Fowler), посвященной шаблону проектирования Presentation Model: martinfowler.com/eaaDev/PresentationModel.html.

1.1.3. Что такое реактивное приложение

Сама по себе парадигма реактивного программирования не нова. Однако в веб-приложениях ее начали использовать относительно недавно, в основном благодаря наличию фреймворков Vue, React и Angular.

В Интернете много информации о теории реактивности, но нас интересует более узкая тема. Чтобы веб-приложение можно было назвать реактивным, оно должно выполнять следующие функции:

- отслеживать изменения в своем состоянии;
- распространять уведомления об изменениях для всех своих компонентов;
- автоматически отрисовывать представления в ответ на изменения состояния;
- своевременно реагировать на действия пользователя.

Для достижения этих целей реактивные веб-приложения по возможности применяют принципы проектирования MVVM, асинхронные методики, позволяющие избежать блокирования взаимодействия с пользователем, и элементы функционального программирования.

Шаблон проектирования MVVM и реактивное программирование не имеют прямого отношения друг к другу, но у них общая направленность: обе концепции стремятся сделать приложение более отзывчивым и надежным с точки зрения пользователя.

ДОПОЛНИТЕЛЬНО

Если хотите узнать больше о реактивной парадигме программирования Vue, почитайте руководство «Подробно о реактивности» по адресу ru.vuejs.org/v2/guide/reactivity.html.

1.1.4. Калькулятор на JavaScript

Чтобы разобраться в понятиях реактивности и связывания данных, напишем калькулятор на чистом JavaScript (листинг 1.1).

Листинг 1.1. Калькулятор на JavaScript: chapter-01/calculator.html

```
<!DOCTYPE>
<html>
  <head>
    <title>A JavaScript Calculator</title>
    <style>
     p, input { font-family: monospace; }
     p, { white-space: pre; }
    </style>
  </head>
  <body>
    <div id="myCalc">
      <p>x <input class="calc-x-input" value="0"></p>
      <p>y <input class="calc-y-input" value="0"></p>
      <p>--------------------</p>
      <p>= <span class="calc-result"></span></p>
    </div>
    <script type="text/javascript">
     (function(){

       function Calc(xInput, yInput, output) {
         this.xInput = xInput;
         this.yInput = yInput;
         this.output = output;
       }

       Calc.xName = 'xInput';
```

Поля для ввода значений x и y, привязанных к функции runCalc

Вывод введенных значений x и y

Конструктор для создания экземпляра Calc

```
Calc.yName = 'yInput';

Calc.prototype = {
  render: function (result) {
    this.output.innerText = String(result);
  }
};

function CalcValue(calc, x, y) {
  this.calc = calc;
  this.x = x;
  this.y = y;
  this.result = x + y;
}

CalcValue.prototype = {
  copyWith: function(name, value) {
    var number = parseFloat(value);

    if (isNaN(number) || !isFinite(number))
      return this;

    if (name === Calc.xName)
      return new CalcValue(this.calc, number, this.y);

    if (name === Calc.yName)
      return new CalcValue(this.calc, this.x, number);

    return this;
  },
  render: function() {
    this.calc.render(this.result);
  }
};

function initCalc(elem) {

  var calc =
    new Calc(
      elem.querySelector('input.calc-x-input'),
      elem.querySelector('input.calc-y-input'),
      elem.querySelector('span.calc-result')
    );
  var lastValues =
    new CalcValue(
      calc,
      parseFloat(calc.xInput.value),
      parseFloat(calc.yInput.value)
    );

  var handleCalcEvent =
```

Конструктор значений для экземпляра Calc

Инициализация компонента Calc

Обработчик событий

```
        function handleCalcEvent(e) {
          var newValues = lastValues,
              elem = e.target;

          switch(elem) {
            case calc.xInput:
              newValues =
                lastValues.copyWith(
                  Calc.xName,
                  elem.value
                );
              break;
            case calc.yInput:
              newValues =
                lastValues.copyWith(
                  Calc.yName,
                  elem.value
                );
              break;
          }

          if(newValues !== lastValues){
            lastValues = newValues;
            lastValues.render();
          }
        };

      elem.addEventListener('keyup', handleCalcEvent, false);

      return lastValues;
    }

    window.addEventListener(
      'load',
      function() {
        var cv = initCalc(document.getElementById('myCalc'));
        cv.render();
      },
      false
    );

  }());
  </script>
 </body>
</html>
```

Регистрация обработчика для события keyup

Это калькулятор на языке ES5 JavaScript (более современной версией JavaScript, ES6/2015, воспользуемся позже). Мы задействуем выражение IIFE (немедленно вызываемая функция), которое запускает наш скрипт. Конструктор отвечает за хранение значений, а обработчик `handleCalcEvent` реагирует на событие `keyup`.

1.1.5. Калькулятор на Vue

Пока что не стоит обращать внимания на синтаксис Vue в этом примере. Мы не стремимся понять каждый фрагмент кода, а лишь пытаемся сравнить две реализации. Хотя, если вы хорошо понимаете пример с JavaScript, вам не составит большого труда разобраться со следующим кодом (листинг 1.2), по крайней мере на теоретическом уровне.

Листинг 1.2. Калькулятор на Vue: chapter-01/calculatorvue.html

```
<!DOCTYPE html>
<html>
<head>
  <title>A Vue.js Calculator</title>
  <style>
    p, input { font-family: monospace; }
    p { white-space: pre; }
  </style>
</head>
<body>
  <div id="app">
    <p>x <input v-model="x"></p>
    <p>y <input v-model="y"></p>
    <p>---------------------</p>
    <p>= <span v-text="result"></span></p>
  </div>

  <script src="https://unpkg.com/vue/dist/vue.js"></script>
  <script type="text/javascript">
  function isNotNumericValue(value) {
    return isNaN(value) || !isFinite(value);
  }
    var calc = new Vue({
      el: '#app',
      data: { x: 0, y: 0, lastResult: 0 },
      computed: {
        result: function() {
          let x = parseFloat(this.x);
          if(isNotNumericValue(x))
          return this.lastResult;

          let y = parseFloat(this.y);
          if(isNotNumericValue(y))
          return this.lastResult;

          this.lastResult = x + y;

          return this.lastResult;
        }
      }
    });
  </script>
</body>
</html>
```

Привязка к DOM для нашей программы

Поля ввода для нашей программы

Результаты будут выводиться в этом элементе span

Элемент script, который подключает библиотеку Vue.js

Инициализация приложения

Подключение к DOM

Переменные, объявленные в приложении

Вычисления выполняются с помощью вычисляемого свойства

1.1.6. Сравнение JavaScript и Vue

По большей части эти две реализации калькулятора имеют разный код. Оба скрипта, представленных на рис. 1.3, доступны в репозитории с кодом для этой главы. Вы можете запустить их по очереди и посмотреть, как они себя ведут.

```
<script type="text/javascript">
 (function(){

   function Calc(xInput, yInput, output) {
     this.xInput = xInput;
     this.yInput = yInput;
     this.output = output;
   }

   Calc.xName = 'xInput';
   Calc.yName = 'yInput';

   Calc.prototype = {
     render: function (result) {
       this.output.innerText = String(result);
     }
   };

   function CalcValue(calc, x, y) {
     this.calc = calc;
     this.x = x;
     this.y = y;
     this.result = x + y;
   }
```

```
<script src="https://unpkg.com/vue/dist/vue.js"></script>
<script type="text/javascript">
function isNotNumericValue(value) {
  return isNaN(value) || !isFinite(value);
}
  var calc = new Vue({
    el: '#app',
    data: { x: 0, y: 0, lastResult: 0 },
    computed: {
      result: function() {
        let x = parseFloat(this.x);
        if(isNotNumericValue(x))
        return this.lastResult;

        let y = parseFloat(this.y);
        if(isNotNumericValue(y))
        return this.lastResult;

        this.lastResult = x + y;

        return this.lastResult;
      }
    }
  });
</script>
```

Рис. 1.3. Сравнение версий реактивного калькулятора, написанных на чистом JavaScript *(слева)* и Vue *(справа)*

Ключевое различие между этими реализациями состоит в том, как инициируется обновление итоговых вычислений и как результаты попадают обратно на страницу. В примере с Vue все обновления и вычисления на странице берет на себя механизм связывания `v-model`. Когда мы создаем экземпляр программы с помощью выражения `new Vue({ ... })`, Vue анализирует код и HTML-разметку, после чего создает все данные и обработчики событий, необходимые для работы приложения.

1.1.7. Как Vue использует MVVM и реактивность

Библиотеку Vue иногда называют *прогрессивным фреймворком*. В общем смысле это означает, что для выполнения простых задач ее можно встраивать в существующие веб-страницы, а еще она способна становиться основой для построения крупномасштабных веб-проектов.

Какой бы способ применения вы ни выбрали, любое Vue-приложение содержит как минимум один *экземпляр Vue*. Самые простые программы ограничиваются единственным экземпляром, который связывает заданную разметку и данные, хранящиеся в модели представления (рис. 1.4).

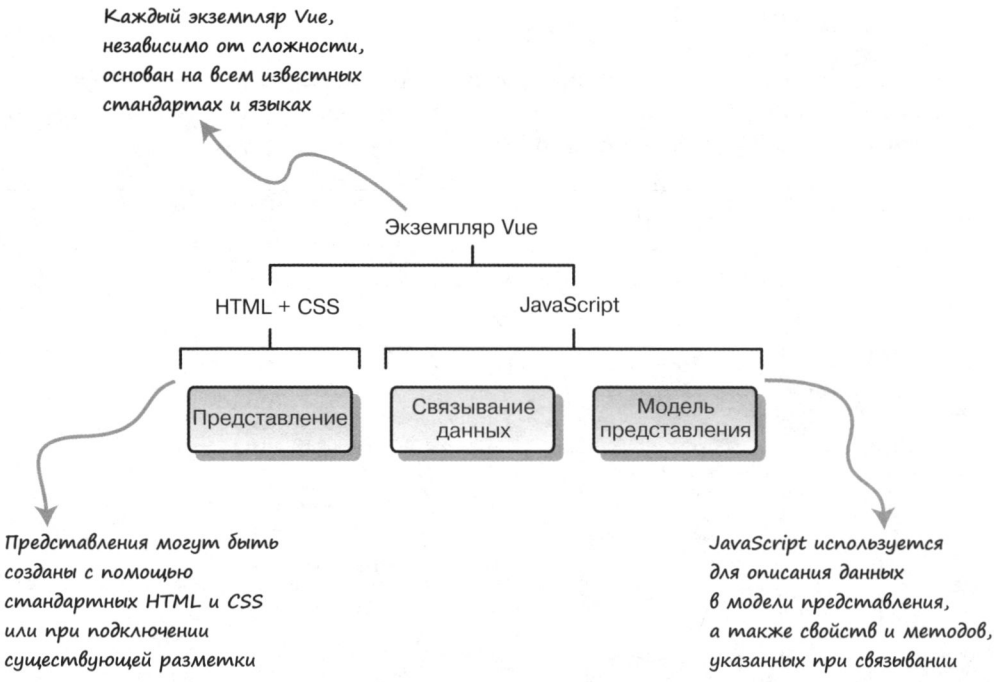


Рис. 1.4. Обычно экземпляр Vue связывает разметку с данными в модели представления

Отдельно взятый экземпляр Vue полностью основывается на веб-технологии и существует исключительно в рамках браузера. В сущности, это означает, что для обновления выводимых данных, выполнения бизнес-логики или любой другой задачи, входящей в область ответственности представления или модели представления, не нужно полагаться на генерацию страниц на сервере. Помня об этом, еще раз пройдемся по примеру с отправкой формы.

Наверное, самая поразительная особенность реактивного подхода по сравнению с клиентской архитектурой MVC — то, что на протяжении всего сеанса работы пользователь очень редко сталкивается с перезагрузкой страницы в браузере, если вообще сталкивается. Представление, модель представления и связывание данных реализованы в HTML и JavaScript, поэтому задачи, которые делегируются модели, могут быть асинхронными и не блокировать взаимодействие с пользователем. Когда модель возвращает новые данные, механизм связывания, встроенный в Vue, вносит в представление необходимые изменения.

Наверное, главная задача Vue — это обеспечение взаимодействия с пользователем путем установления и поддержания связи между представлениями, которые мы создаем, и данными в модели представления. Как вы вскоре увидите при написании нашего первого проекта, в этом смысле Vue обеспечивает надежный фундамент для любого реактивного веб-приложения.

1.2. Почему именно Vue.js

Начало нового проекта всегда связано с принятием тех или иных решений. Одним из ключевых является выбор фреймворка или библиотеки. Неважно, идет речь о компании или программисте-одиночке, подбор подходящего инструмента всегда имеет большое значение. К счастью, Vue.js — очень разносторонняя библиотека, способная справиться с разными задачами.

Далее перечислены несколько самых распространенных проблем, с которыми можно столкнуться при создании нового проекта, а также решения, которые предлагаются Vue или парадигмой реактивного программирования в целом.

- *У нашей команды нет большого опыта работы с веб-фреймворками.* Одно из важнейших преимуществ библиотеки Vue заключается в том, что она не требует никаких специальных знаний. Любое Vue-приложение пишется с помощью HTML, CSS и JavaScript — знакомых всем инструментов, которые сразу позволяют погрузиться в работу. Даже если вам не приходилось раньше заниматься клиентской разработкой, вы всегда сможете опереться на предыдущий опыт работы с шаблоном проектирования MVC, который во многом похож на MVVM.
- *У нас уже есть наработки, от которых мы не хотим отказываться.* Не волнуйтесь, вам не придется отказываться от бережно подобранных стилей или того крутого виджета, над которым вы корпели. Будь то существующий проект с множеством зависимостей или новое приложение, в котором вы хотите использовать знакомые библиотеки, — Vue не доставит вам никаких хлопот. Вы можете дальше работать с такими CSS-фреймворками, как Bootstrap или Bulma, оставить уже написанные компоненты для jQuery или Backbone, интегрировать полюбившуюся библиотеку для выполнения HTTP-запросов или работать с объектами Promise.
- *Нам нужно быстро создать прототип и посмотреть на реакцию пользователей.* Как мы уже видели в первом приложении, чтобы начать программировать с помощью Vue, достаточно подключить библиотеку Vue.js к любой отдельной веб-странице. Не нужно никаких сложных инструментов для сборки! Разработка прототипа от начала до конца занимает одну-две недели, что позволяет собирать отзывы пользователей как можно раньше и чаще.
- *Наш продукт почти всегда применяется на мобильных устройствах.* Минимизированный и сжатый файл Vue.js занимает примерно 24 Кбайт, что довольно скромно, как для клиентского фреймворка. Такую библиотеку легко загрузить через сотовое соединение. В Vue 2 появилась поддержка *отрисовки на стороне сервера* (server-side rendering, SSR). Эта стратегия позволяет минимизировать загружаемые данные на начальном этапе и запрашивать новые представления и ресурсы по мере необходимости. В сочетании с эффективным кешированием компонентов это позволяет еще сильнее снизить потребление трафика.
- *Наш продукт обладает уникальными и нестандартными возможностями.* Vue-приложения изначально проектируются модульными и расширяемыми. Их компоненты поддерживают многократное использование, наследование

и добавление новых функций с помощью примесей. Расширяйте возможности Vue, работая с подключаемыми модулями и пользовательскими директивами.

- *Мы имеем огромную аудиторию и волнуемся о производительности.* Недавно библиотека Vue была переписана с прицелом на надежность, производительность и скорость и теперь задействует модель Virtual DOM. Это означает, что все операции сначала выполняются с представлением DOM, которое не привязано к браузеру, а затем внесенные изменения копируются на саму страницу. В результате Vue стабильно выигрывает в производительности у других клиентских библиотек. Но, поскольку обобщенные тесты часто оказываются слишком абстрактными, я всегда рекомендую заказчикам выбрать несколько типичных и экстремальных случаев, разработать тестовый сценарий и самостоятельно измерить результаты. Подробности о модели Virtual DOM и сравнение Vue с конкурентами можно найти на странице ru.vuejs.org/v2/guide/comparison.html.
- *У нас уже есть готовый процесс сборки, тестирования и/или развертывания.* Мы подробно рассмотрим эти темы в последующих главах, но если коротко, то Vue легко интегрируется с многими популярными фреймворками для сборки (Webpack, Browserify и т. д.) и тестирования (Karma, Jasmine и т. д.). Во многих случаях модульные тесты можно переносить без изменений, если они основаны на существующем фреймворке. Если же вы начинаете с нуля, но хотите использовать эти инструменты, Vue предоставляет шаблоны проектов, которые автоматически интегрируют их. Проще говоря, Vue легко адаптировать и добавить в существующие проекты.
- *Что делать, если в ходе или после разработки нам понадобится помощь?* Двумя бесценными преимуществами проекта Vue являются его сообщество и сопровождающая экосистема. Вам доступна отличная документация в виде отдельных справочных материалов и комментариев внутри кода, а основная команда разработки активна и отзывчива. Не менее, а может, и более важна помощь со стороны сообщества разработчиков. Gitter и форум Vue полны людей, всегда готовых помочь. Каждый день экосистема пополняется новыми подключаемыми модулями, интеграциями и библиотеками, которые делают доступным популярный сторонний код.

Я сам неоднократно задавал все эти вопросы и теперь почти во всех своих проектах рекомендую использовать Vue. Надеюсь, что эта книга вдохновит вас сделать то же самое в своем следующем проекте.

1.3. Мысли на будущее

Во вводной главе мы затронули довольно много тем. Если вы только начинаете заниматься веб-разработкой, это, вероятно, ваш первый контакт с архитектурой MVVM или реактивным программированием. Тем не менее вы уже могли убедиться в том, что процесс построения реактивных приложений не так сложен, как может показаться поначалу.

Наверное, самый полезный вывод, который можно сделать из этой главы, связан даже не с Vue, а с тем, насколько реактивный подход упрощает написание программ и работу ними. Приятно также, что уменьшается объем шаблонного кода, связанного с интерфейсом. Отсутствие необходимости кодировать все взаимодействия с пользователем позволяет нам сосредоточиться на моделировании данных и внешнем виде страницы, а также связывании этих двух аспектов, что благодаря Vue очень просто сделать.

Наверное, вы уже придумали миллион способов улучшить наше скромное приложение. И это замечательно: никогда не прекращайте экспериментировать и играть с кодом. Вот несколько мыслей, которые приходят мне в голову при взгляде на первый пример.

- Как избавиться от повторения одних и тех же текстовых строк в разных местах?
- Можно ли очистить текст по умолчанию, когда пользователь переходит к полю ввода? А что насчет восстановления текста, когда поле пустое?
- Реально ли избежать ручного кодирования каждого поля?

Ответы на эти и многие другие вопросы будут даны в главе 2. Vue растет вместе с вами и вашим кодом, поэтому мы всегда будем стремиться рассматривать разные подходы, оценивать их сильные и слабые стороны и учиться выбирать лучшую методику в конкретной ситуации.

Ладно, теперь посмотрим, как улучшить то, что мы уже написали!

Резюме

- Краткое знакомство с принципами работы моделей, представлений и контроллеров, а также с их устройством в Vue.js.
- Как Vue.js может сэкономить время при создании приложения.
- Почему Vue.js стоит выбрать для своего следующего проекта.

2 Экземпляр Vue

В этой главе

- Создание экземпляра Vue.
- Отслеживание жизненного цикла Vue.
- Добавление данных в экземпляр Vue.
- Привязка данных к разметке.
- Форматирование нашего вывода.

В этой книге мы создадим полноценное веб-приложение — интернет-магазин со списком товаров, выставлением счета, панелью администрирования и т. д. Построение такого проекта может показаться преждевременным, особенно если у вас нет опыта веб-разработки, но Vue позволяет начать с малого и по мере обучения плавно продвигаться вперед. В итоге у вас получится довольно сложный продукт.

Такая плавность на каждом этапе развития программы достигается за счет того, что само Vue-приложение и все его компоненты являются экземплярами Vue. Благодаря этому вы можете расширять Vue, создавая собственные экземпляры с новыми свойствами.

Экземпляр Vue имеет слишком много аспектов, которые не раскроешь в одной-единственной главе, поэтому мы будем двигаться вперед постепенно, по мере развития нашего проекта. Позже при изучении новых возможностей мы будем часто ссылаться на материал об экземпляре и жизненном цикле Vue, представленный в этой главе.

2.1. Наше первое приложение

Первым делом мы заложим фундамент интернет-магазина, выведем его название и создадим описание отдельно взятого продукта. Мы сосредоточимся на создании Vue-приложения и отображении данных с помощью модели представления. Результат, который должны получиться к концу главы, показан на рис. 2.1.

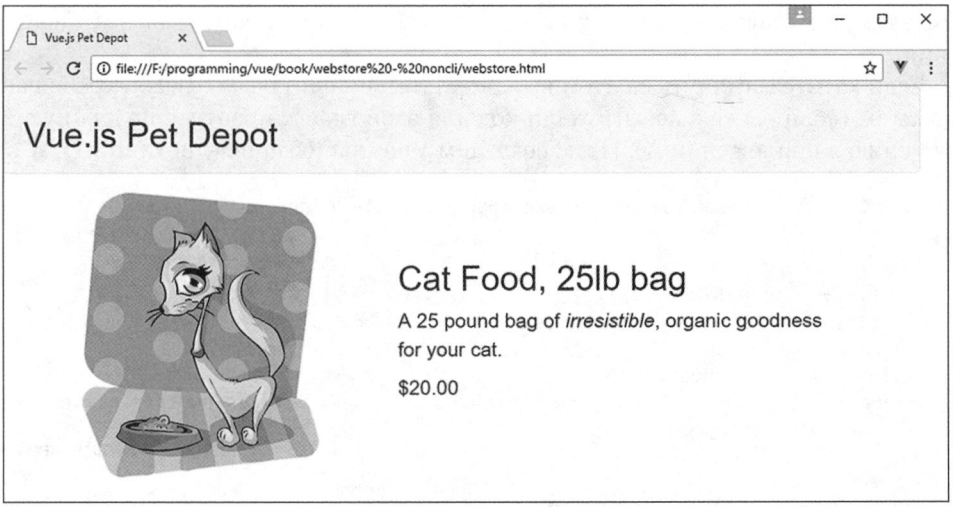


Рис. 2.1. То, с чего начинается наш скромный интернет-магазин

МЕЖДУ ПРОЧИМ

Строго говоря, это ваше второе Vue-приложение, если вы уже запускали пример простого калькулятора из листинга 1.2. Вы уже практически бывалый веб-разработчик!

Перед тем как начать, следует загрузить расширение vue-devtools для вашего браузера. О том, как это сделать, можно узнать в приложении А.

2.1.1. Корневой экземпляр Vue

Сердце любого Vue-приложения, большого или маленького, — это *корневой экземпляр Vue* (или просто экземпляр Vue). Для его создания необходимо вызвать *конструктор Vue* `new Vue()`. Конструктор собирает проект воедино, компилируя HTML-шаблон, инициализируя любые данные экземпляра и создавая обработчики событий, чтобы сделать приложение интерактивным.

Конструктор Vue принимает один объект JavaScript, известный как *options* (опции): `new Vue({ /* здесь размещаются опции */ })`. Наша задача — наполнить этот объект всем, что необходимо для запуска приложения. Но сначала поговорим об отдельно взятой опции под названием `el`.

Опция `el` определяет элемент DOM (`el` от *element*), к которому Vue монтирует приложение. Это будет входная точка для кода.

Далее показан каркас интернет-магазина. Чтобы вам было легче ориентироваться, все листинги помещены в отдельные файлы, доступные в репозитории для этой главы. Чтобы запустить приложение, нужно объединить все фрагменты кода в одном файле `index.html`. Да, со временем он станет довольно крупным, но это нормально.

В следующих главах мы поговорим о способах разбиения программы на отдельные файлы.

Если хотите взглянуть на итоговый результат, поищите файл `index.html` в папке chapter-02 (если вы еще не загрузили код для этой главы, прочтите подробную инструкцию в приложении А). Итак, создадим Vue-приложение (листинг 2.1).

Листинг 2.1. Наше первое Vue-приложение: chapter-02/first-vue.html

```
<html>
  <head>
    <title>Vue.js Pet Depot</title>
    <script src="https://unpkg.com/vue"></script>
    <link rel="stylesheet" type="text/css" href="assets/css/app.css"/>
    <link rel="stylesheet"
    ➥href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
    crossorigin="anonymous">
  </head>
  <body>
    <div id="app"></div>

    <script type="text/javascript">
      var webstore = new Vue({
        el: '#app'
      });
    </script>
  </body>
</html>
```

Подключаем версию Vue.js из CDN

Внутренняя таблица стилей app.css, а также Bootstrap

Элемент, к которому Vue примонтирует приложение

Конструктор Vue

CSS-селектор для поиска точки входа в DOM

Вся разметка состоит из одного элемента `div`, который находят с помощью селектора `#app` и используют в качестве точки входа для приложения. Этот селектор имеет такой же синтаксис, как и CSS (например, `#id`, `.class`).

ПРИМЕЧАНИЕ

Для разметки и задания визуальных стилей в примерах применяется Bootstrap 3. Это отличное решение, которое помогает сосредоточиться на Vue.js. Недавно была выпущена версия Bootstrap 4, но, так как дизайн не основной приоритет этой книги, я решил оставить Bootstrap 3. Приводимые здесь примеры совместимы с версией 4, но при переходе вам, возможно, придется поменять несколько классов. Имейте это в виду.

Если предоставленный нами CSS-селектор вернет несколько элементов DOM, Vue примонтирует приложение к первому из них. Например, если бы в HTML-документе было три элемента `div`, то вызов конструктора `Vue({ el: 'div' })` привел бы к привязыванию приложения к тому, который идет первым.

Если вам нужно запустить несколько экземпляров Vue на одной странице, примонтируйте их к разным элементам DOM, используя уникальные селекторы. Это может показаться странным решением, но, если Vue задействована для построения небольших компонентов, таких как карусель изображений или веб-форма, наличие нескольких корневых экземпляров Vue в одном документе имеет смысл.

2.1.2. Убедимся в том, что приложение работает

Перейдем в Chrome и откроем файл с вашим первым Vue-приложением (см. листинг 2.1). Перед вами появится пустая страница (ведь мы не добавили никакой видимой разметки!).

После загрузки страницы откройте консоль JavaScript, если еще не сделали этого. Если все прошло как нужно, вы должны увидеть... Барабанная дробь!.. Абсолютно ничего (или сообщения об использовании отладочного режима или загрузке расширения vue-devtools, если у вас его нет). Так может выглядеть ваша консоль (рис. 2.2).

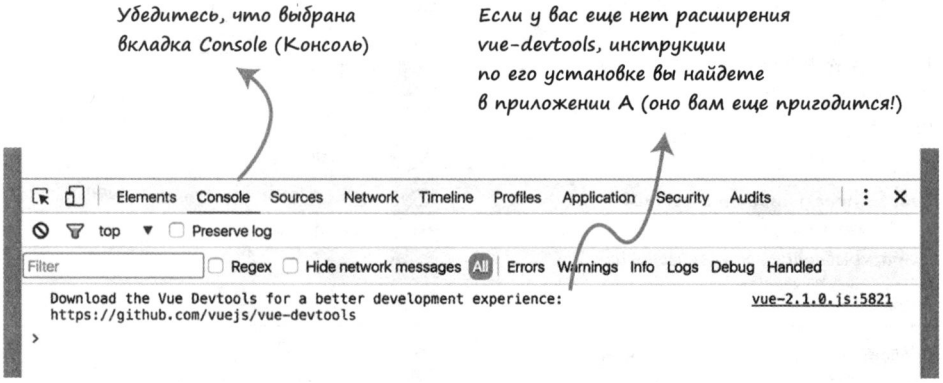


Рис. 2.2. Консоль JavaScript без ошибок и предупреждений

Основы отладки Vue

Даже при загрузке в Chrome такого простого приложения, как наше, что-то может пойти не так. Далее приводятся вероятные проблемы.

- `Uncaught SyntaxError: Unexpected identifier`. Эта ошибка почти всегда свидетельствует об опечатке в коде на JavaScript — обычно пропущенной запятой или фигурной скобке. Вы можете щелкнуть на ссылке с названием файла и номером строки, которая указана справа от ошибки, чтобы перейти к этому коду. Помните, что сама опечатка может находиться несколькими строчками выше или ниже.
- `[Vue warn]: Property or method "propertyname" is not defined ...`. Это предупреждение о том, что в объекте с опциями чего-то не хватает. Проверьте наличие свойства или метода с заданным именем, а если все на месте, поищите опечатки в названии. Кроме того, проверьте имена привязок в своей разметке.

Первые попытки найти проблему могут быть утомительными и безрезультатными, но после устранения нескольких ошибок этот процесс станет более привычным.

Если вы натолкнулись на особенно замысловатую ошибку или никак не удается решить какую-то проблему, можете попросить помощи в разделе Help (Помощь) официального форума Vue (forum.vuejs.org/c/) или в чате Gitter (gitter.im/vuejs/vue).

Завершив инициализацию и примонтировав приложение, Vue возвращает ссылку на корневой экземпляр, которую мы сохраняем в переменную `webstore`. Можно воспользоваться этой переменной, чтобы исследовать программу в консоли JavaScript. Сделаем это прямо сейчас и убедимся в том, что приложение успешно запустилось.

Введите `webstore` в строке приглашения консоли. Результатом будет объект Vue, который можно просматривать отдельно. Пока что просто пощелкайте по треугольникам (►) и посмотрите на свойства корневого экземпляра (рис. 2.3).

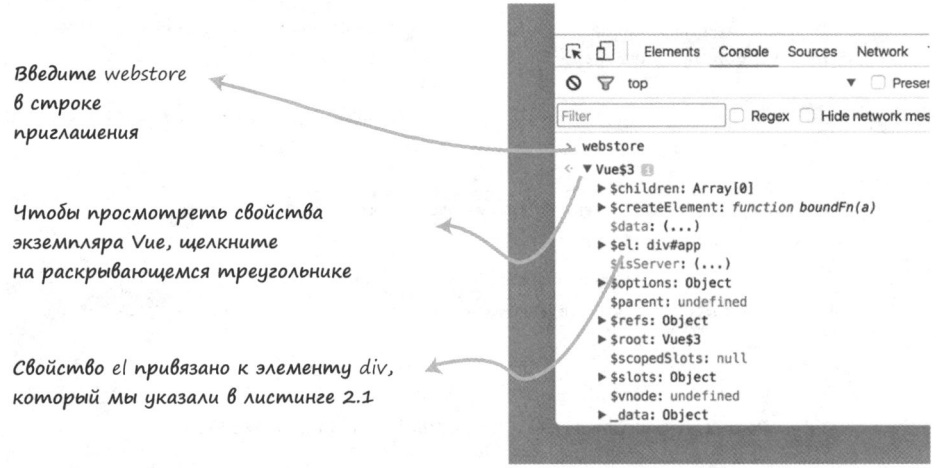


Рис. 2.3. Использование переменной webstore для вывода экземпляра Vue и исследования его свойств

Поиск свойства `el`, которое мы указали среди опций приложения, не должен оказаться слишком трудным, хотя, вероятно, придется немного прокрутить список вниз. В будущем мы будем использовать консоль с целью доступа к экземпляру Vue для отладки, манипуляции данными и вызова определенных функций программы во время ее выполнения. Так мы сможем понять, что код ведет себя так, как было задумано. Чтобы попасть внутрь работающего приложения, используйте также расширение vue-devtools (еще раз напоминаю, что инструкции по его установке приведены в приложении А). Сравним его с консолью JavaScript. Разные элементы vue-devtools показаны на рис. 2.4.

Расширение vue-devtools — это мощное средство исследования Vue-приложения, его данных и иерархии его компонентов. Дерево компонентов поддерживает поиск и показывает взаимоотношения, которых нельзя увидеть в консоли JavaScript, что особенно полезно в ходе работы со сложными программами. Подробнее о компонентах и о том, как они относятся к экземпляру Vue, поговорим позже в этой главе.

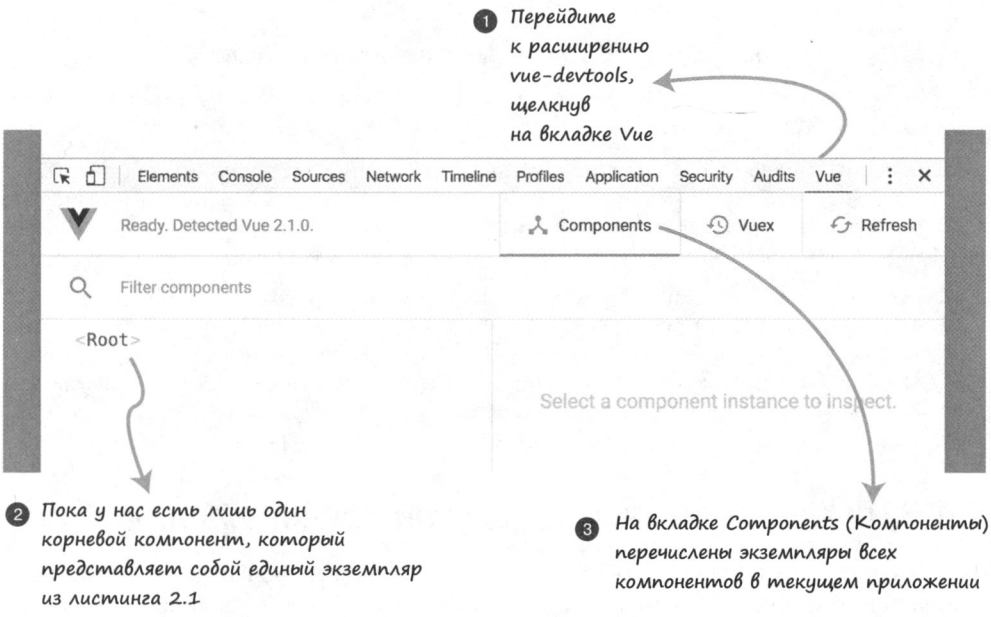


Рис. 2.4. Панель vue-devtools, на которой пока ничего не выбрано

Используем оба этих инструмента для решения проблем, которые будут возникать в ходе разработки проекта. На самом деле vue-devtools предоставляет еще один способ доступа к экземпляру приложения в консоли JavaScript (рис. 2.5).

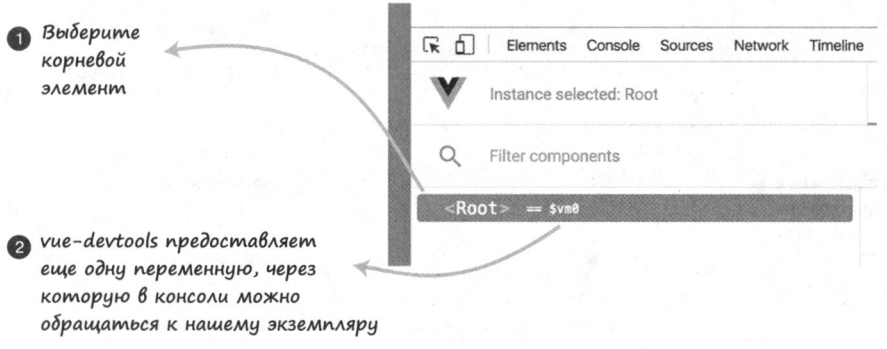


Рис. 2.5. Корневой экземпляр, выбранный в vue-devtools, и переменная, которая динамически с ним связывается

При выборе экземпляра в древовидном представлении vue-devtools присваивает ссылку на него переменной `$vm0`, как сделано ранее с `webstore`. Попробуйте исследовать корневой экземпляр Vue с помощью `$vm0`.

Зачем может понадобиться больше одной ссылки

Наличие нескольких способов доступа к экземпляру может показаться излишним, но иногда это бывает полезно.

Присвоив корневой экземпляр глобальной переменной `webstore`, мы обеспечили возможность обращаться к приложению из другого JavaScript-кода на той же странице. Это позволяет интегрироваться с другими библиотеками, фреймворками или нашим собственным кодом, которому нужно ссылаться на приложение.

Экземпляр Vue, присвоенный переменной `$vm0`, отражает текущий выбор, сделанный в vue-devtools. Когда приложение состоит из сотен или даже тысяч экземпляров, декларативное присваивание каждого из них — не самое удобное занятие. Поэтому при отладке таких сложных приложений очень важно иметь возможность обращаться к отдельным экземплярам, создаваемым программно.

2.1.3. Выведем что-нибудь внутри представления

Пока что все шло как по маслу. Немного усложним задачу и выведем данные экземпляра приложения в его шаблоне. Как вы помните, в качестве основы для шаблона наш экземпляр использует элемент DOM.

Сначала выведем название нашего магазина. Для этого нужно передать данные в конструктор Vue и связать их с представлением. Обновим код программы из листинга 2.1 (листинг 2.2).

Листинг 2.2. Добавление и связывание данных: chapter-02/data-binding.html

```
<html>
  <head>
    <title>Vue.js Pet Depot</title>
    <script src="https://unpkg.com/vue"></script>  </head>
  <body>
    <div id="app">
      <header>
        <h1 v-text="sitename"></h1>
      </header>
    </div>

    <script type="text/javascript">
      var webstore = new Vue({
        el: '#app', // <=== Не забудьте про эту запятую!
        data: {
          sitename: 'Vue.js Pet Depot'
        }
      });
    </script>
  </body>
</html>
```

Привязка данных для свойства sitename

Внутрь div добавлен элемент header

Свойство sitename, которое мы привязали в элементе header

К параметрам Vue добавлен объект data

К опциям, которые мы передаем в конструктор Vue, добавлен объект с данными. Этот объект содержит одно свойство `sitename` с названием магазина.

Название магазина нужно куда-то поместить, поэтому внутрь корневого элемента `div` разметки мы вставили элемент `header`. В заголовочном теге `<h1>` используется директива для связывания данных `v-text="sitename"`.

Директива `v-text` выводит текстовое представление свойства, на которое она ссылается. При запуске приложения мы должны увидеть заголовок с текстом Vue.js Pet Depot внутри.

Если нужно вывести значение свойства посреди длинной строки, воспользуйтесь для его связывания синтаксисом Mustache: `{{ имя-свойства }}`. Например, так можно добавить название магазина в предложение: `<p>Добро пожаловать в {{ имя-сайта }}</p>`.

СОВЕТ

Из всей спецификации Mustache в Vue заимствован лишь синтаксис интерполяции текста {{ ... }}. Если хотите узнать больше, можете почитать руководство по адресу mustache.github.io/mustache.5.html.

Закончив со связыванием данных, откроем браузер и посмотрим, как выглядит новый заголовок.

2.1.4. Исследование свойств в Vue

После перезагрузки приложения в Chrome на странице должен появиться заголовок, гордо отображающий значение свойства `sitename` (рис. 2.6). За внешний вид заголовка отвечает таблица стилей, находящаяся в файле `chapter-02/assets/css/app.css`. Мы будем сочетать ее с Bootstrap для оформления приложения. Если хотите поиграть с визуальной составляющей заголовка, откройте этот файл и найдите определение стилей `header h1`.

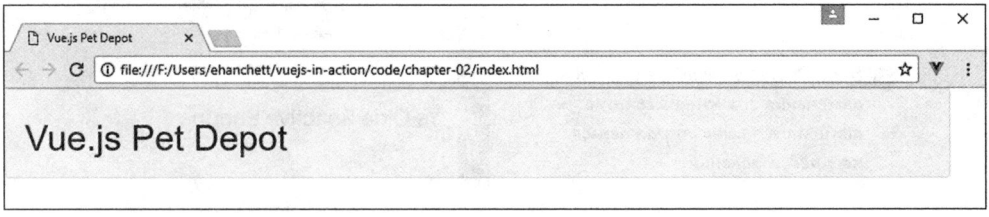


Рис. 2.6. Свойство sitename выводится в заголовке магазина

Vue автоматически создает геттер и сеттер для каждого свойства в объекте данных, когда тот инициализирует приложение. Это позволяет обновлять и извлекать текущее значение любых свойств экземпляра, не требуя написания дополнительного кода. Чтобы увидеть эти функции в действии, выведем значение свойства `sitename` с помощью геттера.

Геттер и сеттер свойства `sitename` доступны на самом верхнем уровне экземпляра приложения (рис. 2.7). Это позволяет обращаться к свойствам из любого кода на JavaScript, который взаимодействует с нашей программой.

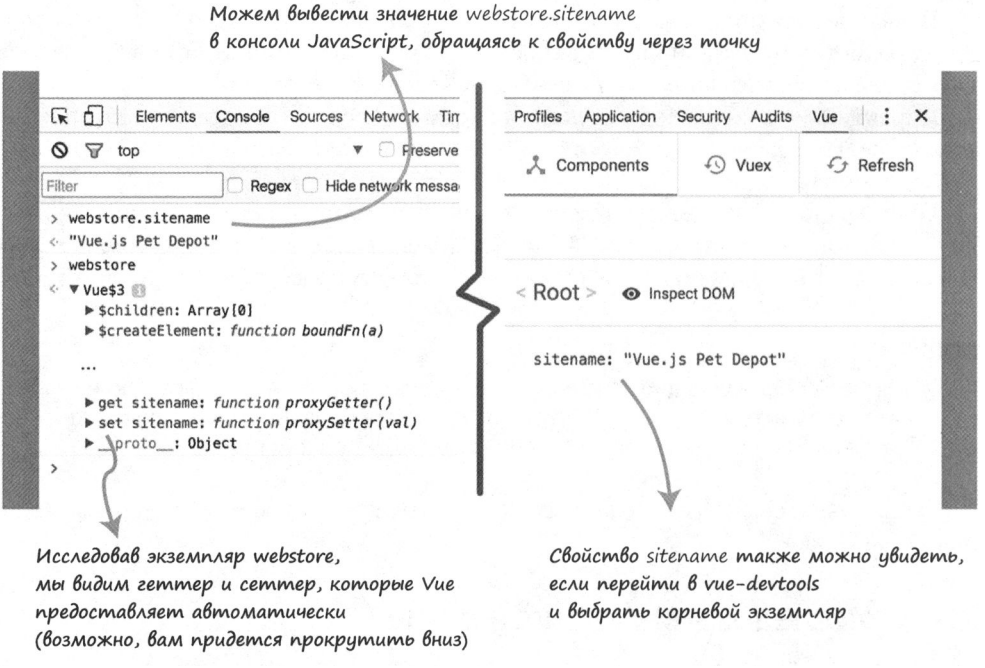


Рис. 2.7. С помощью консоли и vue-devtools можно узнать содержимое свойства sitename

То же свойство увидим в vue-devtools, если выберем экземпляр `<root>`. Теперь взгляните на рис. 2.8, где для присваивания значения `sitename` в консоли JavaScript используется сеттер.

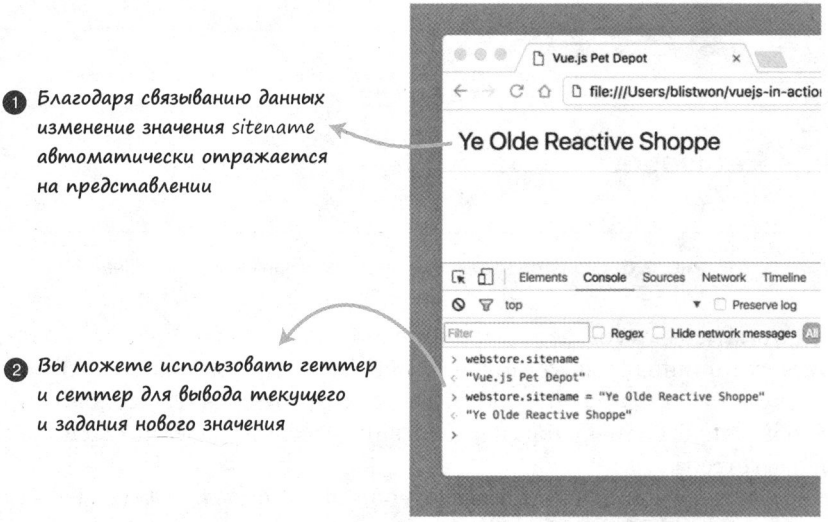


Рис. 2.8. Вывод и обновление свойства sitename с помощью геттера и, соответственно, сеттера

Указав новое значение для свойства `sitename` и нажав Enter, вы увидите, что элемент `header` автоматически обновился. Так работает цикл событий в Vue. Рассмотрим жизненный цикл приложения, чтобы понять, как и когда изменения данных доходят до представления.

2.2. Жизненный цикл Vue

Процесс запуска Vue-приложения состоит из последовательных этапов, совокупность которых называется жизненным циклом. И хотя большую часть времени программа проводит в цикле событий, основную работу библиотека Vue выполняет во время создания приложения. Рассмотрим схему жизненного цикла (рис. 2.9).

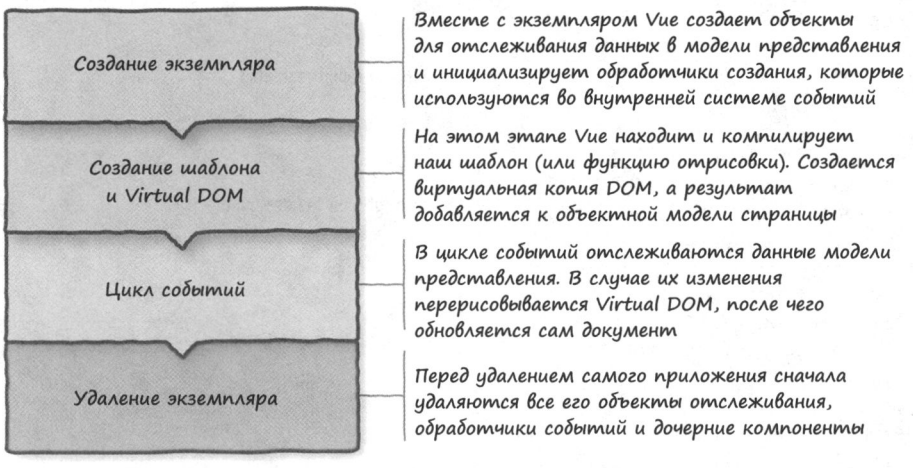


Рис. 2.9. Схема жизненного цикла Vue, разделенного на четыре этапа

Каждый следующий этап жизненного цикла Vue основывается на предыдущем. Вам, наверное, интересно, что такое Virtual DOM и как работает функция отрисовки. *Virtual DOM* – это упрощенная абстракция, которая представляет *объектную модель документа* (Document Object Model, DOM). Она имитирует его дерево, с которым обычно работает браузер. Обновление Virtual DOM происходит намного быстрее, чем изменение настоящей объектной модели. Функция отрисовки отвечает за вывод информации пользователю. Подробнее об экземпляре Vue и хуках жизненного цикла можно почитать в официальном руководстве по адресу ru.vuejs.org/v2/guide/instance.html.

2.2.1. Добавление хуков жизненного цикла

Чтобы увидеть, как экземпляр приложения проходит через разные этапы жизненного цикла, можно написать функции обратного вызова для соответствующих хуков. Обновим код основного файла, `index.html` (листинг 2.3).

Листинг 2.3. Добавление хуков жизненного цикла в наш экземпляр: chapter-02/life-cycle-hooks.js

```
var APP_LOG_LIFECYCLE_EVENTS = true;

var webstore = new Vue({
  el: "#app",
  data: {
    sitename: "Vue.js Pet Depot",
  },
  beforeCreate: function() {
    if (APP_LOG_LIFECYCLE_EVENTS) {
      console.log("beforeCreate");
    }
  },
  created: function() {
    if (APP_LOG_LIFECYCLE_EVENTS) {
      console.log("created");
    }
  },
  beforeMount: function() {
    if (APP_LOG_LIFECYCLE_EVENTS) {
      console.log("beforeMount");
    }
  },
  mounted: function() {
    if (APP_LOG_LIFECYCLE_EVENTS) {
      console.log("mounted");
    }
  },
  beforeUpdate: function() {
    if (APP_LOG_LIFECYCLE_EVENTS) {
      console.log("beforeUpdate");
    }
  },
  updated: function() {
    if (APP_LOG_LIFECYCLE_EVENTS) {
      console.log("updated");
    }
  },
  beforeDestroy: function() {
    if (APP_LOG_LIFECYCLE_EVENTS) {
      console.log("beforeDestroy ");
    }
  },
  destroyed: function() {
    if (APP_LOG_LIFECYCLE_EVENTS) {
      console.log("destroyed");
    }
  }
});
```

Переменная для включения и выключения функций обратного вызова

Выводит событие beforeCreate

Выводит событие created

Выводит событие beforeMount

Выводит событие mounted

Выводит событие beforeUpdate

Выводит событие updated

Выводит событие beforeDestroy

Выводит событие destroyed

ДОПОЛНИТЕЛЬНО

Хук (от hook — «крючок») — это функция, которая «цепляется» за определенный код библиотеки Vue. Когда приложение подходит к такому участку, оно вызывает эту функцию или продолжает работу, если вы ничего не указали.

Первое, что бросается в глаза в листинге 2.3, — это переменная `APP_LOG_LIFECYCLE_EVENTS`, с помощью которой включается и выключается журналирование событий жизненного цикла. Она объявлена за пределами экземпляра Vue, чтобы ее можно было использовать глобально в корневом экземпляре или любом дочернем компоненте, который будет написан позже. Кроме того, если бы мы объявили ее внутри экземпляра приложения, ее бы не существовало на момент выполнения функции обратного вызова `beforeCreate`!

ПРИМЕЧАНИЕ

Название переменной APP_LOG_LIFECYCLE_EVENTS состоит из прописных букв, что характерно для констант. Позже, когда мы перейдем на ECMA-Script 6, для объявления констант будет применяться ключевое слово const. Мы планируем наперед, чтобы впоследствии не пришлось искать и заменять имена на других участках кода.

Остальной код определяет функции, которые фиксируют все события жизненного цикла по мере их возникновения. Вернемся к консоли и посмотрим, что происходит в жизненном цикле Vue, на примере свойства `sitename`.

2.2.2. Исследование кода жизненного цикла

Открыв консоль в Chrome и перезагрузив приложение, вы должны немедленно увидеть вывод нескольких функций обратного вызова (рис. 2.10).

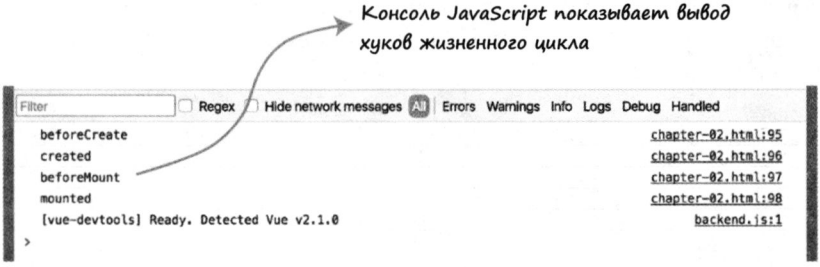


Рис. 2.10. В консоли наблюдаем вывод некоторых функций жизненного цикла

Как можно было ожидать, первые четыре хука срабатывают во время создания и монтирования приложения. Чтобы проверить другие хуки, нужно немного поработать в консоли. Сначала инициируем срабатывание функций обратного вызова, связанных с обновлением. Для этого дадим сайту новое название (рис. 2.11).

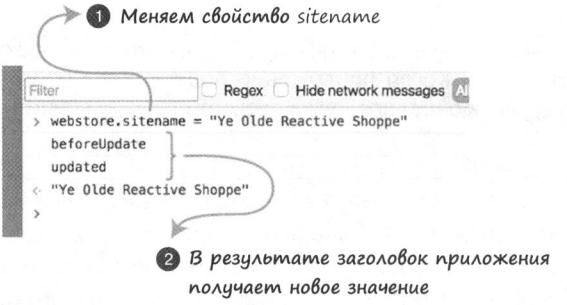


Рис. 2.11. Изменение свойства sitename приводит к срабатыванию хуков, связанных с обновлением

Меняя свойство `sitename`, вы запускаете цикл обновления, в результате которого заголовок привязывается к новому значению. Теперь уничтожим наше приложение (не волнуйтесь, оно вернется на место со следующей перезагрузкой страницы)! Воспользуемся методом экземпляра `$destroy`, чтобы запустить два последних хука жизненного цикла.

СОВЕТ

Специальные методы, которые Vue создает для нашего экземпляра, имеют префикс `$`. Подробнее о методах экземпляра для управления жизненным циклом можно почитать в справочнике API по адресу ru.vuejs.org/v2/api/#Методы-экземпляра-жизненный-цикл.

Последние два хука обычно используются для разного рода очистки в приложении или компоненте. Если мы создали экземпляр сторонней библиотеки, то должны вызвать код, ответственный за ее удаление, или вручную убрать все, что на нее ссылается, в противном случае можно столкнуться с утечками памяти, выделенной приложению. На рис. 2.12 показано, как вызов метода экземпляра `$destroy()` запускает хуки удаления.

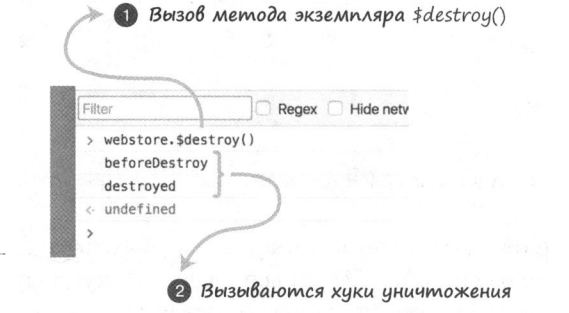


Рис. 2.12. Вызов метода экземпляра $destroy() запускает последние два хука

2.2.3. Стоит ли оставлять код жизненного цикла?

Хуки жизненного цикла позволяют понять, что происходит с приложением во время его работы, но нужно признать, что код вывода сообщений в консоль выглядит повторяющимся и многословным. Вы больше не увидите эти развесистые функции в моих листингах, но время от времени мы будем использовать хуки жизненного цикла для исследования новых возможностей или выполнения каких-то полезных действий в самом приложении.

Если вы решили оставить этот код, но не хотите загромождать консольный вывод, выключите журналирование, присвоив `false` переменной `APP_LOG_LIFECYCLE_EVENTS`. Это можно сделать в файле `index.html`, убрав весь вывод, или динамически, меняя значение во время выполнения или в консоли JavaScript.

2.3. Вывод товара

Отображение названия магазина было хорошим началом, но прежде, чем двигаться дальше, необходимо рассмотреть еще несколько аспектов вывода данных в разметке. Товар будет выводиться несколькими способами: в составе списка и таблицы, в качестве рекомендуемого продукта и на отдельной странице. При проектировании всех этих представлений и создании разметки для них станем использовать одни и те же данные, манипулируя ими с помощью Vue. При этом их исходная структура и значения будут оставаться прежними.

2.3.1. Определение данных товара

Пока мы будем выводить всего один товар. Добавим его в наш объект с данными (листинг 2.4).

Листинг 2.4. Добавление информации о товаре в экземпляр Vue: chapter-02/product-data.js

```
data: {
  sitename: "Vue.js Pet Depot",
  product: {
    id: 1001,
    title: "Cat Food, 25lb bag",
    description: "A 25 pound bag of <em>irresistible</em>,"+
                 "organic goodness for your cat.",
    price: 2000,
    image: "assets/images/product-fullsize.png"
  }
},
```

Объект с информацией о товаре

Атрибуты товара представлены в виде свойств объекта product

Добавление объекта `product` в опцию `data` выглядит довольно просто.

- *Для однозначной идентификации товара используется свойство id.* Оно будет инкрементироваться при добавлении новых товаров.

- *Свойства title и description — строки, но последнее содержит HTML-разметку.* Подробнее об этом поговорим позже, когда дойдем до отображения всех этих свойств в шаблоне товара.
- *Свойство price содержит число и представляет стоимость товара.* Это упростит дальнейшие вычисления и позволит избежать потенциально опасного приведения типов, которое возникает, когда значения хранятся в базе данных в виде вещественных чисел или строк.
- *Свойство image указывает путь к основному изображению товара.* Не смущайтесь при виде фиксированного пути. Мы еще не раз вернемся к этому аспекту и рассмотрим более подходящие альтернативы.

Итак, закончив с данными, займемся представлением.

2.3.2. Разметка представления товара

Теперь сосредоточимся на разметке нашего HTML-документа. Добавим под заголовком элемент `main`, который послужит контейнером для данных приложения. Это новый элемент, он появился в HTML5 и предназначен для хранения основного содержимого веб-страницы или программы.

ДОПОЛНИТЕЛЬНО

Подробнее о main и других элементах можно почитать на странице www.quackit.com/html_5/tags/html_main_tag.cfm.

Разметка товара состоит из двух столбцов, чтобы изображение находилось сбоку от описания (рис. 2.13). В файле `chapter-02/assets/css/app.css` для этого уже предусмотрены все нужные стили, поэтому остается лишь добавить в разметку имя подходящего класса (листинг 2.5).

Листинг 2.5. Добавление разметки товара: chapter-02/product-markup.html

```
<main>
  <div class="row product">
    <div class="col">
      <figure>
        <img v-bind:src="product.image">
      </figure>
    </div>
    <div class="col col-expand">
      <h1 v-text="product.title"></h1>
      <p v-text="product.description"></p>
      <p v-text="product.price" class="price"></p>
    </div>
  </div>
</main>
```

Путь к изображению товара привязан к атрибуту src элемента img с помощью директивы v-bind

Другие свойства товара выводятся с использованием директивы v-text

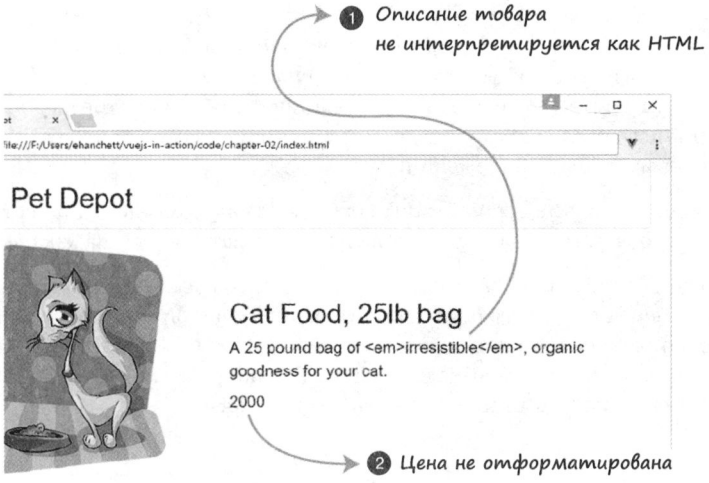


Рис. 2.13. Наш товар отображается, но кое-что следует подчистить

Первое, что бросается в глаза, — доступ к свойствам объекта в стиле JavaScript при связывании данных. Мы должны указывать полный путь к свойствам `product`. Большинство свойств (`title`, `description` и `price`) связываются с помощью директив `v-text`, как это делалось с `sitename` в заголовке.

Для пути изображения товара используется *связывание по атрибуту* с помощью директивы `v-bind`, поскольку атрибуты элемента нельзя привязывать путем простой интерполяции текста. Директива `v-bind` поддерживает любые корректные атрибуты, но для стилей, имен классов и некоторых других случаев предусмотрены особые правила. Подробнее об этом — в следующих главах.

ПРИМЕЧАНИЕ

Директива v-bind имеет сокращенную версию. Каждый раз, когда нужно ее задействовать, просто подставляйте двоеточие (:). Таким образом, вместо v-bind:src=" ... " можно написать :src=" ... ".

Использование выражений при связывании

Vue позволяет связывать не только свойства данных, но и любые корректные выражения на языке JavaScript. Вот несколько примеров того, как этим можно было бы воспользоваться в листинге 2.5:

```
{{ product.title.toUpperCase() }} -> CAT FOOD, 25LB BAG
{{ product.title.substr(4,4) }} -> Food
{{ product.price - (product.price * 0.25) }} -> 1500
<img :src="product.image.replace('.png', '.jpg')"> ->
<img src=" //assets/images/product-fullsize.png">
```

Может оказаться удобно так применять выражения, но тем самым в представление привносится логика, которую почти всегда лучше хранить в коде приложения или компонента, ответственного за данные представления. Кроме того, из-за подобных выражений бывает сложно понять, где именно изменяются данные. Это особенно актуально для сложных приложений.

В целом использование строчных выражений очень помогает при тестировании, прежде чем соответствующая функциональность будет оформлена как следует.

В следующем разделе и дальнейших главах будут представлены рекомендованные методики изменения, фильтрации и наследования существующих значений, которые не сказываются на целостности представлений или данных приложения. Подробнее о том, что именно считается выражением, можно почитать по адресу ru.vuejs.org/v2/guide/syntax.html#Использование-выражений-JavaScript.

Теперь перейдем в Chrome, перезагрузим страницу и убедимся в том, что страница товара отображается так, как было задумано.

Мы видим несколько нюансов, которые стоит подправить.

1. Описание товара выводится в виде строки без интерпретации HTML-разметки.
2. Цена товара показана в виде строкового представления целого числа `2000` и не отформатирована как следует.

Сначала устраним первый недочет. Нам нужно выводить `HTML`, поэтому подставим в разметку товара директиву `v-html` (листинг 2.6).

Листинг 2.6. Обновление разметки товара: chapter-02/product-markup-cont.html

```
<main>
  <div class="row product">
    <div class="col">
      <figure>
        <img v-bind:src="product.image">
      </figure>
    </div>
    <div class="col col-expand">
      <h1 v-text="product.title"></h1>
      <p v-html="product.description"></p>
      <p v-text="product.price" class="price"></p>
    </div>
  </div>
</main>
```

Директива v-html выводит описание товара в виде HTML, а не как простой текст

После перезагрузки приложения в Chrome значение с описанием товара должно выводиться в виде HTML. Таким образом, слово irresistible будет выделено курсивом (рис. 2.14).

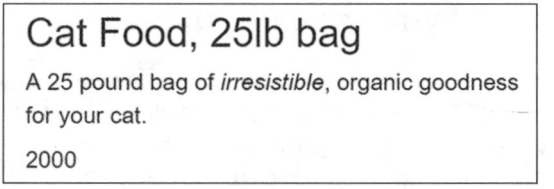


Рис. 2.14. Использование директивы v-html позволяет выводить описание в виде HTML

Директива `v-html` отобразит привязанное свойство как обычную HTML-разметку. Это удобное средство, но не стоит им злоупотреблять. Задействуйте его только для значений, которым доверяете. Теперь следует поправить вывод цены.

Межсайтовый скриптинг

Вставляя HTML непосредственно в представление, мы делаем приложения уязвимыми к *XSS-атакам* (cross-site scripting).

В общем случае, если злоумышленнику удастся сохранить в нашей базе данных вредоносный код на JavaScript, мы станем уязвимыми при выводе этого кода на HTML-странице.

Работая с содержимым и разметкой, нужно придерживаться элементарных мер предосторожности.

- Интерполяции HTML-кода следует подвергать только содержимое, в котором вы уверены.
- Никогда не используйте интерполяцию HTML-кода при выводе содержимого, сгенерированного пользователями, независимо от тщательности проверки.
- В случае крайней необходимости пытайтесь реализовать функцию в виде компонента с собственным шаблоном, не позволяя передавать HTML-разметку через поля ввода.

В статье на сайте excess-xss.com приводится понятный и всесторонний обзор XSS, а углубленную информацию о подобного рода атаках и примеры кода для каждого эксплойта можно найти в вики OWASP: www.owasp.org/index.php/Cross-site_Scripting_(XSS).

2.4. Применение фильтров вывода

Последнее, что нам осталось, — вывести цену товара в знакомом формате, а не просто как целое число. Фильтры вывода позволяют отформатировать значение, перед тем как отображать его на странице. Общий синтаксис фильтра вывода выглядит так: `{{ свойство | фильтр }}`. В нашем примере нужно отображать значение `2000` как `$20.00`.

2.4.1. Написание функции фильтра

Фильтр вывода — это функция, которая принимает значение, выполняет форматирование и возвращает измененный результат. Если речь идет об интерполяции текста, в фильтр передается свойство, к которому мы привязываемся.

Все фильтры вывода находятся в объекте `filters`, который является одной из опций, передаваемых в экземпляр Vue. Именно там создадим функцию форматирования, представленную в листинге 2.7.

Листинг 2.7. Добавление фильтра formatPrice: chapter-02/format-price.js

```
var webstore = new Vue({
  el: '#app',
  data: { … },
  filters: {
    formatPrice: function(price) {
      if (!parseInt(price)) { return ""; }
      if (price > 99999) {
        var priceString = (price / 100).toFixed(2);
        var priceArray = priceString.split("").reverse();
        var index = 3;
        while (priceArray.length > index + 3) {
          priceArray.splice(index+3, 0, ",");
          index += 4;
        }
        return "$" + priceArray.reverse().join("");
      } else {
        return "$" + (price / 100).toFixed(2);
      }
    }
  }
});
```

Фильтры вывода указаны в параметре filters

formatPrice принимает целое число и возвращает его в виде цены

Форматирует значения от $1000 и выше

Если не удается извлечь целое число, немедленно выходим

Преобразует значение в десятичное

Вставляет запятые через каждых три символа

Возвращает отформатированное значение

Если число меньше $1000, возвращает отформатированное значение с десятичным разделителем

Функция `formatPrice` принимает целое число и возвращает строку, которая выглядит как цена в долларах, например $12 345,67. Функция разветвляется в зависимости от величины переданного значения.

- Если ввод больше 99 999 (эквивалент $999,99), нужно будет вставлять пробелы через каждые три символа слева от запятой.
- В противном случае ввод можно преобразовать с помощью метода `.toFixed`, поскольку пробелы вставлять не нужно.

ПРИМЕЧАНИЕ

Вы можете найти множество других способов форматирования значений в долларах, которые будут более эффективными, лаконичными и т. д. В этом примере я пожертвовал целесообразностью ради понятности. Чтобы получить представление о том, насколько сложна это проблема и как много решений для нее придумано, почитайте статью по адресу mng.bz/qusZ.

2.4.2. Добавление фильтра в разметку и проверка разных значений

Чтобы воспользоваться новым блестящим фильтром, мы должны добавить его к привязке для цены товара. Нам нужно также привести привязку к синтаксису Mustache, как показано в листинге 2.8. Фильтры нельзя задействовать в сочетании с директивой `v-text`.

Листинг 2.8. Добавление разметки товара: chapter-02/v-text-binding.html

```
<main>
  <div class="row product">
    <div class="col">
      <figure>
        <img v-bind:src="product.image">
      </figure>
    </div>
    <div class="col col-expand">
      <h1>{{ product.title }}</h1>
      <p v-html="product.description"></p>
      <p class="price">
        {{ product.price | formatPrice }}
      </p>
    </div>
  </div>
</main>
```

Новый фильтр форматирует значение цены товара

Как вы помните, связывание данных с фильтрацией имеет синтаксис вида `{{ свойство | фильтр }}`, поэтому привязка цены выглядит как `{{ product.price | formatPrice }}`.

Вернемся обратно в Chrome и обновим страницу. Вуаля — мы получили отформатированную цену (рис. 2.15).

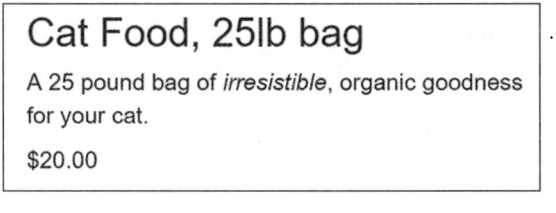


Рис. 2.15. Функция форматирования выводит значение цены с добавлением знака доллара и необходимых знаков пунктуации

Манипулируя данными в консоли, мы можем на лету подставлять разные цены для нашего фильтра. Откройте консоль и установите новое значение для свойства `product.price`, например `webstore.product.price = 150000000`.

Результат обновления цены товара показан на рис. 2.16. Попробуйте подставлять маленькие (меньше 100) и большие (больше 10 000 000) значения и убедитесь в том, что все они форматируются корректно.

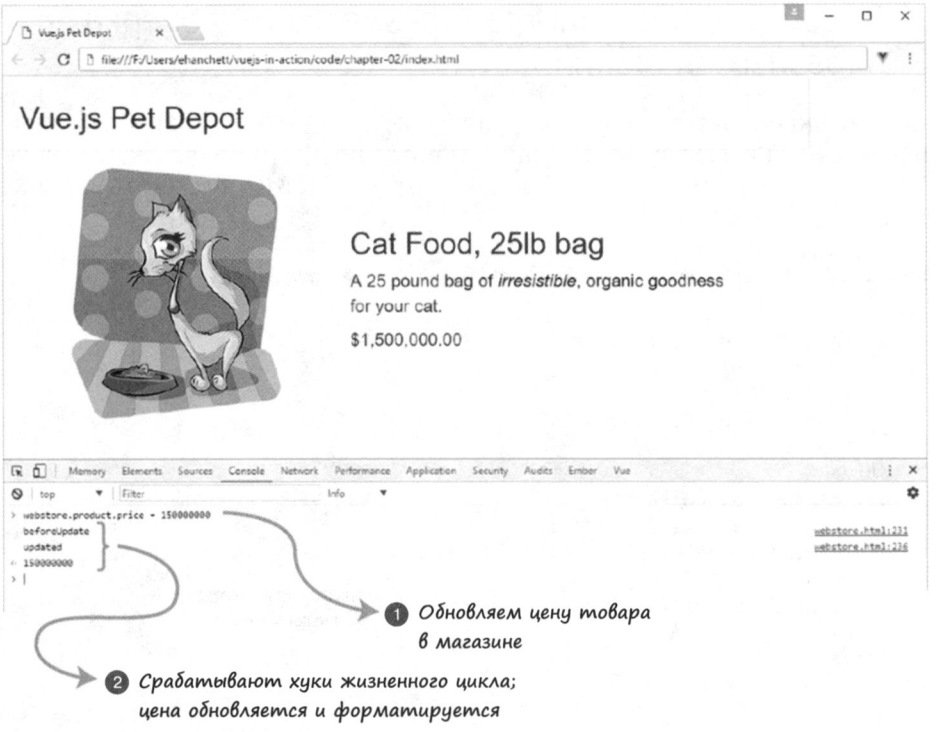


Рис. 2.16. Изменение цены товара запускает цикл событий, если они все еще включены, а обновленное значение проходит через фильтр

Упражнение

Используя знания, приобретенные в этой главе, ответьте на вопрос. В разделе 2.4 мы создали фильтр для цены. Какие еще фильтры могли бы нам пригодиться?

Ответ ищите в приложении Б.

Резюме

- Vue позволяет привнести в приложения интерактивность.
- Мы можем в любой момент вклиниться в жизненный цикл Vue, чтобы выполнить определенные функции.
- Vue.js предлагает концепцию фильтров для вывода информации определенным образом.

Часть II
Представление и модель представления

Это основная часть книги. В следующих главах мы подробно рассмотрим все элементы и составляющие Vue-приложения. Начнем с малого: сделаем нашу программу более интерактивной. Затем перейдем к формам, полям ввода, условным выражениям и циклам.

Несколько важнейших концепций представлены в главах 6 и 7. Они посвящены углубленному изучению компонентов — кирпичиков, из которых строится проект. В главе 3 мы впервые столкнемся с мощным средством из арсенала Vue.js — однофайловыми компонентами.

В последних двух главах рассмотрим переходы, анимацию и методы расширения Vue. Это позволит сделать наше приложение более эффективным и изящным.

3 Поддержка интерактивности

В этой главе

- Получение нового вывода на основе данных с помощью вычисляемых свойств.
- Добавление обработчиков событий в DOM.
- Отслеживание данных на этапе жизненного цикла Vue, отвечающего за обновления.
- Реакция на действия пользователя.
- Вывод разметки в зависимости от выполнения условий.

В это сложно поверить, но мы уже подошли к моменту, когда интернет-магазин можно сделать интерактивным.

Добавление интерактивности в приложение означает привязку к событиям DOM, их обработку в коде и возвращение пользователю результатов взаимодействия. Vue берет на себя создание привязок данных и событий и управление ими, а нам необходимо позаботиться о работе с данными в рамках приложения и о том, как воплотить ожидания пользователей в нашем интерфейсе.

Позволим посетителям добавлять наш единственный товар в корзину. В ходе работы над этим примером вы увидите, как то, что мы делаем, вписывается в общую картину Vue-приложения.

Чтобы лучше понять, что будет происходить в этой главе, взгляните на рис. 3.1 — там представлен результат, к которому мы стремимся.

3.1. Корзина покупок начинается с добавления массива

Прежде чем приступить к реализации суперкрутой корзины, мы должны создать в экземпляре приложения контейнер для хранения новых элементов. К счастью, на этом этапе достаточно простого массива (листинг 3.1), в который будем помещать товары.

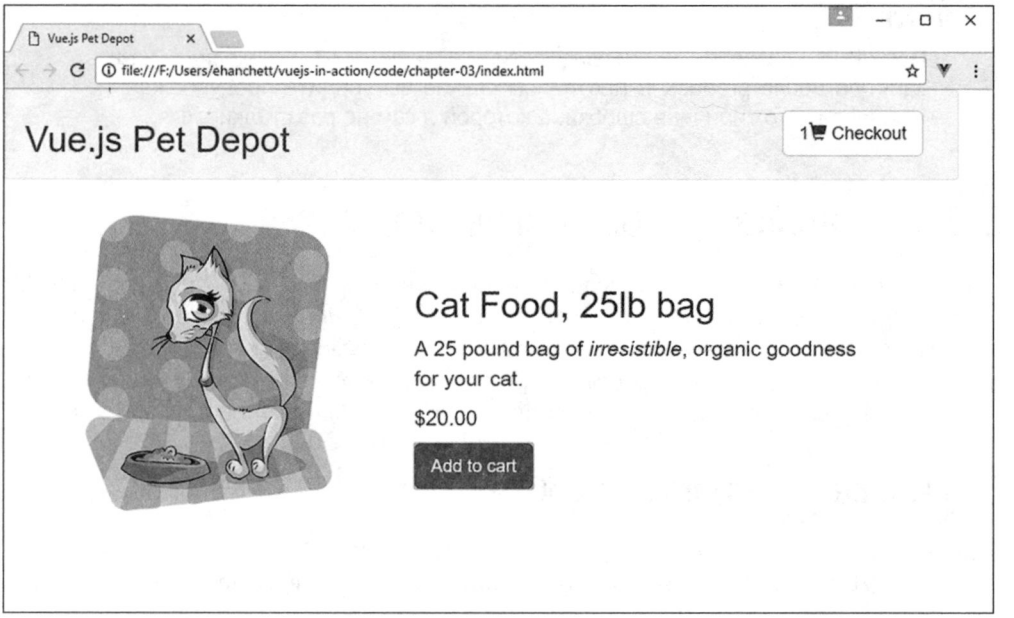


Рис. 3.1. Страница товара с новыми элементами — корзиной покупок и кнопкой добавления

Я разбил код на небольшие фрагменты по аналогии с тем, как было сделано в предыдущей главе. Все это вам нужно добавить в созданный ранее файл `index.html`. При необходимости готовый код можно загрузить из репозитория.

Листинг 3.1. Нам нужен обычный массив: chapter-03/add-array.js

```
data: {
  sitename: "Vue.js Pet Depot",
  product: {
    id: 1001,
    title: "Cat Food, 25lb bag",
    description: "A 25 pound bag of <em>irresistible</em>,
                  organic goodness for your cat.",
    price: 2000,
    image: "assets/images/product-fullsize.png",
  },
  cart: []
},
```

Оставим для наглядности данные о существующем товаре

Массив для хранения товаров в корзине

Вот и готова корзина покупок. Этот массив еще сослужит нам хорошую службу, хотя в какой-то момент мы заменим его отдельным компонентом, инкапсулирующим управление корзиной.

ПОМНИТЕ

Прежде чем добавлять массив для корзины в листинге 3.1, необходимо указать запятую после product. В противном случае вы увидите предупреждение в консоли. Это типичная ошибка, с которой я сам не раз сталкивался.

3.2. Привязка к событиям DOM

Чтобы сделать приложение интерактивным, нужно как-то связать элементы DOM с функциями, определенными в экземпляре Vue. *Привязка событий* позволяет связать элемент с любым стандартным событием документа (`click`, `mouseup`, `keyup` и т. д.). Vue берет на себя всю работу по регистрации обработчиков, чтобы вы могли сосредоточиться на том, как приложение реагирует на действия пользователя.

3.2.1. Основы привязки событий

Для связывания кода на JavaScript с элементом DOM применяется директива `v-on` (рис. 3.2). Привязанный код или функция выполняется в момент срабатывания события.

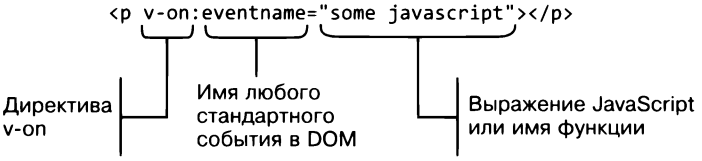


Рис. 3.2. Синтаксис привязки событий

Связывание кода с событиями может производиться двумя способами.

1. *Можно привязать к событию имя функции, определенной в экземпляре.* Привязка вида `v-on:click="clickHappened"` означает, что при щелчке на элементе будет вызвана функция `clickHappened`.
2. *Можно написать выражение JavaScript для работы с одним из доступных свойств.* В этом случае привязка может выглядеть как `v-on:keyup="charactersRemaining -= 1"` (уменьшение свойства `charactersRemaining` на 1).

Оба подхода имеют право на жизнь, но сначала мы рассмотрим обработку событий с помощью функций.

ПРИМЕЧАНИЕ

Существует более короткое обозначение привязки событий. Вместо v-on можно применять символ @, например: v-on:click="..." заменить на @click="...". Станем использовать это сокращение в последующих главах.

3.2.2. Привязывание события к кнопке добавления в корзину

Чтобы добавлять товары в корзину покупок, посетителям нужна отдельная кнопка. С помощью Vue привяжем событие нажатия этой кнопки к функции, которая помещает товар в массив `cart`.

Прежде чем создавать кнопку в разметке, следует написать соответствующую функцию. Для этого нужно добавить объект `methods` в число опций приложения. Вставьте после объекта `filters` следующий код (листинг 3.2) (не забудьте разделить эти объекты запятой!).

Листинг 3.2. Метод addToCart: chapter-03/add-to-cart.js

```
methods: {
  addToCart: function() {
    this.cart.push( this.product.id );
  }
}
```

Определение функции addToCart

Объект methods содержит новую функцию

Пока добавление товара в корзину будет означать помещение его свойства `id` в массив `cart`. Не забывайте использовать ключевое слово `this` для доступа к любым свойствам данных.

Добавляется идентификатор, а не сам объект

В листинге 3.2 было бы проще добавлять в массив `cart` целый объект, например `this.cart.push( this.product );`. Но это не очень элегантный способ. В языке JavaScript аргументы могут передаваться как по ссылке, так и по значению, и чтобы понимать, какой именно вариант передачи используется в той или иной ситуации, требуется определенный опыт.

При добавлении товара в массив `cart` попадала бы не его копия, а ссылка на его объект. Если товар изменится, например, в результате получения новых данных с сервера, содержимое корзины тоже может поменяться или же ссылка станет неопределенной (`undefined`).

Вместо этого мы помещаем в массив `cart` копию идентификатора, а не сам товар. Таким образом, изменение товара не повлияет на значения, хранимые в корзине.

Строго говоря, в языке JavaScript используется концепция *вызовов с разделением ресурсов* (call-by-sharing). Краткое описание этого подхода и его сравнение с другими стратегиями можно найти в «Википедии» на странице ru.wikipedia.org/wiki/Стратегия_вычисления#Вызов_по_соиспользованию_(call_by_sharing).

Итак, у нас есть функция, которая добавляет товары в корзину. Переходим к кнопке. Она будет стоять сразу под ценой внутри элемента `div` с описанием товара (листинг 3.3).

Листинг 3.3. Кнопка для добавления товаров в корзину: chapter-03/button-product.js

```
<button class="default"
  v-on:click="addToCart">
  Add to cart
</button>
```

Привязывает нажатие кнопки к функции addToCart

Кнопка добавления в корзину

Теперь, когда посетитель нажмет эту кнопку, будет вызвана функция `addToCart`. Посмотрим, как это работает.

Перейдите в Chrome, убедитесь в том, что у вас открыта консоль, и откройте вкладку Vue, чтобы видеть, какие данные добавляются в корзину покупок. Массив `cart` должен быть пустым, поэтому, если вы не видите обозначения `Array[0]`, как на рис. 3.3, обновите страницу.

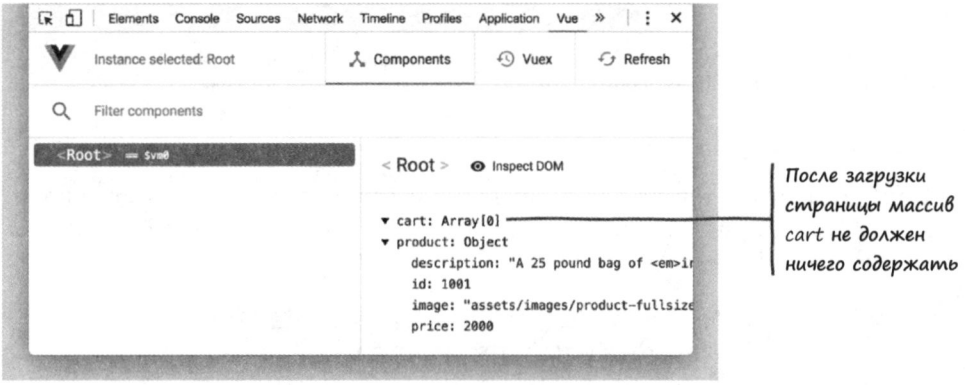


Рис. 3.3. Перед добавлением товаров массив должен быть пустым. В противном случае следует обновить страницу

Теперь несколько раз нажмите кнопку добавления. Откройте панель vue-devtools и щелкните на элементе `<Root>`. С каждым нажатием в массиве должен появляться еще один идентификатор (рис. 3.4).

Мы, как разработчики, можем видеть количество элементов в корзине покупок, используя vue-devtools или консоль. Но чтобы донести эту информацию до посетителей, нужно обновлять представление. Добавим счетчик товаров.

3.3. Кнопка для подсчета и вывода элементов в корзине

Для вывода количества элементов в корзине посетителя задействуем *вычисляемое свойство*. Как и любое другое свойство, определенное в нашем экземпляре, его можно привязать к DOM, но его главное назначение — выводить новую информацию на основании состояния приложения. Мы также добавим кнопку для отображения содержимого корзины.

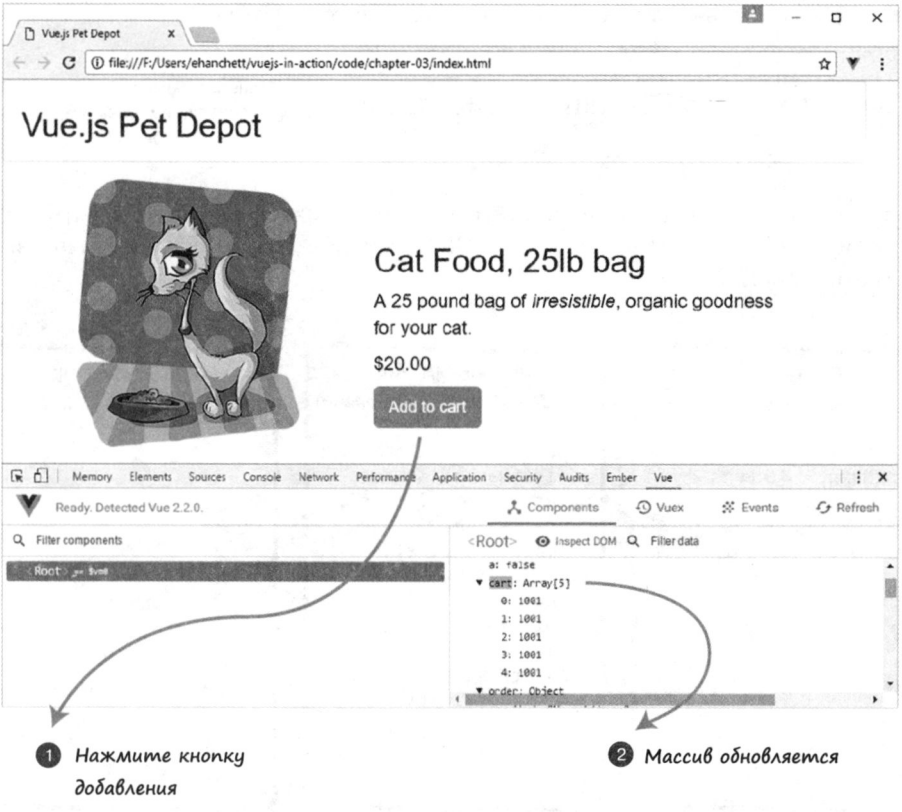


Рис. 3.4. Массив наполняется по мере добавления элементов в корзину

Но сначала поближе познакомимся с вычисляемыми свойствами и посмотрим, как они работают.

3.3.1. Когда следует использовать вычисляемые свойства

Можете воспринимать свойства в объекте `data` как информацию, которая могла бы храниться в базе данных, а вычисляемые свойства — как динамические значения, применяемые в основном в контексте представления. Такой подход кажется слишком обобщенным, но он довольно практичен.

Возьмем в качестве примера типичное вычисляемое свойство, которое выводит полное имя пользователя (листинг 3.4). Имя и фамилию имеет смысл хранить в отдельных полях базы данных, но выделять еще одно поле для полного имени — это лишнее, так как чревато ошибками. Объединение существующих полей с именем и фамилией — идеальный пример применения вычисляемого свойства.

Листинг 3.4. Вычисление полного имени пользователя: chapter-03/computed.js

```
computed: {
  fullName: function() {
    return [this.firstName, this.lastName].join('');
  }
}
```

fullName возвращает имя и фамилию пользователя, разделенные пробелом

Результат, возвращаемый функцией `fullName`, по своей сути эквивалентен одноименному свойству в объекте данных. Это означает, что его можно легко привязать к разметке (рис. 3.5).

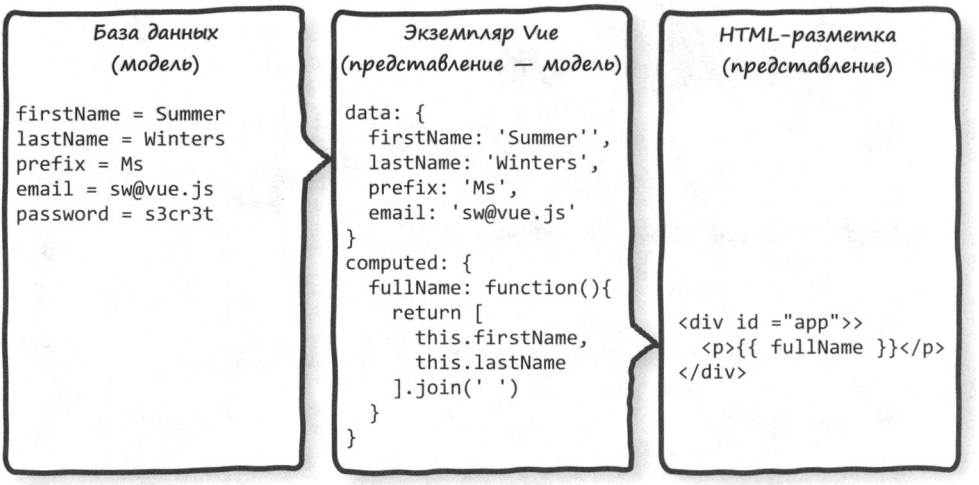


Рис. 3.5. Объединение имени и фамилии пользователя в полное имя, готовое к выводу

Дополнительное преимущество вычисляемых свойств заключается в том, что можно изменить содержимое функции, применяя другие или дополнительные данные. Например, на рис. 3.5 мы могли бы добавить свойство `prefix`, чтобы сделать полное имя пользователя более формальным.

Такое применение вычисляемых свойств позволяет сочетать и изменять любые данные экземпляра, не влияя на работу сервера или базы данных.

3.3.2. Проверка событий обновления с помощью вычисляемых свойств

Поскольку вычисляемые свойства обычно основаны на данных экземпляра, значения, которые они возвращают, автоматически обновляются при изменении исходной информации. Таким образом, любая разметка, привязанная к вычисляемому свойству, всегда отражает актуальное значение.

Такое поведение — неотъемлемая часть цикла обновлений в рамках жизненного цикла экземпляра Vue. Чтобы лучше понять, как обновляются данные, рассмотрим еще одну ситуацию, в которой вычисляемое свойство будет идеальным выбором. Представьте, что нужно определить площадь прямоугольника, зная его длину и ширину (листинг 3.5).

Изначально вычисляемое свойство `area` равно `15`. Любое последующее изменение длины или ширины повлечет за собой последовательность обновлений:

1) при изменении длины или ширины...
2) ...свойство `area` вычисляется заново...
3) ...после чего обновляется вся разметка, привязанная к этим свойствам.

Листинг 3.5. Вычисление площади прямоугольника: chapter-03/computed-rect.js

```
new Vue({
  data: {
    length: 5,
    width: 3
  },
  computed: {
    area: function() {
      return this.length * this.width;
    }
  }
});
```

Объект данных со свойствами legth и width

Вычисляемое свойство, возвращающее значение площади, эквивалентно свойствам объекта данных

Цикл обновления приложения показан на рис. 3.6. Отслеживание изменений данных экземпляра выполняется с помощью *методов-наблюдателей* и хука жизненного цикла `beforeUpdate`, который срабатывает по завершении обновления.

ДОПОЛНИТЕЛЬНО

Функция-наблюдатель работает почти так же, как хуки жизненного цикла, но срабатывает при обновлении данных, за которыми наблюдает. Таким образом, можно отслеживать даже вычисляемые свойства.

В листинге 3.6 вычисление площади показано в контексте всего приложения. Мы видим три метода-наблюдателя, которые выводят сообщения в консоль при изменении свойств `length`, `width` или `area`. Там же находится функция, фиксирующая начало цикла обновления. Эти функции должны находиться в опции `watch` экземпляра Vue, иначе они не будут работать.

СОВЕТ

Код листинга 3.6 вы найдете среди примеров для этой главы (файл chapter-03/area.html). Это полноценное приложение, поэтому можете сразу открывать его в Chrome.

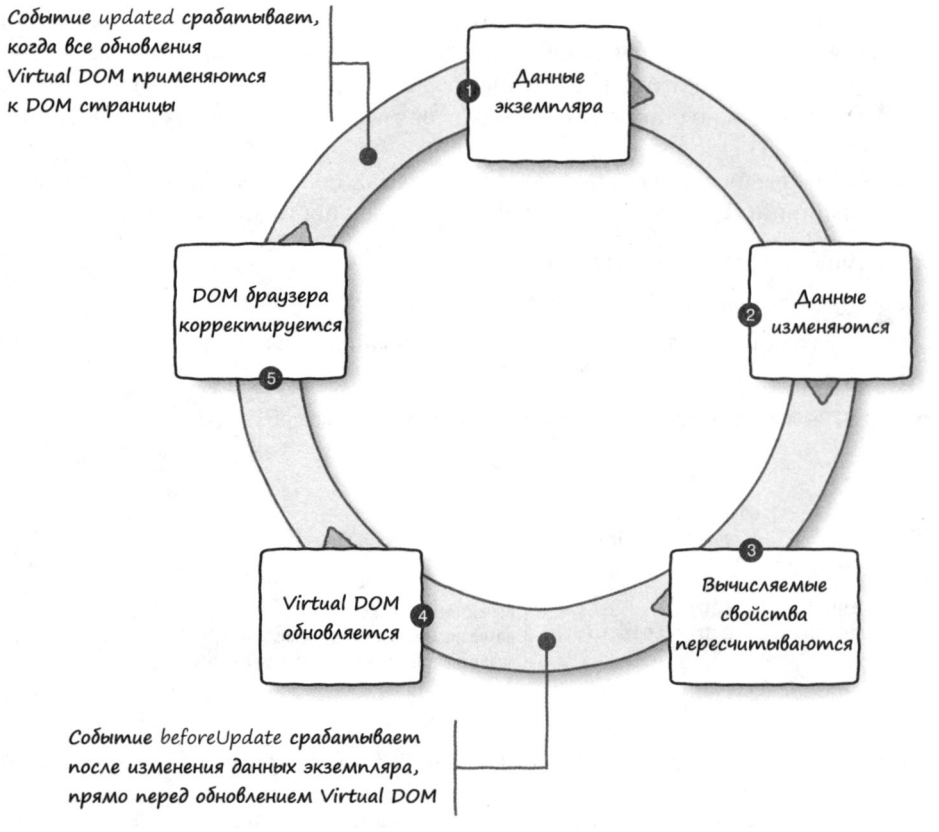


Рис. 3.6. Изменение данных экземпляра запускает цепочку событий в цикле обновления программы

Листинг 3.6. Вычисляемые свойства и фиксация события обновления: chapter-03/area.html

```
<html>
<head>
  <title>Calculating Area - Vue.js in Action</title>
  <script src="https://unpkg.com/vue/dist/vue.js"
    type="text/javascript"></script>
</head>
<body>
  <div id="app">
    <p>
      Area is equal to: {{ area }}
    </p>
    <p>
      <button v-on:click="length += 1">Add length</button>
      <button v-on:click="width += 1">Add width</button>
```

Привязка, отображающая значение площади

Кнопки, увеличивающие длину или ширину на 1

```
    </p>
  </div>
  <script type="text/javascript">
    var app = new Vue({
      el: '#app',
      data: {
        length: 5,
        width: 3
      },
      computed: {
        area: function() {
          return this.width * this.length;
        }
      },
      watch: {
        length: function(newVal, oldVal) {
          console.log('The old value of length was: '
                      + oldVal +
                      '\nThe new value of length is: '
                      + newVal);
        },
        width: function(newVal, oldVal) {
          console.log('The old value of width was: '
                      + oldVal +
                      '\nThe new value of width is: '
                      + newVal);
        },
        area: function(newVal, oldVal) {
          console.log('The old value of area was: '
                      + oldVal +
                      '\nThe new value of area is: '
                      + newVal);
        }
      },
      beforeUpdate: function() {
        console.log('All those data changes happened '
                    + 'before the output gets updated.');
      }
    });
  </script>
</body>
</html>
```

Исходные значения длины и ширины

Свойство для вычисления площади

Функция, фиксирующая изменение длины

Функция, фиксирующая изменение ширины

Функция, фиксирующая изменение площади

Хук жизненного цикла beforeUpdate

При открытии этого файла в Chrome начальное значение `area` будет равно `15` (рис. 3.7). Убедитесь в том, что открыта консоль JavaScript, и попробуйте нажать кнопку, чтобы запустить цикл обновления. При нажатии кнопок Add length (Увеличить длину) и Add width (Увеличить ширину) в консоли должны выводиться сообщения о данных приложения (рис. 3.8).

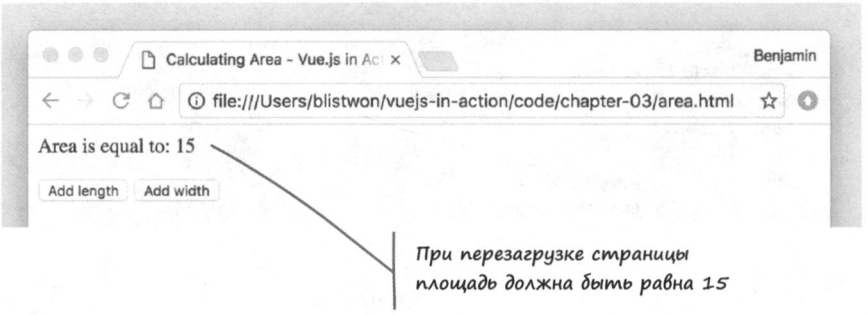


Рис. 3.7. Исходное состояние приложения для вычисления площади

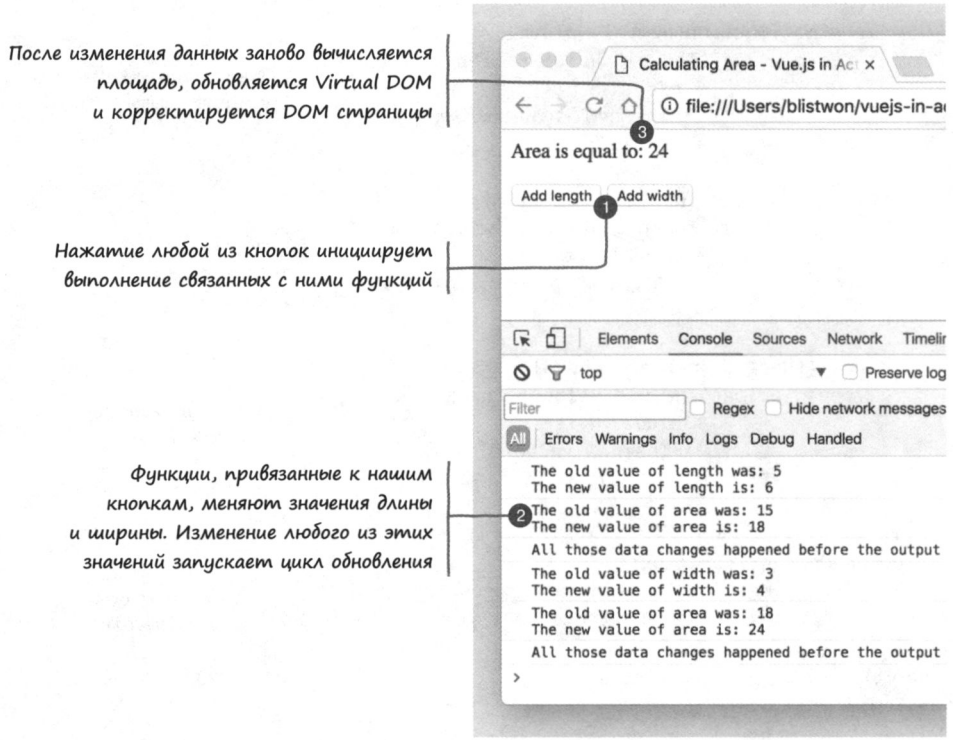


Рис. 3.8. Отслеживание изменений свойств, вызванных нажатиями кнопок

Увидев, как ведет себя приложение, мы можем разместить данные и функции из листинга 3.6 на схеме цикла обновления (рис. 3.9).

В завершение следует отметить, что с удалением привязки `{{ area }}` из кода примера и обновлением страницы поменяется и вывод в консоли, связанный с нажатием любой из кнопок (рис. 3.10).

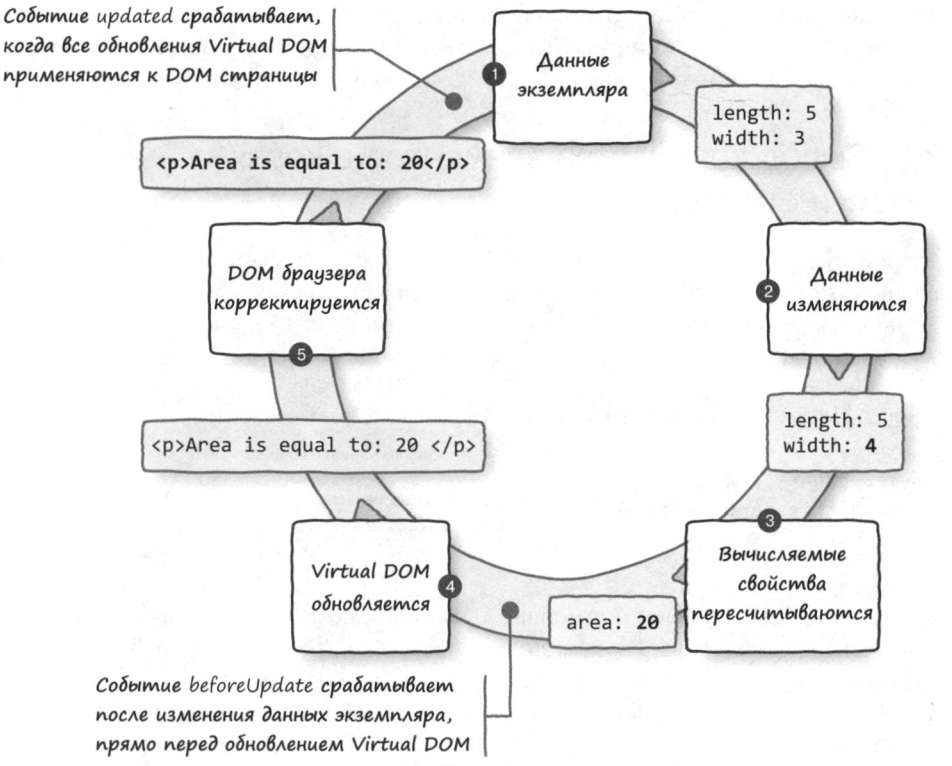


Рис. 3.9. Изменения данных экземпляра запускают последовательность событий в рамках цикла обновления приложения

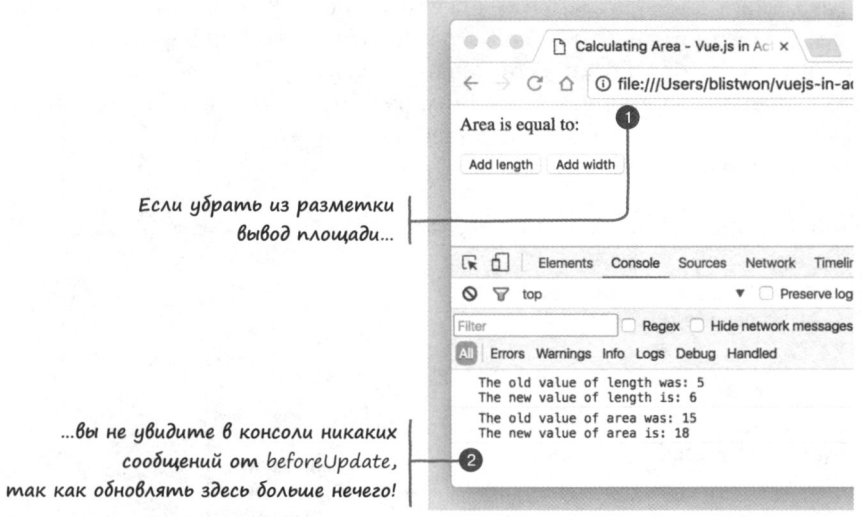


Рис. 3.10. Если ничего не обновляется, сообщение об обновлении не выводится

Если вычисляемое свойство не отображается, ему нечего обновлять, следовательно, у него больше нет причин входить в жизненный цикл. Функция `beforeUpdate` не будет выполнена, поэтому мы не увидим в консоли соответствующего сообщения.

3.3.3. Вывод количества элементов в корзине и тестирование

Получив представление о вычисляемых свойствах, можем вернуться к примеру с корзиной покупок. Добавим в экземпляр Vue вычисляемое свойство, которое будет выводить количество элементов в корзине (листинг 3.7). Не забудьте добавить объект `computed` в число опций, чтобы появилось место для размещения функции.

Листинг 3.7. Вычисляемое свойство cartItemCount: chapter-03/cart-item-count.js

```
computed: {
  cartItemCount: function() {
    return this.cart.length || '';
  }
},
```

Добавляем объект computed

Возвращаем количество элементов в массиве cart

Это простейший пример применения вычисляемого свойства. Мы используем реальное свойство массива, `length`, для извлечения количества элементов, поскольку в создании собственного механизма подсчета нет необходимости.

Сказанное хорошо иллюстрирует то, почему хранение подобного рода информации в объекте `data` — плохая идея: значение `cartItemCount` генерируется при взаимодействии с пользователем, а не берется из базы данных.

Стоит отметить, что в некоторых ситуациях количество элементов сохраняется и в объекте `data`. Например, если пользователь просматривает страницу с предыдущими заказами, у каждого элемента может быть порядковый номер. Это не противоречит реализованному подходу, поскольку в этом случае информация приходит из базы данных после обработки и сохранения заказа.

Разобравшись с функцией, выполним разметку заголовка приложения, чтобы отображались корзина покупок и количество элементов в ней. Обновим заголовок, как показано в листинге 3.8.

Листинг 3.8. Добавление корзины: chapter-03/cart-indicator.html

```
<header>
  <div class="navbar navbar-default">
    <h1>{{ sitename }}</h1>
  </div>
  <div class="nav navbar-nav navbar-right cart">
    <span
      class="glyphicon glyphicon-shopping-cart">
      {{ cartItemCount }}</span>
  </div>
</header>
```

Разместим корзину справа

Привязка данных, отображающая вычисляемое свойство

В заголовок добавлен элемент `div`, который будет служить контейнером для корзины. Значение вычисляемого свойства выводится с помощью привязки данных `cartItemCount` и заключено в элемент `span`, который подключает стили для показа значка корзины рядом со счетчиком. Пришло время проверить все, что мы написали.

Перезагрузим приложение в Chrome. Теперь каждое нажатие кнопки Add to cart (Добавить в корзину) должно приводить к увеличению счетчика. Вы можете перепроверить правильность всех вычислений, отследив содержимое массива `cart` в консоли (рис. 3.11).

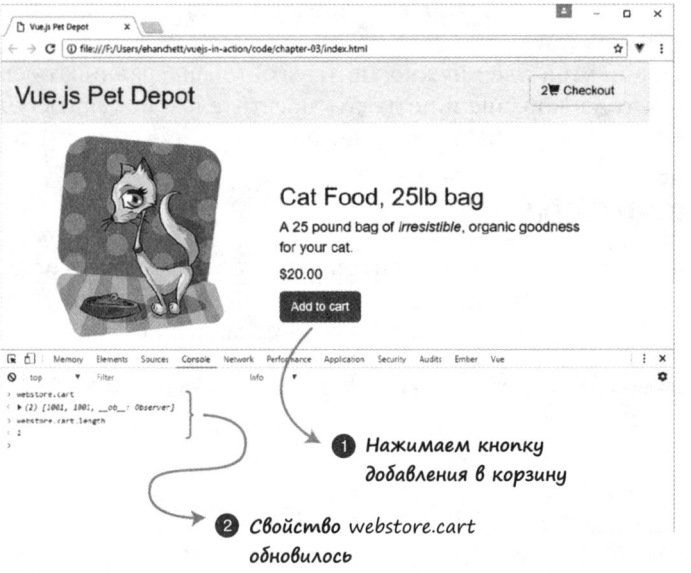


Рис. 3.11. Наблюдение за изменениями в заголовке приложения и исследование массива cart в консоли

3.4. Сделаем кнопку дружественной для пользователей

Посетители, открывающие сайт или веб-приложение, имеют определенные ожидания относительно того, как все должно быть устроено. Одно из самых фундаментальных и глубоко укоренившихся правил заключается в том, что интерактивные элементы должны вести себя предсказуемо. В противном случае продукт может показаться запутанным или неисправным. Концепция *дружественного интерфейса* основана на получении визуальных или иных сигналов и предоставлении обратной связи, делающих приложение соответствующим ожиданиям пользователя.

ДОПОЛНИТЕЛЬНО

Подробнее о дружественных интерфейсах и том, насколько важную роль они играют в приобретении опыта взаимодействия с цифровыми продуктами, можно почитать на сайте Interaction Design Foundation mng.bz/Xv96.

Итак, у нас есть кнопка, с помощью которой посетитель добавляет товар в корзину. Пока что мы никак не контролируем число продуктов, доступных для покупки. Такой подход не годится для целого ряда сценариев, например, когда количество имеющегося товара ограниченно, магазин лимитирует количество покупок, совершенных каждым пользователем, размер скидок и т. д. Если товар закончился, кнопка добавления в корзину должна стать неактивной, чтобы сигнализировать о недоступности действия.

Для реализации этой идеи нужно учитывать товар, сравнивать его количество с числом элементов в корзине и не давать посетителям покупать больше, чем мы можем продать. Для начала займемся учетом.

3.4.1. Учет товара

Чтобы не дать посетителю переусердствовать с покупками, следует добавить в объект `product` новое свойство (листинг 3.9). Оно называется `availableInventory` и хранит количество единиц товара, доступных в магазине.

Листинг 3.9. Добавление свойства availableInventory в объект product: chapter-03/available-inventory.js

```
data: {
  sitename: "Vue.js Pet Depot",
  product: {
    id: 1001
    title: "Cat Food, 25lb bag",
    description: "A 25 pound bag of <em>irresistible</em>,
                  organic goodness for your cat.",
    price: 2000,
    image: "assets/images/product-fullsize.png",
    availableInventory: 5
  }
  cart: []
}
```

Добавляем свойство availableInventory после других свойств товара

При оформлении заказа доступность товара следует перепроверить на случай, если во время транзакции другой пользователь совершил аналогичную покупку, но даже скрытия или отключения кнопки Add to cart (Добавить в корзину) будет достаточно, чтобы большинство посетителей не остались разочарованными.

ОСТОРОЖНО

Когда речь идет о финансовых или иных транзакциях, никогда не полагайтесь на значения, полученные с клиентской стороны. Серверная сторона вашего приложения должна относиться к таким данным как к пожеланиям пользователя, а не как к реальности.

3.4.2. Учет и работа с вычисляемыми свойствами

Значение `availableInventory` должно оставаться неизменным и обновляться только в процессе инвентаризации (к этому мы еще вернемся, но значительно позже). Тем не менее его следует применять при ограничении количества товара, доступного для добавления в корзину.

Нужно следить за тем, сколько элементов в корзине посетителя, и сравнивать это число с количеством имеющихся единиц товара. Чтобы выполнять подсчет на лету, по мере добавления элементов в корзину, воспользуемся вычисляемым свойством (листинг 3.10).

Листинг 3.10. Вычисляемое свойство для оставшегося товара: chapter-03/computed-remaining.js

```
computed: {
  cartItemCount: function() {
    return this.cart.length || '';
  },
  canAddToCart: function() {
    return this.product.availableInventory > this.cartItemCount;
  }
}
```

Используем вычисляемое свойство canAddToCart

Сравниваем свойство availableInventory с количеством элементов в массиве cart

Вычисляемые свойства задействуются так же, как и обычные, поэтому `cartItemCount` можно сочетать с `canAddToCart`. Новое вычисляемое свойство проверяет, превышает ли количество товара число элементов, добавленных в корзину. Положительный ответ означает, что посетитель добавил в корзину весь доступный товар и нужно как-то предотвратить дальнейшее добавление.

Истинность в JavaScript

Как вы, наверное, знаете, определение истинности выражения в JavaScript может оказаться не совсем очевидным. Вот небольшой пример, который реально повторить прямо в консоли.

При использовании нестрогого оператора равенства (`==`) сравнение числа `1` и строки `"1"` возвращает `true`. Дело в том, что язык JavaScript пытается нам помочь и выполняет приведение типов перед сравнением. Строгий оператор равенства (`===`) ожидаемо дает результат `false`.

В функции `canAddToCart` для сравнения двух целых чисел применяется оператор `>`. Если бы у нас были какие-то сомнения относительно происхождения этих значений, мы могли бы выполнить приведение типов с помощью метода `parseInt` или как-то иначе убедиться в том, что сравниваем целые числа.

Существует множество материалов о приведении типов и операторах равенства в JavaScript, но, наверное, самый наглядный пример — это набор диаграмм, находящийся по адресу dorey.github.io/JavaScript-Equality-Table. Обязательно сравните вкладки `==` и `===`.

3.4.3. Знакомство с директивой v-show

Мы научились определять, может посетитель добавлять новые элементы в корзину или нет. Теперь сделаем так, чтобы интерфейс реагировал соответствующим образом. Директива `v-show` отображает разметку только в случае, если указанное выражение возвращает `true`. Благодаря ей кнопка будет исчезать со страницы, если свойство `canAddToCart` равно `false` (листинг 3.11).

Листинг 3.11. Кнопка с директивой v-show: chapter-03/button-v-show.html

```
<button class="default"
    v-on:click="addToCart"
    v-show="canAddToCart"
    >Add to cart</button>
```

Директива v-show привязана к вычисляемому свойству canAddToCart

Если перезагрузить приложение в Chrome и попытаться добавить в корзину покупок шесть единиц товара, после пятого щелчка, то есть при достижении значения `availableInventory`, кнопка должна исчезнуть (рис. 3.12).

Рис. 3.12. Кнопка добавления в корзину исчезает при исчерпании доступного товара

`v-show` немного отличается от других директив, с которыми мы сталкивались прежде. Если выражение возвращает `false`, Vue присваивает CSS-свойству элемента `display` значение `none` с помощью встроенного стиля. Это, в сущности, убирает элемент и его содержимое из представления, хотя он по-прежнему присутствует в DOM. Если позже результат выражения поменяется на `true`, встроенный стиль будет убран и элемент снова станет виден пользователю.

ПРИМЕЧАНИЕ

У этого подхода есть одно побочное действие: любое встроенное свойство display, объявленное ранее, будет перезаписано. Но не волнуйтесь, Vue восстановит оригинальное значение при удалении собственного стиля display:none. Тем не менее вместо встроенных стилей лучше использовать определения классов.

Необходимо также иметь в виду, что директиву `v-show` стоит привязывать к единому контейнеру, а не к нескольким соседним элементам (листинг 3.12).

Листинг 3.12. Завертывание содержимого для v-show: chapter-03/wrap-content.html

```
// Избегайте этого
<p v-show="showMe">Some text</p>
<p v-show="showMe">Some more text</p>
<p v-show="showMe">Even more text</p>
```

Старайтесь не применять директиву v-show к соседним элементам

```
// Предпочтительный вариант
<div v-show="showMe">
  <p>Some text</p>
  <p>Some more text</p>
  <p>Even more text</p>
</div>
```

Вместо этого помещайте соседние элементы в контейнер и используйте единую директиву v-show

Следует уточнить: вы можете свободно использовать директиву `v-show` в любой части своего приложения. Но элементы, которые имеют реактивную связь с данными, стоит объединять: это не только улучшит производительность, но и упростит поддержание разметки в актуальном состоянии. Скрытие кнопки добавления в корзину при исчерпании товара, безусловно, работает. Но давайте попробуем менее радикальный подход.

3.4.4. Отображение и скрытие кнопки с помощью v-if и v-else

Действительно, скрытие кнопки добавления в корзину не даст пользователю купить слишком много единиц товара. Но это не самый изящный способ. Мы можем просто сделать кнопку неактивной — это будет более информативно, не так сильно нарушит целостность интерфейса и сохранит последовательность элементов в документе.

Директивы `v-if` и `v-else` задействуются для отображения одного из двух вариантов в зависимости от истинности заданного выражения. Как и в предыдущем примере, в качестве условия будет применяться свойство `canAddToCart`.

Принцип работы директивы `v-if` показан на рис. 3.13. Если `canAddToCart` равно `true`, кнопка отображается, а если не равно — скрывается.

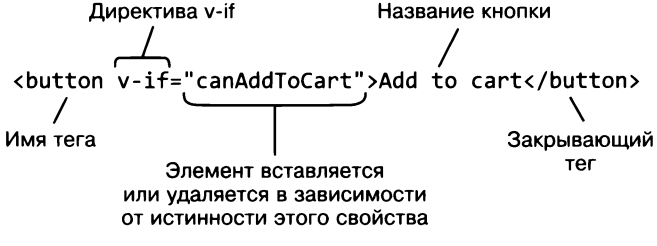


Рис. 3.13. Принцип работы условия директивы v-if

В листинге 3.13 вы видите, как работают директивы `v-if` и `v-else`.

Листинг 3.13. Кнопки с директивами v-if и v-else: chapter-03/v-if-and-v-else.html

```
<button class="default"
    v-on:click="addToCart"
    v-if="canAddToCart"
    >Add to cart</button>

<button class="disabled"
    v-else
    >Add to cart</button>
```

Эта кнопка отображается, когда canAddToCart равно true

Эта кнопка отображается, когда canAddToCart равно false

При совместном применении директив `v-if` и `v-else` в разметке должны быть два элемента, которые поочередно отображаются в зависимости от истинности условия. Кроме того, эти элементы должны идти непосредственно один за другим, иначе Vue не сможет их правильно связать.

В листинге 3.13 используются две кнопки:

- если `canAddToCart` равно `true`, отображаем исходную кнопку, привязанную к событию `addToCart` и CSS-классу `default`;
- если `canAddToCart` равно `false`, отображаем кнопку без привязки к событию и с CSS-классом `disabled`, чтобы она выглядела соответствующим образом.

Если снова загрузить приложение в Chrome, кнопка перестанет быть активной, как только вы добавите в корзину пять единиц товара (рис. 3.14).

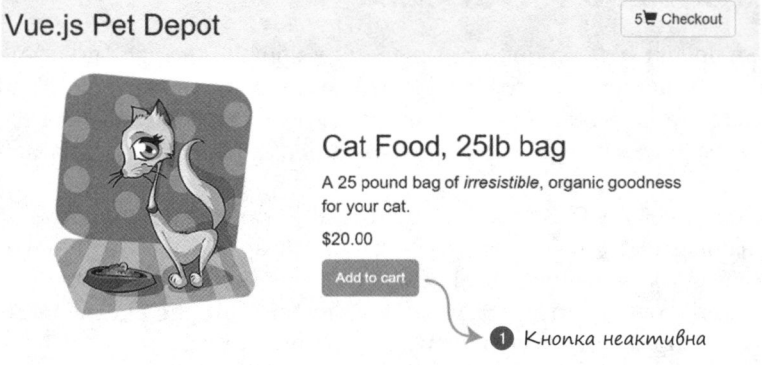


Рис. 3.14. При исчерпании товара директивы v-if и v-else позволяют делать кнопку неактивной, вместо того чтобы скрывать ее

Директивы `v-if` и `v-else` убирают из документа один из элементов в зависимости от истинности условия. Это достигается за счет единого мгновенного обновления DOM. Попробуйте поиграть в консоли со значением `availableInventory`, поглядывая на свойство `display` этих элементов.

Как и в случае с `v-show`, директивы `v-if` и `v-else` следует применять к единому контейнеру, при этом они должны идти одна за другой, как показано в листинге 3.14.

Листинг 3.14. Общие контейнеры для v-if и v-else: chapter-03/single-container.html

```
// Это не сработает
<p v-if="showMe">The if text</p>
<p>Some text related to the if text</p>
<p v-else>The else text</p>

// И это тоже
<div>
  <p v-if="showMe">The if text</p>
</div>
<div>
  <p v-else>The else text</p>
</div>

// Элементы лучше группировать
<div v-if="showMe">
  <p>The if text</p>
  <p>Some text related to the if text</p>
</div>
<div v-else>
  <p>The else text</p>
</div>
```

Это не сработает: v-if и v-else разделены элементом текстового абзаца

Это не сработает: элементы v-if и v-else не соседствуют в разметке

Это будет работать: заверните нужное содержимое в единый элемент и затем привяжите к нему v-if и v-else

Здесь мы стараемся поместить все элементы DOM для заданного условия в единый элемент, который выступает как контейнер. Позже рассмотрим разные стратегии изоляции элементов в условных выражениях с помощью шаблонов и компонентов, что значительно упростит разметку приложения.

3.4.5. Добавление кнопки корзины в виде переключателя

Добавим кнопку для перехода на страницу оформления заказа (листинг 3.15). Для начала создадим в приложении новые метод и свойство.

Листинг 3.15. Добавление кнопки корзины: chapter-03/cart-button.js

```
data: {
  showProduct: true,
…
},
methods: {
…
  showCheckout() {
      this.showProduct = this.showProduct ? false : true;
  },
}
```

Это свойство определяет, нужно ли отображать страницу товара

Метод showCheckout срабатывает после нажатия кнопки корзины

Тернарная операция для переключения между true и false

Переключение свойства `showProduct` приводит к отображению страницы оформления заказа. Рассмотрим этот процесс подробнее. Метод `showCheckout` переключает свойство `showProduct` с помощью конструкции, которую в JavaScript называют

тернарной операцией. Это сокращенный вариант выражения `if`, который принимает три параметра. Первый параметр — это условие, в данном случае `this.showProduct`. Если оно равно `true`, будет возвращено первое выражение, `false`. В противном случае вернется второе выражение, `true`. Тернарный оператор — удобное средство для быстрого создания условных конструкций.

Вы, наверное, заметили, что в определении функции `showCheckout()` отсутствует ключевое слово `function`. Стандарт ES6, известный также как ES2015, поддерживает сокращенный синтаксис объявления методов. Мы будем использовать его в дальнейшем.

Теперь нужно добавить в представление кнопку и привязать ее к событию `click` (листинг 3.16).

Листинг 3.16. Добавление кнопки корзины: chapter-03/add-cart-button.html

```
<div class="nav navbar-nav navbar-right cart">
    <button type="button"
      class="btn btn-default btn-lg"
      v-on:click="showCheckout">
      <span
        class="glyphicon glyphicon-shopping-cart">
        {{ cartItemCount}}</span>
      </span>
      Checkout
    </button>
</div>
```

Событие click, привязанное к кнопке, вызывает метод showCheckout

При нажатии кнопки срабатывает метод `showCheckout`, который меняет состояние свойства `showProduct` на противоположное. Это важная кнопка, поскольку нам нужно где-то разместить информацию о заказе. Подробнее об этом — в следующем разделе.

3.4.6. Отображение страницы оформления заказа с помощью директивы v-if

Пока что приложение умеет отображать лишь одну страницу. Чтобы сделать его более полноценным, нужна еще одна страница для оформления заказа. Это реализуется разными способами. В главе 7 вы познакомитесь с компонентами, которые позволяют легко разбивать приложение на несколько компактных блоков с возможностью повторного использования. Это один из вариантов, который мы могли бы выбрать для данной задачи.

Еще один подход состоит в применении к нашему представлению директивы `v-if`, привязанной к ранее созданному свойству `showProduct`. Ее следует добавить сразу после элементов `main` и `div` на главной странице, как показано в листинге 3.17.

Ранее в этой главе мы создали кнопку корзины. При ее нажатии значение свойства `showProduct` меняется с `true` на `false` или с `false` на `true`. Это приводит к срабатыванию директивы `v-if` из листинга 3.17. Мы увидим либо созданную ранее страницу товара, либо пустой экран с заголовком (рис. 3.15).

Листинг 3.17. Вывод страницы оформления заказа с помощью v-if: chapter-03/v-if-checkout.html

```
<main>
  <div class="row product">
    <div v-if="showProduct">
...
    </div>
    <div v-else>

    </div>
  </div> <!-- Конец строки -->
</main> <!-- Конец блока main -->
```

Элемент отображается, если showProduct равно true

Страница продукта с изображением и описанием

Сюда мы добавим страницу оформления заказа

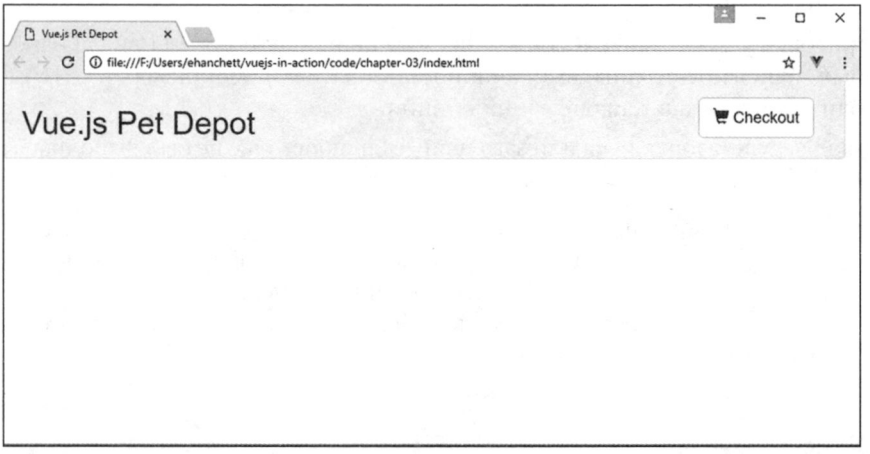


Рис. 3.15. Так выглядит магазин после нажатия кнопки корзины. Повторное нажатие отобразит страницу товара

Пока что забудьте о пустой странице, показанной на рис. 3.15. Мы займемся ею в следующей главе, когда рассмотрим разные виды связывания элементов ввода.

3.4.7. Сравнение v-show с v-if/v-else

Оба подхода, `v-show` и `v-if` и `v-else`, имеют свои преимущества и недостатки — как для пользователя, так и для разработчиков. Как мы знаем, директива `v-show` скрывает и показывает элемент с помощью CSS, тогда как директивы `v-if` и `v-else` удаляют содержимое из DOM. Выбор того или иного способа зависит в основном от поставленных задач, поэтому сравнивать их лучше в контексте реальных примеров.

Директива `v-show` лучше подходит для случаев, в которых отсутствует альтернативный вариант. То есть разметка отображается, только когда условие равно `true`, а если `false`, ничего другого не выводится. Далее перечислены несколько сценариев, подходящих для применения `v-show`.

- Вывод временных сообщений, например объявления о распродаже или об изменении пользовательского соглашения.

- Ссылка на страницу регистрации и другие элементы, которые видны только посетителям, не вошедшим в систему.
- Элементы для перелистывания списков, которые выводятся, только если список занимает больше одной страницы.

Директивы `v-if` и `v-else` подходят в ситуациях, когда есть два фрагмента разметки и *как минимум* один из них всегда должен быть виден. При отсутствии альтернативного варианта (`else`) лучше задействовать `v-show`. Далее приводятся несколько сценариев, в которых предпочтение следует отдавать `v-if` и `v-else`.

- Вывод ссылки Вход для пользователей-гостей и вывод ссылки Выход для уже вошедших посетителей.
- Отрисовка условных фрагментов формы, таких как поля для ввода адреса, которые зависят от страны, выбранной пользователем. Например, в США нужно вводить «штат», а в Канаде — «провинция».
- Вывод результатов поиска или заглушки, если поиск еще не был выполнен (позже мы рассмотрим возможность добавления третьего состояния с помощью `v-else-if`).

Существует бесконечное количество примеров, в которых лучше подходит та или иная условная директива. Наверное, лучший критерий выбора — наличие содержимого, которое нужно выводить по умолчанию или в качестве альтернативы. Дальше мы попробуем сделать магазин более интересным для потенциальных посетителей, расширив ассортимент.

Упражнение

Используя знания, приобретенные в этой главе, ответьте на следующий вопрос: какая разница между методами и вычисляемыми свойствами?

Ответ ищите в приложении Б.

Резюме

- Вывод данных, которые не входят в объект `data`, с помощью вычисляемых свойств.
- Вывод фрагментов приложения в зависимости от условия с использованием директив `v-if` и `v-else`.
- Расширение возможностей приложения с помощью методов.

4 Формы и поля ввода

В этой главе

- Связывание значений с DOM.
- Связывание текста.
- Модификаторы.

Если сравнить приложение с тем, каким оно было в главе 1, то мы увидим, что оно существенно эволюционировало. Мы сформулировали понятие единиц товара и сделали возможным их добавление в корзину. Теперь нужно организовать оформление покупки и позволить посетителям вводить свои данные. Добавим формы, в которых покупатель сможет оставить свой адрес и сведения об оплате. Затем нужно будет сохранить эту информацию для дальнейшего использования.

Чтобы все это реализовать, сначала нужно привязать данные формы к модели приложения. Специально для этого предусмотрена директива `v-model`.

ОПРЕДЕЛЕНИЕ

Директива v-model занимается двунаправленным связыванием между формой или полем ввода и шаблоном. Это гарантирует синхронизацию данных приложения с пользовательским интерфейсом.

Двунаправленное и однонаправленное связывание данных

В реальности двунаправленное связывание данных (рис. 4.1) — не всегда наилучшее решение. В некоторых случаях данные, введенные пользователем, больше не нужно менять. В других фреймворках и библиотеках, таких как React и Angular 2, по умолчанию действует однонаправленное связывание. В Angular 1 от двунаправленного связывания отказались по соображениям производительности во время разработки версии 2. При однонаправленном связывании вводимые данные не синхронизируются между моделью и представлением, а чтобы добиться синхронизации, необходим дополнительный код. Команда Ember.js решила предоставлять двунаправленное связывание

по умолчанию. Директива `v-model` связывает данные в обоих направлениях. Как бы то ни было, свойства в Vue всегда можно сделать однонаправленными, используя директиву `v-once`.

Директива `v-once` отрисовывает элементы и компоненты лишь один раз, делая их статическими. Последующие обновления не приводят к перерисовке и пропускаются. Подробнее об этом говорится в официальной документации к API, расположенной по адресу ru.vuejs.org/v2/api/index.html#v-once.

Позже в этой книге мы обсудим свойства компонентов и их связывание. Значение родительского свойства передается дочернему. Это нам еще пригодится.

Рис. 4.1. Модель обновляет представление, а представление обновляет модель

Директива `v-model` рассчитана на действия со всевозможными элементами формы, такими как однострочные и многострочные поля ввода, флажки, переключатели и раскрывающиеся списки. Все это нам понадобится при построении формы оформления покупки. Обсудим директиву `v-model` и то, как она работает со связанными элементами ввода.

4.1. Связывание с помощью v-model

Директива `v-model` поможет обновлять шаблон с использованием данных, которые вводит посетитель. Ранее мы задействовали объект `data` в основном для вывода статической информации. Взаимодействие с приложением было сведено к нажатию нескольких кнопок. Нам нужно позволить посетителю вводить сведения о доставке во время оформления покупки. Применим `v-model` для отслеживания полей формы и сделаем страницу более интерактивной с помощью простейшего связывания ввода.

Вам, наверное, интересно, какая разница между `v-model` и директивой `v-bind`, использованной в главе 2. Основное различие в том, что `v-model` по большей части задействуется для связывания полей ввода и форм. В этой главе применим данную директиву для привязки текстовых полей на странице оформления покупки. `v-bind` лучше подходит для связывания HTML-атрибутов. Например, с помощью этой

директивы можно привязать атрибут `src` в теге `<img>`. Нельзя сказать, что какой-то из этих подходов лучше другого, — они предназначены для разных ситуаций. Директиву `v-bind` подробно рассмотрим позже в этой главе.

Стоит отметить, что `v-bind` используется внутри `v-model`. Представьте, что у вас есть поле `<input v-model="something">`. В этом случае директива `v-model` — это всего лишь синтаксический сахар для кода `<input v-bind:"something" v-on:input="something=$event.target.value">`. Как бы то ни было, она облегчает написание и восприятие разметки.

На рис. 4.2 мы видим синтаксис директивы `v-model`. Она добавляется в поле ввода и создает двунаправленный объект связывания данных.

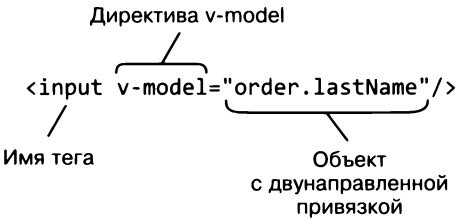


Рис. 4.2. Синтаксис директивы v-model

Для начала нужно добавить новую разметку в приложение. Откройте файл `index.html`, созданный в двух предыдущих главах, и найдите директиву `v-else` (или просто загрузите готовую версию `index.html` для главы 3). Чуть позже мы добавим HTML-код внутрь тега `<div>`. В главе 7 будет рассмотрен более подходящий способ разбиения приложения на компоненты, но пока для переключения на страницу оформления покупки будет использоваться простая директива `v-if`.

Как и в предыдущих главах, каждый фрагмент кода находится в отдельном файле. Чтобы получить готовое приложение, нужно объединить их с `index.html`.

Листинг 4.1. Директива v-model для ввода имени и фамилии: chapter-04/first-last.html

```
<div class="col-md-6">
  <strong>First Name:</strong>
  <input v-model="order.firstName"
    class="form-control"/>
</div>
<div class="col-md-6">
  <strong>Last Name:</strong>
  <input v-model="order.lastName"
   class="form-control"/>
</div>
<div class="col-md-12 verify">
  <pre>
    First Name: {{order.firstName}}
    Last Name:  {{order.lastName}}
  </pre>
</div>
```

Свойства firstName и lastName привязываются с помощью v-model

Свойства firstName и lastName выводятся по мере изменения значений в полях ввода

Код в листинге 4.1 создает два поля для ввода имени и фамилии, каждое из которых привязано к своему свойству и синхронизируется в режиме реального времени. Свойства находятся в объекте данных. Чтобы с ними было проще работать, сгруппируем их внутри другого объекта, `order`. И все это следует добавить в `index.html`.

В объекте `data` нужно создать новое свойство под названием `order`, в котором будут храниться имя и фамилия пользователя. Добавьте следующий код (листинг 4.2) в объект данных внутри файла `index.html`, который мы применяли в главе 3.

Листинг 4.2. Свойство order объекта data внутри экземпляра Vue: chapter-04/data-property.js

```
data: {
  sitename: 'Vue.js Pet Depot',
  showProduct: true,
  order: {
    firstName: '',
    lastName: ''
  },
```

В этом листинге объект `order` — это свойство объекта `data` внутри конструктора Vue. Можно обращаться к нему с помощью двойных фигурных скобок в стиле Mustache, с которым вы познакомились в главе 2, — `{{}}`. Например, вместо `{{order.firstName}}` будет подставлено значение `firstName` из объекта `order`. Благодаря такому группированию информации о заказе будет проще понять, где находятся те или иные данные, когда мы позже к ним вернемся.

Стоит отметить, что объект `order` можно было бы оставить пустым и не создавать свойства `firstName` и `lastName`. Vue.js умеет автоматически добавлять свойства в объект. Но, чтобы сделать код более простым, изящным и наглядным, мы добавили их вручную.

Обратите внимание на то, что данные, которые вводятся в форму для оформления заказа, сразу же появляются на нижней панели (рис. 4.3). В этом вся прелесть двунаправленного связывания данных. Значения автоматически синхронизируются друг с другом и не требуют дополнительного кода.

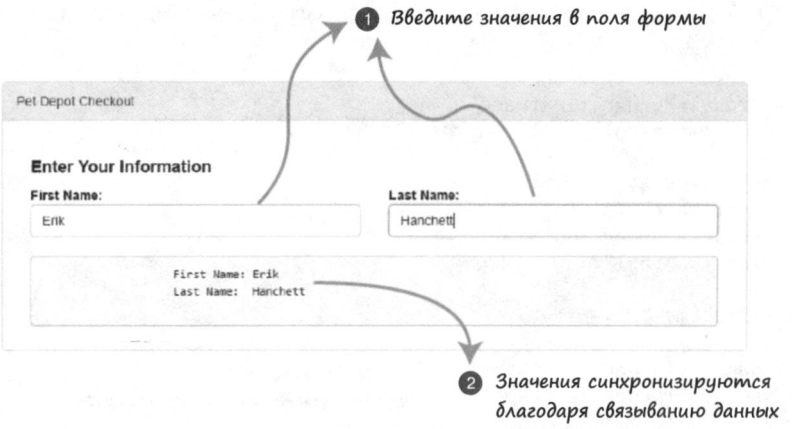


Рис. 4.3. Текст на нижней панели обновляется в режиме реального времени

Это начальная часть страницы оформления покупки. Добавим в нее дополнительные поля, чтобы посетитель мог вводить свой адрес, как показано в листинге 4.3. Этот HTML-код можно разместить сразу после разметки, созданной в листинге 4.1.

Листинг 4.3. Добавление других полей ввода и раскрывающегося списка: chapter-04/text-input.html

```
<div class="form-group">
  <div class="col-md-12"><strong>Address:</strong></div>
  <div class="col-md-12">
    <input v-model="order.address"
      class="form-control" />
  </div>
</div>
<div class="form-group">
  <div class="col-md-12"><strong>City:</strong></div>
  <div class="col-md-12">
    <input v-model="order.city"
      class="form-control" />
  </div>
</div>
<div class="form-group">
  <div class="col-md-2">
  <strong>State:</strong>
    <select v-model="order.state"
      class="form-control">
      <option disabled value="">State</option>
      <option>AL</option>
      <option>AR</option>
      <option>CA</option>
      <option>NV</option>
    </select>
  </div>
</div>
<div class="form-group">
  <div class="col-md-6 col-md-offset-4">
  <strong>Zip / Postal Code:</strong>
    <input v-model="order.zip"
      class="form-control"/>
  </div>
</div>
<div class="col-md-12 verify">
  <pre>
    First Name: {{order.firstName}}
     Last Name: {{order.lastName}}
       Address: {{order.address}}
          City: {{order.city}}
           Zip: {{order.zip}}
         State: {{order.state}}
  </pre>
</div>
```

Раскрывающийся список с v-model

Поля ввода с v-model

Тег <pre> отображает данные

Мы добавили поля для ввода адреса, города, штата и почтового индекса. Все значения, кроме штата, — это текстовые поля, применяющие директиву `v-model` для связывания ввода.

Выбор штата происходит немного иначе. Вместо текстового поля используется раскрывающийся список, `<select>`, в котором указана директива `v-model`.

Вам, наверное, интересно, как лучше добавлять новые пункты в элемент раскрывающегося списка. В этом простом примере их достаточно прописать прямо в коде. Но если бы нам нужно было добавить все 50 штатов, список лучше было бы генерировать динамически. В следующем разделе вы узнаете, как это сделать с помощью связывания значений.

Прежде чем продолжать, не забудьте добавить новые свойства в объект `data` экземпляра Vue (листинг 4.4).

Листинг 4.4. Добавление новых свойств в объект данных экземпляра Vue: chapter-04/data-new-properties.js

```
data: {
  sitename: "Vue.js Pet Depot",
  showProduct: true,
  order: {
    firstName: '',
    lastName: '',
    address: '',
    city: '',
    zip: '',
    state: ''
  },
```

Как вы видели на рис. 4.3, изменение любого из этих свойств внутри любого элемента формы приводит к автоматическому обновлению значений в теге `<pre>`, указанном внизу. После перезагрузки страницы ваша форма будет выглядеть следующим образом (рис. 4.4).

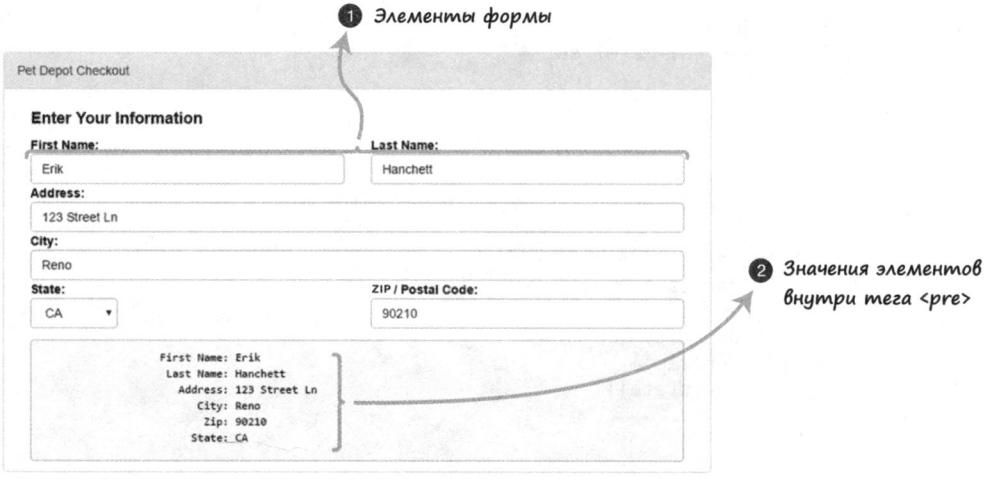


Рис. 4.4. Добавление полей для ввода адреса, города, штата и почтового индекса на страницу оформления покупки

Форма для заказа выглядит неплохо, но нужно позаботиться еще о нескольких моментах. Позволим посетителю отправлять покупку в виде подарка. Для этого достаточно добавить небольшой флажок. Если он установлен, мы отправим товар в подарочной упаковке, если нет — не станем использовать эту услугу. Привязка данных будет отслеживаться с помощью свойства `order.gift`.

Дальше следует позволить посетителю выбирать между домашним и рабочим адресами. Для этого добавим в код переключатель. В Vue директиву `v-model` необходимо указывать для обоих состояний, иначе переключатель не будет обновляться при щелчке кнопкой мыши.

Нужно также добавить свойства `order.method` и `order.gift` в тег `<pre>`, как показано в листинге 4.5. Вставьте этот код в файл `index.html` сразу после разметки из листинга 4.3.

Листинг 4.5. Добавление флажков и переключателей: chapter-04/adding-buttons.html

```
<div class="form-group">
  <div class="col-md-6 boxes">
    <input type="checkbox"
      id="gift"
      value="true"
      v-model="order.gift">                    <-- Флажок с v-model
    <label for="gift">Ship As Gift?</label>
  </div>
</div>
<div class="form-group">
  <div class="col-md-6 boxes">
    <input type="radio"
      id="home"
      value="Home"
      v-model="order.method">                  <-- Переключатель
    <label for="home">Home</label>                 с v-model
    <input type="radio"
      id="business"
      value="Business"
      v-model="order.method">                  <--
    <label for="business">Business</label>
  </div>
</div>
<div class="col-md-12 verify">
  <pre>                                        <-- В тег <pre> добавлены
    First Name: {{order.firstName}}                свойства order.method и order.gift
     Last Name: {{order.lastName}}
       Address: {{order.address}}
          City: {{order.city}}
           Zip: {{order.zip}}
         State: {{order.state}}
        Method: {{order.method}
          Gift: {{order.gift}}
  </pre>
</div>
```

Теперь внесем наши свойства в объект `data`, добавив следующий код (листинг 4.6).

Листинг 4.6. Добавление дополнительных свойств в объект данных: chapter-04/more-props.js

```
data: {
  sitename: "Vue.js Pet Depot",
  showProduct: true,
  order: {
    firstName: '',
    lastName: '',
    address: '',
    city: '',
    zip: '',
    state: '',
    method: 'Home',
    gift: false
  },
```

Вы могли заметить, что свойства `method` и `gift` сразу же инициализируются. Этому есть простое объяснение. Переключатели устанавливаются по умолчанию, а флажки — нет. Таким образом, этим свойствам лучше присвоить начальные значения.

Теперь осталось только добавить кнопку Place Order (Заказать). Мы задействуем ее позже, а пока что она будет здесь лишь для вида. Кнопка создается несколькими способами. Вы можете использовать атрибут формы `action`, который охватывает все элементы ввода (подробнее об этом поговорим в главе 6 при обсуждении событий). Но вместо этого применим директиву `v-on`, с которой вы познакомились в главе 3, — она привязывает функцию к элементу DOM. Добавьте ее в кнопку Place Order (Заказать) для события `click`. Следующий HTML-код (листинг 4.7) можно вставить после кода из листинга 4.5.

Листинг 4.7. Добавление директивы v-on для события click: chapter-04/adding-v-on.html

```
<div class="form-group">
  <div class="col-md-6">
    <button type="submit"
      class="btn btn-primary submit"
      v-on:click="submitForm">Place Order</button>
  </div>
</div>
```

Кнопка отправки заказа привязана к директиве v-on

В последующих главах мы свяжем кнопку отправки заказа с полезным действием. А пока создадим простую функцию для вывода диалогового окна и посмотрим, будет ли она работать. Добавьте функцию `submitForm` в объект `methods` внутри файла `index.html` (листинг 4.8).

Листинг 4.8. Создание нового метода submitForm: chapter-04/submit.js

```
methods: {
  submitForm() {
    alert('Submitted');
...
  }
},
```

Внутри конструктора Vue находится объект `methods`, он содержит все функции, которые могут быть вызваны во время работы приложения. Функция `submitForm` отображает диалоговое окно. Нажмите кнопку Place Order (Заказать), и вы увидите соответствующее сообщение (рис. 4.5).

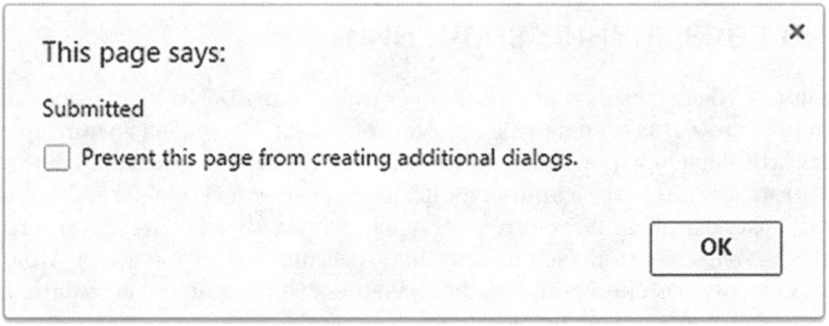


Рис. 4.5. Это диалоговое окно выводится функцией submitForm

На рис. 4.6 показано, как должна выглядеть форма вместе с кнопкой Place Order (Заказать).

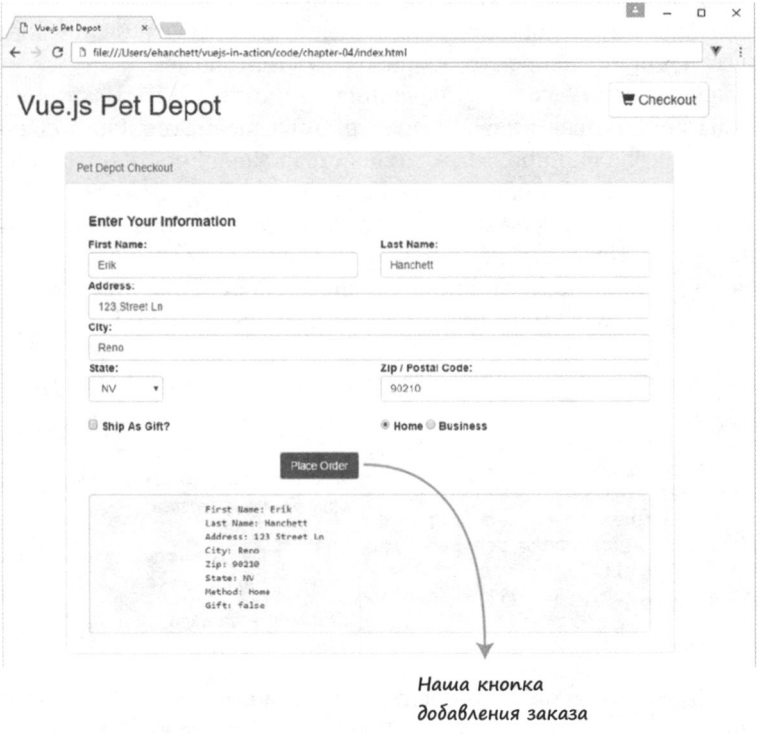


Рис. 4.6. Готовая страница оформления покупки со всеми элементами

Каждое свойство формы привязано к модели Vue.js! Теперь попробуем улучшить привязки для элементов ввода.

4.2. Повторный взгляд на связывание значений

Вы уже могли убедиться в полезности директивы `v-model`. Мы использовали ее для связывания множества стандартных элементов ввода. Но теперь возникает вопрос: как привязать значения для флажков, переключателей и раскрывающихся списков? Вы, вероятно, помните, что в примере у флажка и переключателей есть статические значения, а раскрывающийся список оставлен пустым. Все HTML-элементы могут, а иногда и должны иметь значение, связанное с выбранным вариантом. Модифицируем нашу форму, заменив статические значения свойствами объекта `data`. Начнем с флажка, задействуя директиву `v-bind`.

4.2.1. Привязка значений к флажку

В первом примере флажок был привязан к свойству `order.gift`. Мы могли его устанавливать и сбрасывать, выбирая между `true` и `false`. Но нашим посетителям это ни о чем не говорит. Чтобы они понимали, будет ли заказ упакован в виде подарка, следует вывести для них какое-то сообщение. Этим и займемся.

Директива `v-bind` привязывает значения к атрибутам HTML-элементов. В данном случае мы привязываем к свойству атрибут `true-value`. Он поддерживается только в `v-bind` и позволяет привязывать свойства в зависимости от состояния флажка (`true` или `false`). Таким образом будет меняться значение `order.gift`. В листинге 4.9 `true-value` привязывается к свойству `order.sendGift`, а `false-value` — к `order.dontSendGift`. Когда флажок установлен, выводится сообщение `order.sendGift`, когда сброшен — пользователь видит значение `order.dontSendGift`. Вставьте в `index.html` после разметки из листинга 4.8 следующий HTML-код.

Листинг 4.9. Привязывание значений true и false к флажку: chapter-04/true-false.html

```
<div class="form-group">
  <div class="col-md-6 boxes">
    <input type="checkbox"
      id="gift" value="true"
      v-bind:true-value="order.sendGift"
      v-bind:false-value="order.dontSendGift"
      v-model="order.gift">
    <label for="gift">Ship As Gift?</label>
  </div>
</div>
```

Подставляет свойство order.sendGift, когда флажок установлен

Подставляет свойство order.dontSendGift, когда флажок снят

Привязывает order.gift к элементу ввода

Чтобы эта привязка работала как следует, нужно добавить новые свойства в объект `order`, как показано в листинге 4.10. Создайте свойства `sendGift` и `dontSendGift`.

Листинг 4.10. Добавление свойства sendGift в объект order: chapter-04/prop-gift.js

```
order: {
  firstName: '',
  lastName: '',
  address: '',
  city: '',
  zip: '',
  state: '',
  method: 'Business',
  gift: 'Send As A Gift',
  sendGift: 'Send As A Gift',
  dontSendGift: 'Do Not Send As A Gift'
},
```

По умолчанию флажок использует значение 'Send As A Gift'

Свойство order.sendGift является текстовым сообщением, которое выводится при установке флажка

Свойство order.dontSendGift является текстовым сообщением, которое выводится при снятии флажка

Объект данных становится все больше! Теперь мы можем присваивать текстовые значения в зависимости от состояния флажка. Обновите страницу и снимите флажок Ship As Gift (Отправить как подарок). Посмотрите на нижнюю панель (рис. 4.7): свойство `{{order.gift}}` теперь выводит новое значение.

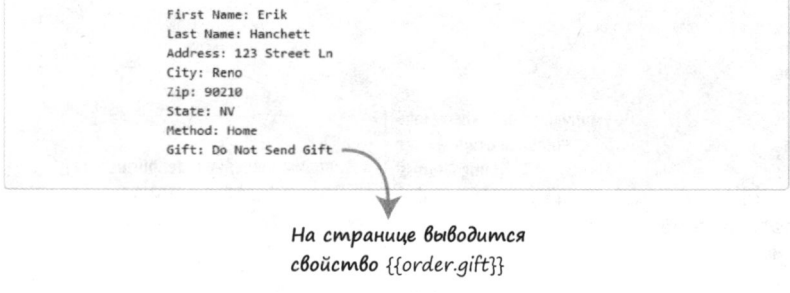


Рис. 4.7. На странице выводится свойство {{order.gift}}

Щелчок на флажке меняет значение с `'Do Not Send As A Gift'` на `'Send As A Gift'`. Поскольку свойство `order.gift` изначально равно `'Send As A Gift'`, флажок устанавливается по умолчанию. При необходимости значение можно поменять, в этом случае флажок по умолчанию будет снят.

4.2.2. Привязка значений и переключатели

Значение можно присваивать не только флажкам, но и переключателям. Для этого оно привязывается напрямую. Будем выводить для пользователя домашний или рабочий адрес в зависимости от выбранного значения — Home (Домой) или Business (На работу). Добавьте в `index.html` следующую разметку (листинг 4.11) после предыдущего кода с флажком.

Листинг 4.11. Привязка значений к переключателям: chapter-04/radio-bind.html

```
<div class="form-group">
  <div class="col-md-6 boxes">
    <input type="radio"
      id="home"
      v-bind:value="order.home"
      v-model="order.method">
    <label for="home">Home</label>
    <input type="radio"
      id="business"
      v-bind:value="order.business"
      v-model="order.method">
    <label for="business">Business</label>
  </div>
</div>
```

Применяем директиву v-bind к атрибуту value первого переключателя

Применяем директиву v-bind к атрибуту value второго переключателя

Директива `v-bind` привязывает `order.home` к первому переключателю, а `order.business` — ко второму. Это очень гибкое решение, так как мы можем в любой момент динамически поменять данные значения.

Чтобы завершить пример, добавим новые свойства в объект `order` внутри `data` (листинг 4.12).

Листинг 4.12. Добавление свойств business и home в объект order: chapter-04/update-order.html

```
order: {
  firstName: '',
  lastName: '',
  address: '',
  city: '',
  zip: '',
  state: '',
  method: 'Home Address',
  business: 'Business Address',
  home: 'Home Address',
  gift:'',
  sendGift: 'Send As A Gift',
  dontSendGift: 'Do Not Send As A Gift'
},
```

По умолчанию переключатель установлен в положение 'Home Address'

Выводим значение order.business при выборе первого варианта

Выводим значение order.home при выборе второго варианта

В этом объекте `order` появились два новых свойства, `home` и `business`, которые привязаны к переключателю. Установка переключателя в одно из этих положений приводит к выводу домашнего или рабочего адреса (рис. 4.8).

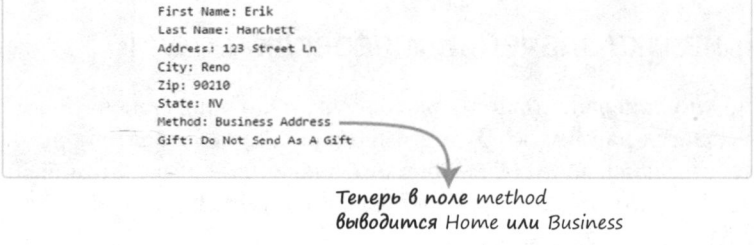


Рис. 4.8. Способ отправки обновляется при установке переключателя в то или иное положение

Посетитель видит, что посылка будет доставлена по рабочему адресу (Erik Hanchett at 123 Street Ln, Reno, NV) без подарочной упаковки! Привязка свойств к значениям атрибутов формы делает все намного проще и понятней. В следующем разделе рассмотрим раскрывающийся список для выбора штата.

4.2.3. Знакомство с директивой v-for

Раскрывающийся список содержит перечень американских штатов, один из которых можно выбрать. Следует модифицировать этот элемент так, чтобы он отображал какой-то штат при обновлении страницы. Посмотрим, как привязать соответствующие значения. Уберите из `index.html` раскрывающийся список и вставьте вместо него следующий код (листинг 4.13).

Листинг 4.13. Привязка значений к раскрывающемуся списку: chapter-04/bind-select.html

```
<div class="form-group">
  <div class="col-md-2">
    <strong>State:</strong>
    <select v-model="order.state" class="form-control">
      <option disabled value="">State</option>
      <option v-bind:value="states.AL">AL</option>
      <option v-bind:value="states.AR">AR</option>
      <option v-bind:value="states.CA">CA</option>
      <option v-bind:value="states.NV">NV</option>
    </select>
  </div>
</div>
```

Директива v-bind присваивает атрибуту значение свойства states.AL

Директива v-bind присваивает атрибуту значение свойства states.AR

Директива v-bind присваивает атрибуту значение свойства states.CA

Директива v-bind присваивает атрибуту значение свойства states.NV

Директива `v-bind` присваивает значения атрибуту, как уже делалось ранее. На этот раз мы создали в объекте данных новое свойство `states`, внутри которого перечислены американские штаты. Объект `states` содержит четыре значения. Можно выбирать их с помощью раскрывающегося списка, используя директиву `v-bind`. Откройте файл `index.html` и добавьте `states` в объект `data` (листинг 4.14).

Листинг 4.14. Добавление свойства states в объект данных экземпляра Vue: chapter-04/states.html

```
states: {
  AL: 'Alabama',
  AR: 'Arizona',
  CA: 'California',
  NV: 'Nevada'
},
```

После обновления кода эти значения должны появиться на панели внизу страницы (рис. 4.9). Теперь посетитель может видеть полное название штата.

Ранее в этой главе я упомянул о важной проблеме, связанной с раскрывающимися списками. В нашем примере перечислены лишь четыре штата. Для каждого нового пункта придется добавлять отдельный элемент `option`. Это хлопотное и монотонное занятие. К счастью, Vue может с этим помочь. Специально для таких случаев предусмотрена директива `v-for`.

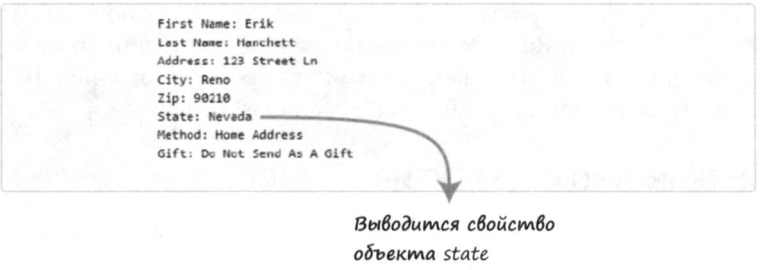


Рис. 4.9. Поле State выводит название выбранного штата

Директива `v-for` позволяет с легкостью перебирать значения в списке или объекте, что идеально подходит для нашей ситуации. Для начала определим все штаты в объекте `states`. Затем пройдемся по каждому штату и подставим подходящие значения с помощью директивы `v-bind`. Итак, приступим!

Здесь нужно многое сделать, поэтому начнем по порядку. Директива `v-for` имеет специальный синтаксис вида `state in states`, где `states` — исходный массив с данными, а `state` — ссылка на текущий элемент цикла. В данном случае значения `state` будут равны `Alabama`, `Arizona`, `California` и т. д. Замените раскрывающийся список в `index.html` следующим HTML-кодом (листинг 4.15).

Листинг 4.15. Применение v-for в раскрывающемся списке: chapter-04/select-drop-down.html

```
<div class="form-group">
  <div class="col-md-2">
    <strong>State:</strong>
    <select v-model="order.state"
      class="form-control">
      <option disabled value="">State</option>
      <option v-for="(state, key) in states"
              v-bind:value="state">
        {{key}}
      </option>
    </select>
  </div>
</div>
```

Директива v-for проходится по всем ключам и значениям объекта states

Директива v-bind присваивает атрибуту значение свойства state

Выводится свойство key

Аргумент `key` необязательный и содержит индекс текущего элемента. Это важно, поскольку в раскрывающемся списке в качестве значений для `key` используются сокращенные обозначения штатов, тогда как полные названия предназначены для атрибута `value`.

Как видно в листинге 4.16, директива `v-bind` привязывает значение `state` к атрибуту `value` тега `<option>`. Подставив эту разметку в приложение, взгляните на исходный код, сгенерированный файлом `index.html`. Для каждого штата в свойстве `states` выводится отдельный элемент `<option>`.

Листинг 4.16. HTML-код, сгенерированный директивой v-for: chapter-04/options.html

```
<option value="Alabama">
    AL
  </option>
  <option value="Alaska">
    AK
  </option>
  <option value="Arizona">
    AR
  </option>
  <option value="California">
    CA
  </option>
  <option value="Nevada">
    NV
  </option>
```

Это очень поможет в разработке приложения. Вместо того чтобы вводить каждый штат вручную, мы можем привязывать значения и перебирать их с помощью `v-for`. Раскрывающийся список будет пополняться динамически в зависимости от содержимого объекта `states`.

4.2.4. Директива v-for без необязательного аргумента key

Уже упоминалось, что аргумент `key` опциональный. Как будет выглядеть директива `v-for`, если мы его опустим? Немного отвлечемся от основного примера и создадим пустой файл `detour.html`, где будет храниться совершенно новое приложение. Нам понадобятся конструктор Vue и объект `data` с массивом `states` (листинг 4.17).

Листинг 4.17. Обновление массива states в объекте data: chapter 04/detour.html

```
<div id="app">
  <ol>
    <li v-for="state in states">
      {{state}}
    </li>
  </ol>
</div>
<script src="https://unpkg.com/vue/dist/vue.js"></script>
<script type="text/javascript">
var webstore = new Vue({
  el: '#app',
  data: {
    states: [
      'Alabama',
      'Alaska',
      'Arizona',
```

Директива v-for с синтаксисом state in states

```
      'California',
      'Nevada'
    ]
  }
})
</script>
```

ДОПОЛНИТЕЛЬНО

Массив states содержит пять значений. Выведем их в виде упорядоченного списка. Нам не нужно беспокоиться о ключах, поскольку их просто нет. Для создания списка понадобятся теги <ol> и <li>. Добавьте их в начало нового файла detour.html.

Директива `v-for` проходится по массиву `states` и выводит каждый элемент `state` внутри списка. Как вы уже знаете, `state` — это ссылка на текущий элемент массива, тогда как `states` — сам массив. Чтобы не запутаться, помните, что сначала идет ссылка, затем опциональный ключ, а в конце — массив или объект, который перебирается.

На странице должен появиться пронумерованный список штатов.

1. Alabama.
2. Alaska.
3. Arizona.
4. California.
5. Nevada.

Теперь можно добавлять новые элементы в объект `states`, не изменяя при этом шаблон.

ПРИМЕЧАНИЕ

Бывают ситуации, когда объектной моделью документа необходимо манипулировать напрямую, что делает работу с директивой v-model нежелательной. На этот случай в Vue.js предусмотрено свойство экземпляра $el, к которому можно обращаться как к this.$el. Оно представляет корневой элемент DOM, которым управляет экземпляр Vue. Из него можно вызывать любые методы типа Document, такие как querySelector(), чтобы получить нужный элемент. По возможности старайтесь использовать для работы с DOM встроенные в Vue.js директивы — они созданы специально для того, чтобы упростить разработку! Больше информации об $el и других API найдете в официальной документации по адресу https://ru.vuejs.org/v2/api/.

4.3. Знакомство с модификаторами

Ранее в этой главе упоминалось, что директива `v-model` способна привязываться к вводимым значениям, которые обновляются с каждым событием ввода. Это поведение можно изменить с помощью модификаторов. Например, приведем значения

к численному типу с помощью `.number` или применим `.trim` (подробнее об этом — на странице ru.vuejs.org/v2/guide/forms.html#Модификаторы). Кроме того, модификаторы можно объединять в цепочку, например `v-model.trim.number`. Сделаем так на странице оформления покупки.

4.3.1. Использование модификатора .number

Модификатор `.number` применяется для автоматического приведения значений в `v-model` к числовому типу. Это может пригодиться для ввода почтового индекса (модификатор `.number` удаляет начальные нули, поэтому будем считать, что в нашем случае таких почтовых индексов не бывает). Обновим поле ZIP Code (Почтовый индекс) в файле `index.html`, добавив использование `.number`, и посмотрим, что из этого получится (листинг 4.18).

Листинг 4.18. Элемент ввода почтового индекса с модификатором .number: chapter-04/number-mod.html

```
<div class="form-group">
  <div class="col-md-6 col-md-offset-4">
    <strong>Zip / Postal Code:</strong>
    <input v-model.number="order.zip"
      class="form-control"
      type="number"/>
  </div>
</div>
```

Директива v-model с модификатором .number

В HTML поля ввода всегда возвращают строку, даже если указать для них атрибут `type="number"`. Добавление модификатора `.number` позволяет возвращать числа. Чтобы в этом убедиться, воспользуемся оператором `typeof` для вывода свойства `order.zip` в шаблоне (листинг 4.19).

Листинг 4.19. Применение оператора typeof к order.zip: chapter-04/type-of.html

```
<div class="col-md-12 verify">
     <pre>
       First Name: {{order.firstName}}
       Last Name: {{order.lastName}}
       Address: {{order.address}}
       City: {{order.city}}
       Zip: {{typeof(order.zip)}}
       State: {{order.state}}
       Method: {{order.method}}
       Gift: {{order.gift}}
     </pre>
</div>
```

JavaScript-оператор typeof возвращает тип заданного операнда

Раньше мы бы получили `string`. Теперь, после добавления модификатора `.number`, на странице выводится `number`. Введите число в поле для почтового индекса, результат должен выглядеть как на рис. 4.10.

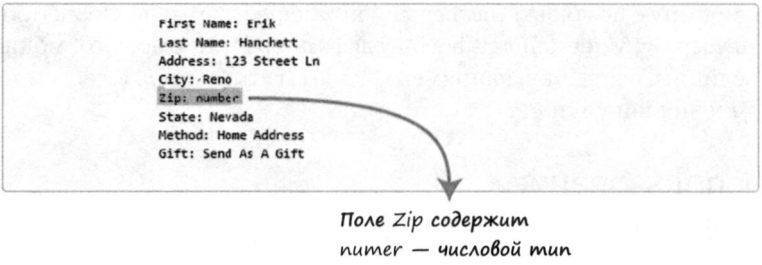


Рис. 4.10. Тип значения, введенного в свойство zip

Как видите, в поле Zip выводится значение `number`. Это тип введенного значения, которое служит операндом для `typeof`. Позже мы вернемся к этой методике, а пока уберите оператор `typeof`, чтобы на странице появлялся почтовый индекс. В скобках должно остаться лишь свойство — `{{order.zip}}`.

4.3.2. Обрезка вводимых значений

Часто при извлечении информации из формы можно столкнуться с лишними пробельными символами в начале или в конце текстового значения. Если пользователь случайно ввел несколько пробелов перед своим именем, нужно от них избавиться. Для автоматизации этой задачи в Vue.js предусмотрен специальный модификатор.

В нашем приложении текстовые поля применяются для ввода имени, фамилии, адреса и города. Модифицируем первые два из них в файле `index.html`, чтобы продемонстрировать, как работает модификатор `.trim` (листинг 4.20).

Листинг 4.20. Имя и фамилия с модификатором .trim: chapter-04/trim-mod.html

```
<div class="form-group">
  <div class="col-md-6">
    <strong>First Name:</strong>
    <input v-model.trim="order.firstName"
      class="form-control" />
  </div>
  <div class="col-md-6">
    <strong>Last Name:</strong>
    <input v-model.trim="order.lastName"
      class="form-control" />
  </div>
</div>
```

Директива v-model использует модификатор .trim для свойства order.firstName

Директива v-model использует модификатор .trim для свойства order.lastName

Чтобы применить модификатор `.trim`, достаточно указать его в конце директивы `v-model`. Теперь пробельные символы будут отсекаться автоматически! То же самое можно сделать с полями для ввода адреса и города (листинг 4.21).

Листинг 4.21. Адрес и город с модификатором .trim: chapter-04/trim-mod-add.html

```
<div class="form-group">
  <div class="col-md-12"><strong>Address:</strong></div>
  <div class="col-md-12">
    <input v-model.trim="order.address"
  class="form-control" />
  </div>
</div>
<div class="form-group">
  <div class="col-md-12"><strong>City:</strong></div>
  <div class="col-md-12">
    <input v-model.trim="order.city"
       class="form-control" />
  </div>
</div>
```

Директива v-model использует модификатор .trim для свойства order.address

Директива v-model использует модификатор .trim для свойства order.city

Обновив страницу и посмотрев на вывод внизу, увидим, что пробельные символы удалены (рис. 4.11).

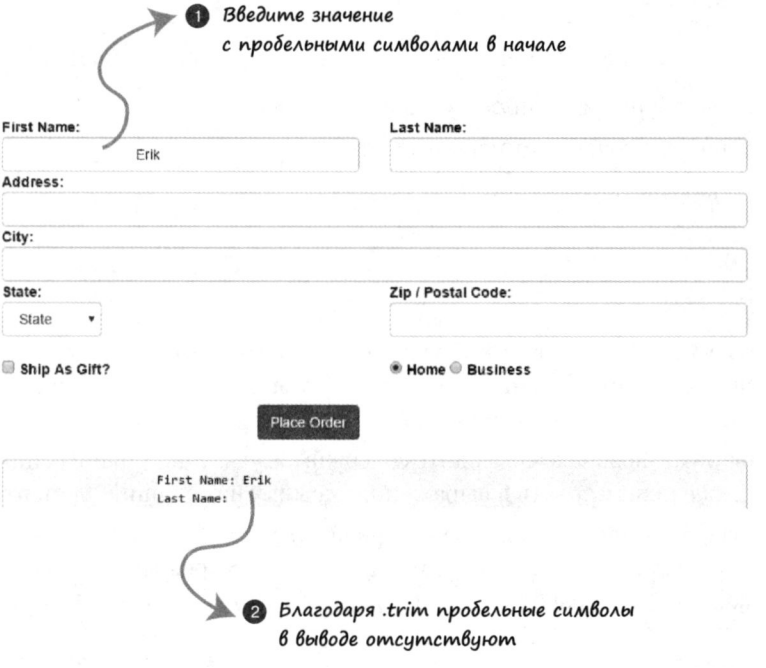


Рис. 4.11. Пример использования модификатора .trim с директивой v-model

Поле ввода имени содержит значение `Erik` с множеством пробелов перед ним. А внизу отображается обрезанный результат. На самом деле, если щелкнуть за пределами поля ввода, набранный текст синхронизируется с обрезанным значением. Это заслуга модификатора `.trim`.

4.3.3. Модификатор .lazy

Последний доступный модификатор — `.lazy`. Как упоминалось ранее, директива `v-model` синхронизируется после каждого события ввода, то есть после набора каждого символа в текстовом поле. Синхронизация происходит в момент нажатия клавиши. Модификатор `.lazy` позволяет синхронизировать значения при наступлении событий `change`. В какой ситуации генерируются события этого типа, зависит от того, какой элемент формы мы используем. Флажки и переключатели генерируют `change` при щелчке, а текстовое поле — при потере фокуса. Событие `change` может по-разному генерироваться в разных браузерах, имейте это в виду.

Обычно модификатор `.lazy` в сочетании с директивой `v-model` выглядит следующим образом:

```
<input v-model.lazy="order.firstName" class="form-control" />
```

Упражнение

Используя знания, приобретенные в этой главе, ответьте на следующие вопросы.

- Как работает двунаправленное связывание данных?
- В каких случаях его следует использовать?

Ответы ищите в приложении Б.

Резюме

- Директива `v-model` позволяет привязывать однострочные и многострочные поля ввода, раскрывающиеся списки и компоненты. Она создает двунаправленное связывание для элементов ввода и компонентов.
- Директива `v-for` позволяет выводить содержимое указанных данных циклически. Можно также использовать в выражениях ссылку на текущий элемент.
- Директива `v-model` поддерживает модификаторы `.trim`, `.lazy` и `.number`. Модификатор `.trim` удаляет пробельные символы, а `.number` превращает строки в числа. Модификатор `.lazy` позволяет изменить момент синхронизации данных.

5 Условные выражения, циклы и списки

В этой главе

- Работа с условными выражениями `v-if` и `v-if-else`.
- Циклы на основе `v-for`.
- Модификация массивов.

В главе 4 мы познакомились с возможностями директивы `v-model` и узнали, как с ее помощью привязывать поля ввода к приложению. Создали страницу оформления покупки, которая содержит все элементы формы, необходимые для сбора информации о пользователе. Для отображения этой страницы применялось условное выражение.

В главе 3 мы создали кнопку корзины, связанную с обработчиком события `click`. Этот метод меняет значение свойства `showProduct`. В нашем шаблоне задействуются директивы `v-if` и `v-else`. Если `showProduct` равно `true`, выводится страница товара, а если `false` — страница оформления покупки. Пользователь способен легко переключаться между этими страницами, нажимая кнопку корзины. Пока этого достаточно, хотя в последующих главах мы структурируем данный код с применением компонентов и маршрутов.

Чтобы расширить возможности приложения, рассмотрим другие виды условных выражений. Например, нам нужно показывать пользователю сообщения о наличии товара. Кроме того, мы должны добавить на страницу новые продукты (подробнее об этом — в разделе 5.2).

5.1. Сообщения о наличии товара

При добавлении в корзину каждой новой единицы товара обновляется вычисляемое свойство `cartItemCount`. Что, если мы хотим держать пользователя в курсе о количестве доступного товара? Предусмотрим разные сообщения на случай, когда запасы начнут истощаться. Применим для этого директивы `v-if`, `v-else-if` и `v-else`.

5.1.1. Сообщение об оставшемся товаре с помощью v-if

Сначала пополним наши запасы, чтобы было легче выводить сообщение при добавлении в корзину новых единиц товара. Для этого следует обновить свойство `product` в объекте данных. Найдите в файле `index.html` свойство `availableInventory` и поменяйте его значение с 5 на 10, как показано в листинге 5.1. Пока что этого достаточно.

Если вы последовательно выполняете все примеры, у вас уже должен быть файл `index.html`. В противном случае можете загрузить подходящую версию для этой главы вместе со всеми фрагментами и CSS-стилями. Как и прежде, каждый листинг разбит на несколько файлов. Продвигаясь дальше, не забывайте добавлять каждый фрагмент кода в `index.html`.

Листинг 5.1. Обновление availableInventory: chapter-05/update-inventory.js

```
product: {
  id: 1001,
  title: "Cat Food, 25lb bag",
  description: "A 25 pound bag of <em>irresistible</em>,
                organic goodness for your cat.",
  price: 2000,
  image: "assets/images/product-fullsize.png",
  availableInventory: 10
},
```

Увеличиваем запасы

Теперь добавим в шаблон условное выражение на тот случай, если запасы начнут истощаться. Пользователь увидит сообщение о том, сколько единиц товара он может добавить в корзину. В листинге 5.2 показан новый элемент `span` с директивой `v-if` и классом `inventory-message`. Этот CSS-класс делает сообщение более заметным и размещает его в нужном месте. Я использовал простое форматирование, чтобы сделать текст чуть приятнее на вид. Гибкость директивы `v-if` позволяет указывать не только свойства, как мы это делали в главе 3 с `showProduct`, но и выражения. Это одна из приятных мелочей, которые поддерживаются в Vue.js.

При нажатии кнопки добавления в корзину увеличивается число вверху страницы. Когда остается меньше пяти единиц товара (`product.availableInventory - cartItemCount`), выводится сообщение о доступных запасах. При дальнейших нажатиях кнопки счетчик продолжит уменьшаться, пока не достигнет нуля.

Найдите в шаблоне кнопку `addToCart`. Добавьте в файл `index.html` новый тег — элемент `span` с директивой `v-if`, как показано далее.

Листинг 5.2. Вывод сообщения в зависимости от количества доступного товара: chapter-05/add-message.html

```
<button class="btn btn-primary btn-lg"
  v-on:click="addToCart"
  v-if="canAddToCart">Add to cart</button>
  <button disabled="true" class="btn btn-primary btn-lg"
    v-else >Add to cart</button>
  <span class="inventory-message"
v-if="product.availableInventory - cartItemCount < 5">
Only {{product.availableInventory - cartItemCount}} left!
  </span>
```

Элементу span назначается класс inventory-message class

Директива v-if отображается, только если выражение истинно

ПОМНИТЕ

Весь исходный код для этой и всех остальных глав, включая файл app.css, доступен для загрузки на сайте Manning по адресу www.manning.com/books/vue-js-in-action.

Мы могли бы использовать в директиве `v-if` вычисляемое свойство, но для простоты остановимся на обычном выражении. Хотя, если выражение становится слишком длинным, его лучше оформить в виде вычисляемого свойства.

Краткий обзор директивы v-show

`v-show` — близкий родственник директивы `v-if` и может действовать сходным образом: `<span v-show=" product.availableInventory - cartItemCount < 5">Message</span>`. Единственное отличие в том, что `v-show` всегда присутствует в DOM, а для отображения/скрытия элемента применяется CSS-свойство `display`. Директиву `v-if` лучше использовать, когда за ней идет `v-else` или `v-else-if`. `v-show` подходит в ситуациях, когда содержимое показывается/отрисовывается большую часть времени или когда видимость элемента меняется несколько раз на протяжении жизненного цикла страницы.

Посмотрим, что у нас получилось (рис. 5.1).

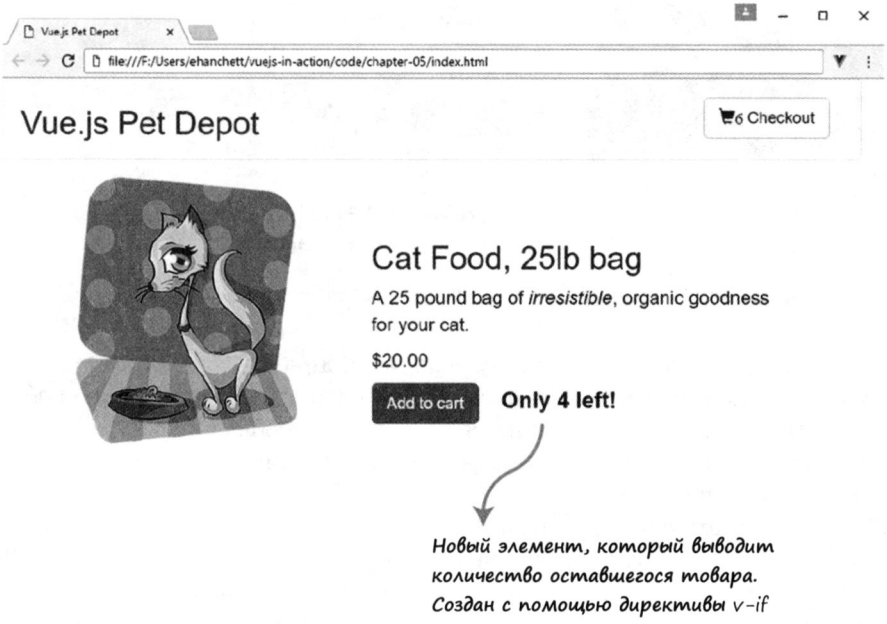


Рис. 5.1. О том, что осталось всего четыре единицы товара, мы знаем благодаря директиве v-if

5.1.2. Дополнительные сообщения с использованием v-else и v-else-if

У нас есть небольшая проблема. Когда количество товара достигает нуля, отображается сообщение Only 0 left!. Очевидно, что это бессмыслица, поэтому попробуем выводить в этой ситуации более подходящий текст. Добавим сообщение, которое поощряет посетителя покупать больше товара. На этот раз задействуем директивы `v-else-if` и `v-else`! Для начала сформулируем, что и как мы хотим сделать. Директива `v-if` будет отображаться, если разница между нашими запасами и количеством элементов в корзине равна нулю. Если в корзину добавлен весь товар, наши запасы полностью исчерпаны.

Результат можно видеть на рис. 5.2. Когда товар закончился, выводится сообщение All Out.

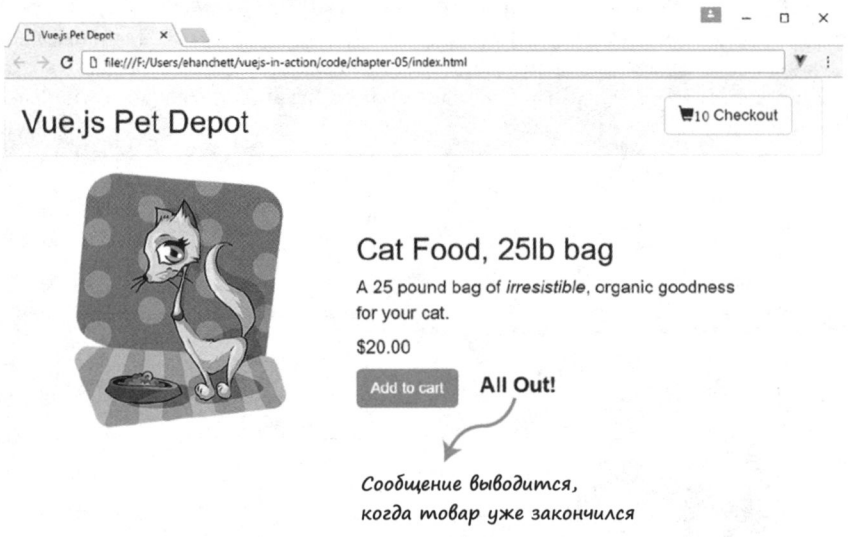


Рис. 5.2. Страница товара сообщает об исчерпании запасов

Если товар еще есть в наличии, переходим к директиве `v-else-if`. Если мы близки к исчерпанию запасов (осталось меньше пяти единиц), появится сообщение Buy Now!. Если нажать кнопку добавления в корзину, сообщение должно поменяться. Когда останется меньше пяти единиц товара, страница будет выглядеть как на рис. 5.1. При исчерпании запасов вернемся к рис. 5.2.

Последнее ответвление отображается, если `v-else` и `v-else-if` равны `false`. Оно предусмотрено на случай, когда не выполняется ни одно иное условие. Мы хотим, чтобы в этой ситуации рядом с кнопкой корзины выводилось сообщение Buy Now! (рис. 5.3). Приведите элемент `span`, добавленный ранее в файл `index.html`, к следующему виду (листинг 5.3).

Рис. 5.3. Выводится сообщение Buy Now!

Листинг 5.3. Добавление нескольких сообщений о наличии товара: chapter-05/multiple-inventory.html

```
<button class="btn btn-primary btn-lg"
  v-on:click="addToCart"
  v-if="canAddToCart">Add to cart</button>
  <button disabled="true" class="btn btn-primary btn-lg"
  v-else >Add to cart</button>
  <span class="inventory-message"
    v-if="product.availableInventory - cartItemCount === 0">All Out!
  </span>
  <span class="inventory-message"
    v-else-if="product.availableInventory - cartItemCount < 5">
    Only {{product.availableInventory - cartItemCount}} left!
  </span>
  <span class="inventory-message"
    v-else>Buy Now!
  </span>
```

Директива v-if, которая отображается только после исчерпания товара

Эта директива отображается, только если не сработало первое условие v-if

v-else отображается, если не сработала ни v-if, ни v-if-else

Работа с условными выражениями

При работе с директивами `v-if`, `v-else` и `v-else-if` следует помнить о нескольких вещах. Директива `v-else` всегда должна идти сразу за `v-if` или `v-else-if`. Между ними нельзя размещать никаких дополнительных элементов. То же самое относится к `v-else-if`. Если нарушить это правило, `v-else-if` или `v-else` просто не будут распознаны.

Помните, что директиву `v-else-if` можно использовать несколько раз в одном и том же блоке. Например, наше приложение могло бы отображать несколько разных

сообщений, когда товар близок к исчерпанию. Это можно было бы сделать с помощью директивы `v-else-if`.

Однако не переусердствуйте с условными выражениями и бизнес-логикой в своих шаблонах. При необходимости лучше работать с вычисляемыми свойствами и методами. Это сделает код более наглядным и понятным.

5.2. Циклический перебор товара

С момента создания зоомагазина в главе 2 мы ограничивались лишь одним наименованием товара. Пока этого достаточно, но, как только мы добавим новый товар, нужно будет как-то отобразить его в шаблоне. Кроме того, не помешало бы выводить внизу каждого продукта простой рейтинг в виде звезд. Директива `v-for` достаточно гибка, чтобы справиться с обеими задачами.

5.2.1. Добавление звездного рейтинга на основе диапазона v-for

Vue.js предоставляет директиву `v-for` для циклического перебора элементов. Мы уже сталкивались с ней в главе 4. Стоит отметить, что ее можно использовать не только с массивами, но и с объектами и даже компонентами. Проще всего передать ей целое число, в этом случае она выведет элемент заданное количество раз. Этот подход иногда называют диапазоном `v-for`.

Для начала добавим для нашего продукта систему рейтинга из пяти звезд. Для простоты применим элемент `span` с директивой `v-for`. `v-for` всегда имеет синтаксис вида `item in items`, где `items` — ссылка на массив перебираемых данных. Как видно на рис. 5.4, `item` является ссылкой на текущий элемент массива `items`.

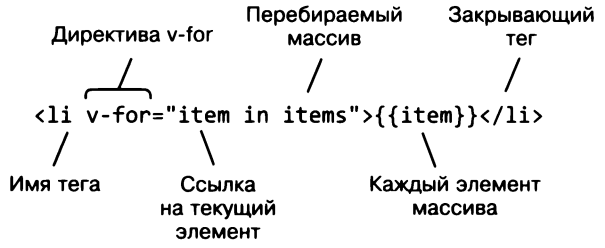


Рис. 5.4. Принцип работы ссылок в v-for

При использовании диапазонов `v-for` исходное число задает количество итераций, то есть сколько раз будет выведен элемент. На рис. 5.5 видно, что `n` повторяется пять раз.

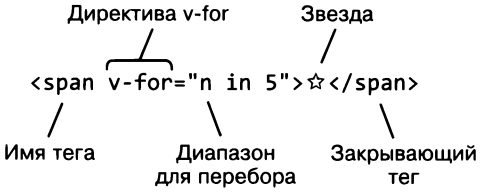


Рис. 5.5. Принцип работы диапазонов v-for

Листинг 5.4 отображает символ ☆ пять раз подряд. Мы также добавили `div` с классом `rating` (не забудьте загрузить файл `app.css` для книги, подробнее об этом — в приложении А). Добавьте этот код в файл `index.html` снизу от сообщений о наличии товара, которые мы создали в разделе 5.1.

Листинг 5.4. Добавление символа звезды с помощью v-for: chapter 05/star-symbol.html

```
<span class="inventory-message"
  v-else>Buy Now!
</span>
<div class="rating">
  <span v-for="n in 5">☆</span>
</div>
```

Выводим символ звезды пять раз подряд

После добавления звезд в шаблон обновите свой браузер. Страница должна выглядеть так, как на рис. 5.6.

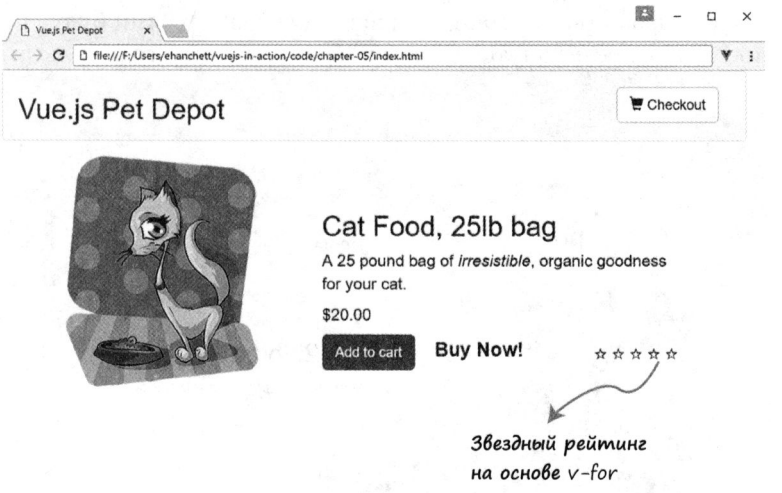


Рис. 5.6. Звездный рейтинг

Как видите, нам пока нечего показывать с помощью этого рейтинга: все звезды выводятся пустыми. Для их заполнения нам нужно динамически привязывать наш класс к CSS.

5.2.2. Привязка HTML-класса к звездному рейтингу

Vue.js позволяет динамически управлять классами в HTML-элементах шаблона. Чтобы определить, какой класс должен применяться, мы можем передавать объекты, массивы, выражения, методы и даже вычисляемые свойства.

Сначала следует отредактировать свойство `product` в объекте данных и добавить в него рейтинг. Последний будет определять, сколько звезд имеет товар. Откройте файл `index.html` и найдите свойство `product` снизу от `order`. Добавьте внизу свойство `rating`, как показано в листинге 5.5.

Листинг 5.5. Добавление свойства product: chapter 05/add-product.js

```
product: {
  id: 1001,
  title: "Cat Food, 25lb bag",
  description: "A 25 pound bag of <em>irresistible</em>,
                organic goodness for your cat.",
  price: 2000,
  image: "assets/images/product-fullsize.png",
  availableInventory:10,
  rating: 3              <-- Новое свойство rating
},
```

Теперь нужно отобразить рейтинг на экране. Проще всего сделать это с помощью CSS и небольшого вкрапления JavaScript. При добавлении CSS-класса звезда внутри элемента `span` станет черной. В нашем случае черными будут первые три звезды, а последние две останутся белыми. Пример итогового результата показан на рис. 5.7.

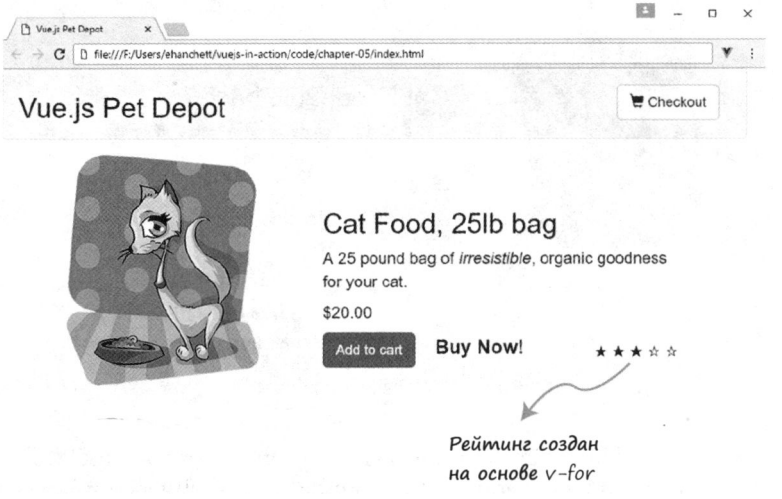


Рис. 5.7. Рейтинг кошачьего корма с закрашенными звездами

Как я уже упоминал, для определения того, какой класс следует применять, можно использовать метод, которому передается заданный диапазон `v-for`.

Добавим новый метод, который считывает рейтинг товара и возвращает `true`, если в `span` необходимо добавить CSS-класс, делающей звезду черной. Для этого методу следует передать переменную `n`, которая берется из диапазона `v-for`: `<span v-for="n in 5">☆</span>`. В шаблоне этого не видно, но `n` инкрементируется от 1 до 5. На первой итерации `n` равно 1, на второй — 2 и т. д. Мы знаем, что на протяжении жизни цикла `n` принимает значения 1, 2, 3, 4 и 5. С помощью нехитрых вычислений можно определить, нужно ли закрашивать звезду.

В нашем примере `n` изначально равно 1, а `this.product.rating` всегда содержит 3; 3 – 1 = 2 — это точно не меньше нуля, поэтому мы возвращаем `true` и добавляем CSS-класс. На следующей итерации `n` равно 2; 3 – 2 = 1 — это тоже не меньше нуля, поэтому опять возвращаем `true`. На следующей итерации `n` равно 3; 3 – 3 = 0, следовательно, класс снова добавляется. На четвертой итерации `n` равно 4; 3 – 4 = –1, поэтому метод возвращает `false`. Проще простого. Откройте файл `index.html` и добавьте в начало объекта `methods` метод `checkRating`, как показано в листинге 5.6.

Листинг 5.6. Метод, определяющий, нужно ли добавлять класс: chapter 05/check.js

```
methods: {
  checkRating(n) {
    return this.product.rating - n >= 0;
},
```

Возвращает true или false в зависимости от рейтинга и n

Чтобы собрать воедино систему рейтинга, следует добавить директиву `v-bind:class` в элемент `span`. Если метод возвращает `true`, она будет создавать новый класс `rating-active`. В противном случае она будет проигнорирована. В этом примере мы передаем в `v-bind:class` объект. Истинность метода `checkRating` определит, нужно ли добавлять класс `rating-active`. Поскольку все это происходит в цикле, мы должны передать также значение `n`, которое увеличивается с каждой итерацией.

Обновите элемент `span` в файле `index.html`, добавив в него директиву `v-bind:class`, как показано в листинге 5.7. Не забудьте поместить `rating-active` в кавычки, иначе в консоли появится сообщение об ошибке.

Листинг 5.7. Добавление привязки для класса: chapter 05/add-class-bind.html

```
<span class="inventory-message"
  v-else>Buy Now!
</span>
<div class="rating">
  <span  v-bind:class="{'rating-active': checkRating(n)}"
         v-for="n in 5" >☆
    </span>
</div>
```

Привязка для rating-active определяется методом checkRating

Это простейший пример связывания HTML-классов. Vue.js позволяет добавлять сразу несколько классов и использовать массивы и компоненты. Подробнее об этом говорится в официальном руководстве по привязке классов и стилей на странице ru.vuejs.org/v2/guide/class-and-style.html.

5.2.3. Информация о товарах

До сих пор мы работали лишь с одним товаром, но в настоящих зоомагазинах ассортимент исчисляется сотнями наименований. Так далеко заходить мы не станем. Посмотрим, насколько сложно будет добавить пять новых товаров и каким образом их можно перебирать на странице.

Для начала взгляните на объект `product`. Сейчас он находится внутри `index.html`, но на данном этапе его лучше вынести в отдельный файл.

Создадим новый файл `products.json` и сохраним его в папке `chapter-05`. Так нам будет проще работать с данными из основного приложения. При желании добавьте собственные товары, как делали это ранее в объекте `data`. Но если не хотите вводить их вручную, просто скопируйте `products.json` из архива с кодом для книги. Инструкции по загрузке кода находятся в приложении А. В листинге 5.8 показаны товары в файле `products.json`.

Листинг 5.8. Товары в файле products.json: chapter 05/products.json

```
{
  "products":[                          ← Массив товаров в JSON
    {
      "id": 1001,                       ← Первый товар
      "title": "Cat Food, 25lb bag",
      "description": "A 25 pound bag of <em>irresistible</em>,
                      organic goodness for your cat.",
      "price": 2000,
      "image": "assets/images/product-fullsize.png",
      "availableInventory": 10,
      "rating": 1
    },
    {                                   ← Второй товар
      "id": 1002,
      "title": "Yarn",
      "description": "Yarn your cat can play with for
                      a very <strong>long</strong> time!",
      "price": 299,
      "image": "assets/images/yarn.jpg",
      "availableInventory": 7,
      "rating": 1
    },
    {                                   ← Третий товар
      "id": 1003,
      "title": "Kitty Litter",
      "description": "Premium kitty litter for your cat.",
      "price": 1100,
      "image": "assets/images/cat-litter.jpg",
      "availableInventory": 99,
      "rating": 4
    },
```

```
    {                                          Четвертый товар
      "id": 1004,
      "title": "Cat House",
      "description": "A place for your cat to play!",
      "price": 799,
      "image": "assets/images/cat-house.jpg",
      "availableInventory": 11,
      "rating": 5
    },
    {                                          Пятый товар
      "id": 1005,
      "title": "Laser Pointer",
      "description": "Drive your cat crazy with
                      this <em>amazing</em> product.",
      "price": 4999,
      "image": "assets/images/laser-pointer.jpg",
      "availableInventory": 25,
      "rating": 1
    }
  ]
}
```

После создания или загрузки файла `products.json` и сохранения его в корневой папке главы 5 вам нужно внести определенные изменения. Если вы последовательно выполняете приведенные здесь примеры, то приложение, скорее всего, загружается локально, прямо с жесткого диска. В этом нет ничего плохого, за исключением одного момента: производители браузеров соблюдают политику безопасности, которая затрудняет загрузку файла `products.json`. Чтобы сделать все правильно, придется создать веб-сервер.

ЗАБЕГАЯ ВПЕРЕД

Приложение, запущенное на локальном веб-сервере, может легко загружать JSON-файлы с жесткого диска, не нарушая политику безопасности. В последующих главах мы будем использовать Vue CLI. Эта утилита командной строки берет на себя создание веб-сервера. Но на данном этапе нам хватит простого облегченного npm-модуля под названием http-server, который обладает аналогичными возможностями. Инструкции по установке npm найдете в приложении A.

Воспользуемся npm для создания веб-сервера. Откройте окно терминала и установите модуль http-server с помощью следующей команды:

```
$ npm install http-server -g
```

После завершения установки перейдите в каталог главы 5. Команда, представленная далее, запускает сервер, который работает на порте 8000 и раздает файл `index.html`:

```
$ http-server -p 8000
```

Если эта команда возвращает ошибки, проверьте, не запущена ли на порте 8000 какая-то другая программа. Можете попробовать поменять порт на 8001.

Если все прошло успешно, запустите свой любимый браузер и перейдите по адресу http://localhost:8000, чтобы открыть нашу веб-страницу. Если страница не отображается, еще раз проверьте командную строку на предмет ошибок. Вероятно, придется указать другой порт, если 8000 уже занят.

5.2.4. Импорт товаров из файла product.json

Помните хуки жизненного цикла Vue.js, которые мы изучили в главе 2? Нужно загрузить наш JSON-файл сразу после загрузки веб-страницы, и один из этих хуков идеально подходит для данной задачи. Догадываетесь, какой именно? Правильный ответ — `created`. Этот хук вызывается после создания экземпляра Vue. Мы можем загружать в нем JSON-файл, но для этого понадобится еще одна библиотека.

Axios — это HTTP-клиент на основе объектов `Promise`, предназначенный для браузеров и Node.js. Он обладает несколькими полезными возможностями, такими как преобразование данных из формата JSON. Добавим эту библиотеку в проект. Создайте в файле `index.html` новый тег `script` внутри `head` (листинг 5.9).

Листинг 5.9. Добавление тега script для Axios: chapter 05/script-tags.html

```
  <link rel="stylesheet"
        href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/
      ➥ bootstrap.min.css" integrity="sha384-BVYiiSIFeK1dGmJRAkycu
      ➥ HAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u"
        crossorigin="anonymous">
  <script src="https://cdnjs.cloudflare.com/ajax/libs/axios/0.16.2/axios.js">
</script>      ◄┐ Тег script со ссылкой
</head>         │ на CDN с Axios
```

Добавив этот тег, вы сможете использовать Axios в хуке `created`. Вставьте хук в файл `index.html` сразу после объекта `filters`. Нужно также написать код, который загружает файл `products.json` с жесткого диска и перезаписывает существующие данные о товарах. Обновите `index.html` и добавьте код Axios (листинг 5.10).

Листинг 5.10. Добавление тега Axios для создания хука жизненного цикла: chapter 05/axios-lifecycle.js

```
...
},
created: function() {                          ┐ Загружаем
  axios.get('./products.json')                 ◄┘ файл products.json
    .then((response) =>{
      this.products=response.data.products;    ◄┐ Добавляем ответ
      console.log(this.products);               │ в объект products
  });                                           ┘
},
```

Метод `axios.get` принимает путь, который в данном случае ведет к локальному файлу. В ответ возвращается объект `Promise`, содержащий метод `.then`. `Promise` выполняется или отклоняется, после чего возвращает объект с ответом. Согласно документации Axios этот ответ имеет свойство `data`. Ссылка `response.data.products` присваивается свойству `this.products`, которое принадлежит экземпляру Vue. Чтобы убедиться в том, что все прошло как следует, мы выводим некоторую информацию в консоль.

Если внимательно присмотреться к коду в листинге 5.10, можно заметить, что данные, взятые из JSON-файла, присваиваются свойству `this.products`, а не `this.product`. Чтобы код был более понятным, это свойство следует создать заранее.

Откройте файл `index.html` и найдите объект данных ближе к середине кода. Добавьте новое свойство `products` и удалите старое, `product`, оно больше не понадобится (листинг 5.11).

Листинг 5.11. Удаление свойства product, добавление свойства products: chapter 05/product-delete.js

```
  business: 'Business Address',
  home: 'Home Address',
  gift:'',
  sendGift: 'Send As A Gift',
  dontSendGift: 'Do Not Send As A Gift'
},
products: [],
```

Объект orders остается неизменным

Новый массив products вместо объекта product

Теперь, если обновить страницу, произойдет ошибка, поскольку мы удалили объект `product`. Исправим это чуть позже, когда добавим директиву `v-for` для перебора товаров.

5.2.5. Рефакторинг приложения с использованием директивы v-for

Прежде чем начать циклически перебирать товары, нужно внести небольшие изменения в классы `div`, которые отвечают за визуальные стили. Мы работаем с Bootstrap 3 и хотим, чтобы каждый товар размещался в отдельной строке (так как теперь у нас больше одного наименования). Итоговый результат должен выглядеть как на рис. 5.8.

Обновите `index.html` и найдите директиву `v-else`, которая выводит страницу оформления покупки. Добавьте еще один тег `div` для новой строки (листинг 5.12).

Листинг 5.12. Исправление таблицы стилей для Bootstrap: chapter 05/bootstrap-fix.html

```
<div v-else>
  <div class="row">
```

Новая строка для Bootstrap

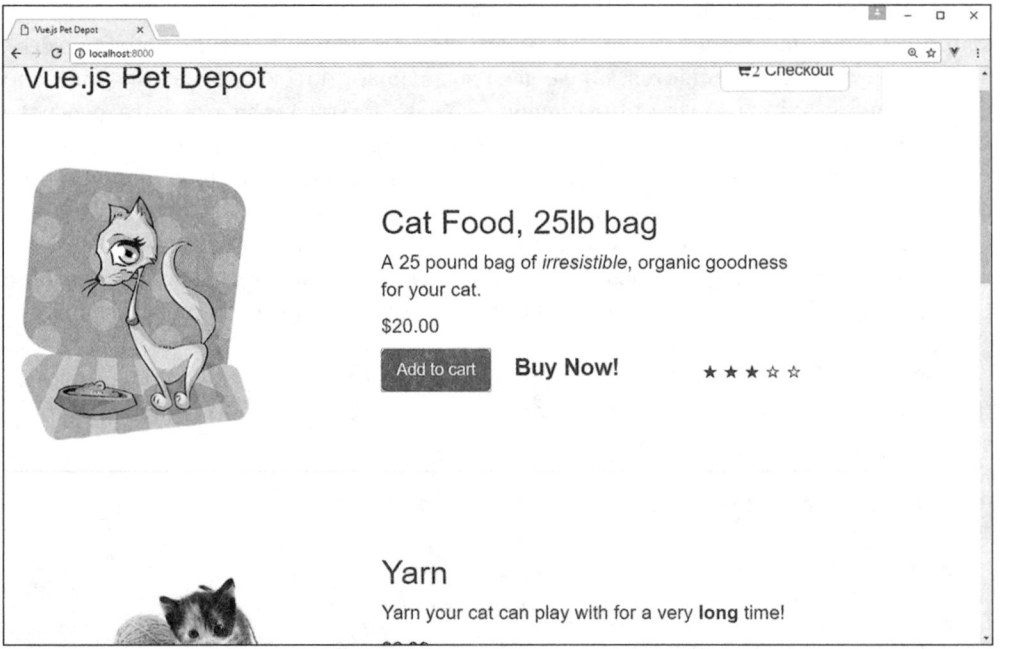


Рис. 5.8. Итоговый внешний вид товаров

Нам нужно переместить элемент `div` с классом `row`, который находится прямо перед директивой `v-if` с условием `showProduct`. Поместите его сразу за `showProduct`, как показано в листинге 5.13. Не забудьте обновить файл `imdex.html`.

Листинг 5.13. Исправление таблицы стилей для Bootstrap: chapter 05/bootstrap-fix-v-if.html

```
<div v-if="showProduct">
  <div class="row product">
```

Помещаем div с классом row под showProduct

Устранив незначительные проблемы с CSS/HTML, мы можем добавить директиву `v-for`, которая будет перебирать и отображать на нашей странице все товары. В данном примере используем синтаксис `product in products`, где `products` — это объект, загруженный ранее, а `product` — ссылка на каждый отдельный товар в `products`. Следует также с помощью Bootstrap изменить ширину столбцов, чтобы товары выводились более аккуратно.

Откройте `index.html` и добавьте директиву `v-for` снизу от `v-if` с условием `showProduct`. Не забудьте закрыть тег `div` внизу страницы, как показано в листинге 5.14.

Мы добавили директиву `v-for`, однако осталось несколько проблемных моментов. Метод `checkRating` и вычисляемое свойство `canAddToCart` все еще ссылаются на `this.product`. Но нам нужно, чтобы этот код работал с массивом `this.products`.

Листинг 5.14. Добавление директивы v-for для products: chapter 05/v-for-product.html

```
<div v-if="showProduct">
  <div v-for="product in products">
    <div class="row">
     <div class="col-md-5 col-md-offset-0">
       <figure>
          <img class="product" v-bind:src="product.image">
        </figure>
     </div>
      <div class="col-md-6 col-md-offset-0 description">
...
    </div><!-- Конец of row -->
    <hr />
  </div><!-- Конец of v-for -->
</div><!-- Конец showProduct -->
```

Перебираем все товары с помощью директивы v-for

Меняем ширину столбца на 5 без смещения

Меняем ширину столбца без смещения

Добавляем горизонтальную линию

Закрывающий тег директивы v-for

Это может оказаться не так уж просто. Сначала исправим метод `checkRating`, который определяет, сколько звезд имеет каждый товар. Передадим ему ссылку `product`. Обновите код `checkRating` внутри `index.html` (листинг 5.15).

Листинг 5.15. Обновление checkRating с использованием данных о товаре: chapter 05/check-rating.js

```
methods: {
  checkRating(n, myProduct) {
    return myProduct.rating - n >= 0;
},
```

Новый метод checkRating, который принимает product

Теперь нужно откорректировать шаблон и передать товар в обновленный метод. Откройте `index.html` и найдите метод `checkRating` под сообщениями о наличии товара. Добавьте к нему `product`, как показано в листинге 5.16.

Листинг 5.16. Обновление шаблона для вывода рейтинга: chapter 05/update-template.html

```
<span class="inventory-message"
  v-else>Buy Now!
</span>
<div class="rating">
  <span  v-bind:class="{'rating-active': checkRating(n, product)}"
    v-for="n in 5" >☆
  </span>
</div>
```

Делаем так, чтобы метод checkRating принимал product

Скопируйте картинки в `assets/images` из архива для этой главы в свою локальную папку `assets/images`, если вы этого еще не сделали. Не забудьте также поместить файл `app.css` в папку `assets/css`.

Чтобы завершить рефакторинг приложения, следует поправить вычисляемое свойство `canAddToCart`. Оно делает неактивной кнопку Add to Cart (Добавить в корзину) после того, как число добавленных элементов сравняется с количеством доступных единиц товара.

Вам, наверное, интересно, как мы этого добьемся? Ранее у нас было лишь одно наименование товара, поэтому вычисление оставшихся запасов не вызывало труда. Теперь же нужно пройтись по каждому элементу в корзине и проверить, можем ли мы добавить еще один.

Это не так сложно, как кажется. Вычисляемое свойство `canAddToCart` необходимо превратить в метод, который принимает товар. Затем мы обновим условное выражение так, чтобы оно извлекало количество элементов. Для этого воспользуемся методом под названием `cartCount`, возвращающим количество элементов с заданным идентификатором. `cartCount` проходится по массиву `cart` с помощью обычного цикла `for`. При каждом совпадении он инкрементирует переменную `count`, которая затем возвращается в качестве результата.

Обновите метод `canAddToCart` в файле `index.html` (листинг 5.17). Можете переместить его из раздела `computed` в `methods`. Не забудьте также создать метод `cartCount`.

Листинг 5.17. Обновление canAddToCart и добавление метода cartCount: chapter 05/update-carts.js

```
canAddToCart(aProduct) {
  return aProduct.availableInventory > this.cartCount(aProduct.id);
},
cartCount(id) {
  let count = 0;
  for(var i = 0; i < this.cart.length; i++) {
    if (this.cart[i] === id) {
      count++;
    }
  }
  return count;
}
```

Определяет, превышает ли объем запасов количество элементов в корзине

Новый метод cartCount, возвращающий количество элементов в корзине с заданным ID

Цикл, проверяющий каждый элемент в корзине

Чтобы завершить обновление метода `canAddToCart`, нужно найти его в шаблоне и передать ему товар. То же самое следует сделать с методом `addToCart`. Откройте `index.html` и передайте ссылку `product` в методы `canAddToCart` и `addToCart` (листинг 5.18).

Листинг 5.18. Обновление шаблона для canAddToCart: chapter 05/update-can-add-cart.html

```
<button class=" btn btn-primary btn-lg"
v-on:click="addToCart(product)"
v-if="canAddToCart(product)">Add to cart</button>
```

Метод addToCart должен принимать товар

Метод canAddToCart должен принимать товар

Это довольно тривиальные изменения. Ранее мы обновили участок шаблона, где применяется метод `addToCart`. Теперь нужно исправить сам метод, чтобы он добавлял идентификатор товара в корзину. Воспользуемся для этого операцией изменения массива `push` (листинг 5.19).

Операции изменения

В Vue имеется множество операций изменения, которые применяются к массивам. Каждый массив в Vue заворачивается в объекты отслеживания. В случае изменения шаблон уведомляется и обновляется. Операции изменения изменяют содержимое массива, из которого они вызываются. Среди них можно выделить методы `push`, `pop`, `shift`, `unshift`, `splice`, `sort` и `reverse`.

Будьте осторожны: Vue не способна отследить некоторые изменения. Это относится, например, к прямому присваиванию значений `this.cart[index] = newValue` и обновлению длины `this.item.length = newLength`. Подробнее об операциях изменения читайте в официальном руководстве, размещенном по адресу ru.vuejs.org/v2/guide/list.html#Методы-внесения-изменений.

Листинг 5.19. Обновление метода addToCart: chapter 05/update-add-to-cart.js

```
addToCart(aProduct) {
  this.cart.push( aProduct.id );
},
```

Помещает ID товара в корзину

Теперь нажатие кнопки Add to cart (Добавить в корзину) не вызывает никаких проблем. При каждом нажатии идентификатор товара помещается в массив `cart`, одновременно автоматически обновляется счетчик добавленных элементов в верхней части экрана.

Последним этапом рефакторинга будет исправление сообщений о наличии товара, которые мы создали ранее. Дело в том, что для определения того, какое из них следует вывести, по-прежнему используется общее количество элементов в корзине. Нам нужно изменить этот код так, чтобы учитывались товары только заданного наименования.

Подставим вместо `cartItemCount` метод `cartCount`, который принимает идентификатор товара. Найдите сообщения в файле `index.html` и поменяйте выражения в директивах `v-if` и `v-else-if` так, чтобы они задействовали `cartCount` (листинг 5.20).

Листинг 5.20. Обновление сообщений о наличии товара: chapter 05/update-inventory.html

```
<span class="inventory-message"
  v-if="product.availableInventory - cartCount(product.id) === 0">
  All Out!
</span>
<span class="inventory-message"
  v-else-if="product.availableInventory - cartCount(product.id) < 5">
  Only {{product.availableInventory - cartCount(product.id)}} left!
</span>
<span class="inventory-message"
  v-else>Buy Now!
</span>
```

Новые выражения для директивы v-if с использованием cartCount

Новые выражения для директивы v-else-if с использованием cartCount

Вот и все! Теперь мы можем открыть страницу и посмотреть на результат. Перезагрузите свой браузер, не забыв запустить HTTP-сервер (`http-server -p 8000`). Вы должны увидеть обновленное приложение, которое берет товары из файла `products.json` и циклически выводит их с помощью директивы `v-for` (рис. 5.9).

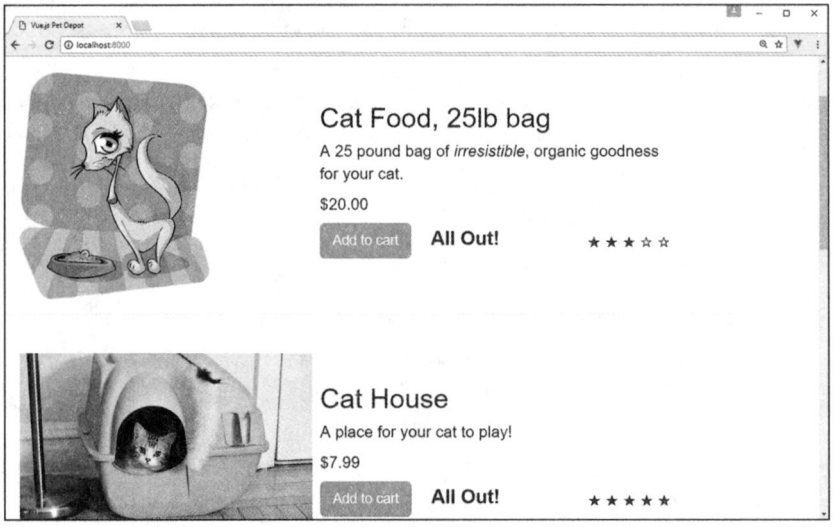


Рис. 5.9. Циклический вывод нескольких товаров из файла product.json

Убедитесь в том, что все работает как следует. Нажатие кнопки Add to cart (Добавить в корзину) должно привести к изменению сообщения. Когда счетчик товаров достигнет нуля, кнопка должна стать недоступной. Попробуйте отредактировать файл `products.json` и перезагрузить браузер. Все изменения должны отразиться на экране.

5.3. Сортировка записей

Часто при выводе массивов или, как в данном случае, объектов с помощью директивы `v-for` возникает необходимость в сортировке значений. Vue упрощает эту задачу. Создадим вычисляемое свойство, которое возвращает отсортированный результат.

Наше приложение загружает товары из JSON-файла. Ассортимент соответствует содержимому `products.json`. Добавим сортировку по названию, чтобы товары отображались в алфавитном порядке. Для этого нужно создать новое вычисляемое свойство `sortedProducts`. Но сначала обновим шаблон.

Найдите в файле `index.html` директиву `v-for`, которая выводит товары. Замените в ней объект `products` на `sortedProducts` (листинг 5.21).

Листинг 5.21. Добавление сортировки в шаблон: chapter 05/add-in-sort.html

```
<div v-if="showProduct">
  <div v-for="product in sortedProducts">
    <div class="row">
```

Добавляем новое вычисляемое свойство sortedProducts

Теперь нужно создать вычисляемое свойство `sortedProducts`, которое используется в шаблоне. Но при этом следует учесть важный момент: данные в свойстве `this.products` сразу после запуска приложения могут оказаться недоступными, поскольку загружаются асинхронно из файла `products.json` в хуке жизненного цикла `created`. Чтобы это не превратилось в проблему, поместим код в блок `if`, который проверяет существование товаров.

Создадим функцию сравнения, которая упорядочивает значения в алфавитном порядке. Она будет применяться в нашем массиве для сортировки элементов по их заголовкам (листинг 5.22).

Листинг 5.22. Вычисляемое свойство sortedProducts: chapter 05/sort-products-comp.js

```
sortededProducts() {
  if(this.products.length > 0) {
    let productsArray = this.products.slice(0);
    function compare(a, b) {
      if(a.title.toLowerCase() < b.title.toLowerCase())
        return -1;
      if(a.title.toLowerCase() > b.title.toLowerCase())
        return 1;
      return 0;
    }
    return productsArray.sort(compare);
  }
}
```

Преобразует объект в массив с помощью стандартного метода slice

Функция сравнения на основе заголовков

Возвращает новый массив товаров

Этого должно быть достаточно. После обновления страницы все товары должны быть отсортированы по названию. Вывод отсортированного массива показан на рис. 5.10. Прокрутите вниз и убедитесь в том, что на странице присутствуют все продукты. Еще раз проверьте, чтобы все работало так, как задумано.

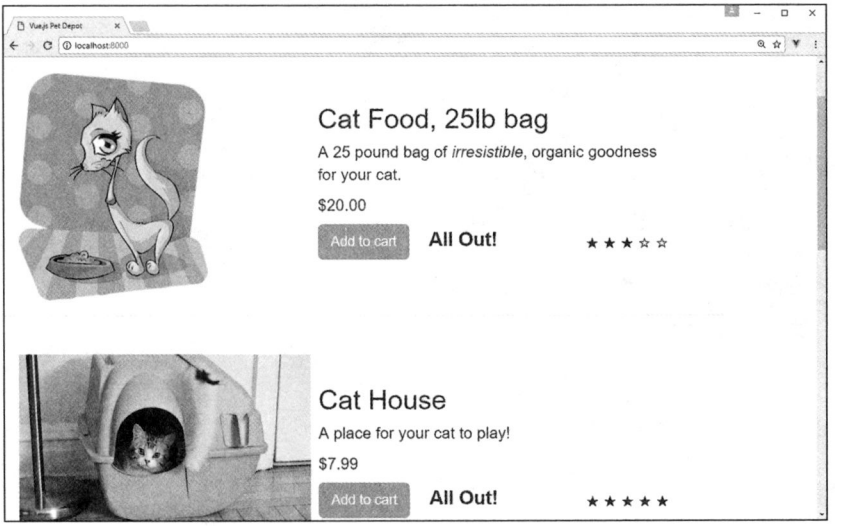


Рис. 5.10. Массив отсортированных товаров

Упражнение

Используя знания, приобретенные в этой главе, ответьте на вопросы: что такое диапазон `v-for` и чем он отличается от обычной директивы `v-for`?

Ответ ищите в приложении Б.

Резюме

- Условные выражения в Vue создаются с помощью директив `v-if`, `v-else-if` и `v-else`. Время от времени, хоть и не так часто, мы будем использовать также директиву `v-show`.
- Директива `v-for` чрезвычайно гибкая. Она позволяет многократно повторять HTML-разметку, перебирая диапазоны положительных целых чисел (то есть начиная с 1), массивы элементов, свойства объектов в виде пар «ключ — значение», шаблоны и компоненты Vue. Для циклического перебора элементов можно использовать любые выражения.
- Мы легко отсортируем значения с помощью вычисляемых свойств. Этот подход можно сочетать с директивой `v-for` для сортировки вывода.

6 Работа с компонентами

В этой главе

- Иерархия родительских и дочерних компонентов.
- Локальная и глобальная регистрация.
- Применение и проверка корректности входных параметров.
- Добавление пользовательских событий.

В предыдущих главах мы познакомились с условными выражениями, циклами и списками. Циклы применялись для того, чтобы избежать повторения кода и упростить разработку. С помощью условных выражений мы выводили разные сообщения в зависимости от действий пользователя. Такой подход работает, но вы уже могли заметить, что приложение стало довольно громоздким и состоит более чем из 300 строк кода. Файл `index.html`, обновляемый в каждой главе, содержит вычисляемые свойства, фильтры, методы, хуки жизненного цикла свойства с данными. Во всем этом легко запутаться.

Чтобы как-то решить эту проблему, следует разделить код на автономные части. Каждая из них должна поддерживать многократное использование и позволять передачу свойств и событий. В этом нам помогут компоненты Vue.js. Сначала я объясню принцип их работы и приведу несколько примеров.

Вы познакомитесь с механизмом локальной и глобальной регистрации компонентов, научитесь передавать им входные параметры и проверять корректность переданных значений. В конце мы создадим шаблоны для приложения и определим пользовательские события.

Если вас интересует судьба зоомагазина — не волнуйтесь. Мы вернемся к нему в следующей главе, посвященной однофайловым компонентам, инструментам сборки и Vue-CLI.

6.1. Что такое компоненты

В Vue.js *компонент* представляет собой мощную концепцию, которая делает код более лаконичным и простым. Большинство Vue-приложений состоят из одного или нескольких компонентов. Такой подход позволяет вычленить повторяющиеся фрагменты кода в отдельные логические модули, которые несут в себе определенный

смысл. Каждый компонент можно многократно использовать в одном и том же приложении. Это такой набор элементов, доступ к которым осуществляется через единый интерфейс. В некоторых случаях компоненты могут выглядеть как стандартные HTML-элементы, для этого предусмотрен специальный атрибут `is` (рассмотрим его позже в этой главе).

На рис. 6.1 показан простой пример объединения нескольких HTML-тегов в единый компонент `my-component`. Ему принадлежит вся разметка между открывающим и закрывающим тегами `div`. Стоит также отметить, что компоненты могут быть самозакрывающимися, например `<my-component/>`. Это требует поддержки со стороны браузера или применения однофайловых компонентов, которые мы обсудим позже.

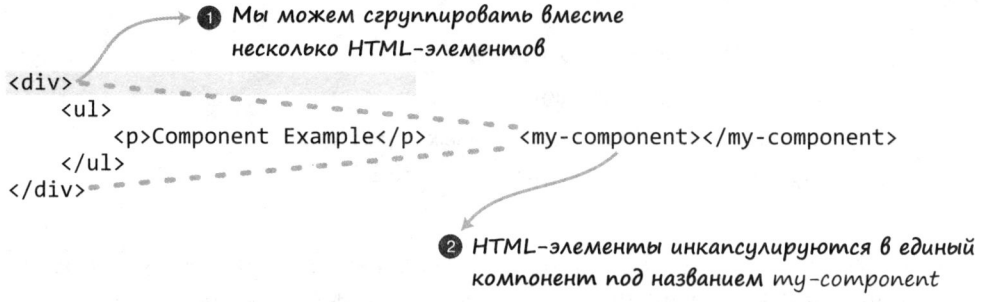


Рис. 6.1. Пример инкапсуляции кода в отдельном компоненте

6.1.1. Создание компонентов

Чтобы создать глобальный компонент, нужно поместить его перед экземпляром Vue. Как видно в листинге 6.1, глобальный компонент `my-component` определяется непосредственно перед созданием с помощью `new Vue`.

Чтобы вывести в компоненте какую-то информацию, следует добавить свойство `template` — это то место, где находится HTML-разметка. Имейте в виду, что любой шаблон должен быть заключен внутрь тега. В примере мы задействуем тег `<div>`. Если этого не сделать, в консоли появится ошибка, а шаблон не будет отрисован на экране.

Последнее, что нужно сделать для отображения компонента, — добавить его в родительский шаблон. Для этого мы укажем пользовательский тег `<my-component></mycomponent>` внутри точки входа в приложение, `<div id="app">` (см. листинг 6.1). В ходе чтения этой главы старайтесь самостоятельно воспроизводить приводимые примеры. Сохраняйте их в файлы с расширением `.html` и загружайте в браузере.

Листинг 6.1. Создаем первый глобальный компонент: chapter-06/global-component-example.html

```
<!DOCTYPE html>
<html>
<head>
<script src="https://unpkg.com/vue"></script>
</head>
```

Тег script для Vue.js

```
<body>
<div id="app">
  <my-component></my-component>
</div>
<script>
Vue.component('my-component', {
  template: '<div>Hello From Component</div>'
});

new Vue({
  el: "#app"
});
</script>
</body>
</html>
```

Добавление компонента в шаблон

Регистрация глобального компонента в Vue

Отрисовка шаблона компонента

Создание экземпляра Vue

Разумеется, наше приложение нельзя назвать полезным. Открыв файл в браузере, вы увидите сообщение Hello From Global Component. Это тривиальный пример работы компонентов, который позволит вам усвоить основную идею. Рассмотрим локальную регистрацию и попробуем найти отличия.

Специальный атрибут is

Использование компонентов в рамках DOM связано со специфическими ограничениями. Некоторые HTML-теги, такие как `<ul>`, `<ol>`, `<table>` и `<select>`, могут содержать только строго определенные элементы. Если не соблюдать эти правила, документ пометит наши компоненты как некорректное содержимое. Решить эту проблему поможет атрибут `is`. Компонент можно добавить в сам элемент, не помещая его внутрь тегов, например `<table><tr is="my-row"></tr></table>`. В этом случае элемент `tr` будет связан с компонентом `my-row`. Мы можем создать собственный компонент `tr` и наделить его любыми возможностями по своему усмотрению. Это ограничение не относится к строчным тегам, шаблону x-template и однофайловым компонентам. Больше об этом атрибуте говорится в официальном руководстве, расположенном по адресу https://ru.vuejs.org/v2/guide/components.html#Особенности-парсинга-DOM-шаблона.

6.1.2. Локальная регистрация

Локальная регистрация ограничивает область видимости одним экземпляром Vue. Ее можно выполнять внутри опций экземпляра компонента. Компонент, зарегистрированный локально, доступен только тому экземпляру Vue, который его зарегистрировал.

Простой пример локального компонента показан в листинге 6.2. Он похож на глобальный компонент, который мы регистрировали ранее. Основное отличие — новый раздел в опциях экземпляра под названием `components`.

В этом разделе объявляются все компоненты, которые нужны данному экземпляру Vue. Каждое объявление представляет собой пару «ключ — значение». В качестве

ключа всегда выступает имя компонента, на которое вы будете ссылаться в родительском шаблоне, а значение — это его определение. В листинге 6.2 компонент называется `my-component`. Значение равно `Component`, это константа `const`, определяющая содержимое компонента.

Можете выбрать любое другое имя, но, как уже упоминалось, старайтесь использовать синтаксис в шашлычной нотации (слова в нижнем регистре, разделенные дефисами). Это рекомендуемый подход.

Листинг 6.2. Регистрация локального компонента: chapter-06/local-component-example.html

```
<!DOCTYPE html>
<html>
<head>
<script src="https://unpkg.com/vue"></script>
</head>
  <body>
  <div id="app">
    <my-component></my-component>
  </div>
  <script>
      const Component = {
          template: '<div>Hello From Local Component</div>'
      }

      new Vue({
          el: '#app',
          components: {'myComponent': Component}
      });
  </script>
  </body>
</html>
```

Константа с объявлением компонента

Шаблон, который будет отображаться на месте этого компонента

Раздел опций components с объявлением компонентов

Выбор между шашлычной и верблюжьей нотациями

Называйте свои компоненты как угодно, но при этом имейте в виду, что в HTML-шаблонах необходимо использовать эквивалент выбранного вами имени в формате *шашлычной нотации* (слова в нижнем регистре, разделенные дефисами). Представьте, что вы зарегистрировали компонент `myComponent`. Это имя соответствует синтаксису верблюжьей нотации. В HTML-шаблонах оно должно переводиться в шашлычную нотацию и выглядеть как `<my-component>`. То же самое касается разновидности верблюжьей нотации под названием *«паскальная нотация»*. Если назвать компонент `MyComponent`, в HTML-шаблоне он будет доступен как `<my-component>`. Это правило распространяется и на входные параметры, о которых мы поговорим позже.

Вскоре, когда вы познакомитесь с однофайловыми компонентами, эта проблема отпадет сама собой. А пока что придерживайтесь формата шашлычной нотации. Подробное сравнение шашлычной и верблюжьей нотаций можно найти в официальной документации, расположенной по адресу https://ru.vuejs.org/v2/guide/components-props.html.

6.2. Иерархия компонентов

Представьте, что вы разрабатываете систему комментариев. Она должна выводить тексты, отправленные всеми пользователями. Каждый комментарий должен содержать имя пользователя, время/дату отправки и сам текст. Пользователи могут редактировать и удалять свои комментарии.

Очевидное решение — применение директивы `v-for`. Именно так мы перебирали список товаров в предыдущей главе. Это могло бы сработать, но представьте, что требования поменялись и теперь нужно выводить древовидные комментарии или создать новый механизм голосования. В этом случае код начнет быстро усложняться.

Проблему можно решить с помощью компонентов. Комментарии будут выводиться внутри компонента `comment-list`, а родитель (корневой экземпляр Vue) станет отвечать за остальную часть приложения. Родитель также должен содержать метод для получения с сервера всех данных о комментариях. Эти данные передаются дочернему компоненту `comment-list`, который отвечает за их отображение.

Помните, что каждый компонент имеет изолированную область видимости, поэтому не должен обращаться к родительским данным напрямую. В связи с этим данные всегда передаются сверху вниз. В Vue.js для этого используются *входные параметры*, которые в коде обозначаются `props` (от *properties* — «свойства»). Дочерние компоненты должны явно задекларировать все свои входные параметры в опции `props`, которая является частью экземпляра Vue.js и содержит массив вида `props: ['comment']`. В данном примере `'comment'` — это параметр, который будет передан компоненту. Если бы у нас было несколько входных параметров, мы бы разделили их запятыми. Этот механизм передачи данных односторонний и всегда направлен от родительского компонента к дочернему (рис. 6.2).

Рис. 6.2. Родительский компонент передает данные дочернему

В главе 4 мы рассматривали двунаправленное связывание данных с помощью `v-model` на примере элементов формы и текстовых полей. Изменения, вносимые в элемент `v-model`, отражаются на свойствах экземпляра Vue.js и наоборот. В компонентах применяется однонаправленное связывание. Когда родитель обновляет

свои свойства, изменения передаются дочернему компоненту, обратное направление не работает. Благодаря этому важному отличию потомок не может случайно изменить состояние родителя. Если он попытается это сделать, в консоли появится сообщение об ошибке (рис. 6.3).

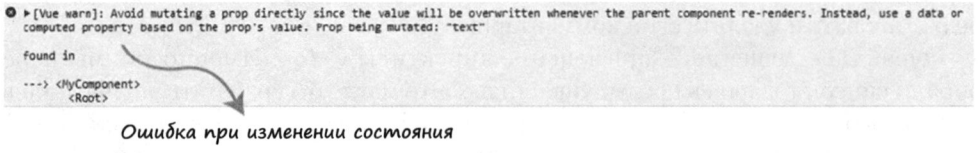


Рис. 6.3. Предупреждение о том, что входные параметры нельзя изменять напрямую

Обратите внимание на то, что все значения передаются по ссылке. Если объект или массив изменится в дочернем компоненте, это затронет состояние родителя. Часто это становится нежелательным побочным эффектом, которого нужно избегать. Изменения следует производить только в родительском компоненте. Позже в этой главе вы узнаете, как передавать данные от потомка к родителю с помощью событий.

6.3. Использование входных параметров для передачи данных

Как упоминалось ранее, входные параметры предназначены для передачи данных от родительских компонентов к дочерним. Данный механизм рассчитан лишь на однонаправленное связывание. Это переменные, которые принадлежат родителю и могут назначаться только в его пределах.

Входные параметры можно проверять на корректность. Мы способны сделать так, чтобы передаваемые значения отвечали определенным критериям. Но об этом чуть позже.

6.3.1. Передача литералов через входные параметры

Простейший способ использования входных параметров состоит в передаче литералов, то есть обычных строк. Компонент, как и прежде, создается в шаблоне, а входной параметр указывается внутри угловых скобок в виде дополнительного атрибута: `<my-component text="World"></my-component>`. В этом примере текст представляет собой строку и передается в созданный нами входной параметр `text`. Шаблон интерполирует его в фигурных скобках.

Новички часто совершают одну и ту же ошибку: при попытке передать во входной параметр настоящее значение оказывается, что компонент получил литерал.

Чтобы этого не произошло, всегда нужно задействовать директиву `v-bind`. Вы увидите, как это делается, в следующем разделе.

В листинге 6.3 показан пример передачи литерала через входной параметр. Скопируйте его в свой текстовый редактор и попробуйте запустить. Компонент `my-component` должен получить переданное значение `"World"`. Вы можете отобразить его с помощью параметра `text`, как показано далее.

Листинг 6.3. Передача литерала в компонент: chapter 06/literal-props.html

```
<!DOCTYPE html>
<html>
<head>
<script src="https://unpkg.com/vue"></script>
</head>
  <body>
  <div id="app">
  <my-component text="World"></my-component>
  </div>
  <script>
  const MyComponent= {
    template:'<div>Hello {{text}}! </div>',
    props:['text']
  }
  new Vue({
    el: "#app",
    components: {'my-component': MyComponent}

  });
  </script>
</html>
```

Компонент с переданным литералом

Шаблон выводит Hello и переданный входной параметр

6.3.2. Динамические входные параметры

Динамическими называются входные параметры, которые привязаны к изменяемым свойствам (в отличие от литералов, представляющих собой статический текст). Для корректной передачи таких значений используется директива `v-bind`. Обновим пример из предыдущего раздела и передадим сообщение в новый входной параметр `text`.

Компонент `<my-component v-bind:text="message"></my-component>` содержит новый атрибут с директивой `v-bind`. Это позволяет привязать параметр `text` к сообщению `message`, возвращаемое функцией данных.

Если вы внимательно читали предыдущие главы, то могли заметить, что в листинге 6.4 `data` больше не является объектом (`data: { }`). Это сделано намеренно. Компоненты ведут себя немного иначе, и данные должны быть представлены в виде функции, а не объекта.

Если создать в компоненте `my-component` объект данных, в консоли появится ошибка. Теперь, чтобы оставаться последовательными, станем использовать `data` в виде функции как в компонентах, так и в корневом экземпляре Vue.

Листинг 6.4 иллюстрирует применение динамических входных параметров. Скопируйте его в свой текстовый редактор и попробуйте запустить.

Листинг 6.4. Использование динамических входных параметров: chapter 06/dynamic-props.html

```
<!DOCTYPE html>
<html>
<head>
<script src="https://unpkg.com/vue"></script>
</head>
  <body>
  <div id="app">
  <my-component :text="message"></my-component>
  </div>
  <script>
  const MyComponent= {
    template:'<div>Hello {{text}}! {{tester1}} </div>',
    props:['text'],
    data() {
      return {
        tester1: 'Testing text'
      }
    }
  }
  new Vue({
    el: "#app",
    components: {'my-component': MyComponent},
    data() {
      return {
        message: 'From Parent Component!'
      }
    }

  });
  </script>
</html>
```

Директива v-bind привязывает message родителя к text потомка

Это шаблон, который отображает параметр text

Функция данных, возвращающая message

Прежде чем двигаться дальше, представьте, что нужно добавить в программу три счетчика, каждый из которых начинается с нуля и инкрементируется независимо от остальных. Все счетчики представлены в виде кнопок, нажатие на которые приводит к инкрементации. Как это реализовать с помощью компонентов?

Возьмем за основу код из листинга 6.4. Добавьте в компонент `MyComponent` функцию данных и счетчик. Очевидным решением было бы использовать глобальную переменную. Давайте так и поступим и посмотрим, что из этого получится.

Как видно в листинге 6.5, мы добавили компонент три раза. А также создали глобальную константу с объектом `counter`, свойство которого равно нулю. В шаблоне

имеется простая привязка к событию `click` на основе директивы `v-on`: при каждом щелчке счетчик будет увеличиваться на 1.

Листинг 6.5. Динамические входные параметры с глобальным счетчиком: chapter 06/dynamic-props-counter.html

```
<!DOCTYPE html>
<html>
<head>
<script src="https://unpkg.com/vue"></script>
</head>
  <body>
  <div id="app">
  <my-component></my-component>
  <my-component></my-component>
  <my-component></my-component>
  </div>
  <script>
  const counter = {counter: 0}
  const MyComponent= {
    template:'<div><button v-on:click="counter +=
      1">{{counter}}</button></div>',
    data() {
      return counter;
    }
  }

  new Vue({
    el: "#app",
    components: {'my-component': MyComponent},
    data() {
      return {
        message: ''
      }
    }

  });
  </script>
</html>
```

Три экземпляра компонента

Глобальная константа counter

Счетчик инкрементируется при каждом щелчке

Функция данных возвращает глобальный счетчик

Откройте написанный код в браузере. Понажимайте кнопки, имеющиеся на странице, и посмотрите, что происходит (рис. 6.4).

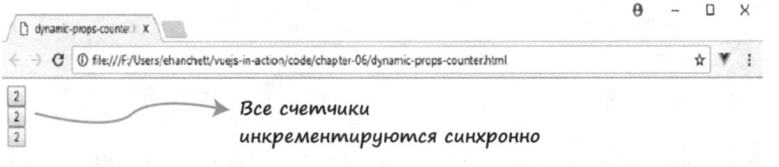


Рис. 6.4. Демонстрация динамических входных параметров в браузере

Вас может удивить то, что все счетчики инкрементируются одновременно при нажатии любой кнопки. Это хорошая иллюстрация разделения области видимости, но нам такое поведение точно не подходит.

Обновим листинг 6.5, чтобы исправить эту проблему. Удалите константу `counter` и отредактируйте функцию данных: теперь она должна возвращать собственный счетчик вместо глобального. Этот счетчик доступен только локально и, следовательно, не может использоваться другими компонентами (листинг 6.6).

Листинг 6.6. Обновление счетчиков с помощью локальных объектов: chapter 06/dynamic-props-counter-correct.html

```
<!DOCTYPE html>
<html>
<head>
<script src="https://unpkg.com/vue"></script>
</head>
  <body>
  <div id="app">
  <my-component></my-component>
  <my-component></my-component>
  <my-component></my-component>
  </div>
  <script>
  const MyComponent= {
    template:'<div><button v-on:click="counter +=
➥   1">{{counter}}</button></div>',
    data() {
      return {
        counter: 0
      }
    }
  }

  new Vue({
    el: "#app",
    components: {'my-component': MyComponent},
    data() {
      return {
        message: ''
      }
    }

  });
  </script>
</html>
```

Функция данных возвращает счетчик

Откройте обновленный код в браузере. Нажмите несколько кнопок и понаблюдайте за поведением счетчика (рис. 6.5).

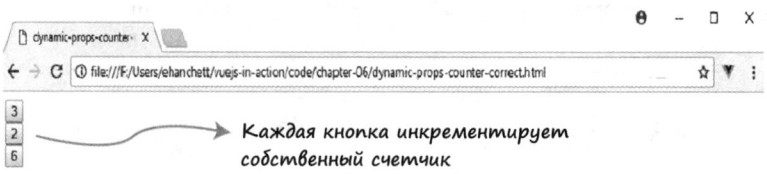


Рис. 6.5. Пример динамических входных параметров с локальным счетчиком

На этот раз все работает как полагается! Нажатие любой кнопки инкрементирует только ее собственный счетчик.

6.3.3. Проверка входных параметров

Vue.js позволяет проверять корректность входных параметров, передаваемых родителем. Особенно это может пригодиться при работе в команде, когда несколько человек используют один и тот же компонент.

Сначала проверим тип входных параметров. Для этого в Vue.js предусмотрены следующие встроенные конструкторы:

- `String`;
- `Number`;
- `Boolean`;
- `Function`;
- `Object`;
- `Array`;
- `Symbol`.

Проверка входных параметров проиллюстрирована в листинге 6.7. Сначала мы создаем компонент `my-component`, а затем передаем ему значения. Переданные данные выводятся в шаблоне.

Вместо создания массива, `prop: ['nameofProp']`, оформим входные параметры в виде объекта. Имя каждого свойства соответствует имени параметра. Внутри укажем тип и одно из полей, `required` или `default`. Поле `default` содержит значение по умолчанию, если входной параметр не инициализирован. Поле `required` требует, чтобы параметр был указан во время создания шаблона.

Последний примечательный момент листинга 6.7 — входной параметр `even`. Он содержит так называемый пользовательский валидатор. Здесь он возвращает `true`, если значение четное, в противном случае в консоли появится ошибка. Стоит отметить, что пользовательские валидаторы могут выполнять функции любого рода — главное, чтобы они возвращали `true` или `false`.

Листинг 6.7. Проверка входных параметров: chapter 06/props-example.html

```
<!DOCTYPE html>
<html>
<head>
<script src="https://unpkg.com/vue"></script>
</head>
  <body>
  <div id="app">
    <my-component :num="myNumber" :str="passedString"
                  :even="myNumber" :obj="passedObject"></my-component>
  </div>
  <script>
  const MyComponent={
    template:'<div>Number: {{num}}<br />String: {{str}} \
              <br />IsEven?: {{even}}<br/>Object:{{obj.message}}</div>',
    props:{
      num: {
        type: Number,
        required: true
      },
      str: {
        type: String,
        default: "Hello World"
      },
      obj: {
        type: Object,
        default: ()=> {
          return {message: 'Hello from object'}
        }
      },
      even: {
        validator: (value) => {
          return (value % 2 === 0)
        }
      }
    }
  }
  new Vue({
    el: '#app',
    components:{'my-component':MyComponent},
    data() {
      return {
        passedString: 'Hello From Parent!',
        myNumber: 43,
        passedObject: {message: 'Passed Object'}
      }
    }

  });
  </script>
  </body>
</html>
```

Передаем значения в my-component

Шаблон MyComponent используется для вывода всех свойств

Значение должно быть числом

Параметр принимает только строки и имеет значение по умолчанию

Параметр принимает только объекты и содержит сообщение по умолчанию

Пользовательский валидатор проверяет, является ли число четным

Не забывайте также, что само по себе двоеточие (`:`) — это короткое обозначение `v-bind`. Для `v-on` аналогичным сокращением будет символ `@`.

Откройте браузер и запустите код этого примера. Результат показан на рис. 6.6.

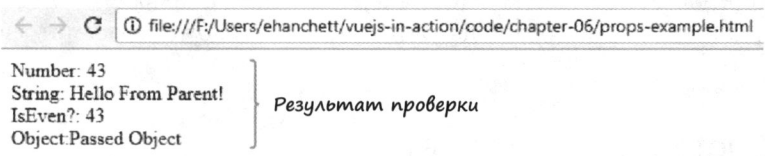


Рис. 6.6. Проверка типов входных параметров (число, строка и объект)

Все как ожидалось! Однако есть небольшая проблема. Если внимательно присмотреться к коду пользовательского валидатора, можно заметить, что он возвращает `false`, если число нечетное. Почему же мы не видим `false` в поле isEven?

На самом деле Vue.js выводит `false`, только не в шаблоне. По умолчанию проверка входных параметров не предотвращает их отображение, а всего лишь показывает предупреждение в консоли. Убедитесь в этом, открыв консоль Chrome, — она должна выглядеть, как на рис. 6.7.

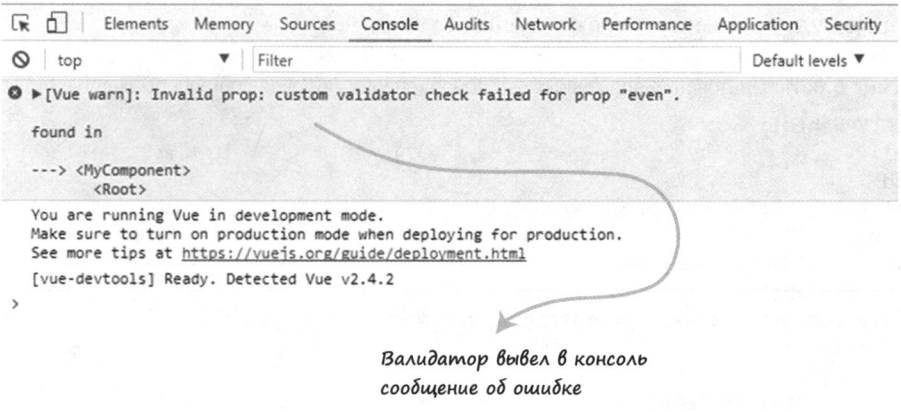


Рис. 6.7. Ошибка, свидетельствующая о непройденной проверке

Ошибка свидетельствует о том, что параметр `even` не прошел проверку. Зная об этом, мы можем поменять число с нечетного на четное. Учитывайте этот тип ошибок при проверке входных параметров.

6.4. Определение шаблона для компонента

Мы научились регистрировать компоненты локально и глобально, до этого момента все шаблоны в них определялись в виде строк. По мере роста и усложнения компонентов это способно превратиться в проблему. Не все среды разработки

поддерживают подсветку синтаксиса для строковых шаблонов, что может сделать работу неудобной. Кроме того, многострочные шаблоны должны содержать символы экранирования, тем самым засоряя определение компонента.

Vue.js предлагает несколько способов решения этой проблемы. Все они, в том числе использование литералов из стандарта ES2015, будут рассмотрены в дальнейшем.

6.4.1. Вложенные шаблоны

Один из самых простых способов вывода разметки — вложенные шаблоны. Чтобы работать с ними, необходимо указать информацию о шаблоне внутри компонента, который добавляется в шаблон родителя.

В листинге 6.8 показано объявление компонента `<my-component :myinfo="message" inline-template>`. Директива `inline-template` заставляет Vue отобразить содержимое компонента между открывающим и закрывающим тегами `my-component`.

Этот подход имеет один недостаток: определения шаблона и самого компонента находятся в разных местах. Для небольших приложений это не составляет проблемы, но в крупных проектах лучше подумать о применении однофайловых компонентов, которые будут рассмотрены в следующей главе.

Листинг 6.8. Использование вложенных шаблонов: chapter 06/inline-component-example.html

```
<!DOCTYPE html>
<html>
<head>
<script src="https://unpkg.com/vue"></script>
</head>
<body>
  <div id="app">
    <my-component :my-info="message" inline-template>
      <div>
          <p>
            inline-template - {{myInfo}}
          </p>
      </div>
    </my-component>
  </div>
  <script>
  const MyComponent = {
    props: ['myInfo']
  };

  new Vue({
      el: '#app',
      components: {'my-component': MyComponent},
      data() {
```

Вложенный шаблон отображает HTML

Переданный параметр

```
        return {
          message: 'Hello World'
        }
      }
  });
  </script>
</body>
</html>
```

6.4.2. Атрибут text/x-template

Еще один способ определения шаблона в компоненте — использование атрибута `text/x-template` в элементе `script`.

В листинге 6.9 с помощью `text/x-template` определяется шаблон для `my-component`. Главное — не забыть указать `type="text/x-template"` в теге `script`.

Этот метод имеет тот же недостаток, что и вложенные шаблоны: определения шаблона и компонента находятся в разных местах. Это подходит только для небольших приложений.

Листинг 6.9. Работа с атрибутом text/x-template: chapter 06/x-template-example.html

```
<!DOCTYPE html>
<html>
<head>
<script src="https://unpkg.com/vue"></script>
</head>
  <body>
  <div id="app">
    <my-component/>

  </div>
  <script type="text/x-template" id="my-component">
    <p>
      Hello from x-template, Hello World
    </p>
  </script>
  <script>
  const MyComponent = {
    template: '#my-component'
  }
  new Vue({
    el: '#app',
    components: {'my-component': MyComponent}
  });
  </script>
  </body>
</html>
```

Тег script типа x-template

6.4.3. Использование однофайловых компонентов

Ранее для определения шаблонов в примерах мы задействовали строки. Некоторые проблемы, присущие этому способу, можно решить за счет шаблонных литералов, поддержка которых появилась в стандарте ES2015. Чтобы превратить строку в шаблонный литерал, ее достаточно заключить в обратные кавычки (`). Такое значение способно быть многострочным, не требуя при этом экранирования. Оно также может содержать встроенные выражения, что упрощает написание шаблонов.

Несмотря на это, шаблонные литералы из состава ES2015 и обычные строки имеют несколько общих недостатков. Они загромождают определение компонента и не имеют подсветки синтаксиса в некоторых средах разработки. У нас есть еще один вариант для решения всех этих проблем — однофайловые компоненты.

Однофайловые компоненты позволяют определять сам компонент и его шаблон в едином файле с расширением `.vue`. Каждый из них имеет свою область видимости, поэтому не нужно беспокоиться о выборе уникальных имен для своих компонентов. То же самое относится и к CSS, что довольно удобно в больших приложениях. В довершение всего можно забыть о строковых шаблонах и необычных тегах `script`.

В листинге 6.10 HTML-код впервые выносится в отдельный тег `template`. Файлы `.vue` используют инструкцию `export` из состава ES2015 для возвращения информации о компоненте.

Листинг 6.10. Однофайловые компоненты: chapter 06/single-file-component.vue

```
<template>
  <div class="hello">
    {{msg}}
  </div>
</template>

<script>
export default {
  name: 'hello',
  data () {
    return {
      msg: 'Welcome to Your Vue.js App'
    }
  }
}
</script>

<!-- Атрибут scoped ограничивает область видимости CSS текущим компонентом -->
<style scoped>
</style>
```

Шаблон выводит данные компонента

Экспорт данных в стиле ES2015

Для применения однофайловых компонентов необходимо познакомиться с несколькими современными средствами сборки. Для компиляции кода в файле `.vue` используются такие утилиты, как Webpack или Browserify. Чтобы упростить этот процесс, Vue.js предоставляет собственную утилиту командной строки под названием Vue-CLI, которая содержит все необходимое для построения проекта. Эти

и другие инструменты будут рассмотрены в следующей главе. А пока вам достаточно знать, что работать с шаблонами можно несколькими способами и что однофайловые компоненты лучше всего подходят для крупных приложений.

6.5. Работа с пользовательскими событиями

У Vue.js есть собственный механизм событий. Пользовательские события, в отличие от обычных, о которых речь шла в главе 3, применяются для передачи данных из дочернего компонента в родительский. Для отслеживания и генерации событий задействуются выражения `$on(eventname)` и `$emit(eventName)` соответственно. `$on(eventname)` обычно используется при обмене событиями между компонентами, не связанными друг с другом напрямую. В случае с родителем и потомком события передаются с помощью директивы `v-on`. Благодаря этому механизму родительские компоненты могут отслеживать своих непосредственных потомков.

6.5.1. Отслеживание событий

Представьте, что вы разрабатываете счетчик. На экране будет видна кнопка, каждое нажатие которой инкрементирует счетчик на 1. При этом кнопка должна находиться в дочернем компоненте, а счетчик — в родительском экземпляре Vue.js. Дочерний компонент не должен изменять значение счетчика напрямую. Вместо этого он должен уведомлять родителя о том, что счетчик следует обновить. Посмотрим, как это сделать.

Начнем с создания компонента. При добавлении его в шаблон нужно задействовать директиву `v-on` и пользовательское событие. Как видно в листинге 6.11, мы добавили компонент и создали пользовательское событие `increment-me`, оно привязано к методу `incrementCounter`, который определен в родительском экземпляре Vue. Мы также добавим кнопку, которая привязана к событию `click` и вызывает `incrementCounter`. Она будет находиться в родительском шаблоне.

Еще одну кнопку нужно поместить внутрь определения компонента. Она тоже будет привязана к событию `click` с помощью директивы `v-on`. Ее нажатие должно приводить к вызову метода `childIncrementCounter`, определенного в дочернем компоненте.

Метод `childIncrementCounter` выполняет одно-единственное действие — генерирует пользовательское событие, которое мы создали ранее. В этом месте можно запутаться. Выражение `this.$emit('increment-me')` применяется для вызова родительского метода `incrementCounter`, связанного с событием, а тот уже инкрементирует счетчик. Такой гибкий подход позволяет изменять значения в родительском компоненте, не нарушая принцип однонаправленного связывания данных.

Листинг 6.11. Инкрементация родительского счетчика с помощью $emit: chapter 06/event-listen.html

```
<!DOCTYPE html>
<html>
<head>
```

```
<script src="https://unpkg.com/vue"></script>
</head>
  <body>
  <div id="app">
    {{counter}}<br/>
    <button v-on:click="incrementCounter">Increment Counter</button>
    <my-component v-on:increment-me="incrementCounter" ></my-component>
  </div>
  <script>
  const MyComponent = {
    template: `<div>
      <button v-on:click="childIncrementCounter"
      >Increment From Child</button>
    </div>`,
    methods: {
      childIncrementCounter() {
        this.$emit('increment-me');
      }
    }
  }
  new Vue({
    el: '#app',
    data() {
        return {
            counter: 0
        }
    },
    methods: {
      incrementCounter() {
        this.counter++;
      }
    },
    components: {'my-component': MyComponent}
  });
  </script>
  </body>
</html>
```

Кнопка, инкрементирующая счетчик из родителя

Компонент, привязывающий событие increment-me к incrementCounter

Кнопка компонента, которая вызывает метод childIncrementCounter

Генерирует событие increment-me

Метод, инкрементирующий счетчик на 1

Открыв этот код в браузере Chrome, вы увидите две кнопки. Одна из них находится в дочернем компоненте, но обе они инкрементируют счетчик, размещенный в родительском экземпляре (рис. 6.8).

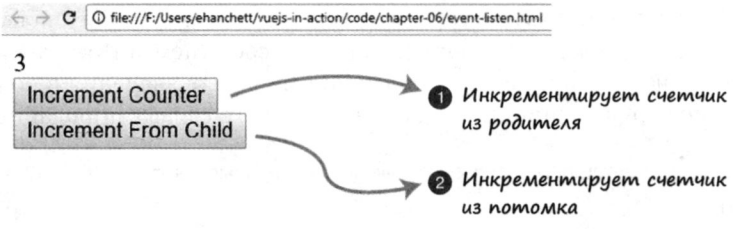


Рис. 6.8. Обе кнопки инкрементируют счетчик в родительском экземпляре

6.5.2. Изменение входных параметров потомка с помощью .sync

В большинстве случаев изменение входных параметров внутри потомка нежелательно. Лучше доверить это родителю. Это одно из фундаментальных правил однонаправленного связывания данных, о котором упоминалось ранее в этой главе. Тем не менее в Vue.js его можно нарушить.

Модификатор `.sync` позволяет изменять значения в родительском компоненте прямо внутри потомка. Эта возможность появилась в ветви Vue 1.x и была убрана с выходом Vue 2.0. Но основная команда разработчиков решила вернуть ее обратно, начиная с версии 2.3.0. Будьте осторожны при ее использовании.

Рассмотрим пример изменения значения с помощью `.sync`. Возьмем за основу листинг 6.11 и обновим `my-component` и `childIncrementCounter`. Для начала разберемся с модификатором `.sync`. Его можно прикрепить к любому входному параметру компонента. В листинге 6.12 это сделано в виде `<my-component :mycounter.sync="counter">`. Параметр `my-counter` привязан к `counter`.

На самом деле это лишь синтаксический сахар для выражения `<my-component :my-counter ="counter" @update:my-counter="val => bar = val"></my-component>`. Мы создали новое событие `update`, которое присваивает входной параметр `my-counter` заданной переменной.

Чтобы все это работало, нужно сгенерировать новое событие и передать значение, которое будет присвоено счетчику. Воспользуемся для этого методом `this.$emit`. Выражение `this.myCounter+1` служит первым аргументом, который передается в событие `update`.

Листинг 6.12. Изменение входных параметров из потомка с помощью .sync: chapter 06/event-listen-sync.html

```
<!DOCTYPE html>
<html>
<head>
<script src="https://unpkg.com/vue"></script>
</head>
  <body>
  <div id="app">
    {{counter}}<br/>
    <button v-on:click="incrementCounter">Increment Counter</button>
    <my-component :my-counter.sync="counter"></my-component>
  </div>
  <script>
  const MyComponent = {
    template: `<div>
      <button v-on:click="childIncrementCounter">Increment From Child</button>
    </div>`,
    methods: {
      childIncrementCounter() {
        this.$emit('update:myCounter', this.myCounter+1);
      }
```

Компонент с модификатором .sync

Генерирует событие update, за которым идет запятая

```
    },
    props:['my-counter']
  }
  new Vue({
    el: '#app',
    data() {
        return {
            counter: 0
        }
    },
    methods: {
      incrementCounter() {
        this.counter++;
      }
    },
    components: {'my-component': MyComponent}
  });
  </script>
  </body>
</html>
```

Открыв этот код в браузере, вы увидите две кнопки. Нажатие любой из них приводит к обновлению счетчика (рис. 6.9).

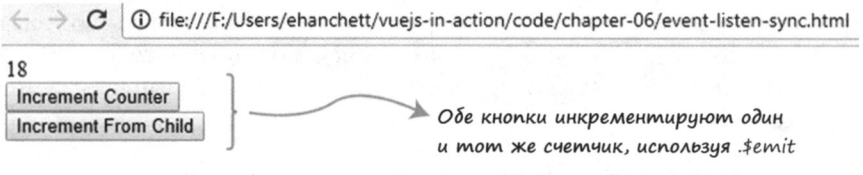


Рис. 6.9. Пример использования .sync для изменения счетчика

Упражнение

Используя знания, приобретенные в этой главе, ответьте на следующие вопросы.

- Как передать информацию из родительского компонента в дочерний?
- Какие средства применяются для передачи информации обратно из дочернего компонента в родительский?

Ответы ищите в приложении Б.

Резюме

- При локальной регистрации компоненты доступны в локальной области видимости. Для этого можно использовать опцию `components` в конструкторе экземпляра Vue.

- Чтобы зарегистрировать компоненты глобально, их определения нужно передать методу экземпляра `Vue.components`.
- Связывание данных между родительскими и дочерними компонентами однонаправленное.
- Входные свойства определяют, что именно можно передавать компоненту.
- Однофайловые компоненты позволяют объединить шаблон и код в едином файле.
- Вы можете передавать информацию родительскому компоненту с помощью `$emit`.

7 Продвинутые компоненты и маршрутизация

В этой главе

- Работа со слотами.
- Применение динамических компонентов.
- Реализация асинхронных компонентов.
- Использование однофайловых компонентов с помощью Vue-CLI.

Мы уже познакомились с компонентами и научились применять их для разбиения кода программы на части. Теперь рассмотрим их более продвинутые возможности, которые помогут придать приложениям динамичность и надежность.

Начнем со слотов. *Слоты* позволяют совмещать содержимое родительского компонента с дочерним шаблоном, упрощая тем самым их динамическое обновление. Затем перейдем к динамическим компонентам, способным переключаться между разными элементами на лету. Эта возможность облегчает изменение целых компонентов в ответ на действия пользователя. Представьте, к примеру, что вы создаете панель администрирования с разными графиками. Пользователь выберет тот, который ему нужен, благодаря динамическим компонентам.

Мы также рассмотрим асинхронные компоненты и попробуем разбить приложение на еще более мелкие части, которые будут загружаться по мере необходимости. Это отличное решение для крупных проектов, при запуске которых необходимо загружать ограниченный объем данных.

Заодно поговорим о Vue-CLI и однофайловых компонентах. Vue-CLI позволяет за несколько секунд подготовить и создать готовое приложение, заменяя собой сложные инструменты разработки. Все знания, приобретенные в этой главе, будут использованы при рефакторинге зоомагазина, и Vue-CLI нам в этом поможет!

В конце вы познакомитесь с механизмом маршрутизации и научитесь создавать параметризированные и дочерние маршруты. Итак, приступим!

7.1. Работа со слотами

Иногда, работая с компонентами, необходимо совмещать родительские и дочерние данные. Представьте, что у вас есть компонент с пользовательской формой, который вы хотите разместить на сайте книжного издательства. Форма состоит из двух полей — имени автора и названия книги. Рядом с полями ввода находятся метки с описанием, содержимое которых уже определено внутри функции данных корневого экземпляра Vue.js.

Вы, наверное, заметили, что данные нельзя добавлять между открывающим и закрывающим тегами. Любое содержимое внутри элемента автоматически убирается (рис. 7.1).

```
<my-component>
    Information here will not be displayed by default.
</my-component>
```

Рис. 7.1. Информация между тегами компонента игнорируется

Как вы вскоре убедитесь, самый простой способ вывести подобного рода информацию состоит в применении слотов. Vue.js поддерживает специальный элемент `slot`, который определяет, где должны выводиться данные между открывающим и закрывающим тегами компонента. В других JavaScript-фреймворках этот процесс называется *распределением содержимого*, в Angular используется термин *«включение»* (transclusion), по своей сути он похож на концепцию дочерних компонентов в React. Независимо от названия или фреймворка идея остается все той же: встраивание родительской информации в дочерние компоненты без непосредственной передачи.

На первый взгляд может показаться, что передача значений из корневого экземпляра Vue.js в дочерний компонент — наилучшее решение. Да, этот вариант тоже работает, но у него есть определенные ограничения. Чтобы их продемонстрировать, рассмотрим пример передачи свойств компоненту.

Создадим новый файл с локальным компонентом под названием `form-component` и простой формой внутри. Этот компонент будет принимать два обычных входных параметра: `title` и `author`. Укажем значения для этих параметров в корневом экземпляре Vue.js (листинг 7.1). Мы выполняли аналогичные действия в главе 6.

Далее приводятся несколько небольших примеров, каждый из которых размещен в отдельном файле. Можете их скопировать или набрать самостоятельно в текстовом редакторе.

Листинг 7.1. Традиционные родительский/дочерний компоненты с входными параметрами: chapter-07/parent-child.html

```
<!DOCTYPE html>
<html>
<head>
```

```
<script src="https://unpkg.com/vue"></script>
</head>
<body>
  <div id="app">
    <form-component
      :author="authorLabel"
      :title="titleLabel">
    </form-component>
  </div>
<script>
const FormComponent ={
  template: `
  <div>
    <form>
      <label for="title">{{title}}</label>
        <input id="title" type="text" /><br/>
      <label for="author">{{author}}</label>
        <input id="author" type="text" /><br/>
      <button>Submit</button>
    </form>
  </div>

  `,
  props: ['title', 'author']
}
new Vue({
  el: '#app',
  components: {'form-component': FormComponent},
  data() {
    return {
        titleLabel: 'The Title:',
        authorLabel: 'The Author:',
    }
  }

})
</script>
</body>
</html>
```

Передача метки author в компонент с формой

Передача метки title в компонент с формой

Вывод метки title, переданной в элемент

Вывод метки author, переданной в элемент

Как уже отмечалось, это вполне рабочий вариант, но по мере усложнения формы придется иметь дело с дополнительными атрибутами. Поля, предназначенные для ISBN, даты и года публикации, потребуют создания в компоненте новых входных параметров и атрибутов. Отслеживание большого количества свойств может оказаться утомительным и привести к ошибкам в вашем коде.

Перепишем этот пример с применением слотов. Для начала добавим текст, который можно выводить в верхней части формы. Вместо входных параметров для передачи значений будут использоваться элементы `slot`. Нам не нужно передавать все данные в компонент в виде свойств. Между его открывающим и закрывающим тегами можно напрямую отображать все что угодно. Итоговая форма показана на рис. 7.2.

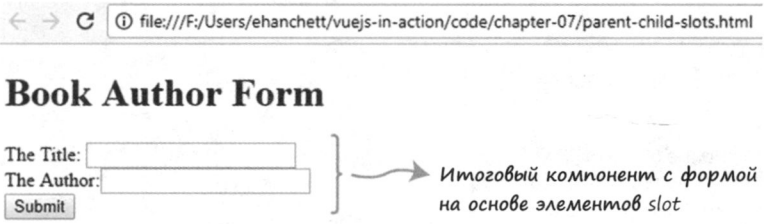


Рис. 7.2. Пример формы для добавления книги

Скопируйте листинг 7.1 в файл, измените функцию данных и добавьте новое свойство `header` (помните, вы всегда можете загрузить код для этой книги на моей странице GitHub по адресу github.com/ErikCH/VuejsInActionCode). Как видно на рис. 7.2, новое свойство `header` будет выводить Book Author Form. Затем найдите в родительском экземпляре Vue.js открывающий и закрывающий теги `form-component`. Вставьте между ними свойство `header`. Нужно также обновить сам компонент `form-component`. Вставьте элементы `<slot></slot>` сразу после первого тега `<form>`. Это то место, где Vue отобразит информацию, указанную между открывающим и закрывающим тегами компонента `form-component`. Итоговый код представлен в листинге 7.2.

Листинг 7.2. Добавление элемента slot: chapter-07/parent-child-slots-extract.html

```
...
<body>
  <div id="app">
    <form-component
      :author="authorLabel"
      :title="titleLabel">
      <h1>{{header}}</h1>
    </form-component>
  </div>
<script>
const FormComponent ={
  template: `
  <div>
    <form>
      <slot></slot>
      <label for="title">{{title}}</label>
        <input id="title" type="text" /><br/>
      <label for="author">{{author}}</label>
        <input id="author" type="text" /><br/>
      <button>Submit</button>
    </form>
  </div>

  `,
  props: ['title', 'author']
}
```

Переменная header внутри form-component

Элемент slot, вставленный родителем

```
new Vue({
  el: '#app',
  components: {'form-component': FormComponent},
  data() {
    return {
        titleLabel: 'The Title:',
        authorLabel: 'The Author:',
        header: 'Book Author Form'
    }
  }

})
</script>
</body>
</html>
```

Новое свойство header

7.2. Именованные слоты

Мы уже добавили один элемент `slot` в наш компонент. Но, как вы уже, наверное, догадываетесь, это не очень гибкое решение. Что, если у нас есть несколько свойств, которые мы хотели бы отображать в разных местах? Опять-таки передавать каждый из них через входной параметр было бы довольно утомительно. Подходят ли слоты для решения такой задачи?

В Vue для этого специально предусмотрены *именованные слоты*. В отличие от обычных их можно поместить в строго определенное место внутри компонента. К тому же каждый компонент способен содержать сразу несколько таких слотов. Воспользуемся этой возможностью в нашем приложении. Создадим внутри дочернего компонента `form-component` два именованных слота, `titleSlot` и `authorSlot`, и разместим их так, как показано в листинге 7.3.

Сначала укажем имена слотов в шаблоне компонента `form-components`. Для этого в HTML-код следует добавить элемент `named-slot`. Возьмите за основу листинг 7.2 и переместите элементы `label` из `form-component` в родительский шаблон, как показано далее. Не забудьте поменять имена свойств в метках: с `title` на `titleLabel` и с `author` на `authorLabel`.

Затем добавьте два новых слота. Каждый из них заменит по одной метке в шаблоне `form-component`. Это должно выглядеть как `<slot name="titleSlot"></slot>` и `<slot name="authorSlot"></slot>`.

Обновите внутри родительского шаблона метки, которые мы переместили ранее. Каждая метка должна содержать атрибут `slot`, например `<label for="title" slot="titleSlot">`. Vue.js проследит за тем, чтобы содержимое этой метки отобразилось в соответствующем именованном слоте. Поскольку мы больше не передаем входные параметры, можем удалить их из `form-component`. Далее показан готовый код.

Листинг 7.3. Использование именованных слотов: chapter-07/named-slots.html

```
<!DOCTYPE html>
<html>
<head>
<script src="https://unpkg.com/vue"></script>
</head>
<body>
  <div id="app">
    <form-component>
      <h1>{{header}}</h1>
      <label for="title" slot="titleSlot">{{titleLabel}}</label>
      <label for="author" slot="authorSlot">{{authorLabel}}</label>
    </form-component>
  </div>
<script>
const FormComponent ={
  template: `
  <div>
    <form>
      <slot></slot>
      <slot name="titleSlot"></slot>
        <input id="title" type="text" /><br/>
      <slot name="authorSlot"></slot>
        <input id="author" type="text" /><br/>
      <button>Submit</button>
    </form>
  </div>

  `
}
new Vue({
  el: '#app',
  components: {'form-component': FormComponent},
  data() {
    return {
        titleLabel: 'The Title:',
        authorLabel: 'The Author:',
        header: 'Book Author Form'
    }
  }

})
</script>
</body>
</html>
```

Вывод метки в слоте titleSlot

Вывод метки в слоте authorSlot

Вставляем именованный слот для titleSlot

Вставляем именованный слот для authorSlot

Именованные слоты значительно упрощают размещение родительских данных в разных участках дочернего компонента. Как видите, код стал более компактным и элегантным. К тому же мы больше не передаем входные параметры и не привязываем атрибуты при объявлении `form-component`. Вы по достоинству оцените эти преимущества, когда начнете разрабатывать более сложные приложения.

Слоты и область видимости на этапе компиляции

В листинге 7.3 свойство корневого экземпляра Vue.js выводится между открывающим и закрывающим тегами `form-component`. Имейте в виду, что дочерний компонент не имеет доступа к этим данным, так как они были добавлены родителем. При использовании слотов очень легко перепутать область видимости элемента. Помните: все, что находится в шаблоне родителя, компилируется в его контексте. То же самое относится и к потомку. Если об этом забыть, можно столкнуться с подобного рода проблемами.

7.3. Слоты с ограниченной областью видимости

Слоты с ограниченной областью видимости похожи на именованные, но играют роль шаблонов, пригодных для многократного использования и передачи данных. В Vue.js для них предусмотрены специальные элемент и атрибут под названием `slot-scope`.

Атрибут `slot-scope` представляет собой временную переменную, которая хранит свойства, переданные из компонента. Это позволяет передавать значения из дочернего компонента в родительский, а не наоборот, как мы делали ранее.

Чтобы это проиллюстрировать, предложим следующую аналогию. Представьте, что у вас есть веб-страница со списком книг. У каждой книги есть автор и название. Мы хотим создать компонент, который вписывается в интерфейс страницы, но при этом позволяет оформить с помощью стилей каждую книгу в списке. Для этого нам нужно передавать книгу из дочернего компонента в родительский. Итоговый результат должен выглядеть как на рис. 7.3.

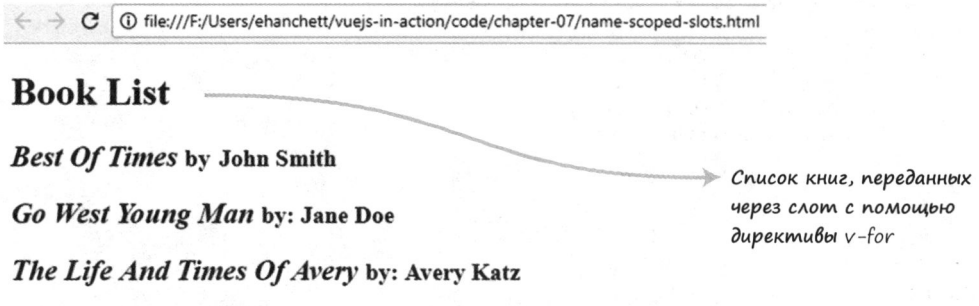


Рис. 7.3. Список книг и их авторов

Этот пример немного надуманный, но он демонстрирует гибкость слотов с ограниченной областью видимости и то, насколько легко с их помощью передавать данные в дочерние компоненты и обратно. В основе этого приложения лежит новый компонент `book-component`, который выводит заголовок и каждую отдельную книгу. В обоих случаях используются именованные слоты. Как видно в листинге 7.4, нужно

добавить директиву `v-for`, которая проходит по списку книг и привязывает к каждой из них подходящее значение.

Массив книг создается в корневом экземпляре Vue.js. По своей сути это список объектов с именем автора и названием книги. Мы можем передать этот массив в `book-component` с помощью директивы `v-bind` в виде `:books`.

Добавим элемент `<template>` внутрь родительского шаблона. Он должен содержать атрибут `slot-scope`, иначе наш пример не будет работать. Этот атрибут привязывает значение, переданное из дочернего компонента. В данном случае `{{props.text}}` приравнивается к дочернему значению `{{book}}`.

Теперь выражение `{{props.text}}` внутри элемента `template` будет выводить то же, что и `{{books}}`. Иными словами, `{{props.text.title}}` равно `{{book.title}}`. Мы визуально выделим заголовок и имя автора каждой книги.

Откройте текстовый редактор и скопируйте код из листинга 7.4. Как видите, мы взяли массив `books` и передали его компоненту `book-component`. Затем вывели каждую книгу в отдельном слоте, переданном в шаблон, который отображается родительским компонентом.

Листинг 7.4. Слоты с ограниченной областью видимости: chapter-07/name-scoped-slots.html

```
<!DOCTYPE html>
<html>
<head>
<script src="https://unpkg.com/vue"></script>
</head>
<body>
  <div id="app">
    <book-component :books="books">
      <h1 slot="header">{{header}}</h1>
      <template slot="book" slot-scope="props">
        <h2>
          <i>{{props.text.title}}</i>
          <small>by: {{props.text.author}}</small>
        </h2>
      </template>
    </book-component>
  </div>
<script>
const BookComponent ={
  template: `
  <div>
      <slot name="header"></slot>
      <slot name="book"
        v-for="book in books"
        :text="book">
      </slot>
  </div>
  `,
  props: ['books']
}
new Vue({
  el: '#app',
```

book-component с переданными книгами

Текст заголовка на основе именованного слота header

Элемент template с входными параметрами, переданными через slot-scope

Отображаем текст для каждой отдельной книги

Именованный слот, который привязывает директиву v-for

Ссылка на текущую книгу в массиве books

```
  components: {'book-component': BookComponent},
  data() {
    return {
      header: 'Book List',
      books: [{author: 'John Smith', title: 'Best Of Times' },
              {author: 'Jane Doe', title: 'Go West Young Man' },
              {author: 'Avery Katz', title: 'The Life And Times Of Avery' }
             ]
    }
  }

})
</script>
</body>
</html>
```

Массив книг

Слоты с ограниченной областью видимости обеспечивают чрезвычайную гибкость, хотя поначалу могут показаться немного запутанными. Мы можем взять значение дочернего компонента и вывести его в контексте родителя с применением специальных стилей. Этот инструмент пригодится во время работы с более сложными компонентами, которые выводят списки элементов.

7.4. Создание приложения с динамическими компонентами

Еще одной сильной стороной Vue.js являются *динамические компоненты*. Они позволяют на лету переключаться между разными элементами, используя зарезервированный элемент `component` и атрибут `is`.

Внутри функции данных можно создать свойство, которое будет определять, какой компонент следует вывести. В шаблон нужно добавить элемент `component` с атрибутом `is`, который будет ссылаться на новое свойство. Рассмотрим конкретный пример.

Представьте, что вы разрабатываете приложение с тремя разными компонентами. Для переключения между ними будет предусмотрена кнопка. Один компонент выводит список книг, другой отображает форму для добавления книги, еще один отвечает за вывод заголовка. Итоговый результат показан на рис. 7.4.

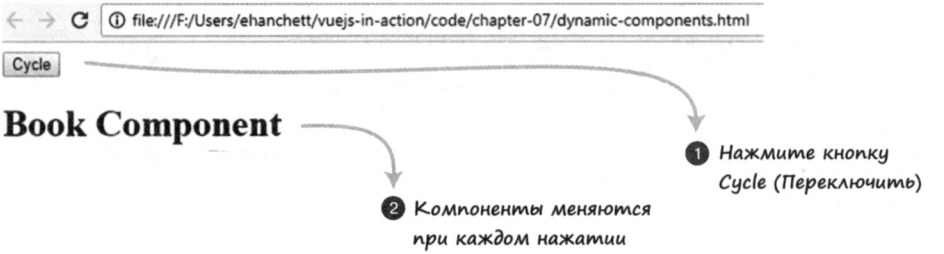


Рис. 7.4. Динамический вывод компонентов по нажатию кнопки

Нажатие кнопки Cycle (Переключить) приводит к отображению следующего компонента. Эта кнопка вызывает простое выражение на языке JavaScript, которое перебирает компоненты для вывода книги, формы и заголовка.

Откройте текстовый редактор и создайте новое приложение на основе Vue.js. Вместо одного компонента создайте три. В каждом шаблоне будет выводиться текст, чтобы пользователь понимал, какой компонент сейчас активен. Код этого примера представлен в листинге 7.5.

Функция данных возвращает одно свойство под названием `currentView`. Изначально оно будет ссылаться на `BookComponent`. Создадим метод `cycle`, который будет обновлять свойство `currentView` при каждом щелчке, перебирая все компоненты.

Завершающим шагом станет добавление кнопки в корневой экземпляр Vue.js. Эта кнопка будет привязана к событию `click`: `<button @click="cycle" >Cycle</ button>`. Создайте под ней тег `<h1>` с новым элементом `component`, который ссылается на `currentView` с помощью своего атрибута `is`. Это позволит динамически переключаться между компонентами. Свойство `currentView` обновляется при каждом щелчке. Чтобы запустить этот пример, создайте файл `dynamic-components.html` и скопируйте в него следующий код.

Листинг 7.5. Динамические компоненты: chapter-07/dynamic-components.html

```
<!DOCTYPE html>
<html>
<head>
<script src="https://unpkg.com/vue"></script>
</head>
<body>
  <div class="app">
    <button @click="cycle">Cycle</button>
    <h1>
      <component :is="currentView"></component>
    </h1>
  </div>
<script>
const BookComponent ={
  template: `
  <div>
    Book Component
  </div>
  `
}

const FormComponent = {
  template: `
  <div>
    Form Component
  </div>
  `
}

const HeaderComponent = {
```

При каждом нажатии срабатывает метод cycle, который меняет currentView

Элемент component, который динамически привязывается к currentView

```
  template: `
  <div>
    Header Component
  </div>
  `
}

new Vue({
  el: '.app',
  components: {'book-component': BookComponent,
               'form-component': FormComponent,
               'header-component': HeaderComponent},
  data() {
    return {
      currentView: BookComponent
    }

  },
  methods: {
        cycle() {
          if(this.currentView === HeaderComponent)
            this.currentView = BookComponent
          else
            this.currentView = this.currentView === BookComponent ?
            FormComponent : HeaderComponent;
        }
    }

})
</script>
</body>
</html>
```

Список создаваемых компонентов

Это свойство изначально ссылается на BookComponent

Метод, который переключается между тремя компонентами

Вы узнали, как нажатием одной кнопки можно переключаться между тремя разными компонентами. Этот пример реально реализовать с помощью нескольких директив `v-if` и `v-else`, но наш способ работает лучше и куда более нагляден.

7.5. Создание асинхронных компонентов

При разработке больших проектов иногда приходится разбивать приложение на отдельные компоненты и загружать их по мере необходимости. Vue упрощает этот процесс с помощью асинхронных компонентов. Каждый компонент можно определить в виде функции, которая загружает компонент асинхронно. Более того, Vue.js кэширует загруженный код для последующей перерисовки.

Рассмотрим простое приложение с симуляцией серверной загрузки на основе предыдущего примера. Представьте, что список книг загружается с сервера и на получение ответа уходит 1 с. Решим эту задачу с помощью Vue.js. Итоговый результат показан на рис. 7.5.

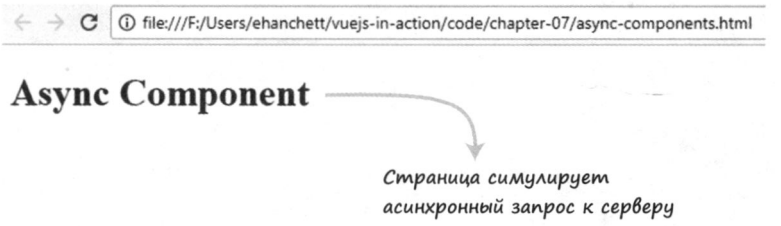


Рис. 7.5. Асинхронный компонент отрисовывается после односекундной задержки

Метод будет иметь две функции обратного вызова, `resolve` и `reject`. Мы также должны адаптировать наш компонент под новые условия. Создайте новое приложение и компонент `book-component` (листинг 7.6).

После загрузки этот простой компонент выводит текст на экран. Воспользуемся функцией `setTimeout`, чтобы этот процесс занимал 1 с. Таким образом мы симулируем сетевую задержку.

Самое важное при создании асинхронного компонента — это определить его в виде метода с двумя функциями обратного вызова, `resolve` и `reject`. Вы можете инициировать разные действия в зависимости от того, был этот метод выполнен или отклонен.

Для запуска этого примера создайте файл под названием `async-componets.html` и скопируйте в него код из следующего листинга. На экране должен появиться простой асинхронный компонент. Мы симулируем сервер, ответ которого занимает 1 с. Можно также предусмотреть функцию `reject` на случай неудачного вызова.

Листинг 7.6. Асинхронные компоненты: chapter-07/async-components.html

```
<!DOCTYPE html>
<html>
<head>
<script src="https://unpkg.com/vue"></script>
</head>
<body>
  <div id="app">
    <book-component></book-component>
  </div>
<script>

const BookComponent = function(resolve, reject) {
  setTimeout(function() {
    resolve({
      template: `
      <div>
        <h1>
          Async Component
        </h1>
```

Вывод book-component в шаблоне

Асинхронный компонент, который либо загружается, либо отказывает

setTimeout симулирует задержку ответа сервера длительностью 1000 мс

```
      </div>
      `
    });

  },1000);

}

new Vue({
  el: '#app',
  components: {'book-component': BookComponent }

})
</script>
</body>
</html>
```

Продвинутые асинхронные компоненты

Начиная с Vue 2.3.0, вы можете создавать более продвинутые асинхронные компоненты, которые позволяют выводить индикатор загрузки. А также определить отдельный компонент на случай ошибки и указать время ожидания. Подробнее об этом говорится в официальном руководстве по адресу https://ru.vuejs.org/v2/guide/components-dynamic-async.html#Управление-состоянием-загрузки.

7.6. Рефакторинг зоомагазина с помощью Vue-CLI

До этого момента все наши приложения помещались в один файл. Но по мере развития зоомагазина этот подход становится все менее удобным. Чтобы сделать код более элегантным, приложение можно разбить на отдельные компоненты.

Как мы видели в главе 6, для этого существует множество разных способов. Один из самых гибких вариантов — применение однофайловых компонентов. Среди его преимуществ выделим CSS-таблицы с локальной областью видимости, подсветку синтаксиса, простой механизм многократного использования и модули ES6.

Мы можем ограничивать область видимости CSS-таблицы текущим компонентом. Это помогает определять стили для каждого отдельного компонента. Улучшается подсветка синтаксиса, так как больше не нужно беспокоиться о том, распознает ли IDE шаблон компонента, присвоенный переменной или свойству. Модули ES6 позволяют легко подключать любимые сторонние библиотеки. Каждая из этих возможностей делает написание Vue-приложений чуть легче.

Чтобы полноценно использовать все преимущества однофайловых компонентов, мы должны задействовать средства сборки вроде Webpack, которые помогут упаковать все наши модули и зависимости. Мы также можем применять такие инструменты, как Babel, которые компилируют JavaScript-код и гарантируют его

совместимость со всеми браузерами. Все это можно делать самостоятельно, но Vue.js значительно упрощает процесс за счет утилиты Vue-CLI.

Vue-CLI — это средство для проектов на основе Vue.js. Оно содержит все, что необходимо для начала разработки Vue-приложений. Vue-CLI поддерживает ряд шаблонов, которые позволяют выбрать подходящие инструменты (больше о Vue-CLI можно узнать на официальной странице ru.vuejs.org/v2/guide/installation.html). Самые распространенные из них перечислены далее.

- *webpack*. Полнофункциональный проект на основе Webpack с поддержкой загрузчика Vue, горячей перезагрузки, анализа кода и извлечения CSS.
- *webpack-simple*. Простой проект для быстрого создания прототипов на основе Webpack и загрузчика Vue.
- *browserify*. Полнофункциональный проект на основе Browserify и Vuetify с поддержкой горячей перезагрузки, анализа кода и модульного тестирования.
- *browserify-simple*. Простой проект для быстрого создания прототипов на основе Browserify и Vuetify.
- *pwa*. Проект для PWA (Progressive Web Applications — прогрессивные веб-приложения) на основе Webpack.
- *Simple*. Простейший Vue-проект в виде единого HTML-файла.

Чтобы начать создавать приложения, нужно сначала установить Node и Git, а затем Vue-CLI (если вы все еще этого не сделали, инструкции можно найти в приложении А).

ПРИМЕЧАНИЕ

На момент написания книги версия Vue-CLI 3.0 все еще находилась на этапе бета-тестирования. В этой главе применяется последняя стабильная версия Vue-CLI, 2.9.2. Если вы используете Vue-CLI 3.0, некоторые параметры будут отличаться. Для создания приложения предусмотрена команда vue create <имя проекта> вместо vue init. Будет предложен новый набор вопросов, вы должны либо согласиться с вариантами по умолчанию, либо выбрать из списка нужные возможности, такие как TypeScript, Router, Vuex и препроцессор для CSS. Если вы воспроизводите представленные здесь примеры, убедитесь в том, что параметры соответствуют листингу 7.7. После этого можно переходить сразу к разделу 7.6.2. Больше подробностей о Vue-CLI 3.0 найдете в официальном файле README по адресу github.com/vuejs/vue-cli/blob/dev/docs/README.md.

7.6.1. Создание нового приложения с помощью Vue-CLI

Создадим приложение для нашего зоомагазина, используя Vue-CLI. Откройте терминал и введите `vue init webpack petstore`. Эта команда создаст проект на основе шаблона Webpack.

На момент написания книги последней версией Vue-CLI была 2.9.2. Если у вас более новая версия, не волнуйтесь: вопросы должны быть похожими и не требующими

отдельных объяснений. Если столкнетесь с какими-либо проблемами, следуйте инструкциям по установке, приведенным в официальном руководстве, и используйте Vue-CLI по адресу ru.vuejs.org/v2/guide/installation.html.

После запуска этой команды будет задано несколько вопросов. Сначала вас спросят название, описание и имя автора проекта. В качестве названия введите `petstore`, а остальные два параметра выберите по своему усмотрению. Дальше вы должны указать, следует ли задействовать компилятор в сочетании со средой выполнения. Я рекомендую выбрать вариант с компилятором, так как он упрощает создание шаблонов. В ином случае шаблоны можно будет размещать исключительно в файлах `.vue`.

Следующий вопрос касается установки vue-router. Введите `yes`. После этого вам предложат установить ESLint. Это библиотека, которая анализирует код при каждом сохранении. Она не играет важной роли в нашем проекте, поэтому выберите `no`. Последние два вопроса связаны с тестированием. В последующих главах я покажу, как создавать тестовые сценарии с помощью библиотеки vue-test-utils, а пока ответьте положительно в обоих случаях. Придерживаясь листинга 7.7, вы сможете создать новое приложение для зоомагазина с применением Vue-CLI.

Листинг 7.7. Работа в терминале

```
$ vue init webpack petstore
? Project name petstore
? Project description Petstore application for book
? Author Erik Hanchett <erikhanchettblog@gmail.com>
? Vue build standalone
? Install vue-router? Yes
? Use ESLint to lint your code? No
? Setup unit tests with Karma + Mocha? Yes
? Setup e2e tests with Nightwatch? Yes

   vue-cli · Generated "petstore".

   To get started:

     cd petstore
     npm install
     npm run dev

   Documentation can be found at https://vuejs-templates.github.io/webpack
```

Команда init создает новое приложение

Перечисление вопросов и ответов

После создания приложения и загрузки шаблона необходимо установить все зависимости. Перейдите в каталог `petstore` и введите `npm install` или `yarn`:

```
$ cd petstore
$ npm install
```

Таким образом, будут установлены все зависимости приложения. Это займет несколько минут. После завершения вы сможете запустить следующую команду:

```
$ npm run dev
```

Откройте браузер и перейдите по адресу localhost:8080. Вы должны увидеть страницу приветствия Vue.js, показанную на рис. 7.6 (пока сервер продолжает работать, любые обновления будут применяться на лету). Если сервер не запустился, убедитесь в том, что на порте 8080, используемом по умолчанию, не работает другое приложение.

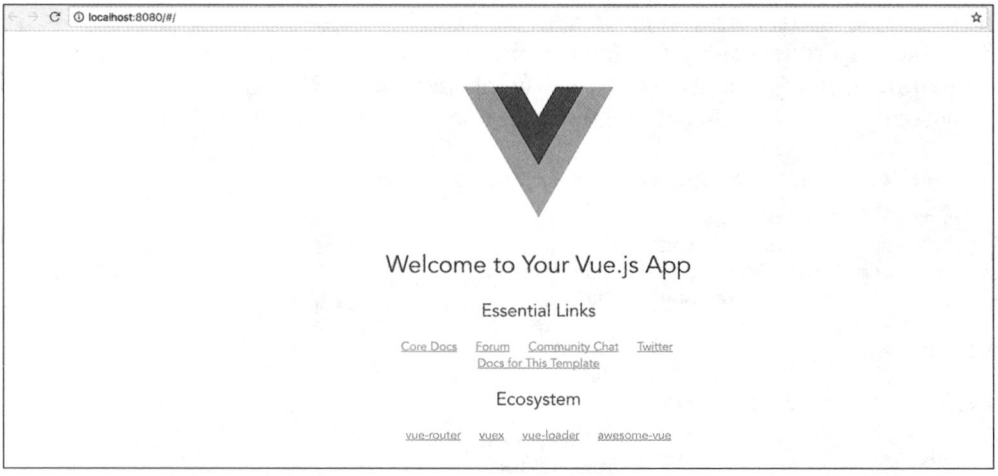


Рис. 7.6. Стандартная страница приветствия для Vue-CLI

Теперь мы можем перейти к зоомагазину.

7.6.2. Настройка маршрутов

Vue-CLI поставляется вместе с продвинутой библиотекой для маршрутизации *vue-router*. Это официальный маршрутизатор для Vue.js, который поддерживает всевозможные функции, включая параметризированные маршруты, параметры в запросах и подстановочные маршруты. Кроме того, он умеет работать в режиме HTML5 history с автоматическим переключением в режим хеша для Internet Explorer 9. С его помощью можно создать нужный маршрут, не беспокоясь о совместимости с браузерами.

Создадим для зоомагазина два маршрута, `Main` и `Form`. Первый будет отображать список товаров из файла `products.json`, а второй — вести на страницу оформления покупки.

Откройте внутри созданного приложения файл `src/router/index.js` и взгляните на массив `routes`. Там уже есть маршрут по умолчанию, `Hello`, можете смело его удалить. Приведите массив `routes` к виду, показанному в листинге 7.8. Каждый хранящийся в нем объект имеет минимум два поля: `path` и `component`. Поле `path` — это URL-адрес, по которому нужно пройти, чтобы посетить данный маршрут, `component` — имя компонента, используемого для маршрута.

При желании укажите также свойство `name`, которое обозначает имя маршрута. Его мы задействуем чуть позже. Есть еще одно необязательное свойство, `props`. Оно определяет, ожидает ли компонент получения каких-либо входных параметров.

Обновив массив, не забудьте импортировать компоненты `Form` и `Main` в файле маршрутизатора. Любой компонент, на который мы ссылаемся, должен быть импортирован. По умолчанию Vue-CLI применяет импорт в стиле ES6. Если компоненты не импортировать, в консоли появится ошибка.

Наконец, стоит отметить, что vue-router по умолчанию работает в режиме хеша. При открытии в браузере адреса form Vue сформирует URL вида #/form вместо /form. Чтобы отключить эту функцию, добавьте в маршрутизатор `mode: 'history'`.

Листинг 7.8. Добавление маршрутов: chapter-07/petstore/src/router/index.js

```
import Vue from 'vue'
import Router from 'vue-router'
import Form from '@/components/Form'
import Main from '@/components/Main'

Vue.use(Router)

export default new Router({
  mode: 'history',
  routes: [
    {
      path: '/',
      name: 'iMain',
      component: Main,
      props: true,
    },
    {
      path: '/form',
      name: 'Form',
      component: Form,
      props: true
    }

  ]
})
```

Импортируем компоненты Form и Main

Режим history с маршрутами без знака решетки

Маршрут iMain по адресу /

Маршрут Form по адресу /form

Новый проект всегда лучше начинать с подготовки маршрутов. Это позволяет определиться с тем, как именно будет устроено ваше приложение.

7.6.3. Добавление CSS, Bootstrap и Axios

Наш зоомагазин использует ряд библиотек, которые необходимо добавить в проект. Это делается разными способами.

Мы можем применять библиотеку для Vue.js. Вместе с Vue развивается и экосистема этого фреймворка. Постоянно появляются новые проекты, ориентированные

на Vue.js. Например, чтобы добавить поддержку Bootstrap, воспользуйтесь библиотекой BootstrapVue. Для Material Design существует популярная библиотека Vuetify. Мы еще вернемся к этим проектам в будущем.

Библиотеки можно подключать также в файле `index.html`. Это удобно в случаях, когда для Vue.js нет специальной обертки.

Итак, откройте файл `index.html` в корневой папке зоомагазина. Чтобы не сильно отклоняться от оригинального примера из главы 5, добавим ссылки на CDN с Bootstrap 3 и Axios (листинг 7.9). В результате получим доступ к этим библиотекам на любом участке приложения.

Листинг 7.9. Добавление Axios и Bootstrap: chapter-07/petstore/index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <script src="https://cdnjs.cloudflare.com/ajax/libs/
➥ axios/0.16.2/axios.js"></script>

    <title>Vue.js Pet Depot</title>
    <link rel="stylesheet"
      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/
➥ css/bootstrap.min.css" crossorigin="anonymous">
  </head>
  <body>
    <div id="app"></div>
    <!-- Встроенные файлы будут добавлены автоматически -->
  </body>
</html>
```

CDN с библиотекой Axios

CDN с библиотекой Bootstrap 3

CSS-таблицу тоже можно добавить несколькими путями. Как вы увидите позже, один из способов — определение CSS в локальной области видимости компонента. Это полезно в случаях, когда к компоненту применяется отдельный набор стилей.

Можем также указать CSS глобально для всего сайта. Чтобы не усложнять, добавим в проект CSS-файл, доступный для всех его компонентов (позже мы рассмотрим CSS с ограниченной областью видимости).

Откройте файл `src/main.js`. Именно в нем находится корневой экземпляр Vue.js. Здесь можно импортировать CSS-таблицу для приложения. Поскольку мы используем Webpack, нужно применить ключевое слово `require` и относительный путь к ресурсу.

ДОПОЛНИТЕЛЬНО

Подробнее о Webpack и ресурсах говорится в документации на странице vuejs-templates.github.io/webpack/static.html.

Поместите файл `app.css` в папку `src/assets` (листинг 7.10). Копию `app.css` можно найти в архиве с кодом для этой книги в приложении А.

Листинг 7.10. Добавление CSS: chapter-07/petstore/src/main.js

```
import Vue from 'vue'
import App from './App'
import router from './router'
require('./assets/app.css')

Vue.config.productionTip = false

/* eslint-disable no-new */
new Vue({
  el: '#app',
  router,
  template: '<App/>',
  components: { App }
})
```

Подключение app.css к приложению

Теперь, когда CSS-файл добавлен в приложение, его может использовать любой компонент.

7.6.4. Подготовка компонентов

Как уже отмечалось, компоненты облегчают разбиение программы на мелкие части, пригодные для многократного использования. Разделим зоомагазин на несколько модулей, чтобы проще было вести разработку. У нас будет три компонента: `Main`, `Form` и `Header`. Компонент `Header` выводит название сайта и панель навигации, `Main` отображает список товаров, а `Form` отвечает за вывод страницы оформления покупки.

Прежде чем начинать, удалите файл `HelloWorld.vue` из папки `src/components`. Он нам не понадобится. Создайте в той же папке файл `Header.vue`. В него мы поместим информацию о заголовке.

Большинство файлов `.vue` имеют сходную структуру. Вверху обычно описывается шаблон, обрамленный открывающим и закрывающим тегами `template`. Как вы уже знаете, вслед за `template` должен идти корневой элемент. Обычно я выбираю для этого элемент `div`, но и `header` тоже подойдет. Не забывайте, что шаблон может содержать только один корневой элемент.

За шаблоном идет элемент `script`, в котором определяется экземпляр Vue. В конце находится элемент `style`, в который можно поместить CSS-код и сделать его локальным для текущего компонента (это будет продемонстрировано в листинге 7.12).

В качестве шаблона скопируйте код из листинга 7.11. Он похож на заголовок, который мы создали в главе 5, но вы уже, наверное, заметили новый элемент `router-link`, который входит в библиотеку vue-router. Этот элемент создает в нашем Vue-приложении локальные ссылки на маршруты. Элемент `router-link` содержит атрибут `to`, который можно привязать к одному из наших именованных маршрутов. Привяжем его к маршруту `Main`.

Листинг 7.11. Шаблон Header: chapter-07/petstore/src/components/Header.vue

```
<template>
<header>
  <div class="navbar navbar-default">
    <div class="navbar-header">
      <h1><router-link :to="{name: 'iMain'}">
            {{ sitename }}
          </router-link>
      </h1>
    </div>
    <div class="nav navbar-nav navbar-right cart">
        <button type="button"
          class="btn btn-default btn-lg"
          v-on:click="showCheckout">
        <span class="glyphicon glyphicon-shopping-cart">
         {{ cartItemCount}}</span> Checkout
      </button>
    </div>
  </div>
</header>
</template>
```

Это ссылка на маршрут iMain

Теперь нужно создать логику нашего компонента. Возьмем код из предыдущей версии зоомагазина и скопируем его в файл `Header.vue`. Здесь следует внести несколько изменений. В последнем примере в главе 5 мы использовали директиву `v-if`, чтобы определить, нужно ли отображать страницу оформления покупки. Для этого применялся метод, который переключал свойство `showProduct` по нажатию кнопки с корзиной.

Вместо этого будем просто переходить по маршруту `Form`. В листинге 7.12 видно, что это делается с помощью метода `this.$router.push`. Как и в случае с `router-link`, следует указать имя нужного маршрута. Таким образом, кнопка корзины будет направлять нас к маршруту `Form`.

Поскольку мы поменяли переменную `sitename` на ссылку с помощью `router-link`, теперь она выглядит немного не так, как прежде. Нужно обновить CSS-стили нового элемента `a` в разделе `style`. Поскольку мы указали ключевое слово `scoped`, Vue.js сделает так, чтобы эта CSS-таблица была доступна только для текущего компонента.

Вы также могли заметить, что в листинге 7.12 больше нет инициализатора экземпляра Vue, который задействовался в предыдущих главах. В проекте Vue-CLI он не нужен. Вместо него используется более простой синтаксис экспорта модулей в стиле ES6, `export default { }`. Поместите туда весь свой код для Vue.js.

В CSS-таблице нужно отключить подчеркивание текста и сделать его черным. Объедините листинги 7.11 и 7.12 в один файл.

Этот компонент можно считать завершенным. Вероятно, вы заметили, что заголовок принимает входной параметр под названием `cartItemCount`. Это значение будет передавать компонент `Main`, который мы создадим чуть позже. `cartItemCount` будет подсчитывать элементы, добавленные в корзину.

Листинг 7.12. Добавление логики и CSS: chapter-07/petstore/src/components/Header.vue

```
<script>
export default {
  name: 'my-header',
  data () {
    return {
    sitename: "Vue.js Pet Depot",
    }
  },
  props: ['cartItemCount'],
  methods: {
    showCheckout() {
      this.$router.push({name: 'Form'});
    }

  }
}
</script>

<!-- Атрибут "scoped" ограничивает область видимости CSS текущим компонентом -->
<style scoped>
a {
  text-decoration: none;
  color: black;
}
</style>
```

Направляет приложение по маршруту Form

CSS с ограниченной областью видимости

7.6.5. Создание компонента Form

Компонент `Form` — это то место, где находится страница оформления покупки. Здесь все останется примерно в том же виде, что и в главе 5. Основным отличием будет использование нового компонента `my-header` в верхней части шаблона. Кроме того, нужно передать `cartItemCount` в заголовок.

Создайте в папке `src/components` компонент под названием `Form.vue`. Как видно в листинге 7.13, HTML-код шаблона почти ничем не отличается от созданного в главе 5. Единственное отличие — добавление нового компонента вверху заголовка. Я не стану приводить здесь весь код, примеры для главы 7 лучше загрузить отдельно (инструкции приводятся в приложении А).

Листинг 7.13. Создание компонента Form: chapter-07/petstore/src/components/Form.vue

```
<template>
  <div>
  <my-header :cartItemCount="cartItemCount"></my-header>
    <div class="row">
      <div class="col-md-10 col-md-offset-1">
      ...

      </div><!--end of col-md-10 col-md-offset-1-->
    </div><!--end of row-->
  </div>
</template>
```

В компонент my-header передается значение cartItemCount

Скриптовый код этого компонента напоминает пример, приведенный в главе 5. Одно из отличий состоит в том, что теперь он принимает входной параметр под названием `cartItemCount`. Мы также должны объявить компонент `Header`, чтобы его можно было использовать в шаблоне. Это продемонстрировано в листинге 7.14.

Листинг 7.14. Добавление элемента script: chapter-07/petstore/src/components/Form.vue

```
<script>
import MyHeader from './Header.vue';
export default {
  name: 'Form',
  props: ['cartItemCount'],
  data () {
    return {
      states: {
...
      },
      order: {
           ...
      }

    }
  },
  components: { MyHeader },
  methods: {
    submitForm() {
      alert('Submitted');
    }
  }
}
</script>
```

Импорт компонента Header

Прием входного параметра cartItemCount

Чтобы закончить с этим компонентом, объедините листинги 7.13 и 7.14. Пока этого достаточно, хотя позже в этой главе мы добавим бизнес-логику к полям ввода.

7.6.6. Добавление компонента Main

Компонент `Main` будет отображать товары в зоомагазине, позволяя посетителю пополнять корзину и просматривать рейтинги. Вся логика для этой страницы уже написана, остается лишь поместить ее в единый файл `.vue`.

Как и в случае с компонентом `Form`, вставим элемент `my-header` в верхнюю часть файла и передадим ему значение `cartItemCount`. Создайте файл `Main.vue` в папке `src/components`. Добавьте в него следующий код (листинг 7.15).

Листинг 7.15. Создание шаблона главной страницы: chapter-07/petstore/src/components/Main.vue

```
<template>
  <div>
  <my-header :cartItemCount="cartItemCount"></my-header>
  <main>
```

Добавляем в код компонент my-header

```
    <div v-for="product in sortedProducts">
      <div class="row">
        <div class="col-md-5 col-md-offset-0">
          <figure>
            <img class="product" v-bind:src="product.image" >
          </figure>
        </div>
        <div class="col-md-6 col-md-offset-0 description">
          <h1 v-text="product.title"></h1>
          <p v-html="product.description"></p>
          <p class="price">
            {{product.price | formatPrice}}
          </p>
          <button class=" btn btn-primary btn-lg"
                  v-on:click="addToCart(product)"
                  v-if="canAddToCart(product)">Add to cart</button>
          <button disabled="true" class=" btn btn-primary btn-lg"
                                  v-else >Add to cart</button>
          <span class="inventory-message"
            v-if="product.availableInventory - cartCount(product.id) === 0">All Out!
          </span>
          <span class="inventory-message"
            v-else-if="product.availableInventory - cartCount(product.id) < 5">
            Only {{product.availableInventory - cartCount(product.id)}} left!
          </span>
          <span class="inventory-message"
            v-else>Buy Now!
          </span>
          <div class="rating">
            <span v-bind:class="{'rating-active' :checkRating(n, product)}"
                  v-for="n in 5" >☆
            </span>
          </div>
        </div><!-- Конец col-md-6 -->
      </div><!-- Конец row -->
      <hr />
    </div><!-- Конец v-for -->
  </main>
  </div>
</template>
```

Вслед за шаблоном нужно добавить код на основе Vue.js. Создайте выражение импорта в верхней части файла, как показано в листинге 7.16. Следует также объявить компонент сразу за функцией данных, используя выражение `components: { MyHeader }`.

Прежде чем добавлять остальной код, скопируйте папку `images` и файл `products.json` в каталог `petstore/static`. Их можно найти в архиве с кодом для главы 7 по адресу github.com/ErikCH/VuejsInActionCode.

При использовании Vue-CLI файлы хранятся в одной из двух папок: `assets` или `static`. Ресурсы в папке `assets` обрабатываются загрузчиками Webpack url-loader

и file-loader, во время сборки они подставляются/копируются/переименовываются, поэтому их можно считать исходным кодом. Чтобы ссылаться на файлы в папке `assets`, необходимо задействовать относительные пути. Например, логотип может иметь путь `./assets/logo.png`.

Статические файлы не проходят никакой обработки со стороны Webpack, а просто копируются как есть в итоговый каталог. Для обращения к ним используются абсолютные пути. Поскольку мы загружаем все ресурсы с помощью файла `products.json`, будет проще скопировать их в папку `static` и ссылаться на них оттуда.

Обновим файл `Main.vue` в папке `src/components` (в листинге 7.16 пропущены фильтры и методы). Скопируйте данные, методы, фильтры и хуки жизненного цикла экземпляра Vue в файл `Main.vue` снизу от шаблона.

Листинг 7.16. Код для Main.vue: chapter-07/petstore/src/components/Main.vue

```
<script>
import MyHeader from './Header.vue';
export default {
  name: 'imain',
  data() {
    return {
      products: {},
      cart: []
    };
  },
  components: { MyHeader },
  methods: {
    ...

},
  filters: {
    ...

  },
  created: function() {
    axios.get('/static/products.json').then(response => {
      this.products = response.data.products;
      console.log(this.products);
    });
  }
};
</script>
```

Импортируем в проект MyHeader

JSON-файл с товарами находится в папке static и доступен по абсолютному пути

После копирования кода удалите из файла `App.vue` стили и элемент `img` со ссылкой на `logo.png`. Если хотите, удалите и сам файл `logo.png` из папки `assets`. Не забудьте перезагрузить сервер Vue-CLI с помощью команды `npm run dev`, если еще этого не сделали. В результате должно запуститься приложение зоомагазина, в котором можно перейти на страницу оформления покупки, нажав кнопку корзины (рис. 7.7). Если возникли какие-то проблемы, еще раз проверьте консоль. Например,

если забыть импортировать библиотеку Axios в файле `index.html`, как делали в листинге 7.9, получится ошибка.

Рис. 7.7. Зоомагазин, открытый с помощью Vue-CLI

7.7. Маршрутизация

Теперь, когда приложение использует Vue-CLI, рассмотрим маршрутизацию подробнее. Ранее в этой главе мы уже создали несколько маршрутов. В этом разделе добавим еще парочку.

В одностраничных приложениях на основе Vue.js маршрутизация помогает переходить между страницами. В нашем зоомагазине есть маршрут `Form`. Чтобы его загрузить, нужно открыть приложение и перейти по адресу `/form`. Для загрузки маршрута не нужно запрашивать никаких данных с сервера, как это делается в традиционных веб-приложениях. При изменении URL-адреса маршрутизатор Vue перехватывает запрос и отображает подходящий маршрут. Благодаря этой важной концепции вся маршрутизация происходит на клиентской стороне и не требует обращения к серверу.

В данном разделе вы научитесь создавать дочерние маршруты, передавать информацию между страницами с помощью параметров и настраивать перенаправление и подстановочные маршруты. Материал, который я не сумел вместить в эту книгу, можно найти в официальной документации для маршрутизатора Vue на сайте router.vuejs.org/ru/.

7.7.1. Добавление параметризированного маршрута product

Наше приложение содержит всего два маршрута, `Main` и `Form`. Создадим еще один, для отдельных товаров. Представьте, что владелец зоомагазина попросил добавить страницу с описанием товара. Это можно реализовать с помощью динамического

сопоставления маршрутов на основе *параметров*. Параметр — это динамическое значение, передаваемое в рамках URL-адреса. На рис. 7.8 показан пример открытия нового динамического маршрута. Обратите внимание на URL-адрес product/1001 вверху.

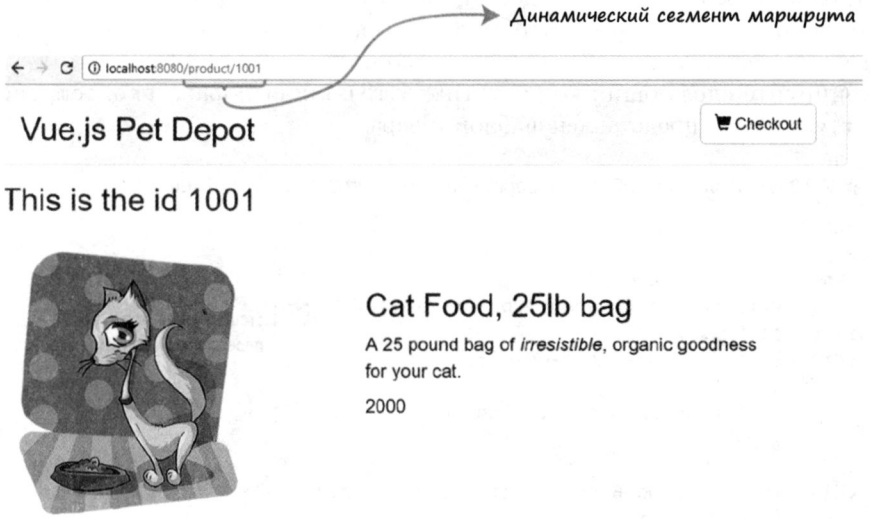


Рис. 7.8. Динамический маршрут для товара 1001

Для обозначения динамических маршрутов в файле маршрутизации используется двоеточие (:). Таким образом, любой маршрут, начинающийся с /product, сопоставляется с компонентом `Product`. Иными словами, компонент `Product` отвечает за обработку маршрутов /product/1001, /product/1002 и им подобных. При этом он принимает значения 1001 и 1002 в качестве параметра `id`.

Загляните в папку `src/router` внутри зоомагазина. В файле `index.js` находятся уже существующие маршруты. Скопируйте фрагмент кода из листинга 7.17 и вставьте его в массив `routes` в файле `src/router/index.js`. Не забудьте импортировать компонент `Product` вверху (его мы создадим чуть позже).

Листинг 7.17. Редактирование файла маршрутизации: chapter-07/route-product.js

```
import Product from '@/components/Product'
...
    },
    {
        path: '/product/:id',
        name: 'Id',
        component: Product,
        props: true
    }
...
```

Динамический сегмент маршрута под названием id

Итак, динамический сегмент содержит параметр `id`. Назовем маршрут `Id`, чтобы позже к нему было легче обращаться с помощью компонента `router-link`. А теперь, как мы и обещали, создадим компонент `Product`.

`Product` будет выводить отдельный товар, заданный в виде параметра маршрута. Наша задача — отобразить информацию о товаре внутри данного компонента.

В шаблоне `Product` у нас будет доступ к значению `$route.params.id`. Это параметр `id`, переданный в URL-адресе. Чтобы убедиться в корректности передачи идентификатора, выведем его в верхней части компонента.

Скопируйте следующий код (листинг 7.18) в новый файл `src/components/Product.vue`. Далее представлен шаблон товара.

Листинг 7.18. Добавление шаблона товара: chapter-07/product-template.vue

```
<template>
  <div>
    <my-header></my-header>
    <h1> This is the id {{ $route.params.id}}</h1>
    <div class="row">
      <div class="col-md-5 col-md-offset-0">
        <figure>
          <img class="product" v-bind:src="product.image" >
        </figure>
      </div>
      <div class="col-md-6 col-md-offset-0 description">
        <h1>{{product.title}}</h1>
        <p v-html="product.description"></p>
        <p class="price">
          {{product.price }}
        </p>
      </div>
    </div>
  </div>
  </template>
...
```

`$route.params.id` выводит переданный идентификатор

В шаблоне компонента нет ничего примечательного. Более интересна логика, содержащаяся в элементе `script`. Чтобы вывести в шаблоне подходящий товар, нам нужно найти товар с переданным идентификатором.

К счастью, это можно сделать с помощью простого выражения на JavaScript. Мы опять воспользуемся библиотекой Axios, чтобы загрузить статичный файл `products.json`. На этот раз применим функцию фильтрации, которая возвращает только те товары, идентификаторы которых совпадают с `this.$route.params.id`. В нашем случае этот фильтр должен вернуть лишь одно значение, так как все идентификаторы уникальные. Если по какой-то причине это не так, перепроверьте файл `products.json` и убедитесь в том, что ни один идентификатор в нем не повторяется.

Напоследок нужно вставить символ `'/'` перед значением `this.product.image`, загруженным из плоского файла (листинг 7.19). Дело в том, что мы применяем

динамическое сопоставление маршрутов, поэтому относительные пути к файлам могут создать проблемы.

Скопируйте код из следующего листинга и добавьте его в конец файла `src/components/Product.vue`. Убедитесь, что в данном файле представлен весь код из листингов 7.18 и 7.19.

Листинг 7.19. Код компонента Product: chapter-07/product-script.js

```
...
  <script>
  import MyHeader from './Header.vue'
  export default {
    components: { MyHeader },
    data() {
    return {
      product: ''
    }
  },
  created: function() {
    axios.get('/static/products.json')
    .then((response) =>{
      this.product = response.data.products.filter(
        data => data.id == this.$route.params.id)[0]
      this.product.image = '/' + this.product.image;
    });
  }
}
</script>
```

Импортируем компонент Header

Загружаем статический файл с помощью библиотеки Axios

Фильтруем полученные данные

Добавляем в this.product только те данные, которые соответствуют параметрам маршрута

Вставляем '/' перед product.image, чтобы сделать путь абсолютным

Закончив с компонентом `Product`, мы можем сохранить файл и открыть браузер. Пока у нас нет прямого доступа к маршруту, но можно ввести URL http://localhost:8080/product/1001 в адресной строке. Отобразится первый товар.

Отладка

Если маршрут не загрузился, откройте консоль и поищите ошибки. Убедитесь в том, что вы сохранили файл с маршрутами, в противном случае маршрут не загрузится. Вы также могли забыть добавить `'/'` перед `product.image`.

7.7.2. Настройка router-link с помощью тегов

Чтобы как-то использовать наши маршруты, следует добавить в приложение соответствующие ссылки. Если этого не сделать, посетителям придется запоминать каждый URL-адрес. Vue-router облегчает эту задачу. Один из самых простых способов, который мы уже встречали в этой главе, заключается в применении компонента `router-link`. Маршрут, к которому нужно перейти, указывается в атрибуте `:to`.

Это может быть как путь, так и объект с именем маршрута. Например, ссылка `<router-link :to="{ name: 'Id' }">Product</router-link>` загрузит именованный маршрут `Id`, который соответствует компоненту `Product`.

У компонента `router-link` есть еще несколько интересных особенностей. Он поддерживает дополнительные функции за счет целого ряда входных параметров. В этом разделе мы рассмотрим параметры `active-class` и `tag`.

Представьте, что у зоомагазина появилось еще одно требование. После перехода к маршруту `Form` кнопка корзины должна становиться нажатой, а когда посетитель покидает этот маршрут — возвращаться в исходное состояние. Для этого мы можем добавлять к кнопке класс `active`, когда маршрут активен, и удалять его, когда пользователь находится на другой странице. Кроме того, нужно сделать так, чтобы посетитель мог перейти к описанию товара, щелкнув на его названии.

На рис. 7.9 показан итоговый результат после нажатия кнопки корзины. Обратите внимание на внешний вид кнопки, когда пользователь находится на странице оформления покупки.

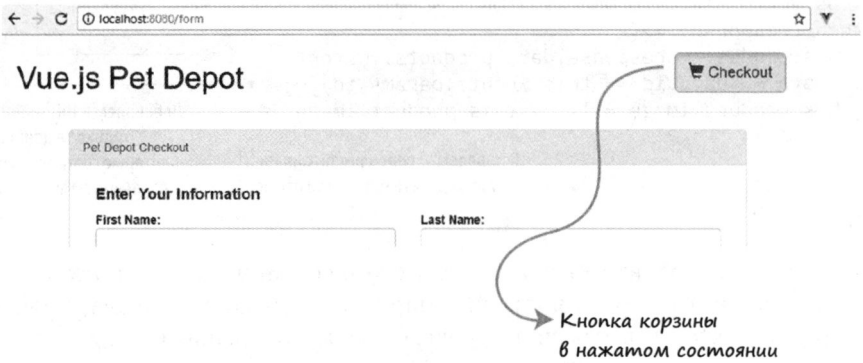


Рис. 7.9. Обновленная кнопка корзины с новым стилем

Добавим ссылку на новую страницу товара. Откройте файл `src/Main.vue` и найдите тег `h1` с `{{product.title}}` внутри. Удалите его и создайте в этом месте элемент `router-link` с входным параметром `tag`. С помощью этого параметра `router-link` преобразуется в заданный тег. В нашем случае ссылка будет отображаться в виде элемента `h1`.

Входной параметр `to` обозначает целевой маршрут ссылки, который можно описать в виде объекта. Для передачи параметров используется синтаксис `params: {id: product.id}`, таким образом, значение `product.id` будет передано в динамический сегмент под именем `id`. Например, если `product.id` равно `1005`, маршрут будет выглядеть как `/product/1005`.

Откройте файл `src/Main.vue` и приведите его в соответствие с листингом 7.20. Обратите внимание, что атрибут `:to` содержит два свойства, `name` и `params`, разделенные запятой.

Листинг 7.20. Обновление router-link в компоненте Main: chapter-07/route-link-example.html

```
...
<div class="col-md-6 col-md-offset-0 description">
  <router-link
      tag="h1"
        :to="{ name : 'Id', params: {id: product.id}}">
        {{product.title}}
  </router-link>
  <p v-html="product.description"></p>
...
```

Открывающий тег router-link

Выводит router-link в виде тега h1

Целевой маршрут Id с передачей параметра

На этом тексте можно будет щелкнуть кнопкой мыши

Сохраните файл и откройте браузер, предварительно запустив команду `npm run dev`. Теперь вы можете перейти к компоненту `Product`, щелкнув на заголовке любого товара. Параметр `id` передается маршруту `Product` и используется для вывода описания.

Параметры запроса

Параметры запроса — это еще один механизм передачи информации между маршрутами. Они добавляются в конец URL-адреса. Мы можем воспользоваться ими вместо динамического сегмента маршрута для отправки идентификатора товара. Чтобы добавить параметр запроса в маршрутизаторе Vue, достаточно указать соответствующее свойство в объекте описания:

```
<router-link tag="h1":to=" {name : 'Id', query:
➥ {Id: '123'}}">{{product.title}}</router-link>
```

Если вы хотите использовать несколько параметров запроса, их следует разделить запятыми, например: `{Id: '123', info: 'erik'}`. В адресной строке это будет выглядеть как `?id=123&info=erik`. К объекту `query` можно обращаться внутри шаблона: `$route.query.info`. Подробности ищите в официальной документации на странице router.vuejs.org/ru/api/#router-link.

7.7.3. Оформление router-link с помощью стилей

Согласно новым требованиям мы должны как-то активизировать кнопку корзины после перехода к маршруту `Form`. Это легко сделать с помощью атрибута `active-class`. Когда маршрут активен, `router-link` автоматически добавляет `active-class` к списку классов тега. Чтобы кнопка выглядела нажатой, воспользуемся классом `active` из состава Bootstrap.

Откройте файл `src/components/Header.vue` и обновите элемент кнопки для `{{cartItemCount}}`. Удалите существующую кнопку и вставьте вместо нее `router-link`, как показано в листинге 7.21. Заодно удалите метод `showCheckout`, так как он нам больше не понадобится.

Листинг 7.21. Обновление ссылки в заголовке при активизации маршрута: chapter-07/header-link.html

```
...
<div class="nav navbar-nav navbar-right cart">
  <router-link
    active-class="active"
    tag="button"
    class="btn btn-default btn-lg"
    :to="{name: 'Form'}">
  <span
    class="glyphicon glyphicon-shopping-cart">
    {{ cartItemCount}}
  </span> Checkout
...
```

Элемент router-link перебрасывает нас на страницу оформления покупки

Атрибут active-class добавляет класс active

Преобразует route-link в элемент h1

Классы для кнопки из состава Bootstrap

Переход к маршруту Form

Сохраните изменения, внесенные в заголовок, и вернитесь к приложению. В консоли браузера можно наблюдать, как при каждом нажатии кнопки корзины к ней добавляется класс `active`. Если опять перейти к маршруту `Main`, класс `active` удаляется, то есть он присутствует только тогда, когда маршрут `Form` активен, и исчезает, когда посетитель переходит на другую страницу.

Vue, начиная с версии 2.5.0, при переключении между маршрутами добавляет новый CSS-класс. Он называется `router-link-exact-active`. При его активизации можно выполнять различные действия, например поменять цвет ссылки на синий.

Добавьте в нижнюю часть файла `src/components/Header.vue` CSS-селектор, скопировав содержимое листинга 7.22. Этот класс будет применяться к элементу `router-link` только при активизации подходящего маршрута.

Листинг 7.22. Класс router-link-exact-active: chapter-07/route-link.css

```
...
.router-link-exact-active {
  color: blue;
}
...
```

Делает элемент синим при активизации маршрута

Сохраните файл и попробуйте походить между страницами приложения в браузере. Вы можете заметить, что текст кнопки корзины в заголовке меняет цвет на синий, когда маршрут становится активным. По эстетическим причинам во всех оставшихся примерах этот CSS-селектор станет устанавливать черный цвет. В случае необходимости он всегда будет под рукой.

7.7.4. Добавление дочернего маршрута edit

Новый маршрут `Id` отображает отдельные товары. Но что, если нам нужно их не только выводить, но и редактировать? Добавим внутрь `Product` еще один маршрут, который активизируется при нажатии кнопки Edit Product (Отредактировать товар).

ЗАМЕЧАНИЕ

Чтобы не усложнять приложение, мы не станем реализовывать функцию редактирования. На текущем этапе у нас нет возможности сохранять изменения в статический файл. Вместо этого сосредоточимся на добавлении дочернего маршрута, а возможность изменять товар прибережем на будущее.

Дочерние маршруты являются вложенными. Они идеально подходят для случаев, когда нужно редактировать или удалять информацию в рамках одного маршрута. Как вы вскоре увидите, для обращения к ним достаточно создать дочерний компонент `router-view` внутри родительского.

После внесения всех изменений новый маршрут `Edit` должен выглядеть так, как на рис. 7.10. Обратите внимание на URL-адрес product/1001/edit.

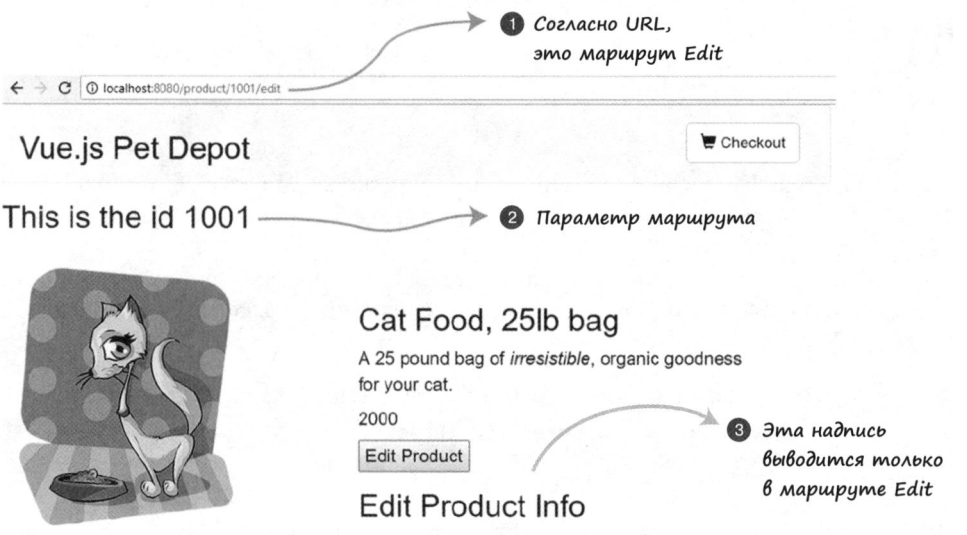


Рис. 7.10. Маршрут Edit — дочерний по отношению к Product

Для начала добавим новый компонент. Создайте в папке `src/components` файл `EditProduct.vue` и скопируйте в него листинг 7.23.

Листинг 7.23. Добавление компонента EditProduct: chapter-07/edit-comp.vue

```
<template>
  <div>
    <h1> Edit User Info</h1>
  </div>
</template>
<script>
    export default {
    // future
    }
</script>
```

Будущая реализация редактирования товара

Создайте элемент `router-view` внутри компонента `Product`. Этот элемент встроен в маршрутизатор Vue и служит конечной точкой для нового маршрута. При его активизации на месте `router-view` будет отображен компонент `EditProduct`.

Скопируйте фрагмент кода, представленный далее (листинг 7.24), в файл `src/components/Product.vue`, чтобы сформировать внизу новую кнопку и компонент `router-view`. Кнопка запускает новый метод `edit`, который активизирует маршрут `Edit`.

Листинг 7.24. Добавление кнопки редактирования товара: chapter-07/edit-button.vue

```
...
  </p>
  <button @click="edit">Edit Product</button>
  <router-view></router-view>
</div>
...
methods: {
    edit() {
      this.$router.push({name: 'Edit'})
    }
},
```

Кнопка, которая запускает метод edit

Элемент router-view служит входной точкой маршрута

$router.push активизирует маршрут Edit

Теперь у нас все готово для обновления списка маршрутов. Создайте внутри маршрута `Id` новый массив под названием `children`. В него мы добавим маршрут `Edit` и компонент `EditProduct`.

Код из листинга 7.25 вставьте в файл `src/router/index.js`. Обновите маршрут `Id`, добавив в него массив `children`. Не забудьте импортировать компонент `EditProduct` вверху.

Листинг 7.25. Добавление дочернего маршрута в маршрутизатор: chapter-07/child-route.js

```
import EditProduct from '@/components/EditProduct'
import Product from '@/components/Product'
...
{
    path: '/product/:id',
    name: 'Id',
    component: Product,
    props: true,
    children: [
      {
        path: 'edit',
        name: 'Edit',
        component: EditProduct,
        props: true
      }
    ]
},
...
```

Новый дочерний маршрут, который активизируется только внутри маршрута Id

Сохраните файл `index.js` и проверьте новый маршрут в своем браузере. При нажатии кнопки Edit Product (Редактировать товар) должно появиться сообщение `'Edit Product Info'`. Если этот маршрут не загружается, поищите ошибки в консоли и еще раз проверьте файл `index.js`.

7.7.5. Перенаправление и подстановочные маршруты

Последними двумя возможностями маршрутизатора Vue, на которых мы остановимся, будут *перенаправление* и *подстановочные маршруты*. Представьте, что к зоомагазину было предъявлено заключительное требование: если посетитель случайно введет неправильный URL-адрес, он должен быть направлен обратно на главную страницу. Это делается с помощью подстановочных маршрутов и перенаправления.

Мы можем использовать подстановочный символ (`*`), чтобы охватить адреса, не предусмотренные существующими маршрутами. Такой маршрут должен находиться в конце списка.

Параметр `redirect` определяет, куда следует перенаправить браузер. Откройте файл `src/routes/index.js` и добавьте внизу следующий фрагмент кода (листинг 7.26).

Листинг 7.26. Добавление подстановочного маршрута: chapter-07/wildcard-route.js

```
...
{
  path: '*',
  redirect:"/"
}
...
```

Захватывает любые адреса

Перенаправляет на страницу "/"

Сохраните этот файл и попробуйте перейти по адресу `/anything` или `/testthis`. В обоих случаях вы должны вернуться на главную страницу `"/"`.

Хуки навигации

Как понятно из названия, *хуки навигации* контролируют переходы между маршрутами, позволяя их перенаправлять или отменять. Это особенно полезно в ситуациях, когда перед открытием страницы нужно проверить пользователя. Например, можно добавить хук `beforeEnter` непосредственно в объект конфигурации маршрута. Это выглядит следующим образом:

```
beforeEnter (to, from, next) => { next() }
```

Хук `beforeEnter(to, from, next)` также можно добавить в любой компонент. Он загружается перед самим маршрутом. Выражения `next()` и `next(false)` позволяют продолжить и остановить загрузку маршрута соответственно. Больше подробностей — в официальной документации по адресу router.vuejs.org/ru/guide/advanced/navigation-guards.html.

Далее представлена полная версия файла `src/routes/index.js` (листинг 7.27).

Листинг 7.27. Файл маршрутизации целиком: chapter-07/petstore/src/router/index.js

```
import Vue from 'vue'
import Router from 'vue-router'
import Form from '@/components/Form'
import Main from '@/components/Main'
import Product from '@/components/Product'
```

```
import EditProduct from '@/components/EditProduct'
Vue.use(Router)

export default new Router({
  mode: 'history',
  routes: [
    {
      path: '/',
      name: 'iMain',
      component: Main,
      props: true,
    },
    {
        path: '/product/:id',
        name: 'Id',
        component: Product,
        props: true,
        children: [
          {
            path: 'edit',
            name: 'Edit',
            component: EditProduct,
            props: true
          }
        ]
    },
    {
      path: '/form',
      name: 'Form',
      component: Form,
      props: true
    },
    {
      path: '*',
      redirect:"/"
    }
  ]
})
```

Динамический сегмент маршрута с id

Дочерний маршрут внутри Id

Перехватывает все оставшиеся пути и перенаправляет на "/"

Ленивая (отложенная) загрузка

Vue-CLI использует Webpack для упаковки кода на JavaScript. Иногда итоговый пакет становится довольно большим, что может сказаться на времени загрузки в условиях медленного сетевого соединения или при работе с крупными приложениями. Для уменьшения размера пакетов можно задействовать асинхронные компоненты в сочетании с ленивой (отложенной) загрузкой. Эта концепция выходит за рамки данной книги, но я настоятельно рекомендую ознакомиться с ней в официальной документации, которую вы найдете по адресу router.vuejs.org/ru/guide/advanced/lazy-loading.html.

Маршруты — неотъемлемая часть большинства Vue-приложений. Они нужны в любом проекте сложнее Hello World. Они должны быть логичными, поэтому не жалейте времени на их проектирование. Используйте дочерние маршруты для реализации таких операций, как добавление или редактирование. Не бойтесь применять параметры для передачи информации между маршрутами. Если вам никак не удается решить проблему, связанную с маршрутизацией, вы всегда можете свериться с официальным руководством на сайте router.vuejs.org/ru.

Упражнение

Используя знания, приобретенные в этой главе, назовите два способа навигации между разными маршрутами.

Ответ ищите в приложении Б.

Резюме

- Использование слотов позволяет передавать информацию в компоненты динамическим образом.
- Можно применять динамические компоненты для переключения между разными элементами.
- Добавление асинхронных компонентов повышает скорость работы приложения.
- Можно задействовать Vue-CLI для рефакторинга приложения.
- Можно передавать значения между компонентами с помощью входных параметров.
- Дочерние маршруты можно использовать для редактирования информации в рамках родительских маршрутов.

8 Переходы и анимация

В этой главе

- Классы переходов.
- Применение анимации.
- Добавление JavaScript-хуков.
- Обновление зоомагазина.

В главе 7 мы рассмотрели продвинутые возможности Vue.js и научились разбивать приложение на мелкие части с помощью однофайловых компонентов. В этой главе поговорим о переходах и анимации, которые в Vue.js основаны на специальных классах. После этого создадим анимацию с помощью JavaScript-хуков и обсудим переходы между компонентами. В конце используем все эти возможности в нашем зоомагазине.

8.1. Что такое переходы

Для создания переходов в Vue.js нужно сначала познакомиться с `<transition>`. Это специальный компонент, который позволяет анимировать один или несколько элементов или выполнить переход между ними. Внутри `<transition>` помещается условное выражение, динамический элемент или корневой узел компонента.

Компонент `<transition>` либо вставляется в документ, либо удаляется из него в зависимости от определенных условий. Например, элемент, который он окружает, добавляется или удаляется с помощью директивы `v-if`. При этом `<transition>` поддерживает анимацию и CSS-переходы, создавая их путем удаления или добавления CSS-классов в нужный момент. Вы также можете указать специальные хуки на JavaScript, если нужно создать более сложный сценарий. Если компонент `<transition>` не обнаружит никакой анимации или переходов, операция добавления или удаления узла документа будет выполнена мгновенно. Рассмотрим пример.

Представьте, что вы разрабатываете сайт со списком наименований книг. Когда пользователь щелкает на заголовке, должно появляться описание книги, а при повторном щелчке оно должно исчезать. Как вы уже знаете из предыдущих глав, это можно сделать с помощью директивы `v-if`.

Но что, если описание должно появляться и исчезать постепенно? Для этого в Vue.js предусмотрены CSS-переходы и компонент `<transition>`.

Откройте свой редактор и создайте простое приложение, показанное в листинге 8.1. Сначала оно выведет вверху страницы название выдуманной книги в виде элемента `h2`. Мы поместим этот элемент внутрь элемента `div` с обработчиком события `@click`. Обработчик будет переключать значение переменной `show`: если она равна `true`, щелчок присвоит ей `false`, а если равна `false`, щелчок присвоит ей `true`.

Внутри элемента `body` следует создать компонент `transition` с атрибутом `name`. Присвойте этому атрибуту значение `fade`. Внутри `transition` имеется директива `v-if`, которая показывает и скрывает описание. Добавьте внизу конструктор Vue с функцией данных — здесь будут находиться все переменные для приложения.

Листинг 8.1. Создание перехода для описания: chapter-08/transition-book-1.html

```
<!DOCTYPE html>
<html>
<head>
  <script src="https://unpkg.com/vue"></script>
</head>
<body>
  <div id="app" >
    <div @click="show = !show">
      <h2>{{title}}</h2>
    </div>
    <transition name="fade">
      <div v-if="show">
        <h1>{{description}}</h1>
      </div>
    </transition>
  </div>
  <script>
  new Vue({
    el: '#app',
    data() {
      return {
        title: 'War and Peace',
        description: 'Lorem ipsum dolor sit amet, consectetur adipiscing elim',
        show: false
      }
    }

  });
  </script>
</body>
</html>
```

Элемент div, переключающий переменную show между true и false

Новый элемент transition с именем fade

Директива v-if, отвечающая за отображение и скрытие описания

Конструктор Vue.js

Функция данных с переменными

Откройте браузер и загрузите приложение. Ваша страница должна выглядеть как на рис. 8.1. Если щелкнуть на заголовке, снизу появится описание. В этом примере нет никакого перехода — описание выводится и скрывается мгновенно. Чтобы сделать процесс более плавным, нужно добавить классы переходов.

Возьмите за основу листинг 8.1 и добавьте элемент `style` внутрь элемента `head`. Чтобы не усложнять пример, используем вложенные CSS-стили, поэтому вам не нужно беспокоиться о создании отдельного CSS-файла.

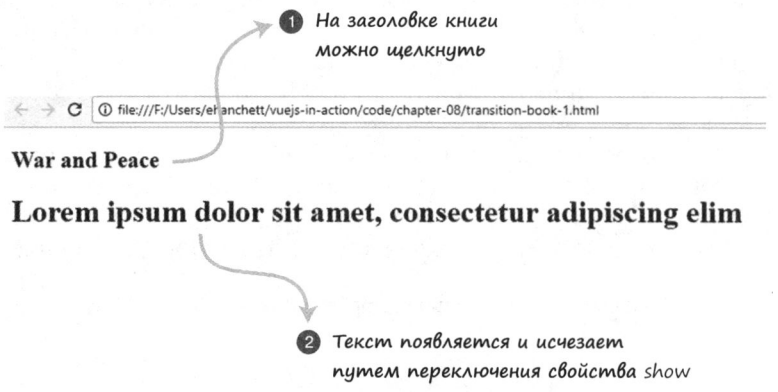


Рис. 8.1. Переключение без перехода

Vue.js предоставляет шесть CSS-классов, которые можно применить к переходам добавления и удаления. В данном примере воспользуемся четырьмя: `v-enter-active`, `v-leave-active`, `v-enter` и `v-leave-to`. Два других, которые мы задействуем позже, называются `v-enter-to` и `v-leave`.

В листинге 8.1 классов анимации нет. Как уже упоминалось, при отсутствии CSS-классов условие директивы `v-if` срабатывает мгновенно, без плавного перехода. Мы должны создать классы CSS-переходов, чтобы реализовать эффект плавного появления/исчезновения. Но сначала посмотрим, какие действия выполняет каждый из этих классов и на каких этапах они добавляются в документ и удаляются из него. Имейте в виду, что свойство `frame` представляет собой список всех элементов в текущем окне. В табл. 8.1 перечислены все классы CSS-переходов, о которых вам следует знать.

Таблица 8.1. Классы CSS-переходов

Класс перехода	Описание
v-enter	Это начальное состояние. Добавляется перед созданием элемента и удаляется через один кадр после его создания
v-enter-active	Этот класс присутствует в элементе, пока тот вставляется в документ. Он добавляется перед вставкой элемента и удаляется после завершения перехода/анимации. Здесь определяются продолжительность, задержка и кривая ослабления всего перехода
v-enter-to	Этот класс появился в Vue.js 2.1.8. Он добавляется через один кадр после вставки элемента и удаляется по окончании перехода/анимации
v-leave	Этот класс добавляется в момент, когда элемент покидает документ, и удаляется через один кадр после завершения этого процесса

Класс перехода	Описание
v-leave-active	Это активное состояние анимации/перехода. Класс похож на v-enter-active. С его помощью можно определить продолжительность, задержку и кривую ослабления. Он добавляется сразу после срабатывания перехода удаления и исчезает по окончании перехода/анимации
v-leave-to	Этот класс похож на v-enter-to и тоже появился в Vue.js 2.1.8. Он представляет собой заключительную фазу удаления. Добавляется через один кадр после срабатывания перехода и исчезает по окончании перехода/анимации

Каждый из этих классов создается и исчезает на разных этапах добавления и удаления элемента. С их помощью можно выполнять переходы и анимацию. Подробности ищите в официальной документации по адресу https://ru.vuejs.org/v2/guide/transitions.html#Классы-переходов.

Создадим переходы в элементе `style` внутри `head`. Но сначала обратите внимание на то, что в листинге 8.1 компоненту `transition` присвоен атрибут `name` со значением `fade`. Все наши CSS-классы будут начинаться с этого имени, а не с `v-`. Если бы мы опустили этот атрибут, названия классов остались бы прежними: `v-enter-active`, `v-leave-active` и т. д. Теперь же они будут называться, к примеру, `fade-enter-active` и `fade-leave-active`.

Создайте классы CSS-переходов `fade-enter-active` и `fade-leave-active` внутри элемента `style`, взяв за основу код приложения из предыдущего листинга. Как уже упоминалось, CSS-переходы с задержкой помещаются в классы `-active`. В данном примере мы зададим прозрачность, задержку длиной 2,5 с и кривую ослабления `ease-out`. В результате получится красивый эффект затухания, который длится 2,5 с.

Теперь создадим стили `fade-enter` и `fade-leave-to`. Изначально свойство `opacity` равно `0`. Это обеспечит корректный эффект появления с полной прозрачностью. Добавьте новые стили в предыдущий пример и посмотрите, что из этого получится (листинг 8.2). Чтобы вы не отвлекались, я убрал лишний код, не изменившийся по сравнению с листингом 8.1. Вы всегда можете свериться с полной версией, которая находится в архиве с кодом для этой книги.

Листинг 8.2. Плавный переход для описания: chapter-08/transition-book.html

```
...
  <style>
  .fade-enter-active, .fade-leave-active {
    transition: opacity 3.0s ease-out;
  }

  .fade-enter, .fade-leave-to {
    opacity: 0;
  }
  </style>
...
```

Активные состояния описывают продолжительность и ослабление перехода

Состояния enter и leave для 'opacity: 0'

Загрузите этот пример в браузере и откройте панель разработки. Если щелкнуть на названии книги и взглянуть на исходный код, можно заметить кое-что интересное: вокруг описания появятся новые классы, `fade-enter-active` и `fade-enter-to`, которые исчезнут по прошествии 3 с (рис. 8.2).

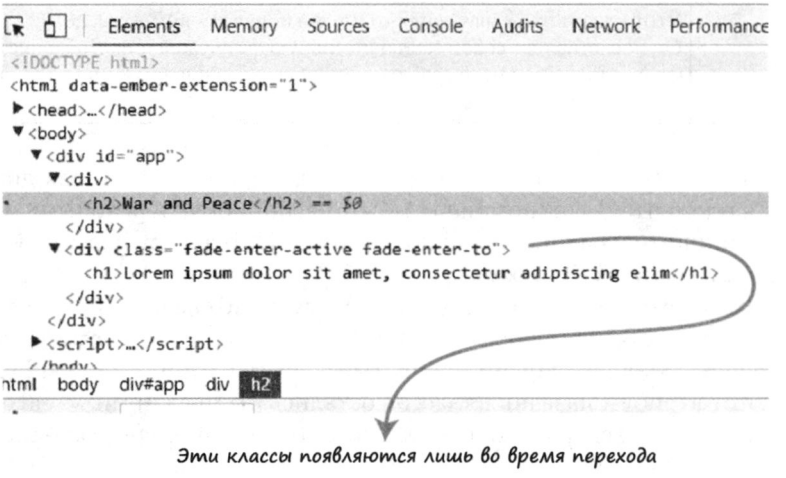


Рис. 8.2. Классы на момент добавления элемента в DOM

Эти классы появляются лишь во время перехода и затем удаляются. Если щелкнуть на заголовке еще раз, текст начнет плавно исчезать. В этот момент в браузере можно наблюдать добавление классов `fade-leave-active` и `fade-leave-to` к HTML-узлу (рис. 8.3).

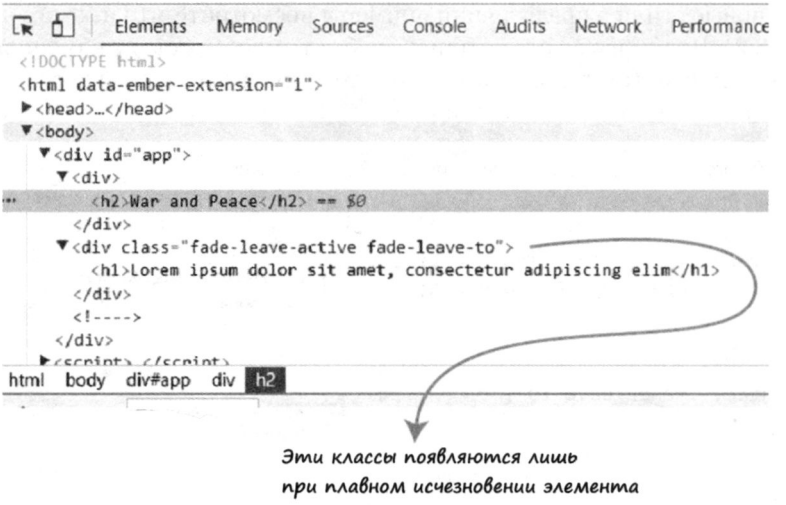


Рис. 8.3. Классы, появляющиеся во время удаления элемента из DOM

Эти классы присутствуют только во время удаления элемента из DOM. Таким образом, добавляя и удаляя классы в определенные моменты, Vue.js удается создавать изящные переходы и анимацию.

8.2. Основы анимации

Анимация — это еще одна задача, с которой Vue.js справляется на отлично. Вам, наверное, интересно, чем она отличается от перехода. Переход выполняется от одного состояния к другому, тогда как анимация способна иметь несколько состояний. В последнем примере текст сначала исчезал со страницы, а затем появлялся вновь.

С анимацией все немного иначе. В рамках одного определения можно указать множество состояний. Это позволяет создавать интересные эффекты, например сложные перемещения. Разные экземпляры анимации можно объединять в единое целое или использовать их как обычные переходы (хотя это разные вещи).

Возьмем пример из листинга 8.2 и добавим в него анимацию. Код останется прежним, но у текста появится эффект отскока, основанный на ключевых кадрах CSS. Мы хотим, чтобы при щелчке на заголовке элемент плавно появлялся и масштабировался. Повторный щелчок должен приводить к плавному исчезновению с обратным масштабированием. Откройте текстовый редактор и скопируйте листинг 8.2. Поменяйте имя перехода на `bounce`, как показано в листинге 8.3.

Листинг 8.3. Анимация с масштабированием: chapter-08/animation-book-1.html

```
<div @click="show = !show">
  <h2>{{title}}</h2>
</div>
<transition name="bounce">  ← Переход bounce
  <div v-if="show">
    <h1>{{description}}</h1>
  </div>
</transition>
```

Теперь нужно создать анимацию. Для этого потребуются классы `enter-active` и `leave-active`. Для начала удалите старые классы CSS-перехода. Затем добавьте вместо них классы `bounce-enter-active` и `bounce-leave-active`. Создайте CSS-анимацию `bounceIn 2s` внутри `bounce-enter-active`. Сделайте то же самое для класса `bounce-leave-active`, только с параметром `reverse`.

Теперь создайте ключевые кадры CSS. Используйте конструкцию `@keyframes` с состояниями `0%`, `60%` и `100%`. Мы применим CSS-трансформацию, изменяя масштаб до 0,1 (`0%`), 1,2 (`60%`) и 1 (`100%`). Изменим также свойство `opacity` с `0` на `1`. Добавьте все это в элемент `style` (листинг 8.4).

Откройте файл в браузере и проверьте анимацию. При щелчке на названии книги вы должны увидеть плавное появление и увеличение текста. При повторном щелчке текст должен плавно исчезнуть. На рис. 8.4 показана промежуточная стадия анимации.

Листинг 8.4. Анимация с масштабированием (целиком): chapter-08/animation-book.html

```
...
  <style>
    .bounce-enter-active {
      animation: bounceIn 2s;
    }
    .bounce-leave-active {
      animation: bounceIn 2s reverse;
    }

    @keyframes bounceIn {
      0% {
        transform: scale(0.1);
        opacity: 0;
      }
      60% {
        transform: scale(1.2);
        opacity: 1;
      }
      100% {
        transform: scale(1);
      }
  </style>
```

Вход в активное состояние с ключевым кадром bounceIn

Выход из активного состояния с ключевым кадром bounceIn

Ключевые кадры анимации

На этапе 0 % масштаб равен 0,1, а opacity — 0

На этапе 60 % масштаб равен 1,2, а opacity — 1

На заключительном этапе 100%-ный масштаб возвращается к 1

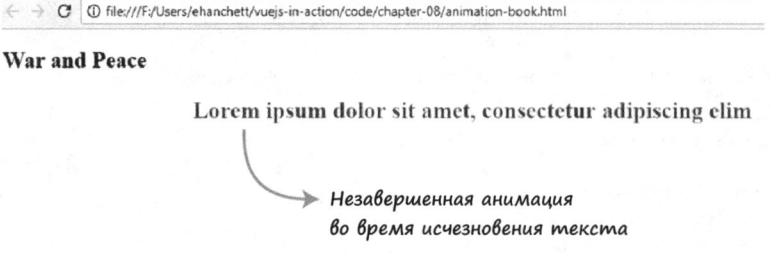


Рис. 8.4. Снимок экрана в момент перехода

Анимация создает эффект, при котором текст сначала увеличивается, а затем сжимается.

8.3. JavaScript-хуки

Классов анимации и переходов, которые предоставляет Vue.js, должно быть достаточно в большинстве ситуаций, но в случае необходимости нам доступно еще более гибкое решение. Для более сложных переходов и анимации предусмотрены специальные хуки, которые можно объединять с кодом на JavaScript для манипуляции и корректирования CSS-стилей.

Этот механизм напоминает хуки жизненного цикла Vue.js, которые мы обсуждали в одной из предыдущих глав. Однако он применяется лишь для переходов/анимации. Прежде чем его использовать, следует запомнить несколько нюансов. Во-

первых, работая с хуками `enter` и `leave`, всегда необходимо задействовать функцию обратного вызова `done`, в противном случае они будут выполнены синхронно, в результате чего переход окажется мгновенным. Кроме того, если переходы написаны исключительно на JavaScript, рекомендуется указывать атрибут `v-bind:css=" false"`, чтобы библиотека Vue могла пропустить обнаружение CSS-переходов. Последнее, о чем нужно помнить, — все хуки, кроме `enter` и `leave`, принимают параметр `el`, который также передается в функцию `done`. Это кажется немного странным, но не волнуйтесь: в следующем разделе вы сами увидите, как это работает.

В начале перехода можно использовать JavaScript-хуки `beforeEnter`, `enter`, `afterEnter` и `enterCancelled`. При завершении перехода доступны хуки `beforeLeave`, `leave`, `afterLeave` и `leaveCancelled`. Все они срабатывают на разных этапах анимации.

Возьмем предыдущий пример и превратим CSS-классы в JavaScript-хуки. Для начала удалим из листинга 8.4 классы `bounce-enter-active` и `bounce-leave-active`. Ключевые кадры пока что оставим. Теперь создадим анимацию на JavaScript, используя хуки для состояний `enter` и `leave`.

Перечислим все хуки в элементе `transition` (листинг 8.5). Для этого нужно применить директиву `v-on` или ее сокращенную версию `@`. Укажите следующие хуки: `before-enter`, `enter`, `before-leave`, `leave`, `after-leave`, `afterenter`, `enter-cancelled` и `leave-cancelled`.

Листинг 8.5. Переход на основе JavaScript-хуков: chapter-08/jshooks-1.html

```
<transition name="fade"                          ← Все хуки для перехода
    @before-enter="beforeEnter"
    @enter="enter"
    @before-leave="beforeLeave"
    @leave="leave"
    @after-leave="afterLeave"
    @leave-cancelled="leaveCancelled"
    :css="false">
```

Далее нужно добавить JavaScript-хуки в объект `methods` внутри экземпляра Vue. Чтобы сделать все правильно, мы должны знать момент завершения анимации. Это позволит убрать все классы из элемента `style` и в ответ на событие запускать функцию `done`, которая передается в качестве параметра в хуки `enter` и `leave`. Она должна выполняться в рамках этих хуков. Поэтому мы создадим новый обработчик для события `animationend`, наступающего при завершении анимации. Когда это происходит, запускается функция обратного вызова, которая сбрасывает стиль и выполняет `done`. Добавьте этот код в свой HTML-файл сверху от конструктора Vue (листинг 8.6).

Листинг 8.6. Обработчик событий для JavaScript-хуков: chapter-08/jshook-2.html

```
function addEventListener(el, done) {
  el.addEventListener("animationend", function() {   ← Обработчик событий реагирует
    el.style="";                                        на завершение анимации
    done();
});
```

Мы используем только два хука, `leave` и `enter`. Остальные просто выводят сообщения в консоль, чтобы вы могли понять, когда они срабатывают. Добавьте в свой экземпляр Vue новый объект `methods` со всеми хуками, которые вы указали в компоненте `transition`. Создайте внутри метода `enter` функцию для вызова `addEventListener`, выполненную ранее, и не забудьте передать параметры `element` и `done`, как показано в листинге 8.7. Теперь опишем анимацию на JavaScript. Анимация на основе ключевых кадров, которую мы создали внутри `style`, называется `el.style.animationName`. Свойству `el.style.animationDuration` будет присвоено `1.5s`.

Укажите те же свойства `animationName` и `animationDuration` внутри хука `leave`. Добавьте также свойство `el.style.animationDirection` с параметром `reverse`. Это позволяет обратить анимацию вспять, когда элемент покидает документ.

Листинг 8.7. Методы для JavaScript-хуков: chapter-08/jshooks.html

```
...
    methods: {
      enter(el, done) {
        console.log("enter");
        addEventListener(el,done);
        el.style.animationName = "bounceIn"
        el.style.animationDuration = "1.5s";
      },
      leave(el, done) {
        console.log("leave");
        addEventListener(el,done);
        el.style.animationName = "bounceIn"
        el.style.animationDuration = "1.5s";
        el.style.animationDirection="reverse";
      },
      beforeEnter(el) {
          console.log("before enter");
      },
      afterEnter(el) {
          console.log("after enter");
      },
      enterCancelled(el) {
          console.log("enter cancelled");
      },
      beforeLeave(el) {
          console.log("before leave");
      },
      afterLeave(el) {
          console.log("after leave");
      },
      leaveCancelled(el) {
          console.log("leave cancelled");
      }
    }

  });
...
```

Хук enter

Внутри хука enter вызывается обработчик addEventListener

Хук leave

Внутри хука leave вызывается обработчик addEventListener

Хук beforeEnter

Хук afterEnter

Хук enterCancelled

Хук beforeLeave

Хук afterLeave

Хук leaveCancelled

Это приложение должно вести себя точно так же, как пример из листинга 8.4. Щелкните на заголовке, чтобы запустить анимацию. При повторном щелчке анимация будет выполнена в обратном порядке. Обратите внимание на консоль. После первого щелчка вы увидите сообщения `before enter`, `enter`, а затем `after enter` (рис. 8.5).

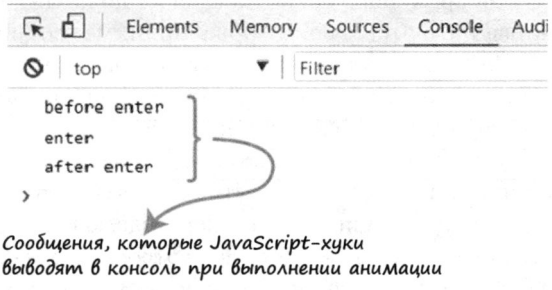


Рис. 8.5. Хуки, срабатывающие после щелчка на заголовке

Это порядок, в котором выполняются хуки. Сообщение `after enter` появляется только после завершения анимации. При повторном щелчке на заголовке мы можем наблюдать новую последовательность хуков: сначала идет `before leave`, потом `leave`, а в конце — `after leave` (рис. 8.6).

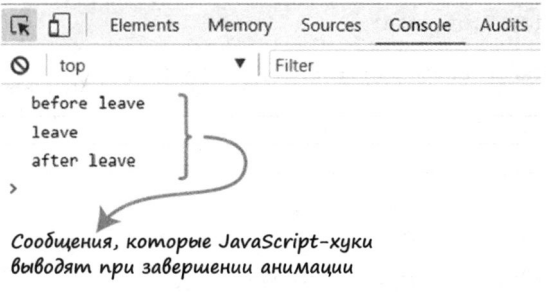


Рис. 8.6. Хуки, срабатывающие при удалении элемента из DOM

Если последить за исходным кодом, можно заметить добавление и удаление CSS-стилей при каждом щелчке. Похожим образом вели себя классы CSS-переходов, которые мы рассматривали ранее.

8.4. Переход между компонентами

В предыдущей главе мы познакомились с динамическими компонентами. Их легко менять с помощью атрибута `is`, который ссылается на текущий компонент.

Мы можем выполнять переход между компонентами по аналогии с тем, как делали ранее с условной директивой `v-if`. Для простоты возьмем за основу пример `dynamic-components.html` из главы 7. Скопируйте листинг 7.5 и внесите в него изменения, описанные далее.

Для начала поместите динамический компонент `<component :is=" currentView" />` внутрь элемента `<transition name=" component-fade" >`. Прежде чем продолжать, познакомимся с режимами перехода.

Вы можете заметить, что по умолчанию появление нового компонента и удаление старого происходит одновременно. Такое поведение подходит не всегда. Элемент `transition` поддерживает атрибут `mode`, который принимает одно из двух значений: `in-out` или `out-in`. Если указать значение `in-out`, старый компонент будет удален только после появления нового. В режиме `out-in` сначала удаляется старый компонент, а затем появляется новый. Именно он подходит для нашего примера, так как мы хотим показать новый компонент только после исчезновения существующего. Итак, поместим динамический компонент внутрь `<transition name=" component-fade" mode=" out-in" >`. Затем нужно добавить классы перехода, как показано в листинге 8.8. Создайте классы `component-fade-enter-active` и `componentfade-leave-active` и укажите для каждого из них свойство `opacity:0`. Далее показан весь код.

Листинг 8.8. Переход между динамическими компонентами: chapter-08/component-transition.html

```
<!DOCTYPE html>
<html>
<head>
<script src="https://unpkg.com/vue"></script>
<style>
.component-fade-enter-active, .component-fade-leave-active {
  transition: opacity 2.0s ease;
}
.component-fade-enter, .component-fade-leave-to {
  opacity: 0;
}

</style>
</head>
<body>
  <div class="app">
    <button @click="cycle">Cycle</button>
    <h1>
      <transition name="component-fade" mode="out-in">
        <component :is="currentView"/>
      </transition>
    </h1>
  </div>
<script>
const BookComponent ={
  template: `
  <div>
    Book Component
  </div>
  `
}

const FormComponent = {
  template: `
  <div>
```

Классы перехода между компонентами

Класс перехода, устанавливающий opacity

Компонент transition в режиме out-in

```
    Form Component
  </div>
  `
}

const HeaderComponent = {
  template: `
  <div>
    Header Component
  </div>
  `
}

new Vue({
  el: '.app',
  components: {'book-component': BookComponent,
               'form-component': FormComponent,
               'header-component': HeaderComponent},
  data() {
    return {
      currentView: BookComponent
    }

  },
  methods: {
        cycle() {
          if(this.currentView === HeaderComponent)
            this.currentView = BookComponent
          else
            this.currentView = this.currentView === BookComponent ?
              FormComponent : HeaderComponent;
        }
    }

})
</script>
</body>
</html>
```

Открыв браузер с текущим кодом, вы должны увидеть кнопку Cycle (Переключить). Если ее нажать, вместо `BookComponent` появится новый компонент, `FormComponent`. На рис. 8.7 показан снимок экрана с частично выполненным переходом.

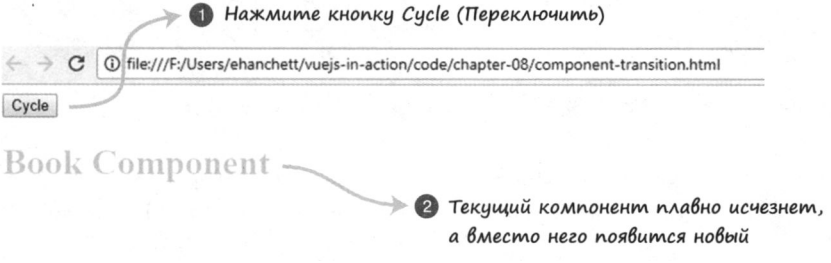


Рис. 8.7. Страница в момент перехода между компонентами

Если нажать кнопку Cycle (Переключить) еще раз, `HeaderComponent` заменит `FormComponent`. Следующее нажатие вернет нас к `BookComponent`.

8.5. Обновление зоомагазина

Мы уже обновляли проект зоомагазина в предыдущей главе при переходе на однофайловые компоненты с применением Vue-CLI. Теперь сделаем его шикарным, используя анимацию и переходы.

Имейте в виду, что в некоторых веб-приложениях анимация и переходы могут оказаться лишними. Не стоит ими злоупотреблять, если проект не высокодинамичный. В нашем примере будет один экземпляр анимации и один переход. Начнем с перехода.

8.5.1. Добавление перехода в проект зоомагазина

Мы хотим, чтобы страницы плавно появлялись и исчезали при переходе между маршрутами. Добавим простой эффект затухания в момент открытия страницы оформления покупки и возвращения на главную страницу. Для этого, как и прежде, используем классы анимации из Vue.js. На рис. 8.8 запечатлен процесс перехода к форме оформления заказа.

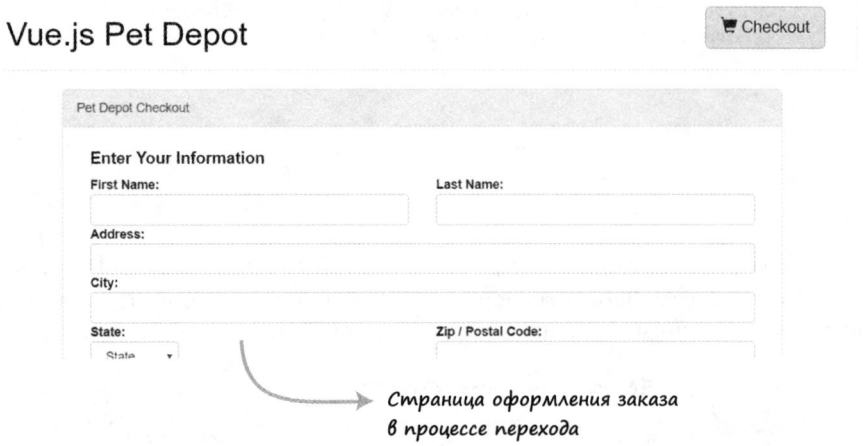


Рис. 8.8. Переход к странице оформления заказа

Найдите в архиве с кодом проект зоомагазина, над которым мы работали в главе 7. Откройте файл `App.vue` в папке `src` — здесь настраивали маршрутизацию. Поместите компонент `router-view` внутрь `transition`, как в предыдущих примерах. Не забудьте добавить режим `out-in` (листинг 8.9).

Теперь создайте в элементе `style`, размещенном внизу, два класса: `fade-enter-active` и `fade-leave-active`. Укажите свойство `opacity` со значением `.5s ease-out`. Добавьте также классы `fade-enter` и `fade-leave-to` со свойством `opacity:0`.

Листинг 8.9. Добавление анимированного перехода в зоомагазин: chapter-08/petstore/src/App.vue

```
<template>
  <div id="app">
    <transition name="fade" mode="out-in">
      <router-view></router-view>
    </transition>
  </div>
</template>

<script>
  export default {
    name: 'app'
  }
</script>

<style>
  #app {
  }
  .fade-enter-active, .fade-leave-active {
    transition: opacity .5s ease-out;
  }
  .fade-enter, .fade-leave-to {
    opacity: 0;
  }
</style>
```

Компонент transition в режиме out-in с router-view внутри

Классы Vue.js, описывающие переход

Классы Vue.js, устанавливающие свойство opacity

После внесения этих изменений сохраните файл и выполните команду `npm run dev`. Запустится веб-сервер, и в браузере должна открыться веб-страница. Если этого не произошло, пройдите по адресу localhost:8081 и проверьте приложение. Нажмите кнопку корзины и проследите за тем, как плавно исчезает главная страница.

8.5.2. Добавление анимации в проект зоомагазина

Для добавления элементов в корзину предусмотрены директивы `v-if`, `v-else-if` и `v-else`. Благодаря им посетитель знает, сколько единиц товара в наличии. Когда запасы исчерпываются, выводится сообщение All Out!. На этот случай добавим анимацию, которая заставляет текст дрожать и временно делает его красным. Для этого воспользуемся CSS-классами анимации из состава Vue.js, с которыми познакомились ранее. На рис. 8.9 показано, как сообщение All Out! двигается влево-вправо.

Откройте файл `Main.vue` и прокрутите его до конца. Для создания анимации мы применим один из встроенных классов, `enter-active`. Нам не нужно анимировать удаление элемента из DOM, поэтому класс `leave-active` не нужен.

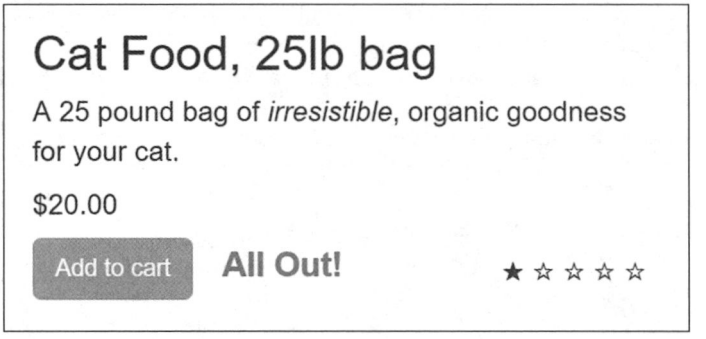


Рис. 8.9. Анимация для сообщения All Out!. Текст двигается влево-вправо

Итак, текст должен двигаться на протяжении 0,72 с. Мы воспользуемся такими возможностями CSS, как `cubic-bezier` и `transform`. Кроме того, подготовим ключевые кадры с шагом 10 %. Скопируйте листинг 8.10 в файл `Main.vue`.

Листинг 8.10. Добавление анимации в зоомагазин: chapter-08/petstore/src/components/Main.vue

```
...
<style scoped>
.bounce-enter-active {                    ← Этот класс инициирует анимацию
  animation: shake 0.72s cubic-bezier(.37,.07,.19,.97) both;
  transform: translate3d(0, 0, 0);
  backface-visibility: hidden;
}

@keyframes shake {                        ← Отдельные ключевые кадры
  10%, 90% {
    color: red;
    transform: translate3d(-1px, 0, 0);
  }

  20%, 80% {
    transform: translate3d(2px, 0, 0);
  }

  30%, 50%, 70% {
    color: red;
    transform: translate3d(-4px, 0, 0);
  }

  40%, 60% {
    transform: translate3d(4px, 0, 0);
  }
}
</style>
...
```

Вдобавок к CSS нужно создать элемент `transition`, в котором будет происходить анимация. Найдите в файле `Main.vue` сообщения о наличии товара. В каждом из них будет класс `inventory-message`. Поместите их внутрь элемента `transition` и не забудьте добавить в последний атрибут `mode="out-in"`, как показано в листинге 8.11.

Листинг 8.11. Применение анимации в элементе transition: chapter-08/petstore/src/components/Main.vue

```
...
<transition name="bounce" mode="out-in">
  <span class="inventory-message"
        v-if="product.availableInventory - cartCount(product.id) === 0"
        key ="0">
        All Out!
  </span>
  <span class="inventory-message"
        v-else-if="product.availableInventory - cartCount(product.id) < 5"
        key="">
    Only {{product.availableInventory - cartCount(product.id)}} left!
  </span>
  <span class="inventory-message"
        v-else key="">Buy Now!
  </span>
</transition>
...
```

Переход для bounce с атрибутом mode

Ключ со значением 0 для v-if

Еще один ключ, который препятствует анимации

В предыдущих примерах мы применяли анимацию и переходы лишь к одному элементу. Чтобы анимировать сразу несколько элементов, в директивы `v-else-if` и `v-if` необходимо добавить атрибут `key`, иначе Vue не сможет корректно преобразовать их содержимое.

Атрибут key

Атрибут `key` необходим в случаях, когда нужно преобразовать несколько элементов с одинаковыми тегами. Это уникальный ключ, который помогает их различить. Рекомендуется добавлять его в ходе работы с несколькими элементами внутри компонента `transition`.

Добавьте пустой атрибут `key` в директивы `v-else-if` и `v-else`. Это делается намеренно, чтобы не анимировать соответствующие сообщения. Для директивы `v-if` укажем ключ `0`, как показано далее.

Запустите веб-сервер с помощью команды `npm run dev` и проверьте работу приложения. Нажимайте кнопку добавления в корзину, пока не закончится товар. Вы увидите, как сотрясается, а затем становится красным сообщение All Out!.

Упражнение

Используя знания, приобретенные в этой главе, ответьте на вопрос: в чем разница между анимацией и переходом?

Ответ ищите в приложении Б.

Резюме

- Переходы могут перемещать элементы на странице.
- Анимация позволяет использовать масштабирование, в том числе программируемое.
- Для сложной анимации можно применять JavaScript-хуки.
- Переходы в сочетании с динамическими компонентами позволяют циклически перебирать элементы страницы.

9 Расширение Vue

В этой главе

- Примеси.
- Пользовательские директивы.
- Функция отрисовки.
- Реализация JSX.

В предыдущей главе мы обсудили переходы и анимацию. Здесь же рассмотрим разные подходы к многократному использованию кода в Vue.js. Это важная тема, поскольку с помощью данных методик мы сможем расширять возможности Vue-приложений и делать их более надежными.

Начнем с *примесей* (mixins). Это механизм распределения функциональности между компонентами, то есть *примешивания* к ним общей информации. Примесь — это объект с методами и свойствами, характерными для любого компонента в Vue.js. Далее мы познакомимся с *пользовательскими директивами*. Это возможность регистрировать собственные директивы с любыми функциями. Мы также затронем *функцию отрисовки* (`render`), которая позволяет создавать шаблоны с помощью JavaScript. В конце вы научитесь применять функцию `render` в сочетании с JSX — синтаксисом шаблонов в стиле XML.

Не волнуйтесь, я не забыл о проекте зоомагазина. В главе 10 мы перепишем его с применением Vuex.

9.1. Повторное использование кода с помощью примесей

Примеси могут пригодиться во многих проектах. Они позволяют выделить некую функциональность и сделать ее общей для нескольких компонентов. При написании Vue-приложений вы заметите, что компоненты начинают походить друг на друга. *DRY* (don't repeat yourself — «не повторяйся») — один из важных принципов

проектирования программного обеспечения. Если один и тот же код встречается в разных компонентах, его следует оформить как примесь.

Представьте, что вам нужно получить номер телефона или адрес электронной почты клиента. Приложение будет содержать два разных компонента. Каждый из них состоит из формы с полем ввода и кнопкой. При нажатии кнопки выводится диалоговое окно с введенным текстом. Итоговый результат показан на рис. 9.1.

Рис. 9.1. Пример компонентов с примесью

Этот слегка искусственный пример демонстрирует, как оформить логику в виде примеси. В данном случае мы выделили код, который обрабатывает нажатие кнопки и выводит диалоговое окно. Это позволяет упорядочить код и избежать повторений.

Сначала создадим файл `mixins.html`. Добавим в него тег `script` для Vue.js и тег `link` для подключения стилей Bootstrap. После этого опишем простую табличную HTML-разметку на основе Bootstrap с одной строкой и тремя столбцами размером `col-md-3`. Наши два компонента находятся в первом и третьем столбцах, при этом первый столбец получает смещение `col-md-offset-2`.

Откройте файл `mixins.html` и скопируйте в него содержимое листинга 9.1. Это лишь первая часть нашего приложения. На протяжении раздела продолжим добавлять в него код. Если вас интересует итоговый результат, его можно найти в одноименном файле в архиве с кодом для этой книги.

Листинг 9.1. HTML/CSS для примера: chapter-09/mixin-html.html

```
<!DOCTYPE html>
<script src="https://unpkg.com/vue"></script>
<link rel="stylesheet"
➥ href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
<html>
<head>
</head>
```

Подключение Bootstrap

```
<body>
  <div id="app">
    <div id="container">
      <h1 class="text-center">{{title}}</h1>
      <div class="row">
        <div class="col-md-3 col-md-offset-2">
          <my-comp1 class="comp1"></my-comp1>
        </div>
        <div class="col-md-3">
            <h2 class="text-center">Or</h2>
        </div>
        <div class="col-md-3">
          <my-comp2 class="comp2"></my-comp2>
        </div> <!-- Конец col-md-2-->
      </div><!-- Конец row -->
    </div> <!-- Конец container -->
  </div> <!-- Конец app -->
```

Первый компонент

Табличная разметка Bootstrap со столбцами

Второй компонент

Добавив HTML-разметку, можем приступить к коду для Vue.js. Откройте файл `mixins.html` и создайте в нем открывающий и закрывающий теги `script`. Если бы мы использовали в этом примере однофайловые компоненты и Vue-CLI, все работало бы точно так же, только каждый компонент и примесь находились бы в отдельных файлах.

Создайте новый экземпляр Vue внутри элемента `script` и добавьте в него функцию `data`, которая возвращает объект со свойством `title`. Мы также должны объявить оба компонента в разделе `components`. Скопируйте код листинга 9.2 в файл `mixins.html`.

Листинг 9.2. Добавление экземпляра Vue: chapter-09/mixins-vue.html

```
<script>
  new Vue({
    el: '#app',
    data() {
      return {
        title: 'Mixin in example using two components'
      }
    },
    components:{
        myComp1: comp1,
        myComp2: comp2
    }

  });
  </script>
</script>
```

Объявление корневого экземпляра Vue

Функция данных, возвращающая свойство title

Объявление компонентов myComp1 и myComp2

Осталось сделать еще несколько вещей. Мы должны создать оба компонента и примесь. Каждый компонент отображает текст, поле ввода и кнопку. При нажатии кнопки следует вывести введенный текст с помощью диалогового окна.

Компоненты имеют некоторое сходство. Они выводят заголовок, поле ввода и кнопку. И нажатие кнопки приводит к похожим результатам. На первый взгляд может показаться, что стоит объединить их в единый компонент, но их внешний вид и поведение разнятся: кнопки имеют разные визуальные стили, а поля ввода принимают разные значения. Этот пример будет реализован в виде двух отдельных компонентов.

Тем не менее, если не принимать во внимание шаблон, мы имеем дело со схожей бизнес-логикой. Нам нужно создать примесь с методом `pressed`, который отображает диалоговое окно. Откройте файл `mixin.html` и добавьте над экземпляром Vue константу `myButton`. Создайте внутри нее функции `pressed` (на основе `alert`) и `data`, последняя должна возвращать объект со свойством `item`, как показано в листинге 9.3.

Листинг 9.3. Добавление примеси: chapter-09/my-mixin.html

```
<script>

const myButton = {
  methods: {
    pressed(val) {
      alert(val);
    }
  },
  data() {
    return {
      item: ''
    }
  }
}
```

Примесь myButton

Методы примеси

Функция pressed, выводящая диалоговое окно

Создав примесь, можем приступить к определению компонентов. Добавьте после `myButton` два новых объекта, `comp1` и `comp2`. Каждый из них будет содержать тег `h1`, форму и кнопку.

В компоненте `comp1` поле ввода привязывается к свойству `item` с помощью директивы `v-model`. Кнопка содержит директиву `v-on` (ее сокращенную версию `@`), которая привязывает событие `click` к методу `pressed`. Этот метод принимает вышеупомянутое свойство. Напоследок необходимо объявить примесь, которую мы создали ранее. Укажите ее внизу внутри массива `mixins`, как показано в листинге 9.4.

В компоненте `comp2` создадим тег `h1` с формой, полем ввода и кнопкой. Для привязки свойства `item` тоже используется директива `v-model`. Как и в `comp1`, нажатие кнопки обрабатывается методом `pressed` с помощью сокращенной версии директивы `v-on`. В этот метод передается свойство `item`. Как и прежде, нужно указать примеси, которые хочется задействовать в этом компоненте, для этого внизу следует создать массив `mixins`.

ПОМНИТЕ

Примеси не разделяются между компонентами — каждый из них получает свою копию. Переменные внутри примесей тоже не являются общими.

Мы также придали стили элементам формы с помощью классов из состава Bootstrap, но я не стану углубляться в подробности этого процесса.

Листинг 9.4. Добавление компонентов: chapter-09/comp1comp2.html

```
...
const comp1 = {                                  ← Объявление первого компонента
    template: `<div>
    <h1>Enter Email</h1>
    <form>
      <div class="form-group">
        <input v-model="email"                   ← Директива v-model,
        type="item"                                связывающая item
        class="form-control"
        placeholder="Email Address"/>
      </div>
      <div class="form-group">
        <button class="btn btn-primary btn-lg"
          @click.prevent="pressed(item)">Press Button 1</button>   ← Директива v-on
      </div>                                                         (сокращенно @),
    </form>                                                          привязывающая
    </div>`,                                                         событие click к pressed
    mixins: [myButton]                           ← Объявление примеси
                                                   для компонента
  }
  const comp2 = {
    template: `<div>
    <h1>Enter Number</h1>
      <form>
        <div class="form-group">                 ← Директива v-model
            <input v-model="item"                  привязывает поле ввода к item
            class="form-control"
            placeholder="Phone Number"/>
        </div>
        <div class="form-group">
          <button class="btn btn-warning btn-lg"
            @click.prevent="pressed(item)">Press Button 2</button>   ← Директива v-on
        </div>                                                         (сокращенно @),
      </form>                                                          привязывающая
    </div>`,                                                           событие click к pressed
    mixins:[myButton]                            ← Объявление примеси внизу

  }
...
```

Откройте браузер и загрузите файл `mixins.html`, над которым мы работали. Вы должны увидеть что-то похожее на рис. 9.1. Введите адрес электронной почты

в поле Enter Email (Введите адрес почты). При нажатии кнопки должно появиться диалоговое окно, как на рис. 9.2.

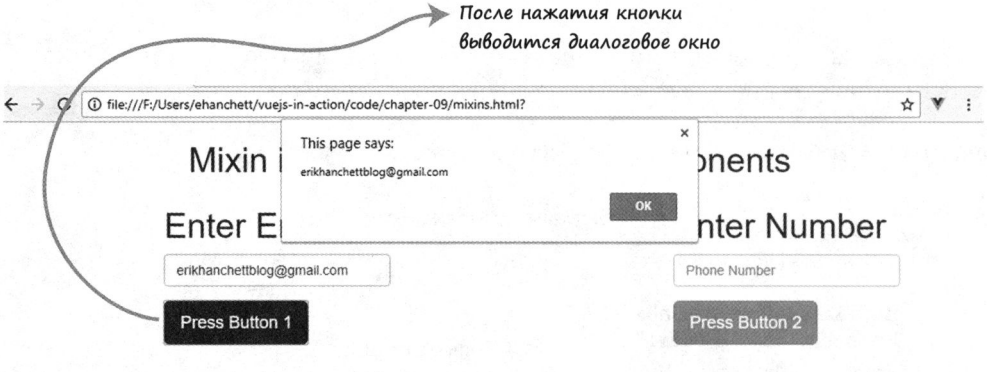


Рис. 9.2. Та же страница, что и на рис. 9.1, но после нажатия первой кнопки

Аналогичным образом работает и ввод телефонного номера.

9.1.1. Глобальные примеси

До сих пор мы использовали именованные примеси, которые объявляются внутри компонентов. Но примеси могут быть также *глобальными* и не требовать никакого объявления. Они применяются ко всем экземплярам Vue, создаваемым в приложении.

С глобальными примесями следует работать осторожно, так как они затрагивают любые сторонние инструменты, с которыми вы работаете. Они хорошо подходят в случаях, когда требуется добавить определенные свойства в каждый компонент и экземпляр Vue.js. Представьте, что вам нужно реализовать аутентификацию в приложении и вы хотите, чтобы информация о пользователе была доступна в любом компоненте. Чтобы не регистрировать примесь каждый раз, когда она нужна, можете сделать ее глобальной.

Для примера рассмотрим приложение из предыдущего раздела. Изменим его так, чтобы использовалась глобальная примесь. Сначала скопируем содержимое `mixins.html` в файл `mixins-global.html`. Здесь выполним рефакторинг приложения.

Найдите строчку `const myButton` в теге `script`. Это наша примесь. Чтобы сделать ее глобальной, вместо константы нужно использовать выражение `Vue.mixin`, которое внедряет примесь в каждый экземпляр Vue.js. Удалите ключевое слово `const` и добавьте вверху `Vue.mixin({`. Затем закройте скобки внизу, как показано в листинге 9.5.

Листинг 9.5. Глобальная примесь: chapter-09/global-mixin.html

```
...
  Vue.mixin({                    ← Объявление
    methods: {                     глобальной примеси
      pressed(val) {
```

```
        alert(val);
      }
    },
    data() {
        return {
            item: ''
        }
    }
  });        ◄—— Закрывающие скобки
...
```

Теперь, когда у нас есть глобальная примесь, можем убрать объявление `myButton`. Удалите строчку `mixins: [myButton]` из каждого компонента. Этого должно быть достаточно. Новый файл `mixins-global.html` должен вести себя точно так же, как и предыдущий.

ОТЛАДКА

Если вы столкнулись с какими-либо проблемами, причиной может быть объявление примеси внизу компонента. Не забудьте удалить все ссылки на myButton, иначе возникнет ошибка.

9.2. Пользовательские директивы с примерами

В предыдущих восьми главах мы имели дело с всевозможными директивами, включая `v-on`, `v-model` и `v-text`. Но что, если ни одна из них не подходит? На этот случай предусмотрены *пользовательские директивы* с низкоуровневым доступом к элементам DOM. С их помощью мы расширим возможности любого элемента на странице.

Имейте в виду, что пользовательские директивы отличаются от компонентов и примесей. Они тоже способствуют повторному применению кода, но делают это немного иначе. Компоненты отлично подходят для разбиения большого участка кода на отдельные модули, оформленные в виде тегов. Обычно они содержат сразу несколько элементов и шаблон. Примеси помогают разделить логику на небольшие фрагменты кода, которые можно подключать к разным компонентам и экземплярам. Пользовательские директивы предоставляют низкоуровневый доступ к элементам DOM. Прежде чем применять любую из этих концепций, следует подумать над тем, какая из них лучше всего подойдет для решения вашей задачи.

Существует два вида директив — локальные и глобальные. Глобальные директивы доступны для любого элемента на любом участке приложения. Они распространены шире, так как обычно разработчики хотят, чтобы директивы можно было задействовать везде.

Локальные директивы доступны только в тех компонентах, в которых зарегистрированы. Это удобно тогда, когда директива создана для какой-то узкоспециализированной задачи. Например, вы можете создать особый раскрывающийся список, который работает только с определенным компонентом.

Прежде чем продолжать, создадим простую локальную пользовательскую директиву, которая устанавливает для элемента цвет, размер шрифта и класс Bootstrap. Результат показан на рис. 9.3.

Рис. 9.3. Текст, оформленный с помощью пользовательской директивы

Создайте новый файл и назовите его `directive-example.html`. Внутри него выполните простую HTML-разметку с тегом `script` для Vue и подключением таблицы стилей Bootstrap. В рамках этого приложения мы создадим новую директиву `v-style-me` и применим ее к тегу `p`, как показано в листинге 9.6.

Листинг 9.6. Локальная пользовательская директива для Vue.js: chapter-09/directive-html.html

```
<!DOCTYPE html>
<html>
<head>
  <script src="https://unpkg.com/vue"></script>
  <link rel="stylesheet"
    href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css
➥ /bootstrap.min.css">
</head>
<body>
  <div id='app'>
      <p v-style-me>
          {{welcome}}
      </p>
  </div>
```

Подключение Bootstrap

Пользовательская директива style-me

Все пользовательские директивы начинаются с `v-*`. Применив `v-style-me` к тегу `p`, можем приступить к созданию логики приложения.

Создайте экземпляр Vue и функцию данных. Она должна возвращать сообщение с приветствием. Затем нужно добавить объект `directives`, в котором будет объявлена и зарегистрирована локальная пользовательская директива.

Создайте функцию `styleMe`. У каждой директивы есть доступ к ряду аргументов, которые она может задействовать:

- `el` — элемент, к которому привязана директива;
- `binding` — объект с такими свойствами, как `name`, `value`, `oldValue` и `expression` (полный список приводится в официальном руководстве по адресу https://ru.vuejs.org/v2/guide/custom-directive.html#Аргументы-хуков);
- `vnode` — виртуальный узел, сгенерированный компилятором Vue;
- `oldVnode` — предыдущий виртуальный узел.

В примере нам пригодится только `el` (элемент). Этот аргумент всегда идет первым. Как вы помните, имя директиве `styleMe` дано в верблюжьем стиле, поэтому в шаблоне должен применяться шашлычный стиль (`v-style-me`).

Любая пользовательская директива должна содержать один или несколько хуков, подобных хукам анимации и жизненного цикла, которые мы рассматривали в предыдущих главах. Эти хуки вызываются на разных этапах работы директивы.

- `bind` вызывается лишь один раз, когда директива привязывается к элементу. Это подходящее место для подготовительных действий.
- `inserted` вызывается, когда привязанный элемент вставляется в родительский узел.
- `update` вызывается после обновления VNode компонента-контейнера.
- `componentUpdate` вызывается после обновления VNode компонента-контейнера и VNode его потомков.
- `unbind` вызывается, когда директива отвязывается от элемента.

Вам, наверное, интересно, что такое VNode. *VNode* (сокращение от virtual node — «виртуальный узел») — это часть механизма Virtual DOM, который Vue.js создает во время запуска приложения. VNode используется в виртуальном дереве, формируемом при взаимодействии Vue.js с документом.

Для нашего простого примера достаточно будет хука `bind`. Он срабатывает, как только директива привязывается к элементу. Это подходящее место для работы со стилями. Мы воспользуемся методами элемента `style` и `className`. Для начала сделаем текст синим, затем установим размер шрифта (`42px`) и в конце присвоим класс `text-center`.

Итак, обновим файл `directive-example.html`. В листинге 9.7 показан итоговый код.

Листинг 9.7. Локальная директива в экземпляре Vue: chapter-09/directive-vue.html

```
<script>
  new Vue({
  el: '#app',
  data() {
    return {
      welcome: 'Hello World'
    }
  },
  directives: {
    styleMe(el, binding, vnode, oldVnode) {
      bind: {
        el.style.color = "blue";
        el.style.fontSize= "42px";
        el.className="text-center";
      }
    }
  }
  });
</script>
</body>
</html>
```

Функция данных возвращает свойство welcome

Здесь регистрируются директивы

Имя локальной пользовательской директивы и ее аргументы

Хук bind

Загрузив страницу в браузере, вы должны увидеть приветствие. Теперь можно применять эту пользовательскую директиву к любому элементу. Создайте новый тег `div` и добавьте в него директиву `v-style-me`. После обновления страницы текст должен выровняться по центру, изменить размер шрифта и стать синим.

9.2.1. Глобальные пользовательские директивы с модификаторами, значениями и аргументами

Познакомившись с локальными директивами, перейдем к глобальным. Сначала преобразуем наш простой пример, а затем рассмотрим связующие аргументы. Это позволит добавить несколько новых возможностей пользовательской директиве. Сделаем так, чтобы ей можно было передавать цвет текста. Добавим также модификатор с возможностью выбора размера текста и передадим имя класса в качестве аргумента.

Когда закончим, страница будет выглядеть так, как на рис. 9.4.

Рис. 9.4. Применение глобальных директив со связующим аргументом

Скопируйте последний пример из `directive-example.html` в файл `directive-global-example.html`. Первым делом нужно избавиться от объекта `directives` в экземпляре Vue.js. Он находится снизу от функции данных.

Теперь создайте глобальную директиву с помощью выражения `new Vue.directive`. Укажите ее имя `style-me` в качестве первого аргумента. Затем выберите имя хука. Здесь, как и в предыдущем примере, используется хук `bind`.

Хук `bind` принимает два аргумента: `el` и `binding`. Первый представляет собой сам элемент. Как и в предыдущем примере, можем задействовать его для изменения свойств `fontSize`, `className` и `color`, принадлежащих элементу, к которому мы привязываемся. Аргумент `binding` — это объект. Нас интересуют его свойства `binding.modifiers`, `binding.value` и `binding.arg`.

Самое простое в применении — свойство `binding.value`. С его помощью новой пользовательской директиве передается значение. Например, мы можем присвоить `binding.value` строку `'red'`:

```
v-style-me="'red'"
```

А также передать несколько значений внутри объекта:

```
v-style-me="{ color: 'orange', text: 'Hi there' }"
```

Каждое из них будет доступно в виде `binding.value.color` и `binding.value.text`. Как показано в листинге 9.8, мы присвоили `binding.value` значение `el.style.color`. Если свойства `binding.value` не существует, по умолчанию применяется синий цвет.

Для доступа к `binding.modifiers` в конце пользовательской директивы указывается точка:

```
v-style-me.small
v-style-me.large
```

Свойство `binding.modifers.large` может быть равно `true` или `false` в зависимости от того, как была объявлена пользовательская директива. Как видно в листинге 9.8, мы проверяем истинность `binding.modifiers.large`. Если свойство равно `true`, устанавливаем `42px` в качестве размера шрифта. Если свойство `binding.modifiers.small` равно `true`, размер шрифта будет `17px`. Если ни одно из этих свойств не установлено, шрифт не меняется.

Напоследок обратите внимание на связующий аргумент, объявленный в пользовательской директиве через двоеточие. В данном примере он называется `text-center`:

```
v-style-me:text-center
```

При этом все три свойства, `modifiers`, `args` и `values`, можно использовать в связке. Например, присвоим `binding.arg`, `binding.modifier` и `binding.value` значения `'red'`, `large` и `text-center` соответственно:

```
v-style-me:text-center.large="'red'".
```

После создания глобальной пользовательской директивы вернитесь к HTML-разметке и добавьте ее второй экземпляр с текстом `'Hi everybody'`. Здесь мы задействуем модификатор `small`, как показано далее.

Листинг 9.8. Окончательная версия глобальной директивы: chapter-09/directive-global-example.html

```
<!DOCTYPE html>
  <script src="https://unpkg.com/vue"></script>
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/
  bootstrap/3.3.7/css/bootstrap.min.css">
<html>
<head>
</head>
<body>
  <div id='app'>
      <p v-style-me:text-center.large="'red'">
          {{welcome}}
      </p>
      <div v-style-me.small>Hi everybody</div>
  </div>
```

Подключение Bootstrap

Объявление пользовательской директивы со значениями, аргументом и модификатором

Вторая пользовательская директива с одним лишь модификатором

```
<script>
  Vue.directive('style-me', {
    bind(el, binding) {
        el.style.color = binding.value || "blue";

        if(binding.modifiers.large)
          el.style.fontSize= "42px";
        else if(binding.modifiers.small)
          el.style.fontSize="17px"

        el.className=binding.arg;
      }
  });

new Vue({
  el: '#app',
  data() {
    return {
      welcome: 'Hello World'
    }
  }
});
</script>
</body>
</html>
```

Объявление глобальной пользовательской директивы с хуком bind

Свойству el.style.color присваивается binding.value или blue

Для изменения шрифта проверяется свойство binding.modifiers

Свойству binding.arg назначается имя класса элемента

Загрузив страницу в браузере, вы должны увидеть сообщение `'Hello World'`. Обратите внимание на то, что вторая строка, `'Hi Everybody '`, не выровнена по центру и меньшего размера. Это связано с тем, что директива `v-style-me`, которую мы к ней применили, использует лишь модификатор `small`. Это влияет на размер шрифта, при этом используется цвет по умолчанию (синий).

Если внимательно присмотреться к исходному коду, можно заметить, что второму элементу `div` назначается класс `undefined`, поскольку в пользовательской директиве атрибуту `el.className` присваивается свойство `binding.arg`. Но, так как мы не объявили это свойство, оно остается неопределенным. Старайтесь избегать подобных ситуаций — перед присваиванием свойство `binding.arg` стоит проверять. Сделайте это самостоятельно.

9.3. Функция отрисовки и JSX

До сих пор во всех наших Vue-приложениях использовались шаблоны. Обычно этого достаточно, но иногда вам может потребоваться вся мощь JavaScript. В таких случаях вместо шаблона определите собственную функцию отрисовки. Она должна быть написана на JavaScript, но задача у нее все та же — выводить HTML.

JSX — это XML-подобный язык разметки, который преобразуется в JavaScript с помощью расширения. У него то же назначение, что и у функции отрисовки. Чаще всего он используется в связке с React — еще одним клиентским фреймворком. Весь

потенциал JSX можно реализовать и в Vue.js, но для этого необходимо установить специальное расширение Babel. Для сравнения: функция отрисовки доступна в любом экземпляре Vue.js и не требует дополнительной конфигурации.

По своему опыту могу сказать, что создание сложной разметки с помощью функции отрисовки — непростая задача. Вам будут недоступны такие стандартные директивы, как `v-for`, `v-if` и `v-model`. Вы можете воспользоваться аналогичными конструкциями, но для этого придется написать дополнительный код на JavaScript. Хорошая альтернатива этому подходу — JSX. У этого языка есть большое сообщество разработчиков, а его синтаксис значительно ближе к шаблонам. Вместе с тем вам будет доступна вся мощь JavaScript. Расширение Babel для Vue.js и JSX хорошо поддерживается, что тоже приятно. Учитывая все сказанное, я сделаю основной акцент на JSX, а функцию отрисовки затрону лишь вскользь.

СОВЕТ

Если вы хотите узнать больше о функции отрисовки, можете почитать официальное руководство по адресу ru.vuejs.org/v2/guide/render-function.html.

9.3.1. Пример функции отрисовки

Создадим простое приложение с применением функции отрисовки. Представьте, что у нас есть глобальный компонент со свойством `welcome`. Нужно вывести это свойство в заголовке страницы. Для выбора типа заголовка (`h1`, `h2`, `h3`, `h4` или `h5`) воспользуемся входным параметром `header`. Кроме того, при щелчке на сообщении будет выводиться диалоговое окно. Чтобы сделать пример аккуратным, возьмем классы из состава Bootstrap. Следует также убедиться в том, что тег `h1` имеет класс `text-center`. Итоговый результат показан на рис. 9.5.

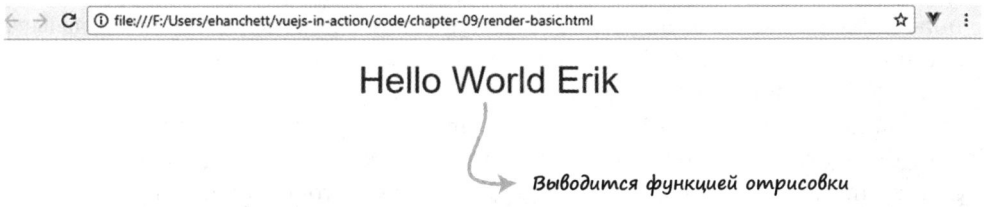


Рис. 9.5. Пример использования функции отрисовки

Создайте файл `render-basic.html`. В нем разместится наше небольшое приложение. Начнем с HTML-разметки. Подключите Vue.js и Bootstrap с помощью тегов `script` и `link`.

Добавьте в тело документа элемент `div` с идентификатором `app` и создайте внутри него компонент `my-comp`. Тег `div` необязателен, мы могли бы присвоить идентификатор самому компоненту. Но, чтобы сделать код более понятным, оставим `div` без изменений. `my-comp` получит входной параметр `header`, которому мы присвоим `1`.

Между открывающим и закрывающим тегами вставим имя `Erik`. Как известно из предыдущей главы, все, что размещается между этими тегами, доступно в виде слота. Чуть позже вы увидите, как работать с содержимым слота из функции отрисовки. Скопируйте следующий код (листинг 9.9) и вставьте его в файл `render-basic.html`.

Листинг 9.9. Основная разметка приложения: chapter-09/render-html.html

```
<!DOCTYPE html>
<html>
<head>
  <script src="https://unpkg.com/vue"></script>
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/
  ➥ bootstrap/3.3.7/css/bootstrap.min.css">
</head>
<body>
  <div id="app">
    <my-comp header="1">Erik</my-comp>
  </div>
<script>
```

Скрипт Vue.js

Подключение Bootstrap

Компонент с параметром header

Закончив с HTML-кодом, создадим корневой экземпляр Vue.js. Перейдите вниз страницы и добавьте внутрь тега `script` выражение `new Vue({el: '#app'})`.

Теперь создадим глобальный компонент. Он должен находиться сверху от корневого экземпляра, иначе получится ошибка. Первым делом нужно определить функцию данных, которая возвращает `welcome`. Назначим этому свойству значение `"Hello World"`. Нам также следует определить входной параметр `header`.

Вместо свойства `template` объявим в компоненте функцию `render`, которая принимает один аргумент, `createElement` (иногда его называют `h`, но это ничего не меняет). `createElement` тоже является функцией, мы должны вернуть ее с подходящими аргументами.

Функция `createElement` формирует Virtual DOM, описывая родительские и дочерние элементы, которые вы хотели бы видеть в своем документе. Их также называют виртуальными узлами (VNode). В сущности, вы строите дерево, представляющее Virtual DOM и состоящее из экземпляров VNode.

Как видно в листинге 9.10, `createElement` принимает три аргумента. Первый может быть строкой, объектом или функцией. Обычно он содержит элементы DOM, такие как `div`. Второй аргумент — это объект, который представляет атрибуты элемента. В качестве третьего аргумента выступает массив или строка. Строка представляет текст, который содержится внутри тега, а массив обычно описывает дочерние виртуальные узлы. Каждый экземпляр VNode эквивалентен еще одному вызову `createElement` и принимает тот же набор аргументов. В этом примере будем использовать только строку.

Создайте функцию `render(createElement)` внутри `Vue.component`. В качестве результата верните вызов `createElement`, как показано в листинге 9.10. Первым аргументом будет тег `header`, чтобы сформировать его значение, воспользуемся входным параметром. В данном случае мы передали `1` — это означает, что нужно

создать элемент h1. Чтобы это сделать, мы можем объединить букву h с параметром this.header.

Следующим аргументом будет объект с атрибутом. Поскольку мы работаем с Bootstrap, для выравнивания текста по центру экрана примените класс text-center. Для этого создадим объект со свойством class, значение которого равно text-center. В JavaScript class — это зарезервированное ключевое слово, поэтому его нужно заключить в кавычки. Следующий атрибут, обработчик события, представлен в функции отрисовки свойством on. В данном примере мы используем событие click и выводим диалоговое окно с текстом 'Clicked'.

В качестве последнего аргумента выберите массив или строку. Тем самым будет определено содержимое тега header. Чтобы сделать этот пример более интересным, объединим приветствие, объявленное в функции данных, с текстом, находящимся между открывающим и закрывающим тегами компонента (Erik). Этот текст считается значением по умолчанию и доступен в виде свойства this.$slots.default[0].text. Теперь можно объединить его с приветствием. Скопируйте следующий код в файл render-basic.html. Этого должно быть достаточно.

Листинг 9.10. Добавление функции отрисовки: chapter-09/render-js.html

```
Vue.component('my-comp',{
  render(createElement) {
      return createElement('h'+this.header,
      {'class':'text-center',
        on: {
          click(e){
            alert('Clicked');
          }
        }
      },
      this.welcome + this.$slots.default[0].text  )
  },
data() {
  return {
    welcome: 'Hello World '
  }
},
  props: ['header']
});
new Vue({el: '#app'})
</script>
</body>
</html>
```

Глобальный компонент my-comp

Функция отрисовки с аргументом createElement

Возвращает аргумент createElement

Объект, описывающий атрибуты для класса и события click

Строка, которая будет выведена в элементе header

Корневой экземпляр Vue.js, который следует добавить

Загрузите файл render-basic.html. Страница должна содержать сообщение Hello World Erik. Поменяйте значение входного параметра header. По мере увеличения числа текст должен становиться все меньше. Щелкнув на сообщении, вы должны получить результат, показанный на рис. 9.6.

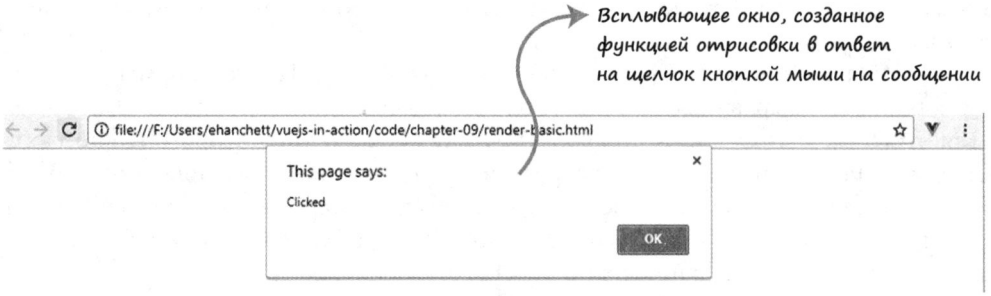


Рис. 9.6. При щелчке на сообщении функция отрисовки выводит диалоговое окно

9.3.2. Пример использования JSX

Подобно шаблонам, язык JSX позволяет создавать HTML-разметку, обладая при этом всей мощью JavaScript. Продемонстрируем это на примере приложения, основанного на Vue-CLI. Наша цель — воссоздать предыдущий пример с применением JSX. Мы должны принимать класс в виде свойства и отображать диалоговое окно при щелчке на сообщении.

Прежде всего убедитесь в том, что у вас установлен пакет Vue-CLI (инструкции — в приложении А). Откройте окно терминала и создайте приложение под названием jsx-example. Вам будет задано несколько вопросов, во всех случаях нажмите клавишу Enter (листинг 9.11).

Листинг 9.11. Консольная команда для создания проекта на основе jsx

```
$ vue init webpack jsx-example

? Project name jsx-example
? Project description A Vue.js project
? Author Erik Hanchett <erikhanchettblog@gmail.com>
? Vue build standalone
? Install vue-router? No
? Use ESLint to lint your code? No
? Setup unit tests No
? Setup e2e tests with Nightwatch? No

   vue-cli · Generated "jsx-example".

   To get started:

     cd jsx-example
     npm install
```

Далее перейдите в каталог jsx-example и выполните команду `npm install`, чтобы установить зависимости. Нужно также установить расширение Babel для JSX.

Если на этом этапе у вас возникнут какие-либо проблемы, пожалуйста, ознакомьтесь с официальной страницей на GitHub по адресу github.com/vuejs/babel-plugin-transform-vue-jsx. Пример, представленный далее, затрагивает лишь основные возможности JSX. Я настоятельно советую почитать официальную документацию на сайте GitHub, чтобы узнать обо всех особенностях этой технологии. Выполните команду `npm install` для всех рекомендуемых библиотек, как показано в листинге 9.12.

Листинг 9.12. Консольная команда для установки расширения

```
$ npm install\
  babel-plugin-syntax-jsx\
  babel-plugin-transform-vue-jsx\
  babel-helper-vue-jsx-merge-props\
  babel-preset-env\
  --save-dev
```

Теперь следует обновить файл `.babelrc`, который находится в корневой папке проекта. В нем перечислено множество предустановок и расширений. Добавьте предустановку `"env"` и расширение `"transform-vue-jsx"` (листинг 9.13).

Листинг 9.13. Обновление .babelrc: chapter-09/jsx-example/.babelrc

```
{
  "presets": [
    ["env", {
      "modules": false
    }],
    "stage-2",
    "env"
  ],
  "plugins": ["transform-runtime", "transform-vue-jsx"],
  "env": {
    "test": {
      "presets": ["env", "stage-2"]    }
  }
}
```

Добавляем env в список предустановок

Добавляем transform-vue-jsx в список расширений

Обеспечив поддержку JSX, приступим к написанию кода. Vue-CLI по умолчанию создает файл `HelloWorld.vue`. Возьмем его за основу, но внесем некоторые изменения. Откройте файл `src/App.vue` и обновите шаблон. Удалите узел `img` и вставьте вместо него компонент `HelloWorld` с двумя входными параметрами: первый, `header`, определяет уровень заголовка, а второй называется `name`. В листинге 9.10 вместо именованных параметров использовались слоты. В этом примере все немного иначе: имя будет передаваться в виде входного параметра. Тем не менее результат будет тем же. Приведите файл `src/App.vue` к следующему виду (листинг 9.14).

Мы используем библиотеку Bootstrap, поэтому должны ее где-то подключить. Найдите файл `index.html` в корне проекта и добавьте элемент `link` со ссылкой на Bootstrap CDN, как показано в листинге 9.15.

Листинг 9.14. Обновление файла App.vue: chapter-09/jsx-example/src/App.vue

```
<template>
  <div id="app">
    <HelloWorld header="1" name="Erik"></HelloWorld>
  </div>
</template>

<script>
import HelloWorld from './components/HelloWorld'

export default {
  name: 'app',
  components: {
    HelloWorld
  }
}
</script>

<style>
</style>
```

Компонент HelloWorld с двумя параметрами, name и header

Листинг 9.15. Обновление файла index.html: chapter-09/jsx-example/index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,initial-scale=1.0">
    <title>jsx-example</title>
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/
    bootstrap/3.3.7/css/bootstrap.min.css">
  </head>
  <body>
    <div id="app"></div>
  </body>
</html>
```

Подключаем Bootstrap

Откройте файл `src/components/HelloWorld.vue` и удалите элемент `template` вверху. Он больше не понадобится, так как мы будем использовать JSX. Сначала подготовим функцию данных внутри `export default`. Она возвращает приветствие и сообщение. Свойство `msg` станет основой для узла `header`, в нем мы объединим текст, который будет выводиться на экран в виде шаблонных литералов в формате ES6.

Добавьте также объект `methods` с методом `pressed`, который станет выводить диалоговое окно с текстом `Clicked`. Напоследок создайте массив `props` для параметров `header` и `name`.

Функция `render` похожа на ту, которую мы использовали в нашем первом примере. Переименуем `createElement` в `h`, чтобы соблюдалось соответствие соглашению об именовании. Затем вернем нужный JSX-код.

Как видно в листинге 9.16, функция `render` возвращает несколько тегов. Первый, `div`, вмещает в себя весь JSX-код. Внутри имеется еще один элемент `div` с классом `text-center`. Добавим к нему обработчик события `click`, который присвоен свойству `this.pressed`. Обычно для привязывания данных в Vue.js применяется интерполяция текста с синтаксисом Mustache (двойные фигурные скобки). В JSX используются одинарные фигурные скобки.

Напоследок нужно добавить специальное свойство `domPropsInnerHTML`, которое предоставляется расширением `babel-plugin-transform-vue`. Если вы знакомы с React, то увидите, что это свойство аналогично параметру `dangerouslySetInnerHTML`. Оно принимает значение `this.msg` и интерпретирует его как HTML. Имейте в виду: преобразование пользовательского ввода в HTML-код чревато *XSS-атаками* (Cross-Site Scripting – межсайтовый скриптинг), поэтому будьте осторожны, применяя `domPropsInnerHTML`. Скопируйте текст следующего листинга, если еще этого не сделали, и сохраните его в своем проекте.

Листинг 9.16. Обновление HelloWorld.vue: chapter-09\jsx-example\src\components\HelloWorld.vue

```
<script>
export default {
  name: 'HelloWorld',
  render(h) {
    return (
      <div>
        <div class="text-center"
        on-click={this.pressed}
        domPropsInnerHTML={this.msg}></div>
      </div>
    )
  },
  data () {
    return {
      welcome: 'Hello World ',
      msg: `<h${this.header}>Hello World ${this.name}
            </h${this.header}>`,
    }
  },
  methods: {
    pressed() {
      alert('Clicked')
    }

  },
  props:['header', 'name']
}
</script>

<style scoped>
</style>
```

Функция для отрисовки JSX

Элемент div, вмещающий в себя JSX-код

Элемент div с атрибутом class

Обработчик события click, создающий диалоговое окно

domPropsInnerHTML добавляет this.msg в div

'Hello World' с добавлением this.name

Метод, создающий диалоговое окно

Два входных параметра, header и name

Сохраните этот файл и запустите сервер с помощью команды `npm run dev` (или `npm run serve`, если используете Vue-CLI 3.0). Затем откройте веб-страницу http://localhost:8080. Вы должны увидеть сообщение Hello World Erik. Попробуйте поменять несколько значений, которые передаются компоненту `HelloWorld` внутри `App.vue`. С помощью параметра `header` можно выбрать уровень отображаемого заголовка.

Упражнение

Используя знания, приобретенные в этой главе, ответьте на следующие вопросы: что такое примесь и в каких ситуациях ее нужно применять?

Ответ ищите в приложении Б.

Резюме

- Примеси позволяют использовать общие фрагменты кода в разных компонентах.
- С помощью пользовательских директив можно модифицировать поведение отдельных элементов.
- Используйте модификаторы, значения и аргументы для передачи информации в пользовательские директивы, делая элементы страницы динамическими.
- Функция отрисовки предоставляет всю мощь JavaScript внутри HTML-разметки.
- Вместо функции отрисовки в Vue.js можно применять язык JSX, который тоже обладает всей мощью JavaScript.

Часть III
Моделирование данных, работа с API и тестирование

Со временем ваше Vue-приложение будет становиться все объемнее и сложнее, и в какой-то момент придется задуматься об эффективном хранении данных. К счастью, нам доступно превосходное решение под названием Vuex, которое упрощает этот процесс. Мы изучим его во всех подробностях в главе 10.

В главе 11 пойдем еще дальше и научимся взаимодействовать с сервером. Обсудим такие темы, как общение с серверными системами и обработка данных. Вы познакомитесь с серверной отрисовкой — новой технологией, которая сделает приложения более быстрыми.

Глава 12 посвящена тестированию. Профессиональный разработчик должен уметь тестировать свой код. Это помогает устранить ошибки и сделать приложение более стабильным. Мы затронем также методики *DevOps* (от Development Operations), и я покажу, как они могут помочь в развертывании приложения и обеспечении его корректной работы.

10 Vuex

В этой главе

- Что такое состояние.
- Использование геттеров.
- Реализация мутаций.
- Добавление действий.
- Работа со вспомогательными методами Vuex.
- Модули и настройка проекта.

В главе 9 мы рассмотрели способы расширения Vue.js и многократное использование функциональности без повторения кода. Здесь вы узнаете, как хранить данные внутри приложения и сделать их доступными для разных компонентов. Один из предпочтительных методов разделения данных — с помощью библиотеки *Vuex*. Это библиотека для управления состоянием, которая создает централизованное хранилище, доступное для любых компонентов приложения.

Сначала мы поговорим о том, в каких ситуациях следует применять Vuex. Для одних приложений эта технология подходит, а для других — не очень. Затем рассмотрим состояние и то, как его можно централизовать. После этого вы познакомитесь с геттерами, мутациями и действиями, эти три концепции позволяют отслеживать состояние приложения. Дальше будут представлены вспомогательные методы Vuex, которые помогут частично избавиться от повторяющегося кода. В конце предложена структура каталогов, которая позволит извлечь максимальную выгоду от использования Vuex в крупных приложениях.

10.1. Для чего нам Vuex

Библиотека Vuex управляет *состоянием*. Она хранит его централизовано, что облегчает обращение к нему любых компонентов. Состояние — это информация или данные, которые поддерживают приложение. Это важно, поскольку нам необходим надежный и понятный механизм для работы с этой информацией.

Если у вас уже есть опыт работы с другими фреймворками для создания одностраничных приложений, некоторые из этих концепций могут показаться знакомыми. Например, React использует похожую систему управления состоянием под названием Redux. Vuex и Redux созданы под влиянием проекта Flux. Это архитектура, предложенная компанией Facebook, которая призвана упростить построение клиентских веб-приложений. Она способствует движению данных в одном направлении: от действий к диспетчеру, затем к хранилищу и в конце — к представлению. Такой подход позволяет отделить состояние от остальной части приложения и поощряет синхронные обновления. Больше о Flux можно узнать из официальной документации на странице facebook.github.io/flux/docs/overview.html.

Vuex работает по тому же принципу, помогая изменять состояние предсказуемо и синхронно. Разработчикам не нужно беспокоиться о последствиях обновления состояния синхронными и асинхронными функциями. Представьте, что мы взаимодействуем с серверным API, который возвращает данные в формате JSON. Что произойдет, если в тот же момент эти данные будут модифицированы сторонней библиотекой? Мы не хотим получить непредсказуемый результат. Vuex помогает избежать подобных ситуаций, исключая любые асинхронные изменения.

Вам, наверное, интересно, зачем нам вообще нужна библиотека Vuex. В конце концов, Vue.js позволяет передавать информацию в компоненты. Как вы знаете из предыдущих глав, для этого предназначены входные параметры и пользовательские события. Мы даже могли бы создать собственную шину событий для передачи данных и межкомпонентного взаимодействия. Пример такого механизма представлен на рис. 10.1.

Это было бы уместно для небольших приложений с горсткой компонентов. В этом случае нужно передавать информацию лишь нескольким адресатам. Но что, если приложение более крупное, сложное и многоуровневое? Понятно, что в большом проекте не так-то просто уследить за всеми функциями обратного вызова, входными параметрами и событиями.

Как раз для таких ситуаций и создана библиотека Vuex. Она позволяет организовать работу с состоянием в виде централизованного хранилища. Представьте сценарий, в котором стоит задуматься о применении Vuex. Например, мы работаем над блогом со статьями и комментариями, которые можно создавать, редактировать и удалять. При этом у нас есть панель администрирования, позволяющая блокировать и добавлять пользователей.

Посмотрим, как это реализуется с помощью Vuex. На рис. 10.2 видно, что компонент `EditBio` является дочерним по отношению к панели администратора. Ему нужен доступ к информации о пользователе, чтобы он мог ее обновить. Работая с Vuex, мы можем обратиться к центральному хранилищу, изменить данные и сохранить изменения непосредственно из компонента `EditBio`. Это намного лучше, чем передавать информацию из корневого экземпляра Vue.js в компонент `Admin`, а затем в `EditBio` с помощью входных параметров. Нам было бы сложно уследить за данными, размещенными в разных местах.

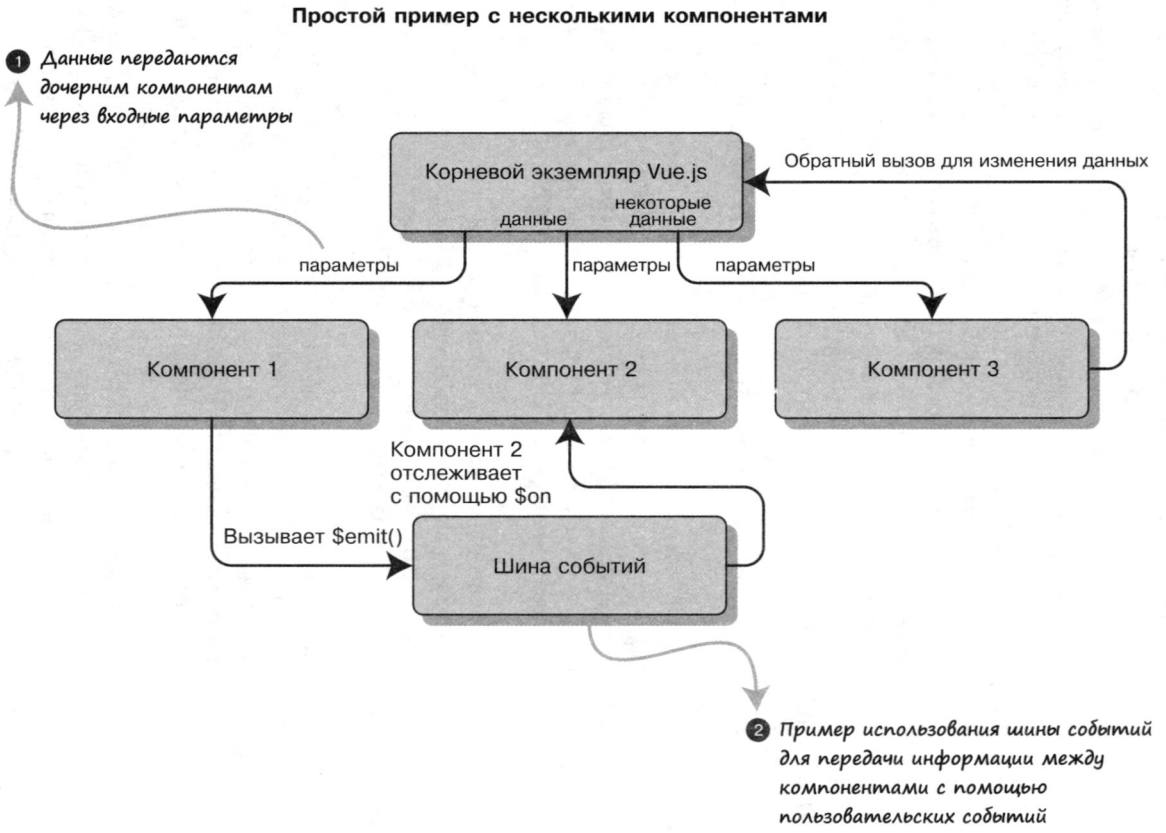


Рис. 10.1. Простой пример использования входных параметров и шины событий

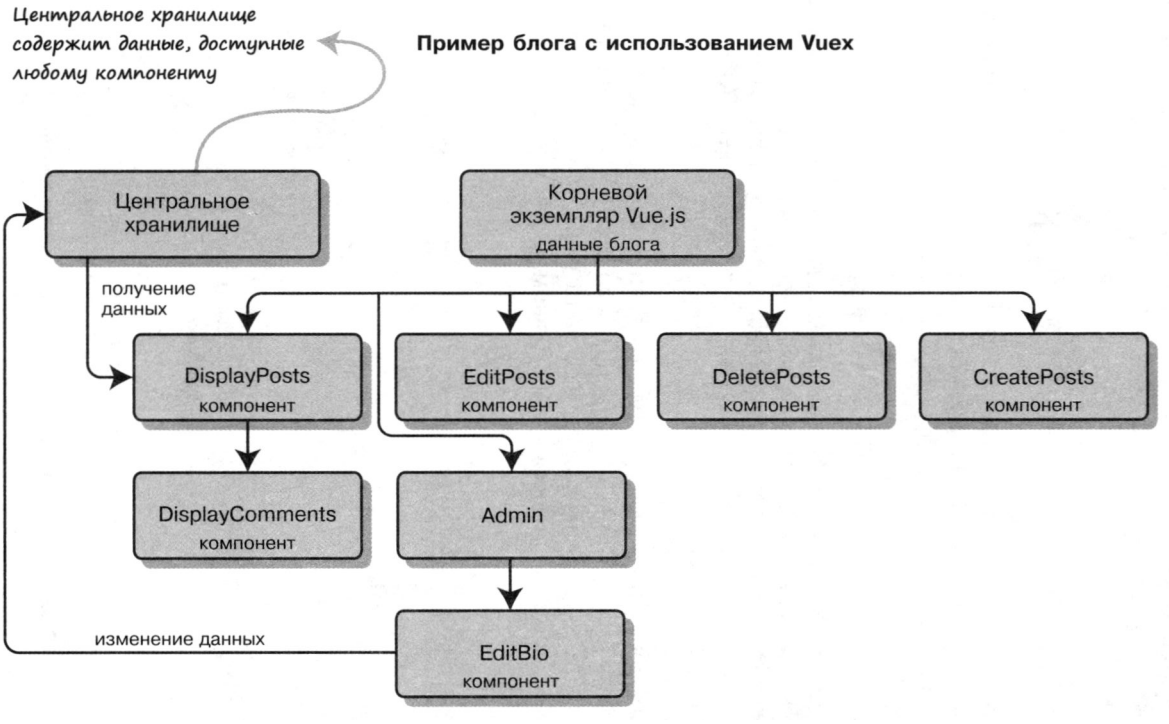


Рис. 10.2. Использование центрального хранилища в Vuex

Тем не менее за применение Vuex приходится платить свою цену в виде дополнительного шаблонного кода и усложнения структуры приложения. Как уже упоминалось, эту библиотеку лучше не задействовать в простых проектах, состоящих из нескольких компонентов. По-настоящему ее потенциал проявляется в крупных приложениях с более сложным состоянием.

10.2. Состояние и мутации в Vuex

Vuex хранит состояние всего приложения в едином объекте, который еще называют *единым источником истины*. Как можно догадаться из названия, все данные собраны в одном месте и не дублируются на других участках кода.

ПОДСКАЗКА

Стоит отметить, что мы не обязаны хранить все свои данные в Vuex. Отдельные компоненты могут иметь собственное локальное состояние. В определенных ситуациях это оказывается предпочтительно. Например, у вашего компонента есть переменная, которая используется только внутри него. Она должна оставаться локальной.

Рассмотрим простой пример работы с состоянием в Vuex. Весь наш код будет размещен в одном файле. Позже вы узнаете, как добавить Vuex в проект с помощью Vue-CLI. Откройте текстовый редактор и создайте файл `vuex-state.html`. Мы будем выводить сообщение, находящееся в центральном хранилище, и счетчик. Итог показан на рис. 10.3.

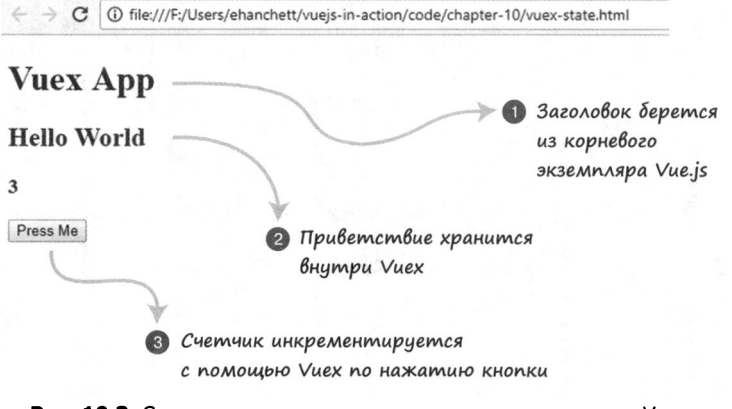


Рис. 10.3. Создание простого приложения с использованием Vuex

Сначала добавим теги `script` со ссылками на Vue и Vuex. Затем создадим HTML-разметку. Мы будем задействовать теги `H1`, `H2`, `H3` и `button`. Тег `h1` отображает заголовок с локальной переменной, объявленной в экземпляре Vue.js. Сообщения `welcome` и `counter` выполняются в виде вычисляемых свойств на основе хранилища Vuex.

Элемент `button` инициирует действие `increment`. Скопируйте код из листинга 10.1 в файл `vuex-state.html`.

Листинг 10.1. HTML-разметка Vuex-приложения: chapter-10/vuex-html.html

```
<!DOCTYPE html>
<html>
<head>
<script src="https://unpkg.com/vue"></script>
<script src="https://unpkg.com/vuex"></script>
</head>
<body>
  <div id="app">
      <h1>{{header}}</h1>
      <h2>{{welcome}}</h2>
      <h3>{{counter}}</h3>
      <button @click="increment">Press Me</button>
  </div>
```

Элемент script для Vue

Элемент script для Vuex

Переменная header

Вычисляемое свойство welcome

Вычисляемое свойство counter

Нажатие кнопки привязано к действию increment

Закончив с HTML-разметкой, приступим к созданию хранилища Vuex. В нем будут находиться все данные приложения, включая свойства `msg` и `count`.

Для обновления состояния используем *мутации*. Это нечто похожее на сеттеры из других языков программирования. *Сеттер* устанавливает значение, мутация обновляет состояние программы. В Vuex мутации должны быть синхронными. В нашем примере счетчик инкрементируется только по нажатию кнопки, поэтому не нужно беспокоиться об асинхронном коде (позже мы рассмотрим действия, которые помогают решить проблемы с асинхронностью).

Создадим внутри объекта `mutations` функцию `increment`, инкрементирующую состояние. Возьмите код, представленный в листинге 10.2, и вставьте его внизу файла `vuex-state.html`.

Листинг 10.2. Добавление состояния и мутаций Vuex: chapter-10/vuex-state-mut.html

```
<script>
  const store = new Vuex.Store({
      state: {
        msg: 'Hello World',
        count: 0
      },
      mutations: {
        increment(state) {
            state.count++;
        }
      }
  });
```

Vuex.Store хранит информацию о состоянии

Мутация, инкрементирующая состояние

Итак, мы подготовили HTML-разметку и хранилище Vuex. Теперь можно добавить логику, которая их свяжет. Мы хотим, чтобы шаблон отображал значения `msg` и `counter`, которые являются частью состояния Vuex. Кроме того, `counter` нужно обновлять.

Создайте экземпляр Vue.js с новой функцией данных, которая будет возвращать локальное свойство `header` с текстом `Vuex App`. В разделе `computed` добавим вычисляемые свойства `welcome` и `counter`. Первое будет возвращать `store.state.msg`, а второе — `store.state.count`.

Напоследок необходимо добавить метод под названием `increment`. В хранилище Vuex объявлена мутация, но мы не можем использовать ее напрямую для обновления состояния. Для этого предусмотрена специальная функция `commit`. Она сообщает Vuex о необходимости обновить хранилище и тем самым сохраняет изменения. Выражение `store.commit('increment')` выполняет мутацию. Вставьте следующий фрагмент (листинг 10.3) сразу за кодом, созданным в листинге 10.2.

Листинг 10.3. Добавление экземпляра Vue.js: chapter-10/vuex-instance.html

```
new Vue({
    el: '#app',
    data() {
      return {
        header: 'Vuex App'
      }
    },
    computed: {
      welcome() {
        return store.state.msg
      },
      counter() {
        return store.state.count;
      }
    },
    methods: {
      increment() {
        store.commit('increment')
      }
    }
});
</script>
</body>
</html>
```

Свойство header, отображающее сообщение

Вычисляемое свойство возвращает состояние msg

Вычисляемое свойство возвращает состояние counter

Метод increment выполняет одноименную мутацию

Это уже полноценное рабочее приложение на основе Vuex! Попробуйте нажимать кнопку — при каждом нажатии счетчик должен увеличиваться на 1.

Обновим код так, чтобы нажатие кнопки инкрементировало счетчик на 10. Если внимательно присмотреться к мутации `increment`, можно заметить, что она принимает лишь один аргумент, `state`. Передадим еще один — назовем его `payload`. Он будет передаваться методом `increment`, который создан в корневом экземпляре Vue.js.

Скопируйте содержимое `vuex-state.html` в новый файл `vuex-state-pass.html`. На примере этого приложения покажем, как передаются аргументы.

Как видно в листинге 10.4, нужно обновить только объект `mutations` и метод `increment`. Добавьте к мутации `increment` еще один аргумент под названием `payload`. Это значение, на которое будет увеличиваться `state.count`. Найдите внутри метода

`increment` вызов `store.commit` и укажите `10` в качестве второго аргумента. Обновите файл `vuex-state.html`, как показано далее.

Листинг 10.4. Передача payload в мутацию: chapter-10/vuex-state-pass-1.html

```
...
mutations: {
  increment(state,payload) {
      state.count += payload;
  }
}
...
methods: {
  increment() {
    store.commit('increment', 10)
  }
...
```

Мутация increment принимает payload и прибавляет его к count

Теперь метод method передает мутации значение 10

Сохраните файл `vuex-state-pass.html` и откройте его в браузере. Теперь при нажатии кнопки счетчик должен увеличиваться на 10, а не на 1. Если что-то пошло не так, проверьте консоль браузера и убедитесь в том, что не допустили никаких опечаток.

10.3. Геттеры и действия

В предыдущем примере мы обращались к хранилищу напрямую из вычисляемых свойств. Но что, если бы у нас было несколько компонентов, которым нужен тот же доступ? Допустим, что мы хотим выводить приветственное сообщение прописными буквами. В этом случае нам помогут геттеры.

Геттеры — часть Vuex. Они позволяют реализовать унифицированный доступ к состоянию во всех компонентах. Возьмем пример из раздела 10.2 и вместо прямого доступа к хранилищу через вычисляемые свойства воспользуемся геттерами. Кроме того, сделаем так, чтобы геттер для свойства `msg` переводил все его буквы в верхний регистр.

Скопируйте содержимое файла `vuex-state-pass.html` в `vuex-state-getter-action.html`. Чтобы упростить задачу, оставим HTML-код неизменным. В конце должно получиться нечто похожее на рис. 10.4.

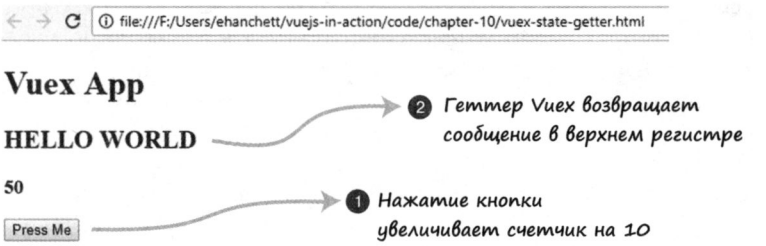


Рис. 10.4. Приложение Hello World с применением сеттеров

Как видите, сообщение `Hello World` теперь выводится прописью. Нажатие кнопки Press Me инкрементирует счетчик так же, как и в предыдущем примере.

Найдите внутри нового файла `vuex-state-getter-action.html` конструкцию `Vuex.Store` сразу под тегом `<script>`. Добавьте после `mutations` новый объект под названием `getters`. Создайте внутри этого объекта методы `msg` и `count`, как показано в листинге 10.5. Оба метода принимают один и тот же аргумент `state`.

Геттер `msg` будет возвращать `state.msg.toUppercase()`. Благодаря этому сообщение всегда выводится в верхнем регистре. В геттере `count` мы вернем `state.count`. После добавления геттеров снизу от мутаций файл `vuex-state-getter-action.html` должен выглядеть следующим образом.

Листинг 10.5. Добавление новых геттеров: chapter-10/vuex-state-getter-action1.html

```
...
mutations: {
  increment(state,payload) {
      state.count += payload;
  }
},
getters: {
    msg(state) {
      return state.msg.toUpperCase();
    },
    count(state) {
      return state.count;
    }
},
...
```

Новый объект getters объявляет геттеры для Vuex

Геттер msg возвращает msg в верхнем регистре

Геттер count

Действия — еще одна неотъемлемая часть Vuex. Ранее я упоминал о том, что мутации должны быть синхронными. Но что, если мы работаем с асинхронным кодом? Как добиться того, чтобы асинхронные вызовы могли изменять состояние? В этом нам помогут действия Vuex.

Представьте, что приложение обращается к серверу и ожидает ответа. Это пример асинхронного действия. К сожалению, мутации асинхронны, поэтому мы не можем их здесь использовать. Вместо этого добавим асинхронную операцию на основе действия Vuex.

Для создания задержки задействуем функцию `setTimeout`. Откройте файл `vuex-state-getter-action.html` и добавьте в него объект `actions` сразу после `getters`. Внутри этого объекта разместим действие `increment`, которое принимает аргументы `context` и `payload`. С помощью `context` будем сохранять изменения. Поместим операцию `context.commit` внутрь `setTimeout`. Таким образом мы симулируем задержку на сервере. Можем также передать в `context.commit` аргумент `payload`, который затем попадет в мутацию. Обновите код на основе листинга 10.6.

После обновления `Vuex.Store` можно переходить к корневому экземпляру Vue.js. Вычисляемое свойство будет обращаться к хранилищу не напрямую, как раньше, а с помощью геттеров. Мы также модифицируем метод `increment`. Для доступа к новому свойству Vuex, которое мы создали ранее, он будет использовать вызов `store.dispatch('increment', 10)`.

Листинг 10.6. Добавление действий: chapter-10/vuex-state-getter-action2.html

```
...
},
actions: {
  increment(context, payload) {
    setTimeout(function(){
      context.commit('increment',payload);
    },2000);
  }
}
...
```

Объект actions содержит асинхронные и синхронные действия

Функция increment принимает context и payload

Запускает мутацию increment и передает ей payload

Первым аргументом вызова `dispatch` служит название действия, а во втором аргументе всегда содержатся дополнительные данные, которые этому действию передаются.

СОВЕТ

Дополнительные данные могут представлять собой обычную переменную или даже объект.

Обновите экземпляр Vue.js в файле `vuex-state-getter-action.html` так, как показано в листинге 10.7.

Листинг 10.7. Обновление экземпляра Vue.js: chapter-10/vuex-state-getter-action3.html

```
...
new Vue({
    el: '#app',
    data() {
      return {
        header: 'Vuex App'
      }
    },
    computed: {
      welcome() {
        return store.getters.msg;
      },
      counter() {
        return store.getters.count;
      }
    },
    methods: {
      increment() {
        store.dispatch('increment', 10);
      }
    }
});
...
```

Вычисляемое свойство welcome возвращает геттер msg

Вычисляемое свойство counter возвращает геттер count

Метод, запускающий действие increment

Загрузите приложение и несколько раз нажмите кнопку. Вы должны заметить задержку, но счетчик будет увеличиваться на 10 после каждого нажатия.

10.4. Использование Vuex в проекте зоомагазина с помощью Vue-CLI

Вернемся к проекту зоомагазина, над которым работали. Если вы не забыли, мы остановились на добавлении анимации и переходов. Теперь интегрируем библиотеку Vuex, с которой познакомились ранее.

Перенесем данные о товарах в хранилище. Как вы помните по предыдущим главам, данные инициализировались в хуке `created` внутри компонента `Main`. Теперь этот хук должен генерировать новое событие, которое инициализирует хранилище Vuex. Мы также добавим вычисляемое свойство `products`, которое извлекает товары с помощью геттера (его создадим позже). Итоговый результат будет выглядеть как на рис. 10.5.

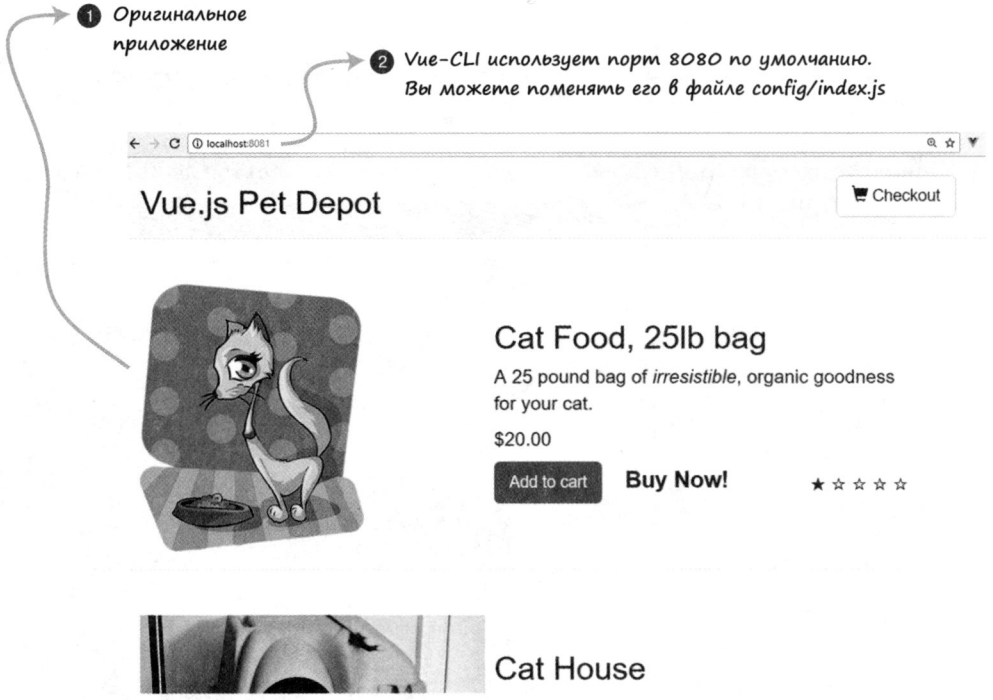


Рис. 10.5. Зоомагазин после перехода на Vuex

10.4.1. Установка Vuex с помощью Vue-CLI

Для начала установим Vuex! Это простой процесс. Подготовьте последнюю версию зоомагазина, над которой мы работали в главе 8. Вы можете также загрузить весь код для этой главы на GitHub по адресу github.com/ErikCH/VuejsInActionCode.

Откройте окно терминала и перейдите в корневой каталог проекта. Чтобы установить последнюю версию Vuex, запустите следующую команду:

```
$ npm install vuex
```

и сохраните запись о ней в файл `package.json` зоомагазина.

Теперь нужно добавить хранилище в файл `main.js`, который находится в папке src. Самого хранилища пока не существует, но мы все равно его импортируем. Обычно оно находится в файле `src/store/store.js`, но вы можете выбрать другой путь — у всех разработчиков свои предпочтения. Остановимся на общепринятом варианте. Позже в этой главе мы обсудим альтернативную структуру каталогов с применением модулей.

Нужно добавить хранилище в корневой экземпляр Vue.js снизу от маршрутизатора, как показано в листинге 10.8. Между прочим, мы используем стандарт ES6, поэтому `store: store` можно сократить до `store`.

Листинг 10.8. Обновление файла main.js: chapter-10/petstore/src/main.js

```
// The Vue build version to load with the `import` command
// (runtime-only or standalone) has been set in webpack.base.conf with an alias
import Vue from 'vue'
import App from './App'
import router from './router'
require('./assets/app.css')
import { store } from './store/store';

Vue.config.productionTip = false

/* eslint-disable no-new */
new Vue({
  el: '#app',
  router,
  store,
  template: '<App/>',
  components: { App }
})
```

Импортируем хранилище в файл main.js

Добавляем его в экземпляр Vue.js

После подключения хранилища к корневому экземпляру можем обращаться к нему с любого участка приложения. Создайте файл `src/store/store.js`. В нем мы разместим хранилище Vuex с информацией о товарах, предлагаемых зоомагазином. Вверху добавьте два выражения `import`, по одному для Vue и Vuex. Затем укажите `Vue.use(Vuex)`, чтобы соединить все вместе.

Мы импортировали хранилище в файле `main.js` из `./store/store`. Теперь нужно экспортировать объект `store` внутри `store.js`. Как видно в листинге 10.9, мы экспортируем значение `const store`, равное `Vuex.Store`.

Сначала добавим объекты с состоянием и мутациями. Состояние будет содержать пустой объект под названием `products`. Вскоре мы наполним его с помощью метода `initStore`. Мутация называется `SET_STORE`, она станет присваивать переданные

товары свойству `state.products`. Вставьте код из следующего листинга в файл `src/store/store.js`, который мы только что создали.

Листинг 10.9. Создание файла store.js: chapter-10/store-part1.html

```
import Vue from 'vue';
import Vuex from 'vuex';

Vue.use(Vuex);

export const store = new Vuex.Store({
  state: {
    products: {}
  },
  mutations: {
    'SET_STORE' (state, products) {
      state.products = products;
    }
  },
  ...
```

Подключаем Vuex к Vue

Экспортируем Vuex.Store для дальнейшего использования в файле main.js

Объект state с товарами

Объект mutations с функцией для инициализации хранилища

Нам нужно создать в хранилище действие и геттер. Геттер будет возвращать объект `products`. С действием все немного сложнее. Следует перенести хук `created`, который считывает файл `static/products.json` с помощью Axios, в объект `actions` внутри Vuex.

Ранее я упоминал, что мутации должны быть синхронными и что только действия внутри Vuex могут принимать асинхронный код. Чтобы обойти это ограничение, поместим код Axios в действие Vuex.

Создайте объект `actions` в файле `store.js` и добавьте в него метод `initStore`. Скопируйте в этот метод содержимое хука `created` из файла `components/Main.vue`. Вместо присваивания `response.data.products` объекту `products` мы вызовем мутацию с помощью функции `commit`. Передадим `response.data.products` в качестве аргумента для `SET_STORE`. Итоговый код должен выглядеть следующим образом (листинг 10.10).

Листинг 10.10. Добавление действий и геттеров в store.js: chapter-10/store-part2.html

```
...
actions: {
  initStore: ({commit}) => {
    axios.get('static/products.json')
    .then((response) =>{
      console.log(response.data.products);
      commit('SET_STORE', response.data.products )
    });
  }
},
getters: {
  products: state => state.products
}
});
```

Объект actions с асинхронным кодом

Действие initStore вызывает мутацию

Геттер products возвращает список товаров

Мы почти закончили, осталось только обновить файл `Main.vue` и перенести товары из локального объекта `products` в хранилище Vuex. Откройте файл `src/components/Main.vue` и найдите функцию данных. Удалите строчку `products: {}`. Мы будем обращаться к товарам из вычисляемого свойства, которое возвращает хранилище.

Найдите вычисляемые свойства `cartItemCount` и `sortedProducts` внутри `Main.vue`, они должны идти сразу после раздела `methods`. Добавьте свойство `products` и сделайте так, чтобы оно возвращало одноименный геттер.

Мы подключили хранилище к корневому экземпляру Vue.js в файле `main.js`, поэтому импортировать его больше не нужно. Кроме того, при использовании Vue-CLI хранилище всегда доступно в виде `this.$store`. Не забудьте знак `$`, иначе получится ошибка. Добавьте вычисляемое свойство `products`, как показано в листинге 10.11.

Листинг 10.11. Добавление вычисляемого свойства products: chapter-10/computed-petstore.html

```
computed: {
  products() {
    return this.$store.getters.products;

  },
...
```

Вычисляемое свойство для файла Main.vue

Вычисляемое свойство products возвращает одноименный геттер

Найдите хук `created`, в котором инициализируется объект `products`, и удалите его содержимое. Вместо этого вставьте вызов действия `initStore`, которое мы создали ранее в хранилище Vuex. Чтобы вызвать действие, используйте функцию `dispatch`, как это сделано в предыдущем примере. В листинге 10.12 показано, как должен выглядеть хук `created` после обновления файла `Main.vue`.

Листинг 10.12. Обновление хука created: chapter-10/created-petstore.html

```
...
},
created: function() {
  this.$store.dispatch('initStore');
}
...
```

Вызывает код для инициализации хранилища Vuex

Этого должно быть достаточно. Выполните в терминале команду `npm run dev`, и на экране должно появиться окно с приложением зоомагазина. Попробуйте положить товары в корзину и убедитесь в том, что все работает как следует. Если что-то пошло не так, поищите ошибки в консоли. В файле `src/store/store.js` вместо `Vuex.store` можно случайно набрать `Vuex.Store`. Помните об этом!

10.5. Вспомогательные методы Vuex

Vuex предоставляет удобные вспомогательные методы, которые позволяют сделать код более лаконичным и избавиться от добавления одних и тех же геттеров, сеттеров, мутаций и действий. Полный список вспомогательных методов Vuex есть в официальном руководстве по адресу vuex.vuejs.org/guide/core-concepts.html. Посмотрим, как они работают.

Основной вспомогательный метод, о котором вы должны знать, называется `mapGetters`. Он применяется для добавления всех имеющихся геттеров в раздел `computed` и не требует перечисления каждого из них. Но прежде, чем использовать, нужно импортировать его внутрь компонента. Еще раз вернемся к зоомагазину и добавим метод `mapGetters`.

Откройте файл `src/components/Main.vue` и отыщите тег `script`. Где-то внутри этого тега должен импортироваться компонент `Header`. Подключите `mapGetters` сразу после этого импорта, как показано в листинге 10.13.

Листинг 10.13. Добавление mapGetters: chapter-10/map-getter.html

```
...
<script>
import MyHeader from './Header.vue';
import {mapGetters} from 'vuex';
export default {
...
```

Импорт mapGetters из Vuex

Теперь нужно обновить вычисляемое свойство. Найдите в разделе `computed` функцию `products` и вставьте вместо нее объект `mapGetters`.

`mapGetters` — уникальный объект, для его правильной работы необходимо использовать оператор `spread` из состава ES6 — он расширяет выражение в ситуации, когда ожидается получение каких-либо аргументов (ноль и больше). Подробнее об этом синтаксисе можно почитать в документации MDN по адресу https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Spread_syntax.

`mapGetters` сделает так, что все геттеры будут добавлены в виде вычисляемых свойств. Это куда более простой и элегантный способ по сравнению с написанием отдельного вычисляемого свойства для каждого геттера. Все геттеры перечислены в массиве `mapGetters`. Добавьте этот вспомогательный метод в файл `Main.vue` (листинг 10.14).

Листинг 10.14. Добавление mapGetters в раздел computed: chapter-10/map-getter2.html

```
...
},
computed: {
  ...mapGetters([
      'products'
  ]),
  cartItemCount() {
...
```

Вспомогательный массив mapGetters

Список геттеров

После запуска команды `npm run dev` зоомагазин должен работать как прежде. Вспомогательный метод `mapGetters` пока выглядит не слишком полезным, но чем больше геттеров мы добавим, тем больше времени он сэкономит.

Существует еще три вспомогательных метода, о которых вам следует знать: `mapState`, `mapMutations` и `mapActions`. Все они работают схожим образом, уменьшая объем шаблонного кода, который приходится писать вручную.

Представьте, что в вашем хранилище находится несколько фрагментов данных и доступ к состоянию осуществляется напрямую из компонента, без использования

каких-либо геттеров. В этом случае вы можете применить метод `mapState` внутри раздела `computed` (листинг 10.15).

Листинг 10.15. Пример использования mapState: chapter-10/map-state.html

```
import {mapState} from 'vuex'      ← Импорт mapState из Vuex
...
computed: {
  ...mapState([                    ← Объявление mapState и переменных
      'data1',                       с помощью оператора spread
      'data2',
      'data3'
  ])
}
...
```

Теперь представьте, что в компоненте требуется использовать несколько мутаций. Чтобы упростить этот процесс, задействуйте вспомогательный метод `mapMutations` (листинг 10.16), как это сделано с `mapState` и `mapGetters`. Далее `mut1` привязывает `this.mut1()` к `this.$store.commit('mut1')`.

Листинг 10.16. Пример использования mapMutations: chapter-10/map-mut.html

```
import {mapMutations} from 'vuex'  ← Импорт mapMutations
...                                  из Vuex в компонент
methods: {
  ...mapMutations([                ← mapMutations добавляет
      'mut1',                        перечисленные методы
      'mut2',
      'mut3'
  ])
}
...
```

Напоследок рассмотрим вспомогательный метод `mapActions`. Он позволяет добавить в приложение действия Vuex, избавляя от необходимости создавать методы с вызовом `dispatch` в каждом отдельном случае. Возвращаясь к предыдущему примеру, представим, что приложение содержит какие-нибудь асинхронные операции. Поскольку использование мутаций исключено, мы должны прибегнуть к действиям. После создания их в Vuex нужно получить к ним доступ в объекте `methods` компонента. Эту задачу можно решить с помощью `mapActions`. `act1` привяжет `this.act1()` к `this.$store.dispatch('act1')`, как показано в листинге 10.17.

Листинг 10.17. Пример использования mapActions: chapter-10/map-actions.html

```
import {mapActions} from 'vuex'    ← Импорт mapActions из Vuex
...
methods: {
  ...mapActions([                  ← mapActions добавляет
      'act1',                        методы act1, act2 и act3
      'act2',
      'act3'
  ])
}
...
```

С ростом приложения эти вспомогательные методы будут приносить все большую пользу, уменьшая объем кода, который необходимо писать. Имейте в виду, что вам необходимо продумывать имена свойств в своем хранилище, так как вспомогательные методы позволяют получить доступ к ним в компонентах.

10.6. Краткое введение в модули

В начале этой главы мы создали файл `store.js` в каталоге src/store. Для небольшого проекта такой подход вполне уместен. Но что, если мы имеем дело с куда более крупным приложением? Файл `store.js` быстро разрастется, и будет сложно уследить за всем, что в нем происходит.

Для решения этой проблемы Vuex предлагает концепцию *модулей*. Модули позволяют разделить хранилище на несколько частей меньшего размера. У каждого модуля есть свои состояния, мутации, действия и геттеры, можно даже делать их вложенными друг в друга.

Перепишем зоомагазин с использованием модулей. Файл `store.js` останется на месте, но рядом с ним следует создать папку modules и поместить туда файл `products.js`. Структура каталогов должна выглядеть так, как на рис. 10.6.

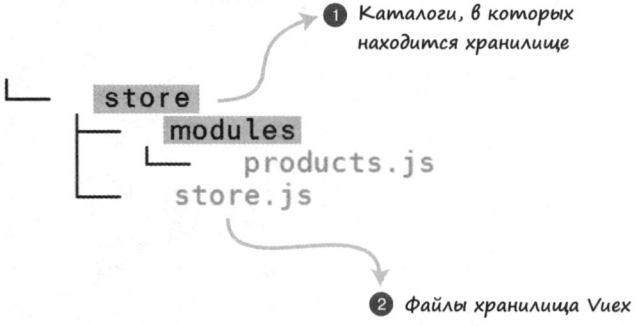


Рис. 10.6. Модульная структура каталогов

Внутри `products.js` нужно создать четыре объекта: `state`, `getters`, `actions` и `mutations`. Содержимое каждого из них следует скопировать из файла `store.js`.

Откройте файл `src/store/store.js` и начните копировать из него код. Когда закончите, файл `products.js` должен выглядеть следующим образом (листинг 10.18).

Теперь нужно экспортировать весь код, который мы добавили в файл `product.js`. Это позволит импортировать его в `store.js`. Внизу файла добавьте выражение `export default`. Это инструкция экспорта в формате ES6, которая позволит импортировать данный код из других файлов (листинг 10.19).

Файл `store.js` следует обновить. Добавим в него объект `modules`, внутри которого можно будет перечислить все новые модули. Не забудьте импортировать файл `modules/products`, который мы создали ранее.

Листинг 10.18. Добавление модулей products: chapter-10/products-mod.js

```
const state = {                          ← Хранит все состояния Vuex
    products: {}
};

const getters = {                        ← Хранит все геттеры Vuex
    products: state => state.products
};

const actions = {                        ← Хранит все действия Vuex
    initStore: ({commit}) => {
      axios.get('static/products.json')
      .then((response) =>{
        console.log(response.data.products);
        commit('SET_STORE', response.data.products )
      });
    }
};

const mutations = {                      ← Хранит все мутации Vuex
    'SET_STORE' (state, products) {
      state.products = products;
    }
};
```

Листинг 10.19. Экспорт кода: chapter-10/products-export.js

```
...
export default {          ← Экспорт всего кода в формате ES6
    state,
    getters,
    actions,
    mutations,
}
```

Наш пример содержит лишь один модуль, поэтому сразу добавим его в объект `modules`. Нужно также удалить все лишнее из `Vuex.Store`, как показано в листинге 10.20.

Листинг 10.20. Новый файл store.js: chapter-10/store-update.js

```
import Vue from 'vue';
import Vuex from 'vuex';
import products from './modules/products';   ← Импорт модуля products

Vue.use(Vuex);


export const store = new Vuex.Store({
  modules: {                                 ← Все модули перечислены в объекте modules
    products
  }

});
```

Импортировав модули, мы завершили процесс рефакторинга. После обновления страницы приложение должно работать точно так же, как и прежде.

Пространства имен в Vuex

В некоторых крупных проектах разбиение на модули способно вызвать определенные проблемы. По мере добавления новых модулей могут возникнуть конфликты с именами действий, геттеров, мутаций и свойств состояния. Например, вы можете случайно присвоить одно и то же имя двум геттерам в разных файлах. И, поскольку в Vuex все находится в общем глобальном пространстве имен, в консоли возникнет ошибка дублирования ключа.

Чтобы избежать этой проблемы, поместите каждый модуль в отдельное пространство имен. Для этого достаточно указать `namespaced: true` вверху `Vuex.store`. Подробнее о данной возможности читайте в официальной документации Vuex на странице vuex.vuejs.org/ru/guide/modules.html.

Упражнение

Используя знания, приобретенные в этой главе, перечислите несколько преимуществ Vuex по сравнению со стандартной передачей данных в приложениях Vue.js.

Ответ ищите в приложении Б.

Резюме

- Вы можете централизовать управление состоянием в своем приложении.
- Можете обращаться к хранилищу данных с любого участка приложения.
- Чтобы избежать проблем с синхронизацией хранилища, используйте мутации и действия Vuex.
- Сократить объем шаблонного кода можно с помощью вспомогательных методов Vuex.
- Для сохранения управляемости хранилища в крупных проектах можно использовать модули и пространства имен.

11 Взаимодействие с сервером

В этой главе

- Применение Nuxt.js для отрисовки на стороне сервера.
- Извлечение сторонних данных с помощью Axios.
- Использование VuexFire.
- Добавление аутентификации.

Мы обсудили Vuex и то, как правильное управление состоянием может помочь в разработке крупных проектов на Vue.js. Теперь пришло время поговорить о взаимодействии с сервером. В этой главе мы рассмотрим технологию *SSR* (*server-side rendering* — отрисовка на стороне сервера) и я покажу, как с ее помощью улучшить отзывчивость приложений. Продемонстрирую также процесс извлечения данных из сторонних API с использованием Axios. Вслед за этим вы познакомитесь с библиотекой VuexFire, которая помогает работать с Firebase — серверным решением для клиентских программ. А в конце я приведу пример добавления аутентификации в VuexFire.

Но прежде, чем продолжать, позвольте мне отметить тот факт, что работа с сервером в Vue.js может быть реализована множеством разных способов. Наряду с объектом `XMLHttpRequest` доступен целый ряд AJAX-библиотек. В прошлом официальным инструментом для работы с AJAX в Vue была библиотека vue-resource. Однако в 2016 году основатель Vue Эван Ю решил, что у этого проекта больше не будет статуса официально рекомендованного. Используйте любой инструмент на свой выбор.

Учитывая сказанное, я выбрал несколько популярных библиотек, которые помогут нам общаться с сервером тем или иным путем: Axios, Nuxt.js и VuexFire. Все они разные. Nuxt.js — это мощный фреймворк для создания приложений с серверной отрисовкой, тогда как Axios — HTTP-клиент, а VuexFire помогает работать с Firebase. Это три разных подхода к взаимодействию.

Цель данной главы заключается в том, чтобы помочь вам приобрести практические навыки использования трех библиотек/фреймворков. Мы рассмотрим примеры работы со всеми этими инструментами, но не станем слишком углубляться. Каждый из них заслуживает отдельной главы или даже целой книги, как в случае с Nuxt. Тем не менее данная глава послужит хорошим введением в эти технологии, а для более глубокого изучения будут предоставлены соответствующие ссылки.

11.1. Отрисовка на стороне сервера

Vue.js — это фреймворк для одностраничных приложений, который выполняет отрисовку на клиентской стороне. Логика и маршрутизация приложения пишутся на JavaScript. Соединившись с сервером, браузер загружает JavaScript-код, он отвечает за его отрисовку и выполнение Vue-приложения. В крупных проектах процесс загрузки и отрисовки может занимать значительное время. Это продемонстрировано на рис. 11.1.

Альтернативный подход обеспечивает технология SSR. При соединении с сервером Vue.js получает готовую HTML-страницу, которую можно сразу отобразить на экране. С точки зрения пользователя загрузка происходит быстрее. После этого сервер отправляет JavaScript-код, который загружается в фоновом режиме. Стоит отметить, что страница лишь выглядит готовой, но, пока Vue не закончит инициализацию, пользователь не способен с ней взаимодействовать (рис. 11.2).

Обычно технология SSR обеспечивает более приятный опыт взаимодействия, поскольку начальная загрузка происходит быстрее. В большинстве своем пользователи не любят ждать загрузки медленного приложения.

SSR также предоставляет уникальные преимущества с точки зрения *поисковой оптимизации* (search engine optimization, SEO). Термин SEO означает естественное (непроплаченное) повышение позиции веб-сайта в поисковой выдаче. Нам по-прежнему мало что известно о методах ранжирования Google и других поисковых систем, однако есть опасение, что поисковые роботы плохо справляются со страницами, сгенерированными на клиентской стороне. Это может повлиять на ранжирование. Избежать проблем помогает SSR.

Библиотека Vue.js не поддерживает SSR, но существуют отличные сторонние инструменты, которые облегчают интеграцию этой технологии в приложение. Две самые популярные — vue-server-renderer и Nuxt.js. Больше информации об SSR можно найти в официальном руководстве по адресу ssr.vuejs.org/ru/. Мы же рассмотрим пример создания SSR-приложения с помощью Nuxt.js.

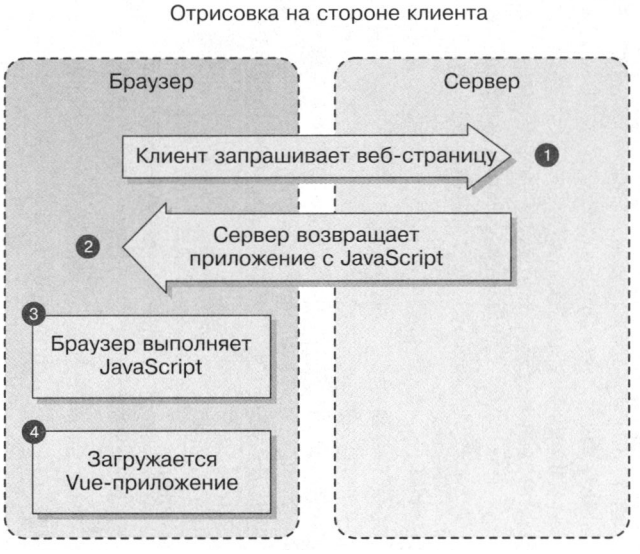


Рис. 11.1. Отрисовка на стороне клиента

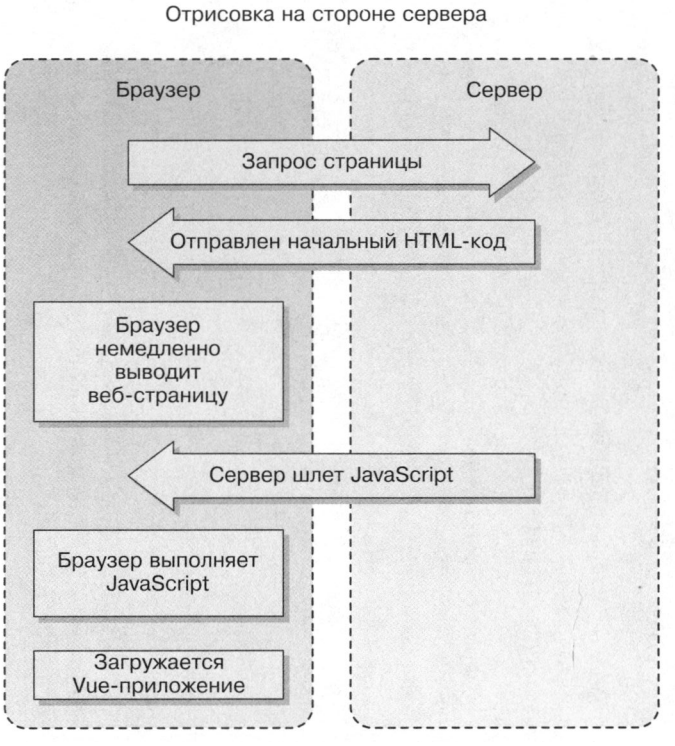


Рис. 11.2. Отрисовка на стороне сервера

11.2. Введение в Nuxt.js

Nuxt.js — это высокоуровневый фреймворк, основанный на экосистеме Vue, который помогает в создании SSR-приложений промышленного уровня и берет на себя все аспекты, связанные с их развертыванием.

Nuxt отвечает за отрисовку пользовательского интерфейса, упрощая бóльшую часть клиент-серверного взаимодействия. Этот фреймворк может выступать в качестве самостоятельного проекта или работать в связке с Node.js. Кроме того, он содержит встроенный генератор статических страниц, с помощью которого можно строить сайты на основе Vue.js.

При создании проекта с помощью Nuxt вы получаете пакеты Vue 2, vue-router, Vuex, vue-server-renderer и vue-meta. Чтобы связать их все воедино, используется Webpack. Это инструмент из разряда «все в одном», который позволяет сразу приступить к разработке.

ДОПОЛНИТЕЛЬНО

> Nuxt можно задействовать в связке с существующими приложениями для Node.js, но мы не станем рассматривать этот вариант конфигурации. Подробнее об этом читайте в официальной документации на странице ru.nuxtjs.org/guide/installation.

Nuxt предоставляет шаблон, с которого можно начать разработку. Загрузите его из официального репозитория на GitHub по адресу mng.bz/w0YV. Он доступен также при использовании Vue-CLI (если вы еще не установили эту утилиту, ознакомьтесь с инструкциями в приложении А).

Для работы с Nuxt вам понадобится Node.js версии 8 или выше. Если использовать более раннюю версию, при запуске проекта возникнут асинхронные ошибки.

ДОПОЛНИТЕЛЬНО

> Проект, представленный в этой главе, разработан с расчетом на Nuxt 1.0. На момент написания книги версия Nuxt 2.0 находилась на стадии beta-тестирования, но она тоже должна подойти. Если столкнетесь с какими-либо проблемами, сверьтесь с нашим официальным репозиторием на GitHub по адресу github.com/ErikCH/VuejsInActionCode. Этот код будет поддерживаться в дальнейшем.

Для создания проекта воспользуемся утилитой Vue-CLI. Введите в терминале следующую команду:

```
$ vue init nuxt-community/starter-template <project-name>
```

Таким образом, мы получим проект на основе начального шаблона. Далее нужно перейти в каталог проекта и установить зависимости с помощью следующих команд:

```
$ cd <project-name>
$ npm install
```

Чтобы запустить проект, выполните команду `npm run dev`:

```
$ npm run dev
```

Приложение запустится локально на порте 3000. Открыв браузер, вы должны увидеть страницу с приветствием (рис. 11.3). Если она не отображается, убедитесь в том, что не пропустили команду `npm install`.

Рис. 11.3. Страница начального шаблона Nuxt.js

Теперь рассмотрим использование Nuxt.js на примере данного приложения.

11.2.1. Создание приложения для поиска музыки

Приложения с отрисовкой на стороне сервера могут оказаться полезными и многофункциональными. Посмотрим, как Nuxt.js помогает в разработке. Представьте, что вам нужно создать приложение, взаимодействующее с API iTunes. iTunes API содержит информацию о миллионах исполнителей и альбомов. Реализуем поиск по музыкантам с выводом их дискографии.

ДОПОЛНИТЕЛЬНО

Больше подробностей об iTunes API можно найти в официальной документации по адресу mng.bz/rm99.

Приложение будет состоять из двух разных маршрутов. Первый из них будет отображать поле для поиска по iTunes API. Внешний вид этой страницы показан на рис. 11.4.

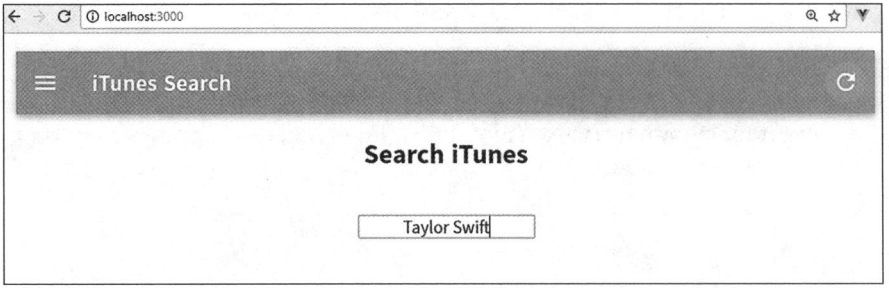


Рис. 11.4. Страница поиска по iTunes API

ДОПОЛНИТЕЛЬНО

Чтобы придать проекту приятный вид, используем библиотеку компонентов Vuetify. Мы вернемся к ней чуть позже.

Чтобы сделать этот пример более интересным, будем передавать информацию между маршрутами с помощью параметра. После ввода в поле поиска имени музыканта (Taylor Swift) на экране появится страница с результатами (рис. 11.5). При этом в адресной строке вверху можно видеть значение `Taylor%20Swift`.

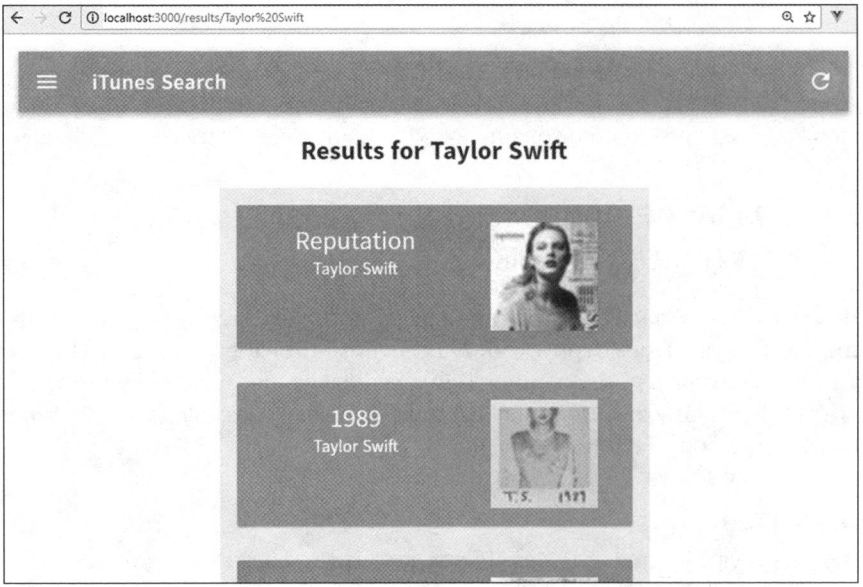


Рис. 11.5. Страница с результатами поиска

Страница поиска отобразит все альбомы, принадлежащие заданному исполнителю, в частности их название, имя музыканта, обложку и карточку со ссылкой на

соответствующую страницу iTunes. В этом примере мы рассмотрим также промежуточные обработчики, которые позволят выполнять код перед отрисовкой маршрута. Вы увидите, как взаимодействовать с iTunes API с помощью библиотеки Axios. В конце мы вернемся к Vuex.

11.2.2. Создание проекта и установка зависимостей

Приступим к созданию поискового приложения, воспользовавшись начальным шаблоном для Vue-CLI. Вслед за этим установим зависимости. Выполните в терминале такую команду:

```
$ vue init nuxt-community/starter-template itunes-search
```

После создания проекта следует установить пакеты Vuetify и Axios с помощью команды `npm install`. Кроме того, Vuetify требует наличия библиотек stylus и stylus-loader для загрузки CSS в формате stylus.

ПРИМЕЧАНИЕ

Vuetify — это графическая библиотека в стиле Material Design для Vue.js 2.0. Она содержит множество красивых и простых в применении компонентов. Библиотека похожа на другие фреймворки для создания пользовательских интерфейсов, такие как Bootstrap. Узнать о ней больше можно на vuetifyjs.com — официальном веб-сайте Vuetify.

Чтобы установить Vuetify, Axios, stylus и stylus-loader, выполните следующие команды:

```
$ cd itunes-search
$ npm install
$ npm install vuetify
$ npm install axios
$ npm install stylus --save-dev
$ npm install stylus-loader --save-dev
```

Будут установлены все нужные нам зависимости. Но для их корректной работы необходимо выполнить некоторые дополнительные действия. Мы укажем Axios и Vuetify в разделе `vendor` конфигурационного файла, зарегистрируем Vuetify и настроим расширение для этой библиотеки. Не забудем также подготовить CSS и шрифты.

Файл `nuxt.config.js` в папке /itunes-search используется для конфигурации Nuxt-приложений. Откройте его и найдите строку, которая начинается с `extend (config, ctx)`. Этот участок применяется для автоматического запуска ESLint при каждом сохранении кода (ESLint — это расширяемая утилита для статического анализа кода, стилей, форматирования и прочих вещей). Мы могли бы изменить стандартные параметры статического анализа, отредактировав файл `.eslintrc.js`, но, чтобы

не усложнять пример, просто удалим этот раздел. Таким образом автоматическая проверка кода будет выключена. Затем создадим параметр `vendor` в разделе `build` и укажем с его помощью библиотеки Axios и Vuetify, как показано в листинге 11.1.

Позвольте объяснить, как это все работает. Код любого модуля, который импортируется в Nuxt.js, добавляется в пакет, создаваемый системой Webpack. Это часть процесса под названием *«разделение кода»*. Webpack делит код на пакеты, которые могут загружаться по требованию или параллельно. Код библиотек, указанных в параметре `vendor`, добавляется только в пакет vendor. Если его не использовать, библиотеки будут добавляться в каждый пакет, в котором они импортируются, что увеличит размер приложения. Все модули проекта желательно указывать в параметре `vendor`, иначе они будут дублироваться (в Nuxt 2.0 этот параметр больше не является обязательным, поэтому можете его убрать). Приведите файл `package.json` в корневом каталоге проекта /itunes-search к следующему виду (листинг 11.1).

Листинг 11.1. Добавление библиотек в пакет vendor: chapter-11/itunes-search/nuxt.config.js

```
...
  build: {
    vendor: ['axios', 'vuetify']
  }
...
```

Добавление Axios и Vuetify в пакет vendor

Мы указали Axios и Vuetify в параметре `vendor`, но этого недостаточно. Vuetify требует дополнительной настройки. В файле `nuxt.config.js` нужно создать раздел `plugins` и указать в нем папку /plugins.

Механизм расширений в Nuxt.js позволяет подключать к приложению внешние модули, что требует небольшой подготовки. Расширения выполняются перед созданием корневого экземпляра Vue.js. В отличие от библиотек, указанных в параметре `vendor`, соответствующие файлы запускаются из папки /plugins.

В официальной документации для Vuetify эту библиотеку рекомендуется импортировать и подключать к Vue в виде расширения. Мы добавим соответствующий код в файл `vuetify.js` внутри /plugins. Создайте этот файл и зарегистрируйте в нем Vuetify, как показано в листинге 11.2.

Листинг 11.2. Добавление расширения Vuetify: chapter-11/itunes-search/plugins/vuetify.js

```
import Vue from 'vue'
import Vuetify from 'vuetify'

Vue.use(Vuetify)
```

Подключение Vuetify к Vue-приложению

Затем нужно указать ссылку на это расширение в конфигурации Nuxt.js. Откройте файл `nuxt.config.js` в корневом каталоге проекта и добавьте раздел `plugins` (листинг 11.3).

Листинг 11.3. Добавление ссылки на расширение: chapter-11/itunes-search/nuxt.config.js

```
...
  plugins: ['~plugins/vuetify.js'],
...
```

Ссылка на файл расширения

Последнее, что нужно сделать для настройки Vuetify, – добавить CSS. В официальной документации рекомендуется импортировать значки Material Design с вебсайта Google и указать ссылку на CSS-файл Vuetify.

Помните, как ранее мы устанавливали stylus-loader? Благодаря этому теперь можем добавить в `nuxt.config.js` ссылку на наш собственный файл формата stylus. В разделе `css` вверху удалите значение `main.css` (если оно там есть) и укажите файл `app.styl`, который мы создадим чуть позже. Не забудьте также добавить таблицу стилей для значков Material Design в разделе `head`. После всех изменений файл `nuxt.config.js` должен иметь следующий вид (листинг 11.4).

Листинг 11.4. Добавление CC и шрифтов: chapter-11/itunes-search/nuxt.config.js

```
module.exports = {
  /*
  ** Headers of the page
  */
  head: {
    title: 'iTunes Search App',
    meta: [
      { charset: 'utf-8' },
      { name: 'viewport', content: 'width=device-width, initial-scale=1' },
      { hid: 'description', name: 'description',
        content: 'iTunes search project' }
    ],
    link: [
      { rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' },
      {rel: 'stylesheet',
       href: 'https://fonts.googleapis.com/css?family=
➥          Roboto:300,400,500,700|Material+Icons'}
    ]
  },
  plugins: ['~plugins/vuetify.js'],
  css: ['~assets/app.styl'],
  /*
  ** Customize the progress bar color
  */
  loading: { color: '#3B8070' },
  /*
  ** Build configuration
  */
  build: {
    vendor: ['axios', 'vuetify']
  }
}
```

Добавляет ссылку на значки Material Design

Удаляет main.css и добавляет ссылку на app.styl

Теперь нужно создать файл `assets/app.styl`, представленный в листинге 11.5. Он импортирует в приложение стили Vuetify.

Листинг 11.5. Добавление стилей в формате stylus: chapter-11/itunes-search/assets/app.styl

```
// Импорт главной таблицы стилей Vuetify
@require '~vuetify/src/stylus/main'
```

Сделав все это, выполните команду `npm run dev` и убедитесь в том, что в консоли нет никаких ошибок. Если ошибки все же возникли, откройте файл `nuxt.config.js` и поищите пропущенные запятые и опечатки. Еще раз проверьте наличие всех зависимостей, включая `stylus` и `stylus-loader`, — они необходимы для работы Vuetify.

11.2.3. Создание элементов интерфейса и компонентов

Компоненты — это кирпичики, из которых состоит приложение. Они позволяют разбить код на отдельные фрагменты, которые затем можно собрать в единое целое. Прежде чем приступать к построению маршрутов, следует обратить внимание на папку components. Именно в ней будут находиться все обычные, простые компоненты.

ПРИМЕЧАНИЕ

Nuxt.js делит компоненты на два вида: заряженные и обычные. Заряженные компоненты имеют доступ к специальным параметрам Nuxt.js и всегда размещаются в папке pages. Эти параметры позволяют обращаться к данным на стороне сервера. В каталоге pages находятся также конфигурация маршрутов и главная страница приложения.

В этом разделе рассмотрим компоненты, размещенные в папке components. Мы создадим два компонента: `Card` (станет хранить информацию о каждом найденном альбоме исполнителя) и `Toolbar`. Последний будет представлять собой простую панель инструментов вверху каждого маршрута. В обоих случаях воспользуемся Vuetify. Я покажу вам HTML- и CSS-код этих компонентов, но без лишних подробностей.

ДОПОЛНИТЕЛЬНО

Если вас интересует полный список параметров для Vuetify, советую почитать инструкцию на странице vuetifyjs.com/en/getting-started/quick-start.

Создайте в папке components файл `Toolbar.vue`. Он будет содержать шаблон панели инструментов с несколькими компонентами из состава Vuetify. Мы также добавим CSS с ограниченной областью видимости, чтобы отключить подчеркивание ссылок. Готовый результат показан на рис. 11.6.

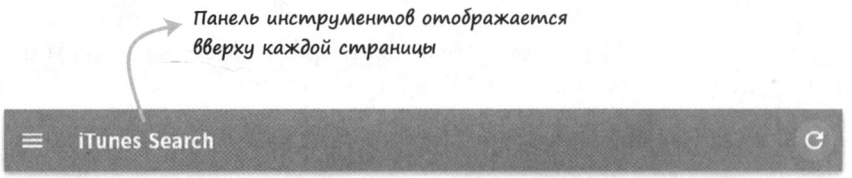


Рис. 11.6. Файл ToolBar.vue с панелью инструментов

В Vue.js для навигации по приложению обычно используется компонент `route-link`, но в Nuxt его не существует. Чтобы переключаться между маршрутами, задействуем его альтернативу `nuxt-link`, которая работает точно так же, как `route-link`. Как видно в листинге 11.6, `nuxt-link` создает ссылку на главную страницу приложения в ответ на щелчок на тексте `iTunes Search` вверху экрана. Добавьте этот код в файл `Toolbar.vue`.

Листинг 11.6. Добавление компонента Toolbar: chapter-11/itunes-search/components/Toolbar.vue

```
<template>
  <v-toolbar dark color="blue">
    <v-toolbar-side-icon></v-toolbar-side-icon>
    <v-toolbar-title class="white--text">
      <nuxt-link class="title" to="/">iTunes Search</nuxt-link>
    </v-toolbar-title>
    <v-spacer></v-spacer>
    <v-btn to="/" icon>
      <v-icon>refresh</v-icon>
    </v-btn>
  </v-toolbar>
</template>
<script>
</script>
<style scoped>
.title {
  text-decoration: none !important;
}
.title:visited{
  color: white;
}
</style>
```

Компонент v-toolbar из состава Vuetify

Компонент nuxt-link указывает на "/"

Локальные стили для этого компонента

Затем необходимо создать компонент `Card`. Он будет отображать каждый альбом исполнителя на странице с результатами. Для его визуального оформления опять воспользуемся Vuetify. Итоговый результат показан на рис. 11.7.

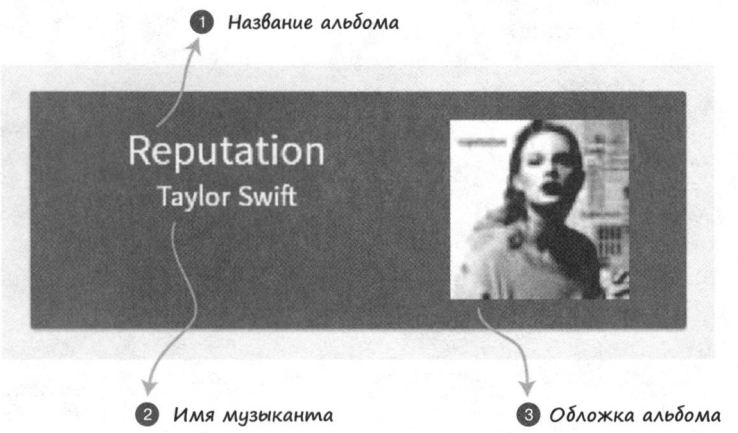


Рис. 11.7. Компонент Card.vue с демонстрационным текстом

Помимо Vuetify, мы будем применять входные параметры `title`, `image`, `artistName`, `url` и `color`. Маршрут `results` отвечает за обращение к API и извлечение информации об альбомах. Входные параметры будут передавать эту информацию в компоненты.

Компонент `v-card` принимает атрибуты `href` и `color`. Мы можем использовать директиву `v-on`, чтобы привязать к ним входные параметры. Компонент `v-card-media` принимает атрибут `img`. Привяжем к нему параметр `image`. Наконец, параметры `artistName` и `title` будут выводиться с помощью класса, который выровняет их по центру карточки. Скопируйте код, представленный в листинге 11.7, и вставьте его в новый файл под названием `Card.vue`.

Листинг 11.7. Добавление компонента Card: chapter-11/itunes-search/components/Card.vue

```
<template>
  <div id="e3" style="max-width: 400px; margin: auto;"
  class="grey lighten-3">
        <v-container
        fluid
        style="min-height: 0;"
        grid-list-lg>
        <v-layout row wrap>
          <v-flex xs12>
            <v-card target="_blank" :href="url" :color="color" class="white--text">
              <v-container fluid grid-list-lg>
                <v-layout row>
                  <v-flex xs7>
                    <div>
                      <div class="headline">{{title}}</div>
                      <div>{{artistName}}</div>
                    </div>
                  </v-flex>
                  <v-flex xs5>
                    <v-card-media
                    :src="image"
                    height="100px"
                    contain>
                    </v-card-media>
                  </v-flex>
                </v-layout>
              </v-container>
            </v-card>
          </v-flex>
        </v-layout>
    </v-container>
  </div>
</template>
<script>
export default {
    props: ['title', 'image', 'artistName', 'url', 'color'],
}
</script>
```

Компонент v-card из состава Vuetify принимает атрибуты href и color

Контейнер div с классом headline отображает название

div, отображающий имя исполнителя

Компонент v-card-media из состава Vuetify принимает путь к изображению

Список входных параметров, которые передаются в компонент

11.2.4. Обновление стандартной разметки

После создания компонентов следует обновить стандартную разметку в папке layouts. Как можно догадаться из названия, файл `default.vue` содержит разметку, в которую по умолчанию заворачивается каждая страница приложения. Это поведение можно переопределить внутри любого маршрута. Страницы в Nuxt представляют собой компоненты, поддерживающие специальные свойства для описания маршрутизации. Мы обсудим их структуру в следующем разделе.

В нашем простом случае достаточно будет обновить файл `default.vue` в папке /layouts и внести несколько мелких изменений. Мы хотим, чтобы компонент `Toolbar` отображался вверху каждого маршрута. Это позволит не добавлять его на каждую страницу приложения. Для этого следует подключить его к разметке по умолчанию. Добавьте в файл `default.vue` новый элемент `section` с классом `container`. Импортируйте компонент `Toolbar` внутри тега `<script>` и пропишите его в разделе `components`. Затем укажите `<ToolBar/>` сверху от `<nuxt/>`, как показано в листинге 11.8.

Листинг 11.8. Обновление разметки по умолчанию: chapter-11/itunes-search/layouts/default.vue

```
<template>
  <section class="container">
    <div>
      <ToolBar/>
      <nuxt/>
    </div>
  </section>
</template>

<script>
import ToolBar from '~/components/Toolbar.vue';
export default {
  components: {
    ToolBar
  }
}
</script>
<style>
...
```

Элемент section класса container с div внутри

Добавляем компонент Toolbar в шаблон

Импортируем компонент Toolbar

Закончив с разметкой, перейдем к Vuex.

11.2.5. Добавление хранилища на основе Vuex

Информация об альбомах, полученная из iTunes API, будет находиться в хранилище Vuex. В Nuxt.js хранилище доступно с любого участка приложения, включая промежуточные обработчики. Последние позволяют создавать код, который выполняется перед загрузкой маршрута. Рассмотрим этот механизм в следующем разделе.

В файле `store/index.js` создадим простое хранилище с использованием Vuex. Оно будет содержать в своем состоянии одно свойство `albums` и мутацию под

названием add, которая принимает дополнительный параметр и присваивает его вышеупомянутому свойству. Создайте файл index.js и вставьте в него следующий код (листинг 11.9).

Листинг 11.9. Добавление хранилища Vuex: chapter-11/itunes-search/store/index.js

```
import Vuex from 'vuex'

const createStore = () => {
  return new Vuex.Store({
    state: {
      albums: []
    },
    mutations: {
      add (state, payload) {
        state.albums = payload;
      }
    }
  })
}

export default createStore
```

Состояние хранилища Vuex содержит лишь одно свойство — albums

Мутация, которая присваивает дополнительный параметр свойству albums

Теперь мы можем обратиться к API и поместить результат в новое хранилище. Все это произойдет в промежуточном обработчике.

11.2.6. Использование промежуточных обработчиков

Термин *«промежуточные обработчики»* используется в Node.js и Express в отношении функций, имеющих доступ к объекту request (запрос) или response (ответ). Похожий механизм существует и в Nuxt.js. Он работает на сервере и клиенте, и его можно применить к любой странице приложения. Промежуточные обработчики выполняются перед отрисовкой маршрута и имеют доступ как к запросу, так и к ответу.

ПРИМЕЧАНИЕ

Промежуточные обработчики и метод asyncData, который мы рассмотрим чуть позже, работают по обе стороны браузера. Это означает, что при первой загрузке маршрута они запускаются на серверной стороне. Однако в дальнейшем при переходе на тот же маршрут выполняются уже на клиенте. Иногда возникает необходимость в выполнении кода исключительно на сервере. На этот случай в файле nuxt.config.js предусмотрен параметр serverMiddleware. Подробнее об этом можно почитать в официальном руководстве по адресу nuxtjs.org/api/configuration-servermiddleware/.

Промежуточные обработчики хранятся в отдельных файлах в каталоге /middleware. Каждый файл содержит функцию, которая принимает объект context. Среди мно-

жества ключей этого объекта можно выделить `request`, `response`, `store`, `params` и `environment`. Их полный список есть в официальной документации на странице nuxtjs.org/api/context.

В своем приложении мы хотим передавать имя исполнителя через параметр маршрута, который находится в объекте `context.params`. С помощью этого параметра можно сформировать запрос к поисковому API iTunes и получить в ответ список альбомов. Затем присвоить этот список свойству `albums` внутри хранилища Vuex.

Чтобы упростить выполнение запроса к серверу, воспользуемся сторонней библиотекой. Среди множества доступных вариантов я предпочитаю Axios — HTTP-библиотеку, с помощью которой можно выполнять HTTP-запросы как в браузере, так и внутри Node.js. Она поддерживает объекты Promise и автоматически преобразует данные в формате JSON. Подробнее о ней рассказывается на официальной странице на GitHub github.com/axios/axios.

Создайте в папке middleware файл `search.js` и добавьте в него код из листинга 11.10. Этот код выполняет HTTP-запрос типа GET к iTunes API, передавая `params.id` в качестве искомой строки. Возвращенный объект Promise вызывает мутацию add с помощью функции `store.commit`. Вы, наверное, заметили, что для `{params, store}` используется деструктурирующее присваивание формата ES6. Вместо того чтобы передавать весь контекст, мы можем его деструктурировать и оставить только нужные ключи.

Листинг 11.10. Создание промежуточного обработчика: chapter-11/itunes-search/middleware/search.js

```
import axios from 'axios'

export default function ( {params, store} ) {
  return axios.get(`https://itunes.apple.com/search?term=${params.id}&entity=album`)
        .then((response) => {
            store.commit('add', response.data.results)
            //  console.log(response.data.results);
        });
}
```

Функция по умолчанию, имеющая доступ к store и params

Результат запроса помещается в хранилище

Теперь у нас есть все необходимое для создания страниц и маршрута.

11.2.7. Создание маршрутов в Nuxt.js

Маршрутизация в Nuxt.js немного отличается от того, что вы видели в обычных Vue-приложениях. Здесь нет объекта `VueRouter`, в котором необходимо прописывать каждый маршрут. Вместо этого маршруты описываются в виде дерева каталогов внутри каталога pages.

Каждый каталог представляет собой отдельный маршрут. То же самое относится и к файлам `.vue`, размещаемым внутри этих каталогов. Представьте, что у вас есть маршрут `pages` с вложенным маршрутом `user`. Чтобы создать такую иерархию, дерево каталогов должно выглядеть как на рис. 11.8.

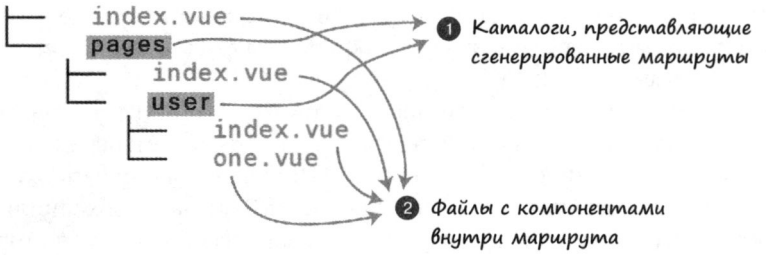


Рис. 11.8. Структура каталогов, описывающая маршруты

Дерево каталогов в папке pages автоматически сгенерирует маршруты, показанные в листинге 11.11.

Листинг 11.11. Автоматическое создание иерархии маршрутов

```
router: {
  routes: [
    {
      name: 'index',
      path: '/',
      component: 'pages/index.vue'
    },
    {
      name: 'user',
      path: '/user',
      component: 'pages/user/index.vue'
    },
    {
      name: 'user-one',
      path: '/user/one',
      component: 'pages/user/one.vue'
    }
  ]
}
```

Маршрут по умолчанию внутри pages

Это небольшой пример маршрутизации в Nuxt.js. Больше информации об этом можно найти в официальном руководстве на странице nuxtjs.org/guide/routing.

В нашем приложении все будет намного проще. Нам понадобится лишь два маршрута, один из которых динамический. Для определения динамического маршрута в Nuxt перед его именем следует указать знак подчеркивания. Как показано на рис. 11.9, в корне папки pages содержится файл `index.vue`. Это корневой компонент, который будет загружаться при запуске приложения. Здесь также есть файл `README.md`. Он всего лишь напоминает о том, что должно находиться внутри каталога, поэтому его можно удалить. Маршрут `_id` динамический. Идентификатор соответствует имени исполнителя и передается в маршрут.

Создайте каталог results внутри папки pages. Затем откройте файл `index.vue`. Удалите все его содержимое и вставьте вместо него код из листинга 11.12. Вверху

находится шаблон с тегом `<h1>` и элементом `<form>`. Директива `v-on` привязана к событию `submit` внутри формы. Мы также используем модификатор события `prevent`, чтобы форму нельзя было отправить.

Рис. 11.9. Структура каталогов приложения для поиска по iTunes

Внутри метода `submit` применяется вызов `this.$router.push`, который перенаправляет приложение на страницу results/. Поисковый запрос будет передаваться в маршрут в виде параметра. Поскольку маршрут `_id` динамический, запрос входит в состав URL-адреса. Например, если мы ищем `Taylor Swift`, адресная строка будет выглядеть как /results/taylor%20swift. Не обращайте внимания на символы %20 — они добавляются автоматически и обозначают пробел.

Создайте внизу страницы тег `style`, как показано в листинге 11.12. Это делается для того, чтобы выровнять текст по центру и добавить небольшой отступ.

Листинг 11.12. Создание страницы index: chapter-11/itunes-search/pages/index.vue

```
<template>
  <div>
    <h1>Search iTunes</h1>
    <br/>
    <form @submit.prevent="submit">
      <input placeholder="Enter Artist Name"
             v-model="search" ref='search' autofocus   />
    </form>
  </div>
</template>
<script>
export default {
  data() {
    return {
      search: ''
    }
  },
  methods: {
      submit(event) {
        this.$router.push(`results/${this.search}`);

      }
  }
}
```

Элемент form с директивой v-on, которая в ответ на событие submit запускает одноименный метод

Направляем приложение на страницу с результатами

```
</script>

<style>
* {
  text-align: center;
}

h1 {
  padding: 20px;
}
</style>
```

Выравниваем текст по центру и добавляем отступ

Заключительный аспект этого приложения — страница `_id`, которая будет выводить каждый найденный альбом в виде карточек. Кроме того, карточки будут поочередно менять свой цвет с синего на красный и обратно.

Ранее я упоминал, что страницы являются «заряженными» компонентами. Иными словами, они поддерживают специальные параметры, доступные только в Nuxt, включая `fetch`, `scrollToTop`, `head`, `transition`, `layout` и `validate`. Далее мы рассмотрим еще два параметра, `asyncData` и `middleware`. Больше о них можно узнать из официальной документации, находящейся по адресу nuxtjs.org/guide/views.

Параметр `middleware` позволяет определить промежуточный обработчик для страницы. Он будет выполняться при каждой загрузке компонента. В листинге 11.13 видно, что файл `_id.vue` задействует обработчик `search`, который мы создали ранее.

Еще один параметр, `asyncData`, позволяет извлекать данные и отрисовывать их на сервере без использования хранилища. В разделе, посвященном промежуточным обработчикам, нам приходилось сохранять информацию в хранилище Vuex, чтобы она была доступной для компонентов. Благодаря `asyncData` можно больше этого не делать. Сначала посмотрим, как обращаться к данным внутри промежуточного обработчика. Затем обновим код с помощью `asyncData`.

Создайте в папке pages/results файл `_id.vue`. Добавьте в этот новый компонент директиву `v-if` для `albumData`. Это гарантирует, что информация об альбомах будет сначала загружена и лишь потом выведена на экран. После этого создайте директиву `v-for`, которая перебирает содержимое `albumData`.

На каждой итерации будем отображать карточку с такими свойствами альбома, как `title`, `image`, `artistName`, `url` и `color`. Свойство `color` будет вычисляться с помощью метода `picker`, который переключается между красным и синим цветами в зависимости от индекса.

Вверху файла будет выводиться значение `{{$route.params.id}}`. Это параметр, полученный из результатов поиска.

Как можно видеть в листинге 11.13, мы добавляем вычисляемое свойство `albumData`. Оно будет извлекать данные из хранилища. Хранилище наполняется с помощью промежуточного обработчика `search`, который срабатывает при загрузке маршрута (см. далее).

Выполните команду `npm run dev` и откройте в браузере localhost с портом 3000. Если сервер уже запущен, остановите его и запустите заново. Вы должны увидеть прило-

жения для поиска по iTunes. Если что-то пошло не так, поищите ошибки в консоли. Это может быть обычная опечатка в имени компонента.

Листинг 11.13. Создание динамического маршрута: chapter-11/itunes-search/pages/results/_id.vue

```
<template>
  <div>
    <h1>Results for {{$route.params.id}}</h1>
    <div v-if="albumData">
      <div v-for="(album, index) in albumData">
        <Card :title="album.collectionCensoredName"
              :image="album.artworkUrl60"
              :artistName="album.artistName"
              :url="album.artistViewUrl"
              :color="picker(index)"/>
      </div>
    </div>
  </div>
</template>
<script>
import axios from 'axios';
import Card from '~/components/Card.vue'
export default {
    components: {
      Card
    },
    methods: {
      picker(index) {
          return index % 2 == 0 ? 'red' : 'blue'
      }
    },
    computed: {
    /*  albumData(){
        return this.$store.state.albums;
      }*/
    },
    middleware: 'search'
}
</script>
```

Параметр маршрута, переданный из поиска

Директива v-if отображается только при наличии albumData

Директива v-for перебирает содержимое albumData

Компонент Card, которому передается информация об альбоме

Метод picker поочередно возвращает красный и синий цвет

Вычисляемое свойство возвращает из хранилища свойство album

Определяет промежуточные обработчики для этого маршрута

Внесем в код еще одно изменение. Как уже упоминалось, у нас есть доступ к методу `asyncData`. Он используется для загрузки данных на стороне сервера во время инициализации компонента. Как и промежуточные обработчики, он принимает контекст приложения.

Будьте осторожны, работая с `asyncData`. Этот метод не имеет доступа к компоненту, поскольку вызывается еще до его инициализации. Тем не менее он объединит полученную информацию с данными компонента, что позволит обойтись без Vuex. Подробнее об `asyncData` можно почитать в официальной документации по адресу ru.nuxtjs.org/guide/async-data/.

Вернитесь к файлу `_id.vue` и удалите вычисляемое свойство `albumData`. Оно больше не понадобится. Создайте вместо него метод `asyncData`, как показано в листинге 11.14. Внутри него выполним HTTP-запрос типа GET с использованием Axios. `asyncData`, как и промежуточные обработчики, имеет доступ к объекту `context`. Мы выполним деструктурирующее присваивание, чтобы извлечь параметры `params`, и затем воспользуемся ими при обращении к iTunes API. Полученный ответ присвоим объекту `albumData`, он станет доступен после инициализации компонента, как показано в листинге 11.14.

Листинг 11.14. Пример применения asyncData: chapter-11/itunes-search/pages/results/_id.vue

```
...
    asyncData ({ params }) {
      return axios.get(`https://itunes.apple.com/search?
                        term=${params.id}&entity=album`)
        .then((response) => {
            console.log(response.data.results);
            return {albumData: response.data.results}
        });
    },
...
```

Метод asyncData имеет доступ к ключу params

Ответ, который iTunes возвращает после выполнения команды axios.get с params.id

Возвращаем новое свойство albumData, доступное внутри компонента

На этом заканчиваем знакомство с `asyncData`. Сохраните файл и снова выполните команду `npm run dev`. Вы должны увидеть ту же страницу, что и прежде. Результаты не изменились, хотя мы больше не используем хранилище Vuex.

11.3. Взаимодействие с сервером при помощи Firebase и VuexFire

Firebase — это продукт от компании Google для быстрого создания мобильных и настольных приложений. Он предоставляет несколько сервисов, включая аналитику, базу данных, обмен сообщениями, отчеты о сбоях, облачное хранилище, хостинг и аутентификацию. Firebase автоматически масштабируется и позволяет сразу же приступать к разработке. Подробная информация обо всех возможностях Firebase размещена на официальной странице firebase.google.com.

В этом разделе мы рассмотрим пример использования двух сервисов Firebase: аутентификации и Realtime Database. Попробуем внедрить их в проект зоомагазина.

Представьте, что нам предъявили новые требования, согласно которым зоомагазин должен находиться в облаке и поддерживать аутентификацию. Как вы помните из предыдущих глав, наши данные хранятся в файле `products.json`. Придется перенести их в базу данных Realtime Database. Затем мы обновим приложение таким образом, чтобы оно извлекало информацию из Firebase, а не из файла.

Еще один важный момент заключается в добавлении простой аутентификации на основе встроенных облачных провайдеров Firebase. Мы создадим в заголовке новую кнопку для входа и выхода, а затем посмотрим, как поместить информацию о сеансе в хранилище Vuex. В итоге приложение должно выглядеть как на рис. 11.10.

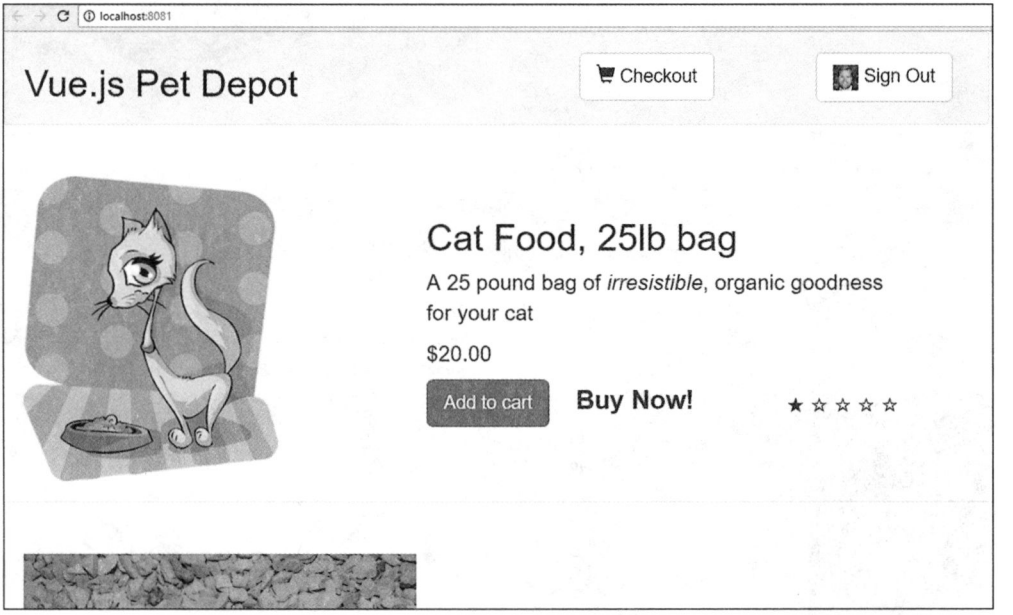


Рис. 11.10. Готовый проект зоомагазина с использованием Firebase

11.3.1. Подготовка к работе с Firebase

Если у вас уже есть учетная запись Google, можете сразу войти в систему на странице https://firebase.google.com/. В противном случае зарегистрируйтесь по адресу http://accounts.google.com — это бесплатно (Firebase начинает взимать плату только после совершения определенного количества транзакций в месяц).

После входа вы попадете на страницу приветствия и получите возможность создать новый проект (рис. 11.11).

После нажатия кнопки Add project (Добавить проект) нужно ввести название проекта и выбрать свою страну. Нажмите кнопку Create Project (Создать проект), чтобы перейти в консоль Firebase. Здесь мы сконфигурируем базу данных и аутентификацию, а также получим ключи, которые нужны для начала работы.

Выберите слева раздел Database (База данных). Будут предложены два варианта: Realtime Database и Cloud Firestore. Мы станем использовать Realtime Database. Нажмите кнопку Get Started (Начать) (рис. 11.12).

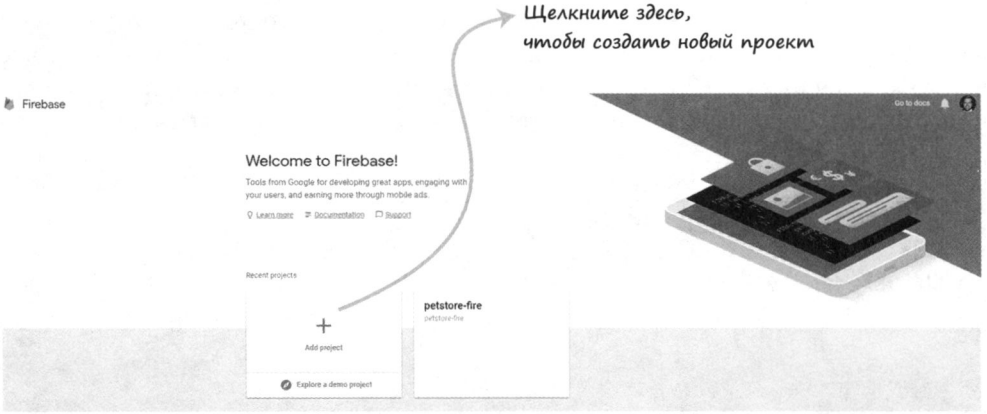


Рис. 11.11. Создание проекта в Firebase

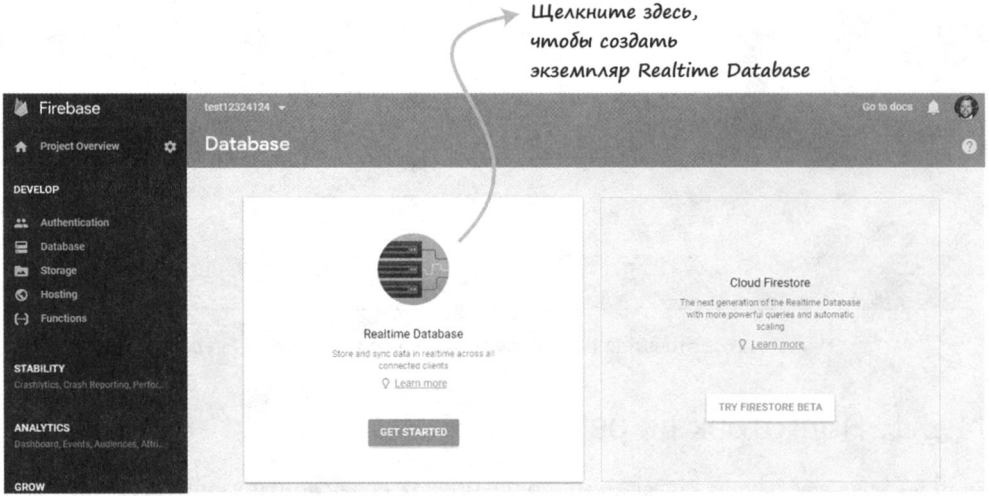


Рис. 11.12. Выбор базы данных

Теперь можно перенести файл `products.json` в базу данных Firebase. Мы могли бы импортировать его автоматически, но сделаем это вручную, чтобы увидеть, как все работает. Нажмите кнопку + рядом с названием базы данных — добавьте дочерний элемент `Products`. Прежде чем сохранить изменения, еще раз нажмите +, чтобы создать второго потомка. Введите число в поле Name (Имя). Еще раз щелкните на + и создайте семь дочерних элементов: `title`, `description`, `price`, `image`, `availableInventory`, `id` и `rating`. Заполните соответствующие поля и повторите все для следующего товара. Готовый результат должен выглядеть как на рис. 11.13.

После нажатия кнопки Add (Добавить) вы увидите в базе данных два товара. Если хотите, добавьте еще несколько, повторив те же шаги.

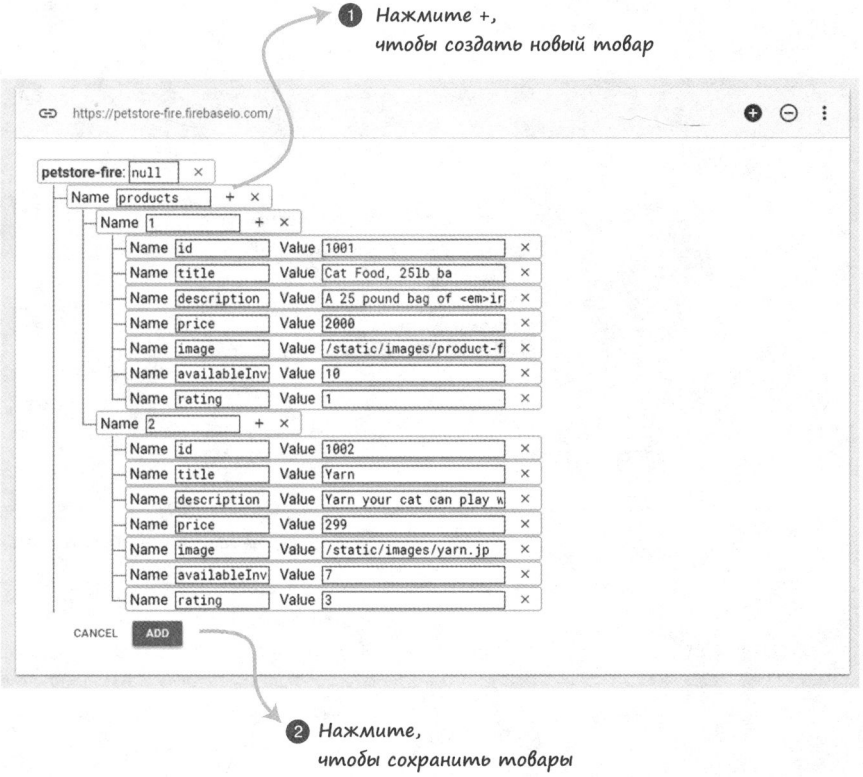


Рис. 11.13. Настройка Realtime Database в Firebase

Закончив с этим, мы должны сконфигурировать аутентификацию. Выберите слева в консоли раздел Authentication (Аутентификация). Перед вами откроется страница с кнопкой SET UP SIGN-IN METHOD (Настроить метод входа). Нажмите ее, как показано на рис. 11.14.

На следующей странице выберите метод Google. Будем использовать его для аутентификации в проекте. Мы могли бы с легкостью сконфигурировать вход через Facebook или Twitter, но в этом примере предполагается, что любой посетитель, желающий войти в наше приложение, должен иметь учетную запись Google. В окне настройки установите переключатель Enable (Включить) и сохраните изменения (рис. 11.15). Этого должно быть достаточно для обеспечения аутентификации через Google.

Напоследок нужно получить конфигурационную информацию. Вернитесь на страницу обзора проекта, выбрав слева пункт Project Overview (Обзор проекта). Вы увидите кнопку Add Firebase to your web app (Добавить Firebase в ваше веб-приложение). Нажав ее, вы перейдете на страницу с ключами Firebase и кодом для инициализации. Сохраните эту информацию для дальнейшего использования, она понадобится во время настройки Firebase в приложении.

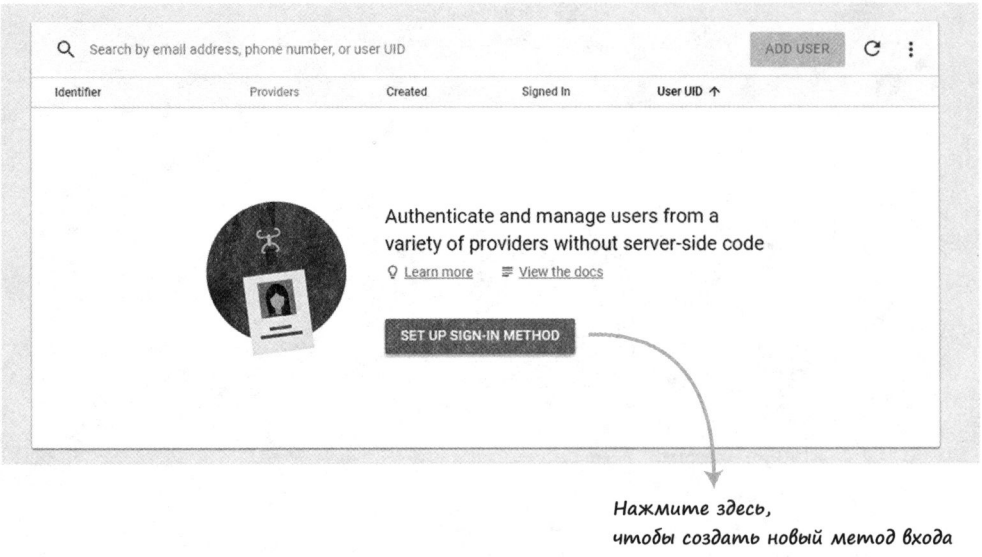


Рис. 11.14. Настройка аутентификации

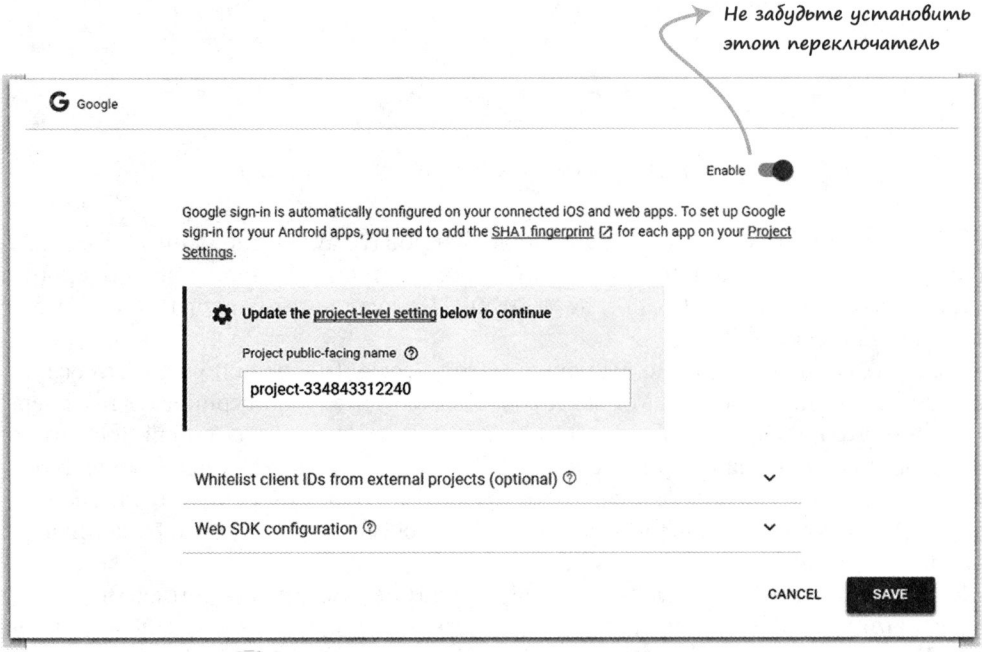


Рис. 11.15. Включение входа через Google

11.3.2. Подключение Firebase к проекту зоомагазина

Закончив с настройкой Firebase, переведем проект на применение этого сервиса. В предыдущий раз мы возвращались к зоомагазину в главе 10, чтобы добавить Vuex. Скопируйте приложение из предыдущей главы или воспользуйтесь архивом с кодом. Примем его в качестве отправной точки.

Для корректной работы с сервисом Firebase в Vue нам понадобится библиотека VueFire. Она поможет общаться с удаленным сервером и создаст нужные привязки.

Больше информации о VueFire содержится на официальной GitHub-странице по адресу github.com/vuejs/vuefire.

Откройте окно терминала и перейдите в каталог с проектом зоомагазина. Установите VueFire и Firebase с помощью следующих команд:

```
$ cd petstore
$ npm install firebase vuefire -save
```

Этим будут установлены и сохранены все необходимые зависимости.

Создайте файл `firebase.js` в папке src, которая находится в корне проекта. Помните код инициализации, который мы скопировали в консоли Firebase? Он сейчас пригодится. Вверху файла напишите `import {initializeApp} from Firebase`. После этого импорта создайте константу с именем `app` и вставьте ранее сохраненный код.

Создайте две инструкции экспорта: одну назовите db, другую — `productsRef`. Это позволит подключиться к базе данных Firebase и извлечь информацию о созданных ранее товарах. Если хотите узнать больше о Firebase API, ознакомьтесь с официальной документацией на странице firebase.google.com/docs/reference/js/. Скопируйте код из листинга 11.15 в файл `src/firebase.js`.

Листинг 11.15. Настройка Firebase и инициализация файлов: chapter-11/petstore/src/firebase.js

```
import { initializeApp } from 'firebase';

const app = initializeApp({
    apiKey: "",
    authDomain: "",
    databaseURL: "",
    projectId: "",
    storageBucket: "",
    messagingSenderId: ""
});

export const db = app.database();

export const productsRef = db.ref('products');
```

Импортируем initializeApp в файл

Ключи для Firebase, полученные в консоли Firebase

Экспорт базы данных в стиле ES6

Экспорт списка товаров в стиле ES6

Теперь нужно сделать так, чтобы файл `main.js` мог видеть библиотеку VueFire, которую мы установили чуть раньше. Строчка `Vue.use(VueFire)` подключит VueFire

к приложению в виде расширения. Приведите файл `src/main.js` к следующему виду (листинг 11.16).

Листинг 11.16. Подготовка главного файла: chapter-11/petstore/src/main.js

```
import Vue from 'vue'
import App from './App'
import router from './router'
require('./assets/app.css')
import { store } from './store/store';
import firebase from 'firebase';
import './firebase';
import VueFire from 'vuefire';

Vue.use(VueFire);
Vue.config.productionTip = false

/* eslint-disable no-new */
new Vue({
  el: '#app',
  router,
  store,
  template: '<App/>',
  components: { App }
})
```

Импорт Firebase в приложение

Импорт файла firebase.js

Импорт VueFire

Подключение VueFire в виде расширения

На этом этапе неплохо бы убедиться в том, что нет никаких ошибок. Сохраните все свои файлы и выполните в терминале команду `npm run dev`, чтобы запустить локальный сервер. Поищите ошибки в консоли. В файле `main.js` можно было случайно пропустить инструкцию импорта.

Итак, все готово. Теперь посмотрим, как сконфигурировать аутентификацию в приложении.

11.3.3. Хранение состояния аутентификации в Vuex

Я уже упоминал, что мы собираемся реализовать в проекте аутентификацию. Для размещения соответствующей информации нужно обновить хранилище Vuex. Чтобы сделать все максимально просто, поместим внутри состояния свойство `session`. После аутентификации пользователя Firebase возвращает объект с данными о текущем сеансе. Эту информацию рекомендуется сохранять так, чтобы она была доступна на любом участке приложения.

Откройте файл `store/modules/products.js` и создайте внутри состояния свойство `session`. Мы также добавим для него геттер и мутацию, как делали в предыдущей главе. Мутация будет называться `SET_SESSION`. Приведите файл `store/modules/products.js` к следующему виду (листинг 11.17).

Листинг 11.17. Обновление Vuex: chapter-11\petstore\src\store\modules/products.js

```
const state = {
    products: {},
    session: false
};

const getters = {
    products: state => state.products,
    session: state => state.session
};

const actions = {
    initStore: ({commit}) => {
      axios.get('static/products.json')
      .then((response) =>{
        console.log(response.data.products);
        commit('SET_STORE', response.data.products )
      });
    }
};

const mutations = {
    'SET_STORE' (state, products) {
      state.products = products;
    },
    'SET_SESSION' (state, session) {
      state.session = session;
    }
};

export default {
    state,
    getters,
    actions,
    mutations,
}
```

Свойство состояния session по умолчанию равно false

Геттер для session

Мутация SET_SESSION устанавливает содержимое session

Теперь добавим код, который будет извлекать данные о сеансе из Firebase и помещать их в хранилище Vuex.

11.3.4. Поддержка аутентификации в компоненте Header

Компонент Header отображает название сайта и кнопку корзины. Добавим в него кнопки для входа и выхода.

На рис. 11.16 показан итоговый результат после аутентификации посетителя. Обратите внимание на значок рядом с текстом Sign Out (Выйти). Он извлекается из объекта `user`, полученного из Firebase.

Рис. 11.16. Пользователь вошел

Когда пользователь выйдет, текст кнопки поменяется на Sign In (Войти) (рис. 11.17).

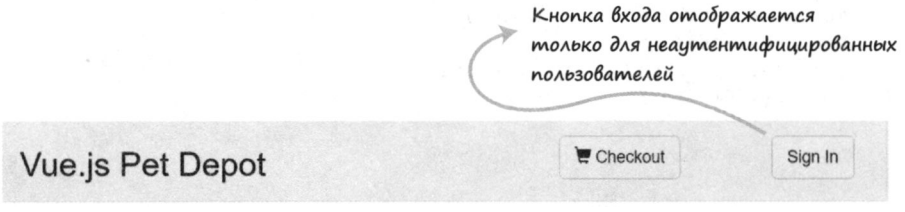


Рис. 11.17. Пользователь вышел

Откройте файл `src/components/Header.vue` и добавьте в его шаблон новые кнопки. Нужно также создать два новых метода для входа и выхода. Создайте снизу от `navbar-header` новые разделы `div` для аутентификации и завершения сеанса (листинг 11.18). Во втором разделе добавьте изображение, которое будет извлекаться из свойства `mySession`.

Внутри второго тега `div` будет находиться директива `v-if`. Если свойство `mySession` равно `false`, она выведет кнопку Sign In (Войти). Если это свойство равно `true`, выведем кнопку Sign Out (Выйти) с помощью директивы `v-else`. Иными словами, если сеанс уже начат, мы увидим кнопку Sign Out (Выйти), а если нет — кнопку Sign In (Войти).

Поскольку компонент `Header` довольно объемный, я разбил его на три листинга — 11.18, 11.19 и 11.20. Объедините их и вставьте результат в файл `src/components/Header.vue` (удалив предыдущее содержимое), как показано далее.

Листинг 11.18. Обновление компонента Header: chapter-11/header-temp.html

```
<template>
  <header>
    <div class="navbar navbar-default">
      <div class="navbar-header">
        <h1><router-link :to="{name: 'iMain'}">
          {{ sitename }}
        </router-link></h1>
      </div>
      <div class="nav navbar-nav navbar-right cart">
```

```
      <div v-if="!mySession">
        <button type="button"
          class="btn btn-default btn-lg"
          v-on:click="signIn">
          Sign In
        </button>
      </div>
      <div v-else>
        <button type="button"
          class="btn btn-default btn-lg"
          v-on:click="signOut">
          <img class="photo"
          :src="mySession.photoURL" />
          Sign Out
        </button>
      </div>
    </div>
    <div class="nav navbar-nav navbar-right cart">
      <router-link
        active-class="active"
        tag="button"
        class="btn btn-default btn-lg"
        :to="{name: 'Form'}">
        <span class="glyphicon glyphicon-shopping-cart">
          {{cartItemCount}}
          </span>
        Checkout
    </button>
    </div>
  </div>
</header>
</template>
```

Кнопка входа выводится, если свойство mySession равно false

Кнопка входа с директивой v-on

Кнопка выхода выводится, если свойство mySession равно true

Изображение из mySession

Создадим в шаблоне два метода — `signIn` и `signOut`. Добавим также новое свойство под названием `mySession`. Теперь создадим для нашего компонента раздел `script` с этими двумя методами и вычисляемым свойством. Не забудьте импортировать `firebase from 'firebase'` в верхнюю часть кода (см. листинг 11.19).

Первым делом нужно добавить хук жизненного цикла `beforeCreate`, который срабатывает перед созданием компонента. Он будет помещать текущий сеанс в хранилище Vuex. В Firebase для этого предусмотрен удобный обработчик `onAuthStateChanged`. Он выполняется при каждом входе или выходе пользователя. С его помощью можно обновлять информацию о сеансе в хранилище, применяя мутацию `SET_STORE`. Подробнее об `onAuthStateChanged` читайте в официальной документации по адресу mng.bz/4F31.

Теперь, когда мы можем следить за тем, как пользователь входит и выходит, пришло время создать соответствующие обработчики. Добавьте метод `signIn` и выполните внутри него вызов `firebase.auth.GoogleAuthProvider()`. Передайте полученный провайдер в функцию `firebase.auth().signInWithPopup`. На экране появится всплывающее окно с предложением войти через учетную запись Google.

Функция `signInWithPopup` создает объект Promise. В случае успешного входа выведем в консоль строку `"signed in"`. В противном случае вы увидите в консоли сообщение `"error"`.

Как вы помните, у нас уже есть обработчик события `onAuthStateChanged` в хуке `beforeCreate`, поэтому после входа пользователя не нужно создавать никаких переменных. Обработчик автоматически обновляет хранилище при входе и выходе.

Метод `signOut` работает по тому же принципу. Когда пользователь завершает сеанс, в консоль выводится сообщение `"signed out"`. В случае возникновения ошибки вы увидите сообщение `"error in signed out!"`.

Вычисляемое свойство `mySession` будет возвращать геттер для значения `session`. Если сеанса не существует, он вернет `false`. Стоит отметить, что здесь можно было бы воспользоваться методом `mapGetters` из состава Vuex. Это позволило бы автоматически привязать геттер `session` к компоненту. Но, поскольку мы имеем дело лишь с одним геттером, я решил вернуть `this.$store.getters.session`.

Скопируйте код листинга 11.19 и добавьте его в конец объединенного файла `src/components/Header.vue`.

Листинг 11.19. Еще одно обновление компонента Header: chapter-11/header-script.js

```
<script>
import firebase from 'firebase';
export default {
  name: 'Header',
  data () {
    return {
      sitename: "Vue.js Pet Depot"
    }
  },
  props: ['cartItemCount'],
  beforeCreate() {
      firebase.auth().onAuthStateChanged((user)=> {
          this.$store.commit('SET_SESSION', user || false)
      });
  },
  methods: {
    showCheckout() {
      this.$router.push({name: 'Form'});
    },
    signIn() {
      let provider = new firebase.auth.GoogleAuthProvider();
      firebase.auth().signInWithPopup(provider).then(function(result) {
        console.log('signed in!');
      }).catch(function(error){
        console.log('error ' + error)
      });
    },
    signOut() {
      firebase.auth().signOut().then(function() {
        // Успешный выход
```

Обработчик onAuthStateChanged внутри хука beforeCreate

Метод signIn для входа пользователя

Метод signOut для выхода пользователя

```
        console.log("error in sign out!")
        // Произошла ошибка
      });
    }
  },
  computed: {
    mySession() {
      return this.$store.getters.session;
    }
  }
}
</script>
```

Вычисляемое свойство mySession получает информацию о сеансе

Наконец, нужно добавить новый класс `photo` в таблицу стилей, чтобы подогнать размер изображения под наши кнопки. Возьмите следующий код и объедините его с предыдущими листингами в рамках файла `Header.vue` в папке src/components.

Листинг 11.20. Обновление стилей компонента Header: chapter-11/header-style.html

```
<style scoped>
a {
  text-decoration: none;
  color: black;
}

.photo {
  width: 25px;
  height: 25px;
}

.router-link-exact-active {
  color: black;
}
</style>
```

Класс photo устанавливает ширину и высоту изображения

Добавив весь этот код в файл `Header.vue`, запустите команду `npm run dev` и убедитесь в отсутствии ошибок. Вы могли допустить опечатку в обработчике `onAuthStateChanged` или забыть сохранить изменения в хранилище Vuex.

11.3.5. Работа с базой данных Realtime Database в файле Main.vue

Разобравшись с аутентификацией, приступим к извлечению информации из базы данных. В конфигурации по умолчанию содержимое БД в Firebase доступно только для чтения. Нам это подходит.

Для начала обновим метод `mapGetters` в файле `src/components/Main.vue`. Как можно заметить, мы извлекаем геттер `products`. Уберите его и вставьте на его место

session. Нам этот код пока что не нужен, но всегда приятно знать, что информация о сеансе доступна в главном компоненте.

Для работы с Realtime Database достаточно импортировать объект productsRef из файла firebase.js. После этого необходимо создать объект firebase, который связывает productsRef с products. Вот и все! Остальной код в файле Main.vue можно не трогать. Обновите файл src/components/Main.vue в соответствии с листингом 11.21.

Листинг 11.21. Обновление файла Main.vue: chapter-11/update-main.js

```
...
import { productsRef } from '../firebase';
export default {
  name: 'imain',
  firebase: {
      products: productsRef
  },
...
  computed: {
    ...mapGetters([
        'session'
    ])
...
```

Импортируем productsRef из файла firebase.js

Привязываем productsRef к products

mapGetters теперь извлекает session вместо products

Сохраните все файлы и запустите команду npm run dev. При загрузке страницы в браузере можно заметить, что товары появляются с небольшой задержкой. Это говорит о том, что они загружаются из Firebase. Если зайти в консоль Firebase и добавить новые товары, они отобразятся на вашей странице.

Вам, наверное, интересно, как еще улучшить этот пример. С помощью свойства session некоторые разделы приложения можно было бы сделать доступными только аутентифицированным пользователям. Этого легко добиться с помощью директивы v-if или маршрутизатора. Например, мы могли бы добавить в маршрутизатор свойство meta и затем при переходе к определенным маршрутам проверять внутри хука router.beforeEach, вошел ли пользователь в систему. Этот механизм называется *хуками навигации*. Подробнее о нем читайте в официальной документации по адресу router.vuejs.org/ru/guide/advanced/navigation-guards.html. В следующей главе мы поговорим о тестировании и покажем, как сделать так, чтобы приложение вело себя предсказуемо.

Упражнение

Используя знания, приобретенные в этой главе, ответьте на следующий вопрос: каким преимуществом обладает метод asycData, доступный в Nuxt-приложениях, по сравнению с применением промежуточных обработчиков?

Ответ ищите в приложении Б.

Резюме

- Для взаимодействия с удаленными API можно использовать такие библиотеки, как Axios.
- Для ускорения загрузки сайтов применяются приложения Nuxt.js с отрисовкой на стороне сервера.
- Вы можете извлекать информацию из удаленного хранилища данных с помощью Firebase.
- Ваше приложение способно поддерживать аутентификацию пользователей.

12 Тестирование

В этой главе

- Зачем нужно тестирование.
- Написание модульных тестов.
- Тестирование компонентов.
- Тестирование Vuex.

В этой книге мы обсудили много важных тем, оставив напоследок одну область, которая часто не получает должного внимания: тестирование. Это чрезвычайно важный аспект разработки любого программного проекта. Тестирование позволяет гарантировать, что приложение ведет себя так, как мы того ожидаем, то есть без ошибок. В этой главе поговорим о том, почему вы должны писать тесты для своего кода, и рассмотрим основы модульного тестирования. А также научимся тестировать компоненты, включая их вывод и методы. В конце вы узнаете, как организовать тестирование хранилища Vuex.

Прежде чем начать, стоит отметить, что тестирование — обширная тема. В этой главе мы рассмотрим лишь несколько наиболее важных ее аспектов, относящихся к Vue.js. Я настоятельно рекомендую ознакомиться с книгой Эдда Йербурга (Edd Yerburgh) *Testing Vue.js Applications* (Manning, 2018). В ней намного подробнее рассматривается процесс создания и разработки тестов. Эдд затрагивает также тестирование SSR, снимков, примесей и фильтров.

Тестирование снимков

Тестирование снимков позволяет исключить непредсказуемые изменения в пользовательском интерфейсе. Здесь я буду применять пакет mocha-webpack, который на момент написания этих строк не поддерживает данную возможность. Однако в официальном руководстве есть информация о том, как тестировать снимки с помощью Jest: vue-test-utils.vuejs.org/ru/guides/#тестирование-однофайловых-компонентов-с-jest.

12.1. Создание тестовых сценариев

В мире разработки программного обеспечения существует два основных подхода к тестированию кода: ручной и автоматический. Сначала поговорим о написании тестов вручную.

Скорее всего, вы начали заниматься ручным тестированием при создании первых своих строчек кода, проверяя, соответствует ли полученный вывод ожиданиям. Например, в нашем зоомагазине есть кнопка для добавления товаров в корзину. В предыдущих главах мы нажимали ее вручную и проверяли количество элементов в массиве `cart`.

Там есть также кнопка для перехода на страницу оформления покупки. Опять-таки мы можем ее нажать и убедиться в том, что она направляет нас в нужное место. Ручное тестирование подходит для мелких проектов, в которых мало что происходит.

Теперь представьте, что вы работаете в команде с другими разработчиками. У вас есть настоящее приложение, над кодом которого трудится много людей. Коллеги постоянно вносят изменения в репозиторий. Как вы понимаете, рассчитывать на то, что каждый разработчик будет тестировать свой код вручную и выискивать потенциальные ошибки, просто нереально. Ручное тестирование превращается в кошмар и не избавляет от потенциальных проблем.

В некоторых организациях за ручное тестирование кода, написанного программистами, отвечает отдел обеспечения качества. Это помогает снизить вероятность попадания проблемного кода в рабочие версии, но в то же время замедляет весь процесс. Кроме того, у многих инженеров, отвечающих за качество кода, нет ни времени, ни ресурсов для проведения полноценного *регрессионного тестирования*.

ОПРЕДЕЛЕНИЕ

> Регрессионное тестирование помогает удостовериться в том, что обновление приложения не ухудшило его функциональность.

Автоматическое тестирование помогает решить некоторые проблемы, характерные для ручных тестов. В нашей гипотетической ситуации можно создать несколько автоматических тестов, которые будут запускаться при сохранении изменений в рабочую ветвь репозитория. Они выполняются намного быстрее по сравнению с тестированием вручную и имеют больше шансов немедленно выловить ошибку. Во многих случаях разработчик может запускать полноценные регрессионные тесты для всей кодовой базы и забыть о кропотливом ручном тестировании.

Несмотря на все свои преимущества, автоматические тесты не лишены недостатков. Например, на их написание уходит больше времени, чем на обычный код, хотя начальные затраты, скорее всего, окупятся в будущем. Как вы увидите в следующем разделе, налаживание таких процессов, как непрерывная интеграция, доставка и развертывание, может сэкономить уйму времени.

12.2. Непрерывная интеграция, доставка и развертывание

Автоматическое тестирование имеет одно дополнительное преимущество: оно делает возможным реализацию таких рабочих процессов, как непрерывная разработка. Она состоит из непрерывной интеграции, доставки и развертывания. Как можно понять из названия, все эти этапы связаны между собой. Далее мы пройдемся по каждому из них.

Представьте, что мы создаем простое приложение, которое подключается к базе данных и извлекает информацию о книгах. У нас есть команда разработчиков, которая занимается кодовой базой, но сталкивается с рядом проблем. Раз в несколько недель у большинства программистов возникают конфликты слияния при сохранении изменений в репозиторий. Кроме того, каждую пятницу кто-то должен вручную запускать *промежуточную среду* с последней версией продукта (в этой среде тестируется готовый к выпуску код). По мере роста и усложнения кодовой базы это начинает занимать все больше времени. Выпуск рабочих версий тоже проходит не без шероховатостей. В половине случаев они просто не собираются и на их исправление уходит несколько часов. Чтобы избавиться от этих проблем, руководитель проекта решил внедрить процесс непрерывной разработки. Первым шагом будет обеспечение непрерывной интеграции.

12.2.1. Непрерывная интеграция

Непрерывная интеграция (continuous integration, CI) — это процесс слияния кода с *основной ветвью* репозитория несколько раз в день.

ОПРЕДЕЛЕНИЕ

> Основная ветвь репозитория обычно содержит код, готовый к выпуску. Термин «ветвь» (branch) используется в системах контроля версий и означает дублирование кодовой базы с параллельным внесением изменений.

Непрерывная интеграция позволяет избежать конфликтов слияния, что является ее очевидным преимуществом. Конфликты слияния происходят в ситуациях, когда несколько разработчиков пытаются слить или объединить свой код в единую ветвь. Чтобы программист не мог испортить чужой код в процессе своей работы, изменения сливаются в основную ветвь по нескольку раз в день. Благодаря непрерывному обновлению основной ветви другие программисты всегда могут использовать в своей среде разработки относительно свежий код.

В нашем гипотетическом случае руководитель хочет сделать процесс CI более гладким, реализовав сервис, который будет запускать автоматические тесты перед обновлением репозитория. В системах контроля версий, таких как Git, разработчи-

ки могут подавать запросы на принятие изменений. Сервисы вроде Travis, CircleCI или Jenkins помогают проверить, проходят ли эти запросы все тестовые сценарии, и только затем позволяют слить код. После налаживания всех этих систем количество конфликтов сократилось, но у команды по-прежнему возникают проблемы с развертыванием.

12.2.2. Непрерывная доставка

Непрерывная доставка — это подход к разработке программного обеспечения, который помогает сделать процесс построения, тестирования и выпуска программ более быстрым и частым. Цель этой методики — обеспечение быстрого и надежного развертывания, которое состоит из цепочки проверок: все они должны пройти успешно, иначе код нельзя будет выпустить. Например, перед выпуском очередной версии программа должна пройти все тесты без ошибок или предупреждений. Это помогает сделать обновления более надежными и последовательными.

Обычно за CI отвечают не разработчики, а команда инженеров, известных как DevOps, которые занимаются настройкой и обслуживанием непрерывной доставки. Но вам все равно будет полезно познакомиться с основами этого процесса и понять, как он соотносится с тестированием.

При непрерывной доставке слияние или сохранение кода в основную ветвь инициирует построение сайта. Это может сэкономить время, исключив лишний шаг с ручным развертыванием в промежуточной среде. Еще одно преимущество состоит в том, что код будет развернут только в случае успешного прохождения всех тестов — это снижает вероятность получения неработоспособного приложения.

Благодаря непрерывной доставке мы можем больше не тратить время на создание и развертывание промежуточной среды. Этот процесс будет выполняться автоматически раз в сутки. Учитывая все сказанное, ответьте: какое отношение непрерывная доставка имеет к непрерывному развертыванию?

12.2.3. Непрерывное развертывание

Непрерывное развертывание идет еще дальше и позволяет при каждом изменении выдавать код прямо в промышленную среду. Как и в случае с непрерывной доставкой, разработчики могут быть уверены в том, что конечное приложение проходит все тесты.

Как можно предположить, развертывание в промышленной среде при каждом изменении чревато проблемами, если автоматические тесты не в состоянии проверить все участки приложения. В худшем случае вы получите неработающий веб-сайт, что потребует отката изменений или экстренного исправления кода.

Итак, я продемонстрировал, как можно внедрить тестирование в рабочий процесс. Теперь посмотрим, какие виды тестов нам доступны и как их использовать в сочетании с Vue.js.

12.3. Виды тестов

В мире тестирования существует несколько видов тестов. В этом разделе мы рассмотрим самые распространенные из них, включая модульные и компонентные.

Модульные тесты проверяют самые мелкие части приложения. Часто, но не всегда они являются частью кода и представляют собой функции или даже компоненты. Начнем с создания простого модульного теста и посмотрим, как он работает.

Модульные тесты обладают уникальными преимуществами. Каждый из них проверяет небольшой участок кода, поэтому их можно сделать быстрыми и надежными. Они также могут играть роль документации. В сущности, это инструкции о том, как код должен себя вести. Модульные тесты можно выполнить тысячи раз и получить один и тот же вывод. В отличие от других тестов они не должны полагаться на API, в которых часты сбои. Представьте, что вы создали приложение, которое переводит температуру по шкале Фаренгейта в градусы Цельсия. Оно состоит из одной-единственной функции, выполняющей преобразование. Мы легко напишем модульный тест для проверки возвращаемого значения. Это продемонстрировано в листинге 12.1.

Листинг 12.1. Простой модульный тест: chapter-12/unit.js

```
function convert(degree) {
  let value = parseFloat(degree);
  return (value-32)/ 1.8;
}

function testConvert() {
  if (convert(32) !== 0) {
    throw new Error("Conversion failed");
  }
}

testConvert();
```

Простой модульный тест для проверки функции convert

Существует еще один вид тестов, которые называют *компонентными*. Они выполняются для каждого компонента, проверяя его поведение. Ввиду более широкого охвата они могут оказаться немного сложнее модульных тестов и потребовать бóльших усилий при отладке. Компонентные тесты позволяют убедиться в том, что компонент соответствует поставленным требованиям и достигает своей цели.

12.4. Подготовка среды для тестирования

Получив представление о том, какие виды тестов нам доступны и зачем они нужны, можем приступить к подготовке среды для тестирования. Нашему зоомагазину не помешают несколько проверок — так давайте же их добавим!

В этом разделе мы модифицируем проект зоомагазина с использованием последних библиотек, рекомендуемых для тестирования Vue.js. На момент написания

книги данные библиотеки не входят в состав проектов, сгенерированных посредством Vue-CLI, поэтому нужно выполнить несколько подготовительных шагов. Мы установим несколько пакетов и отредактируем некоторые файлы.

Сначала следует скопировать приложение. Если вы выполняли все примеры, представленные в книге, можете взять собственный код. В противном случае возьмите проект зоомагазина из главы 11 в репозитории github.com/ErikCH/VuejsInActionCode.

Библиотека vue-test-utils — официальное средство модульного тестирования для Vue.js. Она значительно облегчает написание тестов, поэтому вы должны ее использовать. В этой главе рассмотрим лишь ее основные возможности, а если вы хотите узнать больше о том, как она работает, ознакомьтесь с официальным руководством, находящимся по адресу vue-test-utils.vuejs.org/ru/.

Как вы, наверное, помните, при создании проекта зоомагазина в главе 7 мы согласились с установкой пакетов Nightwatch и Karma. Они подходят для нашей задачи, однако на момент написания книги в Vue-CLI не было встроенной поддержки библиотеки vue-test-utils. Поэтому придется установить ее вручную.

Нужно также определиться с программой для запуска тестов, созданных на основе библиотеки vue-test-utils. При первой установке зоомагазина нам были доступны две такие программы: Mocha и Karma. Последняя совместима с vue-test-utils, но использовать ее не стоит. Разработчики vue-test-utils советуют применять либо Jest, либо mocha-webpack. Поскольку у нас уже есть пакет Mocha, остановимся на втором варианте. Jest тоже отличный инструмент, но мы не станем рассматривать его.

ПРИМЕЧАНИЕ

Все тесты, представленные в этой главе, будут работать, даже если вы выберете Jest. Но это потребует немного другой конфигурации. Инструкции о том, как настроить Jest, вы найдете в официальном руководстве по адресу https://vue-test-utils.vuejs.org/ru/guides/#чем-запускать-тесты.

Мы будем использовать mocha-webpack для запуска тестов, поэтому должны установить несколько дополнительных пакетов. Тесты можно выполнять в настоящем браузере, таком как Chrome или Firefox, но этот подход отличается низкой производительностью и малой гибкостью. Поэтому рекомендую задействовать консольный браузер на основе jsdom и jsdom-global. Эти модули симулируют работу браузера в консоли и применяются для выполнения тестовых сценариев.

ОПРЕДЕЛЕНИЕ

У консольных браузеров нет графического пользовательского интерфейса. Они применяются для автоматизации управления веб-страницами и во многом похожи на обычные современные браузеры. Отличие лишь в том, что взаимодействие с ними происходит с помощью интерфейса командной строки.

Нам также нужно выбрать библиотеку утверждений. Популярные варианты — Chai и Expect. Библиотеки утверждений позволяют проверять корректность

выражений без использования условных операторов `if`. Разработчики vue-test-utils рекомендуют применять Expect в сочетании с mocha-webpack, поэтому остановимся именно на этом пакете. Больше информации о библиотеках утверждений можно найти на странице https://vue-test-utils.vuejs.org/ru/guides/testing-single-file-components-with-mocha-webpack.html.

Напоследок нужно установить библиотеку webpack-node-externals. Она поможет исключить определенные зависимости из тестового пакета.

Загрузите последнюю версию зоомагазина, включенную в архив с кодом для этой книги. Если вы скопировали проект из главы 11, вам нужно перейти в папку src и ввести конфигурацию Firebase в файле `firebase.js` (если вы этого еще не сделали). Если пропустить этот шаг, приложение не запустится!

Загрузите последнюю версию зоомагазина и установите зависимости с помощью следующих команд:

```
$ cd petstore
$ npm install
$ npm install --save-dev @vue/test-utils mocha-webpack
$ npm install --save-dev jsdom jsdom-global
$ npm install --save-dev expect
$ npm install --save-dev webpack-node-externals
```

После установки зависимостей займемся конфигурацией. Откройте файл `webpack.base.conf.js` в папке build нашего проекта и добавьте в его конец код из листинга 12.2, чтобы настроить расширения webpack-node-externals и inline-cheap-module-source-map. Это требуется для их корректной работы.

Листинг 12.2. Настройка карты исходников и внешних зависимостей: chapter-12/setup.js

```
if (process.env.NODE_ENV === 'test') {
  module.exports.externals = [require('webpack-node-externals')()]
  module.exports.devtool = 'inline-cheap-module-source-map'
}
```

Настройка тестовой среды

В папке test можно найти каталоги unit и e2e. Они нам не нужны, поэтому удалите их. Создайте в папке test файл `setup.js`. В нем мы зададим глобальные переменные для `jsdom-global` и `expect`. Таким образом, нам не придется импортировать эти два модуля в каждом тестовом сценарии. Скопируйте код листинга 12.3 и вставьте его в файл `test/setup.js`.

Листинг 12.3. Настройка тестовой среды: chapter-12/petstore/setup.js

```
require('jsdom-global')()
global.expect = require('expect')
```

Подключаем jsdom-global

Делаем доступным модуль expect внутри приложения

Теперь нужно обновить тестовый скрипт в файле `package.json`. Этот скрипт вызовет программу mocha-webpack для запуска тестов. Для простоты все наши тесты будут иметь расширение `spec.js`. Приведите раздел `test` в файле `package.json` к следующему виду (листинг 12.4).

Листинг 12.4. Обновление package.json: chapter-12/testscript.js

```
"test": "mocha-webpack --webpack-config
build/webpack.base.conf.js --require
 test/setup.js test/**/*.spec.js"
```

Тестовый скрипт в файле package.json

Этой конфигурации будет достаточно для нашей среды. Теперь займемся созданием тестовых сценариев!

12.5. Создание первого тестового сценария с помощью vue-test-utils

В первом тесте на основе vue-test-utils попробуем проверить корректность работы компонента Form при нажатии кнопки Place Order (Отправить заказ). Ее нажатие должно приводить к появлению диалогового окна. Мы могли бы отследить вывод окна, но это потребовало бы изменения конфигурации jsdom-global. Специально для этого теста создадим свойство `madeOrder`. По умолчанию оно будет равно `false`. При нажатии кнопки Place Order (Отправить заказ) ему будет присвоено `true`.

Форма оформления покупки будет обновлена и начнет выводить внизу сообщение о выполнении заказа (рис. 12.1). Текст отображается только тогда, когда свойство `madeOrder` равно `true`. Это обеспечивает обратную связь при нажатии кнопки, поэтому мы можем отказаться от использования диалогового окна.

Чтобы внести это изменение, откройте файл `src/components/Form.vue` и добавьте в функцию данных свойство `madeOrder`. Уберите из функции `submitOrder` вывод диалогового окна и вставьте вместо этого выражение `this.madeOrder = true`. Таким образом, при запуске приложения это свойство будет равно `true`. Приведите код файла `src/components/Form.vue` к следующему виду (листинг 12.5).

Листинг 12.5. Обновление компонента Form: chapter-12/form-update.js

```
...
        dontSendGift: 'Do Not Send As A Gift'
      },
      madeOrder: false
...
  methods: {
    submitForm() {
      this.madeOrder = true;
    }
  }
...
```

Новое свойство madeOrder

Присваиваем madeOrder значение true

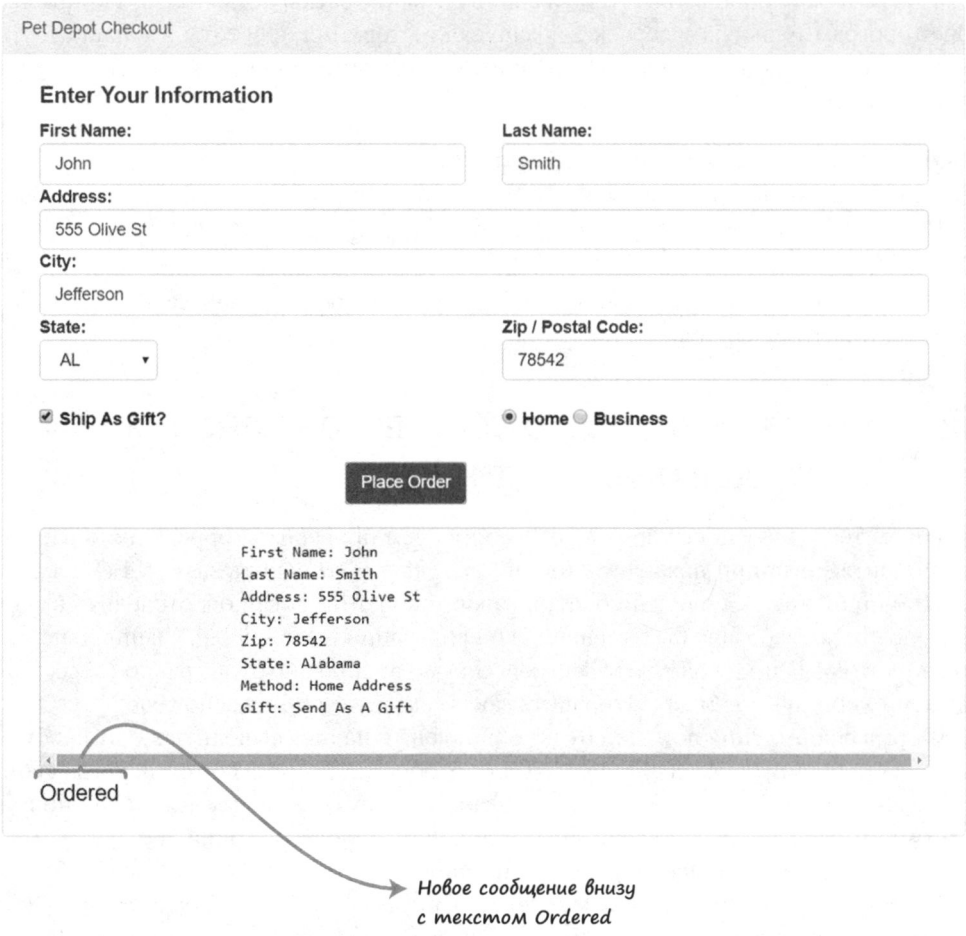


Рис. 12.1. Страница оформления покупки, внизу отображается строка Ordered

Теперь все готово для создания первого тестового сценария. Проверим, имеет ли `madeOrder` значение `true` после нажатия кнопки Place Order (Отправить заказ). Функция `shallow` отрисовывает компонент Vue без учета его потомков. Еще одна часто используемая функция, `mount`, работает по аналогии с `shallow`, но при этом не игнорирует дочерние компоненты.

Еще необходимо импортировать компонент `Form`. Позже мы передадим его в функцию `shallow`, чтобы создать вокруг него обертку. Вы также увидите функцию `describe`, которая позволяет группировать похожие тесты. Работая в командной строке, мы можем следить за успешностью прохождения тестов из определенной группы.

Функция `it` представляет собой тестовый сценарий. Это модульный тест, который проверяет корректность значения `madeOrder` после нажатия кнопки. В каждой группе может быть несколько тестовых сценариев.

Чтобы удостовериться в том, что свойство `madeOrder` равно `true`, воспользуемся библиотекой утверждений Expect. В листинге 12.6 доступ к свойству осуществляется через выражение `wrapper.vm.madeOrder`. Объект `wrapper`, возвращаемый функцией `shallow`, содержит несколько свойств, включая `vm`. С помощью `vm` можно обращаться к любым методам и свойствам экземпляра Vue, то есть запустить любой метод и получить любые значения Vue-компонента. Это удобно.

Объект `wrapper` предоставляет также функцию `find`, которая принимает любой корректный CSS-селектор, например имя тега, идентификатор или класс. Чтобы сгенерировать событие — в данном случае нажатие кнопки, — воспользуйтесь функцией `trigger`. Создайте на основе листинга 12.6 новый файл `Form.spec.js`.

Листинг 12.6. Первый тестовый сценарий: chapter-12/petstore/test/Form.spec.js

```
import { shallow } from '@vue/test-utils'
import Form from '../src/components/Form.vue'

describe('Form.vue', () => {

    it('Check if button press sets madeOrder to true', () => {
      const wrapper = shallow(Form)
      wrapper.find('button').trigger('click')
      expect(wrapper.vm.madeOrder).toBe(true);
    })

})
```

Импорт shallow для использования в тестовом сценарии

Импорт компонента Form

Присваиваем wrapper поверхностную версию компонента

Проверяем, что madeOrder равно true

Находим и затем нажимаем кнопку

Запустите полученный тестовый сценарий. Убедитесь в том, что вы находитесь в корневом каталоге проекта, и выполните команду `npm test`. Это должно запустить тестовую группу. Если увидели ошибку, еще раз проверьте, установлены ли у вас все зависимости, о которых мы говорили ранее, и содержит ли файл `package.json` тестовый скрипт. На рис. 12.2 показан результат успешного прохождения всех тестов.

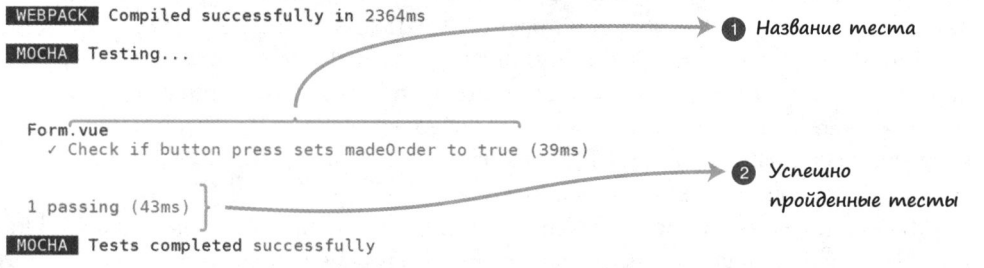


Рис. 12.2. Успешное прохождение первого тестового сценария

Поскольку все прошло как следует, вы увидите сообщение об успехе. Теперь посмотрим, как выглядят проваленные тесты. Вернитесь к файлу `petstore/test/Form.spec.js`, найдите выражение `expect` и поменяйте значение с `true` на `false`. Повторное выполнение команды `npm test` должно завершиться неудачей. На рис. 12.3 показаны ожидаемое и полученное значения.

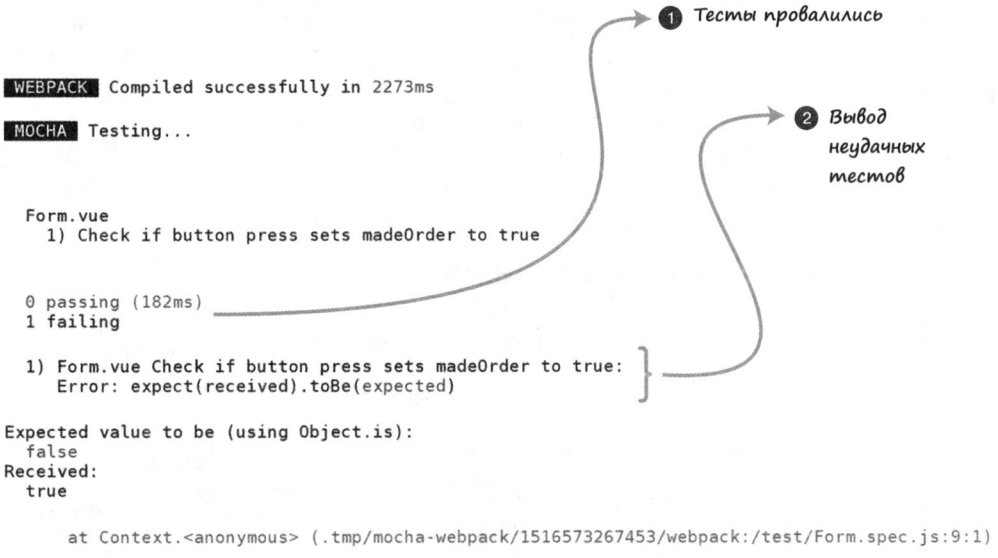


Рис. 12.3. Тестовый сценарий завершился неудачно

Теперь, когда мы познакомились с основами тестирования, протестируем наши компоненты.

12.6. Тестирование компонентов

Прежде чем приступать к тестированию компонентов, необходимо получить общее представление об их технических характеристиках и о том, чего от них следует ожидать. В качестве примера возьмем проект зоомагазина.

Приложение состоит из трех компонентов: `Main`, `Header` и `Form`. `Header` выводит количество элементов в корзине и кнопки входа/выхода. Компонент `Form` отвечает за отображение всех элементов формы и позволяет сделать заказ нажатием кнопки Place Order (Отправить заказ). Компонент `Main` содержит список всех товаров, его содержимое берется из хранилища Firebase.

Мы не станем тестировать каждый компонент, а лишь напишем их спецификации. Это следует сделать перед созданием тестовых сценариев, чтобы понимать, что именно предстоит тестировать.

12.6.1. Тестирование входных параметров

Многие компоненты принимают входные параметры. Например, в зоомагазине компонент `Header` принимает параметр `cartItemCount` и отображает его в правом верхнем углу. Создадим тестовый сценарий, который проверяет факт передачи этого параметра.

Создайте в каталоге petstore/test/ файл `Header.spec.js`. Он будет содержать все тесты для компонента `Header`. Предварительно следует выполнить некоторые подготовительные действия.

В файле `Header.vue` используются Firebase и Vuex. Хук `beforeCreate` вызывает функцию из Firebase и сохраняет полученный сеанс с помощью команды хранилища Vuex. Мы не станем тестировать в этом примере Vuex или Firebase, но подключить их необходимо, иначе получится ошибка. Не забудьте импортировать модули `../src/firebase` и `../src/store/store`, как показано в листинге 12.7.

Вверху файла импортируйте `shallow` из библиотеки vue-test-utils. Заодно импортируйте функцию `createLocalVue` — с ее помощью настраивается Vuex. Далее нужно создать переменную `localVue` и присвоить ей `createLocalVue()`. Эта функция возвращает класс `localVue` — локальную копию Vue. С ее помощью мы подготовим Vuex к тестированию.

В листинге 12.7 опять используется функция `shallow`, но она выглядит немного не так, как в предыдущем тестовом сценарии. Она способна принимать второй опциональный аргумент — объект, содержащий больше информации, чем нужно нашему компоненту. Внутри него можно задать значения для входных параметров, применяя `propsData`, а также `localVue` и `store`.

Чтобы установить входной параметр, нужно ему что-то передать. Проще всего сделать это, создав переменную `cartItemCount`. Мы передадим ее в `propsData`, и она будет инициализирована в компоненте `Header`.

Напоследок следует убедиться в том, что `wrapper.vm.cartItemCount` совпадает с переменной `cartItemCount`. Если они равны, тест проходит успешно. Возьмите код из листинга 12.7 и скопируйте его в файл `petstore/test/Header.spec.js`.

Листинг 12.7. Тестирование входного параметра: chapter-12/header-prop-test.js

```
import { shallow, createLocalVue } from '@vue/test-utils';
import Header from '../src/components/Header.vue';
import Vuex from 'vuex';
import '../src/firebase';
import { store } from '../src/store/store';

const localVue = createLocalVue();
localVue.use(Vuex)

describe('Header.vue', () => {

  it('Check prop was sent over correctly to Header', () => {
    const cartItemCount = 10;
    const wrapper = shallow(Header, {
```

Подключение Vuex к тестовому сценарию

Подключение Firebase к тестовому сценарию

Подключение хранилища Vuex к тестовому сценарию

У новой константы wrapper есть второй аргумент

```
      store, localVue, propsData: { cartItemCount }
    })
    expect(wrapper.vm.cartItemCount).toBe(cartItemCount);
  })
});
```

Входному параметру присваивается cartItemCount

Expect проверяет, совпадает ли cartItemCount с переданным параметром

Мы научились проверять входные параметры. Теперь займемся текстом.

12.6.2. Тестирование текста

Иногда необходимо проверить, выводится ли текст на каком-либо участке компонента. Неважно, в каком элементе он отображается, — нас интересует лишь сам факт отрисовки.

При написании тестов имейте в виду, что каждый тестовый сценарий должен проверять что-то одно. Легко создать несколько утверждений для проверки текста в одном файле, но лучше разнести их по отдельным сценариям. Последуем этому правилу и создадим в нашем сценарии одно утверждение.

Откройте файл `petstore/test/Header.spec.js` и добавьте в него новый тестовый сценарий. В предыдущем примере мы проверяли корректность передачи входного параметра `cartItemCount` в компоненте `Header`. Теперь хотим выяснить, выводится ли содержимое этого параметра в теге `span` того же компонента.

Создадим еще одну обертку, как делали ранее. На этот раз воспользуемся функцией `wrapper.find`, чтобы найти `span`. Затем мы сможем извлечь содержимое `span` с помощью функции `text()` и проверить, совпадает ли оно с `cartItemCount`, задействуя функцию `toContain`. Скопируйте код листинга 12.8 и вставьте его в файл `pet/test/Header.spec.js` сразу после предыдущего тестового сценария.

Листинг 12.8. Тестирование текста: chapter-12/header-text-test.js

```
it('Check cartItemCount text is properly displayed', () => {
  const cartItemCount = 10;
  const wrapper = shallow(Header, {
    store, localVue, propsData: { cartItemCount }
  })
  const p = wrapper.find('span');
  expect(p.text()).toContain(cartItemCount)
})
```

Обертка находит тег span

Утверждение проверяет, совпадает ли текст с cartItemCount

12.6.3. Тестирование CSS-классов

При тестировании можно использовать метод `classes`, который возвращает массив классов, закрепленных за элементом. Добавим простую проверку, чтобы убедиться в корректности класса в одном из элементов `div`.

Добавьте новый тестовый сценарий в файл `petstore/test/Header.spec.js`. Мы создадим еще одну обертку, но на этот раз вызовем из нее функцию `findAll`,

которая вернет все элементы `div` в компоненте. Чтобы извлечь первый элемент, можно воспользоваться методом `at(0)`. А после этого применить выражение `expect` к `p.classes()`, чтобы получить все классы, закрепленные за первым тегом `div`. При обнаружении нужного класса функция `toContain` вернет `true`.

Если заглянуть в файл `Header.vue`, можно заметить, что за первым элементом `div` закреплены классы `navbar` и `navbar-default`. Поскольку мы ищем `navbar`, тест пройдет успешно (листинг 12.9).

Листинг 12.9. Тестирование классов: chapter-12/header-classes-test.js

```
it('Check if navbar class is added to first div', () => {
    const cartItemCount = 10;
    const wrapper = shallow(Header, {
      store, localVue, propsData: { cartItemCount }
    })
    const p = wrapper.findAll('div').at(0);
    expect(p.classes()).toContain('navbar');
})
```

Ищем все теги div и возвращаем первый из них

Проверяем присутствие navbar среди закрепленных классов

Прежде чем двигаться дальше, выполните в терминале команду `npm test` и убедитесь в том, что все тесты проходят успешно (рис. 12.4). Если что-то пошло не так, еще раз проверьте выражения `expect` и корректность подключения всех модулей вверху файла.

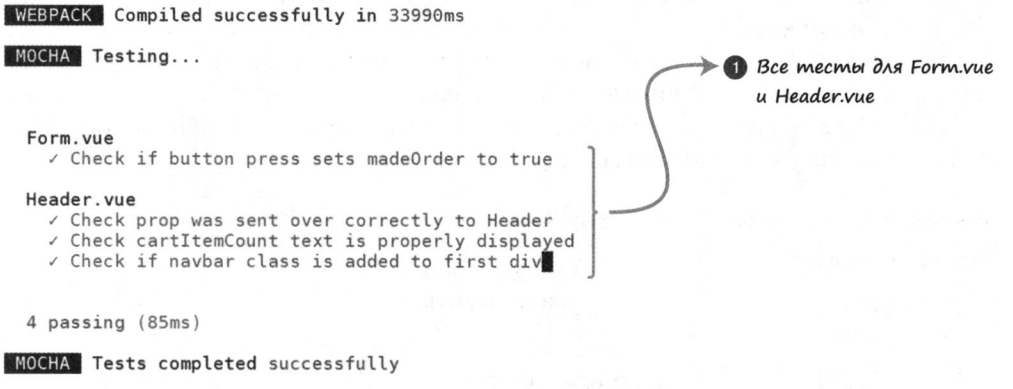


Рис. 12.4. Все тесты проходят успешно

Все тесты успешны. Теперь перейдем к Vuex.

12.6.4. Симуляция Vuex

Vuex служит центральным хранилищем, которое содержит данные всего приложения. В нашем зоомагазине оно используется для хранения информации о сеансе и товарах. Его тоже лучше протестировать.

ПРИМЕЧАНИЕ

Тестирование Vuex отличается повышенной сложностью и имеет множество аспектов. К сожалению, мы не станем рассматривать их здесь в полном объеме. Чтобы узнать больше по этой теме, начните с официального руководства по тестированию Vuex — vue-test-utils.vuejs.org/ru/guides/using-with-vuex.html.

Наши тестовые сценарии будут проверять компонент `Header` и его поведение при разных значениях `session` — `true` и `false`. Мы хотим убедиться в том, что, если сеанс существует, выводится кнопка Sign Out (Выйти), а если нет — Sign In (Войти).

Ранее в этой главе мы импортировали хранилище прямо в тестовом файле. Это было временное решение, которое позволило создавать другие тестовые сценарии для компонента `Header`. Для тестирования Vuex это не годится. Нам придется полностью симулировать хранилище. Но не беспокойтесь — это намного проще, чем может показаться.

Импорт хранилища происходит вверху файла `petstore/test/Header.spec.js`. Удалите эту строчку. Мы создадим *фиктивное* хранилище. Объект будет иметь ту же структуру, что и оригинал, но, в отличие от последнего, мы сможем применять его в тестах, так как контролируем его реализацию. Создайте переменные `store`, `getters` и `mutations` снизу от выражения `describe`, как показано в листинге 12.10. Затем — функцию `beforeEach`. Код внутри `beforeEach` выполняется перед каждым тестовым сценарием. Это подходящее место для подготовительных действий.

Чтобы не усложнять пример, сделаем хранилище очень простым. У нас будут геттер для `session`, который возвращает `false`, и мутация, возвращающая пустой объект. Для создания хранилища можно использовать выражение `new Vuex.Store` (убедитесь в том, что `Store` начинается с прописной S). Скопируйте код листинга 12.10 в верхнюю часть файла `petstore/test/Header.spec.js`.

Листинг 12.10. Симуляция Vuex: chapter-12/header-vuex-mock.js

```
describe('Header.vue', () => {
  let store;
  let getters;
  let mutations;
  beforeEach(() => {
    getters = {
      session: () => false
    }
    mutations = {
      SET_SESSION: () => {}
    }
    store = new Vuex.Store({
      getters,
      mutations
    })
})
```

Переменные store, getters и mutations

Выполняется перед каждым тестом

Геттер session возвращает false

Мутация SET_SESSION возвращает пустой объект

Создается новое хранилище

Создав симуляцию хранилища Vuex, используем ее в тестовых сценариях. Мы исходим из того, что, если `session` равно `false`, на экране выводится кнопка Sign In (Войти). Если вам это кажется немного запутанным, откройте файл `Header.vue` в папке src, там вы увидите, что директива `v-if` зависит от вычисляемого свойства `mySession`. Если `mySession` равно `false`, выводится кнопка Sign In (Войти). Если оно равно `true`, директива `v-else` отобразит кнопку Sign Out (Выйти). Скопируйте код из листинга 12.11 в файл `petstore/test/Header.js`.

Листинг 12.11. Тестирование кнопки входа: chapter-12/header-signin-test.js

```
it('Check if Sign in button text is correct for sign in', () => {
  const cartItemCount = 10;
  const wrapper = shallow(Header, {
    store, localVue, propsData: { cartItemCount }
  })

  expect(wrapper.findAll('button').at(0).text()).toBe("Sign In");

})
```

Утверждение expect удостоверяется в том, что текст на кнопке — Sign In

В то же время мы должны проверить, выводится ли кнопка Sign Out (Выйти) в случае существования сеанса. Это можно сделать несколькими способами, но проще всего создать хранилище с новым геттером `session` и затем добавить его при создании новой обертки. В результате компонент `Header` будет вести себя так, будто `session` равно `true`, а не `false`. Скопируйте содержимое листинга 12.12 и вставьте его в файл `petstore/test/Header.spec.js` в качестве еще одного тестового сценария.

Листинг 12.12. Тестирование кнопки выхода: chapter-12/header-signout-test.js

```
it('Check if Sign in button text is correct for sign out', () => {
  const cartItemCount = 10;
  getters.session = () => true;
  store = new Vuex.Store({ getters, mutations})
  const wrapper = shallow(Header, {
    store, localVue, propsData: { cartItemCount }
  })
  expect(wrapper.findAll('button').at(0).text()).toBe("Sign Out");
})
```

Проверяет, равен ли текст кнопки Sign Out

Запустите тесты. Все они должны пройти успешно. На этом мы заканчиваем тестирование зоомагазина. Если хотите поупражняться, можете добавить еще несколько тестовых случаев для компонентов `Form` и `Main`.

12.7. Настройка отладчика Chrome

Отладка тестов часто сводится к использованию метода `console.log`, который выводит значения переменных во время выполнения кода. Это неплохой подход, но есть кое-что получше. Упростить задачу поможет отладчик Chrome.

Внутри тестовых сценариев можно применять ключевое слово `debugger`. Оно останавливает выполнение кода на любом участке программы. Для его работы необходимо сочетание браузера Chrome и утилиты *Node Instector*. Последняя входит в состав Node.js, начиная с версии 8.4.0, и помогает с отладкой внутри Chrome. Чтобы запустить код с помощью Node Instector, необходимо выполнить следующую команду. Введите ее в терминале или добавьте в раздел `scripts` файла `package.json`, как показано в листинге 12.13.

Листинг 12.13. Добавление команды inspect в файл package.json: chapter-12/petstore/package.json

```
...
  "private": true,
  "scripts": {
...
    "inspect": "node --inspect --inspect-brk node_modules/mochawebpack/
     bin/mocha-webpack --webpack-config build/webpack.base.conf.js -
     require test/setup.js   test/**/*.spec.js"
...
```

Скриптовая команда для запуска Node Instector

Чтобы выполнить эту команду, введите в терминале `npm run inspect`. В результате будет запущена Node Instector. Как вариант, можете ввести следующую команду:

```
$ node --inspect --inspect-brk node_modules/mocha-webpack/bin/mocha-webpack
--webpack-config build/webpack.base.conf.js --require test/setup.js
test/**/*.spec.js
```

В обоих случаях на локальном сервере 127.0.0.1 будет запущен отладчик. Далее показан вывод, который вы должны увидеть:

```
Debugger listening on ws://127.0.0.1:9229/71ba3e86-8c3c-410f-a480-ae3098220b59
For help see https://nodejs.org/en/docs/inspector
```

Откройте браузер Chrome и введите в адресной строке `chrome://inspect`. На экране появится страница Devices (Устройства) (рис. 12.5).

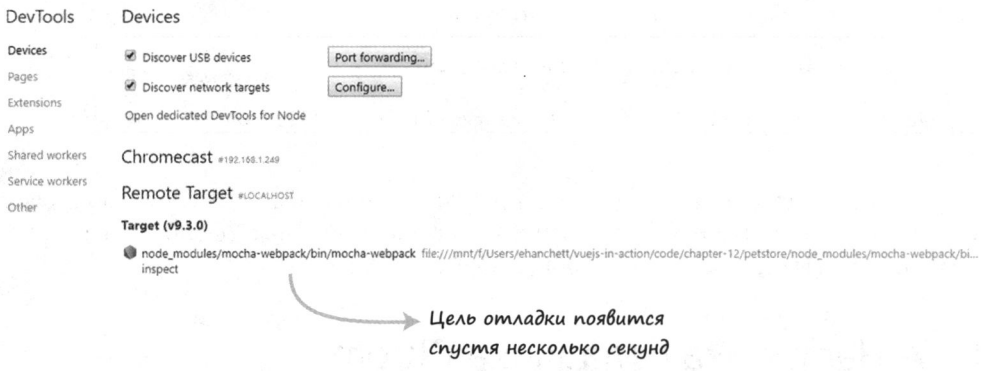


Рис. 12.5. Страница Devices (Устройства) в Chrome

Спустя несколько секунд внизу должна появиться цель отладки со ссылкой Inspect (Проинспектировать). При щелчке на этой ссылке откроется отдельное окно

с отладчиком в приостановленном состоянии. Щелкните на стрелочке, чтобы начать отладку (рис. 12.6).

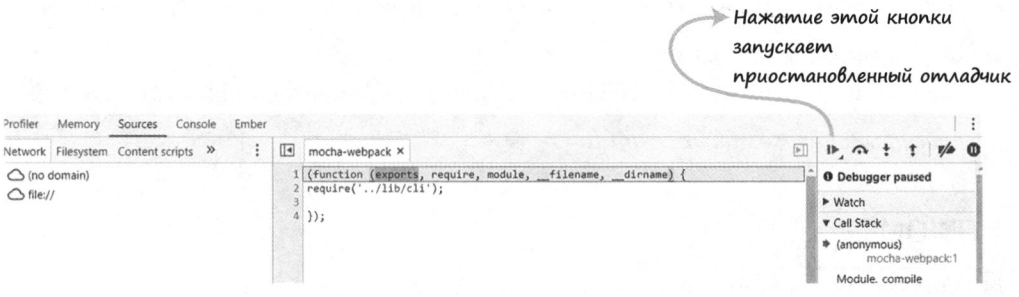


Рис. 12.6. Окно инспектора в Chrome

После запуска отладчик остановится на том участке кода, в который мы добавили выражение `debugger`. После этого можно открыть консоль и просмотреть переменные (рис. 12.7). Например, щелкнув на `wrapper`, `__proto__` и еще раз `__proto__`, вы получите список всех методов объекта `wrapper`.

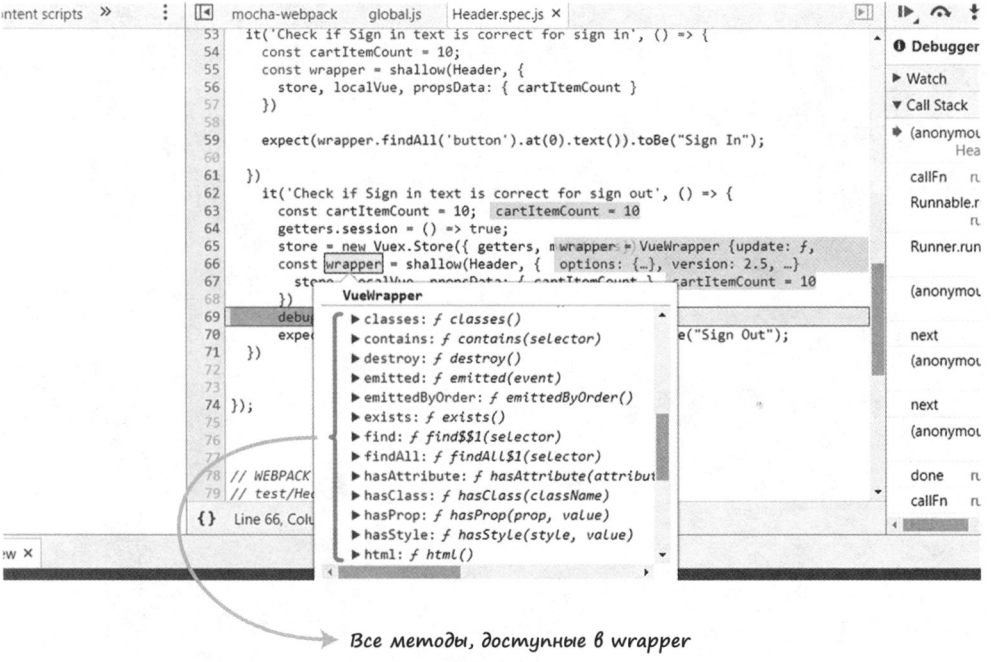


Рис. 12.7. Выражение debugger выводит все методы объекта wrapper

Если вам нужно понять принцип работы теста, но вы не уверены, для чего нужны те или иные переменные, используйте Chrome Inspector.

Упражнение

Используя знания, приобретенные в этой главе, ответьте на следующие вопросы.

- Почему так важно тестировать свой код?
- Какой инструмент может помочь с тестированием приложений на основе Vue.js?

Ответы ищите в приложении Б.

Резюме

- Модульные тесты проверяют работу небольших участков кода.
- С помощью тестов можно проверить свои функции и убедиться в том, что они работают так, как ожидалось.
- Можно отлаживать тестовые сценарии в режиме реального времени, используя браузер Chrome.

Приложения

A Настройка среды разработки

Разработка без использования подходящих инструментов подобна исследованию пещеры без фонаря: выполнимо, но все это время вы проведете в темноте. Тем не менее, если у вас уже установлены инструменты, рассмотренные в данном разделе, или их альтернативы, можете пролистнуть эти страницы.

A.1. Chrome Developer Tools

Нашим лучшим товарищем в путешествии станет набор инструментов Chrome Developer Tools. Установите браузер Chrome, если еще не сделали этого. Его можно найти по адресу www.google.com/chrome/. Чтобы открыть Developer Tools, воспользуйтесь меню View ▸ Developer ▸ Developer Tools (Вид ▸ Разработка ▸ Developer Tools) или щелкните правой кнопкой мыши на странице и выберите пункт Inspect (Просмотр кода элемента) (рис. A.1).

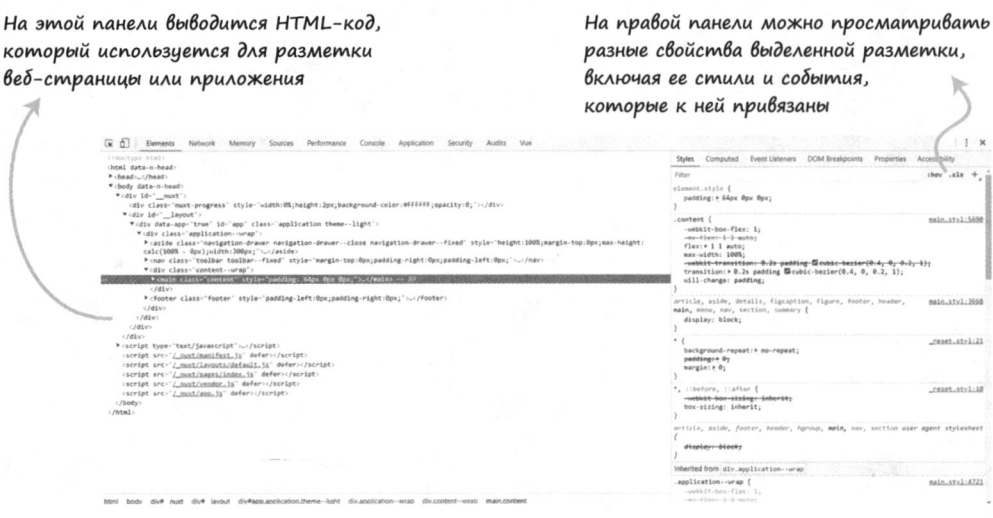


Рис. A.1. По умолчанию панель Developer Tools в Chrome выводит HTML-разметку веб-страницы и CSS-стили, закрепленные за выбранным элементом

Чаще всего при просмотре кода в Developer Tools вы будете применять консоль JavaScript. Можете перейти в нее, воспользовавшись вкладкой Console (Консоль) или напрямую выбрав пункт меню View ▸ Developer ▸ JavaScript Console (Вид ▸ Разработка ▸ Консоль JavaScript).

Консоль можно открыть и работая с любой другой вкладкой в Developer Tools. Для этого достаточно нажать кнопку Escape (рис. А.2). Это, к примеру, позволит вам просматривать HTML одновременно с выполнением JavaScript.

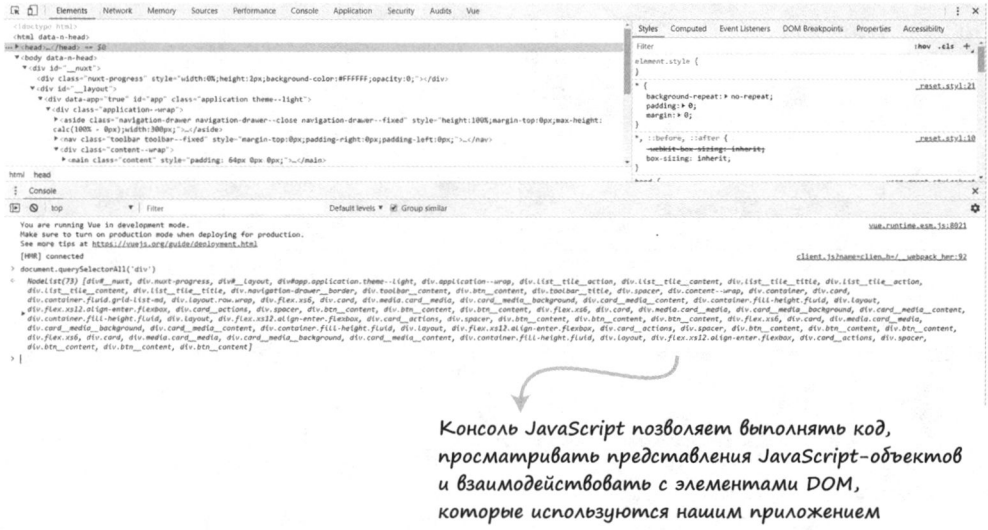


Рис. А.2. Консоль JavaScript позволяет просматривать HTML-разметку и взаимодействовать с ней, не отвлекаясь от выполнения кода Vue-приложения

A.2. vue-devtools для Chrome

Специально для просмотра Vue-приложений в процессе их выполнения в Chrome основная команда разработчиков Vue создала расширение под названием vue-devtools. Вы можете установить его из Chrome Web Store, посетив страницу mng.bz/RpuC. Если любите приключения, клонируйте GitHub-репозиторий, находящийся по адресу github.com/vuejs/vue-devtools, и соберите это расширение самостоятельно или даже модифицируйте его код.

ПОСЛЕ УСТАНОВКИ

Если установить расширение и открыть панель Developer Tools, вкладки Vue может не оказаться на месте. Chrome ведет себя немного капризно в этой ситуации. Попробуйте открыть новую вкладку, а если это не помогло, перезапустите браузер.

После установки расширения вам нужно открыть ему доступ к локальным файлам, поскольку в первых нескольких главах мы не будем использовать веб-сервер. Выберите в меню Window ▸ Extensions (Окно ▸ Расширения) в Chrome и найдите пункт Vue.js devtools. Установите флажок Allow access to file URLs (Открыть доступ к файловой системе), как показано на рис. А.3. Этого будет достаточно.

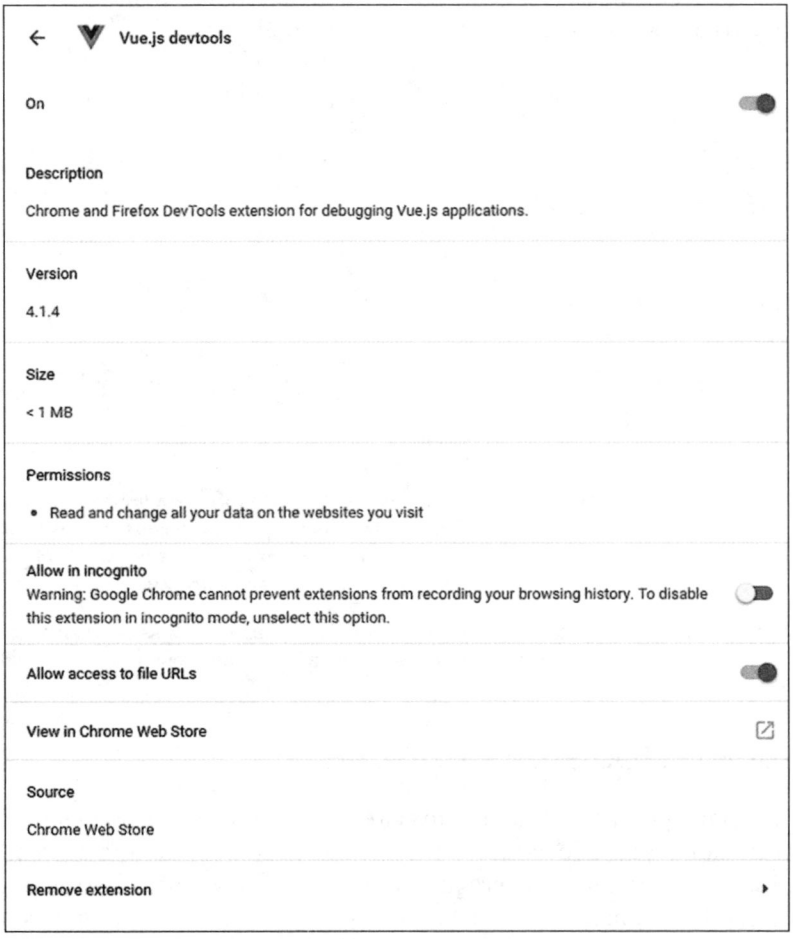


Рис. А.3. Чтобы открыть доступ к локальным файлам в vue-devtools, нужно изменить параметры на странице настроек расширения

После установки расширения вы сможете отслеживать данные, используемые приложением, изолировать его отдельные компоненты и даже перемещаться во времени! В самом простом случае это позволит возвращаться назад и воспроизводить события, уже произошедшие в приложении. Все элементы расширения показаны на рис. А.4.

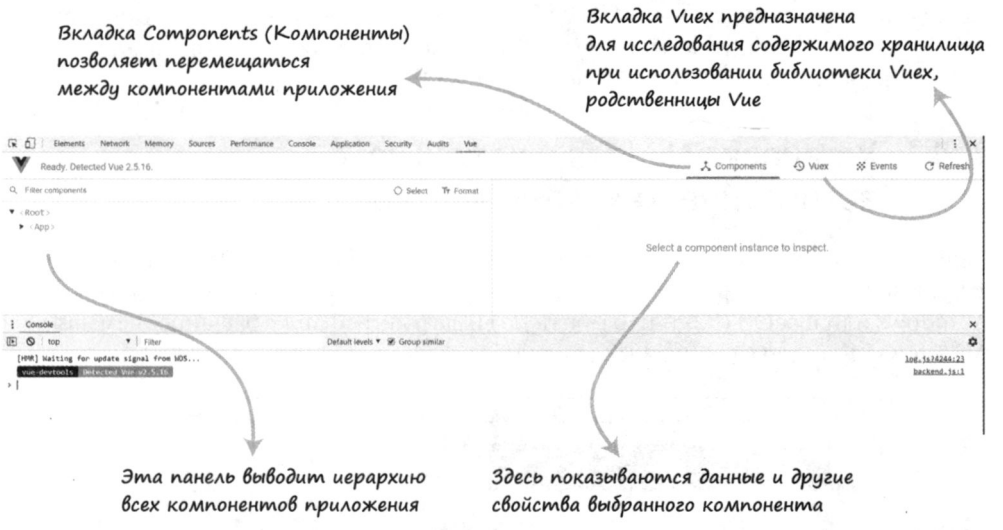


Рис. А.4. Расширение vue-devtools позволяет исследовать Vue-приложения в режиме реального времени

А.3. Загрузка сопутствующего кода для отдельных глав

Исходный код примеров для книги доступен для загрузки на веб-сайте издательства (www.manning.com/books/vue-js-in-action). Код для отдельных глав находится на странице GitHub по адресу github.com/ErikCH/VuejsInActionCode. Там же размещены все иллюстрации.

А.4. Установка Node.js и npm

Чтобы использовать утилиту Vue-CLI и получить доступ к сотням тысяч готовых модулей, понадобится Node.js. Рекомендую устанавливать либо текущую версию Node, либо последний *релиз с долгосрочной поддержкой* (Long Term Support, LTS). Нам подойдут оба варианта.

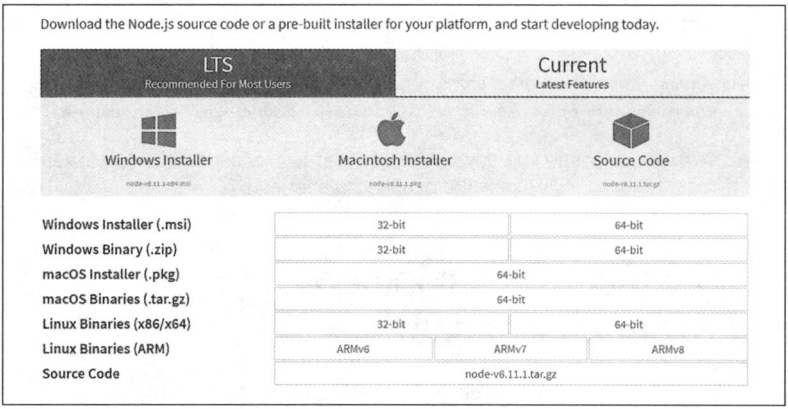
Далее перечислены несколько вариантов получения пакета Node, включающего в себя npm.

- *Homebrew или MacPorts*. Это популярный выбор пользователей macOS.
- *Автоматические установщики*. Доступны в Windows и macOS.
- *Установка с помощью системы управления пакетами в Linux*. Для установки Node в среде Linux можно применять yum, apt-get или pacman.

- *Установка с помощью NVM.* NVM (Node Version Manager — диспетчер версий Node) — это скрипт, который помогает управлять версиями Node.js. Это отличный вариант, доступный в Windows и macOS.

A.4.1. Установка Node.js с помощью автоматических установщиков

Сейчас самый простой способ загрузки Node.js — использование автоматического установщика. Откройте страницу nodejs.org/en/download, выберите свою версию Windows или macOS (32- или 64-битную) и загрузите файл с расширением .msi (для Windows) или .pkg (для macOS) (рис. A.5).



A.5. Страница для загрузки Node

A.4.2. Установка Node.js с помощью NVM

Отличный выбор также NVM. Это скрипт, который помогает управлять несколькими активными версиями Node.js. Для установки даже не нужно заходить на вебсайт. NVM изолирует каждую версию Node, которую вы загружаете. Я советую этот способ большинству новичков, хотя вам все равно потребуются навыки владения командной строкой. NVM можно найти на странице github.com/creationix/nvm. Версия для Windows доступна по адресу github.com/coreybutler/nvm-windows/releases.

Чтобы загрузить и установить последнюю версию NVM для macOS, откройте терминал и выполните следующую команду:

```
$ curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.33.2/
install.sh | bash
```

Для установки NVM в системе Windows щелкните на файле `nvm-setup.zip` на странице nvm-windows/releases. Распакуйте полученный архив и запустите программу `nvm-setup.exe`.

После установки NVM в Windows или macOS выполните следующую команду, чтобы загрузить последнюю версию Node.js:

```
$ nvm install node
```

Вот и все! Теперь в вашей системе установлены Node и npm!

А.4.3. Установка Node.js с помощью системы управления пакетами в Linux

Пакеты Node.js присутствуют в репозиториях всех основных дистрибутивов Linux. Например, в Ubuntu можно использовать apt-get:

```
$ sudo apt-get install nodejs
```

В Fedora можно воспользоваться yum:

```
$ yum install nodejs
```

Чтобы узнать, как устанавливаются пакеты в вашей системе, нужно свериться с документацией дистрибутива. Имейте в виду, что некоторые дистрибутивы могут предлагать для загрузки устаревшие версии Node.js. В таких случаях лучше воспользоваться NVM или загрузить Node.js с официального веб-сайта.

А.4.4. Установка Node.js с помощью MacPorts или Homebrew

MacPorts и Homebrew — это системы управления пакетами для macOS. Вам следует установить одну из них, прежде чем приступать к загрузке Node.js. Последние актуальные инструкции по установке Homebrew и Macports вы найдете на сайтах brew.sh и www.macports.org соответственно.

Установив один из этих диспетчеров пакетов на свой компьютер, можете загрузить Node с помощью единственной команды.

Для Homebrew это:

```
$ brew install node
```

а для MacPorts —

```
$ sudo port install nodejs
```

Этого должно быть достаточно!

А.4.5. Проверяем, установлен ли пакет Node

Чтобы убедиться в успешной установке Node.js, воспользуйтесь ключом `-v`:

```
$ node -v
$ npm -v
```

Эти команды выведут текущие версии установленных пакетов Node и NPM. На момент написания этих строк текущей была версия 8.2.1, а последним релизом с долгосрочной поддержкой был 6.11.

A.5. Установка Vue-CLI

Перед установкой Vue-CLI убедитесь в том, что в вашей системе уже есть Node.js как минимум версии 4.6 (желательно 6.x), npm версии 3 или выше и Git. В главе 11 мы используем Nuxt.js, в связи с чем вам понадобится Node.js 8.0 или более. Vue-CLI будет работать в любой из этих конфигураций. Установите Node, следуя приведенным ранее инструкциям. Инструкции по установке Git находятся на официальном сайте mng.bz/D7zz.

Установив все необходимые пакеты, откройте терминал и выполните следующую команду:

```
$ npm install -g vue-cli
```

Vue-CLI имеет простой интерфейс командной строки. Чтобы создать проект, введите `vue-cli init имя_шаблона` и укажите подходящее название, например:

```
$ vue init <имя_шаблона> <имя_проекта>
$ vue init webpack my-project
```

И, похоже, все.

Стоит отметить, что на момент написания книги последней версией Vue-CLI была 2.9.2. Версия Vue-CLI 3.0 все еще находится на стадии бета-тестирования. Подробнее об установке и работе с Vue-CLI 3.0 можно почитать в официальном руководстве по адресу mng.bz/5t1C.

Б Ответы на вопросы

Далее перечислены ответы на вопросы из глав 2–12.

Глава 2

В разделе 2.4 мы создали фильтр для цены. Какие еще фильтры могли бы нам пригодиться?

В Vue.js фильтры обычно используются для обработки текста. Например, в виде фильтра можно оформить перевод названия товара в верхний регистр.

Глава 3

Какая разница между методами и вычисляемыми свойствами?

Вычисляемые свойства подходят для выведения значений. Значение автоматически обновляется при изменении любого из исходных параметров. Его также кэшируют, чтобы избежать повторяющихся вычислений, которые не приводят ни к каким изменениям, — например, в цикле. Методы представляют собой функции, привязанные к экземпляру Vue. Они выполняются лишь в случае явного вызова. В отличие от вычисляемых свойств методы могут принимать аргументы. Они уместны в тех же ситуациях, в которых вы могли бы применять обычные функции на языке JavaScript. Приложение, не поддерживающее полноценное взаимодействие с пользователем, нельзя назвать эффективным.

Глава 4

Как работает двунаправленное связывание данных? В каких случаях его следует использовать?

Говоря просто, двунаправленное связывание данных — это когда обновление модели изменяет представление, а обновление представления изменяет модель. Этот механизм следует задействовать, работая с формами и полями ввода.

Глава 5

Что такое диапазон v-for и чем он отличается от обычной директивы v-for?

Директива `v-for` используется для отрисовки списков, основанных на массивах. Обычно она имеет вид `item in items`, где `items` — исходный массив, а `item` — ссылка на его текущий элемент. `v-for` можно применять также в качестве диапазона в формате `item in (число)`. В этом случае шаблон будет выведен заданное количество раз.

Глава 6

Как передать информацию из родительского компонента в дочерний? Какие средства применяются для передачи информации обратно из дочернего компонента в родительский?

Обычно для передачи информации из родительского компонента в дочерний задействуют входные параметры. Они должны быть явно заданы внутри потомка. Чтобы передать данные из дочернего компонента в родительский, можно использовать выражение `$emit(имяСобытия)`. Стоит упомянуть, что позже мы рассмотрим другие способы обмена информацией между компонентами, включая применение хранилища данных.

Глава 7

Назовите два способа навигации между разными маршрутами.

Переходить между маршрутами можно двумя способами: добавить в шаблон элемент `router-link` или воспользоваться вызовом `this.$router.push` внутри экземпляра Vue.

Глава 8

В чем разница между анимацией и переходом?

При переходе начальное состояние заменяется конечным, а анимация может включать множество состояний.

Глава 9

Что такое примесь и в каких ситуациях ее следует применять?

Примеси позволяют распределить между компонентами многократно используемую функциональность. Их следует применять в ситуациях, когда в разных компонентах повторяется один и тот же код. Повторение кода противоречит принципу DRY (don't repeat yourself — «не повторяйся») и подлежит рефакторингу. Примеси «примешиваются» непосредственно к параметрам компонента.

Глава 10

Перечислите несколько преимуществ Vuex по сравнению со стандартной передачей данных в приложениях Vue.js.

Vuex размещает состояние приложения в централизованном хранилище. Благодаря этому изменение состояния предсказуемо и происходит синхронно. Это помогает избежать нежелательных последствий. Крупные Vue-приложения могут стать громоздкими, поэтому передачу информации или использование шины событий нельзя назвать идеальными решениями. Vuex помогает абстрагироваться от этой проблемы, предоставляя единое центральное хранилище для всех ваших данных.

Глава 11

Каким преимуществом обладает метод asycData, доступный в Nuxt-приложениях, по сравнению с применением промежуточных обработчиков?

Метод `asyncData` выполняется на стороне сервера перед загрузкой страницы и имеет доступ к объекту `context`. Его преимущество, по сравнению с промежуточными обработчиками, состоит в том, что возвращаемый им результат объединяется с объектом `data` на клиентской стороне. В промежуточных обработчиках иногда приходится использовать хранилище Vuex, чтобы данные, помещенные в него, можно было получить внутри компонента.

Глава 12

Почему так важно тестировать свой код? Какой инструмент может помочь с тестированием приложений на основе Vue.js?

Тестирование должно быть неотъемлемым аспектом написания кода. По сравнению с ручным тестированием автоматические тесты работают намного быстрее и меньше подвержены ошибкам. Их создание требует дополнительных усилий на начальных этапах разработки, но в долгосрочной перспективе они сэкономят вам время. Vue.js предлагает множество инструментов для упрощения тестирования, один из самых важных — библиотека vue-test-utils. Она поможет в создании подходящих тестов для Vue-приложений.

Шпаргалка

Определение компонента

```
Vue.component('my-component', {
  props: {        Параметры, которые принимает компонент
    myMessage: String,
    product: Object,
    email: {
       type: String,
       required: true,
       default: "test@test.com"
       validator: function (value) {
            Возвращает true или false
       }
    }
  },
  data: function() {    Это должна быть функция
    return {
       firstName: 'Vue' ,
       lastName: 'Info'
    }
  },
  methods: { … }
  computed: (    Возвращает закэшированные значения, пока
    fullName: function () {    не поменяются зависимости
      return this.firstName + ' ' + this.lastName
    }
  }
  watch: {    Вызывается при изменении значения firstName
    firstName: function (value, oldValue) {
.. }
  },
  components: { Компоненты, которые можно использовать в шаблоне
    ButtonCómponent, IconComponent
  },
  template: '<span>{{ myMessage }}</span>',
})      Для многострочного текста можно применять обратные кавычки
```

Пользовательские события

Для передачи данных из родительских компонентов в дочерние применяйте входные параметры и соответственно пользовательские свойства.

Установка обработчика событий в рамках родителя:

```
<button-counter
v-on:incrementThis="incVal">
```

Внутри родительского компонента:

```
methods: (
   incVal: function (toAdd) { . . . }
}
```

Внутри кнопки-счетчика:

```
this.$emit('incrementThis', 5)   Данные, отправленные родителю
```

Имя пользовательского события

Хуки жизненного цикла

```
beforeCreate     beforeUpdate
created          updated
beforeMount      beforeDestroy
mounted          destroyed
```

Использование одиночного слота

Шаблон компонента:

```
<div>
   <h2>Заголовок</h2>
      <slot>
        // Отображается, только если нет
        // контента
   </slot>
</div>
```

Подстановка данных в компонент:

```
<my-component>
   <p>This will go in the slot</p>
</my-component>
```

Несколько слотов

Шаблон компонента:

```
<div class="container">
   <header>
      <slot name="header"></slot>
   </header>
   <main>
      <slot>Контент по умолчанию</slot>
   </main>
   <footer>
      <slot name="footer"></slot>
   </footer>
</div>
```

Подстановка данных в компонент:

```
<app-layout>
<h1 slot="header">Название страницы</h1>
<p>основной контент.</p>
<p slot="footer">Контакты</p>
</app-layout>
```

Выражения

```
<div id="app">
  <p>I have a {{ widget }}</p>
  <p>{{ widget + 's' }}</p>
  <p>{{ isWorking ? 'YES' : 'NO' }}</p>
  <p>{{ widget.getSalePrice() }}</p>
</div>
```

Директивы

Добавление/удаление элемента в зависимости от истинности:

```
<p v-if="inStock">{{ widget }}</p>
<p v-else-if="onOrder">. . .</p>
<h1 v-else>. . .</h1>
```

Использует CSS-свойство элемента `display`:

```
<h1 v-show="ok">Hello World!</h1>
```

Двунаправленное связывание данных:

```
<input v-model="firstName" />
```

```
v-model.lazy="..."     Синхронизирует ввод после изменения
v-model.number="..."   Всегда возвращает число
v-model.trim="..."     Убирает пробельные символы
```

Отрисовка списков

```
<li v-for="item in items" :key="item.id">
  {{ item }}
</li>
```

Желательно использовать key

Для доступа к индексу массива:

```
<li v-for="(item, index) in items">. . .
```

Для перебора объектов:

```
<li v-for="(value, key) in object">. . .
```

Использование `v-for` в сочетании с компонентом:

```
<my-item v-for="item in products"
  :products="item" :key="item.id">
```

Связывание

```
<a v-bind:href="url">. . .</a>
```

Сокращенно

```
<a :href="url">...</a>
```

Значения `true` или `false` добавляют или удаляют атрибут:

```
<button :disabled="isButtonDisabled">...
```

Если `isActive` истинно, появится класс `active`:

```
<div :class="{ active: isActive }">. . .
```

Стилю `color` присваивается значение `activeColor`:

```
<div :style="{ color: activeColor }">
```

Действия/события

Вызывает из компонента метод `add` в ответ на щелчок:

```
<button v-on:click="add">. . .
```

Сокращенно

```
<button @click="add">. . .
```

Можно передавать аргументы:

```
<button @click="add(widget)">. . .
```

Чтобы предотвратить перезагрузку страницы...

```
<form @submit.prevent="add">. . .
```

...Срабатывает лишь один раз:

```
<img @mouseover.once="show">. . .
```

```
.stop    Остановить распространение события
.self    Срабатывает, если в event.target указан сам элемент
```

Пример ввода с клавиатуры:

```
<input @keyup.enter="submit">
```

Эрик Хэнчетт, Бенджамин Листуон

Vue.js в действии

Перевел с английского С. Черников

Заведующая редакцией	*Ю. Сергиенко*
Руководитель проекта	*О. Сивченко*
Ведущий редактор	*Н. Гринчик*
Литературный редактор	*Н. Рощина*
Художественный редактор	*С. Заматевская*
Корректор	*Е. Павлович*
Верстка	*Г. Блинов*

Изготовлено в России. Изготовитель: ООО «Прогресс книга».
Место нахождения и фактический адрес: 194044, Россия, г. Санкт-Петербург,
Б. Сампсониевский пр., д. 29А, пом. 52. Тел.: +78127037373.

Дата изготовления: 03.2019. Наименование: книжная продукция. Срок годности: не ограничен.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12 —
Книги печатные профессиональные, технические и научные.

Импортер в Беларусь: ООО «ПИТЕР М», 220020, РБ, г. Минск, ул. Тимирязева, д. 121/3, к. 214, тел./факс: 208 80 01.

Подписано в печать 26.02.19. Формат 70×100/16. Бумага офсетная. Усл. п. л. 24,510. Тираж 1000. Заказ Р-407.

Отпечатано в типографии ООО «ИНФО СИСТЕМ». 420044, г. Казань, пр. Ямашева, д. 36Б.

ИЗДАТЕЛЬСКИЙ ДОМ ПИТЕР®
WWW.PITER.COM

ИЗДАТЕЛЬСКИЙ ДОМ «ПИТЕР» предлагает профессиональную, популярную и детскую развивающую литературу

Заказать книги оптом можно в наших представительствах

РОССИЯ

Санкт-Петербург: м. «Выборгская», Б. Сампсониевский пр., д. 29а
тел./факс: (812) 703-73-83, 703-73-72; e-mail: sales@piter.com

Москва: м. «Электрозаводская», Семеновская наб., д. 2/1, стр. 1, 6 этаж
тел./факс: (495) 234-38-15; e-mail: sales@msk.piter.com

Воронеж: тел.: 8 951 861-72-70; e-mail: hitsenko@piter.com

Екатеринбург: ул. Толедова, д. 43а; тел./факс: (343) 378-98-41, 378-98-42;
e-mail: office@ekat.piter.com; skype: ekat.manager2

Нижний Новгород: тел.: 8 930 712-75-13; e-mail: yashny@yandex.ru; skype: yashny1

Ростов-на-Дону: ул. Ульяновская, д. 26
тел./факс: (863) 269-91-22, 269-91-30; e-mail: piter-ug@rostov.piter.com

Самара: ул. Молодогвардейская, д. 33а, офис 223
тел./факс: (846) 277-89-79, 277-89-66; e-mail: pitvolga@mail.ru,
pitvolga@samara-ttk.ru

БЕЛАРУСЬ

Минск: ул. Розы Люксембург, д. 163; тел./факс: +37 517 208-80-01, 208-81-25;
e-mail: og@minsk.piter.com

Издательский дом «Питер» приглашает к сотрудничеству авторов:
тел./факс: (812) 703-73-72, (495) 234-38-15; e-mail: ivanovaa@piter.com
Подробная информация здесь: http://www.piter.com/page/avtoru

Издательский дом «Питер» приглашает к сотрудничеству зарубежных торговых партнеров или посредников, имеющих выход на зарубежный рынок: *тел./факс: (812) 703-73-73; e-mail: sales@piter.com*

Заказ книг для вузов и библиотек:
тел./факс: (812) 703-73-73, доб. 6243; e-mail: uchebnik@piter.com

---

Заказ книг по почте: *на сайте www.piter.com; тел.: (812) 703-73-74, доб. 6216; e-mail: books@piter.com*

Вопросы по продаже электронных книг: *тел.: (812) 703-73-74, доб. 6217; e-mail: kuznetsov@piter.com*

Эрик Хэнчетт, Бенджамин Листуон

Vue.js

В ДЕЙСТВИИ

Vue.js — это популярная библиотека для создания пользовательских интерфейсов. В ней значительно переосмыслены реактивные идеи, впервые появившиеся в React.js.

Книга «Vue.js в действии» рассказывает о создании быстрых и эластичных пользовательских интерфейсов для Интернета. Освоив ее, вы напишете полноценное приложение для интернет-магазина, где будут присутствовать списки товаров, админка, а также организован полноценный процесс онлайнового заказа.

В КНИГЕ ВЫ НАЙДЕТЕ:

- Подробно аннотированный код и иллюстрации
- Моделирование данных и создание API
- Удобное управление состояниями при помощи Vuex
- Создание собственных директив

Подробно изучайте эту книгу, если хотите оставаться на переднем крае веб-разработки!

ОТЗЫВЫ

Отличное знакомство с библиотекой Vue.js и знакомство с ее экосистемой.

Алекс Миллер, Slalom

Сила книги — в практических примерах! Они не только помогают как следует закрепить теоретические основы, но и пригодятся вам в собственных проектах.

Дуг Уоррен, Java Web Services

Дает четкое понимание всех базовых механизмов Vue.js. Бесценная книга.

Пьер Шарьер, Clever Cloud

ПИТЕР®

Заказ книг:
тел.: (812) 703-73-74
books@piter.com

WWW.PITER.COM
каталог книг и интернет-магазин

instagram.com/**piterbooks**
youtube.com/**ThePiterBooks**
vk.com/**piterbooks**
facebook.com/**piterbooks**

ISBN 978-5-4461-1098-8