Содержание

Об авторе

Пабло Дилеман (Pablo Deeleman) в 90-х годах работал дизайнером пользовательских интерфейсов, когда открыл для себя Интернет. В то время модем со скоростью 14 400 бит/с был ключом в волшебный мир, а самой популярной была игра с названием «Создай свой сайт».

После получения степени бакалавра (с отличием) в области маркетинга и поработав в рекламной индустрии, он остался верен себе и погрузился в самостоятельное изучение, превратившись в увлеченного дизайнера пользовательских интерфейсов и разработчика веб-приложений с блестяще подготовленными CSS-макетами и толстыми клиентами на JavaScript. Пабло создал бесчисленное множество интерактивных проектов и веб-приложений для настольных и мобильных устройств.

В течение нескольких лет он сделал карьеру дизайнера и разработчика, успешно руководившего интернет-проектами для широкого круга клиентов. Работал с европейскими туроператорами, начинающими компаниями из Кремниевой долины, международными веб-сайтами с тяжелым интернет-траффиком, порталами глобальных банковских систем, компаниями, занимающимися созданием игровых программ для мобильных устройств, и это далеко не полный список. Появление Node.js и фреймворков одностраничных приложений стало поворотным пунктом в его карьере, заставив сосредоточиться на создании взаимодействий с пользователем, управляемых с помощью JavaScript.

Пожив и поработав в нескольких странах, Пабло Дилеман обосновался в Барселоне (Испания) и теперь возглавляет разработку интерфейсов в Барселонской студии компании Gameloft, мирового лидера в области разработки игровых приложений для мобильных устройств, известного такими играми, как «Despicable Me: Minions Rush» и «Asphalt 8».

Большую часть свободного времени, когда он не пишет книги и не принимает участия в корпоративных мероприятиях и переговорах, Пабло посвящает своему увлечению – игре на фортепиано и гитаре.

Благодарности

Книга, которую вы сейчас держите в руках, является результатом больших усилий и массы затраченного на нее времени. Однажды кто-то мудро заметил, что писать книгу о фреймворке, находящемся в стадии альфа-версии, – это то же самое, что стрелять навскидку по движущейся мишени, и это действительно так. В процессе написания книги автор и команда проекта сбились со счета, сколько раз пришлось ее переписывать, чтобы привести в соответствие с самой последней реализацией фреймворка. Немногие способны выдержать накал такого сражения, не разочароваться и не согласться с мыслью о прекращении проекта. Поэтому я хочу выразить благодарность команде издательства Packt и прежде всего Саманте Гонсалвес (Samantha Gonsalves). Теплые слова их поддержки подпитывали меня энергией, так необходимой для продвижения проекта.

Я также хотел бы особо поблагодарить своего друга Хорхе Феррандо (Jorge Ferrando) за помощь и подсказки в процессе работы над книгой. Его опыт работы с Angular 2 был бесценен при выборе между разными видами подачи обучающего материала. Кроме того, следует упомянуть моих коллег-разработчиков: Хавьера Гомеса (Javier Gómez), Альфонсо Фернандеса (Alfonso Fernández), Франа Ириэла (Fran Iruela) и Педро Нарцисо (Pedro Narciso).

Еще я хотел бы поблагодарить всех моих наставников, направлявших мою профессиональную деятельность на протяжении нескольких лет, и особенно сотрудников компаний Casumo и Gameloft: Расмуса Свеннингсона (Razmus Svenningson), Кима Ларсен (Kim Larsen), Джосефа Галеа (Josef Galea), Стива Аттарда (Steve Attard), Идена Аззопарди (Iden Azzopardi), Ренальда Далли (Renald Dalli), Мэтью Борга (Matthew Borg), Марка Бусуттила (Mark Busuttil), Жерара Гинэ (Gerard Giné), Антонио Гонсалеса (Antonio González), Альберта Пуэртоласа (Albert Puértolas), Рафаэля Марфил (Rafael Marfil) и всегда вдохновляющего меня Стюарда Лангриджа (Stuart Langridge).

О технических рецензентах

Йоханнес Вебер (Johannes Weber) – увлеченный разработчик и консультант в области применения веб-технологий в промышленных JS-приложениях. Работает в компании Mayflower GmbH (Мюнхен, Германия), где специализируется на миграциях SPA и MPA. В свободное время (со)организует Мюнхенские конференции AngularCamp и JS-Kongress.de по AngularJS. Йоханнес сотрудничает с сайтом ESnextNews.com, который готов еженедельно присылать вам по электронной почте пять отличных ссылок по теме ECMAScript.next.

Предисловие

За последние годы Angular 1.x стал одним из наиболее популярных фреймворков на JavaScript, предназначенных для создания современных веб-приложений, больших и малых. Но по мере увеличения размеров и сложности приложений все явственнее проявлялись его недостатки в отношении производительности и масштабируемости. Angular 2 был полностью переписан для удовлетворения требований разработчиков современных веб-приложений, которым необходимы высокая производительность и гибкость.

Фреймворк Angular 2 разработан в соответствии с современными веб-стандартами и предоставляет определеную свободу в выборе языка, обеспечивая полную поддержку не только ES6 и TypeScript, но и широко распространенных сегодня ES5, Dart и CoffeeScript. Его встроенный механизм внедрения зависимостей позволяет создавать масштабируемые модульные приложения с ясным и выразительным кодом, значительно упрощая сопровождение и разработку, основанную на тестировании. Везде, где бы не применялся Angular 2, он демонстрирует беспрецедентный уровень быстродействия и производительности благодаря новой системе обнаружения изменений, которая работает в пять раз быстрее, чем ее предыдущая реализация. Более прозрачные представления и непревзойденный синаксис стандартных шаблонов предоставляют бесконечный список мощных функций для создания мобильных и настольных веб-приложений нового поколения.

Фреймворк Angular 2 сохранит и упрочит свою роль в разработке современных веб-приложений на годы вперед. Но именно из-за его архитектуры, разрушающей стереотипы, освоение Angular 2 может стать непосильной задачей для новичков.

И здесь на помощь придет эта книга, цель которой – помочь читателю избежать желудочных колик при изучении документации с описанием фреймворка и его программного интерфейса, предложив практической подход к освоению фреймворка, позволяющий сразу перейти к его использованию. И это будет сделано с самого ее начала.

Книга описывает процесс построения веб-проекта на новой платформе, основанной на синтаксисе TypeScript, начиная с основных понятий, примеров компонентов и их использования для создания

функционала, усложняющегося с каждой главой, пока в конце книги не будет получено завершенное, протестированное и готовое к эксплуатации веб-приложение.

Какие темы охватывает книга

Глава 1 «*Создание самого первого компонента в Angular 2*» знакомит с веб-компонентами – строительными блоками всех приложений, реализованных с помощью Angular 2.

Глава 2 «*Введение в TypeScript*» знакомит с синтаксисом и особенностями типизированного надмножества ECMAScript 6, выбранного командой Angular для создания Angular 2.

Глава 3 «*Реализация свойств и событий компонентов*» описывает поведение компонентов, которые изменяют свое состояние в соответствии с данными, полученными через входные свойства, и передают данные в виде событий через выходные свойства.

Глава 4 «*Расширение компонентов с помощью директив и фильтров*» содержит полное описание фильтров, используемых для обработки данных в шаблонах, а также встроенных директив, расширяющих возможности компонентов. В этой главе читатель также узнает, как создавать собственные фильтры и директивы

Глава 5 «*Создание приложений с помощью компонентов Angular 2*» включает повторение изученных ранее элементов и показывает, как обеспечить произвольное масштабирование проектов на Angular 2 и выполнение соглашений об именовании и оформлении кода.

Глава 6 «*Асинхронная обработка данных в Angular 2*» рассматривает внедрение и развертывание HTTP-соединений со сторонними службами с помощью модуля Http для создания собственных клиентов.

Глава 7 «*Маршрутизация в Angular 2*» знакомит с механизмом маршрутизации в Angular 2 и его встроенными директивами, обеспечивающими различные способы загрузки компонентов посредством маршрутов и обработку состояний через программный интерфейс History API.

Глава 8 «*Формы и аутентификация в Angular 2*» демонстрирует различные способы создания веб-форм и двухсторонней приязкой данных к элементам ввода, а также создание сложных форм и проверку их допустимости.

Глава 9 «*Анимация компонентов в Angular 2*» рассматривает инструменты и классы для реализации анимационных эффектов, начиная с использования только стилей CSS, управляемых директивами

Angular 2, и заканчивая сложными переходами, управляемыми из JavaScript через компоновщики анимации Angular 2.

Глава 10 «Модульное тестирование в Angular 2» содержит пошаговое описание приемов тестирования приложений и типичных шаблонов развертывания модульных тестов для компонентов, директив, фильтров, маршрутов и служб.

Что потребуется для работы с книгой

Для работы с примерами в этой книге прежде всего понадобится веб-браузер, обновленный до последней версии. Мы рекомендуем Google Chrome или Mozilla Firefox, хотя Angular 2 поддерживает все современные браузеры.

Также понадобится программа-терминал, так как многие операции будут выполняться через консольные команды npm, для запуска которых также понадобятся установленные в системе Node.js и npm. Прочие необходимые модули и процедуры и порядок их установки будут описаны по ходу работы.

И наконец, для программирования вам потребуется текстовый редактор. В *главе 1 «Создание самого первого компонента в Angular 2»* приводится подробное описание лучших на настоящий момент интегрированных сред разработки приложений, использующих фреймворк Angular 2.

Кому адресована эта книга

Эта книга адресована веб-разработчикам, которые хотят освоить разработку современных мобильных и настольных веб-приложений следующего поколения с помощью Angular 2. От читателя не требуется опыт работы с Angular 1.x или 2, но предполагается хорошее владение JavaScript. Она отлично подойдет новичкам в Angular, нуждающимся в пояснении и определении его концепций.

Соглашения

В этой книге используется несколько разных стилей оформления текста для выделения разных видов информации. Ниже приведены примеры этих стилей с объяснением их назначения.

Программный код в тексте, имена таблиц баз данных, имена папок, имена файлов, расширения файлов, фиктивные адреса URL, пользовательский ввод и ссылки в Twitter будут выглядеть так: «В

результате этого действия в корневой папке проекта появится новый файл `tsconfig.json`, содержащий параметры, необходимые компилятору TypeScript для преобразования кода компонента в обычный код на JavaScript спецификации ECMAScript 5, который способны обработать современные браузеры».

Блоки программного кода оформляются так:

```
<body>
  <nav class="navbar navbar-default navbar-static-top">
    <div class="container">
       <div class="navbar-header">
         <strong class="navbar-brand">My Pomodoro Timer</strong>
       </div>
    </div>
  </nav>
  <pomodoro-timer></pomodoro-timer>
</body>
```

Когда потребуется привлечь ваше внимание к определенному фрагменту в блоке программного кода, он будет выделяться жирным:

```
<body>
  <nav class="navbar navbar-default navbar-static-top">
    <div class="container">
      <div class="navbar-header">
        <strong class="navbar-brand">My Pomodoro Timer</strong>
      </div>
    </div>
  </nav>
  <pomodoro-timer></pomodoro-timer>
</body>
```

Ввод и вывод в командной строке будут оформляться так:

```
$ npm install angular2 es6-shim es6-promise reflect-metadata rxjs zone.js
--save
```

Новые термины и **важные слова** будут выделены жирным. Текст, отображаемый на экране, например в меню или в диалогах, будет оформляться так: «Раздел **learn** предоставляет доступ к обучающим материалам, позволяющим ускорить освоение языка».

Предупреждения и важные сообщения будут выделены подобным образом.

Подсказки и советы будут выглядеть так.

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв прямо на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com, при этом напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и вы заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Поддержка пользователей

Если вы гордый владелец книги, изданной в Packt, мы готовы предоставить вам дополнительные услуги, чтобы ваша покупка принесла вам максимальную пользу.

Загрузка исходного кода примеров

Вы сможете загрузить файлы с примерами кода на странице:

https://github.com/deeleman/learning-angular2.

Кроме того, скачать файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте www.dmkpress.com или www.дмк.рф в разделе «Читателям – Файлы к книгам».

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы удостовериться в качестве наших текстов, ошибки всё равно случаются. Если вы найдёте ошибку в одной из наших книг – возможно, ошибку в тексте или в коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии этой книги.

Если вы найдёте какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу dmkpress@gmail.com, и мы исправим это в следующих тиражах.

Нарушение авторских прав

Пиратство в Интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Packt очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в Интернете с незаконно выполненной копией любой нашей книги, пожалуйста, сообщите нам адрес копии или веб-сайта, чтобы мы могли принять меры.

Пожалуйста, свяжитесь с нами по адресу электронной почты dmkpress@gmail.com со ссылкой на подозрительные материалы.

Мы высоко ценим любую помощь по защите наших авторов, помогающую нам предоставлять вам качественные материалы.

Вопросы

Вы можете присылать любые вопросы, касающиеся данной книги, по адресу dm@dmk-press.ru или questions@packtpub.com. Мы постараемся разрешить возникшие проблемы.

Глава 1

Создание самого первого компонента в Angular 2

Если проследить тенденции за последнюю пару лет, можно отметить всплеск появления современных веб-фреймворков и библиотек на JavaScript для создания интерфейсов. Уже достигнута стадия, когда каждый день рождаются новые фреймворки, что заставляет разработчиков тщательно оценивать их, чтобы определить, стоит ли тратит время и силы на их освоение, а также их пригодность для применения в дальнейших проектах.

В конце концов, некоторые из них признаются наиболее актуальными. Вполне возможно, что следующие названия вам уже известны: Backbone, Ember, Knockout, Angular 1 и т. д.

В битву за господство в мире JavaScript вступили такие новые фреймворки, как React или Aurelia, отдающие предпочтение веб-компонентам и теневой модели DOM, которые являются краеугольным камнем их архитектуры. Приложения, построенные на их основе, получаются более модульными, масштабируемыми и простыми в сопровождении, не говоря уже об их непревзойденном уровне производительности.

С момента создания фреймворка Angular 1 прошло достаточно времени, чтобы его недостатки стали очевидными и с ними нельзя уже было мириться. Пришло время, когда отдельных улучшений в коде оказалось недостаточно. Потребовалось полностью сменить сам подход. В результате появился совершенно новый фреймворк, Angular 2, разработанный с нуля и полностью отвечающий новейшим тенден-

циям в отрасли. Центральное место в нем занимают веб-компоненты, а мощь теневой модели DOM используется для уменьшения времени реакции веб-сущностей на изменение состояния. Кроме того, Angular 2 включает систему обнаружения изменений состояния, встроенную в каждый компонент и отвечающую за организацию связей во всем дереве компонентов, составляющих приложение.

Определяющие черты Angular 2 выводят его за рамки простого фреймворка, основанного на веб-компонентах, поскольку его возможности охватывают практически все необходимое для построения современных веб-приложений: взаимодействие компонентов, универсальную поддержку различных платформ и устройств, превосходную технологию внедрения зависимостей, гибкий расширенный механизм маршрутизации для ослабления связей между компонентами, расширенную поддержку HTTP-сообщений, анимацию и интернационализацию. И это далеко не полный список.

В этой главе мы:

- узнаем причины уникальности Angular 2 в сравнении с предыдущими версиями;
- настроим среду для работы с Angular 2 и TypeScript;
- выберем интегрированную среду для удобной разработки приложений на основе Angular 2;
- создадим первый веб-компонент Angular 2 и поместим его в веб-страницу;
- добавим в веб-компонент простейшие интерактивные возможности;
- познакомимся с основными вспомогательными функциями форматирования выходных данных.

Обновленный подход

Как упоминалось выше, Angular 2 – это совершенно новый фреймворк Angular, основанный на архитектуре, изначально построенный на TypeScript, строгом надмножестве JavaScript, добавляющим статическую типизацию, поддержку интерфейсов и декораторов.

Другими словами, приложения, использующие Angular 2, основаны на архитектуре, включающей взаимосвязанные деревья веб-компонентов с их собственными интерфейсами ввода/вывода. Каждый компонент внутренне использует полностью переделанный механизм внедрения зависимостей. Справедливости ради следует отметить, что это лишь упрощенное описание того, чем Angular 2 яв-

ляется на самом деле. Однако даже простой проект, использующий Angular, наследует все его черты. В следующих главах мы познакомимся с организацией взаимодействий компонентов и внедрением зависимостей, после чего перейдем к маршрутизации, веб-формам и обмену информацией по протоколу HTTP. Это определение также объясняет, почему в книге не будет явных ссылок на Angular 1.x.

Очевидно, что нет смысла тратить время и место, ссылаясь на бесполезную для раскрытия описываемой темы информацию. Кроме того, поскольку предполагается, что читатель может быть незнаком с Angular 1.x, такие ссылки здесь явно излишни.

Веб-компоненты

Веб-компоненты представляют концепцию, которая охватывает четыре технологии, разработанные для создания элементов с высоким уровнем визуальной наглядности и возможностью повторного использования, что ведет к улучшению модульности, последовательности и облегчению обслуживания веб-приложений. К этим технологиям относятся:

- **шаблоны**: фрагменты разметки HTML, определяющие структуру отображаемого содержимого;
- **пользовательские элементы**: эти шаблоны содержат не только традиционные HTML-элементы, но и пользовательские оболочки, улучшающие визуальное представление элементов или расширяющие их функциональные возможности;
- **теневая модель DOM**: обеспечивает защищенную среду для инкапсуляции правил CSS-макета и JavaScript-модели поведения любого пользовательского элемента;
- **импорт HTML**: позволяет размещать в HTML-шаблоне не только HTML-элементы, но и другие HTML-документы.

Вообще говоря, компонент Angular 2 – это пользовательский элемент с шаблоном для размещения HTML-структуры макета, оформление которой определяется CSS-стилем, инкапсулированным в контейнер теневой модели DOM. Попробуем перевести это на обычный язык. Вспомним элемент управления HTML5, ограничивающий ввод числовых значений определенным диапазоном. Его удобно использовать, когда требуется дать возможность вводить значения, заключенные между двумя предопределенными границами. Если вы не использовали его раньше, вставьте следующий фрагмент разметки в пустой HTML-шаблон и загрузите его браузер:

```
<input id="mySlider" type="range" min="0" max="100" step="10">
```

Перед вами появится элемент управления в виде горизонтального ползунка. Если попробовать исследовать этот элемент с помощью инструментов разработчика в браузере, можно заметить скрытый набор HTML-тегов, отсутствующих в определении HTML-шаблона. Это и есть пример использования теневой модели DOM, когда действительный HTML-шаблон управляется встроенными в него CSS-правилами и дополнительными функциями управления ползунком. Согласитесь, что было бы достаточно сложно достичь того же результата самостоятельно. Самое замечательное, что Angular 2 предоставляет набор инструментов, необходимых для поддержки такой функциональности, благодаря чему мы можем строить собственные компоненты (элементы ввода, персонализированные теги и автономные виджеты), отображающие выбранную внутреннюю HTML-разметку с применением индивидуальных стилей, не влияющих (и не подверженных влиянию) на CSS-разметку страницы с такими компонентами.

Почему именно TypeScript?

Приложения на основе Angular 2 можно писать на разных языках, таких как: ECMAScript 5, Dart, ECMAScript 6, TypeScript или ECMAScript 7.

Язык TypeScript – это типизированное надмножество спецификации ECMAScript 6 (также известной как ECMAScript 2015), которое компилируется в обычный код на JavaScript и широко поддерживается современными ОС. Он придерживается объектно-ориентированного подхода, поддерживает аннотации, декораторы и контроль типов.

Причиной, почему в этой книге в качестве синтаксической основы выбран (и рекомендуется) язык TypeScript, является тот факт, что сам Angular 2 написан на этом языке. Опыт работы на TypeScript дает разработчику огромное преимущество при изучении внутреннего механизма фреймворка.

С другой стороны, поддержка аннотаций и контроля типов в TypeScript будет иметь первостепенное значение, когда мы дойдем до управления внедрением зависимостей и связями между компонентами.

Вообще говоря, проекты на основе Angular 2 можно реализовать, используя обычный синтаксис ECMAScript 6. Даже примеры, приведенные в книге, легко можно перенести на ES6, удалив аннотации типов и интерфейсов или заменив способ внедрения зависимостей в Typescript более сложными приемами, используемыми в ES6.

Все примеры в книге написаны исключительно на TypeScript, который мы рекомендуем использовать из-за выразительности аннотаций типов и более элегантного подхода к встраиванию зависимостей, основанного на контроле этих аннотаций типов.

Настройка рабочего окружения

Прежде чем начать реализацию первого компонента Angular 2, необходимо познакомиться со всеми инструментами, необходимыми для реализации программного обеспечения на TypeScript, не говоря уже о самих модулях фреймворка Angular 2.

Прежде всего создадим папку и проверим доступность в системе клиента NPM CLI, должным образом обновленного до последней стабильной версии. Если этот клиент у вас отсутствует, зайдите на сайт https://nodejs.org и установите последнюю версию Node.js.

На момент написания книги последней версией фреймворка была Angular 2 Release Candidate 1, поэтому представленные здесь примеры сборки и развертывания могут потерять актуальность. Автор поддерживает репозиторий по адресу https://github.com/deeleman/learning-angular2, из которого можно получить последнюю версию любого из примеров, содержащихся в этой книге. Репозиторий состоит из папок, соответствующих главам, и каждая папка содержит версии проекта для каждой главы. Если после установки или развертывания примеров возникнут проблемы, обратитесь к этому репозиторию.

Установка зависимостей

Очевидно, что подготовку рабочего окружения следует начать с установки самого фреймворка Angular 2 с его собственными зависимостями. Разработчики Angular 2 постарались сделать установку модульной, чтобы дать возможность включать в проекты только то, что действительно необходимо, в соответствии с предъявляемыми к нему требованиями.

Поэтому фреймворк Angular 2 поставляется не в виде единого установочного пакета, как это обычно бывает. Разработчику дается возможность выбрать только те модули, которые требуются для этого проекта, минимизировав общий объем зависимостей. Некоторые из пакетов, например `common` или `core`, обязательны независимо от вида разрабатываемого проекта. Другие пакеты, такие как `platform-browser-dynamic`, привязаны к виду проекта и целевой платформе.

Ниже приводится неполный список пакетов, которые чаще других используются в проектах:

- `@angular/core`: наиболее важный пакет, содержащий основу Angular и его наиболее общие элементы, такие как директивы и компоненты. Этот модуль всегда следует импортировать в проект;
- `@angular/common`: редко возникает необходимость явно импортировать элементы из этого модуля, но стоит отметить, что этот пакет содержит определения всех директив, служб и фильтров, встроенных в Angular 2, вместе с соответствующими классами;
- `@angular/compiler`: по аналогии с модулем `common` вам редко придется явно импортировать элементы из этого модуля. Он отвечает за компиляцию HTML-шаблонов в код, который позволяет отобразить пользовательский интерфейс приложения;
- `@angular/platform-browser`: содержит классы и функции, необходимые для компоновки и взаимодействия с моделью DOM в контексте веб-браузера. Этот модуль дает возможность обрабатывать такие обычные действия, как обновление заголовка страницы и настройка распознавания жестов. Этот пакет содержит также функции, необходимые для компиляции шаблонов в автономном режиме при производственной эксплуатации;
- `@angular/platform-browser-dynamic`: этот модуль часто будет упоминаться в книге, поскольку он предоставляет функцию загрузки, необходимую для инициализации приложений во время разработки;
- `@angular/http`: это HTTP-клиент Angular 2, который подробно рассматривается в *главе 6* «*Асинхронная обработка данных в Angular 2*»;
- `@angular/router`: это встроенный в Angular 2 маршрутизатор, который на момент написания этой книги все еще находился в стадии бета-версии;
- `@angular/router-deprecated`: предыдущая версия встроенного маршрутизатора, обеспечивающая обратную совместимость с существующими приложениями. В *главе 7* «*Маршрутизация в Angular 2*» мы подробно рассмотрим его основные отличия от нового маршрутизатора, еще находящегося на стадии разработки.

Ниже перечислены все сторонние библиотеки, необходимые как зависимости для проектов с Angular 2, помимо самих модулей Angular 2:

- `es6-shim`: включает полифиллы, обеспечивающие совместимость с ECMAScript 6 для старых движков JavaScript (главным образом для Microsoft Internet Explorer). Необходимость этой библиотеки объясняется отсутствием во многих популярных браузерах широкой поддержки ECMAScript 6, но, надеюсь, это скоро изменится. Некоторые другие реализации вместо этого модуля используют стандартную библиотеку `core-js`. В конце концов, можно выбрать любые полифиллы, обеспечивающие поддержку ядра программного интерфейса ES2015, требуемую Angular 2;
- `zone.js`: полифилл для спецификации Zone. Используется для обработки изменений, обнаруженных в приложениях на основе Angular 2;
- `reflect-metadata`: привносит поддержку декораторов в классах Angular 2 и метаданных компонентов. Использование декораторов будет рассмотрено ниже в этой же главе, а в *главе 2* «*Введение в TypeScript*» будет представлен обзор различных их типов и реализаций. Декораторы являются основной частью Angular 2;
- `rxjs`: эта библиотека разработана в Microsoft Open Technologies, Inc. По утверждениям компании Microsoft, этот набор библиотек предназначен для создания асинхронных и основанных на событиях программ, использующих отслеживаемые коллекции и методы `Array#extras` из языка JavaScript. Короче говоря, библиотека RxJS служит для управления отслеживаемыми объектами и позволяет реализовать асинхронную обработку изменений в состоянии приложения. Работа с отслеживаемыми объектами в будущем станет стандартом, поддерживаемым всеми современными браузерами, поэтому со временем данная зависимость будет исключена.

Эти зависимости могут изменяться без предварительного на то уведомления, поэтому за актуальным списком требований обращайтесь к репозиторию GitHub.

Вас, вероятно, удивило количество библиотек, необходимых Angular 2, и тот факт, что эти зависимости не являются частью пакета Angular. Однако эти требования не являются специфическими для Angular 2, в настоящее время они характерны для подавляющего большинства современных приложений на JavaScript.

Для установки всех этих зависимостей и сторонних библиотек после создания папки проекта запустите в окне терминала следующие bash-команды:

```
$ mkdir learning-angular2
$ cd learning-angular2
$ npm init
$ npm install @angular/common @angular/core @angular/compiler --save
$ npm install @angular/platform-browser @angular/platform-browser-dynamic
--save
$ npm install @angular/router @angular/router-deprecated --save
$ npm install @angular/http --save
$ npm install es6-shim reflect-metadata rxjs zone.js --save
```

Помимо перечисленных зависимостей, для поддержки загрузки модулей и приведения их в соответствие с ES5 нужно также установить пакет универсального загрузчика модулей `systemjs`. Пакет `systemjs` не является единственным возможным вариантом управления загрузкой модулей в Angular 2. На самом деле его можно заменить другим загрузчиком, таким как **WebPack** (https://webpack.github.io/), но в этой книге все примеры используют SystemJS. Установим SystemJS, пометив его как зависимость для разработки, выполнив следующую команду:

```
$ npm install systemjs --save
```

Наконец, установим библиотеку **Bootstrap**, чтобы упростить создание привлекательного пользовательского интерфейса приложения, которое будет постепенно дорабатываться в каждой из глав книги. Это не является требованием Angular 2, а станет зависимостью проекта, реализуемого в этой книге:

```
$ npm install bootstrap --save
```

В процессе установки могут выводиться различные сообщения и предупреждения, что связано с версиями каждой из зависимостей, требуемых Angular 2 на данный момент времени, поэтому в случае появления вопросов я настоятельно рекомендую загрузить последнюю версию файла `package.json` из репозитория к этой книге: https://github.com/deeleman/learning-angular2/blob/master/chapter_01/package.json.

Загрузите этот файл в рабочий каталог и выполните команду `npm install`. NPM автоматически найдет и установит все зависимости.

Пользователям Mac OS, не обладающим правом на запись в каталог, где находится `npm` (обычно это каталог `/usr/local/bin/npm` или `/usr/local/npm` – для версий ОС, предшествующих Mac OS El Capitan), может потребоваться выполнить команду `npm install` с привилегиями суперпользователя.

Установка TypeScript

Итак, у нас имеется полный набор исходных текстов фреймворка Angular 2 и его зависимостей, а также пакеты Bootstrap, для придания привлекательного внешнего вида проекту, и SystemJS для управления загрузкой и связыванием модулей.

Однако в системе все еще отсутствует TypeScript. Установим TypeScript и сделаем его глобально доступным, чтобы получить удобный интерфейс командной строки для компиляции файлов:

```
$ npm install -g typescript
```

Отлично! Мы почти закончили. Осталось лишь уведомить TypeScript, что в проекте будет использоваться компилятор. Для этого просто выполните следующую команду:

```
$ tsc --init --experimentalDecorators --emitDecoratorMetadata --target ES5
--module system --moduleResolution node
```

По сути, эта команда просто добавляет в наш проект на TypeScript (который одновременно является и проектом Angular 2) поддержку экспериментальных декораторов (как уже упоминалось, это новая функция ES7 и TypeScript, широко используемая в Angular 2) и SystemJS как механизма по умолчанию для импорта модулей и зависимостей.

В результате этого действия в корне проекта появится новый файл `tsconfig.json`, включающий параметры, необходимые компилятору TypeScript для преобразования кода компонента в обычный код на JavaScript (соответствующий спецификации ECMAScript 5), понятный современным браузерам.

Не забывайте, что на настоящий момент браузеры не имеют встроенной поддержки ни TypeScript, ни ECMAScript 6, поэтому код должен быть преобразован в код на JavaScript, поддерживаемый целевыми браузерами.

Давайте взглянем на содержимое этого файла:

```
{
  "compilerOptions": {
    "experimentalDecorators": true,
    "emitDecoratorMetadata": true,
    "target": "es5",
    "module": "system",
    "moduleResolution": "node",
    "noImplicitAny": false,
```

```
    "outDir": "built",
    "rootDir": ".",
    "sourceMap": false
  },
  "exclude": [
    "node_modules"
  ]
}
```

Все просто, не так ли? Свойства, включенные в манифест конфигурации, интуитивно понятны, но мне хотелось бы отметить три наиболее интересных из них:

- `rootDir`: указывает на папку, где компилятор будет искать TypeScript-файлы для компиляции (в данном случае это корневая папка примера);
- `outDir`: определяет, куда следует поместить скомпилированные файлы: если в командной строке компилятора отсутствует параметр `--outDir`, по умолчанию будет использоваться папка `built`, созданная во время выполнения там, где находится файл `tsconfig.json`;
- `sourceMap`: определяет режим сопоставления с исходным кодом для нужд отладки.

Присвойте этому свойству значение `true`, если нужно генерировать файлы сопоставления исходных кодов для обратной трассировки инструментами разработчика в браузере при появлении исключений.

Здесь также можно видеть, что папка `node_modules` добавлена в список исключений (свойство `exclude`), а это означает, что команда `tsc` будет пропускать эту папку и все ее содержимое при преобразовании TypeScript-файлов в ES5-файлы во всем дереве приложения.

Полное описание параметров компилятора TypeScript можно найти по адресу: https://github.com/Microsoft/TypeScript/wiki/Compiler-Option.

Установка утилиты typings

Помимо зависимостей проекта, таких как Bootstrap, и зависимостей Angular 2, пакету TypeScript требуются дополнительные библиотеки, облегчающие работу. В частности, ES6 добавляет в среду JavaScript новые методы и прикладные интерфейсы, которые должны быть описаны в компиляторе TypeScript. В противном случае он не распознает их как часть синтаксиса, что вызовет ошибки компиляции. Всякий раз, когда нужно оповестить компилятор TypeScript о программном

интерфейсе на JavaScript – встроенном или в виде сторонней библиотеки, используется файл определения типов TypeScript.

Обычно определения типов для TypeScript хранятся в файле с расширением `d.ts` (подробно об этом рассказывается в *главе 2* «*Введение в TypeScript*»), который используется для проверки типов и предотвращения ошибок компиляции. Подготовка файлов с определениями типов в проектах не слишком сложна и требует лишь наличия специального инструмента: *typings*. Фактически мы обязаны установить файл с определениями типов, чтобы «подружить» компилятор TypeScript с последним прикладным интерфейсом ES6. Однако могу вас обрадовать: при установке инструмента управления определениями типов для TypeScript все необходимые данные можно получить прямо из реестра NPM и тем самым автоматизировать процесс поиска, установки и развертывания файлов с определениями типов. Для этого вернемся в консоль и выполним следующие команды:

```
$ npm install -g typings
$ typings install es6-shim --ambient --save
```

Первая команда устанавливает инструмент `typings` глобально, а вторая устанавливает в проект файл с определениями типов `es6-shim`, создавая при этом файл `typings.json`, который хранит ссылки на источники происхождения всех файлов, уже установленных и которые будут установлены позднее. Она создаст новую папку `typings` и поместит в нее требуемые файлы определений. Без них будут не доступны базовые возможности ES6, такие как функциональные методы класса `Array`. Прежде чем двинуться дальше, необходимо решить еще один вопрос, касающийся поддержки типов в TypeScript. При установке файлов с определениями типов генерируются два фасадных файла: `typings/main.d.ts` и `typings/browser.d.ts`. Однако компилятору TypeScript должен быть доступен только один из них. В противном случае дублирующие определения типов будут вызывать исключения. Поскольку разрабатывается интерфейсное приложение, отдадим предпочтение файлу `browser.d.ts` и отключим `main.d.ts`, а также связанные с ним определения файлов, пометив его в раздел `exclude` в файле `tsconfig.json`:

```
{
  "compilerOptions": {
    "experimentalDecorators": true,
    "emitDecoratorMetadata": true,
    "target": "es5",
    "module": "system",
```

```
    "moduleResolution": "node",
    "noImplicitAny": false,
    "outDir": "built",
    "rootDir": ".",
    "sourceMap": false
  },
  "exclude": [
    "node_modules",
    "typings/main.d.ts",
    "typings/main"
  ]
}
```

Также рекомендуется исключить папку `typings` из дистрибутива проекта, включив ее в файл `.gitignore`, как это обычно делается с папкой `node_modules`. В дистрибутив требуется включить только манифест `typings.json` и выполнять установку файлов определений типов, как действие `postinstall` в файле `package.json`. Таким способом можно одной операцией установить и `npm`-зависимости, и файлы с определениями:

```
"scripts": {
  "typings": "typings",
  "postinstall": "typings install"
},
```

В этом случае пакет `typings` должен быть включен в `package.json` как зависимость для окружения разработки. Поэтому переустановим его с флагом `--save-dev`. И снова за последней версией файла `package.json` для этой главы обращайтесь к репозиторию GinHub.

Hello, Angular 2!

После установки пакета библиотек Angular 2 и полной поддержки TypeScript пришло время произвести тестирование. Во-первых, создадим и очистим файл `hello-angular.ts` (`.ts` – это расширение для TypeScript-файлов) в корневой папке проекта.

Здесь мы сталкиваемся с первым из множества соглашений, используемых в этой книге, касающимся именования файлов. Имена файлов модулей должны содержать буквы в нижнем регистре с дефисом между словами. В *главе 5 «Создание приложений с помощью компонентов Angular 2»* мы подробно рассмотрим соглашения об именах и передовой опыт программирования при-

ложений с применением Angular 2. Но до тех пор будем придерживаться в учебных целях некоторых антишаблонов, что очень скоро заметят более опытные читатели.

Откроем файл и введем следующий текст:

```
import { Component } from '@angular/core';
import { bootstrap } from '@angular/platform-browser-dynamic';
```

Мы только что импортировали самые основные типы и функции, необходимые для создания очень простого компонента в следующем разделе. Синтаксис должен быть знаком тем, кто имел дело с ECMAScript 6. Если вы не принадлежите к их числу, не волнуйтесь, он будет подробно описан в *главе 2* «*Введение в TypeScript*».

Классы TypeScript

А теперь определим класс:

```
class HelloAngularComponent {
  greeting: string;
  constructor() {
     this.greeting = 'Hello Angular 2!';
  }
}
```

В спецификации ECMAScript 6 (и TypeScript) классы определяются как один из основных строительных блоков. Как видно из примера выше, класс содержит поле свойства `greeting` с типом `string`, заполняемое в конструкторе текстовой строкой. Функция-конструктор вызывается автоматически при создании экземпляра класса, и каждое свойство (и функция) должно быть аннотировано соответствующим ему типом (или типом значения, возвращаемого функцией).

Не стоит об этом сейчас беспокоиться. *Глава 2* «*Введение в TypeScript*» содержит все необходимые пояснения, касающиеся механики TypeScript. Теперь же сосредоточимся на макете компонента. Как вы, вероятно. заметили, имя структуры соответствует другому общему соглашению, принятому в Angular 2. Имена классов определяются в стиле языка Pascal с добавлением суффикса, указывающего на его тип (то есть компонент, директива, фильтр и т. д.), в данном случае `Component`.

Введение в декораторы

Только что созданный класс контроллера уже позволяет создать экземпляр объекта со свойством `greeting`, но, чтобы превратить его в на-

стоящий компонент, к нему нужно добавить некоторый синтаксический сахар Angular 2. Ранее мы импортировали метаданные класса `Component`, помните? А теперь используем его, декорировав наш класс следующим образом:

```
@Component({
  selector: 'hello-angular',
  template: '<h1> {{greeting}} </h1>'
})
class HelloAngularComponent {
  greeting: string;
  constructor() {
      this.greeting = 'Hello Angular 2!';
  }
}
```

Декоратор – очень интересная экспериментальная особенность, появившаяся в спецификации ECMAScript 7, которая позднее была заимствована и реализована в TypeScript для добавления метаданных в классы. Существует несколько типов декораторов, и все они легко узнаваемы по префиксу @. *Глава 2 «Введение в TypeScript»* знакомит с декораторами, но подробное описание их базовой логики выходит за рамки этой книги. Тем не менее они будут использоваться на протяжении всей книги.

В данном случае мы сообщили компилятору, что класс `HelloAngularComponent` в действительности является компонентом Angular 2. Компонент предназначен для инкапсуляции пользовательского элемента `<hello-angular>`, свойство `template` которого определяет внутреннюю HTML-структуру компонента. Как уже упоминалось ранее, пользовательские элементы, инкапсулирующие HTML-шаблоны, являются основой веб-компонентов.

Компиляция TypeScript в понятный браузерам код на JavaScript

Фундаментом для нашего примера послужил TypeScript, но, к сожалению, вполне вероятно, что выбранный браузер не будет его поддерживать. Поэтому исходный код следует скомпилировать в старый добрый JavaScript ECMAScript 5.

Хочу вас обрадовать: интерфейс CLI TypeScript включает готовые инструменты для компиляции кода на TypeScript в JavaScript. Для этого откройте окно терминала, перейдите в каталог с файлом `hello-angular.ts` и введите следующую команду:

```
$ tsc --watch
```

Вновь созданный файл `hello-angular.js` с версией кода TypeScript, соответствующей спецификации ECMAScript 5, будет помещен в каталог `built` (или в каталог, указанный в свойстве `outDir` в файле `tsconfig.json`). Этот файл уже содержит некоторый функциональный код для поддержки настроенных метаданных декоратора.

Обратите внимание на наличие флага `--watch` в команде. Этот параметр информирует компилятор, что при внесении изменений в файл требуется автоматически выполнить повторную компиляцию. Опускайте этот флаг, когда нужна лишь одноразовая компиляция и нет необходимости отслеживать изменения в коде.

Компонент выглядит все лучше и лучше, и сейчас самый подходящий момент, чтобы начать его использовать, но прежде его нужно поместить куда-то, чтобы увидеть в действии. Давайте определим HTML-оболочку, или веб-контейнер, где он будет находиться.

HTML-контейнер

Создайте в корневом каталоге проекта HTML-файл с именем `index.html` и добавьте в него следующий код:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Hello Angular 2!</title>

    <script src="node_modules/es6-shim/es6-shim.min.js"></script>
    <script src="node_modules/zone.js/dist/zone.js"></script>
    <script src="node_modules/reflect-metadata/Reflect.js"></script>
    <script src="node_modules/systemjs/dist/system.js"></script>
    <script src="node_modules/rxjs/bundles/Rx.js"></script>

    <script src="systemjs.config.js"></script>

  </head>

  <body>

  </body>

</html>
```

Это простейшая версия HTML-контейнера для приложения, использующего Angular 2. Выглядит он достаточно просто, потому что

большая часть логики отображения и управление зависимостями обрабатываются самим компонентом.

Тем не менее обратите внимание на две вещи. Во-первых, в нем присутствует блок подключения сценариев. Он включает все зависимости полифилла ES2015 (ES6), реализации спецификаций Zone и Observables, обеспечивающих полную поддержку декораторов, и, наконец, функцию динамической загрузки модулей приложения.

Не пытайтесь изменить порядок строк в этих блоках кода, если не хотите столкнуться с неожиданными исключениями.

Во-вторых, сценарий включает ссылку на новый файл `systemjs.config.js`. Этот файл еще не написан, поэтому создайте его в корневой папке проекта для дальнейшей реализации. Рассмотрим его содержимое в следующих абзацах:

```
(function() {

  var pathMappings = {
    '@angular': 'node_modules/@angular',
    'rxjs': 'node_modules/rxjs',
  };

  var packages = [
    '@angular/common',
    '@angular/compiler',
    '@angular/core',
    '@angular/http',
    '@angular/platform-browser',
    '@angular/platform-browser-dynamic',
    '@angular/router',
    '@angular/router-deprecated',
    '@angular/testing',
    'rxjs',
    'built',
  ];

  var packagesConfig = {};

  packages.forEach(function(packageName) {
    packagesConfig[packageName] = {
      main: 'index.js',
      defaultExtension: 'js'
    };
  });
```

```
  System.config({
    map: pathMappings,
    packages: packagesConfig,
  });

})();
```

Как видите, файл `systemjs.config.js` содержит выражение с немедленно вызываемой функцией. То есть данный код будет выполнен в момент обработки его браузером. Рассмотрим поочередно каждую из частей этой процедуры:

```
var pathMappings = {
  '@angular': 'node_modules/@angular',
  'rxjs': 'node_modules/rxjs',
};
```

Здесь определяется объект со свойствами, содержащими полные пути. Далее, в этом же файле, данный объект будет использован для настройки псевдонимов путей в SystemJS, чтобы при выполнении оператора `import '@angular/core'` SystemJS импортировал основной пакет из `node_modules/@angular/core`. Однако пакеты не импортируются как единое целое. Необходимо иметь лишь точку входа в пакет, и, следовательно, нужно сообщить SystemJS, какой фасадный файл следует искать, чтобы импортировать весь модуль. Это достигается определением массива модулей с последующим созданием объекта, где пути к модулям являются ключами, содержащими точки входа, и объекта конфигурации с расширением файла по умолчанию для таких путей:

```
var packages = [
  '@angular/common',
  '@angular/compiler',
  '@angular/core',
  '@angular/http',
  '@angular/platform-browser',
  '@angular/platform-browser-dynamic',
  '@angular/router',
  '@angular/router-deprecated',
  '@angular/testing',
  'rxjs',
  'built',
];

var packagesConfig = {};
```

```
packages.forEach(function(packageName) {
  packagesConfig[packageName] = {
    main: 'index.js',
    defaultExtension: 'js'
  };
});
```

Полученная в результате переменная `packagesConfig` представляет собой объект с описанной выше конфигурацией. Она также включает главный файл для входа и расширение по умолчанию для папки сборки, куда компилятор TypeScript будет сохранять скомпилированные файлы.

В заключение необходимо настроить в SystemJS отображение путей, точку входа и расширения файлов по умолчанию, чтобы связать все пакеты с предоставленными ранее путями:

```
System.config({
  map: pathMappings,
  packages: packagesConfig,
});
```

Вот и вся настройка, необходимая SystemJS. Теперь можно начинать работу над приложением, опираясь на синтаксис ES2015 импорта модулей, как описывается в следующих разделах.

Сопроводительные примеры книги

Прежде чем заняться примером непосредственно, необходимо настроить локальный веб-сервер для его выполнения. Если у вас уже есть рабочий веб-сервер, который можно настроить на использование указанного выше каталога, тогда переходите к чтению следующего раздела. В противном случае вам следует подготовить такой веб-сервер для опробования примеров. Чтобы облегчить эту задачу, рекомендую установить чрезвычайно мощный и компактный модуль `lite-server` из NPM:

```
$ npm install -g lite-server
```

После этого вы сможете запускать веб-сервер, выполнив следующую команду в терминале из папки проекта:

```
$ lite-server
```

Эта команда запустит веб-сервер и откроет окно браузера, указав ему на рабочий каталог. Все доступные параметры можно най-

ти в официальном репозитории модуля NPM (https://github.com/johnpapa/lite-server).

Пакет `lite-server` рекомендуется устанавливать вместе с пакетами `typescript` и `concurrently`, определив их как зависимости времени разработки с помощью флага `--save-dev`. Это даст возможность одновременно запускать компилятор TypeScript в режиме отслеживания и локальный сервер одной командой, которая может быть помещена в сценарий `start`, объявленный в файле `package.json`. После этого вы сможете приступить к сборке, перейдя в папку проекта и выполнив команду `npm start`. Такой подход реализован в примерах для этой книги, хранящихся в репозитории GitHub, так что можете смело загрузить оттуда удобный пример `package.json`.

Соединяем все вместе

Теперь HTML-файл готов для размещения компонента Angular 2. Для этого отредактируйте шаблон и поместите в него пользовательский элемент с тем же именем тега, которое было задано в свойстве `selector` компонента. Затем импортируйте файл с объявлением класса компонента, используя для этого программный интерфейс SystemJS. Сверьте внесенные изменения с кодом ниже:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Hello Angular 2!</title>
    <script src="node_modules/es6-shim/es6-shim.min.js"></script>
    <script src="node_modules/zone.js/dist/zone.js"></script>
    <script src="node_modules/reflect-metadata/Reflect.js"></script>
    <script src="node_modules/systemjs/dist/system.js"></script>
    <script src="node_modules/rxjs/bundles/Rx.js"></script>

    <script src="systemjs.config.js"></script>
      // Импортировать модуль компонента
      // без расширения файла
    <script>
      System.import ('built/hello-angular');
    </script>
  </head>
  <body>
    <!-- Пользовательский тег элемента -->
    <hello-angular></hello-angular>
  </body>
</html>
```

Круто, правда? Теперь мы можем создавать собственные элементы, которые делают все, что в них определено. Давайте откроем страницу в браузере, чтобы увидеть, как она действует, введя адрес http://localhost:3000/ (или указав другое имя хоста и номер порта, на которые настроен локальный веб-сервер).

К сожалению, если перезагрузить страницу в браузере, ничего не произойдет, и он будет отображать пустую страницу без содержимого. Чтобы что-то появилось, необходимо загрузить компонент и создать его экземпляр на HTML-странице.

Вернемся к файлу компонента `hello-angular.ts` и добавим в конец строку:

```
import { Component } from '@angular/core';
import { bootstrap } from '@angular/platform-browser-dynamic';

@Component({
  selector: 'hello-angular',
  template: '<h1> {{greeting}} </h1>'
})
class HelloAngularComponent {
  greeting: string;
  constructor() {
    this.greeting = 'Hello Angular 2!';
  }
}

bootstrap(HelloAngularComponent); // Компонент загружен!
```

Команда `bootstrap` создает экземпляр класса контроллера, передаваемого в аргументе, и включает его в приложение. В основном метод `bootstrap` выполняет следующие действия:

- анализирует конфигурацию компонента, переданного в первом аргументе, и проверяет его тип;
- находит в дереве DOM элемент с тегом, соответствующим свойству `selector` компонента;
- создает дочерний инструмент, который выполнит внедрение зависимостей в этот компонент и всех дочерних директив (включая компоненты, которые тоже являются директивами), что позволят использовать при размещении компонента подход, схожий с ветвлением дерева;
- создает новую зону. Зоны не будут рассматриваться в этой книге, просто отметим, что они отвечают за управление механизмом обнаружения изменений каждого загруженного экземпляра компонента в изолированном режиме;

- создает контекст теневой модели DOM в пользовательском элементе, идентифицируемом свойством компонента `selector`, и отображает его с применением HTML-кода из шаблона компонента;
- сразу после создания экземпляра класса контроллера компонента вызывает механизм обнаружения изменений и затем инициализирует поставщиков, которые заполняют теневую модель DOM, внедряя необходимые данные.

Позже я расскажу, как с помощью команды `bootstrap` организовать отображение отладочной информации и переопределить поставщиков данных во всем приложении, чтобы инструмент внедрения зависимостей, встроенный в Angular 2, смог находить нужные зависимости.

Надеюсь, что вы запустили компилятор TypeScript в режиме отслеживания. Если это не так, выполните команду `tsc`, чтобы скомпилировать код, и перезагрузите страницу в браузере. А теперь можно порадоваться успешному выводу содержимого нашего первого компонента Angular 2 (рис. 1.1)!

Рис. 1.1