Оглавление

Содержание

Из отзывов к пятому изданию книги Крейга Уоллса «Spring в действии»

«Отличный источник знаний о таком сложном фреймворке».

– *Арнальдо Габриэль Айяла Мейер* (Arnaldo Gabriel Ayala Meyer), Consultores Informáticos S.R.L.

«Охватывает все аспекты последней версии Spring и демонстрирует их на практических примерах».

– *Билл Флай* (Bill Fly), Брукхейвенский колледж

«Настольная книга для изучения Spring Framework и отличное справочное руководство».

– *Колин Джойс* (Colin Joyce), Cisco

«На мой взгляд, это лучшая книга о Spring. Новое издание подверглось обширному обновлению, обеспечившему уникальный баланс между теорией и практикой. Она поможет вам быстро приступить к работе и даст подробные пояснения.

– *Дэниел Вон* (Daniel Vaughan), Европейский институт биоинформатики

«Исчерпывающее руководство по разработке облачных приложений с использованием Spring».

– *Дэвид Уизерспун* (David Witherspoon), Parsons Corporation

«Источник истины в последней инстанции для экосистемы Spring».

– *Эдду Мелендес Гонсалес* (Eddú Meléndez Gonzales), Scotiabank

«Я настоятельно рекомендую эту книгу и новичкам в Spring Framework, и опытным разработчикам, которые хотят глубже освоить новейшие возможности, доступные в экосистеме Spring 5».

– *Иэн Кэмпбелл* (Iain Campbell), Tango Telecom

«Даже будучи опытным разработчиком на Spring, я нашел в этой книге много новых практических советов».

– *Джеттро Коэнради* (Jettro Coenradie), Luminis

Предисловие

Фреймворк Spring появился более 18 лет назад, и его главной целью было упрощение разработки Java-приложений. Первоначальная цель состояла в том, чтобы дать облегченную альтернативу EJB 2.x. Но это было только начало. С годами фреймворк Spring охватывал все более широкий круг задач разработки, включая хранение данных, безопасность, интеграцию, облачные вычисления и т. д.

Несмотря на то что возраст Spring приближается к двум десятилетиям и он все шире охватывает сферу разработки корпоративных приложений на Java, в развитии фреймворка не наблюдается никаких признаков замедления. Spring продолжает решать проблемы разработки на Java, будь то приложение, развернутое на обычном сервере приложений, или контейнерное приложение, развернутое в кластере Kubernetes в облаке. А учитывая наличие Spring Boot – фреймворка, обеспечивающего автоматическую конфигурацию, помощь в сборке зависимостей и мониторинг времени выполнения, – никогда не было более удачного времени, чтобы стать разработчиком на Spring!

Это издание книги «Spring в действии» – ваш путеводитель по Spring и Spring Boot. Оно было обновлено и теперь еще полнее отражает все, что могут предложить оба фреймворка. Даже если вы только приступаете к изучению Spring, вы сможете запустить свое первое приложение на Spring еще до конца первой главы. По мере изучения книги вы узнаете, как создавать веб-приложения, работать с данными, защищать приложения и управлять их конфигурациями. Затем вы познакомитесь с вариантами интеграции приложений Spring с другими приложениями и узнаете, как извлечь выгоду из реактивного программирования, используя новый протокол RSocket. Ближе к концу вы увидите, как подготовить приложение к передаче в промышленное окружение, и узнаете о доступных вариантах развертывания.

Кем бы вы ни были – новичком, только начинающим изучать Spring, или ветераном, с многолетним опытом разработки на Spring, – эта книга станет вашим следующим шагом в вашем путешествии. Я очень рад представить вам это руководство и с нетерпением жду появления ваших творений на Spring!

Об этой книге

«Spring в действии, шестое издание» писалась с простой целью: дать вам возможность создавать потрясающие приложения с использованием Spring Framework, Spring Boot и других инструментов из экосистемы Spring. Книга начинается демонстрацией создания веб-приложения на Java с поддержкой базы данных с помощью Spring и Spring Boot. Затем она описывает дополнительные возможности, показывая, как организовать интеграцию с другими приложениями и программами, использующими реактивные типы. Наконец, в ней обсуждаются вопросы подготовки приложения к развертыванию.

Все проекты в экосистеме Spring снабжаются превосходной документацией, однако эта книга дает то, чего не может дать ни один из справочных документов, – практическое руководство по объединению компонентов Spring и созданию действующих приложений.

Кому адресована эта книга

Книга «Spring в действии, шестое издание» адресована всем Java-разработчикам, желающим начать использовать Spring Boot и Spring Framework, а также опытным разработчикам на Spring, которые стремятся выйти за рамки основ и изучить новейшие возможности Spring.

Структура книги

Книга состоит из четырех частей и 18 глав. Часть I охватывает основы создания приложений на Spring:

- глава 1 знакомит с фреймворками Spring и Spring Boot, а также с приемами инициализации проекта на Spring. В этой главе вы сделаете первые шаги к созданию приложения Spring, которое будете развивать на протяжении всей книги;
- глава 2 обсуждает создание веб-уровня приложения с использованием Spring MVC. Здесь вы создадите контроллеры, обрабатывающие веб-запросы, и представления, отображающие информацию в веб-браузере;

- глава 3 рассматривает серверную часть приложения Spring, обеспечивающую хранение информации в реляционной базе данных;
- глава 4 продолжает тему хранения данных и описывает приемы хранения данных в нереляционных базах данных, таких как Cassandra и MongoDB;
- глава 5 демонстрирует приемы использования Spring Security для аутентификации пользователей и предотвращения несанкционированного доступа к приложению;
- глава 6 показывает, как настроить приложение Spring с помощью Spring Boot и как выборочно применять конфигурации с помощью профилей.

Часть II охватывает темы, связанные с интеграцией приложений Spring с другими приложениями:

- глава 7 продолжает обсуждение Spring MVC, начатое в главе 2, и рассматривает вопросы создания и использования REST API в Spring;
- глава 8 показывает, как защитить API, созданные в главе 7, с помощью Spring Security и OAuth 2;
- глава 9 рассматривает приемы асинхронных взаимодействий, позволяющие приложению Spring отправлять и получать сообщения с помощью службы сообщений RabbitMQ или Kafka;
- глава 10 обсуждает декларативную интеграцию приложений с использованием Spring Integration.

Часть III исследует новую захватывающую поддержку реактивного программирования в Spring:

- глава 11 знакомит с Project Reactor – библиотекой реактивного программирования, лежащей в основе реактивных возможностей Spring 5;
- глава 12 вновь рассматривает разработку REST API и представляет Spring WebFlux – новый веб-фреймворк, который многое заимствовал из Spring MVC и предлагает новую реактивную модель для веб-разработки;
- глава 13 разбирает приемы хранения реактивных данных с помощью Spring Data в базах данных Cassandra и Mongo;
- глава 14 знакомит с RSocket, новым коммуникационным протоколом – реактивной альтернативой HTTP для создания API.

В части IV рассказывается, как подготовить и развернуть приложение в промышленном окружении:

- глава 15 знакомит с Spring Boot Actuator – расширением Spring Boot, помогающим экспортировать функции приложения Spring в виде конечных точек REST;
- глава 16 показывает, как использовать Spring Boot Admin для создания удобного браузерного приложения администрирования поверх Actuator;

- глава 17 обсуждает отображение и использование bean-компонентов Spring в виде компонентов JMX MBean;
- наконец, глава 18 рассказывает, как развернуть приложение Spring в различных производственных окружениях, включая Kubernetes.

Разработчики, плохо знакомые с фреймворком Spring, должны начинать чтение с главы 1 и последовательно работать с каждой главой. Опытные разработчики могут предпочесть читать выборочно, переходя к наиболее интересным им темам. Однако каждая следующая глава строится на предыдущей, поэтому вы можете потерять нить рассуждений, начав чтение с середины книги.

О примерах программного кода

Книга содержит множество примеров программного кода и в виде листингов, и в виде отдельных фрагментов в тексте. Этот код всегда будут оформляться моноширинным шрифтом.

Во многих случаях исходный код был переформатирован: добавлены переносы строк и отступы, чтобы примеры уместились по ширине книжной страницы. В редких случаях этого оказалось недостаточно, поэтому там, где это необходимо, я добавил стрелки, обозначающие продолжение строки (➥). Кроме того, я удалил часть комментариев из кода, если он подробно описывается в тексте. Многие листинги сопровождаются дополнительным описанием, чтобы подчеркнуть наиболее важные идеи.

Выполняемые файлы примеров можно получить на странице онлайн-версии этой книги по адресу https://livebook.manning.com/book/spring-in-action-sixth-edition. Все примеры исходного кода доступны для загрузки на веб-сайте издательства Manning по адресу https://www.manning.com/books/spring-in-action-sixth-edition, а также в репозитории GitHub http://github.com/habuma/spring-in-action-6-samples.

Форум книги

Одновременно с покупкой «Spring в действии, шестое издание» вы получаете бесплатный доступ к liveBook – онлайн-платформе издательства Manning. Используя возможности liveBook, вы сможете оставлять свои комментарии к книге в целом или к определенным разделам или абзацам, делать заметки для себя, задавать технические вопросы и отвечать на них, а также получать помощь от автора и других пользователей. Чтобы получить доступ к форуму, перейдите по ссылке https://forums.manning.com/forums/spring-in-action-sixth-edition. Дополнительные сведения о форумах Manning и правилах поведения на них можно получить по адресу https://forums.manning.com/forums/about.

Издательство Manning обязуется предоставить своим читателям место встречи, где может состояться содержательный диалог между отдельными читателями и между читателями и автором. Но со стороны автора отсутствуют какие-либо обязательства уделять форуму какое-то определенное внимание – его присутствие на форуме остается добровольным (и неоплачиваемым). Мы предлагаем задавать автору стимулирующие вопросы, чтобы его интерес не угасал!

Форум и архивы предыдущих обсуждений будут доступны на сайте издательства, пока книга находится в печати.

Другие онлайн-ресурсы

Нужна дополнительная помощь?

- На веб-сайте Spring есть несколько полезных руководств, описывающих, как начать работу (некоторые из них были написаны автором этой книги). Они доступны по адресу https://spring.io/guides.
- Сайт в Stack Overflow (теги *Spring* (https://stackoverflow.com/questions/tagged/spring) и *Spring Boot* (https://stackoverflow.com/questions/tagged/springboot)) – отличное место, где вы сможете задавать вопросы и помогать другим в освоении Spring. Помогать кому-то еще, отвечая на вопросы о Spring, – отличный способ изучения Spring!

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 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 укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и 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.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг, мы будем очень благодарны, если вы сообщите о ней главному редактору по адресу dmkpress@

gmail.com. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Manning Publications очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу электронной почты dmkpress@gmail.com.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Об авторе

Крейг Уоллс (Craig Walls) работает старшим инженером-разработчиком в VMware. Он настойчиво продвигает Spring Framework, часто выступает на встречах в местных группах пользователей и конференциях и пишет о Spring. Когда Крейг не пишет программный код, он обычно планирует свою следующую поездку в Диснейленд и проводит все свободное время со своей супругой, двумя дочерьми, тремя собаками и попугаем.

Часть I

Основы Spring

В части I этой книги вы начнете писать приложение с использованием Spring и попутно будете изучать основы данного фреймворка.

В главе 1 я дам краткий обзор основных принципов, лежащих в основе Spring и Spring Boot, и покажу, как создать проект приложения на Spring на примере создания Taco Cloud – вашего первого приложения на Spring. В главе 2 вы познакомитесь с Spring MVC и узнаете, как представлять модели данных в браузере и обрабатывать и проверять ввод в формах. Вы также получите несколько советов по выбору библиотеки шаблонов представлений. В главе 3 вы добавите в приложение Taco Cloud возможность сохранения данных, где также узнаете об использовании шаблона Spring JDBC и о том, как вставлять данные с помощью параметризованных запросов. Затем вы увидите, как объявлять репозитории JDBC (Java Database Connectivity) и JPA (Java Persistence API) с помощью Spring Data. Глава 4 продолжит рассказ о механизмах хранения данных в Spring и представит еще два модуля Spring Data для сохранения данных в Cassandra и MongoDB. Глава 5 посвящена вопросам обеспечения безопасности в приложениях Spring, включая автоматическую настройку Spring Security, определение пользовательского хранилища, настройку страницы входа и защиту от атак с подделкой межсайтовых запросов. В завершение части I, в главе 6, мы рассмотрим настройку конфигурационных свойств. Вы узнаете, как настраивать автоматически конфигурируемые bean-компоненты, как применять конфигурационные свойства к компонентам приложения и как работать с профилями Spring.

1 Введение в Spring

В этой главе рассматриваются следующие темы:

- основы Spring и Spring Boot;
- инициализация проекта Spring;
- обзор экосистемы Spring.

Греческий философ Гераклит не был известен как разработчик программного обеспечения, однако похоже, что он хорошо разбирался в этом вопросе. Вот его высказывание: «Единственная постоянная вещь – это перемены». Оно отражает фундаментальную истину разработки программного обеспечения.

Наши современные подходы к разработке приложений отличаются от подходов, использовавшихся 5, 10 и, конечно же, 20 лет тому назад, до того, как Spring Framework был представлен в книге Рода Джонсона (Rod Johnson) «Expert One-on-One J2EE Design and Development» (Wrox, 2002, http://mng.bz/oVjy).

В то время наиболее распространенными типами разрабатываемых приложений были браузерные веб-приложения, поддерживающие реляционные базы данных. Приложения этого типа по-прежнему актуальны – и Spring прекрасно подходит для их разработки, – однако в настоящее время нас больше интересует разработка приложений на основе микросервисов, предназначенных для размещения в облаке и хранящих данные в самых разных базах данных. А новый интерес к реактивному программированию направлен на обеспечение боль-

шей масштабируемости и улучшенной производительности за счет неблокирующих операций.

С развитием методик разработки программного обеспечения также менялся и Spring Framework. В нем появлялись все новые и новые средства, помогающие решать проблемы современной разработки, включая поддержку микросервисов и реактивное программирование. Кроме того, создатели Spring решили упростить модель разработки, внедрив Spring Boot.

Независимо от того, что разрабатывается – простое ли веб-приложение на основе базы данных или современное приложение на основе микросервисов, – Spring даст вам все и поможет достичь поставленных целей. Эта глава – ваш первый шаг в путешествии по современной разработке приложений с помощью Spring.

1.1 Что такое Spring?

Понимаю, что вам не терпится приступить к разработке с использованием Spring, и уверяю вас, что до окончания этой главы вы напишете простое приложение. Но сначала позвольте мне представить несколько основных идей, лежащих в основе Spring, которые помогут вам понять суть этого фреймворка.

Любое нетривиальное приложение состоит из множества компонентов, каждый из которых вносит свой вклад в общую функциональность, координируя свои действия с другими элементами приложения. Когда приложение запускается, эти компоненты должны как-то узнать о существовании друг друга.

По сути, Spring предлагает *контейнер*, часто называемый *контекстом приложения Spring*, который создает компоненты приложения и управляет ими. Эти компоненты, или bean-компоненты, объединяются внутри контекста Spring, образуя полноценное приложение, подобно тому, как кирпичи, известковый раствор, древесина, гвозди, водопроводные трубы и проводка соединяются вместе, образуя дом.

Акт объединения bean-компонентов основан на шаблоне, известном как *внедрение зависимостей* (Dependency Injection, DI). В технологии внедрения зависимостей компоненты не создают и не поддерживают жизненный цикл других компонентов, от которых они зависят, а полагаются в этом на отдельный объект (контейнер), который создаст все нужные компоненты и внедрит их в другие компоненты, которые в них нуждаются. Обычно это делается с помощью аргументов конструктора или методов доступа к свойствам.

Например, предположим, что вам нужно обратиться к двум компонентам: службе инвентаризации (для получения информации о наличии запасов на складе) и службе продукта (за сведениями о продукте). Служба продукта зависит от службы инвентаризации, получая от нее полный набор информации о продуктах. Отношения между

этими компонентами и контекстом приложения Spring иллюстрирует рис. 1.1.

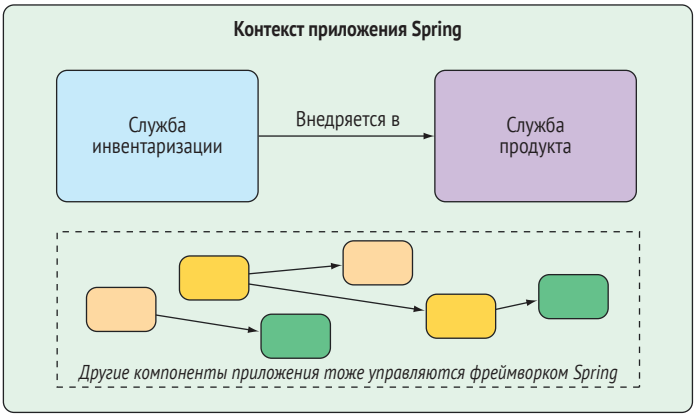


Рис. 1.1 Компоненты приложения управляются и внедряются друг в друга приложением контекста Spring

Помимо основного контейнера, Spring и сопутствующие библиотеки предлагают веб-фреймворк, различные механизмы хранения данных, фреймворк безопасности, интеграцию с другими системами, мониторинг времени выполнения, поддержку микросервисов, модель реактивного программирования и многое другое, необходимое для разработки современных приложений.

Исторически сложилось так, что способ управления контекстом приложения Spring для связывания bean-компонентов был основан на одном или нескольких XML-файлах, описывающих компоненты и их взаимосвязи с другими компонентами.

Например, следующий фрагмент XML объявляет два bean-компонента, `nventoryService` и `ProductService`, и внедряет `InventoryService` в `ProductService` через аргумент конструктора:

```
<bean id="inventoryService"
      class="com.example.InventoryService" />

<bean id="productService"
      class="com.example.ProductService" />
  <constructor-arg ref="inventoryService" />
</bean>
```

Однако в последних версиях Spring принято производить такие настройки в коде на Java в специальном классе-конфигураторе. Следующий Java-класс описывает эквивалентную конфигурацию:

```
@Configuration
public class ServiceConfiguration {
  @Bean
```

```
  public InventoryService inventoryService() {
    return new InventoryService();
  }

  @Bean
  public ProductService productService() {
    return new ProductService(inventoryService());
  }
}
```

Аннотация `@Configuration` подсказывает фреймворку Spring, что это класс конфигурации, который создает bean-компоненты для контекста Spring.

Методы класса конфигурации снабжены аннотацией `@Bean`, указывающей, что возвращаемые ими объекты должны быть добавлены в контекст приложения как bean-компоненты (где эти компоненты по умолчанию будут доступны по идентификаторам, совпадающим с именами определяющих их методов).

Определение конфигурации в Java-коде имеет определенные преимущества перед описанием в XML-файлах, в том числе более высокий уровень безопасности типов и простоту рефакторинга. Тем не менее явное описание конфигурации на Java или в XML необходимо, только если Spring не может автоматически настроить компоненты.

Автоматическая настройка уходит корнями в методы Spring, известные как *автоматическое связывание* (autowiring) и *сканирование компонентов*. Используя механизм сканирования, Spring может автоматически обнаруживать компоненты в пути поиска классов (classpath) приложения и создавать их как bean-компоненты в контексте приложения Spring. Механизм автоматического связывания позволяет фреймворку Spring внедрять компоненты в другие bean-компоненты, от которых те зависят.

Совсем недавно, с появлением Spring Boot, автоматическая настройка вышла далеко за рамки сканирования компонентов и автоматического связывания. Spring Boot – это расширение для Spring Framework, предлагающее несколько улучшений. Наиболее известным из них является *автоконфигурация* – Spring Boot может делать обоснованные предположения о том, какие компоненты следует настроить и связать вместе, опираясь на элементы в пути поиска классов, переменные окружения и другие факторы.

Я хотел бы показать вам пример кода, демонстрирующий автоконфигурацию, но не могу. Автоконфигурация похожа на ветер – вы можете видеть ее последствия, но у нее нет кода, который можно показать вам и сказать: «Смотрите! Вот пример автоконфигурации!» Механизм автоконфигурации просто работает, создает и связывает компоненты – и не требует для этого писать какой-либо код. Именно отсутствие необходимости писать код делает автоконфигурацию такой замечательной.

Автоконфигурация, предлагаемая Spring Boot, значительно сократила объем явного описания конфигурации (с помощью XML или на Java), необходимого для создания приложения. Фактически, закончив пример приложения в этой главе, вы получите действующее приложение Spring, содержащее только одну строку описания конфигурации!

Spring Boot настолько расширяет возможности разработки Spring, что без него трудно представить разработку Spring-приложений. По этой причине мы будем рассматривать Spring и Spring Boot как одно и то же. Мы будем максимально использовать Spring Boot, а явное описание конфигурации будем добавлять только там, где это действительно необходимо. А поскольку описание конфигурации в XML – это старый способ работы со Spring, мы будем описывать конфигурацию приложений Spring почти исключительно в коде на Java.

Но довольно болтовни. В названии этой книги есть слова «в действии», так что давайте будем действовать и начнем писать наше первое приложение на Spring.

1.2 Инициализация приложения Spring

На протяжении всей книги мы с вами будем создавать онлайн-приложение Taco Cloud для заказа самой замечательной еды, созданной человеком, – тако. Конечно, для достижения этой цели мы будем использовать Spring, Spring Boot и множество связанных с ними библиотек и фреймворков.

Инициализировать проекты приложений Spring можно несколькими способами. Я мог бы показать вам все этапы создания структуры каталогов проекта и определения спецификации сборки вручную, но, по большому счету, это пустая трата времени, которое лучше потратить на написание кода. Поэтому мы доверимся Spring Initializr.

Spring Initializr – это веб-приложение, помогающее создать скелетную структуру проекта Spring, которую вы затем сможете наполнить любой функциональностью, какой захотите. Вот несколько способов использования Spring Initializr:

- открыть в браузере страницу http://start.spring.io;
- обратиться к приложению с помощью утилиты `curl`;
- воспользоваться интерфейсом командной строки Spring Boot;
- создать новый проект с помощью Spring Tool Suite;
- создать новый проект с помощью IntelliJ IDEA;
- создать новый проект с помощью Apache NetBeans.

Чтобы не тратить время на обсуждение каждого из этих вариантов в этой главе, я вынес все детали в приложение. А в этой главе и на протяжении всей книги я буду показывать, как создать новый проект, используя мой любимый вариант: с помощью поддержки Spring Initializr в Spring Tool Suite.

Как следует из названия, Spring Tool Suite – это среда разработки (Integrated Development Environment, IDE) на Spring, которая поставляется в форме расширений для Eclipse, Visual Studio Code или Theia IDE. Готовые к использованию двоичные файлы Spring Tool Suite можно получить по адресу https://spring.io/tools. Spring Tool Suite предлагает удобный инструмент Spring Boot Dashboard, позволяющий легко запускать, перезапускать и останавливать приложения Spring Boot из среды разработки.

Даже если вы не хотите использовать Spring Tool Suite, ничего страшного, мы все равно можем остаться друзьями. Перейдите к приложению в конце книги и выберите вариант, который вам больше по душе. Но знайте, что в этой книге я могу время от времени упоминать инструменты, имеющиеся только в Spring Tool Suite, такие как Spring Boot Dashboard. Если вы не используете Spring Tool Suite, то вам придется адаптировать мои инструкции к вашей среде разработки.

1.2.1 Инициализация проекта Spring с помощью Spring Tool Suite

Чтобы начать работу над новым проектом Spring в Spring Tool Suite, откройте меню **File** (Файл) и выберите пункт **New** (Создать), а затем **Spring Starter Project** (Новый проект Spring). На рис. 1.2 для ясности показана структура меню.

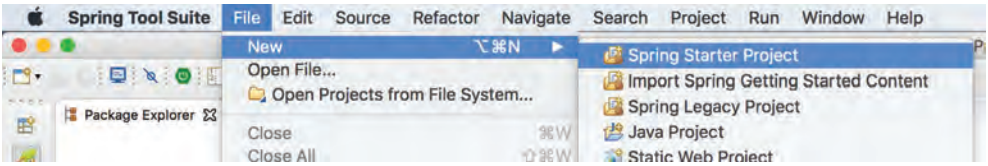


Рис. 1.2 Создание нового проекта с помощью Initializr в Spring Tool Suite

После выбора пункта меню **Spring Starter Project** (Новый проект Spring) появится окно мастера создания нового проекта (рис. 1.3). На первой странице мастер попросит ввести некоторую общую информацию о проекте: имя проекта, описание и т. д. Если вы знакомы с содержимым файла pom.xml для Maven, то в большинстве полей без труда опознаете элементы спецификации сборки Maven. Для приложения Taco Cloud заполните поля, как показано на рис. 1.3, и щелкните на кнопке **Next** (Далее).

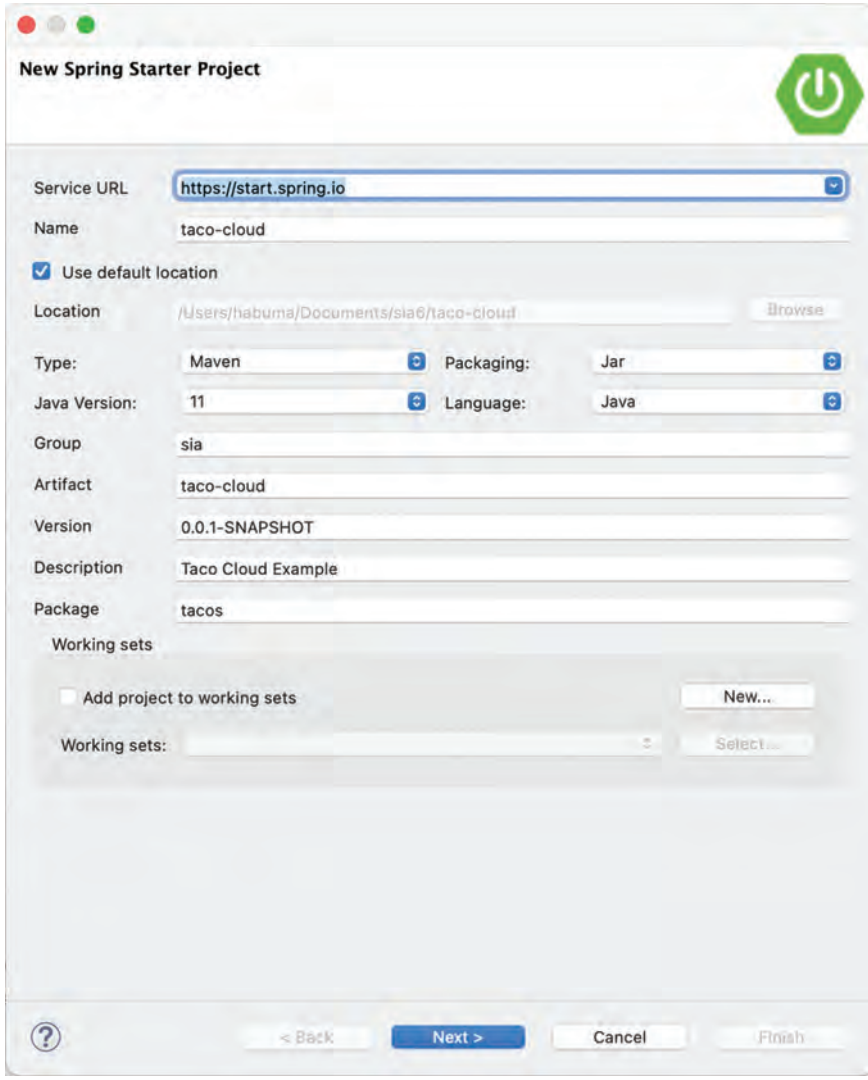


Рис. 1.3 Заполните поля информацией о проекте приложения Taco Cloud

На следующей странице мастера нужно выбрать зависимости для проекта (рис. 1.4). Обратите внимание, что в верхней части диалога можете указать версию Spring Boot, на которой должен основываться проект. По умолчанию используется самая последняя доступная версия. Как правило, рекомендуется оставлять значения по умолчанию, если только вам не нужна какая-то конкретная версия.

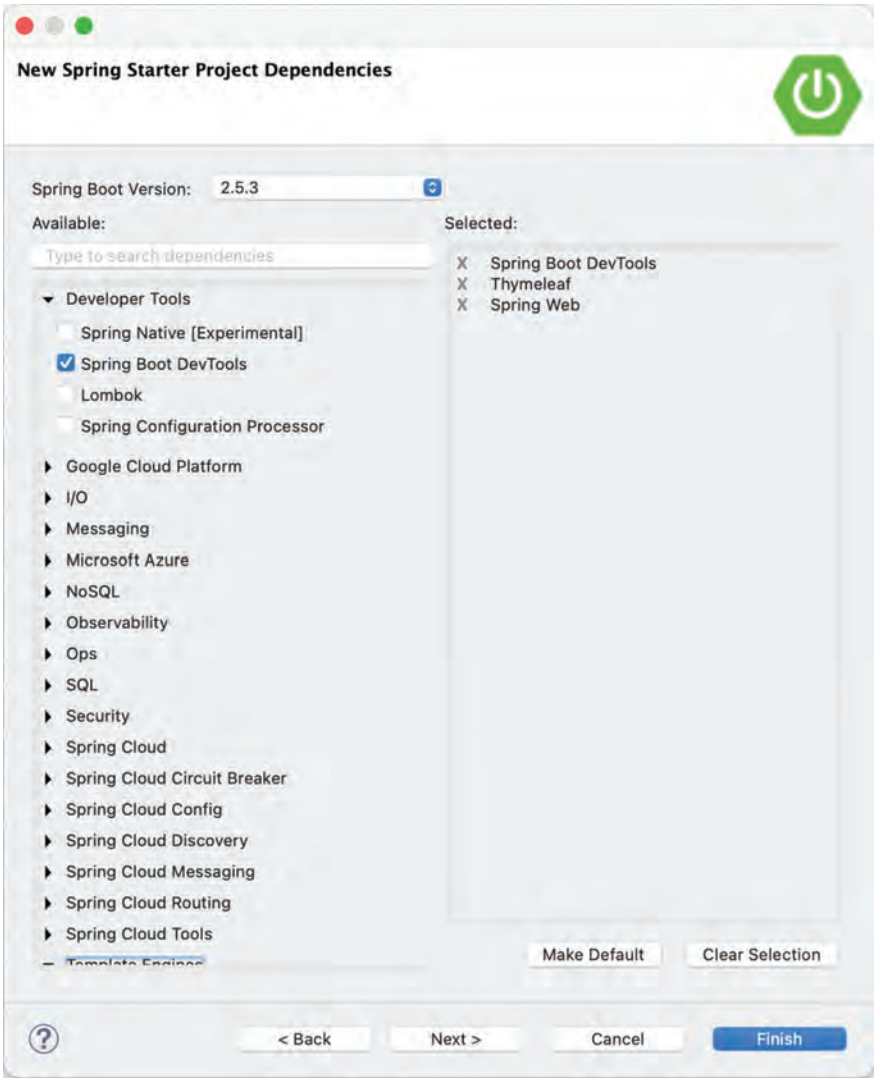


Рис. 1.4 Выбор зависимостей для проекта

При настройке зависимостей можно распахивать различные разделы и выбирать нужные зависимости вручную или искать их с помощью поля **Available** (Доступно) вверху. Для приложения Taco Cloud выберите зависимости, как показано на рис. 1.4.

Теперь можно щелкнуть не кнопке **Finish** (Готово), чтобы сгенерировать проект и добавить его в свое рабочее пространство. Но если вам хочется приключений, щелкните на кнопке **Next** (Далее) еще раз, чтобы увидеть последнюю страницу мастера создания нового проекта, как показано на рис. 1.5.

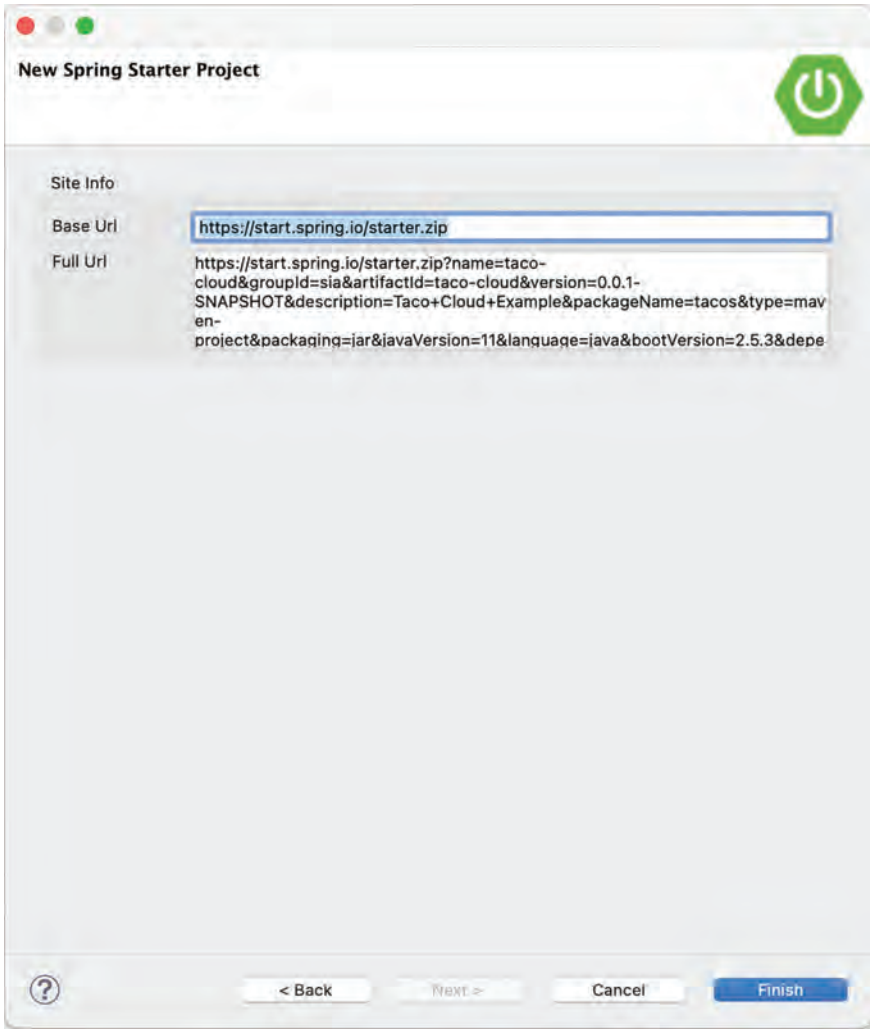


Рис. 1.5 Ввод необязательного альтернативного адреса Initializr

По умолчанию мастер создания нового проекта вызывает веб-приложение Spring Initializr, обращаясь по адресу http://start.spring.io. Обычно нет необходимости переопределять это значение по умолчанию, поэтому можно смело щелкнуть на кнопке **Finish** (Готово) на второй странице мастера. Но если по какой-то причине вы вынуждены использовать свой клон Initializr (локальную копию на своем компьютере или настроенный клон, работающий в локальной сети вашей компании), то вы можете изменить поле **Base Url** (Базовый URL), подставив адрес вашего экземпляра Initializr.

После щелчка на кнопке **Finish** (Готово) проект будет загружен из Initializr и размещен в вашей рабочей области. Подождите несколько

секунд, пока закончится загрузка и сборка, и после этого вы будете готовы приступить к разработке приложения. Но сначала давайте посмотрим, что нам дал Initializr.

1.2.2 Структура проекта Spring

После загрузки проекта в среду разработки распахните его, чтобы увидеть содержимое. На рис. 1.6 показано содержимое проекта Taco Cloud в Spring Tool Suite.

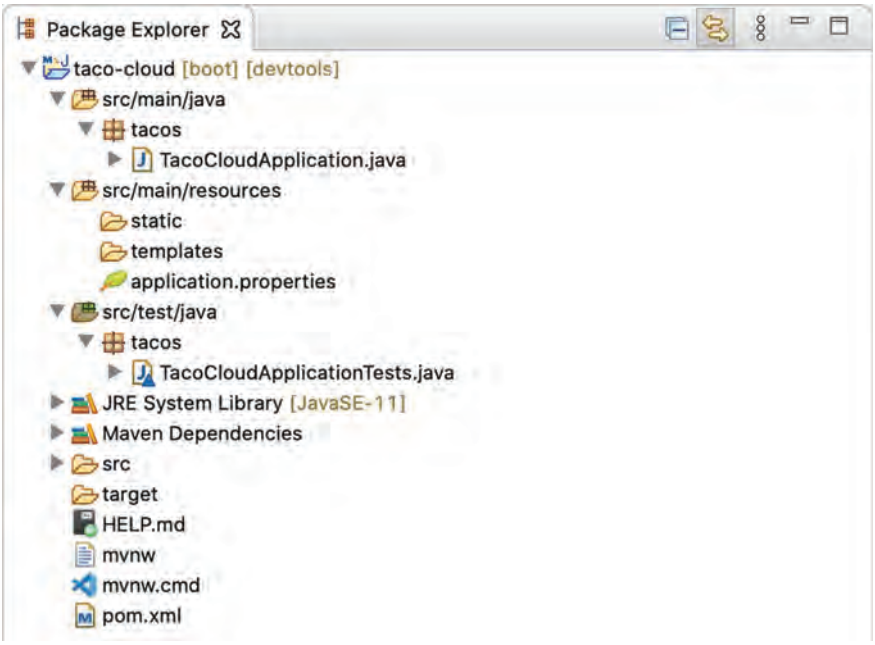


Рис. 1.6 Начальная структура проекта в Spring Tool Suite

Как видите, проект имеет типичную структуру проекта Maven или Gradle, в которой исходный код помещается в каталог *src/main/java*, код тестов – в *src/test/java*, а ресурсы, не являющиеся Java-ресурсами, – в *src/main/resources*. Обратите внимание на следующие элементы внутри этой структуры:

- `mvnw` и `mvnw.cmd` – это сценарии-обертки Maven, эти сценарии можно использовать для создания нового проекта, даже если на вашем компьютере не установлен Maven;
- *pom.xml* – это параметры сборки Maven, вскоре мы рассмотрим ее поближе;
- `TacoCloudApplication.java` – это основной класс Spring Boot, который запускает проект, вскоре мы уделим ему наше внимание;
- *application.properties* – это файл, изначально пустой, используется для определения конфигурационных свойств. Мы не будем

особенно углубляться в него в данной главе, но подробно обсудим в главе 6;

- *static* – в эту папку можно поместить любой статический контент (изображения, таблицы стилей, JavaScript и т. д.), который должен отображаться в браузере. Изначально в этой папке ничего нет;
- *templates* – тут мы разместим файлы шаблонов, используемые для отображения контента в браузере. Изначально в этой папке ничего нет, но скоро мы добавим в нее шаблон Thymeleaf;
- *TacoCloudApplicationTests.java* – это простой тестовый класс, проверяющий успех загрузки контекста приложения Spring. Мы будем добавлять в него дополнительные тесты по мере разработки приложения.

Развивая приложение Taco Cloud, мы будем заполнять эту базовую структуру проекта кодом на Java, изображениями, таблицами стилей, тестами и другими необходимыми ресурсами. А пока поближе рассмотрим некоторые элементы, которые создало веб-приложение Spring Initializr.

Параметры сборки

Заполняя форму Initializr, мы указали, что сборка проекта будет производиться с помощью Maven. Поэтому приложение Spring Initializr создало файл *pom.xml*, уже заполненный необходимыми параметрами. Его содержимое показано в листинге 1.1.

Листинг 1.1 Параметры сборки для Maven

```
<?xml version="1.0" encoding="UTF-8"?><project
      xmlns="http://maven.apache.org/POM/4.0.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
      https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.5.3</version>
    <relativePath /> 
  </parent>
  <groupId>sia</groupId>
  <artifactId>taco-cloud</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>taco-cloud</name>
  <description>Taco Cloud Example</description>

  <properties>
    <java.version>11</java.version>
  </properties>

  <dependencies>
    <dependency>
```

Версия Spring Boot

Зависимости инструмента Starter

```
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-devtools</artifactId>
      <scope>runtime</scope>
      <optional>true</optional>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
      <exclusions>
        <exclusion>
          <groupId>org.junit.vintage</groupId>
          <artifactId>junit-vintage-engine</artifactId>
        </exclusion>
      </exclusions>
    </dependency>

  </dependencies>

  <build>
    <plugins>
      <plugin>                          Плагин Spring Boot
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>

  <repositories>
    <repository>
      <id>spring-milestones</id>
      <name>Spring Milestones</name>
      <url>https://repo.spring.io/milestone</url>
    </repository>
  </repositories>

  <pluginRepositories>
    <pluginRepository>
      <id>spring-milestones</id>
      <name>Spring Milestones</name>
      <url>https://repo.spring.io/milestone</url>
    </pluginRepository>
  </pluginRepositories>
</project>
```

Первое, на что следует обратить внимание, – это элемент `<parent>` и его дочерний элемент `<version>`. Он сообщает, что ваш проект имеет родительский файл POM[1] `spring-boot-starter-parent`. Помимо прочего, родительский POM определяет зависимости от нескольких библиотек, часто используемых в проектах Spring. Для библиотек, определяемых в родительском POM, не нужно указывать версию, потому что она наследуется от родителя. Версия `2.5.6` соответствует Spring Boot 2.5.6, соответственно, управление зависимостями осуществляется, как определено в этой версии Spring Boot. Среди прочего Spring Boot 2.5.6 определяет базовую версию ядра Spring Framework – 5.3.12.

Поскольку речь зашла о зависимостях, обратите внимание на четыре зависимости, объявленные в элементе `<dependencies>`. Первые три должны показаться вам знакомыми. Они напрямую соответствуют зависимостям `Spring Web`, `Thymeleaf` и `Spring Boot DevTools`, которые мы выбрали в мастере создания нового проекта Spring Tool Suite. Четвертая зависимость подключает средства тестирования. Чтобы ее включить, не нужно ставить галочку, потому что Spring Initializr предполагает (надеюсь, правильно), что вы будете писать тесты.

Также обратите внимание, что все зависимости, кроме DevTools, имеют слово *starter* в идентификаторе артефакта. Зависимости от Spring Boot Starter отличаются тем, что они обычно не имеют своего библиотечного кода, а транзитивно подключают другие библиотеки. Эти зависимости предлагают следующие основные преимущества:

- благодаря им файл сборки получится меньше и им будет легче управлять, поскольку отпадает необходимость объявлять зависимости для каждой библиотеки, которая вам может понадобиться;
- зависимости можно рассматривать с точки зрения предоставляемых ими возможностей, а не с точки зрения имен их библиотек. Разрабатывая веб-приложение, вам достаточно добавить зависимость `spring-boot-starter-web` вместо длинного списка библиотек, которые позволят вам написать веб-приложение;
- освобождают от беспокойства о версиях библиотек. Вы можете быть уверены, что версии библиотек, добавленных транзитивно, будут совместимы с данной версией Spring Boot. Вам остается только определиться с версией Spring Boot.

В конце файла с параметрами сборки определяется плагин Spring Boot. Он выполняет несколько важных функций, в том числе:

- определяет цель Maven, которая позволяет запускать приложение с помощью Maven;

---

[1] POM (Project Object Model – объектная модель проекта) – базовый модуль Maven, специальный XML-файл, который всегда хранится в базовом каталоге проекта и имеет имя pom.xml. Файл POM содержит информацию о проекте и различных параметры конфигурации, которые используются Maven для сборки.

- гарантирует включение в выполняемый файл JAR всех библиотек-зависимостей и их доступность в пути поиска классов classpath во время выполнения;
- создает файл манифеста в архиве JAR, который определяет класс начальной загрузки (в вашем случае `TacoCloudApplication`) в качестве главного класса для выполняемого файла JAR.

Давайте рассмотрим класс начальной загрузки поближе.

Загрузка приложения

Поскольку приложение будет запускаться из выполняемого JAR-файла, важно выбрать главный класс, который будет выполняться при запуске этого JAR-файла. Нам также потребуется хотя бы минимальная конфигурация Spring для начальной загрузки приложения. Все это находится в классе `TacoCloudApplication`, показанном в листинге 1.2.

Листинг 1.2 Класс начальной загрузки для проекта Taco Cloud

```
package tacos;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class TacoCloudApplication {

  public static void main(String[] args) {
    SpringApplication.run(TacoCloudApplication.class, args);
  }
}
```

Приложение Spring Boot

Запуск приложения

Несмотря на небольшой размер, класс `TacoCloudApplication` играет очень важную роль. Одна из самых важных строк – одновременно одна из самых коротких. Аннотация `@SpringBootApplication` ясно указывает, что это приложение Spring Boot. Но этим функции аннотации `@SpringBootApplication` не ограничиваются.

`@SpringBootApplication` – это составная аннотация, объединяющая три другие аннотации:

- *`@SpringBootConfiguration`* – определяет этот класс как класс конфигурации. В данный момент в этом классе не определяется никаких конфигурационных параметров, но если понадобится, в него можно добавить настройки Spring Framework. Эта аннотация, по сути, является специализированной формой аннотации `@Configuration`;
- *`@EnableAutoConfiguration`* – включает автоконфигурацию Spring Boot. Подробнее об автоконфигурации мы поговорим позже, а пока просто имейте в виду, что эта аннотация сообщает Spring Boot о необходимости автоматически настраивать любые компоненты, которые могут вам понадобиться;

- `@ComponentScan` – включает сканирование компонентов. Механизм сканирования позволяет объявлять другие классы с аннотациями, такими как `@Component`, `@Controller` и `@Service`, чтобы фреймворк Spring автоматически обнаруживал и регистрировал их как компоненты в контексте приложения Spring.

Другой важной частью `TacoCloudApplication` является метод `main()`. Этот метод будет вызываться в момент запуска файла JAR. В большинстве приложений данный метод содержит шаблонный код; каждое написанное вами приложение Spring Boot будет иметь метод `main()`, аналогичный или идентичный этому (несмотря на различия в именах классов).

Метод `main()` вызывает статический метод `run()` класса `SpringApplication`, который выполняет фактическую загрузку приложения, создавая контекст приложения Spring. Метод `run()` принимает два параметра: класс конфигурации и аргументы командной строки. В общем случае совсем необязательно, чтобы имя класса конфигурации, передаваемого в вызов `run()`, совпадало с именем класса начальной загрузки, но это наиболее удобный и распространенный выбор.

В большинстве случаев вам не придется ничего менять в классе начальной загрузки. Для простых приложений может быть удобно настроить один или два других компонента в классе начальной загрузки, но для большинства приложений лучше создать отдельный класс конфигурации и настраивать в нем все, что не настраивается автоматически. В этой книге мы определим несколько классов конфигурации, так что не теряйте нить рассуждений.

Тестирование приложения

Тестирование – важная часть разработки программного обеспечения. Вы всегда можете протестировать свой проект вручную, создав его, а затем запустив тестирование из командной строки:

```
$ ./mvnw package
...
$ java -jar target/taco-cloud-0.0.1-SNAPSHOT.jar
```

Или, благодаря использованию Spring Boot, плагин Spring Boot Maven упрощает этот шаг еще больше:

```
$ ./mvnw spring-boot:run
```

Но тестирование вручную подразумевает участие человека и, как следствие, вероятность потенциальных человеческих ошибок и непоследовательность тестирования. Автоматизированные тесты более последовательны и воспроизводимы.

Учитывая это, Spring Initializr предоставляет заготовку тестового класса, как показано в листинге 1.3.

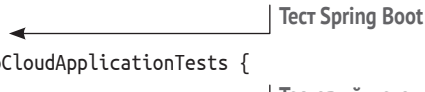
Листинг 1.3 Заготовка класса для тестирования приложения

```
package tacos;

import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
public class TacoCloudApplicationTests {

  @Test
  public void contextLoads() {
  }
}
```



В `TacoCloudApplicationTests` практически ничего нет: единственный тестовый метод в классе пуст. И все же этот тестовый класс выполняет важную проверку – успешность загрузки контекста приложения Spring. Если вы внесете какие-либо изменения, препятствующие созданию контекста приложения Spring, то этот тест завершится ошибкой, и вы сможете отреагировать на нее, устранив проблему.

Аннотация `@SpringBootTest` настраивает JUnit на запуск теста с поддержкой возможностей Spring Boot. Подобно `@SpringBootApplication`, `@SpringBootTest` – это составная аннотация, которая сама снабжена аннотацией `@ExtendWith(SpringExtension.class)`, добавляющей поддержку возможностей тестирования Spring в JUnit 5. Однако вы пока можете думать о ней как об эквиваленте вызова `SpringApplication.run()` в методе `main()`. В этой книге вы несколько раз встретитесь с аннотацией `@SpringBootTest` и мы еще будем обсуждать некоторые ее возможности.

Наконец, в классе есть тестовый метод. Аннотация `@SpringBootTest` выполняет загрузку контекста приложения Spring для теста, но сам тестовый класс ничего не будет делать, если в нем не будет тестовых методов. Даже в таком виде, без всяких проверок и любого другого кода, этот пустой тестовый метод выполнит свою работу за счет аннотаций и загрузит контекст приложения Spring. Если при этом возникнут какие-либо проблемы, тест сообщит об ошибке.

Этот и любые другие тестовые классы можно запустить из командной строки, используя следующее заклинание Maven:

```
$ ./mvnw test
```

На этом мы завершаем обзор кода, сгенерированного веб-приложением Spring Initializr. Вы познакомились с основой, которую можно использовать для разработки приложения Spring, но до сих пор не написали ни строчки кода. Теперь пришло время запустить вашу среду разработки, стряхнуть пыль с клавиатуры и добавить свой код в приложение Taco Cloud.

1.3 Разработка приложения Spring

Поскольку вы только в начале пути, начнем с относительно небольшого изменения в приложении Taco Cloud, но оно продемонстрирует многие достоинства Spring. Вполне уместно, если первым вашим шагом – первой функцией в приложении Taco Cloud – станет домашняя страница. Для этого мы добавим следующие два артефакта:

- класс контроллера, обрабатывающий запросы к домашней странице;
- шаблон представления, определяющий внешний вид домашней страницы.

А поскольку в разработке ПО важную роль играет тестирование, мы напишем также простой тестовый класс для проверки домашней страницы. Но обо всем по порядку... Начнем с класса контроллера.

1.3.1 Обработка веб-запросов

В состав Spring входит мощная веб-платформа, известная как Spring MVC. В основе Spring MVC лежит идея *контроллера* – класса, обрабатывающего запросы и возвращающего некоторую информацию. В веб-приложениях контроллер отвечает, заполняя, при необходимости, модель данных и передавая запрос представлению для создания разметки HTML, которая возвращается браузеру.

Подробнее о Spring MVC мы поговорим в главе 2, а пока просто напишем простой класс контроллера, который обрабатывает запросы с корневым путем (например, /) и передает их представлению домашней страницы без заполнения модели данных. В листинге 1.4 показан простой класс контроллера.

Листинг 1.4 Контроллер домашней страницы

```
package tacos;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class HomeController {

  @GetMapping("/")
  public String home() {
    return "home";
  }
}
```

Контроллер

Обрабатывает запросы с корневым путем /

Возвращает имя представления

Как видите, этот класс снабжен аннотацией `@Controller`. Сама по себе эта аннотация почти ничего не делает. Ее основная цель – идентифицировать класс как компонент, доступный механизму сканиро-

вания компонентов. Поскольку `HomeController` снабжен аннотацией `@Controller`, механизм сканирования в Spring автоматически обнаружит его и создаст экземпляр `HomeController` как bean-компонент в контексте приложения Spring.

Той же цели служат еще несколько аннотаций (включая `@Component`, `@Service` и `@Repository`). Мы могли бы аннотировать класс `HomeController` любой из этих других аннотаций, и он все равно работал бы так же. Однако аннотация `@Controller` лучше описывает роль этого компонента в приложении.

Метод `home()` так же прост, как всякие другие методы контроллера. Он снабжен аннотацией `@GetMapping`, указывающей, что этот метод обрабатывает HTTP-запросы `GET` с корневым путем `/`. При этом он ничего не делает, кроме как возвращает строковое значение `home`.

Это значение интерпретируется как логическое имя представления. Особенности реализации этого представления зависят от нескольких факторов, но поскольку Thymeleaf находится в пути поиска классов, мы можем определить этот шаблон с помощью Thymeleaf.

> **Почему Thymeleaf?**
>
> Возможно, вам интересно, почему из всех механизмов шаблонов я выбрал именно Thymeleaf. Почему не JSP? Почему не FreeMarker? Почему не любой другой из множества вариантов?
>
> В моем выборе нет потаенного смысла, я должен был что-то выбрать, а мне нравится Thymeleaf, и обычно я предпочитаю его другим вариантам. Многим очевидным выбором может показаться JSP, но при использовании JSP с фреймворком Spring Boot необходимо преодолеть некоторые проблемы, а я не хотел спускаться в кроличью нору в главе 1. Но не переживайте, мы рассмотрим другие варианты механизмов шаблонов, включая JSP, в главе 2.

Имя шаблона состоит из имени логического представления, к которому добавляются префикс пути */templates/* и расширение *.html*. В результате полный путь к шаблону будет иметь вид: */templates/home.html*. Это означает, что шаблон должен находиться в папке проекта */src/main/resources/templates/home.html*. Давайте создадим этот шаблон сейчас.

1.3.2 Определение представления

Наша домашняя страница будет простой и содержать лишь приветствие посетителя сайта. В листинге 1.5 показан простой шаблон Thymeleaf, определяющий домашнюю страницу Taco Cloud.

Листинг 1.5 Шаблон домашней страницы Taco Cloud

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">
  <head>
    <title>Taco Cloud</title>
  </head>

  <body>
    <h1>Welcome to...</h1>
    <img th:src="@{/images/TacoCloud.png}"/>
  </body>
</html>
```

Здесь почти нечего обсуждать. Единственная примечательная строка – строка с тегом `<img>`, отображающая логотип Taco Cloud. В ней присутствует Thymeleaf-атрибут `th:src` и выражение `@{...}`, описывающее ссылку на изображение с контекстно-зависимым путем. В остальном это самая обычная страница Hello World.

Давайте поговорим немного об изображении логотипа. Вы можете определить свое изображение логотипа. Но вы должны поместить его в нужное место в проекте.

На изображение ссылается контекстно-зависимый путь */images/TacoCloud.png*. Как вы помните из нашего обзора структуры проекта, статический контент, такой как изображения, хранится в папке */src/main/resources/static*. То есть изображение логотипа Taco Cloud также должно находиться в проекте в пути */src/main/resources/static/images/TacoCloud.png*.

Теперь, когда у нас есть контроллер для обработки запросов к домашней странице и шаблон представления для ее отображения, мы почти готовы запустить приложение и увидеть его в действии. Но сначала посмотрим, как написать тест для контроллера.

1.3.3 Тестирование контроллера

Тестирование веб-приложений может быть сложной задачей, потому что иногда требует проверки содержимого HTML-страницы. К счастью, Spring имеет мощные средства тестирования, упрощающие тестирование веб-приложений.

Для нашей домашней страницы мы напишем тест, сравнимый по сложности с самой домашней страницей. Наш тест (листинг 1.6) выполнит HTTP-запрос `GET` с корневым путем `/` и затем убедится, что контроллер выбрал имя представления `home`, а сама страница содержит фразу «Welcome to...».

Листинг 1.6 Тест для контроллера домашней страницы

```
package tacos;

import static org.hamcrest.Matchers.containsString;
import static
    org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static
    org.springframework.test.web.servlet.result.MockMvcResultMatchers.content;
import static
    org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;
import static
    org.springframework.test.web.servlet.result.MockMvcResultMatchers.view;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.test.web.servlet.MockMvc;

@WebMvcTest(HomeController.class)                    <-- Тест для HomeController
public class HomeControllerTest {

  @Autowired
  private MockMvc mockMvc;                           <-- Внедрить MockMvc

  @Test
  public void testHomePage() throws Exception {
    mockMvc.perform(get("/"))                        <-- Выполнить запрос GET /
      .andExpect(status().isOk())                    <-- Ожидается код ответа HTTP 200
      .andExpect(view().name("home"))                <-- Ожидается имя представления home
      .andExpect(content().string(                   <-- Ожидается наличие строки
          containsString("Welcome to...")));             «Welcome to...»
  }
}
```

Первое, что бросается в глаза, – этот тест немного отличается от класса `TacoCloudApplicationTests` применяемыми к нему аннотациями. Вместо `@SpringBootTest` класс `HomeControllerTest` снабжен аннотацией `@WebMvcTest`. Это специальная тестовая аннотация из Spring Boot, которая организует запуск теста в контексте приложения Spring MVC. В данном случае она обеспечивает регистрацию класса `HomeController` в Spring MVC, чтобы дать возможность отправлять ему запросы.

`@WebMvcTest` также настраивает поддержку тестирования Spring MVC в Spring. Для тестирования можно было бы запустить сервер, но для наших целей вполне достаточно имитировать механику Spring MVC. В тестовый класс внедряется объект `MockMvc`, чтобы тест мог управлять фиктивным объектом.

Метод `testHomePage()` определяет тест для проверки домашней страницы. Он начинается с вызова объекта `MockMvc` для выполнения HTTP-запроса `GET` с `/` (корневым путем). В числе результатов выполнения этого запроса ожидается следующее:

- ответ должен иметь статус HTTP 200 (OK);
- представление должно иметь логическое имя `home`;
- получившаяся страница должна содержать текст «Welcome to...».

Тест можно запустить в среде разработки или с помощью Maven:

```
$ mvnw test
```

Если после выполнения запроса объектом `MockMvc` какое-либо из этих ожиданий не будет выполнено, тест завершится неудачей. Но наш контроллер и шаблон представления написаны так, чтобы удовлетворить эти ожидания, поэтому тест должен выполниться успешно или, по крайней мере, с некоторым оттенком зеленого, указывающим на успешное прохождение.

Контроллер написан, шаблон представления создан, и тест пройден. Похоже, мы успешно справились с реализацией домашней страницы. Но, кроме успешного прохождения теста, есть еще один приятный момент – наглядный результат в браузере. В конце концов, именно его увидят клиенты Taco Cloud. А теперь соберем приложение и запустим его.

1.3.4 Сборка и запуск приложения

У нас есть не только несколько способов инициализации приложения Spring, но и несколько способов запустить его. Если хотите, вы можете перейти к приложению в конце книги и прочитать о некоторых наиболее распространенных способах запуска приложений Spring Boot.

Поскольку мы решили использовать Spring Tool Suite для инициализации проекта и работы над ним, то воспользуемся удобным инструментом под названием Spring Boot Dashboard, который поможет запустить приложение в среде разработки. Spring Boot Dashboard отображается в виде вкладки, обычно в левом нижнем углу окна IDE. На рис. 1.7 показан скриншот Spring Boot Dashboard.

Я не буду тратить много времени на изучение всего, что делает Spring Boot Dashboard, хотя на рис. 1.7 показаны некоторые из наиболее полезных деталей. Сейчас важно знать, как использовать этот инструмент для запуска приложения Taco Cloud. Выберите приложение `taco-cloud` в списке проектов (это единственное приложение, показанное на рис. 1.7), а затем щелкните на кнопке запуска (крайняя левая кнопка с зеленым треугольником и красным квадратом). Приложение должно запуститься сразу после щелчка.

Когда приложение запустится, вы увидите, как в консоли пролетит некоторый логотип Spring, оформленный символами ASCII, а за ним последуют отладочные строки, описывающие шаги, выполняемые при запуске приложения. В их числе вы увидите строку, сообщающую, что сервер Tomcat запущен и прослушивает порт 8080 (http). Это означает, что теперь можно запустить веб-браузер и открыть домашнюю страницу приложения, чтобы увидеть плоды своего труда.

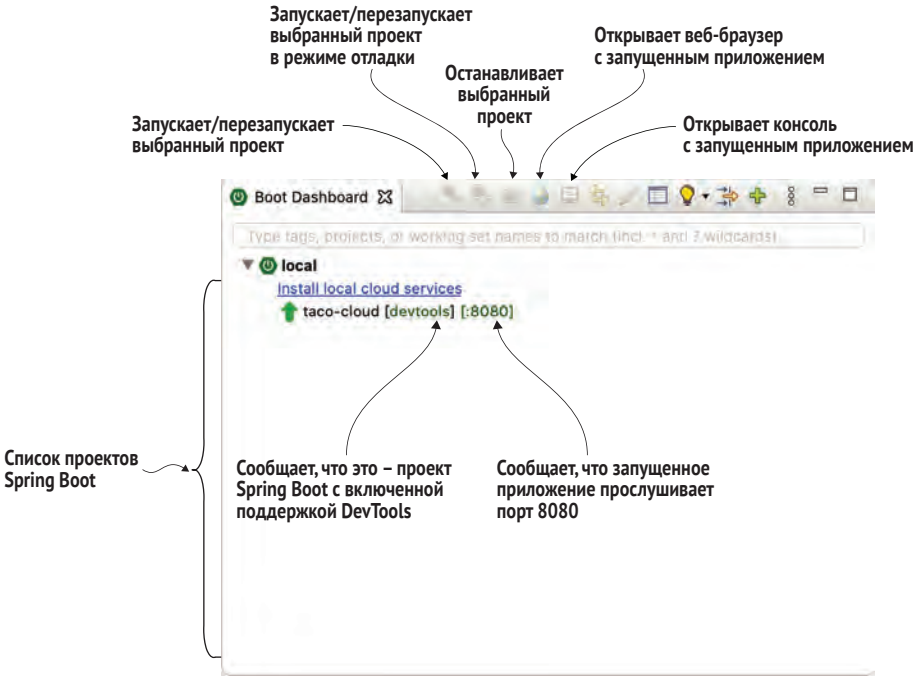


Рис. 1.7 Spring Boot Dashboard

Но постойте! Tomcat запустился? Когда это мы успели развернуть приложение на веб-сервере Tomcat?

Приложения Spring Boot, как правило, несут в себе все, что им нужно, и не требуют развертывания на каком-либо сервере приложений. Мы не развертывали свое приложение на Tomcat – Tomcat является частью нашего приложения! (Я подробно опишу, как Tomcat стал частью приложения, в разделе 1.3.6.)

Теперь, когда приложение запущено, введите в веб-браузере адрес `http://localhost:8080` (или щелкните на кнопке с изображением глобуса на панели инструментов Spring Boot), и вы должны увидеть страницу, как показано на рис. 1.8. У вас страница может отличаться, если вы использовали свое изображение логотипа, но в остальном она не должна отличаться от рис. 1.8.

Страница выглядит незамысловато. Но и эта книга посвящена не графическому дизайну. На данный момент такой скромной домашней страницы более чем достаточно. И она дает нам хорошую отправную точку для дальнейшего знакомства со Spring.

Единственное, что я упустил из виду, – это набор инструментов разработчика DevTools. Мы выбрали его как зависимость при инициализации вашего проекта, и он отображается как зависимость в сгенерированном файле *pom.xml*. А панель Spring Boot Dashboard даже показывает, что в проекте включена поддержка DevTools. Но что это за инструменты DevTools и что они нам дают? Давайте кратко рассмотрим некоторые наиболее полезные функции DevTools.

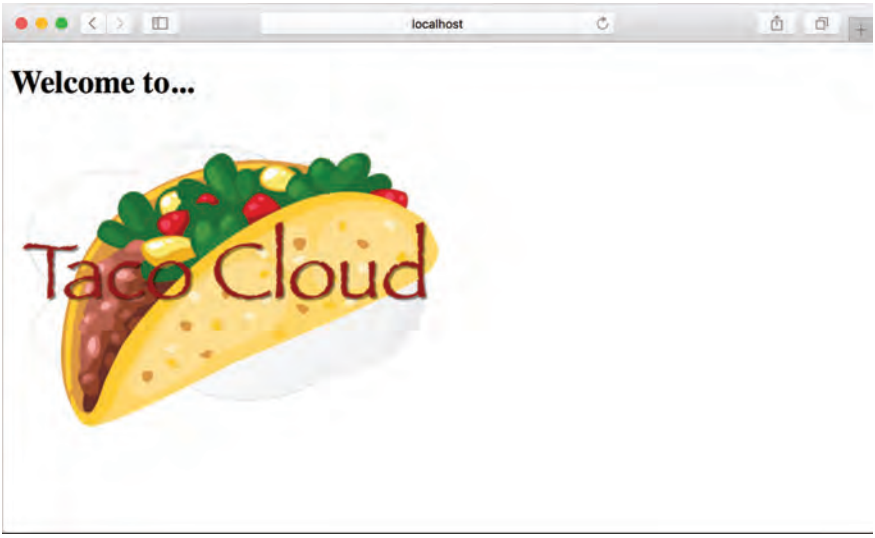


Рис. 1.8 Домашняя страница Taco Cloud

1.3.5 Spring Boot DevTools

Как следует из названия, DevTools – это набор дополнительных инструментов разработчика приложений на Spring, позволяющих:

- автоматически перезапускать приложение при изменении кода;
- автоматически обновлять окно браузера при изменении ресурсов, передаваемых браузеру (таких как шаблоны, сценарии на JavaScript, таблицы стилей и т. д.);
- автоматически отключать кеширование шаблонов;
- открывать встроенную консоль H2, если используется база данных H2.

Важно понимать, что DevTools не является плагином IDE и не требует использования конкретной IDE. Эти инструменты одинаково хорошо работают в Spring Tool Suite, IntelliJ IDEA и NetBeans. Кроме того, будучи предназначенными для разработки, они достаточно интеллектуальны, чтобы отключать себя при развертывании в промышленном окружении. Мы обсудим, как это делается, когда приступим к развертыванию приложения в главе 18. А пока сосредоточимся на наиболее полезных функциях Spring Boot DevTools и начнем с автоматического перезапуска приложения.

Автоматический перезапуск приложения

Наличие DevTools в составе проекта позволяет вносить изменения в код на Java и в файлы свойств проекта и тут же видеть, как эти изменения вступают в силу. DevTools следит за изменениями в файлах и, когда видит, что что-то изменилось, автоматически перезапускает приложение.

Точнее, когда набор инструментов DevTools активен, приложение загружается в два отдельных загрузчика классов в виртуальной машине Java (JVM). Один загружает ваш код на Java, файлы свойств и почти все, что находится в пути *src/main/* проекта. Эти элементы могут часто меняться. Другой загрузчик загружает библиотеки зависимостей, которые вряд ли будут меняться так часто.

Обнаружив изменения, DevTools перезапускает только загрузчик классов, загружающий код вашего проекта, и повторно инициализирует контекст приложения Spring, но оставляет другой загрузчик классов и JVM нетронутыми. Эта стратегия позволяет немного сократить время, необходимое для запуска приложения.

Недостаток данной стратегии – изменения в зависимостях не будут вызывать автоматический перезапуск. Это связано с тем, что загрузчик классов, управляющий библиотеками зависимостей, не перезагружается автоматически. Каждый раз, добавляя, изменяя или удаляя зависимость в своей спецификации сборки, вы должны явно выполнить перезапуск приложения, чтобы эти изменения вступили в силу.

Автоматическое обновление браузера и отключение кеширования шаблонов

По умолчанию результаты анализа шаблонов, таких как Thymeleaf и FreeMarker, кешируются, поэтому шаблоны не нужно повторно анализировать при обслуживании каждого запроса. Это отлично подходит для промышленных условий, потому что дает небольшое преимущество в производительности.

Кеширование, однако, плохо подходит для этапа разработки, так как не позволяет оперативно видеть результаты внесения изменений в шаблоны после обновления окна браузера. Даже если вы внесли изменения, приложение продолжит использовать кешированный шаблон, пока вы не перезапустите его.

DevTools решает эту проблему, автоматически отключая кеширование всех шаблонов. Вносите в свои шаблоны столько изменений, сколько хотите, и знайте, что для просмотра изменений достаточно лишь обновить браузер.

Но если вы так же ленивы, как я, то наверняка не захотите утруждать себя даже такой малостью, как щелчок на кнопке обновления браузера. Было бы намного удобнее, если бы можно было вносить изменения и тут же наблюдать результаты в браузере. К счастью, в DevTools есть что предложить тем из нас, кому лень нажимать кнопку обновления.

DevTools автоматически запускает сервер LiveReload (http://livereload.com/) вместе с приложением. Сам по себе сервер LiveReload не особенно полезен. Но в сочетании с соответствующим плагином LiveReload ваш браузер автоматически будет обновлять страницу при внесении изменений в шаблоны, изображения, таблицы стилей, сценарии JavaScript и т. д. – почти все, что в конечном итоге передается браузеру.

Плагины LiveReload имеются для браузеров Google Chrome, Safari и Firefox. (Извините, любители Internet Explorer и Edge.) Посетите страницу http://livereload.com/extensions/, чтобы найти информацию о том, как установить плагин LiveReload в свой браузер.

Встроенная консоль H2

Наш проект пока не использует базу данных, но это случится в главе 3. Если вы решите использовать базу данных H2 для разработки, то DevTools также автоматически запустит консоль H2, к которой можно получить доступ из веб-браузера. Достаточно ввести в браузере адрес `http://localhost:8080/h2-console`, и вы увидите данные, с которыми работает ваше приложение.

На данный момент мы написали законченное, хотя и очень простое приложение Spring. Мы будем расширять и дополнять его на протяжении всей книги. А сейчас самое время отступить на шаг назад и проанализировать, чего мы достигли и какую роль во всем этом сыграл фреймворк Spring.

1.3.6 Обзор результатов

Давайте перечислим, что мы уже сделали. Проще говоря, перечислим шаги, выполненные для создания приложения Taco Cloud:

- создали начальную структуру проекта с помощью Spring Initializr;
- написали класс контроллера для обработки запроса к домашней странице;
- определили шаблон представления для отображения домашней страницы;
- написали простой тестовый класс, проверяющий работу контроллера домашней страницы.

Как будто ничего сложного, верно? За исключением первого шага – создания проекта, – все последующие действия были нацелены на создание домашней страницы.

Фактически почти каждая строка написанного нами кода нацелена на это. Если не считать операторов импорта, то получается, что мы написали только две строки кода в классе контроллера и ни одной строки в шаблоне представления. И хотя большая часть тестового класса использует поддержку тестирования Spring, в контексте теста это почти незаметно.

Это важное преимущество разработки с использованием Spring. Вы можете сосредоточиться на прикладной логике, а не на удовлетворении требований фреймворка. Конечно, иногда приходится писать код, специфичный для фреймворка, но обычно этот код составляет лишь малую часть кодовой базы. Как я уже говорил, Spring (и Spring Boot) можно считать *фреймворком без фреймворка*.

Как такое вообще возможно? Что делает Spring за кулисами, чтобы гарантировать удовлетворение потребностей вашего приложения?

Давайте начнем расследование с того, что делает Spring, с обзора спецификации сборки.

В файле *pom.xml* мы объявили зависимости `spring-boot-starter-web` и `spring-boot-starter-thymeleaf`. Они транзитивно подключают ряд других зависимостей, в том числе:

- фреймворк Spring MVC;
- встроенный сервер Tomcat;
- механизм шаблонов Thymeleaf и диалект языка разметки Thymeleaf.

Также мы автоматически получили библиотеку автоконфигурации Spring Boot. Когда приложение запускается, механизм автоконфигурации Spring Boot обнаруживает указанные нами зависимости и автоматически выполняет следующие шаги:

- настраивает bean-компоненты в контексте приложения Spring для включения Spring MVC;
- настраивает встроенный сервер Tomcat в контексте приложения Spring;
- настраивает механизм шаблонов Thymeleaf для отображения представлений Spring MVC.

Проще говоря, автоконфигурация выполняет всю рутинную работу, позволяя нам сосредоточиться на прикладном коде, реализующем функциональность приложения. Довольно удобное соглашение, как по мне!

Ваше путешествие по Spring только началось. В приложении Taco Cloud мы затронули лишь малую часть того, что может предложить Spring. Прежде чем сделать следующий шаг, давайте оглядимся вокруг и посмотрим, какие достопримечательности в ландшафте Spring ожидают нас в нашем путешествии.

1.4 Обзор ландшафта Spring

Чтобы получить представление о ландшафте Spring, обратите внимание на огромный список флажков в полной версии веб-формы Spring Initializr. В нем более 100 видов зависимостей, поэтому я даже не буду пытаться перечислить их здесь или предоставить скриншот – посмотрите сами, – но отмечу некоторые основные моменты.

1.4.1 Ядро Spring Framework

Как нетрудно догадаться, ядро Spring Framework является основой для всего остального во вселенной Spring. Это базовый контейнер и инфраструктура внедрения зависимостей. Но оно также предоставляет несколько других важных механизмов.

Среди них Spring MVC – веб-фреймворк Spring. Вы уже пробовали использовать Spring MVC для создания класса контроллера, обрабаты-

вающего веб-запросы. Но вы еще не видели, что Spring MVC можно использовать для создания REST API, генерирующего данные, отличные от HTML. Более детально мы изучим Spring MVC в главе 2, а затем еще раз посмотрим, как использовать его для создания REST API, в главе 7.

Ядро Spring Framework также предлагает некоторую элементарную поддержку хранения данных, в частности поддержку JDBC на основе шаблонов. Так, например, в главе 3 мы познакомимся с `JdbcTemplate`.

Spring включает поддержку реактивного программирования, включая новую реактивную веб-инфраструктуру Spring WebFlux, которая в значительной степени основана на Spring MVC. С моделью реактивного программирования в Spring мы познакомимся в части III книги и с Spring WebFlux в главе 12.

1.4.2 Spring Boot

Мы уже видели многие преимущества Spring Boot, включая подключение начальных зависимостей и автоконфигурацию. В этой книге мы используем Spring Boot в максимально возможной степени, чтобы избежать использования любых форм явной настройки, за исключением ситуаций, когда это абсолютно необходимо. В дополнение к начальным зависимостям и автонастройке Spring Boot предлагает также другие полезные возможности:

- Actuator позволяет исследовать работу внутренних механизмов приложения во время выполнения, включая получение метрик, дампов потоков выполнения и приложения и свойства окружения приложения;
- гибкую спецификацию свойств окружения;
- дополнительную поддержку тестирования, помимо поддержки, предоставляемой ядром фреймворка.

Кроме того, Spring Boot предлагает альтернативную модель программирования, основанную на сценариях Groovy, которая называется Spring Boot CLI (Command Line Unterface – интерфейс командной строки). С помощью Spring Boot CLI можно писать целые приложения в виде набора сценариев Groovy и запускать их из командной строки. Мы не будем тратить много времени на Spring Boot CLI, но будем обращаться к нему при случае, когда это будет соответствовать нашим потребностям.

Spring Boot стал настолько неотъемлемой частью разработки с использованием Spring, что я не могу представить разработку Spring-приложений без него. Поэтому данная книга в равной степени ориентирована и на Spring Boot, и вы часто будете замечать, что я использую слово Spring, когда имею в виду то, что делает Spring Boot.

1.4.3 Spring Data

Ядро Spring Framework уже включает базовую поддержку хранения данных, но Spring Data – это нечто совершенно удивительное! Этот

компонент экосистемы Spring позволяет определять репозитории данных для приложений в форме простых интерфейсов Java, используя соглашение об именах при определении методов, реализующих хранение и извлечение данных.

Более того, Spring Data может работать с несколькими различными типами баз данных, включая реляционные (через JDBC или JPA), документные (Mongo), графовые (Neo4j) и др. Мы будем использовать Spring Data для создания репозиториев для приложения Taco Cloud в главе 3.

1.4.4 Spring Security

Безопасность приложений всегда была важной темой, и кажется, что с каждым днем ее важность только возрастает. К счастью, в Spring есть надежный фреймворк Spring Security.

Spring Security удовлетворяет широкий спектр потребностей в обеспечении безопасности приложений, включая аутентификацию, авторизацию и защиту API. Спектр возможностей Spring Security слишком велик, чтобы его можно было должным образом охватить в этой книге, но все же мы коснемся некоторых наиболее распространенных вариантов его применения в главах 5 и 12.

1.4.5 Spring Integration и Spring Batch

Многим приложениям в какой-то момент требуется интегрироваться с другими приложениями или даже с другими компонентами того же приложения. Для удовлетворения этих потребностей появилось несколько моделей интеграции приложений. Spring Integration и Spring Batch обеспечивают реализацию этих моделей для приложений Spring.

Spring Integration обеспечивает интеграцию в реальном времени, когда данные обрабатываются по мере их поступления. Spring Batch, напротив, предназначен для пакетной интеграции, когда данные могут собираться в течение некоторого времени, пока не произойдет какое-то событие (например, сработает таймер), сообщающее о том, что пришла пора для обработки пакета данных. С фреймворком Spring Integration мы познакомимся в главе 10.

1.4.6 Spring Cloud

Мир разработки приложений вступает в новую эру, когда мы перестанем разрабатывать наши приложения как единые монолиты и будем составлять их из множества отдельных единиц развертывания, известных как *микросервисы*.

Микросервисы – актуальное направление, решающее множество практических проблем разработки и выполнения. Но при этом они привносят свои собственные проблемы. Эти проблемы решает Spring

Cloud – набор проектов для разработки облачных приложений с помощью Spring.

Spring Cloud охватывает множество направлений, и было бы невозможно осветить их все в этой книге. Если вас заинтересует данная тема, то я советую прочитать книгу «Cloud Native Spring in Action» Томаса Витале (Thomas Vitale; Manning, 2020, www.manning.com/books/cloud-native-spring-in-action).

1.4.7 Spring Native

Проект Spring Native является относительно новой экспериментальной разработкой в экосистеме Spring. Он позволяет компилировать проекты Spring Boot в двоичные выполняемые файлы с помощью компилятора GraalVM. Такие файлы запускаются значительно быстрее и занимают меньше места.

За дополнительной информацией о Spring Native обращайтесь по адресу https://github.com/spring-projects-experimental/spring-native.

Итоги

- Spring стремится упростить решение таких задач, как создание веб-приложений, работа с базами данных, защита приложений и микросервисы.
- Spring Boot основан на Spring и делает работу с фреймворком Spring еще проще, предлагая автоматическое управление зависимостями и конфигурацией и возможность анализа параметров выполнения приложений.
- Приложения Spring можно инициализировать с помощью веб-приложения Spring Initializr, доступного в интернете, которое поддерживается большинством сред разработки на Java.
- Компоненты, обычно называемые bean-компонентами, в контексте приложения Spring могут объявляться явно с помощью Java или XML, обнаруживаться путем сканирования компонентов или автоматически настраиваться с помощью механизма автоконфигурации в Spring Boot.