Оглавление

[1.3.5 Getting to know Spring Boot DevTools 3](#_Toc100951014)

[1.3.6 Let’s revlew 5](#_Toc100951015)

[1.4 Surveying the Spring landscape 6](#_Toc100951016)

[1.4.1 The core Spring Framework 6](#_Toc100951017)

[1.4.2 Spring Boot 7](#_Toc100951018)

[1.4.3 Spring Data 8](#_Toc100951019)

[1.4.4 Spring Security 8](#_Toc100951020)

[1.4.5 Spring Integration and Spring Batch 8](#_Toc100951021)

[1.4.6 Spring Cloud 9](#_Toc100951022)

[Summary 9](#_Toc100951023)

[2 Developing web applications 9](#_Toc100951024)

[This chapter covers 10](#_Toc100951025)

[2.1 Displaying information 10](#_Toc100951026)

[2.1.1 Establishing the domain 10](#_Toc100951027)

[2.1.2 Creating a controller class 10](#_Toc100951028)

[2.1.3 Designing the view 10](#_Toc100951029)

[2.2 Processing from submission 10](#_Toc100951030)

[2.3 Validating form imput 10](#_Toc100951031)

1.3.5 Getting to know Spring Boot DevTools

1.3.5 Знакомство с Spring Boot DevTools

Как следует из названия, DevTools предоставляет разработчикам Spring некоторые удобные средства разработки. Среди них:

- Автоматический перезапуск приложения при изменении кода

- Автоматическое обновление браузера, когда ресурсы, предназначенные для браузера (такие как шаблоны, JavaScript, таблицы стилей и т. д.) изменяются

- Автоматическое отключение кэшей шаблонов

- Встроенная H2 Console, если используется база данных H2

Важно понимать, что DevTools не является подключаемым модулем (плагин) IDE и не требует, что бы вы использовали конкретную IDE. Он одинаково хорошо работает в Spring Tool Suite, IntelliJ IDEA, и NetBeans. Кроме того, поскольку он предназначен только для целей разработки, он достаточно умен, чтобы отключить себя при развертывании в производственных условиях. (мы обсудим как это происходит, когда вы приступите к развертыванию своего приложения в главе 19.) А пока давайте сосредоточимся на наиболее полезных функциях Spring Boot DevTools, начиная с автоматического перезапуска приложения.

Automatic application restart

Автоматический перезапуск приложения

Используя DevTools как часть вашего проекта, вы сможете вносить изменения в код Java и properties файлы в проекте и увидеть, как эти изменения применяются спустя короткое время.

DevTools отслеживает изменения и, когда видит, что что-то изменилось, автоматически перезапускает приложение.

Точнее, когда DevTools запущен, приложение загружено в два отдельных загрузчика классов в виртуальной машине Java (JVM). Один загрузчик классов загружается с ваш код Java, файлы свойств и почти все, что находится в пути src/main/ путь проекта. Это элементы, которые могут часто меняться. Другой загрузчик классов загружен с библиотеками зависимостей, которые вряд ли будут меняться так часто.

При обнаружении изменения DevTools перезагружает только загрузчик классов, содержащий ваш код проекта и Spring application context, но оставляет другой класс загрузчик и JVM не тронутым. Несмотря на тонкость, эта стратегия позволяет немного сократить время, необходимое для запуска приложения.

Недостатком этой стратегии является то, что изменения зависимостей не будут доступны при автоматическом перезапуске. Это потому, что загрузчик классов, содержащий библиотеки зависимостей не перезагружается автоматически. Это означает, что каждый раз, когда вы добавляете, изменяете или удаляете зависимость в вашей спецификации сборки, вам нужно будет выполнить жесткий перезапуск приложения чтобы эти изменения вступили в силу.

\_\_\_\_\_

Как происходит загрузка классов в JVM

<https://javarush.ru/groups/posts/646-kak-proiskhodit-zagruzka-klassov-v-jvm>

\_\_\_\_\_

Automatic browser refresh and template cache disable

Автоматическое обновление браузера и отключение кэша шаблона

По умолчанию параметры шаблона, такие как Thymeleaf и FreeMarker, настроены кешировать результаты парсинга шаблонов, чтобы шаблоны не нужно было перепарсить с каждым запросом, который они обслуживают. Это здорово в продакшене, так как дает немного производительности.

Кэшированные шаблоны, однако, не так хороши во время разработки. Кэшированные шаблоны делают невозможным внесение изменений в шаблоны во время работы приложения и увидеть результаты после обновления браузера. Даже если вы внесли изменения, кэшированный шаблон будет по-прежнему использоваться, пока вы не перезапустите приложение.

DevTools решает эту проблему, автоматически отключая кэширование всех шаблонов. Делайте столько изменений, сколько хотите в свои шаблоны, и знайте, что для просмотра результата изменений необходимо всего лишь обновить браузер.

Но если вы похожи на меня, вы даже не хотите обременять себя усилием нажатия кнопки обновления браузера. Было бы намного лучше, если бы вы могли внести изменения и немедленно наблюдать за результатами изменения в браузере. К счастью, в DevTools есть кое-что специально для тех из нас, кто слишком ленив, чтобы нажать кнопку обновления.

Когда DevTools находится в игре, он автоматически включает LiveReload (<http://livereload.com/>) сервер вместе с вашим приложением. Сам по себе сервер LiveReload не очень полезный. Но в сочетании с соответствующим плагином для браузера, LiveReload заставляет ваш браузер автоматически обновляться при внесении изменений в шаблоны, изображения, таблицы стилей, JavaScript и т. д. — практически все, что в итоге подается в ваш браузер.

LiveReload имеет плагины для браузеров Google Chrome, Safari и Firefox. (Извините, любители Internet Explorer и Edge.) Посетите <http://livereload.com/extensions/>, чтобы найти информацию о том, как установить LiveReload для вашего браузера.

Built in H2 Console

Встроен в H2 Console

Хотя в вашем проекте еще не используется база данных, это изменится в главе 3. Если вы выберете использование базы данных H2 для разработки, DevTools автоматически включит консоль H2, к которой вы можете получить доступ из своего веб-браузера. Вам нужно только укажите в веб-браузере <http://localhost:8080/h2-console> , чтобы получить представление о данных, с которыми работает ваше приложение.

На данный момент вы написали полное, хотя и простое приложение Spring. Вы будете расширять его на протяжении всей книги. Но сейчас самое время отступить и просмотреть, чего вы достигли и какую роль сыграл Spring.

1.3.6 Let’s revlew

1.3.6 Давайте рассмотрим

Вспомните, как вы пришли к этому моменту. Короче говоря, это шаги, которые вы предприняли, чтобы создать свое приложение Taco Cloud на базе Spring:

- Вы создали исходную структуру проекта с помощью Spring Initializr.

- Вы написали класс контроллера для обработки запроса домашней страницы.

- Вы определили шаблон представления для отображения домашней страницы.

- Вы написали простой тестовый класс, чтобы проверить свою работу.

Кажется довольно простым, не так ли? За исключением первого шага начальной загрузки проекта, каждое действие, которое вы предприняли, было сосредоточено на достижении цели создания домашней страницы.

Фактически, почти каждая строчка написанного вами кода направлена ​​на достижение этой цели. Не считая операторов импорта Java, я считаю только две строки кода в вашем классе контроллере и нет строк в шаблоне представления, специфичных для Spring. И хотя основная масса тестового класса использует поддержку тестирования Spring, это кажется немного менее инвазивным в контексте теста.

Это важное преимущество разработки с использованием Spring. Вы можете сосредоточиться на коде, который удовлетворяет требованиям приложения, а не требования фреймворка. Хотя вам, несомненно, время от времени, потребуется написать некоторый специфический для фреймворка код, обычно это будет лишь небольшая часть вашей кодовой базы.

Как я уже говорил, Spring (со Spring Boot) можно считать бескаркасным фреймворком (frameworkless framework).

Как это вообще работает? Что Spring делает за кулисами, чтобы убедиться, что потребности вашего приложения удовлетворены? Чтобы понять, что делает Spring, давайте начнем с обзора спецификации сборки.

В файле pom.xml вы объявили зависимость от стартеров Web и Thymeleaf.

Эти две зависимости транзитивно привели к нескольким другим зависимостям, в том числе

- Фреймворк Spring MVC

- Встроенный Tomcat

- Thymeleaf и диалект разметки Thymeleaf

Он также принес с собой библиотеку автоконфигурации Spring Boot. Когда приложение запускается, автоконфигурация Spring Boot обнаруживает эти библиотеки и автоматически:

- Настраивает bean-компоненты в Spring application context для включения в Spring MVC.

- Настраивает встроенный сервер Tomcat в Spring application context.

- Настраивает преобразователь представлений Thymeleaf для рендеринга представлений Spring MVC с шаблонами Thymeleaf.

Короче говоря, автоконфигурация выполняет всю рутинную работу, позволяя вам сосредоточиться на написании кода, который реализует функциональность вашего приложения. Это довольно милая аранжировка, если вы меня спросите!

Ваше Spring путешествие только началось. Приложение Taco Cloud затронуло только малую часть того, что может предложить Spring. Прежде чем сделать следующий шаг, давайте осмотрим Spring пейзаж и узнаем, какие достопримечательности вы встретите в своем путешествии.

1.4 Surveying the Spring landscape

1.4 Осмотр Spring пейзажа

Чтобы получить представление о Spring пейзаже, посмотрите огромный список флажков (чекбоксов) в полной версии веб-формы Spring Initializr. В ней перечислено более 100 варианты зависимостей, поэтому я не буду пытаться перечислить их все здесь или предоставлять скриншот.

Но я призываю вас взглянуть. А пока упомяну несколько основные моментов.

1.4.1 The core Spring Framework

Как и следовало ожидать, ядро ​​Spring Framework является основой всего остального во вселенной Spring. Он предоставляет основной контейнер и фреймворк внедрения зависимостей. Но также он предоставляет несколько других важных функций.

Среди них Spring MVC, веб-фреймворк Spring. Вы уже видели, как используйте Spring MVC для написания класса контроллера для обработки веб-запросов. Что вы еще не видели, так это то, что Spring MVC также можно использовать для создания REST API, которые производят вывод не в формате HTML. Мы собираемся более подробно изучить Spring MVC в главе 2, а затем вы взгляните еще раз на то, как использовать его для создания REST API, в главе 6.

Ядро Spring Framework также предлагает некоторую элементарную поддержку постоянных данных (persistence data), в частности, поддержка JDBC на основе шаблонов. Вы увидите, как использовать JdbcTemplate в главе 3.

В самой последней версии Spring (5.0.8) была добавлена ​​поддержка реактивного стиля программирования, включая новый реактивный веб-фреймворк под названием Spring WebFlux, который сильно заимствован из Spring MVC. Вы посмотрите на реактивное программирование Spring модели в части 3 и Spring WebFlux, в частности, в главе 10.

1.4.2 Spring Boot

Мы уже видели многие преимущества Spring Boot, включая cnfhnjdst зависимости и автоконфигурацию. Будьте уверены, что мы будем использовать как можно больше Spring Boot в этой книге и избегайте любой формы явной конфигурации, если это не абсолютно необходимо. Но помимо начальных зависимостей и автоконфигурации, Spring Boot также предлагает несколько других полезных функций:

- Actuator обеспечивает понимание внутренней работы приложения во время выполнения, включая метрики, информацию о дампе потока, работоспособность приложения и среду свойства, доступные приложению.

\_\_\_\_\_

Введение в Spring Boot Actuator

<https://habr.com/ru/company/otus/blog/452624/>

\_\_\_\_\_

- Гибкая спецификация свойств среды.

- Дополнительная поддержка тестирования в дополнение к поддержке тестирования в Spring core.

Более того, Spring Boot предлагает альтернативную модель программирования, основанную на Groovy-скриптах, которые называются Spring Boot CLI (command-line interface /интерфейс командной строки). С Spring Boot CLI, вы можете писать целые приложения в виде набора скриптов Groovy и запускать их из командной строки. Мы не будем тратить много времени на Spring Boot CLI, но мы коснемся его при случае, когда он будет соответствовать нашим потребностям.

Spring Boot стал такой неотъемлемой частью разработки Spring, что я не могу представить разработку приложения Spring без него. Следовательно, эта книга уделяет Spring Boot основное внимание, и вы можете заметить, что я использую слово Spring, когда я имею в виду о, что делает Spring Boot.

1.4.3 Spring Data

Хотя ядро ​​Spring Framework поставляется с базовой поддержкой data persistence, Spring Data предоставляет кое-что удивительное: возможность определить репозитории данных как простые интерфейсы Java, использующие соглашение об именах (naming convention) при определении методов управления хранением и извлечением данных.

Более того, Spring Data может работать с несколькими различными типами баз данных, в том числе реляционными (JPA), документоориентированными (Mongo), графовыми (Neo4j) и другими.

Вы будете 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, похоже, она становится все более актуальной с каждым днем. К счастью, Spring имеет надежную структуру безопасности в Spring Security.

Spring Security удовлетворяет широкому спектру потребностей в безопасности приложений, в том числе аутентификация, авторизация и безопасность API. Хотя объем Spring Security слишком велик, чтобы быть, должным образом, описанным в этой книге, мы коснемся некоторых наиболее распространенных случаев использования в главах 4 и 12.

1.4.5 Spring Integration and Spring Batch

В какой-то момент большинству приложений потребуется интегрироваться с другими приложениями или даже с другими компонентами того же приложения. Некоторое количество паттернов интеграции приложений возникла для удовлетворения этих потребностей. Spring Integration и Spring Batch обеспечивает реализацию этих шаблонов для приложений на базе Spring.

Spring Integration обеспечивает интеграцию в реальном времени, когда данные обрабатываются по мере их поступления и делаются доступными.

Spring Batch, напротив, предназначен для пакетной интеграции, когда данные разрешено собирать в течение некоторого времени, пока какой-либо триггер (возможно, триггер времени) не просигнализирует, что наступило время обработки пакетных данных. Вы изучите как Spring Batch, так и Spring Integration в главе 9.

1.4.6 Spring Cloud

Пока я пишу это, мир разработки приложений вступает в новую эру, когда мы больше не будем разрабатывать наши приложения как монолиты из одной единицы развертывания, а будем вместо этого создавать приложения из нескольких отдельных единиц развертывания, известных как микросервисы.

Микросервисы — горячая тема, затрагивающая несколько практических аспектов разработки и проблемы во время выполнения. Однако при этом они выдвигают на первый план свои собственные проблемы. Эти проблемы решаются с помощью Spring Cloud, набора проектов для разработки облачных приложении со Spring.

Spring Cloud охватывает много вопросов, и было бы невозможно охватить это все в данной книге. Мы рассмотрим некоторые из наиболее распространенных компонентов Spring Cloud в главах 13, 14 и 15. Для более полного обсуждения Spring Cloud я предлагаю посмотреть Spring Microservices in Action Джона Карнелла (Manning, 2017, [www.manning.com/books/spring-microservices-in-action](http://www.manning.com/books/spring-microservices-in-action) ).

Summary

- Spring стремится упростить задачи разработчиков, такие как создание веб-приложений, работа с базами данных, защита приложений и микросервисов.

- Spring Boot построен на основе Spring, чтобы сделать Spring еще проще за счет упрощения управления зависимостями, автоматической конфигурации (настройки) и анализа во время выполнения.

- Приложения Spring можно инициализировать с помощью веб-интерфейса Spring Initializr, который изначально поддерживается в большинстве сред разработки Java.

- Компоненты, обычно называемые bean-компонентами, в Spring application context могут быть объявлены явно с помощью Java или XML, обнаружен компонентом сканирования или автоматически настроены с помощью автоконфигурации Spring Boot.

2 Developing web applications

This chapter covers

- Представление данных модели в браузере

- Обработка и проверка ввода формы

- Выбор библиотеки шаблонов представлений

Первые впечатления очень важны. Наружное обустройство дома может продать дом задолго до того, как покупатель войдет в дверь. Вишневая окраска автомобиля привлечет больше внимания, чем то, что находится под капотом. А литература изобилует историями о любви с первого взгляда. То, что внутри, очень важно, но то, что снаружи — то, что видно первым — важно.

Приложения, которые вы создадите с помощью Spring, будут делать самые разные вещи, в том числе обработка данных, чтение информации из базы данных и взаимодействие с другими приложениями. Но первое впечатление, которое получат пользователи от вашего приложения, будет пользовательский интерфейс. И во многих приложениях этот пользовательский интерфейс представляет собой то, что веб-приложение показывает в браузере.

В главе 1 вы создали свой первый контроллер Spring MVC для отображения домашней страницы вашего приложения. Но Spring MVC может делать гораздо больше, чем просто отображать статический контент. В этой главе вы разработаете первую крупную функциональность в своем приложении Taco Cloud - возможность создавать собственные тако. При этом вы углубитесь в Spring MVC, и вы увидите, как отображать данные модели и обрабатывать ввод формы.

2.1 Displaying information

По сути, Taco Cloud — это место, где вы можете заказать тако онлайн. Но более того, Taco Cloud хочет дать своим клиентам возможность выразить свою творческую сторону с помощью создания нестандартных тако из богатой палитры ингредиентов.

Поэтому веб-приложению Taco Cloud нужна страница, отображающая выбор ингредиентов для художников тако. Выбор ингредиентов может измениться в любое время, поэтому они не должны быть жестко запрограммированы на HTML-странице. Вернее, список доступных ингредиенты должен быть извлечен из базы данных и передан на страницу для отображается заказчику.

В веб-приложении Spring задачей контроллера является получение и обработка данных. И задача представления — преобразовать эти данные в HTML, который будет отображаться в браузере.

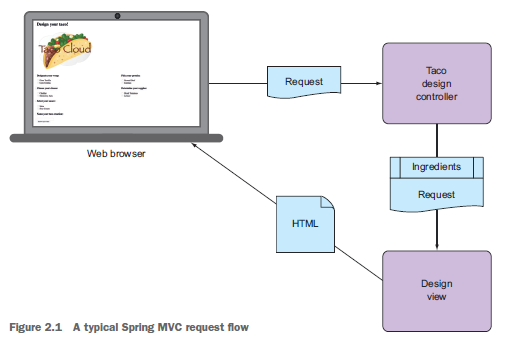
Вы собираетесь создать следующие компоненты для поддержки страницы создания тако:

- Класс предметной области, определяющий свойства ингредиента тако.

- Класс контроллера Spring MVC, который извлекает информацию об ингредиентах и ​​передает ее вместе с видом

- Шаблон представления, отображающий список ингредиентов в браузере пользователя.

Отношения между этими компонентами показаны на рисунке 2.1.



Поскольку в этой главе основное внимание уделяется веб-инфраструктуре Spring, мы отложим все аспекты базы данных до главы 3. На данный момент контролер будет отвечать только за предоставление ингредиентов для обзора. В главе 3 вы переработаете контроллер для совместной работы с репозиторием, который извлекает данные об ингредиентах из базы данных.

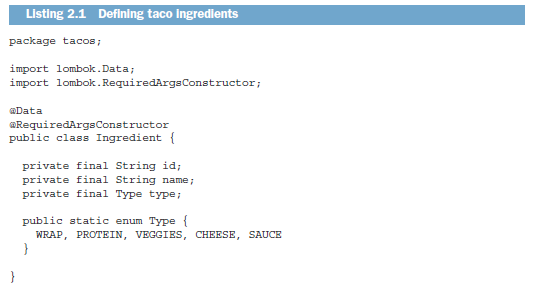
Прежде чем вы напишете контроллер и представление, давайте выработаем тип области (domain type) – java class, который представляет собой ингредиент. Это заложит основу, на которой вы сможете разрабатывать ваши веб-компоненты.

2.1.1 Establishing the domain

Область приложения (application’s domain) — это предметная область, к которой оно обращается, — идеи и концепции, которые влияют на понимание приложения /\* For a much more in-depth discussion of application domains, I suggest Eric Evans’ Domain-Driven Design (Addison-Wesley Professional, 2003) \*/. В приложении Taco Cloud домен включает в себя такие объекты, как дизайн тако, ингредиенты из которых состоит этот дизайн, клиентов и заказов тако размещенных клиентами. Для начала, мы сосредоточимся на ингредиентах тако.

В вашем домене ингредиенты для тако - довольно простые объекты. У каждого объекта есть имя, которое соответствует типу, чтобы его можно было визуально классифицировать (белки, сыры, соусы и т. д.).

У каждого типа также есть идентификатор (ID), по которому на него можно легко и однозначно ссылаться. Следующий класс *Ingredient* определяет нужный вам объект домена.



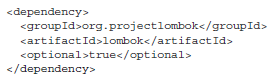
Как видите, это заурядный доменный класс Java, определяющий три свойства необходимых для описания ингредиента. Пожалуй, самое необычное в классе *Ingredient*, определенный в листинге 2.1, состоит в том, что в нем отсутствует обычный набор сеттер и геттер методов, не говоря уже о полезных методах, таких как equals(), hashCode(), toString() и другие.

Вы не видите их в списке отчасти для экономии места, но также и потому, что используете замечательную библиотеку под названием Lombok для автоматической генерации этих методов во время выполнения.

На самом деле аннотация @Data на уровне класса предоставлена ​​ Lombok и сообщает ему, что необходимо генерировать все, вышеперечисленные отсутствующие методы, а также конструктор, который принимает все окончательные свойства в качестве аргументов. Используя Lombok, вы можете сохранить код для класса *Ingredient* кратким и аккуратный.

Чтобы использовать Lombok, нужно добавить его в качестве зависимости в свой проект. Если вы используете Spring Tool Suite, достаточно щелкнуть правой кнопкой мыши файл pom.xml и выбрав Edit Starters в контекстном меню Spring. Та самая подборка зависимостей, которые вы дали в главе 1 (на рис. 1.4), появятся, давая вам возможность добавить или изменить выбранные зависимости. Найдите Lombok и выберите его, убедитесь, что он отмечен, и нажмите OK; Spring Tool Suite автоматически добавит его в вашу спецификацию сборки.

Кроме того, вы можете добавить его вручную с помощью следующей записи в pom.xml:



Эта зависимость предоставит вам аннотации Lombok (например, @Data) которые позволят вам сократить время разработки автоматической генерацией методов во время выполнения. Но вам также понадобится добавить Lombok в качестве расширения в вашу IDE, иначе она будет жаловаться на ошибки об отсутствующих методах и final свойствах, которые не устанавливаются. Посетите <https://projectlombok.org/> , чтобы узнать, как установить Lombok в выбранную вами IDE.

Я думаю, вы найдете Lombok очень полезным, но знайте, что это необязательная опция. Lombok не нужен для разработки приложений Spring. Поэтому, если вы не хотите его использовать и не ленитесь писать недостающие методы вручную, пишите... Я подожду. Когда вы закончите, вы добавите некоторые контроллеры для обработки веб-запросов в вашем приложении.

2.1.2 Creating a controller class

Контроллеры являются основными игроками в среде Spring MVC. Их основная задача состоит в том, чтобы обрабатывать HTTP-запросы либо передавать запрос в представление для отображения HTML (отображается в браузере) или записывать данные непосредственно в тело ответа (RESTful). В этой главе мы сосредоточимся на типах контроллеров, которые используют представления для создания контента для веб-браузеров. Когда мы дойдем до главы 6, мы рассмотрим написание контроллеров, которые обрабатывают запросы в REST API.

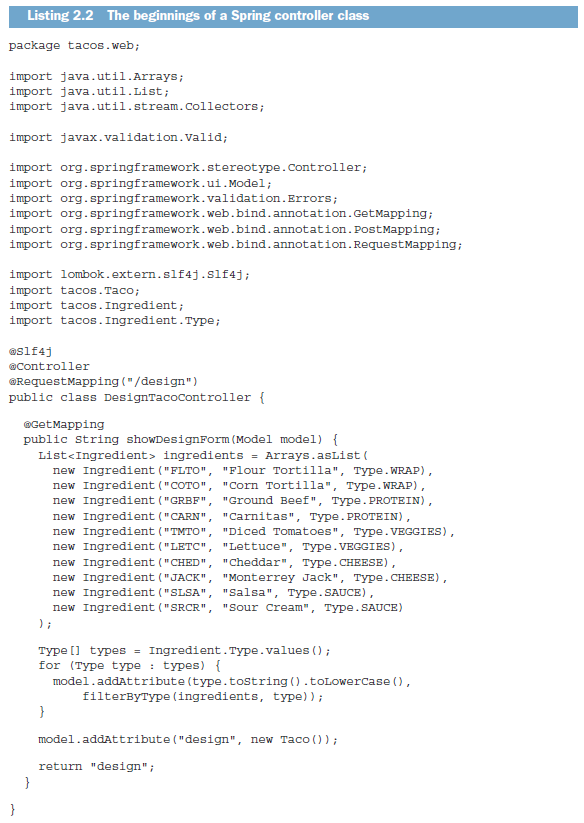
Для приложения Taco Cloud вам понадобится простой контроллер, который будет делать следующее:

- Обрабатывать HTTP-запросы GET, где путь запроса - /design

- Составьте список ингредиентов

- Передать запрос и данные об ингредиентах в шаблон представления для визуализации в формате HTML и отправлять запрашивающему веб-браузеру.

Следующий класс DesignTacoController отвечает этим требованиям.



Первое, что следует отметить в DesignTacoController, — это набор применяемых аннотаций на уровне класса. Первая, @Slf4j, представляет собой предоставленную Lombok аннотацию, которая во время выполнения автоматически сгенерирует SLF4J (простой Logging Facade для Java,

<https://www.slf4j.org/> ) Logger в классе. Эта скромная аннотация имеет такой же эффект, как если бы вы явно добавили следующие строки в класс:



Вы будете использовать этот регистратор чуть позже.

Следующая аннотация, применяемая к *DesignTacoController*, — *@Controller*. Этот аннотация служит для того, чтобы идентифицировать этот класс как контроллер и пометить его как кандидата на сканирование компонентов, так что Spring обнаружит его и автоматически создаст экземпляр *DesignTacoController*, как bean-компонент в Spring application context.

*DesignTacoController* также имеет аннотацию *@RequestMapping*. @Request-Mapping аннотация, применяемая на уровне класса, указывает тип запросов, которые этот контроллер обрабатывает. В этом случае он указывает, что *DesignTacoController* будет обрабатывать запросы, путь которых начинается с /design.

Handling a GET request

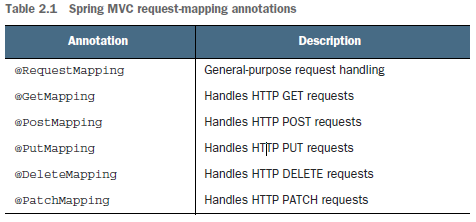
Спецификация *@RequestMapping* на уровне класса уточняется с помощью аннотации *@GetMapping* которая украшает метод showDesignForm(). *@GetMapping* в паре с аннотацией уровня класса *@RequestMapping* указывает, что при получении HTTP-запроса GET для /design, для обработки запроса будет вызван метод *showDesignForm()*.

*@GetMapping* - относительно новая аннотация, представленная в Spring 4.3. До Spring 4.3 вы могли использовать аннотацию *@RequestMapping* уровня метода:

@RequestMapping(method=RequestMethod.GET)

Ясно, что *@GetMapping* является более краткой и специфичной для метода HTTP, на который она нацелена.

*@GetMapping* – это всего лишь один член семейства request-mapping аннотаций. Таблица 2.1 перечисляет все request-mapping аннотации, доступные в Spring MVC.



Making the right thing the easy thing

Делаем правильные вещи простыми

Всегда полезно быть как можно более конкретным при объявлении request mappings на ваших методах контроллера. По крайней мере, это означает объявление пути (или наследуя путь от класса @RequestMapping) и какой HTTP метод это обработает.

Более длинный @RequestMapping(method=RequestMethod.GET) вызывал искушение выберите ленивый выход и оставьте атрибут метода. Благодаря новому Spring 4.3 сопоставление аннотаций, правильное решение — это также простое решение, требующее меньшего набора текста.

Новые аннотации сопоставления запросов имеют все те же атрибуты, что и @Request-Mapping, поэтому вы можете использовать их везде, где в противном случае вы использовали бы @RequestMapping.

Как правило, я предпочитаю использовать @RequestMapping только на уровне класса, чтобы указать базовый путь. Я использую более конкретные @GetMapping, @PostMapping и т. д. для каждого из методов обработчиков.

Теперь, когда вы знаете, что метод showDesignForm() будет обрабатывать запрос, давайте посмотрите на тело метода, чтобы увидеть, как он работает. Основная часть метода строит список ингредиентов объекта. Список пока жестко запрограммирован. Когда мы дойдем до главы 3, вы получать список доступных ингредиентов тако из базы данных.

Как только список ингредиентов будет готов, следующие несколько строк метода showDesignForm() фильтруют список по типам ингредиентов. Список типов ингредиентов затем добавляется в качестве атрибута в объект модели, который передается в showDesignForm(). Модель — это объект, который передает данные между контроллером и любым представлением, ответственным за рендеринг этих данных. В конечном счете, данные, помещенные в атрибуты модели, копируются в атрибуты ответа сервлета, где вид может найти их. Метод showDesignForm() завершается возвращанием строки "disign", которая является логическим именем представления, которое будет использоваться для рендеринга модели в браузере.

Ваш DesignTacoController действительно начинает обретать форму. Если вы запустите приложение сейчас и укажите в браузере путь /design, в классе-контроллере DesignTacoControllers будет задействован метод showDesignForm(), извлекающий данные из репозитория и помещающий их в модель перед передачей запроса в представление. Но поскольку вы еще не определили представление, запрос примет ужасный оборот, что приводит к ошибке HTTP 404 (Not Found). Чтобы исправить это, давайте переключим наше внимание к представлению, где данные будут оформлены с помощью HTML для представления их в веб-браузере пользователя.

2.1.3 Designing the view

2.2 Processing from submission

2.3 Validating form imput