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.

1.4.2 Spring Data