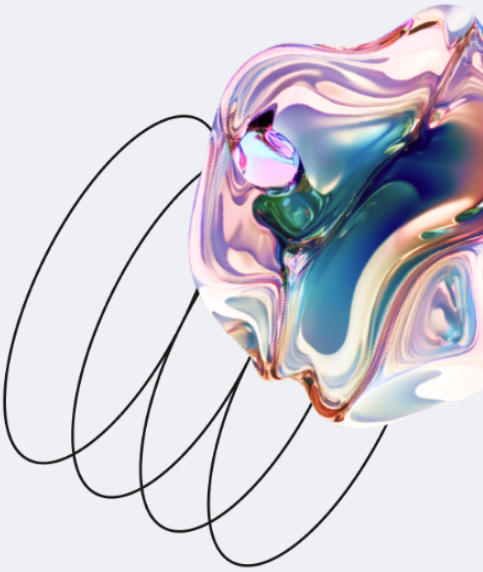
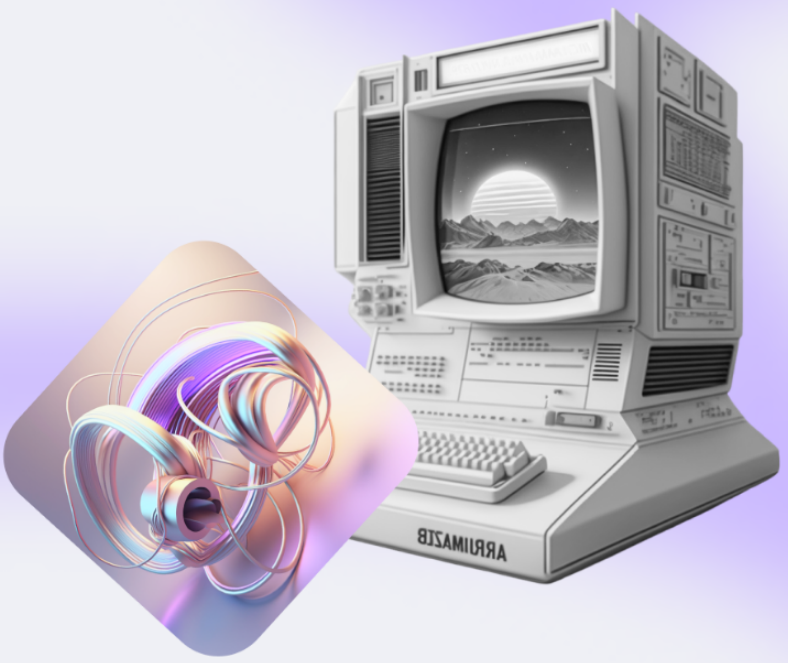
GeekBrains



Основы Spring. Spring Boot

Фреймворк Spring



Оглавление

Введение

Привет, студенты! Сегодня мы затронем одну из самых актуальных тем в мире Java-разработки — фреймворк Spring и его вариацию Spring Boot. Но прежде чем углубиться в детали, давайте разберёмся, как этот урок связан с нашей предыдущей темой: “Использование Maven и Gradle”.

Вспомните, мы говорили о том, что Maven и Gradle – это инструменты, которые помогают нам управлять зависимостями в наших проектах и автоматизировать процессы сборки. А что такое зависимости? Это, в большинстве случаев, библиотеки и фреймворки, которые мы используем в наших проектах. И вот здесь мы плавно переходим к теме нашего сегодняшнего урока.

Скорее всего, вы слышали слова “библиотека” и “фреймворк” множество раз, но чем они отличаются друг от друга? Представьте, что вы строите дом. Библиотека – это как набор инструментов: молоток, отвертка, пила. Вы используете их, как вам угодно, в любом порядке. Фреймворк же - это уже готовый проект дома со всеми

чертежами и планами. Вам необходимо следовать этому проекту, но при этом вы получаете готовый результат быстрее и с меньшими усилиями.

Вернемся к нашему миру разработки. Библиотеки – это наборы готовых функций и классов, которые мы можем использовать в своем коде, как нам угодно. Фреймворки – это более сложные инструменты, они предлагают определенную структуру или “каркас” для нашего приложения. Мы пишем свой код внутри этого каркаса, следуя определенным правилам и структуре, которые нам предлагает фреймворк. Примером библиотеки может быть JUnit, который мы используем для тестирования нашего кода, а примером фреймворка - именно Spring, о котором мы сегодня и поговорим.

Использование фреймворков важно по многим причинам. Во-первых, они значительно ускоряют процесс разработки, предоставляя нам уже готовые решения для многих типичных задач: работы с базой данных, обработки запросов на сервер, авторизации пользователей и многого другого. Во-вторых, они обеспечивают стандартизацию кода. Если все разработчики в команде используют один и тот же фреймворк, то их код будет более структурированным и понятным для всех членов команды.

Сейчас давайте поговорим о Spring. Spring – это один из самых популярных фреймворков в мире Java. Почему? Потому что он предлагает уникальную комбинацию гибкости и мощности. С помощью Spring вы можете создавать различные типы приложений: от простых веб-сервисов до сложных корпоративных систем.

Знание основ Spring очень важно для разработки приложений на этом фреймворке. Почему? Ну, представьте, что вы решили построить дом, но не знаете, как использовать молоток или отвертку. Как вы думаете, насколько успешным будет ваш проект? Точно так же и с Spring. Если вы не понимаете основ, вы не сможете полноценно использовать все возможности, которые предлагает этот фреймворк.

В этом курсе мы начнем с основ, чтобы вы поняли, как работает Spring “изнутри”. Мы поговорим о таких вещах, как контейнер IoC, принципы DI, AOP и многое другое. Затем мы перейдем к более продвинутым темам и покажем вам, как создать полноценное веб-приложение с помощью Spring Boot.

Spring Boot — это версия Spring, которая делает работу с этим фреймворком еще проще. Она предлагает “оптимальные настройки по умолчанию” для быстрого старта разработки, а также множество дополнительных функций, которые делают вашу жизнь проще.

Но обо всем этом мы поговорим позже. Сейчас же самое время заняться основами. В следующем разделе мы начнем разбираться, что такое Spring, как он работает и как начать с ним работать.

Термины, используемые в лекции

Spring – Мощный фреймворк, созданный для упрощения разработки Java-приложений

Java-приложения – Программы, написанные на языке программирования Java

DI (Dependency Injection) – Концепция, позволяющая делать приложения более гибкими и легкими для тестирования путем внедрения зависимостей извне, а не создания их внутри класса

IoC (Inversion of Control) – Концепция, в соответствии с которой не приложение

Макет (Mock) – Используется в программировании для имитации поведения реальных объектов в контексте тестирования

HTTP-запросы – Стандартное средство обмена данными в интернете между клиентом и сервером

Tomcat – Открытый веб-сервер и сервлет-контейнер, реализующий спецификации Java Servlet и JavaServer Pages (JSP)

Файл конфигурации Spring – Файл, который описывает, какие компоненты (или “бины”) должны быть созданы в Spring и как они связаны друг с другом

Бины (Beans) – Объекты, управляемые Spring и добавляемые в контейнер IoC

Java-конфигурация – Способ конфигурирования приложения на Spring с использованием Java, вместо XML

Базы данных – Структурированные наборы данных, с которыми можно работать с помощью Spring

@Component – Общая аннотация Spring, которую можно использовать для определения любого бина

@Service – Специализированная версия аннотации @Component, предназначенная для классов, которые представляют бизнес-логику приложения

@Repository – Специализированная версия аннотации @Component, предназначенная для классов, которые взаимодействуют с системой хранения данных

@Controller – Специализированная версия аннотации @Component, предназначенная для классов, которые обрабатывают HTTP-запросы в веб-приложениях Spring MVC или Spring WebFlux

@Configuration – Аннотация, используемая для классов, которые определяют бины с помощью методов @Bean

@Bean – Аннотация, используемая вместе с @Configuration для определения бинов

Spring Data Access – Механизм Spring, позволяющий взаимодействовать с системами хранения данных

Spring MVC – Подфреймворк Spring, предназначенный для разработки веб-приложений

Spring WebFlux – Подфреймворк Spring, предназначенный для разработки реактивных веб-приложений

Система хранения данных (Data Storage System) – Средства и механизмы для сохранения, поиска и извлечения данных

Бизнес-логика – Правила и процедуры, которые определяют, как бизнес должен функционировать

Исключения Spring DataAccessException – Исключения, связанные с доступом к данным в Spring

Основы Spring

Сейчас мы поговорим о том, что такое Spring и какие преимущества он предлагает нам, разработчикам.

Spring — это мощный фреймворк, созданный для упрощения разработки Java-приложений. Он базируется на нескольких ключевых принципах, и сегодня мы поговорим о двух из них: DI (Dependency Injection) и IoC (Inversion of Control).

DI, или внедрение зависимостей, - это концепция, которая позволяет нам делать наши приложения более гибкими и легкими для тестирования. Представьте, что у вас есть класс “Автомобиль”, который зависит от другого класса “Двигатель”. В классическом подходе вы бы просто создавали объект “Двигатель” внутри класса “Автомобиль”. Но что, если вы захотите заменить “Двигатель” на другой тип двигателя? Или что, если вы захотите тестировать класс “Автомобиль” отдельно от “Двигателя”?

Вот здесь и приходит на помощь DI. Вместо того, чтобы создавать объект “Двигатель” внутри класса “Автомобиль”, мы “внедряем” его извне. Это значит, что мы можем легко заменить “Двигатель” на другой тип или подменить его макетом для тестирования.

IoC, или инверсия управления, - это еще одна важная концепция в Spring. Она означает, что не наше приложение контролирует жизненный цикл его компонентов (как это обычно происходит), а наоборот - фреймворк контролирует наше приложение. Это как если бы вы ехали на автомобиле, но вместо того, чтобы самому управлять, вы передаете руль и педали другому человеку. Этот “другой человек” - это Spring. Он знает, когда создавать и уничтожать объекты, когда их связывать и т.д.

Если вы думаете, что это звучит сложно, представьте себе игру в Лего. Вы хотите построить замок. Вы можете сделать это двумя способами. Первый способ - это найти инструкцию и по ней собрать все детали вместе. Это будет вашим классическим подходом. Второй способ - это отдать все детали и инструкцию другому человеку и попросить его собрать замок за вас. Это будет вашим Spring подходом. Какой способ вы бы выбрали? Я думаю, большинство из нас выберут второй, потому что он проще и менее подвержен ошибкам.

Теперь, когда мы поняли, что такое DI и IoC, давайте поговорим о том, как работает Spring “изнутри”.

Когда вы запускаете приложение на Spring, первое, что происходит, - это запуск встроенного сервера, такого как Tomcat. Этот сервер слушает входящие HTTP-запросы и передает их вашему приложению. Но как именно это работает?

Все начинается с файла конфигурации Spring, который описывает, какие компоненты (или “бины”, как их называют в Spring) должны быть созданы и как они

связаны друг с другом. Когда сервер запускается, Spring читает этот файл и создает все необходимые бины. Затем он “внедряет” эти бины в нужные места, используя DI. После этого ваше приложение готово к работе.

Spring также предлагает множество других возможностей. Он поддерживает различные способы конфигурации, включая аннотации и Java-конфигурацию. Он предлагает мощную поддержку тестирования, что позволяет вам легко тестировать отдельные компоненты вашего приложения. Он также включает в себя множество других модулей для различных задач, таких как работа с базами данных, обработка веб-запросов и многое другое.

В Spring Framework существует множество аннотаций, которые используются для определения и конфигурации бинов. Вот несколько основных из них:

1. @Component: Это общая аннотация, которую можно использовать для определения любого бина. Классы, аннотированные как @Component, автоматически сканируются Spring и регистрируются в контейнере IoC.

2. @Service: Это специализированная версия @Component, предназначенная для классов, которые представляют бизнес-логику приложения. Она не добавляет дополнительной функциональности по сравнению с @Component, но помогает лучше структурировать код.

3. @Repository: Это еще одна специализированная версия @Component, предназначенная для классов, которые взаимодействуют с системой хранения данных. Она может интегрироваться с механизмом перехвата исключений Spring Data Access, который автоматически преобразует исключения хранилища данных в исключения Spring DataAccessException.

4. @Controller: Это специализированная версия @Component, предназначенная для классов, которые обрабатывают HTTP-запросы в веб-приложениях Spring MVC или Spring WebFlux.

5. @Configuration: Эта аннотация используется для классов, которые определяют бины с помощью методов @Bean. Эти классы играют роль источников определения бинов для контейнера IoC.

6. @Bean: Эта аннотация используется вместе с @Configuration для определения бинов. Методы, аннотированные как @Bean, создают объекты, которые управляются Spring и добавляются в контейнер IoC.

Вот пример использования этих аннотаций:

```
@Configuration
public class AppConfig {

    @Bean
    public MyService myService() {
        return new MyServiceImpl();
    }
}

@Service
public class MyServiceImpl implements MyService {
    // ...
}

@Repository
public class MyRepository {
    // ...
}

@Controller
public class MyController {
    // ...
}
```

В этом примере AppConfig — это конфигурационный класс, который определяет бин MyService. MyServiceImpl — это реализация MyService, аннотированная как @Service. MyRepository — это класс, аннотированный как @Repository, а MyController — это класс, аннотированный как @Controller.

Теперь давайте поговорим о разнице между Spring и Spring Boot. В основе Spring Boot лежит Spring, но он предлагает множество дополнительных возможностей. В частности, Spring Boot предоставляет “оптимальные настройки по умолчанию”, что позволяет вам быстро начать разработку без необходимости настраивать все с нуля. Он также включает в себя множество “стартеров”, которые автоматически включают в ваше приложение нужные зависимости.

Подводя итоги вышесказанного, хочу сказать, что Spring - это мощный инструмент в руках разработчика. Он предлагает множество возможностей и позволяет вам сосредоточитьсяна самой разработке, а не на решении технических проблем. Он

упрощает разработку, делает ваш код более чистым и понятным, а также помогает вам быстро и эффективно создавать высококачественные приложения.

Давайте погрузимся глубже в преимущества Spring, посмотрев на пример кода. Допустим, у нас есть класс Car, который зависит от класса Engine.

```
public class Car {
  private Engine engine;

  public Car() {
    this.engine = new Engine();
  }
}
```

В этом случае, мы напрямую создаём объект класса Engine внутри класса Car. Вместо этого, мы могли бы воспользоваться преимуществами Spring и внедрить зависимость через конструктор:

```
public class Car {
  private Engine engine;

  public Car(Engine engine) {
    this.engine = engine;
  }
}
```

Здесь, Spring будет автоматически создавать экземпляр Engine и передавать его в конструктор Car. Это упрощает наш код и делает его более гибким.

Теперь, что касается разницы между Spring и Spring Boot. Spring Boot, по сути, это Spring, но с дополнительными возможностями, предоставляемыми “из коробки”. Он автоматизирует конфигурацию, упрощает управление зависимостями и предлагает множество других удобных функций. Давайте сравним это с поездкой на автомобиле. Если Spring - это автомобиль, то Spring Boot — это автомобиль с встроенной GPS-системой, автоматической коробкой передач и удобными сиденьями с подогревом. Оба варианта доставят вас в пункт назначения, но Spring Boot сделает вашу поездку более комфортной.

Однако стоит помнить, что за всеми этими удобными функциями стоит дополнительный вес. Spring Boot может быть немного более тяжелым, чем обычный Spring, из-за всех дополнительных функций, которые он предлагает. Но на мой взгляд, преимущества, которые он предлагает, вполне оправдывают этот

дополнительный вес.

Типы бинов

В Spring Framework существует несколько типов бинов, основанных на их области видимости. Область видимости бина определяет, когда и как создается новый экземпляр бина. В Spring определены следующие области видимости:

1. Singleton: Это область видимости по умолчанию. Когда бин определен как Singleton, Spring IoC контейнер создает единственный экземпляр бина, и все запросы на получение этого бина возвращают один и тот же объект. Это подходит для бинов, которые не содержат состояния, таких как сервисы или DAO.

2. Prototype: Когда бин определен как Prototype, Spring IoC контейнер создает новый экземпляр бина каждый раз, когда он запрашивается. Это подходит для бинов, которые содержат состояния и не могут быть использованы одновременно в разных компонентах или потоках.

3. Request, Session, и Application: Эти области видимости применяются только в веб-приложениях. Бин области видимости Request создается для каждого HTTP-запроса. Бин области видимости Session создается для каждого HTTP-сеанса. Бин области видимости Application связан с жизненным циклом ServletContext и обычно используется для хранения данных на уровне приложения.

4. WebSocket: Эта область видимости доступна для бинов, которые должны быть связаны с жизненным циклом WebSocket.

Для определения области видимости бина в Spring, вы можете использовать аннотацию @Scope. Например:

```
@Component
@Scope("prototype")
public class PrototypeBean {
    // ...
}
```

В этом примере PrototypeBean будет создаваться каждый раз, когда он запрашивается из Spring контейнера.

Создание простого Spring Boot приложения

Давайте погрузимся в мир Spring Boot и создадим наше первое приложение! Мы начнем с использования инструмента под названием Spring Initializr, который является вашим волшебным ключом к быстрому старту в Spring Boot.

Spring Initializr — это удивительный инструмент, созданный командой Spring. Это прямо как торговый автомат с магическими напитками в видеоиграх. Ты просто выбираешь что тебе нужно, и волшебная машина создает это для тебя! В нашем случае, вместо магических напитков, Spring Initializr выдает проекты Spring Boot.

Вы можете получить доступ к Spring Initializr прямо из вашей IDE или, если вы любите веб-версии, вы можете перейти на сайт https://start.spring.io. Здесь вы можете выбрать параметры вашего проекта, такие как язык программирования (Java, Kotlin или Groovy), версию Spring Boot, а также зависимости, которые вы хотите включить в ваш проект.

Давайте создадим простое веб-приложение. Выберем Java как язык программирования и последнюю версию Spring Boot. В разделе “Dependencies” выберем “Spring Web”. Нажмите “Generate”, и Spring Initializr создаст для вас проект и скачает его в виде .zip-файла.

Распакуйте этот .zip-файл, и вы увидите структуру проекта Spring Boot. Давайте рассмотрим основные компоненты этой структуры:

1. src/main/java: Здесь находится весь ваш исходный код.
2. src/main/resources: Здесь находятся ресурсы вашего приложения, такие как файлы конфигурации, статические веб-ресурсы и т.д.
3. src/test/java: Здесь находятся ваши тесты.
4. pom.xml или build.gradle: Этот файл содержит информацию о вашем проекте и его зависимостях.

В каталоге src/main/java вы найдете файл с именем Application.java (или что-то в этом роде). Это основной класс вашего приложения. Он содержит метод main(), который запускает ваше приложение. Внутри этого метода вызывается метод SpringApplication.run(), который загружает приложение и запускает встроенный сервер.

Подожди, встроенный сервер? Да, именно так. Spring Boot автоматически запускает встроенный сервер (обычно Tomcat), чтобы служить вашему приложению. Вернемся

к нашему примеру с автомобилем. Если вы думаете о Spring Boot как об автомобиле, встроенный сервер — это ваш двигатель. Вместо того чтобы искать двигатель, устанавливать его и настраивать, Spring Boot предоставляет его “из коробки”. Он уже настроен и готов к работе!

Давайте напишем простой контроллер для нашего веб-приложения. В нашем контроллере будет один метод, который будет отвечать на HTTP-запросы GET на корневой URL (“/”). Этот метод будет возвращать простое текстовое сообщение. Вот как это выглядит:

```
package com.example.demo;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloController {

    @GetMapping("/")
    public String hello() {
        return "Hello, Spring Boot!";
    }
}
```

Здесь @RestController — это аннотация Spring, которая говорит, что этот класс является контроллером, и он должен обрабатывать веб-запросы. @GetMapping("/") - это другая аннотация Spring, которая указывает, что метод hello() должен обрабатывать HTTP-запросы GET на URL “/”.

Теперь, если вы запустите ваше приложение (например, через вашу IDE или команду ./mvnw spring-boot:run в командной строке), вы сможете перейти на http://localhost:8080 в вашем браузере и увидеть сообщение “Hello, Spring Boot!”.

Вот так просто создать веб-приложение с помощью Spring Boot!

Теперь давайте заглянем под капот и поговорим немного об Inversion of Control (IoC) и контейнерах в Spring.

IoC — это принцип, в котором контроль над объектами передается на внешний контейнер, вместо того чтобы они контролировались самим приложением. В случае с Spring, этот “контейнер” — это Spring IoC контейнер. Помнишь наш пример с кукольником? Вот здесь он становится действительно полезным. IoC контейнер

Spring — это как кукольник, который управляет объектами (куклами) в приложении. Он создает объекты, связывает их вместе, управляет их жизненным циклом и предоставляет их там, где они нужны.

Давайте посмотрим, как это работает на примере. Вернемся к нашему классу Car и Engine. Если мы хотим, чтобы SpringIoC контейнер управлял нашим объектом Car, мы можем аннотировать его аннотацией @Component:

```
@Component
public class Car {
    ...
}
```

Теперь Spring IoC контейнер создаст экземпляр класса Car для нас. Но что, если наш класс Car зависит от класса Engine? Мы можем использовать аннотацию @Autowired для того, чтобы Spring IoC контейнер внедрил (или “подключил”) экземпляр Engine в наш Car:

```
@Component
public class Car {
    private final Engine engine;

    @Autowired
    public Car(Engine engine) {
        this.engine = engine;
    }
    ...
}
```

Здесь Engine также должен быть аннотирован аннотацией @Component, чтобы Spring IoC контейнер мог создать экземпляр этого класса. Это и есть Dependency Injection (DI) — другой важный принцип в Spring, который позволяет нам легко управлять зависимостями между объектами.

Контейнер в Spring Framework является сердцем, которое управляет жизненным циклом объектов (бинов) и обеспечивает их взаимодействие. Он упрощает зависимости между объектами и предоставляет возможность инверсии управления (IoC) и внедрения зависимостей (DI).

Spring Framework имеет две основные реализации контейнера:

1. BeanFactory: Это базовый контейнер, который предоставляет функциональность IoC. Он определяет основной контракт, который должны выполнять все контейнеры. BeanFactory использует конфигурацию (XML или Java-based) для создания и управления объектами и их зависимостями. Однако он не предоставляет дополнительных возможностей, таких как интеграция с другими компонентами Spring, поддержка аннотаций и т. д.

2. ApplicationContext: Это расширение BeanFactory, которое предоставляет дополнительные возможности, такие как интеграция с другими компонентами Spring (например, AOP, Web), поддержка аннотаций и др. ApplicationContext является предпочтительным контейнером для большинства приложений, поскольку он предоставляет более широкий набор функций.

Контейнер работает следующим образом:

1. Чтение конфигурации: Контейнер считывает конфигурацию приложения, которая может быть определена с помощью XML, аннотаций или Java-based конфигурации.

2. Создание бинов: На основе предоставленной конфигурации контейнер создает и инициализирует объекты (бины) и их зависимости.

3. Управление жизненным циклом бинов: Контейнер управляет жизненным циклом бинов, вызывая их методы инициализации и разрушения в соответствии с конфигурацией.

4. Внедрение зависимостей: Контейнер автоматически внедряет зависимости между бинами, используя конфигурацию и аннотации.

5. Предоставление бинов: Когда приложение запрашивает бин из контейнера, он предоставляет соответствующий экземпляр бина, учитывая его область видимости (singleton, prototype и т. д.).

Итак, мы только что создали простое веб-приложение, рассмотрели основы IoC и DI, а также узнали, как работает встроенный сервер в Spring Boot. Ух, это было немного информации, но вы молодцы, что выдержали! В следующем разделе мы погрузимся глубже в конфигурацию Spring Boot и узнаем, как она может сделать нашу жизнь еще проще. Но пока дайте этой информации немного осесть. Практикуйтесь, создавайте простые приложения, играйте с IoC контейнером и DI, и скоро это все станет второй природой для вас.

Конфигурация Spring Boot приложения

Теперь, когда у нас есть основное представление о том, как создавать простое Spring Boot приложение, давайте перейдем к более интересной теме: настройке нашего приложения. Спойлер: вместе с Spring Boot это будет проще, чем вы думаете!

Итак, у нас есть наше приложение, и оно работает. Но что, если мы хотим изменить некоторые параметры, как, например, порт, на котором работает наше приложение? Или мы хотим иметь разные настройки для различных сред разработки, например, для тестового и продакшн окружения? Или мы хотим изменить название нашего приложения? Все это и многое другое можно сделать с помощью конфигурации в Spring Boot.

В Spring Boot есть два основных способа конфигурации: через файлы application.properties или application.yml и через Java-код. Давайте начнем с файлов конфигурации.

Файлы конфигурации

Spring Boot автоматически читает файлы application.properties или application.yml, которые находятся в корневом каталоге src/main/resources. В этих файлах вы можете задать различные параметры для вашего приложения.

Возможно, вы задаетесь вопросом: “Что такое .yml и .properties файлы, и в чем между ними разница?” Прекрасный вопрос!

.properties - это старый добрый формат файлов конфигурации Java. Каждый параметр задается в виде пары “ключ-значение”, разделенных знаком равенства. Вот пример:

```
server.port=8081
spring.application.name=My Awesome Spring Boot App
```

.yml (YAML Ain’t Markup Language, или YAML - это не язык разметки) - это более современный и гибкий формат. Он позволяет задавать структурированные данные с использованием отступов, что может быть более удобно для сложных конфигураций. Вот как выглядит та же конфигурация в формате .yml:

```yaml
server:
  port: 8081
```

```yaml
spring:
 application:
  name: My Awesome Spring Boot App
```

Какой из этих форматов выбрать - решать вам. Оба они работают отлично с Spring Boot, поэтому выбирайте тот, который вам больше нравится.

Давайте теперь поговорим о некоторых параметрах, которые вы можете задать в этих файлах конфигурации. Не беспокойтесь, мы не будем углубляться в слишком сложные вещи - только в те параметры, которые будут понятны и полезны нам на данном этапе:

- server.port: порт, на котором будет работать ваше приложение. По умолчанию это 8080, но вы можете задать любой другой порт.
- spring.application.name: имя вашего приложения. Оно может быть полезно для логирования и других вещей.
- spring.profiles.active: активные профили Spring. Мы поговорим об этом чуть позже.
- logging.level.root: уровень логирования для вашего приложения. Вы можете задать его, например, как INFO, WARN, ERROR или DEBUG.
- spring.main.banner-mode: режим баннера при запуске вашего приложения. Вы можете отключить баннер, установив этот параметр в OFF.

При запуске приложения Spring Boot автоматически загружает файлы конфигурации. Внутренне Spring использует библиотеку spring-boot-autoconfigure, которая предоставляет функциональность автоматической настройки.

Spring Boot автоматически сопоставляет значения из файлов конфигурации с полями в классах, аннотированных как @ConfigurationProperties. Эти классы обычно используются для группировки связанных свойств конфигурации.

Например, если у вас есть файл application.properties со следующим содержимым:

```
app.name=My Application
app.description=This is a sample application
```

Вы можете сопоставить эти значения с классом Java следующим образом:

```java
@ConfigurationProperties(prefix = "app")
public class AppProperties {
    private String name;
```

```
    private String description;

    // getters
    public String getName() {
        return this.name;
    }

    public String getDescription() {
        return this.description;
    }

    // setters
    public void setName(String name) {
        this.name = name;
    }

    public void setDescription(String description) {
        this.description = description;
    }
}
```

В этом примере Spring Boot автоматически сопоставит значения app.name и app.description из файла конфигурации с полями name и description в классе AppProperties.

Spring Boot также поддерживает валидацию свойств конфигурации с помощью аннотаций JSR-303, таких как @NotNull, @Min, @Max и т.д.

Профили Spring

В больших приложениях у вас, вероятно, будут разные настройки для разных сред разработки. Например, ваши настройки для тестирования могут отличаться от настроек для продакшн. Вот тут на помощь приходят профили Spring.

Профили в Spring Boot - это мощный инструмент, который позволяет разделять конфигурацию приложения на разные части в зависимости от среды, в которой оно работает. Вы можете иметь разные настройки для разработки, тестирования, стейджинга и производства, и легко переключаться между ними.

Профили работают вместе с файлами конфигурации, такими как application.properties или application.yml. Вы можете создать отдельные файлы конфигурации для каждого профиля, добавив имя профиля к имени файла. Например, application-development.properties для профиля разработки и application-production.properties для профиля производства.

В каждом из этих файлов вы можете задать свои собственные значения для различных параметров конфигурации. Например, вы можете иметь разные настройки для базы данных в разных средах.

Чтобы активировать определенный профиль, вы можете установить свойство spring.profiles.active в файле application.properties или application.yml, или установить его через переменную среды или параметр командной строки. Например, чтобы активировать профиль “development”, вы можете добавить следующую строку в application.properties:

spring.profiles.active=development

Или вы можете передать его как параметр командной строки при запуске вашего приложения:

./mvnw spring-boot:run -Dspring-boot.run.profiles=development

Профили также могут быть использованы внутри вашего Java кода с помощью аннотации @Profile. Это позволяет вам определять разные бины или конфигурации для разных профилей. Например, вы можете иметь разные реализации сервиса для разработки и производства:

```
@Service
@Profile("development")
public class DevelopmentMyService implements MyService {
    // реализация для разработки
}

@Service
@Profile("production")
public class ProductionMyService implements MyService {
    // реализация для производства
}
```

В этом примере, в зависимости от активного профиля, будет выбрана соответствующая реализация сервиса. Это дает большую гибкость при конфигурации вашего приложения для разных сред.

Давайте углубимся предыдущий пример и посмотрим, как мы можем использовать разные реализации сервиса в зависимости от профиля. Мы создадим интерфейс MyService и две его реализации - одну для разработки и одну для производства. Затем мы внедрим MyService в контроллер и увидим, какая реализация будет использоваться.

```
public interface MyService {
    String getMessage();
}

@Service
@Profile("development")
public class DevelopmentMyService implements MyService {
    @Override
    public String getMessage() {
        return "Development Service";
    }
}

@Service
@Profile("production")
public class ProductionMyService implements MyService {
    @Override
    public String getMessage() {
        return "Production Service";
    }
}

@RestController
public class MyController {
    private final MyService myService;

    public MyController(MyService myService) {
        this.myService = myService;
    }
```

```
    @GetMapping("/message")
    public String getMessage() {
        return myService.getMessage();
    }
}
```

В этом примере, когда мы делаем запрос на “/message”, контроллер вызывает метод getMessage() из MyService. Если активный профиль - “development”, то будет использоваться DevelopmentMyService, и мы получим “Development Service”. Если активный профиль - “production”, то будет использоваться ProductionMyService, и мы получим “Production Service”.

Для активации нужного профиля, можно добавить в файл application.properties следующую строку:

spring.profiles.active=development

Или для профиля “production”:

spring.profiles.active=production

Этот пример показывает, как можно легко переключаться между разными реализациями сервиса в зависимости от активного профиля, что является мощным инструментом для управления конфигурацией вашего приложения в разных средах.

Конфигурация через Java-код

Хотя файлы конфигурации очень полезны, иногда вам может потребоваться более динамическая конфигурация. В этом случае вы можете использовать Java-код для настройки вашего приложения.

В Spring Boot вы можете создать классы конфигурации, аннотированные @Configuration, и использовать методы с аннотацией @Bean для определения компонентов, которые будут управляться Spring IoC контейнером.

```
@Configuration
public class AppConfig {
    @Bean
    public MyService myService() {
        return new MyServiceImpl();
    }
}
```

В приведенном выше примере мы создаем класс AppConfig с аннотацией @Configuration. Затем мы определяем метод myService(), который возвращает новый экземпляр MyServiceImpl. Этот метод аннотирован @Bean, что означает, что Spring создаст и будет управлять экземпляром MyServiceImpl как bean.

Подобная конфигурация полезна, когда вам нужно выполнить какую-то дополнительную настройку для ваших компонентов или когда вы хотите создать несколько экземпляров одного и того же компонента с разными настройками.

Автоконфигурация

Теперь, когда мы поговорили о ручной конфигурации, давайте рассмотрим одну из самых великолепных особенностей Spring Boot: автоконфигурацию.

Автоконфигурация - это механизм, который позволяет Spring Boot автоматически настраивать ваше приложение на основе тех библиотек, которые присутствуют в вашем classpath. Например, если Spring Boot обнаруживает, что в вашем classpath есть библиотека Thymeleaf, он автоматически настроит шаблонизатор Thymeleaf для вас.

В этом заключается одно из основных преимуществ Spring Boot: он упрощает настройку и позволяет вам быстрее приступить к реальной работе. Вместо того чтобы тратить время на настройку различных библиотек, вы можете просто начать использовать их!

Итак, мы рассмотрели основные аспекты конфигурации Spring Boot приложения, включая файлы конфигурации, профили Spring, конфигурацию через Java-код и автоконфигурацию. Надеюсь, это поможет вам лучше понять, как настраивать ваше Spring Boot приложение и использовать его по максимуму.

Заключение

Поздравляю! Вы справились с огромным блоком информации о Spring Boot. Вы узнали о том, что такое Spring и Spring Boot, какие у них преимущества, как создать и конфигурировать простое Spring Boot приложение. Это было важное начало нашего путешествия в мир Spring, и я надеюсь, что вы наслаждаетесь процессом.

Теперь давайте подведем итоги и посмотрим, как все эти знания могут помочь вам в дальнейшем обучении и разработке приложений на Spring.

1. **Понимание Spring и Spring Boot.** Вы узнали о том, что такое Spring и Spring Boot, и какие преимущества они предлагают разработчикам. Это основа, на которой строится все остальное обучение. Это как знание алфавита перед тем, как начать читать книги.

2. **Создание и конфигурирование Spring Boot приложения.** Вы попрактиковались в создании и конфигурировании простого Spring Boot приложения. Это важный навык, который пригодится вам при создании любого Spring приложения. Считайте это своим первым опытом постройки собственного дома из кирпичей, которыми являются концепции и инструменты Spring.

3. **Понимание DI и IoC.** Вы познакомились с такими ключевыми принципами Spring, как внедрение зависимостей (DI) и инверсия контроля (IoC). Это как понимание того, как работает двигатель в автомобиле — оно помогает вам лучше понимать, как все работает вместе, и позволяет вам максимально эффективно использовать эти инструменты.

4. **Работа с файлами конфигурации и Java-кодом.** Вы научились использовать файлы конфигурации и Java-код для настройки вашего приложения. Это ключевые навыки, которые пригодятся вам при работе с любым Spring приложением. Считайте это своими первыми шагами в мире гастрономии — вы только начинаете узнавать, какие специи и ингредиенты нужны для приготовления вкусного блюда.

Итак, мы многое узнали о Spring и Spring Boot в этом уроке, но наш путь только начинается. В следующих уроках мы углубимся в более сложные аспекты разработки серверных приложений с использованием Spring, такие как работа с базами данных, безопасность и создание RESTful API.

Мы рассмотрим, как Spring Data упрощает работу с базами данных, как Spring Security помогает обеспечить безопасность вашего приложения, и как создать RESTful API с использованием Spring MVC. Эти темы представляют собой сердце любого современного веб-приложения, и я уверен, что вы с нетерпением ждете возможности исследовать их.

Также вы изучите Spring Security — библиотеку, которая помогает обеспечить безопасность вашего приложения. Она позволяет управлять аутентификацией и авторизацией, а также предоставляет множество других функций безопасности.

Кроме того, мы рассмотрим, как создать RESTful API с помощью Spring MVC. Это позволит вашему приложению взаимодействовать с другими приложениями и сервисами через интернет.

Но перед тем, как мы перейдем к этим темам, я хочу убедиться, что вы полностью поняли и усвоили материал этого урока. Поэтому я настоятельно рекомендую вам попрактиковаться в создании и конфигурировании Spring Boot приложений самостоятельно. Попробуйте создать несколько простых приложений, экспериментируйте с разными настройками и посмотрите, как онивлияют на поведение приложения. Это поможет вам лучше понять, как все работает вместе, и подготовит вас к следующему уроку.

Ваши знания и опыт, полученные на этом уроке, будут основой для дальнейшего обучения. Spring Boot — это мощный инструмент, который поможет вам быстро и эффективно разрабатывать приложения. Он обладает большими возможностями, но в то же время остается легким и понятным для новичков. Это делает его идеальным выбором для начинающих разработчиков и тех, кто хочет улучшить свои навыки в Java и Spring.

Уверен, что следующий урок будет еще более интересным и полезным. Помните, что самое важное в обучении - это практика. Поэтому не забывайте применять полученные знания на практике. Это поможет вам уверенно двигаться вперед и достигать новых вершин в мире разработки на Java и Spring.

Домашнее задание

Создайте базовое веб-приложение с использованием Spring Boot, которое будет включать в себя основные компоненты: контроллеры, сервисы и репозитории. Приложение может быть простым, например, приложение для управления книжной библиотекой с операциями CRUD (создание, чтение, обновление и удаление) книг.

Что можно почитать еще?

1. Изучение Spring Boot 2.0. Грег Тернквист
2. Освоение Spring Boot 2.0. Динеш Раджпут

Используемая литература

1. Spring Boot в действии. Крейг Уоллс
2. Spring Microservices в действии. Джон Карнелл
3. Cloud Native Java. Джош Лонг и Кенни Бастани