Folha de Dicas de Codificação: Introdução ao Spring Framework

Esta leitura fornece uma lista de referência de códigos que você encontrará ao aprender e usar o framework Spring em Java. Compreender esses conceitos ajudará você a escrever e depurar programas Java que utilizam o framework Spring. Vamos explorar os seguintes conceitos de codificação em Java:

- Aprendendo anotações do Spring
- Usando o Maven com o Spring
- Definindo projetos Spring

Mantenha esta leitura resumida disponível como referência enquanto avança em seu curso e consulte esta leitura ao começar a codificar com Java após este curso!

Aprendendo anotações do Spring

As anotações do Spring são metadados especiais no framework Spring que ajudam a configurar aplicações, reduzindo a necessidade de configuração baseada em XML. Elas simplificam a injeção de dependência, gerenciamento de beans, manipulação de transações e AOP (Programação Orientada a Aspectos). Anotações comuns como @Component, @Autowired e @Transactional permitem um desenvolvimento eficiente e modular. Ao usar anotações, os desenvolvedores podem escrever um código mais limpo, mais fácil de manter e facilmente testável.

Descrição	Exemplo
`@Component` marca uma classe como um componente gerenciado pelo Spring para auto-detecção e registro no contexto da aplicação.	`import org.springframework.stereotype.Component;` `@Component` `public class BookService {` `  public void listBooks() {` `  System.out.println("Listando todos os livros");` `  }` `}`
`@Controller` é um `@Component` especializado para controladores do Spring MVC que lidam com requisições web.	`import org.springframework.stereotype.Controller;` `import org.springframework.web.bind.annotation.GetMapping;` `@Controller` `public class BookController {` `  @GetMapping("/books")` `  public String showBooks() {` `  return "books"; // Retorna o nome da view "books"` `  }` `}`
`@Autowired` habilita a injeção automática de dependências em beans gerenciados pelo Spring.	`import org.springframework.beans.factory.annotation.Autowired;` `import org.springframework.stereotype.Controller;` `@Controller` `public class BookController {` `  @Autowired` `  private BookService bookService;` `  public void displayBooks() {` `  bookService.listBooks();` `  }` `}`
`@Configuration` define uma classe de configuração que declara beans e configurações para o contêiner Spring.	`import org.springframework.context.annotation.Bean;` `import org.springframework.context.annotation.Configuration;` `@Configuration` `public class AppConfig {` `  @Bean` `  public BookService bookService() {` `  return new BookService();` `  }` `}`

Descrição	Exemplo
`@RequestMapping` mapeia requisições web para métodos manipuladores em aplicações Spring MVC.	`import org.springframework.stereotype.Controller;` `import org.springframework.web.bind.annotation.RequestMapping;` `@Controller` `public class BookController {` `  @RequestMapping("/books")` `  public String getBooks() {` `  return "books";` `  }` `}`
`@PathVariable` extrai valores da URL e os vincula a parâmetros de método.	`import org.springframework.stereotype.Controller;` `import org.springframework.web.bind.annotation.PathVariable;` `import org.springframework.web.bind.annotation.GetMapping;` `@Controller` `public class BookController {` `  @GetMapping("/books/{id}")` `  public String getBookById(@PathVariable("id") String bookId) {` `  System.out.println("ID do Livro: " + bookId);` `  return "bookDetails";` `  }` `}`
`@RestController` é uma combinação de `@Controller` e `@ResponseBody`, usada para construir serviços web RESTful.	`import org.springframework.web.bind.annotation.RestController;` `import org.springframework.web.bind.annotation.GetMapping;` `import java.util.Arrays;` `import java.util.List;` `@RestController` `public class BookRestController {` `  @GetMapping("/api/books")` `  public List<String> getAllBooks() {` `  return Arrays.asList("Spring Boot", "Spring Cloud");` `  }` `}`
`@RequestParam` extrai parâmetros de consulta da URL e os vincula a parâmetros de método.	`import org.springframework.web.bind.annotation.GetMapping;` `import org.springframework.web.bind.annotation.RequestParam;` `import org.springframework.web.bind.annotation.RestController;` `@RestController` `public class BookRestController {` `  @GetMapping("/api/book")` `  public String getBookByTitle(@RequestParam("title") String title) {` `  return "Título do livro: " + title;` `  }` `}`
`@ResponseBody` indica que o valor de retorno de um método deve ser escrito diretamente no corpo da resposta HTTP.	`import org.springframework.web.bind.annotation.GetMapping;` `import org.springframework.web.bind.annotation.ResponseBody;` `import org.springframework.web.bind.annotation.RestController;` `@RestController` `public class BookRestController {` `  @GetMapping("/api/message")` `  @ResponseBody` `  public String getMessage() {` `  return "Olá, Spring!";` `  }` `}`
`@Value` injeta valores de arquivos de propriedades ou variáveis de ambiente em beans do Spring.	`import org.springframework.beans.factory.annotation.Value;` `import org.springframework.stereotype.Component;` `@Component` `public class Library {` `  @Value("${library.name}")` `  private String libraryName;`

Descrição	Exemplo
	`public void printLibraryName() {` `System.out.println("Nome da Biblioteca: " + libraryName);` `}` `}`
`@Scope` define o escopo de um bean, como singleton ou prototype.	`import org.springframework.context.annotation.Scope;` `import org.springframework.stereotype.Component;` `@Component` `@Scope("prototype")` `public class Book {` `// Bean com escopo prototype` `}`

Usando Maven com Spring

Maven é utilizado no Spring para gerenciar dependências, construir projetos e automatizar tarefas como compilar, empacotar e implantar aplicações. Ele simplifica a configuração do projeto com um arquivo pom.xml padronizado, garantindo builds consistentes e fácil integração das dependências do Spring.

Descrição	Exemplo
Gerenciando bibliotecas externas com dependências: Esta parte do arquivo `pom.xml` garante que as bibliotecas externas necessárias sejam incluídas no projeto. Cada dependência especifica um `<groupId>` (organização ou fornecedor), um `<artifactId>` (nome da biblioteca) e uma `<version>` (lançamento específico). O Maven baixa e gerencia automaticamente essas dependências.	`<dependencies>` `<dependency>` `<groupId>org.springframework.boot</groupId>` `<artifactId>spring-boot-starter-web</artifactId>` `<version>2.7.0</version>` `</dependency>` `</dependencies>`
Configurando o processo de build: Esta parte define como o projeto é compilado e empacotado. Inclui plugins como o `maven-compiler-plugin`, que especifica a versão do Java para compatibilidade do código-fonte.	`<build>` `<plugins>` `<plugin>` `<groupId>org.apache.maven.plugins</groupId>` `<artifactId>maven-compiler-plugin</artifactId>` `<version>3.8.1</version>` `<configuration>` `<source>1.8</source>` `<target>1.8</target>` `</configuration>` `</plugin>` `</plugins>` `</build>`
Adicionando repositórios personalizados para dependências: Se as dependências necessárias não estiverem disponíveis no repositório Maven Central padrão, esta parte permite especificar repositórios adicionais onde o Maven pode procurá-las.	`<repositories>` `<repository>` `<id>spring-releases</id>` `<url>https://repo.spring.io/release</url>` `</repository>` `</repositories>`
Definindo propriedades em todo o projeto: Este recurso permite definir valores reutilizáveis, como a versão do Java, facilitando a manutenção da configuração em todo o projeto.	`<properties>` `<java.version>1.8</java.version>` `</properties>`

Descrição	Exemplo
Gerenciando diferentes ambientes com perfis: Perfis ajudam a configurar diferentes configurações para vários ambientes (por exemplo, desenvolvimento, teste, produção). Eles podem ser ativados usando opções de linha de comando.	`<profiles>` `  <profile>` `  <id>dev</id>` `  <properties>` `  <env>development</env>` `  </properties>` `  </profile>` `</profiles>`
Exemplo completo de um projeto Maven (pom.xml): Este é um exemplo completo de um arquivo `pom.xml` que gerencia dependências, configurações de build e plugins para uma aplicação simples do Spring Boot.	`<project xmlns="<a href="http://maven.apache.org/POM/4.0.0"" target="_blank" rel="noopener noreferrer">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 http://maven.apache.org/xsd/maven-4.0.0.xsd">` `  <modelVersion>4.0.0</modelVersion>` `  <groupId>com.example</groupId>` `  <artifactId>spring-demo</artifactId>` `  <version>1.0-SNAPSHOT</version>` `  <packaging>jar</packaging>` `  <dependencies>` `  <dependency>` `  <groupId>org.springframework.boot</groupId>` `  <artifactId>spring-boot-starter-web</artifactId>` `  <version>2.7.0</version>` `  </dependency>` `  </dependencies>` `  <build>` `  <plugins>` `  <plugin>` `  <groupId>org.springframework.boot</groupId>` `  <artifactId>spring-boot-maven-plugin</artifactId>` `  </plugin>` `  </plugins>` `  </build>` `</project>`

Definindo projetos Spring

Definir projetos Spring é importante para estabelecer uma estrutura clara, gerenciar dependências de forma eficiente e garantir uma integração suave com frameworks como o Spring Boot. Um projeto bem definido simplifica o desenvolvimento, teste e implantação, mantendo a escalabilidade e a manutenibilidade.

Descrição	Exemplo
Verifique a instalação: Abra seu terminal ou prompt de comando e execute os seguintes comandos para verificar as instalações.	`java -version` `mvn -version` Ambos os comandos devem retornar informações de versão se instalados corretamente.
Crie um novo projeto Maven usando a linha de comando: Abra seu terminal, navegue até o diretório desejado e execute o seguinte comando.	`mvn archetype:generate -DgroupId=com.example -DartifactId=spring-beginner-project -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false` `groupId`: Um identificador exclusivo para seu projeto (por exemplo, com.example). artifactId: O nome do seu projeto (por exemplo, spring-beginner-project).

Descrição	Exemplo
Entenda a estrutura do projeto: Um layout padrão de projeto Maven se parece com isto.	``` spring-beginner-project ├── src │ ├── main │ │ ├── java │ │ └── resources │ └── test │ ├── java │ └── resources ├── pom.xml ```
Adicione dependências do Spring: Abra `pom.xml` e adicione as dependências necessárias do Spring.	``` <dependencies> <dependency> <groupId>org.springframework</groupId> <artifactId>spring-context</artifactId> <version>5.3.28</version> </dependency> <dependency> <groupId>org.springframework</groupId> <artifactId>spring-webmvc</artifactId> <version>5.3.28</version> </dependency> </dependencies> ``` Execute `mvn clean install` para baixar as dependências.
Crie uma classe de configuração: Define beans e configurações para a aplicação.	``` package com.example; import org.springframework.context.annotation.Bean; import org.springframework.context.annotation.Configuration; @Configuration public class AppConfig { @Bean public HelloWorld helloWorld() { return new HelloWorld(); } } ```
Crie um bean simples: Uma classe básica para demonstrar um bean gerenciado pelo Spring.	``` package com.example; public class HelloWorld { public void sayHello() { System.out.println("Hello, World!"); } } ```
Crie uma classe principal da aplicação: Carrega o contexto da aplicação Spring e recupera o bean.	``` package com.example; import org.springframework.context.ApplicationContext; import org.springframework.context.annotation.AnnotationConfigApplicationContext; public class MainApp { public static void main(String[] args) { ApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class); HelloWorld helloWorld = context.getBean(HelloWorld.class); helloWorld.sayHello(); } } ```

Descrição	Exemplo
Execute sua aplicação: Compile e execute a aplicação usando os seguintes comandos.	`mvn compile` `mvn exec:java -Dexec.mainClass="com.example.MainApp"`

Author(s)

Ramanujam Srinivasan
Lavanya Thiruvali Sunderarajan