Folha de Dicas de Codificação: Mergulho Profundo no Spring Boot

Esta leitura fornece uma lista de referência de código que você encontrará enquanto aprende e utiliza 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:

- Criando uma aplicação Spring Boot
- Gerenciando uma aplicação Spring Boot
- Versionamento de API REST
- Enviando e recebendo dados usando parâmetros de API REST

Mantenha esta leitura resumo disponível como referência à medida que avança em seu curso e consulte esta leitura ao começar a codificar em Java após este curso!

Criando uma aplicação Spring Boot

Spring Boot simplifica o desenvolvimento de aplicações Java ao otimizar a configuração e a configuração.

Descrição	Exemplo

Criando um controlador simples: Um controlador lida com solicitações web no Spring Boot. A classe `BookController` define um endpoint que retorna informações sobre livros.	`package com.example.bookfinder.controller;`
	
	`import org.springframework.web.bind.annotation.GetMapping;`
	`import org.springframework.web.bind.annotation.RestController;`
	
	`@RestController`
	`public class BookController {`
	
	`  @GetMapping("/book")`
	`  public String getBook() {`
	`  return "Effective Java by Joshua Bloch";`
	`  }`
	`}`

Usando getters e setters: Em Java, getters e setters são usados para acessar as variáveis privadas de uma classe. Vamos criar uma classe `Book` simples com algumas propriedades.	`package com.example.bookfinder.model;`
	
	`public class Book {`
	
	`  private String title;`
	`  private String author;`
	
	`  public Book(String title, String author) {`
	`  this.title = title;`
	`  this.author = author;`
	`  }`
	
	`  public String getTitle() {`
	`  return title;`
	`  }`
	
	`  public void setTitle(String title) {`
	`  this.title = title;`
	`  }`
	
	`  public String getAuthor() {`
	`  return author;`
	`  }`
	
	`  public void setAuthor(String author) {`
	`  this.author = author;`
	`  }`
	`}`

Descrição	Exemplo

Registro no Spring Boot: O registro ajuda a rastrear o comportamento da aplicação. O `Logger` é usado para registrar mensagens em uma aplicação Spring Boot. Aqui, `LoggerFactory` cria um logger para `BookController`.	`package com.example.bookfinder.controller;`
	
	`import org.slf4j.Logger;`
	`import org.slf4j.LoggerFactory;`
	`import org.springframework.web.bind.annotation.GetMapping;`
	`import org.springframework.web.bind.annotation.RestController;`
	
	`@RestController`
	`public class BookController {`
	
	`private static final Logger logger = LoggerFactory.getLogger(BookController.class);`
	
	`@GetMapping("/book")`
	`public String getBook() {`
	`logger.info("getBook() method called");`
	`return "Effective Java by Joshua Bloch";`
	`}`
	`}`

Gerenciando uma aplicação Spring Boot

Descrição	Exemplo

Usando Maven: Certifique-se de que seu `pom.xml` tenha a seguinte configuração sob a tag `<build>`.	`<build>`
	`<plugins>`
	`<plugin>`
	`<groupId>org.springframework.boot</groupId>`
	`<artifactId>spring-boot-maven-plugin</artifactId>`
	`</plugin>`
	`</plugins>`
	`</build>`

Executando o comando Maven package: Isso criará um arquivo JAR no diretório `target`.	`mvn clean package`
Versionamento no JAR Especifique a versão no `pom.xml` para rastrear alterações e atualizações.	`<version>1.0.0-SNAPSHOT</version>`

Adicionando dependências	`<dependencies>`
	`<dependency>`
	`<groupId>org.springframework.boot</groupId>`

Descrição	Exemplo

no Maven: Adicione dependências dentro da seção `<dependencies>` do `pom.xml`.	`<artifactId>spring-boot-starter-web</artifactId>`
	`</dependency>`
	`</dependencies>`

Tratamento de exceções - Exceção personalizada: Defina uma classe de exceção personalizada para um melhor tratamento de erros.	`public class CustomException extends RuntimeException {`
	`public CustomException(String message) {`
	`super(message);`
	`}`
	`}`

Tratamento de exceções - Usando `@ExceptionHandler`: Trate exceções no nível do controlador.	`@RestController`
	`public class MyController {`
	`@GetMapping("/example")`
	`public String example() {`
	`throw new CustomException("Esta é uma mensagem de exceção personalizada.");`
	`}`
	
	`@ExceptionHandler(CustomException.class)`
	`public ResponseEntity<String> handleCustomException(CustomException ex) {`
	`return new ResponseEntity<>(ex.getMessage(), HttpStatus.BAD_REQUEST);`
	`}`
	`}`

Tratamento de exceções - Usando `@ControllerAdvice`: Trate exceções globalmente em todos os controladores.	`@ControllerAdvice`
	`public class GlobalExceptionHandler {`
	`@ExceptionHandler(CustomException.class)`
	`public ResponseEntity<String> handleCustomException(CustomException ex) {`
	`return new ResponseEntity<>(ex.getMessage(), HttpStatus.BAD_REQUEST);`
	`}`
	
	`@ExceptionHandler(Exception.class)`
	`public ResponseEntity<String> handleGeneralException(Exception ex) {`
	`return new ResponseEntity<>("Ocorreu um erro: " + ex.getMessage(), HttpStatus.INTERNAL_SERVER_ERROR);`
	`}`
	`}`

Propriedades da aplicação em YAML: Converta as propriedades da aplicação para o formato YAML.	`server:` `port: 9090`

Versionamento da API REST

Descrição	Exemplo

Definindo um controlador REST simples (Versão 1):
Crie um controlador REST do Spring Boot para lidar com solicitações da API para a versão 1.
	`@RestController`
	`public class GreetingControllerV1 {`
	`  @GetMapping("/api/v1/greeting")`
	`  public String getGreetingV1() {`
	`  return "Olá, Mundo! - v1";`
	`  }`
	`}`

Definindo um controlador REST simples (Versão 2):
Crie outro controlador para lidar com a versão 2 da API.
	`@RestController`
	`public class GreetingControllerV2 {`
	`  @GetMapping("/api/v2/greeting")`
	`  public String getGreetingV2() {`
	`  return "Olá, Universo! - v2";`
	`  }`
	`}`

Executando a aplicação Spring Boot: Inicie a aplicação usando o Maven.	`mvn spring-boot:run`
Acessando os endpoints da API: Use um navegador ou uma ferramenta como `curl` para testar a API.	http://localhost:8080/api/v1/greeting http://localhost:8080/api/v2/greeting

Enviando e recebendo dados usando parâmetros da API REST

Descrição	Exemplo

Descrição	Exemplo

Enviando dados com uma solicitação GET: Uma solicitação GET recupera dados e pode incluir parâmetros na URL.	`@RestController`
	`@RequestMapping("/api")`
	`public class DataController {`
	`  @GetMapping("/data")`
	`  public String getData(@RequestParam(value = "name", defaultValue = "World") String name) {`
	`  return "Olá, " + name + "!";`
	`  }`
	`}`

Enviando dados com uma solicitação POST: Uma solicitação POST é usada para criar novos recursos.	`@RestController`
	`@RequestMapping("/api")`
	`public class UserController {`
	`  @PostMapping("/users")`
	`  public String createUser(@RequestBody User user) {`
	`  return "Usuário criado com o nome: " + user.getName();`
	`  }`
	`}`

Enviando dados com uma solicitação PUT: Uma solicitação PUT atualiza um recurso existente.	`@RestController`
	`@RequestMapping("/api")`
	`public class ProductController {`
	`  @PutMapping("/products/{id}")`
	`  public String updateProduct(@PathVariable Long id, @RequestBody Product product) {`
	`  return "Produto com ID " + id + " atualizado para o nome: " + product.getName();`
	`  }`
	`}`

Enviando dados com uma solicitação DELETE: Uma solicitação DELETE remove um recurso pelo seu ID.	`@RestController`
	`@RequestMapping("/api")`
	`public class OrderController {`
	`  @DeleteMapping("/orders/{id}")`
	`  public String deleteOrder(@PathVariable Long id) {`
	`  return "Pedido com ID " + id + " foi deletado.";`
	`  }`
	`}`

Exemplo cumulativo - Gerenciando livros: Uma API simples para gerenciar uma coleção de livros.	`@RestController`
	`@RequestMapping("/api")`
	`public class BookController {`
	`  private Map<Long, Book> bookRepository = new HashMap<>();`
	`  @GetMapping("/books/{id}")`
	`  public Book getBook(@PathVariable Long id) {`
	`  return bookRepository.get(id);`
	`  }`
	`  @PostMapping("/books")`
	`  public String addBook(@RequestBody Book book) {`
	`  bookRepository.put(book.getId(), book);`
	`  return "Livro adicionado com ID: " + book.getId();`
	`  }`
	`  @PutMapping("/books/{id}")`

Descrição	Exemplo

	`public String updateBook(@PathVariable Long id, @RequestBody Book book) {`
	`  if (!bookRepository.containsKey(id)) {`
	`  return "Livro não encontrado!";`
	`  }`
	`  bookRepository.put(id, book);`
	`  return "Livro atualizado com ID: " + id;`
	`  }`
	`  @DeleteMapping("/books/{id}")`
	`  public String deleteBook(@PathVariable Long id) {`
	`  bookRepository.remove(id);`
	`  return "Livro deletado com ID: " + id;`
	`  }`
	`}`

Testando a API:
Comandos curl de exemplo para enviar dados.
	`# Solicitação GET com parâmetros de consulta`
	`curl -X GET "http://localhost:8080/api/products?category=electronics&sort=price"`
	
	`# Solicitação POST com corpo JSON`
	`curl -X POST "http://localhost:8080/api/users" \`
	`  -H "Content-Type: application/json" \`
	`  -d '{"name": "John Doe", "email": "john.doe@example.com"}'`

Author(s)

Ramanujam Srinivasan
Lavanya Thiruvali Sunderarajan