

Desenvolvimento Web Back-end

Java Spring Boot

Spring

- Spring é um projeto de software livre que disponibiliza uma abordagem simplificada e modular para criar aplicativos com Java.
- A família de projetos Spring começou em 2003 como uma resposta às complexidades iniciais de desenvolvimento em Java e fornece suporte para o desenvolvimento de aplicativos Java.
- O nome, Spring, sozinho, geralmente se refere à própria estrutura de aplicativos ou a todo o grupo de projetos ou módulos. Java Spring Boot é um módulo específico criado como uma extensão da estrutura do Spring.

Spring

- O Spring é um framework para o desenvolvimento de aplicativos Java que foi criado para resolver muitos dos desafios enfrentados pelos desenvolvedores de software.
- Ele fornece um conjunto abrangente de recursos para simplificar o desenvolvimento de aplicativos empresariais, incluindo gerenciamento de dependências, injeção de dependência, segurança, integração com bancos de dados e muito mais.
- O Spring tornou-se extremamente popular entre os desenvolvedores devido à sua flexibilidade e facilidade de uso.

Qual a diferença entre o Spring e o Spring Boot?

- Spring é uma estrutura de aplicativo de software livre baseada em Java que abrange muitos projetos menores sob o seu guarda-chuva.
- Outros projetos populares de Spring incluem o Spring Data, o Spring Cloud e o Spring Security, para citar apenas alguns.
- Para entender totalmente as diferenças entre o Spring e o Spring Boot, primeiro precisaremos definir alguns termos principais.

O que são microserviços?

- Microserviços são uma abordagem para a arquitetura de desenvolvimento de software.
- O "micro" em microserviços se refere ao código sendo fornecido em pequenas partes ou componentes gerenciáveis, e cada "serviço" ou função principal é criado e implantado independentemente dos outros serviços.
- Os componentes independentes trabalham em conjunto e se comunicam por meio de documentos de API prescritos chamados de contratos.

O que são microsserviços?

- A pequena escala e o isolamento relativo desses microsserviços disponibilizam muitos benefícios.
- Por exemplo, como esse tipo de arquitetura é distribuído e acoplado de forma flexível, o aplicativo inteiro não será interrompido se um componente falhar.
- Outros benefícios incluem a produtividade aprimorada, a manutenção mais fácil, o melhor alinhamento de negócios e maior tolerância a falhas.

O que é injeção de dependências (DI)?

- É um conceito de programação utilizado em desenvolvimento de software.
- Ele se refere à técnica de fornecer as dependências de um objeto a partir de uma fonte externa, em vez de criá-las dentro do objeto. Isso ajuda a tornar o código mais flexível, modular e fácil de testar, pois as dependências podem ser facilmente substituídas por outras implementações.
- Na programação orientada a objeto como Java, os objetos que dependem de outros objetos são chamados de dependências.

O que é injeção de dependências (DI)?

- Normalmente o objeto de recebimento ou dependente é chamado de cliente e o objeto do qual o cliente depende é chamado de serviço.
- Portanto, a injeção de dependência passa o serviço para o cliente ou "injeta" a dependência usando o código chamado de injetor.
- A DI elimina a necessidade do cliente especificar qual serviço usar, o injetor executa esse trabalho para o cliente.

Exemplo de injeção de dependência

- Um exemplo simples de injeção de dependência em Java pode ser quando uma classe depende de outra classe para funcionar corretamente.
- Em vez de instanciar a classe dependente dentro da classe principal, a dependência é passada por meio de um construtor ou método, permitindo que a classe principal seja mais flexível e reutilizável.

Exemplo em Java

```
// Interface que define o contrato da dependência
interface Servico {
    void executar();
}
```

```
// Classe que implementa a interface Servico
class ServicoImpl implements Servico {
    @Override
    public void executar() {
        System.out.println("Executando o serviço...");
    }
}
```

// Classe que recebe a dependência por injeção

```
class Cliente {
```

```
    private Servico servico;
```

// Construtor que recebe a dependência

```
    public Cliente(Servico servico) {
```

```
        this.servico = servico;
```

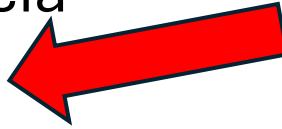
```
    }
```

```
    public void usarServico() {
```

```
        servico.executar();
```

```
    }
```

```
}
```



// Exemplo de uso

```
public class Main {
```

```
    public static void main(String[] args) {
```

```
        Servico servico = new ServicoImpl(); // Instância da implementação da dependência
```

```
        Cliente cliente = new Cliente(servico); // Injeção da dependência
```

```
        cliente.usarServico(); // Chamada do método que utiliza a dependência
```

```
    }
```

```
}
```

O que é a convenção sobre a configuração?

- A convenção sobre configuração, às vezes chamada de codificação por convenção, é um conceito usado em estruturas de aplicativos para reduzir o número de decisões que um desenvolvedor precisa tomar.
- Ela segue o princípio "não se repetir" para evitar escrever código redundante.
- A codificação por convenção se esforça para manter a flexibilidade, enquanto permite que um desenvolvedor escreva apenas código para os aspectos não convencionais do aplicativo que está criando.

O que é a convenção sobre a configuração?

- Quando o comportamento desejado do aplicativo corresponde às convenções estabelecidas, o aplicativo será executado por padrão sem precisar gravar arquivos de configuração.
- O desenvolvedor só precisará escrever explicitamente os arquivos de configuração se o comportamento desejado se desviar da "convenção."

O que é uma classe de configuração no Spring?

- Em um aplicativo Spring, uma classe de configuração é responsável por definir a configuração do container de IoC (Inversão de Controle) do Spring.
- Essa classe geralmente é anotada com `@Configuration` e contém métodos anotados com `@Bean` que definem como os beans (objetos gerenciados pelo Spring) devem ser criados e configurados.

Exemplo em Java Spring

- Em Java Spring, a injeção de dependência é feita de forma automática pelo **container de IoC** (Inversão de Controle) do Spring. Aqui está um exemplo simples de como você pode usar a injeção de dependência em um aplicativo Spring:
- Suponha que temos uma interface `Servico` e sua implementação `ServicoImpl`, assim como no exemplo anterior.
- O Spring irá gerenciar a criação do **bean** `ServicoImpl` e injetá-lo automaticamente no construtor da classe `Cliente` quando ela for criada. Isso demonstra como o Spring facilita a injeção de dependências em um aplicativo Java.

// Interface que define o contrato da dependência

```
public interface Servico {  
    void executar();  
}
```

// Classe que implementa a interface Servico

```
public class ServicoImpl implements Servico {  
    @Override  
    public void executar() {  
        System.out.println("Executando o serviço...");  
    }  
}
```

Em seguida, vamos criar uma **classe de configuração** que define como os beans (objetos gerenciados pelo Spring) serão criados e injetados:

```
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;
```

```
@Configuration
```

```
public class AppConfig {
```

```
    @Bean
```

```
    public Servico servico() {  
        return new ServicoImpl();
```

```
    }
```

```
}
```

Definição de um bean em spring



- Agora, vamos criar uma classe que utiliza a injeção de dependência do Spring:

```
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Component;
```

```
@Component
```

```
public class Cliente {
```

```
    private Servico servico;
```

```
    @Autowired
```

```
    public Cliente(Servico servico) {
```

```
        this.servico = servico;
```

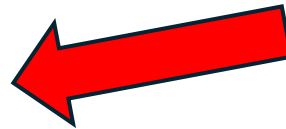
```
    }
```

```
    public void usarServico() {
```

```
        servico.executar();
```

```
    }
```

```
}
```



Por fim, vamos criar uma classe principal para executar o aplicativo Spring:

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class Main {
    public static void main(String[] args) {
        ApplicationContext context=new AnnotationConfigApplicationContext(AppConfig.class);

        Cliente cliente = context.getBean(Cliente.class);
        cliente.usarServico();
    }
}
```

O que é um bean?

- Em termos de desenvolvimento de software, no contexto do Spring Framework, um "bean" é um objeto gerenciado pelo container de IoC (Inversão de Controle) do Spring. Em outras palavras, um bean é uma instância de uma classe que é configurada e gerenciada pelo Spring.
- Os beans são definidos nas configurações do Spring (como classes de configuração ou anotações) e o container do Spring é responsável por instanciar, configurar e injetar esses beans quando necessário. Os beans do Spring são componentes fundamentais para a aplicação, pois representam os objetos que colaboram para fornecer funcionalidades específicas.
- Em resumo, no Spring Framework, um bean é um objeto gerenciado pelo container IoC do Spring, que é configurado e injetado em outras partes da aplicação conforme necessário.

O que é container de IoC (Inversão de Controle) do Spring

- O container de IoC (Inversão de Controle) do Spring é um mecanismo fundamental do Spring Framework que gerencia a criação, configuração e controle de objetos (beans) em uma aplicação. A IoC é um princípio de design de software no qual o controle da criação e do ciclo de vida dos objetos é invertido, ou seja, em vez de ser responsabilidade do desenvolvedor criar e gerenciar manualmente as dependências dos objetos, essa responsabilidade é transferida para o container de IoC.
- No contexto do Spring Framework, o container de IoC é responsável por instanciar, configurar e injetar as dependências dos beans de forma automática, seguindo as configurações definidas pelo desenvolvedor. Isso ajuda a reduzir o acoplamento entre os componentes da aplicação, facilita a manutenção do código e promove a reutilização de componentes.

Alguns pontos-chave sobre o container de IoC do Spring:

- **Inversão de Controle (IoC):** O princípio da Inversão de Controle é central no Spring Framework. Em vez de ser o desenvolvedor a controlar a criação e a configuração dos objetos (beans), essa responsabilidade é invertida para o container de IoC do Spring. O container é responsável por gerenciar o ciclo de vida dos objetos, criando, configurando e injetando as dependências automaticamente.
- **Gerenciamento de Dependências:** O container de IoC do Spring gerencia as dependências entre os beans da aplicação, permitindo a injeção de dependências de forma automática. Isso reduz o acoplamento entre os componentes da aplicação e facilita a reutilização e a manutenção do código.
- **Configuração Declarativa:** A configuração dos beans no Spring é feita de forma declarativa, por meio de anotações, XML ou classes de configuração. Isso permite uma configuração mais flexível e organizada, separando a lógica de negócios da configuração técnica.

Estrutura do Spring versus Spring Boot

| | Spring | Spring Boot |
|------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| O que é? | Uma estrutura de aplicativo Web de software livre baseada em Java. | Uma extensão ou módulo criado na estrutura do Spring. |
| O que faz? | Fornecer um ambiente flexível e completamente configurável usando ferramentas e bibliotecas de códigos predefinidos para criar aplicativos Web personalizados e flexíveis. | Fornecer a capacidade de criar no Spring aplicativos autônomos que podem ser executados imediatamente sem a necessidade de anotações, configuração XML ou gravação de muitos códigos adicionais. |

Estrutura do Spring versus Spring Boot

Quando devo utilizar?

Use o Spring quando desejar:

- Flexibilidade.
- Uma abordagem sem conceito.*
- Para remover dependências do código personalizado.
- Para implementar uma configuração muito exclusiva.
- Para desenvolver aplicativos empresariais.

Use o Spring Boot quando desejar:

- Facilidade de uso.
- Uma abordagem com conceito.*
- Para obter aplicativos de qualidade para executar rapidamente e reduzir o tempo de desenvolvimento.
- Para evitar escrever código clichê ou configurar XML.
- Para desenvolver APIs REST.

Estrutura do Spring versus Spring Boot

| Qual é o seu principal recurso? | Injeção de dependência | Configuração automática |
|---------------------------------|---------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| Tem servidores inseridos? | Não. No Spring, você precisará configurar explicitamente os servidores. | Sim, o Spring Boot vem com servidores HTTP internos, como <u>Tomcat</u> e Jetty. |
| Como ele é configurado? | A estrutura do Spring fornece flexibilidade, mas sua configuração precisa ser criada manualmente. | O Spring Boot automaticamente configura o Spring e outras estruturas de terceiros pelo princípio padrão de "convenção sobre a configuração". |

Estrutura do Spring versus Spring Boot

| | | |
|-------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| Preciso saber como trabalhar com XML? | No Spring, é necessário conhecer a configuração do XML. | O Spring Boot não requer configuração XML. |
| Tem ferramentas da CLI para aplicativos de desenvolvimento/teste? | A estrutura do Spring Framework por si só não fornece ferramentas da CLI para desenvolver ou testar aplicativos. | Como um módulo do Spring, o Spring Boot tem uma ferramenta de CLI para desenvolver e testar aplicativos baseados no Spring. |
| Funciona com uma abordagem com ou sem conceito? | Sem conceito* | Com conceito* |

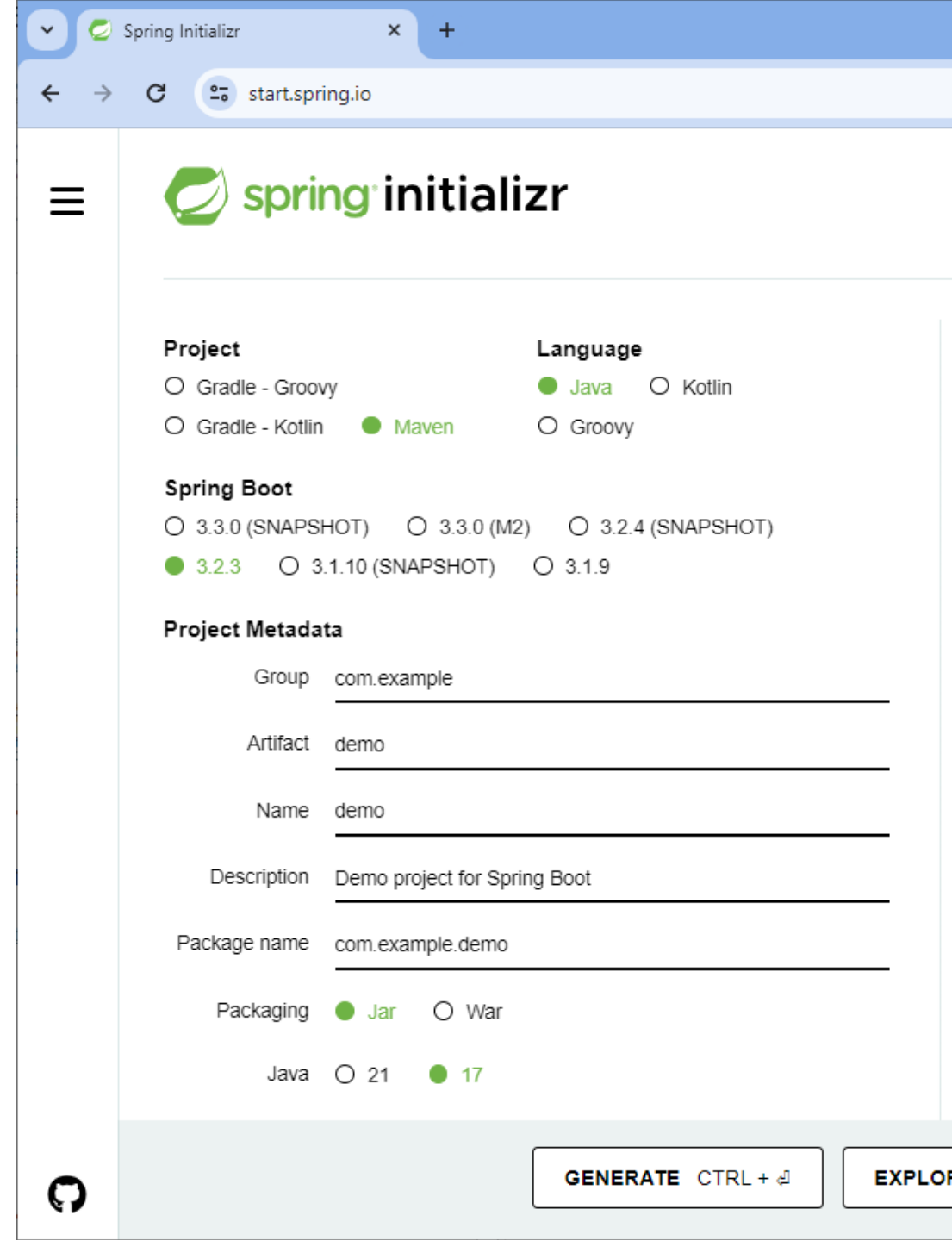
Ferramentas da CLI do Spring Boot

As ferramentas disponíveis na CLI do Spring Boot que ajudam os desenvolvedores a criar, configurar e executar aplicativos Spring de forma mais eficiente e produtiva.

- **Spring Initializr:** A CLI do Spring Boot inclui o Spring Initializr, uma ferramenta que permite inicializar rapidamente projetos Spring Boot com base em modelos predefinidos. Com o Spring Initializr, é possível configurar as dependências, versões e estrutura do projeto de forma simples e rápida. (<https://start.spring.io/>)
- **Spring Boot CLI:** A própria CLI do Spring Boot é uma ferramenta que permite criar, executar e testar aplicativos Spring Boot a partir da linha de comando. Com a CLI do Spring Boot, é possível criar novos projetos, executar scripts Groovy, iniciar a aplicação, entre outras funcionalidades. (<https://docs.spring.io/spring-boot/docs/current/reference/html/cli.html>)
- **Spring Boot DevTools:** Embora não seja exclusivamente uma ferramenta da CLI, o Spring Boot DevTools é uma coleção de ferramentas que auxiliam no desenvolvimento de aplicações Spring Boot. Ele fornece recursos como reinicialização automática da aplicação, recarregamento de propriedades, entre outros, para facilitar o desenvolvimento e aumentar a produtividade. (<https://www.baeldung.com/spring-boot-devtools>)

1º Projeto em Spring Boot

1. Entre no site <https://start.spring.io/>
2. Escolhas as configurações a seguir:
3. Escolha a opção “Generate”
4. Descompacte a pasta
5. Carregue o SpringToolsSuite

A screenshot of the Spring Initializr website in a web browser. The browser tab is titled 'Spring Initializr' and the address bar shows 'start.spring.io'. The website has a green and white color scheme. On the left, there is a hamburger menu icon. The main content area is titled 'spring initializr'. Below this, there are sections for 'Project', 'Language', 'Spring Boot', and 'Project Metadata'. The 'Project' section has radio buttons for 'Gradle - Groovy', 'Gradle - Kotlin', and 'Maven' (which is selected). The 'Language' section has radio buttons for 'Java' (selected), 'Kotlin', and 'Groovy'. The 'Spring Boot' section has radio buttons for '3.3.0 (SNAPSHOT)', '3.3.0 (M2)', '3.2.4 (SNAPSHOT)', '3.2.3' (selected), '3.1.10 (SNAPSHOT)', and '3.1.9'. The 'Project Metadata' section has input fields for 'Group' (com.example), 'Artifact' (demo), 'Name' (demo), 'Description' (Demo project for Spring Boot), and 'Package name' (com.example.demo). There are also radio buttons for 'Packaging' (Jar selected, War) and 'Java' (21, 17 selected). At the bottom right, there are two buttons: 'GENERATE CTRL + G' and 'EXPLORE'.

Estrutura de pastas

- Quando você cria um projeto Spring Boot usando o Spring Initializr, a estrutura de pastas padrão criada pode variar dependendo das configurações e das dependências selecionadas durante a inicialização do projeto.
- No entanto, vou descrever a estrutura de pastas comuns que você pode encontrar em um projeto Spring Boot típico são:

Estrutura de pastas



src/main/java: Esta pasta é onde você coloca todos os seus arquivos Java. Aqui você encontrará o pacote base da sua aplicação e as classes Java que compõem a lógica de negócios do seu aplicativo.

src/main/resources: Nesta pasta, você coloca recursos estáticos, como arquivos de propriedades, arquivos de configuração, arquivos de template, arquivos de dados, etc. Esses recursos são usados pela aplicação durante a execução.

src/test/java: Esta pasta contém os arquivos de teste unitário para testar a lógica do seu aplicativo. Aqui você escreve testes para garantir que o código funcione conforme o esperado.

src/test/resources: Assim como em src/main/resources, esta pasta contém recursos usados nos testes unitários, como arquivos de configuração de teste, dados de teste, etc.

pom.xml ou build.gradle: Estes são arquivos de configuração do Maven (no caso do pom.xml) ou do Gradle (no caso do build.gradle). Eles contêm informações sobre as dependências do projeto, plugins utilizados, configurações de compilação, entre outras configurações relacionadas ao build do projeto.