Spring Framework na Prática

Sumario

O que é o Spring?

O **Spring** é um framework open source para a plataforma **Java** criado por Rod Johnson e descrito em seu livro "Expert One-on-One: JEE Design e Development". Trata-se de um framework não intrusivo, baseado nos padrões de projeto inversão de controle (IoC) e injeção de dependência.

Por que não Intrusivo?

Por que não impõem padrões e fluxos de vida complexos, apesar deles existirem.

O que é inversão de controle?

É uma técnica utilizada para eliminar fortes acoplamentos, as responsabilidades entre Classes é forçadamente isolada. Aplicando a inversão de controle, retiramos responsabilidades de trechos de código e a isolamos.

O que é injeção de dependência?

É uma técnica utilizada em conjunto com a inversão de controle e, como consequência da inversão de controle, é necessário efetuar a injeção de dependência.

Ela consiste em realizar construções de componentes isoladamente e injetá-los através de construtores ou propriedades onde ela são requisitadas.

Como o Spring faz IOC?

Através de esteriótipos representados por anotações e por Profiles, também representados por anotações.

O que vamos aprender?

- Montar um projeto a partir do zero com spring boot e seus módulos, utilizando o Maven para o gerenciamento de dependências;

- Construir um front-end utilizando o gerenciador de templates Thymeleaf integrado como Spring MVC;

- Disponibilizar interfaces com protocolo HTTP utilizando o Spring MVC;

- Construir Fachadas de negócio utilizando o Spring Core.

- Construir Componentes para acesso a Banco de Dados com Spring Data.

A quem se destina este treinamento?

Este treinamento se destina à pessoas que possuem conhecimento básico em linguagem de programação Java e querem atuar profissionalmente no mercado de trabalho utilizando Spring.

Principais Módulos do Spring

Spring Core: principal componente do framework - é o responsável por fazer a injeção de dependência através de mecanismos de Service Locator (Padrão de Projeto para identificação de Implementações diversas para uma mesma interface) com a utilização de configurações, que podem ser por arquivo XML (já em desuso) e com anotações Java.

Spring MVC: construído sobre os componentes Servlet para atender a construção de aplicações web com protocolo HTTP. Utiliza o padrão de projetos MVC (model-view-controller).

Spring Data: construído sobre o JDBC e o JPA, utiliza interfaces e templates para abstrair o acesso a um Banco de Dados relacional, minimizando bruscamente a quantidade de código que precisa ser escrito para efetuar operações no SGDB.

Spring Security: promove implementações para segurança de aplicações Spring MVC.

Spring Annotation: anotações utilizadas pelo core para criar os esteriótipos padrões de um projeto.

Spring Configuration: promove diversos mecanismos para resolução de parâmetros configuráveis através de arquivo ou Classes.

Spring Transaction: promove abstrações e implementações para controle de transações.

Spring Batch: construído sobre a especificação JSR-352, promove interfaces e implementações para a construção de Jobs para diversas finalidades. Aplicativos batchs são aqueles que executam diretamente sobre o sistema operacional através de CLI (command line interface).

Existem diversos outros módulos do spring framework que atendem a resolução de problemas específicos. Para mais detalhes sobre cada um deles, consultar o link https://spring.io/projects.

É alvo desse treinamento os módulos **Spring Core, Spring MVC, Spring Data.**

Para entendimento desses módulos serão abrangidos os módulos **Spring Annotation, Spring Configuration e Spring Transaction.**

É esperado que após o treinamento, o aluno seja capaz de compreender de forma independente qualquer outro módulo do spring após algum estudo, o que pode variar dependendo da complexidade de cada módulo. A média quatro horas serão suficientes para a compreensão básica da maioria dos módulos.

Principais Esteriótipos do Spring

A criação de esteriótipos no spring, como já vimos é feita através de anotações Java.

São elas:

@Bean

Essa anotação indica a presença de um método que efetua a construção de uma Classe que deve ser gerenciada pelo Contexto do Spring e esta é injetável.

@Component

Essa anotação indica que a classe é um componente injetável do core da sua aplicação.

@Service

Essa anotação indica que a classe é um serviço, ou seja, possui regras e fluxos encapsulados conforme o modelo de negócio.

@Repository

Essa anotação indica que a classe é um repositório, ou seja, faz acesso a um banco de dados e efetua operações no SGDB.

@Configuration

Essa anotação indica a presença de métodos para construção de beans do Spring (@Bean).

@Controller

Essa anotação indica que a classe é um controlador de requisições HTTP. Através dela, são disponibilizados as implementações que atendem a chamadas de métodos HTTP, como GET, POST, PUT, DELETE entre outros.

@RestController

Essa anotação indica que a classe é uma API Rest. Os contratos de dados e as operações disponibilizadas seguem as convenções RestFul e são implementadas como um @Controller.

@ControllerAdvice

Essa anotação indica que a classe é um interceptador de notificações das requisições HTTP e, através dela, é possível capturar exceções que ocorreram e tratar cada uma delas de forma adequada.

Spring Core

O spring core é o principal componente do spring framework e é ele quem faz a injeção de dependência através de mecanismos de Service Locator.

O rastreamento de componentes candidatos do spring carrega no contâiner (spring framework) gerenciável as classes que são anotadas como algum tipo de esteriótipo.

No momento de inicialização de uma aplicação Spring, o contâiner irá tentar eleger um candidato de acordo com os esteriótipos para resolver a implementação de uma interface injetável.

Mas o que são interfaces injetáveis?

São aquelas que foram escolhidas como elegíveis durante o processo de desenho da solução como possíveis alvos de fortes acoplamentos. Ou seja, são os beans estereotipados conforme anotações vistas anteriormente. Para eleger um Bean o spring por convenção, deve-se fazer pelo nome do Bean e sua respectiva inteface.

A injeção de dependência tem como alvo a Anotação @Autowired.

Como exemplo, temos uma implementação sem interface declarada:

```
@Service
public class MeuServico{

        @Autowired
        private RegraA regraA;
        …
}

@Component
public class  RegraA {
        ...
}
ou
@Configuration
public class MinhaConfiguracao{

        @Bean
        public RegraA regraA(){
                return new RegraA();
        }
}
```

Quando temos uma implementação e sua interface é declarada, não é necessário nomear o candidato, por exemplo:

```
@Service
public class MeuServico{

        @Autowired
        private RegraA regraA;

        ...
}

@Component
public class  RegraAImp implemets RegraA {
        ...
}

public interface RegraA{
        ...
}
```

ou

```
@Configuration
public class MinhaConfiguracao{

        @Bean
        public RegraA regraA(){
                return new RegraAImp();
        }

        ...
}
```

Quando temos duas ou mais implementações para uma mesma interface, é necessário nomear o candidato, por exemplo:

```
@Service
public class MeuServico{

	@Autowired
	@Qualifier(“regraAImpY”)
	private RegraA regraA;

	...
}

@Component(“regraAImpX”)
public class  RegraAImpX implements RegraA {
	...
}

@Component(“regraAImpY”)
public class  RegraAImpY implements RegraA{
	...
}

public interface RegraA{
	...
}
```

ou

```
@Configuration
public class MinhaConfiguracao{

	@Bean(“regraAImpX”)
	public RegraA regraA(){
		return new RegraAImpX();
	}

	@Bean(“regraAImpY”)
	public RegraA regraA(){
		return new RegraAImpY();
	}
...
}
```

Definindo o @Scope de um bean Spring

O escopo de um bean define como o spring framework deve gerenciar e construir um objeto.

Utilizando escopos por nome, teremos a definição sobre como o spring framework deve gerenciar os objetos. Os escopos por nome:

@Scope(name="singleton") ou @Scope("singleton")

O contêiner garante uma única instância de objeto a ser injetada para a classe estereotipada. Ou seja, todos os alvos (@Autowired) receberão o mesmo objeto quando forem afetados pela injeção de dependência. Quando não declararmos a propriedade name do @Scope, o spring irá utilizar singleton por padrão.

@Scope(name="prototype") ou @Scope("prototype")

O contêiner irá criar uma nova instância de objeto a ser injetada para a classe estereotipada. Ou seja, cada alvo (@Autowired) recebe uma nova instância quando afetado pela injeção de dependência.

@Scope(name="request") ou @Scope("request")

O contêiner irá criar uma nova instância de objeto a ser injetada para a classe estereotipada a cada request existente dentro do contexto.

@Scope(name="session") ou @Scope("session")

O contêiner irá criar uma nova instância de objeto a ser injetada para a classe estereotipada e o manterá disponível durante o tempo de vida de uma session existente dentro do contexto.

É importante notar que quando temos um **singleton**, qualquer campo desse objeto quando modificado terá seu valor alterado para todos que possuem sua dependência.

Note que os escopos **request e session** existem apenas para contextos web.

Tipo de Proxy do @Scope

A anotação @Scope possui uma propriedade chamada proxyMode. Esta anotação especificará ao contêiner como o spring framework deverá construir o objeto, são eles:

@Scope(proxyMode=ScopedProxyMode.NO)

O spring não utilizará um proxy para a construção do bean estereotipado.

@Scope(proxyMode=ScopedProxyMode.DEFAULT)

Para a maioria dos cenários, ele se comporta como proxyMode=ScopedProxy.NO. Quando não declaramos nenhuma valor para a propriedade, o spring utilizará esse modo como padrão.

@Scope(proxyMode=ScopedProxyMode.TARGET_CLASS)

Para esse modo o Spring irá utilizar recursos da CGLIB para criação do Bean. Detalhes sobre este modo estão descritos no link: https://github.com/cglib/cglib/wiki/How-To

@Scope(proxyMode=ScopedProxyMode.INTERFACES)

Para esse modo, o spring irá criar dynamic proxys da JDK. Detalhes sobre este modo estão descritos no link: https://docs.oracle.com/javase/8/docs/technotes/guides/reflection/proxy.html

Note que as duas propriedades, name e proxyMode podem ser combinadas entre si.

Desafio

Dado um número de colunas e um número de linhas, retornar uma matriz em espiral de fora para dentro no sentido horário.

Este problema é mais fácil de ser compreendido através de exemplos:

Entrada: 3 4

Saída:

```
 1  2 3
10 11 4
 9 12 5
 8  7 6
```

Entrada: 5 6

Saída:

```
 1  2  3  4  5
18 19 20 21  6
17 28 29 22  7
16 27 30 23  8
15 26 25 24  9
14 13 12 11 10
```

Spring MVC

É o módulo mais utilizado do spring framework pelo mercado na atualidade. Com o avanço da internet e a proliferação de APIs para integrações B2B, B2C e P2P, a popularidade desse módulo alavancou o framework como um todo.

O Spring MVC foi construído para facilitar acessos aos métodos HTTP, que são providos no Java por Servlets.

Servlets são burocráticos e complicados, exigem uma programação imperativa e fugir de fortes acoplamentos é oneroso.

Para disponibilizar acesso aos métodos HTTP de um servidor de aplicações (ou contêiner no jargão técnico) o spring framework disponibiliza anotações java para métodos que serão expostos. São elas:

@Controller

Essa anotação indica que a classe é um controlador de requisições HTTP. Através dela são disponibilizados as implementações que atendem a chamadas de métodos HTTP, como: GET, POST, PUT, DELETE, entre outros.

@RestController

Essa anotação indica que a classe é uma API Rest. Os contratos de dados e as operações disponibilizadas seguem as convenções RestFul e são implementadas como um @Controller.

Para expor métdos Java em métdos HTTP temos as seguintes Anotações:

@GetMapping

Essa anotação expõem um método java, tornando-a acessível através de uma requisição HTTP com método GET.

@PostMapping

Essa anotação expõem um método java, tornando-a acessível através de uma requisição HTTP com método POST.

@PutMapping

Essa anotação expõem um método java, tornando-a acessível através de uma requisição HTTP com método PUT.

@DeleteMapping

Essa anotação expõem um método java, tornando-a acessível através de uma requisição HTTP com método DELETE.

@PatchMapping

Essa anotação expõem um método java, tornando-a acessível através de uma requisição HTTP com método PATCH.

Todas as anotações acima possuem as propriedades:

- **Path** – indica qual URL que esse método deve atender;
- **Params** – indica quais parâmetros HTTP esse método possui;
- **Headers** – indica quais elementos do cabeçalho HTTP serão utilizados;
- **Consumes** – indica qual o formato consumido no método, ou seja qual MediaType deve ser utilizado para efetuar o Bind dos dados recebidos;
- **Produces** – indica qual o formato fornecido no método, ou seja, qual MediaType deve ser utilizado para efetuar o Parse dos dados fornecidos.

@RequestBody

Essa anotação indica que o conteúdo recebido pelo método está contido no corpo do protocolo HTTP.

@ResponseBody

Essa anotação indica que o conteúdo fornecido pelo método está contido no corpo do protocolo HTTP.

@ResponseStatus

Indica qual o statuscode do protocolo HTTP é esperado que esse método responda.

Desafio

Vamos implementar uma API Restful para controle de acesso de usuários.

Essa API deve conter as seguintes operações:

- Cadastrar usuário:

  Para isso, iremos ter uma método POST, que atende no caminho /usuario e recebe o seguinte payload em formato Json:

  {

  “username”:”string”,

  “password”:”string”,

  “confirmPassword”:”string”

  “email”:”string”

  }

  Essa API deve responder:

  201 CREATE quando tiver sucesso.

  400 BAD REQUEST quando password é diferente de confirmPassword

  400 BAD REQUEST quando email não é válido

  409 CONFLIT quando usename ou email já existem.

- Verificar usuário:

  Para isso, iremos ter uma método POST, que atende no caminho /usuario/verificar e recebe o seguinte payload em formato Json:

  {

  “usernameOrEmail”:”string”,

  “password”:”string”

  }

  Essa API deve responder:

  200 OK quando tiver sucesso.
  403 ACCESS DENIED quando username ou email e senha não forem de algum usuário já cadastrado.

- Consultar usuário:

  Para isso, iremos ter uma método GET, que atende no caminho /usuario e recebe o seguinte payload em formato Json:

  {“usernameOrEmail”:”string”}

  Essa API deve responder:

  200 OK quando tiver sucesso.

  E no corpo, deve conter os dados do usuário:

  {“username”:”string”,“email”:”string”}

- Listar usuários:

  Para isso, iremos ter uma método GET, que atende no caminho /usuario/listar.

  Essa API deve responder:

  200 OK quando tiver sucesso.

  E no corpo, deve conter os dados do usuário:

  [{

  “username”:”string”,

  “email”:”string”

  },

  {

  “username”:”string”,

  “email”:”string”

  }

  ….

  ]

Spring Data

Em conjunto com Spring MVC, o Spring Framework torna-se incrivelmente ágil. O que antes com JDBC e Hibernate era extremamente demorado de se fazer, com muitas chances de cometer pequenos erros que levavam muito tempo para se perceber, o Spring Data tornou essa tarefa divertida e simples.

Com a interface JpaRepository é possível implementar de forma elegante a maioria das operações executadas em Banco de Dados relacional.

Baseada em convenções, o spring data em conjunto com hibernate e jdbc transformam nome de métodos em comandos SQL, fazendo isso com expressões bastante intuitivas e de fácil memorização.

Além disso, já possui implementações para boa parte das operações normalmente realizadas.
Se isso não é suficiente, ele ainda possui mecanismos para que você declare exatamente a operação que deseja efetuar.

Segue abaixo um exemplo de como escrever comandos SQL com o Spring Data e a interface JpaRepository:

```
interface PersonRepository extends Repository<User, Long> {

    List<Person> findByEmailAddressAndLastname(EmailAddress emailAddress, String lastname);

   List<Person> findDistinctPeopleByLastnameOrFirstname(String lastname, String firstname);

   List<Person> findPeopleDistinctByLastnameOrFirstname(String lastname, String firstname);

  List<Person> findByLastnameIgnoreCase(String lastname);

   List<Person> findByLastnameAndFirstnameAllIgnoreCase(String lastname, String firstname);

  List<Person> findByLastnameOrderByFirstnameAsc(String lastname);

  List<Person> findByLastnameOrderByFirstnameDesc(String lastname);

}
```

À partir dessas interfaces, o Spring Data sabe criar exatamente as queries que precisam ser executadas no SGDB relacional. Não importa qual o Banco de Dados que está sendo utilizado, pois com o trabalho em conjunto com JDBC e JPA, esse problema é resolvido.

Além disso, é possível customizar de forma bem simples e ágil comportamentos bastante elaborados, como por exemplo, paginações em consultas, como podemos observar no exemplo abaixo:

```
Page<User> findByLastname(String lastname, Pageable pageable);

Slice<User> findByLastname(String lastname, Pageable pageable);

List<User> findByLastname(String lastname, Sort sort);

List<User> findByLastname(String lastname, Pageable pageable);
```

E se isso ainda não atende a query que é preciso executar, declare sua própria query:

```
@Query("select u from User u")
List<User> findAllByCustomQuery();

@Query("select u from User u")
List<User> findllPaged(Pageable pageable);
```

Desafio

Vamos reescrever a API do desafio de Spring Mvc para utilizar Spring Data.

Projeto E-commerce (com metodologia Ágil)

Mãos à massa!!!

O Projeto deve ser realizado em times de 03 à 05 pessoas.

Nesse projeto teremos os seguintes papéis:

PO → O Instrutor do treinamento. Deve fornecer todas as informações que o time precisa para efetuar suas tarefas, sem deixar dúvidas ou lacunas no que deve ser entregue. O PO deve dizer o que ele quer, jamais o como deve ser feito.

SM → Um membro eleito do time, o SM deve levantar com o time todos os impedimentos e dúvidas que o time possui e buscar saná-las o mais rapidamente possível. Caso a dificuldade não possa ser resolvida, o SM deve buscar alternativas juntamente com o PO para a entrega ou não da estória.

Testador → Um membro com afinidade para essa atividade. As atividades de testes podem ser divididas entre o time, mas é importante que haja um responsável. O SM é quem deve direcionar as tarefas para o time.

O Testador deve certificar-se de que as estórias entregues cumprem os requisitos desejados e não possuem defeitos que impedem ou comprometam a utilização do sistema.

Desenvolvedor → 01 até 03 membros com afinidade para essa atividade. O desenvolvedor é o responsável por codificar as estórias na linguagem combinada, escrever os testes unitários e garantir requisitos não funcionais, tais como performance, disponibilidade, escalabilidade, segurança, resiliência, entre outros.

Arquiteto → O Instrutor do treinamento. O arquiteto deve auxiliar o time de desenvolvimento com relação as tecnologias escolhidas para o projeto, determinar como os componentes do projeto serão integrados e fazer inspeção de código.

Deverão ser feitas duas reuniões de time por dia, uma no início do dia e outra após o horário de almoço, as reuniões não devem levar mais do que 5 min.

É esperado que cada membro do time diga com precisão:

- o que foi feito, quanto tempo levou, qual a dificuldade.
- o que está fazendo e mais quanto tempo irá levar.
- o que será feito em seguida.
- se existe algum impedimento, qual o motivo.
- teve ajuda de alguém?
- ajudou alguém?
- aprendizagem com a tarefa.

Estórias do Projeto:

PEC-01

Eu como usuário gostaria de visualizar os produtos disponíveis na loja para venda assim que eu entro no site.

- Quero poder colocar um produto no carrinho de compras.
- Quero saber o quanto custa um produto.
- Quero poder dizer a quantidade de itens que desejo desse produto.
- Quero saber o prazo de entrega.

PEC-02

Eu como usuário após escolher todos os produtos que desejo, quero finalizar a compra.

- Caso eu não tenha um cadastro, quero fazer nesse momento.
- Caso não tenha informado forma de pagamento e/ou endereço, quero fazer nesse momento.
- Quero receber um e-mail informando que uma compra foi efetuada.
- Quero poder acompanhar o pedido.

PEC-03

Eu como usuário quero poder cancelar um produto após efetivar a compra.

- Caso o produto já tenha saído para entrega, me comprometo a enviar o produto por sedex e o rastreio do correio pela forma escolhida pela loja virtual.
- Quero receber o valor pago de volta.

PEC-04

Eu como vendedor preciso cadastrar os produtos que tenho disponíveis para venda através de uma arquivo csv.

- Caso o produto já exista, suas informações devem ser atualizadas.
- O produtos possuem os seguintes campos:
  - código do produto, ele é único para cada produto
  - nome
  - preço
  - caminho da imagem

PEC-05

Eu como vendedor preciso ser avisado quando um produto é vendido para poder iniciar o processo de separação e entrega do mesmo.

PEC-06

Eu como analista de segurança preciso ter garantia de que quem comprou um produto é realmente ele mesmo.

Macro atividades do projeto

O projeto deverá durar 3 dias.

Dia 1 (planejamento e desenvolvimento):

- divisão das estórias entre os times. (tempo previsto de 10 min)
- refinamento técnico da estórias. (tempo previsto 1h e 55 min)
- definição de arquitetura. (tempo previsto 1h e 55 min)
- reunião para divisão das atividades (5 min)
- desenvolvimento das atividades. (3h e 55 min)

Dia 2 (desenvolvimento):

- reunião de status report. (5 min)
- desenvolvimento das atividades. (3h e 55 min)
- intervalo para almoço. (1 hora)
- reunião de status report. (5 min)
- desenvolvimento das atividades. (3h e 55 min)

Dia 3 (desenvolvimento e testes):

- reunião de status report. (5 min)
- desenvolvimento das atividades. (3h e 55 min)
- intervalo para almoço. (1 hora)
- reunião de status report. (5 min)
- homologação do projeto. (3h)
- discussão sobre os temas abordados, feedbacks e aprendizado.