DESENVOLVIMENTO DE SOFTWARE PARA A WEB 1

Módulo 8 SpringMVC, Thymeleaf & Spring Data JPA

Delano Medeiros Beder delano@dc.ufscar.br

Injeção de Dependências (DI)

Inversão de Controle (IoC)





Injeção de dependências

- Injeção de dependências (Dependency Injection, em inglês) auxilia-nos a manter nossas classes tão independentes quanto possível
 - Padrão de projeto usado para evitar o alto nível de acoplamento de código dentro de uma aplicação.
 - Aumenta o reuso por proporcionar o baixo acoplamento
 - Aumento na facilidade de manutenção
 - Uma dependência é simplesmente um objeto que a sua classe precisa para funcionar plenamente.

Relação entre DI e loC

- Injeção de dependência é um padrão de desenvolvimento de programas de computadores utilizado quando é necessário manter baixo o nível de acoplamento entre diferentes módulos de um sistema.
 - As dependências entre os módulos não são definidas programaticamente, mas sim pela configuração de uma infraestrutura de software (container) que é responsável por "injetar" em cada componente suas dependências declaradas. A Injeção de dependência se relaciona com o padrão Inversão de controle mas não pode ser considerada um sinônimo deste.
- Injeção de dependência é uma das duas maneiras de implementar a inversão de controle (Inversion of Control, em inglês). Inversão de controle é um termo mais amplo, onde a responsabilidade de informar a implementação a ser utilizada deixa de ser da classe, e passa a ser do consumidor da classe.

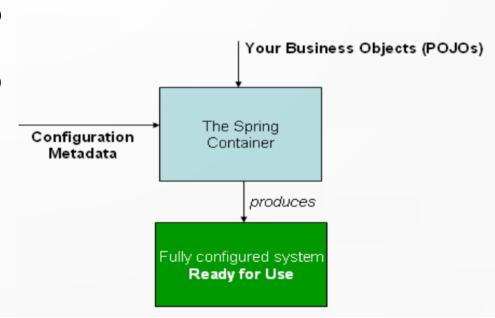
Injeção de Dependências - IoC Container

- O contêiner Spring (loC contêiner) encontra-se no núcleo do Spring Framework.
- O loC contêiner é responsável por criar os objetos, conectá-los, configurá-los e gerenciar seu ciclo de vida completo, desde a criação até a destruição.



Injeção de Dependências - IoC Container

- O contêiner determina quais objetos instanciar, configurar e gerenciar através dos metadados de configuração fornecidos.
- Os metadados de configuração podem ser representados por:
 - Arquivos XML,
 - Anotações Java,
 - Código Java.



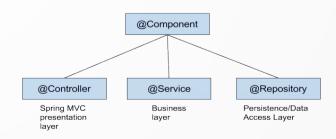
Injeção de Dependências - IoC Container

- No Spring, os objetos que formam a espinha dorsal da sua aplicação e que sejam gerenciados pelo Spring são chamados de beans. Um bean é um objeto que é instanciado, montado e gerenciado pelo Spring IoC container.
- O Spring IoC container busca informações em XML, anotações ou código java sobre como os beans devem ser instanciados, configurados e montados, e como se relacionam com outros beans.
 - A resolução do relacionamento entre beans é conhecida como injeção de dependências.
 - Se você cria uma classe que depende de alguns bean, só precisa se preocupar com o que a sua classe depende, e não com o que suas dependências dependem.

Anotações

Spring MVC





@Component

É uma anotação básica para criar qualquer tipo de *bean* gerenciado pelo **Spring Framework**.

Normalmente usada quando não se define um bean como @Repository ou @Service.

```
package br.ufscar.dc.dsw.conversor;
import org.springframework.beans.factory.annotation.Autowired;
@Component
public class EditoraConversor implements Converter<String, Editora>{
    private IEditoraService service;
    public Editora convert(String text) {
}
```

@Repository

Define um *bean* como sendo do tipo persistente para uso em classes de acesso a banco de dados.

A partir desta anotação o **Spring** pode usar recursos referentes a persistência, como tratar as exceções específicas para este fim.

@Controller

Transforma a classe em um controlador do Spring MVC

```
package br.ufscar.dc.dsw.controller;
import javax.validation.Valid;
@Controller
@RequestMapping("/editoras")
public class EditoraController {
    @Autowired
    private IEditoraService service:
   @GetMapping("/cadastrar")
    public String cadastrar(Editora editora) {
        return "editora/cadastro":
    public String listar(ModelMap model) {
    public String salvar(@Valid Editora editora, BindingResult result, RedirectAttributes attr) {
    public String preEditar(@PathVariable("id") Long id, ModelMap model) {
    public String editar(@Valid Editora editora, BindingResult result, RedirectAttributes attr) {
    public String excluir(@PathVariable("id") Long id, ModelMap model) {
```

@Autowired

Anotação usada para informar ao Spring que ele deve injetar a variável anotada na classe em que está declarada.

```
package br.ufscar.dc.dsw.controller:
import javax.validation.Valid;
@Controller
@RequestMapping("/editoras")
public class EditoraController {
                                                                  Injeção
    @Autowired
    private IEditoraService service;
                                                                  IEditoraService
    @GetMapping("/cadastrar")
    public String cadastrar(Editora editora) {
        return "editora/cadastro":
    public String listar(ModelMap model) {
    public String salvar(@Valid Editora editora, BindingResult result, RedirectAttributes attr) {
    public String preEditar(@PathVariable("id") Long id, ModelMap model) {
    public String editar(@Valid Editora editora, BindingResult result, RedirectAttributes attr) {
    public String excluir(@PathVariable("id") Long id, ModelMap model) {
```

@Service

Usado para classes do tipo serviço (Service Layer), que possuem, por exemplo, regras de negócios.

```
package br.ufscar.dc.dsw.service.impl;
import java.util.List;
@Service
@Transactional(readOnly = false)
public class EditoraService implements IEditoraService {
                                      Injeção
   @Autowired
   IEditoraDAO dao:
                                      IEditoraDAO
   public void salvar(Editora editora) {
   public void excluir(Long id) {
   public Editora buscarPorId(Long id) {
   public List<Editora> buscarTodos() {
   public boolean editoraTemLivros(Long id) {
```

@Autowired

@Autowired pode ser declarada sobre variáveis de instancia, construtores e métodos set() das variáveis de instancia.

@RequestMapping

Usada para mapear URLs de acesso a um controlador e aos métodos contidos nele. Também podemos definir verbos HTTP (POST, GET, ...) de acesso aos métodos.

```
@Controller
@RequestMapping("/editoras")
public class EditoraController {
    @Autowired
    private IEditoraService service:
    @RequestMapping(path = "/cadastrar", method = RequestMethod.GET)
    public String cadastrar(Editora editora) {
        return "editora/cadastro":
    @GetMapping("/listar")
    public String listar(ModelMap model) {
        model.addAttribute("editoras".service.buscarTodos()):
        return "editora/lista":
    public String salvar(@Valid Editora editora, BindingResult result, RedirectAttributes attr) {
    public String preEditar(@PathVariable("id") Long id, ModelMap model) {
    @PostMapping("/editar")
    public String editar(@Valid Editora editora, BindingResult result, RedirectAttributes attr) {
        if (result.hasErrors()) {
            return "editora/cadastro";
        service.salvar(editora);
        attr.addFlashAttribute("sucess", "Editora editada com sucesso.");
        return "redirect:/editoras/listar";
    public String excluir(@PathVariable("id") Long id, ModelMap model) {
```

@GetMapping
@PostMapping
@PutMapping
@DeleteMapping
@PatchMapping

@PathVariable

Tem o objetivo de extrair da URL um parâmetro que foi incluído como path da URL.

http://localhost:8080/editoras/excluir/123

```
@Controller
@RequestMapping("/editoras")
public class EditoraController {
   @Autowired
   private IEditoraService service:
   public String cadastrar(Editora editora) {
   public String listar(ModelMap model) {
   public String salvar(@Valid Editora editora, BindingResult result, RedirectAttributes attr) {
   public String preEditar(@PathVariable("id") Long id, ModelMap model) {
   public String editar(@Valid Editora editora, BindingResult result, RedirectAttributes attr) {
   @GetMapping("/excluir/{id}")
   public String excluir(@PathVariable("id") Long id, ModelMap model) {
       if (service.editoraTemLivros(id)) {
           model.addAttribute("fail", "Editora não excluída. Possui livro(s) vinculado(s).");
       } else {
           service.excluir(id):
           model.addAttribute("sucess", "Editora excluída com sucesso.");
                                    id receberá o valor 123
       return listar(model);
```

@RequestParam

Tem o objetivo capturar um parâmetro de consulta (Query Param) enviado por uma solicitação.

http://localhost:8080/livros/listarPorAno?ano=2020

```
@Controller
@RequestMapping("/livros")
public class LivroController {
   @Autowired
   private ILivroService livroService:
   @Autowired
   private IEditoraService editoraService:
   public String cadastrar(Livro livro) {
   public String listar(ModelMap model) {
   @GetMapping(path = "/listarPorAno")
   public String listarPorAno(@RequestParam(name = "ano") int ano, ModelMap model) {
       // implementação omitida
       return "livro/lista";
                                    ano receberá o valor 2020
   public String salvar(@Valid Livro livro, BindingResult result, RedirectAttributes attr) {□
   public String preEditar(@PathVariable("id") Long id. ModelMap model) {
   public String editar(@Valid Livro livro, BindingResult result, RedirectAttributes attr) {□
   public String excluir(@PathVariable("id") Long id, RedirectAttributes attr) {
   public List<Editora> listaEditoras() {
```

@ModelAttribute

Pode ser usado sobre a assinatura de um método ou como argumento de um método.

```
@Controller
@RequestMapping("/livros")
public class LivroController {
   @Autowired private ILivroService livroService:
    @Autowired private IEditoraService editoraService:
    public String cadastrar(Livro livro) {
    public String listar(ModelMap model) {
    @PostMapping("/salvar")
    public String salvar(@ModelAttribute Livro livro, BindingResult result, RedirectAttributes attr) {
        if (result.hasErrors()) {
            return "livro/cadastro":
        livroService.salvar(livro):
       attr.addFlashAttribute("sucess", "Livro inserido com sucesso");
        return "redirect:/livros/listar";
    public String preEditar(@PathVariable("id") Long id, ModelMap model) {
    public String editar(@Valid Livro livro, BindingResult result, RedirectAttributes attr) {
    public String excluir(@PathVariable("id") Long id, RedirectAttributes attr) {
    @ModelAttribute("editoras")
   public List<Editora> listaEditoras() {
        return editoraService.buscarTodos():
```

@Valid / BindingResults

Responsável por injetar a validação back-end via Hibernate Validator ou Spring Validator.

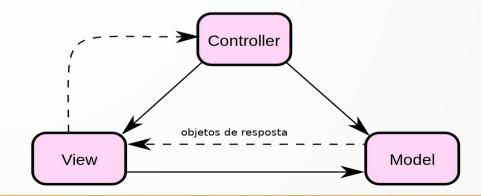
```
@Controller
@RequestMapping("/livros")
public class LivroController {
   @Autowired private ILivroService livroService;
   @Autowired private IEditoraService editoraService;
   public String cadastrar(Livro livro) {
   public String listar(ModelMap model) {
   @PostMapping("/salvar")
   public String salvar(@Valid Livro livro, BindingResult result, RedirectAttributes attr) {
       // Implementação omitida
       return "redirect:/livros/listar";
                                           resultado da validação
                                           erros que podem ter ocorrido
   @PostMapping("/editar")
   public String editar(@Valid Livro livro, BindingResult result, RedirectAttributes attr) {
       // Implementação omitida
       return "redirect:/livros/listar";
   public String preEditar(@PathVariable("id") Long id, ModelMap model) {
   public String excluir(@PathVariable("id") Long id, RedirectAttributes attr) {
   public List<Editora> listaEditoras() {
```

@Configuration / @Bean

Classes de configuração (classe anotada com @Configuration) podem conter métodos de definição de beans (métodos anotados com @Bean)

```
@Configuration
public class MvcConfig implements WebMvcConfigurer {
    public void addViewControllers(ViewControllerRegistry registry) {
    @Bean
    public LocaleResolver localeResolver() {
        SessionLocaleResolver slr = new SessionLocaleResolver():
        slr.setDefaultLocale(new Locale("pt", "BR"));
        return slr;
   @Bean
    public LocaleChangeInterceptor localeChangeInterceptor() {
        LocaleChangeInterceptor lci = new LocaleChangeInterceptor();
        lci.setParamName("lang");
        return lci:
    public void addInterceptors(InterceptorRegistry registry) {
```





ModelMap

Objeto usado para enviar dados a página (visão) como resposta de uma solicitação.

```
@Controller
@RequestMapping("/livros")
public class LivroController {
                                              Dados enviados
   @Autowired
   private ILivroService livroService;
                                                para a visão
   @GetMapping("/listar")
   public String listar(ModelMap model) {
       model.addAttribute("livros", livroService.buscarTodos());
                                    Visão a ser invocada
       return "livro/lista";
      Demais métodos
```

ModelAndView

Objeto usado para enviar dados a página (visão) como resposta de uma solicitação.

```
@Controller
@RequestMapping("/livros")
public class LivroController {
                                       Visão a ser invocada
   @Autowired
   private ILivroService livroService;
   @GetMapping("/listar")
    public ModelAndView listar() {
       ModelAndView model = new ModelAndView("livro/lista");
       model.addObject("livros", livroService.buscarTodos());
       return model;
      Demais métodos
                                            Dados enviados
                                              para a visão
```

ModelAndView & ModelMap

Adicionamos parâmetros de resposta no objeto *ModelMap* e retornamos um *ModelAndView*.

```
@Controller
@RequestMapping("/livros")
public class LivroController {
                                              Dados enviados
   @Autowired
                                                para a visão
    private ILivroService livroService;
   @GetMapping("/listar")
    public ModelAndView listar(ModelMap model) {
       model.addAttribute("livros", livroService.buscarTodos());
       return new ModelAndView("livro/lista", model);
      Demais métodos
                          Visão a ser invocada
```

Redirect

Redirect é uma operação usada para redirecionar a resposta de uma solicitação a outra solicitação.

```
@Controller
@RequestMapping("/livros")
public class LivroController {
    @Autowired
    private ILivroService livroService;
   @GetMapping("/listar")
    public String listar(ModelMap model) {
        model.addAttribute("livros", livroService.buscarTodos());
       return "livro/lista";
                                                                 Envio de atributos
   @PostMapping("/salvar")
    public String salvar(@Valid Livro livro, BindingResult result, RedirectAttributes attr) {
        if (result.hasErrors()) {
           return "livro/cadastro";
        livroService.salvar(livro):
       _attr.addFlashAttribute("sucess", "Livro inserido com sucesso");
        return "redirect:/livros/listar":
    // Demais métodos
```

Thymeleaf





Loop: Tabela (dados dinâmicos)

```
<thead>
    #
      Título
      Autor
      Editora
      Ano
      Preco
      Ação
    </thead>
 <a class="btn btn-info btn-sm"
          th:href="@{/livros/editar/{id} (id=${livro.id}) }" role="button">
          <span class="oi oi-brush" title="Editar" aria-hidden="true"></span></a>
        <button th:id="${#strings.concat('btn livros/excluir/',livro.id)}"</pre>
          type="button" class="btn btn-danger btn-sm" data-toggle="modal" data-target="#myModal">
          <span class="oi oi-circle-x" title="Excluir" aria-hidden="true"></span>
        </button>
```

Condicionais & Loops

```
<th:block th:each="produto:${produtos}">
  <div>
     <span th:if="${produto.pgtoVista()}" th:text="${produto.precoVista}"></span>
     <span th:unless="${produto.pgtoVista()}" th:text="${produto.precoPrazo}"></span>
  </div>
</th:block>
<th:block th:each="produto:${produtos}">
  <div>
     <span th:text="${produto.pgtoVista() ? produto.precoVista :</pre>
     product.precoPrazo}"></span>
  </div>
</th:block>
```

Escopo de Variáveis

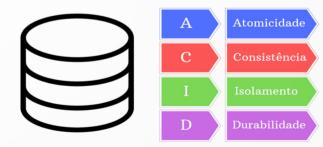
```
<span>
    [[${usuario.nome}]][[${usuario.sobrenome}]] <br/>
    [[${usuario.endereco.logradouro}]] <br/>
    [[${usuario.endereco.numero}]] <br/>
    [[${usuario.endereco.cidade}]], [[${usuario.endereco.estado}]]
</span>
<span th:object="${usuario.endereco}">
    [[${usuario.nome}]] [[${usuario.sobrenome}]] <br/>
    [[*{logradouro}]] <br/>
    [[*{numero}]] <br/>
    [[*{cidade}]], [[*{estado}]]
</span>
```

Escopo de Variáveis

```
<span th:object="${usuario.endereco}>
   [[${usuario.nome}]][[${usuario.sobrenome}]] <br/>
   [[*{logradouro}]] <br/>
   [[*{numero}]] <br/>
   [[*{cidade}]], [[*{estado}]]
</span>
<span th:object="${usuario.endereco}</pre>
      th:with="nomeCompleto=${usuario.nome + ' ' + usuario.sobrenome}">
            [[${nomeCompleto}]] <br/>
            [[*{logradouro}]]<br/>
            [[*{numero}]]<br/>
            [[*{cidade}]], [[*{estado}]]
</span>
```

Gerenciamento de Transações





Gerenciamento de Transações via DAO

```
@Transactional
public class ProfessorDAO implements IProfessorDAO {

    @Override
    public void insert(Professor professor) {
        // Implementação
    }
}
```

```
@Transactional
public class DisciplinaDAO implements IDisciplinaDAO {

    @Override
    public void insert(Disciplina disciplina) {
        // Implementação
    }
}
```

```
public class AlocacaoService implements IAlocacaoService{
    @Autowired
    IProfessorDAO profDAO;

@Autowired
    IDisciplinaDAO discDAO;

@Override
    public void aloca(Professor professor, Disciplina disciplina) {
        profDAO.insert(professor);
        disciplina.setProfessor(professor);
        discDAO.insert(disciplina);
    }
}
2 transações distintas
```

Gerenciamento de Transações via Service

```
public class ProfessorDAO implements IProfessorDAO {
    @Override
    public void insert(Professor professor) {
        // Implementação
    }
}
```

```
public class DisciplinaDAO implements IDisciplinaDAO {
    @Override
    public void insert(Disciplina disciplina) {
        // Implementação
    }
}
```

```
@Transactional
public class AlocacaoService implements IAlocacaoService{
    @Autowired
    IProfessorDAO profDAO;

    @Autowired
    IDisciplinaDAO discDAO;

    @Override
    public void aloca(Professor professor, Disciplina disciplina) {
        profDAO.insert(professor);
        disciplina.setProfessor(professor);
        discDAO.insert(disciplina);
    }
}

    1 única transação
```

FIM