



Construção de Sistemas de Software

Padrões para a Camada de Apresentação
de
Aplicações Web

UI das Aplicações Web

- UI das Aplicações Web depende de **Web Browsers**
- A interação com utilizador realiza-se através de **páginas** apresentadas num *browser*
 - dados, estrutura e aparência
- Páginas construídas à custa de **Templates** que definem o que é fixo e deixam em aberto o que varia (e depende dos dados e do comportamento da aplicação)
 - a parte fixa é escrita em HTML + CSS
 - a parte variável é programada (eg., em Java ou JavaScript)
- Onde é construída a página?
 - *Browser* : **Client-Side Templates**
 - *Servidor Web* : **Server-Side Templates**



Client-side vs Server-side templates

- Num **template** quem executa o código correspondente à parte dinâmica da apresentação?
- Duas alternativas:
 - Servidor Web -> ***Server-side template***
 - Exemplos: ASP, JSP, PHP
 - Browser -> ***Client-side template***
 - JavaScript. Exemplos: AngularJS, React, ...
 - *Single Page Application*
- Em ambos os casos há um **Servidor Web** responsável por servir pedidos relativos à aplicação



Client side vs Server side templates: Eventos

- ***Client Side Template***
 - a aplicação *JavaScript* apanha o evento
 - valida e guarda a informação
 - acede à camada de negócio, por exemplo através de uma API REST, caso em que são gerados pedidos HTTP
 - *JavaScript* usa a resposta para alterar o DOM (subordinado aos *templates* existentes)
- ***Server Side Template***
 - ...

Client side vs Server side templates: Eventos

- ***Client Side Template***
 - ...
- ***Server Side Template***
 - Eventos traduzidos por pedidos HTTP para o servidor Web
 - Um pedido *GET* a solicitar os dados do cliente
 - Um pedido *POST* a solicitar a criação de um novo cliente
 - *Controller* do lado do servidor que
 - recebe o pedido,
 - valida a informação (mas não a lógica de negócio),
 - converte os tipos de dados
 - despacha o pedido para a camada de negócio
 - dados recebidos + *template* usados para construir a resposta

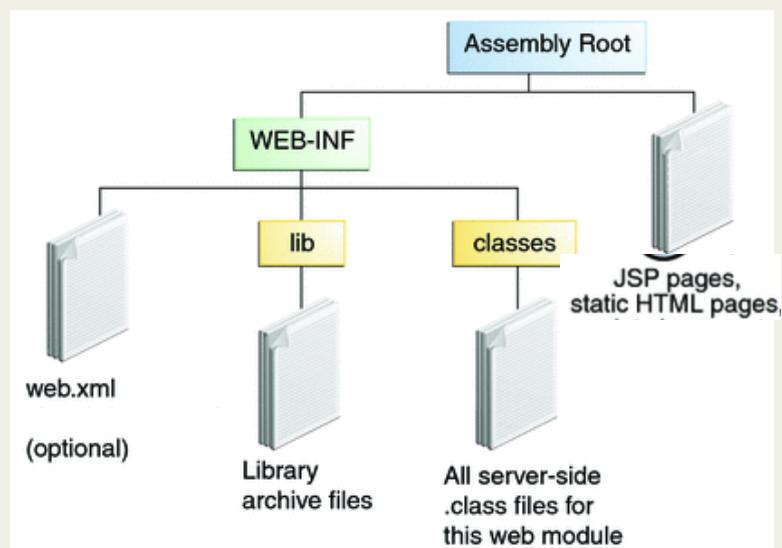
Aplicações Web Java

- Aplicações **programadas** recorrendo a um conjunto de APIs, nomeadamente
 - Java Servlet
 - Java Server Pages (JSP)
 - Java Expression Language (EL)
 - JSP Standard Tag Language (JSTL)
- Aplicações **executadas** recorrendo a um **Servidor** que fornece implementações destas APIs, eg. um **Java HTTP Web Server**
 - Apache Tomcat
 - Catalina – JVM
 - Coyote – trata dos pedidos e respostas HTTP
 - Jasper – compila o JSP
 - Eclipse Jetty
 - Undertow (antes JBoss, agora RedHat)
 - Wildfly (JBoss/RedHat)

Aplicação Web Java

- Estas aplicações são constituídas por
 - Páginas estáticas HTML
 - *Server templates JSP*
 - Classes Java, nomeadamente *servlets* (.class)
 - Classes Java e outros recursos empacotados no formato **java archive**, em ficheiros com a extensão **.jar**
- Estas aplicações são instaladas num servidor, responsável pela sua execução
 - Para isso, precisam de ter os seus elementos empacotados no **formato web archive**, em ficheiros com a extensão **.war**
 - Tem de se fornecer alguma informação específica para o ato de instalação, através de um ficheiro **deployment descriptor** (*web.xml*) ou de anotações

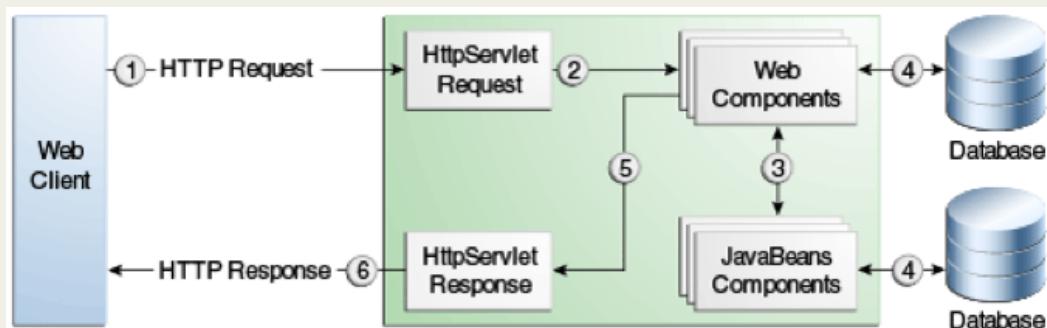
Aplicação Web Java: Organização do war



Web Container (Servlet Container)

- Pode ser visto como uma extensão da JVM
- É o interface entre as **Componentes Web** e o Servidor Web
 - em particular com objetos *HttpServlet*
- Este *container*:
 - Efetua a gestão do ciclo de vida das componentes da aplicação
 - Entrega os pedidos HTTP às componentes, fazendo o mapeamento dos URLs nos servlets
 - Serve de interface para a informação de contexto, tal como informação sobre o pedido ou sobre a sessão

Web Container: Pedidos/Respostas



Servlets

- Uma Aplicação Web é um caso particular de uma aplicação cliente/servidor
 - O cliente, o *browser*, efetua pedidos
 - O servidor, o servidor *Web*, aceita estes pedidos e responde
- Os **Servlets** são um mecanismo utilizado pelo Java para implementar aplicações cliente/servidor
 - classes Java que estendem as capacidades de um servidor
- No contexto das aplicações web vamos usar um tipo específico de Servlets, os **HTTP Servlets**
 - que servem para responder a **pedidos HTTP**
 - estendem a classe **HttpServletRequest**



HttpServlet

- Classe abstracta que implementa `javax.servlet.Servlet`
 - Processa pedidos e produz respostas, normalmente no formato HTML, utilizando o protocolo HTTP
 - O método **service** analisa o pedido HTTP e invoca o método **doGet**, **doPost**, **doPut** ou **doDelete** em função do tipo de pedido Http recebido
- A definição de um *servlet* específico para uma página (ou site) envolve criar uma classe que estende esta classe e implementar os métodos **do***
- As anotações **@WebServlet** em cada *servlet* permitem informar o *web container* a que componente web deve entregar cada pedido



@WebServlet

- Anotações são processadas pelo *container* quando é feita a instalação da aplicação web

```
@WebServlet(  
    name = "MyOwnServlet",  
    urlPatterns = {"/path1", "/path2"}  
)  
public class MyServlet extends HttpServlet{...}  
  
@WebServlet("/processForm")  
@WebServlet("/path/*")
```

```
<servlet-mapping>  
    <servlet-name>redteam</servlet-name>  
    <url-pattern>/red/*</url-pattern>  
</servlet-mapping>
```

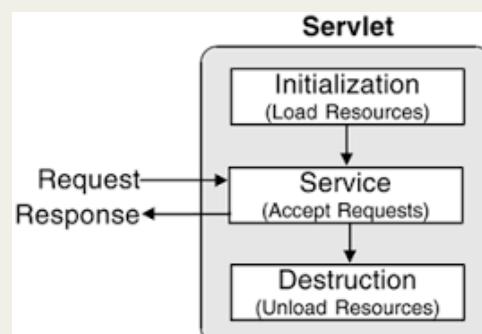
```
<servlet>  
    <servlet-name>redteam</servlet-name>  
    <servlet-class>mysite.server.TeamServlet</servlet-class>
```



Ciências
ULisboa | Informáti

Ciclo de vida de uma *Servlet*

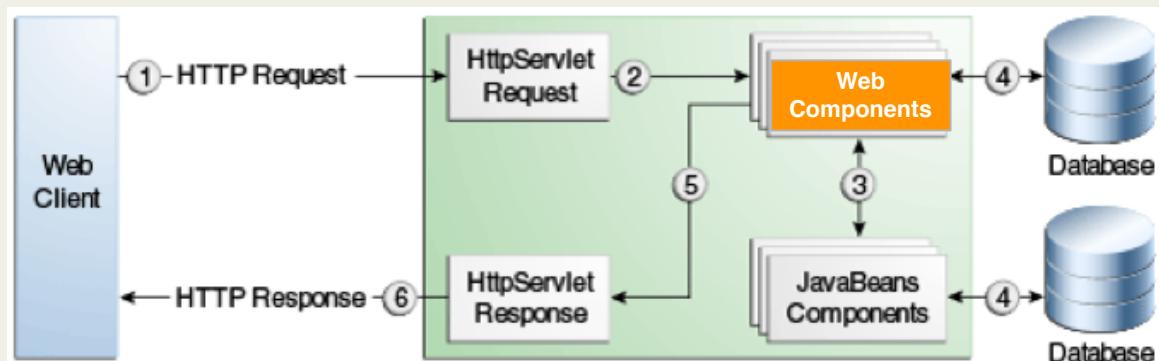
- O *container* cria a *Servlet* e chama o seu método **init()**
- A cada pedido, o *container* chama o método **service()**
 - cada pedido é servido numa *thread* diferente
 - existe um só objeto, mas várias *thread* ativas
 - é preciso ter cuidado com as *race conditions* sobre os atributos da *Servlet*
- O *container* chama o método **destroy()** quando pretende desativar a *Servlet*
- O *container* lança eventos para propagar este tipo de informação (*init*, *destroy*, etc), à qual é possível reagir através da implementação de *listeners*



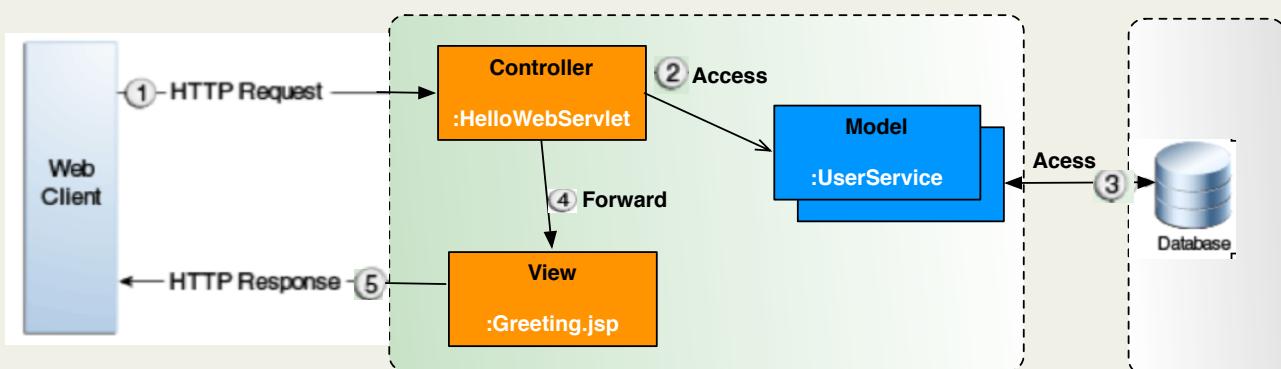
Papel do *Web Container* no tratamento do pedido

- Recebe o pedido HTTP
- Cria um objeto Java *HttpRequest* e preenche-o com a informação do pedido HTTP
 - Existe um **novo objeto HttpRequest por cada pedido**
- A partir do URL determina qual a *Servlet* a que deve ser entregue o pedido
- Cria uma nova *thread* para executar a chamada ao método **service** desta *Servlet*
 - Há um **único objeto por Servlet durante a vida da aplicação**
- Chama o método **service** e passa-lhe o objeto *HttpRequest*

Padrão MVC nas Aplicações Web Java



Model-View-Controller nas Aplicações Web Java



- **View**: Lida com **output**, que dados são mostrados, com que estrutura e aparência
- **Controller**: Lida com **input**, determina reação aos estímulos, fazendo validações, as mudanças necessárias no *model* e decidir o que acontece a seguir
- **Model**: Dá acesso ao **estado** da aplicação e funcionalidades

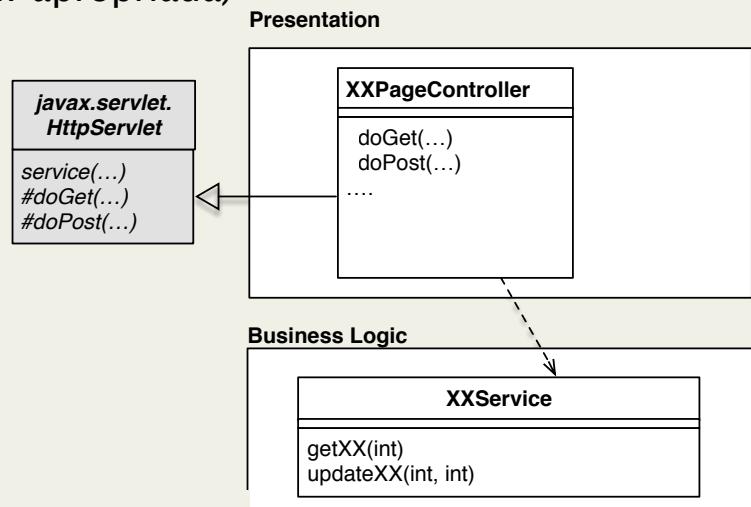
Padrões para Controller nas Aplicações Web

- É um **Controller** que lida com **input** e por isso é chamado **Input Controller**
- **Page Controller**
 - Um objeto que trata dos pedidos para uma página específica ou ação específica do *web site*
 - Existe um controlador para cada página lógica do *web site*
- **Front Controller**
 - Um objeto que trata de todos os pedidos de um *web site*
 - Todos os pedidos são canalizados para este objeto que despacha cada pedido para um objeto *action* apropriado (seguindo o padrão *command*)

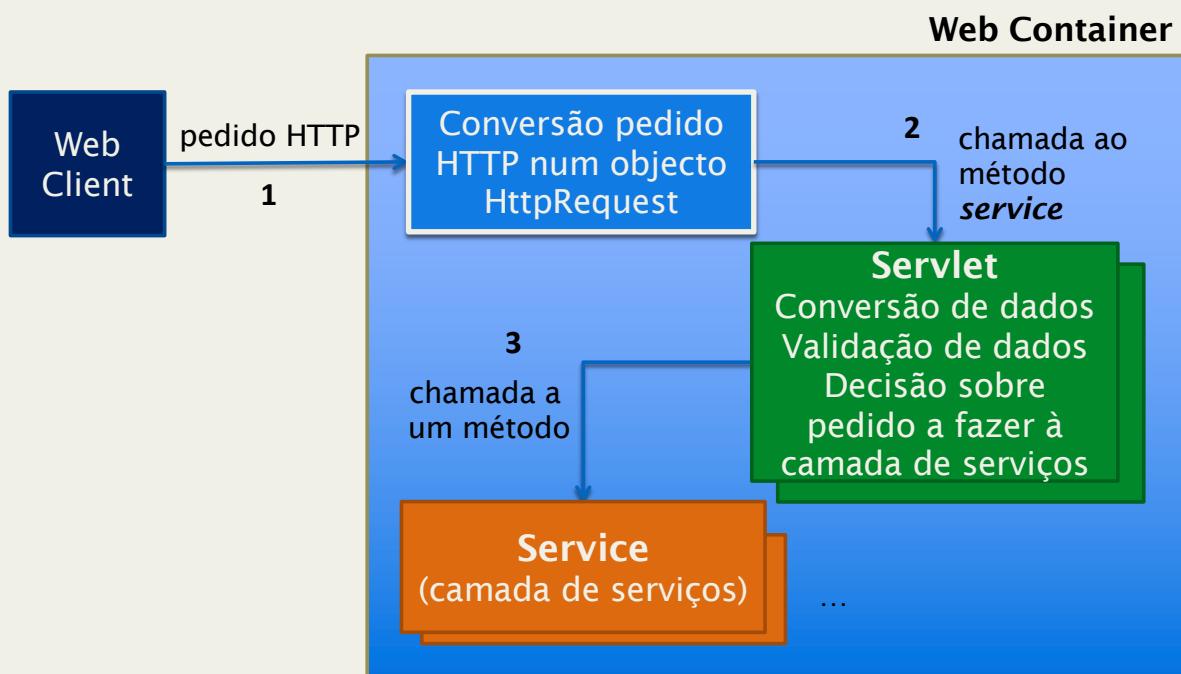


Page Controller

- Pode ser concretizado com **um servlet por página** lógica do *web site*
- No contexto de uma arquitetura em camadas com uma camada de serviços, estes controladores usam esses serviços para tratar dos pedidos e decidir como responder (por exemplo, selecionam a *view* apropriada)



Controller: Processamento dos pedidos



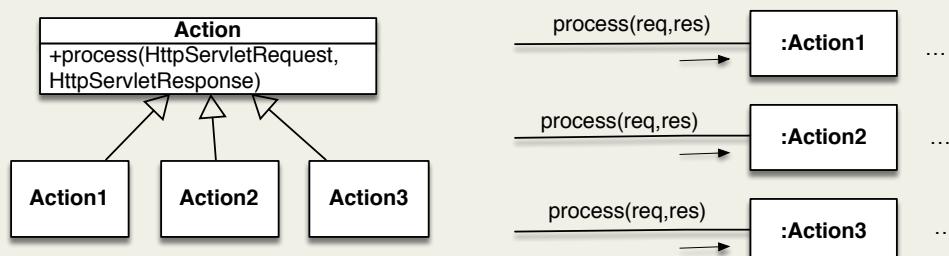
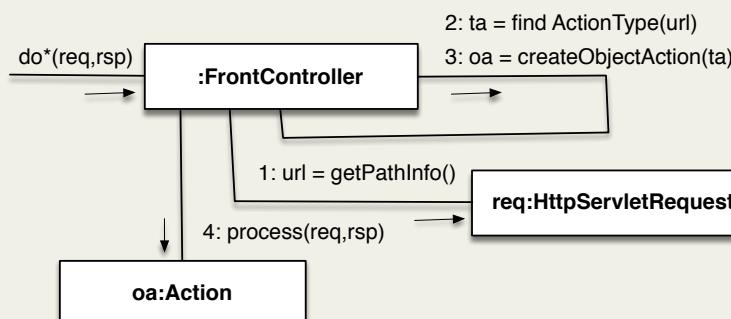
Front Controller: Utilização do padrão GoF Command

- Neste caso existe um único objeto *controller* que tem o controlo sobre como são tratados todos os pedidos de um *site*
- Pode ser concretizado
 - com um *servlet* para quem são encaminhados todos os pedidos do site (`@WebServlet(PATH+“/*”)`)
 - recorrendo ao padrão **Command**
 - um comando por cada tipo de pedido
 - um objeto que sabe executar cada tipo de comando, i.e., como tratar cada tipo de pedido
 - o **frontcontroller** identifica o tipo de cada pedido que recebe e despacha-o para o objeto que sabe tratar esse tipo de pedido específico
 - com o *URL wiring* baseado em carregamento dinâmico de classes



`appRoot/action/vendas/novaVenda`
`controller.web.input.actions.NewSaleAction`

Front Controller: Utilização do padrão Command (GoF)



Padrões para *Controller* nas Aplicações Web

- **Page Controller**

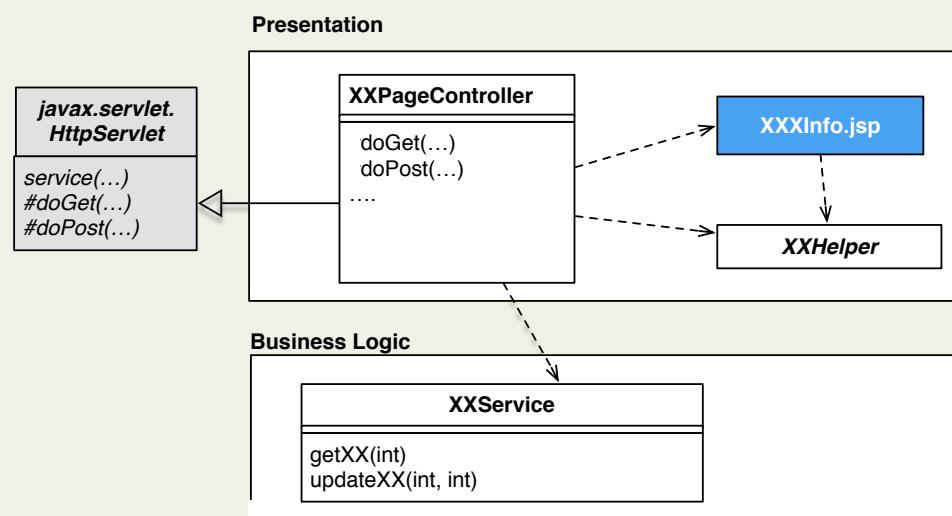
- A solução de desenho é mais simples
- Apropriado quando a lógica de controlo é simples

- **Front Controller**

- A solução de desenho é mais complicada
- Apenas um controlador tem de ser registado no *web server*
- As ações são criadas sempre de novo para cada pedido pelo que não vão ser partilhadas por *threads* concorrentes (é ainda assim preciso ter atenção aos objetos que manipulam, se estes forem partilhados)

Controller -> View:

- Recorrendo à utilização de classes *Model/Helper*, que lidam com a maior parte da lógica ou simplesmente agregam os dados a apresentar na view (*ViewModel*)

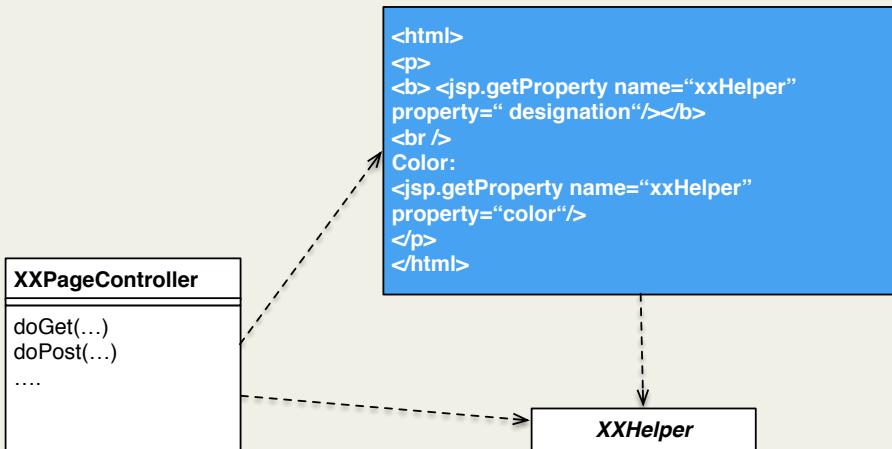


Template View

- O acesso à informação nos objetos *helpers* pode ser concretizada recorrendo ao **Template View**

Renders information into HTML by embedding markers in an HTML page

e tirando partido dos diferentes **âmbitos** existentes



Âmbitos numa Aplicação Web

- Há informação disponível na forma de uma **mapa** de *String* para *Object*
 - nome do atributo -> valor do atributo*em diferentes **âmbitos (scopes)** de uma aplicação web
 - Aplicação**
 - Informação disponível a todas as sessões
 - Sessão**
 - Informação disponível entre pedidos da mesma sessão
 - Pedido**
 - Informação disponível para um pedido
- É possível adicionar/obter informação com *setAttribute/getAttribute*

Âmbitos numa Aplicação Web: Exemplos

- no âmbito da aplicação pode ser colocada informação sobre o objeto inicial do domínio

```
SaleSys app = new SaleSys();
try {
    app.run();
} catch (ApplicationException e) {
    e.printStackTrace();
}
event.getServletContext().setAttribute("app", app);
```

- no âmbito do pedido pode ser colocada informação sobre objetos que são úteis na view selecionada (*helpers/view models*)

```
NewCustomerModel model = createModel(request, app);
request.setAttribute("model", model);
```



Exemplo

The screenshot shows a web browser window with the URL <http://localhost:8080/domain-model-jpa-web-v0/action/clientes/novoCliente>. The page title is "Adicionar Cliente". There are four input fields: "Designação" (Client 1), "Número de pessoa colectiva" (168027852), "Telefone" (12345), and "Tipo de desconto" (Percentagem do Total (acima de limiar)). A red circle highlights the "Criar Cliente" button at the bottom right of the form.



Exemplo

Adicionar Cliente

Designação:

Número de pessoa colectiva:

Telefone:

Tipo de desconto:

Adicionar Cliente

Designação:

Número de pessoa colectiva:

Telefone:

Tipo de desconto:

Mensagens

- Cliente criado com sucesso.

Ciências
ULisboa | Informática

Exemplo

Adicionar Cliente

Designação:

Número de pessoa colectiva:

Telefone:

Tipo de desconto:

Mensagens

- É obrigatório preencher o número de pessoa colectiva
- Erro ao validar cliente

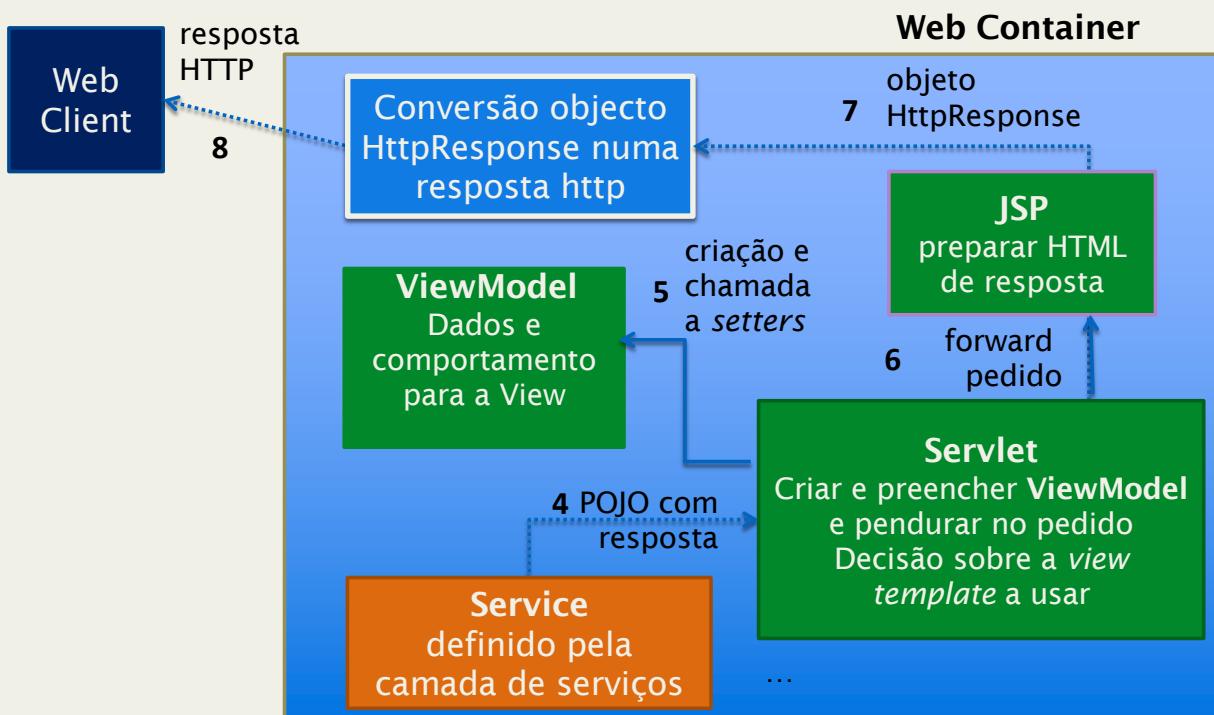
Ciências
ULisboa | Informática

Âmbitos numa Aplicação Web: Exemplos

```
public class CreateCustomerAction extends Action {  
  
    @Override  
    public void process(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        SaleSys app = (SaleSys) request.getServletContext().getAttribute("app");  
        CustomerService addCustomerService = app.getCustomerService();  
  
        NewCustomerModel model = createModel(request, app);  
        request.setAttribute("model", model);  
  
        if (isValid(model)) {  
            try {  
                addCustomerService.addCustomer(intValue(model.getVATNumber()), model.getDesignation(),  
                    intValue(model.getPhoneNumber()), intValue(model.getDiscountType()));  
                model.clearFields();  
                model.addMessage("Cliente criado com sucesso.");  
            } catch (ApplicationException e) {  
                model.addMessage("Erro ao criar cliente: " + e.getMessage());  
            }  
        } else {  
            model.addMessage("Erro ao validar cliente");  
        }  
        request.getRequestDispatcher("/addCustomer/newCustomer.jsp").forward(request, response);  
    }  
}
```



Controller -> View: Geração das Respostas



Controllers → Services

- Como é que os controladores conhecem os serviços?
 - Por exemplo, existe um objeto que sabe fabricar as instâncias dos serviços
 - Candidato: Objeto inicial do domínio
- E quando é criado o objeto inicial do domínio?
 - Através da implementação de *listeners* apropriados é possível reagir a eventos do ciclo de um *Servlet*, nomeadamente à inicialização e finalização
 - É um sítio apropriado para arrancar com a aplicação e os recursos de que ela depende
 - E “injetar” atributo com objeto inicial



Controllers → Services

```
@WebListener
public class Startup implements ServletContextListener {

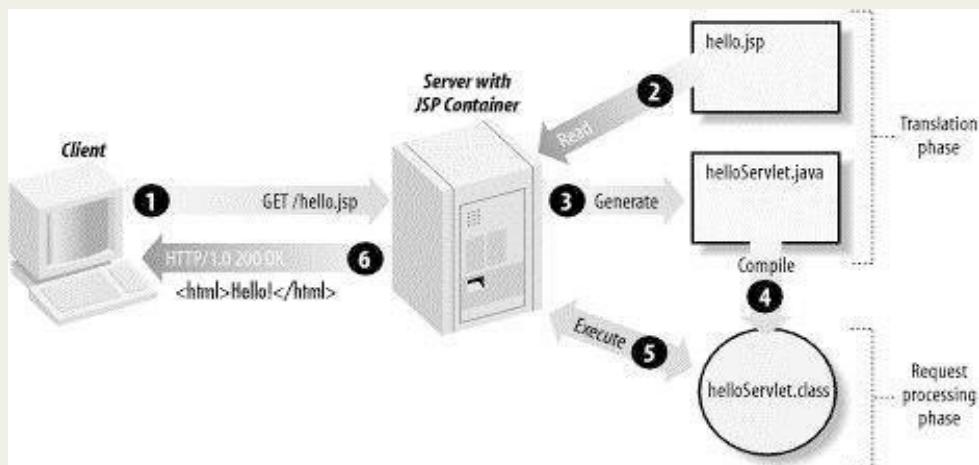
    /**
     * @see ServletContextListener#contextInitialized(ServletContextEvent)
     */
    public void contextInitialized(ServletContextEvent event) {
        SaleSys app = new SaleSys();
        try {
            app.run();
        } catch (ApplicationException e) {
            e.printStackTrace();
        }
        event.getServletContext().setAttribute("app", app);
    }

    /**
     * @see ServletContextListener#contextDestroyed(ServletContextEvent)
     */
    public void contextDestroyed(ServletContextEvent event) {
        SaleSys app = (SaleSys) event.getServletContext().getAttribute("app");
        app.stopRun();
    }
}
```



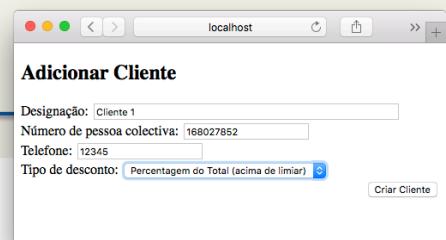
Java Server Pages (JSP)

- Permite definir os *server templates*
- Os programas JSP são estruturado em torno da estrutura da resposta — uma página de texto, no formato HTML
- Na verdade é uma forma de definir um Java servlet sem ter de programar em Java



Exemplo

```
<h2>Adicionar Cliente</h2>
<form action="criarCliente" method="post">
    <div class="mandatory_field">
        <label for="designacao">Designação:</label>
        <input type="text" name="designacao" size="50" value="${model.designation}" />
    </div>
    <div class="mandatory_field">
        <label for="npc">Número de pessoa colectiva:</label>
        <input type="text" name="npc" value="${model.VATNumber}" /> <br/>
    </div>
    <div class="optional_field">
        <label for="telefone">Telefone:</label>
        <input type="text" name="telefone" value="${model.phoneNumber}" />
    </div>
    <div class="mandatory_field">
        <label for="desconto">Tipo de desconto:</label>
        <select name="desconto">
            <c:forEach var="desconto" items="${model.discounts}" %>..</c:forEach>
        </select>
    </div>
    <div class="button" align="right">
        <input type="submit" value="Criar Cliente" />
    </div>
</form>
```



JSP Standard Tag Library (JSTL)

- Biblioteca standard de *tags* que permite definir tarefas comuns na criação dinâmica de HTML
 - sem ser necessário ter conhecimentos de Java
 - mantendo a apresentação separada do resto da aplicação
- Oferece vários tipos de *tags*
 - <c:xxx> para definir o fluxo de controlo na página JSP
 - <fmt:xxx> para definir formatação de datas e números e internacionalização
 - ...
- Fácil de aprender e usar

<comando arg1="..." ... argN="..."> ... </comando>



JSP Standard Tag Library (JSTL)

- Introduzida para substituir as *scriptlets*, i.e., páginas JSP com código Java embebido <% ... %>
 - código não reutilizável
 - código não testável
 - código não documentável
 - complica a estrutura do JSP
 - torna mais difícil a leitura



Exemplo

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<label for="desconto">Tipo de desconto:</label>
<select name="desconto">
    <c:forEach var="desconto" items="${model.discounts}">
        <c:choose>
            <c:when test="${model.discountType == desconto.id}">
                <option selected="selected" value="${desconto.id}">${desconto.de
            </c:when>
            <c:otherwise>
                <option value="${desconto.id}">${desconto.description}</option>
            </c:otherwise>
        </c:choose>
    </c:forEach>
</select>

<c:if test="${model.hasMessages}">
    <p>Mensagens</p>
    <ul>
        <c:forEach var="mensagem" items="${model.messages}">
            <li>${mensagem}</li>
        </c:forEach>
    </ul>
</c:if>
```

JSP Expression Language (EL)

- Permite a resolução dinâmica de objetos e métodos no JSP, evitando a escrita de código Java

`${person.address.city}`

vs

```
<%=request.getAttribute("person") .
    getAddress().getCity() %>
```

- Assenta na especificação do **JavaBeans** no acesso às propriedades
 - classes serializáveis,
 - com construtor sem argumentos,
 - que permitem acesso às suas propriedades com *setters* e *getters* com nomes que seguem convenções
 - ...



JSP Expression Language (EL)

Template JSP

```
<jsp:useBean id="model"
              class="presentation.web.model.NewCustomerModel"
              scope="request"/>

<label for="npc">Número de pessoa colectiva:</label>
<input type="text" name="npc" value="${model.VATNumber}"/>
```

JavaBean

```
public class NewCustomerModel extends Model {

    private String designation;
    private String vatNumber;
    private String phoneNumber;
    private String discountType;

    public void setDesignation(String designation) {}

    public String getDesignation() {}

    public void setVATNumber(String vatNumber) {}

    public String getVATNumber() {}
```



Guided tour – preencher o formulário

The screenshot shows a web browser window with the URL <http://localhost:8080/domain-model-jpa-web-v0/action/clientes/novoCliente>. The page title is "Adicionar Cliente". The form fields are:

- Designação:
- Número de pessoa colectiva:
- Telefone:
- Tipo de desconto:

A red oval highlights the "Criar Cliente" button at the bottom right of the form.



As entradas do formulário

localhost

Adicionar Cliente

Designação: Cliente 1
Número de pessoa colectiva: 168027852
Telefone: 12345
Tipo de desconto: Percentagem do Total (acima de limiar)

Criar Cliente

```
<form action="criarCliente" method="post">
    <div class="mandatory_field">
        <label for="designacao">Designação:</label>
        <input type="text" name="designacao" size="50"
value="${model.designation}" />
    </div>
    <div class="mandatory_field">
        <label for="npc">Número de pessoa
colectiva:</label>
        <input type="text" name="npc"
value="${model.VATNumber}" /> <br/>
    </div>
    <div class="optional_field">
        <label for="telefone">Telefone:</label>
        <input type="text" name="telefone"
value="${model.phoneNumber}" />
    </div>
    <div class="mandatory_field">
        <label for="desconto">Tipo de desconto:</label>
        <select name="desconto">
    </div>
    <div class="button" align="right">
        <input type="submit" value="Criar Cliente">
    </div>
</form>
```



Ciências
ULisboa

| Informática

Envio do pedido HTTP

<http://localhost:8080/domain-model-jpa-web/action/clients/criarCliente>

localhost

Adicionar Cliente

Designação: Cliente 1
Número de pessoa colectiva: 168027852
Telefone: 12345
Tipo de desconto: Percentagem do Total (acima de limiar)

Criar Cliente

HTTP POST request

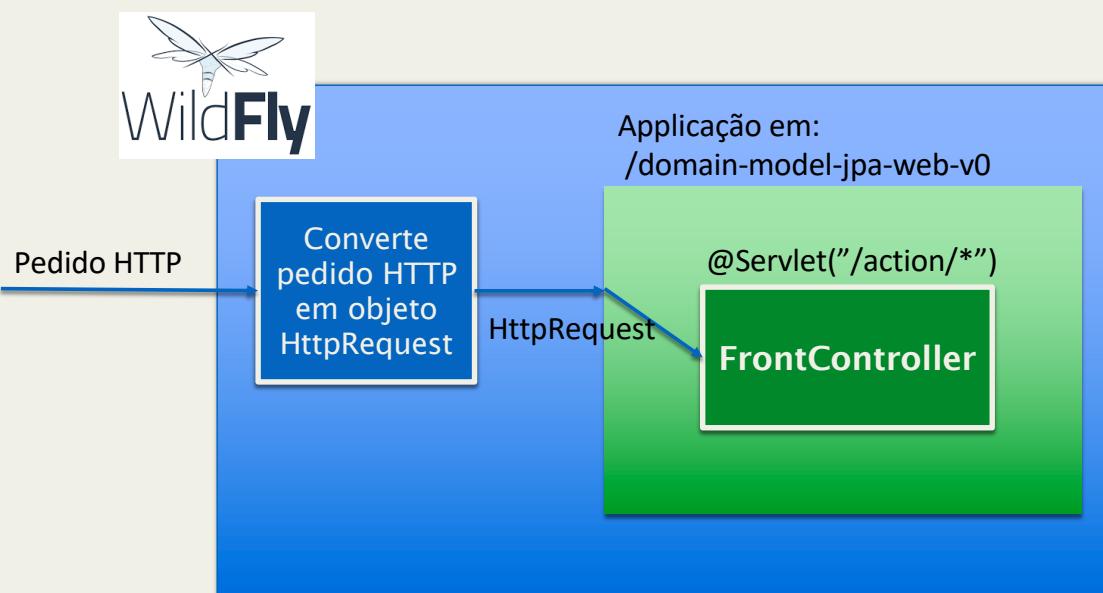
```
{
    designacao: cliente1
    npc: 168027852
    telefone: 12345
    desconto: 2
}
```



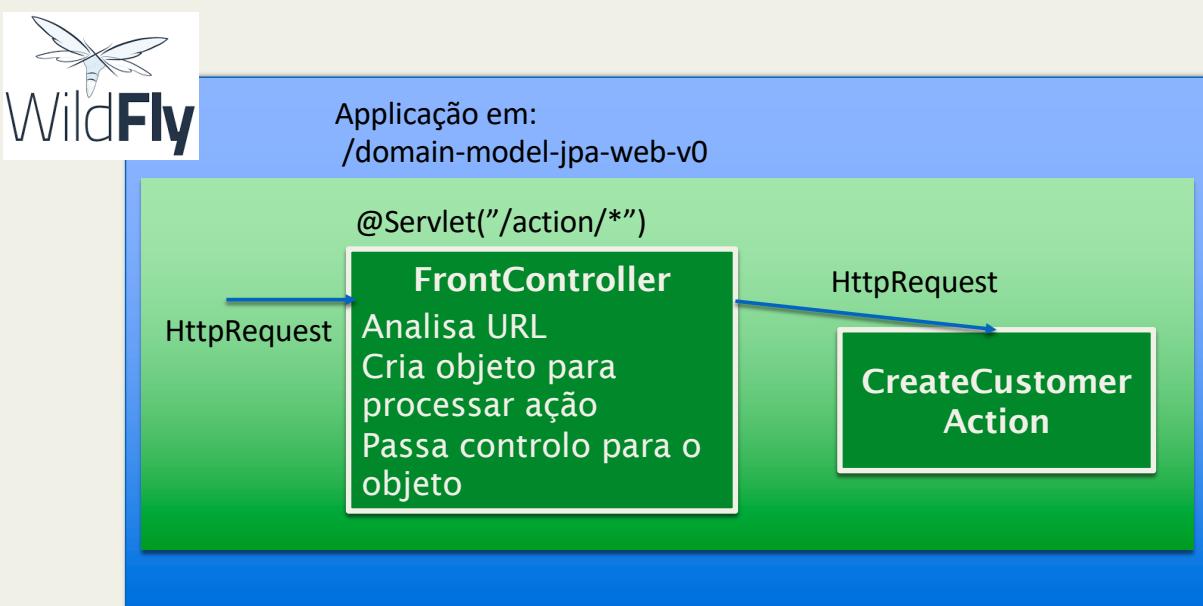
Ciências
ULisboa

| Informática

Conversão do pedido & entrega ao SaleSys



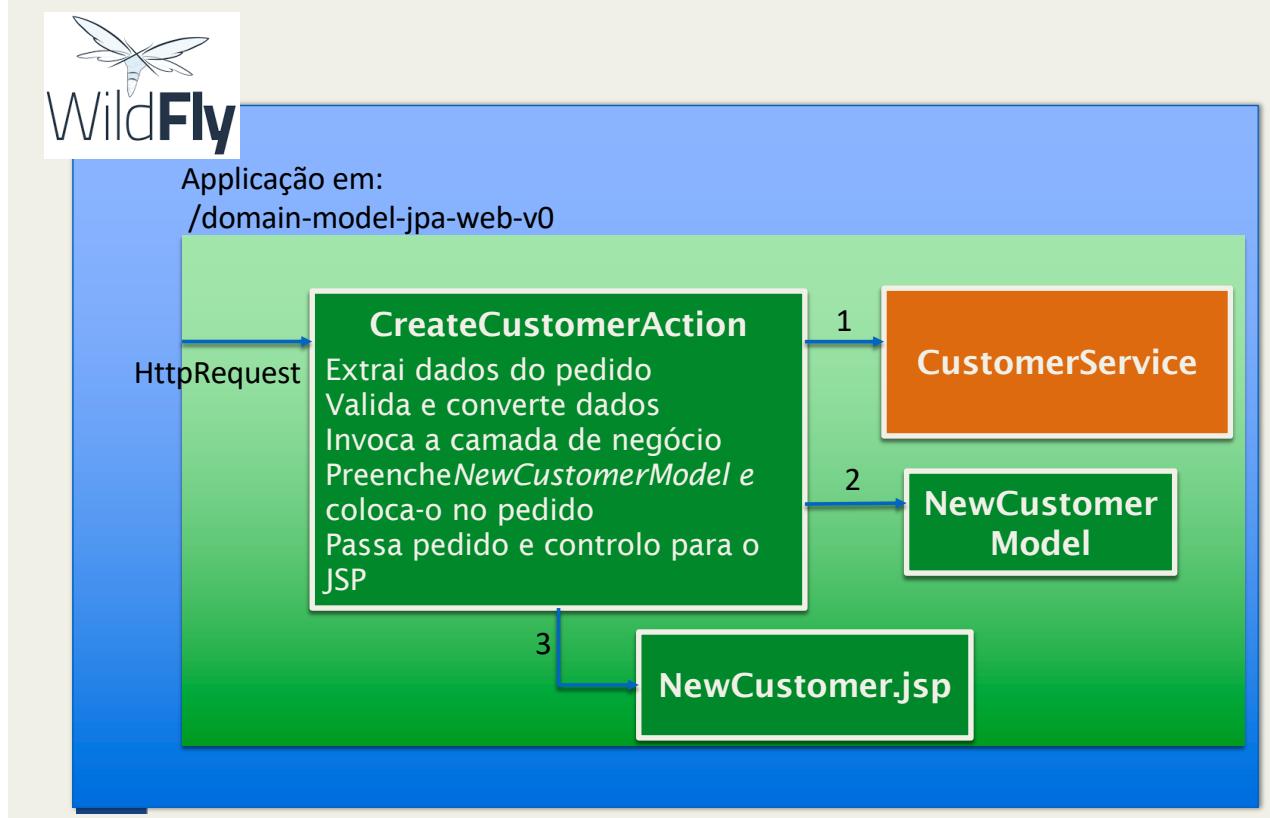
O trabalho do *FrontController*



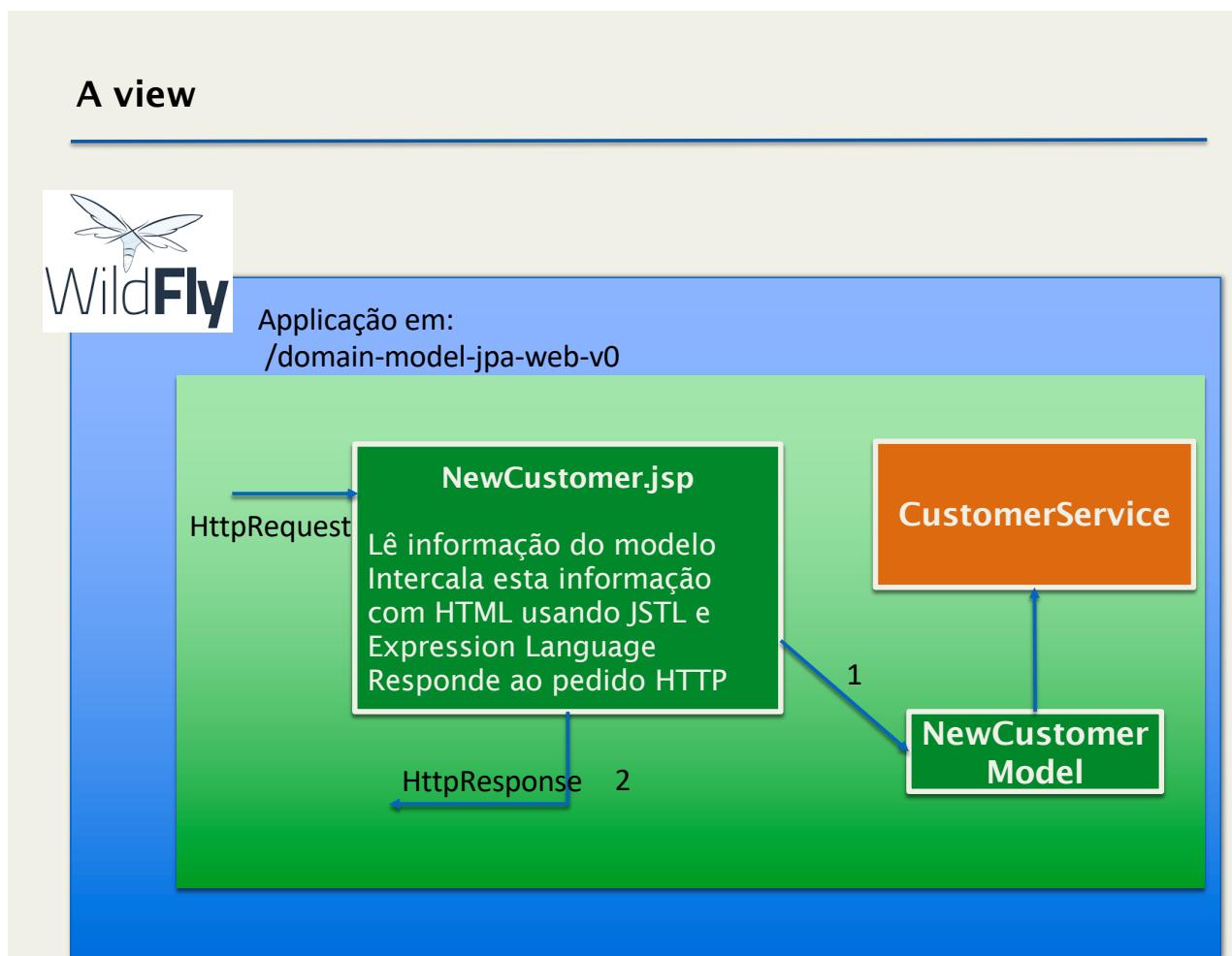
```
appRoot/action/clientes/criarCliente = presentation.web.inputcontroller.actions.CreateCustomerAction
```



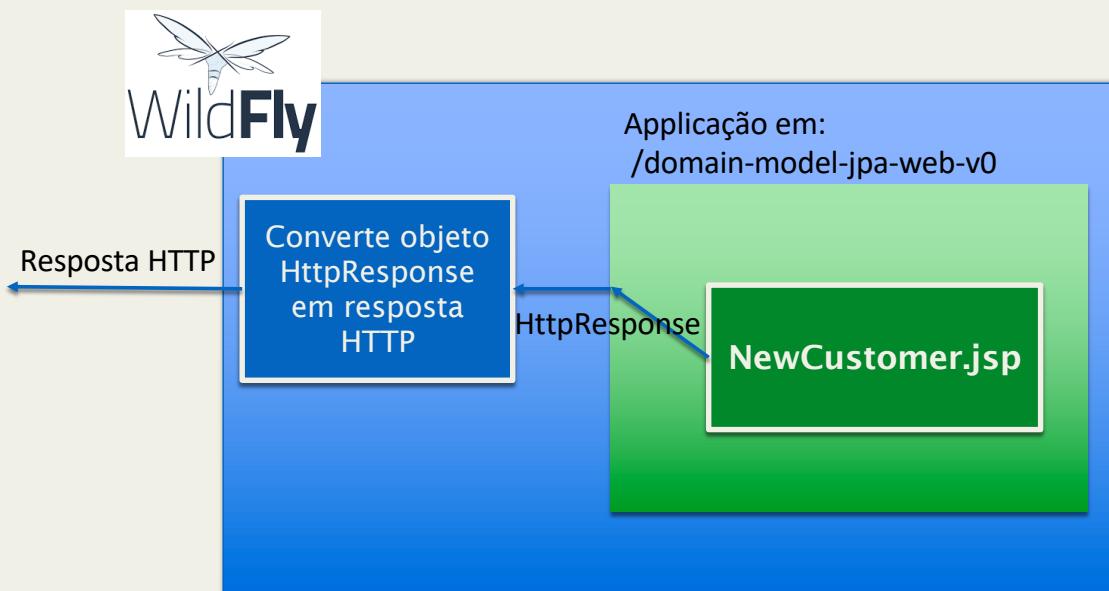
O trabalho de uma Action



A view



Conversão do pedido HTTP num objeto HttpRequest + entrega Sale Sys



Envio HTTP da resposta

