



***Open-source Education***

**Desenvolvimento Web com design-patterns e Struts**  
**Iniciativa Globalcode**

## Agenda

- **Introdução básica**
- **Patterns e certificações Sun**
- **Exemplos GoF patterns**
- **J2EE patterns**
- **Conclusões**

## Palestrante

---

- Vinicius M. Senger
  - Trabalha com **desenvolvimento** de softwares a mais de **12 anos**;
  - Foi **instrutor** e **consultor** Java da: Sun do Brasil e Oracle;
  - **Palestrante** em diversos eventos Java: Fenasoft, Objetos Distribuídos, JustJava, JavaOne EUA, ;
  - **Certificações**: Sun Enterprise Architect (2ª fase), Java Programmer 1.4, Sun Official Instructor, Oracle Instructor, Microsoft Certified Professional, Microsoft Certified Trainner;
  - Programador, diretor técnico e instrutor Globalcode;

## Agenda

- **Introdução básica**
- **Patterns e certificações Sun**
- **Exemplos GoF patterns**
- **J2EE patterns**
- **Conclusões**

## Introdução básica

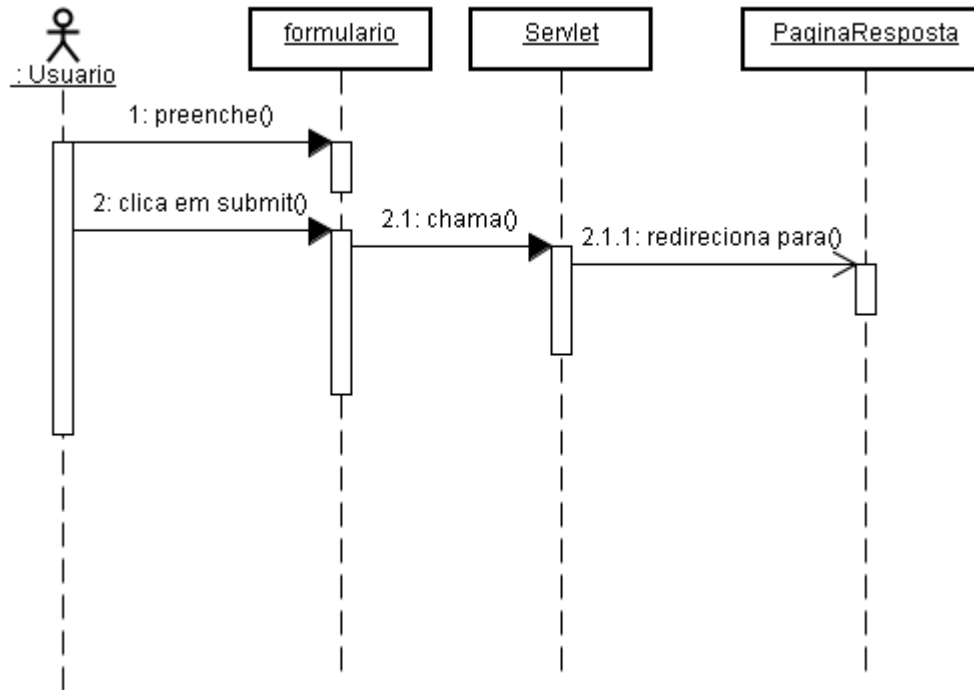
- Um design-pattern é...
  - Uma forma **padrão** de organizar classes e objetos;
  - Nomes para soluções que você já modelou;
  - Uma forma de compartilhar conhecimentos sobre POO;
  - Soluções POO para problemas que incidem em diversos cenários de desenvolvimento;
  - Uma definição de conjunto finito de responsabilidades para uma classe;

## Introdução básica

- Ao adotar design-patterns...
  - Seu código fica mais organizado;
  - Aumento de qualidade;
  - Menor complexidade;
  - Aumenta comunicação dentro da equipe de desenvolvimento;

## Introdução básica

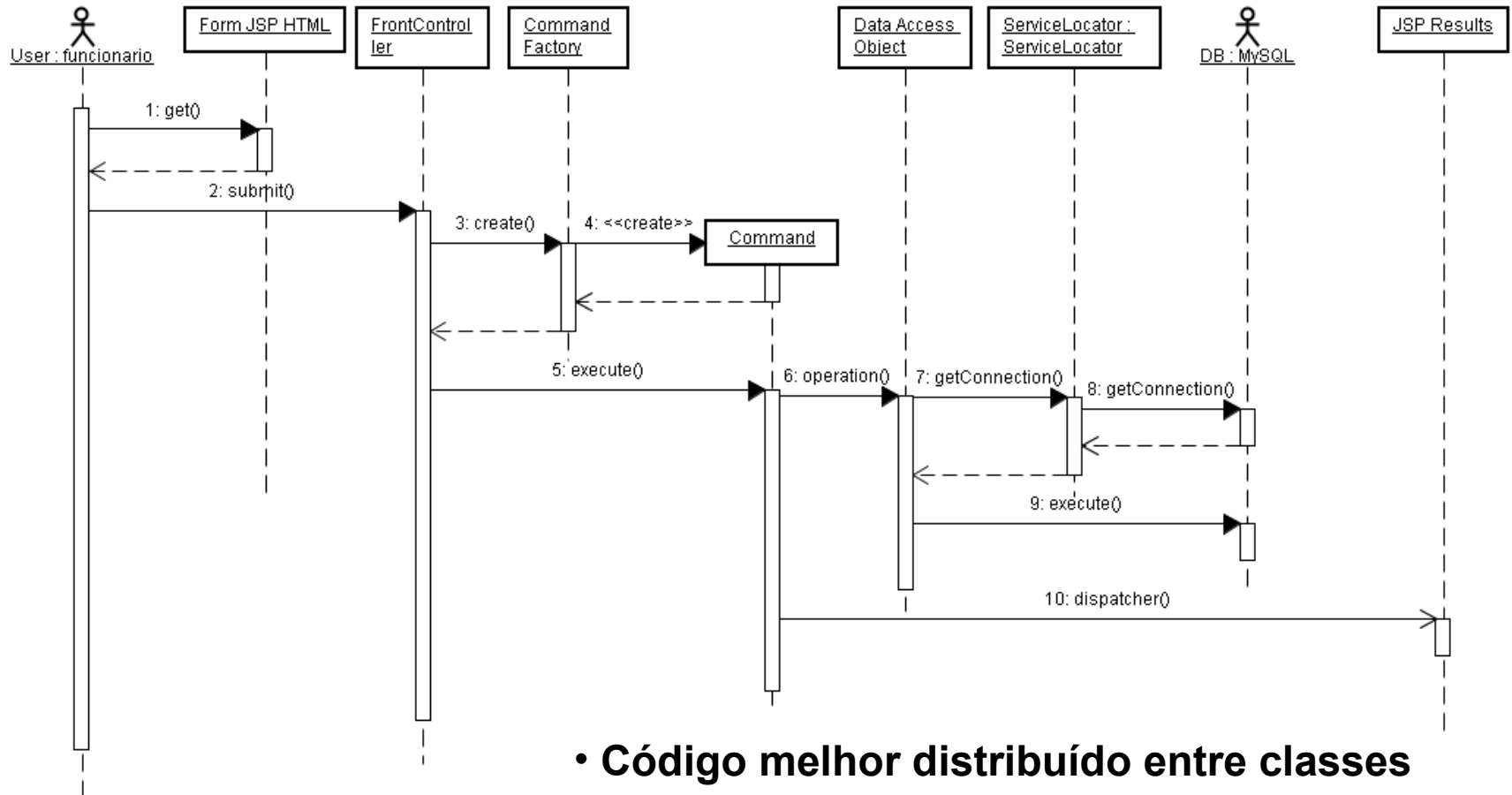
- Diagrama de seqüência UML, código sem pattern



- Código extenso
- Muitas responsabilidades por classe

# Introdução básica

- Com patterns...



- Código melhor distribuído entre classes
- Poucas responsabilidades por classe



## Introdução básica

- A definição de um pattern pode conter...
  - **Um nome:** Transfer Object
  - **Um outro nome (also know as):** Value Object
  - **Um problema:** algumas entidades contém dados que são sempre lidos em grupo...
  - **Uma solução:** serializar todos os dados da entidade em um objeto que...

## Introdução básica

- Famílias de patterns
  - **GoF**: 23 patterns
    - *Criação*: Abstract Factory, Builder, Factory Method, Prototype, Singleton
    - *Estrutura*: Adapter, Bridge, Composite, Decorator, Façade, Flyweight, Proxy
    - *Comportamento*: Chain of Resp., Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method, Visitor

## Introdução básica

- Famílias de patterns
  - **J2EE:** Business Delegate, Composite Entity, Composite View, Data Access Object, Fast Lane Reader, Front Controller, Intercepting Filter, Model-view-controller, Service Locator, Session Façade, Transfer Object, Value List Handler, View Helper
  - Outros: [theserverside.com](http://theserverside.com), [securitypatterns.org](http://securitypatterns.org), Microsoft .NET patterns

## Agenda

- **Introdução básica**
- **Patterns e certificações Sun**
- **Exemplos GoF patterns**
- **J2EE patterns**
- **Conclusões parciais sobre patterns**

## Patterns e Certificação

- As seguintes certificações Sun exigem conhecimentos de patterns:
  - Sun Certified Web Components Developer;
  - Sun Certified Business Component Developer;
  - Sun Certified Enterprise Architect;
- O que e quanto estudar?
  - Todos patterns J2EE;
  - Aplicar na prática os principais GoF e os mais obscuros conhecer a teoria básica;

## Patterns e Certificação

- Qual das opções não é um benefício da utilização dos design-patterns:
  - a) Eles fornecem uma linguagem comum para discussões sobre o design.
  - b) Eles fornecem soluções para os problemas “do mundo real”.
  - c) Ele comunicam a experiência obtida previamente.
  - d) Eles fornecem soluções aos problemas totalmente inusitados.

## Patterns e Certificação

- Qual das opções não é um benefício da utilização dos design-patterns:
  - a) Eles fornecem uma linguagem comum para discussões sobre o design.
  - b) Eles fornecem soluções para os problemas “do mundo real”.
  - c) Ele comunicam a experiência obtida previamente.
  - d) Eles fornecem soluções aos problemas totalmente inusitados.

## Patterns e Certificação

- O design pattern Decorator aparece frequentemente em qual pacote Java:
  - a) java.io
  - b) java.awt
  - c) java.lang
  - d) java.util



## Patterns e Certificação

- O design pattern Decorator aparece frequentemente em qual pacote Java:

a) **java.io**

b) java.awt

c) java.lang

d) java.util

## Agenda

- **Introdução básica**
- **Patterns e certificações Sun**
- **Exemplos GoF patterns**
- **J2EE patterns**
- **Conclusões parciais sobre patterns**

# Singleton

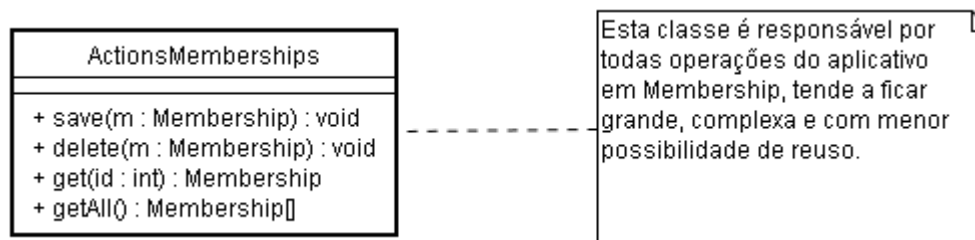
---

- **Definição:** garantir que uma classe tenha somente uma instance.
- **Warning:** Devemos tomar cuidado com NullPointerException em servidores em cluster com Singleton implementados com static

```
public class FormatHelper {  
    private static FormatHelper instance = new FormatHelper();  
    ...  
    public static FormatHelper getInstance() {  
        return instance;  
    }  
    protected FormatHelper() {  
    }  
    public String fullDateFormat(java.util.Date data) {  
        if(data==null || data.equals("")) return "";  
        else return dataCompleta.format(data);  
    }  
}
```

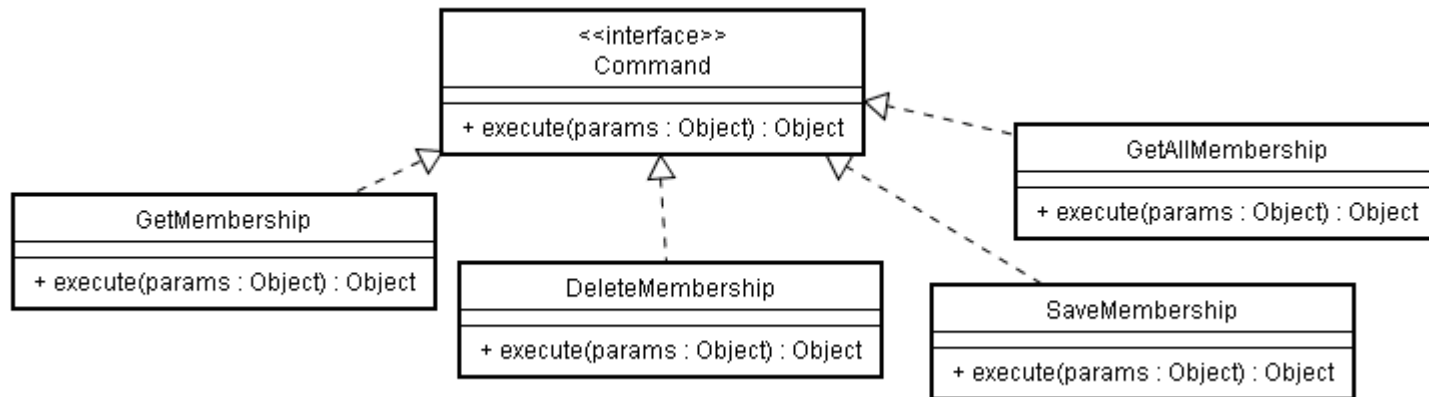
## Command / Action

- **Definição:** encapsula uma requisição ao software em um objeto.
- Action do Struts é o principal exemplo de implementação deste pattern.
- Transformação do método / código em objeto
- Anti-pattern:



## Command / Action

- Após pattern



## Factory

- **Definição:** as responsabilidades de criar objetos a partir de um tipo (interface) é atribuída à uma classe que contempla tal lógica. Exemplo do anti-pattern:

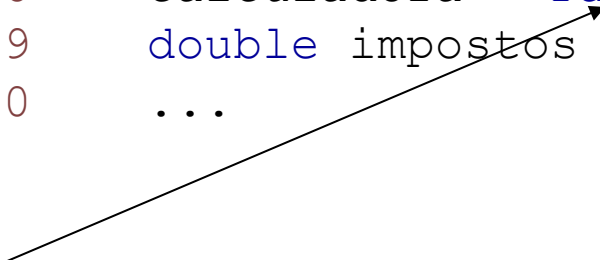
```
16      ...
17      CalculadoraDeImpostos calculadora = null;
18      if(p.fornecedor.getUF().equals("SP")) {
19          calculadora = new CalculadoraSaoPaulo();
20      }
21      else if(p.fornecedor.getUF().equals("RJ")) {
22          calculadora = new CalculadoraRio();
23      }
24      else if(p.fornecedor.getUF().equals("SC")) {
25          calculadora = new CalculadoraSC();
26      }
27      double impostos = calculadora.processar(p);
28      ...
```

## Factory

---

- Ao aplicarmos o pattern factory a responsabilidade sob o operador **new** é da factory...

```
16    ...  
17    CalculadoraDeImpostos calculadora = null;  
18    calculadora = FabricaDeCalculadoras.criar(p) ;  
19    double impostos = calculadora.processar(p) ;  
20    ...
```



- Toda a responsabilidade de criar objetos foi transferida. A factory tem a oportunidade de criar cache / pool de objetos, transparecer qual a classe que implementa tal interface (jdbc), além de transparência de localidade.

## Factory

- Podemos programar fábricas que criam objeto baseados em uma interface;
- Podemos criar fábricas dinâmicas utilizando a API `java.reflection`:
- A classe `DriverManager` é uma factory de `Connections` JDBC que cria dinamicamente um conexão com RDBMS, baseando-se em um URL;



# Composite

- **Definição:** define um estrutura de objetos em formato de árvore de dados

```
public class Produto {  
    private Componente componente;  
    public void setComponente(Componente componente) {  
        this.componente = componente;  
    }  
    public Componente getComponente() {  
        return componente;  
    }  
}  
class Componente {  
    private Componente componente;  
    public void setComponente(Componente componente) {  
        this.componente = componente;  
    }  
    public Componente getComponente() {  
        return componente;  
    }  
}  
class Motor extends Componente {}  
class Cilindro extends Componente {}
```

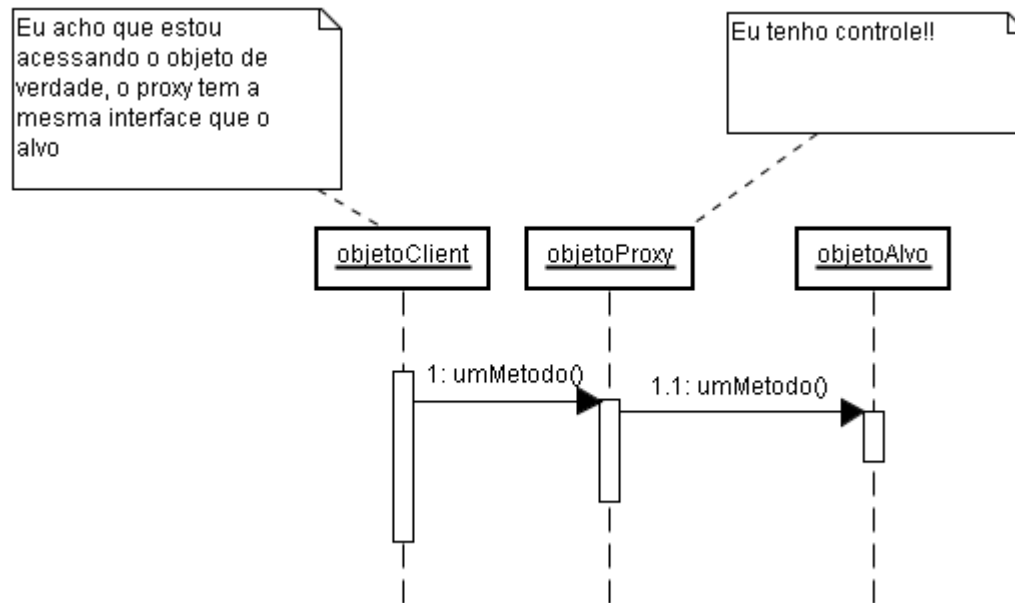
# Composite

---

```
class Usa {  
    public void test() {  
        Produto p = new Produto();  
        Componente motor = new Motor();  
        motor.setComponente(new Cilindro());  
        p.setComponente(motor);  
    }  
}
```

# Proxy

- **Definição:** prover um objeto intermediário para acessar outro objeto.
- O maior exemplo de Proxy em Java são Stubs e Skeletons RMI.



## Agenda

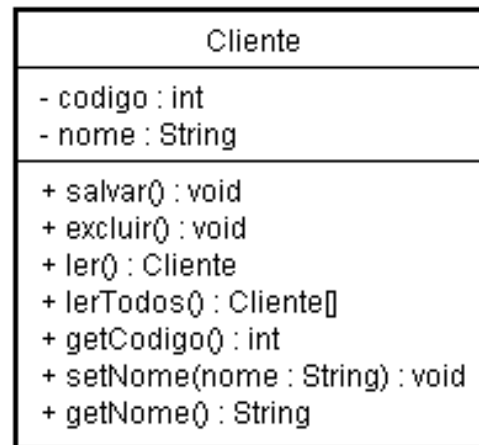
- **Introdução básica**
- **Patterns e certificações Sun**
- **Exemplos GoF patterns**
- **J2EE patterns**
- **Conclusões parciais sobre patterns**

## Service Locator

- **Definição:** simplifica o acesso a recursos J2EE em um aplicativo centralizando lookups JNDI em classes específicas de localização de serviços.
- Evita que sua solução tenha alto acoplamento com JNDI Naming Service;
- Tomar cuidado com Service Locator e cluster;
- Utilize sempre que possível ENC;
- Exemplo no JAREF

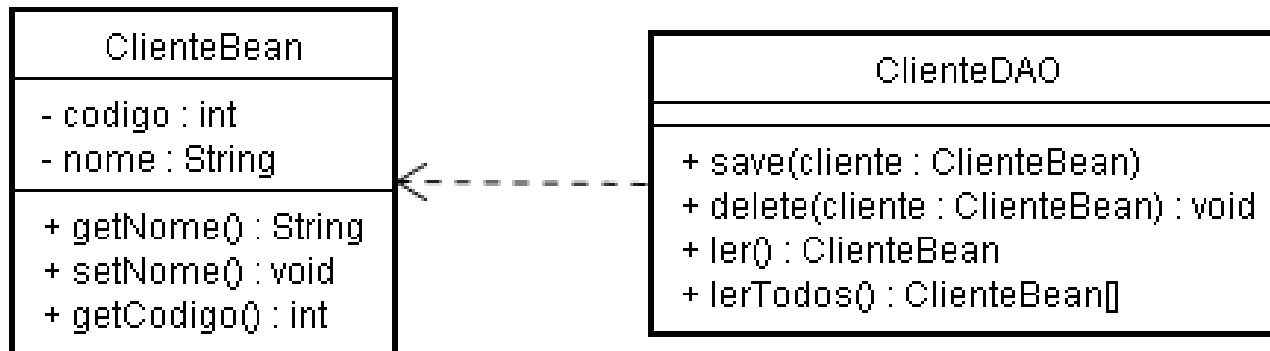
## Data Access Object

- **Definição:** centraliza o serviço de persistência de objetos em um pequeno conjunto de classes, evitando por exemplo que código SQL se espalhe pelo código da solução.
- Anti-pattern:



## Data Access Object

- **Aplicando o pattern:**



## Model-view-controller

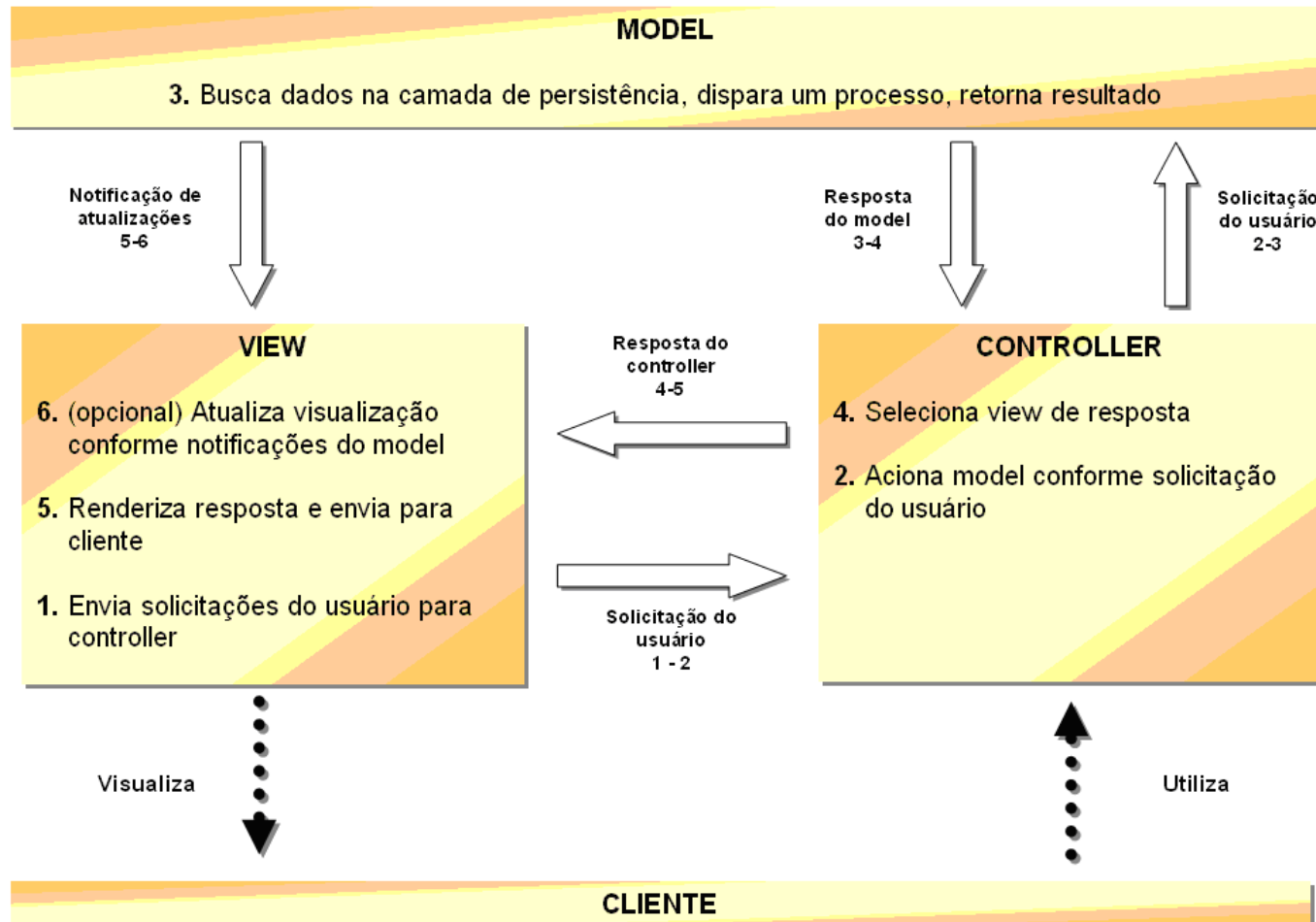
- **Definição:** divide o aplicativo em dados, comportamento e apresentação.
- Aplicando MVC podemos reaproveitar o mesmo dado para múltiplas visualizações;
- Podemos reaproveitar o comportamento (eventos) da solução;
- É um “pattern” de arquitetura, criado há muito tempo. Pode ser aplicado em qualquer linguagem, mais facilmente com OOP.



## **Model-view-controller**

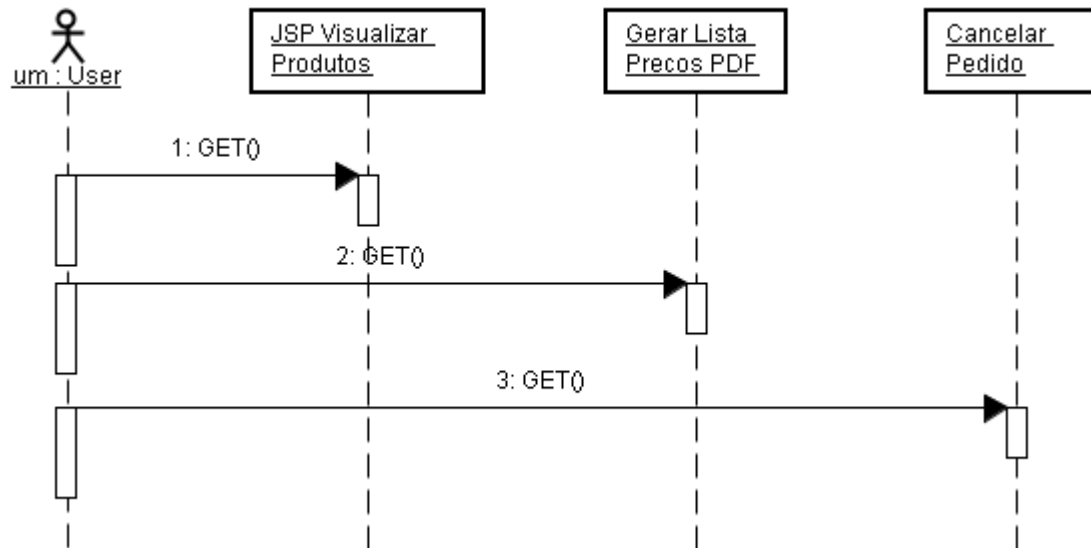
- 115.000 resultados na busca sobre framework MVC no google
- Struts, WebWorks, Spring, PicoContainer são exemplos de frameworks J2EE

# Model-view-controller



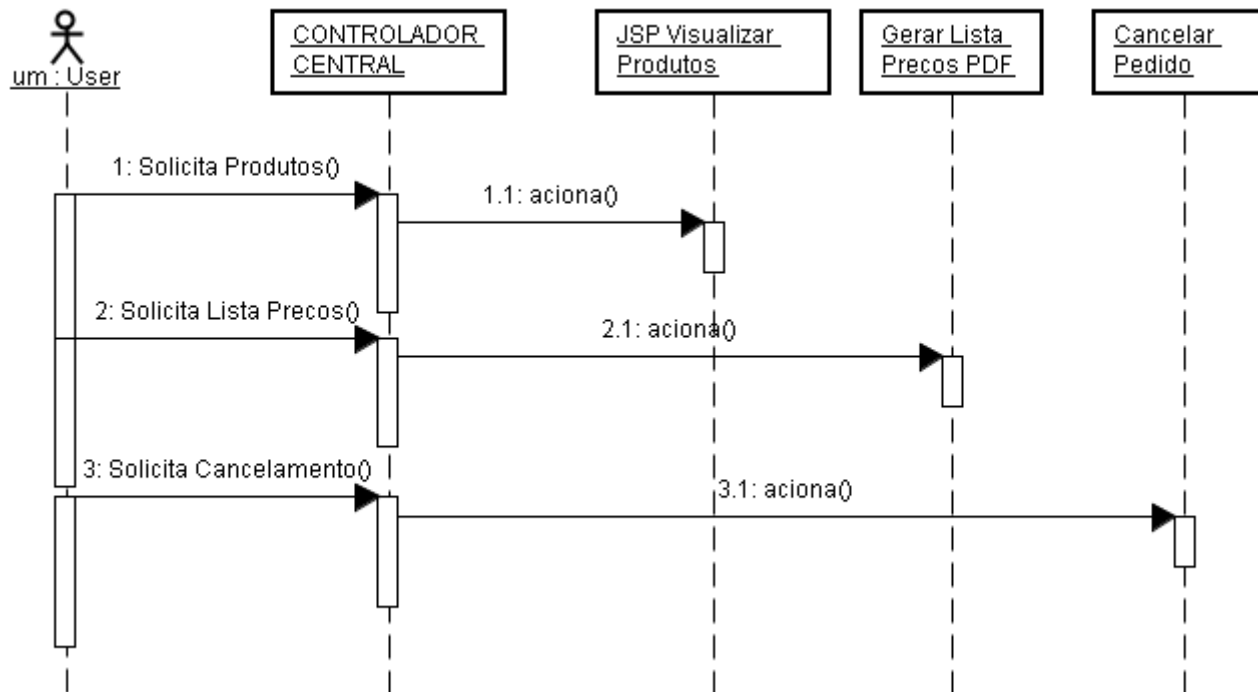
## Front-controller

- **Problema:** em aplicativos com múltiplos pontos de entrada (em Web, múltiplas URL's de acesso; em GUI, métodos espalhados para tratamento de eventos)



## Front-controller

- **Definição:** criar um ponto único de entrada no aplicativo. Para o caso de aplicativos Web, uma única URL permite o acesso aos recursos do servidor.



## **Front-controller**

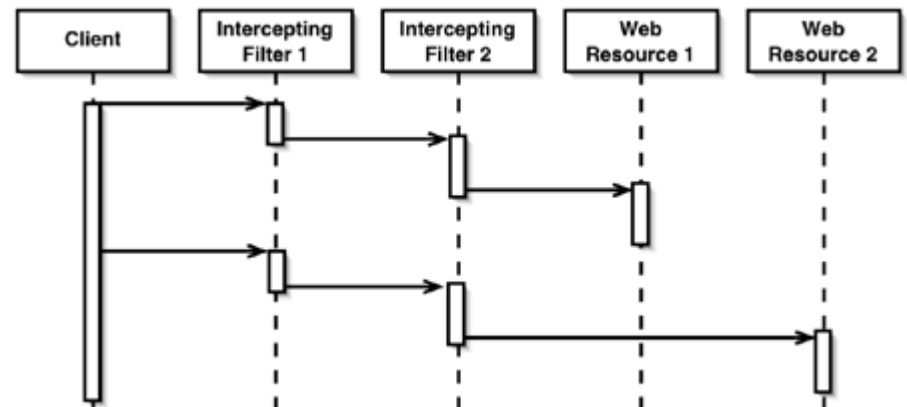
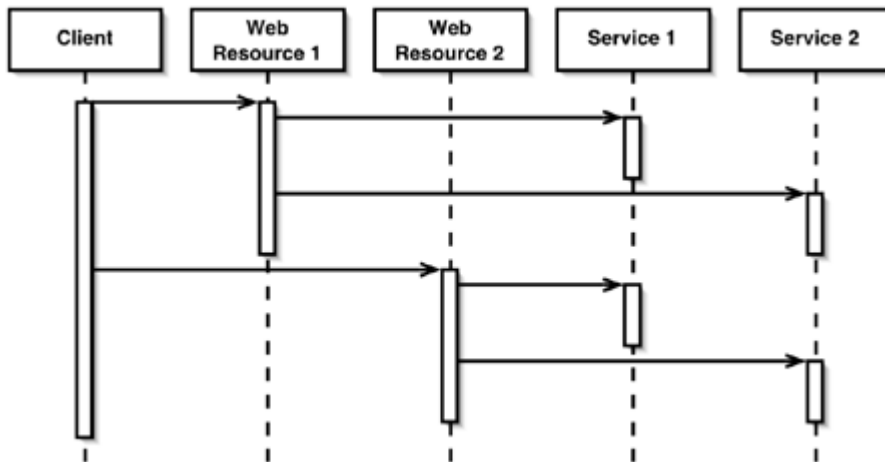
---

- **Um controlador central pode oferecer serviços para o aplicativo:**
  - Logging
  - Segurança
  - Undo / Redo
  - Internacionalização
  - Leitura de formulários
  - Validação de dados
  - Filtro de conteúdo

## Intercepting Filter

- **Definição:** forma para executar pré e pós processamento em requests da solução
- Um Servlet Filter é um exemplo de implementação de Intercepting Filter para interceptar requests no Web Container;
- **Ainda** não temos na espec J2EE intercepting filter para EJB's

# Intercepting Filter



## View Helper

- **Definição:** simplifica a “renderização” de objetos em views com formatação.
- Uma Custom Tag pode representar um View Helper;
- Uma simples classe convencional com métodos estáticos também;
- Exemplo no JAREF



## Composite View

- **Definição:** em views complexas, dividimos a “tela” em diversos pedaços. Depois criamos uma view que reúne determinados componentes / pedaços de tela para compor um front-end.
- Em Web / JSP's utilizamos diversas técnicas de **include**;
- Com aplicativos Swing, uma view composta pode ter diversos panels: panel para botões, panel do menu de opções, panel para apresentar o dado etc.;

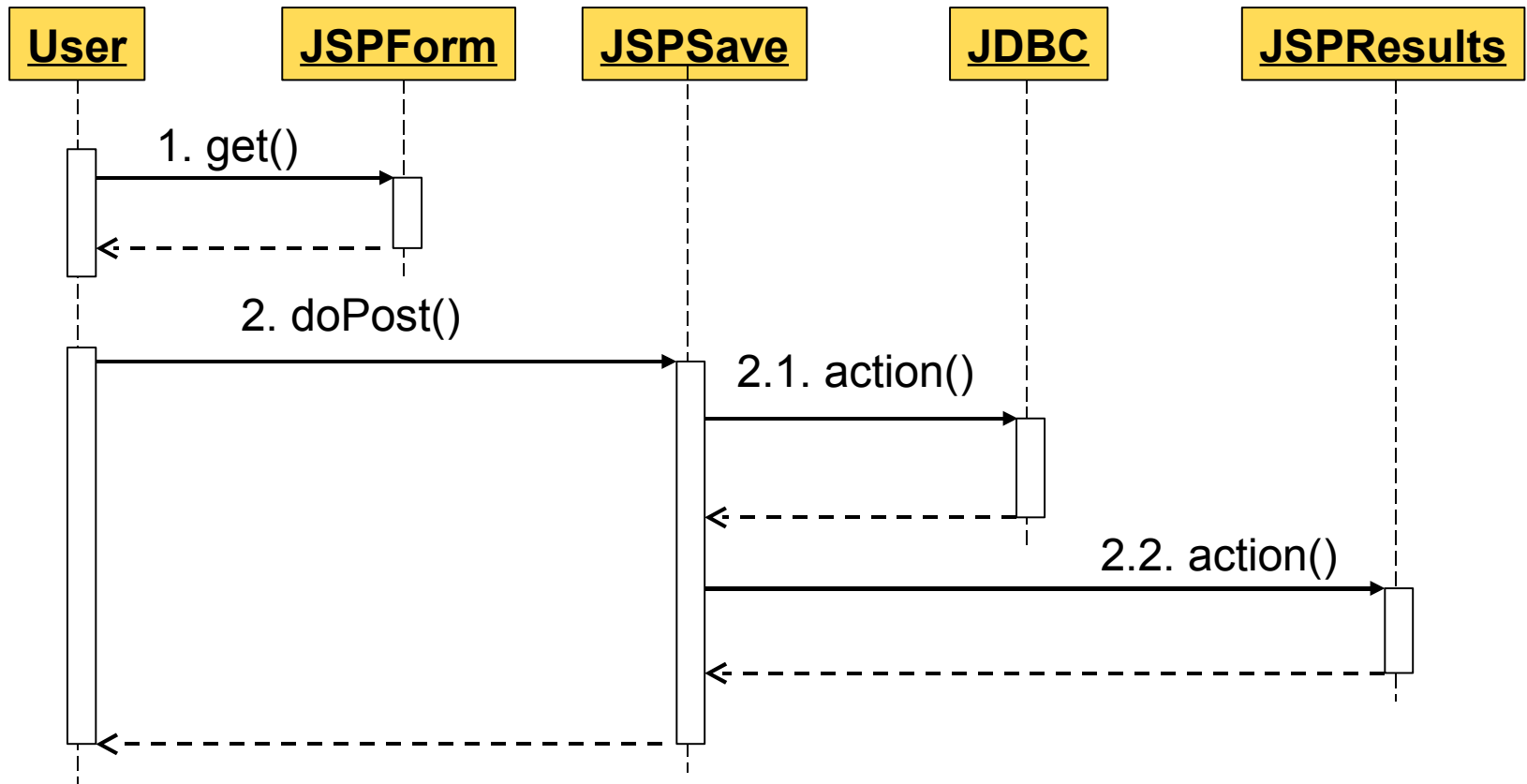
## J2EE Patterns Catalog

Each pattern in this catalog includes sample code from Java™ BluePrints reference applications such as the Java Pet Store.

Pattern Name	Description
<a href="#">Business Delegate</a> <a href="#">[ACM01]</a>	Reduce coupling between Web and Enterprise JavaBeans™ tiers
<a href="#">Composite Entity</a> <a href="#">[ACM01]</a>	Model a network of related business entities
<a href="#">Composite View</a> <a href="#">[ACM01]</a>	Separately manage layout and content of multiple composed views
<a href="#">Data Access Object (DAO)</a> <a href="#">[ACM01]</a>	Abstract and encapsulate data access mechanisms
<a href="#">Fast Lane Reader</a>	Improve read performance of tabular data
<a href="#">Front Controller</a> <a href="#">[ACM01]</a>	Centralize application request processing
<a href="#">Intercepting Filter</a> <a href="#">[ACM01]</a>	Pre- and post-process application requests
<a href="#">Model-View-Controller</a>	Decouple data representation, application behavior, and presentation
<a href="#">Service Locator</a> <a href="#">[ACM01]</a>	Simplify client access to enterprise business services
<a href="#">Session Facade</a> <a href="#">[ACM01]</a>	Coordinate operations between multiple business objects in a workflow
<a href="#">Transfer Object</a> <a href="#">[ACM01]</a>	Transfer business data between tiers
<a href="#">Value List Handler</a> <a href="#">[ACM01]</a>	Efficiently iterate a virtual list
<a href="#">View Helper</a> <a href="#">[ACM01]</a>	Simplify access to model state and data access logic

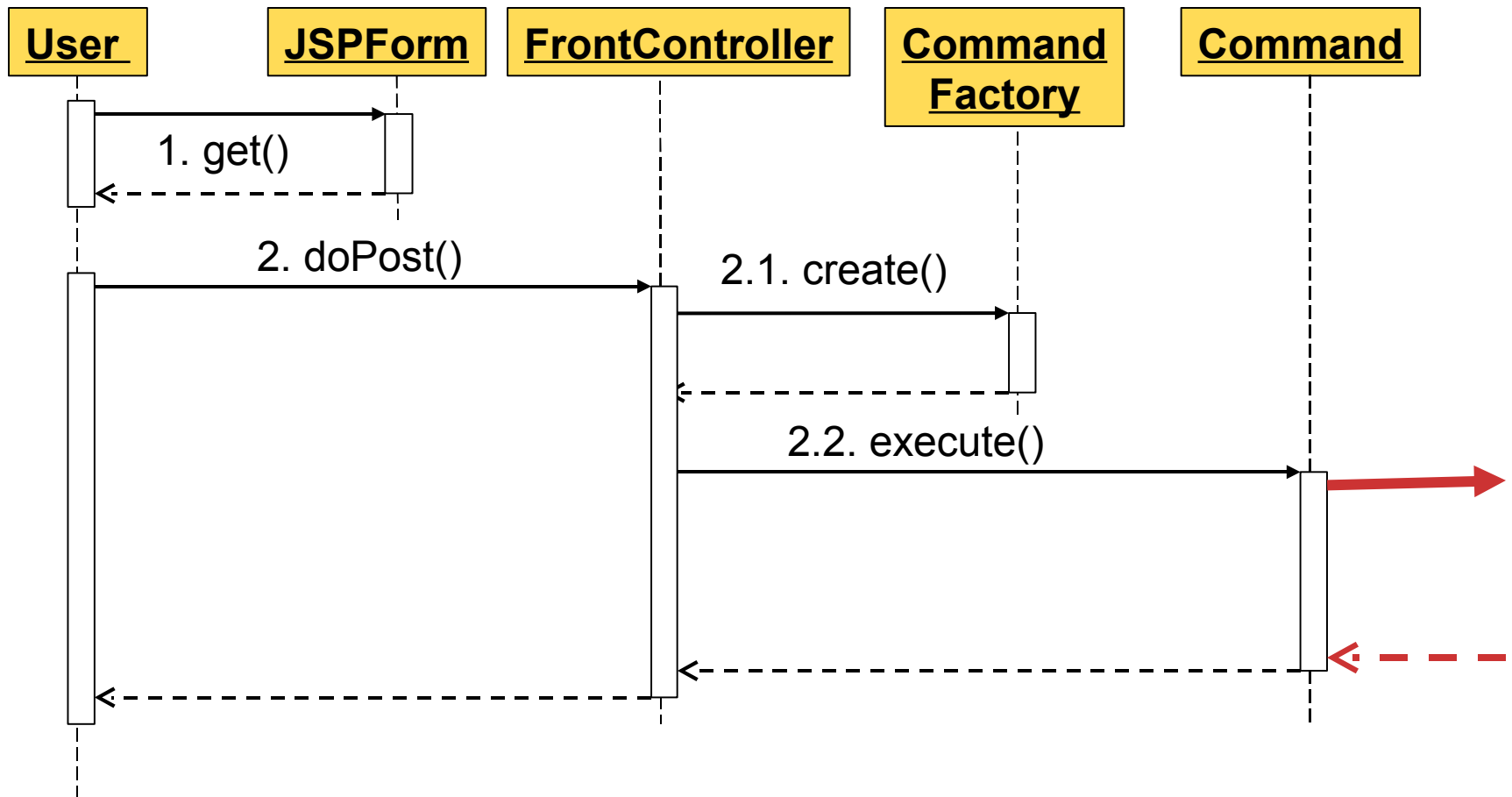
# Arquitetura sem patterns

## #1—Model One



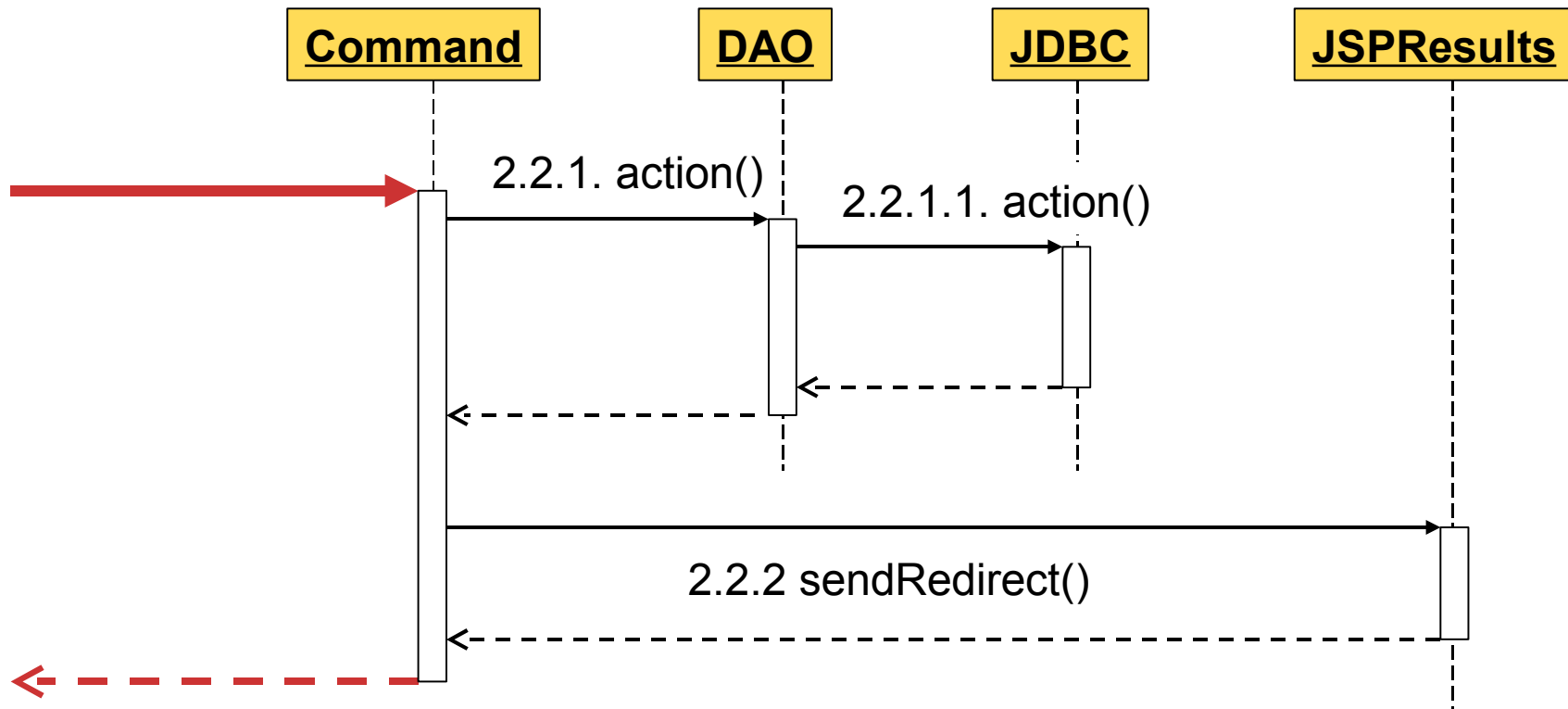
# Arquitetura com patterns

## Arquitetura MVC



# Arquitetura com patterns

## Arquitetura MVC



## Agenda

- **Introdução básica**
- **Patterns e certificações Sun**
- **Exemplos GoF patterns**
- **J2EE patterns**
- **Introdução ao Jakarta Struts**

## Jakarta Struts

- Utilizando patterns você cria soluções com código-fonte mais claro e organizado, porém esta prática requer planejamento e modelagem;
- Planejamento e modelagem de código representa menor produtividade no curto prazo;
- Frameworks unem qualidade de código, com patterns pré-moldados, sem queda de produtividade no curto prazo;

## Jakarta Struts

- Um framework cuida da infra-estrutura técnica comum para diversos tipos de aplicativos
- Um framework disponibiliza um software pré-moldado, basta “recheiar” com as funcionalidades do seu domínio de negócio;
- O Struts é framework de desenvolvimento de aplicativos Web J2EE MVC;
- O Struts trabalha de acordo com diversos patterns, oferecendo alta qualidade de código aliada à alta produtividade no curto prazo;



## Download e instalação

- Download: <http://jakarta.apache.org>;
- Instalação: É um zip/jar, basta descompactar...

jakarta-struts-1.2.4

- contrib
- lib
- webapps

Name	Size
struts-blank.war	1.013 KB
struts-documentation.war	4.519 KB
struts-examples.war	1.110 KB
struts-mailreader.war	1.158 KB
tiles-documentation.war	1.307 KB

Name	Size
antlr.jar	380 KB
commons-beanutils.jar	116 KB
commons-collections.jar	172 KB
commons-digester.jar	107 KB
commons-fileupload.jar	22 KB
commons-logging.jar	38 KB
commons-validator.jar	83 KB
jakarta-oro.jar	64 KB
struts.jar	515 KB
struts-bean.tld	9 KB
struts-html.tld	66 KB
struts-logic.tld	15 KB
struts-nested.tld	64 KB
struts-tiles.tld	15 KB
validator-rules.xml	14,3 KB
struts-config_1_0.dtd	35 KB
struts-config_1_1.dtd	35 KB
struts-config_1_2.dtd	13 KB
tiles-config_1_1.dtd	16 KB
web-app_2_2.dtd	33 KB
web-app_2_3.dtd	

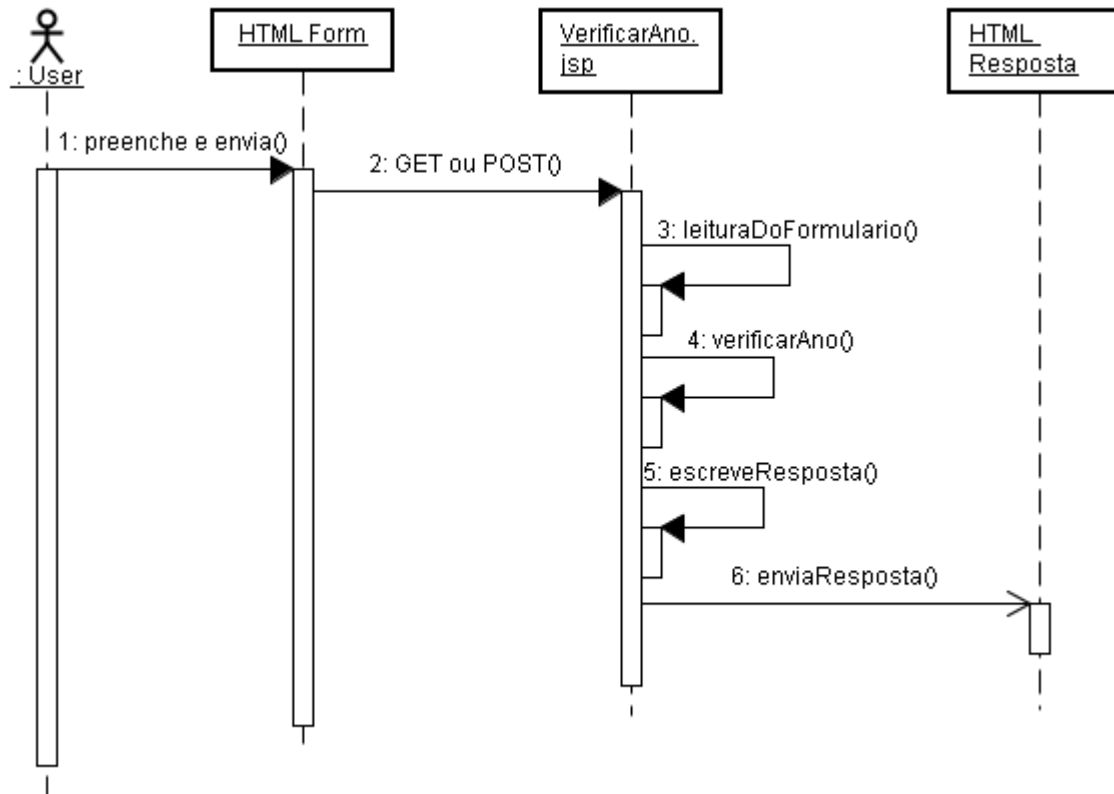
Type: TLD File  
Date Modified: 12/9/2001  
Size: 14,3 KB

## Primeiro aplicativo

- Vamos imaginar um simples exemplo de um aplicativo Web J2EE para verificar se um ano é bissexto ou não;
- Conforme discutimos sobre patterns, temos três abordagens para desenvolver uma solução J2EE Web:
  1. Sem patterns (somente JSP's);
  2. Com patterns (aplicamos patterns com classes de infra-estrutura proprietárias);
  3. Com patterns e framework (utilizando Struts);

## Arquitetura básica

- Com a **primeira alternativa**, teríamos a seguinte solução:

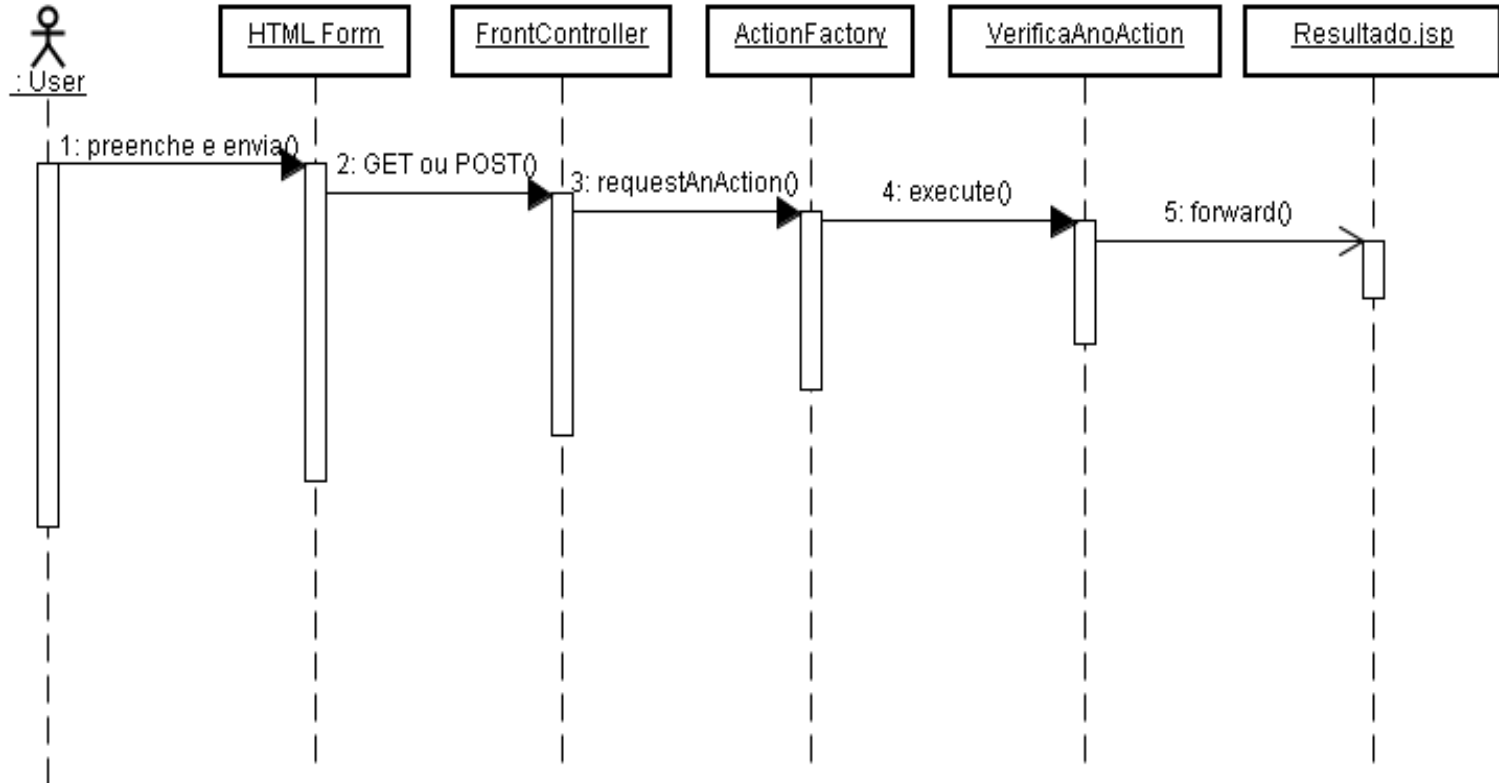


## Arquitetura básica

- Para esta arquitetura nosso trabalho seria o de desenvolver:
  1. Um HTML com formulário de entrada do ano;
  2. VerificarAno.jsp;

## MVC proprietário

- A **segunda opção**, é aplicarmos os patterns “na raça”:

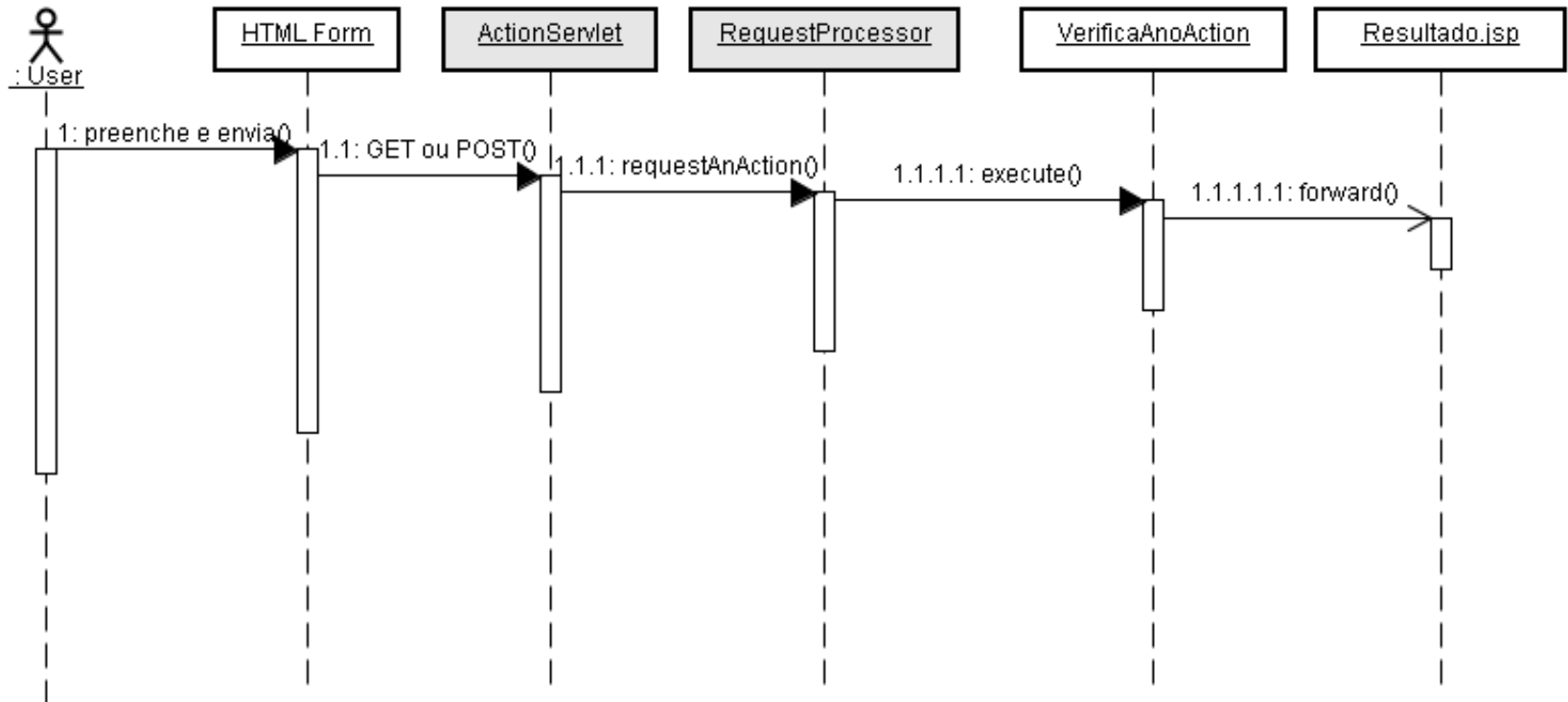


## MVC proprietário

- Esta abordagem é mais elaborada que a anterior, porém teríamos que desenvolver:
  1. Um HTML com formulário de entrada do ano;
  2. Um Java Servlet Front Controller;
  3. Uma classe ActionFactory;
  4. Uma interface para definir o padrão de Action;
  5. Uma classe com a implementação VerificaAnoAction;
  6. A página Resultado.jsp;

## MVC com Struts

- A terceira opção é desenvolver o aplicativo com Struts:



## MVC com Struts

- Abordagem ideal, precisamos desenvolver apenas:
  1. Formulário HTML ou link em uma página;
  2. Classe para responder pelo evento submit do formulário ou clique no link (Action);
  3. Página de reposta;



## Vantagens do Struts

- Struts cuida do controle (de eventos) do aplicativo;
- Struts oferece serviços para facilitar a construção da actions para formulários de dados e validação;
- Struts facilita (muito) a criação de views / página de apresentação através de diversas tag libraries;
- Struts facilita internacionalização do aplicativo;
- Struts é modular e extensível;

## Exemplo com Struts

### Resumo de tarefas

3. Instalamos a biblioteca struts.jar no Application Server;
4. Configuramos o Servlet Front-controller do Struts no web.xml;
5. Criamos uma sub-classe de Action para responder ao evento;
6. Configuramos a Action no struts-config.xml

## Exemplo com Struts

- Instalação do struts.jar:
  - Cada servidor pode oferecer uma forma diferente;
  - Muitos trabalham com o lib/ext;
  - No Jboss podemos simplesmente copiar para o diretório deploy;
  - Podemos empacotar o jar no WEB-INF/lib (+ / -);

# Exemplo com Struts

## web.xml com Struts

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
  <display-name>Acme Struts</display-name>

  <!-- Declaração Padrão do Front-Controller ActionServlet (com debugging) -->
  <servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>
    <load-on-startup>2</load-on-startup>
  </servlet>
  <!-- Padrão para mapeamento para Struts Action -->
  <servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>
</web-app>
```

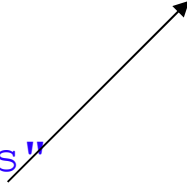
→ Poderia ser .exe, .asp, .php...

# Exemplo com Struts

## Formulário HTML

```
<html>
<head>
<title></title>
</head>
<body>
<FORM name="ExemploStruts"
  action="VerificarAno.do" method="get">
  <p>Exemplo de Struts - Curso Globalcode</p>
  <p>Entre com o ano:
    <input maxlength="4" size="4" value="2000"
      name="textAno">
  </p>
  <p><input type="submit"></p>
</FORM>
</body>
</html>
```

Qualquer coisa .do vai chegar no  
controlador central do Struts



# Exemplo com Struts

## Criação da classe Action

```
package br.com.globalcode.struts.action;

import javax.servlet.http.*;
import org.apache.struts.action.*;

public class VerificarAno extends Action {

    public ActionForward execute(ActionMapping mapping, ActionForm form,
        HttpServletRequest request, HttpServletResponse response) throws Exception {

        /* Podemos ler os dados de entrada e processa-los aqui mesmo, ou apenas
        receber e ler os dados e em seguida delegar o processamento para outra
        classe (business delegate)*/
        boolean resposta = false;
        int ano = Integer.parseInt(request.getParameter("textAno"));
        resposta = ((ano%4==0 && ano%100!=0) || ano%400==0);
        request.setAttribute("resposta", new Boolean(resposta));
        // Em seguida, chamamos uma página através de um nome
        // "mapping" / "alias" definido no struts-config.xml
        return (mapping.findForward("resposta"));
    }
}
```

# Exemplo com Struts

## struts-config.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<!DOCTYPE struts-config PUBLIC
    "-//Apache Software Foundation//DTD Struts
    Configuration 1.1//EN"
    "http://jakarta.apache.org/struts/dtds/struts-
    config_1_1.dtd">

<struts-config>
    <action-mappings>
        <action path="/VerificarAno"

            type="br.com.globalcode.struts.action.VerificarAno" >

                <forward name="resposta" path="/resultado-ano.jsp" />
            </action>
        </action-mappings>
    </struts-config>
```

# Exemplo com Struts

## Página de resultados

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%
Boolean resposta = (Boolean) request.getAttribute("resposta");
%>

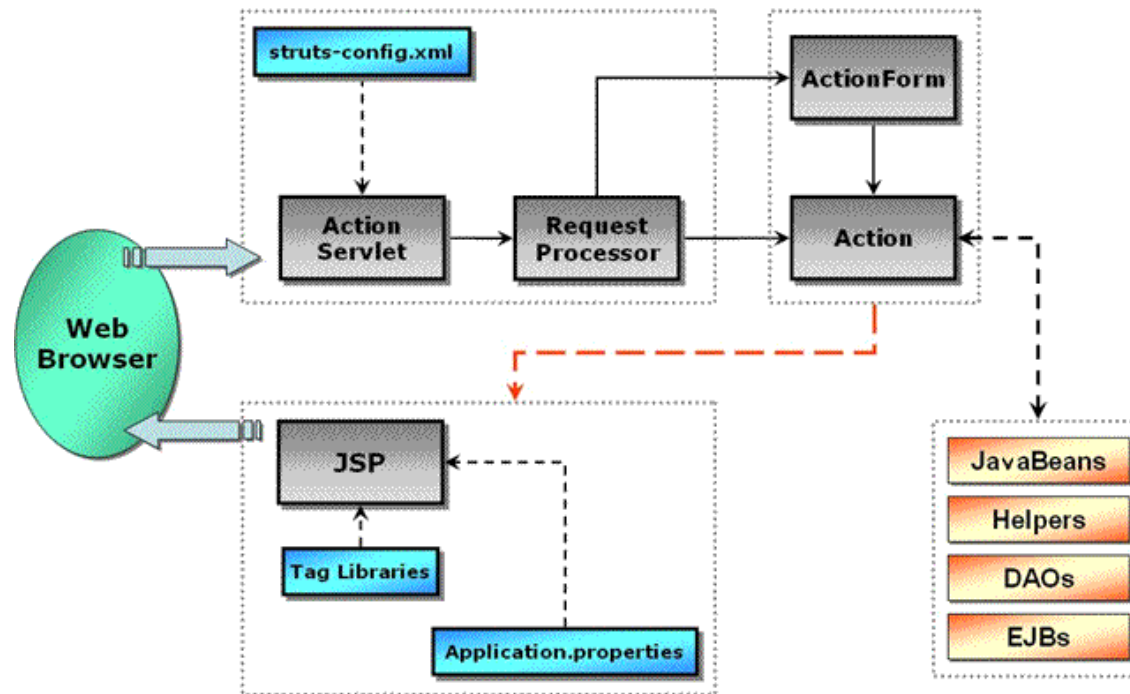
<html>
  <head>
    <title>Resultado verificação AnoBissexto</title>
  </head>

  <body>
    O ano <%= (resposta.booleanValue() ? "" : " não ") + " é
    bissexto" %><br>
  </body>
</html>
```



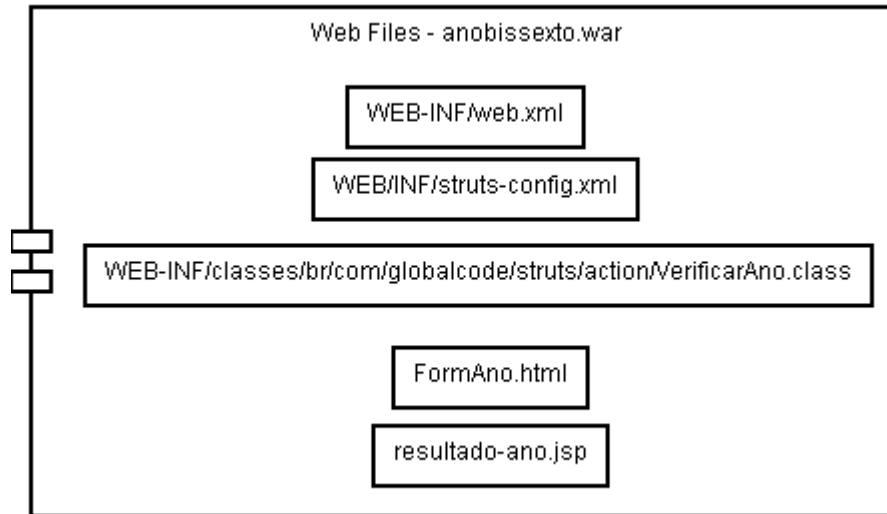
# Exemplo com Struts

## Resumo da arquitetura com Struts



## Exemplo com Struts

Empacotamento do aplicativo em .war



## Demo

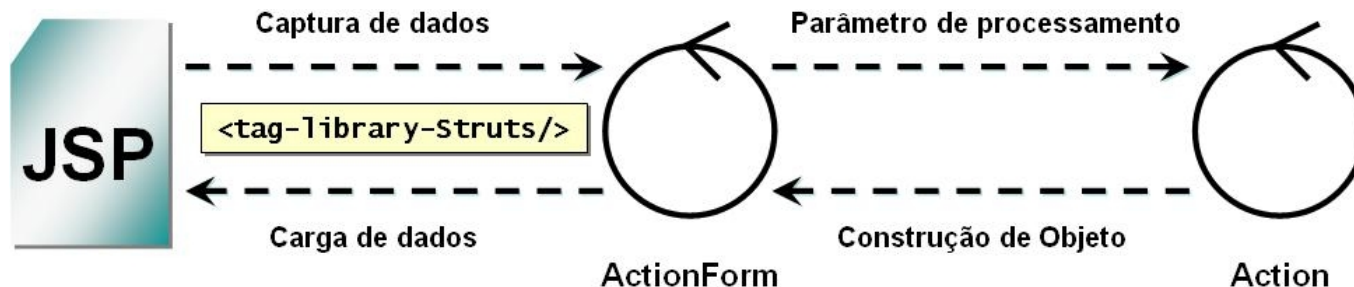
---

- Ano bissexto com Struts

## Mais sobre Struts

- Oferece também recurso de leitura de formulários, ou seja, ele pode fazer automaticamente o código abaixo:

```
String nome = request.getParameter("textNome");  
If(nome.equals("")) ...
```



## Mais sobre Struts

- Oferece um recurso chamado de synchronized tokens, que ajuda a resolver problema de submits repetidos “dejavu”;
- Oferece Tag Libs para simplificar a criação de views:
  - HTML Tag Lib: criação de formulários com dados de JavaBeans, validação e JavaScript;
  - Logic Tag Lib: laços, percorrer coleções, condições entre outros;
  - Bean Tag Lib: includes, variáveis entre outros “utilitários”;

## Mais sobre Struts

- Exemplo de JSP com Tag lib do Struts para formulário:

```
1 <%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html"%>
2   <html:html>
3     <html:form action="RegistrarPesquisa.do">
4       <p>Qual seu time de futebol?</p>
5       <p><html:text property="resposta"/></p>
6       <p>Qual seu nome?</p>
7       <html:submit/>
8     </html:form>
9     <html:errors />
10  </html:html>
11
```

## **Demo**

---

- Ferramentas e / ou outros exemplos