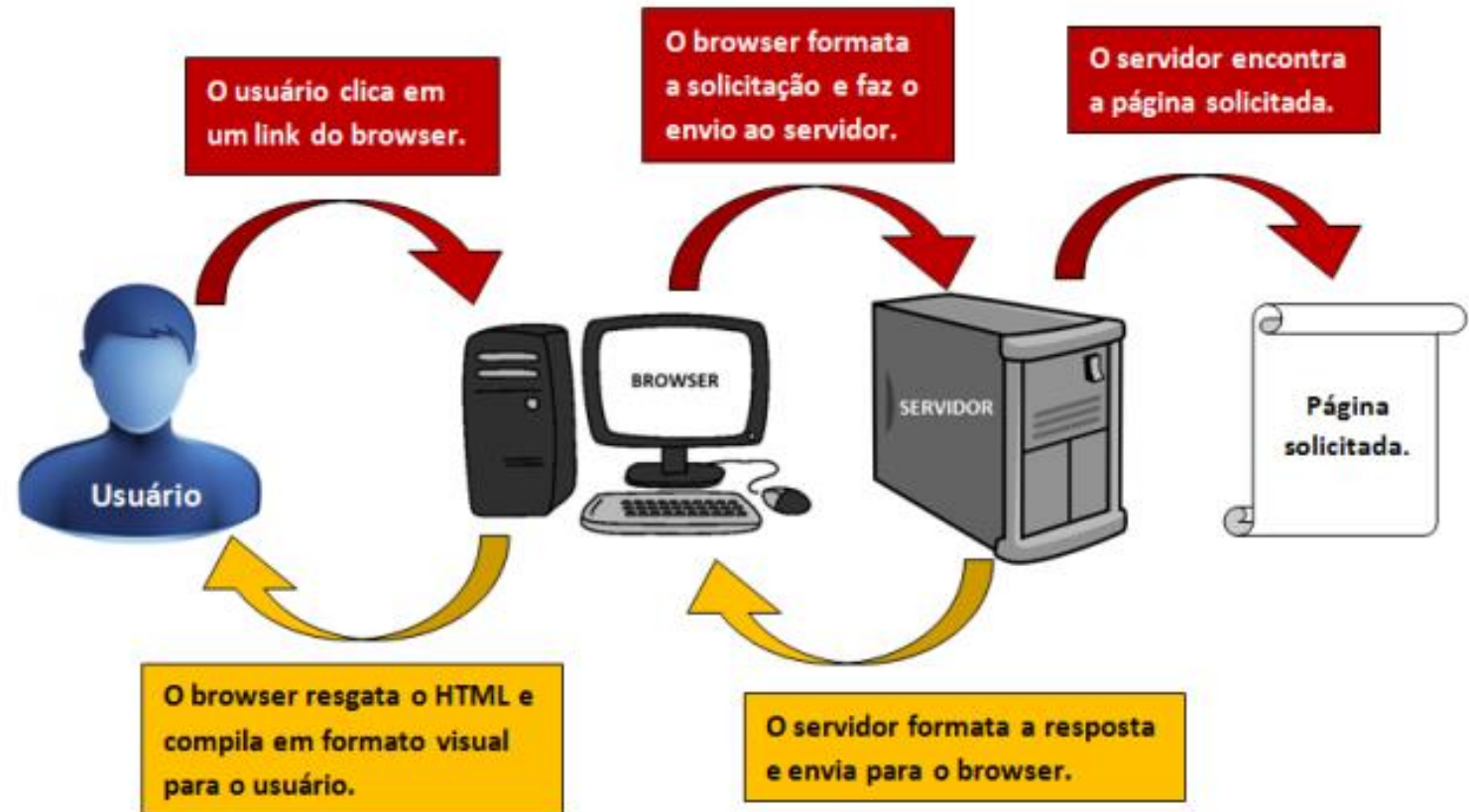




Prog Web

Edson Orivaldo Lessa Junior

Requisições dos usuários



Principais requisitos de aplicações Web



Geração de páginas dinâmicas



Acesso a banco de dados



Gerenciamento de sessões de usuários



Utilização de componentes



Tratamento de erros

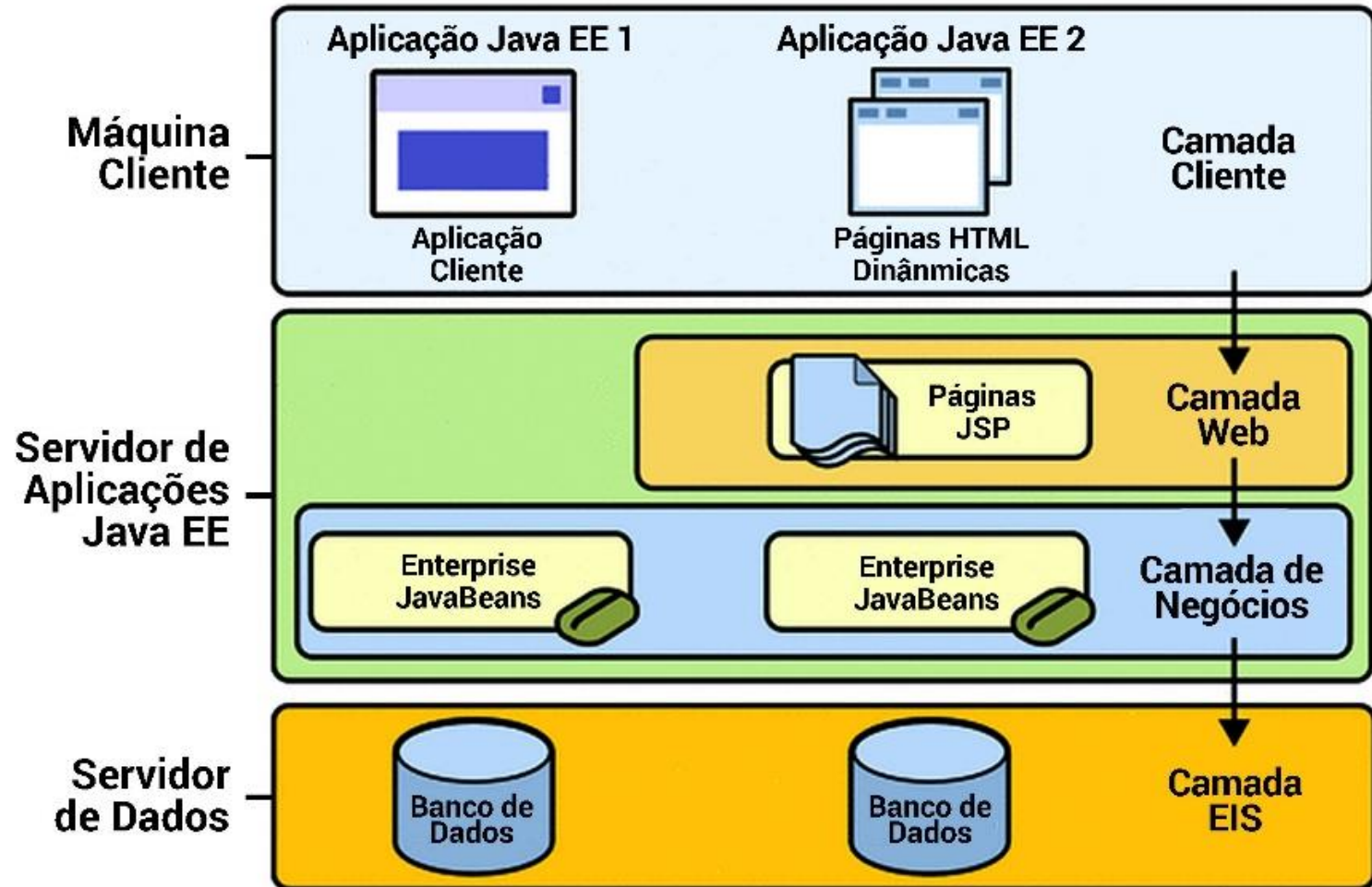


Recebimento e tratamento de requisições



Envio de respostas para o cliente

Servidor de aplicação



GET Request

GET url HTTP/1.1

Headers

Information about
the request

POST Request

POST url HTTP/1.1

Headers

Information about
the request

Body

Information sent as
part of the request,
usually intended for
a web application

HTTP - Requisição

HTTP - Resposta

Response
Status Line

Headers
Information about
the response

Body
HTML, JPG, or a file
in another format



Servlets

Servlets são classes Java instanciadas

Executam em associação com servidores Web

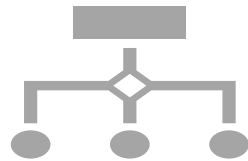
Respondem as requisições realizadas por meio do protocolo HTTP

Os objetos Servlets podem enviar a resposta na forma de uma página HTML

Servlet é uma API para construção de componentes do lado do servidor

Objetivo é de fornecer um padrão para a comunicação entre clientes e servidores.

Benefícios



Desempenho

Não há processo de criação para cada solicitação de cliente

Solicitação é gerenciada pelo processo contentor de servlet

Após o servlet finalizar o processamento de uma solicitação, ele permanece na memória, aguardando por outra solicitação.

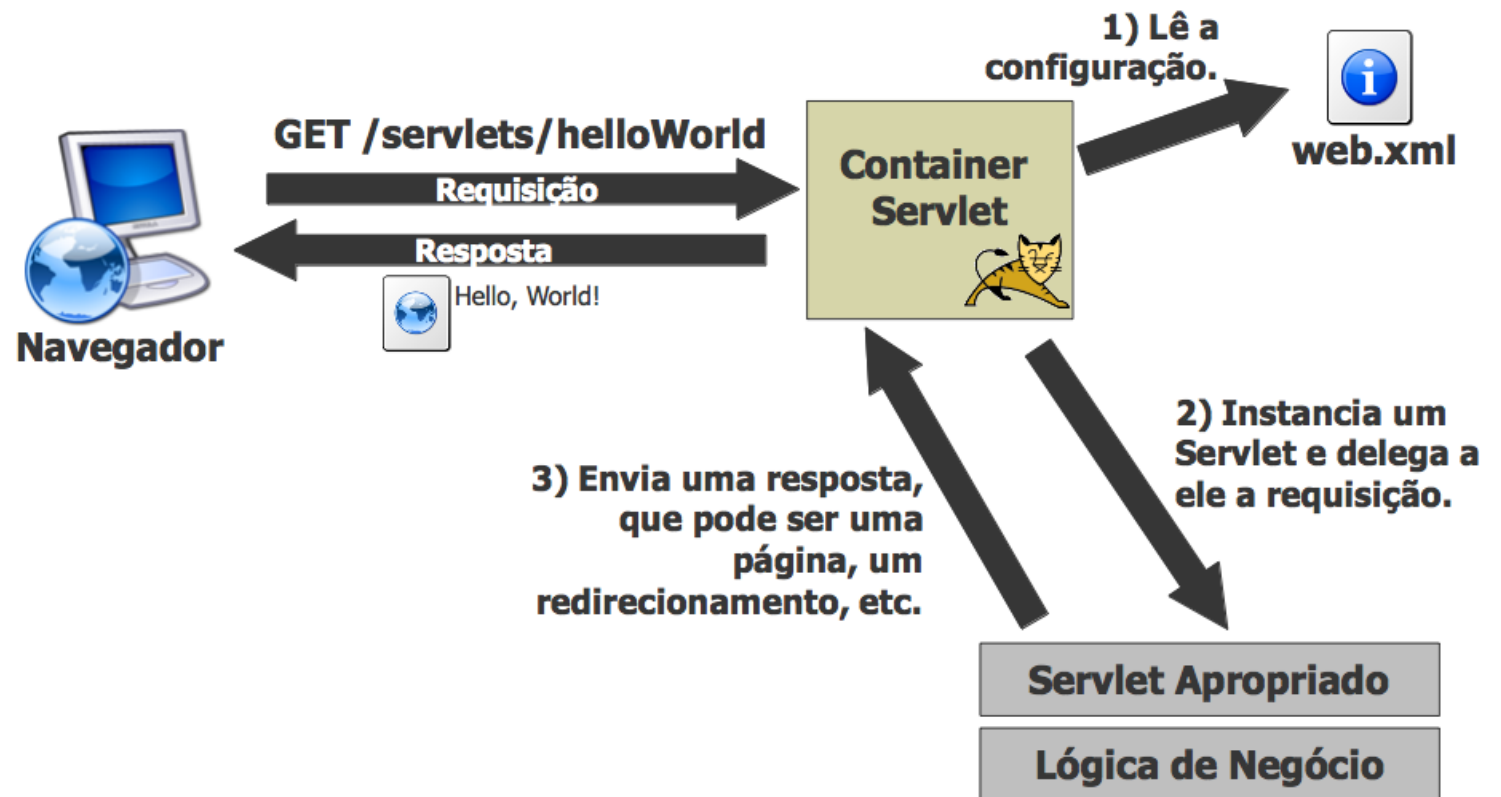


Portabilidade

São portáteis

Pode-se mover para outros sistemas operacionais sem dificuldades.

Servlets





Web Application Resource

As aplicações desktop possuem o padrão de distribuição JAR

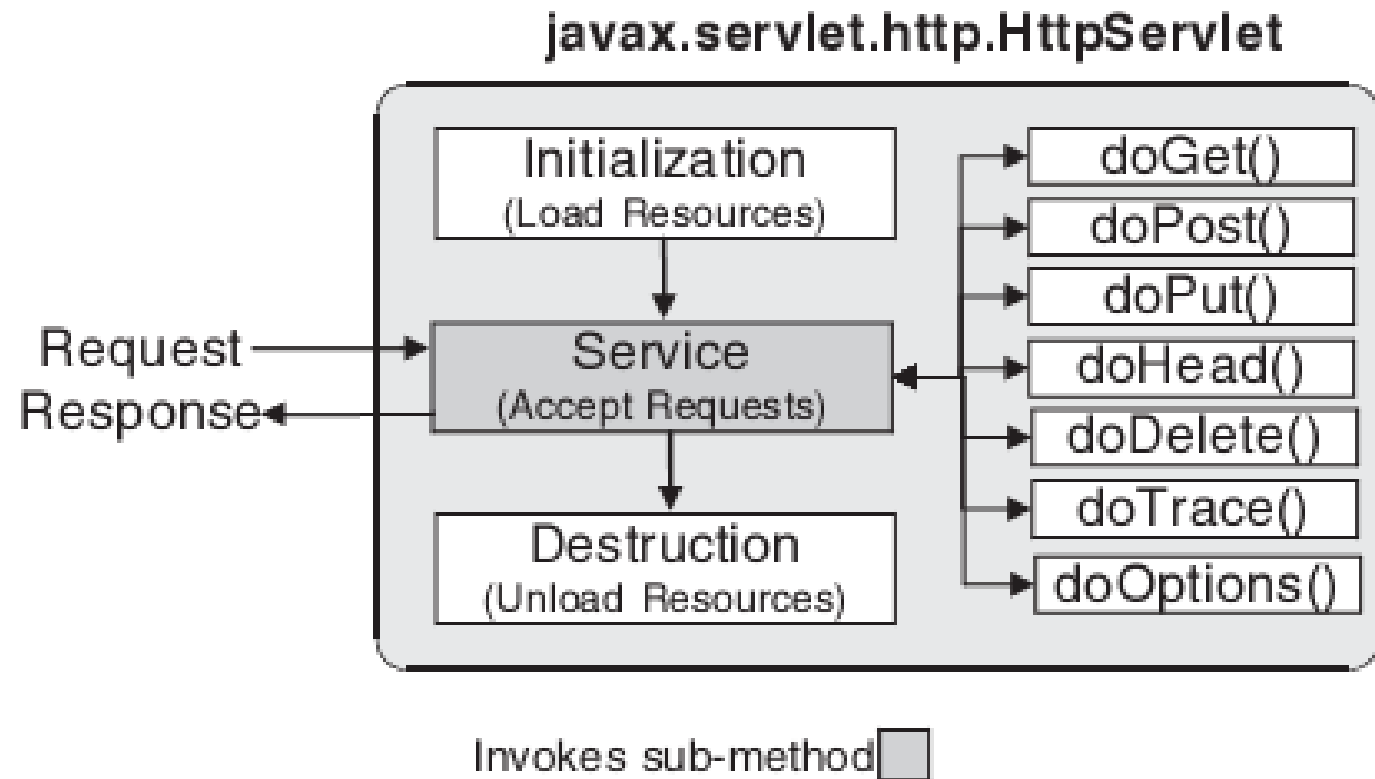
As aplicações web possuem o padrão WAR

Arquivo com extensão .war

Compactação da aplicação

Independência de plataforma

Servlet – HttpServlet



Facilidades Servlets 3.0

Definir mais de uma URL para acessar a Servlet utiliza-se: `urlPatterns`

```
@WebServlet(name = "MinhaServlet3", urlPatterns = {"/oi", "/ola"})  
public class OiServlet3 extends HttpServlet{  
    ...  
}
```

Servlet estando anotado com `@WebServlet()`, ele deve obrigatoriamente realizar um `extends javax.servlet.http.HttpServlet`

Atributo chamado `metadata-complete` da tag `<web-app>` no `web.xml`

- `True` as classe com anotação não serão procuradas pelo servidor de aplicação
- `False` as classes na `WEB-INF/classes` ou em algum `.jar` dentro `WEB-INF/lib` serão examinadas

Facilidades Servlets 3.0

- `@WebInitParam()`: declara parâmetros como padrão para acessá-los dentro de um vetor


```
@WebServlet(  
    name = "OiServlet3",  
    urlPatterns = {"/oi"},  
    initParams = {  
        @WebInitParam(name = "param1", value =  
            "value1"),  
        @WebInitParam(name = "param2", value =  
            "value2")  
    }  
)  
public class OiServlet3 {  
    ...  
}
```

Escrevendo - Response

```
PrintWriter out = response.getWriter();  
out.println("<html>");  
out.println("<body>");  
out.println("Helo world");  
out.println("</body>");  
out.println("</html>");
```



HttpSession

- Cada conexão de um cliente no servidor pode ou não possuir uma Sessão. Uma sessão é uma forma de armazenar temporariamente os dados do usuário (ex: login, carrinho de compras, etc.).
- 

HttpSession

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
```

```
    throws ServletException, IOException {
```

```
        HttpSession session = request.getSession(true); // se não existe, então cria
```

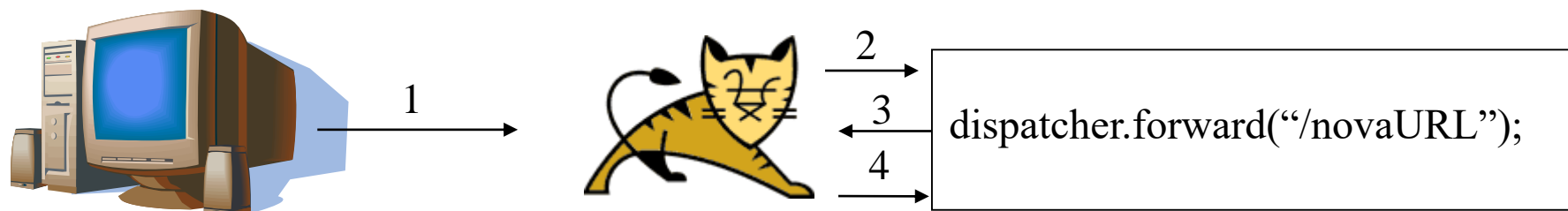
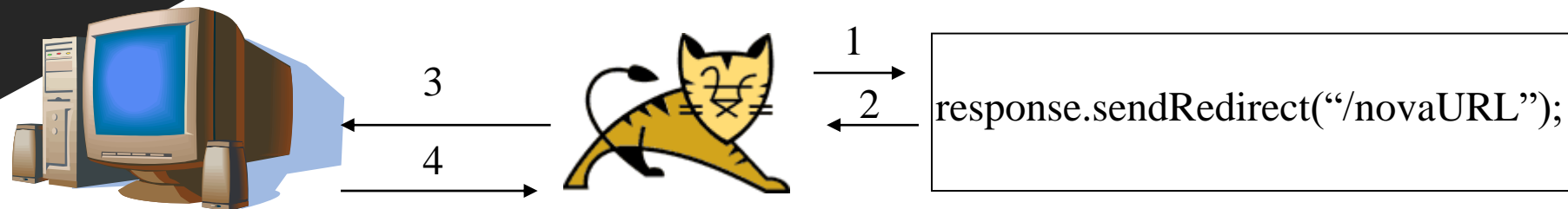
```
        session.setAttribute("usuario", new String("Nome Do Usuario"));
```

```
        session.getAttribute("usuario"); // retorna Nome Do Usuario
```

```
        session.removeAttribute("usuario");
```

```
    }
```


Forward x Redirect



Diretiva Page Importar pacotes

```
<%@ page import="java.util.Date" %>
<html>
<head>
<title>Hello World</title>
</head>
<body>
Hello World <br/>
Hoje é <%=new Date()%>
</body>
</html>
```

Diretiva Include

```
<html>
<head>
<title>Hello World</title>
</head>
<body>
<%@ include file="cabecalho.jsp" %>
Hello World <br/>
Hoje é <%=new Date()%>
</body>
</html>
```

Declarações JSP e Expressões JSP

```
<html>
<head>
<title>Contador</title>
</head>
<body>
<%! int count=0; %>
Esta página foi teve <%= ++count%>
    acessos.
</body>
</html>
```

Scriptlet

```
<html>
```

```
<head>
```

```
<title>Scriptlet</title>
```

```
</head>
```

```
<body>
```

```
<%
```

```
    String nome = "Tom Jobim";
```

```
    int quantidade = nome.lenght();
```

```
%>
```

```
O nome digitado foi <%= nome%> e ele possui  
    <%= quantidade%> letras.
```

```
</body>
```

```
</html>
```

Recursos

- Comentário JSP
 - `<% // xxx %>`
 - `<%-- xxx-- %>`
 - `<% /** xxx **/ %>`

conteudo.jsp

```
<%@ page contentType="text/html; charset=UTF-8"
language="java" %>
<html>
<head>
    <title>Java Server Pages</title>
</head>
<body>
<%@ include file="cabecalho.jsp" %>
<form action="recuperaInformacoes.jsp" method="post">
    Nome: <input type="text" name="nome"/><br/>
    Data de Nascimento: <input type="date"
name="dtaNascimento"/>
    <input type="submit" value="Enviar"/>
</form>
<%@ include file="rodape.jsp" %>
</body>
</html>
```

recuperaInformacoes.jsp

```
<%@ page import="java.text.SimpleDateFormat" %>
<%@ page import="java.util.Date" %>
<%@ page contentType="text/html; charset=UTF-8"
language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
<%@ include file="cabecalho.jsp" %>
Você informou os seguintes dados: <br/>
Nome : <%=request.getParameter("nome") %> <br/>
<%
    SimpleDateFormat formatarData = new
SimpleDateFormat("dd/MM/yyyy");
    Date data = new SimpleDateFormat("yyyy-mm-
dd").parse(request.getParameter("dataNascimento")) ;
%>
Data de Nascimento: <%=formatarData.format(data) %>
<br/>
<%@ include file="rodape.jsp" %>

</body>
</html>
```


Páginas restantes

cabecalho.jsp

```
<%@ page contentType="text/html; charset=UTF-8"
language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <h1>Cabeçalho</h1>
</body>
</html>
```

rodape.jsp

```
<%@ page contentType="text/html; charset=UTF-8"
language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <h2>Rodapé</h2>
</body>
</html>
```

Objetos Implícitos

São objetos que estão implicitamente disponíveis para serem utilizados nas páginas JSP

Servlet	page
	config
IO	request
	response
	out
Contextuais	session
	application
	pageContext
Exceções	exception

Java Bean

Objeto de uma classe Java

A classe tem que ter um construtor-padrão público

Todos os métodos devem ser nomeados com get e set.

- Ex.: setNome, getNome

Ações-padrão de Java Bean (objeto de uma classe Java)

- Declarando e inicializando um Bean
- `<jsp:useBean id="pessoa" class="unisul.progweb.Personalidade" scope="session">`
 - Se o Bean não existir ele é criado.
- Obtendo o valor da propriedade de um Bean
- `<jsp:getProperty name="pessoa" property="nome"/>`
- Armazenando um valor na propriedade de um Bean
- `<jsp:setProperty name="pessoa" property="cor"/>`

Exemplo

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
  <head>
    <title>Exemplo Ações-Padrão de Bean</title>
  </head>
  <body>
    <form action="telacor.jsp" method="post">
      <label for="idNome">Nome</label>
      <input type="text" name="txtNome" id="idNome"><br>
      <label for="idIdade">Idade</label>
      <input type="number" name="nrIdade" id="idIdade"><br>
      <input type="submit" value="Enviar">
    </form>
  </body>
</html>
```

Exemplo

```
<%@ page contentType="text/html; charset=UTF-8"
language="java" %>
<jsp:useBean id="pessoa"
class="br.unisul.progweb.Pessoalidade"
scope="session">
    <jsp:setProperty name="pessoa"
property="nome" param="txtNome"/>
    <jsp:setProperty name="pessoa"
property="idade" param="nrIdade"/>
</jsp:useBean>
<html>
<head>
    <title>Title</title>
</head>
<body>
<form action="resultado.jsp">
    <label for="idCor" >Cor:</label>
    <input type="text" name="txtCor"
id="idCor"/><br/>
    <input type="submit" value="Enviar"/>
</form>
</body>
</html>
```

```

<%@ page contentType="text/html; charset=UTF-8"
language="java" %>
<jsp:useBean id="pessoa"
class="br.unisul.progweb.Personalidade"
scope="session">
    <jsp:setProperty name="pessoa" property="cor"
param="txtCor"/>
</jsp:useBean>
<html>
<head> <title>Title</title> </head>
<body>
    Nome:
    <jsp:getProperty name="pessoa" property="nome"/>
    <br>
    Idade:
    <jsp:getProperty name="pessoa" property="idade"/>
    <br>
    <jsp:setProperty name="pessoa" property="cor"
param="txtCor"/>
    Cor:
    <jsp:getProperty name="pessoa" property="cor"/>
    <br>
    <%
        String cor = pessoa.getCor();
        if (cor.equalsIgnoreCase("branco")) {
    %>
    Pessoa calma
    <% } %>
</body>
</html>

```

Exemplo

MVC - Model-View-Controller

- A essência do MVC é separar a lógica do negócio da apresentação (interface), de modo que essa lógica possa servir para qualquer forma de apresentação (HTML, Swing, Web service, etc).
- Quando as aplicações misturam códigos de apresentação, de acesso a dados e de lógica ou regra de negócios, tornam-se difíceis de manter.
- Primeiro por que a interdependência dos componentes faz com que alterações no código se propaguem.
- Segundo, por que o forte acoplamento torna difícil ou impossível a reutilização das classes;
- Aumenta a complexidade do projeto.

Gerenciamento de Dependências

- Praticamente todo projeto utiliza bibliotecas para executar algum recurso específico
- Maven administra as bibliotecas a partir do arquivo pom.xml no qual contém a lista de dependências que um projeto utiliza.
- O Maven analisa e tenta localizá-las para disponibilizar para o projeto. Os lugares onde o Maven procura por dependências chamam-se repositórios.
- Na máquina local fica armazenado m2/repository normalmente no home do usuário
- O repositório normalmente é configurado para o endereço <http://repo.maven.apache.org/maven2/>



JPA

- O sistema é desenvolvido na grande maioria utilizando somente objetos, sem referências específicas de cada banco de dados.
- Oferece uma linguagem semelhante ao SQL chamada JPA Query, que possibilita escrever consultas e instruções de persistência.
- Principais Características
 - Persistência de POJOs (Plain Old Java Objects)
 - Permite elaboração de domínios ricos (herança, polimorfismo, navegabilidade, multiplicidade, associações e composições, etc)
 - Acessos sob demanda (Lazy)
 - Suporte à annotations - sem arquivos de mapeamento
 - Permite conectar frameworks de persistência (ex: Hibernate)

Metadados das Entidades

- Uma entidade, entretanto, possui um metadados. Ou seja, cada entidade no sistema deve ter informações sobre os dados que estão armazenando, com o objetivo de guiar o JPA no mapeamento adequado dos Objetos vs. Banco de Dados Relacional.
- Os metadados são, basicamente, realizados de duas formas:
 - Arquivos XML: utilizados nas versões mais antigas
 - Annotations: utilizados a partir do JPA 2
- As annotations vem para simplificar a implementação, reduzindo a quantidade de arquivos de configuração e facilitando a visualização e leitura do código.

Definindo uma PersistenceUnit

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
  version="2.0">
  <persistence-unit name="EmployeeService"
    transaction-type="RESOURCE_LOCAL">
    <properties>
      <property name="javax.persistence.jdbc.driver"
        value="com.mysql.jdbc.Driver" />
      <property name="javax.persistence.jdbc.url"
        value="jdbc:mysql://localhost:3306/dbprojetos?createDatabaseIfNotExist=true"/>
      <property name="javax.persistence.jdbc.user" value="APP"/>
      <property name="javax.persistence.jdbc.password" value="APP"/>
    </properties>
  </persistence-unit>
</persistence>
```

Construindo uma entidade

```
@Entity
public class Employee {
    @Id
    private String name;
    private long salary;

    public Employee() {}
    public Employee(int id) { this.id = id; }
    public gets();
    public sets();
}
```

EntityManager

- O entity manager compreende o mecanismo que, como o próprio nome diz, gerencia as entidades do sistema. Ou seja, é responsável por realizar operações de inserção, busca, alteração e remoção de entidades.
- É importante destacar:
 - Um EntityManager é criado a partir de uma fábrica, chamada de EntityManagerFactory;
 - Uma factory pode criar vários entity manager;
 - Uma entity manager gerencia entidades, indentificando-a pelo Nome e verificando se a mesma está devidamente registrada no JPA.

Obtendo uma EntityManager

- Uma persistence Unit é identificada no sistema por um nome e é responsável por estabelecer uma unidade de persistência, ou seja:
 - possui informações de conexão
 - possui tipo do banco de dados
 - schema
 - outras informações
- Cada módulo pode ter uma unidade de persistência, porém, usualmente utilizamos somente 1 PersistenceUnit em sistemas mais simples.
- Para obter uma entity manager, usamos a factory e passamos qual é a persistence unit responsável.

```
EntityManagerFactory emf =  
Persistence.createEntityManagerFactory("EmployeeService");
```

```
EntityManager em = emf.createEntityManager();
```

- Com a instância da EntityManager, é possível realizar operações com as Entidades

EntityManager

- A partir de uma EntityManager, é possível criar transações. Uma transação é utilizada principalmente quando se altera um conjunto de entidades e o valor somente deve ser efetivamente persistido se, no final de todas as operações, nenhum erro for encontrado.
- Informando que uma transação é iniciada:
`em.getTransaction().begin();`
- Informando que a transação foi realizada com sucesso e, portanto, os dados dever efetivamente ir para o banco de dados: `em.getTransaction().commit();`
- Informando que a transação toda deve ser cancelada:
`em.getTransaction().rollback();`

EntityManager

- Consultas (Queries)
- É possível escrever consultas com JPA utilizando uma linguagem semelhante ao SQL padrão Ansi, porém, deve ficar claro que:
 - Independentemente do fabricante do Banco de Dados, o SQL utilizado no JPA será o mesmo;
 - A consulta não é em uma tabela e sim em uma Entidade (classe do java);
 - O retorno geralmente é uma lista de Objetos. Este retorno depende do modo que o Select foi construido e, geralmente, a lista é de objetos do tipo da Entidade selecionada. Deve-se ter cuidado em Queries mais complexas.

Annotations do JPA

- Mapeamento de uma Tabela
- Annotation `@Entity`, informa ao JPA que uma determinada classe poderá ser manipulada por uma `EntityManager`,
 - Poderá ser persistida em um banco de dados definido pela `PersistenceUnit`.
- Por padrão, o JPA define o nome da tabela que será gerada no banco de dados em função do nome da classe.
 - Exemplo: se a classe se chamar `Usuario`, a tabela será a `"usuario"`. Muitas vezes, a base de dados já existe, ou ainda, o DBA prefere utilizar nomes personalizados para as tabelas, o que desconfiguraria o nome dos objetos em java, se o desenvolvedor optasse em seguir o padrão do DBA.

Annotations do JPA

- Annotation @Column
- É possível mapear o nome da coluna no banco de dados com um nome diferente do nome do atributo
- Definição da coluna
- Permite que o desenvolvedor faça a definição da coluna de forma manual, de acordo com seu banco de dados. Isso deve ser evitado, visto que o JPA vem para ser um framework independentemente do banco de dados utilizado.
- Define a coluna no banco de dados com o tipo DATE e que o valor default será através do SYSDATE.

Annotations do JPA

- length / Tamanho
- Define o tamanho da coluna. O número de caracteres ou dígitos que serão utilizados para armazenar o valor.
- nullable / pode ser nulo?
- Informa se o valor pode (ou não) ser nulo (vazio). Caso nullable=true, então o valor pode ser nulo. Caso contrário, é de preenchimento obrigatório.

Visão geral de mapeamento com JPA

- Relacionamento com valores "Únicos" (many-to-one, one-to-one)
- Um relacionamento onde um dos lados (roles) possui a cardinalidade (multiplicidade) 1(um) é chamado de relacionamento com valor único (single-valued association). Estes relacionamentos podem ser:
 - Muitos para um (many-to-one); e,
 - Um para um (one-to-one).
- Muitos para um
- O relacionamento Muitos para Um (many-to-one) indica que, dependendo da direcionalidade, em A teremos uma lista de B e em B, pode-se ter uma referência única para A.
- O JPA estabelece este relacionamento utilizando a annotation `@ManyToOne` e, de acordo com a direcionalidade, também a `@OneToMany`.
- A annotation `@ManyToOne` deverá ser utilizada na Entidade que terá apenas uma instância da outra relacionada a ela.

Visão geral de mapeamento com JPA

- Utilizando colunas de Junção
- É possível informar ao JPA qual é o nome da coluna que será gerada para armazenar o código (ID) da outra Entidade. No caso de não informar, por exemplo, seria criado uma coluna `turma_id` dentro da tabela `Aluno`.

Visão geral de mapeamento com JPA

- Associação Muitos para Muitos (many-to-many)
- Já é conhecido que um relacionamento muitos-para-muitos (many-to-many) necessita de tabelas entidades no banco de dados para fazer as associações de duas entidades que podem ter muitas instâncias da outra, ou vice-versa.
- O JPA oferece este tipo de mapeamento e gera automaticamente esta entidade responsável pela junção das entidades Origem e Destino.
- A annotation utilizada é a `@ManyToMany`. Esta annotation deverá ser colocada primeiramente na entidade "dona" do relacionamento e, em seguida - e se for necessário -, na entidade que faz o relacionamento inverso, definindo o parâmetro `mappedBy`.

Visão geral de mapeamento com JPA

- Ajustando a Tabela de Junção
- Um relacionamento muitos para muitos resulta numa terceira tabela responsável pelo relacionamento das chaves primárias das entidades Origem e Destino.
- Na tabela resultante deste relacionamento serão encontrados somente duas colunas: KEY de origem e KEY de destino.
- No caso de desejar fazer ajustes nesta tabela resultante, é possível utilizando a annotation `@JoinTable`, na classe "dona" do relacionamento.