

TI-510 Java Enterprise Edition Básico

ANTONIO CARVALHO - TREINAMENTOS

Disciplina

TI-510 Java Enterprise Edition Básico

Objetivo : Ensinar a criar sistemas corporativos, distribuídos via Web utilizando tecnologias convencionais da plataforma Java EE existentes no container Web como Servlets, JSPs e Filtros.

Carga horária : 40 horas

Professor

Antonio Rodrigues Carvalho Neto
antoniorcn@hotmail.com

Ao enviar emails favor colocar no cabeçalho:
<TURMA>-<MATRICULA>-<NOME>-<Assunto>

Ementa

Arquitetura Enterprise Edition

Protocolo HTTP

Estrutura da Aplicação JEE

Servlets

Padrão de Projetos Model View Controller (MVC)

JSP

Recursos Utilizados

Para baixar o **Eclipse EE** acesse o site --> www.eclipse.org

Para baixar o **Java** acesse o site -->

<http://www.oracle.com/technetwork/java/javase/downloads/java-se-jdk-7-download-432154.html> procure a versão 7 do JDK

Para baixar o **Apache Tomcat** acesse o site → <http://tomcat.apache.org/>

Arquitetura Enterprise Edition

O que é Java Enterprise Edition ?

Java EE é um conjunto de especificações que suportam a criação de sistemas corporativos. Como especificação a Oracle deixa que a JEE seja implementada nos servidores de aplicação, existentes no mercado.

O Java EE reúne uma série de especificações independentes

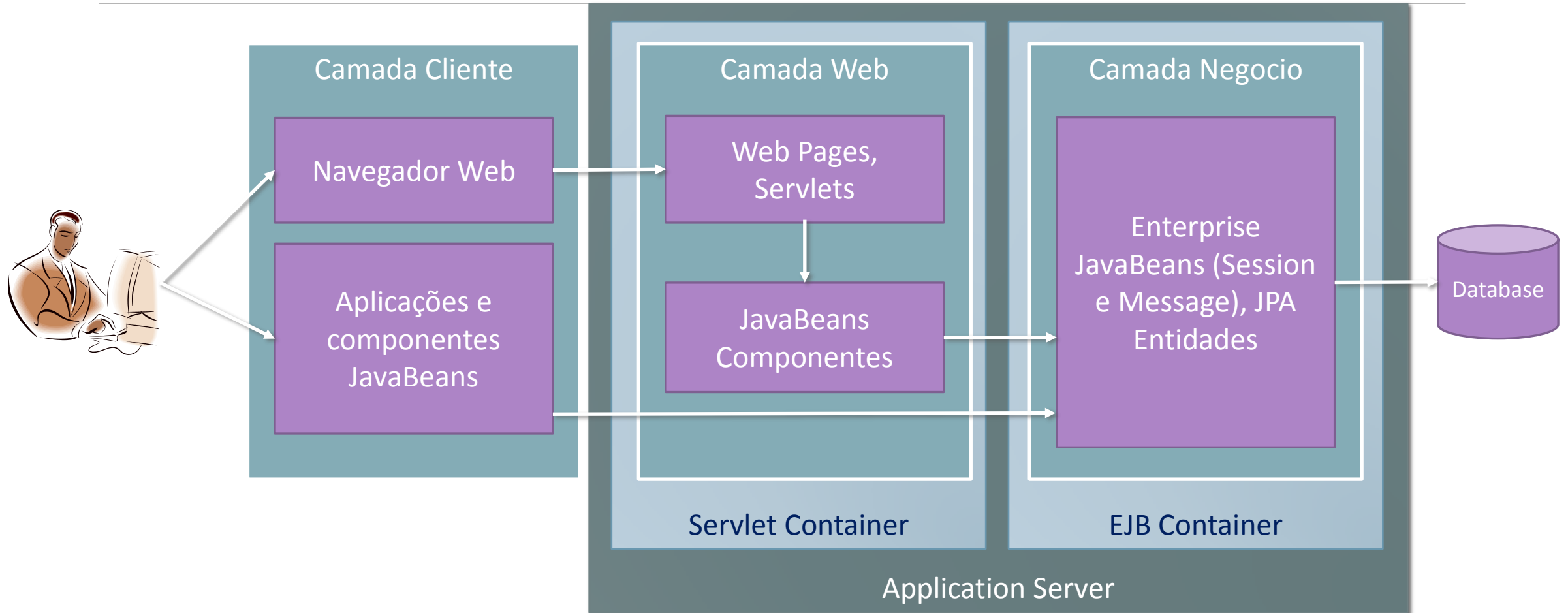
- **Java Servlet 3.0 and JavaServer Pages 2.2/Expression Language 2.2** - JSR 315 and JSR 245
- **JavaServer Faces 2.0** - JSR 314
- **Managed Beans** – Container leve de objetos (POJO)
- Contexts and Dependency Injection for Java - JSR 299 and JSR 330 – Define um conjunto de contextos fornecidos pelos containers Java EE
- Java Transaction API (JTA) – Prove uma interface padrão para demarcar transações
- Java Message Service API (JMS) – Permite aos componentes Java EE, criar, enviar, receber e ler mensagens
- **Java Persistence API (JPA) 2.0** - JSR 317 – Solução padrão para persistência em Java utilizando ORM (Object Relational Mapping)
- Bean Validation 1.0 JSR 303 – Prove um metamodelo e uma API para fazer a validação de dados em componentes JavaBeans
- Java Connector Architecture (JCA) – Possibilita a criação de conectores para acessar sistemas EIS
- JavaMail API – API que permite o envio de emails
- Enterprise JavaBeans (EJB3) 3.1 - JSR 318
- Java API for RESTful Web Services (JAX-RS) 1.1 - JSR 311 – Prove API para desenvolvimento de Web Services REST

O que é Java Enterprise Edition ?

Especificações que são oriundas do JSE

- **Java Database Connectivity (JDBC)** – Possibilita a execução de comandos SQL através do Java
- **Java Naming and Directory Interface (JNDI)** – Possibilita as aplicações utilizarem serviços de nomes e diretórios
- **Java API for XML Processing (JAXP)** – Suporta o acesso a documentos XML usando o DOM, SAX e XSLT
- **Java Architecture for XML Binding (JAXB)** – Permite a conversão de schemas XML em programas Java
- **Java API for XML-Based Web Services (JAX-WS) 2.2 - JSR 224** – Suporta Web Services através do JAXB
- **Java Authentication and Authorization Service (JAAS)** – Permite que o sistema faça autorização e autenticação externamente

Arquitetura JEE



Arquitetura JEE

Com esta arquitetura a forma de interação com a aplicação será ligeiramente diferente.

O código que é executado não está mais na mesma máquina do usuário, ele se encontra agora em um servidor.

O Cliente (navegador do usuário) e o Servidor trocarão apenas informações através do processo de requisições.

- O Cliente envia uma **requisição**
- O Servidor processa esta **requisição**
- O Servidor envia uma **resposta** para o Cliente
- O Cliente exibe esta **resposta** na tela do usuário

Arquitetura JEE

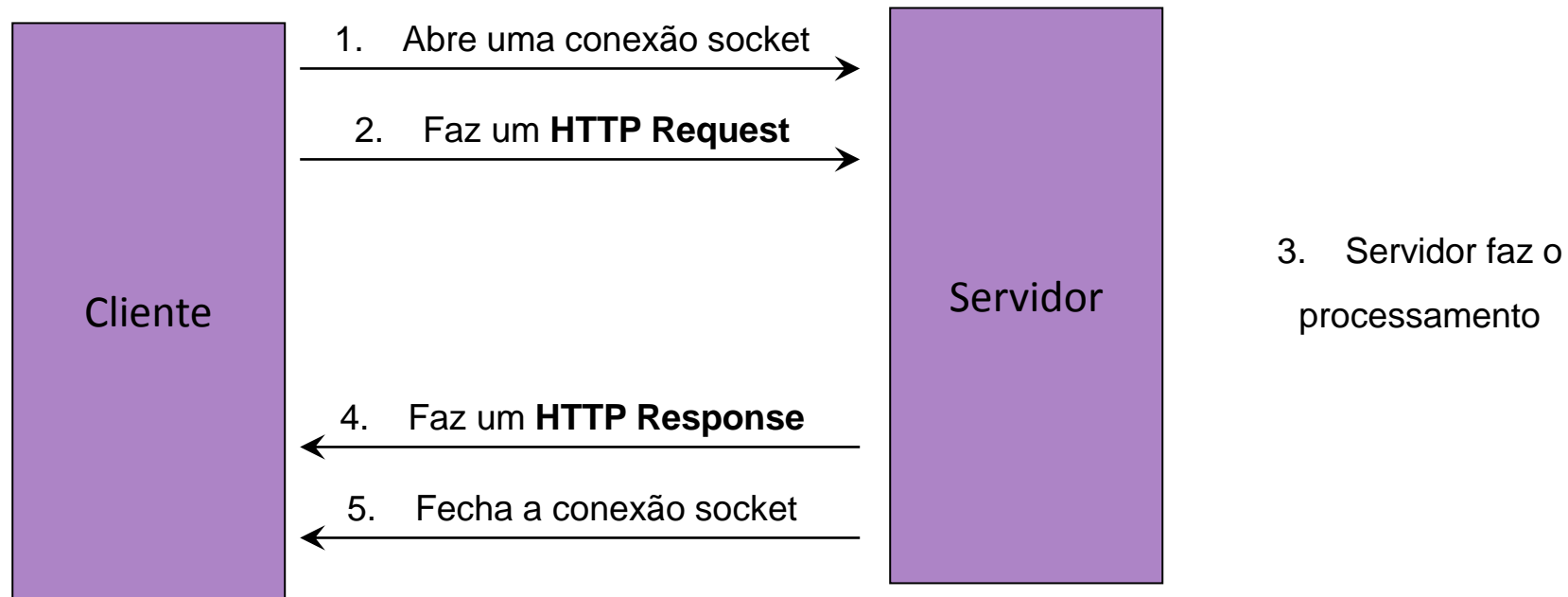
Estas requisições e respostas serão baseadas em um protocolo chamado HTTP (Hyper Text Transfer Protocol).

No momento que o cliente faz a requisição ele precisará passar todas as informações que o servidor necessita conhecer para executar a atividade, como exemplo, qual rotina deve ser executada, e quais dados serão informados para esta rotina.

Nota: Os dados somente serão passados nesta requisição, quando o servidor não puder obtê-los do banco de dados.

Arquitetura JEE

Funcionamento do processo de **requisição e resposta**



JEE - Exercícios

- Instale o Apache Tomcat
- Inicie e pare o application server através do Tomcat Monitor, do Serviço do Windows e através da linha de comando.
- Verifique o resultado das inicializações na LOG do sistema

Protocollo HTTP

Protocolo HTTP

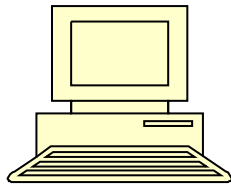
O protocolo HTTP consiste em textos enviados na requisição e na resposta

A requisição, chamada de **HTTP Request** é composto pelo **método** invocado, **valores** do cabeçalho, uma linha em branco e as informações organizados como **chave** e **valor**.

Já a resposta chamada de **HTTP Response** é composto pelas informações de cabeçalho (**Header**), e o corpo (**Body**) usualmente um texto em HTML .

Protocolo HTTP

Envia **HTTP Request**

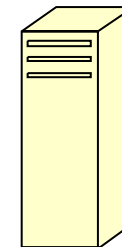


Recebe **HTTP Response**

```
GET /TesteJEE/music.html HTTP/1.1
Host: localhost:8080
Connection: keep-alive
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/
webp,*/*;q=0.8
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/47.0.2526.111 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: pt-BR,pt;q=0.8,en-US;q=0.6,en;q=0.4
```

} HTTP
Header

} HTTP
Body



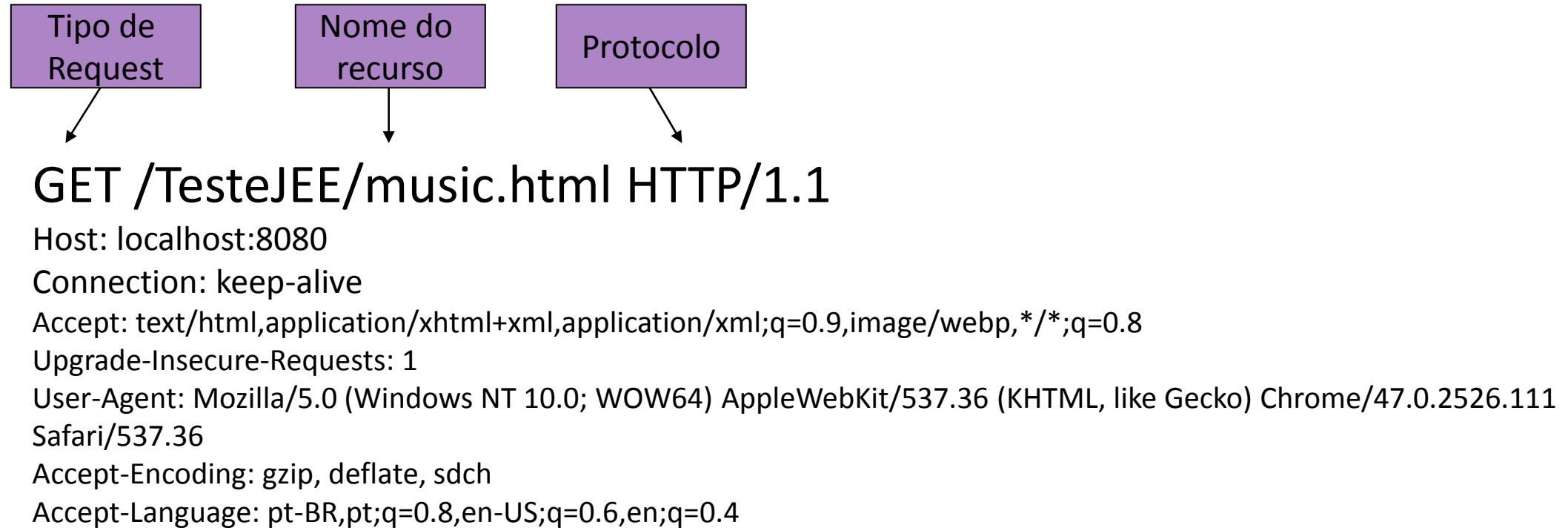
```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Accept-Ranges: bytes
ETag: W/"661-1453209477644"
Last-Modified: Tue, 19 Jan 2016 13:17:57 GMT
Content-Type: text/html
Content-Length: 661
Date: Tue, 19 Jan 2016 13:19:57 GMT
```

} HTTP
Header

} HTTP
Body

```
<html>
<body>
...
<tr>
  <td>Nome Musica : </td>
  <td><input type="text" name="nome"/></td>
</tr>
...
</body>
</html>
```


Protocolo HTTP (Request)



Protocolo HTTP (Request)

As requisições podem ser feitas de tres formas

- **GET** (HTTP 1.0) – Obtém informações do servidor
- **HEAD** (HTTP 1.0) – Solicita apenas o cabeçalho da resposta
- **POST** (HTTP 1.0) – Postam dados para o servidor
- **PUT** (HTTP 1.0) – Permite o envio de um arquivo para o servidor
- **OPTIONS** (HTTP 1.1) – Retorna a versão do protocolo HTTP e os métodos disponíveis
- **DELETE** (HTTP 1.1) – Apaga um recurso do servidor
- **TRACE** (HTTP 1.1) – Faz com que a resposta mostre o mesmo texto enviado na requisição
- **CONNECT** (HTTP 1.1) – Permite a criação de um túnel HTTP

Protocolo HTTP (Response)

Protocolo

Reposta
(Status code)

HTTP/1.1 200 OK

Server: Apache-Coyote/1.1

Accept-Ranges: bytes

ETag: W/"661-1453209477644"

Last-Modified: Tue, 19 Jan 2016 13:17:57 GMT

Content-Type: text/html

Content-Length: 661

Date: Tue, 19 Jan 2016 13:19:57 GMT

<!DOCTYPE html>

<html>

<head>

<meta charset="ISO-8859-1">

<title>Insert title here</title>

</head>

<body>

<form method="GET" action="/MusicServlet">

<table>

<tr>

<td>Nome Musica : </td>

<td><input type="text" name="nome"/></td>

</tr>

<tr>

<td>Nome Artista : </td>

<td><input type="text" name="artista"/></td>

</tr>

<tr>

<td>Tempo da musica : </td>

<td><input type="text" name="tempo"/></td>

</tr>

<tr>

<td><input type="submit" name="cmd" value="Adicionar"/></td>

<td><input type="submit" name="cmd" value="Procurar"/></td>

</tr>

</table>

</form>

</body>

</html>

Protocollo HTTP (Response)

HTTP Status Codes

1xx Informational

2xx Success

3xx Redirection

4xx Client Error

5xx Server Error

[http://en.wikipedia.org/wiki/List of HTTP status codes](http://en.wikipedia.org/wiki/List_of_HTTP_status_codes)

Protocolo HTTP

Envia **HTTP Request**

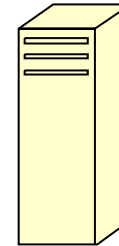


Recebe **HTTP Response**

```
GET
/TesteJEE/MusicServlet?nome=Enter+Sandman&artista=Metallica&tempo=6&cmd=A
dicionar HTTP/1.1
Host: localhost:8080
Connection: keep-alive
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/47.0.2526.111 Safari/537.36
Referer: http://localhost:81/TesteJEE/music.html
Accept-Encoding: gzip, deflate, sdch
Accept-Language: pt-BR,pt;q=0.8,en-US;q=0.6,en;q=0.4
```

HTTP
Header

HTTP Body

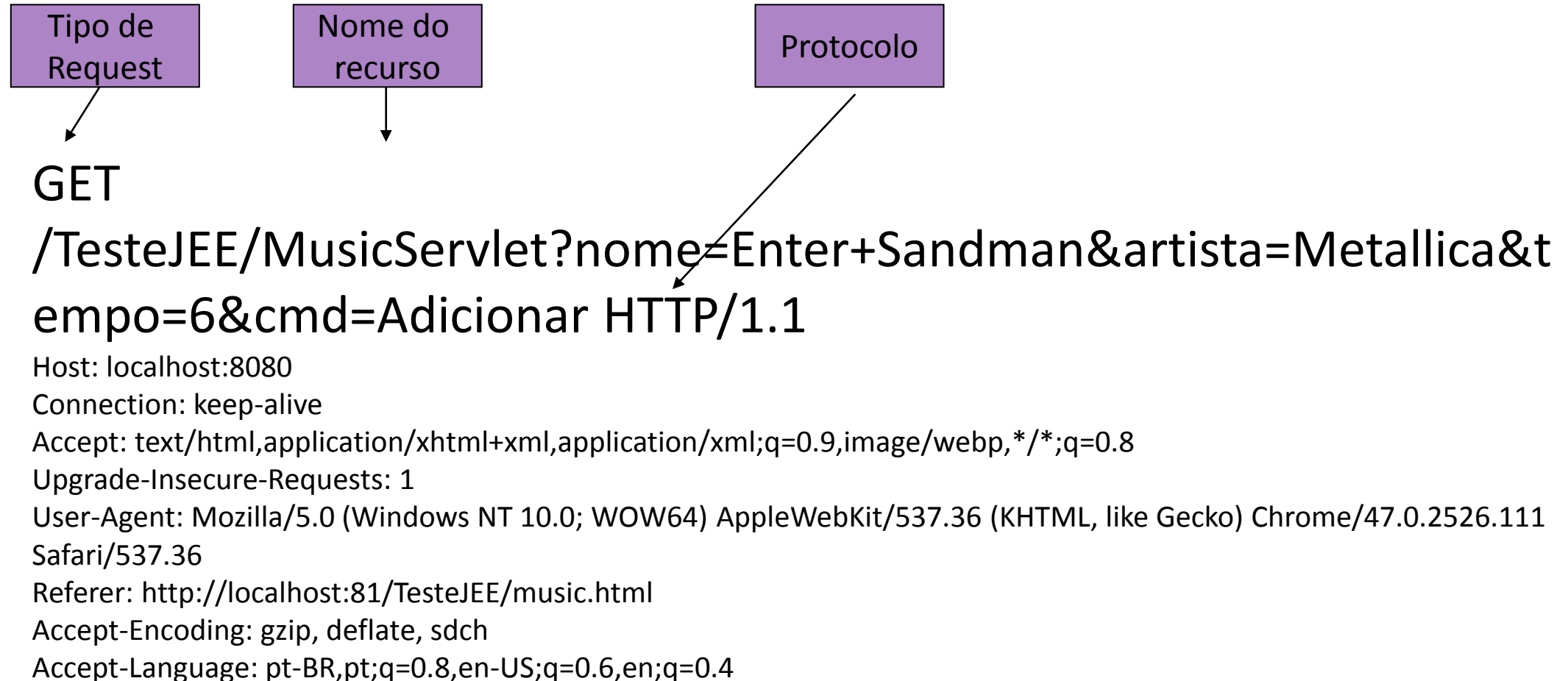


```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Length: 0
Date: Tue, 19 Jan 2016 13:44:53 GMT
```

HTTP
Header

HTTP
Body

Protocolo HTTP (Request)



Protocolo HTTP (Request)

Tipo de
Request

Nome do
recurso

Protocolo

POST /TesteJEE/MusicServlet HTTP/1.1

Host: localhost:8080

Connection: keep-alive

Content-Length: 58

Cache-Control: max-age=0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8

Origin: http://localhost:81

Upgrade-Insecure-Requests: 1

User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.111 Safari/537.36

Content-Type: application/x-www-form-urlencoded

Referer: http://localhost:81/TesteJEE/music.html

Accept-Encoding: gzip, deflate

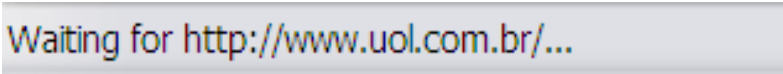
Accept-Language: pt-BR,pt;q=0.8,en-US;q=0.6,en;q=0.4

nome=Enter+Sandman&artista=Metallica&tempo=6&cmd=Adicionar

JEE - Exercícios

Ao colocar uma URL no navegador ex. www.uol.com.br o navegador envia um **HTTP Request** para o servidor www.uol.com.br.

Ele ficará aguardando o servidor retornar a resposta (**HTTP Response**), e mostrará na barra de status a seguinte mensagem



Waiting for [http://www.uol.com.br/...](http://www.uol.com.br/)

Assim que o servidor retornar com a resposta o navegador extrai o HTML da resposta e mostra na tela com os textos estáticos.

Se houver tags HTML apontando para imagens, então o navegador fará novos acessos ao servidor para fazer o download dessas imagens.

JEE - Exercícios

Abra a view TCP/IP Monitor do Eclipse e crie uma nova escuta na porta 8085, apontando para o servidor

<http://autocomplete.wunderground.com/aq>

Faça o navegador acessar o endereço `http://127.0.0.1:8085`

Faça agora um acesso pelo navegador passando um parâmetro no request indicando o nome da cidade

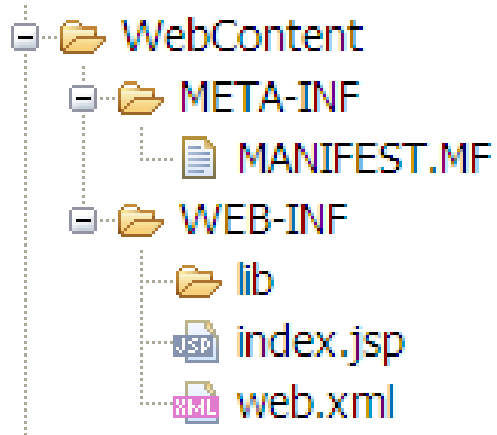
<http://autocomplete.wunderground.com/aq?query=Washington+DC>

Consulte o TCP/IP monitor e verifique as informações no request e no response

Estrutura da Aplicação JEE

Estrutura da Aplicação JEE

A aplicação Enterprise Edition, possui uma estrutura de diretórios que visa separar os arquivos que podem ser acessados pelos usuários, daqueles que somente são acessados pelo servidor.



- **WebContent** é a raiz da aplicação nela ficam os elementos visíveis aos usuários.
- **META-INF** contém os arquivos de metadados, são arquivos que indicam como será feito o deploy da aplicação EE. (Visível apenas para o servidor)
- **WEB-INF** Todo o conteúdo desta pasta fica escondido do usuário. Dentro do WEB-INF existem duas outras pastas:
 - **lib** contém os arquivos .JAR e .ZIP usados como bibliotecas para a aplicação
 - **classes** contém os códigos fontes dos servlets e os byte codes.

Estrutura da Aplicação JEE

Além da estrutura de pastas existe um arquivo importante que delimita as configurações da aplicação Web.

O arquivo **web.xml** tem um descritivo de como a aplicação se comportará quando for instalada no servidor.

Este arquivo fica localizado na pasta **WEB-INF** dentro do **WebContent**

JEE - Exercícios

Crie um projeto **Web Dynamic** no Eclipse.

- Preencha o nome do projeto como **MeuPrimeiroProjetoWeb**
- Defina o rootcontext como sendo **ProjetoWeb**
- Crie um arquivo **index.html** contendo apenas o a frase "Meu Primeiro Projeto EE"
- Verifique a estrutura de diretórios criada para este projeto
- Exporte o projeto como **.WAR** e faça o *deploy* no Tomcat.

Servlets

O que é um Servlet

Servlet é uma classe Java que implementa a interface `javax.servlet.Servlet`. Um objeto dessa classe é instanciado uma vez quando o container web é iniciado.

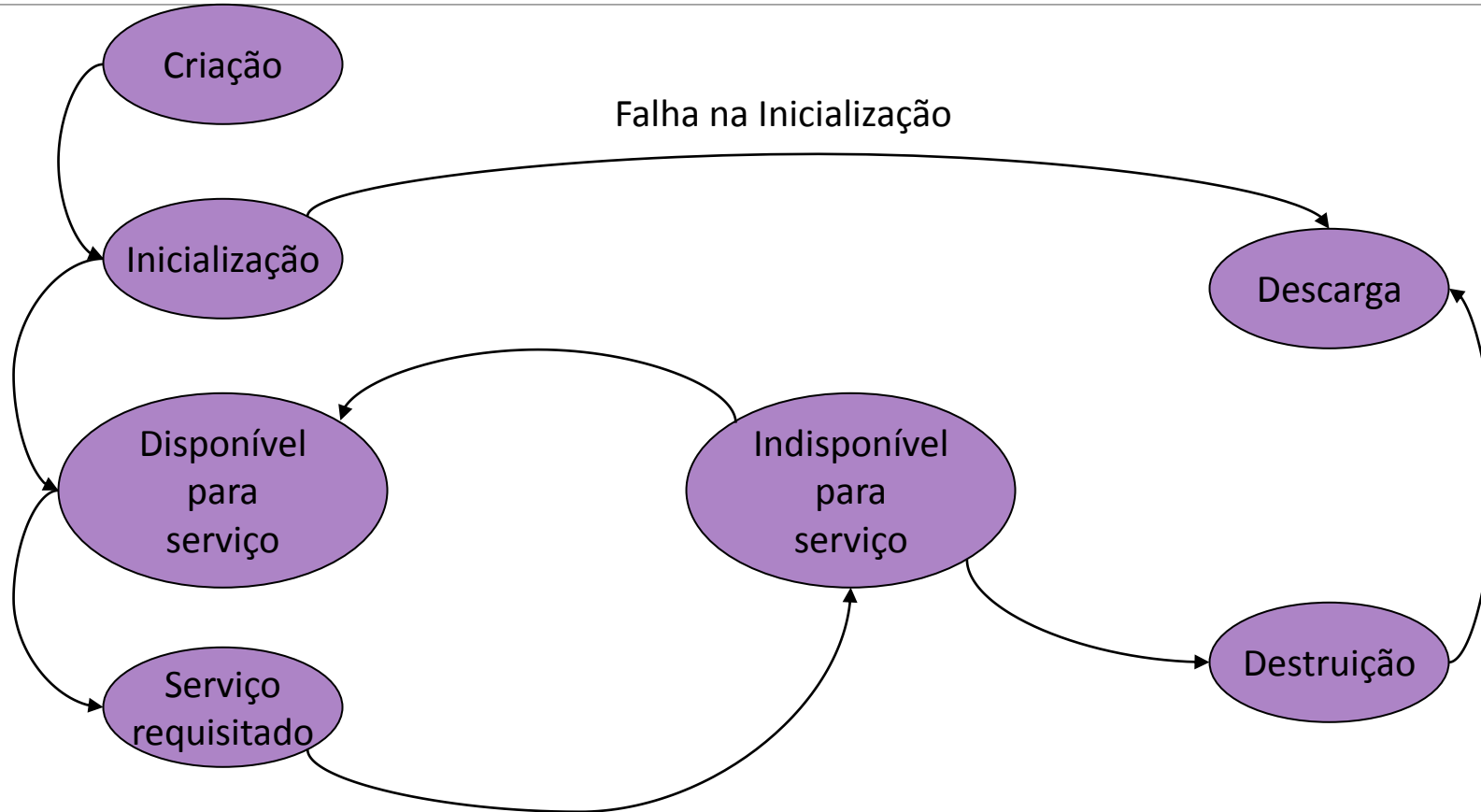
Através do **servlet** é possível hospedar um código que será acessado através do processo de **Request** e **Response**

Ciclo de vida de um Servlet

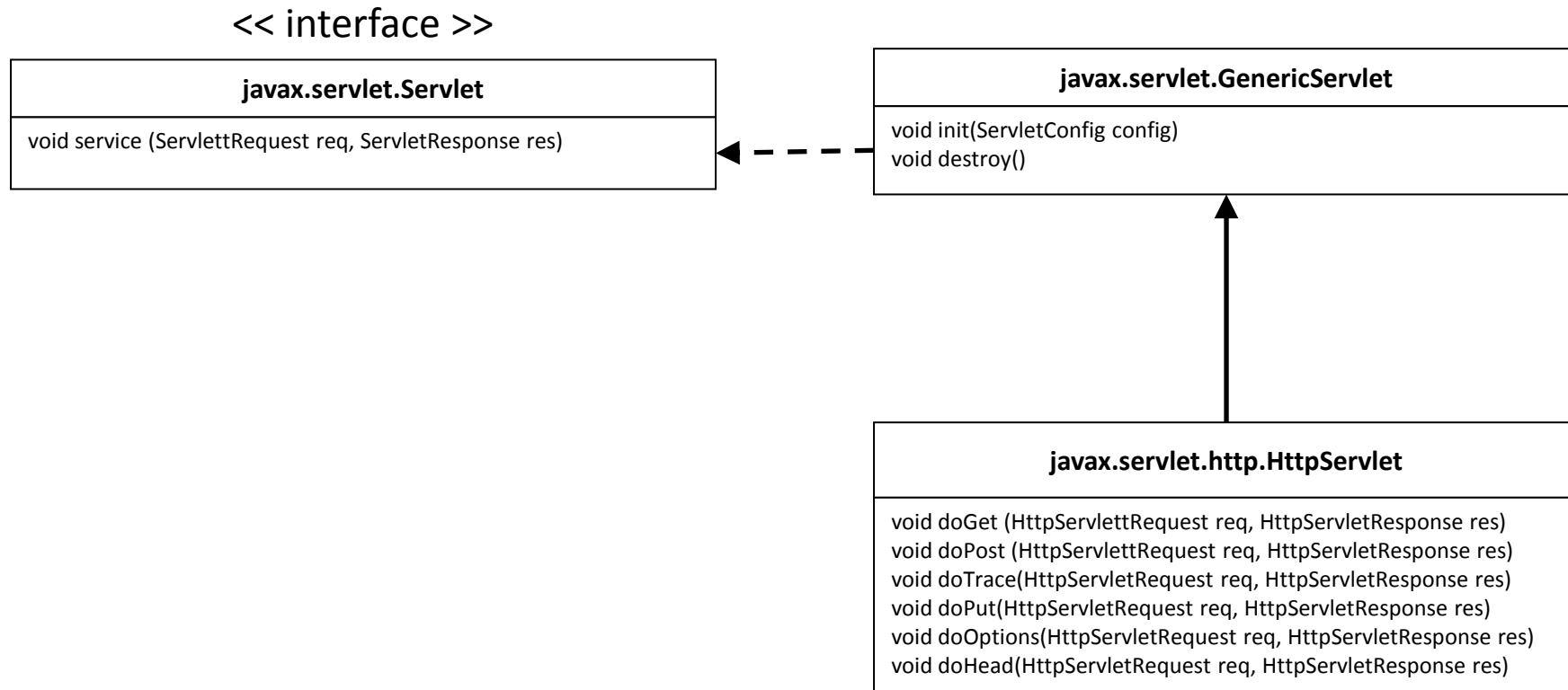
O ciclo de vida de um Servlet é controlado pelo Contêiner de Servlets, quando uma requisição é enviada ao Servlet o contêiner executa as seguintes atividades:

1. Caso ainda não exista uma instância do Servlet
 - a) Carrega a classe do Servlet
 - b) Cria uma nova instância do Servlet
 - c) Invoca o método **init()** para inicializar a instância do Servlet com os dados do Contexto
2. Aciona o método **service()** do Servlet (ver métodos **doGet**, **doPost** e outros da classe **HttpServlet**), passando objetos do tipo **ServiceRequest** e **ServiceResponse**.

Ciclo de vida de um Servlet



Hierarquia de um Servlet



Métodos de um Servlet

```
doGet(HttpServletRequest req, HttpServletResponse res)
doPost(HttpServletRequest req, HttpServletResponse res)
doTrace(HttpServletRequest req, HttpServletResponse res)
doPut(HttpServletRequest req, HttpServletResponse res)
doOptions(HttpServletRequest req, HttpServletResponse res)
doHead(HttpServletRequest req, HttpServletResponse res)
init(ServletConfig config)
destroy()
```

HttpServletRequest

O objeto do tipo **HttpServletRequest** contém todas as informações provenientes do cliente quando o Servlet foi acionado.

Exemplo de informações:

- Content Type, tipo da informação que está sendo enviada pelo cliente
 - `getContentType() : String;`
- Headers, acessa os elementos do cabeçalho da requisição
 - `getHeaderNames() : Enumeration<String>`
 - `getHeader(String name) : String;`
- Atributos e Sessão, acessa os objetos disponíveis no escopo de request e session
 - `getAttributeNames() : Enumeration<String>`
 - `getAttribute(String name) : Object;`
 - `getSession() : HttpSession`
- Parameters, acessa os parametros enviados pelo cliente
 - `getParameterNames() : Enumeration<String>`
 - `getParameter(String name) : String;`
 - `getParameterValues(String name) : String[]`

HttpServletResponse

O objeto do tipo **HttpServletResponse** contém todas as informações que o servlet pode encaminhar de volta para o cliente.

Exemplo de informações:

- Headers, acessa e define os elementos do cabeçalho de resposta
 - `getHeaderNames()` : `Collection<String>`
 - `getHeader(String name)` : `String`;
 - `setHeader(String name, String value)`;
- Demais métodos que definem informações da resposta
 - `setContentLength(int size)`;
 - `setContentType(String type)`;
 - `setStatus(int status)`;
 - `sendRedirect(String url)`;
- `Writer`, permite o envio de dados para o corpo da resposta
 - `getWriter()` : `PrintWriter`

Criando um Servlet

Para criar um Servlet é preciso seguir as etapas abaixo:

1. Criar uma classe que herde da classe `HttpServlet`.
2. Sobrescreva os métodos desejados (`doGet`, `doPost`, etc...)
3. Escreva o código neste método
4. Coloque a anotação **@WebServlet** na classe para mapear o acesso a uma URL
5. Coloque o projeto no servidor e acesse a URL pelo navegador

Servlet - Exercícios

Crie um banco de dados com uma tabela chamada **tbl_usuarios** conforme a figura ao lado.

Crie um **servlet** chamado **CheckLogin** que ira implementar o método **doGet()**. A requisição irá conter dois parâmetros um chamado **txtLogin** e outro chamado **txtSenha**

Faça com que o servlet pesquise na tabela **tbl_usuarios** por um registro cuja o campo **usuario** seja igual ao conteúdo da variável **txtLogin**. Caso não seja encontrado nenhum registro, então o servlet deverá exibir uma mensagem para o usuário final mostrando "Usuário informado não existe".

Caso este usuário seja encontrado o servlet deverá comparar o campo **password** com o conteúdo da variável **txtSenha**. Se o campo **password** for diferente o servlet deverá exibir uma mensagem para o usuário final informando "A senha está incorreta".

tbl_usuarios
USUARIO CHAR (20) PK
NOME VARCHAR (100)
EMAIL VARCHAR (100)
TELEFONE VARCHAR (20)
PASSWORD VARCHAR (20)
PERFIL VARCHAR (20)

Padrão de Projetos

Model View Controller

(MVC)

Padrão de projetos Model View Controller (MVC)

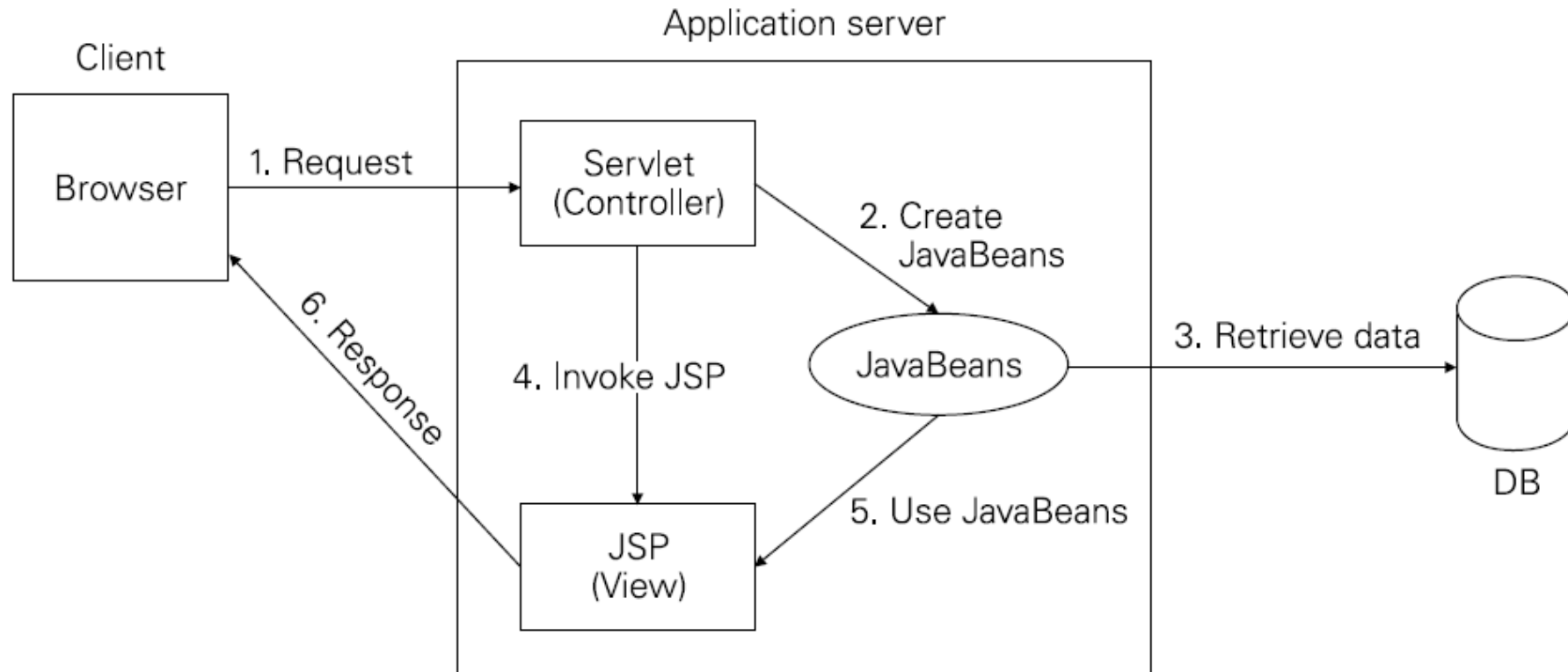
A arquitetura cliente, servidor permite múltiplas possibilidades para implementar aplicações distribuídas.

Para facilitar o desenvolvimento e a manutenção dessas aplicações, surgiu um padrão chamado MVC (Model View Controller) que divide as funcionalidades da aplicação corporativa em componentes de forma que cada um deles represente um domínio específico.

Em outras palavras ao usar o MVC a aplicação será dividida da seguinte forma:

- **Model**, são componentes responsáveis para lidar com a manipulação dos dados, regras de negócios e atualização do banco de dados.
- **View**, são componentes voltados a mostrar os dados para o usuário final sem se preocupar com as regras de negócio dos dados, nem como estes dados serão extraídos ou inseridos no banco de dados.
- **Controller**, este último tipo de componente interage com o usuário acionando a view solicitada e o Modelo (Objeto) que será mostrado nesta view. O controller, aguarda uma nova solicitação do usuário para decidir qual view será acionada para responder esta solicitação.

Arquitetura MVC



View

Os componentes de visualização dos dados são feitas normalmente em uma linguagem própria que possua comandos fáceis para exibição dos dados.

Exemplo Java Server Pages (JSP) que é editável utilizando uma linguagem XHTML ou HTML e que aceita comandos de Java.

O JSP pode ser utilizado em ferramentas de Web Design como DreamWeaver, NetObjects e outras pelo fato de ser uma linguagem baseado em tags.

Model

Os componentes de modelos normalmente são classes JavaBeans ou classes de acesso de dados que possuem métodos para gravar, ler e atualizar a si próprio no banco de dados.

A classe de modelo contém atributos de uma determinada entidade e estes atributos são acessados apenas por meio de métodos **gets** e **sets**.

Control

O componente de controle interage com o usuário recebendo a sua requisição. A partir deste recebimento ele direciona qual modelo deverá ser acionado e qual view (pagina JSP) será disponibilizada para responder a solicitação.

Normalmente o Controller é uma classe do tipo **servlet**.

JSP

Conteúdo Dinâmico x Estático

Toda requisição feita pela Web tem como intuito obter algum dado como resposta do servidor.

Os dados provenientes do servidor podem ser classificados em **estáticos** ou **dinâmicos**.

Conteúdo Estático

Estáticos, são os dados que não sofrem transformação mediante a requisição do usuário, ou seja o dado será sempre o mesmo não variando a cada requisição.

- Como exemplo de informações estáticas há arquivos de imagens (.gif, .jpeg, .bmp e outros), páginas com extensão .html (que não usam tecnologia AJAX, arquivos de script com extensão .css, .js.

Conteúdo Dinâmico

Dinâmicos, são os dados que sofreram processamento antes de serem encaminhados de volta ao usuário. Estes dados podem ser diferentes a cada requisição.

- Como exemplo de conteúdo dinâmico há alguns códigos em linguagens ou scripts como PHP, JSP, ASP, CGI. Pois arquivos deste tipo serão primeiro executados e podem alterar os dados a serem retornados para o usuário.

Java Server Pages

Java Server Pages (JSP) utilizam dados do modelo, elementos personalizados, linguagens de criação de scripts e objetos Java do lado do servidor para retornar um conteúdo dinâmico a um cliente geralmente dentro de um navegador Web. Uma JSP é uma combinação de sintaxe HTML e sintaxe Java executadas em tempo de execução para criar dinamicamente conteúdo aos clientes baseados na Web. (ALLEN 2002).

Vantagem ao usar o JSP

- Conteúdo estático separado do conteúdo dinâmico. Normalmente o conteúdo estático está localizado nas TAGs XHTML ou HTML, e o conteúdo dinâmico encontra-se nas TAGs JSP e nos scripts em Java.
- Fácil adesão ao modelo MVC.
- Reuso de componentes e bibliotecas de TAG, agilizando o trabalho.
- Utiliza a tecnologia de Servlets.
- Suportada por ferramentas de produção de páginas.

Modelo de execução do JSP

O JSP é executado no WebContainer

JSP é convertido em um Servlet

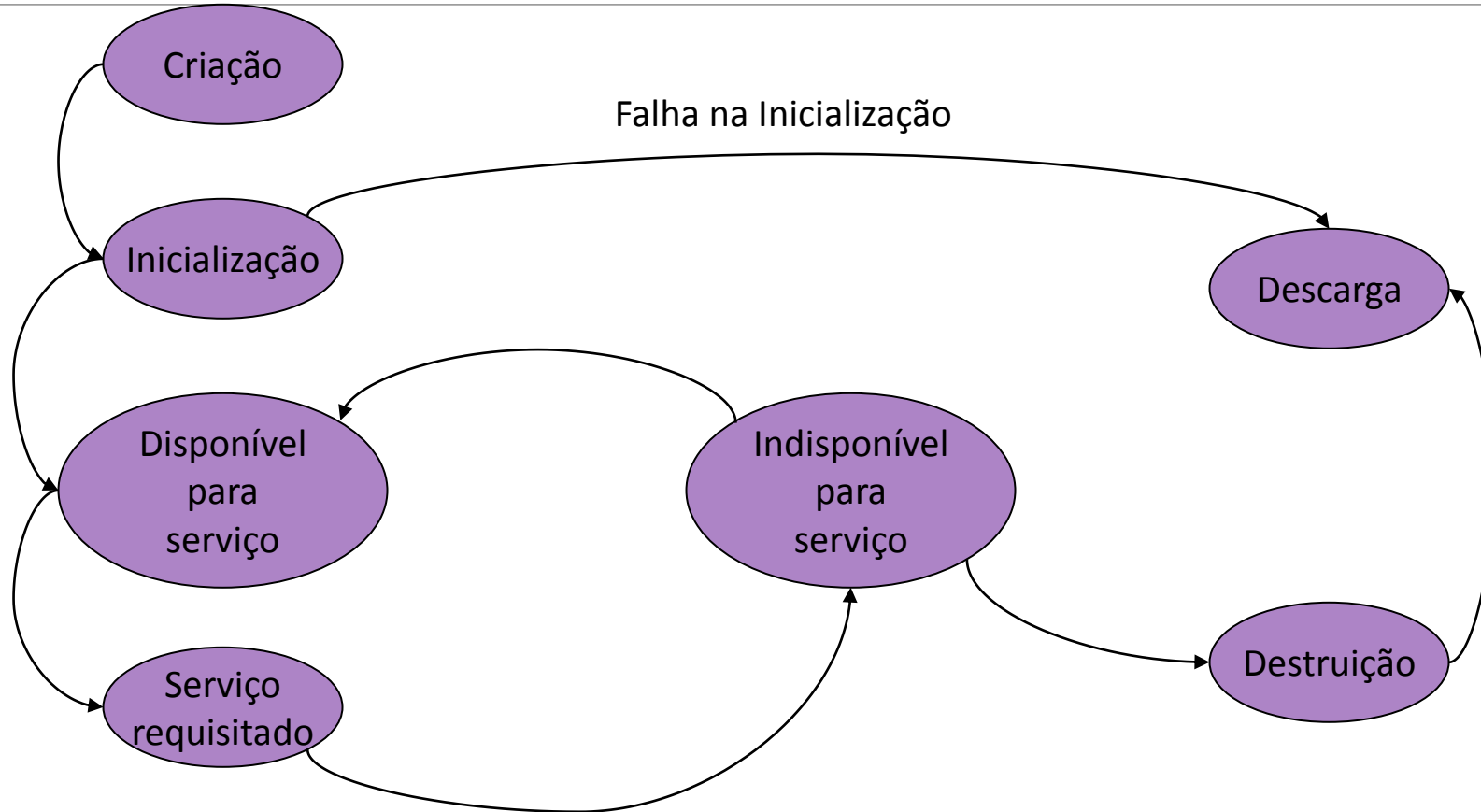
- Este processo é chamado de *Page Compilation*, e ele ocorre somente se não houver um arquivo .class já criado para este JSP ou se houver atualização no arquivo JSP.

Pré compilação, o container Web permite a pré compilação do código JSP para evitar que ele seja compilado em tempo de execução.

- A pré-compilação deve ser ativada no *Application Server* ou através do parâmetro chamado **jsp_precompile**.
- Ex : http://127.0.0.1/MyApp/index.jsp?jsp_precompile=true

Assim como jsp_precompile, os parâmetros iniciados com jsp_ são parâmetros reservados e normalmente são ignorados pelas páginas JSP

Ciclo de vida do JSP



Componentes do JSP

- Diretivas.
- Ações.
- Elementos de script.
- Bibliotecas de tags

Diretivas

São informações encaminhadas para o contêiner que converte e compila o arquivo JSP

Através das diretivas é possível especificar:

- Inclusão de outros recursos dentro da página JSP, como outros JSPs ou páginas HTML. Inclusão de outras classes para serem importadas no código do JSP.
- Configurações
- Definição das bibliotecas de tags personalizadas a serem utilizadas no JSP

Diretivas

Diretivas são delimitadas por `<%@` e `%>`.

As diretivas existentes são:

- **taglib**, carrega uma biblioteca de tags para ser utilizada
- **include**, realiza a inserção de outro recurso no conteúdo deste JSP em tempo de *Page Compilation*
- **page**, define informações da página no momento do *Page Compilation*, esta diretiva é utilizada com os parâmetros:
 - **language**, especifica a linguagem, o único valor válido para este atributo é **java**
 - **extends**, especifica a classe pai da classe JSP a ser criada
 - **import**, importa uma classe Java para ser usada no *Page Compilation*, deixando esta classe disponível para ser utilizada no código dentro da JSP.
 - **session**, especifica se a página participa de uma sessão (padrão é **true**)
 - **buffer**, especifica o tamanho do buffer de saída, a quantidade de informação acumulada, antes de ser encaminhada para o navegador do usuário, os valores válidos são **none** ou um número com o valor ex. **8kb** padrão, **16kb**

Diretivas

-
- **autoflush**, indica se o buffer será esvaziado automaticamente (**true** padrão) caso esteja desativado, o programador deve esvaziar o buffer através do comando **out.flush()** ou ocorrerá uma **exception**
 - **isThreadSafe**, indica se a página está segura para ser utilizada com Thread (**true** padrão), caso contrario a implementação do JSP seguirá a interface **java.lang.SingleThreadModel** e a página somente poderá ser acessada uma **Thread** por vez.
 - **info**, especifica uma **String** para descrever a página
 - **errorPage**, indica para qual página deverá ser encaminhado caso ocorra erro nesta página JSP
 - **isErrorPage**, especifica se a página atual é uma página de erro (**false** padrão)
 - **contentType**, especifica o tipo MIME de resposta a ser encaminhado para o navegador do cliente, o padrão é ("**text/html**")
 - **pageEncoding**, especifica a codificação de caracteres utilizada na página JSP, valor padrão é ("**ISO-8859-1**")
 - **isELIgnored**, indica se as expressões EL serão ignoradas no processamento da página (**false** padrão)

Ações

São tags pré-definidas em JSP que são convertidas para comandos em Java no momento do *Page Compilation*, encapsulando funcionalidades

Podem ser utilizadas para criar e carregar instâncias, modificar e consultar propriedades.

Ações

As tags de ação registradas no JSP são:

<jsp:include>, inclui um recurso em um JSP de maneira dinâmica, a inclusão ocorre no momento da execução.

Atributos:

- **page**, indica o nome do recurso a ser incluído, este recurso deve fazer parte da aplicação Web
- **flush**, especifica se o buffer deve ser esvaziado antes de executar o processo de inclusão

<jsp:forward>, encaminha o processamento para outro recurso, normalmente JSP, Servlet ou uma página com conteúdo estático. Este outro recurso deve estar contido na mesma aplicação Web

<jsp:plugin>, possibilita a adição de um objeto na página JSP como um applet ou flash.

<jsp:param>, é utilizado em conjunto com as demais ações (include, forward e plugin) para especificar parâmetros utilizando os pares de chave e valor.

Ações

As tags de ação registradas no JSP são:

<jsp:useBean>, pode criar ou utilizar uma instância de JavaBean já existente. É possível especificar em qual escopo pode ser criada a instância do JavaBean.

- **id**, nome da variável utilizada para manipular o JavaBean
- **scope**, nome do escopo, ver tópico de **Escopos**
- **class**, nome da classe do JavaBean
- **beanName**, O nome de um JavaBean que pode ser utilizado com o método **instantiate** da classe **java.beans.Beans** para carregar um JavaBean na memória
- **type**, tipo da variável do JavaBean

<jsp:getProperty>, resgata uma propriedade do JavaBean

<jsp:setProperty>, define o valor de uma ou mais propriedades de um JavaBean.

- **name**, nome do JavaBean o qual terá sua propriedade alterada ou resgatada.
- **property**, nome da propriedade a acessar ou alterar, no caso de alteração é possível usar o *wildcard* *
- **param**, pode-se vincular a propriedade de um JavaBean a um parâmetro de request, por padrão o parâmetro já é vinculado com a propriedade se existir um parâmetro com o mesmo nome da propriedade
- **value**, define o valor a ser associado a propriedade (usado apenas na tag **jsp:setProperty**)

Elementos de Script

- Através dos elementos de script é possível inserir código Java dentro da página JSP, o qual pode envolver tags e outros elementos, com a finalidade de implementar páginas dinâmicas
- Os elementos de script podem ser divididos em:
- Scriptlets:
 - Possibilita a inserção de código em Java
- Scriptlet Expression:
 - Permite a inserção de um código Java, onde o resultado é enviado para a tela do navegador do usuário

Elementos de Script

- Scriptlets são blocos de código em Java delimitados por `<% e %>`.
- Comentários de JSP:
 - Comentários na JSP (delimitados por `<%-- e --%>`).
 - Comentários XHTML (delimitados por `<!-- e -->`).
 - Comentários do Java (delimitados por `// e /* e */`).
- Expressões (delimitadas por `<%= e %>`).
- Declarações (delimitadas por `<%! e %>`).

Biblioteca de tags

- Possibilita a criação de bibliotecas de tags que podem desempenhar funções.
- Dessa forma é possível gerar conteúdo dinâmico para a página JSP com mais facilidade.

JSP Exemplo

```
<%@ page language="java" contentType="text/html; charset=US-ASCII" pageEncoding="US-ASCII"%>
<%@ page import="java.util.Date, java.text.SimpleDateFormat" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=US-ASCII">
    <title>Primeiro JSP</title>
  </head>
  <body>
    <h3>Ola</h3>
    <br/>
    <%
      Date d = new Date();
      SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
    %>
    <strong>Horário Atual</strong>: <%=sdf.format(d)%>
  </body>
</html>
```


JSP Exemplo

```
<%@ page language="java" contentType="text/html"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head><title>Segundo JSP</title></head>
<body>
    <%
        double num = Math.random();
        if (num > 0.95) {
    %>
        <h2>Você terá um dia de sorte</h2><p>(<%= num %>)</p>
    <% } else { %>
        <h2>Bem, a vida segue ...</h2><p>(<%= num %>)</p>
    <% } %>
    <a href="<%= request.getRequestURI() %>"><h3>Tente novamente</h3></a>
</body>
</html>
```

JSP - Exercícios

- Crie no eclipse um projeto Web Dinamic e uma pequena página em JSP para acompanhar o processo de **compilação do JSP**
- Cole o seguinte código dentro da tag **<BODY>** na página JSP :

```
<h3> Pagina de Teste em JSP </h3>
<h2> Este código esta em JSP </h2>
<%
    out.println("Este código mostra a tabuada do número 7");
    for (int i = 1; i <= 7; i++ ) {
%>
        <h4> 7 X <%=i%> = <%= (i * 7) %> </h4>
<%
    }
%>
<h2> Fim da Tabuada </h2>
```

- Salve o arquivo e coloque o projeto para rodar no TomCat
- Em seguida acesse o diretório abaixo:

<WorkSpace do Eclipse>

\\.metadata\\.plugins\\org.eclipse.wst.server.core\\tmp1\\work\\Catalina\\localhost\\TesteJSP\\org\\apache\\jsp

- Neste diretório haverão os arquivos **index_jsp.java** e **index_jsp.class**
- Observe o conteúdo do arquivo **index_jsp.java** principalmente na parte em que traduz o código
for (int i = 1; i <= 7; i++) {

JSP - Exercícios

Agora limpe o cache do Tomcat clicando no servidor com o botão direito e escolhendo a opção **Clean Tomcat Work Directory...**

Esta opção limpa todos os arquivos já compilados (.class) pelo Tomcat.

Agora execute a página novamente usando o parâmetro **jsp_precompile**. Exemplo :

- http://localhost:8090/TesteJSP/index.jsp?jsp_precompile

Isso fará com que a página seja compilada, mas não mostrará o conteúdo da página no navegador.

Escopos de memória

Escopos

Objetos implícitos:

- Fornece acesso a muitas capacidades de servlet dentro de uma JSP.
- Quatro escopos:
- Escopo *application*:
 - Objetos possuídos pela aplicação contêiner.
 - Qualquer servlet ou JSP pode manipular esses objetos.
 - Representado pelo objeto **application** no JSP e pelo método **getServletContext()** do servlet
- Escopo *page*:
 - Objetos que só existem na página em que eles são definidos.
 - Cada página tem sua própria instância desses objetos.
 - Representado pelo objeto **page** no JSP

Escopos

Objetos implícitos:

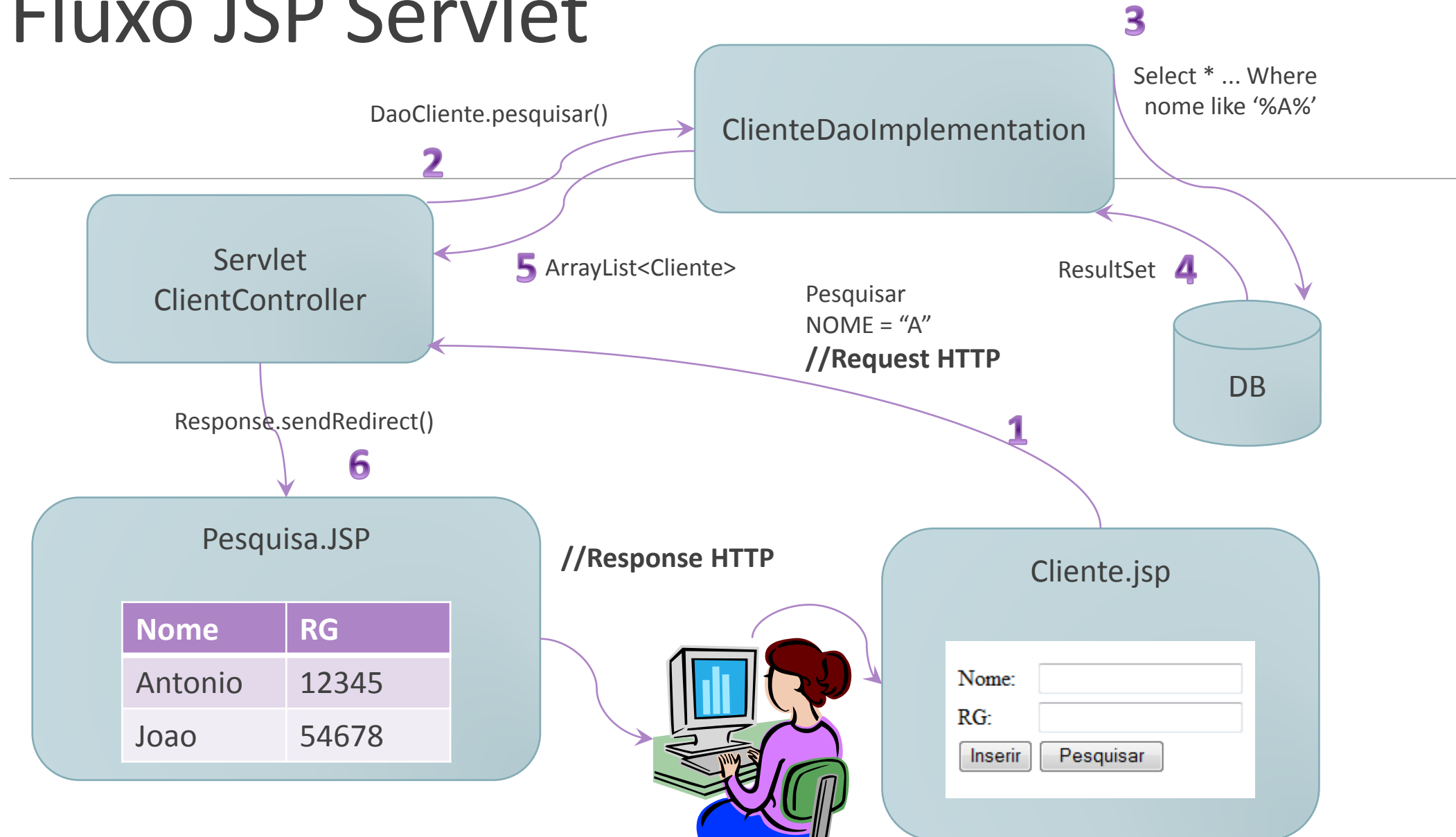
- Escopo *request*:
 - Objetos existem pela duração da solicitação do cliente.
 - Os objetos saem de escopo quando a resposta é enviada ao cliente.
 - Representado pelo objeto **request** no JSP e pelo parâmetro **HttpServletRequest** recebido nos métodos do servlet
- Escopo *session*:
 - Os objetos existem pela duração da sessão de navegação do cliente.
 - Os objetos saem de escopo quando o cliente termina a sessão ou quando o tempo limite de sessão ocorre.
 - Representado pelo objeto **session** no JSP, pode ser obtido no servlet executando o método **getSession()** do objeto **HttpServletRequest** recebido nos métodos do servlet

Escopos

Para pegar ou colocar objetos nas áreas de memória, é possível utilizar os métodos abaixo, os quais estão disponíveis em todos os escopos.

- `getAttribute(String name)` – Retorna um objeto contido no escopo
- `setAttribute(String name, Object obj)` – Coloca um objeto no escopo para ser acessado através do nome

Fluxo JSP Servlet



Bibliografia

ALLEN, PAUL R., Sun Certified Enterprise Architect For J2ee: Guia Oficial de Certificação 2002, Campus

ALUR, DEEPAK, Core J2EE Patterns, Best practices and design strategies - 1ª edição - pg. 371 à 388, Sun Microsystems

BARNES, DAVID J. Programação Orientada a Objetos com Java

Core JEE Pattern Catalog -

<http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html>

DEITEL, Java - Como Programar - 6ª Edição, Pearson Education

KURNIAWAN, BUDI, Java para a Web com Servlets, JSP e EJB, Ed. Ciência Moderna

SIERRA, KATHY e BATES BERT, Use a Cabeça Java, Alta Books