UNIP
UNIVERSIDADE PAULISTA

Aplicações de Linguagem de Programação Orientada a Objetos

MÓDULO 9

Professor: Me. Pablo I. Gandulfo
Data: 01/08/2020
Versão: 1.1

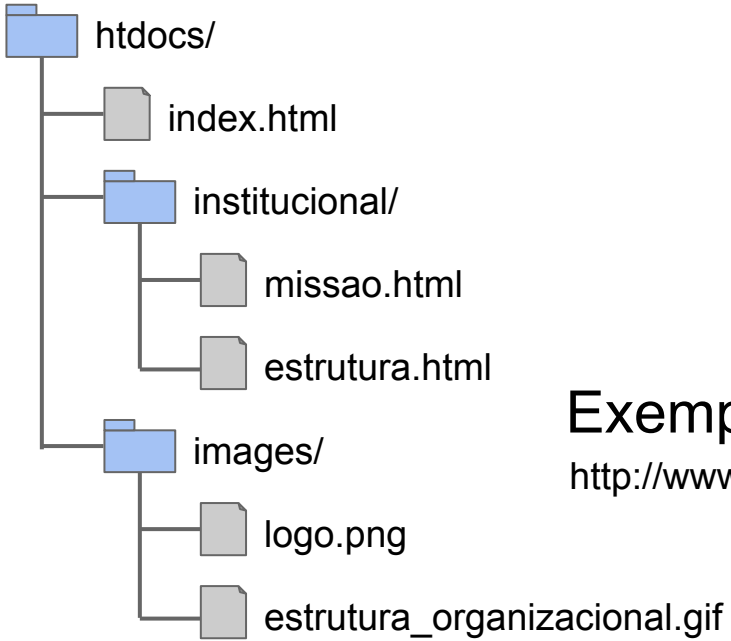
Professor: Me. Pablo I. Gandulfo
Data: 01/08/2020
Versão: 1.1

Introdução a aplicação Web

- Introdução conceitual do JavaEE e de desenvolvimento WEB via JSP e Servlet
  - Conceitos de Programação WEB
  - Servidores (Containers) e Clientes
  - Tecnologias para Desenvolvimento WEB
  - Servlets
- Primeiros Servlets e JSP's
- Exercícios

Estrutura de um Site Web

- Um site Web é uma hierarquia de documentos HTML estáticos, arquivos de mídia e os diretórios que formam a estrutura:



Exemplo de uma URL HTTP:

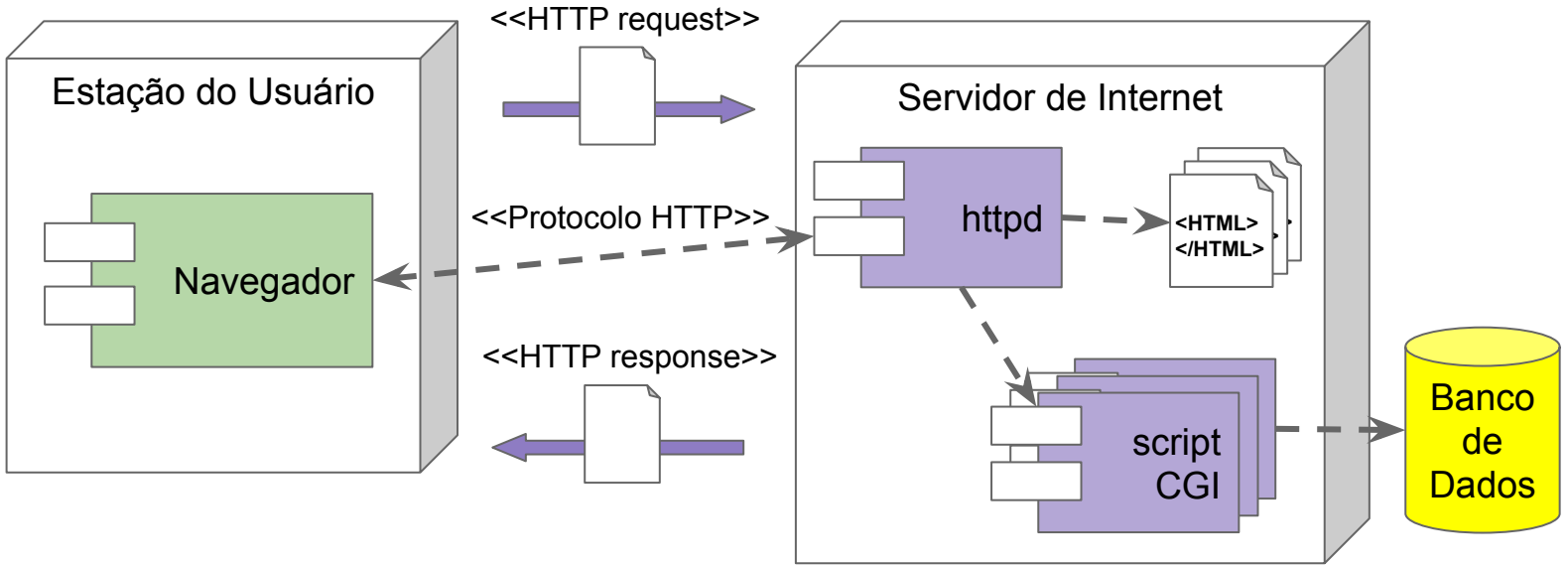
http://www.negociosbr.com.br/institucional/missao.html

Aplicação Web

- Uma aplicação Web é um site Web com funcionalidade dinâmica no servidor (ou, às vezes, no cliente, através de *applets* e/ou outros elementos interativos)
- Aplicações Web usam formulários HTML como interface entre o usuário e o código que executa do lado do servidor:
  - Dados são passados do formulário HTML ao servidor Web através do protocolo HTTP
  - O servidor Web repassa os dados à aplicação Web, e depois devolve a resposta correspondente através de algum tipo de mecanismo. Um dos mecanismos conhecidos, que segue uma especificação padronizada entre diferentes servidores Web é o CGI - Common Gateway Interface.

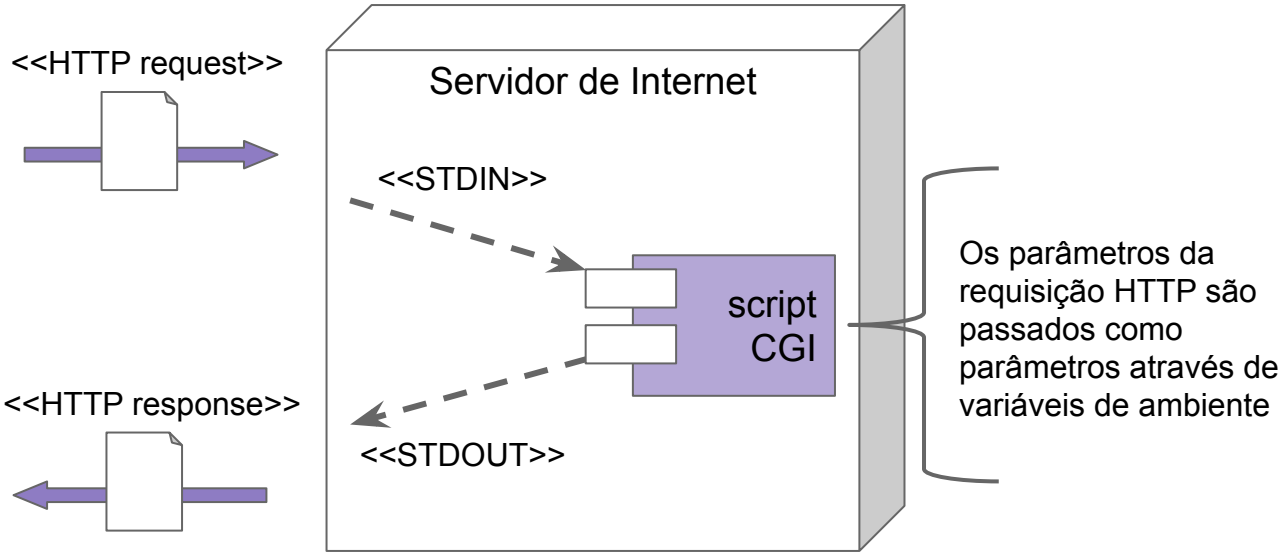
Programas CGI no Servidor Web

- Diagrama do servidor Web com programas CGI:



Obs.: cada requisição aciona o programa (script) CGI num processo separado

Fluxo de um Programa CGI



Vantagens e Desvantagens de Programas CGI

- Vantagens:
  - Escritos numa variedade de linguagens
  - Relativamente fácil de um *web designer* referenciar

- Desvantagens:
  - Cada processo é relativamente pesado
  - Não é escalável
  - Código de processamento CGI (lógica de negócio) é entrelaçada com HTML (lógica de apresentação)
  - A linguagem nem sempre é segura, ou orientada a objetos
  - A linguagem nem sempre é independente de plataforma

Servlets JAVA

- Servlet é uma tecnologia de componente JAVA que executa no servidor
- Servlets atuam de forma similar aos programas CGI, mas executam num ambiente diferente

- Um Container Web é uma JVM especial que é responsável por manter o ciclo de vida de servlets, tanto quanto acionar o servlet correspondente para tratar uma requisição HTTP

Obs.: cada requisição executa o servlet numa *thread* separada

Vantagens e Desvantagens de Servlets JAVA

- Vantagens:
  - Desempenho (*threads* são mais rápidas que processos)
  - Escalabilidade
  - A linguagem de programação JAVA é robusta e orientada a objetos
  - A linguagem de programação JAVA é independente de plataforma

- Desvantagens:
  - Dependente da linguagem de programação
  - Questões envolvendo concorrência

Páginas Dinâmicas

- Páginas dinâmicas parecem páginas HTML estáticas, mas contém código para executar geração dinâmica de dados e HTML

- Exemplo:

```
<TABLE BORDER="1" CELLSPACING="0" CELLPADDING="5">
<TR><TH>número</TH><TH>quadrado</TH></TR>
<% for (int i = 0; i < 5; i++) { %>
<TR><TD><%= i %></TD><TD><%= (i * i) %></TD></TR>
<% } %>
</TABLE>
```

número	quadrado
0	0
1	1
2	4
3	9
4	16

Outras Tecnologias de Páginas Dinâmicas

- PHP - PHP Hypertext Preprocessor

```
<? for ($i = 0; $i < 5; $i++) { ?>
<TR><TD><? echo $i ?></TD><TD><? echo ($i * $i) ?></TD></TR>
<? } ?>
```

- ASP - Active Server Pages

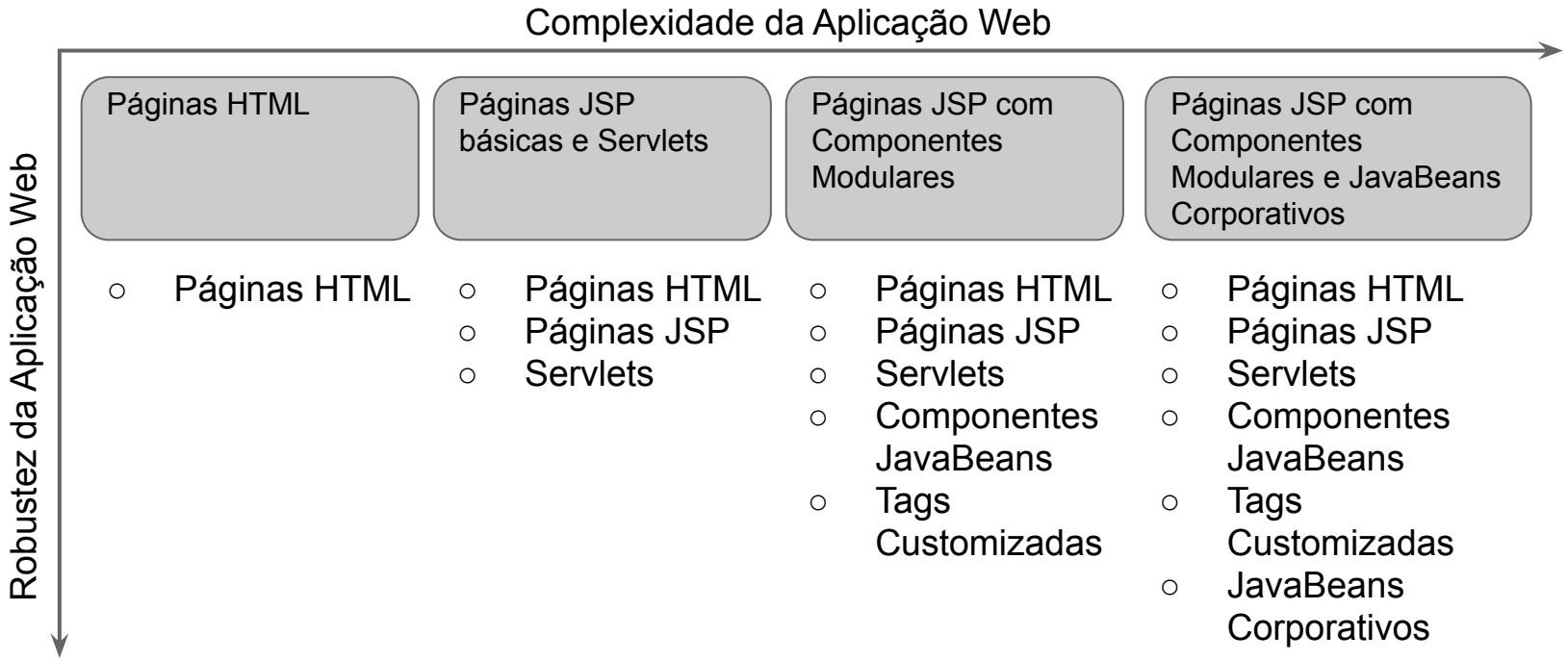
```
<% FOR I = 0 TO 4 { %>
<TR><TD><%= I %></TD><TD><%= (I * I) %></TD></TR>
<% NEXT %>
```

- JSP - JavaServer Pages

```
<% for (int i = 0; i < 5; i++) { %>
<TR><TD><%= i %></TD><TD><%= (i * i) %></TD></TR>
<% } %>
```

Migração das Aplicações Web (Robustez)

- Matriz mostrando a relação entre a robustez e complexidade da arquitetura, baseado nas tecnologias utilizadas:



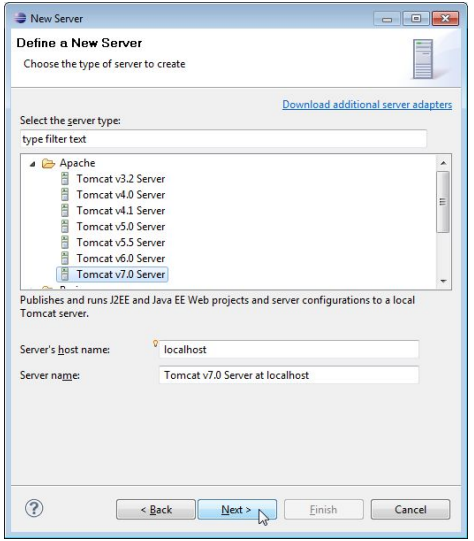
Servidores (e Containers) Web

- Container é um componente que pode conter outros componentes, incluídos ou removidos dinamicamente (em tempo de execução)
- Um Container Web é responsável por prover servlets, controlando o ciclo de vida dos mesmos, oferecendo suporte a *multithread*, segurança e suporte a páginas JSP
- Um Servidor Web é um serviço capaz de interpretar requisições HTTP de múltiplos clientes e enviar respostas HTTP correspondentes
- Exemplos:
  - Servidor Apache (o mais popular e mais utilizado no mundo)
  - Microsoft IIS
  - Wamp
  - Zeus Web Server

Container Web Apache Tomcat

- Desenvolvido pela Apache Software Foundation, é um Container Web distribuído como software livre

- O download pode ser feito a partir de http://tomcat.apache.org/
  - Exemplo
    - http://mirror.nbtelecom.com.br /apache/tomcat/tomcat-8/v8.5.12 /bin/apache-tomcat-8.5.12.zip



- Deve ser adicionado no Eclipse como um Servidor (menu Window \ Preferences \ Server \ Runtime Environments)

Clientes Web - Navegadores

- O Navegador é um cliente web capaz de se comunicar com um Servidor Web para enviar requisições HTTP, receber respostas HTTP e renderizá-las apropriadamente
  - Interpreta um documento HTML para processar os comandos envolvidos
  - Lê e processa vários tipos de arquivos, alguns de forma nativa (geralmente HTML e imagens) e outros através de *plugins* (Flash, Java, etc.)
- Exemplos:
  - Google Chrome
  - Mozilla Firefox
  - Microsoft Internet Explorer
  - Apple Safari

Tecnologias para Desenvolvimento WEB

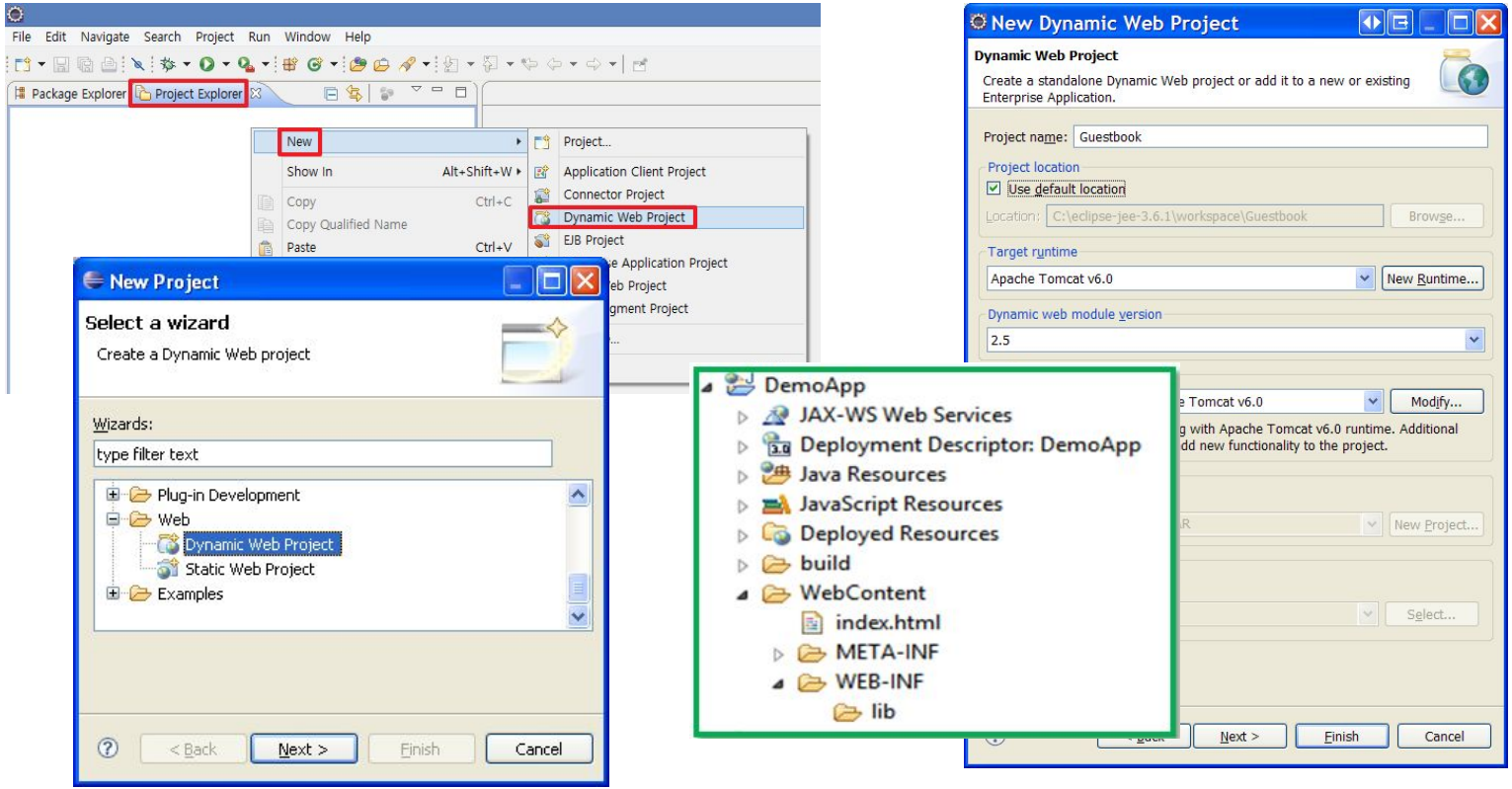
- JAVA Servlet API: os servlets processam requisições HTTP e geram respostas HTTP dinamicamente

- JavaServer Pages (JSP): páginas JSP são parecidas com os servlets, mas são construídas a partir de documentos baseados em texto para receberem trechos de código para geração dinâmica de conteúdo
  - Diferente de outras páginas dinâmicas (ASP e PHP), seu código é traduzido em servlet para depois ser compilado

- JavaServer Pages Standard Tag Library (JSTL): encapsula funcionalidade comum a várias páginas JSP através de tags customizadas

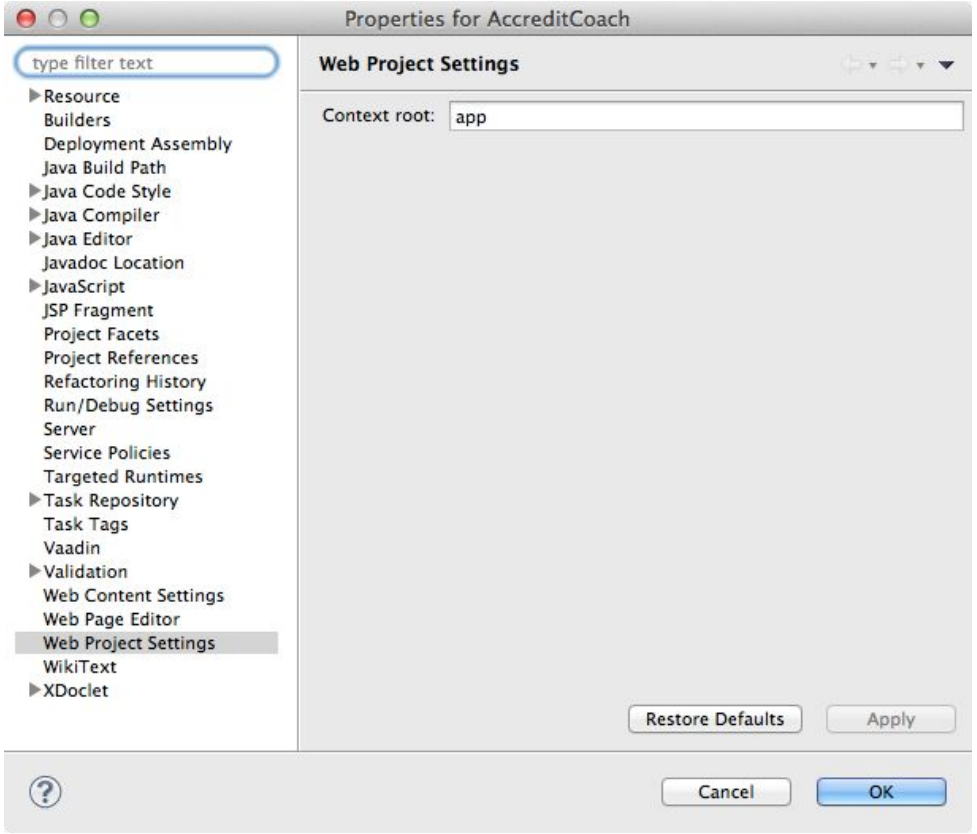
- JavaServer Faces: framework de I.U. para a construção de aplicações web

Projeto *Dynamic Web Project* do Eclipse



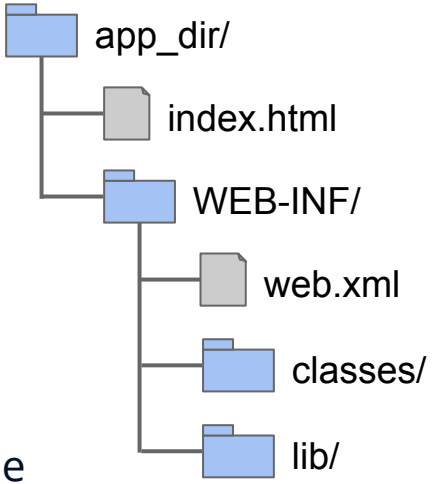
Configuração *Context Root*

- A configuração *Context Root* define o nome da aplicação web no âmbito do *container web*
- Uma vez criado o projeto, qualquer mudança nessa configuração requer parar o servidor web e executar o comando *Clean* no mesmo



Estrutura Padrão de uma Aplicação Web no Tomcat

- Uma aplicação web normalmente segue uma estrutura definida pelo *container web* que a publicará
- No Tomcat, a estrutura segue a seguinte formação:
  - **app_dir**: diretório raiz da aplicação
  - **index.html** ou quais outros arquivos e diretórios: representam as páginas HTML (estáticas ou dinâmicas), documentos, imagens, sons, vídeos, e outros recursos necessários para o completo funcionamento da aplicação
  - **WEB-INF**: diretório protegido contendo arquivos de definição ou configuração, classes compiladas e/ou bibliotecas JAVA
  - **web.xml**: descritor da aplicação web
  - **classes**: diretório contendo as classes compiladas JAVA
  - **lib**: diretório contendo bibliotecas JAVA



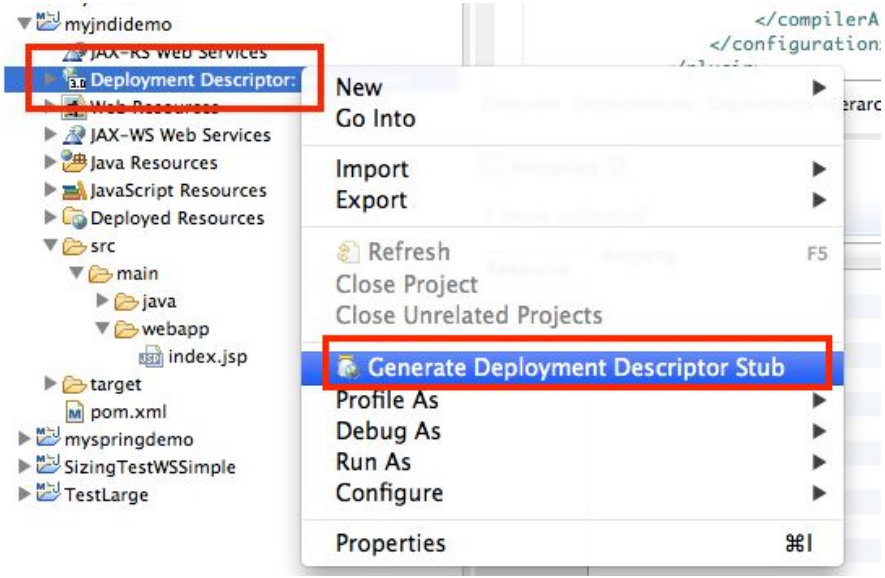
Cadastro ou Registro da Aplicação no Tomcat

- Existe mais de uma forma de informar ao Tomcat que uma aplicação deverá ser publicada para uso. Estas são algumas delas:
  - Editar o arquivo [dir_tomcat]\conf\server.xml e acrescentar uma linha parecida com esta na seção <Host ...>...</Host>:
    - <Context docBase="[app_dir]" path="/[context root]" reloadable="true"/>
  - Criar um arquivo no caminho [dir_tomcat]\conf\Catalina\[host]\[context root].xml com este conteúdo:
    - <?xml version="1.0" encoding="UTF-8"?>
    - <Context docBase="[app_dir]" path="/[context root]" reloadable="true"/>

- Algumas dessas formas permitem mudanças enquanto o Tomcat está no ar, mas outras requerem o seu reinício

Descritor de Distribuição web.xml

- O arquivo web.xml é opcional dependendo da versão do Tomcat. Portanto, não é mais gerado inicialmente pelo Eclipse. Mas pode ser gerado a qualquer momento através da seguinte opção do menu:



Servlets – O Servlet OlaMundoServlet.java

```
import java.io.IOException; import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet; import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest; import javax.servlet.http.HttpServletResponse;


@WebServlet("/OlaMundoServlet")
public class OlaMundoServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        response.setContentType("text/html"); // Specify the content type is HTML
        PrintWriter out = response.getWriter();
        out.println("<html><body>Olá, mundo!</body></html>");
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        doGet(request, response);
    }
}
```

URL
http://localhost:8080/intermediariojee/OlaMundoServlet

Saída
<html><body>Olá, mundo!</body></html>

Conceitos - URL para Chamada de um Servlet

- Vamos entender a URL utilizada no exemplo anterior:
  - http://localhost:8080/intermediariojee/OlaMundoServlet
- Campos:
  - **http**: protocolo de conversação utilizado, podendo ser também https (http + SSL)
  - **localhost**: domínio ou servidor envolvido
  - **8080**: porta padrão em que o Tomcat executa, mas pode ser outra dependendo das configurações do Tomcat
  - **intermediariojee**: configuração *context root* da aplicação *web*
  - **/OlaMundoServlet**: caminho pra se chegar ao servlet, definido na própria classe através da anotação @WebServlet("/OlaMundoServlet"). É importante lembrar que o caminho publicado do servlet não precisa ter relação com o pacote e nome real da classe (similar a outros mapeamentos feitos num servidor *web*).

URL para Chamada de um Servlet (cont.)

- Campos:
  - **/OlaMundoServlet** (cont.): O mapeamento do servlet pode também ser feito explicitamente no arquivo web.xml da aplicação. Exemplo de configuração para o mesmo servlet, pertencente ao pacote **servlets**:
    - <web-app>
      <servlet>
      <servlet-name>OlaMundoServlet</servlet-name>
      <servlet-path>servlets.OlaMundoServlet</servlet-path>
      </servlet>
      <servlet-mapping>
      <servlet-name>OlaMundoServlet</servlet-name>
      <url-pattern>/OlaMundoServlet</url-pattern>
      </servlet-mapping>
      </web-app>

A Página OlaMundo.jsp

```
<%@ page language="java" contentType="text/html" %>
<html><body>Olá, mundo!</body></html>
```

URL
http://localhost:8080/intermediariojee/OlaMundo.jsp

Servlet Traduzido
[WorkspaceDir]\.metadata\.plugins\org.eclipse.wst.server.core\tmp0\work\Catalina\localhost
\intermediariojee\org\apache\jsp\OlaMundo_jsp.java

Saída
<html><body>Olá, mundo!</body></html>

URL para Chamada de um JSP

- Vamos entender a URL utilizada no exemplo anterior:
  - http://localhost:8080/intermediariojee/OlaMundo.jsp
- Campos:
  - **http** / **localhost** / **8080** / **intermediariojee**: descritos anteriormente
  - **/OlaMundo.jsp**: caminho pra se chegar ao jsp, correspondendo diretamente ao caminho em que o arquivo OlaMundo.jsp se encontra a partir do diretório raiz da aplicação web.

Exercício 1

- Desenvolva um servlet a partir do código do arquivo FormularioExemplo.html

- Teste o mesmo código utilizando os dois métodos de requisição, GET e POST, e avalie as diferenças entre eles