

DESENVOLVIMENTO DE APLICAÇÕES PARA INTERNET

Parte IV

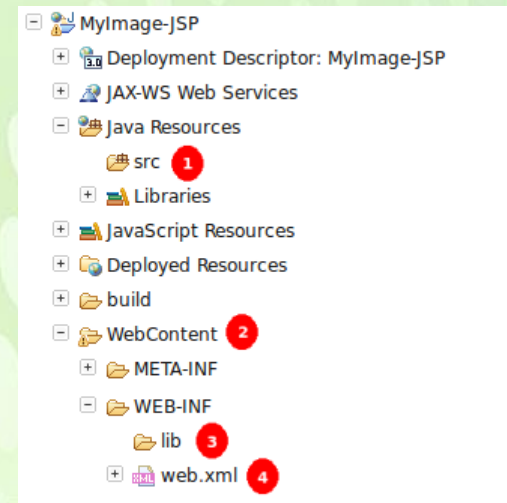
Prof. Msc. Felipe Diniz Dallilo



Estrutura para aplicação WEB

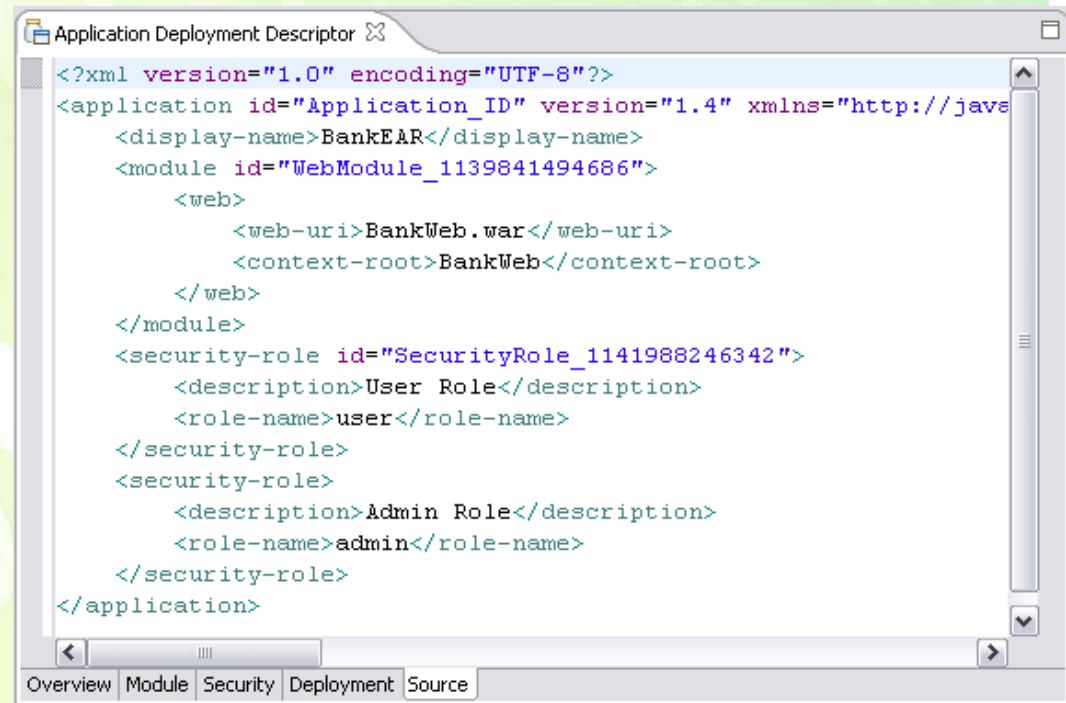
- Ao criar uma aplicação web, a seguinte estrutura é apresentada:

1. Codificação Java
2. Todos os recursos web, como as páginas JSPs, CSSs e Javascripts
3. Bibliotecas Java.
4. Configuração da aplicação / Deployment Descriptor



Deployment Descriptor

- Um deployment descriptor declara como uma aplicação web deve ser implantada/compilada.
- Uma aplicação deve conter apenas um DD e o mesmo é definido em XML (sobrescrito pelo exemplo de mapeamento por annotations na aula passada).
- Efetua o mapeamentos dos servlets, filtros, atribuições de segurança, páginas de erro, bibliotecas de tags, configurações iniciais...



```
<?xml version="1.0" encoding="UTF-8"?>
<application id="Application_ID" version="1.4" xmlns="http://java.sun.com/xml/ns/javaee">
  <display-name>BankEAR</display-name>
  <module id="WebModule_1139841494686">
    <web>
      <web-uri>BankWeb.war</web-uri>
      <context-root>BankWeb</context-root>
    </web>
  </module>
  <security-role id="SecurityRole_1141988246342">
    <description>User Role</description>
    <role-name>user</role-name>
  </security-role>
  <security-role>
    <description>Admin Role</description>
    <role-name>admin</role-name>
  </security-role>
</application>
```

The screenshot shows a web IDE window titled "Application Deployment Descriptor". The main text area contains an XML document. The XML defines an application named "BankEAR" with version "1.4". It includes a web module named "WebModule_1139841494686" which has a web-uri of "BankWeb.war" and a context-root of "BankWeb". There are also two security roles defined: "SecurityRole_1141988246342" with description "User Role" and role-name "user", and another security role with description "Admin Role" and role-name "admin". The IDE has tabs at the bottom: Overview, Module, Security, Deployment, and Source.

O protocolo HTTP Teoria de Funcionamento

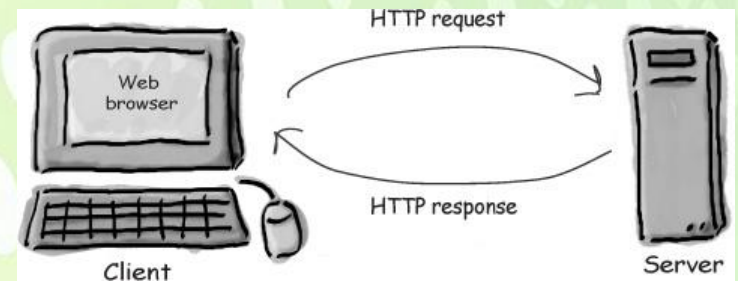
- HTTP (acrônimo de Hypertext Transfer Protocol) é um protocolo que executa no topo da TCP/IP.
- TCP é responsável por garantir que um arquivo enviado de um nó de rede seja entregue inteiro ao seu destino (mesmo dividido em blocos)
- IP é o protocolo de sustentação, transfere/roteia os blocos (pacotes) de um host para outro no seu caminho até o destino.
- HTTP é outro protocolo de rede, no entanto, depende do TCP/IP para obter a solicitação e a resposta completas de um lugar para outro.

A estrutura de uma conversa HTTP é uma sequência simples de **Solicitação/Resposta** (navegador solicita e servidor responde)

Solicitações HTTP prática

- O protocolo HTTP possui diversos métodos,
- No entanto, os mais habituais são o **GET** e o **POST**.

Principais características:



GET – Solicita dados de um recurso especificado

- **POST** - Submete dados a ser processado para um recurso especificado

Posso submeter dados pelo método get?

- A resposta é sim, apesar do método get ser utilizado para requisitar dados, pode ser adicionado parâmetros a sua requisição como ilustrado abaixo:

`/webtest/servletTeste?parm1=ola&parm2=mundo`

- Alguns pontos interessantes desta requisição:
- **?** – início dos parâmetros
- **&** - concatenação de parâmetros

chave=valor – os parâmetros são passados como chave/valor para o servidor.

- **URL** = Parâmetros passados pela URL da requisição

Por que não utilizar?

- Dentre os diversos motivos, os principais são:
- **Tamanho limitado** = A URL possui um tamanho máximo para ser solicitada.
- **Segurança** = Parâmetros são salvos no histórico do browser
- **Visibilidade** = Deseja mesmo passar uma senha pela URL para todos visualizarem?
- **ASCII** = Apenas caracteres ASCII

Como o Post funciona?

- **Requisição**
- **URL:**
 - /webtest/servletTeste
- **Body:**
 - POST /webtest/servletTeste HTTP/1.1
Host: uniara.com
name1=value1&name2=value2

Anatomia da Requisição GET

RequestHTTP GET

GET/servlet/login.do?n=ia&me=ra

Host:localhost:8080

User-Agent: mOzilla/4.0(compatible; Windows NT 5.0)

Refer: http://localhost/login.jsp

Accept: text/xml, application/xml

Accept-Language: en-us

Accept-Encoding: gzip, deflate

Connection: Keep-Alive

Anatomia da Requisição POST

RequestHTTP POST

POST/servlet/login.do

Host:localhost:8080

User-Agent: mOzilla/4.0(compatible; Windows NT 5.0)

Refer: http://localhost/login.jsp

Accept: text/xml, application/xml

Accept-Language: en-us

Accept-Encoding: gzip, deflate

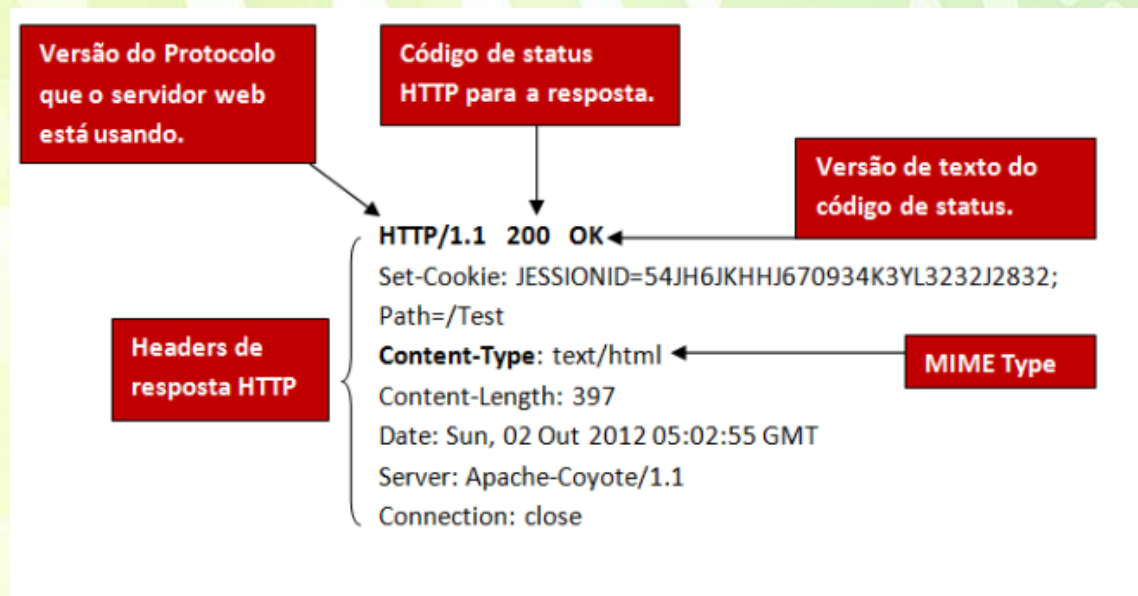
Connection: Keep-Alive

Quais outros métodos o Servlet pode tratar?

- Apesar dos métodos GET e POST serem os mais utilizados (únicos suportados pelo HTML4), também é possível utilizar:
 - **HEAD** - Mesmo que GET, mas retorna apenas os cabeçalhos HTTP e nenhum corpo documento
 - **TRACE** - Para permitir que um servlet para lidar com um pedido TRACE/Depuração
 - **PUT** - Carrega uma representação do URI (Uniform Resource Identifier) especificado
 - **DELETE** – Deleta um recurso específico
 - **OPTIONS** – Retorna os métodos HTTP que o servidor suporta
 - **CONNECT** – Converte a conexão pedido para um túnel de transporte TCP/IP
 - **PATCH** – Não implementado no Apache (utilizar o service) , utilizado para atualizações parciais

Respostas HTTP prática

- Além do arquivo com o código HTML, o servidor responde uma linha de status e linhas de cabeçalho como pode ser visualizado na imagem abaixo:



CÓDIGOS DE ERRO

- Os códigos seguem uma padronização simples de serem definidos:
- 1xx: Informações gerais
- 2xx: Sucesso na requisição e resposta
- 3xx: Redirecionado para outra url
- 4xx: Erro (por parte do cliente);
- 5xx: Erro (por parte do servidor);

CÓDIGOS DE ERRO – Mais específicos

- 100 Continuar
- 101 Mudando protocolos
- 102 Processamento (WebDAV) (RFC 2518)
- 122 Pedido-URI muito longo
- 200 OK
- 201 Criado
- 202 Aceito
- 203 não-autorizado (desde HTTP/1.1)
- 204 Nenhum conteúdo
- 205 Reset
- 206 Conteúdo parcial
- 207-Status Multi (WebDAV) (RFC 4918)
- 300 Múltipla escolha
- 301 Movido
- 302 Encontrado
- 304 Não modificado
- 305 Use Proxy (desde HTTP/1.1)
- 306 Proxy Switch
- 307 Redirecionamento temporário (desde HTTP/1.1)
- 400 Requisição inválida
- 401 Não autorizado
- 402 Pagamento necessário
- 403 Proibido
- 404 Não encontrado
- 405 Método não permitido
- 406 Não Aceitável
- 407 Autenticação de proxy necessária
- 408 Tempo de requisição esgotou (Timeout)
- 409 Conflito
- 410 Gone
- 411 comprimento necessário
- 412 Pré-condição falhou
- 413 Entidade de solicitação muito grande
- 414 Pedido-URI Too Long
- 415 Tipo de mídia não suportado
- 416 Solicitada de Faixa Não Satisfatória
- 417 Falha na expectativa
- 418 Eu sou um bule de chá
- 422 Entidade improcessável (WebDAV) (RFC 4918)
- 423 Fechado (WebDAV) (RFC 4918)
- 424 Falha de Dependência (WebDAV) (RFC 4918)
- 425 coleção não ordenada (RFC 3648)
- 426 Upgrade Obrigatório (RFC 2817)
- 450 bloqueados pelo Controle de Pais do Windows
- 499 cliente fechou Pedido (utilizado em ERPs/VPs)
- 500 Erro interno do servidor (Internal Server Error)
- 501 Não implementado (Not implemented)
- 502 Bad Gateway
- 503 Serviço indisponível (Service Unavailable)
- 504 Gateway Time-Out
- 505 HTTP Version not supported

Na prática- Requisição com parâmetros

```
<html>
<head><title>Página de Login</title></head>
<body>
<form action=" ${pageContext.request.contextPath}/login.do" method="get">
    Nome.: <input type="text" name="nome" />
<br />
    Sobrenome.: <input type="text" name="sobrenome" />
<br />
<br />
<input type="submit" value="Enviar"/>
</form>
</body>
</html>
```

SERVLET

Características:

- Tradução significa “Servidorzinho”.
- **Classe Java** utilizada como interface do servidor web para receber as **requisições** do cliente e efetuar as **respostas**

Frequentemente usados para:

- Processar ou armazenar dados que foram submetidos de um formulário HTML
- Fornecer conteúdo dinâmico, como os resultados de uma consulta a um banco de dados
- Gerenciar a informação de estado que não existe no protocolo sem estado HTTP, como inserir/retirar os itens de uma cesta de compras de um cliente específico (**Session**)

Coletando os parâmetros

- No servlet (ou JSP), para coletar os parâmetros de entrada da requisição é necessário utilizar o comando:
- **String parametro = Request.getParameter("parm1");**

Servlet

```
import java.io.IOException;
import java.io.PrintWriter;
//Bibliotecas da classe Servlet
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class HelloWorld extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        out.println("<html>\n" +
            "<head><title>Hello WORD</title></head>\n" +
            "<body>\n" +
            "<h1>Hello WORD</h1>\n" +
            "</body></html>");
    }
}
```


Servlet – do get()

- Entrada e Saída de um Servlet:

```
String parametro = Request.getParameter("nome_html"); //  
propriedade name do html
```

```
PrintWriter saida = response.getWriter(); // coleta o objeto de  
resposta
```

```
String serverResponse = "Servidor recebeu requisição";  
saida.println("<html> <head> <title> Meu site Cliente-Servidor  
</title></head> <body> " + serverResponse + "</body> </html>");
```

Servlet Mapping

No Deployment Descriptor, o XML abaixo é utilizado (dentro da tag “web-app”) para o mapeamento dos Servlets:

```
<servlet>
```

```
  <servlet-name> Internal name xpto </servlet-name> // este name é  
utilizado para fazer a
```

```
  <servlet-class> Teste </servlet-class> // O servlet tem este nome  
</servlet>
```

```
<servlet-mapping>
```

```
  <servlet-name> Internal name xpto </servlet-name> //ligação com este  
mapping aqui
```

```
  <url-pattern>/Public1 </url-pattern> //Mas o usuário digita isto  
no navegador  
</servlet-mapping>
```

Servlet mapping

- Com a annotation `@WebServlet`, é possível mapear a chamada da requisição url com o Servlet, sem a necessidade de um arquivo xml. Segue abaixo um exemplo:

```
@WebServlet("/Teste")                Pode ser feito assim também !!!!
public class Teste extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public Teste() {
        super();
        // TODO Auto-generated constructor stub
    }
}
```

Referências

- Deitel, Paul J.; Deitel, Harvey M. Java: como programar - 8ª edição. Editora Pearson. ISBN: 9788576055631
- Barnes, David.; Kölling, Michael. Programação Orientada a Objetos com Java: uma introdução prática usando o Blue J - 1ª edição. 2013. Editora Pearson. ISBN: 9788576050124
- Puga, Sandra; Rissetti, Gerson. Lógica de Programação e Estruturas de Dados: com aplicações em Java. 1ª edição. 2013. Editora Pearson. ISBN: 9788587918826
- Lemay, Laura; Colburn, Rafe; Tyler, Denise. Aprenda a Criar Páginas Web com HTML e XHTML em 21 Dias. Editora Pearson. 2013. 1ª Edição. ISBN: 9788534614283
- Chak, Andrew. Como Criar Sites Persuasivos. Editora Pearson. 2012. 1ª Edição. ISBN: 9788534615112
- Fábio Flatschart; Clécio Bachini; Cesar Cusin. Open Web Platform. Editora Pearson. 2013. 1ª Edição. ISBN: 9788574526140