Contextos

Helder da Rocha (helder@acm.org)
www.argonavis.com.br

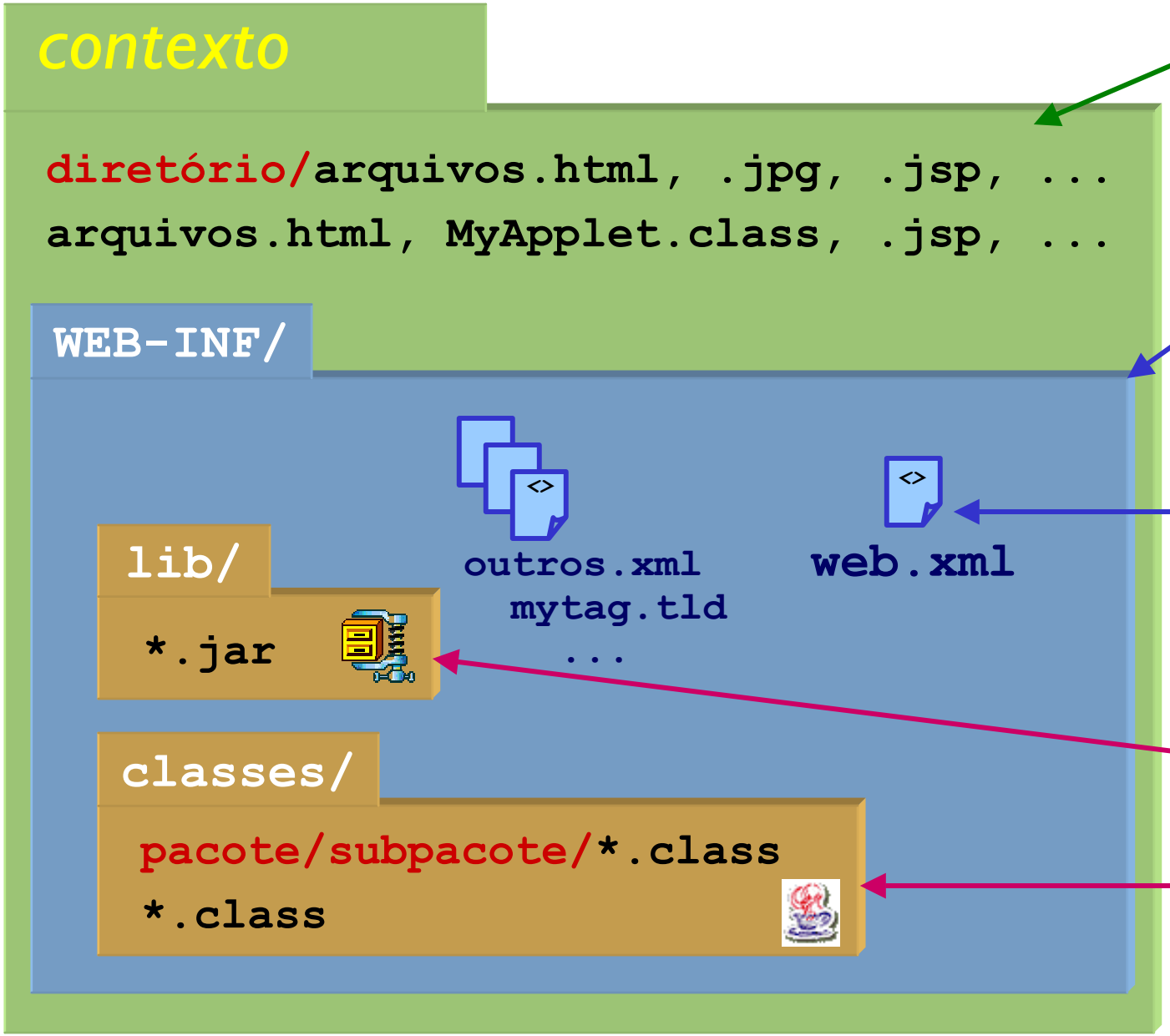
Sobre este módulo

- Neste módulo serão apresentadas aplicações Web configuráveis através de um deployment descriptor
- Aplicações Web são quase a mesma coisa que contextos. Contextos possuem
  - Diretório próprio que define o escopo máximo da aplicação
  - Um CLASSPATH próprio onde rodam os servlets
  - Um arquivo de configuração (web.xml) - o deployment descriptor: usado na implantação da aplicação quando o servidor inicia
- Também serão abordados
  - Passagem de atributos através do contexto e da requisição
  - Logging

Aplicações Web (contextos)

- *Web Containers suportam a implantação de múltiplas aplicações Web*
  - *Definem contextos separados para execução de servlets*
- *No Tomcat, essas aplicações estão na pasta webapps/*
  - *Veja o conteúdo de webapps no seu servidor*
- *Todo diretório de contexto tem uma estrutura definida, que consiste de*
  - *Área de documentos do contexto (/), acessível externamente*
  - *Área inacessível (/WEB-INF), que possui pelo menos um arquivo de configuração padrão (web.xml)*
  - *O WEB-INF pode conter ainda dois diretórios reconhecidos pelo servidor: (1) um diretório que pertence ao CLASSPATH da aplicação (/WEB-INF/classes) e (2) outro onde podem ser colocados JARs para inclusão no CLASSPATH (/WEB-INF/lib)*

Estrutura de uma aplicação Web



Arquivos acessíveis ao cliente a partir da raiz do contexto

Área inaccessível ao cliente

Arquivo de configuração (WebApp deployment descriptor)

Bibliotecas

Classpath (Contém Classes, JavaBeans, Servlets)

Componentes de um contexto

- *A raiz define (geralmente) o nome do contexto.*
  - *Na raiz ficam HTMLs, páginas JSP, imagens, applets e outros objetos para download via HTTP*

{Contexto}/WEB-INF/web.xml

  - *Arquivo de configuração da aplicação*
  - *Define parâmetros iniciais, mapeamentos e outras configurações de servlets e JSPs.*

{Contexto}/WEB-INF/classes/

  - *Classpath da aplicação*

{Contexto}/WEB_INF/lib/

  - *Qualquer JAR incluido aqui será carregado como parte do CLASSPATH da aplicação*

Nome do contexto e URL

- *A não ser que seja configurado externamente, o nome do contexto aparece na URL após o nome/porta do servidor*

```
http://serv:8080/contexto/subdir/pagina.html
http://serv:8080/contexto/servlet/pacote.Servlet
```

- *Para os documentos no servidor (links em páginas HTML e formulários), a raiz de referência é a raiz de documentos do servidor, ou* `DOCUMENT_ROOT`: `http://serv:8080/`
- *Documentos podem ser achados relativos ao* `DOCUMENT_ROOT`

```
/contexto/subdir/pagina.html
/contexto/servlet/pacote.Servlet
```

- *Para a configuração do contexto (web.xml), a raiz de referência é a raiz de documentos do contexto:* `http://serv:8080/contexto/`
- *Componentes são identificados relativos ao contexto*

```
/subdir/pagina.html
/servlet/pacote.Servlet
```

`servlet/` *é mapeamento virtual definido no servidor para servlets em* `WEB-INF/classes`

Tipos e fragmentos de URL

- *URL absoluta: identifica recurso na Internet. Usada no campo de entrada de localidade no browser, em páginas fora do servidor, etc.*

  `http://serv:8080/ctx/servlet/pacote.Servlet/cmd/um`

- *Relativa ao servidor (Request URI): identifica o recurso no servidor. Pode ser usada no código interpretado pelo browser nos atributos HTML que aceitam URLs (para documentos residentes no servidor)*

  `/ctx/servlet/pacote.Servlet/cmd/um`

- *Relativa ao contexto: identifica o recurso dentro do contexto. Pode ser usada no código de servlets e JSP interpretados no servidor e web.xml. Não contém o nome do contexto.*

  `/servlet/pacote.Servlet/cmd/um`

- *Relativa ao componente (extra path information): texto anexado na URL após a identificação do componente ou página*

  `/cmd/um`

Criando um contexto válido

- *Para que uma estrutura de diretórios localizada no webapps/ seja reconhecida como contexto pelo Tomcat, na inicialização, deve haver um arquivo web.xml no diretório WEB-INF do contexto*
  - *O arquivo é um arquivo XML e deve obedecer às regras do XML e do DTD definido pela especificação*
  - *O conteúdo mínimo do arquivo é a declaração do DTD e um elemento raiz <web-app/>*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app/>
```

  - *Se houver qualquer erro no web.xml, a aplicação não será carregada durante a inicialização*

Exercício

- 1. Construa um novo contexto exercicio
  - Crie um diretório exercicio dentro de webapps/
  - Monte a estrutura esperada, com diretório WEB-INF, diretório classes para os servlets e web.xml vazio (copie o que está no WEB-INF do ROOT)
  - Copie alguns dos servlets e páginas Web que você fez em exercícios anteriores para ele
  - Reinicie o servidor
  - Teste o contexto chamando servlets e páginas Web localizadas dentro dele

Configuração de instalação

- *Toda aplicação Web possui um arquivo web.xml que configura a implantação de todos os seus componentes*
  - *Configura inicialização de instâncias de servlets*
  - *Define mapeamentos de nomes a servlets (aliases)*
  - *Pode conter instrução para carregar previamente páginas JSP*
  - *Configura inicialização do contexto (aplicação)*
  - *Define permissões e perfis de usuário*
  - *Configura tempo de timeout de sessão*
  - ...
- *Para criar e editar o web.xml, use um editor XML que possa validá-lo através do DTD:* **`web-app_2_3.dtd`**
  - *O DTD é comentado e explica para que serve cada elemento*
  - *Ache-o em:* **`java.sun.com/dtd/web-app_2_3.dtd`**
  - *Instale o DTD em sua máquina local e use-o para validar seu documento caso não esteja conectado à Internet*

XML e DTDs

- *Um DTD (Document Type Definition) declara todos os elementos e atributos de um documento XML*
  - *Define quais elementos e atributos são válidos e em que contexto*
  - *A sintaxe é SGML. Para definir um elemento:*

    ```
    <!ELEMENT nome-do-elemento (modelo de conteudo)>
    ```

  - *O DTD do web.xml define principalmente elementos*
- *Exemplo: DTD para um documento simples*

```
<!ELEMENT pessoa (nome, web?, telefone+)>
<!ELEMENT nome (prenome, inicial*, sobrenome)>
<!ELEMENT prenome (#PCDATA)>
<!ELEMENT inicial (#PCDATA)>
<!ELEMENT sobrenome (#PCDATA)>
<!ELEMENT web (email|website)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT website (#PCDATA)>
<!ELEMENT telefone (#PCDATA)>
```

pessoa *tem* **nome**, *seguido de zero ou um* **web** *e um ou mais* **telefone**

nome *tem um* **prenome**, *seguido de zero ou mais* **inicial** *e um* **sobrenome**

web *pode conter ou um* **email** *ou um* **website**

Elementos que só podem conter texto

Documentos válidos segundo o DTD

- *Os documentos abaixo são válidos segundo o DTD mostrado na página anterior*

```
<pessoa>
  <nome><prenome>Giordano</prenome>
        <sobrenome>Bruno</sobrenome></nome>
  <telefone>1199343232</telefone>
</pessoa>
```

```
<pessoa>
  <nome><prenome>Giordano</prenome>
        <sobrenome>Bruno</sobrenome></nome>
  <web><website>www.site.com</website></web>
  <telefone>1199343232</telefone>
</pessoa>
```

```
<pessoa>
  <nome><prenome>Giordano</prenome>
        <inicial>F</inicial><inicial>R</inicial>
        <sobrenome>Bruno</sobrenome></nome>
  <web><email>giordano@web.net</email></web>
  <telefone>1199343232</telefone>
  <telefone>1134999992</telefone>
</pessoa>
```

Documentos inválidos segundo o DTD

- *Os documentos abaixo não são válidos de acordo com o DTD.*
- *Por que?*

```
<pessoa>
  <nome><prenome>Giordano</prenome>
        <sobrenome>Bruno</sobrenome></nome>
</pessoa>
```

```
<pessoa>
  <nome><prenome>Giordano</prenome>
        <sobrenome>Bruno</sobrenome></nome>
  <web><website>www.site.com</website>
       <email>giordano@web.net</email></web>
  <telefone>1199343232</telefone>
</pessoa>
```

```
<pessoa>
  <nome><prenome>Giordano</prenome>
        <sobrenome>Bruno</sobrenome></nome>
  <telefone>1199343232</telefone>
  <telefone>1134999992</telefone>
  <web><email>giordano@web.net</email></web>
</pessoa>
```

Elemento <!DOCTYPE>

- O elemento <!DOCTYPE> é um elemento do DTD que deve ser usado dentro da página XML
  - Identifica o elemento raiz
  - Associa o arquivo a um DTD através de uma URL e/ou de um identificador formal público
- <!DOCTYPE> deve aparecer no início do documento XML

```
<!DOCTYPE pessoa SYSTEM "/docs/dtd/pessoa.dtd">
<pessoa>
...
</pessoa>
```

Deve ser o mesmo

Onde está o DTD

- Alguns DTDs possuem um identificador formal público (FPI)
  - Neste caso, são declarados com a palavra PUBLIC e duas strings: o identificador seguido de uma URL onde pode ser encontrado

```
<!DOCTYPE pessoa  PUBLIC  "-//ACME, Inc.//DTD Pessoa//PT"
                          "http://localhost/pessoa.dtd">
```

Arquivo web.xml

- O arquivo web.xml necessita de declaração <!DOCTYPE> pública, que tem a seguinte sintaxe

```
<!DOCTYPE web-app PUBLIC
        "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
        "http://java.sun.com/dtd/web-app_2_3.dtd">
```

- O identificador formal deve ser sempre o mesmo. A URL pode ser alterada para apontar para um caminho local ou outro endereço, se necessário
- Uma aplicação Web sempre tem um arquivo web.xml. Se não for necessária configuração alguma em seus servlets e JSPs, pode-se usar o web.xml mínimo:

```
<!DOCTYPE web-app PUBLIC
        "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
        "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app></web-app>
```

Elementos XML do web.xml

- Consulte o DTD (faz parte da especificação) para maiores detalhes sobre os elementos que podem ser usados no web.xml
  - A ordem dos elementos é importante: se eles estiverem na ordem incorreta, o XML não validará com o DTD
  - Existem elementos que exigem a presença de outros (e isto não é validado pelo DTD - leia os comentários no DTD)
- Use um editor XML
  - O JEdit habilitado com plug-in XML valida qualquer XML com seu DTD desde que se informe a localização do DTD em sua declaração <!DOCTYPE> (no web.xml, altere, se necessário, o segundo argumento, que representa a URL onde o DTD pode ser achado)
  - Se estiver online, o JEdit achará o DTD no site da Sun. Se não estiver, copie-o e use-o localmente
  - Não tente instalar um contexto sem antes validar o seu web.xml. Se ele não estiver válido a instalação não irá funcionar.

Exemplo de web.xml (1/2)

```
<web-app>
  <context-param>
    <param-name>tempdir</param-name>
    <param-value>/tmp</param-value>
  </context-param>

  <servlet>
    <servlet-name>myServlet</servlet-name>
    <servlet-class>exemplo.pacote.MyServlet</servlet-class>
    <init-param>
      <param-name>datafile</param-name>
      <param-value>data/data.txt</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet>
    <servlet-name>myJSP</servlet-name>
    <jsp-file>/myjsp.jsp</jsp-file>
    <load-on-startup>2</load-on-startup>
  </servlet>
                    ...
```

Parâmetro que pode ser lido por todos os componentes

Instância de um Servlet

Parâmetro que pode ser lido pelo servlet

Ordem para carga prévia do servlet

Declaração opcional de página JSP

Ordem para pré-compilar JSP

Exemplo de web.xml (2/2)

```
...
  <servlet-mapping>
    <servlet-name>myServlet</servlet-name>
    <url-pattern>/myservlet</url-pattern>
  </servlet-mapping>

  <session-config>
    <session-timeout>60</session-timeout>
  </session-config>

  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>

  <error-page>
    <error-code>404</error-code>
    <location>/notFound.jsp</location>
  </error-page>

</web-app>
```

Servlet examples.MyServlet foi mapeado à URL /myservlet

Sessão do usuário expira expira em 60 minutos

Lista de arquivos que serão carregados automaticamente em URLs terminadas em diretório

Redirecionar para esta página em caso de erro 404

Exercícios

- 2. Use os arquivos XML exemplo mostrados e valide-os com o DTD para comprovar se realmente são válidos ou não
  - Inclua uma declaração
    <!DOCTYPE pessoa SYSTEM "pessoa.dtd">
- 3. Pesquise o DTD do web.xml, descubra para que servem e use os elementos:
  - <welcome-file-list>
  - <description>
  - <error-page>

Instâncias de servlets

- *Uma instância de um servlet pode ser configurada no web.xml através do elemento <servlet>*

```
<servlet>
   <servlet-name>myServlet</servlet-name>
   <servlet-class>exemplo.pacote.MyServlet</servlet-class>
   <!-- elementos de configuração opcionais aqui -->
</servlet>
```

  - *<servlet-name> e <servlet-class> são obrigatórios*
- *É uma boa prática escolher nomes de servlets seguindo as convenções Java*
  - *Use caixa mista, colocando em maiúsculas cada palavra, mas comece com letra minúscula. Ex: banco, pontoDeServico*
- *Pode-se criar múltiplas instâncias da mesma classe definindo blocos servlet com <servlet-name> diferentes*
  - *Não terão muita utilidade a não ser que tenham também configuração diferente e mapeamentos diferentes*

Servlet alias (mapeamento) no web.xml

- *É uma boa prática definir aliases para os servlets*
  - *Nomes grandes são difíceis de digitar e lembrar*
  - *Expõem detalhes sobre a implementação das aplicações*
- *Para definir um mapeamento de servlet é necessário usar* *<servlet>* e *<servlet-mapping>*
- *<servlet-mapping> associa o nome do servlet a um padrão de URL relativo ao contexto. A URL pode ser*
  - *Um caminho relativo ao contexto iniciando por /*
  - *Uma extensão de arquivo, expresso da forma *.extensao*

```
<servlet>
    <servlet-name>umServlet</servlet-name>
    <servlet-class>pacote.subp.MeuServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>umServlet</servlet-name>
    <url-pattern>/programa</url-pattern>
</servlet-mapping>
```

Sintaxe de mapeamentos

- *Mapeamento exato*
  - *Não aceita /nome/ ou /nome/x na requisição*

```
<url-pattern>/nome</url-pattern>
<url-pattern>/nome/subnome</url-pattern>
```

- *Mapeamento para servlet default*
  - *Servlet é chamado se nenhum dos outros mapeamentos existentes combinar com a requisição*

```
<url-pattern>/</url-pattern>
```

- *Mapeamento de caminho*
  - *Aceita texto adicional (path info) após nome do servlet na requisição*

```
<url-pattern>/nome/*</url-pattern>
<url-pattern>/nome/subnome/*</url-pattern>
```

- *Mapeamento de extensão*
  - *Arquivos com a extensão serão redirecionados ao servlet*

```
<url-pattern>*.ext</url-pattern>
```

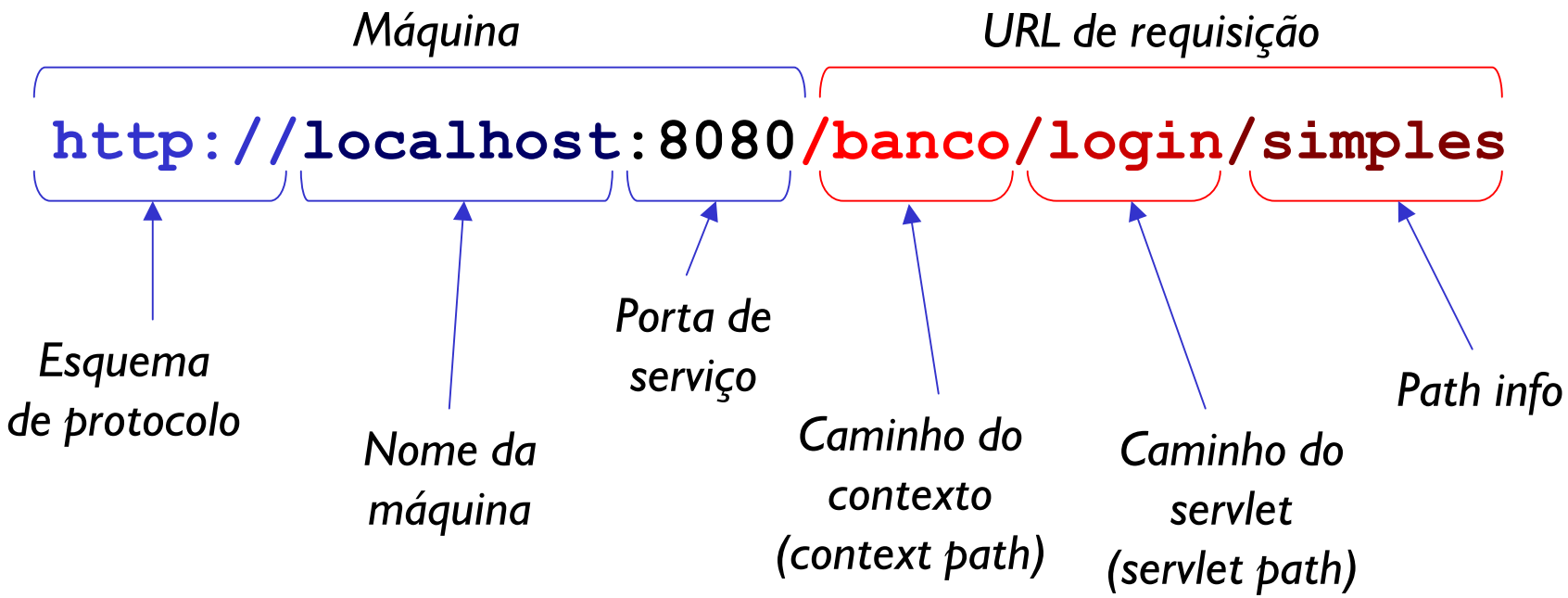
Processamento de URLs e mapeamentos

- O Web container procura dentre os mapeamentos existentes no web.xml, o maior que combine com a URL recebida
  1. Procura primeiro um mapeamento exato
  2. Se não achar, procura entre os caminhos que terminam em `*`.
  3. Por último, procura pela extensão do arquivo, se houver
  4. Não havendo combinação, redireciona ao servlet default se este tiver sido declarado ou exibe um erro, se não houver servlet default
- Após uma combinação, texto adicional à direita na URL recebida será considerado path info
  - Pode ser recuperado com `request.getPathInfo()`
- Considere, por exemplo, o mapeamento `/um/*` e a URL

  `http://localhost:8080/contexto/um/dois/tres/abc.txt`
- Mesmo que exista um mapeamento para `*.txt`, este não será considerado pois antes haverá combinação para `/um`
  - `/dois/tres/abc.txt` é path info!

Anatomia de uma URL

- *Diferentes partes de uma URL usada na requisição podem ser extraídas usando métodos de HttpServletRequest*
  - `getContextPath()`: */banco*, *na URL abaixo*
  - `getServletPath()`: */login*, *na URL abaixo*
  - `getPathInfo()`: */simples*, *na URL abaixo*



Exercícios

- 4. Crie uma aplicação Web chamada miniforum
  - Construa a estrutura com WEB-INF e web.xml
- 5. Escreva um servlet (j550.miniforum.ForumServlet) que receba dois parâmetros: uma mensagem e o e-mail de quem a enviou
  - Crie um formulário HTML novaMensagem.html
  - Grave os dados em um arquivo
  - Mostre na tela a mensagem enviada como resposta
- 6. Escreva um servlet (j550.miniforum.MensagemServlet) que liste todas as mensagens
- 7. Faça mapeamentos no web.xml para que os dois servlets possam ser chamados pelas seguintes URLs no HTML:
  - /forum/gravar - para ForumServlet
  - /forum/listar - para MensagemServlet

ServletConfig

- A interface ServletConfig serve para que um servlet possa ter acesso a informações de configuração definidas no web.xml
- Todo servlet implementa ServletConfig e, portanto, tem acesso aos seus métodos
- Principais métodos de interesse
  - String getInitParameter(String nome): lê um parâmetro de inicialização <init-param> do web.xml
  - Enumeration getInitParameterNames(): obtém os nomes de todos os parâmetros de inicialização disponíveis
- Os métodos de ServletConfig devem ser chamados no método init(), do servlet

Definição de parâmetros de inicialização

- Parâmetros de inicialização podem ser definidos para cada instância de um servlet usando o elemento <init-param> dentro de <servlet>
  - Devem aparecer depois de <servlet-name> e <servlet-class> (lembre-se que a ordem foi definida no DTD)
  - Requer dois sub-elementos que definem o nome do atributo e o seu valor

```
<servlet>
    <servlet-name>umServlet</servlet-name>
    <servlet-class>pacote.subp.MeuServlet</servlet-class>
    <init-param>
      <param-name>dir-imagens</param-name>
      <param-value>c:/imagens</param-value>
    </init-param>
    <init-param> ... </init-param>
  </servlet>
```

Leitura de parâmetros de inicialização

- *Parâmetros de inicialização podem ser lidos no método init() e guardados em variáveis de instância para posterior uso dos métodos de serviço*

```
private java.io.File dirImagens = null;

public void init() throws ServletException {
    String dirImagensStr =
              getInitParameter("dir-imagens");
    if (dirImagens == null) {
        throw new UnavailableException
                    ("Configuração incorreta!");
    }
    dirImagens = new File(dirImagensStr);
    if (!dirImagens.exists()) {
       throw new UnavailableException
                    ("Diretorio de imagens nao existe!");
    }
}
```

Exercícios

- 8. Guarde a cor do texto e a cor de fundo da página que mostra as mensagens como parâmetros de inicialização do servlet j550.cap03.MensagemServlet
  - Use cores HTML (red, blue, #FF0000, ou CSS)
  - Leia os parâmetros na inicialização e guarde-os em variáveis de instância
  - Monte a página HTML com os dados dos parâmetros
- 9. Crie uma segunda instância do mesmo servlet MensagemServlet (use outro nome e outro mapeamento no web.xml)
  - Defina os mesmos parâmetros com valores diferentes
  - Chame o segundo servlet e veja os resultados

Três níveis de configuração

- 1. Nível do servidor (não faz parte da especificação)
  - Configuração definida fora do contexto,em arquivos de configuração do fabricante (ex: server.xml, jboss-web.xml) ou na configuração de um arquivo EAR (J2EE)
  - Permite configuração do nome do contexto raiz
- 2. Nível da aplicação (contexto) - web.xml
  - Aplicada a todos os componentes da aplicação
  - Lista de componentes, mapeamentos, variáveis compartilhadas, recursos e beans compartilhados, serviços compartilhados, timeout da sessão, etc.
- 3. Nivel do componente (servlet, JSP) - <servlet>
  - Configuração de servlets, filtros e páginas individuais
  - Parâmetros iniciais, regras de carga, etc.

ServletContext

- A interface ServletContext encapsula informações sobre o contexto ou aplicação
- Cada servlet possui um método getServletContext() que devolve o contexto atual
  - A partir de uma referência ao contexto atual pode-se interagir com o contexto e compartilhar informações entre servlets
- Principais métodos de interesse de ServletContext
  - String getInitParameter(String): lê parâmetros de inicialização do contexto (não confunda com o método similar de ServletConfig!)
  - Enumeration getInitParameterNames(): lê lista de parâmetros
  - InputStream getResourceAsStream(): lê recurso localizado dentro do contexto como um InputStream
  - setAttribute(String nome, Object): grava um atributo no contexto
  - Object getAttribute(String nome): lê um atributo do contexto
  - log(String mensagem): escreve mensagem no log do contexto

Inicialização de contexto

- No web.xml, <context-param> vem antes de qualquer definição de servlet

```
<context-param>
    <param-name>tempdir</param-name>
    <param-value>/tmp</param-value>
</context-param>
```

- No servlet, é preciso primeiro obter uma instância de ServletContext antes de ler o parâmetro

```
ServletContext ctx = getServletContext();
String tempDir = ctx.getInitParameter("tempdir");
if (tempDir == null) {
   throw new UnavailableException("Configuração errada");
}
```

Carregamento de arquivos no contexto

- *O método getResourceAsStream() permite que se localize e se carregue qualquer arquivo no contexto sem que seja necessário saber seu caminho completo*
  - *Isto é importante pois contextos podem ser usados em diferentes servidores e armazenados em arquivos WAR*
- *Exemplo*

```
ServletContext ctx = getServletContext();
String arquivo = "/WEB-INF/usuarios.xml";
InputStream stream = ctx.getResourceAsStream(arquivo);
InputStreamReader reader =
                    new InputStreamReader(stream);
BufferedReader in = new BufferedReader(reader);
String linha = "";
while ( (linha = in.readLine()) != null) {
   // Faz alguma coisa com linha de texto lida
}
```

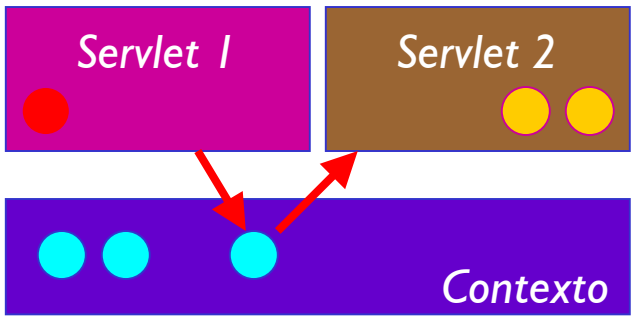
Logging

- *Para cada contexto criado, o servidor fornece um arquivo de log, onde mensagens serão gravadas*
  - *O arquivo só passará a existir quando a primeira mensagem for gravada. Procure em logs/ no Tomcat.*
- *Há dois métodos disponíveis*
  - `log(String mensagem)`: *grava uma mensagem*
  - `log(String mensagem, Throwable exception)`: *grava uma mensagem e o stack-trace de uma exceção. Use nos blocos try-catch.*
- *Log é implementado em GenericServlet também. Pode, portanto, ser chamado tanto do contexto como do servlet*

```
log("Arquivo carregado com sucesso.");
ServletContext ctx = getServletContext();
ctx.log("Contexto obtido!");
```

Gravação de atributos no contexto

- *Servlets podem compartilhar objetos pelo contexto usando*
  - `setAttribute("nome", objeto);`
  - `Object getAttribute("nome");`



- *Exemplo de uso*

Servlet 1

```
String[] vetor = {"um", "dois", "tres"};
ServletContext ctx = getServletContext();
ctx.setAttribute("dados", vetor);
```

Servlet 2

```
ServletContext ctx = getServletContext();
String[] dados = (String[])ctx.getAttribute("dados");
```

- *Outros métodos*
  - `removeAttribute(String nome)` - *remove um atributo*
  - `Enumeration getAttributeNames()` - *lê nomes de atributos*

ServletContextListener

- Não existem métodos init() ou destroy() globais para realizar operações de inicialização/destruição de um contexto
  - A forma de controlar o ciclo de vida global para um contexto é através da implementação de um ServletContextListener
- ServletContextListener é uma interface com dois métodos
  - public void contextInitialized(ServletContextEvent e)
  - public void contextDestroyed(ServletContextEvent e)

  que são chamados respectivamente depois que um contexto é criado e antes que ele seja destruído. Para isto é preciso registrá-lo no web.xml usando o elemento <listener>

```
<listener>
   <listener-class>ex01.OuvinteDeContexto</listener-class>
</listener>
```

- ServletContextEvent possui um método getServletContext() que permite obter o contexto associado

Outros listeners de contexto

- É possível saber quando um atributo foi adicionado a um contexto usando ServletContextAttributeListener e ServletContextAttributeEvent
- Métodos a implementar do Listener
  - attributeAdded(ServletContextAttributeEvent e)
  - attributeRemoved(ServletContextAttributeEvent e)
  - attributeReplaced(ServletContextAttributeEvent)
- ServletContextAttributeEvent possui dos métodos para recuperar nome e valor dos atributos
  - String getName()
  - String getValue()
- É preciso registrar o listener no web.xml

Contexto externo no Tomcat

- *Sempre que possível, crie seus contextos no webapps*
- *Se for necessário, é possível fazer o Tomcat criar contextos em outro lugar, alterando a configuração. Acrescente um elemento <Context> em* `$TOMCAT_HOME/conf/server.xml`

```
<Server ...> ...
   <Service ...> ...
      <Engine ...> ...
         <Host ...>
           ...
             <Context path="/sistema"
                      docBase="c:\sistema\build" />
...
```

Específico do Tomcat!

- *docBase também pode conter endereço relativo a webapps (esses são gerados automaticamente) - veja DTD!*
- *É preciso reiniciar o servidor*

Exercício

- 10. Guarde o nome do arquivo compartilhado por ForumServlet e MensagemServlet como um parâmetro de inicialização de contexto
  - Guarde o arquivo dentro de WEB-INF e o caminho no parâmetro de inicialização
  - Recupere o parâmetro no init() de seu servlet e guarde-o em uma variável de instância. Cause uma Unavailable Exception caso o parâmetro seja null.
  - Use getResourceAsStream() para recuperar um stream para o arquivo.
- 11. Guarde, como atributo de contexto, um número, e incremente-o a cada acesso
  - Imprima na página o número de acessos.

Introdução a Model View Controller

- Apesar de servlets não separarem código de resposta do código de requisição explicitamente, isto pode ser feito pelo desenvolvedor
  - Melhor separação de responsabilidades: cada método cuida de uma coisa - ou lógica de negócios, ou controle de requisição ou geração de respostas
  - Maior facilidade para migrar para solução JSP-servlet
- Lógica de negócios deve ficar em classes externas, executadas pelos métodos controladores e pesquisadas pelos métodos de geração de resposta
- Controlador deve selecionar o método de resposta adequado após o processamento
  - Dados podem ser passados através da requisição usando atributos de requisição (não use variáveis de instância)

Exemplo: MVC com servlets

```
public class MVCServlet extends HttpServlet {
   private Servico servico;
   public void init() {
      ... inicializa servico
   }
   public void doGet(...) { processar(...);}
   public void doPost(...) { processar(...);}

   public void processar(... request, ... response) {
      Produto p = new Produto(123, "Livro");
      servico.adicionar("produto", p);
      ...
      if (funcionou) {
         request.setAttribute(p);
         sucesso(request, response);
      } ...
   }
   public void sucesso(... request, ... response) {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      Produto p = (Produto) request.getAttribute("produto"); // ...
      out.println("<td>"+p.getCodigo()+"</td>"); // ...
   }
}
```

*Variável de instância compartilhada contendo objeto do **Model***

Controller

*Objetos do **Model** sendo manipulados no Controller*

*Objetos do **Model** sendo lidos no View*

View

Passagem de atributos pela requisição

- Para compartilhar dados entre métodos de serviço, não use variáveis estáticas ou de instância
  - Elas são compartilhadas por todos os clientes!
- Use atributos de requisição (HttpServletRequest)
  - `setAttribute("nome", objeto);`
  - `Object getAttribute("nome");`
- Atributos são destruídos junto com a requisição
  - Não são compartilhados entre clientes
  - É a forma recomendada de comunicação entre métodos de serviço e objetos na mesma requisição
  - Se desejar reter seu valor além da requisição, copie-os para um objeto de persistência maior (por exemplo, um atributo de contexto)

Escopo e threads

- Geralmente, só há uma instância de um servlet rodando para vários clientes
  - Atributos de instância são compartilhados!
- Se não desejar compartilhar dados entre clientes, use sempre objetos thread-safe
  - Atributos guardados no request
  - Variáveis locais
- Quaisquer outros atributos, como atributos de sessão, atributos de instância e de contexto são compartilhados entre requisições
  - Caso deseje compartilhá-los, use synchronized nos blocos de código onde seus valores são alterados.

Exercício

- 12. Altere MensagemServlet e ForumServlet para que contenham apenas lógica de processamento Web, delegando as operações lógicas com mensagens para a classe Mensagem
  - Acesso ao arquivo deve estar em Mensagem
  - Servlets devem criar objeto e manipular métodos de Mensagem: get/setTexto(), get/setEmail() sem se preocupar com arquivos
  - Passe o stream como argumento do construtor de Mensagem
- 13. Separe os métodos relacionados com saída (View) dos métodos relacionados com entrada e processamento (Controller e Model)
  - Métodos Controller controlam a requisição, chamam métodos de processamento no Model (criam objeto, preenchem dados) e escolhem o método de View desejado
  - Métodos View lêem o Model (getXXX()) e geram a resposta
  - Passe objetos entre os métodos usando atributos de requisição

Repasse de requisição

- Objetos *RequestDispatcher* servem para repassar requisições para outra página ou servlet. Seus dois principais métodos são

```
include(request, response)
```

```
forward(request, response)
```

- Esses métodos não podem definir cabeçalhos
  - *forward()* repassa a requisição para um recurso
  - *include()* inclui a saída e processamento de um recurso no servlet
- Para obter um RequestDispatcher use o ServletRequest

```
RequestDispatcher dispatcher =
  request.getRequestDispatcher("url");
```

- Para repassar a requisição para outra máquina use

```
dispatcher.forward(request, response);
```

- No repasse de requisição, o controle não volta para o browser.
  - Todos os parâmetros e atributos da requisição são preservados

Redirecionamento x Repasse

- Pode-se enviar um cabeçalho de redirecionamento para o browser usando

```
response.sendRedirect("url");
```

- Isto é o mesmo que fazer

```
response.setHeader("Location", "url");
```

- Location é um cabeçalho HTTP que instrui o browser para redirecionar para outro lugar
- Sempre que o controle volta ao browser, a primeira requisição terminou e outra foi iniciada
  - Os objetos HttpServletResponse e HttpServletRequest e todos seus atributos e parâmetros foram destruídos
- Com repasse de requisições, usando RequestDispatcher, o controle não volta ao browser mas continua em outro servlet (com forward())ou no mesmo servlet (com include())

Exercícios

- 14. Crie j550.miniforum.PortalServlet que redirecione a resposta para MensagemServlet ou formulário para entrada de mensagens dependendo do comando recebido como parâmetro
  - Implemente dois links para o mesmo servlet: Listar Mensagens e Nova Mensagem
  - comando=listar - redirecione para /forum/listar
  - comando=criar - redirecione para /forum/gravar (ou para forum/novaMensagem.html)
  - Crie um mapeamento para o servlet: /miniforum/portal
- 15. Use repasse de requisição para refazer o exercício 14.

helder@acm.org

argonavis.com.br