Módulo 6

Programação WEB

Lição 1

Introdução à Programação WEB

Versão 1.0 - Nov/2007

Autor
Daniel Villafuerte

Equipe
Rommel Feria
John Paul Petines

Necessidades para os Exercícios

Sistemas Operacionais Suportados

NetBeans IDE 5.5 para os seguintes sistemas operacionais:

- Microsoft Windows XP Profissional SP2 ou superior
- Mac OS X 10.4.5 ou superior
- Red Hat Fedora Core 3
- Solaris™ 10 Operating System (SPARC® e x86/x64 Platform Edition)

NetBeans Enterprise Pack, poderá ser executado nas seguintes plataformas:

- Microsoft Windows 2000 Profissional SP4
- Solaris™ 8 OS (SPARC e x86/x64 Platform Edition) e Solaris 9 OS (SPARC e x86/x64 Platform Edition)
- Várias outras distribuições Linux

Configuração Mínima de Hardware

Nota: IDE NetBeans com resolução de tela em 1024x768 pixel

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	500 MHz Intel Pentium III workstation ou equivalente	512 MB	850 MB
Linux	500 MHz Intel Pentium III workstation ou equivalente	512 MB	450 MB
Solaris OS (SPARC)	UltraSPARC II 450 MHz	512 MB	450 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Série 1.8 GHz	512 MB	450 MB
Mac OS X	PowerPC G4	512 MB	450 MB

Configuração Recomendada de Hardware

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	1 GB
Linux	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	850 MB
Solaris OS (SPARC)	UltraSPARC IIIi 1 GHz	1 GB	850 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Series 1.8 GHz	1 GB	850 MB
Mac OS X	PowerPC G5	1 GB	850 MB

Requerimentos de Software

NetBeans Enterprise Pack 5.5 executando sobre Java 2 Platform Standard Edition Development Kit 5.0 ou superior (JDK 5.0, versão 1.5.0_01 ou superior), contemplando a Java Runtime Environment, ferramentas de desenvolvimento para compilar, depurar, e executar aplicações escritas em linguagem Java. Sun Java System Application Server Platform Edition 9.

- Para **Solaris**, **Windows**, e **Linux**, os arquivos da JDK podem ser obtidos para sua plataforma em http://java.sun.com/j2se/1.5.0/download.html
- Para **Mac OS X**, Java 2 Plataform Standard Edition (J2SE) 5.0 Release 4, pode ser obtida diretamente da Apple's Developer Connection, no endereço: http://developer.apple.com/java (é necessário registrar o download da JDK).

Para mais informações: http://www.netbeans.org/community/releases/55/relnotes.html

Colaboradores que auxiliaram no processo de tradução e revisão

Aécio Júnior
Alexandre Mori
Alexis da Rocha Silva
Allan Souza Nunes
Allan Wojcik da Silva
Angelo de Oliveira
Aurélio Soares Neto
Bruno da Silva Bonfim
Carlos Fernando Gonçalves
Denis Mitsuo Nakasaki
Emanoel Tadeu da Silva Freitas
Felipe Gaúcho
Jacqueline Susann Barbosa
João Vianney Barrozo Costa
Luciana Rocha de Oliveira
Luiz Fernandes de Oliveira Junior
Marco Aurélio Martins Bessa
Maria Carolina Ferreira da Silva
Massimiliano Giroldi
Mauro Cardoso Mortoni
Paulo Oliveira Sampaio Reis
Pedro Henrique Pereira de Andrade
Ronie Dotzlaw
Sergio Terzella
Thiago Magela Rodrigues Dias
Vanessa dos Santos Almeida
Wagner Eliezer Roncoletta

Auxiliadores especiais

Revisão Geral do texto para os seguintes Países:

- **Brasil** – Tiago Flach
- **Guiné Bissau** – Alfredo Cá, Bunene Sisse e Buon Olossato Quebi – ONG Asas de Socorro

Coordenação do DFJUG

- **Daniel deOliveira** – JUGLeader responsável pelos acordos de parcerias
- **Luci Campos -** Idealizadora do DFJUG responsável pelo apoio social
- **Fernando Anselmo -** Coordenador responsável pelo processo de tradução e revisão, disponibilização dos materiais e inserção de novos módulos
- **Rodrigo Nunes -** Coordenador responsável pela parte multimídia
- **Sérgio Gomes Veloso -** Coordenador responsável pelo ambiente JEDI™ (Moodle)

Agradecimento Especial

John Paul Petines – Criador da Iniciativa JEDI™
Rommel Feria – Criador da Iniciativa JEDI™

1. Objetivos

Bem-vindo a este curso de programação WEB. Começaremos com uma ampla introdução, pois é interesse das companhias (empresas) e dos programadores, conhecer sobre programação para a WEB.

Ao final desta lição, o estudante será capaz de:

- Descrever como funciona a WEB
- Definir a arquitetura do tipo Cliente-Servidor
- Entender sobre o protocolo HTTP
- Definir o básico sobre a arquitetura Java EE
- Saber o que são *Servlets* e *Java Server Pages*

2. Porque migrar para WEB?

2.1. Ambiente de Tecnologia Neutra

Um dos principais benefícios em aplicações para a Internet, é que a Internet é um ambiente de tecnologia neutra. A comunicação em qualquer aplicação na WEB é feita através de protocolos populares (HTTP/FTP), que não requerem que o usuário nem o cliente tenham qualquer sistema operacional em particular instalado em sua máquina ou seja desenvolvida em uma linguagem de programação específica. Tudo que os clientes necessitam é de um navegador WEB (denominado *Browser*), o acesso a aplicação e qualquer sistema operacional. Isto traz diversas possibilidades dentro de uma ampla gama de aplicações baseadas na WEB.

2.2. Facilidade de distribuição e atualização

Visto que o navegador WEB é o único programa instalado que os usuários irão necessitar, não há necessidade de fornecer programas adicionais através de CDs ou outra mídia. Assim, como não existe a necessidade do usuário repassar uma seqüência de instalação, possivelmente demorada, o que será necessário é o local e acesso a aplicação na internet, e tudo estará pronto para funcionar.

Outro benefício de se ter o código binário exato do programa, que residirá em um único servidor acessível, ao invés de instalado no computador do usuário, pois evita-se possíveis problemas comuns relatados com atualizações de programas, tais como a necessidade de checar periodicamente novas versões de programas. O problema de conseguir novas atualizações dos programas e eliminado completamente. O usuário não precisa ser informado sobre uma atualização do programa; tudo que será necessário é atualizar o código no servidor WEB e, automaticamente, todos os usuários irão usufruir da nova versão.

3. Arquitetura Cliente-Servidor

3.1. Cliente Pesado e Cliente Magro

Uma aplicação WEB é um tipo de aplicação que faz uso da chamada estrutura Cliente-Servidor. Neste tipo, um programa cliente conecta a um servidor para acessar informações que necessita para completar as tarefas que os usuários desejam para realizar. Há os que são chamados **clientes magros**, e os que são chamados **clientes pesados**.

Clientes magros são os que contém apenas o mínimo necessário para que o usuário execute o sistema. Em geral, é apenas uma casca. Todos as regras de negócio, dados, exceto os que são fornecidos pelo usuário, residem dentro de um servidor.

Clientes pesados são os que contém, além de uma interface, alguns, se não muitos, dos processos de regra de negócio requeridos pelas tarefas específicas dos usuários.

3.2. Arquitetura Cliente-Servidor na visão WEB

Na definição acima, podemos citar que o cliente utilizado para aplicações WEB são os que nós chamamos de **clientes magros**. O programa cliente, um navegador, neste caso, é apenas uma interface que o usuário utiliza para realizar tarefas. Tudo mais, como os dados que o usuário precisa para operar num determinado fluxo lógico de execução do programa, reside no servidor. De uma perspectiva mais focada na WEB, estas são as funções do servidor e do cliente:

3.2.1. Servidor WEB

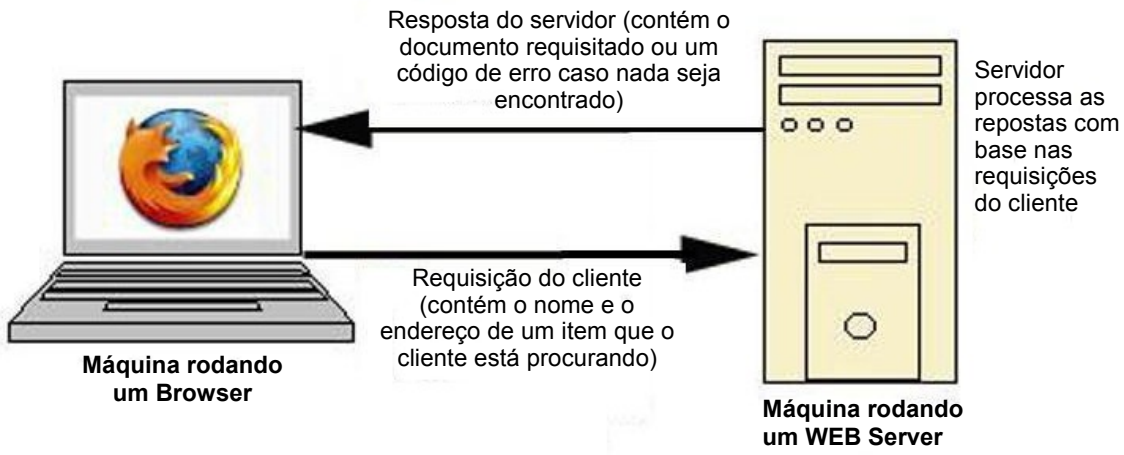

Figura 1: Funções do Servidor e do Cliente

O servidor recebe uma requisição do cliente através do navegador WEB e retorna uma resposta. Qualquer requisição vinda de um cliente inclui o nome e o endereço do item que o cliente está procurando, assim como, qualquer dado fornecido pelo usuário. O servidor assimila a requisição, a processa e retorna uma resposta dos dados procurados pelo cliente ou exibe um código de erro indicando que o item requisitado não existe no servidor.

3.2.2. Cliente WEB

É da responsabilidade do navegador prover ao usuário uma interface para exibir a resposta fornecida pelo servidor. Quando o usuário emite uma requisição para o servidor, por exemplo, buscar um documento ou submeter um formulário, o navegador deve enviar esta requisição de modo que o servidor possa entender.

Uma vez que o servidor finalizou o processo requisitado e enviou uma resposta, o navegador, que recebeu os dados requeridos da resposta do servidor, mostra a página resultante ao usuário.

4. HTML

4.1. Como o browser sabe o que deve ser exibido ao usuário?

A maioria dos sites WEB não tem apenas um conteúdo simples de texto. Ao invés disso, emprega gráficos ou tem formas que acessam dados.

4.2. Como o navegador sabe o que deverá exibir?

A resposta reside em HTML, que são as iniciais para ***H****yper****t****ext* ***M****arkup* ***L****anguage*. HTML pode ser pensado como um conjunto de instruções para um navegador WEB que define como apresentar conteúdo para o usuário. É um padrão aberto, atualizado pela W3C ou a *World Wide Web Consortium*. Sendo este um padrão aberto, qualquer um pode acessá-lo. Isto também significa que os navegadores são desenvolvidos sobre esse padrão. Isso significa que todos os navegadores sabem o que fazer quando se deparam com HTML, embora alguns navegadores mais antigos possam ter problemas em interpretar algumas páginas que foram escritas usando novas versões de HTML que foram criadas após o desenvolvimento destes.

5. HTTP

5.1. Definição

HTTP significa ***H**yper**t**ext **T**ransfer **P**rotocol*. É um protocolo de internet de rede de comunicações com características específicas da WEB, que funciona no topo de duas outras camadas de protocolo, TCP e IP.

TCP é um protocolo que é responsável por garantir que um arquivo enviado através de uma rede de comunicações seja entregue completamente e com sucesso no destino. **IP** é um protocolo que encaminha parte dos arquivos de um ponto para outro no seu destino.

O HTTP utiliza estes dois protocolos para ter certeza de que as requisições e respostas serão entregues corretamente ao final de cada pacote transferido.

O protocolo HTTP usa uma seqüência Requisição/Resposta (*Request/Response*): um cliente HTTP abre uma conexão e envia uma mensagem de requisição para um servidor HTTP. O servidor, então, retorna uma mensagem resposta, geralmente contendo o recurso que foi requerido. Após a entrega da resposta o servidor fecha a conexão fazendo uso de um protocolo *stateless HTTP* (não mantém qualquer informação de estado sobre a conexão).

O formato das mensagens *Requisição/Resposta* é semelhante e orientado. Ambos os tipos de mensagem consistem em:

- Um cabeçalho inicial
- Zero ou mais cabeçalhos adicionais
- Uma linha em branca (CRLF)
- O corpo de mensagem (opcional). Ex. Um arquivo, dado ou serviço

5.2. Requisições HTTP

Requisições de um cliente para o servidor contêm a informação sobre o tipo de dado que o usuário necessita. Um dos itens de informação encapsulados no *HTTP Request* é a item *method*. Isto diz ao servidor como que este tipo de requisição está sendo feita, e como o resto da mensagem do cliente está formatada. Há dois métodos que serão encontrados: *GET* e *POST*.

5.3. Método GET

É o método mais simples que é usado principalmente para requisitar um recurso qualquer de um servidor, como uma página WEB, um arquivo de imagem gráfica, um documento, etc.

O método GET também pode ser utilizado para enviar dados para o servidor, embora isto tenha suas limitações. A quantidade de caracteres que podem ser encapsulados em uma requisição GET é limitada, então, para as situações onde muitas informações precisam ser enviadas para o servidor, é provável que nem todas as mensagens "sobreviverão".

Outra limitação do método GET é no envio dos dados. Os dados enviados utilizando este método são simplesmente anexados à **URL** enviada ao servidor.

Por enquanto, pense na URL como um único endereço que é enviado ao servidor. Denota o destino ao qual será enviada a requisição. Um dos problemas encontrados neste método são as muitas URLs que podem ser exibidas na barra de navegação dos navegadores. Isto significa que dados confidenciais, tais como senhas ou informações do contato, serão expostas para qualquer pessoa.

A vantagem em se utilizar o método GET para enviar dados para o servidor é que a URL pode ser gravada como um *bookmarker* pelo navegador. Isto significa que o usuário pode simplesmente selecionar o item desejado e acessá-lo em vez de ter que passar por todo o processo novamente. Vale ressaltar que isto também pode ser perigoso, se a facilidade do *bookmarker* não é algo que os usuários deveriam ter.

Aqui está o que a URL criada com uma GET Request pode parecer:

```
http://jedi-master.dev.java.net/Servlets/NewsItemView?
newsItemID=2359&filter=true
```

O item antes do sinal de interrogação (?) representa a URL original da requisição (neste caso é http://jedi-master.dev.java.net/Servlets/NewsItemView). Tudo após, são os parâmetros ou dados que são enviados juntos para o servidor. Olharemos mais atentamente para esta segunda parte. Estes são os parâmetros adicionados pela requisição:

```
newsItemID=2359&filter=true
```

Em requisições GET, parâmetros são codificados com pares nomes e valores. Não são enviados valores excedentes de dados para o servidor sem que este conheça especificamente seus nomes. Os pares nomes e valores são codificados como:

```
name=value
```

Além disso, caso exista mais de uma série de parâmetros, estes serão separados utilizando o símbolo &. Neste caso, os nomes dos parâmetros que estudamos são *newsItemID* e *filter*, e os valores 2359 e true, respectivamente.

5.4. Método POST

Outro método de requisição utilizado é o POST. Este tipo de requisição é utilizada de tal forma que as solicitações ao navegador podem se tornar complexas, isto é, para que o usuário, através do navegador, possa enviar muitos dados para o servidor. Formas complexas são geralmente envidas utilizando requisições POST, assim como, também, formas simples.

Uma diferença aparente entre os métodos GET e POST é o modo como eles enviam dados para o servidor. Como declarado antes, o método GET simplesmente anexa os dados à URL enviada. O método POST, por outro lado, esconde os dados dentro do corpo da mensagem enviada. Quando o servidor recebe a requisição e determina que ela é uma POST, procurará no corpo da mensagem pelos dados.

5.5. HTTP response (Resposta HTTP)

O *HTTP response* do servidor contém o cabeçalho e o corpo da mensagem, de maneira semelhante ao *HTTP request*. É utilizado um conjunto diferente de cabeçalhos. Os cabeçalhos contêm informações sobre a versão do protocolo HTTP que o servidor está vendo, assim como também o tipo de conteúdo que é escondido dentro do corpo da mensagem. O valor para o tipo de conteúdo é chamado de *MIME-type*. Informa ao navegador que a mensagem contém um HTML, imagem ou outros tipos de conteúdo.

5.6. Páginas Dinâmicas ou Estáticas

O tipo de conteúdo que pode ser oferecido por um servidor WEB pode ser estático ou dinâmico. Conteúdo estático é o conteúdo que não é modificado. Este tipo de conteúdo geralmente não executa nada quando o servidor é acessado e é criado durante sua requisição. Quando estes conteúdos são enviados através da resposta do servidor, são enviados exatamente o mesmo conteúdo armazenado no servidor. Exemplos de conteúdos estáticos incluem artigos de jornal arquivados, fotos de família, ou uma cópia de um documento.

O conteúdo dinâmico, por outro lado, muda de acordo com a requisição do cliente. Tais aplicações no servidor acessam este tipo de conteúdo seguindo um modelo pré-determinado, de acordo com o documento a ser enviado.

Este modelo é então preenchido de acordo com os parâmetros enviados pelo usuário e retornam ao cliente. É suficiente dizer que páginas dinâmicas tem muito mais flexibilidade e tem mais utilidade que as páginas estáticas. Aqui estão cenários onde o conteúdo dinâmico será a única que irá atender às necessidades do usuário.

- **A página WEB é baseada em dados submetidos pelo usuário**. Por exemplo, os resultados das páginas de pesquisa são gerados deste modo. Programas que processam

pedidos e sites de comércio fazem isto muito bem.

- **Dados mudam freqüentemente**. Uma página com a previsão do tempo ou novas notícias atualizadas constantemente podem construir a página dinamicamente, talvez retornando a uma página previamente construída se ela ainda estiver atualizada.
- **A página WEB utiliza informações de um banco de dados**. Realizado pelo acesso e busca ao banco de dados da empresa.

É importante perceber que o servidor WEB sozinho não tem capacidade de apresentar conteúdo dinâmico. Os servidores WEB precisaram separar das aplicações as informações que iriam armazenar informações pertinentes ao cliente (tais como dados coletados em formulários) dentro do depósito de páginas. Não se pode esperar que, a partir de um formulário, ter os dados do cliente, quando submetidos ao servidor, este conhecerá automaticamente o que deverá ser feito com estes dados.

Estamos, agora, no ponto da discussão onde podemos, explicitamente, apontar que é esta a questão principal das aplicações WEB e formam a base para o nosso curso.

Neste curso iremos nos voltar, primeiramente, às tecnologias baseadas em Java para criar aplicações WEB. Mais especificamente, faremo um uso excessivo de bibliotecas fornecidas a nível de especificação.

6. Java EE – Visão sobre a camada WEB

A plataforma Java EE foi criada para o desenvolvimento de aplicações corporativas (baseada em componentes). O modelo de aplicação utilizada para esta plataforma é chamado Modelo de Aplicação Multi-Camadas Distribuídas, ou, simplesmente, *multi-tier*. O aspecto de distribuição deste modelo simplesmente significa que a maior parte das aplicações programadas e desenvolvidas com esta plataforma em mente podem ter diferentes componentes instalados em diferentes máquinas. A parte *multi-tier* significa que as aplicações são concebidas visando a separação entre as camadas dos vários componentes importantes da aplicação. Um exemplo de uma aplicação *multi-tier* é uma aplicação WEB: a camada de apresentação (representada pelo navegador), a camada de lógica de negócio (o programa que reside no servidor WEB), e a camada de dados (a base de dados que irá armazenar e gerenciar os dados da aplicação) são distintamente separadas, entretanto são necessários para se criar uma aplicação para o usuário.

Um dos níveis na plataforma Java EE, como previamente mencionado é a *web-tier*. Este nível é destinado a camada que interage com o navegador para criar o conteúdo dinâmico. Existem duas tecnologias dentro desta camada: *Servlets* e *JavaServer Pages – JSP*.

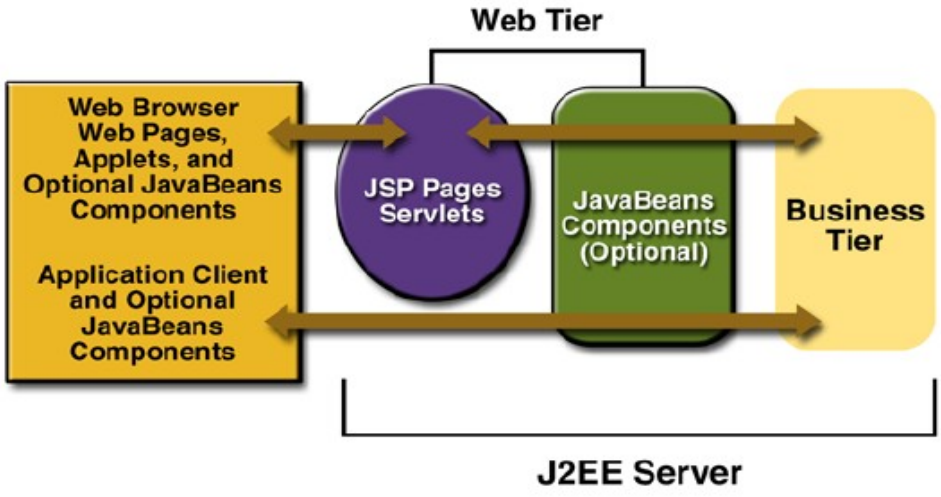

Figura 2: A camada WEB da plataforma Java EE (Imagem do documento J2EE Tutorial)

6.1. Servlets

A tecnologia *Servlet* foi a resposta inicial de Java para acrescentar a funcionalidade adicional aos servidores que utilizavam o modelo requisição/resposta. Possui a habilidade de ler dados contidos na requisição (request) passada ao servidor e gerar uma resposta dinâmica baseada nestes dados.

Servlet não é necessariamente limitada as situações baseadas em HTTP. Como declarado antes, são aplicáveis para qualquer caso que utilize o modelo requisição/resposta. As situações baseadas em HTTP são, atualmente, o uso desta tecnologia. Então, Java forneceu uma versão de classes específicas que implementam as características de HTTP.

6.2. JavaServer Pages

Uma das desvantagens de se utilizar a tecnologia de classes *Servlets* para gerar uma resposta ao cliente, é que primeiro essa resposta deve ser formatada em HTML para ser reenviada. Já que que *Servlets* são simplesmente classes em linguagem de Java, produzem resultados de modo semelhante a que outras classes Java fariam: através da impressão de caracteres em formato String pelo canal de saída, neste caso a *HTTP response*. Entretanto, HTML pode se tornar bastante complexo e ser muito difícil codificar HTML através do uso de uma *String*. Além disso, o emprego de recursos gráficos dedicados e páginas WEB programadas para ajudar na parte estática das

páginas é difícil. Estaríamos supondo que a equipe de desenvolvimento deva ter um bom conhecimento de Java.

Deste modo surgiu a tecnologia *JavaServer Pages*. A JSP se parece com HTML, tendo acesso a todas as habilidades dinâmicas das classes *Servlets* através do uso de *scripts* e linguagem de expressão. Visto que se parece muito com HTML, os projetista podem se concentrar no simples formato HTML e realizar as modificações necessárias e permite que os desenvolvedores ocupem-se do conteúdo dinâmico.

6.3. Contêiner

O conceito central de qualquer aplicação Java EE é o contêiner. Todo componente Java EE, inclui componentes WEB (*Servlet* ou *JSP*) que dependem da existência de um contêiner. Sem o contêiner apropriado, não seriam executados.

Talvez outro modo de explicar isto seria pensar sobre o modo normal de execução dos programas Java. Programas Java, para serem executados, devem ter um método principal definido. Isto marca o começo da execução do programa, sendo este método processado quando o programa é executado na linha de comando.

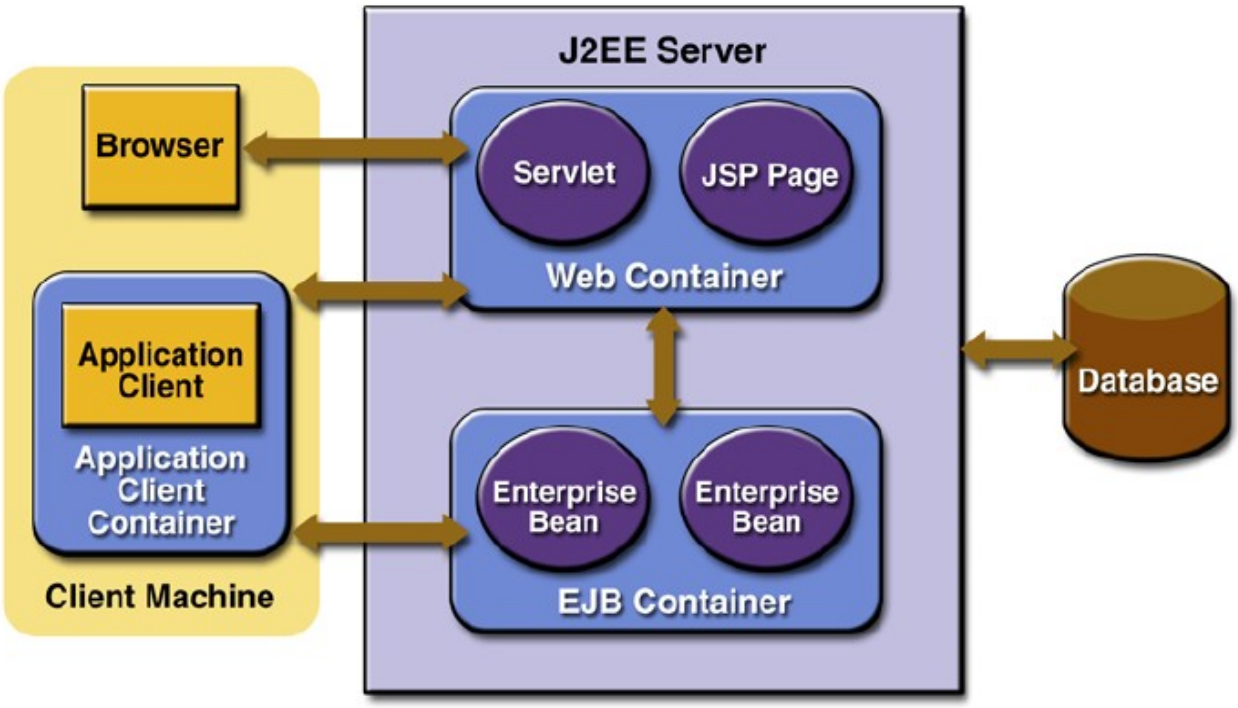

Figura 3: Contêineres da plataforma Java EE (Imagem do documento J2EE Tutorial)

Como veremos mais tarde, a classe *Servlet*, não têm um método principal definido. E se existe algum definido (*bad programming design*) este não marca o começo da execução do programa. Quando o usuário faz uma requisição HTTP (*http request*) para uma classe *Servlet*, seu método não é chamado diretamente.

Em vez disso, o servidor passa a requisição não para a classe *Servlet*, mas para o contêiner na qual a *Servlet* está inserida. O contêiner é o único responsável pela chamada ao método apropriado na *Servlet*, dependendo do tipo de requisição do usuário.

6.4. Vantagens fornecidas pelo contêiner

Suporte de comunicações. O contêiner passa todo código necessário para a *Servlet* para se comunicar com o servidor WEB. Sem o contêiner, desenvolvedores precisariam escrever o código

que iria criar uma conexão *socket* do servidor para a *Servlet* e vice-versa e ainda deveria gerenciar como eles falam um ao outro a cada momento.

Gerenciamento do Ciclo de Vida. O contêiner conhece o que se passa na vida de suas classes *Servlets* desde seu carregamento, instalação, inicialização e recolhimento pelo *Garbage Collector*.

Suporte a múltiplas tarefas. O contêiner controla a função de criar uma nova *thread* a cada momento que uma requisição para a classe *Servlet* é realizada. NOTA: o contêiner NÃO será responsável pelas thread de sua *Servlet*.

Declaração de Segurança. Um contêiner suporta o uso de um arquivo de configuração XML que pode repassar diretivas de segurança para sua aplicação WEB sem precisar de um código complexo qualquer dentro do *Servlet*.

Suporte JSP. Páginas JSP, para funcionarem, devem ser transformadas em uma classe Java (*Servlet*). O contêiner controla a tarefa de traduzir as páginas JSP para uma classe *Servlet*, compilar e executar.

7. Estrutura Básica de uma aplicação WEB

Para um determinado contêiner reconhecer sua aplicação como aplicação WEB válida, esta deve se adequar a uma estrutura específica de diretório conforme a figura a seguir.

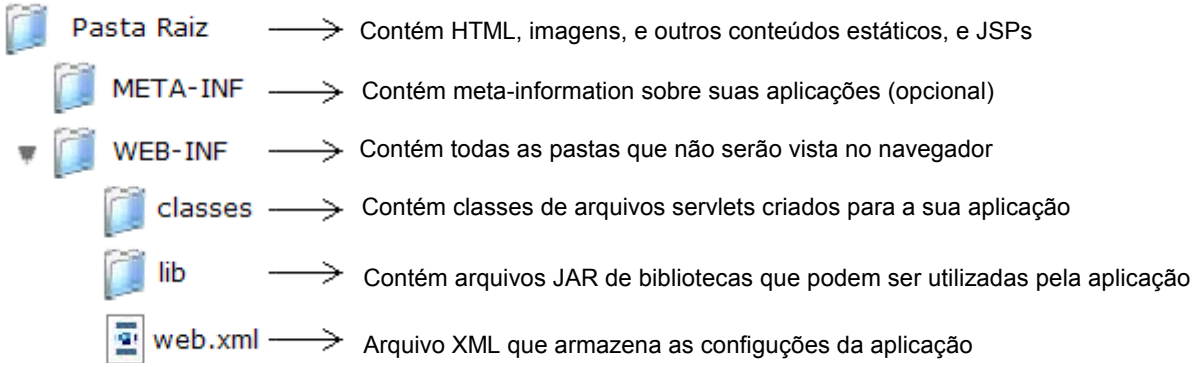

Figura 4: Estrutura do Directório de uma aplicação Java WEB

A figura mostra a estrutura do diretório requisitado pelo contêiner para que sua aplicação seja reconhecer. Alguns pontos relativos a esta estrutura são:

1. A **Pasta Raiz** (contém a aplicação) não precisa ser nomeada como Pasta Raiz. Pode ser, qualquer nome que se deseje, embora seja altamente recomendado que o nome desta pasta seja o nome da sua aplicação.
2. Qualquer outra pasta pode ser criada dentro desta estrutura do diretório. Por exemplo, para desenvolvedores que desejam organizar seus conteúdos, devem ser criadas pastas dentro da pasta inicial para organizar os arquivos.
3. As letras maiúsculas na pasta **META-INF** e **WEB-INF** são intencionais e obrigatórias assim como as letras minúsculas nas pastas **classes** e **lib**. Não seguir o formato correto em qualquer destas pastas resultará que sua aplicação não será capaz de localizar seus conteúdos.
4. Todo conteúdo da pasta **WEB-INF** não será visto a partir do navegador (por exemplo, acessando seu endereço). O contêiner automaticamente gerencia isto, ou seja, para o navegador, esta pasta não existe. Este mecanismo protege suas fontes confidenciais, como classe de arquivos Java, as configurações de aplicações, entre outros. O conteúdo desta pasta, somente pode ser acessado pela aplicação.
5. Deve existir um arquivo chamado *web.xml* dentro da pasta *WEB-INF*. Mesmo que, por exemplo, sua aplicação WEB tenha apenas conteúdo estático e não faça uso de classes Java ou arquivos externos, o contêiner ainda irá requerer que sua aplicação tenha estes dois itens.

8. Exercício

Responda às seguintes questões:

- Que tipo de arquitetura as aplicações WEB usam? Quem são os participantes de tais estruturas e o quais são suas funções?
- Qual o tipo de linguagem utilizada para indicar ao navegador como apresentar conteúdo para o usuário?
- HTTP é um protocolo de conexão stateful ou stateless?
- Quais são os dois métodos *HTTP Request* mais utilizados? E como eles são diferentes? Quando é melhor usar um ao invés do outro?
- Qual componente é absolutamente necessário para ser possível executar aplicações WEB?
- Quais são os elementos não opcionais da estrutura de diretórios da aplicação WEB?
- Qual é o nome do arquivo XML utilizado para configurar aplicações WEB? Em qual diretório pode ser encontrado?
- Qual pasta contém os arquivos JAR das bibliotecas usadas pelas aplicações?
- Qual pasta irá conter os arquivos de classe Java usados pelas aplicações?

Parceiros que tornaram JEDI™ possível

Instituto CTS
Patrocinador do DFJUG.

Sun Microsystems
Fornecimento de servidor de dados para o armazenamento dos vídeo-aulas.

Java Research and Development Center da Universidade das Filipinas
Criador da Iniciativa JEDI™.

DFJUG
Detentor dos direitos do JEDI™ nos países de língua portuguesa.

Banco do Brasil
Disponibilização de seus *telecentros* para abrigar e difundir a Iniciativa JEDI™.

Politec
Suporte e apoio financeiro e logístico a todo o processo.

Borland
Apoio internacional para que possamos alcançar os outros países de língua portuguesa.

Instituto Gaudium/CNBB
Fornecimento da sua infra-estrutura de hardware de seus servidores para que os milhares de alunos possam acessar o material do curso simultaneamente.