



APLICAÇÕES JAVA NA INTERNET

Rodrigo Rodrigues Gonçalves

Uberlândia, dezembro/2000.

APLICAÇÕES JAVA NA INTERNET

Rodrigo Rodrigues Gonçalves

Monografia apresentada ao Curso de Ciência da Computação do Centro Universitário do Triângulo - Unit, como requisito básico à obtenção do grau de Bacharel em Ciência da Computação, sob a orientação do Prof. Henrique Barbosa Leite.

Uberlândia, dezembro/2000.

APLICAÇÕES JAVA NA INTERNET

Rodrigo Rodrigues Gonçalves

Monografia apresentada ao Curso de Ciência da Computação do Centro Universitário do Triângulo - Unit, como requisito básico à obtenção do grau de Bacharel em Ciência da Computação.

Henrique Barbosa Leite, Msc.

(Orientador)

Marcos Ferreira de Rezende, Msc.

(Coordenador do Curso)

Alex Dias, Msc.

(Avaliador)

Marcos Ferreira de Rezende, Msc.

(Avaliador)

Uberlândia, dezembro/2000.

DEDICATÓRIA

Para aquelas pessoas que desejam utilizar os recursos desta magnífica linguagem computacional na Internet.




























AGRADECIMENTOS









Agradeço aos professores Henrique Barbosa, Silvia Fernanda, Marcos Rezende e Cleudair Nery, que muito me ajudaram ao longo da minha vida acadêmica e profissional, contribuindo também para o sucesso deste trabalho, transmitindo seus conhecimentos e suas experiências de vida.

RESUMO

A plataforma *Java* é essencialmente um novo caminho da computação, este projeto apresentará os conceitos sobre a plataforma *Java* na utilização da Internet, abordando as funcionalidades dos *Applets* que são utilizados nos clientes (*browsers*), *Servlets* e *Java Server Pages* (JSP) que oferecem grandes recursos aos servidores *Web*, e o JDBC (*Java Database Connectivity*) para a integração dos documentos dinâmicos com os Bancos de Dados . O projeto destaca ainda que a plataforma *Java* não é mais mito e sim realidade na rede mundial de computadores, oferece argumentos verídicos das vantagens da utilização na Internet, ensina como pode ser aplicada, mostrando algumas soluções que podem ser introduzidas juntamente com as redes de comunicações para a integração com os sistemas computacionais existentes, que aumenta negócios com clientes (*business-to-customers*), negócios com empresas (*business-to-business*), parceiros, e fornecedores na Internet ou em outra rede de comunicação.

SUMÁRIO

 INTRODUÇÃO.....	
2. <i>APPLETS</i>	
2.1. O que é um <i>Applet</i> ?.....	
2.2. Applets e Aplicações <i>standalone</i>	
2.2.1. As Aplicações.....	
2.2.2. Os Applets.....	
2.3. Adicionando um Applet a um documento HTML.....	
2.4. Atributos Opcionais para Applets.....	
2.5. <i>Browsers</i> que não suportam Applets.....	
2.6. Um Applet simplificado.....	
2.7. Conclusão.....	
3. <i>SERVLETS</i>	
3.1. O que é um <i>Servlet</i> ?.....	
3.2. Vantagens do Servlet sobre o “tradicional” CGI.....	
3.3. Servlets, o Java no ambiente do servidor.....	
3.4. Servlets e a <i>Web</i> – o novo CGI.....	
3.5. Um <i>tour</i> pelo JSDK (<i>Java Servlet Development Kit</i>).....	
3.6. Um Exemplo.....	
3.7. Conclusão.....	
4. <i>JAVA SERVER PAGES</i> (JSP).....	
4.1. O que é <i>Java Server Pages</i> ?.....	
4.2. As vantagens do Java Server Pages sobre o <i>Active Server Pages</i>	
4.3. Comparação entre Java Server Pages e <i>Active Server Pages</i>	
4.4. Um Exemplo.....	
4.5. Conclusão.....	
5. <i>JAVA DATABASE CONNECTIVITY</i> (JDBC).....	

5.1. O que é JDBC (<i>Java Database Connectivity</i>) ?.....	
5.2. Como funciona o JDBC.....	
5.2.1. JDBC-ODBC.....	
5.3. Exemplos.....	
5.4. Conclusão.....	
6. CONCLUSÃO.....	
REFERÊNCIA BIBLIOGRÁFICA.....	
BIBLIOGRAFIA.....	



INTRODUÇÃO

A princípio, o WWW (*World Wide Web*) funciona como um ambiente cliente-servidor típico. Nesse conceito, os papéis representados pelos sistemas estão bem divididos. O cliente geralmente é um microcomputador, com uma configuração de *hardware* modesta e com um aplicativo instalado que permite que essa máquina se conecte ao servidor. Este, em contrapartida, conta com uma capacidade de *hardware* mais significativa e está preparado para atender a diversos pedidos, vindos de diferentes clientes. Seguindo esse conceito, o cliente faz pedidos, o servidor realiza todo o processamento, e envia a resposta.

O ambiente WWW difere um pouco. Na grande maioria das vezes, o cliente divide o trabalho de processamento com o servidor, que envia uma série de instruções que são seguidas pelo cliente. Essas instruções são a respeito de como montar a página, de como criar os formulários, de onde as imagens devem ficar, do acionamento dos *plugin's* (componentes), etc. Mas não foi sempre assim.

No início existia apenas o HTML (*Hypertext Markup Language*), em suas primeiras versões. Era uma linguagem que basicamente permitia ao "programador" criar *links* (caminhos) para outras páginas.

Com o tempo, essa linguagem foi evoluindo, e gradualmente teve melhorias. No início, era a possibilidade de incluir imagens, o advento das tabelas, mas a parte interessante foi o surgimento dos formulários. Os formulários tinham a capacidade de enviar informações digitadas pelos usuários para serem processadas no servidor. Nascia o CGI.

CGI é uma sigla que significa *Common Gateway Interface*. *Gateway* é uma palavra que significa (à grosso modo) um elemento que liga dois ambientes diferentes. Nesse caso, o servidor WWW e o sistema operacional da máquina.

Mas o CGI apresenta uma série de limitações. Como um servidor WWW foi construído para atender dezenas de clientes simultaneamente, o CGI tem que se desdobrar para atender vários pedidos simultâneos, coisa que invariavelmente acontece. A programação deve ser feita de maneira mais robusta, sempre pensando no atendimento à vários usuários.

Depois de 2 ou 3 anos, com os conceitos de Intranet e redes privadas aparecendo, surgiu a necessidade de ambientes que colocassem disponíveis na *Web* os dados da empresa, ou que permitissem a um cliente fazer um pedido de compra *online* (imediatos), ou que um gerente pudesse ter acesso a dados de venda de onde quer que ele estivesse. E tudo isso mantendo a segurança, performance, modularidade, escalabilidade e permitisse a manutenção do sistema a baixo custo.

A linguagem Java surgiu de uma proposta de uma empresa chamada *Sun Microsystems* de criar uma linguagem que pudesse ser utilizada em eletrodomésticos, possibilitando a comunicação entre eles. A linguagem era baseada em uma versão mais simplificada do C++ (portanto, orientada a objetos) e, como esta, previa a troca de mensagens entre entidades. Alguns observaram que se imaginássemos o cliente como um objeto e o servidor como outro, seria possível usar o Java para comunicação na *Web*. E foi o que aconteceu.

A primeira visão que se teve da linguagem na *Web* foi com o que chamamos de *Applets* (que são programas desenhados para execução no cliente). Pela primeira vez, poderia ser executado programas

complexos no cliente, e esses programas tinham a capacidade de se comunicarem com um servidor para recuperar e enviar informações.

Os *Applets* não são 100% perfeitos, possuem alguns problemas, a maioria dos *Applets* são muito grandes para serem usados por pessoas com acesso lento à rede, mas atualmente este problema está sendo resolvido, com as novas tecnologias de comunicação o acesso à Internet está melhorando cada vez mais. Um outro problema é que existem diferenças entre os *browsers* (navegadores) dos principais *players* (fabricantes) do mercado, e um *Applet*, dependendo da funcionalidade implementada, não pode ser executado em um desses *browsers*. Deste modo, o desenvolvimento de um sistema usando *Applets* requer um estudo aprofundado desses pontos, e muitas vezes pode ser inviabilizado.

Se o grande problema na adoção do Java é o cliente, a solução mais prática é retirar o código do cliente. Essa idéia gerou um novo conceito, a de um programa em Java que pudesse interagir com o cliente como se fosse um CGI (ou seja, via formulários). Desta forma, todos os clientes poderiam usar um sistema baseado nessa tecnologia, (sacrificando apenas um pouco a interface). Essa tecnologia foi chamada de *Servlet*.

O *Servlet* é mais interessante que o CGI, em diversos aspectos. Executam mais rápido, seus *byte-codes* (programas em formato .class) podem ser transferidos para outro servidor WWW sem necessidade de alterações no código ou recompilações, tem uma grande facilidade de ligação com Banco de Dados (através de ODBC ou JDBC). Além disso, permite a programação de maneira multitarefa, o aproveitamento do código e todas as vantagens que uma linguagem orientada a objetos possui.

O ambiente utilizado para a execução de *Servlets* Java é um ambiente especial. Necessitando de um servidor *Web* e um *servlet runner*, que é um produto (normalmente feito em Java) responsável pela execução dos *Servlets*.

A implementação dos *Servlets* por parte dos “programadores” foi sendo dirigida cada vez mais para a facilidade na escrita dos códigos Java, a partir deste ponto de vista o JSP (*Java Server Pages*) foi criado. Facilitando a implementação de códigos tão poderosos quanto os *Servlets*, mas com recursos que provêm uma interação direta com os códigos HTML na manipulação das páginas dinâmicas.

O suporte às operações a um banco de dados são oferecidas pelo Java, isto devido ao JDBC (*Java Database Connectivity*). O JDBC pode ser utilizado juntamente com os *Servlets* ou JSP oferecendo uma interatividade muito maior do que simplesmente a construção de páginas dinâmicas. Esta integração com os sistemas computacionais existentes, é convertida em negócios com clientes (*business-to-customers*), negócios com empresas (*business-to-business*), parceiros, e fornecedores na Internet ou em outra rede de comunicação.

Este trabalho está organizado em três partes principais, a utilização do Java nos clientes (*Applets*), nos servidores (*Servlets* e JSP) e a integração das páginas dinâmicas com os Bancos de Dados (JDBC).

2. *APPLETS*

Programas Java podem ser classificados tanto como aplicações Java ou *Applets* Java, baseado no conteúdo dos módulos de execução. Enquanto *Applets* requerem a presença de um *appletviewer* (utilitário para testes de *Applets*) ou de um *browser* (navegador) como *Netscape Communicator* ou *Microsoft Internet Explorer*, as aplicações Java são projetadas para executar em um computador através do interpretador Java e inicializados através de uma janela de comando (*prompt*), sem a necessidade de uma ajuda especial de outras aplicações.

Applets são pequenos programas que podem ser adicionados as páginas *Web*. Entretanto, para começar a trabalhar com *Applets* é necessário conhecer um pouco o seu funcionamento interno, a diferença de um *Applet* e uma aplicação *standalone* (independente), e assim como adicionar um *Applet* a um documento HTML.

Serão descritos os seguintes assuntos:

- O que é um *Applet* e como ele funciona.
- As diferenças entre *Applets* e aplicações *standalone*.
- Trabalhando juntamente com dois tipos diferentes de programas Java, um tipo como parte de uma página *Web* (*Applets*) e um outro tipo, como qualquer outra aplicação, aplicações Java *standalone*.
- Adicionando um *Applet* a um documento HTML.
- Como escrever um simples *Applet*.

2.1. O que é um *Applet*?

Um *Applet* é simplesmente uma parte de uma página *Web*, como uma imagem ou uma linha de texto. Da mesma forma que um *browser* “cuida” de mostrar uma referência de uma imagem em um documento HTML, o *browser* localiza e executa o *Applet* Java. Quando o *browser* carrega o documento HTML, o *Applet* Java é também carregado e executado. Não importa se o *Applet* esta disponível no disco rígido ou não, porque caso seja necessário, o *browser* automaticamente efetua o *download* (cópia de códigos do servidor para o cliente) do *Applet* antes de executá-lo.

Existe uma relação cliente-servidor (*client-server*) entre um *browser* que mostra um *Applet* e um sistema que fornece o *Applet*. O cliente é o computador que requer o serviço de um outro sistema, o servidor é o computador que provê estes serviços. No caso do *Applet*, o cliente é o computador que esta tentando exibir um documento HTML que contém uma referência a um *Applet*. O servidor é o computador que efetua *upload* (fornece) do *Applet* para o cliente, através desta relação que é permitido ao cliente usar o *Applet*.

Muitos programas nos quais são efetuados o *download* da Internet podem conter vírus que são descobertos somente depois do usuário ter executado tais *softwares*, estes podem contaminar todo o sistema operacional e destruir as informações, mas este problema é apenas uma das pequenas conseqüências que estes programas podem trazer, entretanto os *Applets* estão em toda a Internet e representam uma forma segura para transferir programas pela Internet. Isto se deve ao fato que o interpretador Java não permite que o *Applet* seja executado até que ele tenha confirmado que o *byte-code* do *Applet* não foi corrompido ou mudado de alguma forma (Figura 2.1.). Além disso o interpretador Java determina se a representação do *byte-code* do *Applet* está dentro das regras do Java, estas regras são estabelecidas pela *sandbox* (caixa de areia).

Por exemplo, um Applet nunca pode ter acesso a uma porção de memória do sistema operacional quando não lhe pertencer tal memória. Não somente seguros são os Applets, como na prática é garantido que, nunca um Applet poderá travar (*crash*) o sistema operacional.



Figura 2.1. – “Caminho” que o *byte-code* a ser executado percorre.

2.2. Applets e Aplicações *standalone*

Ainda que a maioria dos programadores Java estejam entusiasmados com criação de Applets, o Java pode também ser utilizado para criar aplicações *standalone*, ou seja, aplicações que não precisam de ser embutidas num documento HTML. A aplicação mais famosa é o *HotJava Web Browser*. Este *browser* é completamente escrito na linguagem Java.

Enquanto que um Applet pode ser transmitido automaticamente através da Internet, uma aplicação *standalone* Java reside no disco rígido local de um computador. De fato, uma aplicação *standalone* Java é exatamente como uma outra aplicação qualquer instalada no sistema operacional. A única diferença é que foi escrita em Java e a outra aplicação em uma linguagem computacional como C++ por exemplo.

Para executar uma aplicação Java, é necessário a utilização do interpretador Java, no qual efetua a leitura do *byte-code* contido no arquivo e posteriormente o executa. Por causa do interpretador Java ser executado a partir da linha de comando, é necessário inicializar a aplicação Java através desta mesma linha de comando, mesmo que a aplicação seja no modelo de janelas (*windows*), mesmo que de qualquer forma seja inicializada a partir do *prompt* de comando, a aplicação aparece em uma janela como qualquer outra aplicação em modelo de janela que estiver instalada no sistema operacional.

Uma outra grande diferença entre Applets e aplicações *standalone* é o modo como tratam o assunto de segurança. A habilidade de uma Applet acessar um arquivo, como exemplo, é controlada por variáveis do ambiente que o usuário configura com o seu *browser*. Um Applet não pode acessar nenhum arquivo ou diretório que ele não possua permissão. Por isso, a maioria dos Applets não tratam com arquivos em caso nenhum, porque nunca se tem certeza exatamente qual o tipo de acesso ao disco rígido estarão permitidos.

As aplicações *standalone*, por outro lado, não possuem esta segurança rígida que é aplicada aos Applets. Podem ter acesso normalmente a arquivos, como qualquer outra aplicação. Esta segurança rígida não é necessária as aplicações *standalone* porque as aplicações não são transmitidas para o computador através da Internet sem o conhecimento do usuário. Ao contrário dos Applets, no qual são transmitidos para o sistema automaticamente como parte de uma página *Web*. A segurança aplicada aos Applets previne que o servidor (de onde foi enviado o Applet) possa obter acesso as informações em seu sistema.

2.2.1. As Aplicações

Aplicações Java são projetadas para rodar na máquina cliente sem a necessidade de um *browser*. As aplicações possuem uma função de acesso público, *main()*, no qual representa o ponto de partida da execução. O interpretador tem a responsabilidade de chamar apenas uma vez a função *main()*, isto ocorre no exato momento que a execução do programa começa, o resto fica a cargo do programador.

Aplicações que utilizam serviços de rede (*network services*), como *sockets*, são responsáveis pela implementação, suporte a segurança e integridade no programa no nível de aplicação. No nível do usuário a segurança mostra ser adequada para aplicações *standalone*, desde que durante a execução de tal programa, o ambiente possa somente exercer privilégios do usuário que iniciou a execução.

2.2.2. Os Applets

Diferente das aplicações, a execução dos Applets requer um *browser* com suporte a Java ou um *appletviewer*. O “coração” de um Applet é composto por quatro métodos, descritos abaixo. Estes métodos são automaticamente chamados pelos *browsers*.

- *Applet.init()*, quando um Applet é carregado pela primeira vez para o sistema, o método *init()* é chamado pelo ambiente de execução (*browser* ou *appletviewer*) para inicializar o Applet. Esta função é chamada apenas uma vez, antes que o Applet se torne visível, serve como um método de configuração, para inicializar variáveis e outros objetos.
- *Applet.start()*, este método é chamado pelo *browser* toda hora que o Applet torna-se ativo. O Applet pode ser ativado logo após a inicialização, ou quando uma página HTML que contém a *tag applet* é exibida novamente. Os Applets que possuem códigos para o processamento em *multi-threaded* (vários processos), utilizam este método para criar ou reiniciar a execução de tais *threads* (processos).
- *Applet.stop()*, quando o usuário sai da página que continha o Applet, o *browser* efetua a chamada a este método. *Threads* e conexões a sistemas através da rede são geralmente suspensas quando este método é acionado.
- *Applet.destroy()*, Este método é chamado pelo *browser* a todo momento que um *applet* necessita ser descartado. Uma chamada ao método *destroy()* é seguida pelo despejo da instância do Applet da memória, portanto este método é utilizado principalmente para o propósito de “limpeza” da memória.

Estes métodos são chamados pelo *browser* em uma ordem específica. É impossível fazer uma chamada ao método *start()*, se o Applet não foi inicializado. O diagrama representado na Figura 2.2. ilustra estes passos na execução de um Applet.

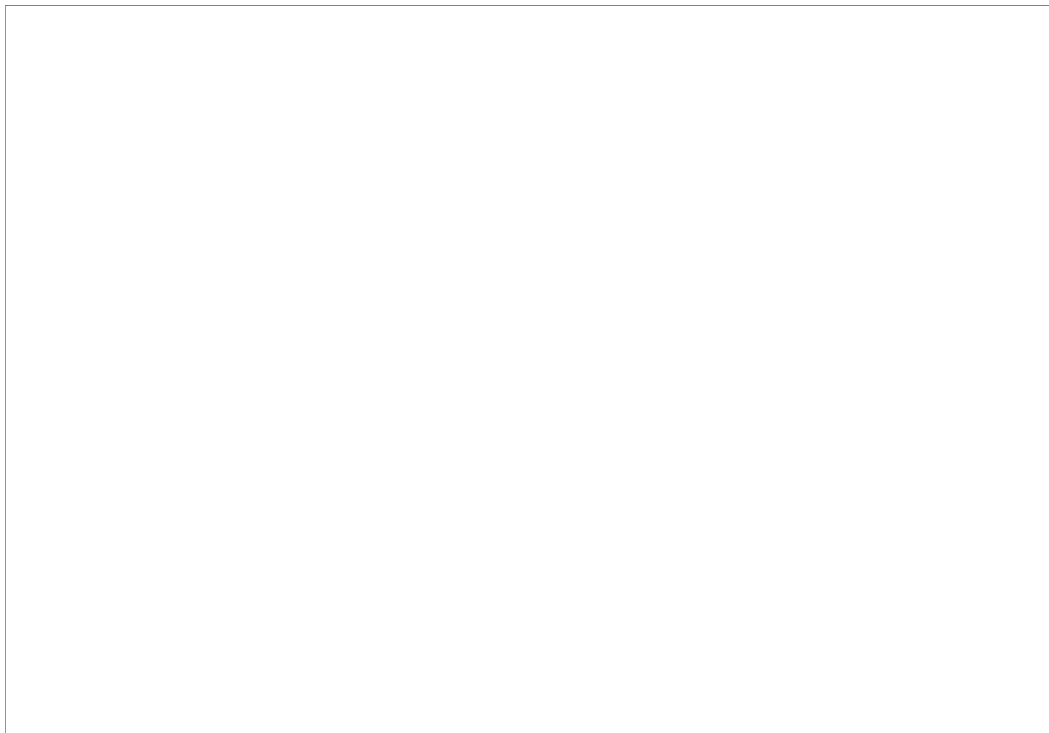


Figura 2.2. –
Execução de um
Applet.

Enquanto *init()* e *destroy()* são chamados apenas uma vez, no início e no fim da execução respectivamente, o Applet deve estar preparado para responder aos métodos *start()* e *stop()* a toda hora. Para melhorar a performance do sistema, quando *threads* e conexões a sistemas pela rede são utilizados, estes recursos devem ser suspensos e/ou liberados através da chamada do método *stop()*.

2.3. Adicionando um Applet a um documento HTML

Quando a *Sun Microsystems* desenvolveu o Java, foi sugerido uma extensão para os documentos HTML que habilitava o funcionamento dos Applets nas páginas *Web*, a extensão é a *tag* (expressão) `<applet>`, que é definida abaixo.

```
<applet attributes>

parameters

alternate-content

</applet>
```

A *tag* `<applet>` é similar a qualquer outra *tag* HTML. Por exemplo, a *tag* é iniciada com `<applet attributes>` e finalizada com `</applet>`, na qual não se difere do formato de um *tag* HTML. A primeira linha (`<applet attributes>`) e a última são obrigatórias (`</applet>`). As outras linhas são opcionais.

A seção de atributos da *tag* `<applet>` contém informações importantes sobre todo o conjunto da *tag*, incluindo o arquivo associado *.class* e a largura e altura do Applet a ser exibido no documento HTML. A última linha mostra ao *browser* que a *tag* chegou ao fim, como no exemplo abaixo.

```
<applet code=MeuApplet.class width=120 height=120>

</applet>
```

No exemplo acima, o atributo *code* é o nome do arquivo *.class* que é o Applet propriamente dito. Como descrito anteriormente o arquivo *.class* contém o *byte-code* que representa o Applet, no qual pode

ser executado pelo interpretador Java. Os atributos *width* e *height* controlam o tamanho do Applet no documento HTML. Os atributos *code*, *width* e *height* são obrigatórios como a última linha que é a tag `</applet>`.

2.4. Atributos Opcionais para Applets

Existem vários atributos opcionais que podem ser utilizados com a tag `<applet>`. O primeiro é *codebase*, que especifica o diretório base do Applet ou URL (*Uniform Resource Locator*). Este diretório ou URL é utilizado na combinação com o nome do arquivo especificado no atributo *code* para encontrar o código Applet. No caso de um diretório, o atributo *codebase* é relativo a localização do documento HTML que contém a tag `<applet>`. Quando não informado o atributo *codebase* o *browser* procura pelo arquivo Applet no mesmo diretório ou URL que se encontra o documento HTML. A tag `<applet>` no exemplo abaixo mostra a utilização desse atributo.

```
<applet codebase=MeuDiretorio code=MeuApplet.class width=120
height=120>

</applet>
```

O exemplo acima mostra ao *browser* que o arquivo “MeuApplet.class” esta localizado no diretório chamado “MeuDiretorio”. Este diretório está no mesmo nível da estrutura de diretórios no qual se encontra o documento HTML que contém este Applet. Caso o arquivo HTML esteja localizado no diretório “java\demo”, então o caminho para o arquivo *.class* deverá estar em “java\demo\meudiretorio\meuapplet.class”. Podendo ser utilizando a URL, como <http://www.servidor.com.br/paginas/meudiretorio>, para o atributo *codebase*. Isto faz com que o Applet seja carregado de um site específico.

Outros atributos opcionais que podem ser utilizados na tag `<applet>` são *alt*, *name*, *align*, *hspace*, e *vspace*. O atributo *alt* habilita um texto específico que é mostrado como uma pequena informação sobre o Applet, o atributo *name* fornece ao Applet um nome simbólico que é utilizado como referência ao Applet (utilizado para comunicação entre Applets).

Os atributos *align*, *hspace* e *vspace* trabalham juntos para posicionar o Applet em um documento HTML. Estes atributos trabalham exatamente como na tag `` que é utilizada para mostrar imagens nas páginas *Web*. O atributo *align* pode ser um dos valores: *left*, *right*, *middle*, *absmiddle*, *bottom*, *absbottom*, *baseline*, *top* ou *texttop*. Os atributos *hspace* e *vspace* controlam o tamanho dos espaços em branco em torno do Applet, ou seja, a borda do Applet, isto quando o atributo *align* é configurado para *left* ou *right*. No exemplo abaixo mostramos um documento HTML simples utilizando estes atributos opcionais.

```
<title>Exemplo</title>
<hr>Este é um texto simples de um documento HTML

<applet codebase=MeuDiretorio code=MeuApplet.class width=120
height=120 alt="Este é meu applet no documento HTML."
name=MeuApplet align=middle>

</applet>

que demonstra o uso de atributos opcionais</hr>
```

2.5. Browsers que não suportam Applets

É um comportamento padrão dos *browsers* ignorar as *tags* que não são reconhecidas, assim ocorre com a *tag* `<applet>`, caso o *browser* não suporte Applets. Entretanto, pode-se prover uma resposta mais amigável ao usuário que esteja tentando visualizar o Applet com estes tipos de *browser*. Isto é facilmente resolvido adicionando um conteúdo antes da *tag* `</applet>`. No exemplo abaixo, é mostrado um código HTML simples para executar o Applet “MeuApplet.class” com o conteúdo adicional para aqueles *browsers* que não suportam Java.

```
<applet code=MeuApplet.class width=120 height=120>
```

```
Se o browser suportar Java, o applet estará sendo executado agora!
```

```
</applet>
```

O conteúdo informado é compreendido por qualquer comando padrão HTML e é ignorado para os *browsers* que suportam Java. Este conteúdo somente aparece para os *browsers* que não suportam Java.

2.6. Um Applet simplificado

A linguagem de programação Java e bibliotecas permite a criação de Applets que podem ser simples ou complexos de acordo com a necessidade. De fato, pode-se escrever um Applet simples em apenas algumas linhas de código, como é mostrado.

```
import java.applet.*;

public class MeuApplet extends Applet
{
}
```

A primeira linha do exemplo mostra ao compilador Java que este Applet estará utilizando alguma ou todas as classes definidas no pacote *java.applet*. Todas as capacidades básicas de um Applet são fornecidas por estas classes, o qual facilita a criação de um Applet em poucas linhas de código.

A segunda linha de código declara uma nova classe chamada “MeuApplet”. Esta nova classe é declarada como pública para que a classe possa ser acessada quando o Applet é executado em um *browser* ou em uma aplicação como o *appletviewer*. Caso a classe Applet não seja declarada como pública, o código é compilado com sucesso, mas o Applet recusa-se a ser executado. Em outras palavras, todas as classes Applets devem ser públicas.

Nota-se que existem vantagens da programação orientada a objetos (POO) utilizando a herança na declaração de um Applet através da super-classe Applet. Esta herança funciona exatamente do mesmo modo que qualquer outra hierarquia criada na POO.

A classe Applet está incluída no JDK (*Java Developer's Kit*). No exemplo exibido acima representa um Applet simplificado que quando compilado criará o arquivo “MeuApplet.class”, no qual representa o *byte-code* a ser executado no sistema. Para executar este Applet, apenas faz-se necessário a criação de um documento HTML como foi descrito anteriormente.

2.7. Conclusão

Criar um Applet pode ser relativamente um processo muito fácil. As classes que acompanham os pacotes do JDK oferecem estas facilidades para a construção, desde capturar eventos do mouse até conexões com servidores *Web*. Existe uma relação cliente-servidor (*client-server*) entre um *browser* que mostra um Applet e um sistema que fornece o Applet, pois os Applets são programas desenhados para

execução no cliente, com isso pela primeira vez pode-se executar programas complexos no cliente, e esses programas possuem a capacidade de se comunicarem com um servidor para recuperar e enviar informações com um certo nível de segurança.

Destaca-se nesta facilidade da criação de um Applet os seus quatro métodos (*init*, *start*, *stop*, *destroy*), que provem os recursos necessários para o gerenciamento da aplicação Applet.

Os Applets não são 100% perfeitos, possuem alguns problemas, a maioria dos Applets são muito grandes para serem usados por pessoas com acesso lento à rede. Um outro problema é que existem algumas diferenças entre os *browsers* (navegadores) dos principais *players* (fabricantes) do mercado, e um Applet, dependendo da funcionalidade implementada, não pode ser executado em um desses *browsers*. Deste modo, a criação de um sistema usando Applets requer um estudo aprofundado desses pontos, e muitas vezes pode ser inviabilizada.

Se o grande problema na adoção do Java é o cliente, a solução mais prática é retirar o código do cliente. Essa idéia gerou um novo conceito, a de um programa em Java que pudesse interagir com o cliente como se fosse um CGI (ou seja, via formulários). Desta forma, todos os clientes poderiam usar um sistema baseado nessa tecnologia, (sacrificando apenas um pouco a interface). Essa tecnologia foi chamada de *Servlet*, que será abordada no capítulo seguinte.

3. *SERVLETS*

Uma das soluções Java para os servidores são os *Servlets*, que tem o intuito de substituir, com vantagens, as atuais aplicações CGI (*Common Gateway Interface*).

Aplicações CGI são aqueles programas que rodam nos servidores e tem a capacidade de receber dados vindos da Internet, processar estes dados e enviar uma resposta ao cliente. Na maioria das vezes que um usuário preenche um formulário *online* pela Internet estes dados são processados por um programa localizado no servidor e as respostas enviadas de volta ao usuário.

As páginas dinâmicas também são processadas no servidor por aplicações CGI, ASP (*Active Server Pages*), *ColdFusion* ou programas similares.

Toda vez que o servidor *Web* recebe uma requisição CGI ele carrega um novo programa, executa-o, envia a resposta para o cliente e é finalizado.

No início da Internet, um site que recebia 100 visitas por dia, era um site bastante visitado, mas hoje alguns sites recebem mais de 50 mil acessos por dia e todo este processo de carregar, executar e finalizar um CGI resulta num significativo *overhead* (inicialização, execução e finalização de processos) do servidor. Os *Servlets* são persistentes, independentes de plataforma e com eles pode-se usar todos os recursos disponíveis para as aplicações Java incluindo RMI (objetos distribuídos), JDBC e interações com Applets Java.

A definição exata e as vantagens da utilização de um *Servlets* serão abordadas a seguir, estas vantagens serão comparadas com as aplicações CGI.

3.1. O que é um *Servlet*?

Segundo Davidson[1], “um Servlet é um componente *Web*, que gera conteúdo dinâmico. Servlets são pequenas classes Java compiladas, independentes de plataforma para uma arquitetura indiferente onde seu *bytecode* pode ser carregado dinamicamente e executado por um servidor *Web*. Servlets interagem com os clientes *Web* (*browser*) no paradigma de *request* (pedido) e *response* (resposta). Este modelo *request-response* é baseado no comportamento do *Hypertext Transfer Protocol* (HTTP).”

Os Servlets são programas que rodam em um servidor *Web* e constróem páginas *Web*. Construir páginas dinâmicas em tempo de execução é útil (e freqüentemente utilizado) por várias razões:

- A página *Web* é baseada no envio de dados pelo usuário. Por exemplo a página de resultados dos *sites* de procura são geradas dinamicamente neste modelo, e programas que processam pedidos para o comércio eletrônico também.
- Mudanças freqüentes em dados. Por exemplo, páginas de relatórios sobre o tempo ou as notícias principais de um jornal podem ser construídas dinamicamente, talvez retornando a própria página anterior se ela ainda estiver com os dados atualizados.
- A utilização de páginas *Web* a partir de banco de dados corporativos e outras fontes semelhantes. Por exemplo, a construção de páginas dinâmicas para uma lista *online* de preços atualizados de uma loja e número de itens em estoque.

3.2. Vantagens do Servlet sobre o “tradicional” CGI

Java Servlets são mais eficientes em processamento, mais fáceis de utilizar, com mais recursos, mais portáteis, e custo baixo (tecnologia Java, servidores e aprendizado) que o CGI tradicional e possui outras alternativas similares a tecnologia CGI.

- Eficiência em processamento, com o CGI tradicional, um novo processo é inicializado para cada solicitação HTTP. Se o programa CGI possui uma operação relativamente rápida, o *overhead* para inicialização do processo pode dominar o tempo de execução do sistema. Com Servlets, a *Java Virtual Machine* fica no “ar”, e cada solicitação é gerenciada por uma *thread* simples, ágil e otimizada, não sobrecarregando a operação de processos do sistema. No CGI, se existem N solicitações (*request*) simultâneas para o mesmo programa CGI, o código do programa CGI é carregado para a memória N vezes. Com Servlets, de qualquer forma, as N *threads* são carregadas mas somente uma cópia do código da classe Servlet permanece na memória. Servlets também possuem mais alternativas de otimização do que o programa CGI como por exemplo o controle de *cache* (local de armazenamento que contém dados em que o computador precisará usar em curto tempo ou usa com mais frequência), conexões com Banco de Dados que permanecem abertas e outras ações semelhantes. Os Servlets possuem um método *init()*, que assim como nos Applets, ocorre apenas na primeira vez que a classe é carregada. É geralmente no método *init()* que, por exemplo, estabelecemos uma conexão ao Banco de Dados. Cada uma das *threads* geradas pode usar a mesma conexão aberta no método *init()*. Este tipo de tratamento aumenta em muito o tempo de performance do Servlet, já que podemos fazer a conexão ao Banco de Dados apenas uma vez e todos as outras requisições usam esta conexão.

- Conveniente e fácil, Servlets possuem uma grande infra-estrutura para análise automática de palavras (*parsing*) e decodificação de dados de formulários HTML, obtendo e configurando cabeçalhos HTTP, gerenciamento de *cookies*, controle de *sessions*, e muitas outras utilidades semelhantes. Programar em Java é muito mais fácil que programar em C ou PERL. Algumas IDEs (*Interface Developer Enviroment*) Java, como o *Visual J++*, *Visual Age* da IBM e *Borland JBuilder*, auxiliam na criação de Servlet Java.

- Mais Recursos, Servlet permitem facilmente efetuar diversas ações que são consideradas difíceis ou impossíveis com um CGI. Servlets podem compartilhar dados entre outros Servlet, tornando altamente proveitoso e fácil de implementar recursos como um concentrador de conexões a um Banco de Dados (*database connection pools*). Podem manter as informações de cada solicitação, simplificando ações como controle de *sessions* e gerenciamento de *caches*.

- Independência de plataforma ou portabilidade, Servlet são escritos em Java e seguem a padronização da API (*Application Program Interface*) Java. Consequentemente, Servlets são suportados diretamente ou através de extensões adicionais (*plugin*) a quase todos os principais servidores *Web*. Os Servlets podem rodar em qualquer plataforma sem serem rescritos ou até mesmo compilados novamente. Os *scripts* PERL, por serem interpretados, também possuem esta funcionalidade mas são muito lentos. As aplicações CGI que necessitam de alta performance são escritas na linguagem C/C++. Com Java você pode criar aplicações muito mais modulares, tirar todo o proveito da orientação a objetos e utilizar o grande número de APIs disponíveis pela Javasoft (divisão da Sun Microsystems que gerencia a linguagem Java) e terceiros.

- Custo baixo, existem um número de servidores *Web* gratuitos ou com preços acessíveis disponíveis que são ótimos para o uso pessoal ou para volumes pequenos dos *sites Web* que suportam Servlets, destacando o servidor de *Web* Apache, que é gratuito e com vários recursos no gerenciamento de grandes volumes dos *sites Web*. No entanto, uma vez com um servidor *Web*, não importando o custo do servidor, adicionar um suporte a Servlet nele (caso o servidor não esteja pré-configurado para suporte a Servlets) é geralmente gratuito ou possui um custo baixo.

3.3. Servlets, o Java no ambiente do servidor

Servlets são executados em um servidor como Applets são executados em um cliente, com os mesmos tipos de checagem de erros e o mesmo *garbage collection* (gerenciamento de memória).

Servlets são mais capacitados e robustos que Applets, porque rodam em um ambiente muito mais controlado. Os administradores que utilizam os Servlets podem oferecer as permissões necessárias para que estes sejam executados da melhor forma possível e cumprindo assim, o trabalho que ele tem como objetivo.

3.4. Servlets e a *Web* – o novo CGI

O Servlet pode ser considerado a melhor ferramenta que surgiu para ser utilizado juntamente com desenvolvimento em servidores *Web* desde a utilização do Perl e várias bibliotecas CGI. Oferece uma substituição para as aplicações CGI e muito mais. Desenvolvedores que já utilizaram Applets encontrarão nos Servlets uma familiaridade extrema, requerendo apenas uma mudança no foco e nas bibliotecas das classes.

Os Servlets podem utilizar várias ferramentas que auxiliam o desenvolvimento para a *Web*, temos entre estas ferramentas o *Java Database Connectivity* (JDBC, que será abordado no capítulo 5) que é considerada a ferramenta chave para as conexões com Banco de Dados, o *Remote Method Invocation* (RMI) uma outra ferramenta utilizada nas aplicações distribuídas, e várias outras contidas na biblioteca Java.

3.5. Um *tour* pelo JSDK (*Java Servlet Development Kit*)

Para criar um Servlet é necessário o *Java Servlet Development Kit* (JSDK) da *JavaSoft*. O JSDK consiste em vários *packages* (pacotes), incluindo os pacotes *javax.servlet* e *javax.servlet.http* que contém as classes e interfaces necessárias para criar os Servlets. Existe também o pacote *sun.servlet*, no qual consiste em classes que são utilizadas pelo *Web Server* na execução dos Servlets.

Uma vez com o JSDK, utiliza-se na criação de um Servlet a classe *GenericServlet* que define três métodos principais: *init()*, *service()* e *destroy()*. O método *init()* oferece ao Servlet a oportunidade de executar qualquer inicialização requerida, como conexões com Bancos de Dados. O método *destroy()* oferece a oportunidade de finalização de recursos, como o término de conexões com Bancos de Dados. O método *service()* é processado cada vez que o Servlet é requisitado. O Servlet utiliza um objeto para controlar a requisição (*ServletRequest*) e um de resposta (*ServletResponse*) para processar uma solicitação do cliente.

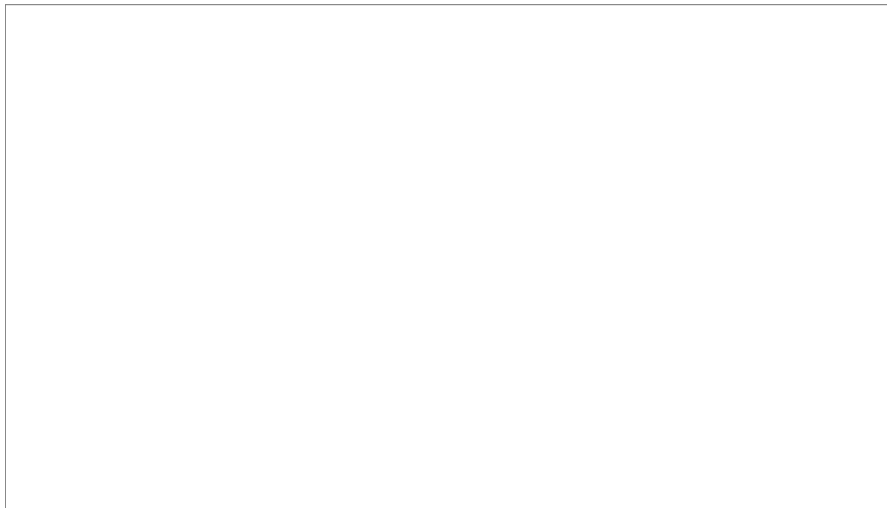


Figura 3.1. –
Execução de um Servlet.

Para ajudar no tratamento de transações baseadas na *Web*, o JSDK possui a classe *HttpServlet* no pacote *javax.servlet.http.HttpServlet* que é derivada da classe *GenericServlet*. *HttpServlet*, provê uma implementação mais fácil e inteligente do método *service()* para automaticamente gerenciar as requisições (*request*) *GET* e *POST*. *HttpServlet* delega apropriadamente para os métodos *doGet()* e *doPost()*. Um típico Servlet CGI precisaria apenas ter implementações nos métodos *doGet()* e *doPost()*.

Os métodos *doGet()* e *doPost()* recebem os parâmetros *HttpServletRequest* e *HttpServletResponse*. *HttpServletRequest* é uma interface que provê métodos para capturar informações de uma solicitação (*request*) de um cliente. Com o método *getParameter()* pode-se obter um par de valores (chave/valor) da requisição de um cliente. *HttpServletResponse* provê meios de “saída” no qual os Servlets podem escrever informações HTML de volta para os clientes.

3.6. Um Exemplo

Um programa cliente, como um *browser*, acessa um servidor *Web* e faz uma requisição HTTP. Esta solicitação é processada pelo servidor *Web*. O servidor *Web* determina qual Servlet será invocado, baseado na sua configuração interna e o chama na representação dos objetos de requisição (*request*) e resposta (*response*). O Servlet pode rodar em diferentes processos no mesmo servidor, ou em um servidor diferente do servidor *Web* que sofreu a solicitação.

O Servlet utiliza o objeto de requisição para encontrar qual é o usuário remoto, quais parâmetros do *form* (formulário) HTML podem ter sido enviados como parte desta requisição, e outros dados relacionados. O Servlet então pode executar qualquer lógica que foi programada e poderá gerar dados para enviar novamente ao cliente da requisição inicial, enviando estes dados de volta para o cliente através do objeto de resposta.

Uma vez que é processado a solicitação, o Servlet assegura-se que a resposta foi devidamente entregue e retorna o controle novamente para o servidor *Web*.

Um código básico de um Servlet é apresentado no, exemplo abaixo, mostrando como pode ser utilizado o Servlet no lugar de um programa CGI. O Servlet simplesmente lista o parâmetro e seu valor informado através de um *form* HTML. A classe do exemplo abaixo, *BasicCGIServlet* controlam os métodos *POST* e *GET* da mesma forma, então *doPost()* e *doGet()* simplesmente invocam *doService()*.

```
import javax.servlet.*;
```

```
import javax.servlet.http.*;
```

```

import java.io.*;

import java.util.*;

public class BasicCGIServlet extends HttpServlet

{

    public BasicCGIServlet ()

    {

        super();

    }


    public void doPost(HttpServletRequest req, HttpServletResponse
resp) throws ServletException, IOException

    {

        doService(req, resp);

    }


    public void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException

    {

        doService(req, resp);

    }


    protected void doService(HttpServletRequest req,
HttpServletResponse resp) throws ServletException, IOException

    {

        // captura o objeto de saída para escrita
        ServletOutputStream out = resp.getOutputStream();

        // configura o tipo do documento a ser respondido
        resp.setContentType ("text/html");

        out.println("<html><head><title>Parâmetros de um Form </title>
</head> <body> bgcolor = \"white\">");

        out.println ("<table border=1>");

        out.println ("<th>Key</th><th>Valor</th>");

        for(Enumeration e=req.getParameterNames();
e.hasMoreElements(); )

        {

            String key = (String) e.nextElement();

```

```

        out.println("<tr><td>" + key + "</td><td>" +
            req.getParameter(key) + "</td></tr>");
    }

    out.println("</table></body></html>");
}
}

```

Quando a instrução `req.getParameterNames()` é invocada, o objeto de requisição do Servlet lê os dados do *form* e decodifica em pares, sendo parâmetros/valores (na verdade, quando qualquer das instruções `getParameterNames()`, `getParameterValues()` ou `getParameter()` são requisitadas pela primeira vez, a decodificação ocorre).

O método `getParameterNames()` retorna uma lista com todos os parâmetros. O método `getParameter()` retorna o valor associado com um parâmetro em particular. Quando o Servlet solicita uma análise dos pares parâmetros/valores, os valores para cada parâmetro são armazenados como um *array* de *String*, no caso de um parâmetro possuir múltiplos valores utilizando o método `getParameterValues()`, retornará um *array* de *String* para o parâmetro determinado, enquanto que `getParameter()` apenas retorna o primeiro elemento do *array* de *String* de um parâmetro que possui múltiplos valores.

Geralmente Servlets são processados em um ambiente *multi-thread*. Sabendo-se que somente uma instância de um objeto Servlet é criado pelo servidor *Web*, o Servlet deve prover serviços de requisições simultâneos. Os Servlets, então, precisam assegurar, quando necessário, o acesso sincronizado para recursos compartilhados, como suas variáveis de uma instância, conexões com Banco de Dados e acesso a arquivos.

Digitado e compilado com sucesso, este Servlet do exemplo anterior, poderá ser colocado em funcionamento, basta colocar o código compilado, extensão “*.class” no diretório específico para Servlets em seu servidor de *Web*. Qualquer arquivo colocado no diretório específico para Servlets será executado como um Servlet a todo momento que a requisição a este diretório seja parte da URL. Basicamente a instância do Servlet será acionada pela URL: <http://www.servidor.com.br/servlet/BasicCGIServlet>. O Servlet também pode ser acionado através do *action* de um *form* HTML.

3.7. Conclusão

Os Servlets provêem um conjunto de ferramentas padronizadas para a construção de aplicações Java, chamadas *server-side* (processamento no servidor). Enquanto são apenas uma única parte de todo o conjunto das ferramentas Java, que cresce rapidamente, o Servlet oferece aos desenvolvedores Java, fácil acesso a toda a infra-estrutura Java sem requisitar muito tempo de treinamento adicional.

A API (*Application Program Interface*) e as implementações são simples, provendo um alívio as dificuldades de múltiplos sistemas operacionais (*write once run anywhere*) e na despadronização de máquinas virtuais como ocorre em outras linguagens.

Em comparações aos tradicionais CGIs e outras soluções existentes atualmente, os Servlets são eficientes em processamento, convenientes, fácil de implementar, com mais recursos, independentes de plataforma e principalmente, a tecnologia Java possui um custo baixo (servidores e aprendizado), que para as corporações que desejam ou já fazem parte da Internet é uma das características que mais chama a atenção.

Os Servlets executam mais rápido, seus *byte-codes*, podem ser transferidos para outro servidor WWW sem necessidade de alterações no código ou recompilações, tem uma grande facilidade de ligação com Banco de Dados (através de ODBC ou JDBC – capítulo 5). Além disso, permite a programação de maneira multi-tarefa, aproveitamento de código e todas as vantagens que uma linguagem orientada a objetos possui.

Com o surgimento de novas tecnologias *server-side*, para a criação de conteúdo dinâmico de documentos HTML, como o ASP (*Active Server Pages*) a implementação dos Servlets por parte dos “programadores” foi sendo dirigida cada vez mais para a facilidade na escrita dos códigos Java, a partir deste ponto de vista o JSP (*Java Server Pages*) foi criado. O próximo capítulo (JSP) aborda os conceitos e características desta nova tecnologia *server-side* Java. Facilitando a implementação de códigos tão poderosos quanto os Servlets, mas com recursos que provem uma interação direta com os códigos HTML na manipulação das páginas dinâmicas.

4. *JAVA SERVER PAGES (JSP)*

Observa-se grandes mudanças nos sites da Internet. Já há algum tempo todos os grandes sites têm tentado interagir e descobrir o gosto de seus visitantes, guardando as suas preferências e características. Isto pode ser visto na própria *Amazon.com*, por exemplo, que a partir da busca de um livro apresenta outras publicações similares dentro do mesmo tema, focando assim a sua propaganda.

Entre os representantes mais populares desta tecnologia que utiliza página HTML “misturada” com *script*, temos as páginas ASP (*Active Server Pages*) da *Microsoft*, PHP (*br.php.net*), que surgiu para fortalecer a geração de páginas em servidores *Unix* mas já está disponível para *Linux* e *Windows NT*, o *JScript Server Side* da *Netscape* e o *ColdFusion*.

Além de todas estas soluções citadas acima temos o *Java Server Pages* (JSP). Assim como as páginas ASP ou o PHP, as páginas JSP contém pequenas porções de código (*scriptlets*) junto ao código HTML que são processadas pelo servidor e depois disto enviadas para o cliente. Existem algumas vantagens desta nova tecnologia frente às outras. É discutido neste capítulo principalmente, as diferenças entre JSP (*Java Server Pages*) e ASP (*Active Server Pages*).

A tecnologia JSP prove um método simples e rápido para a criação de páginas *Web* que exibem conteúdos gerados dinamicamente. *Java Server Pages* foi projetado para efetuar construções rápidas e fáceis quando se trata de aplicações *Web* que devem ser funcionais numa grande variedade de servidores *Web*, servidores de aplicações, *browsers*, e ferramentas de desenvolvimento.

A definição de uma página JSP e suas vantagens na utilização desta tecnologia e comparações com uma outra tecnologia (ASP) serão abordadas a seguir.

4.1. O que é *Java Server Pages* ?

Segundo a Sun Microsystems[2], a tecnologia Java Server Pages (JSP) “...é uma página criada por desenvolvedores *Web* que inclui *tags* específicas da tecnologia JSP, declarações, e possíveis *scriptlets*, em combinação com outras *tags* estáticas (HTML ou XML). A tecnologia JSP possui a extensão *.jsp*, que é a identificação para o servidor *Web* que suporta a tecnologia JSP e processa elementos nesta página.”

Java Server Pages é parte da família Java, utiliza a linguagem Java como base do seu *script* de desenvolvimento, e as páginas JSP são compiladas em *Servlets*. As páginas JSP podem utilizar componentes Java, chamados de *JavaBeans*, para executarem processos no servidor. Assim, a tecnologia JSP é o componente chave para uma arquitetura altamente escalonável nas aplicações *Web*.

O *engine* (motor, núcleo de processamento) do JSP interpreta *tags* e *scriptlets*, e gera o conteúdo requerido, como exemplo, chamando um *bean* (componente), que acessa um Banco de Dados utilizando JDBC (capítulo 5) ou que acesse um arquivo. Envia o resultado de volta no formato de páginas HTML (ou XML) para o *browser*. A lógica que gera o conteúdo dinâmico é encapsulada através de *tags*, *beans* e *scriptlets* que são processados no servidor. *Scriptlets* permitem aos desenvolvedores “amarrarem” todo o código em um único lugar (HTML e *scriptlets* Java), onde as *tags* HTML controlam a formatação do conteúdo da página a ser exibido.

As páginas JSP, não são restritas a nenhuma plataforma específica ou servidor *Web*.

4.2. As vantagens do Java Server Pages sobre o *Active Server Pages*

O grande ponto fraco do ASP é sem dúvida sua dependência do sistema operacional e do servidor *Web*, ou seja, ele está casado com o *Windows NT Server* e o IIS (*Internet Information Web Server*, da *Microsoft*). E as soluções que permitem que estas páginas sejam usadas nos servidores *Unix*, ainda não foram aprovadas pela maioria dos *designers* e desenvolvedores, devido principalmente à baixa performance. Outro ponto é que a tecnologia ASP tem seu padrão fechado, ou seja, somente a *Microsoft* tem total conhecimento de como as coisas funcionam internamente.

Já as páginas JSP podem ser processadas por diversos servidores *Web*, como o *Apache*, *Netscape* e IIS, e sob várias plataformas, como o *Solaris*, *Linux*, *Mac OS*, *Windows NT*, *Windows 95* e outras. Uma outra vantagem é o padrão aberto para a indústria, contando com o apoio de empresas como *Oracle*, *Netscape*, *IBM* e *WebLogic*. Isto significa que mais empresas poderão fornecer servidores *Web* compatíveis com JSP e outros adendos para facilitar o trabalho dos desenvolvedores.

Para desenvolver a lógica em páginas ASP é possível o uso das “linguagens” *VBScript* ou *Jscript*, enquanto páginas JSP podem ser desenvolvidas com *scriptlets* escritos na linguagem Java ou *Jscript*, ou usar *tags* XML que serão processadas pelo JSP engine.

A *JavaSoft*, divisão da *Sun Microsystems* que controla o Java, tem prometido a adoção de outras linguagens para a criação das páginas, o que não é um processo muito complicado. Esta diferença é importante, pois uma equipe que tem total conhecimento em *Visual Basic* e já possui um ambiente baseado em soluções *Microsoft* deve pensar muito bem antes de uma mudança como esta.

Em contrapartida, se a equipe já possui algum conhecimento em Java e se existe a possibilidade de adoção de novas plataformas (*Linux*) ou novos servidores *Web*, então olhe com bons olhos para o JSP.

Mas uma das vantagens mais marcantes do JSP é a possibilidade de integração com todas as APIs Java já existentes (*java.net*, *java.rmi*, *java.util*, *java.sql*, *java.bean*, *java.text*, *java.etc*) e principalmente a utilização de componentes *JavaBeans* e *Enterprise JavaBeans*. A tecnologia de componentes tem sido adotada cada vez mais pela comunidade de desenvolvimento de software e tem respondido com presteza às necessidades. Todos os componentes de software que você já criou para suas aplicações Java podem ser utilizados pelas páginas JSP. Além disso, outros recursos como criação de *tags* personalizadas também é possível, o que permite maior flexibilidade, podendo até criar *tags* que sejam específicas para os seu componentes *JavaBeans*.

E finalmente a promessa de aumento de performance aparece como mais um ponto positivo para a adoção desta tecnologia, pois uma página JSP é convertida em um *Servlet* Java antes de ser executada e fica na memória do servidor já compilada.

4.3. Comparação entre Java Server Pages e *Active Server Pages*

Atualmente sempre existe a dúvida de qual a melhor escolha tecnológica para um *site* que utiliza páginas dinâmicas, a utilização de ASP ou JSP. Na verdade não se pode afirmar com certeza. Cada uma das tecnologias envolvidas possui suas vantagens e desvantagens. As duas tecnologias oferecem uma escolha excelente para a construção de *sites* dinâmicos. Atualmente a principal consideração para a escolha entre as tecnologias ASP e JSP pode depender do seguinte critério:

- 1) O quão grande é o projeto ?
- 2) Qual a especialidade que o estabelecimento possui ?

- 3) Como e onde será hospedado o *site* ?
- 4) Quantos usuários concorrentes será esperado ?

Analisando com maior clareza cada critério:

- 1) O quão grande é o projeto ?

Para projetos grandes, recomenda-se a utilização da tecnologia JSP. Quando usada corretamente, pode-se determinar tarefas específicas a equipe de desenvolvedores. Alguns da equipe podem concentrar na construção de componentes (*JavaBeans*) enquanto outros membros da equipe permanecem concentrados nas partes de apresentação do projeto (*design* e códigos de interface com o usuário). Com a utilização nativa da programação orientada a objetos oferecida pelo Java, facilmente consegue-se subdividir o projeto entre equipes. Em projetos grandes, ASP tende a tornar-se um código *spaghetti* (complicado) onde a manutenção tornar-se-a impraticável. JSP é muito mais puro e fácil de efetuar manutenções em grandes projetos. Lembrando que isto deve-se também a aplicação correta da tecnologia JSP.

Para projetos pequenos, a tecnologia ASP tende a ser melhor. Devido a três fatores que podemos identificar:

- ASP é muito simples.
- Existem poucas discussões na configuração e manutenção em *sites* que utilizam ASP. Encontra-se inúmeros documentos que mostram a simplicidade da tarefa de uma configuração de um *site* baseado na tecnologia ASP.
- Objetos ADO (*Access Data Object*, da *Microsoft*). A utilização do ADO oferece ao projeto um meio simples e rápido na manipulação de dados. Funciona com a grande maioria dos Banco de Dados. A flexibilidade de recursos e velocidade do projeto é passada integralmente para a equipe do projeto, que facilita o término do projeto no tempo previsto.

- 2) Qual a especialidade que o estabelecimento possui ?

Em outras palavras, permanecer com a tecnologia que melhor atende. Não existe razão para trocar de tecnologia no caso de ASP para JSP se a especialidade da companhia for o ASP. Se não existe uma especialidade em nenhuma das tecnologias, existe uma tendência a tecnologia ASP, já que ASP possui um aprendizado mais fácil sobre o JSP/Servlets/Java. Mas por outro lado se a equipe já conhece o ambiente Java e seus recursos, JSP poderá ser a melhor opção.

- 3) Como e onde será hospedado o *site* ?

Caso seja necessário hospedar o *site Web* fora do negócio da empresa o ASP pode ser a melhor opção. Atualmente é muito mais fácil hospedar um *site* ASP do que um *site* na tecnologia JSP.

Se a hospedagem do *site* for efetuada pela própria empresa, que deseja colocar o negócio na Internet, estas limitações citadas acima deixarão de existir. Existe uma grande variedade de recursos tecnológicos envolvendo servidores *Web* e sistemas operacionais que o *site* JSP poderia estar hospedado.

- 4) Quantos usuários concorrentes será esperado ?

Pequenos *sites* que esperam cerca de 500 usuários simultâneos, podem com qualquer uma das duas tecnologias (ASP e JSP) funcionarem perfeitamente. Entretanto, quando existe a necessidade de aumentar este *site*, esperando um número maior de usuários concorrentes, e que este aumento deve ser feito rapidamente, a tecnologia JSP possui suporte muito mais amplo que o ASP para grandes *sites*. Principalmente se esta mudança depender da troca significativa de equipamentos e plataformas diferentes que iram suportar este novo *site*.

Para resumir, na Tabela 4.1, existe uma comparação entre várias características das duas tecnologias JSP e ASP.

Tabela 4.1 – Comparação entre ASP e JSP



 Recurso	<i>Active Server Pages</i>	Java Server Pages
Uso com múltiplos Servidores <i>Web</i>	Não. (somente com o IIS)	Sim. (<i>Apache, Netscape, IIS</i>)
Uso com múltiplas plataformas	Não. Somente <i>Microsoft Windows</i>	Sim. (<i>Solaris, plataformas Windows NT e outros Windows, Mac OS, Linux, etc.</i>)
<i>Tags</i> customizáveis	Não.	Sim.
Componentes reutilizáveis multiplataformas	Não.	Sim. arquitetura de componentes <i>JavaBeans</i>
Padrão aberto para a indústria	Não.	Sim. (Parceiros incluem: <i>Oracle, Netscape, IBM & WebLogic</i>)
Segurança contra <i>crash</i> (travamento) do sistema	Não.	Sim (Todos os recursos de segurança na plataforma Java incluem proteção para <i>crash</i> do sistema)
Habilidade para separar a geração do conteúdo da apresentação.	Sim. Usando Objetos COM	Sim. Usando <i>beans</i> (<i>JavaBeans</i>)
Linguagens de programação suportadas	<i>VBScript, JScript</i>	Java, <i>JavaScript</i> (intenção de adicionar mais)
Geração dinâmica de HTML	Sim.	Sim.

Tabela 4.1 – Comparação entre ASP e JSP (cont.)

 Recurso	Active Server Pages	Java Server Pages
Escalabilidade desde pequenas até grandes aplicações <i>Web</i> .	Sim.	Sim.
Integração automática dos arquivos gerados.	Sim.	Sim.
Compatibilidade com Banco de Dados legados	Sim. (usando COM)	Sim. (usando a API JDBC™)
Extensão do arquivo	.asp	.jsp
Manutenção de estado	Sim.	Sim.
Capaz de integrar com fontes de dados	Trabalha com qualquer banco de dados compatível com ODBC	Trabalha com qualquer banco de dados compatível com ODBC e JDBC
Componentes	COM	<i>Beans</i> ou <i>Tags</i>
Editável em qualquer editor de texto	Sim	Sim

Ambas as soluções, ASP e JSP, são sólidas para os *sites Web*. O mercado atualmente está dividido entre estas duas opções. Existem algumas variações no uso das duas tecnologias ao longo da Internet, mas no geral as duas soluções estão balanceadas nos “prós” e “contras”. Nota-se que o ASP está se tornando bem popular entre os *sites* de pequeno porte, enquanto o JSP já caminha para os *sites* de grande porte.

4.4. Um Exemplo

A seguir é apresentado um exemplo simples da estrutura de um *script* ASP e da tecnologia JSP. Estes exemplos visam apenas a exibição simples da estrutura das duas tecnologias destacadas e comparadas neste capítulo.

Tecnologia ASP, exemplo:

```
<HTML>
<BODY>
<%for i=1 to 6%>
<H<%=i%>>Fonte tamanho=<%=i%> <BR>
<% next %>
</BODY>
</HTML>
```

Este mesmo script em uma página JSP:

```
<HTML>
<BODY>
<% int i;
for (i=1; i<=6; i++) {%>
<H<%=i%>> Fonte tamanho=<%=i%> <BR>
<%}%>
</BODY>
</HTML>
```

A única diferença no código é exatamente a sintaxe de Java e *VBScript*.

Um exemplo mais complexo de JSP está abaixo, fazendo chamadas a componentes *JavaBean* e *tags* personalizadas.

```
<HTML>

<%@ page language=="java" imports=="com.servidor.JSP.*"%>

<H1>Bem Vindo!</H1>
<P>Hoje é </P>

<jsp:useBean id=="clock" class=="calendar.jspCalendar" />

<UL>
<LI>Dia: <%=clock.getDayOfMonth() %>
<LI>Ano: <%=clock.getYear() %>
</UL>

<% if (Calendar.getInstance().get(Calendar.AM_PM) ==
Calendar.AM) { %>

    Bom Dia
  <% } else { %>
    Boa Tarde<% } %>
<%@ include file=="copyright.html" %>
</HTML>
```

4.5. Conclusão

Com o mesmo impacto ascendente da linguagem Java, a “nova” tecnologia JSP mostra muita promessa no mundo do desenvolvimento *Web*. O JSP foi modelado depois do ASP, e “tomou” emprestado muito de seus recursos, separando o conteúdo dinâmico do *design*. JSP utiliza a linguagem mais discutida nos últimos tempos, Java, na qual permite mais opções aos desenvolvedores, do que as linguagens de *script* como o ASP (Vbscript ou Jscript). JSP, possui a vantagem de ser executado em várias plataformas, como qualquer aplicação Java.

É importante estar observando o JSP, devido ao seu potencial e os melhoramentos crescentes em sua arquitetura. Desenvolvedores que estão trabalhando no ambiente *Unix* podem contar com a habilidade de desenvolver páginas com conteúdo dinâmico em Java Server Pages com a mesma facilidade e eficiência que *Active Server Pages*, mas principalmente com uma grande diferença, a utilização da linguagem Java.

Um dos recursos que enriquecem qualquer aplicação, é a possibilidade de acesso as informações

que se localizam em um Banco de Dados, como já discutida anteriormente em várias oportunidades, existe o JDBC. O JDBC é o destaque do próximo capítulo.

5. *JAVA DATABASE CONNECTIVITY* (JDBC)

A linguagem de programação Java, tornou-se uma das principais ferramentas na construção de aplicações corporativas na Internet, desde o seu lançamento oficial pela *Sun Microsystems*. Mas um elemento crucial que torna o Java bastante útil, unificando os desenvolvedores de aplicações Java nos acessos aos Bancos de Dados, é o *Java Database Connectivity* (JDBC), ou seja, uma interface comum aos Bancos de Dados, que foi apresentada logo após o lançamento do Java.

A utilização do JDBC nos programas Java permite solicitar conexões com um Banco de Dados, enviar *statements* (comandos) SQL e receber resultados das operações SQL para processamento.

De acordo com a *Sun Microsystems*, os *drivers* JDBC estão disponíveis para a grande maioria dos Bancos de Dados, incluindo os Bancos de Dados relacionais da *Oracle*, *IBM*, *Microsoft*, *Informix* e *Sybase*.

A definição, o funcionamento, a arquitetura e alguns exemplos serão abordados nos tópicos abaixo.

5.1. O que é JDBC (*Java Database Connectivity*) ?

JDBC é uma API (*Application Program Interface*) que oferece objetos para trabalharem com diversos *drivers* de Banco de Dados, conexões, *statements* SQL, e tratamento de resultados. O JDBC torna fácil a escrita de códigos Java para acesso aos Bancos de Dados. Estes objetos simplificam a programação através de abstração, herança, encapsulamento, e polimorfismo. Inclui classes que “empacotam” (encapsula) os tipos comuns da linguagem SQL e métodos úteis para interagir com tipos de datas, horas e até mesmo *time stamps* (tipo de dado onde é representado a data e hora).

O JDBC possui suporte a diferentes modelos de execução de comandos SQL. Desde *stored procedures*, *prepared queries*, e execuções diretas (dinâmicas). Segue os padrões SQL-92 para interpretar os comandos SQL de uma aplicação o que representa uma grande facilidade na utilização por parte de qualquer desenvolvedor .

5.2. Como funciona o JDBC

Um programa Java, primeiro abre uma conexão com um Banco de Dados, cria um objeto responsável pela execução de comandos SQL, passa o comando SQL para o Banco de Dados, para ser processado, através do objeto que foi criado anteriormente, e obtém o resultado bem como informações sobre todo o conjunto do resultado.

As classes JDBC pertencem ao pacote *java.sql*, e todos os programas Java que acessem um Banco de Dados utilizam os objetos e métodos deste pacote, para ler e escrever em um Banco de Dados. Um programa que utiliza JDBC precisará de um *driver* para o acesso aos dados, que representa uma interface com o Banco de Dados.

5.2.1. JDBC-ODBC

Como parte do JDBC, a *JavaSoft* (divisão da *Sun Microsystems*, que administra a tecnologia Java) fornece um *driver* para acesso as fontes de dados ODBC, através do JDBC. Este *driver* foi desenvolvido

juntamente com a *Intersolv* e é chamado de *JDBC-ODBC Bridge*. O *JDBC-ODBC Bridge* foi implementado como *JdbcOdbc.class* e uma biblioteca nativa do Java para o acesso aos *drivers* ODBC. Para a plataforma *Windows*, a biblioteca nativa é uma DLL (*Dynamic Link Library*) chamada *JdbcOdbc.dll*.

Como o JDBC é semelhante a implementação ODBC, a ponte ODBC (*ODBC Bridge*) é uma pequena camada sobre o JDBC. Internamente, este *driver* traduz os métodos JDBC para as chamadas ODBC e então interage com qualquer *driver* ODBC disponível. A grande vantagem desta “ponte” (ligação), é que o JDBC possui a capacidade de acessar a maioria dos Bancos de Dados, por causa da grande difusão de *drivers* ODBC que estão no mercado.

5.3. Exemplos

Dois exemplos são descritos a seguir, no primeiro exemplo obtém-se uma lista de todos os estudantes em um Banco de Dados através do comando SQL *SELECT*. Para cada passo, do programa Java escrito com a API JDBC segue uma breve descrição.

```
// Declara o método

public void ListaEstudantes() throws SQLException

{

    int i, NroColunas;
    String sNumero,sNome,sSobreNome;

    // Inicializa e carrega o driver JDBC-ODBC
    Class.forName("jdbc.odbc.JdbcOdbcDriver");
    // Cria um objeto de conexão
    Connection ExlCon = DriverManager.getConnection(
"jdbc:odbc:MeuBandoDeDados;uid="admin";pw="sa");

    // Cria um objeto para execução de statement
    Statement ExlStmt = ExlCon.createStatement();
    // Cria uma string SQL, executa no sistema de BD
    ResultSet Exlrs = ExlStmt.executeQuery(
"SELECT Numero, Nome, SobreNome FROM Estudantes");

    // Processa cada linha até não existir mais linhas

    System.out.println("Número do Estudante - Nome - SobreNome");

    while (Exlrs.next()) {
        // Obtém o valor das colunas e coloca nas variáveis
        sNumero = Exlrs.getString(1);
        sNome = Exlrs.getString(2);
        sSobreNome = Exlrs.getString(3);

        // Mostra o resultado
```

```

        System.out.println(sNumero, sNome, sSobreNome);
    }

}

```

O programa ilustra os passos básicos necessários para acessar uma tabela e listar alguns campos de um registro.

Neste próximo exemplo, é atualizado o campo Nome na tabela de Estudantes, sabendo-se o número do estudante a ser atualizado. Como no exemplo anterior, cada passo do código é seguido de uma breve descrição.

```

// Declara um método

public void AtualizaNomeEstudante(String sNome, String sSobreNome,
String sNumero) throws SQLException

{
    int RetValue;
    // Inicializa e carrega o driver JDBC-ODBC
    Class.forName ("jdbc.odbc.JdbcOdbcDriver");
    // Cria um objeto de conexão
    Connection Ex1Con = DriverManager.getConnection
("jdbc:odbc:StudentDB;uid="admin";pw="sa");

    // Cria um objeto para a execução do statement
    Statement Ex1Stmt = Ex1Con.createStatement();
    // Cria uma string e passa para o Banco de Dados executar

    String SQLBuffer = "UPDATE Estudantes SET Nome = "+ sNome+",
SobreNome = "+sSobreNome+" WHERE Numero = "+sNumero

    RetValue = Ex1Stmt.executeUpdate( SQLBuffer);
    System.out.println("Atualizado " + RetValue + " registros no Banco
de Dados.");

}

```

Neste exemplo, é executado um *statement* SQL e obtido o número de linhas afetadas no Banco de Dados.

Estes dois exemplos acima, mostram como é simples e fácil a manipulação do SQL junto a API JDBC em um ambiente de programação Java, sendo ele um Servlet, JSP ou até mesmo um Applet.

5.4. Conclusão

Lançado pela *Sun Microsystems*, logo após a linguagem Java ter “nascido”, oficialmente, o JDBC tornou-se um elemento fundamental na criação de aplicações baseadas na tecnologia Java. Uma interface comum na manipulação das operações efetuadas em um Banco de Dados.

Disponível para ser utilizado com a grande maioria dos Bancos de Dados que atualmente estão no mercado, o JDBC torna fácil a escrita de códigos Java que necessitam interagir com Bancos de Dados.

O JDBC não é somente um conjunto de APIs para desenvolvedores de aplicações Java, mas também um conjunto de APIs que oferecem suporte aos desenvolvedores de *drivers* para Bancos de Dados.

Sem dúvida alguma o JDBC contribui em muito, na verdade era o que faltava na tecnologia Java quando surgiu a primeira versão, para que as corporações pudessem tirar mais proveito de todo o ambiente Java.

6. CONCLUSÃO

O conjunto de soluções Java (Applets/Servlets/JSP/JDBC), oferecidos para a construção de aplicações a serem utilizadas na Internet, possuem a total capacidade de suportar os requisitos necessários para a elaboração de negócios com clientes, empresas, parceiros e fornecedores. Sendo este negócio de pequeno, médio e grande porte.

A utilização de qualquer destas soluções pode ser relativamente um processo muito fácil, permitindo aos desenvolvedores o acesso a toda a infra-estrutura Java sem requisitar muito tempo de treinamento adicional. As classes que acompanham os pacotes oferecidos pela Sun Microsystem e outros fornecedores, fornecem tais facilidades para a construção das aplicações *Web*. O conjunto de ferramentas Java não são 100% perfeitos, possuem alguns problemas como qualquer outra tecnologia, mas a sua utilização deve ser analisada e estudada, viabilizando a elaboração de qualquer negócio.

As dificuldades enfrentadas por diversas tecnologias “concorrentes”, na construção de aplicações para a Internet, como as dificuldades de múltiplos sistemas operacionais (*write once run anywhere*) e na despadronização de máquinas virtuais, conseguem ser superadas pelas soluções citadas ao longo deste trabalho, que fazem parte da “família” Java.

Lembrando que uma das características que mais chamam a atenção é o custo baixo, que para as corporações que desejam ou já fazem parte da Internet é um dos principais motivos na adoção destas soluções Java para uso na Internet. Uma outra característica marcante, para os desenvolvedores, é a programação de multitarefa, aproveitamento de código, e todas as vantagens que uma linguagem orientada a objetos possui.

Através das tecnologias abordadas neste trabalho, é identificado que pela primeira vez, na Internet, pode-se executar programas complexos no cliente, utilizando os Applets, e que esses programas possuem a capacidade de se comunicarem com um servidor para recuperar e enviar informações com um certo nível de segurança. Outras tecnologias que foram abordadas, como o JSP, são analisadas com um certo nível de importância, devido ao seu potencial e os melhoramentos crescentes em sua arquitetura.

A tecnologia Java para a Internet não para, a cada dia novas soluções vão surgindo e compondo esta “família”, que já mostrou não ser mais um mito, mas sim realidade na rede mundial de computadores (Internet).

REFERÊNCIA BIBLIOGRÁFICA

[1] Davidson, James Duncan e Coward, Danny. Java Servlet Specification, v2.2, Sun Microsystems, Final Release, 1999, p11.

[2] Sun Microsystems, Java Server Pages – Frequently Asked Questions, Julho de 2000.

(Internet : <http://java.sun.com/products/jsp/faq.html>)

BIBLIOGRAFIA

Hamilton, Graham e Catell, Rick. JDBC Database Access with Java - A Tutorial and Annotated Reference, Addison Wesley, 1997.

Thomas, Michael D. e Patel, Pratik R.. Programando em Java para a Internet, Makron Books, 1997.

Deitel, H. M. e Deitel, P. J.. Java How to Program, Prentice Hall, 3ª ed., 1999.

Eckel, Bruce. Thinking in Java, Prentice Hall, 1998.

Microsoft, Mastering Web Site Development Using Visual InterDev, Course Number 793, Microsoft, 1997.

Sun Microsystems, Java Server Pages Technology White Paper, Sun Microsystems, 1999.

Java Developer's Jornal, Janeiro de 2000.

(Internet: <http://www.sys-con.com/java>)