

Java

Aplicações Web

Paulo Henrique Cayres

A RNP – Rede Nacional de Ensino e Pesquisa – é qualificada como uma Organização Social (OS), sendo ligada ao Ministério da Ciência, Tecnologia e Inovação (MCTI) e responsável pelo Programa Interministerial RNP, que conta com a participação dos ministérios da Educação (MEC), da Saúde (MS) e da Cultura (MinC). Pioneira no acesso à Internet no Brasil, a RNP planeja e mantém a rede Ipê, a rede óptica nacional acadêmica de alto desempenho. Com Pontos de Presença nas 27 unidades da federação, a rede tem mais de 800 instituições conectadas. São aproximadamente 3,5 milhões de usuários usufruindo de uma infraestrutura de redes avançadas para comunicação, computação e experimentação, que contribui para a integração entre o sistema de Ciência e Tecnologia, Educação Superior, Saúde e Cultura.



RNP

MINISTÉRIO DA
DEFESA

MINISTÉRIO DA
CULTURA

MINISTÉRIO DA
SAÚDE

MINISTÉRIO DA
EDUCAÇÃO

MINISTÉRIO DA
**CIÊNCIA, TECNOLOGIA,
INOVAÇÕES E COMUNICAÇÕES**





Java – Aplicações Web

Paulo Henrique Cayres



Java – Aplicações Web

Paulo Henrique Cayres

Rio de Janeiro
Escola Superior de Redes
2017

Copyright © 2017 – Rede Nacional de Ensino e Pesquisa – RNP
Rua Lauro Müller, 116 sala 1103
22290-906 Rio de Janeiro, RJ

Diretor Geral
Nelson Simões

Diretor de Serviços e Soluções
José Luiz Ribeiro Filho

Escola Superior de Redes

Coordenação
Leandro Marcos de Oliveira Guimarães

Edição
Lincoln da Mata

Coordenador Acadêmico da Área de Desenvolvimento de Sistemas
John Lemos Forman

Equipe ESR (em ordem alfabética)

Adriana Pierro, Alynne Pereira, Celia Maciel, Edson Kowask, Elimária Barbosa, Evellyn Feitosa, Felipe Nascimento, Lourdes Soncin, Luciana Batista, Luiz Carlos Lobato, Renato Duarte e Yve Marcial.

Capa, projeto visual e diagramação
Tecnodesign

Versão
1.1.0

Este material didático foi elaborado com fins educacionais. Solicitamos que qualquer erro encontrado ou dúvida com relação ao material ou seu uso seja enviado para a equipe de elaboração de conteúdo da Escola Superior de Redes, no e-mail info@esr.rnp.br. A Rede Nacional de Ensino e Pesquisa e os autores não assumem qualquer responsabilidade por eventuais danos ou perdas, a pessoas ou bens, originados do uso deste material.

As marcas registradas mencionadas neste material pertencem aos respectivos titulares.

Distribuição
Escola Superior de Redes
Rua Lauro Müller, 116 – sala 1103
22290-906 Rio de Janeiro, RJ
<http://esr.rnp.br>
info@esr.rnp.br

Sumário

1. Soquets

Exercício de nivelamento – Soquets 1

Protocolos de comunicação 1

Soquetes 3

Socket TCP (stream) 6

Socket UDP 6

Raw Sockets 6

Conexões 6

Implementação de soquetes TCP 8

Implementação do socket servidor 8

Implementação do socket cliente 9

Sockets UDP 10

Multicast 13

Raw Sockets 15

Exercícios de fixação – Socket 16

2. RMI – Remote Method Invocation

Introdução 17

Arquitetura RMI 20

Implementação de uma Aplicação RMI 21

Executando uma aplicação RMI 25

Exercícios de fixação – RMI 26

3. Applets

Exercício de nivelamento – Applets 27

Introdução 27

Ciclo de vida de um applet 29

Um primeiro exemplo de applet 30

A classe applet 30

Executando um applet 31

Uso de parâmetros em applets 34

Conversão de aplicação em applet 35

Manipulação de eventos no applet 36

Outros métodos disponíveis para um applet 37

Restrições de segurança 40

4. Servlets

Definição 41

Arquitetura Servlets 42

Um exemplo de Servlet 43

Estrutura de uma aplicação web 46

Arquivo web.xml 47

Executando um Servlet 47

Servlet com passagem de parâmetros 48

Exercícios de fixação – Servlets 50

Atividades Práticas 50

5. JSP (Java Server Pages)

Introdução 51

Elementos sintáticos 52

Diretivas 53

Ações 54

Elementos de script 54

Exemplos JSP 56

Objetos implícitos 57

Objeto page	59
Objeto config	60
Objeto request	61
Objeto response	62
Objeto session	65
Objeto application	66
Objeto pageContext	67
Objeto exception	69
Exercícios de fixação – JSP	70
Atividades Práticas	70

6. JSP e banco de dados

JDBC	71
Padronizando a conexão ao BD	74
DAO e JavaBean	75
Arquitetura MVC	78
Definição da interface gráfica com o usuário	78
Exemplo JSP – Listagem geral	81
Exemplo JSP – Exclusão	82
Exemplo JSP – alteração	82
Ferramenta de relatório JasperReport	84
Exercícios de fixação – JSP e banco de dados	86

7. Taglib e JSTL

Taglib	87
JSTL	90
Estrutura JSTL	91
Expression Language	92
Core taglib	93
Saída básica	94
Acesso a parâmetros	94
Manipulação de variáveis	95
Decisão simples	95
Decisão múltipla	96

Repetição 96
Tokenização 97
Inclusão de páginas (import, redirect e catch) 97
Internacionalização 97
Internacionalização 98
Formatação 98
SQL 99
SQL com DAO 100
Exercícios de fixação – Taglib e JSTL 100

8. Cookie e session

Cookies 101
Session 105
Exercício de fixação – Cookie e session 108

9. Javabeans

Componentes de software 109
JavaBeans 110
Criando um JavaBean 113
JavaBeans no IDE (Netbeans) 115
JSP com JavaBeans 116
Exercício de fixação – JavaBeans 120

10. Deployment de aplicação Java Web

Apache Tomcat 121
Deployment 123
Deployment de arquivo WAR 123
Deployment automático do arquivo WAR 124
Deployment manual 124
Exercícios de fixação – Deployment 125

Escola Superior de Redes

A Escola Superior de Redes (ESR) é a unidade da Rede Nacional de Ensino e Pesquisa (RNP) responsável pela disseminação do conhecimento em Tecnologias da Informação e Comunicação (TIC).

A ESR nasce com a proposta de ser a formadora e disseminadora de competências em TIC para o corpo técnico-administrativo das universidades federais, escolas técnicas e unidades federais de pesquisa. Sua missão fundamental é realizar a capacitação técnica do corpo funcional das organizações usuárias da RNP, para o exercício de competências aplicáveis ao uso eficaz e eficiente das TIC.

A ESR oferece dezenas de cursos distribuídos nas áreas temáticas: Administração e Projeto de Redes, Administração de Sistemas, Segurança, Mídias de Suporte à Colaboração Digital e Governança de TI.

A ESR também participa de diversos projetos de interesse público, como a elaboração e execução de planos de capacitação para formação de multiplicadores para projetos educacionais como: formação no uso da conferência web para a Universidade Aberta do Brasil (UAB), formação do suporte técnico de laboratórios do Proinfo e criação de um conjunto de cartilhas sobre redes sem fio para o programa Um Computador por Aluno (UCA).

A metodologia da ESR

A filosofia pedagógica e a metodologia que orientam os cursos da ESR são baseadas na aprendizagem como construção do conhecimento por meio da resolução de problemas típicos da realidade do profissional em formação. Os resultados obtidos nos cursos de natureza teórico-prática são otimizados, pois o instrutor, auxiliado pelo material didático, atua não apenas como expositor de conceitos e informações, mas principalmente como orientador do aluno na execução de atividades contextualizadas nas situações do cotidiano profissional.

A aprendizagem é entendida como a resposta do aluno ao desafio de situações-problema semelhantes às encontradas na prática profissional, que são superadas por meio de análise, síntese, julgamento, pensamento crítico e construção de hipóteses para a resolução do problema, em abordagem orientada ao desenvolvimento de competências.

Dessa forma, o instrutor tem participação ativa e dialógica como orientador do aluno para as atividades em laboratório. Até mesmo a apresentação da teoria no início da sessão de aprendizagem não é considerada uma simples exposição de conceitos e informações. O instrutor busca incentivar a participação dos alunos continuamente.



As sessões de aprendizagem onde se dão a apresentação dos conteúdos e a realização das atividades práticas têm formato presencial e essencialmente prático, utilizando técnicas de estudo dirigido individual, trabalho em equipe e práticas orientadas para o contexto de atuação do futuro especialista que se pretende formar.

As sessões de aprendizagem desenvolvem-se em três etapas, com predominância de tempo para as atividades práticas, conforme descrição a seguir:

Primeira etapa: apresentação da teoria e esclarecimento de dúvidas (de 60 a 90 minutos).

O instrutor apresenta, de maneira sintética, os conceitos teóricos correspondentes ao tema da sessão de aprendizagem, com auxílio de slides em formato PowerPoint. O instrutor levanta questões sobre o conteúdo dos slides em vez de apenas apresentá-los, convidando a turma à reflexão e participação. Isso evita que as apresentações sejam monótonas e que o aluno se coloque em posição de passividade, o que reduziria a aprendizagem.

Segunda etapa: atividades práticas de aprendizagem (de 120 a 150 minutos).

Esta etapa é a essência dos cursos da ESR. A maioria das atividades dos cursos é assíncrona e realizada em duplas de alunos, que acompanham o ritmo do roteiro de atividades proposto no livro de apoio. Instrutor e monitor circulam entre as duplas para solucionar dúvidas e oferecer explicações complementares.

Terceira etapa: discussão das atividades realizadas (30 minutos).

O instrutor comenta cada atividade, apresentando uma das soluções possíveis para resolvê-la, devendo ater-se àquelas que geram maior dificuldade e polêmica. Os alunos são convidados a comentar as soluções encontradas e o instrutor retoma tópicos que tenham gerado dúvidas, estimulando a participação dos alunos. O instrutor sempre estimula os alunos a encontrarem soluções alternativas às sugeridas por ele e pelos colegas e, caso existam, a comentá-las.

Sobre o curso

O curso tem como principal característica a prática em laboratório para desenvolvimento de aplicações em Java voltadas para a web/internet. São abordados desde o uso de sockets e RMI, passando pela utilização de applets, servlets e JSP (incluindo a manipulação de banco de dados com JSP) até a implementação do gerenciamento de sessões e uso de cookies. São abordados ainda a TagLib e JSTL, finalizando com o processo de Deployment de aplicações em servidores Web.

A quem se destina

O curso destina-se aos desenvolvedores de sistemas que já tenham conhecimentos relacionados com programação em Java e que queiram aprender funcionalidades e técnicas específicas para o desenvolvimento de aplicações que irão funcionar na web/internet. É um curso de nível intermediário que demanda um bom conhecimento dos fundamentos da linguagem Java e prática de programação.

Convenções utilizadas neste livro

As seguintes convenções tipográficas são usadas neste livro:

Itálico

Indica nomes de arquivos e referências bibliográficas relacionadas ao longo do texto.

Largura constante

Indica comandos e suas opções, variáveis e atributos, conteúdo de arquivos e resultado da saída de comandos.

Conteúdo de slide 

Indica o conteúdo dos slides referentes ao curso apresentados em sala de aula.

Símbolo 

Indica referência complementar disponível em site ou página na internet.

Símbolo 

Indica um documento como referência complementar.

Símbolo 

Indica um vídeo como referência complementar.

Símbolo 

Indica um arquivo de áudio como referência complementar.

Símbolo 

Indica um aviso ou precaução a ser considerada.

Símbolo 

Indica questionamentos que estimulam a reflexão ou apresenta conteúdo de apoio ao entendimento do tema em questão.

Símbolo 

Indica notas e informações complementares como dicas, sugestões de leitura adicional ou mesmo uma observação.

Permissões de uso

Todos os direitos reservados à RNP.

Agradecemos sempre citar esta fonte quando incluir parte deste livro em outra obra.

Exemplo de citação: TORRES, Pedro et al. *Administração de Sistemas Linux: Redes e Segurança*.

Rio de Janeiro: Escola Superior de Redes, RNP, 2013.

Comentários e perguntas

Para enviar comentários e perguntas sobre esta publicação:

Escola Superior de Redes RNP

Endereço: Av. Lauro Müller 116 sala 1103 – Botafogo

Rio de Janeiro – RJ – 22290-906

E-mail: info@esr.rnp.br

Sobre os autores

Paulo Henrique Cayres possui graduação no curso Superior de Tecnologia em Processamento de Dados pela Universidade para o Desenvolvimento do Estado e da Região do Pantanal (UNIDERP), especialização em Análise de Sistemas pela Universidade Federal de Mato Grosso do Sul (UFMS) e mestrado em Ciências da Computação pela Universidade Federal do Rio Grande do Sul (UFRGS). Atuou como coordenador curso de Bel. em Sistemas de Informação e Superior de Tecnologia em Redes de Computadores na Faculdade da Indústria do Sistema FIEP, onde também coordenou as atividades do SGI - Setor de Gestão de Informações. Atualmente é coordenador do Núcleo de Educação a Distância - NEaD da Faculdade da Indústria do Sistema FIEP. Sócio-diretor da CPP Consultoria e Assessoria em informática Ltda. Tem experiência na área de Ciência da Computação, com ênfase em Engenharia de Software, atuando principalmente nos seguintes temas: linguagens de programação, engenharia de software, modelagem de sistemas, desenvolvimento de aplicações para web e gerência de projetos. Professor titular em cursos de graduação e pós-graduação ministrando disciplinas de desenvolvimento de sistemas desde 1995. Instrutor de treinamento na linguagem Java de programação junto ao CITS em Curitiba e na ESR-RNP.

John Lemos Forman é Mestre em Informática (ênfase em Engenharia de Software) e Engenheiro de Computação pela PUC-Rio, com pós-graduação em Gestão de Empresas pela COPPEAD/UFRJ. É vice-presidente do Sindicato das Empresas de Informática do Rio de Janeiro – TI RIO, membro do Conselho Consultivo e de normas Éticas da Assespro-RJ e Presidente do Conselho Deliberativo da Riosoft. É sócio e Diretor da J.Forman Consultoria e atua como especialista em desenvolvimento de sistemas para a Escola Superior de Redes da RNP. Acumula mais de 30 anos de experiência na gestão de empresas e projetos inovadores, com destaque para o uso das Tecnologias da Informação e Comunicação na Educação, Saúde e Gestão Pública e Privada. Mais recentemente vem desenvolvendo projetos que fazem uso de Ciência de Dados e Aprendizagem de Máquina (Machine Learning).

1

Soquets

objetivos

Apresentar aos alunos o processo de desenvolvimento de aplicações distribuídas por meio de comunicação via socket entre programas cliente utilizando a linguagem de programação Java.

Protocolos de comunicação; Modelo OSI; TCP/IP; Socket TCP (stream); Socket UDP, Raw Socket; Conexões; Portas e MultiCast.

conceitos

Exercício de nivelamento

Soquets

Você já desenvolveu aplicações distribuídas? Que recurso e/ou linguagem de programação você já utilizou para desenvolvimento?

Protocolos de comunicação

Utilizado para tratar detalhes da camada de comunicação.

- Entre aplicações.

Tipos de protocolos de aplicação.

- FTP, SMTP, DNS, HTTP e TCP/IP.

Implementados via processos.

- Interação usando o modelo cliente-servidor.
- Usam os serviços da camada de transporte.
- Interação por meio de APIs.

Para que exista comunicação em rede, faz-se necessário o uso de um protocolo de comunicação. Um protocolo funciona como “linguagem” entre computadores em rede e para validar essa comunicação é preciso que os computadores utilizem o mesmo protocolo.

Existem diversos tipos de protocolos, tais como FTP, SMTP, DNS, HTTP e TCP/IP.

Camada de aplicação.

- Implementada por meio de processos.
 - Representação para programas em execução no SO.
- Arquitetura TCP/IP.
 - Uma aplicação é apenas um programa em execução.
 - Padrão de interação denominado modelo cliente-servidor.



- Demais camadas.
 - Implementadas diretamente no kernel do SO.



Modelo de Referência OSI



Figura 1.1
Representação
do modelo de
Referência OSI.

O TCP/IP é, na realidade, um conjunto de protocolos. Os mais conhecidos dão justamente o nome desse conjunto: TCP (Transmission Control Protocol, Protocolo de Controle da Transmissão) e IP (Internet Protocol), que operam nas camadas de transporte e internet, respectivamente.

O TCP/IP atualmente é o protocolo mais usado em redes locais. Isso se deve, basicamente, à popularização da internet. Uma das grandes vantagens do TCP/IP em relação aos outros protocolos existentes é que ele é roteável, isto é, foi criado pensando em redes grandes e de longa distância, onde pode haver vários caminhos para o dado atingir o computador receptor.

TCP-IP:

- Protocolo mais usado em redes locais.
 - Popularização da internet.
 - Protocolo roteável > redes grandes e de longa distância.
 - Arquitetura aberta.
- Servidor escolhe uma determinada porta e fica aguardando conexões.
 - Não precisa conhecer a porta utilizada pelo cliente.
 - Descobre a porta somente após receber a requisição.
- O cliente deve saber previamente qual a máquina servidora (HOST ou IP) e a porta que o servidor está aguardando conexões para solicitar uma conexão.



Outro fato que tornou o TCP/IP popular é que ele possui arquitetura aberta e qualquer fabricante pode adotar a sua própria versão de TCP/IP em seu Sistema Operacional.

Na transmissão de dados com o protocolo TCP/IP, este divide as informações que serão transmitidas através da rede em “pacotes” de dados. Todo pacote que for enviado do servidor para o cliente, ou vice-versa, terá, além dos dados que estão sendo enviados, uma série de dados de controle, tais como o número do pacote, código de validação dos dados e também o número da porta que o software utiliza. Quando o pacote chega a seu destinatário, o sistema lê no pacote o número da porta e sabe para qual aplicativo vai encaminhar o pacote. Esses controles são conferidos ao chegarem à outra ponta, pelo protocolo do receptor.



Todos os fabricantes de Sistemas Operacionais adotaram o TCP/IP, transformando-o em um protocolo universal, possibilitando que todos os sistemas possam comunicar-se entre si sem dificuldade.

Isso garante que um dado qualquer chegue a outro ponto da mesma forma que foi transmitido. Sendo assim, a integridade dos dados é mantida, independente do meio utilizado na transmissão, seja ele por linha telefônica, canal de dados, canais de voz, satélite ou qualquer outro meio de transmissão, já que o controle é feito nas pontas. Se na transmissão ocorre algum erro, o protocolo deve enviá-los novamente até que cheguem corretamente.

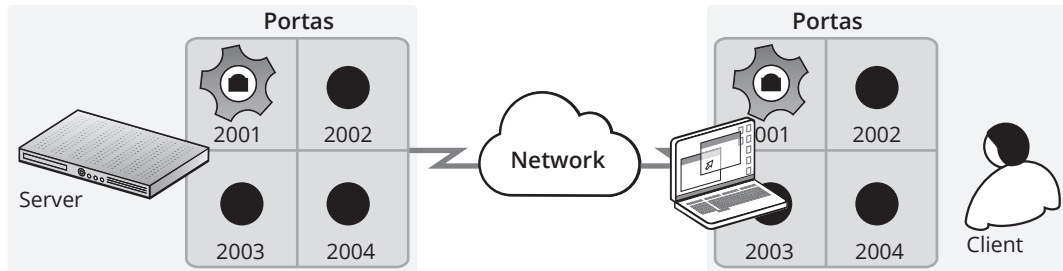


Figura 1.2
Servidores, clientes
e portas.

O servidor fica em um loop aguardando novas conexões e gerando sockets para atender as solicitações de clientes. Os segmentos TCP são encapsulados e enviados em pacotes de dados.

Uma forma de visualizar o funcionamento de socket TCP seria compará-lo a uma ligação telefônica onde alguém faz uma ligação para outra pessoa e, quando essa atende, é criado um canal de comunicação entre os dois falantes.

Soquetes

Essencialmente é uma conexão de dados transparente entre dois computadores em uma rede.

- Um dos computadores é chamado servidor.
 - ▣ Abre um socket e presta atenção às conexões.
- Demais computadores denominam-se clientes.
 - ▣ Chamam o socket servidor para iniciar a conexão.

Um socket é uma representação interna do SO para um ponto de comunicação.

- Identificado pelos endpoints local e remoto.
- Utilizados para representar a comunicação entre um programa cliente e um programa servidor.

As aplicações de sockets são feitas baseadas no protocolo TCP/IP.

Cada endpoint é representado pelo par: IP e porta.

Primeiro o socket do servidor é aberto e fica monitorando as conexões, depois o socket do cliente chama o socket do servidor para iniciar a conexão.

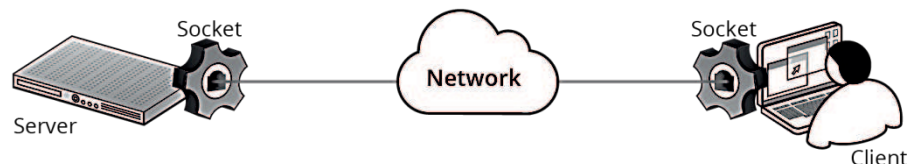


Figura 1.3
Interface Socket.

As operações em rede envolvem a capacidade de estabelecer conexões do aplicativo cliente ou aplicativo servidor através da rede. Essas operações na tecnologia Java envolvem as classes do pacote `java.net`, que oferecem abstrações entre plataformas para operações comuns em rede, como a conexão e recuperação de arquivos utilizando protocolos comuns da web e criando soquetes básicos como os do UNIX^(tm).

Originalmente proposta para sistema Unix e linguagem C.

Passou a ser amplamente utilizada.

Exemplos:

- Winsocket em SO Windows.
- Socket, DatagramSocket, ServerSocket.
 - ▣ Classes Java.

Quando os processos se comunicam por uma rede, a linguagem Java utiliza novamente o modelo de fluxos. Nesse caso, um soquete contém dois fluxos: um de entrada e outro de saída. Um processo envia dados para outro processo através da rede simplesmente gravando no fluxo de saída associado ao soquete. O processo lê os dados que foram gravados pelo processo na outra extremidade da conexão pela simples leitura do fluxo de entrada associado ao soquete. A figura 1.4 apresenta a hierarquia de classes do pacote `java.net`.



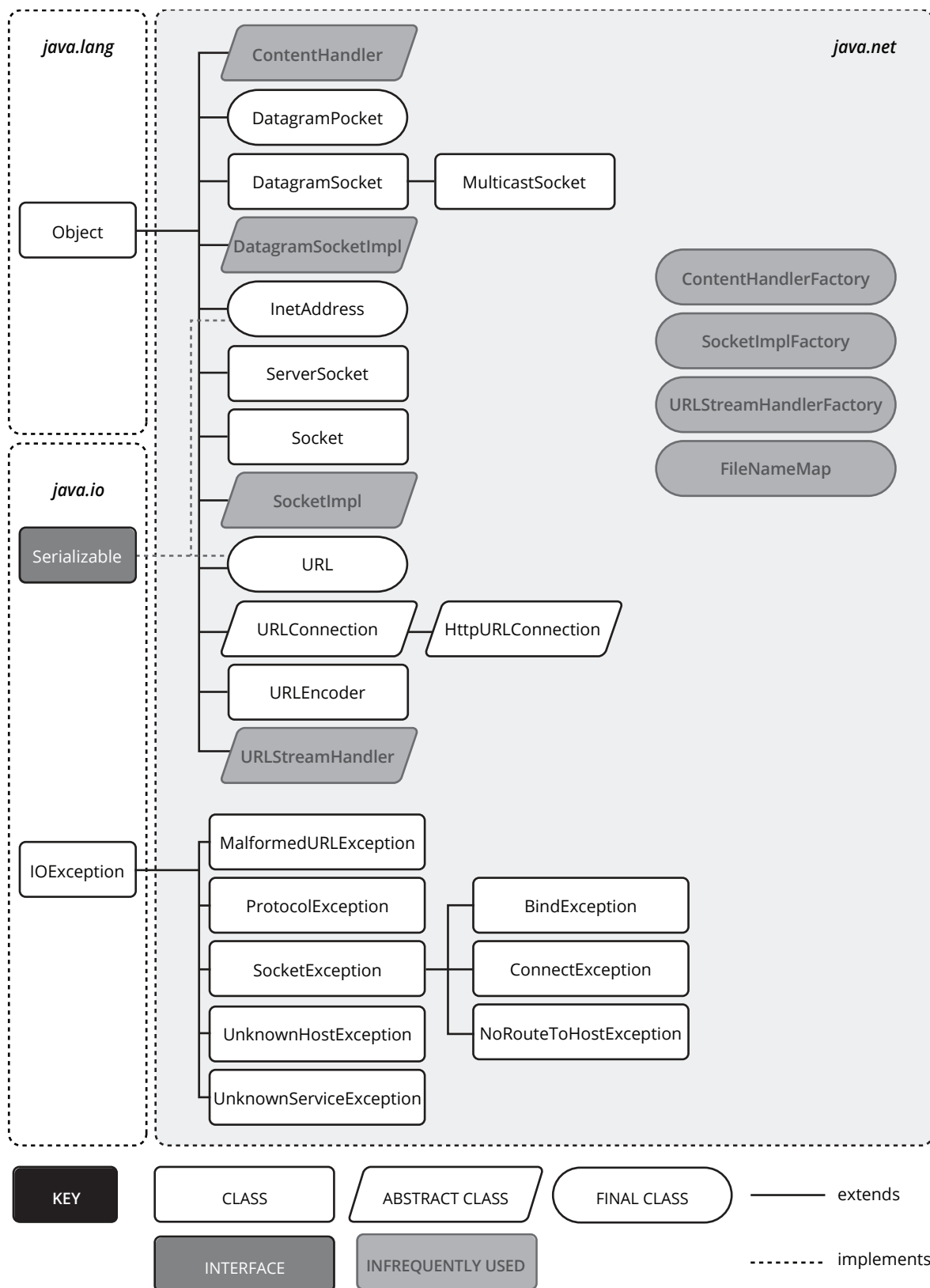


Figura 1.4
 Hierarquia de
 classes pacote
 java.net.

Os computadores em rede direcionam os streams de dados recebidos da rede para programas receptores específicos, associando cada programa a um número diferente, que é a porta do programa. Da mesma forma, quando o tráfego de saída é gerado, o programa de origem recebe um número de porta para a transação. Determinados números de porta são reservados no TCP/IP para protocolos específicos, por exemplo, 80 para HTTP.

O protocolo TCP/IP oferece os seguintes modos de utilização de sockets:

- **Socket TCP (stream):** modo orientado a conexão, que funciona sobre o protocolo TCP.
- **Socket UDP:** modo orientado a datagrama, funciona sobre o protocolo UDP, que fica na camada de transporte do protocolo TCP/IP.
- **Raw Socket:** forma mais rústica que faz uso do Internet Control Message Protocol (ICMP). Essa interface de socket não fornece serviço ponto-a-ponto tradicional.

Socket TCP (stream)

O processo de comunicação entre aplicativos no modo orientado à conexão ocorre da seguinte forma: o servidor escolhe uma determinada porta e fica aguardando conexões dessa porta. O cliente deve saber previamente qual a máquina servidora (HOST ou IP) e a porta que o servidor está aguardando conexões para solicitar uma conexão.

Socket UDP

O User Datagram Protocol (UDP) é usado por alguns programas, em vez de TCP para o transporte rápido de dados entre HOSTS dentro de uma rede TCP/IP. Em UDP não se estabelece conexão, pois a comunicação ocorre apenas com o envio de dados. Nesse tipo de socket, uma mensagem é um datagrama, que é composto por um remetente, um destinatário ou receptor e a mensagem. Caso o destinatário não esteja aguardando uma mensagem, ela é perdida, pois esse protocolo não realiza a verificação de transmissão de dados.

Raw Sockets

Envia o pacote sem utilizar as camadas de transporte. Como foi visto anteriormente, é usada de forma mais rústica na camada de rede (IP) e na Internet Control Message Protocol (ICMP). Essa interface de socket não fornece serviços ponto-a-ponto tradicional.

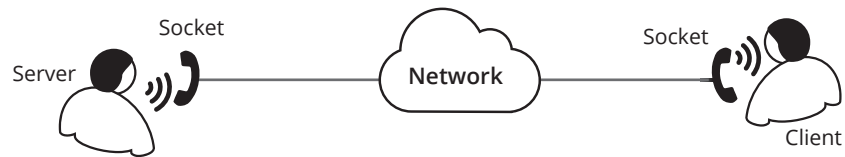
Conexões

Para que seja efetuada a comunicação em rede através de sockets, é necessário ter conhecimento do número de IP ou HOST do computador e o número de porta do aplicativo ao qual se quer realizar a conexão.

O endereço IP identifica uma máquina específica na internet e o número de porta é uma maneira de diferenciar os processos que estão sendo executados no mesmo computador. É exatamente a combinação do IP com o número de porta do aplicativo que se denomina sockets.

Após a conexão ser estabelecida, utilizar os fluxos associados a essa conexão não é muito diferente de utilizar qualquer outro fluxo, como mostra a figura 1.5.

Figura 1.5
Comunicação
cliente/Servidor
via socket.



Para estabelecer a conexão, uma máquina deve estar executando um programa que aguarda a conexão e a outra extremidade deve tentar alcançar a primeira. Tal como um sistema de telefonia, uma parte deve efetuar a chamada enquanto a outra parte deve aguardar junto ao fone quando a chamada é efetuada, conforme ilustrado na figura 1.6.

Ao efetuar uma ligação telefônica, é necessário conhecer o número a ser discado. Ao estabelecer uma conexão de rede, é necessário conhecer o endereço ou o nome da máquina remota. Além disso, uma conexão de rede exige o número de uma porta, que você pode associar a um ramal. Após conectar-se ao computador apropriado, é necessário identificar um objetivo em particular para a conexão. Portanto, da mesma forma que você utiliza um determinado número de ramal para conversar com o departamento de contabilidade, é possível utilizar um determinado número de porta para se comunicar com o programa de contabilidade.

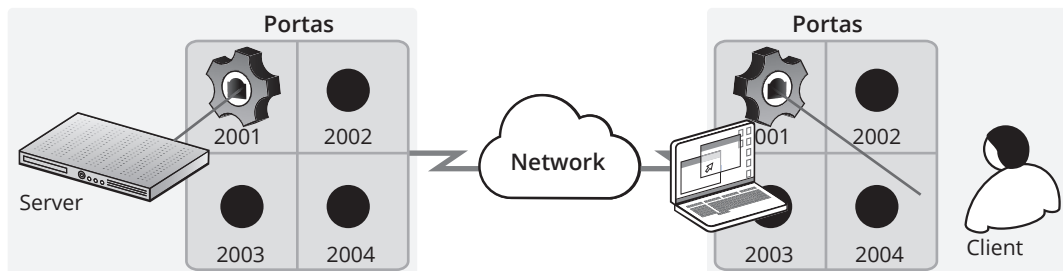


Figura 1.6
Estabelecimento
de conexão.

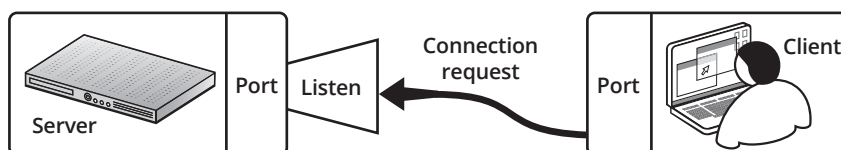
Números de porta em sistemas TCP/IP.

- São números de 16 bits, no intervalo de 0-65535.
- Números de porta a seguir de 1024 são reservados para serviços predefinidos.
 - ▣ Não devem ser utilizados, a menos que deseje se comunicar com um desses serviços (p.e., TELNET, correio SMTP, FTP e outros).

É necessário que os dois programas cheguem a um acordo com antecedência, tanto o cliente que inicia a conexão quanto o servidor que “aguarda o telefonema”. Se os números de porta utilizados pelas duas partes do sistema não estiverem de acordo, a comunicação não ocorrerá.

O servidor atribui o número de uma porta. Quando o cliente requisita uma conexão, o servidor abre a conexão do soquete com o método `accept()`.

Figura 1.7
Servidor aguarda
uma solicitação
de conexão.



O cliente estabelece uma conexão com o host na porta `port_number`.

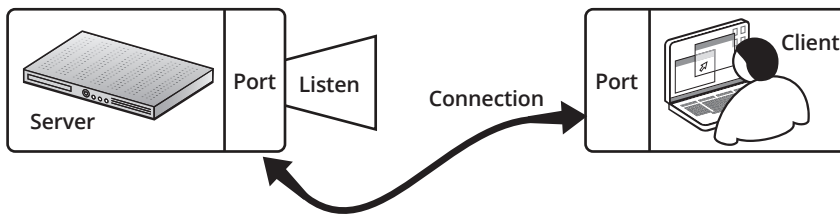


Figura 1.8
Cliente usa Socket
para se comunicar
com o servidor.

O cliente e o servidor se comunicarão por meio de um InputStream e um OutputStream.

Implementação de soquetes TCP

Basicamente, temos de nos preocupar com as tarefas descritas a seguir para realizar a implementação de uma aplicação que vai utilizar-se de soquets para possibilitar a comunicação entre máquinas clientes e servidor:

Tarefas da máquina servidora:

- Criar um socket de escuta (LISTENING), em uma determinada porta.
- Processar requisições de conexões.
- Processar pedido de fluxo.
- Utilizar o fluxo, conversa com a outra máquina.
- Fechar o fluxo.
- Fechar o socket.

Tarefas da máquina cliente:

- Criar a conexão, socket cliente e socket servidor.
- Criar um fluxo de comunicação no socket.
- Utilizar o fluxo, conversa com a outra máquina.
- Fechar o fluxo.
- Fechar o socket.



Implementação do socket servidor

Os aplicativos servidores TCP/IP baseiam-se nas classes de operação em rede ServerSocket e Socket disponibilizadas pela linguagem Java. A classe ExServidor01 tem a maior parte da responsabilidade no estabelecimento da conexão.

O código para implementar esse modelo simples é mostrado a seguir.

```
package visao;
import java.net.ServerSocket;
import java.net.Socket;
public class ExServidor01 {
    public static void main(String argv[]) {

        while (true) {
            try {
                ServerSocket socketRecepcao = new ServerSocket(3000);
                System.out.println("Servidor esperando conexão na porta 3000");
                Socket socketConexao = socketRecepcao.accept();
```

```

        System.out.println("Conexão estabelecida na porta " +
                           socketConexao.getPort());

        socketConexao.close();
        socketRecepcao.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
}
}

```

O primeiro passo é criar o server socket: `ServerSocket serverSocket = new ServerSocket(3000)`. O segundo passo é implementar a instrução que vai aguardar conexões de clientes: `Socket clientSocket = serverSocket.accept()`; na sequência a aplicação pode implementar streams de entrada e saída para comunicação com o cliente:

```

DataInputStream inbound = new DataInputStream(clientSocket.getInputStream());
DataOutputStream outbound = new DataOutputStream(clientSocket.getOutputStream());

```

O próximo passo é o uso dos streams de entrada e saída para realizar a troca de informações entre clientes e servidor.

```

int k = inbound.readInt();
String s = inbound.readUTF();
outbound.writeInt(3);
outbound.writeUTF("Hello");

```

Para finalizar, devemos providenciar o fechamento dos streams e socket do cliente e do servidor:

```

inbound.close();
outbound.close();
clientSocket.close();
serverSocket.close();

```

Implementação do socket cliente

Já o lado cliente de um aplicativo TCP/IP baseia-se na classe `Socket`. Novamente, muito do trabalho envolvido no estabelecimento das conexões foi realizado pela classe `Socket`. Esse cliente se conecta ao servidor por meio da especificação do seu nome ou IP e da porta de conexão. A classe `ExCliente01` exemplifica a implementação de uma classe cliente que vai se comunicar com o servidor implementado na classe `ExServidor01`, apresentada anteriormente.

```

package visao;
import java.net.Socket;
public class ExCliente01 {
    public static void main(String argv[]) throws Exception {
        Socket socketCliente = new Socket("localhost",3000);
        System.out.println("Socket cliente é criado e finalizando...");
        socketCliente.close();
    }
}

```

A primeira linha implementada no método `main()` da classe `ExCliente01` vai instanciar um objeto da classe `Socket`. Esse objeto é o responsável por estabelecer a conexão com o servidor, sendo este recebido pelo método `accept()` da classe `ServerSocket`. A partir dessa linha, o programa cliente é identificado pelo programa servidor e esses poderão realizar operações de envio e recepção de informação como, por exemplo, o envio de uma mensagem, leitura de um arquivo em disco, entre outras operações desejadas entre o cliente e servidor.

Ou seja, o primeiro passo é a abertura de uma conexão por meio da instância de um objeto da classe `Socket`: `Socket clientSocket = new Socket("localhost", 3000)`. Em seguida, o cliente pode se utilizar de streams de entrada e saída para realizar a troca de informações com o servidor

```
DataInputStream inbound = new DataInputStream(clientSocket.getInputStream());
DataOutputStream outbound = new DataOutputStream(clientSocket.getOutputStream());
```

O passo seguinte é o uso dos métodos dos objetos de streams enviar ou receber informações do servidor.

```
outbound.writeInt( 3 );
outbound.writeUTF( "Hello" );
int k = inbound.readInt( );
String s = inbound.readUTF( );
```

Para finalizar, devemos providenciar o fechamento dos streams e socket do cliente e do servidor:

```
inbound.close();
outbound.close();
clientSocket.close();
```

Percebam que o processo de criação das aplicações cliente/servidor é basicamente o mesmo, com a diferença que o servidor deve utilizar um objeto da classe `ServerSocket` para poder especificar um canal de comunicação que fica à espera de uma solicitação de conexão de possíveis clientes.

Sockets UDP

Enquanto o TCP/IP é um protocolo orientado a conexão, User Datagram Protocol é um protocolo sem conexão. As diferenças entre esses dois protocolos são semelhantes às diferenças entre uma chamada telefônica e uma correspondência postal ou, o envio de uma mensagem em uma garrafa, como mostra a figura 1.9.

UDP (User Datagram Protocol):

- Usado para o transporte rápido de dados entre HOSTs dentro de uma rede TCP/IP.
- Não se estabelece conexão.
- A comunicação ocorre apenas com o envio de dados.
- Uma mensagem é um datagrama.
 - ▣ Composto de um remetente, um destinatário ou receptor e a mensagem.
- Não realiza a verificação de transmissão de dados.

O User Datagram Protocol é suportado por meio de duas classes Java, `DatagramSocket` e `DatagramPacket`. O pacote é uma mensagem autossuficiente que inclui informações sobre o remetente, a própria mensagem e seu comprimento. As principais aplicações do UDP são aplicações como transmissões de vídeo, Skype, VOIP etc.



As chamadas telefônicas garantem a existência de uma comunicação síncrona: as mensagens são enviadas e recebidas na ordem pretendida. A conversação através de cartões postais enviados por correio não é garantida: as mensagens podem ser recebidas fora de ordem, se forem recebidas.



Figura 1.9
Conexão UDP –
protocolo sem
conexão.



A classe `DatagramPacket` possui dois construtores: um para receber e outro para enviar dados, descritos a seguir:

- **`DatagramPacket (byte[] recvBuf, int readLength)`**: utilizado para definir uma matriz de bytes para receber um pacote UDP. A matriz de bytes está vazia quando é passada ao construtor e a variável `int` é definida com o número de bytes a serem lidos (e também não pode ser maior que o tamanho da matriz de bytes).
- **`DatagramPacket(byte[] sendBuf, int sendLength, InetAddress iaddr, int iport)`**: utilizado para definir um pacote UDP para transmissão. `sendLength` não pode ser maior que a matriz de bytes `sendBuf`.

Já a classe `DatagramSocket` é utilizada para ler e gravar pacotes UDP. Essa classe possui três construtores que permitem especificar a que porta e endereço da internet deve ser vinculada:

- **`DatagramSocket()`**: vincular a qualquer porta disponível no host local.
- **`DatagramSocket(int port)`**: vincular à porta especificada no host local.
- **`DatagramSocket(int port, InetAddress iaddr)`**: vincular ao número de porta especificado no endereço especificado.

O código a seguir exemplifica a implementação de uma classe java criada para receber uma mensagem por meio do protocolo UDP:

```
package visao;
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
public class UDPReceptor {
    public static void main(String[] args) {
        try {
            //Argumento inteiro (numero da porta)
            int port = 1234;
            //Cria o DatagramSocket para aguardar mensagens, nesse momento o
            // método fica bloqueando até o recebimento de uma mensagem
            DatagramSocket dsReceptor = new DatagramSocket(port);
            System.out.println("Ouvindo a porta: " + port);
            //Preparando o buffer de recebimento da mensagem
            byte[] msg = new byte[256];
            //Prepara o pacote de dados
            DatagramPacket pkg = new DatagramPacket(msg, msg.length);
            //Recebimento da mensagem
            System.out.println("Pronto para receber mensagem.");
            dsReceptor.receive(pkg);
            javax.swing.JOptionPane.showMessageDialog(null, new
                String(pkg.getData()).trim(), "Mensagem recebida", 1);
        }
    }
}
```

```

        System.out.println("Mensagem recebida.");
        dsReceptor.close();
    }
    catch(IOException ex) {
        System.out.println(ex.toString());
    }
}
}

```

Quando executada, a classe vai produzir a seguinte saída no console:

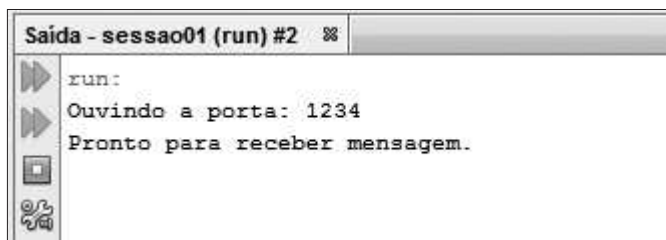


Figura 1.10
Execução do
receptor UDP
Socket no console.

Isso indica que a classe criada para receber uma mensagem UDP está em execução e o método `receive()` da classe `DatagramSocket` está à espera do pacote de dados. Após a execução desse método, a seguinte saída será gerada:



Figura 1.11
Mensagem escrita
na interface UDP
receptora.

O código a seguir exemplifica a implementação de uma classe java criada para enviar uma mensagem por meio do protocolo UDP:

```

package visao;
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
public class UDPRemetente {
    public static void main(String[] args) {
        try {
            //Primeiro argumento é o nome do host destino
            InetAddress addr = InetAddress.getByName("localhost");
            int port = 1234;
            byte[] msg = "Exemplo de Socket UDP".getBytes();
            //Monta o pacote a ser enviado
            DatagramPacket pkg =
                new DatagramPacket(msg,msg.length, addr, port);

            // Cria o DatagramSocket que será responsável por enviar a mensagem
            DatagramSocket ds = new DatagramSocket();

```

```

        //Envia a mensagem
        ds.send(pkg);
        System.out.println("Mensagem enviada para: " +
            addr.getHostAddress() + "\n" + "Porta: " + port + "\n" +
            "Mensagem: " + "Exemplo de Socket UDP");
        //Fecha o DatagramSocket
        ds.close();
    }
    catch(IOException ex) {
        System.out.println(ex.toString());
    }
}
}

```

Quando executada, a classe vai produzir a seguinte saída no console:

```

Saída
sessao01 (run) #2  sessao01 (run) #3
run:
Mensagem enviada para: 127.0.0.1
Porta: 1234
Mensagem: Exemplo de Socket UDP
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)

```

Figura 1.12
Execução do
remetente UDP
Socket no console.

Multicast

Técnica que possibilita enviar datagramas para um grupo de clientes conectados na rede.

- É suportado pelo UDP.
- O que consiste em mais uma diferença em relação ao TCP.
- Java implementa a classe `MulticastSocket` do pacote `java.net`.
 - ▣ Assemelha-se bastante com um socket datagram.
 - ▣ Capacidades adicionais, como formação de grupos multicast.
- Endereços reservados.
 - ▣ Classe D.
 - ▣ Entre 224.0.0.0 e 239.255.255.255.
 - ▣ Endereços com prefixo 239 reservados para uso em intranets. Veja a seguir o código para implementar um servidor MultiCast.

Veja a seguir o código para implementação de um Servidor MultiCast.

```

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

public class ExMultiCastSocketServer {

    public static void main(String[] args) {

```



```
//Criando um grupo
// MULTICAST Classe D (224.0.0.0 - 239.255.255.255)

try {
    byte[] b = javax.swing.JOptionPane.showInputDialog("Digite uma
        mensagem: ").getBytes();

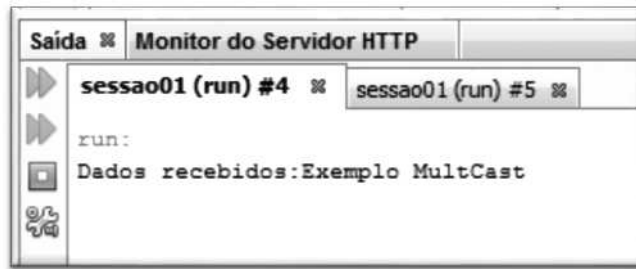
    InetAddress addr = InetAddress.getByName("239.0.0.1");
    DatagramSocket ds = new DatagramSocket();
    DatagramPacket pkg = new DatagramPacket(b, b.length, addr, 12347);
    ds.send(pkg);
} catch (IOException ex) {
    System.out.println(ex.toString());
}
}
```

E também o código para implementação de um cliente MultiCast.

```
import java.net.DatagramPacket;
import java.net.InetAddress;
import java.net.MulticastSocket;
public class ClienteMulticast {
    public static void main(String[] args) {
        while(true) {
            try {
                MulticastSocket mcs = new MulticastSocket(12347);
                InetAddress grp = InetAddress.getByName("239.0.0.1");
                mcs.joinGroup(grp);
                byte rec[] = new byte[256];
                DatagramPacket pkg = new DatagramPacket(rec, rec.length);
                mcs.receive(pkg);
                String data = new String(pkg.getData());
                System.out.println("Dados recebidos:" + data);
            }
            catch(Exception e) {
                System.out.println("Erro: " + e.getMessage());
            }
        }
    }
}
```

Quando executada, a classe vai produzir a seguinte saída no console:

Figura 1.13
Execução
do cliente MultCast
no console.



Raw Sockets

Envia o pacote sem utilizar as camadas de transporte.

- Utiliza a forma mais rústica na camada de rede (IP) e na Internet Control Message Protocol (ICMP).
- Essa interface de socket não fornece serviços ponto-a-ponto tradicionais.
- Permite que você manipule o TCP/IP no mais baixo nível.
 - Atravessando as camadas normais de encapsulamento/desencapsulamento na pilha do TCP/IP.
 - Quando um pacote TCP/IP é enviado ele não está processado.
 - É um pacote cru (RAW).
- A aplicação que está usando o pacote é agora responsável por examinar os cabeçalhos e retirá-lo, analisando o pacote, e todo o material que está na pilha de TCP/IP.

Um socket Raw é um socket que faz exame de pacotes, contorna o TCP/IP normal processado e emite-o à aplicação que o quer.

Muito usado para:

- Escrever aplicações farejadoras “sniflers”.
- Dar pings, traceroute.

Examinando onde o cabeçalho IP é constituído.

Também pode ser usado para:

- Escrever Aplicações Gerenciadoras de Rede.
- Monitoramento de erros ICMP (Internet Control Message Protocol).

Java não suporta Raw Sockets diretamente.

- Elimina a possibilidade de se criar pacotes ICMP.
- Elimina implementação de rotinas que realizem as operações de um ping.
- Uma solução para esse problema está no uso de chamada a métodos nativos.
- A criação de métodos nativos nesse caso não parece ser complicada.

Raw sockets sobre ICMP só podem ser criados por administradores.

- Nas chamadas o SO verifica se o usuário possui direito de administrador ou não.
- Se não possuir tais direitos à chamada ao SO, retorna erro e o raw sockets não é criado.



Exercícios de fixação

Socket

Qual a diferença entre socket TCP e UDP?

Atividades Práticas

2

RMI – Remote Method Invocation

objetivos

Apresentar aos alunos a estrutura da linguagem de programação Java utilizada no desenvolvimento de aplicações que possibilitam o acesso a métodos de objetos remotamente como se fossem objetos locais em um ambiente de rede distribuído.

Arquitetura RMI; registry; interfaces remotas; stubs; skeletons; camada de transporte; Classe naming; rebind(); lookup().

conceitos

Introdução

O que é? Para que serve?

- Remote Method Invocation: RMI
- API que permite que programas Java chamem métodos de objetos remotos como se fossem objetos locais.
- União da:
 - ▣ Natureza distribuída do Java.
 - ▣ Capacidade de carregamento dinâmico de classes.
- Compreendem frequentemente dois programas separados.
 - ▣ Um servidor e um cliente.



RMI (Remote Method Invocation) habilita a chamada remota de métodos. Também possibilita que o método de um objeto, em uma máquina virtual, chame um método de outro objeto, em outra máquina virtual. Isso é feito com a mesma sintaxe e facilidade de uma invocação de objetos locais. Com RMI um programa cliente pode invocar um método em um objeto remoto da mesma forma que faz com um método de um objeto local. Todos os detalhes de conexão de rede estão ocultos, permitindo assim que o modelo de objetos tenha sua interface pública mantida através da rede, sem necessitar expor detalhes irrelevantes como conexões, portas ou endereços.

RMI trabalha basicamente com um programa servidor, um programa cliente e um programa de interface, responsável pela interligação entre o cliente e o servidor. O computador servidor é utilizado para definir o corpo de cada um dos métodos que poderão ser executados remotamente. Um passo importante para que esse método seja executado com eficiência é propiciar o registro de um ou mais objetos com o rmiregistry, que deverá estar presente no servidor. Com isso, o cliente poderá acessar os métodos dos objetos das aplicações ativas no servidor. Nesse processo é indispensável a definição de uma interface que especifique quais os métodos poderão ser executados remotamente pelo cliente no servidor.



Servidor:

- Tipicamente cria alguns objetos remotos e torna a referência a esses objetos acessíveis.

Cliente:

- Obtém a referência remota de um ou mais objetos disponíveis no servidor.
- Invoca a execução de métodos remotos.

Tal aplicação é muitas vezes referenciada como uma aplicação de objetos distribuídos

A figura 2.1 ilustra o esquema de funcionamento de uma aplicação RMI.

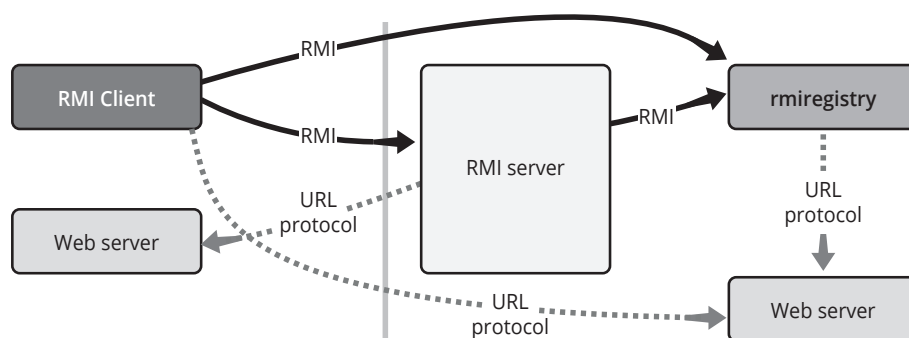


Figura 2.1
Esquema de funcionamento de uma aplicação RMI.

Aplicações de objetos distribuídos disponibilizam mecanismos para:

- Localização de objetos remotos.
- Comunicação com objetos remotos.
- Carregar definições de classes de objetos remotos.

Características centrais e singulares de RMI:

- A capacidade de baixar a definição de classe de um objeto se a classe não está definida na máquina virtual Java do cliente.
- Todos os tipos e comportamentos de um objeto, anteriormente disponível apenas em uma única máquina virtual Java, pode ser transmitida para outra máquina virtual Java, possivelmente remota.
- RMI passa objetos por suas classes reais, de modo que o comportamento dos objetos não se altera quando eles são enviados para outra máquina virtual Java.

Vantagens da leitura dinâmica de códigos:

- Essa capacidade permite que novos tipos e comportamentos possam ser introduzidos máquina virtual Java cliente.
- Aumento dinâmico do comportamento de uma aplicação.

Aplicações distribuídas RMI são desenvolvidas como qualquer outra aplicação Java, fazendo uso de interfaces, classes e objetos.

Interfaces:

- Declaram assinaturas de métodos.

Classes:

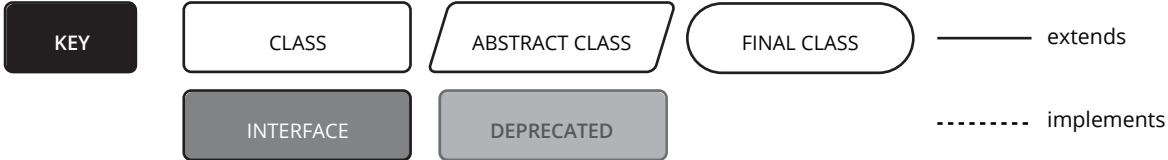
- Implementam os métodos declarados na interface.
- Podem conter métodos adicionais.



- 1

Pacote Java RMI.

em um pacote que pode ser visto na figura 2.2, a seguir.



cláusula throws, conforme pode ser visto no trecho de código a seguir.

```
public interface Calculadora extends java.rmi.Remote {
    public long add(long a, long b) throws java.rmi.RemoteException;
    public long sub(long a, long b) throws java.rmi.RemoteException;
    public long mul(long a, long b) throws java.rmi.RemoteException;
    public long div(long a, long b) throws java.rmi.RemoteException;
}
```

Arquitetura RMI

O sistema RMI consiste em três camadas:

- A camada de stub/skeleton:
 - stubs do lado cliente (proxies).
 - skeletons do lado servidor.
- A cama de referência remota – comportamento de referência remota (como invocação para um único objeto ou para um objeto reproduzido).
- A camada de transporte – configura a gerencia, conexão e localização do objeto remoto.

A figura 2.3 mostra uma representação dessa arquitetura.

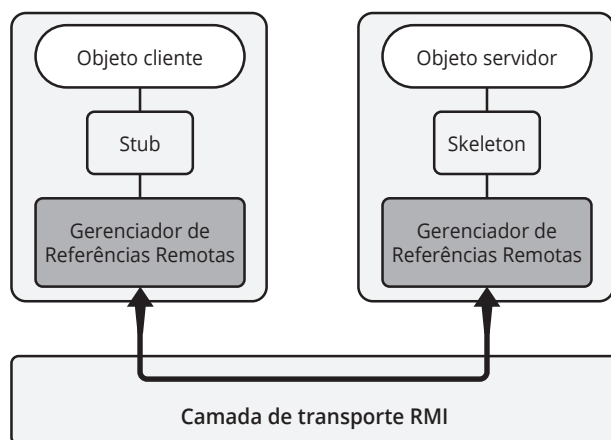


Figura 2.3
Arquitetura RMI.

Quando se utiliza a RMI, no cliente é disponibilizada uma classe chamada Stub e, no servidor, uma classe chamada Skeleton. Essas duas classes são geradas por um programa fornecido como parte do JDK (Java Developers Kit). A classe Stub aparece para o cliente como se fosse um objeto real, permitindo a chamada a cada um de seus métodos. Quando um método é chamado via Stub, os parâmetros são passados ao servidor via serialização e chegam até a classe Skeleton, que tem a finalidade de pegar os valores passados e efetuar a chamada de função do objeto real. O retorno do objeto real é transmitido do Skeleton para o Stub, e o cliente não faz a menor ideia de que o objeto está em outro local.

Ao se utilizar a RMI:

- No cliente é disponibilizada uma classe chamada Stub.
 - Aparece para o cliente como se fosse um objeto real, permitindo a chamada a cada um de seus métodos.
- No servidor, uma classe chamada Skeleton.
 - Quando um método é chamado via Stub, os parâmetros são passados ao servidor via serialização e chegam até a classe Skeleton, que tem a finalidade de pegar os valores passados e efetuar a chamada de função do objeto real.



O retorno do objeto real é transmitido do Skeleton para o Stub, e o cliente não faz a menor ideia de que o objeto está em outro local.

Essas duas classes são geradas por um programa fornecido como parte do JDK (Java Developers Kit), o `rmic`.

Como primeira etapa para a criação de uma classe acessada via RMI, precisa-se de interfaces para o objeto remoto. Essa classe deve possuir a interface pública e todos os seus métodos têm de lançar uma `RemoteException()`. Além disso, a interface da classe definida deve ser estendida da classe `Remote()`, parte da arquitetura RMI. Criadas e implementadas as Interfaces, deve-se compilar os códigos e gerar os arquivos.class. Estes serão utilizados pelo utilitário `RMIC` para a criação de classes `Skeleton` e `Stub`.

! O compilador utiliza-se dos arquivos compilados e, por isso, eles são necessários.

Cada objeto a ser compartilhado deve passar por um processo especial no servidor, onde deverá ser registrado e receber um nome. O programa que gerencia todo o registro e acesso de nomes no servidor é o `rmiregistry`. Ele deve estar em execução (modo stand by) quando o servidor tentar registrar o objeto, bem como quando o objeto for acessado. Se algum objeto for solicitado ao servidor, ele cria, executa, registra e termina a execução desse objeto instanciado através de uma chamada feita pelo cliente. Este, por sua vez, deve seguir a convenção de uma URL (Uniform Resource Locator), onde o protocolo RMI deve fornecer um servidor e o nome do método a ser executado remotamente pelo cliente no servidor.

A seguir serão apresentados cada um dos passos necessários para construir uma aplicação RMI (considerando que o sistema RMI está corretamente configurado em sua máquina).

Implementação de uma Aplicação RMI

Deve ser realizada seguindo os seguintes passos:

- 1º: Escrever e compilar o código java para as interfaces.
- 2º: Escrever e compilar o código java para as classes de implementação.
- 3º: Gerar os arquivos com as classes stubs e skeletons geradas a partir das classes de implementação.
- 4º: Escrever o código java para o serviço remoto do programa host.
- 5º: Desenvolver o código java para o programa cliente RMI.



Passo 1: Escrever e compilar o código java para as interfaces

Esse primeiro passo tem como objetivo escrever e compilar o código java para o serviço de interface. A classe de interface `Calculadora` define todas as características remotas oferecidas por esse serviço.

```
public interface Calculadora extends java.rmi.Remote {  
  
    public long add(long a, long b) throws java.rmi.RemoteException;  
    public long sub(long a, long b) throws java.rmi.RemoteException;  
    public long mul(long a, long b) throws java.rmi.RemoteException;  
    public long div(long a, long b) throws java.rmi.RemoteException;  
  
}
```

Passo 2: Escrever e compilar o código java para as classes de implementação

Esse segundo passo determina quais serão as instruções a serem executadas remotamente, quando o cliente solicitar um serviço para o servidor. Os métodos aqui implementados são herdados da classe de interface implementada anteriormente (Calculadora).

```
public class CalculadoraImp extends java.rmi.server.UnicastRemoteObject
implements Calculadora {

    // Implementations must have an
    //explicit constructor
    // in order to declare the
    //RemoteException exception

    public CalculadoraImp() throws java.rmi.RemoteException {
        super();
    }

    public long add(long a, long b) throws java.rmi.RemoteException {
        long resp = a + b;
        System.out.println("\nResultado da Adição de "+a+" + "+b+" eh "+resp);
        return a + b;
    }

    public long sub(long a, long b) throws java.rmi.RemoteException {
        long resp = a - b;
        System.out.println("\nResultado da Subtração de "+a+": "+b+" e "+resp);
        return a - b;
    }

    public long mul(long a, long b) throws java.rmi.RemoteException {
        long resp = a * b;
        System.out.println("\nResultado da Multiplicação de "
                           + a + " * " + b + " eh " + resp);

        return a * b;
    }

    public long div(long a, long b) throws java.rmi.RemoteException {
        long resp = a / b;
        System.out.println("\nResultado da Divisão de "
                           + a + " / " + b + " eh " + resp);

        return a / b;
    }
}
```

Passo 3: Gerar os arquivos com as classes stubs e skeletons geradas a partir das classes de implementação

Nesse ponto serão gerados os arquivos com as classes stubs e skeletons. Isso é possível através do uso do compilador RMI, o `rmic`. Depois de executar o compilador RMI serão criados os arquivos `<classname>_Stub.class` e `<classname>_Skel.class`, onde `<classname>` corresponde ao nome da classe de interface. A instrução para a geração dessas classes é: `rmic CalculadoraImp`. A compilação é feita na linha de comando e não deve ser realizada no diretório onde se encontra o arquivo `.class`. Sendo assim, retorne um nível na estrutura de diretório e execute o comando `rmic calculadora.CalculadoraImp`, onde `calculadora` indica o pacote onde a implementação da classe de interface se localiza.

A figura a seguir mostra os arquivos já criados no diretório `calculadora`.

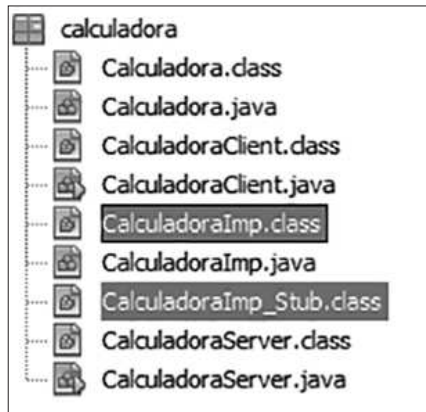


Figura 2.4
Arquivos com as
classes stubs e
skeletons.

Passo 4: Escrever o código java para o serviço remoto do programa host

Serviços remotos via RMI devem ser hospedados em um servidor. A seguir, o código que implementa o serviço remoto.

```
import java.rmi.Naming;

public class CalculatorServer {

    public CalculatorServer() {
        try {
            Calculator c = new CalculatorImpl();
            Naming.rebind("rmi://localhost/CalculatorService", c);
            // CalculadoraServer pronta!
        } catch (Exception e) {
            System.out.println("Trouble: " + e);
        }
    }

    public static void main(String args[]) {
        new CalculatorServer();
    }
}
```

Cabe destacar que a classe Naming fornece métodos para armazenar e obter referências a objetos remotos, em um registro remoto do objeto. Já rebind(name, obj) religa o nome especificado para um novo objeto remoto. Qualquer vinculação existente para o nome é substituída.

Passo 5: Desenvolver o código java para o programa cliente RMI

No trecho de código a seguir, podemos ver a interface da aplicação que será acessada pelo cliente.

```
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.net.MalformedURLException;
import java.rmi.NotBoundException;

public class CalculatorClient {

    public static void main(String[] args) {
        try {
            Calculator c = (Calculator)
                Naming.lookup("rmi://localhost/CalculatorService");
            System.out.println( c.sub(4, 3) );
            System.out.println( c.add(4, 5) );
            System.out.println( c.mul(3, 6) );
            System.out.println( c.div(9, 3) );
        }
        catch (MalformedURLException murl) {
            System.out.println();
            System.out.println("MalformedURLException");
            System.out.println(murl);
        }
        catch (RemoteException re) {
            System.out.println();
            System.out.println("RemoteException");
            System.out.println(re);
        }
        catch (NotBoundException nbe) {
            System.out.println();
            System.out.println("NotBoundException");
            System.out.println(nbe);
        }
    }
}
```

O método lookup(name) retorna uma referência (ao stub) para o objeto remoto associado ao nome especificado. Ele liga o stub ao skeleton, sendo necessário realizar downcast. Também libera execução de métodos remotos.

Executando uma aplicação RMI

Agora que todos os arquivos do projeto foram criados e devidamente compilados, estamos prontos para rodar o sistema! Você precisará abrir três diferentes consoles PROMPT-DOS no seu Windows, ou outro, caso utilize um Sistema Operacional diferente.

3 Consoles:

- O RMI Registry.
- O programa servidor java <classe_servidor>.
- O programa cliente java <classe_cliente>.

Em um dos consoles vai rodar o programa servidor; no outro o cliente e no terceiro o RMI Registry. Inicie com o RMI Registry. Você deve estar no mesmo diretório em que estão gravados seus arquivos para rodar o aplicativo. Execute a seguinte linha de comando:

```
start rmiregistry
```

Isso vai iniciar o RMI Registry e rodá-lo. O resultado da execução desse comando pode ser visto na figura 2.5, a seguir.



Figura 2.5
rmiregistry
em execução.

No segundo console, vamos executar o programa servidor. Você deve estar no mesmo diretório em que estão gravados seus arquivos para rodar o aplicativo. Execute o seguinte comando:

```
start java calculadora.CalculadoraServer
```

Isso vai iniciar, carregar a implementação na memória e esperar pela conexão cliente. Veja o resultado na figura 2.6.



Figura 2.6
Servidor da calculadora em execução.

Finalmente, no último console, rode o programa cliente. Você deve estar no mesmo diretório em que estão gravados seus arquivos para rodar o aplicativo. Excute o comando:

```
java calculadora.CalculadoraClient
```

Imediatamente, você terá os resultados dos cálculos sendo visualizados no console do cliente, bem como mensagens dando conta da realização dos cálculos que estão sendo realizados remotamente no servidor, conforme a figura 2.7.

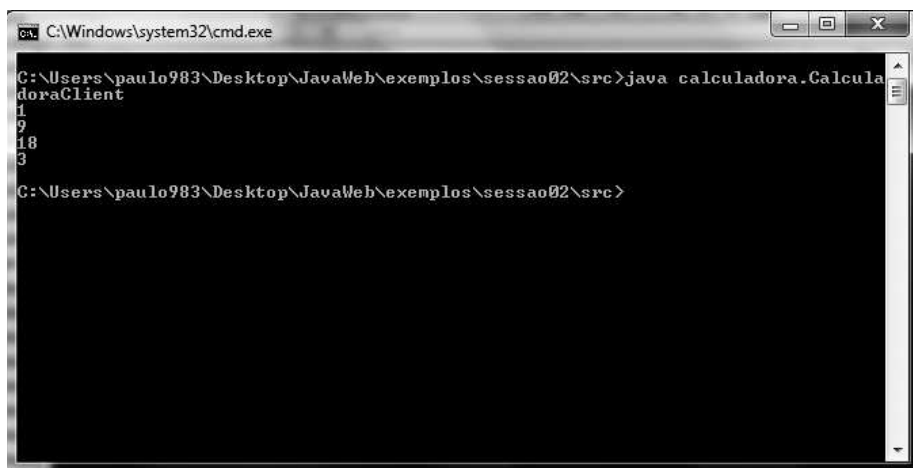


Figura 2.7
Interface cliente executando método remoto.

Os passos descritos nos possibilitam criar um sistema utilizando a tecnologia RMI. Apesar de você ter rodado os programas na mesma máquina, o RMI usa a pilha de rede TCP/IP para se comunicar entre as três diferentes instâncias da JVM.

Exercícios de fixação

RMI

Quantos e quais são os passos para implementação de uma aplicação Java RMI?

Atividades Práticas

3

Applets

objetivos

Apresentar aos alunos o processo de desenvolvimento de aplicações com Applets Java que utilizam elementos de interface gráfica para a interação com o usuário final, através de aplicações que executam remotamente por meio de navegadores internet.

conceitos

Applet; java-enabled; javapowered; ciclo de vida do applet; tag <applet> e tag <object>.

Exercício de nivelamento

Applets

Você já desenvolveu aplicações para internet? Que recurso e/ou linguagem de programação você já utilizou para desenvolvimento?

Introdução

Applet:

- Tipo especial de programa Java.
 - ▣ Browser com suporte à tecnologia Java é dito Java-enabled.
 - ▣ Pode ser baixado pela internet.
 - ▣ Appletviewer: aplicativo do Java para testar a execução de applets.
- Estende as funcionalidades do browser.
 - ▣ Adiciona som, animação etc.
 - ▣ Proveniente de URLs locais ou remotas.
 - ▣ Cada página html pode conter uma ou mais applets.
 - ▣ A cada mudança de página, os applets associados são automaticamente destruídos.
 - ▣ Uma página HTML contendo applets é dita Javapowered.

Os applets são aplicações Java que utilizam recursos da máquina virtual Java dentro de navegadores web com algumas limitações de segurança. Quando um usuário visualiza uma página HTML que contém um applet, o código do applet é baixado para a máquina do usuário.

Um applet é um programa em Java que é executado em um browser. Um applet pode ser uma aplicação Java totalmente funcional, pois tem toda a API Java à sua disposição.



Normalmente embutido em uma página web.

É executado no contexto do browser.

Subclasse: java.applet.Applet ou javax.swing.JApplet.

Applet ou JApplet fornece a interface padrão (entre o applet e browser).

- Apresentam GUIs (AWT ou Swing).

Plug-in Java.

- Instalado no browser.
- Gerencia o ciclo de vida de um applet.
- JVM que interpreta o bytecode.

O desenvolvimento ou não de programas em Java sob a forma de applets depende do objetivo da aplicação. Sua execução necessita de um navegador e pode ser disponibilizada para execução via internet.

Applets sempre executam nos clientes web, nunca nos servidores. É importante considerar que a carga das classes pode levar algum tempo.

Existem algumas diferenças importantes entre um applet e uma aplicação Java standalone, conforme é mostrado na tabela a seguir:

Aplicações	Applets
<ul style="list-style-type: none">▪ Programa autônomo.▪ Executa em uma JVM autônoma.▪ Ponto de entrada é o método main(), chamado pelo carregador de classes.▪ Geralmente contém construtores.▪ Acesso a todos os recursos do ambiente.▪ Pode ser gráfica.▪ Funcionalidades plenas: menus, diálogos, arquivos...▪ Janela base da aplicação estende de Frame ou JFrame.	<ul style="list-style-type: none">▪ Executada por um navegador.▪ O navegador fornece a JVM.▪ Vários pontos de entrada são chamados pelo navegador como init().▪ Construtores tendem a ser vazios.▪ Acesso limitado por razões de segurança.▪ São sempre gráficas.▪ Devem fornecer uma classe que estenda applet.▪ Janela base da aplicação estende de applet ou JApplet.▪ Carregador de classes remotas.

Tabela 3.1
Diferenças entre aplicações Java e applets.

Uma vez que um applet é executado a partir de um browser web, ele tem a vantagem de já ter uma janela e a capacidade de responder aos eventos da interface com o usuário, por meio do browser. Também pelo fato de os applets serem projetados para uso em redes, Java é muito mais restritivo nos tipos de acesso que os applets poderão ter no seu sistema de arquivos do que com os aplicativos que não estão necessariamente rodando em rede.

Ciclo de vida de um applet

Todo applet passa por diferentes estágios, desde a sua criação até o fim das suas atividades, estágios esses que podem ser visualizados na figura 3.1.

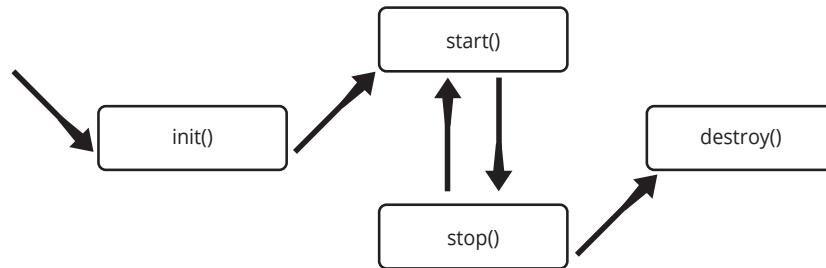


Figura 3.1
Ciclo de Vida
de um applet.

Na verdade, cada um desses estágios pelo qual passa um applet corresponde a um método da classe applet, que disponibiliza o framework necessário a partir do qual podemos construir o código Java que é executado em um browser. São eles:

init(): chamado pelo browser quando o applet é carregado.

- Responsável pela inicialização do applet.
- Faz processamento de parâmetros.
- Criação e adição de componentes de interface.

start(): chamado pelo browser para informar que o applet deve iniciar sua execução.

- Após a chamada ao método init().
- Executado toda vez que o usuário retorna à página.

stop(): chamado pelo browser para informar que o applet deve interromper sua execução.

- Navegação entre páginas.
- Antes da destruição de um applet.
- Atividades que foram interrompidas com stop() são retomadas com start().

destroy(): chamado pelo browser quando do seu encerramento.

- Libera recursos alocados para o Applet (via garbage collector).
- Fechamento da página HTML que contém o applet.

Um applet vai trabalhar de forma coordenada com o HTML (a linguagem de marcação de hipertexto) e que vai descrever o layout de uma página da web. O HTML é simplesmente um veículo para indicar elementos de uma página de hipertexto. Por exemplo, <TITLE> indica o título da página e todo texto após essa tag se torna o título da página. Você indica o final do título com a tag </TITLE>.

Deve-se mencionar ainda que o método paint(), do java.awt, deve ser invocado logo após o método start(), e sempre que o applet precisar se redesenhar no próprio navegador.

Um primeiro exemplo de applet

A seguir apresentamos um código Java simples que implementa um applet que vai escrever a mensagem “Primeiro exemplo de Applet Java” no browser quando a página HTML que ele estiver inserido for acessada.

Para criar um applet:

- 1º: cria-se uma sub-classe de Applet.
- 2º: redefine-se certos métodos:
 - ▣ `init()`, `start()`, `stop()`, `destroy()`, `paint()` e `update()`

A seguir, podemos ver o código desse primeiro exemplo:

```
import java.applet.*;
import java.awt.*;
public class HelloWorldApplet extends Applet
{
    public void paint (Graphics g)
    {
        g.drawString ("Primeiro exemplo de Applet Java ", 25, 50);
    }
}
```

As declarações de importação trazem as classes para o âmbito da nossa classe applet. Sem essas declarações de importação, o compilador Java não reconheceria a classe applet, e tampouco os objetos visuais gráficos que a classe do applet referencia. Vamos descrever em seguida a classe applet com mais detalhes. Para ajudar nesse detalhamento, apresentamos na figura 3.2 a hierarquia de classes em um applet.

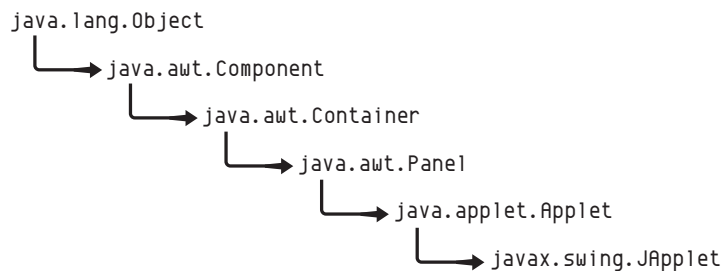


Figura 3.2
Hierarquia
de classes em
um applet.

A classe applet

Cada applet é uma extensão da classe `java.applet.Applet` ou da classe `javax.swing.applet.JApplet`. A classe `Applet` base fornece métodos que uma classe `Applet` derivada pode chamar para obter informações e serviços a partir do contexto do browser.

Métodos da Classe `Applet` permitem:

- Iniciar, controlar ou terminar a execução de uma aplicação em um browser.
- Obter a localização de rede do arquivo HTML que contém o applet e do diretório da classe do applet.
- Obter parâmetros de applet.
- Imprimir uma mensagem de status no navegador.
- Buscar e executar componentes de som e imagem.
- Redimensionar o applet.

Além disso, a classe applet fornece uma interface pela qual o usuário ou o navegador obtém informações sobre o applet e controla sua execução.

A interface permite:

- Pedir informação sobre o autor, a versão e o copyright do applet.
- Solicitar uma descrição dos parâmetros que o applet reconhece.
- Inicializar o applet.
- Começar ou interromper a execução do applet.
- Destruir o applet.

A classe applet fornece implementações padrão de cada um desses métodos. Essas implementações podem ser substituídas quando necessário. Veja que o applet apresentado como um primeiro exemplo em 3.2 está completo, tal como está. O único método nele substituído é o método de pintura `paint()`.

O pacote `java.applet` fornece as classes para criar applets e permitir que applets se comuniquem com seu contexto.

Contexto de um applet:

- Uma aplicação que é responsável por carregar e executar applets.
 - Exemplo: navegadores e “`appletviewer`”.
- O código HTML que a contém e as demais applets contidas no mesmo documento.

Executando um applet

Fazendo novamente referência ao ciclo de vida de um applet exibido na figura 3.1, não custa reforçar cada uma das etapas observadas no seu processo de execução, listadas a seguir.

Acesso à página que contém um applet:

- Carrega o applet e executa o método `init()`.

Applet carregada e o browser pode chamar:

- O browser pode chamar `start()`, `paint()`, `stop()`.

O browser sai da página atual:

- Ocasiona a chamada de `stop()` ou `destroy()`.

A estrutura geral de um applet pode ser vista no trecho de código a seguir.

```
package visao;
import java.applet.Applet;
import java.awt.Graphics;
public class ExemploApplet extends Applet {
    @Override
    public void init( ){
        // Inicializa a applet
    }
    @Override
    public void start( ){
        // Dispara a execução de uma applet
    }
    @Override
```

```

    public void stop( ){
        // Para a execução de uma applet
    }
    @Override
    public void destroy( ){
        // Destrói definitivamente uma applet
    }

    @Override
    public void paint(Graphics g){
        //...
    }
} // fim da classe

```

Um segundo exemplo de código de um applet, que corresponde ao arquivo “ExemploAppletAWT.java” no diretório de exemplos desta sessão, pode ser visto a seguir:

```

package visao;
import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.Label;
public class ExemploAppletAWT extends Applet{
    //Executado quando o applet for carregado no browser.
    @Override
    public void init() {
        setSize(250, 100);
        Label lbl = new Label("Exemplo java.applet.Applet",Label.CENTER);

        setLayout(new BorderLayout());
        add(lbl,BorderLayout.CENTER);
    }
}

```

O que não mostramos ainda é como fazer para associar um applet a uma página web. A ideia é muito simples! É preciso fazer com que a página HTML diga ao navegador quais applets devem ser carregados e, então, onde colocar cada applet na página da web.

Logo que os applets foram “inventados”, era utilizada uma tag especial capaz de dizer ao navegador exatamente isso: onde estão os arquivos de classe e como o applet se posiciona na página da web (tamanho, localização etc.).

```
<APPLET code="visao.ExemploAppletAWT.class" width=350 height=200></APPLET>
```

A tag <applet> é a base para a incorporação de um applet em um arquivo HTML. É necessário utilizar o atributo code (código) do tag <applet>. Ele especifica a classe applet a ser executada. Também é necessário indicar a largura (width) e a altura (height) de modo a especificar o tamanho inicial do painel em que o applet é executado. A diretiva applet deve ser fechada com a tag </applet>.

A seguir, mostramos o código HTML do arquivo “ExemploAppletAWT.html”, também disponível no diretório de exemplos desta sessão.

```

<HTML>
<HEAD>
  <TITLE>Applet HTML Page</TITLE>
</HEAD>
<BODY>
  <H3><HR WIDTH="100%">Applet HTML Page<HR WIDTH="100%"></H3>
  <P>
    <APPLET code="visao.ExemploAppletAWT.class" width=350 height=200></APPLET>
  </P>
  <HR WIDTH="100%"><FONT SIZE=-1><I>Generated by NetBeans IDE</I></FONT>
</BODY>
</HTML>

```

Outros atributos que podem ser usados com a tag <applet> podem ser vistos na figura 3.3.

```

<APPLET
  <!-- Obrigatórios -->
  CODE = appletFile      <!-- Fornece o nome da classe que contém a applet -->
  WIDTH = pixels         <!-- Fixa a largura inicial da applet em pixels -->
  HEIGHT = pixels        <!-- Fixa a altura inicial da applet em pixels -->

  <!-- Opcionais -->
  [ CODEBASE = codebaseURL ]    <!-- Define a base URL da classe do applet -->
  [ ALT = alternateText ]      <!-- Permite exibir uma mensagem para o browser -->
  [ NAME = appletInstanceName ] <!-- Fornece a instância de uma applet em
                                uma página HTML, de modo a estabelecer uma
                                comunicação entre applets -->
  [ ALIGN = alignment ]        <!-- Posicionamento de uma applet em uma página HTML. Os
                                valores permitidos são: left, right, top, texttop, middle,
                                absmiddle, baseline, bottom, absbottom. -->
  [ VSPACE = pixels ]          <!-- Número de pixels (vertical) entre a applet e o resto
                                da página HTML -->
  [ HSPACE = pixels ]          <!-- Número de pixels (horizontal) entre a applet e o resto
                                da página HTML -->
>
  <!-- Único modo de passar argumentos a uma applet -->
  [ < PARAM NAME = appletParameter1 VALUE = value >]
  [ < PARAM NAME = appletParameter2 VALUE = value >]
</APPLET>

```

Figura 3.3
Atributos que
podem ser usados
com a tag <applet>.

Convém lembrar que o navegador procura o código do applet na mesma URL do arquivo HTML. Para especificar um local diferente, é preciso usar o atributo CODEBASE, conforme mostrado:

```

<applet codebase="http://amrood.com/applets"
code="HelloWorldApplet.class" width="320" height="120">

```

Se um applet for implementado em um pacote diferente do padrão, o caminho para esse pacote deverá ser especificado no atributo CODE, usando o caractere ponto (.) para separar os componentes do pacote/classe. Por exemplo:

```

<applet code="mypackage.subpackage.TestApplet.class"
width="320" height="120">

```

Navegadores não habilitados para Java não processam as tags `<applet>` e `</applet>`. Portanto, qualquer coisa que aparece entre as tags, não relacionada com o applet, é visível em navegadores não habilitados para Java.



Convém esclarecer também que houve um movimento para unificar a forma de referenciar dados externos a uma página HTML através da tag `OBJECT`. Assim, o W3C chegou a recomendar que não fosse mais utilizada a tag `APPLET`, mas os browsers mais antigos habilitados para Java não reconheciam a nova tag `OBJECT`. Os navegadores mais novos reconhecem as duas tags, o que acabou contribuindo para a tag `APPLET` nunca ter realmente saído de uso, razão pela qual continuaremos a usá-la neste curso. O HTML5 reforçou essa tendência ao adotar novas tags `AUDIO` e `VIDEO` e mantendo o uso de `IMG` (tags essas que poderiam ser substituídas por `OBJECT`).

Finalmente, além do applet em si, a página da web pode conter todos os outros elementos HTML disponíveis: múltiplas fontes, listas com bullets, elementos gráficos, links etc. Os applets são apenas uma parte da página de hipertexto.

É sempre interessante lembrar que a linguagem de programação Java não é uma ferramenta para projetar páginas HTML; ela é uma ferramenta para dar vida a elas. Isso não quer dizer que os elementos de projeto de GUI em um applet Java não são importantes, mas que eles devem trabalhar com (e, na verdade, são subservientes a) o projeto HTML subjacente da página da web.

Uso de parâmetros em applets

Veja que se um applet precisar usar parâmetros, estes podem ser passados através do uso do atributo `PARAM` na página HTML. No exemplo a seguir é passado o parâmetro “número” contendo o valor “10” para o applet “VerificarParImparApplet.class”.

```
<HTML>
<HEAD>
  <TITLE>Applet HTML Page</TITLE>
</HEAD>
<BODY>
<APPLET codebase="classes" code="visao/VerificarParImparApplet.class" width=350
height=200>
  <PARAM name="numero" value="10">
</APPLET>
</BODY>
</HTML>
```

Os parâmetros especificados em uma página HTML serão repassados ao applet que poderão tratá-los para executar alguma funcionalidade desejada. Durante a escrita do código Java no applet, o programador deverá se preocupar com a conversão dos parâmetros, que são Strings, para o tipo necessário para utilização dessa informação.

O código Java do applet precisa estar preparado para receber e trabalhar com parâmetros. Isso pode ser feito no método `init()` ou no método `paint()`. No entanto, é melhor receber os valores e prepará-los para o uso logo no início, em vez de fazer isso a cada atualização ao applet. Não apenas é mais conveniente, como também mais eficiente.

Veremos a seguir como o método `Applet.getParameter()` é usado para receber um parâmetro, que chega sempre como um `string`.

```
package visao;
import java.awt.BorderLayout;
import javax.swing.JApplet;
import javax.swing.JLabel;
public class VerificarParImparApplet extends JApplet {

    //Executado quando o applet for carregado no browser.
    @Override
    public void init() {
        setSize(250, 100);
        JLabel lbl =
            new JLabel("Exemplo com passagem de parâmetros",JLabel.CENTER);
        setLayout(new BorderLayout());
        add(lbl, BorderLayout.CENTER);

        int num = Integer.parseInt(getParameter("numero"));

        if(num%2 == 0)
            javax.swing.JOptionPane.showMessageDialog(rootPane, num+"
            é um número par!", "Mensagem",
            javax.swing.JOptionPane.INFORMATION_MESSAGE);
        else
            javax.swing.JOptionPane.showMessageDialog(rootPane, num+"
            é um número ímpar!", "Mensagem",
            javax.swing.JOptionPane.INFORMATION_MESSAGE);
    }
}
```

Veja que `getParameter()` indica o mesmo nome do parâmetro definido na tag `<PARAM>`, presente na chamada `<APPLET>` na página html. O método `Integer.parseInt` (Classe `Wrapper`) converte o `string` e retorna um número inteiro.

Na sequência, o applet realiza um teste de condição e chama o método `showMessageDialog()` de `javax.swing.JOptionPane` para exibir uma mensagem informando se o número é par ou ímpar. Note que estamos utilizando os mesmos componentes visuais gráficos utilizados para desenvolver aplicativos desktop em Java.

Conversão de aplicação em applet

É fácil converter um aplicativo Java gráfico (isto é, um aplicativo que usa o AWT ou SWING) em um applet que você pode inserir em uma página web. Basta seguir os passos a seguir:

- Criar uma página HTML com a tag `<APPLET>`/`</APPLET>` apropriada para carregar o código Java.
- Fornecer uma subclasse da classe `JApplet`. Essa subclasse deve ser declarada como pública (senão, o applet não poderá ser carregado).
- Eliminar o método principal na aplicação e também a construção do seu `Frame` (já que a aplicação será exibida dentro do navegador).





- Mover qualquer código de inicialização do construtor para o método `init()` do applet. Não é preciso construir explicitamente uma instância do objeto para o navegador. O applet vai instanciá-lo automaticamente através da chamada ao método `init()` quando o applet for carregado pela página HTML contendo a tag `<APPLET>` correspondente.
- Retirar a chamada para `setSize()`. O dimensionamento dos applets é feito com a especificação da largura e altura como atributos da tag `<APPLET>` no arquivo HTML.
- Retirar a chamada para `setDefaultCloseOperation()`. Um applet não pode ser fechado. Sua execução termina quando o usuário navega para uma próxima página ou fecha o navegador.
- Eliminar a chamada para o método `setTitle()`. Um applet não pode ter barras de título (mas pode utilizar o título da própria página web).
- Eliminar a chamada para o método `setVisible(true)`. O applet é exibido automaticamente durante a carregamento da página HTML.

Manipulação de eventos no applet

Applets herdam um grupo de métodos da classe `Container` de manipulação de eventos. A classe `Container` define vários métodos, tais como `processKeyEvent` `processMouseEvent`, bem como métodos para o tratamento de tipos particulares de acontecimentos.

Para possibilitar o controle de eventos, um applet deve implementar a classe de interface específica do evento apropriado e reescrevê-lo para atender as particularidades da aplicação em desenvolvimento.

O código a seguir implementa a interface `MouseListener` para controlar eventos do mouse no applet:

```
package visao;
import java.awt.Graphics;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import javax.swing.JApplet;
public class ExemploControleEventoMouseApplet extends JApplet implements
MouseListener {
    StringBuffer strBuffer;
    public void init() {
        addMouseListener(this);
        strBuffer = new StringBuffer();
        addItem("initializing the apple ");
    }
    public void start() {
        addItem("starting the applet ");
    }
    public void stop() {
        addItem("stopping the applet ");
    }
    public void destroy() {
        addItem("unloading the applet");
    }
    void addItem(String word) {
```

```

        System.out.println(word);
        strBuffer.append(word);
        repaint();
    }
    public void paint(Graphics g) {
        //Draw a Rectangle around the applet's display area.
        g.drawRect(0, 0,
            getWidth() - 1,
            getHeight() - 1);
        //display the string inside the rectangle.
        g.drawString(strBuffer.toString(), 10, 20);
    }

    @Override
    public void mouseClicked(MouseEvent e) {
        addItem("mouse clicked! ");
    }
    @Override
    public void mousePressed(MouseEvent e) {
    }
    @Override
    public void mouseReleased(MouseEvent e) {
    }
    @Override
    public void mouseEntered(MouseEvent e) {
        addItem("mouse entered! ");
    }
    @Override
    public void mouseExited(MouseEvent e) {
        addItem("mouse exited! ");
    }
}

```

Outros métodos disponíveis para um applet

Um método que pode ser bastante útil é o `resize()`, que como o nome indica, permite redefinir o tamanho do applet na janela do navegador.

Muitos applets fazem uso do método `getImage()`, que retorna um objeto `Image` que pode ser exibido na tela pelo método `paint`. Podem ser exibidas imagens nos formatos GIF, JPEG, BMP e outros. Mas para exibir uma imagem é preciso usar o método `drawImage()` encontrado na classe `java.awt.Graphics`. O exemplo a seguir implementa um applet com todos os passos necessários para visualizar imagens:

```

package visao;
import java.applet.AppletContext;
import java.awt.Graphics;
import java.awt.Image;
import java.net.MalformedURLException;
import java.net.URL;
import javax.swing.JApplet;
public class ExemploImgApplet extends JApplet {
    private Image image;
    private AppletContext context;
    public void init() {
        context = this.getAppletContext();
        String imageURL = this.getParameter("image");
        if(imageURL == null) {
            imageURL = "alert.gif";
        }
        try {
            URL url = new URL(this.getDocumentBase(), imageURL);
            System.out.println(url);
            image = context.getImage(url);
        } catch(MalformedURLException e) {
            e.printStackTrace();
            // Display in browser status bar
            context.showStatus("Could not load image!");
        }
    }

    public void paint(Graphics g) {
        context.showStatus("Displaying image");
        g.drawImage(image, this.getWidth()/2,
                    this.getHeight()/2, 33, 34, null);
        repaint();
    }
}

```

Um applet pode também reproduzir um arquivo de áudio através da interface AudioClip no pacote java.applet. A interface AudioClip tem três métodos:

- **public void play():** reproduz o clipe de áudio uma vez, desde o início.
- **public void loop():** faz com que o clipe de áudio seja reproduzido continuamente.
- **public void stop():** para a reprodução do clipe de áudio.

Para obter um objeto AudioClip, você deve chamar o método da classe Applet `getAudioClip()`. O método `getAudioClip()` retorna imediatamente, ou não resolve a URL para um arquivo de áudio real. O arquivo de áudio não é baixado até que seja feita uma tentativa de reproduzir o clipe de áudio.

O mecanismo de som que reproduz os clipes de áudio suporta vários formatos de arquivo de áudio, incluindo:



- Formato de arquivo Sun Audio (.au).
- Windows wave (.wav).
- Macintosh AIFF (.aif ou .AIFF).
- Musical Instrument Digital Interface – MIDI (.mid ou .rmi).

Segue-se o exemplo mostrando todos os passos para ter um áudio:

```
package visao;
import java.applet.AppletContext;
import java.applet.AudioClip;
import java.net.MalformedURLException;
import java.net.URL;
import javax.swing.JApplet;
public class ExemploAudioApplet extends JApplet {

    private AudioClip clip;
    private AppletContext context;
    public void init() {
        context = this.getAppletContext();
        String audioURL = this.getParameter("audio");
        if(audioURL == null) {
            audioURL = "teste.wav";
        }
        try {
            URL url = new URL(this.getDocumentBase(), audioURL);
            clip = context.getAudioClip(url);
        } catch(MalformedURLException e) {
            e.printStackTrace();
            context.showStatus("Could not load audio file!");
        }
    }

    public void start() {
        if(clip != null) {
            clip.loop();
        }
    }

    public void stop() {
        if(clip != null) {
            clip.stop();
        }
    }
}
```

Existem também métodos para obter informações relacionadas com o applet, como por exemplo, `getAppletInfo()` que retorna um string com informações sobre este. Outro método é o `getParameterInfo()`, que retorna um array de string com informações sobre os parâmetros recebidos applet.

Restrições de segurança

De modo a garantir a segurança no uso de aplicações remotas, existem uma série de restrições em relação ao que um applet Java pode ou não fazer em uma máquina cliente. Por default, uma applet não pode:

- Acessar arquivos na máquina local.
- Chamar outro programa na máquina local.
- Comunicar-se com nenhuma outra máquina, a não ser a que contém a página HTML em questão (servidor).



Essas limitações podem ser ocasionalmente autorizadas por meio de um arquivo especial (policy file), através do qual é definido o que é permitido para um código-fonte particular. Acesse o AVA e veja os links para esta sessão.

Atividades Práticas  

4

Servlets

objetivos

Apresentar aos alunos o processo de desenvolvimento de aplicações web com o uso de Servlets, tecnologia disponibilizada para o processo de desenvolvimento de aplicações Java que utilizam a internet como base para seu funcionamento.

Servlets; ServletRequest; ServletResponse; anotação @WebServlet; métodos GET e POST.

conceitos

Definição

É um componente da web, administrado por um programa que gera um conteúdo de página web dinâmica.

Possui classes da linguagem de programação Java independentes de plataforma.

- bytecode que pode ser dinamicamente carregado em um servidor web.

Interage com clientes web.

- Modelo pedido-resposta (request-response).
- Baseado no comportamento do protocolo de comunicação HTTP.

Os servlets são programas em Java usados para estender as funcionalidades de um servidor web. Servlets são pequenos e possuem classes da linguagem de programação Java independentes de plataforma, que, quando compilados, geram o byte-code que pode ser dinamicamente carregado em um servidor web.



Servlets são para o servidor, enquanto que applets são para o cliente.



Servlets interagem com clientes da web por um pedido-resposta implementado no programa Servlet. Esse modelo de pedido-resposta é baseado no comportamento do protocolo de comunicação HTTP (Hypertext Transfer Protocol), como ilustrado na figura 4.1.

Um Servlet define um conjunto de métodos, sem interface gráfica para usuário, que trabalha juntamente com o Servlet Engine, ambiente escrito por uma distribuição de um servidor web de acordo com a especificação Java Servlet API, que roda no servidor web, e que por sua vez trata das requisições e respostas aos clientes.

O programa cliente é qualquer programa (escrito em Java), capaz de fazer conexões e requisições ao servidor web. Essa requisição é processada pela Servlet. Depois de feita a requisição do cliente para Servlet Engine, ela retorna uma resposta ao Servlet. Após isso, o Servlet envia uma resposta dentro do protocolo HTTP para o cliente.



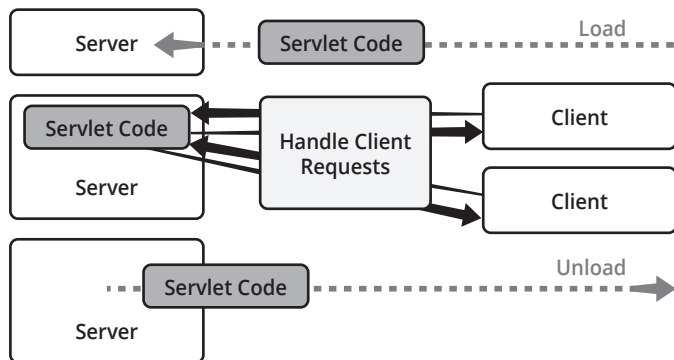


Figura 4.1
Estrutura de execução de um Servlet.

Servlet é controlado pelo container WEB onde foi implementado.

- Quando uma requisição é solicitada ao servlet, o container WEB executa os seguintes passos:
 - Se a instância do servlet não existir.
 - Carrega a classe do servlet.
 - Cria uma instância da classe do servlet.
 - Inicializa a instância do servlet chamando o método `init()`.
 - Chama o método `processRequest()`.
 - Passa como parâmetros os objetos `request` e `response`.
 - Funcionalmente, Servlets estão entre programas CGI e extensões servidoras proprietárias NSAPI. Entre as vantagens de usar Servlets, destacamos:
 - Fornecem uma maneira de gerar documentos dinâmicos que são fáceis de escrever e rápidos para executar.
 - São mais rápidos que scripts CGI.
 - Usam uma API padrão que é suportada por muitos servidores web.
 - Possuem todas as vantagens da linguagem Java, incluindo facilidade de desenvolvimento e independência de plataforma.
 - Podem ter acesso a um grande número de APIs disponíveis para a plataforma Java.

No código, o ciclo de vida de um Servlet é definido pela interface `javax.servlet.Servlet` (disponível após instalação do JSDK), sendo que todos os Servlets devem implementar direta ou indiretamente essa interface que roda no Servlet Engine.

Arquitetura Servlets

O Servlet Engine instancia e carrega um Servlet. A instanciação e carregamento geralmente acontecem quando a Servlet Engine inicia ou quando é necessário um Servlet para responder por uma requisição. Ele pode carregar um Servlet oriundo de um sistema de arquivos local ou remoto. A figura 4.2 apresenta os estágios de execução de um Servlet.

Quando um Servlet aceita a chamada de um cliente, ele recebe dois objetos como parâmetros:

- `ServletRequest`:
 - Encapsula as mensagens do cliente para o servidor.
- `ServletResponse`:
 - Encapsula as mensagens do servidor para o cliente.

Quando um Servlet aceita a chamada de um cliente, ele recebe dois objetos como parâmetros: o `ServletRequest` e o `ServletResponse`. O primeiro encapsula as mensagens do cliente para o servidor, e o segundo encapsula as mensagens do servidor para o cliente. Existem subclasses das classes já citadas que permitem a manipulação de dados mais específicos para determinados protocolos, como é o caso das classes `HttpServletRequest` e `HttpServletResponse`.

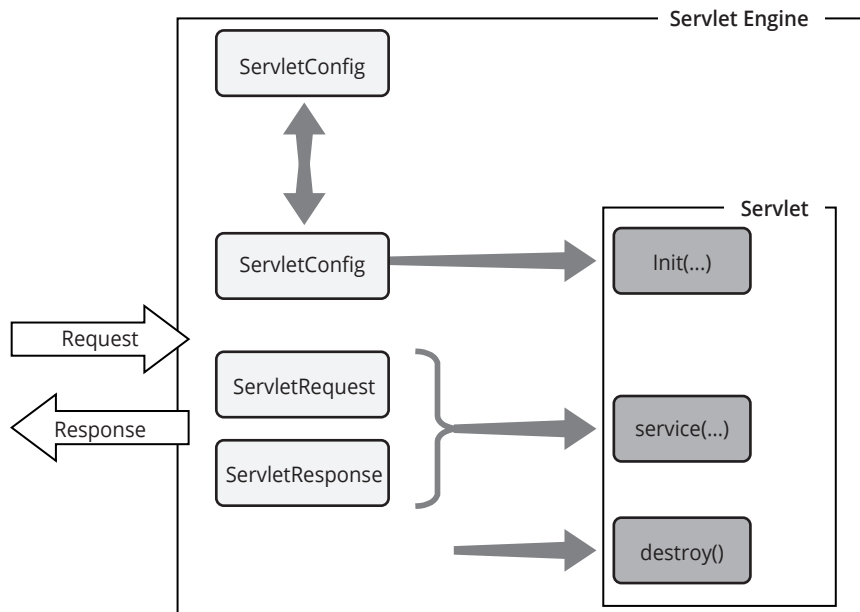


Figura 4.2
Estágios de execução de um Servlet.

A interface `ServletRequest` permite que o Servlet acesse informações como os nomes dos parâmetros enviados pelo cliente. Ainda é provido o `ServletInputStream`, através do qual o Servlet pode obter dados do cliente, utilizando os métodos POST e PUT do protocolo HTTP. Já a interface `ServletResponse` dá ao Servlet a capacidade de responder ao cliente. De forma análoga, é provido um objeto de saída `ServletOutputStream` e outro objeto, `Writer`, pelos quais os dados são respondidos.

Interface `ServletRequest`: permite que o Servlet acesse informações enviadas pelo cliente.

- Nomes e conteúdos de parâmetros enviados.
- O protocolo que está sendo usado.
- Nome do host remoto que fez a requisição, entre outras.

Interface `ServletResponse`: dá ao Servlet a capacidade de responder ao cliente.

- Permite que o Servlet especifique o tipo de dado que será enviado.
- Provê um objeto de saída `ServletOutputStream` e outro objeto, `Writer`, pelos quais os dados são respondidos.

Um exemplo de Servlet

O código Java a seguir implementa uma classe Servlet Java, que apresenta alguns métodos criados por default.

```

@WebServlet(name = "NewServlet", urlPatterns = "/NewServlet")
public class NewServlet extends HttpServlet {
    protected void processRequest(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {...}    @Override

```

```

protected void doGet(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {...}
@Override
protected void doPost(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
    processRequest(request, response);
}
@Override
public String getServletInfo(){
    return "Short description";
}
}

```

Cabe destacar os seguintes trechos do código:

@WebServlet: uma anotação que define um componente Servlet em uma aplicação web.

- Substitui a necessidade de referenciar o Servlet no descritor de implantação (web.xml)

processRequest(): método padrão para realizar a codificação da página de resposta ao cliente.

doGet(): processa requisição do cliente com método de submissão GET.

- Tamanho limitado de informação a ser submetida.
- Segurança comprometida > Informações podem ser visualizadas.

doPost(): processa requisição do cliente com método de submissão POST.

- Tamanho ilimitado de informação a ser submetida.
- Mais Seguro > Informações ocultas.

getServletInfo(): retorna uma string com uma breve descrição sobre o Servlet.

Cada método tem suas vantagens e desvantagens. GET tem um tamanho limitado para a informação que é submetida, tornando-se fácil de anexar na última URL do seu Java Servlet. Por sua vez, POST não tem nenhuma limitação no tamanho da informação enviada, sendo que a informação fica oculta a partir da URL.

Por exemplo, GET exibirá algo como:

```
http://localhost:8084/ExemploServlet/ServMsg?nome=Paulo&idade=40
```

Já POST não exibirá o nome e a idade no final da URL, gerando algo como:

```
http://localhost:8084/ExemploServlet/ServMsg
```

É obvio que GET pode impactar a segurança, uma vez que podemos claramente ver informação enviada para o Java Servlet. POST seria a melhor opção, mas às vezes é preferível usar GET para tornar o desenvolvimento mais fácil. O mais comum é utilizar o GET na criação de links dinâmicos para a navegação da aplicação web em desenvolvimento.



No trecho de código a seguir, temos a implementação de requisições tanto via GET como POST.

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

@WebServlet(name = "NewServlet", urlPatterns = {"/NewServlet"})
public class ExemploServlet01 extends HttpServlet {
    protected void processRequest(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            /* TODO output your page here. You may use following sample */
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet ExemploServlet 01</title>");
            out.println("</head>");
            out.println("</body>");
            out.println("</html>");
        } finally {
            out.close();
        }
    }

    /** Handles the HTTP <code>GET</code> method.
     */
    @Override
    protected void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        processRequest(request, response);
    }

    /** Handles the HTTP <code>POST</code> method.
     */
    @Override
    protected void doPost(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        processRequest(request, response);
    }

    /** Returns a short description of the servlet.
     */
    @Override
    public String getServletInfo() {
        return "Short description";
    } // </editor-fold>
}
```

Estrutura de uma aplicação web

Qualquer aplicação em Java deve ser organizada de forma que seus recursos possam ser facilmente identificados e utilizados durante o desenvolvimento e execução de um determinado projeto. A figura 4.3 ilustra a estrutura de diretórios criada para uma aplicação web através do netBeans.

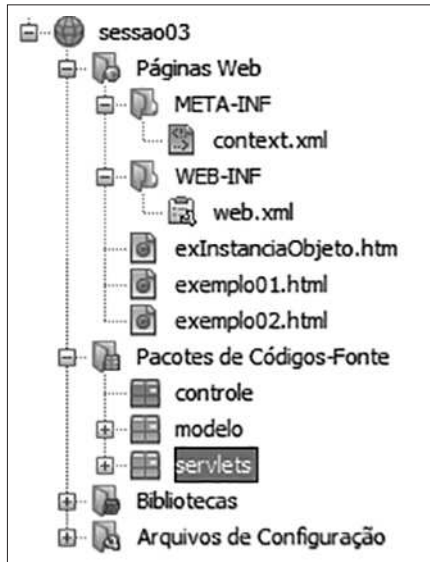


Figura 4.3
Estrutura de diretórios de uma aplicação web no netBeans.

A pasta Páginas Web é o diretório-raiz (document root) da aplicação em desenvolvimento. Na raiz dessa pasta deverá estar o arquivo de inicialização da aplicação quando essa for executada no browser. Esse arquivo é conhecido como “welcome file” e geralmente recebe o nome de index, sendo que no NetBeans ele também recebe a extensão.jsp por padrão.

Páginas Web:

- Contêm arquivos com a interface para o cliente.
- META-INF.
 - Armazena arquivos de configuração necessários para a execução dos recursos disponíveis na aplicação.
 - Geralmente são encontrados o arquivo context.xml e manifest.mf.
- WEB-INF.
 - Especialmente importante por ser o local da aplicação em desenvolvimento onde serão armazenados os Servlets e JavaBeans.

Pacotes de Códigos-Fonte:

- Contêm os códigos Java que serão executados pelas requisições do cliente.

Bibliotecas:

- Recursos adicionais necessários à execução do projeto.
- Driver de acesso ao banco de dados.
- Biblioteca para geração de relatórios (p.e. JasperReport).

A pasta “WEB-INF” é especialmente importante por ser o local da aplicação em desenvolvimento no qual serão armazenados os Servlets e JavaBeans. A pasta “META-INF” armazena arquivos de configuração necessários para a execução dos recursos disponíveis na aplicação. Geralmente são encontrados o arquivo contexto.xml e manifest.mf.

Arquivo web.xml

O arquivo web.xml é utilizado para se definir as configurações das aplicações em desenvolvimento, como por exemplo, o nome do Servlet, a localização da classe, o url-pattern e o timeout de sessão. A seguir apresentamos o conteúdo de um arquivo web.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/
javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
  <servlet>
    <servlet-name>ExemploServlet01</servlet-name>
    <servlet-class>servlets.ExemploServlet01</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>ExemploServlet01</servlet-name>
    <url-pattern>/ExemploServlet01</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>
      30
    </session-timeout>
  </session-config>
</web-app>
```

Executando um Servlet

É possível executar um Servlet diretamente na barra de endereços do browser, bastando indicar a URL + <url-pattern>. Seria algo como:

```
http://localhost:8084/sessao04/exemplo01
```

O mais comum, contudo, é usar uma página HTML para fazer uma referência a um Servlet que foi mapeado no arquivo web.xml. A seguir, mostramos um exemplo de página HTML com essa referência.

```
<html>
  <head>
    <title></title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  </head>
  <body>
    <p align="center">&nbsp;</p>
    <p align="center">Um Exemplo de Execução de Servlets</p>
    <p align="center">&nbsp;</p>
    <p align="center"><a href="ExemploServlet01">Clique aqui para executar o
Servlet</a></p>
  </body>
</html>
```

A página HTML apresentada faz referência a <url-pattern> do <servlet-mapping>, definida dentro do arquivo web.xml. A referência a ExemploServlet01 possibilita o link necessário para que a aplicação em desenvolvimento consiga identificar um Servlet e colocá-lo em execução. A figura 4.4 procura demonstrar essas relações.

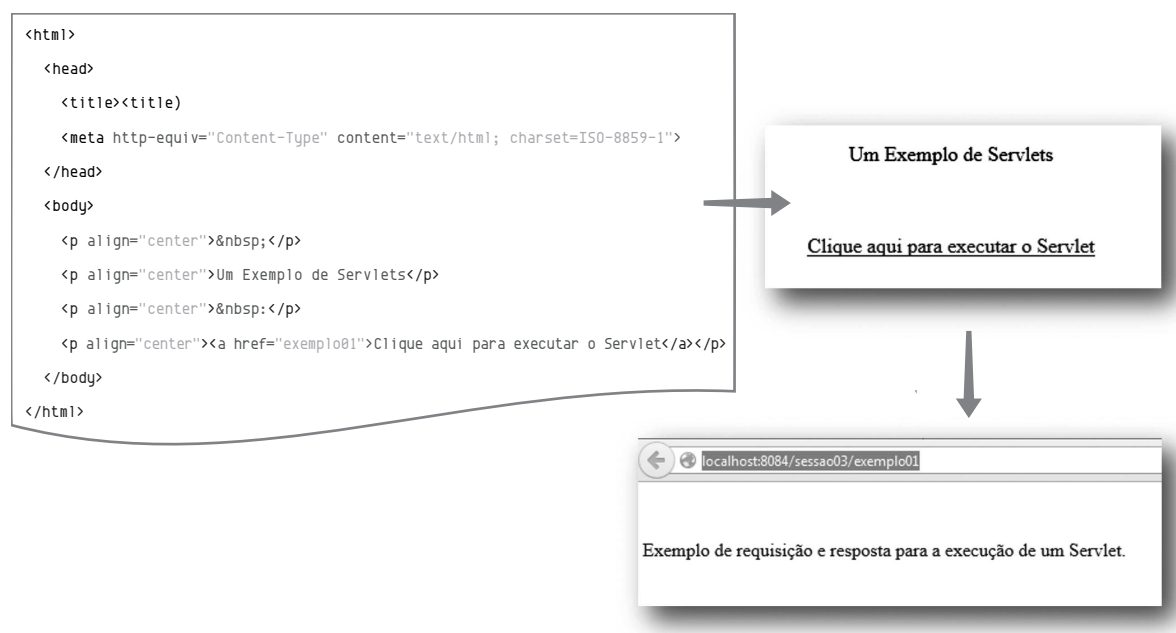


Figura 4.4
Servlet
referenciado via
página HTML.

Servlet com passagem de parâmetros

Vimos nos exemplos anteriores que podemos chamar um Servlet em uma página HTML. As páginas HTML disponibilizam vários recursos que podem ser utilizados para o desenvolvimento de interfaces amigáveis com o usuário, sendo essas acessíveis em browsers. Entre eles temos a possibilidade de criar um formulário para entrada de dados e posterior envio desses dados ao servidor. Para isso devemos utilizar a tag html <form> e configurar o método de submissão e o Servlet que vai receber os valores para que possam ser processados de acordo com a necessidade da aplicação em desenvolvimento.

O código HTML, a seguir, ilustra a criação de uma página que possibilita o envio de dados do cliente para que possam ser processados por um Servlet no lado servidor da aplicação em desenvolvimento. A execução desse código em um browser tem como resultado o que é exibido na figura 4.5.

```
<html>
  <head>
    <title></title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  </head>
  <body>
    <form method="POST" action="ExemploServlet02">
      Digite uma frase: <input name="frase" value="">
      <br><br>
      <input type="submit" value="Enviar">
    </form>
  </body>
</html>
```

Figura 4.5
Formulário HTML
de entrada
de dados.

Um formulário HTML simples com um campo de texto e um botão de envio. O campo de texto contém o texto "Digite uma frase:" seguido de um campo de entrada vazio. Abaixo do campo de texto, há um botão com o texto "Enviar".

Digite uma frase:

Servidor:

- Uso do doPost().
- Uso do request.getParameter(String param).
- Uso de Wrappers e casting.

A seguir, apresentamos o método doPost() da classe Servlet, que vai receber os valores enviados do formulário HTML e realizar o processamento das informações. As classes responsáveis pela comunicação entre cliente e servidor são: HttpServletRequest e HttpServletResponse.

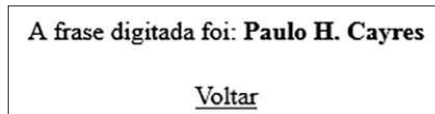
```
protected void doPost(HttpServletRequest request,
                      HttpServletResponse response)
    throws ServletException, IOException {
    processRequest(request, response);

    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
        /* TODO output your page here. You may use following sample code. */
        out.println("<html>");
        out.println("<head>");
        out.println("    <title>Servlet ExemploServlet02</title>");
        out.println("</head>");
        out.println("    <body>");
        out.println("        <center>");
        out.println("            A frase digitada foi: <b>" + request.
getParameter("frase") + "</b>");
        out.println("            <br><br><a href='exemplo02.html'>Voltar</a>");
        out.println("        </center>");
        out.println("    </body>");
        out.println("</html>");
    } finally {
        out.close();
    }
}
```

O objeto criado a partir da interface HttpServletRequest encapsula todos os dados referentes à solicitação do usuário. Esses dados são posteriormente acessados pelo método da própria interface, bem como métodos herdados da interface ServletRequest.

O objeto criado a partir da interface HttpServletResponse realiza o caminho inverso da interface HttpServletRequest. O objeto gerado a partir da instanciação dela envia dados encapsulados ao usuário em resposta à solicitação do cliente.

Figura 4.6
HTML resultante
do processo de
Servlet.

O resultado HTML gerado pelo servidor. O texto "A frase digitada foi: Paulo H. Cayres" está exibido em negrito. Abaixo dele, há um link "Voltar" sublinhado.

A frase digitada foi: **Paulo H. Cayres**

[Voltar](#)

Exercícios de fixação _ Servlets

Qual a diferença entre applet e Servlet?

Atividades Práticas

5

JSP (Java Server Pages)

objetivos

Apresentar aos alunos os mecanismos de controle de aplicações Java para internet com o uso de scriplets JSP, tecnologia utilizada no desenvolvimento de páginas dinâmicas acessíveis por meio de navegadores internet.

conceitos

JSP; dados estáticos; elementos sintáticos e objetos implícitos.

Introdução

JSP (Java Server Pages) é uma tecnologia para desenvolvimento de aplicações web semelhante ao Microsoft Active Server Pages (ASP), porém tem a vantagem da portabilidade de plataforma podendo ser executado em outros Sistemas Operacionais além daqueles oferecidos pela Microsoft. Permite ao desenvolvedor de sites produzir aplicações que acessam banco de dados ou arquivos-texto, que captem informações a partir de formulários, quem captem informações sobre o visitante ou sobre o servidor, o uso de variáveis e loops, entre outras coisas.

JavaServer Pages (JSP).

- Tecnologia que ajuda os desenvolvedores de software a criarem páginas web geradas dinamicamente.
- Baseadas em HTML, XML e outros tipos de documentos.

Lançada em 1999 pela Sun Microsystems.

- Similar ao PHP e ASP, mas usa a linguagem de programação Java.

Não oferece nada que você não possa conseguir com os servlets puros.

- Vantagem de ser facilmente codificado, facilitando assim a elaboração e manutenção de uma aplicação.

Quem conhece servlets verá que o JSP não oferece nada que você não possa conseguir com os servlets puros. O JSP, entretanto, oferece a vantagem de ser facilmente codificado, facilitando assim a elaboração e manutenção de uma aplicação. Além disso, essa tecnologia permite separar a programação lógica (parte dinâmica) da programação visual (parte estática), facilitando o desenvolvimento de aplicações mais robustas, onde programador e designer podem trabalhar no mesmo projeto, mas de forma independente. Outra característica do JSP é produzir conteúdos dinâmicos que possam ser reutilizados.



Quando uma página JSP é requisitada pelo cliente através de um browser, essa página é executada pelo servidor, e a partir daí será gerada uma página HTML que será enviada de volta ao browser do cliente. A figura 5.1 ilustra esse funcionamento.

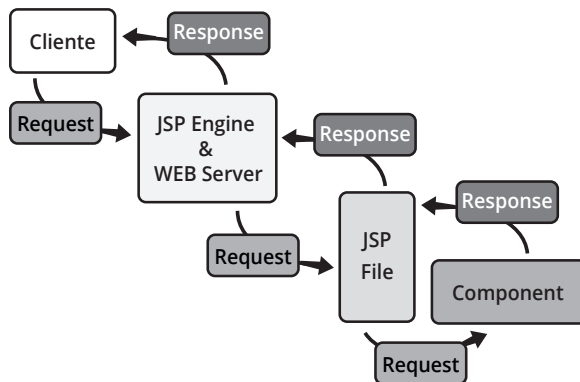


Figura 5.1
Funcionamento da arquitetura JSP.

Quando o cliente faz a solicitação de um arquivo JSP, é enviado um object request para a JSP engine. A JSP engine envia a solicitação de qualquer componente (podendo ser um JavaBeans component, servlet ou enterprise Bean) especificado no arquivo. O componente controla a requisição possibilitando a recuperação de arquivos em banco de dados ou outro dado armazenado e, em seguida, passa o objeto response de volta para a JSP engine. A JSP engine e o web server enviam a página JSP revisada de volta para o cliente, onde o usuário pode visualizar os resultados através do browser. O protocolo de comunicação usado entre o cliente e o servidor pode ser HTTP ou outro protocolo.

Por definição, JSP usa Java como sua linguagem de scripts. Por esse motivo, O JSP se apresenta mais flexível e mais robusto do que outras plataformas baseadas simplesmente em JavaScripts e VBScripts.

Dificuldades no uso de servlets:

- Problemas de manutenção.
- Legibilidade do código.
- Código Java misturado com código HTML.

Vantagens do JSP:

- Utilização de forma direta no HTML.
- Possibilita a inclusão de instruções Java.



Elementos sintáticos

A sintaxe JSP especifica a presença de dois tipos genéricos de dados para a formação de uma página JSP:

- Dados estáticos.
- Os próprios elementos sintáticos.

Os dados estáticos são todos os que não podem ser interpretados por serem desconhecidos da sintaxe JSP, como texto, HTML e XML.

Os elementos sintáticos são os próprios componentes da sintaxe JSP, aqueles que podem ser interpretados por um servidor web que processe códigos JSP.

A sintaxe JSP especifica a presença de dois tipos genéricos de dados para a formação de uma página JSP:

- Dados estáticos: não podem ser interpretados por serem desconhecidos da sintaxe JSP
 - ▣ Textos, HTML e XML.
- Os próprios elementos sintáticos: podem ser interpretados.
 - ▣ Os elementos do tipo diretiva.
 - ▣ Os elementos do tipo ação.
 - ▣ Os elementos do tipo scripting.

Diretivas

As diretivas são elementos vitais para a fase de tradução da página JSP. Por meio das diretivas são obtidas informações globais e que não dependem de qualquer solicitação, como, por exemplo, a inclusão de uma classe Java, a especificação de que a página JSP faz parte de uma sessão, entre outras. Por isso, sempre são conceitualmente válidas. A sintaxe básica é:

```
<%@   diretiva %>
```

Diretivas – formato de uso.

- <%@ nome-da-diretiva [nome-do-atributo = “valor-do-atributo”] %>

Diretivas disponíveis:

- <%@ page lista-de-atrributos %>
- <%@ include file=“url-relativa”
- <%@ taglib uri=“url-relativa-biblioteca-tags” prefix=“prefixo-tags” %>

A seguir, mostramos um trecho de código exemplificando o uso da diretiva page.

```
<%@ page contentType="text/html" pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
    <title>JSP Page</title>
  </head>
  <body>
    <title>Exemplo diretiva page</title>
  </body>
</html>
```

No exemplo anterior, a diretiva page contentType é usada para definir o tipo MIME do conteúdo a ser enviado na resposta da execução de uma página JSP, sendo o seu valor padrão “text/html”. Já a diretiva pageencoding é utilizada para definir o conjunto codificador de caracteres para a página JSP. Nesse exemplo, estamos utilizando o conjunto de caracteres Latin 1.

Já no exemplo a seguir, definimos o contentType como sendo um arquivo Excel. Desta forma, o conteúdo a ser processado na página será disponibilizado para visualização ou download como sendo uma planilha Excel e não como um conjunto de tags HTML a ser interpretada pelo browser.

```
<%@ page contentType="application/vnd.ms-excel" pageEncoding="ISO-8859-1"%>
<html>
  <head>
    <title>Exemplo diretiva page-contentType</title>
  </head>
  <body>
  </body>
</html>
```

O arquivo de exemplo ExemploContentType.jsp ilustra a utilização do contentType que indica o tipo de arquivo planilha Excel, fazendo com que o Sistema Operacional abra uma janela de diálogo já identificando o tipo do arquivo, conforme pode ser visto na figura 5.2.

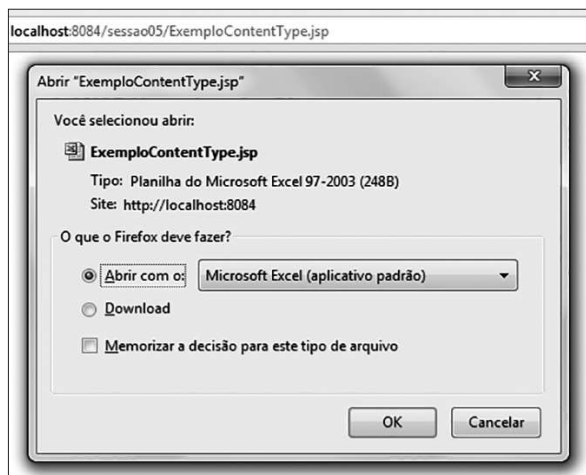


Figura 5.2
Janela de diálogo usando informação definida no atributo contentType.

Ações

As informações fornecidas pelas ações são empregadas na fase de execução. Por estar ligada à fase de solicitação, a interpretação de uma ação depende diretamente dos detalhes da solicitação recebida. A seguir, podemos ver a sintaxe desse tipo de elemento com alguns exemplos práticos.

<jsp:<identificador-da-ação>/>

- <jsp:include page="{ relativeURL | <%= expression %> }" />
- <jsp:forward page="{ relativeURL | <%= expression %> }" />



Elementos de script

Os elementos de script estabelecem um mecanismo que une os dados estáticos e os dados dinâmicos.

Existem três tipos de elementos de script:

- Declarações
 - <%! declaração; %>.
- Scriptlets
 - <% código; %>.
- Expressões
 - <%= expressão; %>.



Um quarto tipo de elemento de scripting é o comentário, que serve para documentar o código JSP e que não são enviados para as solicitações realizadas pelos browsers clientes. Comentários podem usar as seguintes sintaxes: `<!-- comentário -->`, `<%-- comentário --%>` e `<%/* comentário */ %>`. A última alternativa é utilizada dentro da página através de um scriptlet, usando a sintaxe de comentário nativa da linguagem Java de criação de scripts.

As Declarações são empregadas na declaração de métodos e variáveis a serem utilizados pela página JSP, conforme exemplificado a seguir.

```
<!-- Exemplo de Declaração de variáveis -->
<%!
    private String mensagem="Exemplo de váriáveis e métodos";
    Date agora = new Date();
    DateFormat df = DateFormat.getDateInstance(DateFormat.FULL);
%>
<!-- Exemplo de Declaração de método -->
<%!
public String getMensagem(){
    return this.mensagem;
}
%>
```

Os scriptlets são empregados com códigos de programação que serão executados a cada nova solicitação da página. Fazem uso da estrutura da linguagem de programação Java conforme podemos ver no trecho de código a seguir.

```
<!-- Uso de scriptlet -->
<%
    out.println("<hr>" + this.getMensagem());
    int num = 25;
    if( num % 2 == 0)
        out.println(num + " é par!");
    else
        out.println(num + " é impar!");
%>
```

Expressões, por sua vez, são empregadas na geração de dados a serem exibidos ao usuário na resposta. Assim como os scriptlets, as expressões são executadas a cada nova solicitação. Veja o exemplo:

```
<!-- Uso de expressões -->
<hr>
<%= this.getMensagem() %>
<hr>
Essa página foi acessada em <%= df.format(agora) %>
```

O resultado da execução do arquivo `declarandovariaveis.jsp`, contendo os exemplos anterior e disponível no diretório de exemplos desta sessão, pode ser visto na figura 5.3.

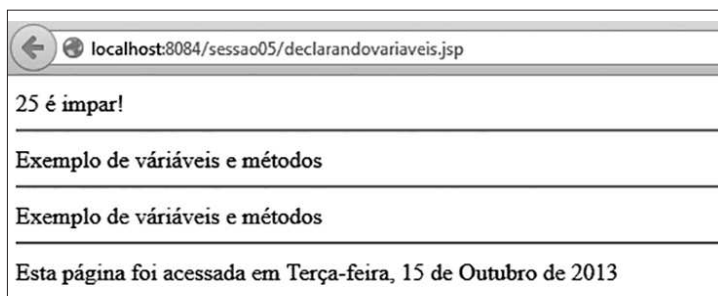


Figura 5.3
Resultado da
execução de
declarando
variaveis.jsp.

Exemplos JSP

O código JSP a seguir apresenta uma página que será construída dinamicamente com dados que serão enviados a ela por meio de uma página HTML com um formulário de entrada de dados. O valor enviado será processado por meio do elemento de script `<%= {expressão} %>`. Nesse exemplo, utilizamos o elemento implícito `request` para capturar o valor submetido pelo método `getParameter()`.

```
<%@page contentType="text/html" pageEncoding="UTF-8" language="java"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Page</title>
  </head>
  <body>
    <center>
      <h1>A frase digitada foi: <%= request.getParameter("frase") %></h1>
    </center>
  </body>
</html>
```

O código HTML a seguir é utilizado para possibilitar a digitação de uma frase e seu posterior envio pelo scriptlet JSP apresentado anteriormente para processamento no servidor.

```
<html>
  <head>
    <title>Executando um jsp</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  </head>
  <body>
    <form action="mensagem.jsp" method="POST">
      <p>Digite uma Frame:<input type="text" size="20" name="frase"></p>
      <p align="center">
        <input type="submit" name="B1" value="Enviar">
        <input type="reset" name="B2" value="Limpar">
      </p>
    </form>
  </body>
</html>
```

Já nesse outro exemplo mostramos como proceder para receber e alterar o tipo básico de um parâmetro enviado através de uma requisição web. Assim é possível realizar cálculos, tirando proveito da classe parser Integer para realizar a conversão do parâmetro, que é por definição textual, para inteiro.

```
<%@page language="java"
    import="java.io.*"
%>
<HTML>
  <HEAD>
    <TITLE>Teste JSP</TITLE>
  </HEAD>
  <BODY>
    <%int num = Integer.parseInt(request.getParameter("num")); %>
    <center>
      <h1>Tabuada do número <%= request.getParameter("num") %> </h1><br>
    </center>
    <% for(int i = 1; i <= 10; i++){
      out.print("<br>" + num + " * " + i + " = " + (num*i));
    } %>
    <br><br>
    <center>
      <a href="javascript:history.back();">voltar</a>
    </center>
  </BODY>
</HTML>
```

Objetos implícitos

Podemos criar, dentro de scriptlets na página JSP, instâncias de uma classe Java e manipulá-las a fim de produzir conteúdo dinâmico. Por exemplo, podemos criar um objeto de uma classe que acessa uma base de dados e então usar métodos desse objeto para exibir na página uma consulta ou transação com a base de dados. Através da manipulação desse objeto, quer seja acessando seus métodos ou suas variáveis, podemos gerar conteúdo dinâmico para a página JSP.

Além de objetos como esses, que estão completamente sob o controle do programador, o container JSP se encarrega de instanciar automaticamente, durante a execução de uma página JSP, alguns objetos. Tais objetos podem ser usados dentro da página JSP e são conhecidos como “Objetos Implícitos”.

- O container JSP se encarrega de instanciar automaticamente, durante a execução de uma página, alguns objetos.
- Tais objetos podem ser usados dentro da página JSP e são conhecidos como “Objetos Implícitos”.
- Um Objeto Implícito é sempre uma instância de uma classe ou interface pertencente ao container JSP.



Assim como todo objeto em Java, cada objeto implícito é uma instância de uma classe ou interface e segue uma API correspondente. A tabela 5.1 apresenta um resumo dos objetos implícitos disponíveis em JSP, suas respectivas classes ou interfaces e uma pequena descrição do objeto.

Objeto	Classe ou interface	Descrição
Page	javax.servlet.jsp.HttpJspPage	Instância de servlet da página.
Config	javax.servlet.ServletConfig	Dados de configuração de servlet.
Request	javax.servlet.http.HttpServletRequest	Dados de solicitação, incluindo parâmetros.
Response	javax.servlet.http.HttpServletResponse	Dados de resposta.
Out	javax.servlet.jsp.JspWriter	Fluxo de saída para conteúdo da página.
session	javax.servlet.http.HttpSession	Dados de sessão específicos de usuário.
application	javax.servlet.ServletContext	Dados compartilhados por todas as páginas de aplicação.
pageContext	javax.servlet.jsp.PageContext	Dados de contexto para execução da página.
exception	javax.lang.Throwable	Erros não capturados ou exceção.

Páginas JSP não são executadas diretamente.

JSP é convertido em servlets.

- Na primeira chamada ao JSP.
- O que ocorre: servlet é gerado com o uso das marcações JSP.

Em tempo de execução.

- Não existe interpretação de páginas JSP.
- Somente código servlet é executado.

Os nove objetos implícitos acima listados são analisados em maior detalhe a seguir.

Tipos de Objetos Implícitos:

- Objetos relacionados ao servlet.
 - ▣ page e config.
- Objetos relacionados ao input e output de páginas.
 - ▣ request, response e out.
- Objetos contextuais (acesso ao contexto da página JSP).
 - ▣ application, session, request e pageContext.
- Objetos relacionados a diferentes tipos de erro.
 - ▣ exception

Começamos com os objetos relacionados ao servlet da página, que são o page e config. Eles se baseiam na implementação da página JSP como um servlet.

Depois seguimos com os objetos relacionados com o input (entrada de dados) e com o output (saída de informação) de uma página JSP, que são o request, response e out.



Tabela 5.1

Objetos implícitos de JSP e suas APIs para aplicações de HTTP.



Temos também os objetos contextuais `application`, `session`, `request` e `pageContext`. Esses objetos fornecem à página JSP o acesso ao contexto dentro do qual ela está respondendo. Todos eles têm a capacidade de armazenar e recuperar valores de atributos. Os atributos de página armazenados no objeto `pageContext` duram apenas enquanto o processamento de uma única página ocorre. Os atributos de solicitação, armazenados no objeto `request`, também têm pouca duração, mas podem ser transferidos entre páginas quando for transferido o controle.

Os atributos de sessão, armazenados no objeto `session`, duram enquanto o usuário continuar a interagir com o servidor da web. Os atributos de aplicação, armazenados no objeto `application`, são mantidos enquanto o container JSP mantiver uma ou mais páginas de uma aplicação carregada na memória (enquanto o container JSP estiver rodando).

A tabela 5.2 apresenta os métodos comuns a esses quatro objetos implícitos e que são usados para armazenar e recuperar valores de atributos.

Métodos	Descrição
<code>void setAttribute(String key, Object value)</code>	Associa um valor de atributo com um nome.
<code>Enumeration getAttributeNames()</code>	Recupera os nomes de todos os atributos associados com o objeto.
<code>Object getAttribute(String key)</code>	Recupera o valor de atributo associado com a chave.
<code>void removeAttribute(String key)</code>	Remove o valor de atributo associado com a chave.

Tabela 5.2
Os métodos
comuns aos objetos
contextuais.

Finalmente, temos o objeto que é resultante de diferentes tipos de erro: o objeto `exception`. Ele tem por propósito permitir o tratamento de erros dentro de uma página JSP.

Conforme já mencionado, cada um desses objetos implícitos será apresentado em mais detalhes a seguir.

Objeto page

Representa a própria página JSP ou, mais especificamente, uma instância da classe de servlet na qual a página foi traduzida.

Implementa as interfaces.

- `javax.servlet.jsp.HttpJspPage`.
- `javax.servlet.jsp.JspPage`.

No trecho de código a seguir, temos um exemplo utilizando o objeto implícito `page`.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"
    info = "Autor: Paulo Henrique Cayres"
%>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
    <title>Exemplo Objeto Page</title>
</head>
<body>
    <%= ((javax.servlet.jsp.HttpJspPage)page).getServletInfo() %>
</body>
</html>
```



Objeto config

O objeto config armazena dados de configuração do servlet: na forma de parâmetros de inicialização: para o servlet na qual uma página JSP é compilada. Pelo fato de as páginas JSP raramente serem escritas para interagir com parâmetros de inicialização, esse objeto implícito raramente é usado na prática. O objeto config é uma instância da interface `javax.servlet.ServletConfig`.

Objeto config:

- Armazena dados de configuração de servlet para o servlet na qual uma página JSP é compilada.
 - Faz isso na forma de parâmetros de inicialização.
- Esse objeto implícito raramente é usado na prática.
 - Páginas JSP raramente são escritas para interagir com parâmetros de inicialização.
- É uma instância da interface `javax.servlet.ServletConfig`.

Os métodos fornecidos por essa interface para recuperar parâmetros de inicialização de servlet estão listados a seguir:

Métodos	Descrição
<code>Enumeration getInitParameterNames()</code>	Recupera os nomes de todos os parâmetros de inicialização.
<code>String getInitParameter(String name)</code>	Recupera o valor do parâmetro de inicialização a partir de um nome.

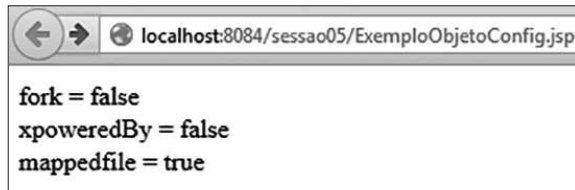
Apresentamos um exemplo do uso desse tipo de objeto implícito a seguir.

```
<%@page import="java.util.Enumeration"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Page</title>
  </head>
  <body>
    <%
      Enumeration temp = getInitParameterNames();
      while (temp.hasMoreElements()){
        String name = (String)temp.nextElement();
        out.println(name + " = " + getInitParameter(name)+"<br>");
      }
    %>
  </body>
</html>
```

A figura 5.4 apresenta o resultado da execução do servlet.

Tabela 5.3
Métodos do objeto config.

Figura 5.4
Exemplo do
uso do objeto
implícito config.



Objeto request

Representa a solicitação que requisitou a página.

Implementa a interface `javax.servlet.http.HttpServletRequest`.

- Uma subinterface de `javax.servlet.ServletRequest`.

Também é classificado como um objeto contextual.

- É um dos mais complexos e mais utilizados na construção de páginas JSP.

Podemos dividir os métodos desse objeto em quatro categorias:

- Armazenar e Recuperar valores de atributos.
- Recuperar parâmetros de solicitação e cabeçalho de HTTP.
- Recuperar cabeçalhos de solicitação e cabeçalhos de HTTP.
- Outros métodos diversos.

Por ser um dos objetos mais utilizados em páginas JSP vale a pena mostrar os principais métodos a ele associados, agrupados de acordo com as categorias recém-listadas.

Tabela 5.4
Métodos do objeto
request: armazenar
e recuperar valores
de atributos.

Método	Descrição
<code>void setAttribute (String key, Object value)</code>	Associa um valor de atributo com um nome.
<code>Enumeration getAttributeNames()</code>	Recupera os nomes de todos os atributos associados com o objeto.
<code>Object getAttribute (String key)</code>	Recupera o valor de atributo associado com a chave.
<code>void removeAttribute (String key)</code>	Remove o valor de atributo associado com a chave.

Tabela 5.5
Métodos do objeto
request: recuperar
parâmetros de
solicitação e
cabeçalho de HTTP.

Métodos	Descrição
<code>Enumeration getParameterNames()</code>	Retorna os nomes de todos os parâmetros de solicitação.
<code>String getParameter (String name)</code>	Retorna o primeiro valor (principal) de um único parâmetro de solicitação.
<code>String[] getParameterValues(String name)</code>	Recupera todos os valores para um único parâmetro de solicitação.

Métodos	Descrição
Enumeration getHeaderNames()	Recupera os nomes de todos os cabeçalhos associados com a solicitação.
String getHeader (String name)	Retorna o valor de um único cabeçalho de solicitação, como uma cadeia.
Enumeration getHeaders (String name)	Retorna todos os valores para um único cabeçalho de solicitação.
int getIntHeader (String name)	Retorna o valor de um único cabeçalho de solicitação, com um número inteiro.
long getDateHeader (String name)	Retorna o valor de um único cabeçalho de solicitação, como uma data.
Cookies[] getCookies()	Recupera todos os cookies associados com a solicitação.

Tabela 5.6
Métodos do objeto request: recuperar cabeçalhos de solicitação e cabeçalhos de HTTP..

Métodos	Descrição
String getMethod()	Retorna o método de HTTP (e.g. POST, GET etc.) para a solicitação.
String getRequestURI()	Retorna a URL de solicitação (não inclui a cadeia de consulta).
String getQueryString()	Retorna a cadeia de consulta que segue a URL de solicitação, se houver algum.
HttpSession getSession()	Recupera os dados da sessão para a solicitação (i.e, o objeto implícito session).
HttpSession getSession(boolean flag)	Recupera os dados da sessão para a solicitação (i.e, o objeto implícito session), opcionalmente criando-o se ele ainda não existir.
RequestDispatcher getRequestDispatcher(String path)	Cria um dispatcher de solicitação para o URL local indicado.
String getRemoteHost()	Retorna o nome totalmente qualificado do host que enviou a solicitação.
String getRemoteAddr()	Retorna o endereço de rede (IP) do host que enviou a solicitação.
String getRemoteUser()	Retorna o nome do usuário que enviou a solicitação, se conhecido.

O trecho de código a seguir apresenta um exemplo da utilização do objeto implícito request na construção de scriptlets JSP para páginas dinâmicas.

```
.....
Seu IP é :<%= request.getRemoteAddr() %><br>
Seu Host é :<%= request.getRemoteHost() %><br>
.....
```

Tabela 5.7
Métodos do objeto request: outros métodos diversos.

Objeto response

Representa a resposta que será enviada de volta para o usuário como resultado do processamento da página JSP.

Implementa a interface `javax.servlet.http.HttpServletResponse`.

- É uma subinterface de `javax.servlet.ServletResponse`.

Podemos dividir os métodos desse objeto em quatro categorias:

- Especificar o tipo de conteúdo e codificação da resposta.





- Definir cabeçalhos da resposta.
- Definir códigos de resposta.
- Reescrita da URL.

Por ser também um dos objetos mais utilizados em páginas JSP, vamos igualmente mostrar os principais métodos a ele associados, agrupados de acordo com as categorias anteriormente listadas.

Tabela 5.5
Métodos do objeto response: especificar o tipo de conteúdo e codificação da resposta.

Métodos	Descrição
void setContentType (String type)	Define o tipo MIME e, opcionalmente, a codificação de caracteres do conteúdo da resposta.
String getCharacterEncoding()	Retorna o conjunto de estilos de codificação de caracteres para o conteúdo da resposta.

Tabela 5.8
Métodos do objeto response: Definir cabeçalhos da resposta.

Métodos	Descrição
void addCookies (Cookie cookie)	Adiciona o cookie especificado.
boolean containsHeader (String name)	Verifica se a resposta inclui o cabeçalho.
void setHeader(String name, String value)	Atribui o valor definido pela variável "value" ao cabeçalho especificado por "name".
void setIntHeader (String name, int value)	Atribui o valor de número inteiro especificado por "value" ao cabeçalho especificado por "name".
void setDateHeader (String name, long date)	Atribui o valor de data especificado por "value" ao cabeçalho especificado por "name".
void addHeader (String name, String value)	Adiciona o valor definido por "value" ao cabeçalho especificado por "name".
void addIntHeader (String name, int value)	Adiciona o valor de número inteiro especificado por "value" ao cabeçalho especificado por "name".
void addDateHeader (String name, long date)	Adiciona o valor de data especificado por "value" ao cabeçalho especificado por "name".

O exemplo a seguir ilustra uma das utilidades desse objeto. Vários cabeçalhos são definidos para evitar que a página seja armazenada em cache por um navegador.

```
<%
    response.setHeader("Expires", 0);
    response.setHeader("Pragma", "no-cache");
    if(request.getProtocol().equals("HTTP/1.1")){
        response.setHeader("Cache-Control", "no-cache");
    }
%>
```

Esse script primeiro define o cabeçalho Expires para uma data no passado. Isso significa que o conteúdo da página já expirou, como uma dica que seu conteúdo não deve ser armazenado em cache.



Métodos	Descrição
<code>void setStatus(int code)</code>	Define o código de status para a resposta (para circunstâncias sem erro).
<code>void sendError(int status, String msg)</code>	Define o código de status e mensagem de erro para a resposta.
<code>void sendRedirect(String url)</code>	Envia uma resposta para o navegador indicando que ele deveria solicitar uma URL alternativa (absoluto).

Tabela 5.9
Métodos do objeto `response`: definir códigos de resposta.

Métodos	Descrição
<code>String encodeRedirectURL(String url)</code>	Codifica uma URL para uso com o método <code>sendRedirect()</code> para incluir informações de sessão.
<code>String encodeURL(String url)</code>	Codifica um URL usado em um link para incluir informações de sessão.

Tabela 5.10
Métodos do objeto `response`: Reescrita da URL.

Objeto out

- Esse objeto representa o fluxo de saída para a página.
- O conteúdo será enviado para o navegador com o corpo de sua resposta.
- É uma instância da classe `javax.servlet.jsp.JspWriter`.
- Implementa todos os métodos `print()` e `println()` definidos por `java.io.Writer`.
- Pode ser usado dentro de um script para adicionar conteúdo à página gerada.



Vejamos um exemplo do uso do objeto `out`:

```
<%
int i = (int)(Math.random()*10);
if(i%2==0){
    out.print("O Número escolhido "+ i +" é par!");
}
else {
    out.print("O Número escolhido "+ i +" é impar!");
}
%>
```

Esse objeto é muito utilizado para gerar conteúdo dentro do corpo de um script, sem ter de fechá-lo temporariamente para inserir conteúdo de página estática. Contudo, deve-se evitar usar os métodos `print()` ou `println()` para inserir cadeias de caracteres muito grandes. No próximo exemplo, é mais aconselhável fechar o script e inserir o conteúdo estático. Veja o resultado:

```
<!-- Não Aconselhável: --%>
<%
if(i %2 == 0){
out.print("<h6>"+
"<font face='verdana'>"+
"O número é par."+
"</font>"+
"</h6>");
```

```

}
%>
<%-- Aconselhável: --%>
<% if(i == 1) {%>
<h6>
<font face='verdana'>
0 número é par
</font>
</h6>
<% } %>

```

O segundo exemplo do quadro anterior utiliza tags HTML sem realizar a concatenação de string através de código Java. Isso torna a criação e resposta da página HTML dinâmica para o cliente solicitante mais rápida, uma vez que o código dinâmico se restringe apenas ao teste de condição if. Já o primeiro exemplo tem-se o gasto de processamento de concatenação de string além do já mencionado teste de condição.

Objeto session

Representa a sessão atual de um usuário individual e armazena informações sobre ela.

- Todas as solicitações feitas por um usuário são consideradas parte de uma sessão.
- Após um período de tempo sem receber qualquer nova solicitação do usuário, a sessão expira.

Implementa a interface `javax.servlet.http.HttpSession`.

Um de seus principais usos é armazenar e recuperar valores de atributos para transmitir as informações específicas de usuários entre as páginas.

A seguir, um exemplo que armazena dados na sessão, na forma de um objeto que é instância de uma classe hipotética “Usuario”:

```

<%
    Usuario u = new Usuario(nome, senha);
    session.setAttribute("usuario", u);
%>

```

Uma vez que um objeto tenha sido armazenado através do método `setAttribute()`, ele pode ser recuperado na mesma página ou em outra acessada pelo usuário. O código a seguir ilustra a recuperação do objeto armazenado no código anterior.

```

<%
    Usuario u = (Usuario)session.getAttribute("usuario");
    ....
%>

```

Perceba que o método `getAttribute()` retorna um objeto da classe `Object`, portanto, é necessário fazermos um cast para converter o objeto retornado em uma instância da classe desejada.

A seguir, os principais métodos utilizados por esse objeto, além daqueles descritos anteriormente na tabela 5.2:



Métodos	Descrição
Object getAttribute(String nome)	Recupera o objeto identificado por “nome”.
String getId()	Retorna o Id da sessão.
long getCreationTime()	Retorna a hora na qual a sessão foi criada.
long getLastAccessedTime()	Retorna a última vez que uma solicitação associada com a sessão foi recebida.
int getMaxInactiveInterval()	Retorna o tempo máximo (em segundos) entre solicitações pelo qual a sessão será mantida.
void setMaxInactiveInterval(int time)	Define o tempo máximo (em segundos) entre solicitações pelo qual a sessão será mantida.
boolean isNew()	Retorna se o navegador do usuário ainda não tiver confirmado o ID de sessão.
boolean invalidate()	Descarta a sessão, liberando quaisquer objetos armazenados como atributos.

Tabela 5.11
Métodos do objeto session.



Objeto application

- Representa a aplicação à qual a página JSP pertence.
- É uma instância da interface javax.servlet.ServletContext.
- Os containers JSP tipicamente tratam do primeiro nome de diretório em uma URL como uma aplicação.
- Podemos dividir os métodos desse objeto em quatro categorias:
 - ▣ Recuperar informações de versão do container servlet.
 - ▣ Interagir com arquivos e caminhos no servidor.
 - ▣ Suporte para log de mensagens.
 - ▣ Acessar parâmetros de inicialização.

Exemplo:

```
http://localhost:8080/curso/index.jsp
http://localhost:8080/curso/jsp/ex01/index.jsp
http://localhost:8080/curso/jsp/excalc/index.jsp
```

No exemplo anterior, todas as páginas JSP são consideradas parte da mesma aplicação, no caso, a aplicação curso.

Além dos métodos descritos na tabela 5.2, o objeto application também apresenta os seguintes outros métodos que podem ser divididos em quatro categorias:

Tabela 5.12
Métodos do objeto application – Recuperar informações de versão do container servlet.

Método	Descrição
String getServerInfo()	Retorna o nome e versão do container servlet.
int getMajorVersion()	Retorna a versão principal da API do servlet para o container servlet.
int getMinorVersion()	Retorna a versão secundária da API do servlet para o container servlet.

Tabela 5.13
Métodos do
objeto application
– Interagir
com arquivos
e caminhos no
servidor.

Método	Descrição
String getMimeType (String filename)	Retorna o tipo MIME para o arquivo indicado, se conhecido pelo servidor.
URL getResource (String path)	Traduz uma cadeia especificando uma URL em um objeto que acessa os conteúdos das URLs, localmente ou na rede.
InputStream getResourceAsStream(String path)	Traduz uma cadeia especificando uma URL em um fluxo de entrada para ler seu conteúdo.
String getRealPath(String path)	Traduz uma URL local em um nome de caminho no sistema de arquivo local.
ServletContext getContext (String path)	Retorna o contexto de aplicação para a URL local especificada.
RequestDispatcher getRequestDispatcher(String path)	Cria um dispatcher de solicitação para a URL local indicada.

Tabela 5.14
Métodos do objeto
application –
Suporte para log de
mensagens.

Método	Descrição
void log(String message)	Grava a mensagem o arquivo de log.
void log(String message, Exception e)	Grava a mensagem no arquivo de log, junto com a trilha de pilha para a exceção especificada.

Tabela 5.15
Métodos do objeto
application –
Acessar parâmetros
de inicialização.

Método	Descrição
Enumerations getInitParameterNames()	Recupera os nomes de todos os parâmetros de inicialização.
String getInitParameter (String name)	Recupera o valor do parâmetro de inicialização como o nome dado.

Objeto pageContext

O objeto pageContext fornece várias facilidades como gerenciamento de sessões, atributos, páginas de erro, inclusões e encaminhamento de requisições de fluxo de resposta. O objeto pageContext é uma instância da classe javax.servlet.jsp.PageContext.

Objeto pageContext:

- Fornece várias facilidades, tais como o gerenciamento de sessões, atributos, páginas de erro, inclusões e encaminhamento de requisições de fluxo de resposta.
- É uma instância da classe javax.servlet.jsp.PageContext.
- Podemos dividir os métodos desse objeto em três categorias:
 - ▢ Acessar outros objetos implícitos de JSP.
 - ▢ Envio de solicitações de uma página JSP para outra.
 - ▢ Acessar atributos através de múltiplos escopos.



Além dos métodos descritos na tabela 5.2, os principais métodos desse objeto podem ser divididos em três categorias:

Método	Descrição
Object getPage()	Retorna a instância de servlet para a página atual (objeto implícito page).
ServletRequest getRequest()	Retorna a solicitação que iniciou o processamento da página (objeto implícito request).
ServletResponse	Retorna a resposta para a página (objeto implícito response).
JspWriter getOut	Retorna o fluxo de saída atual para a página (objeto implícito out).
HttpSession getSession()	Retorna a sessão associada com a solicitação da página atual, se houver alguma (objeto implícito session).
ServletConfig getServletConfig()	Retorna o objeto de configuração de servlet (objeto implícito config).
ServletContext getServletContext()	Retorna o contexto no qual o servlet da página roda (objeto implícito application).
Exception getException()	Para páginas de erro, retorna a exceção passada para a página (objeto implícito exception).

Tabela 5.16
Métodos do objeto
pageContext –
Acessar outros
objetos implícitos
de JSP.

Método	Descrição
void forward(String path)	Encaminha o processamento para outro URL local dado pela String path.
void include(String path)	Inclui o output do processamento de outro URL local.

Tabela 5.17
Métodos do objeto
pageContext
– Envio de
solicitações de uma
página JSP para
outra.

Método	Descrição
void setAttribute(String key, Object obj, int scope)	Associa o valor do atributo “obj” com a chave “key” no escopo “scope”.
Enumeration getAttributeNamesInScope(int scope)	Recupera os nomes de todos os atributos associados com “key” no escopo “scope”.
Object getAttribute(String name, int scope)	Recupera o valor de atributo associado com “name” no escopo “scope”.
removeAttribute(String name, int scope)	Remove o valor de atributo associado com “name” no escopo “scope”.
Object findAttribute(String name)	Procura em todos os escopos pelo atributo associado com “name”.
int getAttributesScope(String name)	Retorna o escopo no qual o atributo associado com “name” está armazenado.

Tabela 5.18
Métodos do objeto
pageContext –
Acessar atributos
através de
múltiplos escopos.

Os métodos Object findAttribute(String name) e int getAttributesScope(String name) permitem a procura, através de todos os escopos definidos, por um atributo associado com uma String passada como parâmetro. Nos dois casos, o objeto pageContext vai realizar uma busca através dos escopos na seguinte ordem: pageContext, request, session e application.

Tabela 5.19
Objeto pageContext
– variáveis para
representação de
escopo.

A tabela anterior traz métodos que recebem parâmetros para especificar o escopo. A classe javax.servlet.jsp.PageContext fornece variáveis estáticas para representar esses quatro escopos diferentes. A tabela a seguir resume essas variáveis:

Variável	Descrição
PAGE_SCOPE	Escopo para atributos armazenados no objeto pageContext.
REQUEST_SCOPE	Escopo para atributos armazenados no objeto request.
SESSION_SCOPE	Escopo para atributos armazenados no objeto session.
APPLICATION_SCOPE	Escopo para atributos armazenados no objeto application.

O exemplo a seguir ilustra o uso do método “getAttribute” e das variáveis estáticas descritas na tabela anterior:

```
<%
User uPag=(User)pageContext.getAttribute("user",pageContext.PAGE_SCOPE)
//Recupera o object "usuario" do escopo pageContext
User uReq=(User)pageContext.getAttribute("user",pageContext.REQUEST_SCOPE)
//Recupera o object "usuario" do escopo request

User uSes=(User)pageContext.getAttribute("user",pageContext.SESSION_SCOPE)
//Recupera o object "usuario" do escopo session

User uApp=(User)pageContext.getAttribute("user",pageContext.APPLICATION_SCOPE)
//Recupera o object "usuario" do escopo application
%>
```

Objeto exception

O objeto exception não está automaticamente disponível em todas as páginas JSP. Esse objeto está disponível apenas nas páginas que tenham sido designadas como páginas de erro, usando o atributo isErrorPage configurado com true na diretiva page. O objeto exception é uma instância da classe java.lang.Throwable correspondente ao erro não capturado que fez com que o controle fosse transferido para a página de erro.

Objeto Exception.

- Não está automaticamente disponível em todas as páginas JSP.
 - ▣ Disponível apenas nas páginas que tenham sido designadas como páginas de erro.
 - ▣ Uso do atributo isErrorPage configurado com true na diretiva <@ page %>.
- É uma instância da classe java.lang.Throwable.
 - ▣ Correspondente ao erro não capturado.
 - ▣ Faz com que o controle seja transferido para a página de erro.

Os principais métodos da classe java.lang.Throwable que são utilizados dentro das páginas JSP são listados na tabela a seguir:



Método	Descrição
String getMessage()	Retorna a mensagem de erro descritiva associada com a exceção quando ela foi lançada.
void printStackTrace(PrintWriter out)	Imprime a pilha de execução em funcionamento quando a exceção foi lançada para o fluxo de saída especificado pelo parâmetro out.
String toString()	Retorna uma cadeia combinando o nome da classe da exceção com sua mensagem de erro, se houver alguma.

Tabela 5.20
Métodos do objeto exception.

O trecho a seguir ilustra o uso do objeto exception em uma página de erro JSP:

```
<@ page isErrorPage=true %>
<h1>Erro Encontrado</h1>
O seguinte erro foi encontrado:<br>
<b><%= exception %></b><br>
<% exception.printStackTrace(out); %>
```

O resultado do printStackTrace(out) pode ser visto na figura 5.5.

```
org.apache.jasper.JasperException: An exception occurred processing JSP page /ExemploObjetoException.jsp at line 19
16:   <body>
17:       O seguinte erro foi encontrado:<br>
18:       <b><%= exception %></b><br>
19:       <% exception.printStackTrace(); %>
20:   </body>
21: </html>

Stacktrace:
    org.apache.jasper.servlet.JspServletWrapper.handleJspException(JspServletWrapper.java:568)
    org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:470)
    org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:390)
    org.apache.jasper.servlet.JspServlet.service(JspServlet.java:334)
    javax.servlet.http.HttpServlet.service(HttpServlet.java:722)
    org.netbeans.modules.web.monitor.server.MonitorFilter.doFilter(MonitorFilter.java:393)

root cause
java.lang.NullPointerException
    org.apache.jsp.ExemploObjetoException_jsp._jspService(ExemploObjetoException_jsp.java:81)
    org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:70)
    javax.servlet.http.HttpServlet.service(HttpServlet.java:722)
    org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:432)
    org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:390)
    org.apache.jasper.servlet.JspServlet.service(JspServlet.java:334)
    javax.servlet.http.HttpServlet.service(HttpServlet.java:722)
    org.netbeans.modules.web.monitor.server.MonitorFilter.doFilter(MonitorFilter.java:393)
```

Figura 5.5
Resultado do
printStackTrace
(out).

Exercícios de fixação

JSP

Quais são os três elementos sintáticos disponibilizados em JSP?

Atividades Práticas

6

JSP e banco de dados

objetivos

Apresentar aos alunos o mecanismo de desenvolvimento de aplicações Java para internet com o uso de scripts JSP que possibilitam a manipulação de banco de dados com o uso da tecnologia Java JDBC e os padrões de projeto DAO, ConnectionFactory e MVC.

JDBC; Processamento em duas ou três camadas; Design Pattern; ConnectionFactory; Properties; JavaBeans; DAO; MVC; HTML DOM; Ferramenta de Relatórios.

conceitos

JDBC

Como foi visto na sessão 5, JSP é uma tecnologia para desenvolvimento de aplicações web que permite ao desenvolvedor ter acesso a arquivos-texto, captar informações a partir de formulários, usar variáveis, estruturas de controle e muito mais.

Nesta sessão, vamos tratar do uso de JSP para acessar informações armazenadas em bancos de dados. Para isso, faremos uso da tecnologia Java JDBC.

JDBC (Java Database Connectivity).

API que contém um conjunto de classes e interfaces necessário para que uma aplicação Java possa acessar SGBDRs.

- Acesso a qualquer tipo de fonte de dados (BDRs, planilhas e arquivos de dados).
- Acesso ao BD baseado em SQL.

É composta pelos pacotes java.sql e javax.sql.

Auxilia na escrita de aplicativos Java que gerenciam três atividades de programação:

- Ligação a uma fonte de dados.
- Envio de consultas e instruções de atualização para a fonte de dados.
- Recupera e processa os resultados recebidos do banco de dados em resposta à sua consulta.

O JDBC permite o processamento de informações em duas ou três camadas, conforme ilustrado nas figuras 6.1 e 6.2.



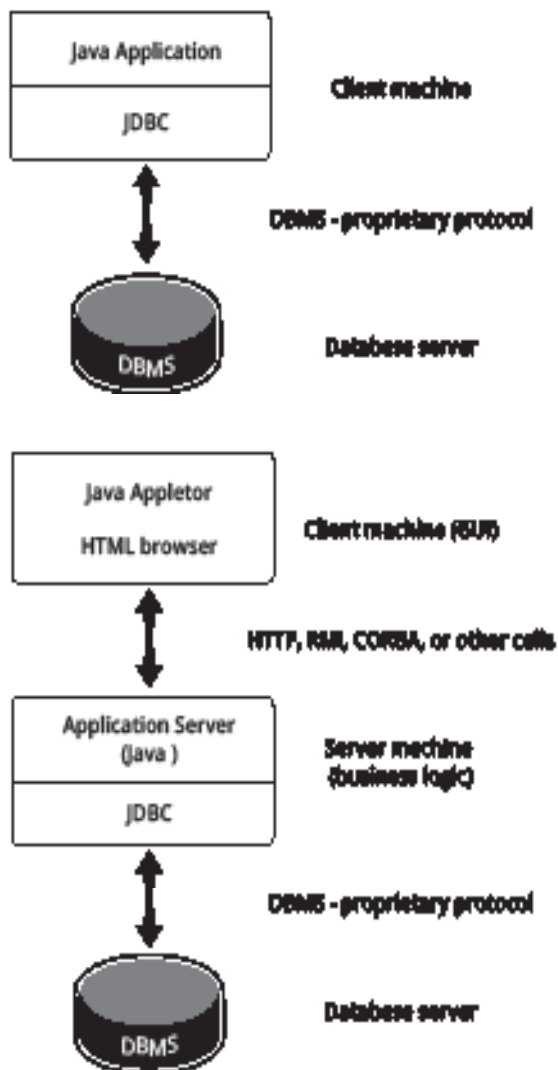


Figura 6.1
Modelo de
processamento em
duas camadas.

Figura 6.2
Modelo de
processamento em
três camadas.

Para tanto, o JDBC faz uso de uma arquitetura que é apresentada na figura 6.3.

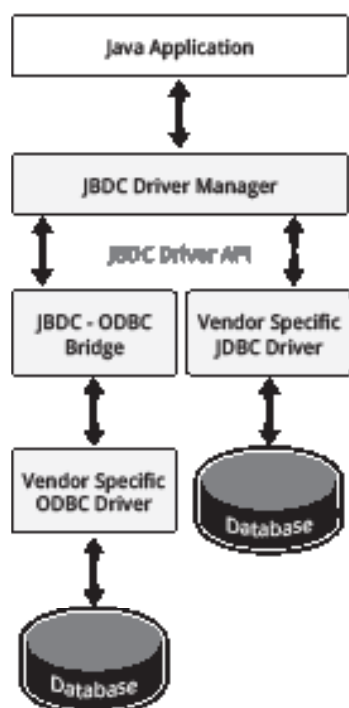


Figura 6.3
Arquitetura JDBC.

Essa arquitetura tem duas partes principais: JDBC API e JDBC Driver Manager. A primeira é uma API java-based pura, ou seja, é construída com a própria linguagem de programação Java. Já a JDBC Driver manager é usada para se comunicar com drivers específicos de fabricantes de BD/SGBD.

O uso do JDBC pressupõem os seguintes passos:

- Carregar o driver do BD/SGBD.
- Conectar ao BD.
- Especificar a operação SQL.
- Manipular os resultados obtidos.

A figura 6.4 demonstra o uso em conjunto de JSP com JDBC.

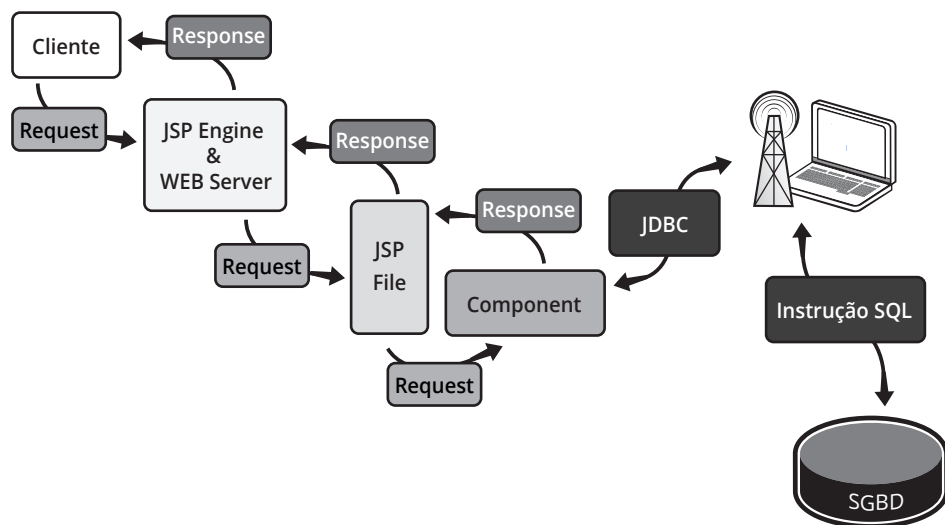


Figura 6.4
JSP e JDBC.

Neste curso, o banco de dados que será utilizado em todos os exemplos é o PostgreSQL. Por conta disso, será necessário fazer a importação da biblioteca de classes específica para esse SGBD, que é a biblioteca postgresql-9.0-801.jdbc4.jar (disponível no subdiretório lib do diretório de exemplos da sessão 6).

Vamos também fazer uso da tabela curso, cujo script SQL para sua criação pode ser visto a seguir.

```
CREATE TABLE curso
(
    idcurso serial NOT NULL,
    nome character varying(30) NOT NULL,
    CONSTRAINT curso_pkey PRIMARY KEY (idcurso)
)
WITH (
    OIDS=FALSE
);
ALTER TABLE curso
    OWNER TO postgres;
```

Padronizando a conexão ao BD

Cabe lembrar que não é desejável fazer a conexão ao BD no meio do código, podendo inclusive ser repetitivo caso essa conexão for feita sempre que se precisar acessar o banco. O melhor é fazer isso uma única vez, e para tanto podemos utilizar um Design Pattern, ou “padrão de projeto”, que é um método para resolver problemas que são comuns (e por isso mesmo repetitivos). No nosso caso, vamos utilizar um Design Pattern chamado Connection Factory.

Por que criar uma ConnectionFactory?

- Controlar o número de conexões requeridos pela aplicação.
- Controlar melhor as exceções de conexão com o banco de dados (em um único lugar).
 - É desejável configurar a conexão com o BD externamente ao código.
 - Podemos utilizar arquivos de configuração:
 - No Netbeans: Novo Arquivo / Outro / Arquivo de Propriedades.
 - Preencha o nome do arquivo como “BD.properties”.

Java disponibiliza uma classe onde podemos especificar algumas configurações que podem ser acessadas por uma determinada aplicação. A classe Properties possibilita a leitura de valores por meio de campos-chave que estão armazenados em um arquivo de propriedades separado da aplicação. O arquivo de propriedades é uma ótima opção para passar configurações para uma determinada aplicação que necessita de configurações externas.

O arquivo de propriedade é um arquivo simples salvo com a extensão.properties, que deve fazer parte da estrutura do projeto em desenvolvimento. Nos exemplos que vamos estudar, o arquivo de propriedades estará dentro do pacote útil. A seguir, apresentamos a estrutura do arquivo de propriedades BD.properties com um par de identificador e valor que serão lidos pela aplicação java que necessita dessas informações externas para seu funcionamento.

```
db.driver=org.postgresql.Driver
db.url=jdbc:postgresql://localhost:5432/pesquisa
db.user=postgres
db.pwd=123456
```

Um exemplo seria um programa que conecta a um banco de dados e precisa de dados para realizar a conexão, sem que o código-fonte deste seja alterado.

Figura 6.5
Estrutura
do arquivo
DB.properties.

Uma vez criado o arquivo de propriedades, uma aplicação Java pode acessá-lo e realizar a leitura de cada um dos valores dos identificadores nele armazenados. Para isso, devemos utilizar o método getProperty() da classe Properties, conforme o exemplo a seguir:

```
public class ConnectionFactoryComProperties{

    public Connection getConnection() {
        try {
            // Criação de um objeto de propriedades
            Properties prop = new Properties();
            // leitura/carga do arquivo texto com as propriedades
            prop.load( getClass().getResourceAsStream(
                "../../../../util/bancoDeDados.properties"));

            // obtém o valor de cada parâmetro através do método getProperty
            String dbDriver = prop.getProperty("db.driver");
            String dbUrl = prop.getProperty("db.url");
```



```

        String dbUser = prop.getProperty("db.user");
        String dbPwd = prop.getProperty("db.pwd");

        Class.forName(dbDriver);
        return DriverManager.getConnection(dbUrl, dbUser, dbPwd);

    } catch (ClassNotFoundException | IOException | SQLException ex) {
        throw new RuntimeException(ex);
    }
}
}
}

```

DAO e JavaBean

Veremos a seguir duas alternativas para a implementação do acesso ao banco de dados, ambas dentro do paradigma de orientação a objetos.

JavaBeans

- Classes que possuem o construtor sem argumentos e métodos de acesso do tipo get e set para acesso aos seus atributos (e serializável).

DAO (Data Access Object)

- Pattern que separa o acesso ao banco de dados da lógica de negócios.

JavaBean são classes que possuem o construtor padrão (sem argumentos) e métodos de acesso getters e setters, podendo ser serializável ou não. A serialização depende da fonte de dados utilizada. O código Java a seguir apresenta a estrutura do JavaBean para a tabela curso.

```

package modelo;
public class Curso {

    private int idcurso;
    private String nomeCurso;
    public Curso() {
        super();
        // TODO Auto-generated constructor stub
    }
    public Curso(int idcurso, String nomecurso) {
        super();
        this.idcurso = idcurso;
        this.nomeCurso = nomecurso;
    }
    public Curso(String nomeCurso) {
        super();
        this.nomeCurso = nomeCurso;
    }
    public Curso(int idcurso) {
        this.idcurso = idcurso;
    }
    public int getIdcurso() {
        return idcurso;
    }
}

```

```

    }
    public void setIdcurso(int idcurso) {
        this.idcurso = idcurso;
    }
    public String getNomeCurso() {
        return nomeCurso;
    }
    public void setNomeCurso(String nomecurso) {
        this.nomeCurso = nomecurso;
    }
}

```

O padrão Data Access Object ou padrão DAO é usado para separar a camada de acesso ao banco de dados da regra de negócio da aplicação, ficando responsável por gerenciar a conexão com a fonte de dados.

O DAO deve implementar os mecanismos de acesso à fonte de dados que se deseja manipular na interface da aplicação em desenvolvimento. Os detalhes de sua implementação não são de conhecimento do cliente. O código Java a seguir apresenta a estrutura da classe DAO para a tabela curso.

```

package controle;
import java.sql.*;
import java.util.ArrayList;
import modelo.Curso;
public class CursoDAO {
    private Connection con;
    public CursoDAO() throws Exception {
        this.con = new ConnectionFactoryComProperties().getConnection();
    }
    public void inserirCurso(Curso c){
        try{
            PreparedStatement stmt = this.con.prepareStatement(
                "insert into curso (nome) values(?)");
            stmt.setString(1, c.getNomeCurso());
            stmt.execute();
        } catch(SQLException e){
            System.out.println("incluirCurso(): "+e.toString());
        }
    }
    public Curso consultarCurso(int cod){
        try{
            PreparedStatement stmt = this.con.prepareStatement(
                "select * from curso where idcurso = ?");
            stmt.setInt(1, cod);
            ResultSet res = stmt.executeQuery();
            if(res.next())
                return new Curso(res.getInt("idcurso"),res.getString("nome"));
            else
                return null;
        }
    }
}

```

```

    } catch(SQLException e){
        System.out.println("consultarCurso(): "+e.toString());
    }
    return null;
}
public void atualizarCurso(Curso c){
    try{
        PreparedStatement stmt = this.con.prepareStatement(
            "update curso set nome=? where idcurso=?");
        stmt.setString(1, c.getNomeCurso());
        stmt.setInt(2, c.getIdcurso());
        stmt.executeUpdate();
    } catch(SQLException e){
        System.out.println("atualizarCurso(): "+e.toString());
    }
}
public void excluirCurso(Curso c){
    try{
        PreparedStatement stmt = this.con.prepareStatement(
            "delete from curso where idcurso=?");
        stmt.setInt(1, c.getIdcurso());
        stmt.executeUpdate();
    } catch(SQLException e){
        System.out.println("excluirCurso(): "+e.toString());
    }
}
public ArrayList<Curso> getCursos(){
    try{
        PreparedStatement stmt = this.con.prepareStatement(
            "select * from curso order by nome");
        ResultSet res = stmt.executeQuery();
        ArrayList<Curso> dados = new ArrayList<Curso>();
        while(res.next()) {
            Curso temp = new Curso(res.getInt("idcurso"),
                                   res.getString("nome"));
            dados.add(temp);
        }
        stmt.close();
        return dados;
    } catch (SQLException e) {
        System.out.println("getCursos(): "+e.toString());
    }
    return null;
}
}

```

Arquitetura MVC

Vamos aproveitar esta sessão para também tratar da organização dos componentes da aplicação web a partir da arquitetura MVC (Model-View-Controller), que é constituída das camadas modelo, visão e controle.

A figura 6.6 apresenta a estrutura de diretórios que será utilizada no desenvolvimento da aplicação web.

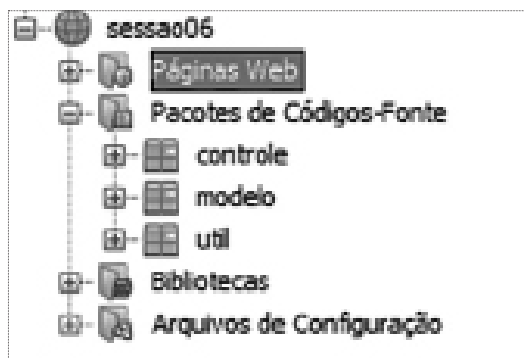


Figura 6.6
Estrutura de
diretórios da
aplicação web.

Camada visão:

- Será construída dentro da pasta “Páginas Web”.
- Disponibiliza as GUIs a que o usuário terá acesso.

Camada modelo.

- Será construída dentro de Pacotes de Códigos-Fonte.
- Um pacote de classes (new package).
- Conterá os JavaBeans do projeto.

Camada controle:

- Um pacote de classes (new package).
- Disponibilizará as classes Java com recursos para criação de conexão com o banco de dados.
- Interface DAO que vai possibilitar a manutenção de registro.

O pacote útil é uma extensão.

- Define o arquivo com as propriedades necessárias para a realização da conexão com o banco de dados.

Definição da interface gráfica com o usuário

O primeiro passo para a criação da aplicação JSP que possibilita a manipulação de banco de dados é a criação de uma interface gráfica com o usuário. Vamos utilizar páginas HTML como a porta de entrada para a construção de GUIs, tomando como referência o modelo DOM (Document Object Model) que foi padronizado pelo W3C. Nesse modelo, um documento qualquer pode ser descrito por uma estrutura de árvore, conforme o exemplo mostrado na figura 6.7.

Essas páginas vão possibilitar a interação dos usuários com informações armazenadas em um banco de dados, bem como o desenvolvimento de uma aplicação web no padrão MVC.

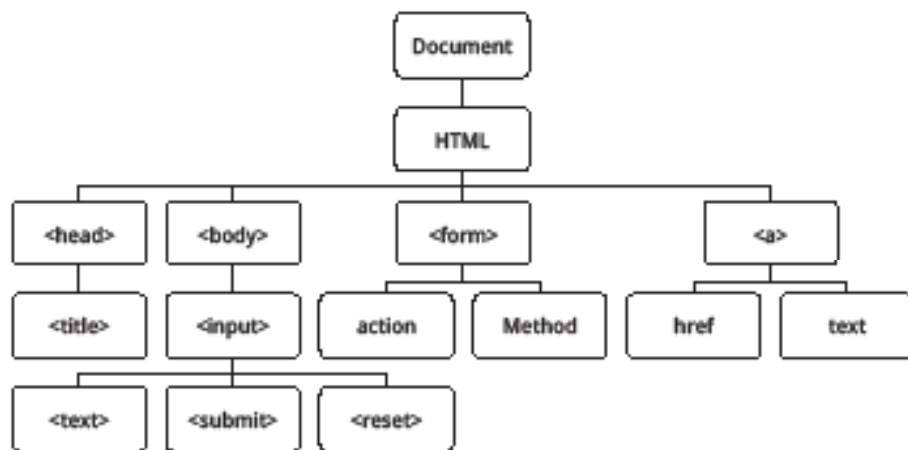


Figura 6.7
DOM: Document
Object Model.

O código HTML a seguir ilustra a criação de uma GUI com um formulário de entrada de dados que será utilizado pelo usuário da aplicação.

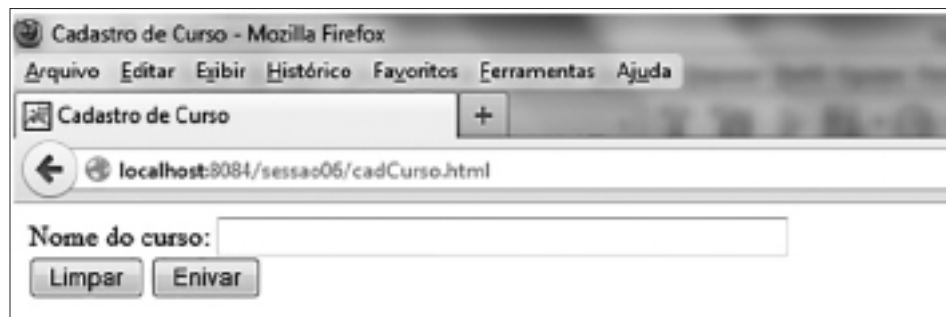
```

<html>
  <head>
    <title>Cadastro de Curso</title>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
  </head>
  <body>
    <form name="cadcurso" action="cadCurso.jsp" method="POST">
      Nome do curso:
      <input type="text" name="nome" size="50">
      <br>
      <input type="reset" name="b1" value="Limpar">
      <input type="submit" name="b2" value="Enviar">
    </form>
  </body>
</html>

```

Esse código HTML vai produzir a página HTML apresentada na figura 6.8. O endereço localhost:8084/sessao06/CadCurso.html possibilitará ao usuário a visualização do código HTML já interpretado pelo browser.

Figura 6.8
Página HTML para
entrada de dados.



Dentro dessa página HTML temos um link com o scriplet JSP que vai receber os valores digitados no formulário e fazer a interação com a camada de modelo e controle da aplicação para possibilitar a instância de um objeto da classe curso, a conexão com o banco de dados e execução do método de inserção do registro no BD descrito na classe DAO para a tabela curso.

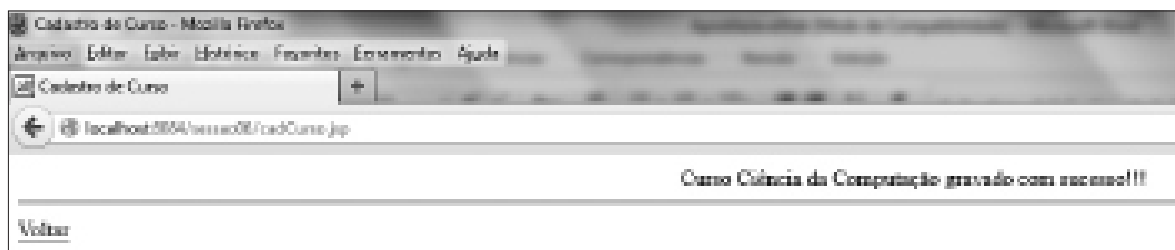
O scriplet JSP a seguir ilustra o processo de integração da camada de visão com as camadas modelo e controle da aplicação.

```
<%@page contentType="text/html"
    pageEncoding="UTF-8"
    import="controle.CursoDAO, modelo.Curso"
    language="java"
%>

<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
        <title>Cadastro de Curso</title>
    </head>
    <body>
        <%
            CursoDAO dao = new CursoDAO();
            Curso c = new Curso(request.getParameter("nome"));
            dao.inserirCurso(c);
        %>
        <center>Curso ${param.nome} gravado com sucesso!!!</center>
        <hr>
        <a href="index.jsp">Voltar</a>
    </body>
</html>
```

No scriplet JSP apresentado, destacamos a cláusula import na diretiva page que informa os recursos adicionais necessários para a instância dos objetos para executar a opção de gravação de um novo registro no banco de dados. Na sequência, devemos utilizar um elemento scriplet para escrever a codificação Java necessária para instância e uso dos objetos para efetivar a ação de gravação de um registro no banco de dados. A figura 6.9 apresenta o resultado da execução da página JSP cadCurso.jsp.

Figura 6.9
Página HTML
dinâmica produzida
por cadCurso.jsp.



Exemplo JSP – Listagem geral

O script JSP a seguir ilustra uma solução para a construção de uma página HTML dinâmica com os dados dos cursos cadastrados na tabela curso.

```
<%@page import="java.util.ArrayList"%>
<%@page contentType="text/html"
    pageEncoding="ISO-8859-1"
    import="modelo.Curso, controle.CursoDAO"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Lista de cursos cadastrados</title>
</head>
<body>
<center>
<table width="20%">
<tr><td>Código</td><td>Nome</td><td></td><td></td></tr>
<%
    CursoDAO dao = new CursoDAO();
    ArrayList<Curso> cursos = dao.getCursos();
    String tabela="";
    for(Curso temp : cursos){
        tabela += "<tr><td>" + temp.getIdcurso() + "</td><td>" +
            temp.getNomeCurso() + "</td>";
        tabela += "<td><a href='alterarCurso.jsp?id=' +
            temp.getIdcurso() + "'><img src='img/edit.png'
            alt='Editar' width='22' height='22'
            border='0'></a></td>";
        tabela += "<td><a href='excluirCurso.jsp?id=' +
            temp.getIdcurso() + "'><img src='img/editdelete.png'
            alt='Excluir' width='22' height='22'
            border='0'></a></td>";
        tabela += "</tr>";
    }
    out.println(tabela);
%>
</table>
<hr>
<a href="index.jsp">Voltar</a>
</center>
</body>
</html>
```

O método `getCursos()` da classe DAO de cursos foi construído para realizar uma consulta SQL que retorna todos os registros existentes na tabela em um objeto `ArrayList`. Isso possibilita que o desenvolvedor da GUI não se preocupe com as instruções de programação Java necessárias para acessar o banco de dados, manipular e retornar um objeto `ResultSet` a ser trabalhado na GUI.

Em vez disso, o método retorna um ArrayList já com os objetos da classe Curso instanciados. Isso facilita o processamento das informações durante o processo de construção da página dinâmica.

Para finalizar, criamos dois links que serão utilizados para ação de alteração e exclusão de registros gravados no banco de dados. Esse é um exemplo de utilização de método GET, onde passamos os valores das variáveis aberto após a URL. A figura 6.10 ilustra a execução do script JSP que monta a listagem de cursos dinamicamente.







Código	Nome		
5	Administração		
1	Ciência da Computação		
2	Direito		
Voltar			

Figura 6.10
Página HTML
dinâmica –
Listagem geral
de cursos.

Exemplo JSP – Exclusão

A seguir, o script JSP ilustra a codificação das instruções necessárias para realizar a exclusão de um registro no banco de dados.

```
<%@page import="modelo.Curso"%>
<%@page import="controle.CursoDAO"%>
<%@page contentType="text/html" pageEncoding="ISO-8859-1"%>
<%
    CursoDAO dao = new CursoDAO();
    Curso c = new Curso(Integer.parseInt(request.getParameter("id")));
    dao.excluirCurso(c);
%>
<jsp:forward page="listarCursos.jsp"></jsp:forward>
```

Esse script está acessível pelo link de exclusão de registro presente na listagem geral de cursos, conforme pode ser visto na figura 6.10 acima. Com isso, o usuário tem uma facilidade maior na hora de realizar a exclusão de um registro.

Exemplo JSP – alteração

A alteração de um registro existente deve ser feita em duas etapas. A primeira é colocar o registro que se deseja alterar para edição. A segunda é atualizar o registro no banco de dados com as novas informações fornecidas pelo usuário da aplicação.

O script JSP a seguir ilustra a primeira etapa. A tela que disponibiliza os dados do curso para edição está acessível pela listagem geral de cursos.


```

<%@page import="modelo.Curso"%>
<%@page import="controle.CursoDAO"%>
<%@page contentType="text/html" pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<%!
    Curso c;
%>
<%
    CursoDAO dao = new CursoDAO();
    c = dao.consultarCurso(Integer.parseInt(request.getParameter("id")));
%>
<html>
    <head>
        <title>Alterar Curso</title>
        meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"
    </head>
    <body>
        <form name="altcurso" action="updateCurso.jsp" method="POST">
            <input type="hidden" name="id" value="<%= c.getIdcurso() %>">
            Nome do curso:
            <input type="text" name="nome" size="50" value="<%=c.getNomeCurso()%>">
            <br>
            <input type="submit" name="b2" value="Enivar">
        </form>
    </body>
</html>

```

Esse script recebe o id do curso que se deseja alterar, instancia o objeto e utiliza seus métodos getters para escrever os valores atualmente gravados no banco de dados na GUI. Tomamos o cuidado de deixar o código, que é chave primária, escondido. Ele só será necessário na segunda etapa do processo de atualização do registro em edição.

O script JSP a seguir apresenta a codificação necessária para proceder com a atualização do registro em edição, usando os novos dados fornecidos pelo usuário na GUI.

```

<%@page import="modelo.Curso"%>
<%@page import="controle.CursoDAO"%>
<%@page contentType="text/html" pageEncoding="ISO-8859-1"%>
<%
    CursoDAO dao = new CursoDAO();
    Curso c = new Curso(Integer.parseInt(request.getParameter("id")),
                        request.getParameter("nome"));
    dao.atualizarCurso(c);
%>
<jsp:forward page="listarCursos.jsp"></jsp:forward>

```

A figura 6.11 ilustra a janela exibida para o usuário quando da edição do registro.

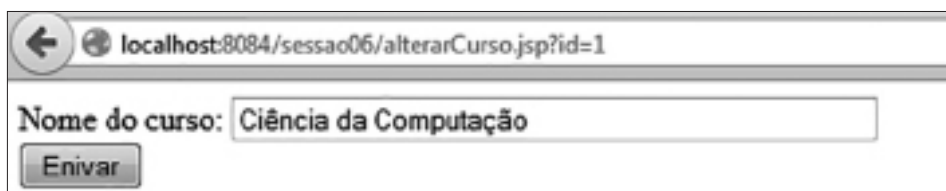


Figura 6.11
Página HTML para
edição do registro.

Ferramenta de relatório JasperReport

Interface amigável para geração de relatórios.

Escrita em Java.

Capaz de usar dados provenientes de qualquer tipo de fonte de dados.

- Atualmente considerada o mecanismo de relatórios mais popular do mundo open source.

Arquivos fonte com extensão .jrxml.

O layout do relatório é definido em um arquivo XML.

- Todas as informações de formatação do relatório.
- Todos os campos de dados a serem preenchidos pela fonte de dados.

Após a construção do relatório dentro da interface do iReport, você deverá fazer a inclusão desse novo recurso dentro do projeto da aplicação Java em desenvolvimento. Para isso, você deverá copiar o arquivo .jasper resultante do processo de criação dentro do iReport e importar as bibliotecas do JasperReports para dentro do projeto em desenvolvimento.

As bibliotecas do JasperReports são:

- commons-digester-2.1.jar
- commons-beanutils-1.8.2.jar
- commons-collections-3.2.1.jar
- commons-javaflow-20060411.jar
- commons-logging-1.1.jar
- groovy-all-1.7.5.jar
- JasperReports-4.7.0.jar
- iText-2.1.7.jar

Esses arquivos encontram-se dentro da pasta "ireport\modules\ext" no diretório de instalação da interface iReport.

Lembre-se de importar a biblioteca de acesso ao banco de dados PostgreSQL, para que o relatório possa estar acessando a fonte de dados que foi informada durante o processo de desenvolvimento. Após a importação das bibliotecas, você deverá desenvolver um programa Java e nele fazer a chamada ao relatório referenciado pelo arquivo com a extensão .jasper (resultante do processo de criação dentro da interface iReport).

O script JSP a seguir representa uma estrutura genérica para geração e visualização de relatórios no formato PDF tendo como base o JasperReport.



```

<%@page import="controle.ConnectionFactoryComProperties"%>
<%@page import="net.sf.jasperreports.engine.JasperRunManager"%>
<%@page import="java.io.File"%>
<%@page import="java.util.HashMap"%>
<%@page contentType="text/html" pageEncoding="ISO-8859-1"%>
<%
    try{
        //map para passagem de parâmetros
        HashMap map = new HashMap();
        //endereço para o arquivo jasper
        File reportFile =
            new File(application.getRealPath("/reports/reiCurso.jasper"));
        byte[] bytes = JasperRunManager.runReportToPdf(reportFile.getPath(),
            map, new ConnectionFactoryComProperties().getConnection());
        response.setContentType("application/pdf");
        response.setContentLength(bytes.length);
        ServletOutputStream outStream = response.getOutputStream();
        outStream.write(bytes, 0, bytes.length);
        outStream.flush();
        outStream.close();

    } catch (Exception e) {
        javax.swing.JOptionPane.showMessageDialog(null, "Erro: " +
            e.getMessage());
    }
%>

```

Fechando a série de exemplos, sugerimos que seja acessado, no diretório de exemplos da sessão, o arquivo recursos.jsp, que ao ser executado vai exibir a tela que é reproduzida na figura 6.12, a seguir.



Figura 6.12
Aplicação JSP
Cursos.

Exercícios de fixação

JSP e banco de dados

Qual foi o padrão de projeto utilizado para a construção dos exemplos desta sessão?

Onde deverão estar localizados cada um dos seus recursos dentro da estrutura do projeto?

Atividades Práticas

7

Taglib e JSTL

objetivos

Apresentar aos alunos os mecanismos de desenvolvimento e uso de tags personalizadas no processo de desenvolvimento de aplicações Java para internet, bem como o uso da JSTL no processo de construção de páginas dinâmicas sem a presença de código Java.

conceitos

Taglib; tag personalizada; JSLT e expression language (EL).

Taglib

O grande objetivo de taglib é possibilitar o uso de tags personalizadas entre páginas JSP. As tags personalizadas, implementadas por bibliotecas de tags, foram desenvolvidas com base na sintaxe XML. Isso significa que é possível desenvolver conjuntos de bibliotecas de tags virtualmente ilimitados, pois como são baseadas na sintaxe XML, não fazem restrição alguma quanto ao nome e número de tags que podem ser desenvolvidas.

- Possibilita o uso de tags personalizadas dentro de páginas JSP.
- São implementadas por bibliotecas de tags.
- Desenvolvidas com base na sintaxe XML, mas é diferente de um simples documento XML.
- Não existe restrição alguma quanto ao nome e número de tags que podem ser desenvolvidas.
- Usadas para otimizar o código JAVA, em especial das aplicações JAVA J2EE.
- Vários frameworks utilizam.
 - ▣ Struts, SpringMVC, JSTL etc.



Mas diferentemente de um simples documento XML, as tags personalizadas JSP não são meros elementos de marcação; elas são de fato elementos que implementam ações nas páginas. Podemos concluir ser possível a criação de verdadeiras sublinguagens que fornecerão funcionalidade às páginas JSP.

A diretiva <@taglib @> permite indicar quais bibliotecas de tags poderão ser utilizadas pela página.

- Faz uso dos atributos uri e prefix.

Após ser carregada, uma biblioteca de tags não será carregada novamente.

- O trabalho de carregá-la só será realizado uma única vez pelo contêiner.
- Nas próximas ocasiões em que for necessário utilizá-la, ela já estará disponível.



Uma mesma página JSP pode conter diversas ocorrências da diretiva taglib, significando que várias bibliotecas podem ser carregadas para a mesma página.

Cada ocorrência da diretiva taglib pode carregar uma biblioteca por vez.

Uma tag personalizada é, portanto, um elemento de linguagem JSP definida pelo usuário. O conteúdo da tag personalizada é traduzido em um servlet, sendo a tag convertida para operações em um objeto chamado de manipulador de tag.

O contêiner da web chama essas operações quando o servlet da página JSP é executado.

Utilize arquivos de tag para criar tags JSP personalizadas.

- Autores de arquivos de tag não são obrigados a saber como criar classes Java

Localização:

- Devem ser colocados a seguir do diretório “/WEB-INF/tags”.

Dispensa

- Descritor de biblioteca de classes.

A implementação das tags requer que os arquivos com tags personalizadas sejam colocados a seguir do diretório “/WEB-INF/tags” da aplicação web em desenvolvimento, conforme ilustrado na figura 7.1.

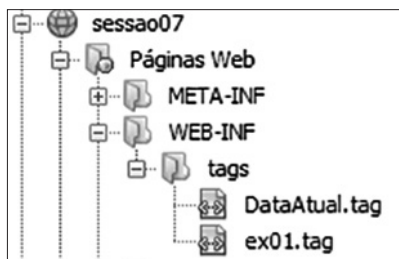


Figura 7.1
Sugestão de estrutura de diretório para aplicações JSP com taglibs.

Depois de configurado o ambiente para que as páginas JSP possam trabalhar com taglibs (tags personalizadas) podemos criar, por exemplo, uma página JSP que faça uso de uma taglib dentro de sua estrutura, como ilustra o trecho de código JSP:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib prefix="exemplo" uri="/WEB-INF/tlds/extag_library.tld" %>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Exemplo TagLib com TLD</title>
  </head>
  <body>
    <exemplo:Mensagem />
  </body>
</html>
```

No exemplo anterior, é preciso definir a tag personalizada <exemplo:Mensagem />. Para tanto, será preciso estender a classe SimpleTagSupport, reescrever o método doTag() e colocar seu código para gerar conteúdo para a tag. Isso pode ser visto no trecho de código a seguir.

```
package com.extaglib;
import java.io.IOException;
```

```
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.SimpleTagSupport;
public class ExemploTagLib extends SimpleTagSupport {

    @Override
    public void doTag() throws JspException, IOException {
        JspWriter out = getJspContext().getOut();
        out.println("Exemplo de classe java como taglib e uso de tld!");
    }
}
```

O próximo passo é a criação do descritor da biblioteca de tags através de um arquivo.tld em WEB-INF/tlds. O trecho do conteúdo do arquivo extag_library.tld pode ser visto a seguir.

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.1" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/
javaee http://java.sun.com/xml/ns/javaee/web-jsptaglibrary_2_1.xsd">
  <tlib-version>1.0</tlib-version>
  <short-name>extag_library</short-name>
  <uri>/WEB-INF/tlds/extag_library</uri>
  <tag>
    <name>Mensagem</name>
    <tag-class>com.extaglib.ExemploTagLib</tag-class>
    <body-content>empty</body-content>
  </tag>
</taglib>
```

O resultado da execução da nova tag personalizada pode ser visualizada na figura 7.2:

Figura 7.2
Nova Tag
Mensagem.



Nesse outro exemplo a seguir, temos o código JSP do arquivo DataAtual.tag com a codificação de uma nova tag personalizada.

```
<%@tag import="java.io.IOException"%>
<%@tag import="java.text.SimpleDateFormat"%>
<%@tag import="java.util.Calendar"%>
<%@ tag pageEncoding="ISO-8859-1"%>
<%
    try {
        String formatoLong = "EEEEEE", dd 'de' MMMM 'de' yyyy";
        SimpleDateFormat formatter = new SimpleDateFormat(formatoLong);
        String horaAtual = formatter.format(Calendar.getInstance().getTime());
        out.print(horaAtual);
    }
```

```

    } catch (IOException e) {
        throw new JspException(e.getMessage());
    }
}
%>

```

Vejamos então como essa tag é referenciada em uma página JSP.

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
    <title>Data Atual</title>
</head>
<body>
    <%@ taglib prefix="aux" tagdir="/WEB-INF/tags" %>
    <aux:DataAtual />
</body>
</html>

```

Nesse exemplo, a diretiva `<@taglib` que é utilizada para informar a localização dos controladores de tags continua a fazer uso do atributo `prefix` através do qual se define uma referência (namespace) para referenciar a tag dentro da página JSP.

Já o atributo `uri` é utilizado para indicar a localização do arquivo da taglib quando este é implementado em Java através de uma subinterface de `"javax.servlet.jsp.tagext.JspTag"`, ou como um tagfile JSP puro (arquivo.tld). Mas no exemplo anterior o atributo `uri` foi substituído pelo atributo `tagdir`, que faz referência a tagfiles JSP disponíveis em `/WEB-INF/tags` (ou subdiretórios) e que não podem ser Classes Java.

De qualquer modo, após ser informada a localização e referência da taglib `DataAtual.tag`, a mesma é referenciada por `<aux:DataAtual />` dentro da página em desenvolvimento. Isso faz com que o fluxo de interpretação seja transferido para a taglib que vai incorporar o seu conteúdo à página JSP em execução.

Com a disponibilização desse mecanismo na linguagem Java, houve uma enorme proliferação de taglibs, destinadas a uma grande variedade de objetivos.

Enorme Proliferação de taglibs.

- Duplicação de funcionalidades.
- Incompatibilidade entre taglibs.

JSTL

JSTL é o acrônimo de Java Server Pages Standard Template Library. E consiste em uma coleção de bibliotecas, tendo cada uma um propósito bem definido.

Java Server Pages Standard Tag Library.

- Desenvolvido pela SUN.
- Coleção de bibliotecas, cada uma com um propósito bem definido.
- Cobrem a maioria das funcionalidades básicas de uma aplicação JAVA J2EE.



Para prover algum grau de padronização, a SUN Microsystem propôs a criação de uma biblioteca padrão que fornece tags prontas para atividades comuns em aplicações web, tais como execução de laços, controle de decisão, formatação de texto, entre outros recursos. Com isso surgiu a JSTL (JSP Standard Tag Library).



Permitem escrever páginas JSP sem scriptlets (sem código Java).

- Aumento da legibilidade do código.
- Interação entre desenvolvedores e web designers.

É usado como framework básico para usar a API nativa da SUN.

Surgiu para padronizar bibliotecas sendo criadas por diversos grupos.

- Atender necessidades de mercado.

Uma página JSTL é uma página JSP contendo um conjunto de tags JSTLs. Cada tag JSTL realizará um determinado tipo de processamento equivalente a um possível código Java dentro de uma página JSP.

API que encapsula funcionalidades para páginas web em tags simples.

- Controle de fluxo.
 - For, if-else.
- Manipulação.
 - Banco de dados, XML, Coleções.
- Internacionalização de aplicações.

Estrutura JSTL

A JSTL está dividida em quatro bibliotecas correspondentes às suas áreas funcionais básicas.

Core:

- Tags destinadas às tarefas comuns (saída, repetição, tomada de decisão e seleção).

Database Access:

- Tags que oferecem acesso a banco de dados.

Formatting:

- Contém tags destinadas à internacionalização e à formatação de dados.

XML processing:

- Destinadas ao processamento de documentos XML.

A tabela 7.1 apresenta a estrutura geral do JSTL com um pouco mais de informação.

Tabela 7.1
Estrutura da
biblioteca JSTL

Biblioteca JSTL	Prefixo	URL	Tipos de uso	Exemplo de tag
core	c	http://java.sun.com /jstl/core	<ul style="list-style-type: none">▪ Acessar e modificar dados em memória▪ Comandos condicionais▪ Loop	<c:forEach>
Processamento de XML	x	http://java.sun.com /jstl/xml	<ul style="list-style-type: none">▪ Parsing (leitura) de documentos▪ Impressão de partes de documentos XML▪ Tomada de decisão baseado no conteúdo de um documento XML	<x:forEach>

Biblioteca JSTL	Prefixo	URL	Tipos de uso	Exemplo de tag
Internacionalização e formatação	fmt	http://java.sun.com /jstl/fmt	<ul style="list-style-type: none"> Leitura e impressão de números Leitura e impressão de datas Ajuda a sua aplicação funcionar em mais de uma língua 	<fmt:formatDate>
Acesso a banco de dados via SQL	sql	http://java.sun.com /jstl/sq	<ul style="list-style-type: none"> Leitura e escrita em banco de dados 	<sql:query>

Tabela 7.1 (cont.)

Estrutura da biblioteca JSTL

Instalação

Alguns servidores de aplicações já trazem a biblioteca JSTL em sua distribuição. Em outros casos, é necessário realizar a instalação da JSTL para que seus web containers JavaEE possam utilizar a JSP.

Instalando o JSTL:

- Fazer o download de uma implementação da JSTL.
 - <http://archive.apache.org/dist/jakarta/taglibs/standard/binaries/>
- Fazer o download do arquivo contendo a implementação.
 - jakarta-taglibs-standard-1.1.2.zip
- Descompacte o arquivo.
- Localize os arquivos jstl.jar e standard.jar na pasta bin e os adicione no projeto em desenvolvimento.
- Vá em "bibliotecas", clique com o botão direito e selecione a opção "adicionar JAR/Pasta".
- Localize os arquivos.jar na sua estrutura de diretório e clique em "abrir".



Expression Language

A EL (expression. Language) foi introduzida na JSTL versão 2.0 e permite a realização de operações aritméticas e lógicas, bem como efetuar a leitura de propriedade de JavaBeans. Seu propósito é facilitar o acesso a informações, pois sua sintaxe é mais simples do que a empregada nas tags-padrão e, usualmente, não requer associação com código Java.

Introduzida na JSTL versão 2.0.

Permite:

- A realização de operações aritméticas e lógicas.
- Efetuar a leitura de propriedade de JavaBeans.

Propósito de facilitar o acesso a informações.

- Sintaxe é mais simples do que a empregada nas tags-padrão.
- Usualmente não requer associação com código Java.

As expressões EL são delimitadas pelos símbolos \${ }.

O conteúdo dessa estrutura é interpretado no momento da execução da página e imediatamente avaliado, ou seja, o valor de qualquer expressão pode ser acessado da seguinte forma: \${<expressão>}.



A tabela 7.2 ilustra o uso dos operadores suportados na EL e o resultado que estes produzem quando aplicados em expressões EL.

Operador	Descrição	Exemplo	Resultado
<code>==</code> <code>eq</code>	Igualdade	<code>\${5 == 5}</code>	true
<code>!=</code> <code>ne</code>	Desigualdade	<code>\${5 != 5}</code>	false
<code><</code> <code>lt</code>	Menor que	<code>\${5 < 7}</code>	true
<code>></code> <code>gt</code>	Maior que	<code>\${5 > 7}</code>	false
<code><=</code> <code>le</code>	Menor ou igual que	<code>\${5 le 5}</code>	true
<code>>=</code> <code>ge</code>	Maior ou igual que	<code>\${5 ge 6}</code>	false
<code>empty</code>	Checa se um parâmetro está vazio	<code>\${user.lastname}</code>	depende
<code>and</code> <code>&&</code>	E	<code>\${param.month == 5 and param.day == 25}</code>	depende
<code>or</code> <code> </code>	OU	<code>\${param.month == 5 or param.month == 6}</code>	depende
<code>+</code>	soma	<code>\${4 + 5}</code>	9
<code>!</code> <code>not</code>	Negação	<code>\${not true}</code>	false

Tabela 7.2
Operadores EL.

Core taglib

Como já vimos, essa taglib inclui tags de propósito geral.

Entre os tags da Core podemos destacar:

- Saída básica.
- Manipulação de variáveis.
- Ações condicionais.
- Estruturas de repetição.
- Manipulação de URLs.
- Inclusão e redirecionamento de páginas.

São identificadas pelo prefixo `c`.

Declaração.

- `<@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>`

Apresentaremos a seguir uma série de exemplos que ilustram o uso de Core taglib na construção de páginas JSP.



Saída básica

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSTL Core - Saída Básica</title>
  </head>
  <body>
    <c:out value="Exemplo JSTL Core - Saída Básica" />
    <br>
    <c:out value="${nome}" default="Nome não foi informado!" />
    <br>
    <c:out value="${15*3}" />
  </body>
</html>
```

Acesso a parâmetros

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSTL Core - Acesso a Parâmetros</title>
  </head>
  <body>
    <form method="POST" action="exJSTL-CoreAcessoParametros.jsp">
      Digite seu nome:
      <input type="text" name="nome" size="50">
      <br>
      <input type="submit" value="Enviar Dados">
    </form>
    <hr>
    Nome digitado: <c:out value="${param.nome}"
                      default="Nenhum nome digitado ainda!" />
    <br>
    Nome digitado: ${param.nome}
    <hr>
    Parâmetros da Página:
    <br>
    ${paramValues}
  </body>
</html>
```

Manipulação de variáveis

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSTL Core - Manipulação de Variáveis</title>
  </head>
  <body>
    <!--declarando e exibindo conteúdo de variáveis -->
    <c:set var="centigrados" value="${param.centigrados}" />
    <br>Graus centigrados: <c:out value="${centigrados}" default="0" />
    <c:set var="fahrenheit" value="${centigrados*5/9 + 32}" />
    <br>Graus Fahrenheit: <c:out value="${fahrenheit}" />

    <!--removendo variáveis -->
    <c:remove var="centigrados" />
    <c:remove var="fahrenheit" />

    <!--Tentativa de exibição de conteúdo de variáveis -->
    <br>Graus centigrados: <c:out value="${centigrados}"
                        default="Variável Removida!" />
    <br>Graus Fahrenheit: <c:out value="${fahrenheit}"
                        default="Variável Removida!" />

  </body>
</html>
```

Decisão simples

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
  <title>JSTL Core - Decisão Simples</title>
</head>
<body>
  <c:if test="${param.idade >= 18 }">
    ${param.nome }, você é maior de idade!
  </c:if>
  <c:if test="${param.idade < 18 }">
    ${param.nome }, você é menor de idade!
  </c:if>
</body>
</html>
```

Decisão múltipla

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
    <title>JSTL Core - Decisão Simples</title>
</head>
<body>
    <%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
    <c:choose>
        <c:when test="${param.numero < 0}">
            0 número ${param.numero} é negativo!
        </c:when>
        <c:when test="${param.numero >= 0}">
            0 número ${param.numero} é positivo!
        </c:when>
        <c:otherwise>
            <!-- trecho de código caso nenhuma tag c:when seja válida-->
        </c:otherwise>
    </c:choose>
</body>
</html>
```

Repetição

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
    <title>Tabuada do número ${param.numero}</title>
</head>
<body>
    <center>Tabuada do número ${param.numero}
    <br><br>
    <c:forEach var="i" begin="1" end="10">
        ${param.numero} * ${i} = ${param.numero * i}<br>
    </c:forEach>
    </center>
</body>
</html>
```

Tokenização

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
<meta http-equiv="Content-Type" content="text/html;">
<title>Manipulação de Tokens</title>
</head>
<body>
    <br><br>
    <ul>
        <c:forEach var="i" delims="," items="Maria,Pedro,Ana,José">
            <li><c:out value="${i}" /></li>
        </c:forEach>
    </ul>
</body>
</html>
```

Inclusão de páginas (import, redirect e catch)

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<!DOCTYPE html>
<c:catch var="excecao">
    <c:import url="/algo.jsp" context="/" />
</c:catch>
<c:if test="${not empty excecao}">
    <c:redirect url="/erro.jsp" />
</c:if>
```

Internacionalização

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ page import="java.io.*,java.util.Locale" %>
<%@ page import="javax.servlet.*,javax.servlet.http.*" %>
<!DOCTYPE html>
<%
    //capturando informações do Locale do cliente
    Locale locale = request.getLocale();
    String language = locale.getLanguage();
    String country = locale.getCountry();
%>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Internacionalização</title>
```

```

    </head>
    <body>
<h1>Informações de localização (Locale)</h1>
        <%
            out.println("Idioma : " + language + "<br />");
            out.println("País : " + country + "<br />");
        %>
        <hr>
    </body>
</html>

```

Internacionalização

```

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ page import="java.io.*,java.util.Locale" %>
<%@ page import="javax.servlet.*,javax.servlet.http.*" %>
<%@ page import="java.text.NumberFormat,java.util.Date" %>
<%
    String title = "Formatação monetária por localidade";
    //Get the client's Locale
    Locale locale = request.getLocale( );
    NumberFormat nft = NumberFormat.getCurrencyInstance(locale);
    String formattedCurr = nft.format(1000000);
%>
<html>
    <head>
        <title><% out.print(title); %></title>
    </head>
    <body>
        <center>
            <h1><% out.print(title); %></h1>
        </center>
        <div align="center">
            <p>Valor monetário formatado: <% out.print(formattedCurr); %></p>
        </div>
    </body>
</html>

```

Formatação

```

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>JSTL - Internacionalização e Formatação </title>
</head>
<body>

```



```

<jsp:useBean id="agora" class="java.util.Date"/>
<br>
Data Versão Curta: <fmt:formatDate value="${agora}" />
<br><br>
Data Versão Longa: <fmt:formatDate value="${agora}" dateStyle="full"/>
<br><br>
Número: <fmt:formatNumber value="12345.6789" pattern="#,##0.00" />
<br><br>
Moeda Corrente: <fmt:formatNumber value="123.67" type="currency" />
<br><br>
Número: <fmt:formatNumber value="123.6" type="number"
minFractionDigits="2" />

</body>
</html>

```

SQL

```

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<!DOCTYPE html>
<sql:setDataSource url="jdbc:postgresql://localhost:5432/sessao08"
driver="org.postgresql.Driver"
user="postgres"
password="123456"
var="dtSource" />
<sql:query var="resultado" dataSource="${dtSource}">
select * from curso order by nome
</sql:query>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JSTL Core - SQL</title>
</head>
<body>
<h1>Listagem de cursos</h1>
<table border='1'>
<tr>
<th>Código</th>
<th>Nome Curso</th>
</tr>
<c:forEach var="reg" items="${resultado.rows}">
<tr>
<th><c:out value="${reg.idcurso}" /></th>
<th><c:out value="${reg.nome}" /></th>
</tr>
</c:forEach>
</table>
<hr>

```

```

        <i>Total de registros: ${resultado.rowCount}</i>
    </body>
</html>

```

SQL com DAO

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Listagem de Cursos Cadastrados</title>
</head>
<body>
    <jsp:useBean id="dao" class="controle.CursorDAO"></jsp:useBean>
    <table border="1" width="40%">
    <c:forEach var="temp" items="${dao.cursos}">
        <tr>
            <td>${temp.idcurso}</td>
            <td>${temp.nomeCurso}</td>
        </tr>
    </c:forEach>
    </table>
</body>
</html>

```

Exercícios de fixação

Taglib e JSTL

Cite as principais vantagens que você consegue identificar no uso de taglib e JSTL no desenvolvimento de aplicações WEB com Java.

Atividades Práticas

8

Cookie e session

objetivos

Apresentar aos alunos os mecanismos para controle dos objetos cookies e session, geralmente utilizados no processo de identificação e permissão de acesso a áreas restritas de aplicações web.

Cookies e sessions.

conceitos

Cookies

Cookie é um mecanismo padrão fornecido pelo protocolo HTTP que permite gravarmos pequenas quantidades de dados persistentes de um usuário no navegador. Tais dados podem ser recuperados posteriormente. Esse mecanismo é usado quando queremos recuperar informações de algum usuário. Com os cookies, podemos reconhecer quem entra em um site, de onde vem, com que periodicidade costuma voltar.

É um mecanismo padrão fornecido pelo protocolo HTTP.

- Permite gravação de pequenas quantidades de dados.
 - ▣ Persistentes no navegador de um usuário.
- Podem ser recuperados posteriormente pelo navegador.

Uso:

- Recuperar informações de algum usuário.
- Reconhecer quem entra em um site, de onde vem, com que periodicidade costuma voltar.

Para se ter uma ideia de como eles fazem parte da sua vida, dê uma olhada na sua máquina.

Se você usa o Internet Explorer no Windows 7, vá à pasta "C:\Users\Usuário\AppData\Roaming\Microsoft\Windows\Cookies". No Firefox, os cookies ficam dentro de um lugar chamado pasta do perfil. Ela é diferente da pasta onde o Firefox está instalado. Para visualizar os cookies, vá em "Ajuda" > "Dados para Suporte". Na página que abrirá, clique em "Abrir Pasta". Eis os arquivos do seu perfil. Eles armazenam suas extensões, cookies, senhas, favoritos, histórico e preferências. Os cookies estão armazenados no arquivo cookies.sqlite.



Onde estão armazenados?

- IE 7:
 - Na pasta “C:\Users\Usuario\AppData\Roaming\Microsoft\Windows\Cookies”
- Firefox:
 - Dentro de um lugar chamado pasta do perfil (diferente da pasta onde o Firefox está instalado).
 - Para visualizar, acesse “Ajuda” > “Dados para Suporte” e clique em “Abrir Pasta”.
 - Armazenam suas extensões, cookies, senhas, favoritos, histórico e preferências.
 - Os cookies estão armazenados no arquivo cookies.sqlite.

Os cookies em si não atrapalham ninguém, se propriamente usados. Como padrão, os cookies expiram tão logo o usuário encerra a navegação naquele site. Porém, podemos configurá-los para persistir por vários dias. Além dos dados que ele armazena, um cookie recebe um nome, através do qual pode ser posteriormente identificado. Um servidor pode definir múltiplos cookies, sendo que cada cookie é associado à URL da página que o manipula.

Expiram tão logo o usuário encerra a navegação no site, mas podem ser configurados para persistir por vários dias.

São definidos por um servidor da web.

- Recebem um nome.
- Associados à URL da página que os manipulam.

São enviados de volta para o servidor quando o usuário solicita a URL a qual estejam associados.

Quando um usuário solicita uma URL cujo servidor e diretório correspondam àqueles de um ou mais de seus cookies armazenados, os cookies correspondentes são enviados de volta para o servidor. As páginas JSP acessam os seus cookies associados através do método `getCookies()` do objeto implícito `request`. De forma similar, as páginas JSP podem criar ou alterar cookies através do método `addCookie()` do objeto implícito `response`. Esses métodos são resumidos na tabela a seguir:

Tabela 8.1
Métodos para gerenciamento de cookies.

Objeto implícito	Método	Descrição
Request	<code>getCookies()</code>	Retorna uma matriz de cookies acessíveis da página.
response	<code>addCookie()</code>	Envia um cookie para o navegador para armazenagem/modificação.

A classe cookie

Manipulamos um cookie através de instâncias da classe `javax.servlet.http.Cookie`. Essa classe fornece apenas um tipo de construtor que recebe duas variáveis do tipo `String`, que representam o nome e o valor do cookie.

Manipulação de um cookie.

Instância da classe `javax.servlet.http.Cookie`.

- Fornece apenas um tipo de construtor.
- Recebe duas variáveis do tipo `String`, representando o nome e o valor do cookie

Sintaxe.

- `Cookie cookie = new Cookie("nome do cookie", "valor do cookie");`

A seguir, apresentamos os métodos fornecidos pela classe cookie:

Método	Descrição
getName()	Retorna o nome do cookie
getValue()	Retorna o valor armazenado no cookie
getDomain()	Retorna o servidor ou domínio do qual o cookie pode ser acessado
getPath()	Retorna o caminho de URL do qual o cookie pode ser acessado
getSecure()	Indica se o cookie acompanha solicitações HTTP ou HTTPS
setValue()	Atribui um novo valor para o cookie
setDomain()	Define o servidor ou domínio do qual o cookie pode ser acessado
setPath(nome do path)	Define o caminho de URL do qual o cookie pode ser acessado
setMaxAge(inteiro)	Define o tempo restante (em segundos) antes que o cookie expire
setSecure(nome)	Retorna o valor de um único cabeçalho de solicitação como um número inteiro

Tabela 8.2
Métodos da
classe cookie.

Depois de construir uma nova instância ou modificar uma instância recuperada através do método `getCookies()`, é necessário usar o método `addCookie()` do objeto `response`, com a finalidade salvar no navegador do usuário as alterações feitas no cookie.

Para apagar um cookie, utilizamos a seguinte técnica: chamamos o método `setMaxAge(0)` com valor zero e depois mandamos gravar chamando o método `addCookie()`. Isso faz com que o cookie seja gravado e imediatamente expire (após zero segundos).

Exemplos de uso de cookies

O primeiro passo para usar um cookie dentro de uma página é defini-lo. Isso é feito criando uma instância da classe `cookie` e chamando os métodos `sets` para definir os valores de seus atributos. O script JSP a seguir apresenta o código de uma página JSP utilizada para gerar cookies.

```
<%
    String email = request.getParameter("email");
    String cookieName = "Exemplocookie";

    Cookie Excookie = new Cookie(cookieName, email);
    //define o tempo de vida como 7 dias (604800 segundos)
    Excookie.setMaxAge(7 * 24 * 60 * 60);
    //versão 0 da especificação de cookie
    Excookie.setVersion(0);
    //indica que o cookie deve ser transferido pelo protocolo HTTP padrão
    Excookie.setSecure(false);
    //insere um comentário para o cookie
    Excookie.setComment("Email do visitante");
    //grava o cookie na máquina do usuário
    response.addCookie(Excookie);
%>
```

O código da página HTML a seguir possui um formulário com chamada através de um input submit ao script JSP que vai criar o cookie.

```
<html>
  <body>
    <form action="criacookie.jsp">
      <input type="text" name="email">
      <input type="submit" value="ok">
    </form>
  </body>
</html>
```

O código JSP, visto anteriormente, tem a finalidade de receber um valor (email) passado através de um formulário de uma página HTML. Esse valor é armazenado de forma persistente em um cookie e pode ser acessado por outras páginas JSP que compartilham o domínio e o caminho originalmente atribuído ao cookie.

O trecho de código a seguir demonstra a recuperação do valor de um cookie.

```
<%
String cookieName = "Exemplocookie";

Cookie listaPossiveisCookies[] = request.getCookies();
Cookie Excookie = null;

if (listaPossiveisCookies != null) {
//Se não existem cookies associados o método getCookies() retorna null
    int numCookies = listaPossiveisCookies.length;

    for (int i = 0 ; i < numCookies ; ++i) {
        //procura pelo cookie
        if (listaPossiveisCookies[i].getName().equals(cookieName)) {
            Excookie = listaPossiveisCookies[i];
            break;
        }
    }
}
%>
```

Esta página HTML demonstra o resultado obtido quando da recuperação do cookie:

```
<html>
<body>
  <h1>Lê Cookie</h1>
  <% if (cookieJSP != null) { %>
    A pagina "criacookie.jsp" gravou o email: <%= Excookie.getValue() %>
  <% }
  else { %>
    O cookie não gravou ou o prazo do cookie expirou.
  <% } %>
</body>
</html>
```

Considerações finais sobre cookies

Apesar da praticidade de se utilizar os cookies oferecidos pelo protocolo HTTP, devemos fazer algumas considerações quanto à sua utilização.

O tamanho dos dados armazenados (nome e valor) não devem ultrapassar 4K.

O navegador pode armazenar múltiplos cookies.

- Armazena até 20 cookies por configuração de domínio.
- Armazena 300 cookies em geral.
- Se qualquer um desses dois limites forem alcançados, espera-se que o navegador exclua cookies menos utilizados.

Configuração do domínio atribuído a um cookie.

- Método `setDomain(nome do domínio)`.
- Quando nenhum domínio é especificado na sua criação, o cookie só poderá ser lido pelo host que originalmente o definiu.



Session

Session é um mecanismo padrão fornecido pelo protocolo HTTP que permite gerenciar o acesso a páginas restritas de uma determinada aplicação web. Também serve para manter objetos de classes de objetos dentro das páginas HTML. Para que o controle de páginas restritas tenha maior segurança, é preciso que se configure um tempo máximo para que esta expire. Isso dará a uma determinada aplicação a segurança de acesso a páginas restritas e, quando estas forem acessadas, terão um tempo limite de controle da sua não utilização. Caso esse tempo seja ultrapassado, o usuário será obrigado a acessá-la novamente. Esse acesso geralmente é feito através de páginas de login.

Mecanismo padrão fornecido pelo protocolo HTTP.

- Permite:
 - Gerenciar o acesso a páginas restritas.
 - Manter objetos de classes de objetos dentro das páginas.
- Para que o controle de páginas restritas tenha maior segurança:
 - Configurar tempo máximo para que ela expire.
 - Caso o tempo limite seja ultrapassado, o usuário será obrigado a acessá-la novamente.
 - Acesso geralmente feito por meio de páginas de login.



O objeto session representa a sessão atual de um usuário individual. Todas as solicitações feitas por um usuário são consideradas parte de uma sessão. Desde que novas solicitações por aqueles usuários continuem a ser recebidas pelo servidor, a sessão persiste. Se, no entanto, um certo período de tempo passar sem que qualquer nova solicitação do usuário seja recebida, a sessão expira.

Representa a sessão atual de um usuário individual.

- Todas as solicitações feitas por um usuário são consideradas parte de uma sessão.

Persistente:

- Desde que novas solicitações do usuário continuem a ser recebidas pelo servidor



Expira:

- Se um certo período de tempo passa sem que qualquer nova solicitação do usuário seja recebida.

O objeto `session` armazena informações a respeito da sessão. Um dos principais usos para o objeto `session` é armazenar e recuperar valores de atributos, a fim de transmitir as informações específicas de usuários entre as páginas.

A seguir, um exemplo que armazena dados na sessão, na forma de um objeto que é instância de uma classe hipotética `IdSessao`.

```
<%@ page import="IdSessao" session="true"%>
<%! IdSessao idsession = new IdSessao(); %>
<%
    session.setMaxInactiveInterval(5);
    idsession.setIdSessao("123456789");
    pageContext.setAttribute("atSession", idsession, pageContext.SESSION_SCOPE);
%>
```

No exemplo anterior, estamos utilizando a diretiva `page` para importar a classe `IdSessao` e também fazemos o uso de atributo `session` com o valor igual a `TRUE`. Esse atributo especifica que a página faz parte da sessão atual. Caso ainda não exista uma sessão, uma nova será criada. Em seguida, ocorre a declaração de um objeto da classe `IdSessao` e depois são definidos valores para esse objeto.

O método `setMaxInactiveInterval()` define, em segundos, o tempo máximo que a sessão pode permanecer inativa. A sessão será perdida caso a página fique inativa por mais tempo do que o informado pelo método `setMaxInactiveInterval()`.

Vejamos agora o exemplo de código de um formulário de login.

```
<form method='post' action='validaUsuario'>
    <table width="30%" border="0" align="center">
        <tr>
            <td colspan="2" align="center">Efetuar Logon</td>
        </tr>
        <tr>
            <td width="40%">Usuário:</td>
            <td width="60%"><input type="text" name="usuario"></td>
        </tr>
        <tr>
            <td width="40%">Senha:</td>
            <td width="60%"><input type="password" name="senha"></td>
        </tr>
        <tr>
            <td colspan="2" align="center">
                <input type="submit" name="b1" value="Efetuar Logon">
            </td>
        </tr>
    </table>
</form>
```

Já no trecho de código a seguir, podemos ver o servlet de autenticação da sessão.



Lembramos que o intervalo máximo de inatividade padrão é de 15 minutos, que equivale a 900 segundos. Isso não quer dizer que não se possa trabalhar com um tempo maior.


```

@WebServlet(name="/LoginUsuario", urlPatterns="/validaUsuario")
public class LoginUsuario extends HttpServlet {

    /** ... */
    public LoginUsuario() {

        /** ... */
        protected void doGet(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOException {

            /** ... */
            protected void doPost(HttpServletRequest request,
                HttpServletResponse response) throws ServletException, IOException {
                // TODO Auto-generated method stub
                UsuarioDAO dao = new UsuarioDAO();
                Usuario u = dao.validarUsuario(request.getParameter("usuario"),
                    request.getParameter("senha"));
                if (u != null){
                    request.getSession().setMaxInactiveInterval(30);
                    request.getSession().setAttribute("atUsuario", u);
                    response.sendRedirect("menu.jsp");
                }
                else
                    response.sendRedirect("login.jsp?msg=Erro ao tentar efetuar logon!
                        Favor verificar usu rio e senha.");
            }
        }
    }
}

```

O exemplo de p gina JSP a seguir ilustra como podemos desenvolver p ginas que tenham controle de acesso por sess o. Nesse exemplo, a p gina espera por um objeto j  definido para a sess o atual, ou seja, ela espera que seja enviado um objeto da classe `IdSessao` para que este seja validado, dando, assim, acesso aos recursos dessa p gina. Uma mensagem de erro ser  retornada para o usu rio caso este tente o acesso direto a essa p gina. Isso ocorrer  devido a especifica  o do atributo `errorPage`, da diretiva `page`, onde se define a p gina que ser  retornada caso ocorra algum erro durante o processamento dos scriptlets.

```

<%@ page import="IdSessao" session="true"%>
<%@ page errorPage="erro.htm"%>
<html>
<head>
    <title>Teste de validade da sess o</title>
</head>
<body>
<%
    IdSessao atRecuperado = new IdSessao();
    String ID = null;
    atRecuperado = (IdSessao)pageContext.getAttribute("atSession",
                                                pageContext.SESSION_SCOPE);
    ID = atRecuperado.getIdSessao();

```

```

if (ID == null) { %>
    A sessão foi perdida.<br>
    <a href="ExemploJSP15.jsp">Voltar</a>
<% } else { %>
    Dados da sessão recuperada a partir do escopo SESSION:<br>
    ID: <%= ID %><br>
    Tempo máximo de vida desta sessão:<%= session.getMaxInactiveInterval()%>
<% } %>
</body>
</html>

```

Finalmente, apresentamos trecho do arquivo ListarCursos.jsp, onde é feita a validação de uma sessão.

```

<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<c:if test="${empty sessionScope.atUsuario.nome}">
    <c:redirect
        url="login.jsp?msg=Sessão expirou! Fazer login novamente.">
    </c:redirect>
</c:if>
<jsp:useBean id="dao" class="controle.CursorDAO"></jsp:useBean>
<center>Listagem de Cursos Cadastrados</center>
<hr>
<table border="0" width="60%" align="center">
    <thead>
        <tr>
            <td>Código</td>
            <td>Nome</td>
        </tr>
    </thead>
    <c:forEach var="temp" items="${dao.cursos}">
        <tr>
            <td>${temp.idcurso}</td>
            <td>${temp.nomeCurso}</td>
        </tr>
    </c:forEach>
</table>
<hr>
<a href="menu.jsp">Voltar</a>

```

Exercício de fixação

Cookie e session

Cite as principais diferenças entre uso de cookies e sessions que você consegue identificar.

Atividades Práticas

9

Javabeans

objetivos

Apresentar aos alunos os mecanismos de criação e uso de componentes no processo de desenvolvimento de software e o uso desses em ferramentas RAD.

conceitos

Componente de software, bean, API e JAR.

Componentes de software

Peças de software.

- Aderem a uma especificação definida.
- Podem ser distribuídos e usados em diversas aplicações.

Transformar classe em componente.

- Adicionar algumas funcionalidades.
- Respeitar convenções.

São interligados de modo a construir uma aplicação.

- Desenvolvidos com um objetivo específico.
- Não para uma aplicação específica.



A definição oficial de bean, conforme dada na especificação JavaBeans, é: “Um bean é um componente de software reutilizável, baseado na especificação JavaBeans da Sun, que pode ser manipulado visualmente em uma ferramenta construtora.”

Modelo.

- Define um conjunto de interfaces e classes.
 - Serão utilizadas de forma determinada.
- Especifica como serão expostos.
 - Propriedades, métodos e eventos.
- JavaBean.
 - Para construção de aplicação de forma visual.
- Enterprise JavaBeans (EJB).
 - Modelo de componentes para arquitetura distribuída.



- JavaBean e EJB.
 - ▣ Os modelos não são relacionados.
 - ▣ EJB não é uma extensão de JavaBean.



Os programadores com experiência em Windows saberão imediatamente por que os beans são tão importantes. Atualmente, as ferramentas RAD (Rapid Application Development) são um bom ponto de apoio para o desenvolvimento de aplicações nas principais linguagens visuais existentes no mercado. Alguns exemplos podem ser citados, como Visual Studio, Delphi, VisualAge, Android Studio, entre outros.

Nestas ferramentas você tem a possibilidade de trabalhar com componentes já desenvolvidos, em sua maioria pelo próprio fornecedor da linguagem, através do processo de construção de interfaces soltando componentes sobre a aplicação em construção. Cada um desses componentes já traz consigo suas particularidades a serem trabalhadas, tais como características visuais ou eventos que respondem a estímulos do usuário (clique do mouse, digitação no teclado etc.).

JavaBeans

JavaBeans é o nome de um projeto JavaSoft para definir um conjunto de componentes de software padrão acessíveis através de uma API (Interface de Programação de Aplicativos) para a plataforma Java.

Nome de um projeto JavaSoft.

Define um conjunto de componentes de software padrão.

- Reutilizáveis.

API (Interface de Programação de Aplicativos) para a plataforma Java.

- Facilitam aos desenvolvedores a ligação entre modelos de componentes existentes.
 - ▣ ActiveX da Microsoft, OpenDoc da Apple...

JavaBeans é a tecnologia Java que lhe permite escrever programas Java e “convertê-los” em Beans que poderão ser utilizados por programadores em suas atividades de desenvolvimento de aplicativos.

Descreve uma arquitetura.

- Componentes para Java baseado em classes Java.
- Definem regras.
 - ▣ Possibilita a criação de classes configuráveis e reutilizáveis.
 - ▣ Possibilita o uso por diferentes ferramentas.
- Definem classes e métodos.
 - ▣ Possibilita a obtenção de informações sobre os componentes.
- Padrões definem convenções.
 - ▣ Nomes de atributos e métodos.



Uma vez que você tenha construído um bean, os usuários de qualquer ambiente compatível com a tecnologia JavaBeans podem usá-lo prontamente. Eles podem inclusive ser usados em outros ambientes, como o Visual Studio ou o Delphi, através da Ponte Beans para ActiveX (Beans to ActiveX Bridge).



Características:

- Baseados em programas Java “normais”.
- Bean pode ser visto como um “componente” de software.
- Podem ser incorporados em ambientes RAD que suportam JavaBeans.
- Padronizar.
 - ▣ Nomes de métodos para aceitação comercial (getters e setters).
 - ▣ Nome da classe sempre com a primeira letra maiúscula.
 - ▣ Nome de métodos com a primeira letra minúscula.
 - ▣ Atributos com a primeira letra minúscula.
 - ▣ Documentação de todas as funcionalidades (javadoc).
- Acessível para usuários não especialistas através de ferramentas visuais.
- Possibilidade de uso em uma ampla variedade de contextos.
 - ▣ Local, distribuída, web, dispositivos móveis, objetos visuais gráficos etc.
- Ter um construtor padrão.
- Ser serializável.

Escrever beans em Java não é uma tarefa tecnicamente difícil. Existem apenas algumas classes e interfaces novas para serem dominadas. Em particular, o tipo mais simples de bean nada mais é do que uma classe da plataforma Java que segue algumas convenções de atribuição de nomes bastante restritas.

Por outro lado, como um mesmo bean pode ser usado em uma ampla variedade de contextos, eles podem se tornar bastantes complexos. Por exemplo, um controle gráfico popular em média apresenta 60 propriedades, 14 métodos, 47 eventos e 178 páginas de documentação. Além disso, os beans devem ter um construtor padrão (ou nenhum construtor no caso em que um construtor padrão é fornecido automaticamente) e devem ser serializáveis. Isso permite aos ambientes instanciar novos beans e salvar beans entre sessões.

A seguir, mostramos um trecho de programa que corresponde a um código Java para um bean.

```
import java.io.*;
import java.swing.*;

public class TesteBean implements Serializable{
    private String filename = "";

    public void setFileName(String f){
        filename = f;
    }

    public String getFileName(String f){
        return filename;
    }
}
```



A classe TesteBean é um exemplo de bean onde podemos citar algumas convenções de atribuição de nomes, principalmente para métodos pertencentes às classes. Nesse caso, a classe possui um atributo, o filename, que precisa ser acessado pelo programador que estará utilizando-a. Para tanto, foram implementados os métodos getFileName() e setFileName() que, respectivamente, vão retornar ou atribuir o nome de um arquivo informado ao atributo filename. Note também que a classe implementa a interface Serializable, respeitando a necessidade de ser um objeto serializável para poder ser utilizado em mais de uma sessão.

Iremos, agora, preparar a classe TesteBean para ser um JavaBean. A princípio, faremos algumas modificações que permitiram utilizar a classe como um JavaBean. Primeiro, vamos adicionar uma instrução package (linha 1) ao arquivo TesteBean.java. Normalmente, as classes que representam um bean são colocadas em pacotes. A segunda modificação é a implementação da interface Serializable (linha 3), já mencionada anteriormente. Como resultado, temos a classe TesteBean, apresentada a seguir:

```
package aulabeans;

import java.io.*;

public class TesteBean implements Serializable {
    private String filename="";

    public TesteBean(){
        this.setFileName("teste.txt");
        System.out.println("Nome do Arquivo: "+this.getFileName());
    }

    public void setFileName(String f){
        filename = f;
    }

    public String getFileName(){
        return filename;
    }

    public static void main(String args[]){
        TesteBean tb = new TesteBean();
    }
} // End of Class TesteBean
```

Agora temos de compilar a classe TesteBean.java. A instrução de compilação é:

```
javac -d. TesteBean.java
```

A opção -d indica que o arquivo .class deverá ser criado dentro do diretório de pacotes de classes criado. A opção "." representa o diretório em que o pacote ordenador deve ser colocado, sendo, nesse caso, o diretório atual. Como resultado da compilação, teremos um subdiretório na pasta onde esse arquivo TesteBean.java se encontra com o nome do package informado no arquivo TesteBean.java. Esse diretório vai conter o arquivo TesteBean.class, indicando que esse é um diretório de pacotes de classes Java.

Criando um JavaBean

Para utilizar uma classe como um JavaBean, ela deve primeiro ser colocada em um Java ArchiveFile (arquivo JAR).

Classes beans.

- Organizadas em pacotes de classes (package).

Implementam a interface Serializable.

Criar o arquivo “.jar” correspondente.

- Utilitário JAR do JDK.
- Descreve o conteúdo do arquivo JAR.

Criar arquivo de manifesto.

- manifest.tmp

O primeiro passo seria criar um arquivo chamado manifest.tmp. Esse arquivo é utilizado pelo utilitário jar para descrever o conteúdo do arquivo JAR. Quando um arquivo JAR contendo um JavaBean é carregado em um IDE, este examina o arquivo de manifesto para determinar as classes no JAR que representam JavaBeans e disponibilizá-las para o programador de uma maneira visual. As instruções do arquivo de manifesto manifest.tmp são:

```
Main-Class: aulabeans.TesteBean
Name: aulabeans\TesteBean.class
Java-Bean: True
```

arquivo manifest.tmp

- Main-Class
 - Descreve a classe principal do arquivo JAR.
 - Método main() quando necessário.
- Name
 - Especifica o nome do arquivo contendo a classe bean.
 - Incluir a extensão .class
- Java-Bean
 - indica se a classe indicada por Name é, de fato, um JavaBean.
 - True | False

A seguir, criamos o arquivo JAR para o bean TesteBean através do utilitário jar com a seguinte linha de comando:

```
jar cvfm ButtonRelatorio.jar relatorio\manifest.tmp relatorio\*.*
```

Gerar o arquivo .jar

- Utilizar o utilitário jar do JDK.
- No comando, cvfm:
 - ▣ criação de um arquivo JAR.
 - ▣ verbose – mostra na tela o que está acontecendo.
 - ▣ file, o nome do arquivo JAR a ser criado.
 - ▣ manifest, o arquivo de manifesto.
 - ▣ Utilizado para a geração do MANIFEST.MF
 - ▣ Incluído no diretório META-INF do arquivo JAR

As diretiva v, de verbose, e t, de table, são usadas para visualizar o conteúdo de um arquivo .jar qualquer, podendo ser usadas juntamente com outras diretivas. Ainda no comando anterior, relatorio\manifest.tmp é o caminho/nome do arquivo .tmp, enquanto que relatorio*. * indica que todos os arquivos nesse diretório devem ser incluídos no arquivo JAR. Na figura 9.1, podemos ver a estrutura dos arquivos gerados após a execução do comando.



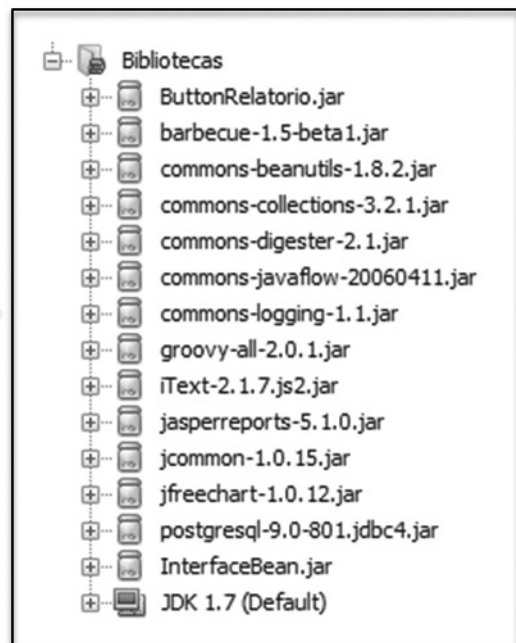
Figura 9.1
Estrutura de
diretórios após
criação do
arquivo.jar.

Podemos utilizar pacotes de classes externas.

- Devem ser disponibilizados no projeto em construção.
 - ▣ Importar .jar em bibliotecas.
 - ▣ JavaBean e classes externas.



Figura 9.2
Pacotes externos
importados
para o diretório
'Bibliotecas'.



JavaBeans no IDE (Netbeans)

O uso de JavaBeans no Netbeans pode ser configurado através de Ferramentas/Paletas/Componentes Swing/AWT, conforme pode ser visto na figura 9.3.

Arquivo .jar do JavaBean

- Pelo gerenciador de paletas.
 - ▣ Adicionar.
 - ▣ Remover.
- Em qualquer patela.
- Criar novas.
- Preferencialmente.
 - ▣ Paleta Beans.



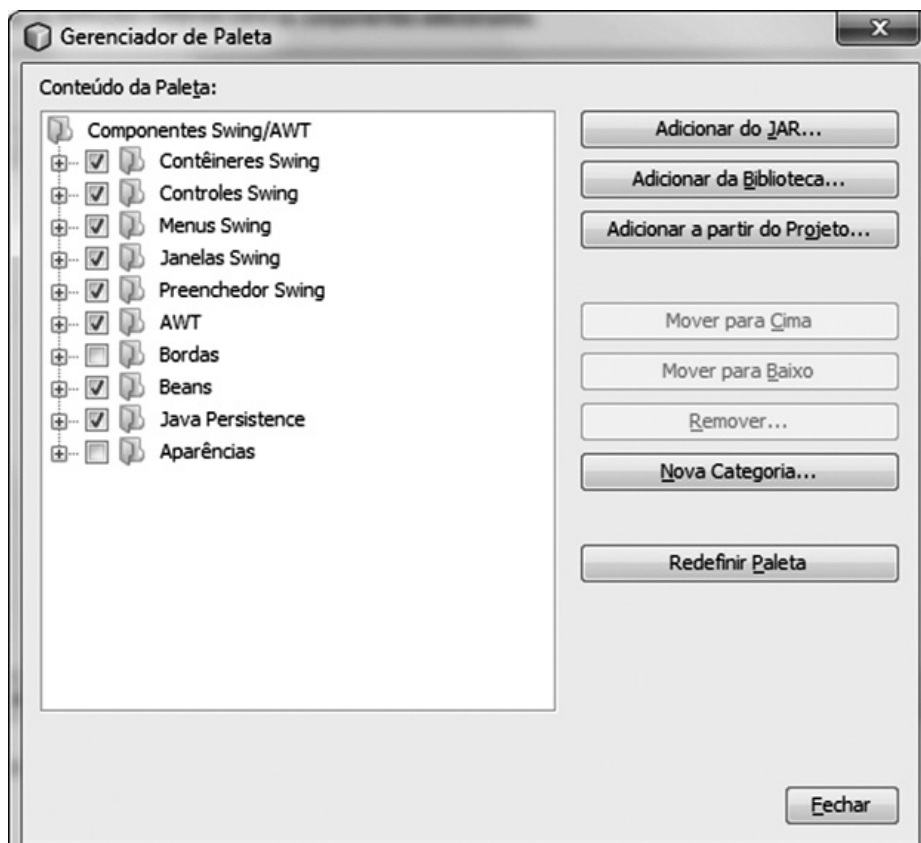


Figura 9.3
Gerenciador
de paletas do
Netbeans.

JSP com JavaBeans

O atributo ação da tag page possibilita a realização de tarefas complexas. Embora se obtenha resultados equivalentes com o uso de código Java dentro de scriptlets, o uso das tags de ações promove reusabilidade de seus componentes e aumenta a capacidade de manutenção da página.

Uso do atributo ação da tag `<@page />` permite:

- Instanciação de objetos.
- Comunicação com recursos do lado do servidor como páginas JSP e servlets.
- Sem a necessidade de código Java.

Resultado equivalente com o uso de código Java dentro de scriptlets.

Uso das tags de ações.

- Promove reusabilidade de seus componentes.
- Aumenta a capacidade de manutenção da página.

A tecnologia de componentes para JSP é baseada em componentes JavaBeans. Antes de acessar um bean dentro de uma página JSP, é necessário identificar o bean e obter uma referência dele.



Tecnologia de componentes para JSP.

- Baseada em componentes JavaBeans.
- Identificar o bean.
 - Antes de acessá-lo dentro de uma página JSP.
 - Necessário identificar e obter sua referência.
- tag `<jsp:useBean />`
 - Tenta obter uma referência de um bean que já esteja criado.
 - Depende do escopo em que ele foi definido.
 - `scope="page | request | session | application"`

Por exemplo:

```
<jsp:useBean id="user" class="Global" scope="session"/>
```

id

- `id="beanInstanceName"`

class

- `class="package.class"`

scope

- `scope="page | request | session | application"`

beanName

- `beanName="{package.class | <%= expression %>}"`

Type

- `type="package.class"`

A tag acima instanciará um objeto que será identificado por user, a partir da classe Global. A definição do escopo depende muito do projeto em questão. No exemplo anterior, esse objeto criado será compartilhado por toda sessão que será mantida com o usuário. Caso encontre-se uma nova tag de criação de um objeto da classe Global, o objeto já instanciado será usado, não havendo a necessidade de criação de outro.

Passos para a criação do objeto:

- Tentativa de localizar o objeto.
- Baseado nas informações (id, scope).
- Inspeção é feita sincronizada.
- Definir a variável de script identificado por id.
- Objeto encontrado.
 - A variável é inicializada com a referência à localizada e então é realizado um casting para o tipo específico.
- Se o Cast falhar:
 - Uma exceção do tipo `java.lang.ClassCastException` vai ocorrer e o processamento da tag acaba.

- Objeto não encontrado no escopo e a classe não pode ser localizada:
 - ▣ Uma exceção do tipo `java.lang.InstantiationException` vai ocorrer e o processamento da tag acaba.
- Objeto não encontrado no escopo e a classe for encontrada com um construtor padrão (sem argumentos):
 - ▣ O objeto é instanciado e relacionado com a variável de scriptlet identificada por id.
- Se um construtor sem argumentos não for encontrado:
 - ▣ Então uma exceção do tipo `java.lang.InstantiationException` ocorrerá e o processamento da tag acabará.



Caso o processamento do corpo da tag venha a ser realizado, ele poderá realizar algumas inicializações desejadas, como no exemplo a seguir, onde o objeto `clientes` é instanciado com as propriedades `nome` e `telefone`.

```
<jsp:useBean id="clientes" class="Clientes" scope="page">
<jsp:setProperty name="clientes" property="nome" value="Rafael dos Santos">
<jsp:setProperty name="clientes" property="telefone" value="648-2343"/>
</jsp:useBean>
```

Cabe entender melhor os escopos possíveis para a definição de um bean. São eles:

- **page:** objetos são vistos apenas pela página onde ele foi criado. As referências para esses objetos são armazenados no objeto `pageContext`.
- **request:** objetos são vistos nas páginas usadas para responder a requisição do usuário. Se uma página é redirecionada para outra, esses objetos são preservados, uma vez que fazem parte da mesma requisição. As referências para esses objetos são armazenadas no objeto implícito `request`.
- **session:** objetos são acessíveis por páginas que fazem parte da mesma sessão. As referências para esses objetos são armazenadas no objeto `session`.
- **application:** objetos são acessíveis por toda aplicação JSP em questão e são armazenados no objeto implícito `application`.



Vejamos um exemplo de uso de JSP com JavaBeans. No código JSP a seguir, é realizada a chamada a um `JavaBean` que é a classe `Global`. Através da tag `<jsp:useBean>` é criado um objeto chamado `user`, que será utilizado ao longo do código para realizar chamadas aos métodos implementados para essa classe. Podemos citar a linha do comando `user.checkUser()`; como exemplo de execução de um método da classe `Global` dentro do código JSP.

```
<html>
<head>
  <title>Página Principal</title>
</head>

<%@ page import = "Global" %>

<jsp:useBean id="user" scope="page" class="Global" />
<jsp:setProperty name="user" property="login" />
<jsp:setProperty name="user" property="password" />

<body>
```

Nesse outro exemplo, começamos mostrando o código HTML de um formulário de entrada de dados.

O código do bean a ser utilizado é o seguinte.

Capítulo 9 - Javabeans



```

</head>
<body>
    <h1>Informações do objeto instanciado do Bean</h1>
    <hr>
    Nome: ${pessoa.nome}
    <br>
    Sexo: ${pessoa.sexo}
    <hr>
    <a href="ex01UseBean.html">Voltar</a>
</body>
</html>

```

Uma alternativa é utilizar o bean com atribuição genérica, conforme podemos ver no trecho de código a seguir.

```

<%@page contentType="text/html" pageEncoding="ISO-8859-1"%>
<jsp:useBean id="pessoa" class="modelo.Pessoa" />
<jsp:setProperty name="pessoa" property="*" />
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
        <title>JSP e JavaBean</title>
    </head>
    <body>
        <h1>Informações do objeto instanciado do Bean</h1>
        <hr>
        Nome: ${pessoa.nome}
        <br>
        Sexo: ${pessoa.sexo}
        <hr>
        <a href="ex02UseBean.html">Voltar</a>
    </body>
</html>

```

Exercício de fixação

JavaBeans

Para você, qual é a principal característica e como JavaBeans pode auxiliar no processo de desenvolvimento de softwares?

Atividades Práticas

10

Deployment de aplicação Java Web

objetivos

Apresentar aos alunos os passos necessários para a organização de um projeto de aplicação web a ser colocado em ambiente de produção no servidor web Apache TomCat.

Deployment.

conceitos

Apache Tomcat

Neste curso, vamos apresentar o Deployment de aplicação Java WEB no Apache Tomcat. O Apache Tomcat Servlet/JSP Container é uma implementação rápida e eficiente da especificação dos servlets, de um web container que suporta servlets e aplicações JSP ou JSF.

É oriundo do projeto Jakarta da ASF (Apache Software Foundation), originalmente criado para desenvolver tecnologias server-side baseadas na plataforma Java, tendo recebido um grande impulso quando a Sun resolveu doar sua implementação de referência dos servlets e JSP.

Tomcat Servlet/JSP Container.

- Implementação rápida e eficiente da especificação dos servlets.
- Web container que suporta servlets e aplicações JSP ou JSF.
 - ▣ Composto por três elementos principais:
- **Catalina:** web container que fornece a infraestrutura para a execução e gerenciamento dos servlets.
- **Jasper:** compilador que transforma páginas JSP em servlets equivalentes, compilando-os em bytecode.
- **Connectors:** elementos de conexão entre o Tomcat e outros servidores.
 - ▣ Possibilita integração com outros servidores web.
 - ▣ Apache HTTPd ou MS IIS.

A instalação do Tomcat é bastante simples e pode ser realizada seguindo o roteiro disponível no AVA.



Após a instalação, de modo a verificar o correto funcionamento do Tomcat, acesse o seguinte endereço através do navegador/browser:

```
http://localhost:8080
```

Se estiver tudo Ok, deverá ser exibida a tela reproduzida na figura 10.1, a seguir.

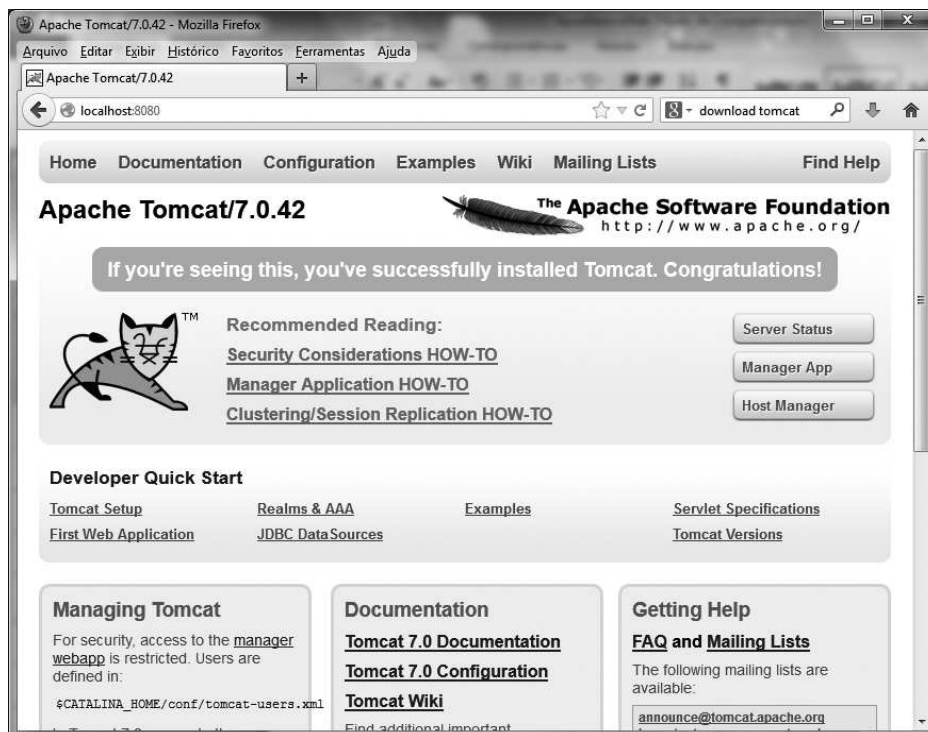


Figura 10.1
Página inicial
do Tomcat.

O gerenciamento das aplicações hospedadas pelo Tomcat pode ser feito através do manager webapp, acessível através do link que aparece no canto inferior esquerdo na figura 10.1, logo a seguir do título "Managing Tomcat". Essa aplicação pode ser também acessada diretamente através da URL:

```
http://localhost:8080/manager/html
```

Qualquer que tenha sido a opção escolhida, caso tenha sido definido um login/senha para o administrador do Tomcat durante o seu processo de instalação, essas informações terão de ser corretamente fornecidas para que o sistema exiba a tela que é reproduzida na figura 10.2.

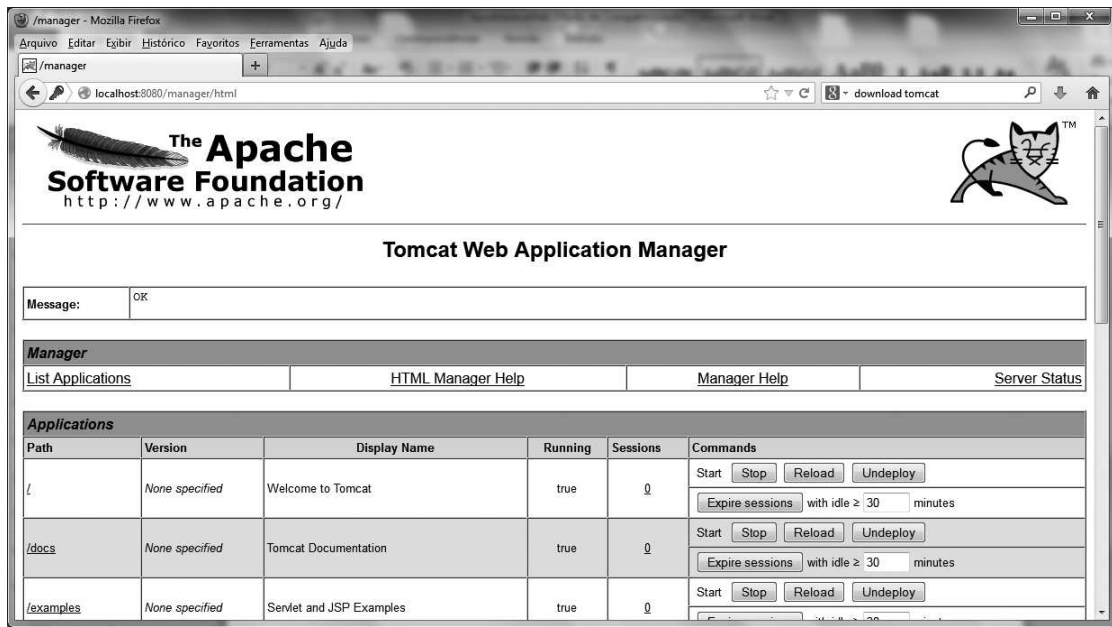


Figura 10.2
Gerenciador
de aplicações
do Tomcat.

Deployment

O deployment ou instalação web no Tomcat é bastante simples e pode ser realizado de três formas:

- Deployment de arquivo WAR.
- Deployment automático do arquivo WAR.
- Deployment manual.

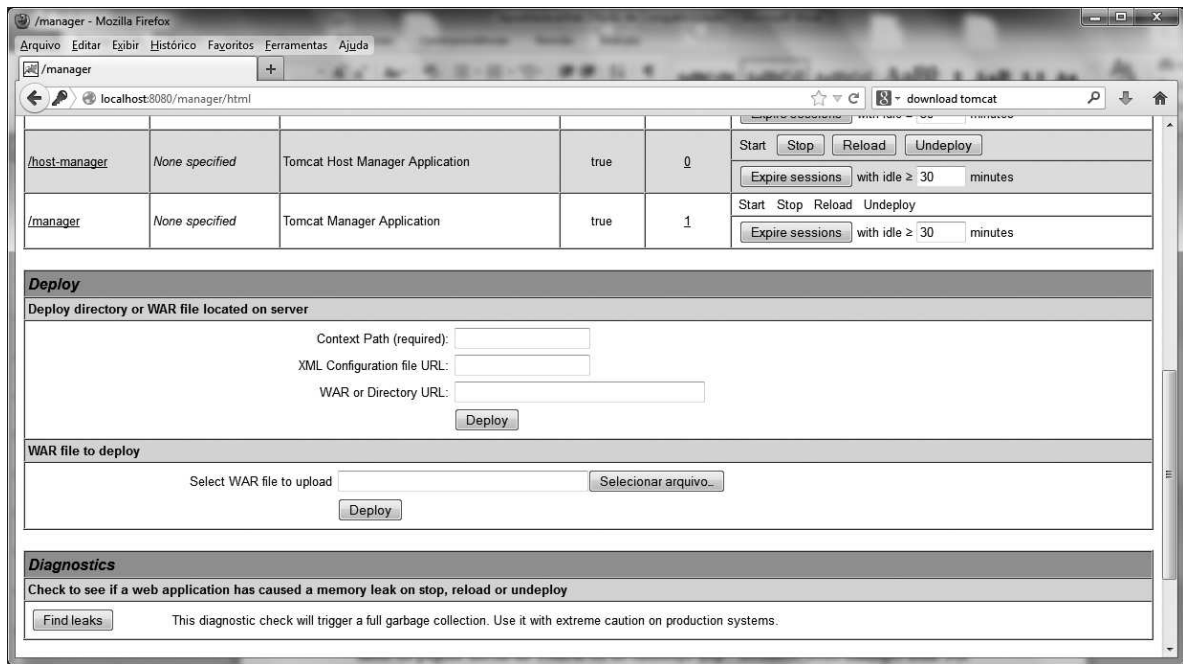


Deployment de arquivo WAR

O deployment de arquivo WAR pode ser realizado pelo gerenciador de aplicativos do Tomcat, mais especificamente através da sessão Deploy, exibida mais para baixo na página, após a lista de aplicações disponíveis. Essa sessão pode ser visualizada na figura 10.3.

- No Manager Webapp.
- Seção Deploy.
 - ▢ WAR file to deploy.
 - ▢ Selecione o arquivo.war com a aplicação a ser instalada.
 - ▢ Clique no botão Deploy.





A nova aplicação passará a figurar entre as aplicações disponíveis no Tomcat.

Figura 10.3
Ativação/Deploy
de aplicações com
arquivos war.

Deployment automático do arquivo WAR

Quando o Tomcat é inicializado, quaisquer arquivos WAR presentes em seu diretório wabapps são automaticamente instalados, tornando-se aplicações web disponíveis.

- Ao ser inicializado, o Tomcat automaticamente disponibiliza todas as aplicações cujos arquivos WAR estejam em seu diretório webapps.
- Novos arquivos WAR colocados no diretório wabapps são também automaticamente instalados e disponibilizados se o Tomcat estiver em modo autodeploy.
- O modo autodeploy pode ser configurado através do arquivo server.xml (configuração do Tomcat).
 - ▣ Localizado no diretório "conf".
 - ▣ Propriedade do Host.

Destacamos a seguir o trecho do arquivo server.xml onde o AutoDeploy é configurado:

```
<Host name="localhost" appBase="webapps" unpackWARs="true" autoDeploy="true"
      xmlValidation="false" xmlNamespaceAware="false">
```

Deployment manual

O deployment manual é bem mais trabalhoso.

- Toda a estrutura de diretórios da aplicação deverá ser criada manualmente:
- Raiz do contexto, WEB-INF, classes, lib etc.
- Os arquivos de cada diretório deverão ser copiados manualmente.
- A aplicação só se tornará disponível após a reinicialização do Tomcat.



- Bibliotecas Adicionais – 2 alternativas:
 - ▣ Adicionando o arquivo JAR em um subdiretório lib a ser criado no subdiretório WEB-INF da aplicação.
 - ▣ Será empacotado e distribuído por meio do arquivo WAR da aplicação web.
 - ▣ O deployment da aplicação a tornará completamente ativa, sem necessidade de configurações adicionais.
 - ▣ Adicionando o arquivo JAR no subdiretório lib do Apache Tomcat.
 - ▣ Disponíveis para todas as aplicações instaladas.
 - ▣ O Apache Tomcat deverá ser reiniciado para que os novos JAR se tornem disponíveis.

Exercícios de fixação

Deployment

Qual sua impressão sobre o processo de deploy de aplicação Java WEB no Apache Tomcat?

Atividades Práticas







Paulo Henrique Cayres possui graduação no curso Superior de Tecnologia em Processamento de Dados pela Universidade para o Desenvolvimento do Estado e da Região do Pantanal (UNIDERP), especialização em Análise de Sistemas pela Universidade Federal

de Mato Grosso do Sul (UFMS) e mestrado em Ciências da Computação pela Universidade Federal do Rio Grande do Sul (UFRGS). Atualmente é coordenador do Núcleo de Educação a Distância - NEaD da Faculdade da Indústria do Sistema FIEP. Sócio-diretor da CPP Consultoria e Assessoria em informática Ltda. Tem experiência na área de Ciência da Computação, com ênfase em Engenharia de Software. Professor titular em cursos de graduação e pós-graduação ministrando disciplinas de desenvolvimento de sistemas desde 1995. Instrutor de treinamento na linguagem Java de programação junto ao CITS em Curitiba e na ESR-RNP.

O curso tem como principal característica a prática em laboratório para desenvolvimento de aplicações em Java voltadas para a web/internet. São abordados desde o uso de sockets e RMI, passando pela utilização de applets, servlets e JSP (incluindo a manipulação de banco de dados) até a implementação do gerenciamento de sessões e uso de cookies. São abordados ainda a TagLib e JSTL, finalizando com o processo de Deployment de aplicações em servidores Web.

