UNICAMP/IC/MO410-051/

Capítulo 7 - Aplicações para Internet

Resumo do Capítulo 7 – Aplicações para Internet do livro:
Ramarkrishnan, R. and Gehrke, J. (2003) "Data Management Systems", McGraw-Hill, 3rd edition.

Alunos:
Roberto Alves Gallo Filho
Fernando Gobbi Bianchini

Resumo do Capítulo

Os principais tópicos abordados neste capítulo são: (a) Conceitos da Internet, (b) Formato de dados para a web, (c) Introdução à arquitetura "três-camadas", (d) A camada de apresentação (presentation tier) e (e) A camada lógica (middle tier).

Conceitos da Internet

URI (Uniform Resource Identifiers)

URIs são esquemas para a identificação, i. e., endereçamento de recursos na Internet. Entende-se recursos por qualquer objeto disponível na Internet, desde páginas estáticas e dinâmicas (e.g. index.html, login.jsp), até arquivos multimídia (e.g. song.mp3, picture.jpg e saídas de programas.

Exemplos de URIs:

```
http://www.cs.wisc.edu/~dbbook/index.html
mailto:webmaster@ic.unicamp.br
```

URIs são estruturadas em três parte. A primeira descreve o protocolo de acesso ao recursos. No primeiro exemplo, "http". A segunda parte, informa a máquina hospedeira (host) do recursos que se quer acessar (www.cs.wisc.edu). Finalmente, a terceira parte indica o nome do recurso. No primeiro exemplo "~dbbook/index.html"

As conhecidas URLs são subconjuntos das URIs

URLs (Uniform Resource Locator) são subconjuntos das URIs e representam recursos através de seu mecanismo de acesso.

Protocolo HTTP

O protocolo http, Hypertext Transfer Protocol, é um protocolo de comunicação. Protocolos comunicação são conjuntos de padrões e regras que definem e disciplinam determinadas trocas de informação.

O HTTP é o procotolo de alto nível mais comum para a transferência de dados na Internet. Na pilha da Internet existem outros protocolos de comunicação utilizados, como o TCP e o IP, que formam as pilhas TCP/IP.

Quando um usuário clica em um endereço em seu browser, por exemplo: http://www.cs.wisc.edu/~dbbook/index.html, o cliente envia requisições para o servidor e este as responde com o header do protocolo e possivelmente com dados. Neste caso, o cliente pode fazer automaticamente mais requisições, possivelmente trazendo recursos indicados nos dados anteriores (e.g. imagens de uma página HTML).

Existem algumas variações do protocolo HTTP, como o HTTPS, que é basicamente o HTTP sobre uma conexão SSL.

A seguir um exemplo de requisição HTTP feita por um browser:

```
GET ~/index.html HTTP/1.1
User-agent: Mozilla/4.0
Accept: text/html, image/gif, image/jpeg
```

E a resposta do servidor:

```
HTTP/1.1 200 OK
Date: Mon, 04 Mar 2002 12:00:00 GMT
Server: Apache/1.3.0 (Linux)
Last-Modified: Mon, 01 Mar 2002 09:23:24 GMT
Content-Length: 1024
Content-Type: text/html
<HTML> <HEAD></HEAD>
<BODY>
<h1>Barns and Nobble Internet Bookstore</h1>
Our inventory:
<h3>Science</h3>
<b>The Character of Physical Law</b>
...
```

Mensagens do protocolo HTTP têm estrutura simples. Requisições HTTP são compostas de linhas de texto ASCII com uma linha em branco para finalizar. As requisições usualmente contém 3, mas podem ter mais, dependendo do uso de Cookies:

A primeira é a linha de requisição: GET ~/index.html HTTP/1.1. O primeiro campo, GET, indica o método utilizado (GET ou POST). O segundo campo,

indica qual o nome do recurso e é derivado da URI. Finalmente, o terceiro campo diz a versão do protocolo utilizada: HTTP/1.1.

A segunda linha informa o tipo do cliente. Neste caso: User-agent: Mozilla/4.0

A terceira linha indica os arquivos aceitáveis pelo cliente: Accept: text/html, image/gif, image/jpeg

As repostas têm estrutura também em linhas, mas diferenciam-se em seu conteúdo. Como se observa no exemplo acima, a primeira linha, retorna o status da resposta gerada pela requisição: HTTP/1.1 200 OK. Neste caso três campos são identificáveis: (a) versão do protocolo: HTTP/1.1, (b) o código de Status: 200, e (c) mensagem associada do servidor: OK.

Os campos (b) e (c) são relativamente redundantes. Alguns códigos de status comuns e suas descrições são:

- "200 OK": Requisição feita com sucesso
- "400 Bad Request": Requisição não pode ser preenchida pelo servidor
- "404 Not Found": objeto da requisição não existe no servidor
- "505 HTTP Version not Supported": Versão não suportada

A segunda linha da resposta indica a data da criação do objeto: "Last-Modified: Mon, 01 Mar 2002 09:23:24 GMT"

As demais linhas contêm outras informações relevantes, como número de bytes enviados: Content-Length: 1024, tipo do objeto: Content-Type: text/html e etc. Finalmente, ao final do header, os dados para o cliente são retornados. No caso acima, um documento HTML.

Apesar de muito útil o protocolo HTTP não contempla algumas facilidades: (a) protocolo é *"stateless"*, não guarda estado das trocas de mensagens, (b) não existe o conceito de sessões.

O resultado é que as conexões são mantidas somente até a resposta de uma mensagem pelo servidor, não suportando interações cliente-servidor. Estas características representam um *trade-off* entre a facilidade de implementação do protocolo e a facilidade de implementação das aplicações.

Se manutenção de estados e dados é importante, a aplicação deve mantê-las, em níveis superiores, através do envio de das próprias informações ou de ids de sessões a cada requisição e resposta HTTP.

As abordagens mais comuns para manutenção de estados do lado do cliente são (a) Cookies e, (b) URLs geradas dinamicamente.

Formatos de Dados para a web

Os três principais formatos em uso para dados na Internet são: (a) HTML, a linguagem de apresentação para a Internet, (b) XML, modelo de dados hierárquico e auto-descritivo, (c) DTD, esquemas de padronização para XMLs e (d) XSLT (XML + informações para apresentação)

HTML, breve introdução

O HTML, ou "Hypertext Markup Language", é uma linguagem para formatação de texto. Seus comandos são representados por tags. Cada comando possui uma tag de início e uma de fim.

Exemplo:

```
<HTML> … </HTML>
<UL> … </UL>
```

Diversos editores de páginas geram código HTML a partir da apresentação gráfica do documento, como o Macromedia Dreamweaver ou o Microsoft Word.

O HTML não é case-sensitive quanto às tags. Algumas das principais tags são:

- "<HTML>": marca o início do documento.
- "<UL>": representa uma lista
- "<LI>": uma entrada na lista
- "<h1>": texto de cabeçalho (<h2>, <h3>, ...) de diferentes tamanhos.
- "<B>Título</B>": Negrito
- "<TABLE>": tabela
- "<TR>": linha de uma tabela
- "<TD>": divisão de uma linha

A seguir, um exemplo de código HTML.

```
<HTML>
<HEAD></HEAD>
<BODY>
<h1>Barns and Nobble Internet Bookstore</h1>
Our inventory:

<h3>Science</h3>
<b>The Character of Physical Law</b>
<UL>
     <LI>Author: Richard Feynman</LI>
     <LI>Published 1980</LI>
     <LI>Hardcover</LI>
</UL>
<h3>Fiction</h3>
<b>Waiting for the Mahatma</b>
<UL>
     <LI>Author: R.K. Narayan</LI>
     <LI>Published 1981</LI>
</UL>
<b>The English Teacher</b>
<UL>
     <LI>Author:  R.K. Narayan</LI>
     <LI>Published 1980</LI>
     <LI>Paperback</LI>
</UL>
</BODY>
</HTML>
```

XML (Extensible Markup Language)

O XML, ou Extensible Markup Language, é uma linguagem para comunicação de informação e permite um conjunto organizado e bem definido de dados. Enquanto o HTML se preocupa com a apresentação, o XML se preocupa com o conteúdo.

O XML utiliza meta-dados: servem para descrever dados e a própria linguagem. Seu conjunto de tags é extensível; permitindo um número infinito de novas linguagens ou conjunto de dados.

O XML é uma excelente linguagem para integração entre a visão orientada a documentos do HTML com a orientada a relações de SGBDs, pois permite a união, em um só documento, entre a descrição dos dados e aquilo que eles representam.

Uma utilidade do XML é a definição de linguagens específicas ou protocolos. Um exemplo é a Chemical Markup Language:

```
<molecule>
     <weight>234.5</weight>
     <Spectra>…</Spectra>
     <Figures>…</Figures>
</molecule>
```

Os principais objetivos do grupo que desenvolveu o XML eram: (a) manter compatibilidade com SGML (meta-linguagem para definição de outras linguagens de troca de documentos e dados – HTML), (b) permitir processadores fáceis de implementar, (c) o design deveria ser formal e preciso.

Desta forma, a estrutura do XML é uma fusão entre SGML e HTML. A sua aparência é idêntica a do HTML e é composta de uma hierarquia de tags definidas pelo usuário. Tais tags são chamados elementos. Elementos podem conter atributos e dados.

Os dados de um elemento podem ser outros elementos. A seguir, um exemplo de tag XML:

```
<BOOK genre="Science" format="Hardcover">…</BOOK>
```

Algumas regras sintáticas do XML são:

- Sensível a espaço e caso
- Tags de início e de fim de um elemento devem ter o mesmo nome
- Tags de início: "<" + nome do elemento + ">"
- Tags de fim: "</" + nome do elemento + ">"
- Elementos vazios não possuem dados e nem tag de fim:
- Começam com "<" e terminam com "/>"

Atributos XML fornecem informações adicionais para os elementos, que podem ter zero ou mais atributos. Cada atributo é declarado no formato nome_do_atributo=‘valor_do_atributo’. Algumas regras que atributos devem obedecer são:

- Não deve haver espaços entre os nomes o e o “=”
- Valores devem estar entre “ ou ‘
- Atributos são separados por espaços (ou tabs)

Os dados contidos em elementos XML podem ser outros elementos ou qualquer outra informação, contanto que não tenham: (a) os caracteres “<“ e “>” e , (b) caracteres especiais: &, <, >, “ e ;

A hierarquia XML é bem definida. Ela pode ser descrita com as seguintes regras:

- Tags XML podem ser agrupadas em uma hierarquia de árvore n-ária.
- Documentos XML podem ter apenas uma tag raiz
- Entre uma tag de início e uma tag de fim, é possível inserir:
    - Dados
    - Outros elementos
    - Combinação de elementos e dados

Um exemplo de documento XML para uma lista de livros:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<BOOKLIST>
      <BOOK genre="Science" format="Hardcover">
            <AUTHOR>
            <FIRSTNAME>Richard</FIRSTNAME>
                  <LASTNAME>Feynman</LASTNAME>
            </AUTHOR>
            <TITLE>The Character of Physical Law</TITLE>
            <PUBLISHED>1980</PUBLISHED>
      </BOOK>
      <BOOK genre="Fiction">
            <AUTHOR>
                  <FIRSTNAME>R.K.</FIRSTNAME>
                  <LASTNAME>Narayan</LASTNAME>
            </AUTHOR>
            <TITLE>Waiting for the Mahatma</TITLE>
            <PUBLISHED>1981</PUBLISHED>
      </BOOK>
</BOOKLIST>
```

DTD (Document Type Definition)

Enquanto grande parte do poder do XML vem de sua extesibilidade, alguns problemas podem naturalmente ocorrer, pois a semântica de um documento pode ser deturpada. Para garantir tipagem de documento XML, o DTD foi criado.

Desta forma, protocolos e linguagens definidas por XML podem ser padronizadas por DTDs. Um documento DTD especifica quais elementos e seus atributos são necessários e obrigatórios ou não. Ele permite a definição formal da estrutura da linguagem

A seguir, um exemplo de documento DTD:

```
<?xml version='1.0'?>
<!ELEMENT Cesta (Cereja+, (Maçã | Laranja)*) >
     <!ELEMENT Cereja EMPTY>
          <!ATTLIST Cereja sabor CDATA #REQUIRED>
     <!ELEMENT Maçã EMPTY>
          <!ATTLIST Maçã cor CDATA #REQUIRED>
     <!ELEMENT Laranja EMPTY>
          <!ATTLIST Laranja localização 'Florida'>
```

As tags DTD podem determinar regras sobre os elementos, dados e atributos. A tag !ELEMENT declara um elemento, com seu nome e seus sub-elementos (conteúdo). Os tipos de conteúdo válidos são:

- Outros elementos
- #PCDATA (parsed character data)
- EMPTY (sem conteúdo)
- ANY (sem restrições)
- Uma expressão regular

As expressões regulares utilizadas seguem o padrão ISO. Algumas expressões regulares simples são:

- exp1, exp2, exp3, …, expk: Lista de expressões regulares
- exp*: expressão opcional (zero ou mais ocorrências)
- exp+: expressão obrigatória (uma ou mais ocorrências)
- exp1 | exp2 | … | expk: disjunção de expressões

A linha DTD a seguir é um exemplo de uso de expressões regulares:

```
<!ELEMENT Cesta (Cereja+, (Maçã | Laranja)*) >
```

A tag !ATTLIST, por sua vez, define uma lista de atributos para um elemento. Atributos podem ser de diferentes tipos, podem ser obrigatórios ou não, e podem ter valores padrão.

Arquiteturas de Três Camadas

Sistemas intensivos em dados são usualmente compostos por três distintas funcionalidades ou componentes funcionais: (a) o gerenciamento de dados, (b) a lógica da aplicação e, (c) a apresentação.

A o gerenciamento de dados é usualmente feito através do uso de um SGDB, enquanto as implementações da lógica e da apresentação ficam por conta da aplicação, que pode ou não, estar diretamente ligada a o SGDB.

A arquitetura do sistema determina se esses componentes estarão em um único sistema (camada) ou distribuídos em diversas camadas.

Arquiteturas de uma camada

Arquiteturas de uma camada são usualmente implementadas em infra-estruturas de mainframes. Neste tipo de arquitetura, todos os três componentes executam na mesma máquina hospedeira.

Múltiplos usuários podem fazer uso do sistema, mas o mainframe deverá fazer desde o processamento das transações até o processamento de display das informações. As principais vantagens deste tipo de abordagem são: (a) facilidade de atualização, (b) maior segurança de dados.

As principais desvantagens são: (a) consumo excessivo de processamento e memórias do mainframe para implementar a lógica e a apresentação, (b) baixa capacidade de escalonamento.

Arquiteturas de duas camadas: cliente – servidor

Um dos problemas da arquitetura de uma camada é que ela requer que o processador central cuide de todas as tarefas da aplicação, inclusive da apresentação, que usualmente é bastante cara não só em termo de processamento.

A arquitetura cliente – servidor procura resolver este problema. Existem duas classificações de abordagens para este tipo de arquitetura, baseadas na divisão de componentes, e portanto trabalho, entre os peers.

Na divisão chamada de “thin client”, o cliente implementa apenas a interface gráfica. O servidor implementa a lógica do negócio e o gerenciamento de dados.

Na divisão do trabalho: “thick client”, cliente implementa interface gráfica e a lógica do negócio. O servidor implementa o gerenciamento de dados.

Atualmente a primeira divisão, a "thin client" é bastante adotada, em parte pelas desvantagens da segunda aproximação, que são: (a) atualização: da lógica do negócio não centralizada, (b) segurança: o servidor deve confiar nos clientes, de forma que controle de acesso e autenticação precisam ser gerenciados pelo primeiro, (c) cliente precisa deixar a base de dados do servidor em estado consistente e (d) dificuldade no escalonamento: grande transferência de dados entre cliente e servidor. No caso de existência de mais de um servidor, a quantidade de conexões é o produto de cliente e conexões.

A arquitetura de três camadas

Cada uma das camadas da arquitetura de três camadas resolve e cuida de problemas específicos. Cada uma das camadas envolve um componente e sua separação é facilmente identificável.

A Camada de Apresentação (presentation tier): cuida da interface com o usuário, que pode incluir a adaptação da apresentação da aplicação para diferentes dispositivos, como displays significativamente diferentes (por exemplo: PC, PDA, telefones celulares, acesso por voz)

A Camada Lógica (midle tier): implementa a lógica do negócio, isto é, garante a semântica da aplicação, operando ações complexas e mantendo diferentes estados. A camada lógica pode acessar diferentes sistemas de gerenciamento de dados.

A Camada de Gerenciamento de Dados (data management tier): representada por um ou mais SGBDs.

Um exemplo de sistema em três camadas é apresentado a seguir:

Matrículas em cursos Utilidade: Sistema on-line para matrículas de estudantes em cursos. O Sistema de Gerenciamento de Dados cuida das informações dos estudantes, dos cursos, das turmas, requisitos e qualquer outra informação O Servidor de Aplicação mantém a lógica para adicionar cursos, turmas, fazer matrículas e outras operações pertinentes. Programa cliente, possivelmente Web, cuida do login de diferentes tipos de usuários (estudantes, operadores), da apresentação de formulários e saídas e entradas de dados.

A separação em entre camadas procura maximizar as vantagens que a arquitetura pode oferecer e são principalmente: (a) sistemas heterogêneos, permitem que a melhor tecnologia seja empregada em cada camada, (b) manutenção, modificação e substituição das camadas independentemente, (c) permite clientes “leves”, apenas a camada de apresentação apenas nos clientes (web browsers), (e) acesso aos dados integrado, diversos sistemas de banco de dados podem ser manipulados transparentemente pela camada lógica, (f) gerenciamento centralizado das conexões, (g) escalabilidade, a replicação da camada lógica permite aumento da lógica do negócio, (h) código da lógica centralizado e (i) componentização

Camada de apresentação

No detalhamento da camada de apresentação, abordaremos as tecnologias que existem do lado cliente da arquitetura de três camadas. Serão discutidos nesta sessão formulários HTML, Java Script e Style sheets.

Formulário HTML

Os formulários HTML são uma das maneiras mais comum que encontramos para comunicação entre a camada de apresentação e a camada lógica. Podemos ter vários formulários dentro de um documento HTML e geralmente um formulário HTML tem o seguinte formato:

```
<FORM ACTION = "page.jsp" METHOD ="GET" NAME="LoginForm">
…
</FORM>
```

Cada atributo da tag "FORM", tem um significado e utilidade:

ACTION: Especifica a URI para onde será encaminha o conteúdo do formulário quando ocorrer o "submit". Nesta URI ocorrerá o processamento dos dados de entrada dos tags apropriados do formulário (e.g. INPUT, SELECT, TEXTAREA...).

METHOD: Especifica o método pelo qual serão enviados os dados. Existem dois métodos GET e POST, os quais serão abordados com mais detalhes a seguir.

NAME: Especifica o nome do formulário, de forma que este possa ser utilizado por scripts do lado do cliente. Esta abordagem é bastante utilizada com a tecnologia JavaScript.

Dentro dos formulários especificamos os elementos de entrada para os usuários, os quais podem ser um INPUT, SELECT ou TEXTAREA. Os formulários podem conter vários elementos de cada tipo.

Um exemplo de tag INPUT:

```
<INPUT TYPE="text" NAME="title" >
```

A tag INPUT poder ter vários atributos, sendo que os mais importantes são:

TYPE: Especifica o tipo do campo de entrada. Este tipo pode ser "text" (campo de texto), "password" (campo de texto aparecendo "*"), "reset" (botão onde todos os campos do formulário voltam para o seu valor default) e "submit" (botão que dispara o envio dos campos de dados do formulário para o servidor).

NAME: Especifica o nome do campo para que este possa ser identificado na camada lógica. Podemos definir o tag NAME para todos os tipos exceto reset e submit.

VALUE: Especifica o valor defaut do campo.

Exemplo de um formulário HTML utilizando a tag INPUT:

```
<form method=”POST” action=”TableOfContents.jsp”>
     <input type=”text” name=”userid”>
     <input type=”password” name=”password”>
     <input type=”submit” value=”Reset Values”>
     <input type=”reset” value=”Log on”>
</form>
```

Como mencionado anteriormente, existem dois métodos para o envio dos dados do formulário HTML para o servidor, “GET” e “POST”. No caso do GET, o conteúdo do formulário é codificado na URI, no seguinte formato:

```
Action?nome1=valor1&nome2=valor2&nome3=valor3
```

Onde action é o nome da URI especificada no formulários e os pares (nome,valor) são os campos de entrada do formulário.

No caso do método POST os campos de entrada do formulário vão embutidos na requisição http, invisível para o usuário. A aplicação e os seus requisitos é o que determina as escolha do método a ser utilizado.

JavaScript

O JavaScript é uma linguagem de script utilizada na camada de apresentação que roda diretamente do lado do cliente. Geralmente ela é utilizada para as seguintes finalidades: (a) detectar o tipo de browser, (b) validar formulários e controlar o browser (abrir/fechar janelas).

Para se utilizar scripts Java, deve-se utilizar a tag <SCRIPT>. Se a tag não conter o atributo “SRC”, o script estará embutido no próprio código HTML da página em questão. Do contrário, os scripts serão buscados na URI relativa apontada por SRC.

JavaScript é uma linguagem bem completa: pode-se utilizar variáveis, atribuições, operadores, expressões e funções.

A seguir, um exemplo de script JavaScript:

Formulário HTML:	JavaScript associado:
<form name=”LoginForm” method="POST“ action="TableOfContents.jsp" onSubmit=return testLoginEmpty ()”> <input type="text" name="userid"> <input type="password" name="password"> <input type="submit" value="Login“ name="submit"> <input type=“reset” value=“Clear”> </form>	<script language="javascript"> function testLoginEmpty() { loginForm = document.LoginForm if ((loginForm.userid.value == "") || (loginForm.password.value == "")) { alert('Please enter values for userid and password.'); return false; } else return true; } </script>

Stylesheets

A função dos stylesheets é de separar a formatação da página de seu conteúdo, de maneira que seja fácil a adaptação da apresentação da página (i.e. de seu *render*) à aplicações e clientes diferentes.

Duas principais vantagens da utilização de stylesheets são: (a) adaptação para formatos diferentes de displays e (b) padronização da apresentação gráfica de uma aplicação utilizando somente um conjunto específico de stylesheets.

No caso de uma mudança de fonte ou tamanho, por exemplo, não seria necessário alterar toda a aplicação e sim apenas o documento que descreve este padrão.

Existem duas principais linguagens de stylesheets: (a) Cascading Style Sheets (CSS), para documento HTML e (b) Extensible Stylesheet Language(XSL), para documento XML.

Cascading Style Sheets (CSS)

É a linguagem que define como os documentos HTML serão mostrados. Vários documentos HTML podem acessar um único css, facilitando a padronização de aplicações.

Um exemplo de um documento HTML referenciando um css:

```
<LINK REL=”style sheet”  TYPE=”text/css” HREF=”books.css”>
```

Cada linha de um css tem o seguinte formato:

```
selector {property:value}
```

Onde "selector" é a tag para o qual o formato está sendo definido, "property" é um atributo da tag cujo valor será ajustado e "value" é o valor do atributo.

XSL

O XSL é uma linguagem avançada para expressar style sheets. Pode ser empregada para reordenação, adição ou remoção de delimitadores e atributos do documento XML.

Camada lógica

Nesta camada é onde é epxressa a lógica do negócio, a conexão com o banco de dados, recebimento das entradas da camada de apresentação e a geração de saídas para camada de apresentação.

Referente a estada camada, serão apresentados os seguintes tópicos: CGI, Servidor de aplicação, Servlets, JSP e Manutenção de estados.

Common Gateway Interface (CGI)

Protocolo para passagem de argumentos para programas que rodam na camada lógica.

O CGI (Common Gateway Interface) permite que um navegador de Internet execute programas residentes no servidor web. Tais programas podem processar dados recebidos de formulários de páginas HTML, além de armazenar e ler dados no servidor.

Apesar de muito utilizado, principalmente em aplicações legadas, o protocolo tem algumas desvantagens, como: (a) para cada nova requisição, o programa de aplicação é chamado em um novo processo, causando uso excessivo de recursos no caso de muitos acessos simultâneos e, (b) Não existe compartilhamento de recursos entre os programas de aplicação (p.e., conexões com o banco de dados).

Uma solução para esses problemas é a utilização de um Servidor de aplicações.

Servidor de aplicações

Um servidor de aplicação é uma forma moderna de se implementar a camada de aplicação.

Um bom servidor de aplicação deve executar aplicações de negócio com níveis garantidos de desempenho, disponibilidade e integridade, e deve suportar múltiplos padrões e arquiteturas, de acordos com as necessidades específicas de cada negócio.

A idéia do Servidor de aplicação veio para evitar o overhead do CGI. Para tanto, um Servidor de aplicação é capaz de manter um Pool de threads e processos, gerenciar conexões acesso aos recursos de dados heterogêneos. O Servidor de aplicações ainda disponibiliza uma API para o gerenciamento de sessões.

Servlets e Java Server Pages (JSP)

Os Servlets, assim como as paginas JSP tratam chamadas (requests) e respostas (responses) do protocolo HTTP. A principal diferença entre os JSP's e os servlets é que as páginas JSP's são mais fáceis de se trabalhar, aproveitando com o front-end do usuário, ou seja o HTML.

Ambos são independentes de plataforma.
Os Servlets geram HTML através de escritas de strings contendo código HTML, pelo uso de seu objeto "PrintWriter". Desta forma, as páginas HTML estão contidas no código Java.

Os JSP's, por sua vez são códigos Java embutidos em páginas HTML, assim como linguagens como ASP e PHP.

Segue exemplos de Servlet e de JSP:
Servlets:

```
import java.io.*;
import java.servlet.*;
import java.servlet.http.*;
public class ServetTemplate extends HttpServlet {
     public void doGet(HTTPServletRequest request,
                HTTPServletResponse response)  throws
                SerletExpection, IOException {
          PrintWriter out=response.getWriter();
          out.println("Hello World");
     }
}
```

Java Server Pages:

```
<html>
<head><title>Welcome to B&N</title></head>
<body>
<h1>Welcome back!</h1>
<% String name="NewUser";
if (request.getParameter("username") != null) {
     name=request.getParameter("username");
}
%>
You are logged on as user <%=name%>
<p>
</body>
</html>
```

Manutenção de estados

O protocolo http não possui estados. Isso implica em vantagens e desvantagens. Como vantagens, pode-se apontar a facilidade de uso, a simplicidade de implementação e uso reduzido de memória, sendo ideal para aplicações estáticas.

Já as desvantagens são: (a) impossibilidade de manter nativamente no servidor estado e informações das requisições anteriores e (b) dificuldade para implementação de segurança (deve-se utilizar SSL/TSL, i.e. https).

Entretanto, pode-se utilizar algumas técnicas de manutenção de estados. Do lado do cliente, isso é possível pelo uso de "estados escondidos" e Cookies. Já do lado do lado servidor, estados e informações podem ser mantidas com o uso banco de dados ou na memória lógica da camada lógica.

Uma vez que o overhead para o acesso ao banco de dados é considerável e o alto consumo de memória do servidor é um problema, estados transitórios usualmente são mantidos pelo cliente, enquanto a consolidação se dá no servidor.

Pode-se utilizar Cookies para controlar os estados no lado do cliente. Cookies são coleções de pares (Nome,Valor) que são passados ao servidor através das requisições http.

Os Cookies devem ser utilizados quando se deseja armazenar estados interativos, como informação de login do usuário atual, carrinho de compras atual e qualquer opção não-permanente feita pelo usuário.

As vantagens de se utilizar Cookies são: (a) fáceis de usar, inclusive em Java, e (b) maneira simples de permitir persistência de dados no cliente mesmo após o web browser ser fechado. Existem, entretanto, algumas desvantagens como: (a) limite de 4 KB por Cookie e (b) usuários podem desabilitá-los.

Criando um Cookie em Java:

```
Cookie myCookie =new Cookie("username", "jeffd");
response.addCookie(userCookie);
```

Acessando um Cookie em Java:

```
Cookie[] cookies = request.getCookies();
String theUser;
for(int i=0; i<cookies.length; i++) {
     Cookie cookie = cookies[i];
     if(cookie.getName().equals("username"))
     theUser = cookie.getValue();
}
/* neste ponto theUser == "username" */
```

Para manter estado, pode-se também utilizar "estados escondidos". Estes ficam ocultos em dois possíveis lugares: (a) campos "hidden" em um formulário ou, (b) utilizando a informação do caminho (da URI).

No caso dos campos hidden, o usuário não vê tal informação (a não ser que visualize o código HTML). Se usado sem cautela, poderá prejudicar seriamente a performance, pois a cada pagina deve estar contida em um formulário.

Exemplo:

```
<input type='hidden' name='user' value='username'/>
```

No outro caso a informação é armazenada na URL de requisição.
Exemplo:

```
http://server.com/index.htm?user=jeffd
```