Aplicações para Internet

Capítulo 7

Tópicos

- Conceitos da Internet
- Formato de dados para a web
  - HTML, XML, DTDs
- Introdução à arquitetura "três-camadas"
- A camada de apresentação (*presentation tier*)
  - Formulários HTML; os métodos "Get" e "Post" do protocolo HTTP, URL; Javascript; Stylesheets; XSLT
- A camada lógica (*middle tier*)
  - CGI, servidores de aplicação, Servlets, JSP (*JavaServerPages)*, passagem de parâmetros, manutenção de estados (*cookies*)

URI (Uniform Resource Identifiers)

- Esquema de identificação única de recursos na Internet
- Um recurso pode ser qualquer coisa disponível pela Internet:
  - Páginas estáticas ou dinâmicas (Index.html, login.jsp)
  - Arquivos (mysong.mp3, picture.jpg)
  - Scripts, saídas de programas

- Exemplos:

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

Estrutura de URIs

http://www.cs.wisc.edu/~dbbook/index.html

- Três partes:
  - Nome do protocolo (*http*)
  - Nome do computador hospedeiro (*host*) (www.cs.wisc.edu)
  - Nome do recurso (~dbbook/index.html)
- URLs (*Uniform Resource Locator*) são subconjuntos das URIs

Protocolo HTTP

(*Hypertext Transfer Protocol*)

- O que é um protocolo de comunicação?
  - Conjunto de padrões que definem a estrutura de mensagens, a fim de disciplinar e tornar possível sua troca
  - Exemplos: TCP, IP, HTTP
- O mais comum usado pela Internet para transferir qualquer tipo de recurso
- O que acontece quando você clica em www.cs.wisc.edu/~dbbook/index.html?
  - Cliente (*web browser*) envia requisições para o servidor
  - Servidor responde às requisições recebidas
  - Cliente recebe as respostas; faz novas requisições
- Variantes (SSL, HTTPS, etc)

HTTP (Cont.)

Requisição (cliente):

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

Resposta (servidor):

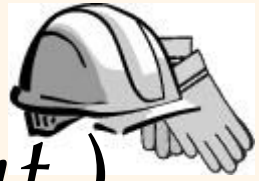
HTTP/1.1 200 OK
Date: Mon, 04 Mar 2002 12:00:00 GMT
Server: Apache/1.3.0 (Linux)
Last-Modified: Mon, 01 Mar 2002 09:23:24 GMT
Content-Length: 1024
Content-Type: text/html
<HTML> <HEAD></HEAD>
<BODY>
<h1>Barns and Nobble Internet Bookstore</h1>
Our inventory:
<h3>Science</h3>
The Character of Physical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

Estrutura do protocolo HTTP

Requisições HTTP – linhas de texto ASCII com uma linha em branco no fim

- Linha de requisição: GET ~/index.html HTTP/1.1
  - GET: campo do método http (GET ou POST)
  - ~/index.html: campo da URI
  - HTTP/1.1: versão do protocolo
- Tipo do cliente: User-agent: Mozilla/4.0
- Tipos de arquivos aceitáveis pelo cliente:
  Accept: text/html, image/gif, image/jpeg

Estrutura do protocolo HTTP(Cont.)



Respostas HTTP

- Linha de Status: HTTP/1.1 200 OK
  - Versão: HTTP/1.1
  - Código de Status: 200
  - Mensagem do servidor associada: OK
  - Combinações mais comuns para Códigos de Erro /mensagens do servidor:
    - 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
- Data da criação do objeto:
  Last-Modified: Mon, 01 Mar 2002 09:23:24 GMT
- Número de bytes enviados: Content-Length: 1024
- Tipo do objeto: Content-Type: text/html
- Outras informações (tipo do servidor, hora, etc)

HTTP - Observações importantes

- Protocolo "stateless": não há informações de estados
  - Não existem"sessões"
  - Conexões são mantidas somente até a resposta de uma mensagem
  - Não suporta interações entre cliente - servidor
  - *Tradeoff* entre facilidade de implementação e facilidade de implementação de aplicações – qualquer outra funcionalidade deve ser construída em outras camadas
- Implicações:
  - Qualquer informação de estados (informação de login, carrinho de compras) precisam ser acrescentados em cada requisição/resposta HTTP
  - Abordagens mais comuns para controle de estados:
    - Cookies
    - URLs geradas dinamicamente pelo servidor

Formatos de Dados para a web

- HTML
  - A linguagem de apresentação para a Internet
- Xml
  - Modelo de dados hierárquico e auto-descritivo
- DTD
  - Esquemas de padronização para XMLs
- XSLT (XML + informações para apresentação)

HTML: Exemplo

```
<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>
```

HTML: Breve introdução

- *Hypertext Markup Language:* formatação de texto
- Comandos são representados por *tags:*
  - Cada comando possui uma *tag* de início e de fim
  - Exemplo:
    - <HTML> … </HTML>
    - <UL> … </UL>
- Diversos editores que geram código HTML graficamente ou a partir de documentos (Microsoft Word)

HTML: Alguns comandos

- <HTML>:
- <UL>: lista
- <LI>: uma entrada na lista
- <h1>: aumento de cabeçalho (<h2>, <h3>, ...)
- <B>Título</B>: Negrito
- <TABLE>: tabela
- <TR>: linha de uma tabela
- <TD>: divisão de uma linha

XML: Exemplo

```
<?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>
  <BOOK genre="Fiction">
    <AUTHOR>
      <FIRSTNAME>R.K.</FIRSTNAME><LASTNAME>Narayan</LASTNAME>
    </AUTHOR>
    <TITLE>The English Teacher</TITLE>
    <PUBLISHED>1980</PUBLISHED>
  </BOOK>
</BOOKLIST>
```

XML (Extensible Markup Language)

- Linguagem
  - Uma maneira para comunicação de informação (conjunto organizado e bem definido de dados)
  - Meta-Dados que descrevem os dados/linguagens
- Extensível
  - Conjunto de "*tags*" infinitas: novas linguagens ou conjunto de dados
- Modo de integração dos SGBDs com aplicações web: ponte entre a visão orientada a documentos do HTML e a visão orientada a esquemas de um SGBD

XML (Cont.)

- União em um só documento dos dados e descrição do que eles representam
  - Útil para definição de linguagens próprias ou protocolos
- Exemplo: Chemical Markup Language

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

- Objetivos principais:
  - Compatível com SGML (meta-linguagem para definição de outras linguagens de troca de documentos e dados – HTML)
  - Processadores fáceis de implementar
  - O design deve ser formal e preciso

XML – Estrutura

- XML: fusão entre SGML e HTML
- Aparência idêntica à HTML
- Hierarquia de tags definidas pelo usuário chamados elementos com atributos e dados
- Dados são descritos pelos elementos, elementos são descritos por atributos

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

atributo

Tag de início

Valor do atr.

dado

Tag de fim

XML – Elementos

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

Tag de inicio
Nome do elem

atributo

Valor do atr.

dado

closing tag

- 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 "/>"

  <BOOK/>

XML – Atributos

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

Tag de início
Nome do elemento

atributo

Valor do atr.

dado

Tag de fim

- Atributos fornecem informação adicional para os elementos
- Podem existir zero ou mais atributos para cada elemento, cada um com o formato:

  Nome_atributo='valor_atributo'

  - Sem espaços entre o nome e o "='"
  - Valores devem estar entre " ou '
- Atributos são separados por espaços (ou *tabs*)

XML – Dados e comentários

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

Tag de início
Nome do elemento

atributo

Valor do atr.

dado

Tag de fim

- Dados são qualquer informação entre as tags de início e de fim
- Os dados não podem conter os caracteres “<“ e “>”
- Caracteres especiais: &, <, >, “ e ;

- Comentários:
  <!- comentário ->

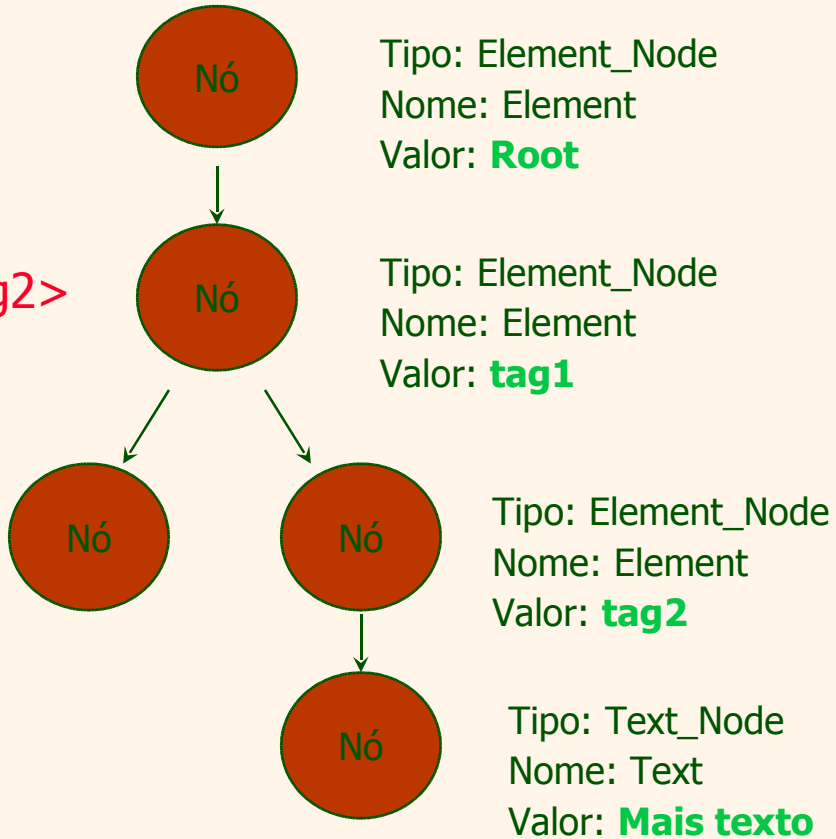
XML – Agrupamento e hierarquia

- Tags XML podem ser agrupadas em uma hierarquia de árvore
- Documentos XML podem ter apenas uma tag raiz
- Entre uma tag de início e uma tag de fim, é possível inserir:
    1. Dado
    2. Outros elementos
    3. Combinação de elementos e dados
- Documento XML "bem formado": declaração, elemento raiz, elementos devidamente agrupados

XML – estrutura de armazenamento

- Armazenamento é feito como em qualquer outra árvore *n-ária* (DOM)

```
<root>
   <tag1>
      Algum texto
      <tag2>Mais texto</tag2>
   </tag1>
</root>
```



DTD (Document Type Definition)

- Esquema de verificação para documentos Xml
- Protocolos e linguagens definidas por Xml podem ser padronizadas por DTDs
- Um DTD especifica quais elementos e seus atributos são necessários (obrigatórios ou não)
  - Definição formal da estrutura da linguagem

DTD – Exemplo

```
<?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'>
```

```
<Cesta>
    <Cereja sabor='bom'/>
    <Maçã cor='vermelha'/>
    <Maçã cor='verde'/>
</Cesta>
```

```
<Cesta>
    <Maçã/>
    <Cereja sabor='bom'/>
    <Laranja/>
</Cesta>
```

DTD - !ELEMENT

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

Nome Sub-elementos

- !ELEMENT declara um elemento, com seu nome e seus sub-elementos (conteúdo)
- Tipos de conteúdo:
  - Outros elementos
  - #PCDATA (parsed character data)
  - EMPTY (sem conteúdo)
  - ANY (sem restrições)
  - Uma expressão regular

DTD - !ELEMENT (Cont.)

- Estrutura de uma expressão regular:
  - exp_1, exp_2, exp_3, …, exp_k: Lista de expressões regulares
  - exp*: expressão opcional (zero ou mais ocorrências)
  - exp+: expressão obrigatória (uma ou mais ocorrências)
  - exp_1 | exp_2 | … | exp_k: disjunção de expressões

DTD - !ATTLIST

<!ATTLIST Cereja sabor CDATA #REQUIRED>

Elemento Artibuto Tipo Flag

<!ATTLIST Laranja localização CDATA #REQUIRED
cor 'laranja'>

- !ATTLIST define uma lista de atributos para um elemento
- Atributos podem ser de diferentes tipos, podem ser obrigatórios ou não, e podem ter valores padrão

DTD – Bem-formado e válido

```
<?xml version='1.0'?>
<!ELEMENT Cesta (Cereja+)>
    <!ELEMENT Cereja EMPTY>
        <!ATTLIST Cereja sabor CDATA #REQUIRED>
```

Mal-formado

```
<cesta>
  <Cereja sabor=bom>
</Cesta>
```

Bem-formado mas inválido

```
<Trabalho>
  <Localização>Casa</Localização>
</Trabalho>
```

Bem-formado e válido

```
<Cesta>
  <Cereja sabor='bom'/>
</Cesta>
```

XML e DTDs

- Desenvolvimento crescente de DTDs padronizados
  - MathML
  - Chemical Markup Language
- Permite troca 'leve' de informações sem perda na semântica

- Existência de linguagens sofisticadas de consultas:
  - Xquery
  - XPath

Tópicos

- Conceitos da Internet
- Formato de dados para a web
  - HTML, XML, DTDs
- Introdução à arquitetura "três-camadas"
- A camada de apresentação (*presentation tier*)
  - Formulários HTML; os métodos "Get" e "Post" do protocolo HTTP, URL; Javascript; Stylesheets; XSLT
- A camada lógica (*middle tier*)
  - CGI, servidores de aplicação, Servlets, JSP (*JavaServerPages)*, passagem de parâmetros, manutenção de estados (*cookies*)

Componentes de sistemas de dados (data-intensive)

Três tipos diferentes de funcionalidades (ou componentes funcionais):

- Gerenciamento de dados
- Lógica da aplicação
- Apresentação

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

Arquiteturas de uma camada

Todas as funcionalidades combinadas em uma única camada (normalmente um *mainframe*)

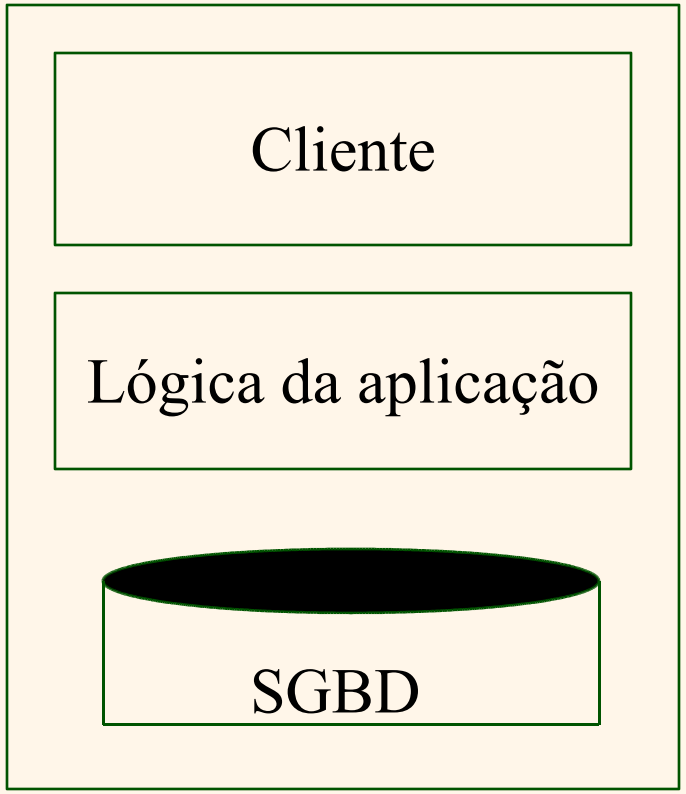
- Acesso através de terminais simples

Vantagens:

- Facilidade de manutenção e administração

Desvantagens:

- Exigência por interfaces gráficas
- Processamento centralizado para todas as funcionalidades é muito para um sistema central



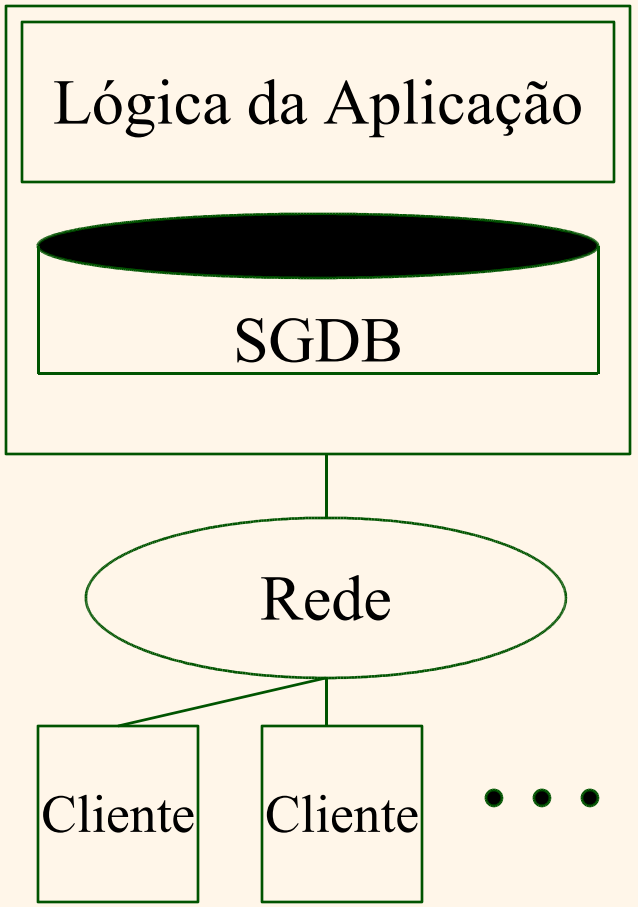
Arquiteturas Cliente-Servidor

Divisão do trabalho: *thin client*

- Cliente implementa apenas a interface gráfica
- Servidor implementa a lógica do negócio e o gerenciamento de dados

Divisão do trabalho: *thick client*

- Cliente implementa interface gráfica e lógica do negócio
- Servidor implementa gerenciamento de dados



Arquiteturas Cliente-Servidor(Cont.)

Desvantagens de *thick-client*

- Atualização da lógica do negócio não centralizada
- Segurança: servidor deve confiar nos clientes
  - Controle de acesso e autenticação precisam ser gerenciados no servidor
  - Cliente precisa deixar a base de dados do servidor em estado consistente
  - Uma possível solução: encapsulamento de todo o acesso à base de dados através de *stored procedures*
- Escalonamento
  - Grande transferência de dados entre cliente e servidor
  - Mais de um servidor: x clientes e y servidores – x*y conexões

A arquitetura três-camadas

Camada de Apresentação

Progama Cliente (*web brownser*)

Camada lógica

Servidor de Aplicação

Camada de Gerenciamento de dados

Sistema de banco de dados

As três camadas

Camada de Apresentação (*Presentation tier*)

- Interface com o usuário
- Necessidade de adaptação para diferentes dispositivos de display(PC, PDA, tel. celulares, acesso a voz?)

Camada lógica (*Midle tier*)

- Implementa lógica do negócio (ações complexas, mantém diferentes estados)
- Acessa diferentes sistemas de gerenciamento de dados

Camada de gerenciamento de dados(*Data management tier*)

- Um ou mais SGBDs

Exemplo 1: reservas de passagens

- Sistema de reservas de passagens aéreas on-line
- Sistema de Banco de Dados
  - Informação das linhas aéreas, assentos disponíveis, informação do cliente, etc.
- Servidor de Aplicação
  - Lógica para fazer as reservas, cancelar reservas, adicionar linhas aéreas, etc.
- Programa Cliente
  - Login de diferentes usuários, disponibilização de formulários e saídas de dados

Exemplo 2: Matrículas em cursos

- Sistema on-line para matrículas de estudantes em cursos
- Sistema de gerenciamento de dados
  - Informação dos estudantes, dos cursos, das turmas, requisitos, etc.
- Servidor de Aplicação
  - Lógica para adicionar cursos, turmas, fazer matrículas, etc
- Programa cliente
  - Login de diferentes tipos de usuários (estudantes, operadores), formulários e saídas de dados

Tecnologias

Programa Cliente *(Web Browser)*	*HTML* *Javascript* *XSLT*
Servidor de Aplicação *(Tomcat, Apache)*	*JSP* *Servlets* *Cookies* *CGI*
Sistema de banco de dados (DB2)	*XML* *Stored Procedures*

Vantagens da Arquitetura três-camadas

- Sistema heterogêneos
  - Manutenção, modificação e substituição das camadas independentemente
- Clientes "leves"
  - Apenas a camada de apresentação nos clientes (*web browsers*)
- Acesso aos dados integrado
  - Diversos sistemas de banco de dados podem ser manipulados transparentemente pela camada lógica
  - Gerenciamento centralizado das conexões
- Escalabilidade
  - Replicação da camada lógica permite escalabilidade da lógica do negócio
- Desenvolvimento do *software*
  - Código da lógica centralizado
  - Componentização

Tópicos

- Conceitos da Internet
- Formato de dados para a web
  - HTML, XML, DTDs
- Introdução à arquitetura "três-camadas"
- A camada de apresentação (*presentation tier*)
  - Formulários HTML; os métodos "Get" e "Post" do protocolo HTTP, URL; Javascript; Stylesheets; XSLT
- A camada lógica (*middle tier*)
  - CGI, servidores de aplicação, Servlets, JSP (*JavaServerPages)*, passagem de parâmetros, manutenção de estados (*cookies*)

Overview da Camada de Apresentação

- Revisão: Funcionalidade da camada de apresentação
  - Interface com o usuário
  - Necessidade de suporte a diferentes dispositivos (PC, PDA, tel. Celulares, acesso a voz?)
  - Funcionalidade simples (validação de entradas)
- Assuntos que serão cobertos
  - Formulários HTML: como passar os dados para a camada lógica
  - JavaScript: funcionalidade simples na camada de apresentação
  - Style sheets: separar dados de sua formatação

Formulários HTML

- Maneira mais comum de transferência de dados entre o cliente e a camada lógica
- Formato geral:
  - <FORM ACTION="page.jsp" METHOD="GET"
    NAME="LoginForm">
    …
    </FORM>
- Componentes da tag FORM:
  - ACTION: especifica a URI que manipula o conteúdo
  - METHOD: especifica o método utilizado para enviar os dados (GET ou POST)
  - NAME: nome do formulário; pode ser usado em scripts no cliente para a referência ao formulário

Formulários HTML (Cont.)

- Tag INPUT
  - Atributos:
    - TYPE: text (campo de texto), password (campo de texto com "*"), reset (limpa todos os campos do formulário)
    - NAME: nome simbólico, usado para identificar o valor do campo na camada lógica
    - VALUE: valor padrão
  - Exemplo: <INPUT TYPE="text" Name="título">
- Formulário exemplo:

```
<form method="POST" action="TableOfContents.jsp">
    <input type="text" name="userid">
    <input type="password" name="password">
    <input type="submit" value="Login" name="submit">
    <input type="reset" value="Limpar">
</form>
```

Passagem dos parâmetros

Dois métodos: GET e POST

- GET
  - Conteúdo do formulário vai na URI
  - Estrutura:
    action?nome1=valor1&nome2=valor2&nome3=valor3
    - Action: nome da URI especificada no formulário
    - Os pares (nome,valor) vêm de campos de entrada (input) do formulário; campos vazios possuem valores vazios (“nome=“)
  - Exemplo (formulário anterior):
    TableOfContents.jsp?userid=john&password=johnpw
  - Note que a ação da página precisa ser um programa, um script ou uma página que irá processar a entrada do usuário

Método GET: codificação dos campos

- Campos de formulários podem conter caracteres ASCII que não podem aparecer na URI
- Uma convenção especial de codificação converte esses valores em caracteres compatíveis com a URI:
  - Converte todos os caracteres especiais em %xyz, onde xyz é o código ASCII do caractere. Caracteres especiais: &, =, +, %, etc.
  - Converte espaços para "+"
  - Junta os pares (nome, valor) das tags INPUT com "&" como separador para formar a URI

Formulários HTML: Exemplo

```
<form method="POST" action="TableOfContents.jsp">
    <table align = "center" border="0" width="300">
    <tr>
        <td>Userid</td>
        <td><input type="text" name="userid" size="20"></td>
    </tr>
    <tr>
        <td>Password</td>
        <td><input type="password" name="password" size="20"></td>
    </tr>
    <tr>
        <td align = "center"><input type="submit" value="Login“
                name="submit"></td>
    </tr>
    </table>
</form>
```

JavaScript

- Objetivo: adicionar funcionalidade na camada de apresentação
- Tarefas mais comuns:
  - Detectar tipo do *browser* para carregar uma página específica
  - Validação de formulários: validar as entradas dos campos
  - Controle do *browser*: Abrir / fechar janelas (pop-up)
- Normalmente embutido diretamente no código HTML, com a tag <SCRIPT> … </SCRIPT>
- Tag <SCRIPT> possui diversos atributos:
  - LANGUAGE: linguagem do script
  - SRC: arquivo com o código do script
  - Exemplo
    <SCRIPT LANGUAGE="JavaScript" SRC="validate.js>
    </SCRIPT>

JavaScript (Cont.)

- Se a tag <SCRIPT> não possui o atributo SRC, o script fica no próprio código HTML
- Exemplo:
  <SCRIPT LANGUAGE="JavaScript">
  <!-- alert("Bem vindo à Unicamp!!")
  //-->
  </SCRIPT>
- Comentários:
  - <!– comentário para o código HTML (ignorar o código do script caso o *browser* não suporte tal linguagem)
  - //: comentário JavaScript para finalizar o comentário HTML

JavaScript (Cont.)

- Linguagem de script completa
  - Variáveis
  - Atribuições (=, +=, …)
  - Operadores de comparação (<,>,…), e booleanos (&&, ||, !)
  - Expressões
    - if (condition) {statements;} else {statements;}
    - Loops for, do-while e while
  - Funções com valores de retorno
    - f(arg1, …, argk) {statements;}

JavaScript: Exemplo

Formulário HTML:

```
<form method="POST“
    name=“LoginForm”
     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>
```

JavaScript associado:

```
<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

- Idéia: Separar o display do conteúdo, e adaptar o display a diferentes formatos de apresentação
- Dois aspectos:
  - Transformações do documento para identificar quais partes serão mostradas e em que ordem
  - Renderizações do documento para decidir como cada parte do documento será mostrada.
- Porque Stylesheets?
  - Reuso do mesmo documento para diferentes displays
  - Adaptação do formato à preferência do usuário
  - Reuso do mesmo documento em contextos diferentes
- Duas linguagens de stylesheets:
  - Cascading style sheets (CSS): para documentos HTML
  - Extensible stylesheet language (XSL): para documentos XML

CSS: Cascading Style Sheets

- Define como os documentos HTML serão mostrados
- Diferentes documentos HTML podem referenciar um mesmo css
  - Possibilidade de mudar um website apenas alterando um css
  - Exemplo:
    <LINK REL="style sheet" TYPE="text/css" HREF="books.css"/>

Cada linha possui três partes:

selector {property: value}

- Selector: Tag cujo formato é definido
- Property: atributo da tag cujo valor será ajustado
- Value: valor do atributo

CSS: Cascading Style Sheets

Exemplo de stylesheet:

```
body {background-color: yellow}
h1 {font-size: 36pt}
h3 {color: blue}
p {margin-left: 50px; color: red}
```

A primeira linha tem o mesmo efeito da seguinte tag:

<body background-color="yellow>

XSL

- Linguagem para descrever style sheets
  - http://www.w3.org/Style/XSL/
- Três componentes:
  - XSLT: XSL Transformation language
    - Transforma um documento em outro diferente
    - http://www.w3.org/TR/xslt
  - XPath: XML Path Language
    - Seleciona partes de um documento XML
    - http://www.w3.org/TR/xpath
  - Objetos de formatação XSL
    - Formata a saída de uma transformação XLS
    - http://www.w3.org/TR/xsl/

Tópicos

- Conceitos da Internet
- Formato de dados para a web
  - HTML, XML, DTDs
- Introdução à arquitetura "três-camadas"
- A camada de apresentação (*presentation tier*)
  - Formulários HTML; os métodos "Get" e "Post" do protocolo HTTP, URL; Javascript; Stylesheets; XSLT
- A camada lógica (*middle tier*)
  - CGI, servidores de aplicação, Servlets, JSP (*JavaServerPages)*, passagem de parâmetros, manutenção de estados (*cookies*)

Overview da Camada lógica

- Revisão: Funcionalidade da camada lógica
  - Codificação da lógica do negócio
  - Conexão com a base de dados
  - Recebimento de entradas da camada de apresentação
  - Geração de saídas para a camada de apresentação
- O que será coberto:
  - CGI: protocolo para passagem de argumentos para programas que rodam na camada lógica
  - Servidor de Aplicação: ambiente de *runtime* para a camada lógica
  - Servlets: programas Java na camada lógica
  - JavaServerPages: scripts Java na camada lógica
  - Manutenção de estados: como manter estados na camada lógica (foco: Cookies).

CGI: Common Gateway Interface

- Objetivo: transmitir argumentos dos formulários HTML para programas que rodam na camada lógica
- Detalhes do protocolo não precisam ser levados em consideração → interfaces de alto-nível disponibilizadas por bibliotecas

- Desvantagens:
  - O programa de aplicação é chamado em um novo processo a cada nova requisição
  - Não há compartilhamento de recursos entre os programas de aplicação (p.e.,conexões com o banco de dados)
  - Solução: servidores de aplicação

CGI: Exemplo

- Formulário HTML:

```
<form action="findbooks.cgi" method=POST>
Type an author name:
<input type="text" name="authorName">
<input type="submit" value="Send it">
<input type="reset" value="Clear form">
</form>
```

- Código Pearl relacionado:

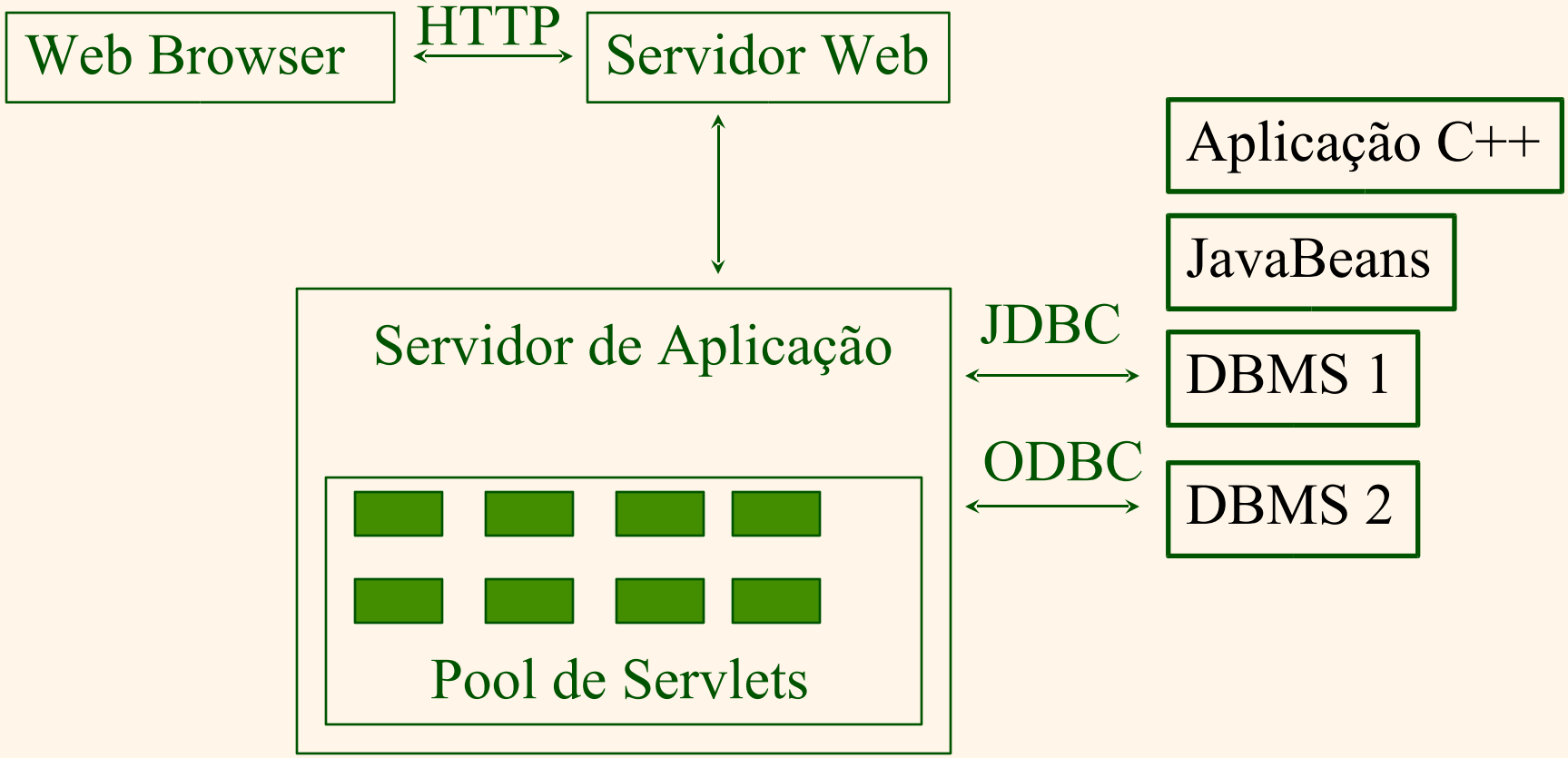
```
use CGI;
$dataIn=new CGI;
$dataIn->header();
$authorName=$dataIn->param('authorName');
print("<HTML><TITLE>Argument passing test</TITLE>");
print("The author name is " + $authorName);
print("</HTML>");
exit;
```

Servidores de Aplicação

- Idéia: Evitar o *overhead* do CGI
  - Pool de threads e processos
  - Gerenciamento de conexões
  - Permite acesso a recursos de dados heterogêneos
  - Outras funcionalidades como APIs para gerenciamento de sessões

Servidor de Aplicação: estrutura de processos

Estrutura de Processos



Servlets

- Java Servlets: código Java que roda na camada lógica
  - Independente de plataforma (Java)
  - APIs Java disponíveis, incluindo JDBC

Exemplo:

```
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");
    }
}
```

Servlets (Cont.)

- Vida de a servlet?
  - Servidor web encaminha requisições para o servlet container
  - Container cria uma instância do servlet (chama o método init(); destroy() na desalocação)
  - Container chama o método service()
    - service() chama doGet() ou doPost(), dependendo de qual método HTTP foi submetido

Servlets: Exemplo

```
public class ReadUserName extends HttpServlet {
    public void doGet(          HttpServletRequest request,
                                HttpSevletResponse response)
        throws ServletException, IOException {
        reponse.setContentType("text/html");
        PrintWriter out=response.getWriter();
        out.println("<HTML><BODY>\n <UL> \n" +
                "<LI>" + request.getParameter("userid") + "\n" +
                "<LI>" + request.getParameter("password") + "\n" +
                "<UL>\n<BODY></HTML>");
    }
    public void doPost(         HttpServletRequest request,
                                HttpSevletResponse response)
        throws ServletException, IOException {
        doGet(request,response);
    }
}
```

Java Server Pages

- Servlets
  - Geram HTML através de escritas do código HTML no seu objeto "PrintWriter"
  - Pagina contida no código
- JavaServerPages
  - Escrito em HTML, código "servlet" embutido no HTML
  - Código contido na página
  - Internamente são transformados em um servlet

JavaServerPages: Exemplo

```
<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

HTTP é um protocolo sem estados

- Vantagens
  - Facilidade de uso
  - Ótimo para aplicações estáticas
  - Não necessita de espaço em memória adicional
- Desvantagens
  - Não manter informações de requisições anteriores não permite:
    - Carrinhos de compras
    - Login de usuários
    - Conteúdo dinâmico
  - Segurança difícil de implementar

Estado da aplicação

- No lado do servidor
  - Informação é mantida no banco de dados ou na memória local da camada lógica
- No lado do cliente
  - Informação é mantida no computador do cliente através de Cookies
- Estado "escondido"
  - Informação é escondida entre páginas dinâmicas

Estado da aplicação

Estados no lado do servidor

- Diferentes tipos de manutenção de estados no lado do servidor:
- 1. Informação armazenada no banco de dados
  - Dados estão seguros no banco de dados
  - Mas: exige um acesso ao banco de dados para consultar ou atualizar tal informação (overhead)
- 2. Informação "armazenada" na memória local da camada lógica
  - Mapeamento do IP do usuário em estados
  - Mas: informação volátil e consome muita memória do servidor 5 M IPs = 20 MB

Estados armazenados no servidor

- Deveria usar armazenamento de estados no servidor para informações persistentes
  - Pedidos antigos de clientes
  - Cliques de usuários pelo site
  - Escolhas permanentes dos usuários

Estados no lado cliente: Cookies

- Armazenamento de texto no cliente enviada ao servidor através de toda requisição HTTP
  - Pode ser desabilitado pelo cliente
  - Comumente e erroneamente encarados como "perigosos", e por tal motivo acabam sendo rejeitados por usuários[1]
- Coleção de pares (Nome, Valor)

[1] http://www.webdevelopersjournal.com/columns/stateful.html

Estados no lado cliente: Cookies

- Vantagens
  - Fácil de se usar em Java (Servlets / JSP)
  - Oferecem uma maneira simples de persistência de dados no cliente mesmo após o *web browser* ter sido fechado
- Desvantagens
  - Limite de 4 KB
  - Usuários podem desabilitá-los
- Devem ser usados para armazenar estados interativos
  - Informação de login do usuário atual
  - O carrinho de compras atual
  - Qualquer opção não-permanente feita pelo usuário

Criando um Cookie

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

- É possível criar um cookie em qualquer momento

Acessando um Cookie

```
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"
```

- Cookies precisam ser acessados antes de setar o cabeçalho de resposta:

```
response.setContentType("text/html");
PrintWriter out = response.getWriter();
```

Funcionalidades dos Cookies

- Cookies podem ter
  - Duração (expira ou persiste após o fechamento do *browser*)
  - Filtros para quais caminhos de domínios / diretórios o cookie é enviado
- Maiores informações: Java Servlet API (Javadoc), Tutoriais de Servlets, internet, site da sun, etc

Estados escondidos

- Normalmente o usuário desabilita os cookies
- Os dados podem ser "escondidos" em dois lugares:
  - Campos "hidden" em um formulário
  - Utilizando a informação do caminho
- Não exige "armazenamento" de informação pois a informação é passada por cada página web

Estados escondidos: Campos "hidden"

- Declaração de campos "hidden" em um formulário:
  - <input type='hidden' name='user' value='username'/>
- Usuário não vê tal informação (a não ser que ele visualize o código HTML)
- Se usado sem cautela, irá prejudicar seriamente a performance, pois CADA página deve estar contida em um formulário

Estados escondidos: informação no Path

- Informação armazenada na URL de requisição:

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

- Separação dos campos através do caractere '&':

  `index.htm?user=jeffd&preference=pepsi`

- Existem "ferramentas" em Java que parseiam tais campos (`javax.servlet.http.HttpUtils parserQueryString()`)

Múltiplos métodos de armazenamento de estados

- Normalmente todos os métodos de armazenamento de estados são utilizados:
  - Usuário faz login no sistema e essa informação é armazenada no cookie
  - Usuário solicita uma busca que é armazenada através de um Path
  - Usuário acrescenta um item no carrinho de compras - cookie
  - Usuário realiza a compra, e o número de seu cartão de crédito é armazenado /obtido do banco de dados
  - Usuário gera um "histórico de cliques" pelo site, o qual é armazenado no servidor web (o que pode ser analisado posteriormente)

Resumo

Assuntos cobertos:

- Conceitos da Internet
- Formato de dados para a web
  - HTML, XML, DTDs
- Introdução à arquitetura "três-camadas"
- A camada de apresentação (*presentation tier*)
  - Formulários HTML; os métodos "Get" e "Post" do protocolo HTTP, URL; Javascript; Stylesheets; XSLT
- A camada lógica (*middle tier*)
  - CGI, servidores de aplicação, Servlets, JSP (*JavaServerPages)*, passagem de parâmetros, manutenção de estados (*cookies*)