Modulo I
Introdução a Programação Web

Prof. Ismael H F Santos

Ementa

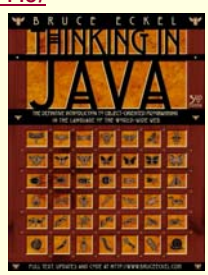
- Modulo VII – Programação Web com Java
  - Arquitetura da World Wilde Web - WWW
  - URI e URL
  - Protocolo HTTP
  - Tecnologias do lado do Cliente
  - Tecnologias do lado do Servidor

Bibliografia

- *Linguagem de Programação JAVA*
  - *Ismael H. F. Santos, Apostila UniverCidade, 2002*
- *The Java Tutorial: A practical guide for programmers*
  - *Tutorial on-line: http://java.sun.com/docs/books/tutorial*
- *Java in a Nutshell*
  - *David Flanagan, O´Reilly & Associates*
- *Just Java 2*
  - *Mark C. Chan, Steven W. Griffith e Anthony F. Iasi, Makron Books.*
- *Java 1.2*
  - *Laura Lemay & Rogers Cadenhead, Editora Campos*

Livros

- **Core Java 2**, Cay S. Horstmann, Gary Cornell
  - Volume 1 (Fundamentos)
  - Volume 2 (Características Avançadas)
- **Java: Como Programar**, Deitel & Deitel
- **Thinking in Patterns with JAVA**, Bruce Eckel
  - **Gratuito**. http://www.mindview.net/Books/TIJ/

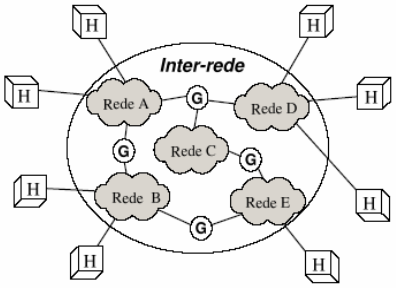


POO-Java

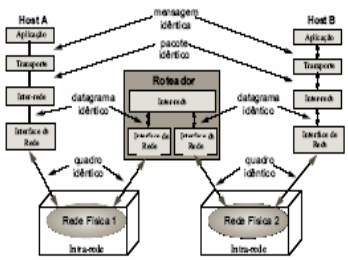
Arquitetura WWW

Arquitetura TCP/IP

Arquitetura TCP/IP



- A arquitetura Internet TCP/IP é organizada em quatro camadas conceituais construídas sobre uma quinta camada que não faz parte do modelo, a camada intra-rede. A Figura abaixo mostra as camadas e o tipo de dados que é passado de uma para outra.

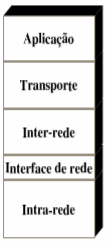


Arquitetura TCP/IP

Transporte

- Protocolos: TCP e UDP
  - TCP *(Transmission Control Protocol)*
    - serviço orientado a conexão
      - seqüenciação
      - controle de erro
      - controle de fluxo
  - UDP *(User Datagram Protocol)*
    - serviço não orientado a conexão
  - múltiplos pontos de acesso ao serviço (Portas)
    - multiplexação

Aplicação



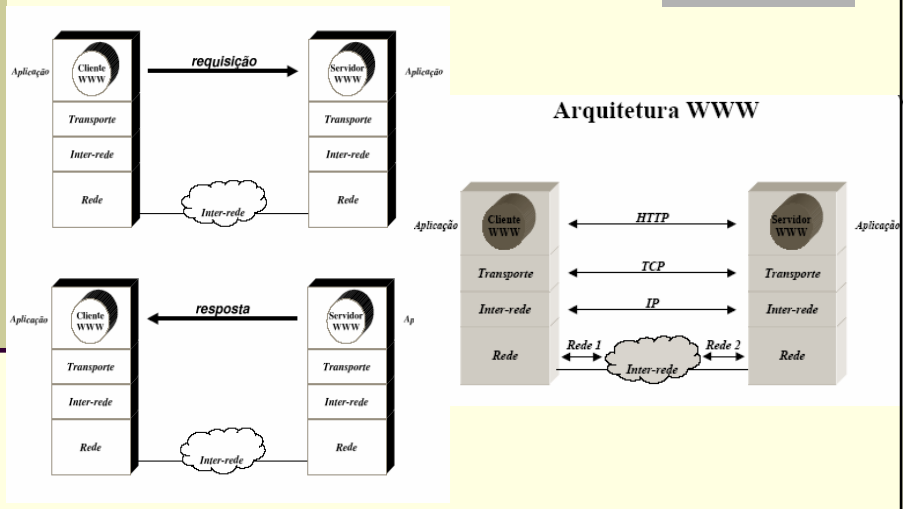
- Onde residem as aplicações inter-rede
- Utilizam os serviços oferecidos pela camada de transporte
- WWW
  - exemplo de aplicação TCP/IP
  - utiliza o serviço confiável da camada de transporte (TCP)
  - aplicação baseada no paradigma cliente/servidor

Arquitetura WWW

Modelo de Arquitetura Cliente-Servidor

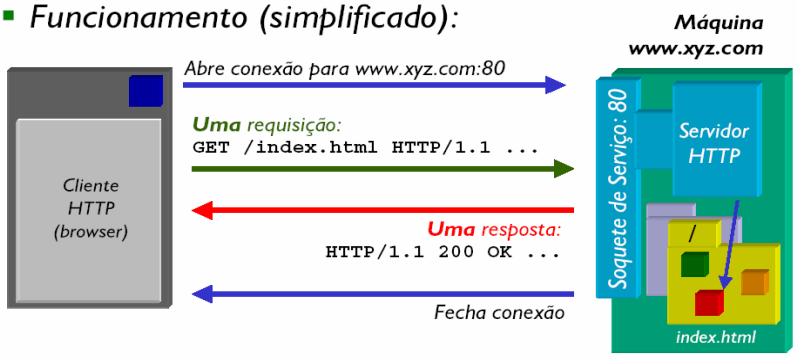
- O termo *servidor* se aplica a qualquer programa que oferece um serviço que pode ser alcançado através da rede
- Um programa em execução torna-se um *cliente* quando envia uma requisição a um servidor e aguarda por uma resposta

Arquitetura WWW



A Arquitetura WWW (cont)

- *Baseada em HTTP (RFC 2068)*
  - *Protocolo simples de transferência de arquivos*
  - *Sem estado (não mantém sessão aberta)*
- *Funcionamento (simplificado):*



Clientes WWW

- **Browsers**
  - **exibem e permitem a navegação através de documentos**
  - **exemplos**
    - **Netscape Navigator**
    - **Internet Explorer**
    - **Amaya**
    - **HotJava**
    - **NCSA Mosaic**
    - **Lynx**
- **Máquinas de busca**
- **Qualquer programa utilizando os serviços oferecidos por um servidor Web**

Servidores WWW

- **Não necessitam ser dedicados**
  - **Exemplos**
    - **Apache**
    - **Internet Information Server (IIS)**
    - **Netscape Enterprise Server**
    - **NCSA httpd**
    - **Jigsaw**

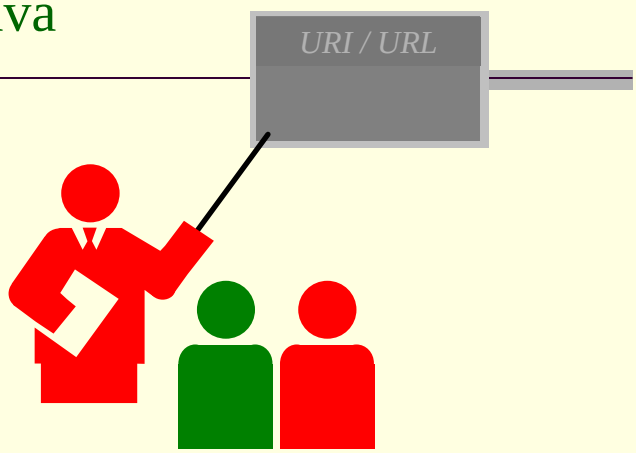
Conteúdo Estático x Conteúdo Dinâmico

- **Conteúdo estático**
  - ausência de um processamento adicional para entregar/exibir o documento
  - principal interação é pela navegação através de hiper-links
- **Conteúdo dinâmico**
  - inclusão de processamento adicional além da pura entrega de documentos e interpretação das marcações HTML

Porque Conteúdo Dinâmico

- **Permitir que sistemas de informação aproveitem a infra-estrutura oferecida pela Web**
  - simplicidade e portabilidade (em alguns casos) para os projetistas
  - infra-estrutura de distribuição para o projetista
  - simplicidade para o usuário final
    - browser como desktop
- **Aplicações**
  - home banking, comércio eletrônico, bibliotecas digitais, máquinas de busca, etc.

POO-Java



Universal Resource Identifier (URI)

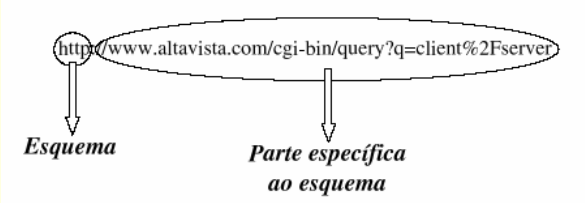
- **Como identificar os recursos (documentos)?**
  - **URL (Uniform Resource Locator)**
- **Como recuperar um documento?**
  - **HTTP (Hypertext Transfer Protocol)**
- **Como definir o formato do conteúdo dos documentos?**
  - **HTML (Hypertext Markup Language)**

Sintaxe de URIs

- **RFC 1630: descreve a notação de URIs em um nível sintático**
- **Separação em duas partes**
  - URI = scheme ":" scheme-specific-part
- **Esquema**
  - **identifica o esquema de definição dos nomes (naming scheme)**
  - **IANA *(Internet Assigned Numbers Authority)* uma lista dos esquemas e referências para suas definições**
- **Parte específica ao esquema**
  - **identificação propriamente dita de um objeto particular para um dado esquema**
  - **inteiramente dependente do esquema sendo utilizado**

URL e URN

- URI = scheme ":" scheme-specific-part



- URL – Uniform Resource Locator
  - Identificação e localização de recursos através de endereços
- URN – Uniform Resource Name
  - Identificação e localização de recursos através de nomes
- Definem as semânticas para URIs

URL

- Sintaxe para parte específica do esquema

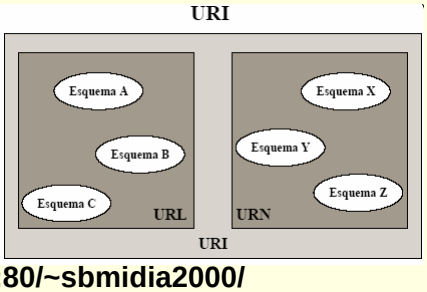
```
"//" [user [":" password] "@"] host [":"port] "/" url-path
```

- Principais esquemas URL registrados (IANA)

file	ldap	prospero
ftp	mailto	telnet
http	news	wais
https	nntp	

URL para esquema HTTP

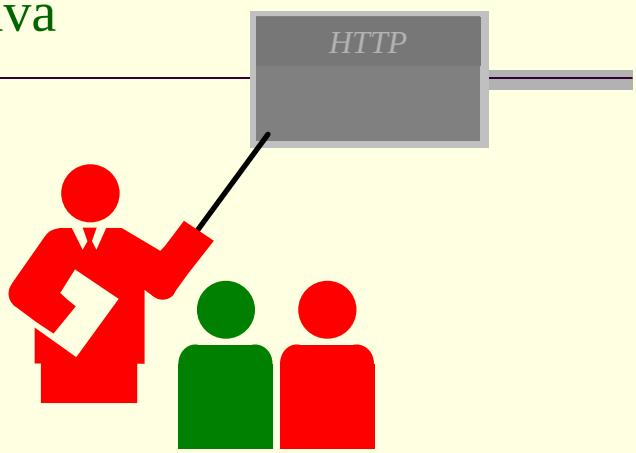
- **URI engloba URL e URN**
- **Exemplos de URL (esquema HTTP)**



  - **http://www.dimap.ufrn.br:80/~sbmidia2000/**
  - **http://www.telemidia.puc-rio.br/index.html**
  - **http://www.altavista.com/cgi-bin/query?q=client%2Fserver**
  - **http://139.82.95.14/index.html**

```
"http://" host [":"port] "/" [abs_path ["?" query ]]
```

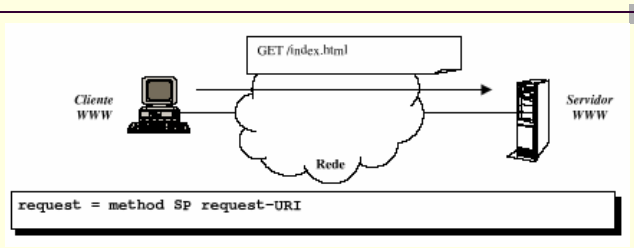
POO-Java



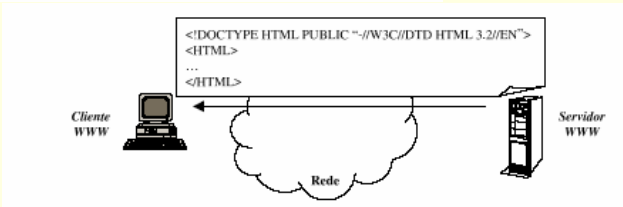
HTTP – Hypertext Transfer Protocol

- **Objetivo original**
  - **capacidade de recuperar, de um servidor, documentos simples baseados na mídia texto**
  - **protocolo leve e rápido**
- **Baseado em um modelo simples de arquitetura clienteservidor**
  - **pedido/resposta**
  - **protocolo sem estado**
- **Utiliza um serviço de transporte confiável, orientado a conexão (TCP)**
- **Protocolo mais utilizado na Internet, na atualidade**
- **Versões: HTTP/0.9, HTTP/1.0 e HTTP/1.1**

Mensagens HTTP/0.9




```
request = method SP request-URI
```




```
response = [message-body]
```


HTTP/1.0 – maio/96 (RFC 1945)

- Permitiu ao servidor responder códigos de erro e informações sobre a entidade, por exemplo, o tipo de conteúdo.
  - Definiu o conceito de tipo de mídia
    - MIME – Multipurpose Internet Mail Extensions, como padrão para identificação de conteudo.
      - MIME possui arquitetura aberta permitindo a uma aplicação incorporar suporte a novos tipos de dados
- Formato flexível de mensagem. O cliente passou a poder enviar dados ao servidor.
- Mecanismos de autenticação.

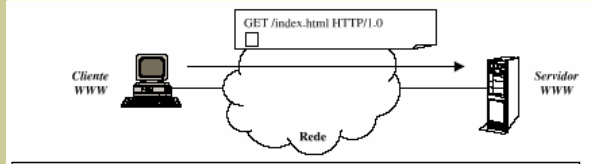
Alguns MIME Types

TYPE	SUBTYPE	TIPO DE DOCUMENTO
text	html	Arquivo HTML
text	plain	Arquivo texto simples
image	gif	Arquivo de imagem no formato GIF
image	jpeg	Arquivo de imagem no formato JPEG
audio	basic	Arquivo de audio PCM (RDSI – Lei Mi)
application	msword	Arquivo do aplicativo Microsoft Word
application	octet-stream	Arquivos executáveis ou binários genéricos
application	X-cceweb	Formato experimental

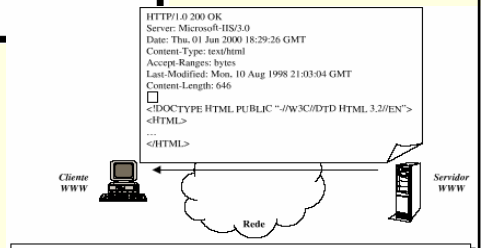
Tipos MIME

- text/plain **- arquivo no formato texto (ASCII);**
- text/html **- dosumento no formato HTML, o padrão para documentos Web;**
- application/zip **- arquivo compactado;**
- image/gif **- imagem codificada no formato GIF;**
- image/jpeg **- imagem codificada no formato JPEG.**

Mensagens HTTP/1.0



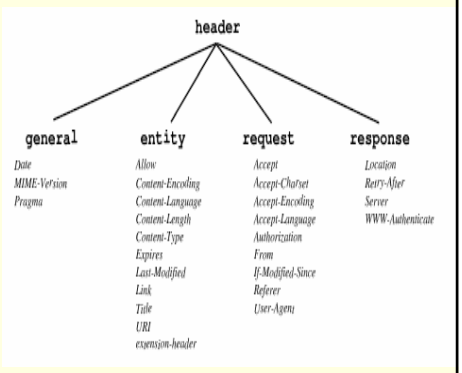

```
request = method SP request-URI SP HTTP-version CRLF
          *(general-header | request-header | entity-header)
          CRLF
          [message-body]
```



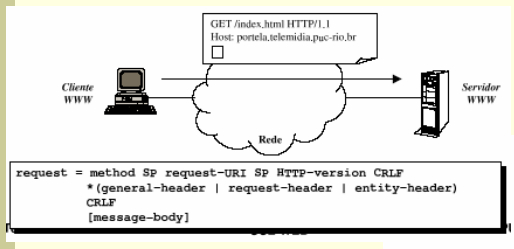

```
response = HTTP-version SP status-code SP reason-phrase CRLF
           *(general-header | response-header | entity-header)
           CRLF
           [message-body]
```


Cabeçalhos HTTP/1.0

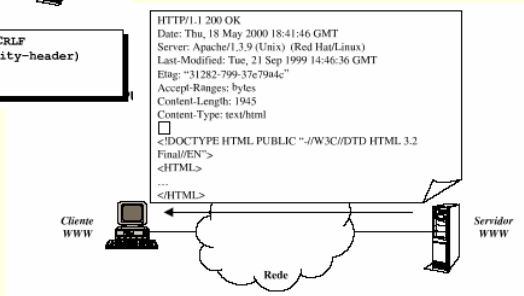
- General (requisição e resposta)
  - não se aplicam a entidades
- Entity (requisição e resposta)
  - usados para transmitir meta-informações de uma entidade
- Request (requisição)
  - contêm informações do cliente
- Response (resposta)
  - contêm informações que não podem ser transmitidas na status-line



Mensagens HTTP/1.1




```
request = method SP request-URI SP HTTP-version CRLF
          *(general-header | request-header | entity-header)
          CRLF
          [message-body]
```

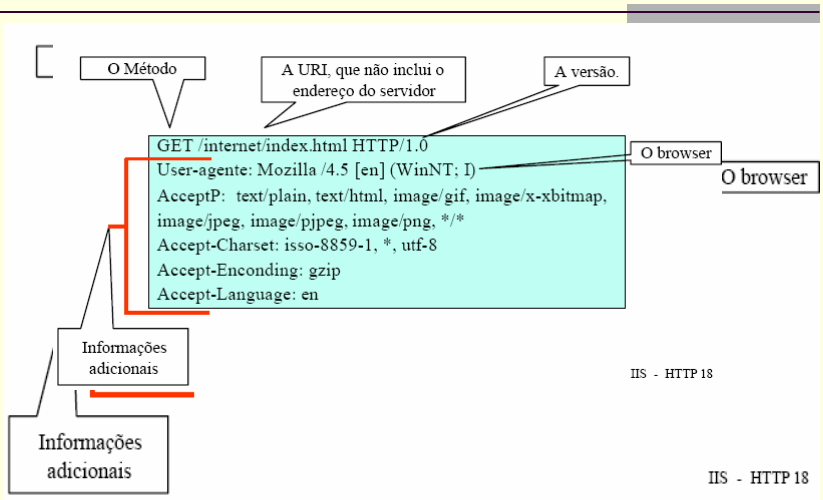



```
response = HTTP-version SP status-code SP reason-phrase CRLF
           *(general-header | response-header | entity-header)
           CRLF
           [message-body]
```

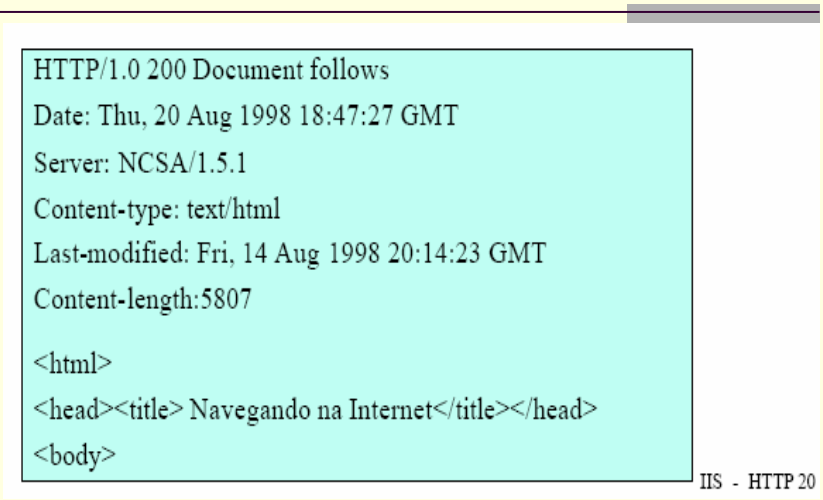

Métodos de Requisição em HTTP/1.1

- Métodos
  - GET - **retorna o objeto, ou seja, a informação requisitada.**
  - HEAD - **retorna somente informações sobre o objeto, como tamanho, data de criação etc.**
  - POST - **envia informações para o servidor Web**
  - PUT - **envia uma cópia de um** objeto/informação **para ser armazenado num servidor Web.**
  - DELETE - **apaga um objeto armazenado no servidor Web.**
  - OPTIONS
  - CONNECT
  - TRACE

Pedido HTTP completo



Resposta HTTP



Resposta HTTP

- **Uma resposta HTTP é formada por três elementos:**
  - **Linha de status**
    - **indicando sucesso ou falha do pedido.**
  - **Descrição da informação**
    - **contida na resposta (Metainformação/MIME).**
  - **A própria informação que foi requisitada.**

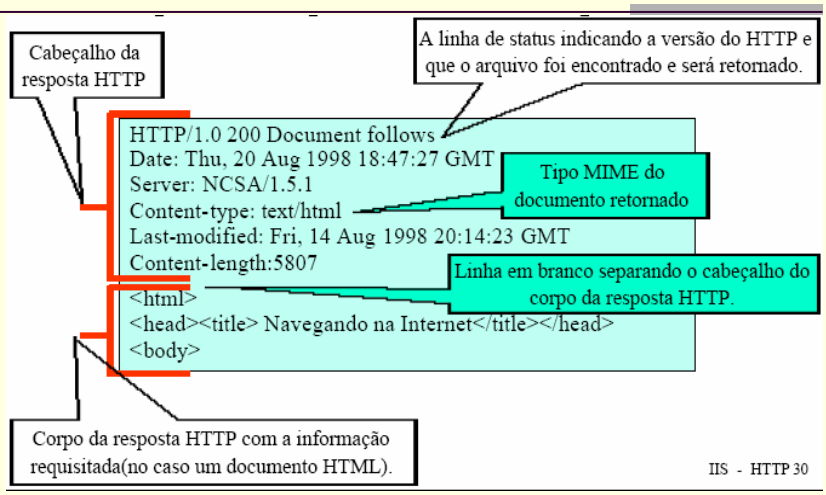
HTTP – Códigos de Retorno

- **A linha de status traz as seguintes informações:**
  - **A versão do protocolo HTTP;**
  - **O código de status que define o resultado do pedido;**
  - **Uma pequena frase explicando o que significa o código.**
- Código status é compostos de 3 dígitos, divididos em categorias em função do primeiro dígito
  - **1xx – informativo**
  - **2xx – sucesso**
  - **3xx – redireção**
  - **4xx – erro do cliente**
  - **5xx – erro do servidor**
- Podem ser estendidos

Resposta HTTP - Status

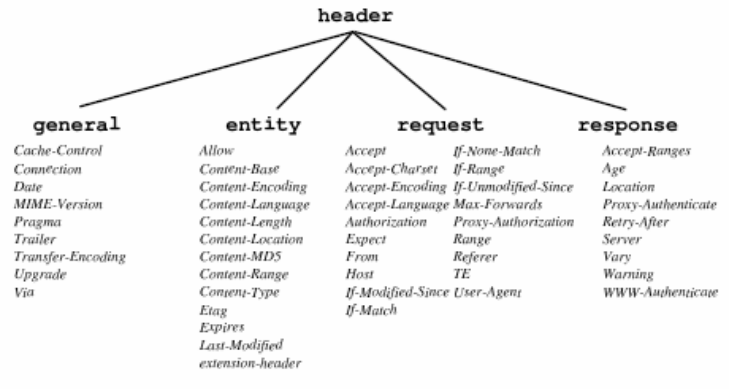
- **Os principais códigos de status existentes:**
  - 200 **(Document follows) - pedido bem sucedido. A informação requisitada será retornada.**
  - **401 (Unautorized) - a informação requisitada é de acesso restrito, sendo necessário se autenticar.**
  - 403 **(Forbidden) - acesso proibido.**
  - 404 **(Not found) - a informação requisitada não foi encontrada ou teve permissão de acesso negada. A primeira opção é muito freqüente na Internet e pode ocorrer por erro de digitação de uma URL.**
  - 500 **(Server Error) - erro no servidor Web. Comum quando da execução de scripts.**

Resposta HTTp



Métodos de Requisição em HTTP/1.1

- Cabeçalhos



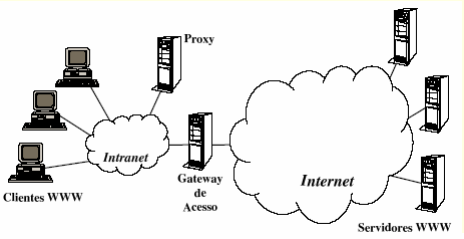
Modificações HTTP/1.1

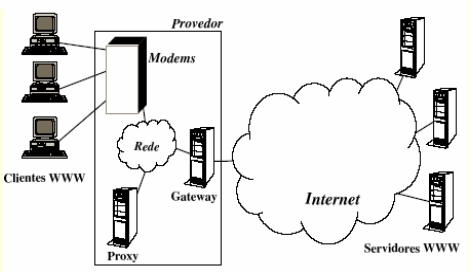
- Melhora no modelo de conexão TCP por requisição/resposta
  - HTTP persistente (P-HTTP)
    - Mantém uma conexão aberta durante várias requisições para um mesmo servidor
  - novos métodos de requisição
    - CONNECT, OPTIONS e TRACE
  - melhor suporte para cache
  - esquema mais seguro de autenticação
    - elimna a transferência de nome e senha de forma limpa
  - suporte à transferência parcial de entidades
  - suporte à negociação de conteúdo

Proxy

- Motivação
  - Cache
    - redução de carga no servidor e do trafego na conexão com a Internet
    - redução do tempo de resposta para os usuários
  - Segurança
    - filtragem de requisições
    - conversão de protocolos

Proxy – Cenários de uso





Criticas HTTP

- **Sem estado**
  - **requisições em paralelo numa mesma conexão precisam ser enfileiradas**
- **Implementação integral complexa**
- **Fundamentado no TCP como protocolo de transporte**
- **Requisições em um único sentido**
- **Ausência de um padrão para definição de extensões**
- **Mecanismo de negociação de conteúdo ainda restrito**

Cliente e servidor HTTP

- *Servidor HTTP*
  - *Gerencia sistema virtual de arquivos e diretórios*
  - *Mapeia pastas do sistema de arquivos local (ex: c:\htdocs) a diretórios virtuais (ex: /) acessíveis remotamente (notação de URI)*
- *Papel do servidor HTTP*
  - *Interpretar requisições HTTP do cliente (métodos GET, POST, ...)*
  - *Devolver resposta HTTP à saída padrão (código de resposta 200, 404, etc., cabeçalho RFC 822* e dados*
- *Papel do cliente HTTP*
  - *Enviar requisições HTTP (GET, POST, HEAD, ...) a um servidor. Requisições contém URI do recurso remoto, cabeçalhos RFC 822 e opcionalmente, dados (se método HTTP for POST)*
  - *Processar respostas HTTP recebidas (interpretar cabeçalhos, identificar tipo de dados, interpretar dados ou repassá-los.*

* *Padrão Internet para construção de cabeçalhos de e-mail*

Principais métodos HTTP (requisição)

- ***GET*** *- pede ao servidor um arquivo (informado sua URI) absoluta (relativa à raiz do servidor)*

```
GET <uri> <protocolo>/<versão>
<Cabeçalhos HTTP>: <valores> (RFC 822)
<linha em branco>
```

  - *GET pode enviar dados através da URI (tamanho limitado)*

```
<uri>?dados
```

  - *Método HEAD é idêntico ao GET mas servidor não devolve página (devolve apenas o cabeçalho)*
- ***POST*** *- envia dados ao servidor (como fluxo de bytes)*

```
POST <uri> <protocolo>/<versão>
<Cabeçalhos HTTP>: <valores>
<linha em branco>
<dados>
```


Cabeçalhos HTTP

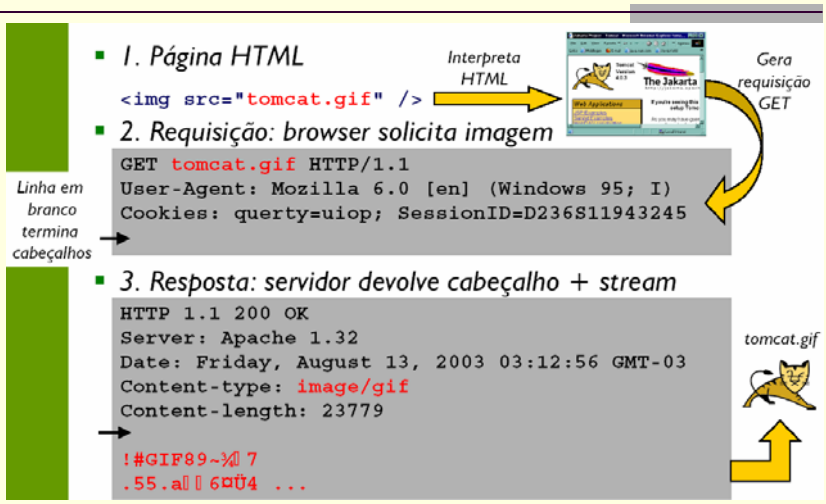
- *Na **requisição**, passam informações do cliente ao servidor*
  - *Fabricante e nome do browser, data da cópia em cache, cookies válidos para o domínio e caminho da URL da requisição, etc.*
- *Exemplos:*

```
User-Agent: Mozilla 5.5 (Compatible; MSIE 6.0; MacOS X)
If-Modified-Since: Thu, 23-Jun-1999 00:34:25 GMT
Cookies: id=344; user=Jack; flv=yes; mis=no
```

- *Na **resposta**: passam informações do servidor ao cliente*
  - *Tipo de dados do conteúdo (text/xml, image/gif) e tamanho, cookies que devem ser criados. endereço para redirecionamento, etc.*
- *Exemplos:*

```
Content-type: text/html; charset-iso-8859-1
Refresh: 15; url=/pags/novaPag.html
Content-length: 246
Set-Cookie: nome=valor; expires=Mon, 12-03-2001 13:03:00 GMT
```

Comunicação HTTP



POO-Java

Tecnologias no lado do Cliente

Plug-ins

- Tecnologia originalmente projetada pela Netscape
  - Netscape Navigator 2.0
  - Internet Explorer 3.0 passou também a oferecer suporte
- Permite também que aplicações existentes sejam facilmente integradas à Web
- Principal utilidade: exibir conteúdo cujo formato não é tratado pelo browser
  - conteúdos específicos das aplicações (PDF, PostScript, etc), áudio, vídeo

Plug-ins

- Módulo de código separado que se comporta como se fosse parte do browser
  - associado a um ou mais tipos de mídia (tipo MIME)
  - biblioteca de código nativo C
    - específico a uma plataforma (sistema operacional)
    - dependente da interface de programação do browser

Inserindo plug-ins em páginas HTML

- **Elementos HTML utilizados para inserção de plug-ins**
  - ***OBJECT***
    - **quando o browser não sabe tratar a especificação, o conteúdo do elemento deve ser apresentado**
    - ***Objects* podem ser aninhados**
      - ***<object data="clock.avi" type="video/msvideo" height="100%" width="100%" classid="http://microsoft.com/plugins/" >***
        - ***< object data="clock.gif" type="image/gif">***
        - ***<p>Hora certa.***
        - ***</object>***
      - ***</object>***

Inserindo plug-ins em páginas HTML

- **Elementos HTML utilizados para inserção de plug-ins**
  - ***EMBED* (não faz parte da especificação HTML 4.01)**
    - **elemento não mais padronizado na DTD HTML**
      - ***embed src="clock.avi" type="video/msvideo" width="100%" height="100%">***
- **Modos de exibição de um plug-in**
  - **Embutido, escondido ou página inteira**

Modelo de Execução de Plug-ins

- Plug-ins executam no mesmo espaço de memória do browser
  - DLLs, objetos compartilhados, bibliotecas compartilhadas, etc.
- Ciclo de vida de um plug-in está associado ao ciclo de vida da página que o aciona
- Quando o browser encontra em uma página uma referência (URI) para um arquivo que está associado a um Plug-in
  - browser carrega o código do plug-in na memória (se ainda não otiver feito)
  - cria uma nova instância do plug-in (o browser pode criar várias instâncias de um mesmo plug-in simultaneamente)
- Quando o browser sai da página que contém a referência para o plug-in ou tem sua janela fechada, a instância do plug-in é removida da memória
  - quando a última instância de um plug-in é removida, o código do plug-in é retirado da memória

Modelo de Execução de Plug-ins

- Quando um plug-in não está carregado em memória, o mesmo só ocupa espaço em disco
- Plug-ins são dependentes de plataforma e browser e não permitem interagir diretamente com o conteúdo HTML para por exemplo:
  - substituir imagens (simular animações)
  - simular menus de opções
  - mudar características de apresentação do documento de acordo com a interação do usuário
  - acrescentar conteúdo dinamicamente

Scripts

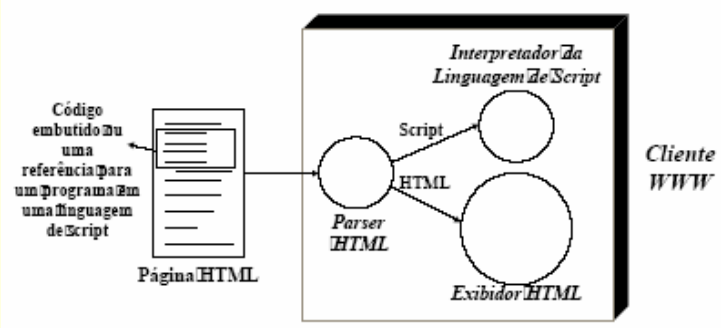
- **Usados para adicionar funcionalidades dinâmicas a páginas HTML estáticas. Página HTML carrega (de forma embutida ou através de uma referência) scripts que são executados pelo browser**
  - **alterar a especificação de apresentação dos elementos**
  - **acrescentar conteúdo dinamicamente ao documento**
  - **verificar a entrada de dados em um formulário**
  - **controlar o browser**
- **Principais linguagens de script utilizadas**
  - **Tcl, JavaScript (inicialmente chamado de LiveScript) - Netscape**
  - **Jscript e VBScript - Microsoft**

Scripts

- **Padrão para linguagens de script interpretadas no cliente**
  - **ECMAScript (European Computer Manufactors Associations Script)**
    - **padrão de junho de 1997, JavaScript e JScript são implementações**
- **Por que Linguagens de Script?**
  - **Interpretadas (não exigem compilação) oferecendo independência de plataforma**
  - **Simples de programar, sendo mais adequadas para usuários não experts em programação. Ideais para tarefas simples**
- **Desvantagens**
  - **Ineficiência e recursos limitados por isso são indicadas para tarefas simples**

Scripts

- **Para executar os scripts , o cliente WWW (browser) precisa de um interpretador da linguagem de script utilizada no documento**



Scripts

- **Pode aparecer várias vezes, tanto no *Head* como no *Body* do documento HTML. Fica a cargo de cada linguagem de script oferecer uma sintaxe para referenciar elementos HTML no documento**
  - **<p>Última atualização feita em:**
  - **<script type="text/javascript">**
  - **<!--** ← **evita que browsers que**
  - **document.write(document.lastModified);** **não dão suporte a scripts**
  - **-->** **exibam o conteúdo do**
  - **</script>** **script na tela !**
  - Informa a data da última modificação do documento

Scripts

- Exemplo de Script Associado a Eventos HTML
  - **<form>**
  - **<input type="button" value="Aperte!" onclick="alert('Clicou no botão!')">**
  - **</form>**
- **Os eventos intrínsecos normalmente são utilizados em conjunto com funções declaradas na área de SCRIPT**

Style Sheets

- **HTML pode ser visto como aplicação SGML**
  - **DTD SGML + Declaração SGML**
- **Style Sheets: especificação da apresentação do documento**
  - **linguagens para especificação de folhas de estilo**
  - ***DSSSL (Document Style Semantics and Specification Language)* para SGML**
  - ***CSS (Cascading Style Sheets)* para HTML**

HTML

- **Apesar do SGML focar em conteúdo, as primeiras versões do HTML tinham elementos e atributos para especificar a apresentação do documento**
  - **tamanhos e cores das fontes - <FONT>, <BASEFONT>, ...**
  - **alinhamentos e margens - <CENTER>, atributo *Align*, ...**
  - **formatação de texto - <U> (sublinhado), <STRIKE> (riscado)**
- **A partir da versão 4.0, o uso desses elementos e atributos foi desaconselhado (*deprecated*), devendo os mesmos serem substituídos por especificações nas *style sheets*. HTML 4 define três DTDs**

HTML

- ***Transitional DTD***
  - **utilizada só para interpretação,** não para gerar novos documentos**, pois contém elementos e atributos deprecated. Esses elementos e atributos eram utilizados para especificar formatação nas versões antigas e agora devem ser substituídos nas *style sheets* (CSS)**
- ***Strict DTD***
  - **utilizada para gerar novos documentos**
- ***Frameset DTD***
  - **usado para especificar documentos contendo uma estrutura de frames**

CSS - Cascading Style Sheets

- **Formatação do texto**
  - cores, fontes (tipo e tamanho), negrito, sublinhado, alinhamento dos parágrafos, ...
- **Formatação espacial**
  - margens superior, inferior, direita, esquerda, bordas, ...
- **Formatação das tabelas**
- **Sintaxe de CSS:**
  - Grupo de declarações
    - propriedade1:valor1; propriedade2:valor2
      - Ex.: color: red; font-size:16px
  - Associados a elementos HTML
    - elemento { grupo de declarações }
      - Ex.: P{color: red; font-size: 16px}

Por que CSS ?

- **Especificar diferentes apresentações para um único documento**
- **Possibilitar uma aparência consistente para um conjunto de documentos (site)**
- **Facilita a manutenção**
  - <H1><FONT COLOR="yellow" FACE="Arial">A heading</FONT></H1> ←no .html
  - H1 {font-family:Arial; color: yellow} ←no .css
- **HTML não foi projetado e não oferece suporte suficiente para controlar a apresentação**
  - ex.: espaçamento entre linhas, sombreado nas fontes, ...

Por que CSS ?

- Style sheet embutida no cabecalho do .html

```
<html>
  <head>
    <style type="text/css">
     <!--
          body {color: red}
          h1 {color: black}
     -->
    </style>
  </head>
  <body>
    ...
  </body>
</html>
```

- **Diretamente em cada elemento do documento - *inline style*:**
  - **qualquer elemento, exceto <html>, pode ter um estilo associado**
    - ** Este é um parágrafo. **

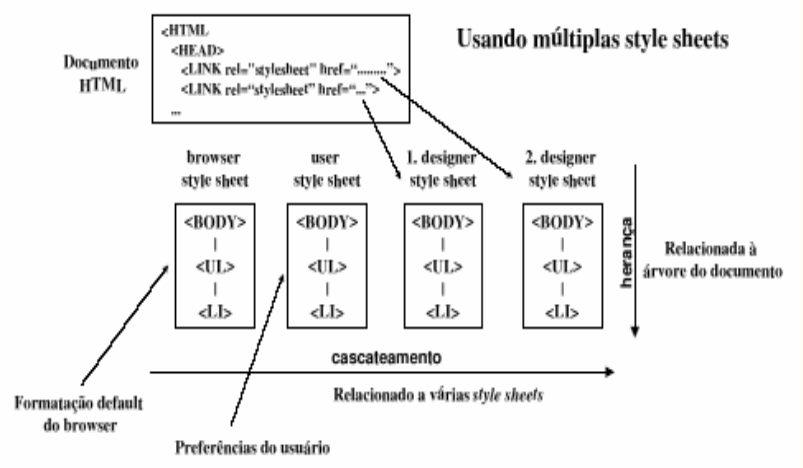
Por que CSS ?

- **Vantagem de usar a style sheet por fora**
  - **Reuso, facilidade de modificação e manutenção, maior poder de expressão para os recursos de apresentação**
  - **reduz o tamanho dos documentos e portanto o tempo de download**
- **Desvantagem de usar a style sheet por fora**
  - **quando uma página for lida offline, o usuário deve salvar, além da página HTML, o arquivo contendo a style sheet**

Por que CSS ?

- **Como especificar um estilo?**
  - **seletores associam grupos de declarações a elementos**
    - ***type selector* => Ex.: p{color: red; font-size: 16px}**
    - ***class selector***
  - **Documento HTML**
    - ** .... **
  - **Style Sheet**
    - **P.introductoryparagraph {color: blue}**
    - **.introductoryparagraph {color: blue}**

Múltiplas style sheets



Exemplo DHTML

- **Sublinha a âncora só quando o mouse estiver sobre a mesma**

```
<html>
<head>
      <style>
      .on {text-decoration:underline; color:blue;}
      .off {text-decoration:none;  color:black;}
      </style>
</head>
<body>
  <ul>
    <li> <a href="apresentacoes.html" class="off"
    onmouseover="this.classname='on';"
    onmouseout="this.classname='off';">Apresentações</a>
   <li> <a href="artigos.html" class="off"
    onmouseover="this.classname='on';"
    onmouseout="this.classname='off';">Artigos</a>
  </ul>
</body>
</html>
```


O que pode ser feito com Scripts + CSS

- **Acrescentar dinamicamente conteúdo a uma página HTML enquanto a página está sendo processada pelo browser**
- **Alterar o conteúdo de alguns elementos HTML**
- **Alterar características de apresentação dos elementos**
- **Obter a lista de links "elos" do documento**
- **Controlar o comportamento do browser**
  - **exibir mensagens através de dialog boxes ou na linha de status do browser**
  - **abrir e fechar novas janelas do browser**
  - **navegar no histórico de documentos simulando o comportamento dos botões back e forward do browser**
- **Ler e escrever valores em elementos para entradas de dados em qualquer formulário do documento**
- **Ler e escrever em Cookies**

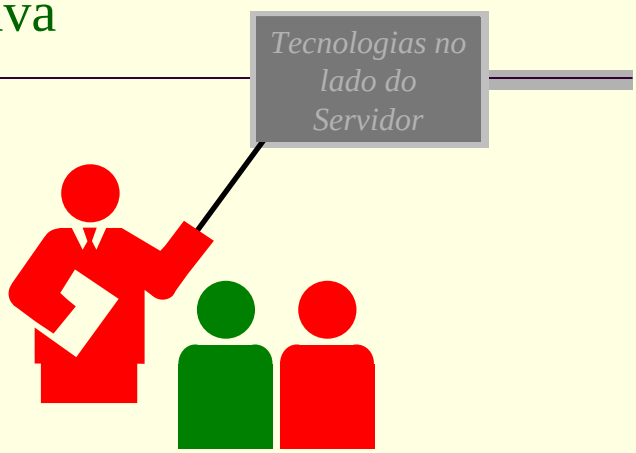
O que NÃO pode ser feito com Scripts

- As linguagens Script não possuem recursos gráficos
- Por razões de segurança, não se tem acesso para ler ou escrever em arquivos no cliente.
- Não oferece suporte para comunicação em rede a não ser baixar o conteúdo de uma URL
- Não tem capacidade para abrir múltiplas threads

Linguagens de Sistema

- Outra possibilidade para introduzir dinamismo em documentos na Web é através de uma linguagem de sistema
  - programas precisam ser distribuídos e executados em uma variedade de plataformas (clientes WWW)
  - programa (compilado) precisa ser independente de plataforma
- Applets Java
  - Programas que podem ser baixados de qualquer servidor WWW
  - Executados localmente em qualquer cliente WWW que saiba executar código Java
    - browsers possuem uma máquina virtual Java (JVM) embutida ou implementada como um plug-in
  - Quando o browser carrega uma página Web que faz referência a um applet, ele traz o applet de um servidor WWW (bytecode) e o executa localmente

POO-Java



Servidores WWW

- **Primeiros servidores HTTP**
  - **simples**
  - **traduziam o nome do recurso requisitado em um arquivo, enviando o conteúdo do arquivo como resposta**
- **Diversos fatores tornaram complexa a configuração apropriada e a gerência eficiente de servidores HTTP**
  - **servidores hospedando uma quantidade grande de documentos**
  - **aumento na complexidade do protocolo HTTP**

Servidores WWW

- **Fazem o mapeamento entre URL-path e o recurso local**
  - http://www.inf.puc-rio.br/index.html

URL - caminho virtual

  - c:\inetpub\wwwhome\index.html

caminho físico no sistema de arquivos do servidor

Servidores WWW

- **Tipos de recursos**
  - **estáticos**
    - **resposta é gerada pelo servidor sem a ajuda de um outro processo externo**
    - **tradução da URL-path em um path físico do recurso**
    - **envio da resposta acrescida de algumas informações (tipo MIME, tamanho, data de última modificação, etc.)**
  - **· dinâmicos**
    - **resposta é gerada dinamicamente através de algum processamento externo ao servidor**
    - **tradução da URL-path em um path físico de um programa**
    - **programas são normalmente identificados por extensões ou por prefixos especiais para URL-paths (diretórios virtuais)**

Tecnologias Server-side

- ***Estendem*** *as funções básicas de servidor HTTP:*
  - *CGI - Common Gateway Interface*
  - *APIs: ISAPI, NSAPI, Apache API, Servlet API, ...*
  - *Scripts: ASP, JSP, LiveWire (SSJS), Cold Fusion, PHP, ...*
- *Rodam do lado do servidor, portanto, não dependem de suporte por parte dos browsers*
  - *browsers fornecem apenas a interface do usuário*
- *Interceptam o curso normal da comunicação*
  - *Recebem dados via requisições HTTP (GET e POST)*
  - *Devolvem dados através de respostas HTTP*

CGI – Common Gateway Interface

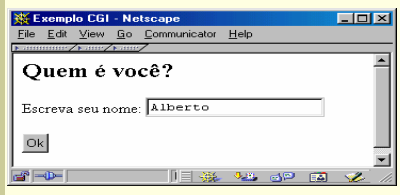
- ***Especificação*** *que determina como construir uma aplicação que será executada pelo servidor Web*
- *Programas CGI podem ser escritos em qualquer linguagem de programação. A especificação limita-se a determinar os formatos de entrada e saída dos dados (HTTP).*
- *O que interessa é que o programa seja capaz de*
  - *Obter dados de entrada a partir de uma requisição HTTP*
  - *Gerar uma resposta HTTP incluindo os dados e parte do cabeçalho*

- *Escopo: camada do servidor*
  - *Não requer quaisquer funções adicionais do cliente ou do HTTP*

Ineficiência do CGI

- A interface CGI requer que o servidor sempre execute um programa
  - Um novo processo do S.O. rodando o programa CGI é criado para cada cliente remoto que o requisita.
  - Novos processos consomem muitos recursos, portanto, o desempenho do servidor diminui por cliente conectado.
- CGI roda como um processo externo, logo, não tem acesso a recursos do servidor
  - A comunicação com o servidor resume-se à entrada e saída.
  - É difícil o compartilhamento de dados entre processos

CGI: Exemplo

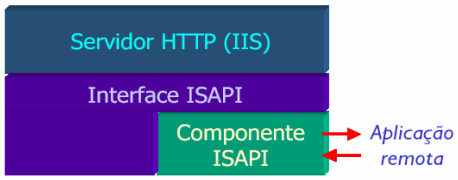



```
<HTML>
<HEAD>
<TITLE> Exemplo CGI </TITLE>
</HEAD>
<BODY>
<H2> Quem &eacute; voc&ecirc;? </H2>
<FORM METHOD=POST ACTION="../cgi-bin/uncgi/form-
nome">
<P>Escreva seu nome:
<INPUT TYPE="TEXT" NAME="Nome">
</P>
<P><INPUT TYPE="Submit" VALUE="Ok">
</FORM>
</BODY> </HTML>
```


```
#! /bin/sh
echo "Content-type: text/html"
echo
echo "<HTML><HEAD>"
echo "<TITLE>Ol&aacute;</TITLE>"
echo "</HEAD><BODY>"
echo "<P><H3>"
if [ ! -z "$WWW_Nome" ]; then
  echo "Ol&aacute;, "
  echo $WWW_Nome
else
  echo "Voc&ecirc; n&atide;o tem nome?"
echo "</H3></BODY></HTML>"
```

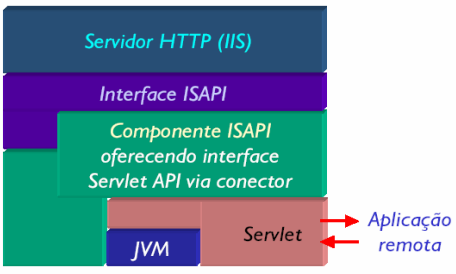
APIs do Servidor

- *Podem substituir totalmente o CGI, com vantagens:*
  - *Toda a funcionalidade do servidor pode ser usada*
  - *Múltiplos clientes em processos internos (threads)*
  - *Muito mais rápidas e eficientes (menos overhead)*
- *Desvantagens:*
  - *Em geral dependem de plataforma, fabricante e linguagem*
  - *Soluções proprietárias*
- *Exemplos*
  - *ISAPI (Microsoft)*
  - *NSAPI (Netscape)*
  - *Apache Server API*
  - *??SAPI*



Servlet API

- *API independente de plataforma e praticamente independente de fabricante*
- *Componentes são escritos em Java e se chamam servlets*
- *Como os componentes SAPI proprietários, rodam dentro do servidor, mas através de uma Máquina Virtual Java*
- *Disponível como 'plug-in' ou conector para servidores que não o suportam diretamente*
  - *Desenho ao lado mostra solução antiga de conexão com IIS*
- *Nativo em servidores Sun, IBM, ...*



Vantagens dos Servlets

- ... sobre CGI
  - Rodam como parte do servidor (cada nova requisição inicia um novo thread mas não um novo processo)
  - Mais integrados ao servidor: mais facilidade para compartilhar informações, recuperar e decodificar dados enviados pelo cliente, etc.
- ... sobre APIs proprietárias
  - Não dependem de único servidor ou sistema operacional
  - Têm toda a API Java à disposição (JDBC, RMI, etc.)
  - Não comprometem a estabilidade do servidor em caso de falha (na pior hipótese, um erro poderia derrubar o JVM)

Problemas dos Servlets, CGIs e APIs

- Para gerar páginas dinâmicas (99% das aplicações), é preciso embutir o HTML ou XML dentro de instruções de uma linguagem de programação:

```
out.print("<h1>Servlet</h1>");
for (int num = 1; num <= 5; i++) {
    out.print("<p>Parágrafo " + num + "</p>");
}
out.print("<table><tr><td> ... </tr></table>");
```

- Maior parte da informação da página é estática, no entanto, precisa ser embutida no código
- Afasta o Web designer do processo
  - Muito mais complicado programar que usar HTML e JavaScript
  - O design de páginas geradas dinamicamente acaba ficando nas mãos do programador (e não do Web designer)

Solução: scripts de servidor

- *Coloca a linguagem de programação dentro do HTML (e não o contrário)*

```
<h1>Servlet</h1>
 <% for (int num = 1; num <= 5; i++) { %>
   <p>Parágrafo <%= num %></p>
 <%}%>
 <table><tr><td> ... </tr></table>
```

- *Permite o controle da aparência e estrutura da página em softwares de design (DreamWeaver, FrontPage)*
- *Página fica mais legível*
- *Quando houver muita programação, código pode ser escondido em servlets, JavaBeans, componentes (por exemplo: componentes ActiveX, no caso do ASP)*

Controle de sessão

- *HTTP não preserva o estado de uma sessão. É preciso usar mecanismos artificiais com CGI (ou qualquer outra tecnologia Web)*
  - *Seqüência de páginas/aplicações: desvantagens: seqüência não pode ser quebrada; mesmo que página só contenha HTML simples, precisará ser gerada por aplicação*
  - *Inclusão de dados na URL: desvantagens: pouca flexibilidade e exposição de informações*
  - *Cookies (informação armazenada no cliente): desvantagens: espaço e quantidade de dados reduzidos; browser precisa suportar a tecnologia*

Cookies

- *Padrão Internet (RFC) para persistência de informações entre requisições HTTP*
- *Um cookie é uma pequena quantidade de informação que o servidor armazena no cliente*
  - *Par nome=valor. Exemplos: usuario=paulo, num=123*
  - *Escopo no servidor: domínio e caminho da página*
  - *Pode ser seguro*
  - *Escopo no cliente: browser (sessão)*
  - *Duração: uma sessão ou tempo determinado (cookies persistentes)*
- *Cookies são criados através de cabeçalhos HTTP*

```
Content-type: text/html
Content-length: 34432
Set-Cookie: usuario=ax343
Set-Cookie: lastlogin=12%2610%2699
```


Exemplo com cookies: Loja Virtual