6. Fundamentos HTTP

O PROTOCOLO HTTP consiste de um conjunto de comandos, códigos e procedimentos que garantem a comunicação entre um browser e um servidor Web. Para desenvolver programas usando CGI, em qualquer linguagem, é essencial conhecer os fundamentos do protocolo HTTP – Hypertext Transfer Protocol. Este é o objetivo deste capítulo.

6.1. O que é HTTP

HTTP significa Hypertext Transfer Protocol. É o principal protocolo utilizado nas comunicações na Internet e o protocolo que dá sustentação ao serviço conhecido como World Wide Web. A especificação, elaborada pelo W3C[1], o define como:

> *HTTP é um protocolo de nível de aplicação para sistemas de hipermídia, colaborativos e distribuídos. É um protocolo genérico, sem estado e orientado a objetos que pode ser usado para diversas tarefas, tais como servidores de nomes e sistemas de gerenciamento de objetos distribuídos, através da extensão de seus métodos de requisição [RFC2068].*

A W3C define HTTP como um protocolo sem estado (stateless). Isto quer dizer que o protocolo não preserva informações de estado entre requisições ao servidor. Se um servidor recebe uma seqüência de 3 requisições de três clientes ao mesmo tempo ele não saberá separar as requisições por cliente de forma a considerar informações da primeira requisição que podem influenciar na segunda ou terceira. Isto quer dizer que não será possível, por exemplo, implementar aplicações que dependam de informações de estado definidas em páginas separadas, como aplicações de comércio online, sem recorrer a mecanismos externos ao protocolo.

6.2. Métodos

HTTP define um conjunto de mensagens chamados de *métodos de requisição HTTP* que são usados pelo cliente, geralmente um browser como o Netscape Navigator ou Microsoft Internet Explorer, para enviar uma requisição ao servidor. Esta requisição é, na maior parte das vezes, uma solicitação para que o servidor devolva uma página HTML, uma imagem, um com-

[1] World Wide Web Consortium (http://www.w3.org)

ponente ou recurso multimídia. O método HTTP usado neste tipo requisição é o método **GET** que, além de requisitar páginas, imagens ou programas, pode requisitar informações geradas na saída de um dispositivo ou programa executado pelo servidor.

Vários outros métodos são definidos pelo protocolo HTTP. O protocolo também garante a possibilidade de extensões. A versão HTTP 1.1 define 7 métodos básicos, que são: `GET`, `HEAD`, `POST`, `PUT`, `DELETE`, `TRACE`, `OPTIONS`. Um servidor Web mínimo, que suporte HTTP 1.1, deve ser capaz de entender pelo menos os métodos `GET` e `HEAD`. Um servidor Web típico recebe normalmente requisições `GET`, `HEAD` e `POST`, sendo a grande maioria requisições `GET`.

As mensagens passadas em HTTP podem ser requisições ou respostas. As requisições são formadas e enviadas por um cliente HTTP. As respostas são retornadas por um servidor. A porta de comunicações onde o servidor aguarda requisições é, por default, a porta 80. Se outra porta for utilizada, ela deverá constar da requisição.

6.3. Requisições HTTP e método GET

A requisição HTTP obedece a um formato padrão [RFC2068]:

```
<método HTTP> <URL do recurso solicitado> <versão HTTP>
<cabeçalhos formato RFC 822>
<dados>
```

O cabeçalho inclui informações adicionais essenciais ou não. Tipicamente contém o nome do host (se separado da URL na requisição), a data, o nome e versão do cliente, etc. Sempre termina com uma linha em branco. Algumas requisições enviam dados adicionais após o cabeçalho. Nestes casos, o cabeçalho deve conter ainda o tamanho e tipo dos dados enviados.

O método `GET` tem a finalidade de recuperar informações do servidor. A especificação [RFC] o define como um método seguro e idempotente, ou seja, deve produzir o mesmo resultado se repetido várias vezes. `GET` não envia dados após o cabeçalho e possui a seguinte sintaxe mínima:

```
GET <URL_relativa> HTTP/1.1
Host: <host>
<linha em branco (CRLF)>
```

Por exemplo:

```
GET /index.html HTTP/1.1
Host: www.canopus.com.br
```

Faz uma conexão à porta 80 da máquina www.canopus.com.br e solicita ao servidor o arquivo `index.html` localizado na raiz de documentos. A requisição acima normalmente contém várias outras linhas de cabeçalho antes da linha em branco. Abaixo está uma requisição para buscar a página `index.html`, formada ao se clicar em um link no Microsoft Internet Explorer:

```
GET /index.html HTTP/1.1
User-Agent: Mozilla 4.0 (compatible; MSIE 4.01; Windows98)
Accept: image/gif, image/x-bitmap, image/jpeg, image/pjpeg, image/png
Host: www.canopus.com.br
```

Os cabeçalhos presentes dependem do método usado na requisição. Alguns dos cabeçalhos mais usados em requisições GET são:

- **User-Agent** - contém uma string que identifica o browser que faz o pedido.
- **Host** - endereço internet do servidor.
- **Date** – data da requisição.
- **Accept** - pode haver vários campos accept. Cada um contém um tipo MIME (tipo/subtipo) que é aceito pelo browser que faz o pedido. Exemplos: Accept: text/plain, text/html; Accept: text/x-dvi; q=.8; mxb=100000; mxt=5.0, text/x-c; Accept: *.*, q=0.1
- **Accept-Encoding** - tipos de codificação aceitas pelo browser.
- **Referer** – contém uma string com a URL do documento onde está a referência (link) do recurso solicitado. Não aparece se requisição não foi causada por um link.
- **If-Modified-Since** – usada com o método GET para torná-lo condicional. Se o documento não mudou desde a última vez em que foi recuperado, ele não será enviado e o browser deve buscá-lo no cache.

6.4. Respostas HTTP

Após receber uma requisição, um servidor HTTP sempre responde. A resposta pode consistir de uma mensagem de erro, de redireção ou de sucesso. A sintaxe das respostas é:

```
<versão HTTP> <código de status> <informações de status>
<cabeçalhos formato RFC822>
<dados>
```

O código de status é um número de três dígitos que pode ter a forma `1xx`, `2xx`, `3xx`, `4xx` ou `5xx`, onde `x` é qualquer dígito de `0` a `9`. Códigos começando em `4` ou `5` indicam mensagens de erro. Iniciando em `2` indicam sucesso. `3xx` é usado para indicar requisição incompleta ou redirecionamento. `1xx` é usado para retornar informações durante o processamento. Os códigos de erro `4xx` indicam que o cliente deve ter errado. Os códigos `5xx` indicam que o servidor está consciente que o erro é dele, e não pode ser solucionado pelo cliente. As informações de status contém um texto descrevendo a ocorrência. Veja alguns codigos:

- **OK 200** – A requisição foi atendida.
- **No Response 204** – O servidor recebeu a requisição mas não há nada a ser retornado. Browser deve se manter na mesma página.
- **Bad request 400** - A sintaxe do pedido era inadequada ou impossível de ser satisfeita.
- **Unauthorized 401** – O cliente não tem autorização para obter a informação

- **Forbidden 403** – O objeto não pode ser enviado. Não adianta ter autorização.
- **Not found 404** – O servidor não encontrou nada que combinasse com a URL que recebeu.
- **Internal Error 500** – Erro interno
- **Not implemented 501** – Recurso não implementado
- **Moved 301** – a informação se mudou para outro lugar. A resposta contém a nova URL do recurso.
- **Not Modified 304 –** a página não foi modificada desde o último acesso. Recupere-a do cache.

Por exemplo, uma resposta iniciando em:

```
HTTP 1.0 404 Not Found
```

será enviada ao browser caso o servidor não consiga atender à requisição do browser por não encontrar o recurso solicitado, ou

```
HTTP 1.0 200 OK
```

em caso de sucesso.

Como resposta à requisição `GET` acima (seção anterior), o servidor poderia ter retornado:

```
HTTP/1.0 200 OK
Server: Netscape-FastTrack/2.0a
Date: Mon, 4 Jan 1999 18:37:47 GMT
Content-type:text/html
Content-length: 19920
Last-modified: Sun, 13 Dec 1998 5:11:59 GMT

<!DOCTYPE ... </html> (19920 bytes de informação)
```

Poderia também ter retornado um erro. Por exemplo, ao requisitar uma página HTML ao `servletrunner` (servidor especial da Sun que só serve servlets) ele retorna um erro:

```
HTTP/1.0 403 Forbidden
Server: servletrunner/2.0
Content-type:text/html
Content-length: 135
Date: Mon, 4 Jan 1999 18:39:55 GMT

<html><head><title></title></head>
<h1>403 Forbidden</h1><body>
Will not serve files, only servlets
</body></html>
```

Observe a linha em branco separando os dados (o conteúdo de uma página) do cabeçalho (gerado pelo servidor).

Os cabeçalhos que o servidor devolve ao cliente referem-se a propriedades do objeto retornado mais informações de ambiente. A maioria deles são gerados automaticamente pelo servidor. Quando se usa CGI, é preciso completar o cabeçalho com uma linha em branco. Antes de completar o bloco do cabeçalho, porém, o programador CGI pode acrescentar novos cabeçalhos, como o cabeçalho "Content-type", que informa o tipo do conteúdo gerado pelo CGI, e geralmente é obrigatório. Alguns dos principais cabeçalhos de resposta estão listadois a seguir:

- **Content-Length:** Conteúdo em bytes da informação enviada.
- **Content-Type:** indica o tipo de mídia dos dados que estão sendo enviados. Tipo MIME ou tipos de multiplas partes: Multipart/alternative, Multipart/related, Multipart/mixed ou Multipart/parallel *Exemplo*: Content-type: text/html.
- **Content-Encoding:** codificação utilizada (se utilizada). Formatos suportados pelo padrão W3C são x-compress e x-gzip.
- **Date:** indica a data de criação do objeto. O formato deve seguir RFC850.
- **Server:** inclui informações sobre o servidor.
- **Expires:** indica uma data em que a informação no documento não é mais válida (usada por browser que fazem cache). O formato é o mesmo de Date.
- **Last-Modified:** este cabeçalho indica a data e a hora da última modificação de um recurso. O formato é igual do de Date.
- **Location**: define a localização exata de um recurso, através de uma URL. Este deve ser usado em separado de qualquer outro. Se o browser ler uma linha contendo Location: http://novaURL.com/novapag.html, a janela vai ser redirecionada para buscar as informações na nova URL.
- **Set-Cookie:** Define um cookie.
- **Pragma** – pode ter o valor "no-cache" indicando que o servidor deve retornar um documento mesmo que ele não tenha mudado (usado pelo Reload do browser).

O cabeçalho `Content-type` é um dos mais importantes. Ele informa o tipo dos dados que o browser está recebendo. Sem ele, o browser não seria capaz de distinguir uma página HTML de uma imagem GIF. O cabeçalho `Content-type` contém o tipo MIME dos dados que virão a seguir. O cabeçalho `Content-length`, não menos importante, informa quantos caracteres o browser terá que ler antes que toda a informação seja recebida.

Sempre que se cria uma página dinâmica, produzida por um programa CGI, é necessário montar (imprimir) a parte do cabeçalho HTTP que o servidor não faz sozinho, que é o Content-Type) e terminar o bloco com uma linha em branco.

O servidor Web sempre retorna alguma coisa. Qualquer coisa que esteja armazenada nos diretórios de documentos do servidor HTTP será retornada para o browser se houver permissão. Pode ser uma página HTML, mas também pode ser um arquivo executável Windows. Se a

raiz de documentos (/) de um servidor Web rodando em Windows é `C:\WINDOWS\COMMAND` e o browser solicitar

```
GET /format.com HTTP/1.0
```

o servidor devolverá:

```
HTTP/1.0 200 OK
Server: Caranguejeira/1.0
Date: Mon, 4 Jan 1999 21:10:33 GMT
Content-type: application/x-msdownload
Content-length: 41604
Last-modified: Sat, 24 Aug 1996 11:11:00 GMT

é¨Converted▯▯▯▯MZx▯U▯I▯ ▯▯▯ÿÿ▯▯ð (...)
```

6.5. Método HEAD

O método `HEAD` é semelhante ao método `GET`. A única diferença entre eles é a resposta do servidor. O servidor que recebe `HEAD`, retorna o cabeçalho completo dos dados requeridos mas não retorna os dados.

`HEAD` é normalmente usada para verificar a data de última modificação do arquivo, o tamanho do arquivo, tipo dos dados, etc. para fins de otimização da transferência ou gerenciamento de cache ou proxy.

6.6. Extensões HTTP

A maior parte das requisições de um browser geralmente são para recuperar algum recurso no servidor. Às vezes, porém, o cliente quer que o servidor execute um programa. É preciso que o servidor esteja configurado para identificar o que o cliente quer a partir da URL da requisição. Um servidor que suporte CGI, por exemplo, pode identificar um pedido de execução se a URL solicita um recurso contido em um diretório especial:

```
GET /cgi-bin/format.com HTTP/1.0
```

aqui chamado de CGI-BIN. Desta vez, o programa será executado na máquina servidora e a saída do programa poderá ser retornada para o cliente.

A extensão de um arquivo também é freqüentemente usada pelo servidor para distinguir um recurso estático de um recurso executável. Em servidores Microsoft IIS:

```
GET /texto.asp HTTP/1.0
```

fará com que o servidor tente interpretar comandos especiais na página HTML `texto.asp` e envie o resultado para o browser.

Freqüentemente, um programa no servidor requer parâmetros enviados pelo cliente para poder executar corretamente. Tome por exemplo um contador que precisa saber o nome do

arquivo onde armazenar a contagem. A linha de informações que segue a "?" em uma URL tem um comprimento limitado e é chamada de *Query String* e pode ser usada para passar parâmetros. A requisição ao contador poderia ser da forma:

```
GET /cgi-shl/count.pl?arq=pg34.txt HTTP/1.0
```

O programa em Perl `count.pl` será então executado pelo servidor recebendo o parâmetro `arq=pg34.txt` como argumento.

6.7. Método POST

`POST` é usado pelo cliente para enviar dados ao servidor durante uma requisição. O alvo da requisição deve ser um agente capaz de realizar alguma coisa com os dados recebidos. A URL da requisição `POST`, portanto, geralmente contém informações que fazem o servidor executar algum procedimento em vez de simplesmente retornar uma imagem ou página.

`POST` é indicado para enviar grandes quantidades de dados ao servidor já que não é limitado como o *Query String* usado para enviar informações via `GET`. O método `POST` não precisa ser seguro ou idempotente. Pode provocar alterações nos dados e duas requisições não precisam retornar os mesmos dados.

Veja um exemplo com o contador mostrado anteriormente. Se o browser enviasse uma requisição `POST` em vez de `GET` os dados seriam enviados da seguinte maneira:

```
POST /cgi-shl/count.pl HTTP/1.0
Content-type: text/plain
Content-length: 12

arq=pg34.txt
```

Mas o browser não usa `POST` a não ser que seja instruído a fazê-lo. A forma mais comum de fazer isto, é usar formulários HTML, apresentados superficialmente na próxima seção.

6.8. Exercícios

3.1 Conecte-se via telnet à porta do servidor Web instalado em qualquer servidor. Digite os comandos GET e POST como mostrados nos exemplos acima. Veja as respostas retornadas pelo servidor. Se quiser, defina alguns cabeçalhos. Quando você imprimir duas novas-linhas, o pedido será enviado. Logo que o servidor atender ao seu pedido, ele desconectará.

7. A Interface CGI

CGI é um serviço oferecido pelo servidor. É uma extensão ao serviço HTTP. Servidores que suportam CGI podem ser configurados a executarem programas externos ao servidor, passar-lhes parâmetros recebidos de uma requisição do browser e redirecionar sua saída como uma resposta HTTP.

CGI significa Common Gateway Interface. Sua finalidade é permitir que o servidor funcione como gateway para aplicações externas. A especificação define regras para a construção de aplicações que poderão ser executadas pelo servidor HTTP. As regras só se aplicam à entrada e saída das aplicações. É uma especificação de "caixa-preta". Não interessa o conteúdo. As aplicações podem ser escritas em qualquer linguagem desde que:

- sejam capazes de decodificar strings passadas como argumentos de linha de comando na entrada
- construam corretamente o final do cabeçalho HTTP para os dados que gerarem na saída

Com esta flexibilidade, CGI tornou-se extremamente popular. Existem aplicações em C, C++, Perl, AppleScript, Java, MS-DOS, Basic, VB, Pascal, Bourne-shell e várias outras linguagens rodando em todas as platarormas onde há servidores Web.

7.1. Formulários HTML

A interface de formulários oferecida por HTML permite que o leitor de uma página Web possa enviar informações de volta para o servidor através de componentes como botões, caixas de checagem, caixas de texto, caixas de seleção pull-down, etc. Os objetos de formulários em HTML são definidos dentro de um bloco `<form> ... </form>`. Podem ter um botão para enviar os dados e outro para limpá-los (reinicializá-los aos valores *default*) e vários campos de entrada de dados.

Os atributos mais comuns de `<form>` são a URL que inicia a aplicação que irá manusear os dados e opcionalmente, o método que será usado na requisição. Por exemplo, o formulário:

```
<FORM ACTION="/cgi-bin/bd.exe" METHOD="POST" >
...
</FORM>
```

quando for enviado, usará o método `POST` ao requisitar a execução de `/cgi-bin/bd.exe` ao servidor. Se o atributo `METHOD` for omitido, a requisição será realizada usando o método `GET`.

Cada componente de entrada de dados de um formulário tem um atributo `NAME` e um atributo `VALUE`. `NAME`, contém o `nome` que será utilizado pelo CGI para recuperar o `valor` que o leitor da página enviou, que fica armazenado em `VALUE`. Cada objeto de formulário envia seus dados no formato `nome=valor`.

Geralmente existem vários campos de entrada de dados, cada um com seu nome e valores diferentes. Ao enviar um formulário, todos os valores são concatenados com um "`&`". Não pode haver espaço entre os nomes e valores, portanto os espaços são convertidos em "+". Finalmente, caracteres que não são representados em ASCII e caracteres reservados são convertidos em hexadecimal, da forma `%hh`, antes de serem enviados.

Tome como exemplo o formulário mostrado na figura 12-3. Quando o leitor apertar o botão enviar, o browser enviará a seguinte requisição ao servidor:


```
<FORM ACTION="/cgi-bin/bd.exe"
      METHOD="POST" >
<p>Nome: <input type=text name=nome>
<p>Idade: <input type=text name=idade>
<p>Escolha uma cor:
<select name=cor>
  <option value=red>Vermelho</option>
  <option value=yellow>Amarelo</option>
  <option value=blue>Azul</option>
</select>
<p><input type=submit value="Enviar">
<input type=reset value="Limpar">
</FORM>
```

```
POST /cgi-bin/bd.exe HTTP/1.0
Content-type: text/plain
Content-length: 36

nome=Jo%E3o+da+Silva&idade=23&cor=yellow
```

Esses dados são enviados ao servidor que os repassa ao `/cgi-bin/bd.exe`. A requisição também pode ser realizada usando o método `GET`, e os dados serão enviados no *Query String*:

```
GET /servlet/bdservlet?nome=Jo%E3o+da+Silva&idade=23&cor=yellow HTTP/1.0
```

7.2. Programas CGI

Programas que seguem a especificação CGI devem ser capazes de obter e decodificar um string como

```
nome=Jo%E3o+da+Silva&idade=23&cor=yellow HTTP/1.0
```

A obtenção dos dados geralmente também requer a capacidade de ler a entrada padrão, possivelmente ler um arquivo ou conexão de rede e obter informações armazenadas em variáveis de ambiente definidas pelo servidor.

A decodificação consiste em:

- converter os códigos hexadecimais `%hh` nos seus caracteres equivalentes
- converter "+" em espaço
- separar os pares nome/valor pelo "`&`"
- identificar, para cada par nome/valor, o nome (à esquerda do "=") e o valor (à direita) e montar uma associação que permita obter o valor a partir do nome.

Com os dados obtidos, o programa CGI pode realizar qualquer tarefa como por exemplo, iniciar uma outra aplicação, montar uma requisição SQL para envio a um banco de dados, recuperação de uma imagem de um dispositivo externo. No final, deve retornar para o cliente uma imagem, uma página ou outra seqüência de bytes qualquer. O servidor não tem como saber o tipo da seqüência de bytes para montar o cabeçalho para o cliente então deixa-o incompleto para que seja completado pelo programa CGI.

O programa precisa então imprimir na sua saída no mínimo:

- cabeçalho "`Content-type`", informando o tipo de dados enviados de volta ao servidor
- uma linha em branco, indicando o fim do cabeçalho
- os dados.

Supondo que o programa CGI seja escrito em C, para imprimir uma mensagem em HTML com o texto "Hello CGI World" em resposta a uma requisição de um browser, ele conteria as linhas:

```
printf("Content-type: text/html\n");  /* observe o '\n' extra! */
printf("<html><body><h1>Hello CGI World</h1></body></html>");
```

7.3. Variáveis de Ambiente

A requisição do browser gera várias variáveis de ambiente no servidor. Outras são definidas pelo servidor e pelo sistema. Todas têm um escopo definido pelo sistema, servidor, aplicação, cliente ou requisição. As principais variáveis de ambiente suportadas pela maioria dos servidores são:

SERVER_SOFTWARE

O nome e a versão do software do servidor que responde ao pedido.

Formato: nome/versão

Exemplo: CERN/3.0

SERVER_NAME
O nome da máquina onde roda o servidor, o alias DNS, ou o endereço IP.

GATEWAY_INTERFACE
A número da revisão da especificação CGI que o servidor utiliza.
Formato: CGI/revisão

CONTENT_LENGTH
O comprimento do conteúdo enviado pelo cliente.

CONTENT_TYPE
Tipo do conteúdo dos dados para buscas que têm informação anexada.

DOCUMENT_NAME
Nome do documento.

DOCUMENT_ROOT
Caminho absoluta da raiz de documentos HTML.

DOCUMENT_URI
URL (URI) do documento.

DATE_LOCAL
Data e hora locais.

DATE_GMT
Data e hora locais no formato Greenwich Mean Time..

LAST_MODIFIED
Data da última modificação do documento.

HTTP_COOKIE
Informação armazenada em um Cookie.

HTTP_ACCEPT
Tipos MIME que o cliente aceitará, dados por cabeçalhos HTTP. Cada item desta lista deve ser separado por vírgulas.
Formato: type/subtype, type/subtype, etc.

HTTP_USER_AGENT
O browser que o cliente está usando para enviar o pedido.
Formato: software/version library/version.

HTTP_REFERER
URL da página que contém o link para a página atual.

PATH
Caminho (de subdiretórios).

PATH_INFO

Informação extra de caminho (de subdiretórios), como fornecida pelo cliente.

PATH_TRANSLATED

Versão de PATH_INFO traduzida pelo servidor.

QUERY_STRING

A informação que segue a ? na URL que referencia o programa CGI. É a informação de busca (query).

QUERY_STRING_UNESCAPED

A mesma informação contida em QUERY_STRING, mas com os caracteres de escape (%nn) traduzidos.

REMOTE_HOST

O nome da máquina que faz o pedido. Se o servidor não tem esta informação, não deve estabelecer um valor para esta variável, mas para REMOTE_ADDR.

REMOTE_ADDR

O endereço IP da máquina remota que faz o pedido.

REQUEST_METHOD

O método com o qual o pedido foi feito. Para HTTP, esse método é "GET", "HEAD", "POST", etc.

SERVER_PROTOCOL

O nome e a revisão do protocolo de informações utilizado pelo pedido.
Formato: protocolo/revisão.
Exemplo: HTTP/1.0

SERVER_PORT

O número da porta para o qual foi enviado o pedido.

SERVER_ROOT

O diretório absoluto da localização do servidor na rede.

SCRIPT_NAME

Um caminho virtual para o programa CGI que está sendo usado.

7.4. Exercícios

1. Escreva um programa CGI usando uma linguagem de sua escolha (no Unix, você pode escolher C, shell ou Perl) que imprima as variáveis de ambiente definidas pelo browser. Não precisa gerar HTML. Pode ser text/plain (o programa só funcionará se as primeiras linha imprimirem “Content-type: text/plain” e a segunda for em branco). Transfira o programa para o subdiretório cgi-bin do seu servidor Web.

8. Server-side Includes

SERVER-SIDE INCLUDES são instruções para o servidor, embutidas em uma página HTML. Normalmente, um servidor não analisa o conteúdo de uma página HTML, mas simplesmente a recupera e envia para o cliente, após uma requisição. Se o suporte a server-side includes estiver instalado, e a página for identificada como uma página que o servidor deve analisar (server-parsed HTML), este procura dentro da página por instruções especiais, em comentários HTML, e realiza transformações nos dados da página antes de enviá-la.

As páginas que podem sofrer transformações terão um conteúdo diferente no isco do servidor e no browser do cliente. São geralmente identificados pelo subtipo MIME server-parsed-html através de uma extensão especial (.shtml, por exemplo). O servidor tenta processar todos os comandos encontrados antes de enviar a página completa ao browser. Caso o servidor não realiza a substituição, os comandos chegarão ao browser as não serão exibidos na página, pois estarão entre comentários HTML.

8.1. Configuração

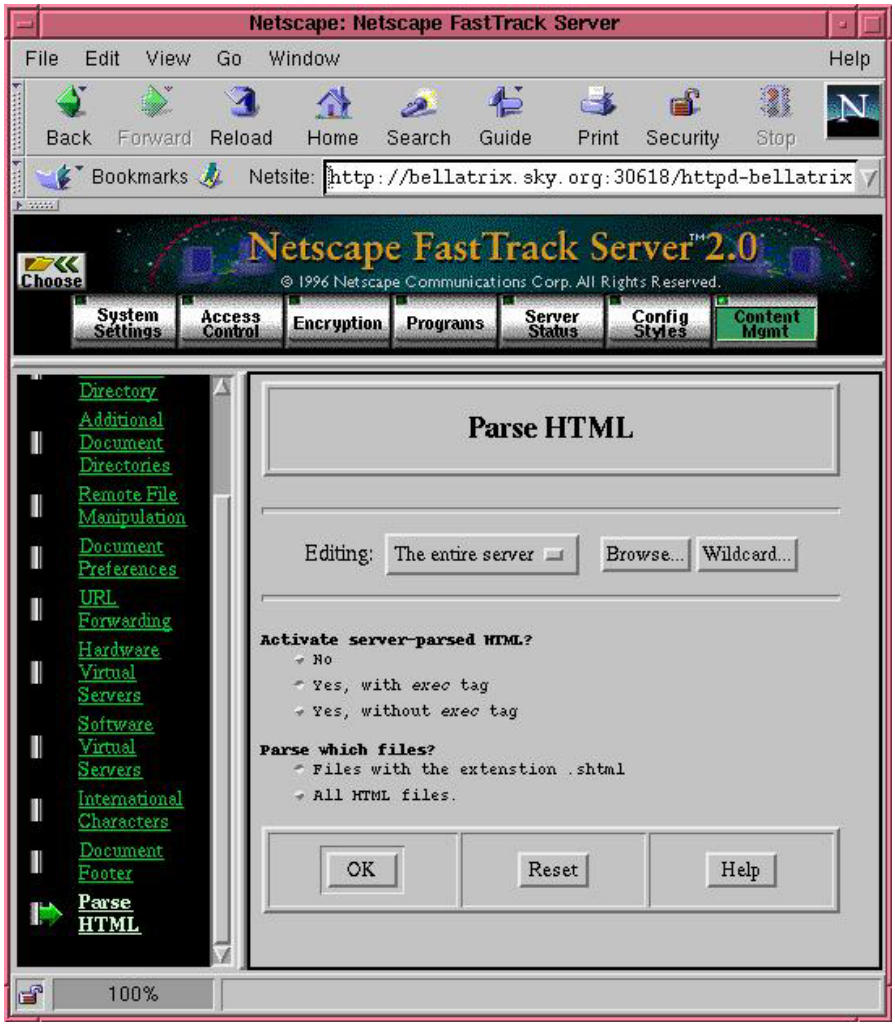
Para usar server-side includes é necessário configurar o servidor. Os servidores que suportam permitem dois níveis de segurança. Um, que permite todos os includes disponíveis no servidor e outro que permite todos mas sem o comando `EXEC` – um include especial que causa a execução de um programa no servidor. Nos servidores UNIX, é preciso alterar o arquivo *access.conf*. As alterações abaixo habilitam o diretório `htdocs` e subdiretórios a armazenar arquivos que contém server-side includes comuns e que suportam o include `EXEC`.

```
<Directory /var/lib/httpd/htdocs>
        Options Indexes FollowSymLinks Includes ExecCGI
        AllowOverride None
        order allow,deny
        allow from all
</Directory>
```

Para definir a extensão `.shtml` como indicação de server-parsed HTML, também é necessário definir o tipo no arquivo *srm.conf* (servidores Apache, NCSA):

```
AddType text/html .shtml
AddHandler server-parsed .shtml
```

Nos servidores Netscape, em Unix ou Windows, tudo isto é definido na janela "Content-Mgmt", opção "Parse-HTML":



8.2. Server-side includes mais comuns

Server-side includes geralmente são dependentes do fabicante do servidor. Há, porém, uma coleção deles que existem desde os primeiros servidores (NCSA e CERN) e estão disponíveis na maioria dos servidores modernos.

Se o suporte a server-parsed HTML estiver instalado, os comandos listados abaixo podem ser utilizados nos documentos HTML analisados pelo servidor, para incluir informação dinâmica na página, antes do envio ao browser. Os principais comandos estão listados a seguir.

<!--#ECHO *atributo*--**>**

Inclui o conteúdo de uma variável de ambiente CGI no documento.

Atributos:

- `VAR="variável-de-ambiente"` armazena o nome da variável de ambiente a retornar. *Exemplo:*

  ```
  <!--#ECHO VAR="LAST_MODIFIED"-->
  ```

  inclui na página a data da última modificação do documento.

<!--#INCLUDE *atributo***-->**

Adiciona todo o conteúdo de um arquivo ao documento, antes de enviá-lo ao cliente. Suporta um atributo, que pode ser FILE ou VIRTUAL.

Atributos:

- `FILE="arquivo"`. Informa a localização de arquivo em relação ao documento que o referencia (caminho relativo).
- `VIRTUAL="arquivo"`. Informa a localização de arquivo em relação à raiz de documentos do servidor (caminho absoluto).

<!--#EXEC *atributo***-->**

Inclui no documento os valores retornados pela execução de um programa no servidor (provoca a execução desse programa). Suporta um atributo, que pode ser CMD ou CGI.

Atributos:

- `CMD="/path/command"`. Informa o comando do sistema a ser executado.
- `CGI="/cgi-bin/programa.pl"`. Informa, a partir do diretório raiz do servidor, o caminho (diretório CGI) e o nome de um programa CGI a ser executado. *Exemplo:*

  ```
  <!--#EXEC CGI="/cgi-bin/counter.pl?doc=includes.html"-->
  ```

  aciona o programa `counter.pl` que conta o número de vezes que o documento foi acessado:

<!--#FSIZE *atributo***-->**

Inclui a informação sobre o tamanho de um determinado arquivo no documento. Suporta um atributo, que pode ser FILE ou VIRTUAL.

Atributos:

- `FILE="arquivo".` Informa a localização de arquivo em relação ao documento que o referencia. *Exemplo*:

  ```
  <!--#FSIZE FILE="filme.mpg"-->
  ```

  inclui na página o tamanho do arquivo `filme.mpg`, que está no mesmo diretório que o documento HTML que contém a linha acima.
- `VIRTUAL="arquivo"`. Informa a localização de arquivo em relação à raiz de documentos do servidor.

<!--#FLASTMOD *atributo***-->**

Inclui a informação sobre a data da última modificação de um determinado arquivo no documento. Suporta um atributo, que pode ser FILE ou VIRTUAL.

Atributos:

- `FILE="arquivo"`. Informa a localização de arquivo em relação ao documento que o referencia. *Exemplo:*

  ```
  <!--#FLASTMOD FILE="texto3.txt"-->
  ```

  informa a data em que foi modificado o arquivo `texto.txt`, que está no mesmo diretório que o documento HTML que contém a linha acima.
- `VIRTUAL="arquivo"`. Informa a localização de arquivo em relação à raiz de documentos do servidor. *Exemplo:*

  ```
  <!--#FLASTMOD VIRTUAL="~helder/logs/setembro.log"-->
  ```

  inclui no documento HTML o arquivo setembro.log, que está no diretório `~helder/logs/`.

8.3. Exemplos e aplicações

O programa a seguir testa o suporte de Server Side Includes do seu servidor.

```
<HTML>
<HEAD>
<TITLE>Server-Side Includes</TITLE>
</HEAD>

<body bgcolor="#FFFFFF">

<H1>Server-Side Includes</H1>

<P>Variáveis de configuração SSI (observe que comentários podem ser usados
dentro dos descritores):
     <PRE>&lt;!--#config timefmt="%c" formato simples de data/hora --&gt;
     <!--#config timefmt="%c" formato simples de data/hora -->
     &lt;!--#config sizefmt="%d bytes"--&gt;
     <!--#config sizefmt="%d bytes"-->
     &lt;!--#config errmsg="##ERRO!##"--&gt;
     <!--#config errmsg="##ERRO!##"--></PRE>

<p>As diretivas SSI abaixo estão executadas ao lado. Para ver o arquivo original
     é necessário visualizá-lo no seu diretório e não através do servidor.
     <p><b>Configurações necessárias:</b>
     1) Para testar a diretiva EXEC é necessário informar o nome um arquivo CGI que retorne
texto e colocá-lo no lugar de "progcgi.exe". 2) Para testar as diretivas INCLUDE, FSIZE e FLASTMOD
é preciso substituir ssitext.txt e ssifile.ext por arquivos existentes no mesmo diretório onde
está a página SHTML.

<PRE><B>DIRETIVA SSI                                                RESULTADO</B>
<HR>
```

```
&lt;!--#exec cgi="/cgi-bin/progcgi.exe"--&gt;  <!--#exec cgi="/cgi-bin/ssicgi.exe"-->
&lt;!--#include file="ssitext.txt"--&gt;   <br><!--#include file="ssitext.txt"-->
&lt;!--#fsize file="ssifile.txt"--&gt;         <!--#fsize file="ssifile.txt"-->
&lt;!--#flastmod file="ssifile.txt"--&gt;      <!--#flastmod file="ssifile.txt"-->
&lt;!--#echo var="DOCUMENT_NAME"--&gt;         <!--#echo var="DOCUMENT_NAME"-->
&lt;!--#echo var="DOCUMENT_URI"--&gt;          <!--#echo var="DOCUMENT_URI"-->
&lt;!--#echo var="LAST_MODIFIED"--&gt;         <!--#echo var="LAST_MODIFIED"-->
&lt;!--#echo var="QUERY_STRING"--&gt;          <!--#echo var="QUERY_STRING"-->
&lt;!--#echo var="QUERY_STRING_UNESCAPED"--&gt;<!--#echo var="QUERY_STRING_UNESCAPED"-->
&lt;!--#echo var="DATE_LOCAL"--&gt;            <!--#echo var="DATE_LOCAL"--&gt;
&lt;!--#echo var="DATE_GMT"--&gt;              <!--#echo var="DATE_GMT"-->
&lt;!--#echo var="SERVER_SOFTWARE"--&gt;       <!--#echo var="SERVER_SOFTWARE"-->
&lt;!--#echo var="SERVER_NAME"--&gt;           <!--#echo var="SERVER_NAME"-->
&lt;!--#echo var="SERVER_PROTOCOL"--&gt;       <!--#echo var="SERVER_PROTOCOL"-->
&lt;!--#echo var="REQUEST_METHOD"--&gt;        <!--#echo var="REQUEST_METHOD"-->
&lt;!--#echo var="REMOTE_HOST"--&gt;           <!--#echo var="REMOTE_HOST"-->
&lt;!--#echo var="HTTP_ACCEPT"--&gt;           <!--#echo var="HTTP_ACCEPT"-->
&lt;!--#echo var="HTTP_USER_AGENT"--&gt;       <!--#echo var="HTTP_USER_AGENT"-->
&lt;!--#echo var="REFERER"--&gt;               <!--#echo var="REFERER"-->
&lt;!--#echo var="BOGUS"--&gt;                 <!--#echo var="BOGUS"--></PRE>
<HR>
</BODY>
</HTML>
```

8.4. Exercícios

1. Escreva uma página HTML que inclua informações sobre a última modificação do arquivo no final da página usando SSI.
2. Escreva uma página HTML que inclua o número gerado em arquivo pelo programa contador counter.pl (é necessário configurar suporte ao descritor exec) em uma área da página.
3. Escreva uma página HTML com uma tabela vazia que seja capaz de incluir linhas adicionais através de arquivo de texto carregado via SSI.

9. Cookies

A TECNOLOGIA CONHECIDA COMO HTTP Cookies, surgiu em 1995 como um recurso proprietário do browser Netscape, que permitia que programas CGI gravassem informações em um arquivo de textos controlado pelo browser na máquina do cliente. Por oferecer uma solução simples para resolver uma das maiores limitações do HTTP – a incapacidade de preservar o estado das propriedades dos documentos em uma mesma sessão – os cookies logo passaram a ser suportados em outros browsers e por linguagens e tecnologias de suporte a operações no cliente e servidor. Hoje, embora não seja ainda um padrão formal, é um padrão de fato adotado pela indústria de software voltada à Web e Internet.

Um cookie não é um programa de computador, portanto não pode conter um vírus executável ou qualquer outro tipo de conteúdo ativo. Pode ocupar no máximo 4 kB de espaço no computador do cliente. Um servidor pode definir no máximo 20 cookies por domínio (endereço de rede) e o browser pode armazenar no máximo 300 cookies. Estas restrições referem-se ao browser Netscape e podem ser diferentes em outros browsers.

Há várias formas de manipular cookies:

- Através de CGI ou outra tecnologia de servidor, como LiveWire, ASP ou Servlets, pode-se criar ou recuperar cookies.
- Através de JavaScript também pode-se criar ou recuperar cookies.
- Através do descritor `<META>` em HTML, pode-se criar novos cookies ou redefinir cookies existentes, mas não recuperá-los.

Um cookie é enviado para um cliente no cabeçalho HTTP de uma resposta do servidor. Além da informação útil do cookie, que consiste de um par nome/valor, o servidor também inclui um informações sobre o domínio onde o cookie é válido, e o tempo de validade do mesmo.

9.1. Criação de cookies via cabeçalhos HTTP

Cookies podem ser criados através de um cabeçalho HTTP usando CGI. Um bloco de cabeçalhos de resposta é gerado pelo servidor Web sempre que o browser solicita uma página estática. Como vimos nos capítulos anteriores, parte ou todo o bloco de cabeçalhos também pode ser gerado por um programa CGI ou equivalente. Quando um programa CGI gera um

cabeçalho, pode incluir outros campos de informação sobre a página que o servidor não inclui por *default*. Pode, por exemplo, definir um ou mais cabeçalhos `Set-Cookie`, que irão fazer com que o browser guarde a informação passada em cookies:

```
HTTP/1.0 200 OK
Date: Friday, June 13, 1997
Server: Apache 1.02
Set-Cookie: cliente=jan0017
Set-Cookie: nomeclt=Marie
Content-type: text/html

<HTML><HEAD>
<TITLE> Capitulo 11</TITLE>
(...)
```

Quando receber a resposta do servidor e interpretar os cabeçalhos acima, o browser irá gravar dois novos cookies na memória contendo as informações `cliente=jan0017` e `nomeclt=Marie`. Essas informações podem ser recuperadas em qualquer página que tenha origem no servidor que definiu os cookies enquanto a presente sessão do browser estiver aberta.

Um cabeçalho `Set-Cookie` pode conter muito mais informações, que alteram a forma como o cookie é tratado pelo browser. Por exemplo, se o cookie tiver um campo `expires` com uma data no futuro, as informações do cookie serão gravadas em arquivo e persistirão além da sessão atual do browser:

```
Set-Cookie: nomeclt=Marie; expires=Monday, 15-Jan-99 13:02:55 GMT
```

A sintaxe completa do cabeçalho `Set-Cookie` está mostrada abaixo. Os campos são separados por ponto-e-vírgula. Todos, exceto o primeiro campo que define o nome do cookie, são opcionais.

```
Set-Cookie: nome_do_cookie=valor_do_cookie;
            expires=data no formato GMT;
            domain=domínio onde o cookie é válido;
            path=caminho dentro do domínio onde o cookie é válido;
            secure
```

Os campos do cabeçalho `Set-Cookie` são usados na definição de cookies tanto em CGI quanto em JavaScript. O significado dos campos está relacionado na tabela abaixo:

Campo	Descrição
nome=valor	*Este campo é obrigatório.* Seqüência de caracteres que não incluem acentos, ponto-e-vírgula, percentagem, vírgula ou espaço em branco. Para incluir esses caracteres é preciso usar um formato de codificação estilo URL. Em JavaScript, a função `escape()` codifica informações nesse formato e a função `unescape()` as decodifica.
`expires=`*data*	*Opcional.* Se presente, define uma data com o período de validade

Campo	Descrição

	do cookie. Após esta data, o cookie deixará de existir. Se este campo não estiver presente, o cookie só existe enquanto durar a sessão do browser. A data deve estar no seguinte formato:
	`DiaDaSemana, dd-mes-aa hh:mm:ss GMT`
	Por exemplo:
	`Monday, 15-Jan-99 13:02:55 GMT`
	O método toGMTString() dos objetos *Date* gera uma data compatível com este formato.

`domain=`*`domínio`*	*Opcional*. Se presente, define um *`domínio`* onde o cookie atual é válido. Se este campo não existir, o cookie será válido em todo o domínio onde o cookie foi criado.
`path=`*`caminho`*	*Opcional*. Se presente, define o *`caminho`* onde um cookie é válido em um domínio. Se este campo não existir, será usado o caminho do documento que criou o cookie.
`secure`	*Opcional*. Se presente, impede que o cookie seja transmitido a menos que a transmissão seja segura (baseada em SSL ou SHTTP).

9.2. Criação de cookies via HTML

Um cookie pode ser criado através de HTML usando o descritor <META> e seu atributo HTTP-EQUIV. O atributo HTTP-EQUIV deve conter um cabeçalho HTTP. O valor do cabeçalho deve estar presente no seu atributo CONTENT. A presença do um descritor <META> dentro de um bloco <HEAD> de uma página HTML, criará um cookie no cliente quando este for interpretar a página.

```
<HEAD>
<META HTTP-EQUIV="Set-Cookie"
      CONTENT="nomeclt=Marie; expires=Monday, 15-Jan-99 13:02:55 GMT">
(...)
</HEAD>
```

9.3. Espaço de nomes de um Cookie

Várias páginas de um site podem definir cookies. O espaço de nomes de um cookie é determinado através de seu domínio e caminho. Em um mesmo espaço de nomes, só pode haver um cookie com um determinado nome. A definição de um cookie de mesmo nome que um cookie já existente no mesmo espaço, sobrepõe o cookie antigo.

Por *default*, o espaço de nomes de um cookie é todo o domínio onde foi criado. Para definir um novo domínio, mais restritivo, é preciso definir o campo `domain`. Por exemplo, se o

domínio de um cookie é `.biscoitos.com`, ele pode ser lido nas máquinas `agua.biscoitos.com` e `chocolate.biscoitos.com`. Para restringi-lo à máquina `chocolate.biscoitos.com`, o campo `domain` deve ser especificado da forma:

```
domain=chocolate.biscoitos.com
```

Somente máquinas dentro do domínio `.biscoitos.com` podem redefinir o domínio. Ele necessariamente tem que ser mais restritivo que o *default*.

O caminho dentro do domínio onde o cookie é válido é o mesmo caminho onde foi criado. O caminho pode ser alterado de forma que tenha um valor mais restritivo definindo o campo `path`. Por exemplo, se um cookie é válido em todos os subdiretórios a partir da raiz, seu `path` é `/`. Para que só exista dentro de `/bolachas/`, o campo `path` pode ser especificado da forma:

```
path=/bolachas/
```

Um cookie chamado `bis` definido em `/` não colide com um cookie também chamado `bis` definido em `/bolachas/`.

Um cookie pode ser apagado se for definido um novo cookie com o mesmo nome e caminho que ele e com data de vencimento (campo `expires`) no passado.

9.4. Recuperação de cookies

Toda requisição de um browser ao servidor consiste de uma linha que contém o método de requisição, URL destino e protocolo, seguida de várias linhas de cabeçalho. É através de cabeçalhos que o cliente passa informações ao servidor, como, por exemplo, o nome do browser que enviou o pedido. Uma requisição HTTP típica tem a forma:

```
GET /index.html HTTP/1.0
User-Agent: Mozilla/4.5 (WinNT; I) [en]
Host: www.alnitak.org.br
Accept: image/gif, image/jpeg, */*
```

Quando um cookie é recuperado pelo browser, ele é enviado em todas as requisições à URLs que fazem parte do seu espaço de nomes, através do cabeçalho do cliente `Cookie`. Apenas o par nome/valor é armazenado no cabeçalho. As informações dos campos `expires`, `path`, e `domain` não aparecem:

```
Cookie: cliente=jan0017; nomeclt=Marie
```

O servidor pode recuperar as informações do cookie através do cabeçalho ou através da variável de ambiente `HTTP_COOKIE`, definida quando o servidor recebe uma requisição com o cabeçalho Cookie. Para fazer uso dos dados de HTTP_COOKIE, é preciso tratar a string que a variável contém, separando cada cookie pelo ponto-e-vírgula e identificando nome e valor através do sinal de igualdade.

Cookies são armazenados em ASCII e contém vários caracteres reservados, entre eles o ponto-e-vírgula e o sinal de igualdade. Para evitar problemas, o cookie deve ser armazenado com esses caracteres convertidos em formatos codificados (URL-encoding, por exemplo). Na recuperação dos dados, será necessário fazer uma decodificação. Várias linguagens oferecem ferramentas para esse tipo de codificação. Exemplos são Perl, C e JavaScript.

9.5. Exercícios

1. Crie um programa CGI (assacookie.pl) que ofereça um formulário HTML que pede o nome do usuário. Grave o nome do usuário como um cookie. Em outra página (comecookie.pl), leia a variável de ambiente HTTP_COOKIE e imprima na página o nome do usuário.