

Criação de Web Sites II

2

Formulários HTTP e CGI

Conteúdo

6. Formulários	65
6.1. Elemento <FORM>	65
Botões de envio	66
Entrada e envio de dados	66
Múltiplos campos de entrada de dados	67
Outros elementos de entrada de dados	68
6.2. Elemento <INPUT>	68
6.3. Botões (TYPE=BUTTON, SUBMIT, RESET ou IMAGE)	69
6.4. Campos de texto de uma linha (TYPE=TEXT ou PASSWORD).....	70
6.5. Campos ocultos (TYPE=HIDDEN)	70
6.6. Chaves booleanas do tipo “botão de rádio” (TYPE=RADIO).....	71
6.7. Elemento <LABEL>.....	71
6.8. Chaves booleanas tipo “caixas de checagem” (TYPE=CHECKBOX).....	72
6.9. Upload de arquivos (TYPE=FILE).....	73
6.10. Elemento <TEXTAREA>	73
6.11. Elementos <SELECT> e <OPTION>	74
6.12. Grupos de opções: elemento <OPTGROUP>	75
6.13. Subgrupos de componentes <FIELDSET> e <LEGEND>	76
6.14. Botões HTML 4.0 <BUTTON>	76
6.15. Exercícios	77
7. Formulários e CGI.....	78
7.1. Programas CGI	78
7.2. Interligando formulários HTML e CGI	78
7.3. Tarefas básicas dos programas CGI.....	79
7.4. Variáveis de Ambiente	80
7.5. Exercícios.....	82
8. Princípios de HTTP	83
8.1. O que é HTTP	83
8.2. Métodos	83
8.3. Requisições HTTP e método GET.....	84
8.4. Respostas HTTP.....	85
8.5. Método HEAD.....	87
8.6. Execução de programas	88
8.7. Método POST.....	88
9. Uso de programas CGI.....	89
9.1. Tipos de programas CGI.....	89
9.2. Onde conseguir programas CGI	90
9.3. Como configurar e instalar os programas	90
9.4. Como criar programas CGI simples	91
9.5. Exercícios.....	95

6. Formulários

É possível construir, usando apenas HTML e nenhuma programação adicional, interfaces gráficas para entrada de dados através de uma página, exibida em um browser. Componentes como botões, chaves booleanas e campos para entrada de texto são criados usando o elemento HTML `<INPUT>` com atributos diferentes. Menus podem ser criados com o elemento `<SELECT>` que agrupa elementos `<OPTION>` contendo as suas opções. Existe ainda um elemento especialmente criado para a entrada de grandes quantidades de texto: o elemento `<TEXTAREA>`. Os componentes devem ser agrupados em blocos chamados de *formulários*, que contêm o endereço do programa, localizado em algum servidor Web, encarregado de fazer alguma coisa com os dados recebidos. Os blocos são delimitados pelo elemento `<FORM>`.

Embora sejam apenas três os elementos para entrada de dados (`<INPUT>`, `<SELECT>` e `<TEXTAREA>`), eles produzem 12 componentes gráficos diferentes. Existem outros seis elementos usados para agrupar componentes e criar botões, mas a maior parte deles não aparece no *Netscape* ou em browsers que não suportem HTML 4.0.

Qualquer elemento de entrada de dados só deve ser usado dentro de um bloco `<FORM>`. Embora um bloco `<FORM>` possa ter mais de um botão de envio, todos enviarão os dados para a única URL que ele indicar, no seu atributo `ACTION`.

As seções a seguir irão explorar cada um dos elementos usados para construir formulários no HTML 4.0. Se possível, tente repetir cada exemplo no seu editor de código, visualizando em seguida os efeitos no seu browser.

6.1. Elemento `<FORM>`

O elemento `<FORM>` é o mais importante dos elementos de formulário. Ele define o “bloco” de formulário que deve ser atrelado a um programa no servidor. A sua sintaxe, contendo os seus atributos (opcionais) mais importantes, está mostrada abaixo:

```
<FORM ACTION="URL para onde será enviado o formulário"
      METHOD="método HTTP (pode ser GET ou POST)"
      ENCTYPE="formato de codificação"
      TARGET="nome da janela que mostrará a resposta do formulário" >
      ... corpo do formulário (permite qualquer coisa permitida em <BODY>) ...
</FORM>
```

Embora todos os atributos do elemento `FORM` sejam opcionais, não é possível criar uma página que envie dados para o servidor sem que esse elemento contenha pelo menos o atributo: `ACTION`, que indica a URL do programa CGI (ou equivalente) que irá processar o conteúdo do formulário (o que o usuário digitar ou escolher).

Botões de envio

O bloco `<FORM>` vazio não mostra coisa alguma na tela. Para que ele possa pelo menos funcionar como um vínculo de hipertexto, ele precisa ter um botão de envio. Esse é o formulário mínimo: Um bloco `<FORM>` com um botão do tipo `SUBMIT`.

```
<FORM ACTION="/dados/tutorial.html">
  <INPUT TYPE="submit" VALUE="Tutorial de Formulários">
</FORM>
```

O botão `SUBMIT`, que é construído com o elemento `<INPUT>` não serve realmente para a entrada de dados como o nome do elemento sugere. É responsável apenas por executar a ação do formulário. O seu atributo `TYPE` é obrigatório para que tenha a aparência e funcionalidade do botão. O seu rótulo é definido pelo atributo `VALUE`. `<FORM>` e `<SUBMIT>` funcionam em conjunto. O trecho acima funciona de forma idêntica ao vínculo de hipertexto abaixo:

```
<A HREF="/dados/tutorial.html">Tutorial de Formulários</A>
```

porém o primeiro mostra um botão no lugar da âncora sublinhada que aparece no segundo. Ambos enviam a mesma requisição ao servidor:

```
GET /dados/tutorial.html HTTP/1.0
```

Mas formulários não foram criados para substituir vínculos de hipertexto. Eles servem principalmente para se enviar informações. O atributo `METHOD` indica o método de requisição HTTP que o browser utilizará para transferir as informações. Se ele estiver ausente, o método será `GET`, como vimos acima. Mas `GET` foi criado para o browser *receber* informação e não *enviá-la*. Não há bons motivos práticos para se usar outro método que não seja `POST` para enviar dados via formulários. Portanto, `ACTION` e `METHOD` são os atributos que devem estar sempre presentes em um bloco `<FORM>`, e `METHOD` deve conter o valor `POST`.

Entrada e envio de dados

Além do botão *Submit*, um formulário deverá conter outros componentes, já que o botão sozinho apenas executa o evento de enviar o formulário, mas não tem a capacidade de receber dados. Os outros elementos que são usados dentro do bloco `<FORM>`, e que podem armazenar informações que seguirão para o servidor, sempre têm um atributo obrigatório: `NAME`. Esse *nome* serve para identificar a *variável* que irá armazenar um *valor* fornecido pelo usuário/visitante da página Web. Depois de preencher um formulário e apertar um botão *Submit*, o nome da variável e o seu valor correspondente serão enviados ao servidor. Veja um exemplo:

```
<FORM ACTION="/cgi-bin/cadastro.pl" METHOD="GET">
  <P>Digite seu Nome: <INPUT TYPE="text" NAME="usuario">
    <INPUT TYPE="Submit" VALUE="Enviar para o servidor">
</FORM>
```

O método `GET` não é uma forma eficiente de se enviar dados, mas tem algumas vantagens didáticas, por isto está sendo usado neste exemplo. O elemento `<INPUT>`, acima, quando recebe o atributo `TYPE` com o valor `"text"` funciona como um campo para entrada de textos. Se não tiver outros atributos terá 20 caracteres de largura. Suponha agora que o usuário digite no campo de textos o nome "Florbela". Quando o usuário apertar o botão *Submit*, o browser construirá a seguinte requisição:

```
GET /cgi-bin/cadastro.pl?usuario=Florbela HTTP/1.0
```

O método GET envia os dados como parte da requisição, logo depois da URL do programa, separando-os por um sinal de interrogação “?”. Isto é ineficiente e arriscado, pois, além deles ficarem visíveis, eles poderão ser truncados se as informações forem muito longas. Essa é a razão porque é melhor usar POST.

Mas, deixando de lado o método, observe como os dados do formulário foram montados pelo browser. Foi criado um par *nome=valor*. O *nome* é o identificador escolhido no atributo NAME. O *valor* é o texto que foi digitado.

Múltiplos campos de entrada de dados

Formulários geralmente servem para se enviar mais do que um nome. Atribuir um nome às variáveis passa a ter alguma utilidade quando há mais de um campo de entrada de dados:

```
<FORM ACTION="/cgi-bin/cadastro.pl" METHOD="GET">
  <P>Digite seu Nome: <INPUT TYPE="text" NAME="usuario">
  <P>Digite seu Número: <INPUT TYPE="text" NAME="numero">
  <INPUT TYPE="Submit" VALUE="Enviar para o servidor">
</FORM>
```

Este segundo bloco tem dois campos de entrada de dados. Suponha que o usuário preencha o formulário com as seguintes informações:

Digite seu Nome:

Digite seu Número:

Desta vez a requisição do browser será a seguinte:

```
GET /cgi-bin/cadastro.pl?usuario=Florbela&numero=12345 HTTP/1.0
```

Observe que os dois pares nome/valor foram separados por um “&”. Se houvesse mais campos de entrada de dados, os dados seriam agrupados da mesma forma, organizados em pares nome=valor e separados pelo “&”. O arquivo `cadastro.pl` deve ser um programa executável e configurado para funcionar como CGI. Ele será responsável por separar os dados recebidos e fazer alguma coisa com eles.

Existem ainda outras conversões feitas pelo browser quando um formulário é enviado, como a conversão de caracteres especiais e espaços. Em uma URL, os caracteres `:`, `/`, `?`, `&`, `#`, `*`, `%` e vários outros têm significado especial. URLs também não suportam caracteres acentuados nem espaços. Quando um usuário preenche um formulário contendo esses caracteres, é preciso que o browser os transforme antes de enviar. Para isto ele usa o código numérico do caractere dentro do alfabeto latino (alfabeto ISO-8859-1) em hexadecimal. Um “ç”, por exemplo, é representado pelo número hexadecimal E7. O código do “ã” é E3 e assim por diante. O browser coloca um “%” antes do código, para indicar que o número não deve ser considerado literalmente, e monta a linha de dados da mesma forma como antes. Espaços são a única exceção. Em vez de serem convertidos no código %20, eles são substituídos pelo caractere “+”. Suponha que os dados digitados no formulário sejam:

Digite seu Nome:

Digite seu Número:

Agora a requisição irá construir a seguinte linha de dados (que será enviada após a ? da URL do programa CGI se o método for GET):

```
usuario=Concei%E3%E7o+da+Silva&numero=333%2301
```

Mais uma vez, é o programa CGI que terá que “se virar” para decodificar os dados recebidos, localizando os sinais de percentagem, os sinais de + e fazendo a conversão.

Se o método usado em um formulário for POST, os dados serão codificados da mesma forma para envio ao servidor. A única diferença é a forma de envio. POST não envia dados através da URL da requisição. Os dados, que podem ser muito longos, são enviados em pedaços através do corpo da requisição do browser, e lidos pelo servidor na sua entrada padrão (porta de serviços). Um browser geraria algo parecido com a requisição a seguir:

```
POST /cgi-bin/cadastro.pl HTTP/1.0
Content-type: text/x-www-form-urlencoded
Content-length: 42

usuario=Concei%E3%E7o+da+Silva&numero=333%2301
```

Observe que os dados fazem parte do corpo da mensagem, que pode ter várias linhas se for necessário. O espaço reservado na URL da requisição é limitado a no máximo uns 2000 caracteres. Formulários mais longos, portanto, irão perder dados se enviados via GET. O tipo dos dados codificados e separados por “&” é `text/x-www-form-urlencoded`. É a forma mais simples. Essa codificação pode ser alterada com o atributo ENCTYPE de <FORM>.

Outros elementos de entrada de dados

Os elementos usados para entrada de dados via formulários são apenas <INPUT>, <SELECT> e <TEXTAREA>. Como vimos, <INPUT> também serve para construir botões. Mudando-se o atributo TYPE de <INPUT>, se obtém diferentes componentes de entrada de dados. São dez diferentes componentes.

O elemento <SELECT> representa uma caixa de seleção com uma ou mais opções contidas entre os descritores <OPTION> e </OPTION>. <TEXTAREA> representa uma caixa de texto. Todos os componentes de entrada de dados, sejam elementos <INPUT>, <SELECT> ou <TEXTAREA> devem ter um atributo NAME que contém o nome da variável que será usada pelo CGI para identificar as informações enviadas. Nas seções a seguir, cada componente será explorado em maiores detalhes.

6.2. Elemento <INPUT>

O elemento <INPUT> é usado para construir botões, e dispositivos de entrada de dados como caixas de texto e chaves booleanas (botões de rádio e caixas de checagem), componentes para o envio de arquivos e para armazenar variáveis invisíveis. O que distingue um objeto do outro é apenas o seu atributo TYPE. A sintaxe geral de <INPUT> é:

```
<input atributos>
```

onde os atributos podem ser diferentes, dependendo do valor do atributo TYPE, que pode ter um dos seguintes valores:

▪ type= text	campo de entrada de texto
▪ type= password	campo de entrada de texto oculto
▪ type= radio	chave booleana para única seleção por grupo
▪ type= checkbox	chave booleana para múltiplas seleções por grupo
▪ type= submit	botão que implementa o evento “enviar dados”
▪ type= reset	botão que implementa o evento “reinicializar formulário”
▪ type= button	botão sem evento associado
▪ type= image	imagem que implementa o evento “enviar dados”
▪ type= hidden	campo oculto para armazenar pares nome/valor
▪ type= file	componente para realizar o upload de arquivos

Cada um dos tipos de <INPUT> será ilustrado nas seções a seguir.

6.3. Botões (TYPE=BUTTON, SUBMIT, RESET ou IMAGE)

Os botões são criados com os atributos type=button, type=reset e type=submit. Usando o atributo type=image, o botão do tipo *Submit* poderá ser substituído por uma imagem. A sintaxe típica de um botão é:

```
<INPUT TYPE="button" VALUE="rótulo do botão">    ou
<INPUT TYPE="submit" VALUE="rótulo do botão">    ou
<INPUT TYPE="reset" VALUE="rótulo do botão">
```

Se o atributo VALUE for omitido, o botão utilizará um rótulo padrão, escolhido pelo browser. A exceção são os botões do tipo *Button*, que não possuem um rótulo *default*. Para os botões do tipo *Submit* usando imagens, é necessário informar a URL onde reside a imagem através do atributo SRC:

```
<INPUT TYPE="image" SRC="http://a.com/imagens/botao.gif">
```

Os botões SUBMIT e RESET reagem a eventos. O botão BUTTON nada faz. Ele só tem utilidade se tiver um evento programado em JavaScript ou VBScript. Ao apertar o botão SUBMIT, o usuário envia os dados do formulário ao programa no servidor indicado pelo atributo ACTION de <FORM>. Ao apertar RESET, o usuário reinicializa o formulário, restaurando os valores iniciais de cada campo de entrada de dados.

O atributo VALUE permite alterar o texto que aparece dentro do botão. A figura ao lado mostra a aparência dos botões em um browser *Netscape Navigator* rodando em *Windows 95* com e sem o atributo VALUE.

O botão IMAGE envia como dados não só os valores digitados pelo usuário mas também as coordenadas em pixels da área da imagem que foi clicada. Isto permite que o programa CGI possa responder de forma diferente a partes diferentes da imagem.

A aparência dos botões, como sua borda, sua cor de fundo, a cor e o tamanho do texto, pode ser alterada por folhas de estilo. O suporte não é total no *Netscape*.

Há ainda vários novos atributos do HTML 4.0 que são apenas parcialmente suportados atualmente. O atributo ACCESSKEY aceita uma tecla que pode ser usada como atalho para o botão:

```
<INPUT TYPE=submit ACCESSKEY="s">
```

TITLE pode ser usado em botões para incluir informações adicionais de acessibilidade. Botões também podem ser desabilitados se tiverem o atributo DISABLED:

```
<INPUT TYPE=button VALUE="Não disponível" DISABLED>
```

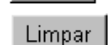
Button

	<input type=button>
	<input type=button value="Apertar">

Submit

	<input type=submit>
	<input type=submit value="Enviar">

Reset

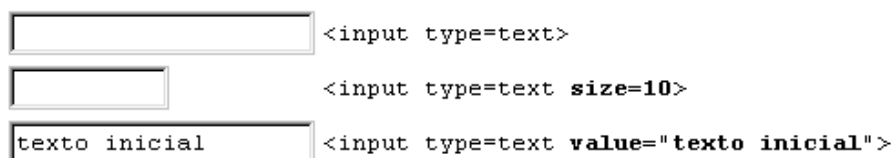
	<input type=reset>
	<input type=reset value="Limpar">

6.4. Campos de texto de uma linha (TYPE=TEXT ou PASSWORD)

Campos de texto de uma única linha são definidos pelos componentes `<INPUT>` que possuem atributos `TYPE` com os valores `PASSWORD` ou `TEXT`. Ambos têm a mesma aparência, mas o texto dos objetos *Password* não é exibido na tela. No lugar dos caracteres digitados aparecem asteriscos (*Windows*) ou marcadores (*Macintosh*). A sintaxe de um elemento `TEXT` e seus principais atributos em HTML é a seguinte:

```
<INPUT TYPE="text"
    NAME="nome_do_campo_de_texto"
    VALUE="texto inicial do campo de textos"
    SIZE="número de caracteres visíveis"
    MAXLENGTH="número máximo de caracteres permitido">
```

Todos os atributos, exceto o atributo `TYPE` são opcionais. Se `SIZE` não for definido, a caixa de texto terá 20 caracteres de largura. Se `MAXLENGTH` não for definido, não haverá limite para o número de caracteres digitado no campo de textos. A figura abaixo ilustra a aparência de alguns componentes `TEXT` em um browser *Netscape 4.5* rodando em *Windows 95*.

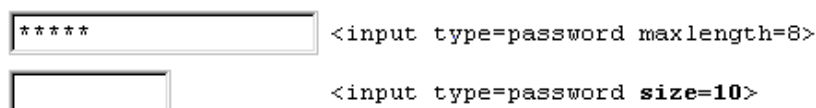


O atributo `VALUE` pode ser usado para definir o valor inicial do texto exibido. Esse valor poderá ser alterado pelo usuário.

O elemento do tipo `PASSWORD` é criado da mesma forma, mas com um atributo `TYPE` diferente:

```
<INPUT TYPE="password" ... >
```

Os caracteres do texto digitado em componentes `PASSWORD` não aparecem na tela, como mostrado na figura abaixo (*Windows95*):



Assim como os botões, campos de texto pode ter sua aparência alterada por folhas de estilo e conter os atributos `ACCESSKEY`, `DISABLED`, `TITLE` e outros que funcionarão em browsers compatíveis com HTML 4.0. Também foi introduzida no HTML 4.0 a possibilidade de tornar um campo de textos somente-leitura (para exibição de informações apenas). Isto é feito com o atributo `READ-ONLY`:

```
<INPUT TYPE=text VALUE="Mensagem inapagável" READ-ONLY>
```

6.5. Campos ocultos (TYPE=HIDDEN)

O componente do tipo `HIDDEN` é um campo de entrada de dados invisível, que o usuário da página não tem acesso. Serve para que o *autor da página* passe informações ao servidor, ocultando-as no código HTML da página. É muito utilizado para enviar informações de configuração para programas CGI usados por várias pessoas. Programas de envio de e-mail, por exemplo, podem obter o endereço destino (do autor da página) através de um

campo oculto. Esse campo também é muito usado na transferência de informações entre formulários que são formados por mais de uma página. Sua sintaxe é a seguinte:

```
<INPUT TYPE="hidden"
      NAME="nome_do_campo_oculto"
      VALUE="valor armazenado" >
```

Os atributos NAME e VALUE são obrigatórios para que o campo HIDDEN tenha alguma utilidade. A exceção é quando se usa programas como JavaScript, que rodam no próprio browser e têm condições de alterar o valor de tais campos durante a exibição da página.

6.6. Chaves booleanas do tipo “botão de rádio” (TYPE=RADIO)

O componente do tipo RADIO representa um dispositivo de entrada booleano cuja informação relevante consiste em saber se uma opção foi selecionada ou não. Botões de radio são organizados em grupos de descritores com o mesmo nome (atributo NAME). Cada componente aparece na tela como um botão ou caixa de dois estados: *ligado* ou *desligado*. Dentro de um grupo de componentes (todos com o *mesmo* atributo NAME), somente um deles poderá estar ligado ao mesmo tempo. A sintaxe típica de um componente RADIO em HTML é a seguinte:

```
<INPUT TYPE="radio"
      VALUE="valor (o valor que será enviado ao servidor)"
      CHECKED      <!-- previamente marcado -->
      > Rótulo do componente
```

A figura abaixo mostra dois grupos de botões de rádio (em um browser *Netscape* rodando em *Windows95*). Observe que os atributos NAME distinguem um grupo do outro. O atributo CHECKED indica um botão previamente ligado mas que pode ser desligado pelo usuário ao clicar em outro botão.

Se não for definido um campo VALUE, com o valor a ser enviado ao servidor, será enviado o texto “on” ou “1”. Se nenhum dos componentes de um grupo de botões de rádio for selecionado, a maior parte dos brow-

GRUPO I

☐	pela manhã	<input type=radio name=turno value="Manhã"> pela manhã
☐	à tarde	<input type=radio name=turno value="Tarde"> à tarde
☒	à noite	<input type=radio name=turno value="Noite" checked> à noite

GRUPO II

☐	Masculino	<input type=radio name=sexo value="M"> Masculino
☒	Feminino	<input type=radio name=sexo value="F"> Feminino

sers sequer envia o nome do elemento ao servidor, mas esse comportamento não é garantido (a especificação não prevê sua ocorrência), portanto, é uma boa idéia pré-marcou ou verificar, usando JavaScript, se um usuário marcou uma das opções antes que os dados sejam enviados ao servidor.

Todos os atributos HTML 4.0 aplicáveis aos botões podem ser usados nos botões de rádio. Usando folhas de estilo, poderá ser possível alterar a sua cor.

6.7. Elemento <LABEL>

Os botões de rádio não possuem um rótulo associado. O texto que o identifica deve ser colocado ao seu lado e não tem relação alguma com ele. O HTML 4.0 permite associar um rótulo de texto com elementos <SELECT>, <TEXTAREA> e <INPUT>, que inclui botões de rádio, usando o elemento <LABEL>:

```
<label accesskey="m">
  <input type=radio name=sexo value=M>Masculino
</label>
```

Em browsers que suportam esse recurso do HTML 4.0, será possível mudar o estado do botão de rádio clicando no texto ou digitando a letra “m” no teclado. Nos browsers que não suportam o recurso, o componente se comportará da mesma forma como sempre se comportou, ou seja, ignorando cliques do mouse sobre o rótulo do botão.

Nem sempre é possível ter o rótulo e o componente dentro do mesmo bloco <LABEL>. Uma situação é quando são usadas tabelas para organizar formulários. É comum o rótulo estar em uma célula e o componente em outra. Nessas situações não será possível ter os dois dentro do bloco <LABEL>. A solução nesses casos é usar <LABEL> com o atributo FOR. O componente precisará ser identificado através de sua propriedade ID e FOR informa ao browser que aquele rótulo pertence ao componente indicado. ID e NAME devem ter o mesmo valor, sempre:

```
<tr>
<td><label for="usuario">Nome: </label></td>
<td><input type=text name=usuario id=usuario></td>
</tr>
```

6.8. Chaves booleanas tipo “caixas de checagem” (TYPE=CHECKBOX)

Cada elemento <INPUT> do tipo CHECKBOX aparece na tela como um botão ou caixa que pode assumir dois estados: *ligado* ou *desligado*. Diferentemente dos componentes RADIO, vários componentes CHECKBOX de um mesmo grupo podem estar ligados ao mesmo tempo. Não há, portanto, sequer a necessidade de organizar tais objetos em um grupo.

A sintaxe de um elemento CHECKBOX em HTML é praticamente idêntica à de RADIO, mudando apenas o valor do atributo TYPE:

```
<INPUT TYPE="checkbox" ...
  VALUE="valor (o valor que será enviado ao servidor)"
  CHECKED      <!-- previamente marcado -->
  > Rótulo do componente
```

A figura abaixo mostra um grupo de caixas de checagem (em um browser Netscape rodando em Windows. O atributo CHECKED indica um botão previamente ligado mas que pode ser desligado pelo usuário ao clicar em outro botão.

```
☑ Café    <input type=checkbox name=refeicoes value="Café" checked> Café
☐ Almoço  <input type=checkbox name=refeicoes value="Almoço"> Almoço
☑ Jantar  <input type=checkbox name=refeicoes value="Jantar"> Jantar
```

Assim como RADIO, o elemento CHECKBOX pode ser alterado por folhas de estilo e com atributos HTML 4.0 como ACCESSKEY e DISABLED. Pode também ser associado a um rótulo através do elemento <LABEL>.

Como dois elementos CHECKBOX podem ter o mesmo nome e ambos podem ter sido marcados pelo usuário, o programa CGI terá que separar os dois para evitar sobreposição. Os dois elementos marcados do desenho acima não são anulados pelo browser, mas são concatenados e enviados da forma:

```
refeicoes=Caf%E9&refeicoes=Jantar
```

Como os grupos não têm importância em elementos CHECKBOX, pode-se usar nomes diferentes em vez de ter que se preocupar com a separação das variáveis no CGI:

```
<input type=checkbox name=cafe>
<input type=checkbox name=almoco>
<input type=checkbox name=jantar>
```

Os elementos marcados serão enviados como:

```
cafe=on&jantar=on
```

6.9. Upload de arquivos (TYPE=FILE)

O último elemento <INPUT> serve para realizar a transferência de arquivos da máquina do cliente até o servidor. Com o atributo TYPE=file, o elemento <INPUT> se transforma em um par de componentes que consiste de uma caixa de texto e um botão.

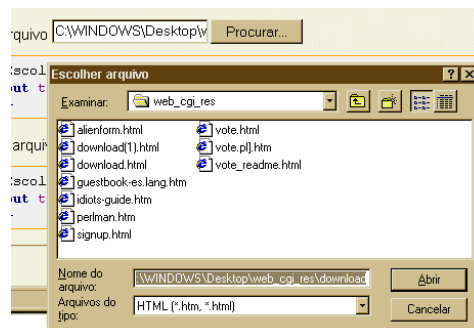
```
<label>Escolha um arquivo
  <input type=file name=arq2 title="Localizar arquivo">
</label>
```

O botão possui um rótulo que não é possível alterar. Sua ação, ao ser apertado, é abrir uma janela de escolha de arquivos através da qual o usuário poderá escolher o arquivo que deseja transferir. Uma vez escolhido o arquivo, o caminho do mesmo aparece na caixa de texto (veja figura).

Infelizmente, não é possível alterar a aparência ou o tamanho do elemento FILE de forma alguma. O campo de textos e o botão sempre têm tamanho fixo. Não pode haver texto inicial na caixa de texto nem outro rótulo para o botão.

A transferência de arquivos para o servidor exige que lá exista um programa CGI capaz de receber e tratar formulários no qual os dados são enviados em múltiplas partes. Não é possível enviar arquivos através da codificação padrão do formulário, que é do tipo text/x-www-form-urlencoded. Os dados têm que ser enviados usando a forma text/multipart-form-data, que deve ser expressa no atributo ENCTYPE. Esse formato, por necessitar de múltiplas linhas, não é suportado pelo método GET. Em suma, o bloco <FORM> de um formulário que contém elementos FILE deve ter a forma:

```
<FORM ACTION="/cgi-bin/programa.exe"
  METHOD=POST
  ENCTYPE="text/multipart-form-data"> ... </FORM>
```



6.10. Elemento <TEXTAREA>

O elemento <TEXTAREA> é um bloco (possui descritor inicial e final) e define uma área onde se pode ler ou digitar texto em várias linhas. A sintaxe para criar um elemento <TEXTAREA> em HTML é a seguinte. Os atributos em negrito são obrigatórios e definem a altura e largura da área de texto:

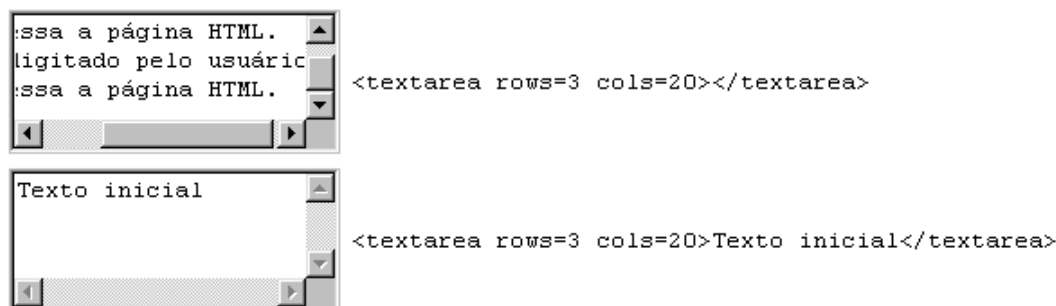
```
<TEXTAREA
  ROWS="número de linhas visíveis"
```

```

COLS="número de colunas visíveis"
NAME="nome_do_campo_de_texto"
WRAP="hard | soft | virtual | physical | off">
    Texto inicial
</TEXTAREA>
    
```

Não é possível controlar a aparência exata dos elementos de texto como o <TEXTAREA> e <INPUT TYPE=TEXT> porque eles são dimensionados não em pixels mas em número de caracteres e linhas visíveis. As fontes usadas mudam de um browser para outro fazendo, por exemplo, que um campo <TEXTAREA> apareça 30 a 40% maior em browsers *Netscape* que no *Internet Explorer*. Pode-se usar folhas de estilo para controlar a aparência mas elas só funcionarão no *Internet Explorer*.

A figura abaixo mostra a aparência de componentes <TEXTAREA> no *Netscape*:



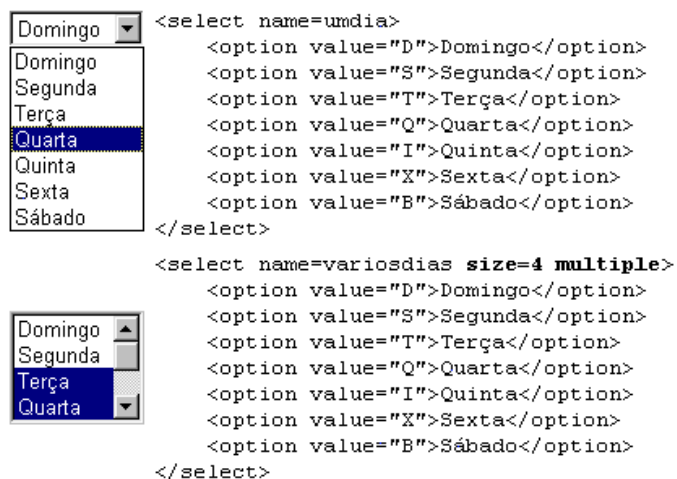
O atributo WRAP pode ser usado para controlar a maneira como o texto é enviado e visualizado no campo de texto. Com WRAP=off, o texto rola lateralmente e as linhas não quebram. Com WRAP=soft ou WRAP=virtual as linhas quebram na tela mas chegam no servidor como uma única linha. WRAP=hard ou WRAP=physical introduz os caracteres de novas-linhas que farão com que as quebras de linha cheguem ao servidor.

Elementos <TEXTAREA> podem ser desabilitados com o atributo DISABLED e tornados somente-leitura com o atributo READ-ONLY. Esses recursos, assim como alguns recursos de folhas de estilo, só são suportados por browsers que oferecem suporte amplo ao HTML 4.0.

6.11. Elementos <SELECT> e <OPTION>

Menus e listas de seleção como as mostradas nas figuras ao lado são criadas com o elemento HTML <SELECT>. Este componente não pode conter texto. Contém apenas elementos <OPTION> que representam as opções do menu.

Os componentes <SELECT> podem ter uma aparência e comportamento diferente dependendo se possuem ou não os atributos SIZE e MULTIPLE. A figura ao lado ilustra o efeito desses atributos, transformando um menu simples em uma lista com barra de rolagem lateral que permite a seleção de múltiplas opções. SIZE altera o número de linhas visíveis. MULTIPLE, se presente, habilita a seleção de



mais de uma opção. A sintaxe de um elemento HTML `<SELECT>` com seus principais atributos está mostrada abaixo:

```
<SELECT NAME="nome_do_componente"
      SIZE="número de opções visíveis"
      MULTIPLE      <!-- Suporta seleção múltipla -->
  >
      <OPTION ...> Opção 1 </OPTION>
      ...
      <OPTION ...> Opção n </OPTION>
</SELECT>
```

Todos os atributos são opcionais. A existência do atributo NAME é obrigatória em formulários que terão dados enviados ao servidor. Cada um dos elementos `<OPTION>` poderá ter a seguinte sintaxe básica:

```
<OPTION VALUE="Valor da opção" SELECTED >
      Texto descrevendo a opção
</OPTION>
```

O atributo VALUE informa o que deve ser enviado ao servidor quando a opção correspondente tiver sido selecionada. Mas VALUE é opcional. Na ausência de um elemento VALUE o browser envia o texto visível, contido entre `<OPTION>` e `</OPTION>`. Esse texto não será enviado se VALUE estiver presente.

O atributo SELECTED marca a opção como previamente selecionada. Nos casos onde só é permitida a seleção simples, apenas uma das opções deverá conter o atributo.

Nos casos onde se permite seleção múltipla, os pares nome/valor são repetidos e o comportamento é semelhante ao caso de objetos CHECKBOX de mesmo nome. Por exemplo, se na lista de opções mostrada o usuário tiver selecionado os dias terça e quarta, o browser enviará:

```
variosdias=T&variosdias=Q
```

Folhas de estilo podem ser aplicadas tanto no objeto SELECT como um todo, como nas suas opções individuais, onde se pode mudar a cor de fundo, a cor e tamanho da fonte usada, etc. Um elemento `<SELECT>` também pode ser desligado usando DISABLED. Todos esses recursos só são suportados por browsers que implementam HTML 4.0.

6.12. Grupos de opções: elemento `<OPTGROUP>`

Nos browsers que implementam HTML 4.0, é possível organizar grupos de opções `<OPTION>` através do elemento `<OPTGROUP>`, também aceito dentro de um bloco `<SELECT>`. Esse elemento cria um rótulo e indenta ou destaca de alguma outra forma as opções de menu que contém, criando submenus.

```
<select name=cores>
  <option value="nenhuma">Escolha uma cor</option>
  <optgroup title=luminosas>
    <option value="R">Vermelho</option>
    <option value="G">Verde</option>
    <option value="B">Azul</option>
    <option value="W">Branco</option>
  </optgroup>
  <optgroup title=opacas>
    <option value="C">Azul Ciano</option>
```

```

<option value="M">Magenta</option>
<option value="Y">Amarelo</option>
<option value="K">Preto</option>
</optgroup>
</select>

```

6.13. Subgrupos de componentes <FIELDSET> e <LEGEND>

Em formulários longos, pode ser necessário identificar suas partes com informações específicas para indicar a organização de um grupo de elementos. O HTML 4.0 oferece os elementos <FIELDSET> e <LEGEND> para essa finalidade.

<FIELDSET> deve ser colocado em volta dos elementos que se quer agrupar. <LEGEND> é usada para identificar o subgrupo.

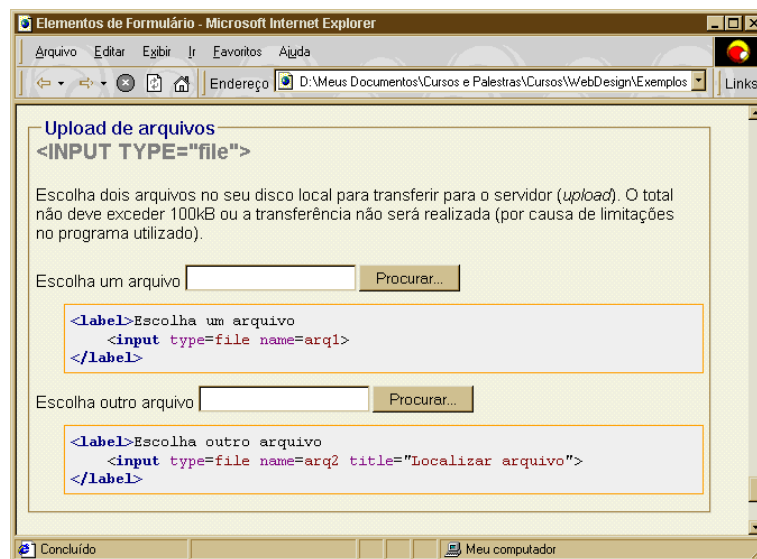
```

<fieldset> <!-- HTML 4.0! Pode ã funcionar em alguns browsers! -->
  <legend>Upload de Arquivos</legend>

  <!-- vários elementos de formulário -->

</fieldset>

```



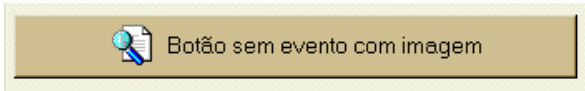
6.14. Botões HTML 4.0 <BUTTON>

Você deve ter achado estranho um botão que não serve para a entrada de dados ser construídos a partir de um elemento chamado <INPUT>. O grupo de trabalho que desenvolve o HTML também e por isso propuseram um novo elemento que tem a finalidade apenas ser um botão e que oferece diversas vantagens sobre os botões tradicionais.

O elemento <BUTTON> pode ter seu rótulo definido através do texto que contém ou através do atributo VALUE. Possui um atributo TYPE que pode assumir os valores submit, reset e button.

Com <BUTTON>, fica simples criar um botão que contém uma imagem. Basta colocar a imagem dentro do bloco <BUTTON>:

```
<button title="Botão compatível com HTML 4.0" type="button">
  
  Botão sem evento com imagem
</button>
```



É preciso ter cuidado já que o botão é um elemento que pode fazer falta se não aparece na página. Browsers que não suportam HTML 4.0 poderão não mostrar nada no lugar onde deveria estar o botão, portanto, vale a pena esperar por um suporte mais amplo antes de deixar de usar os antigos <INPUT TYPE=SUBMIT>, <INPUT TYPE=RESET> e <INPUT TYPE=BUTTON>.

6.15. Exercícios

1. Pratique com os componentes de formulário mostrados neste capítulo. Faça uma ou mais páginas HTML que utilizem todos os elementos. Veja o resultado em browsers diferentes.
2. Construa um formulário de feedback para sua página. O formulário deverá ter um lugar para que o visitante deixe seu nome, seu e-mail e mande uma mensagem. Você pode acrescentar outros campos se quiser.
3. Faça o formulário de feedback acima funcionar. Se você tem conta no IBPINET, informe “/cgi-bin/FormMail.pl” como sendo o ACTION do seu formulário e inclua um campo HIDDEN com o nome “recipient” e valor contendo o seu e-mail. Agora qualquer visitante poderá mandar e-mail para você via Web.
4. Mude a página de resposta do exercício anterior acrescentando outro campo hidden, desta vez com o nome “redirect”. O campo VALUE deverá conter o nome da página que será mostrada após o envio do formulário.

7. *Formulários e CGI*

Para nada serve um formulário se não há um programa no servidor capaz de fazer alguma coisa com os dados preenchidos pelo cliente. Dependendo do servidor, de sua plataforma e configurações, o programa pode ser uma página ASP, PHP ou JSP, pode ser um servlet Java, um arquivo *.dll, um arquivo *.exe, um programa em Perl, um roteiro Shell ou outro mecanismo que seja capaz de interagir com o formulário.

A forma mais universal de interação com programas iniciados pelo servidor é o mecanismo chamado de CGI, suportado por quase todos os servidores Web. Este capítulo mostra o que é preciso fazer para que um programa CGI obtenha informações de um formulário.

7.1. *Programas 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 software servidor, passar-lhes parâmetros recebidos de uma requisição do browser e redirecionar sua saída como uma resposta HTTP.

A finalidade do CGI é permitir que o servidor funcione como ponte (ou *gateway*) para aplicações externas. A especificação CGI 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 gerar corretamente o final do cabeçalho do servidor (imprimir uma linha em branco).

Um programa CGI, para ser útil, precisa fazer algo mais. O cabeçalho de resposta é deixado incompleto pelo servidor porque ele espera que o programa CGI gere dados de algum tipo. Como o servidor não tem como saber que tipo de dados o CGI irá gerar, é o programa CGI que terá que dar essa informação, construindo (imprimindo) uma linha de cabeçalho “Content-type” antes de encerrar o cabeçalho com uma linha em branco.

Para tratar formulários, o programa tem que ser capaz de ler eficientemente os dados na entrada, identificando o método usado pelo browser, extraíndo as informações da requisição do browser (se o método tiver sido GET) ou da entrada padrão (se tiver sido POST) e depois decodificando as informações geradas pelo browser.

Por não exigir o conhecimento de uma linguagem de programação específica, CGI tornou-se uma solução extremamente popular, mesmo depois que surgiram tecnologias mais eficientes e mais simples para programar a Web. Existem diversas aplicações comerciais ou gratuitas na Internet que foram escritas em C, C++, Perl, AppleScript, Java, Visual Basic, Delphi, Bourne-shell e várias outras linguagens rodando em todas as plataformas onde há servidores Web.

7.2. *Interligando formulários HTML e CGI*

Como vimos em um capítulo anterior, 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, menus, etc.

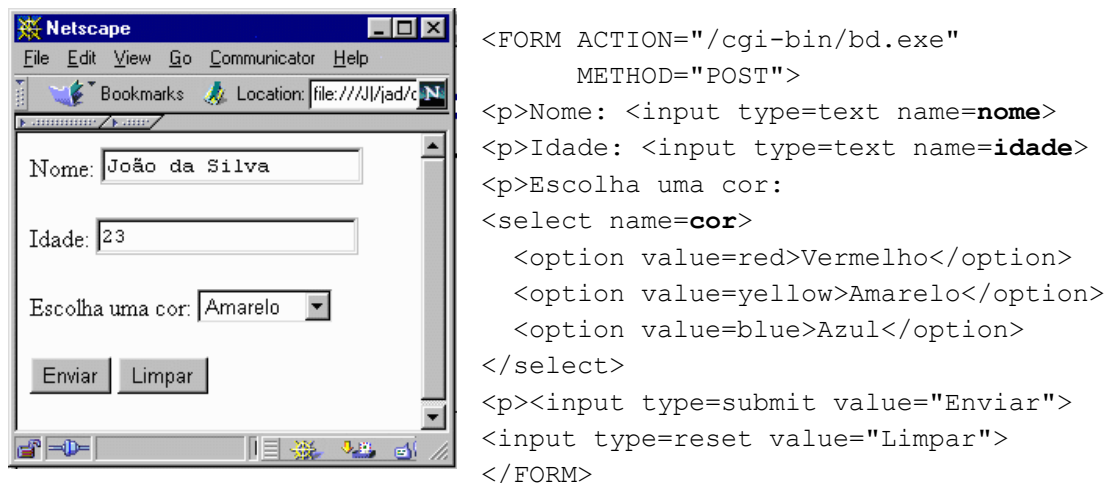
Cada componente de entrada de dados de um formulário tem um atributo NAME e pode ter um atributo VALUE. NAME, geralmente um nome escolhido pelo autor da página, contém o nome que será utilizado pelo programa CGI para recuperar o valor que o leitor da página enviou. Esse valor também pode ser fixado pelo autor da página através dos atributos VALUE de cada componente. Como vimos, cada um dos elementos de um formulário envia seus dados no formato nome=valor.

Ao enviar um formulário com mais de um campo de entrada de dados, todos os pares nome/valor são concatenados com um “&”. Os espaços também são convertidos em “+” e caracteres reservados convertidos em hexadecimal, da forma %hh. Essa é a forma padrão de codificação dos dados de um formulário feita pelo browser e representada pelo tipo MIME text/x-www-form-urlencoded.

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

```
POST /cgi-bin/bd.exe HTTP/1.0
Content-type: text/x-www-form-urlencoded
Content-length: 36
```

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



Esses dados são enviados ao servidor que os repassa ao programa localizado em /cgi-bin/bd.exe. A requisição também poderia ter sido realizada usando o método GET, mas nesse caso os dados seriam enviados na linha de requisição após a “?” em uma parte da URL chamada de *Query String*.

```
GET /cgi-bin/bd.exe?nome=Jo%E3o+da+Silva&idade=23&cor=yellow HTTP/1.0
```

7.3. Tarefas básicas dos programas CGI

Programas que seguem a especificação CGI e irão ler dados enviados por formulários devem ser capazes de obter e decodificar uma linha como

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

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 todos os “+” em espaços
- 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.

Uma vez extraídos os dados, o programa CGI pode realizar qualquer tarefa no servidor. Pode, por exemplo, iniciar uma outra aplicação, montar uma requisição em linguagem SQL para envio a um banco de dados ou recuperar uma imagem de um dispositivo externo. No final, o programa deve retornar para o cliente uma imagem, uma página ou outra sequência de bytes qualquer. O servidor não tem como saber o tipo da sequência de bytes para montar o cabeçalho para o cliente, então, deixa-o incompleto para que seja completado pelo programa CGI. O programa precisa, portanto, imprimir na sua saída no pelo menos:

- uma linha de cabeçalho “Content-type: *tipo_MIME*”, informando o tipo de dados enviados de volta ao servidor (por exemplo, Content-type: text/html)
- uma linha em branco, indicando o fim do cabeçalho
- os dados (página HTML, código GIF, código JPEG, código de máquina, etc)

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

```
printf("Content-type: text/html\n\n"); /* \n\n quebra linha duas vezes */
printf("<html><body><h1>Hello CGI World</h1></body></html>");
```

7.4. Variáveis de Ambiente

Um programa CGI precisa freqüentemente obter várias informações sobre o servidor, sobre o sistema operacional local e sobre o browser e máquina cliente que fez a requisição. Toda requisição do browser faz com que o servidor defina algumas variáveis de ambiente, que geralmente duram o mesmo tempo da transação.

Essas variáveis são como variáveis de ambiente do DOS ou do Unix. Se você escrever um programa CGI como um roteiro Shell, por exemplo, poderá imprimir o nome do browser que fez o pedido com o seguinte programa:

```
#!/bin/sh

echo "Content-type: text/html"
echo ""
echo "<h1>Teste de CGI</h1>"
echo "<p>O seu browser é " $HTTP_USER_AGENT
```

A variável HTTP_USER_AGENT é criada pelo servidor durante a transação. Como qualquer outra variável Unix, ela pode ser lida usando o cifrão antes do nome (se você tivesse criado um programa em MS-DOS, a variável seria lida como %HTTP_USER_AGENT%).

Variáveis de ambiente são definidas tanto por servidores Unix como por servidores Windows. Nem todas são suportadas em todos os servidores. As que são suportadas pela maioria dos servidores estão listadas abaixo:

SERVER_SOFTWARE

O nome e a versão do software do servidor que responde ao pedido. *Formato:* nome/versão. *Exemplo:* Apache/1.2

SERVER_NAME

O nome da máquina onde roda o servidor (DNS) ou o endereço IP. *Exemplo:* www.servidor.org

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 (se programa for um documento HTML).

DOCUMENT_ROOT

Caminho absoluta da raiz de documentos HTML.

DOCUMENT_URI

URL (URI) do documento ou do programa CGI.

DATE_LOCAL

Data e hora locais. *Exemplo:* Friday, Aug 13, 1999 0:0:0 GMT-03

DATE_GMT

Data e hora locais no formato *Greenwich Mean Time*. *Exemplo:* Thursday, Aug 12, 1999 21:0:0 GMT

LAST_MODIFIED

Data da última modificação do documento ou do programa.

HTTP_COOKIE

Lista de cookies para o caminho e domínio atuais, separados por ponto-e-vírgula.

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.

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.

7.5. 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 gerar text/plain (o programa só funcionará se a primeira linha imprimir “Content-type: text/plain” e a segunda for em branco). Transfira o programa para o diretório CGI do seu servidor Web e teste-o. (A forma de imprimir uma variável de ambiente em Perl é

```
print $ENV{'NOME_DA_VARIÁVEL'};
```

Não se esqueça de imprimir o cabeçalho *antes*.

2. Rode o formulário-exemplo localizado no site do curso (http://site_do_curso/allform.html), preencha-o e veja o resultado.

8. Princípios de HTTP

Este é um capítulo de referência. Não é preciso conhecer os assuntos tratados aqui para desenvolver ou usar CGI.

8.1. 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 W3C¹, 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

[REC2068]

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 sequência 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.

8.2. 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 componente 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.

¹ World Wide Web Consortium (<http://www.w3.org>)

8.3. Requisições HTTP e método GET

Uma requisição HTTP obedece a um formato padrão que consiste de uma linha de requisição, linhas de cabeçalho, uma linha em branco e talvez, linhas de dados:

```
Método   URL_solicitada   Protocolo/versão
Propriedade: valor
Propriedade: valor ...
```

Várias linhas de dados (informações enviadas pelo browser)

A primeira linha é a mais importante. Ela usa um método HTTP para operar sobre uma determinada URL, e diz qual o protocolo e versão que deve ser usado para a resposta.

O cabeçalho pode conter várias linhas com informações adicionais essenciais ou não. As linhas sempre são da forma Propriedade: valor e o formato de todo o cabeçalho é padronizado pela especificação Internet RFC822, de define o formato para cabeçalhos de e-mail. Essa especificação declara que o cabeçalho termina quando o receptor encontra uma linha em branco. Tudo o que segue será considerado informação a ser consumida (dados).

Tipicamente, as linhas de cabeçalho contêm o nome da máquina destino (se separado da URL na requisição), a porta de serviços, a data, o nome e versão do cliente, etc. A maior parte das requisições HTTP não envia dados adicionais após o cabeçalho. As que enviam dados precisam ainda informar o tamanho e tipo dos dados enviados para que o servidor saiba o que fazer com eles.

O método GET tem a finalidade de recuperar informações do servidor. É o método mais usado pelo browser. A especificação [RFC2068] o define como um método seguro e *idempotente*, ou seja, deve produzir o mesmo resultado se repetido várias vezes. GET nunca envia dados após o cabeçalho e possui a seguinte sintaxe mínima:

```
GET URL_relativa HTTP/1.1
Host: nome_da_maquina
<--- linha em branco (CRLF)
```

Por exemplo:

```
GET /docs/index.html HTTP/1.1
Host: www.festa.org.br
```

A requisição acima faz uma conexão à porta 80 da máquina `www.festa.com.br` e solicita ao servidor o arquivo `/docs/index.html`. A requisição acima normalmente contém várias outras linhas de cabeçalho antes da linha em branco. Abaixo está uma provável requisição para buscar a página `index.html`, formada ao se clicar em um vínculo de hipertexto no *Microsoft Internet Explorer*:

```
GET /docs/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, *.*
Referer: http://www.casa.com.br/pagina2.html
Host: www.festa.com.br
```

As propriedades de cabeçalho presentes dependem do método usado na requisição. Algumas das propriedades frequentes em requisições GET são:

- **User-Agent:** Contém uma linha que identifica o browser que faz o pedido.
- **Host:** Endereço ou nome do servidor.

- **Date:** Data da requisição no formato GMT ou local.
- **Accept:** Pode haver vários cabeçalhos Accept. Cada um contém um tipo MIME (tipo/subtipo) que é aceito pelo browser que faz o pedido.
- **Cookie:** Envia os cookies relacionados com o domínio e caminho atual.
- **Accept-Encoding:** Tipos de codificação aceitas pelo browser.
- **Referer:** Contém uma linha com a URL do documento onde está a referência (link) do recurso solicitado. Não aparece se a 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 seu diretório de arquivos Internet temporários.

8.4. Respostas HTTP

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

```
Protocolo/Versão  Código  Informações de status
Propriedade: valor
Propriedade: valor ...
```

Várias linhas de dados (informações enviadas pelo servidor)

A resposta começa com uma linha que informa o estado da resposta. 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 (o cliente poderá corrigir o erro, se souber como). Os códigos 5xx indicam que o *servidor* está consciente que o erro é dele, e não pode ser solucionado pelo cliente (o servidor inclui possíveis programas CGI e arquivos .htaccess). As informações de *status* contêm um texto descrevendo a ocorrência. Veja o significado de alguns códigos:

- **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 (pode ser erro em programas CGI, erro em arquivos .htaccess, etc.)
- **Not implemented 501** – Este recurso não foi implementado no servidor
- **Moved 301** – A informação se mudou para outro lugar (houver redirecionamento). A resposta contém a nova URL onde o recurso pode ser localizado.
- **Not Modified 304** – A página não foi modificada desde o último acesso. Recupere-a do diretório de arquivos da Internet temporários.

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 e não aceita pedido de páginas) 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). Isto ocorre em qualquer resposta do servidor.

Os cabeçalhos que o servidor devolve ao cliente referem-se a propriedades do objeto retornado, mais informações sobre o ambiente. A maioria deles são gerados automaticamente pelo servidor. Para criar uma aplicação CGI, é preciso completar o cabeçalho com uma linha em branco, pois nessas condições, o servidor não o completa. Antes de completar o bloco do cabeçalho, porém, o programador CGI pode redefinir ou acrescentar novos cabeçalhos, como o cabeçalho “Content-type”, que informa o tipo do conteúdo gerado pelo CGI. Alguns dos principais cabeçalhos de resposta estão listados a seguir:

- **Content-Length:** Conteúdo em bytes da informação enviada.
- **Content-Type:** Indica o tipo de mídia dos dados que estão sendo enviados. Isto pode ser o tipo MIME ou tipos de múltiplas partes como: 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, x-gzip, x-www-form-urlencoded.
- **Date:** Indica a data de criação do objeto devolvido. O formato deve seguir a especificação RFC850 (datas GMT).
- **Server:** Inclui informações sobre o servidor como o seu nome e versão.

- **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. Um usuário poderá fazer com que o CGI imprima um cabeçalho Expires com a data atual ou no passado para evitar que ele seja armazenado em pastas de arquivos temporários (*cache*).
- **Last-Modified:** Este cabeçalho indica a data e a hora da última modificação de um recurso. O formato é igual do de Date. O browser usa esta informação para guardar os arquivos no *cache*.
- **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 (cria) um cookie. *Por exemplo:* `Set-Cookie: usuario=fulano`. Esta propriedade poderá ser usada várias vezes, para criar mais de 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).

Entre as propriedades da cabeçalho do servidor, a propriedade `Content-type` é uma das mais importantes. Ela informa ao browser o tipo dos dados que ele está recebendo. Sem essa informação, 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 (o `Content-Type`, por exemplo) 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* for `C:\WINDOWS\COMMAND` e o browser solicitar

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

o servidor devolverá o arquivo para download (jamais o executará):

```
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
```

```
éü ``Converted____MZx_U_I_ ____ÿÿ__ø (...)
```

8.5. Método HEAD

O método HEAD é semelhante ao método GET. A única diferença entre os dois 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*.

8.6. Execução de programas

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
```

neste exemplo chamado de `/cgi-bin` (é preciso que o servidor esteja configurado para reconhecer `/cgi-bin` como um diretório que *não* fornece documentos, mas inicia a execução de programas.) Desta vez, supondo o mesmo servidor *Windows* do exemplo anterior, o programa será executado na máquina servidora e a saída do programa poderá ser retornada para o cliente (essa segunda parte provavelmente não ocorrerá nesse caso pois o arquivo não gera os cabeçalhos CGI.).

A extensão do nome 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 porque ele foi instruído a analisar o código de arquivos terminados em `.asp` e tentar interpretá-los antes de devolvê-los para o browser.

Geralmente 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.

8.7. 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.

9. *Uso de programas CGI*

Para rodar programas CGI é preciso configurar o servidor e, muitas vezes, os próprios programas. Este capítulo distingue vários tipos de programas CGI e mostra como um Web designer pode usá-los sem precisar saber nada sobre a linguagem de programação na qual foram escritos.

A maior parte dos provedores de acesso possui diversos CGIs mais populares já instalados e disponíveis através de um diretório cgi-bin central. Antes de transferir e tentar instalar um programa, procure saber se o seu provedor já não possui o serviço. Programas que são geralmente disponibilizados pelos provedores incluem contadores, leitores de formulários, programas que enviam e-mail, livros de visitas e sistemas de busca local. Se o seu provedor já possui um desses programas instalados, descubra como usá-los. Pode ser tão simples como escrever a URL do programa no campo ACTION do seu formulário.

Se você decidir aprender CGI para criar aplicações, terá que aprender uma linguagem de programação. Nas seções finais deste capítulo, alguns programas CGI escritos em linguagens interpretadas são analisados. Nessas seções será mostrado como alterar programas e como criar aplicações simples sabendo-se um mínimo de programação.

9.1. *Tipos de programas CGI*

Programas CGI podem ser interpretados ou compilados. Geralmente não é possível descobrir qual a linguagem em que foram escritos os programas compilados. Eles são os mais fáceis de instalar. Alguns (os comerciais) possuem uma rotina de instalação (tipo *InstallShield*). Outros só precisam ser copiados para um diretório executável para começarem a funcionar. Grande parte dos programas necessita de configuração, mesmo que seja através do formulário HTML.

Programas interpretados são mais fáceis de modificar, já que o código-fonte está sempre disponível. A desvantagem é que eles são mais difíceis de configurar. Não basta copiar o programa para um diretório executável. É preciso associá-lo ao interpretador e, muitas vezes, realizar configurações adicionais no servidor Web.

Programas em MS-DOS e Shell

Pode-se criar programas CGI simples usando a linguagem MS-DOS, interpretada pelo programa Command.COM do Windows. MS-DOS é uma linguagem muito limitada e pode-se tornar complexa e ineficiente quando usada para gerar programas longos. Ela é útil apenas para testes em servidores que rodam em Windows.

Alguns servidores poderão ser incapazes de rodar programas em MS-DOS já que tais aplicações rodam de maneira diferente das aplicações Windows. Programas MS-DOS só podem ser instalados em servidores que distinguem CGI do Windows de CGI do MS-DOS.

Shell é a linguagem de linha de comando do Unix. Apesar da semelhança inicial, é bem mais poderosa que MS-DOS e permite a criação de programas bem mais sofisticados. Mas um programa em Shell pode ficar muito complexo e ineficiente se for muito grande e realizar muitas tarefas repetitivas. Grandes programas em Shell costumam precisar usar recursos de outras linguagens como AWK e Tcl/Tk no Unix.

Shel é, porém, útil para realizar testes em servidores e para rotinas curtas e simples. Antes da adoção de Perl como linguagem preferencial, a maior parte dos programas CGI disponíveis na Web eram escritas em Shell.

Programas em Perl

Perl é a linguagem mais popular usada no desenvolvimento de programas CGI interpretados. Ela possui vários recursos que auxiliam o desenvolvimento de tais aplicações e é interpretada, facilitando a modificação e a sua portabilidade. Existem interpretadores Perl para os sistemas Windows, Unix, Macintosh e diversos outros.

Diversos sites na Internet disponibilizam programas CGI em Perl que, geralmente, podem ser usados gratuitamente. Muitos são configuráveis sem que seja preciso mexer no código. Além de ser aplicado em CGI, Perl também é bastante usado na administração de sistemas Unix e Windows (substituindo o editor de registro). A última versão estável é Perl 5.005. A implementação do Perl no Windows pode ser obtida na ActiveState (www.activestate.com).

Para instalar programas CGI em Perl é preciso associar o endereço do interpretador com o programa fonte. A forma de realizar essa associação varia entre servidores e sistemas operacionais.

9.2. Onde conseguir programas CGI

Há vários sites na Internet que disponibilizam programas CGI gratuitamente para download. Outros cobram taxas após um período de uso. Os melhores programas permitem a instalação e configuração sem que seja preciso alterar uma linha sequer do código. Costumam ser distribuídos com um manual (em HTML) e exemplos de aplicações completas (páginas HTML integradas aos programas). A documentação informa ao usuário como configurar o programa para que possa rodá-lo no seu servidor.

Abaixo está uma lista de alguns sites que distribuem programas CGI. Na página dedicada a este curso você encontrará outros endereços e programas prontos como contadores, leitores de formulários, livros de visitas, sistemas de busca, entre outros.

- www.freescrpts.com/ - diversos scripts CGI de domínio público
- www.geocities.com/SiliconValley/Orchard/6104/ - CGIs gratuitos e comerciais.
- cgi-lib.berkeley.edu/ - biblioteca para construção de aplicações CGI em C e Perl
- www.cpan.org/ - repositório de módulos Perl
- www.cgi-resources.com/ - diversos scripts em Perl, C, C++, Tcl, AppleScript
- perfect.com/freescrpts/ - diversos programas gratuitos
- www.nethernet.com/free/ - diversos programas gratuitos
- www.cgi-world.com/ - repositório de programas CGI em várias linguagens
- www.worldwidemart.com/scripts/ - programas úteis em C e em Perl

9.3. Como configurar e instalar os programas

Instalar programas compilados é simples: basta copiar o programa para um diretório que permita a sua execução pelo sistema operacional. Programas interpretados são mais complicados de instalar pois requerem a existência entre o código-fonte e o interpretador. Não há uma forma padrão de realizar essa associação. Vários servidores a fazem de maneiras diferentes. No Unix, que faz a associação é sempre o sistema operacional. No Windows, na maior parte das vezes é preciso criar uma associação no Windows e no servidor. No Personal Web Server a associação precisa ser realizada no Registro do Windows.

Para programas escritos em Perl para execução através do servidor Apache, o caminho até o programa interpretador deverá ser feita na primeira linha, após os caracteres `#!`. Quem acha o interpretador para o programa no Apache Linux é o sistema operacional. Já no Windows, é o próprio Apache quem o localiza.

Para instalar um programa CGI em Perl escrito por outra pessoa, é necessário descobrir qual o endereço onde está instalado o seu interpretador Perl. A forma mais fácil de fazer isto é perguntar ao seu administrador do sistema. Locais típicos no Linux são `/usr/local/bin/` e `/usr/bin/`. Se o programa estiver localizado no primeiro subdiretório, a primeira linha do programa, portanto, deverá ser alterada para:

```
#!/usr/local/bin/perl
```

O Apache Windows associa o programa ao interpretador da mesma forma. Suponha que o Perl esteja instalado em `c:\perl\`. O programa executável deverá estar em `c:\perl\bin\`, portanto, a primeira linha do programa deverá conter:

```
#!c:\perl\bin\perl.exe
```

Há outras variáveis que talvez precisem ser mudadas. Um bom script de propósito geral contém documentação suficiente para explicar cada configuração necessária. Geralmente tais alterações necessárias são realçadas com comentários e aparecem bem no início do programa. A maioria requer que o usuário digite informações sobre o sistema que está utilizando, como endereço do interpretador, caminho completo de seu diretório CGI, nome e IP de sua máquina, endereço do programa de e-mail, localização do banco de dados, etc.

Programas de propósito geral também costumam requerer configuração especial na página HTML, geralmente através de elementos `<input type=hidden>`. Os melhores programas são totalmente configuráveis através de tais campos e por isso podem ser reutilizados mais de uma vez e por autores diferentes.

As seções a seguir mostrarão como configurar um dos mais populares programas para tratamento de formulários e seu envio por e-mail. O roteiro `FormMail.pl` disponível gratuitamente em *Matt's Script Archive* (www.worldwidemart.com/scripts/).

Configuração e uso do FormMail.pl

Consulte anexos distribuídos no site deste curso.

Alterações no código-fonte

Os programas distribuídos pela Internet geralmente são bastante configuráveis pelo autor de um site através de HTML pois destinam-se a um público que não pode ou não quer mexer com programação. Mesmo assim, há vários programas configuráveis que não são flexíveis o suficiente. Por exemplo, o `FormMail`, embora possa sempre ser redirecionado para páginas escolhidas pelo autor da página que contém o formulário, ainda mostra mensagens em inglês que não podem ser alteradas a menos que se manipule o código. A própria mensagem enviada para quem preencheu o formulário contém partes em inglês que não se pode remover sem ter acesso ao código.

Não basta mudar o texto contido nos comandos `print` para alterar as mensagens. Perl coloca variáveis (palavras começando com `$`) dentro do texto e removê-las acidentalmente poderá causar erros. Essas alterações no código fogem ao escopo deste curso, mas, se você decidir explorar a linguagem Perl (apêndice desta apostila), não deixe de voltar e analisar o código desse programa. Ele se tornará bem mais simples.

9.4. Como criar programas CGI simples

É fácil criar programas CGI simples com um conhecimento mínimo de linguagens de programação. Ir além disso requer uma dedicação maior aos recursos oferecidos pela linguagem. Nesta seção serão apresentados pequenos programas que listam variáveis de ambiente e isolam variáveis passadas via formulário. Eles requerem conhecimentos mínimos de Bourne Shell (linha de comando Unix), Perl (veja apêndice desta apostila) e Visual Basic (não abordada neste curso). Não são analisados exemplos de programas que salvam em disco nem conectam-se a bancos de dados pois tais recursos não dependem de CGI mas da linguagem utilizada.

Listagem das variáveis de ambiente em Shell e MS-DOS

O programa a seguir lista diversas variáveis de ambiente geradas pela requisição do browser. É um programa escrito em *Bourne Shell*, a linguagem usada para interpretar a linha de comando do Unix. Observe que na primeira linha foi informado o caminho até o interpretador. Isto é essencial. Também é necessário mudar o bit executável (usando `chmod`) para que o programa rode de forma independente.

O programa abaixo não gera HTML, por isso o tipo de dados é `text/plain`. Se você quiser mudar o tipo para `text/html` não se esqueça de imprimir também a marcação HTML como elementos `<P>` antes de cada parágrafo e um elemento `<h1>` em volta do título.

```
#!/bin/sh

echo "Content-type: text/plain"
echo ""
echo "Variáveis de ambiente"
echo "-----"

echo "SERVER_SOFTWARE: " $SERVER_SOFTWARE
echo "SERVER_NAME: " $SERVER_NAME
echo "GATEWAY_INTERFACE: " $GATEWAY_INTERFACE
echo "CONTENT_LENGTH: " $CONTENT_LENGTH
echo "CONTENT_TYPE: " $CONTENT_TYPE
echo "DOCUMENT_NAME: " $DOCUMENT_NAME
echo "DOCUMENT_ROOT: " $DOCUMENT_ROOT
echo "DOCUMENT_URI: " $DOCUMENT_URI
echo "DATE_LOCAL: " $DATE_LOCAL
echo "DATE_GMT: " $DATE_GMT
echo "LAST_MODIFIED: " $LAST_MODIFIED
echo "HTTP_COOKIE: " $HTTP_COOKIE
echo "HTTP_ACCEPT: " $HTTP_ACCEPT
echo "HTTP_USER_AGENT: " $HTTP_USER_AGENT
echo "HTTP_REFERER: " $HTTP_REFERER
echo "PATH: " $PATH
echo "PATH_INFO: " $PATH_INFO
echo "PATH_TRANSLATED: " $PATH_TRANSLATED
echo "QUERY_STRING: " $QUERY_STRING
echo "QUERY_STRING_UNESCAPED: " $QUERY_STRING_UNESCAPED
echo "REMOTE_HOST: " $REMOTE_HOST
echo "REMOTE_ADDR: " $REMOTE_ADDR
echo "REQUEST_METHOD: " $REQUEST_METHOD
echo "SERVER_PROTOCOL: " $SERVER_PROTOCOL
echo "SERVER_PORT: " $SERVER_PORT
echo "SERVER_ROOT: " $SERVER_ROOT
echo "SCRIPT_NAME: " $SCRIPT_NAME
echo "-----"
```

Para rodar o programa, depois que ele estiver instalado em um diretório CGI, simplesmente chame-o através de uma URL no seu browser. Utilize-o para receber dados de formulários e veja os diferentes resultados.

O programa acima poderá rodar também em sistemas *Windows* se for salvo em um arquivo de texto com a extensão `.BAT`. Para isto é necessário que as variáveis de ambiente sejam chamadas não precedidas por um `$` mas entre `%` e `%`. A primeira linha também deve ser eliminada pois ela não faz sentido em MS-DOS. Não é preciso identificar o local do interpretador pois o interpretador `Command.COM`, que interpreta o MS-DOS é chamado automaticamente quando a extensão do arquivo é `*.BAT`. Deve-se substituir a primeira linha por:

```
@echo off
```

Para que não seja impressa na página enviada para o browser o nome dos comandos (`echo`) e o *prompt* do MS-DOS. Também é necessário substituir as linhas

```
echo ""
```

que imprime uma linha em branco no Unix, por

```
echo.
```

que faz o mesmo em MS-DOS.

Alguns servidores Web têm problemas ao rodar arquivos `.BAT`. Eles não rodam em servidores que suportam apenas Win-CGI. É preciso ter um diretório `cgi-bin` que esteja preparado para rodar CGI via MS-DOS. Uma melhor alternativa para o desenvolvimento de programas CGI em *Windows* é a linguagem *Visual Basic* ou *Perl*.

Listagem de variáveis com *Visual Basic*

Programas em Visual Basic devem ser compilados antes que sejam utilizados em CGI, usando a biblioteca `CGI.BAS` (ou outra equivalente). O código abaixo lista os dados enviados através de um formulário HTML (os pares nome/valor).

```
Sub CGI_main()
    Dim loop as Integer
    Dim info as String

    Send ("Content-type: text/html")
    Send ("")
    Send ("<h1>Dados digitados no formulário</h1>")

    For loop = 0 to CGI_NumFormTuples - 1
        Send ("<p><b>" & CGI_FormTuples(loop).key)
        Send ("</b>: " & CGI_FormTuples(loop).value)
    Next loop
End Sub
```

O programa compilado pode ser armazenado em qualquer diretório que suporte Win-CGI como o diretório `cgi-bin` do *Personal Web Server*. As funções e variáveis iniciadas em `CGI_` (em negrito) fazem parte da biblioteca `CGI.BAS`.

Perl

No capítulo anterior havia um pequeno formulário HTML que enviava para o servidor as variáveis nome, idade e cor. O seu código está repetido abaixo:

```
<FORM ACTION="/cgi-bin/bd.pl" 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>

```

Em Perl há uma biblioteca de funções para CGI que permite a decodificação automática dos dados. É a biblioteca `cgi-lib.pl`². O programa a seguir, escrito em Perl, usa essa biblioteca para transformar os dados recebidos e colocá-los em uma variável Perl que chamamos de `%dados`. Essa variável é capaz de armazenar pares nome valor que podem ser recuperados através da sintaxe `$dados{ 'nome' }`. Isto quer dizer que, para armazenar o valor contido no atributo NAME “cor” do HTML em uma variável chamada `$cor`, pode-se fazer

```
$cor = $dados{'cor'};
```

O programa `bd.pl` listado abaixo faz isto. Primeiro importa a biblioteca (com a instrução `require`). Depois chama a função `ReadParse`, que obtém os dados da entrada padrão e coloca os pares nome/valor no vetor `%dados`. Em seguida obtém o valor de cada atributo (nome, idade e cor). No final, imprime o cabeçalho `Content-type` e gera uma página HTML contendo os dados digitados.

```

#!/usr/local/bin/perl

require "cgi-lib.pl";
&ReadParse(\%dados);

$nome = $dados{'nome'};
$idade = $dados{'idade'};
$cor= $dados{'cor'};

print "Content-type: text/html\n\n";

print "<h1>Dados enviados</h1> \n";
print "<p>Nome: $nome\n";
print "<p>Idade: $idade\n";
print "<p>Cor: $cor\n";

```

As informações devolvidas em uma página HTML também poderiam ser salvas em disco ou enviadas por e-mail.

Para rodar o programa acima no *Apache Windows*, basta mudar a primeira linha informando o endereço do interpretador Perl instalado. Em outros servidores isto pode não ser necessário.

² Per 5 possui um módulo chamado CGI:: que faz a mesma coisa e é tão simples de usar quanto o `cgi-lib.pl`. Embora o módulo CGI:: seja parte da distribuição do Perl 5, decidimos não usá-lo porque ele não está disponível em algumas distribuições mais antigas do Perl para Windows e Macintosh.

9.5. *Exercícios*

1. Instale a página de feedback que você fez em exercício de um capítulo anterior no seu servidor Web local (Apache). Escreva um programa CGI em Perl (como o do exemplo acima) que leia os dados digitados e os imprima na tela.
2. Faça um construtor de frases: um formulário HTML que receba várias palavras e escolhas do usuário e construa uma frase com ela, devolvendo uma página HTML com a frase construída.