11
Cookies

UM COOKIE É UMA PEQUENA QUANTIDADE DE INFORMAÇÃO que pode ser armazenada na máquina do cliente através de instruções enviadas pelo servidor ou contidas em uma página HTML. É uma informação que pode persistir por toda uma sessão do cliente em um site, ou por mais tempo ainda. Um cookie pode ser gravado em uma página e recuperado em outra, permitindo o acesso a propriedades, informações ou preferências do usuário a qualquer momento, de qualquer página do site. .

Um cookie está sempre associado a um browser e a um domínio. Não é um padrão formal ou especificação, e a implementação dos cookies é dependente de browser e fabricante. A sua manuipulação, porém, é baseada em padrões HTTP (cabeçalhos) e tem amplo suporte tanto de tecnologias client-side, como JavaScript, como de tecnologias server-side como ASP, Servlets, LiveWire e CGI.

O objetivo deste capítulo é demonstrar o uso de cookies em JavaScript, e apresentar algumas aplicações como o "Carrinho de Compras". A próxima seção, introduz a arquitetura de cookies cujo conhecimento é essencial para o uso eficiente de cookies com JavaScript.

Cookies em HTTP

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 o domínio onde o cookie é válido, e o tempo de validade do mesmo.

Criação de cookies via cabeçalhos HTTP

Cookies podem ser criados através de um cabeçalho HTTP usando CGI. Toda resposta do servidor a uma requisição do browser sempre contém um conjunto de cabeçalhos com informações sobre os dados enviados. Essas informações são essenciais para que o browser consiga decodificar os dados, que ele recebe em pedaços, como um fluxo irregular de bytes. Os cabeçalhos trazem o comprimento total dos dados, o tipo dos dados (se é imagem, página HTML, som, etc.), a versão e nome do servidor, entre outras informações. Um exemplo típico de resposta de um servidor Web a um browser que solicita uma página HTML está mostrada abaixo:

```
HTTP/1.0 200 OK
Date: Friday, June 13, 1997
Server: Apache 1.02
Content-type: text/html

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

A primeira linha acima não é um cabeçalho, mas o *status* da resposta do servidor. No caso acima, indica sucesso através do código 200. Um outro código freqüente na Web é o código 404, correspondente à recurso não encontrado.

Toda linha de cabeçalho HTTP tem a forma:

```
NomeDoCabeçalho: Valores ↵
```

onde ↵ corresponde a uma quebra de linha. O nome do cabeçalho será ignorado pelo browser, se ele não souber o seu significado. Os valores têm um formato específico para cada cabeçalho. O conjunto de caracteres suportado é ASCII de 7 bits, portanto, é necessário converter acentos e outros caracteres antes de usá-los como cabeçalhos.

O bloco de cabeçalhos é separado dos dados por uma linha em branco (dois caracteres de nova-linha seguidos). Ao receber a resposta, o browser separa o cabeçalho do restante da informação, identifica o formato e comprimento dos dados (que vêm depois da linha em branco) e os formata na sua área de visualização, se o seu tipo de dados for suportado.

Um bloco de cabeçalhos de resposta é gerado pelo servidor Web sempre que o browser solicita uma página estática. 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:

```
(... outros cabeçalhos ...)
Set-Cookie: cliente=jan0017
Set-Cookie: nomeclt=Marie
Content-type: text/html

(... dados ...)
```

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 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).

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>
```

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.

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 na maior parte dos servidores.

Cookies em JavaScript

Cookies podem ser manipulados em JavaScript através da propriedade `document.cookie`. Esta propriedade contém uma *String* com o valor de todos os cookies que pertencem ao espaço de nomes (domínio/caminho) do documento que possui a propriedade. A propriedade `document.cookie` é usada tanto para criar como para ler cookies.

Para definir um novo cookie, basta atribuir um string em um formato válido para o cabeçalho HTTP `Set-Cookie` à propriedade `document.cookie`. Como cookies não podem ter espaços, ponto-e-virgula e outros caracteres especiais, pode-se usar a função `escape(String)` antes de armazenar o cookie, para garantir que tais caracteres serão preservados em códigos hexadecimais:

```
nome="usuario";
valor=escape("João Grandão"); // converte para Jo%E3o%20Grand%E3o
vencimento="Monday, 22-Feb-99 00:00:00 GMT";
document.cookie = nome + "=" + valor + "expires=" + vencimento;
```

A propriedade `document.cookie` não é um tipo *string* convencional. Não é possível apagar os valores armazenados simplesmente atribuíndo um novo valor à propriedade. Novos valores passados à `document.cookie` criam novos cookies e aumentam a *string*:

```
document.cookie = "vidacurta=" + escape("É só por hoje!");
document.cookie = "vidalonga=" + escape("É por duas semanas!") +
                  "expires=" + vencimento;
```

Os cookies estão todos armazenados na propriedade `document.cookie`, em um string que separa os cookies pelo “`;`”. Se o valor de `document.cookie` for impresso agora:

```
document.write(document.cookie);
```

O texto a seguir será mostrado na página, com os três cookies separados por “`;`”:

```
usuario=Jo%E3o%20Grand%E3o; vidacurta=%C9%20s%F3%20por%20hoje%21;
vidalonga=%C9%20por%20duas%20semanas%21
```

As letras acentuadas, espaços e outros caracteres especiais foram substituídos pelo seu código hexadecimal, após o “`%`”. Para decodificar o texto, pode-se usar `unescape()`:

```
document.write(unescape(document.cookie));
```

Se não for definido um campo *expires* com uma data no futuro, o cookie só sobreviverá à sessão atual do browser. Assim que o browser for fechado, ele se perderá. Por exemplo, se a sessão atual do browser for encerrada e o browser for novamente iniciado,

carregando a página que imprime `document.cookie`, teremos apenas dois cookies, e não três como antes. Isto porque o cookie `vidacurta` foi definido *sem* o campo expires:

```
usuario=Jo%E3o%20Grand%E3o; vidalonga=%C9%20por%20duas%20semanas%21
```

Geralmente queremos definir o tempo de validade de um cookie de acordo com um intervalo relativo de tempo e não uma data absoluta. O formato de data gerado pelo método `toGMTString()` de *Date* é compatível com o formato aceito pelos cookies. Sendo assim, podemos criar, por exemplo, uma função para definir um cookie com validade baseada em um número de dias a partir do momento atual:

```
function setCookie(nome, valor, dias) {
   diasms = (new Date()).getTime() + 1000 * 3600 * 24 * dias;
   dias = new Date(diasms);
   expires = dias.toGMTString();
   document.cookie = escape(nome) + "=" +
                     escape(valor) + "; expires=" + expires;
}
```

A função acima pode ser chamada tanto para criar cookies como para *matar* cookies no mesmo espaço de nomes. Para criar um novo cookie, com duração de 12 horas:

```
setCookie("cook", "Um, dois, tres", 0.5);
```

Para matar o cookie criado com a instrução acima, basta criar um homônimo com data de vencimento no passado. Podemos fazer isto passando um *número negativo* como tempo de validade em `setCookie()`:

```
setCookie("cook", "", -365);
```

Para extrair as informações úteis de um cookie, usamos os métodos de *String* que realizam operações com caracteres (`indexOf()`, `substring()`, `split()`). Uma invocação do método `split(";")` coloca todos os pares nome/valor em células individuais de um vetor.

```
cookies = document.cookie.split(";");
// cookies[0] = "usuario=Jo%E3o%20Grand%E3o"
// cookies[1] = "vidacurta=%C9%20s%F3%20por%20hoje%21"
// cookies[2] = "vidalonga=%C9%20por%20duas%20semanas%21"
```

Uma segunda invocação de `split()`, desta vez sobre cada um dos pares nome/valor obtidos acima, separando-os pelo "=", cria um vetor bidimensional. O string `cookies[i]` se transforma em um vetor para receber o retorno de `split("=")`. Criamos então duas novas propriedades: `name` e `value` para cada cookie, que contém respectivamente, o nome e valor, já devidamente decodificados:

```
for (i = 0; i < cookies.length; i++) {
   cookies[i] = cookies[i].split("=");
   cookies[i].name = unescape(cookies[i][0]);
   cookies[i].value = unescape(cookies[i][1]);
}
```

Carrinho de compras

Os cookies são essenciais na construção de aplicações de comércio eletrônico, pois permitem passar informações de uma página para outra e manter os dados persistentes na máquina do cliente por mais de uma sessão.

O carrinho de compras virtual consiste de um ou mais cookies que guardam as preferências do usuário enquanto ele faz compras pelo site. No final, quando o usuário decide fechar a conta, o(s) cookie(s) são lido(s), os dados são extraídos, formatados e mostrados na tela ou enviados para o servidor. Mesmo que a conexão caia ou que ele decida continuar a compra no dia seguinte, os dados podem ser preservados, se os cookies forem persistentes (terem um campo *expires* com uma data de vencimento no futuro). No final, depois que o usuário terminar a transação, o cookie não é mais necessário e é descartado.

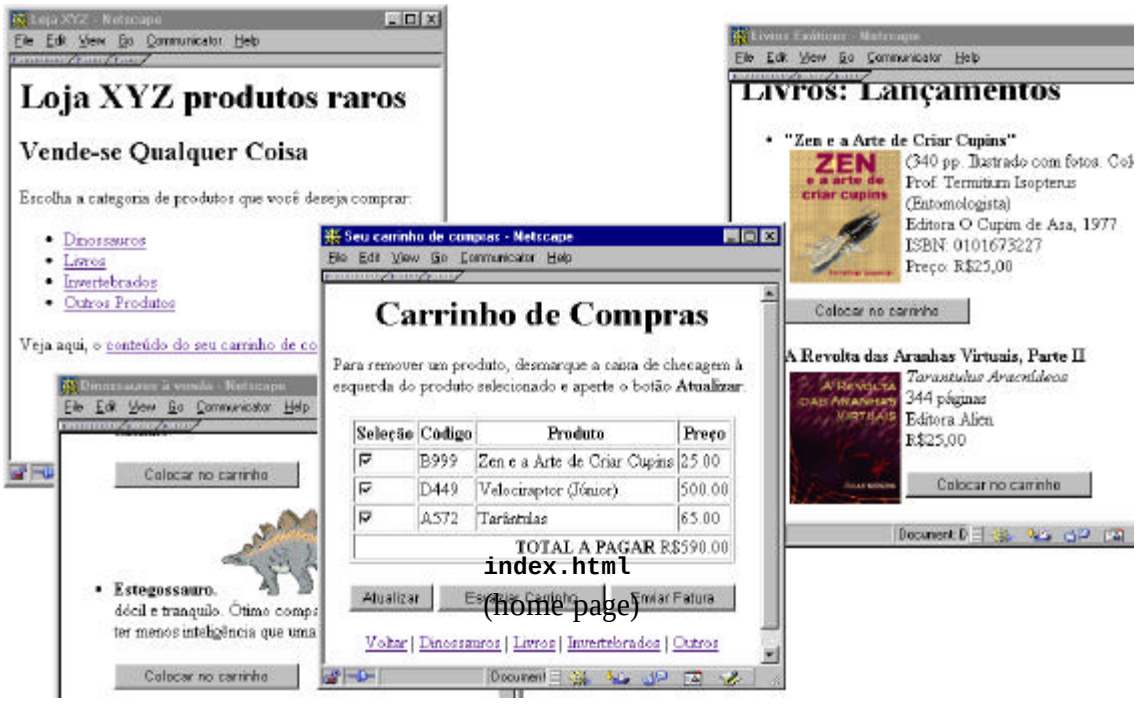
No exercício a seguir, desenvolveremos uma pequena aplicação de comércio eletrônico usando cookies e JavaScript.

Exercício Resolvido

A Loja XYZ S/A deseja vender seus produtos esquisitos na Web. A implantação do serviço consiste de duas etapas. A primeira é a criação de um carrinho de compras virtual para que os clientes possam selecionar seus produtos. A segunda etapa envolve questões relacionadas ao servidor, como o acesso ao banco de dados da empresa, segurança, etc. Ficamos encarregados de desenvolver a primeira etapa e decidimos usar JavaScript.

A sua tarefa é desenvolver os requisitos mínimos para lançar a versão 0.1 da aplicação. Esta versão ainda não é adequada à publicação no site do cliente, mas já possui as características mínimas para demonstrar os principais recursos do site. Os arquivos HTML já estão prontos no subdiretório `cap11/carrinho/`.

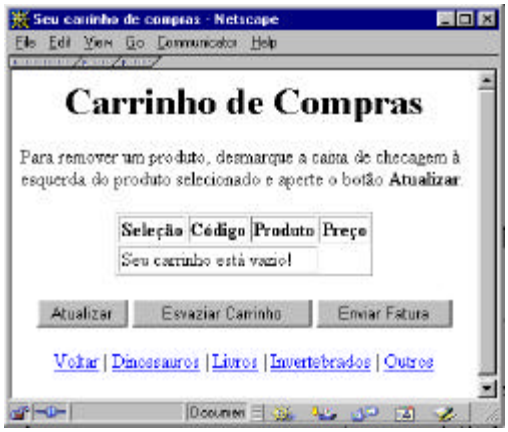
Várias páginas da aplicação estão mostradas na figura abaixo. A primeira é a home



a) Implementar os botões “Colocar no Carrinho” de cada produto para que armazenem um cookie contendo:

   - um código de uma letra e três dígitos (por exemplo, X123, F555)
   - uma descrição do produto
   - preço do produto (por exemplo 5.99, 123.25)

   O código e o preço podem ter valores arbitrários. O cookie deve durar 5 dias.



b) Implementar a página carrinho.html para exibir os cookies armazenados. Esta página pode ser alcançada a partir de qualquer página do site (através de links comuns). Quando for carregada, deve mostrar na tela uma tabela com as informações armazenadas nos cookies. Se não houver cookies, ela deve ter a aparência da figura ao lado. A tabela deve ter o seguinte formato:

   - *Primeira coluna: Checkbox* marcado. Se o usuário atualizar a página, o valor deste componente deve ser verificado. Se for `false`, o cookie correspondente deve ser eliminado.
   - *Segunda e terceira colunas:* código do produto e descrição obtidas a partir do cookie armazenado.
   - *Quarta coluna:* preço do produto obtido a partir do cookie. Todos os valores desta coluna devem ser somados e o total exibido no final da tabela.

c) Implementar o botão “Atualizar”, que deverá recarregar a página e eliminar os cookies que não estiverem marcados.

d) Implementar os botões “Esvaziar Carrinho”, que deverá eliminar todos os cookies e atualizar a página, e “Enviar Fatura”, que deverá eliminar todos os cookies e mostrar na tela um diálogo de alerta informando a ocorrência.

Solução

A solução do problema consiste de três partes: a criação dos cookies (a colocação dos produtos no carrinho), a leitura dos cookies (a visualização do conteúdo do carrinho) e a remoção dos cookies (o esvaziamento do carrinho).

Para criar os cookies (a), usamos a função `setCookie()` abaixo. Ela pode estar presente em todas as páginas de produtos ou em um arquivo externo (`.js`), importado em

cada página. A função recebe três argumentos apenas (estamos supondo que este domínio não terá outros cookies) que são um nome, um valor e um período de validade em dias:

```
<script>
   function setCookie(nome, valor, dias) {
      diasms = (new Date()).getTime() + 1000 * 3600 * 24 * dias;
      dias = new Date(diasms);
      expires = dias.toGMTString();
      document.cookie = escape(nome) + "=" +
                        escape(valor) + "; expires=" + expires;
   }
</script>
```

Precisamos armazenar três informações por produto. Se usássemos três cookies para cada produto, em pouco tempo ficaríamos sem cookies, pois o browser limita o número de cookies em 20 por domínio. Precisamos, portanto, armazenar as informações em o mínimo de cookies possível. Optamos, nesta primeira versão, por definir um cookie por produto[1]. Para separar os dados, usamos o "&" como delimitador do *string*:

```
<form>
  <input type=button value="Colocar no carrinho"
         onclick="setCookie('dino1','D372&Brontossauro&1500.00',5)">
</form>
```

Os códigos, nomes de cookie e preços escolhidos são totalmente arbitrários. Tendo todos os botões implementados da forma acima, com nomes distintos para cada cookie, podemos armazenar as informações no carrinho de compras apertando o botão ao lado de cada produto.

Para recuperar os cookies (b), precisamos alterar apenas a página `carrinho.html`. Nesta página, também iremos precisar da função `setCookie()` para apagar os cookies, como foi pedido no requisito (c). Para ler os cookies, definimos duas funções: `getCookies()`, retorna um vetor bidimensional com todos os cookies disponíveis:

```
<script>
   // devolve todos os cookies em uma matriz (num_cookies x 2)
function getCookies() {
      cookies = document.cookie.split("; ");
      for (i = 0; i < cookies.length; i++) {
         cookies[i] = cookies[i].split("=");
         cookies[i][0] = unescape(cookies[i][0]); // nome
         cookies[i][1] = unescape(cookies[i][1]); // valor
      }
      return cookies;    // retorna matriz[n][2]
   }
```

[1] Ainda não é a melhor idéia, pois aproveitamos pouco dos 4kB permitidos a cada cookie. Idealmente colocaríamos vários produtos em um único cookie e evitaríamos armazenar informações como descrição de produtos que poderiam ser recuperadas do banco de dados.

e `getCookie(nome)`, que chama `getCookies()` e retorna um cookie pelo nome:

```
      // retorna o valor de um cookie fornecendo-se o nome
function getCookie(name) {
      cookies = getCookies();
      for (i = 0; i < cookies.length; i++) {
         if(cookies[i][0] == name) {
            return cookies[i][1];      // valor
         }
      }
      return null;
   }
</script>
```

Com essas funções, temos condições de montar a tabela com os produtos. Como estamos supondo que não há outros cookies neste domínio[2] podemos verificar se o carrinho está vazio, simplesmente verificando se o string `document.cookie` está vazio:

```
<table border><tr>
<th>Seleção</th><th>Código</th><th>Produto</th><th>Preço</th></tr>
<script>
if (!document.cookie)
   document.write("<tr><td colspan=3>Seu carrinho está vazio!</td></tr>");
(...)
```

Caso existam cookies, os recuperamos e colocamos na variável `cooks`. Partimos as informações no valor de cada cookie (`cooks[i][1]`) pelo "`&`" e utilizamos a informação de preço (última das três informações armazenadas em cada cookie) para montar o total. Usamos o nome de cada cookie para definir o nome de um *Checkbox*, que é criado previamente ligado (contém atributo `CHECKED`):

```
else {
   cooks = getCookies();
   total = 0;
   for (i = 0; i < cooks.length; i++) {
      partes = cooks[i][1].split("&"); // partes eh Array
      total += parseFloat(partes[partes.length-1]);
      partes = partes.join("</td><td>");  // agora partes eh String
      partes = "</td><td>" + partes + "</td></tr>";
      partes = "<tr><td><input type=checkbox name='"
               + cooks[i][0] + "' checked size=3>" + partes;
      document.write(partes + "\n");
   }
   document.write("<tr><td colspan=4 align=right><b>TOTAL A PAGAR</b> R$"
                  + formata(total) + "</td></tr>");
}
```

[2] Teremos que levar em conta a possibilidade de haver outros cookies neste domínio em uma versão definitiva.

```
</script>
</table>
```

Utilizamos a função `formata(numero)` para formatar o total (não está mostrada aqui[3]), para que ele seja exibido com duas casas após o ponto decimal.

O requisito (c) pede para implementar uma função que atualize o carrinho de compra, verificando se algum cookie deve ser removido. O botão atualizar chama a função `atualiza()`, mostrada abaixo, que localiza todos os *Checkbox* da página e extrai o seu nome (que corresponde ao nome de um cookie existente). Se algum *Checkbox* está desligado, um cookie com o seu nome será definido com data de vencimento no passado, o que provocará a morte do cookie:

```
function atualiza() {
   for(i = 0; i < document.forms[0].elements.length; i++) {
      if(document.forms[0].elements[i].type == "checkbox") {
         chkbox = document.forms[0].elements[i];
         nome = chkbox.name;
         if(!chkbox.checked) {
            setCookie(nome,"nada",-365); // mata  cookie
         }
      }
    }
    location.reload(true);     // atualiza a página
}
```

O último requisito (d) consiste da programação dos botões “Esvaziar Carrinho” e “Enviar Fatura”, que nesta versão, fazem praticamente a mesma coisa: matam todos os cookies. Em uma versão definitiva desta aplicação, o botão enviar fatura deverá enviar os dados para um lugar seguro antes de matar os cookies.

Criamos uma função especialmente para eliminar todos os cookies: `killAll()`. Ela localiza os cookies um a um e os redefine com uma data de um ano atrás.

```
function killAll() {
   if (document.cookie) {
      cooks = document.cookie.split("; ");
      for (i=0; i < cooks.length; i++) {
         nome = cooks[i].split("=");
         setCookie(unescape(nome[0]), "aaa", -365); // mata
      }
   }
}
```

A chamada do botão “Esvaziar Carrinho”, além de matar todos os cookies, recarrega a página para que a mudança seja visível, usando `location.reload()`:

[3] Veja o código desta função na solução em cap10/carrinhosol/carrinho.html

```
<input type=button value="Esvaziar Carrinho"
       onclick="killAll(); location.reload(true)">
```

Veja a aplicação completa do carrinho de compras explorado neste exercício no subdiretório `cap10/carrinhosol/`.

Exercícios

11.1 Escreva um conjunto de funções gerais para definir cookies com todos os parâmetros possíveis (a função utilizada nos nossos exemplos apenas admite três parâmetros e não define cookies fora do espaço de nomes *default*). A função deve poder definir cookies com apenas os parâmetros nome e valor, com os três parâmetros que usamos nos exemplos ou com os 6 parâmetros possíveis. Deve ser possível criar um cookie das formas:

```
setCookie("visitante","Fulano de Tal", 10, "/sub","abc.com", true)
```

ou

```
setCookie("visitante","Fulano de Tal")
```

11.2 Escreva uma aplicação que informe ao usuário quantas vezes ele já visitou a página, quando foi, e de onde ele tinha vindo antes.