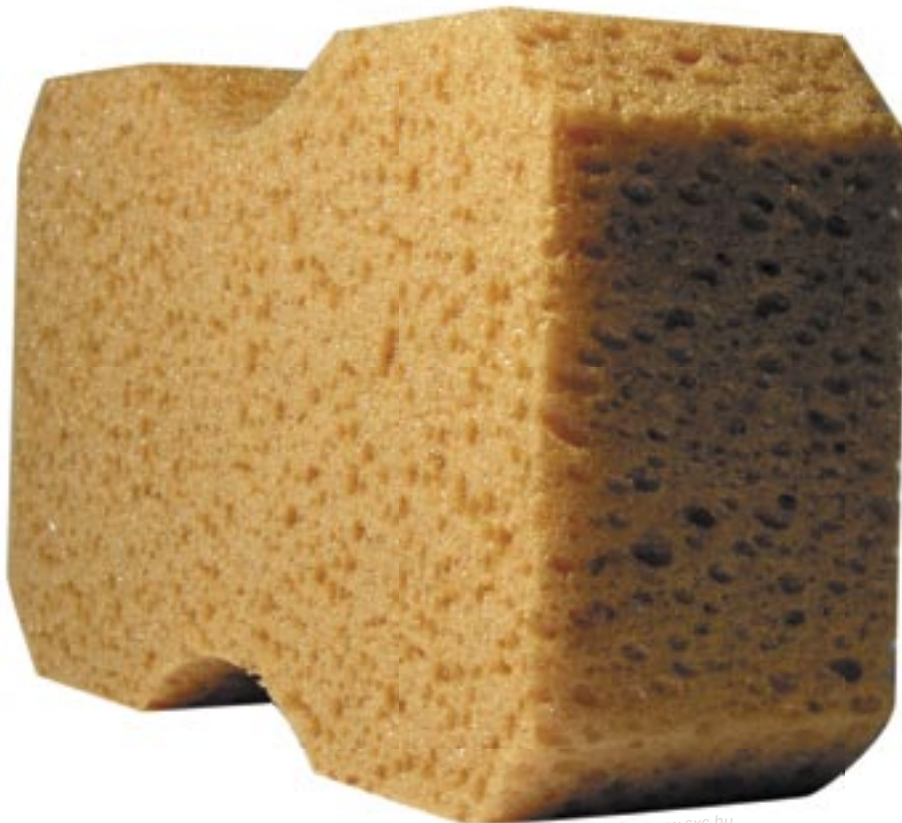


A nova tecnologia da Web

AJAX

AJAX é o termo da moda. O Google usa AJAX, o Yahoo usa AJAX...todo mundo quer usar o AJAX. Mas, e você, usa? E, o mais importante: que diabos é AJAX?

por José María Ruiz
e Pedro Orantes



massimo bassi - www.sxc.hu

“**B**rave New World” (“Admirável mundo novo”) é o nome do famoso romance de Aldous Huxley, que nos mostra um mundo diferente e aterrador, mas que parecia e parece cada vez mais próximo.

Não temos uma visão tão pessimista do mundo, mas é provável que esse título explique todo o alvoroço que o AJAX vem causando. O termo foi cunhado por Jesse James Garrett em [1].

Durante muito tempo as GUIs – interfaces gráficas de usuário – dominaram a informática. Os profissionais da Web estavam sempre tentando convencer o mundo de que, para a maioria dos programas, uma interface Web era o bastante. Mas os usuários estavam acostumados a certas características, como campos que são completados automaticamente, ou o arrastar e soltar, que eram impossíveis na Web.

Com o passar do tempo, cada vez mais pessoas propunham soluções. A lista é interminável: *JavaScript*, *Java Applets*, *Active X*, *Tcl*, *VBScript*, *Macromedia Flash*...

Mas todas falhavam, de uma maneira ou de outra. No caso do *Java*, para se executar o *applet* era necessário ter instalado a máquina virtual *Java*, e a maioria dos usuários nem sabiam o que era aquilo que estava sendo pedido. O mesmo ocorria com o *Macromedia Flash*.

O pior era que quando o tema “instalação do software adequado” estava

solucionado, os desenvolvedores criavam (como ainda criam) páginas horríveis, cheias de coisas se movendo, que distraem e irritam. Sentiam-se impulsionados a usar até a última funcionalidade das novas ferramentas e acabavam gerando monstruosidades.

Essa fase quase já passou, e agora felizmente se busca a simplicidade, e nesse exato momento surgiu o AJAX. Para mais informações, veja o site em [2].

O que é AJAX?

Muito boa pergunta. A verdade é que o AJAX estava o tempo todo debaixo de nossos narizes, esperando que alguma mente mais atenta o descobrisse. O termo “AJAX” consiste de um acrônimo de *Asynchronous JavaScript And XML* (*JavaScript* e *XML* assíncronos), e curiosamente sua existência se deve a uma dessas famosas violações dos padrões que a Microsoft costuma realizar com seus produtos.

Em 1998, a Microsoft introduziu dentro de seus produtos uma biblioteca que permitia fazer consultas usando o protocolo HTTP de maneira autônoma e assíncrona. Quando o seu navegador acessa uma página que contém código *JavaScript*, esse código por sua vez pode trazer informações dessa ou de outras páginas de maneira independente. Se, além disso, fizermos com que esse código

permaneça em execução respondendo a eventos, teremos nas mãos a possibilidade de trazer informação para o navegador sem carregar a página.

Isso é útil para algumas tarefas, mas não muito, já que faltam algumas peças em nosso quebra-cabeças. A primeira peça é a adoção dessa biblioteca por quase todos os navegadores, para que o código possa ser de aplicação universal. Além disso, o resultado é que podemos modificar o conteúdo da página em tempo real, usando a chamada *árvore DOM*. Como se isso não fosse suficiente, quando o AJAX foi definido, os programadores começaram a usar protocolos XML para se comunicarem com servidores.

E o que isso quer dizer? Que agora, com o AJAX, podemos carregar uma página e, sem ter que recarregá-la, obter

Os problemas com o IE

O *Internet Explorer*, apesar de ter sido o primeiro a introduzir o *XMLHttpRequest*, é o que mais apresenta problemas de uso. O código aqui mostrado nem sequer funciona no IE, devido ao fato de ele utilizar um componente *ActiveX* para estabelecer a conexão.

Existem diversas técnicas para permitir a compatibilidade entre navegadores, mas devido à limitação de páginas deste artigo e sua complexidade, não iremos mostrá-las. O leitor que estiver interessado na compatibilidade pode estudar o código de sistemas de código livre que implementam AJAX, como é o caso do *Sarissa* [3].

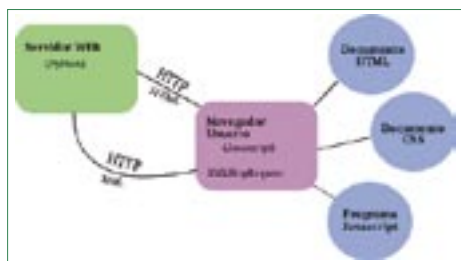


Figura 1 Esquema da nossa aplicação AJAX com todos os seus componentes.

informação, modificar a página em tempo real e interagir com servidores remotos usando protocolos XML.

Basicamente, uma vez carregada a página Web, temos em mãos todas as possibilidades de programação de uma GUI tradicional. E tudo isso sem a necessidade de plugins nem instalações – toda essa tecnologia está em nossos computadores, esperando para ser usada.

E onde entra o Python?

Vamos construir um pequeno servidor de conteúdo usando *Python*, que possa ser consultado usando-se AJAX. Criaremos uma página Web com um pouco de código JavaScript, que acessará nosso servidor Python em intervalos pré-definidos e modificará o aspecto da página Web.

Os cinco ingredientes

Os cinco ingredientes necessários para elaborar nosso produto são CSS, JavaScript, HTML, XML e Python, conforme mostrado na **figura 1**. Cada um deles terá uma função específica em nosso projeto.

O HTML é a base sobre a qual iremos trabalhar; definiremos uma página Web onde tudo vai acontecer. Na verdade, com o passar do tempo, o próprio HTML aca-

bou se convertendo em uma espécie de palmilha na qual o CSS e o JavaScript se moldam confortavelmente. O CSS nos permite outorgar propriedades visuais aos elementos do HTML. O JavaScript é o encarregado de atuar na máquina cliente, mais especificamente no navegador, e pode modificar tanto o HTML como as propriedades visuais que o CSS define. Com a chamada *XMLHttpRequest*, são disparadas as suas atribuições. Agora o HTML é visto como uma linguagem de programação de pleno direito. Nos próximos anos, pode ser que adquira muito mais importância do que tem tido até agora.

O XML é a nova linguagem de intercâmbio de informações. Praticamente qualquer linguagem já dispõe de bibliotecas para gerar e analisar documentos XML. Dentro do círculo de profissionais Web, o AJAX tornou-se o padrão para o intercâmbio de informações com o servi-

Exemplo 1: python.py

```

01 #!/usr/local/bin/python
02
03 import BaseHTTPServer
04 import os
05 import cgi
06
07 class AJAXHTTPRequestHandler (BaseHTTPServer.BaseHTTPRequestHandler):
08 """
09 Responde a requisições HTTP
10 """
11 def do_GET(self):
12     "Gerencia os GET"
13     acoes = {
14         "/" : ["envia_arquivo","index.html"],
15         "/ps.xml" : ["envia_comando", "ps afx"],
16         "/df.xml": ["envia_comando", "df"],
17         "/who.xml": ["envia_comando","who"],
18         "/uname.xml": ["envia_comando","uname -a"]
19     }
20     if self.path in acoes.keys():
21         acao = acoes[self.path]
22         (getattr(self,acao[0]))(self.path,acao[1])
23     else:
24         if (self.path[-3:] == ".js" or
25             self.path[-4:] == ".css"):
26             self.envia_arquivo("",self.path[1:])
27         else:
28             self.envia_arquivo("", "404.html")
29     def envia_arquivo(self,caminho,arquivo):
30         # Não usamos caminho, mas assim simplificamos o código
31         p = Pagina(arquivo)
32         self.enviar_resposta(p.tipo(), p.conteudo())
33
34     def envia_comando(self, caminho, comando):
35         c = Comando(comando)
36         self.enviar_resposta(c.tipo(), c.conteudo())
37     def enviar_resposta(self, tipo, conteudo):
38         self.enviar_cabecalho(tipo)
39         self.wfile.write(conteudo)
40
41     def enviar_cabecalho(self, tipo):
42         self.send_response(200)
43         self.send_header("Content-type", "text/" + tipo)
44         self.end_headers()
45     class Pagina:
46         def __init__(self,nome):
47             self.nome = nome
48             self.texto = ""
49             arquivo = file(self.nome)
50             self.texto = arquivo.read()
51             arquivo.close()
52         def conteudo(self):
53             return self.texto
54         def tipo(self):
55             tipo = "html"
56             ext = self.nome[-4:]
57             if (ext == "html"):
58                 tipo = "html"
59             elif (ext == ".xml"):
60                 tipo = "xml"
61             elif (ext == ".css"):
62                 tipo = "css"
63             return tipo
64     class Comando:
65         def __init__(self,comando):
66             self.lixo = os.popen(comando)
67             self.xml = ""
68
69         def conteudo(self):
70             # arquivo XML
71             if not self.xml:
72                 self.xml = "<?xml version='1.0\' ?>"
73                 self.xml += "<saida>"
74                 linha = self.lixo.readline()[:-1] # para eliminar \n
75                 while linha:
76                     self.xml += "<linha>" + cgi.escape(linha) + "</linha>"
77                     linha = self.lixo.readline()[:-1]
78                 self.xml += "</saida>"
79             return self.xml
80         def tipo(self):
81             return "xml"
82     def test(HandlerClass = AJAXHTTPRequestHandler, ServerClass =
83             BaseHTTPServer.HTTPServer):
84         BaseHTTPServer.test(HandlerClass, ServerClass)
85     if __name__ == '__main__':
86         test()

```

dor e para a serialização de objetos com protocolos como JSON.

É, como não poderia deixar de ser, o Python. Em nosso caso, ele se encarregará tanto da realização das tarefas de servidor HTTP quanto de recolher informação importante e usá-la para gerar arquivos XML.

Teremos que reunir todos esses elementos para realizar nosso projeto. O objetivo é que os componentes HTML, JavaScript, CSS e XML sejam os mais estáticos possíveis, já que todos eles devem interagir com o usuário.

É no servidor Python que devemos depositar a flexibilidade necessária, como adicionar novas características sem ter que mudar nenhum dos outros elementos.

A parte do Python: BaseHTTPRequest

O Python dispõe, em suas bibliotecas padrão, de muitos “esqueletos” para diferentes tipos de servidores. Entre elas encontramos a `BaseHttpRequest`. Essa classe nos permite construir servidores HTTP sem muito esforço, e por isso vamos empregá-la.

Mas não devemos usá-la diretamente, a não ser através da classe `BaseHttpRequestHandler`, que é a encarregada de gerenciar os eventos que acontecem no servidor. Portanto, partiremos dela e criaremos uma classe chamada `AJAXHttpRequestHandler` – ver **exemplo 1** (todos os exemplos deste artigo podem ser baixados em [4], em sua versão original em espanhol).

Um servidor HTTP recebe diferentes tipos de comandos, mas o que nos interessa é o comando GET. Ele é usado para solicitar informação ao servidor. Cada vez que uma solicitação GET é realizada, o método `do_GET` de `BaseHttpRequest` é invocado. Por isso vamos redefini-lo em nossa classe.

Ao se invocar `do_GET` na variável de instância `self.path`, é encontrada a rota solicitada pelo cliente. Iremos confrontar essa rota com as que aceitamos. Caso não se encontre entre elas, respondemos com a célebre página 404, indicando que a página solicitada não existe. A informação é devolvida usando-se o método `self.wfile.write()`; o que for escrito nele será devolvido ao cliente que fez a solicitação.

Além do arquivo `index.html`, oferecemos uma série de serviços em forma de arquivos XML. Esses serviços se basearão na execução de um comando de sistema, e a conversão de sua saída em um arquivo XML. O formato do arquivo será muito simples:

```
<?xml version="1.0" ?>
<saida>
<linha>linha de saida</linha>
...
<linha>linha de saida</linha>
...
<linha>linha de saida</linha>
</saida>
```

Por isso iremos gerar o arquivo XML à mão, sem fazer uso de bibliotecas. Dessa maneira, quando o cliente solicitar o arquivo `ps.xml`, nosso servidor executará o comando `ps afx`, criará o arquivo `ps.xml` com a saída do comando e o enviará ao cliente.

Definição de serviços

Para permitir que a definição de serviços seja a mais simples possível, basta introduzir uma nova entrada no dicionário `acoes` da classe `AJAXHttpRequestHandler`, com a seguinte estrutura:

```
<caminho> : [<metodo_a_invocar>
->.<comando_a_executar>],
```

Quando se solicita o comando HTTP GET, o caminho é buscado nesse dicionário. Caso ele esteja presente, será executado o método armazenado, usando como parâmetro o caminho e o comando. Isso nos dá uma grande flexibilidade; adicionar um novo serviço consiste em introduzir uma nova linha de código.

Há um detalhe importante: todo arquivo devolvido usando HTTP deve ter um cabeçalho com uma série de linhas com

Exemplo 2: arquivo index.html

```
01 <html>
02 <head>
03 <title>Testes com AJAX</title>
04 <link rel="stylesheet" href="estilo.css" type="text/css" />
05 <script language="JavaScript"
06 src="ajax.js">
07 </script>
08 </head>
09 <body>
10 <div id="documento">
11 <h3 id="titulo">Informação do sistema</h3>
12 <input type="button" name="button" value="Processos"
13 onclick="JavaScript:Requisita('ps.xml');" />
14 <input type="button" name="button" value="Disco"
15 onclick="JavaScript:Requisita('df.xml');" />
16 <input type="button" name="button" value="Usuários"
17 onclick="JavaScript:Requisita('who.xml');" />
18 <input type="button" name="button" value="Máquina"
19 onclick="JavaScript:Requisita('uname.xml');" />
20 <div id="contêiner">
21 <div id="dados"></div>
22 </div>
23 </div>
24 </body>
25 </html>
```

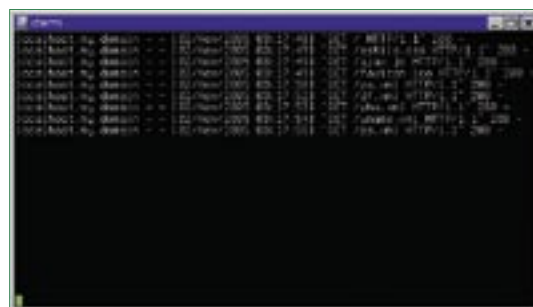


Figura 2 A classe `BaseHttpRequest` gera uma entrada por comando.

formato `chave: valor`, que dá informação ao cliente – o navegador – sobre o arquivo devolvido. Devido a problemas de espaço, decidimos devolver apenas o formato do arquivo. Para isso, invocamos o método `envia_cabecalho` a partir de `envia_resposta`, antes de enviar o arquivo em si. É dessa maneira que o navegador determina o tipo de arquivo que vai receber e suas características.

Ao iniciarmos o servidor, veremos que vão aparecendo mensagens correspondentes aos diferentes comandos enviados. A classe `BaseHttpRequest` gera uma entrada para cada uma (ver **figura 2**).

Gestores de serviços

Para administrar os serviços, foram criadas as classes `Página` e `Comando`, que respondem aos mesmos métodos. A primeira se encarrega dos pedidos de arquivos de texto, que em nosso programa resumem-se ao arquivo `index.html` e seus arquivos suplementares,

Exemplo 3: ajax.js.

```

01 // GLOBAIS
02 var http_request = false;
03
04 function Requisita(url) {
05     http_request = false;
06     http_request = new XMLHttpRequest();
07     if (http_request.overrideMimeType) {
08         http_request.overrideMimeType('text/xml');
09     }
10
11     if (!http_request) {
12         alert('Erro criando a instancia de XMLHttpRequest.');
```

arquivo JavaScript e CSS. Basicamente, eles são carregados em uma variável e, mediante o método `conteudo`, as demais classes têm acesso aos mesmos. Nesse caso, o parâmetro `caminho` não afeta essa classe, mas foi definido para que se preserve a compatibilidade com a classe `Comando`.

A classe `Comando` executa um comando especificado e permite o acesso ao texto devolvido por esse comando, através do mesmo método que a classe `Página`. Dessa maneira, as duas classes são intercambiáveis.

Ambas as classes possuem um método `tipo()`, que devolve o tipo de arquivo que armazenam. No caso de `Comando`, sempre será um arquivo XML, mas `Página` deve “adivinhar” o tipo dos arquivos que devolve. Para isso, ela examina a extensão do arquivo. Para manter a simplicidade, consideramos três extensões.

Quando um comando é executado por `Comando`, tem-se um cuidado especial em utilizar a função `cgi.escape` em cada linha. Essa função realiza algumas conversões dentro do texto que lhe é passado, para que possa ser visualizado corretamente pelo navegador web. O problema se agrava porque certos caracteres, tais como “<” ou “(aspas)” são especiais e, se não forem omitidos, ou forem precedidos de um “\”, causarão problemas.

Com isso, conseguimos definir um servidor web básico. O Python nos permite realizar tarefas complexas com pouco código, e esse é um desses casos. Agora vamos colocar o AJAX para compreendê-lo e interpretá-lo.

HTML

Talvez este seja um dos artigos onde o Python tenha o papel menos importante. Mas com cinco outros atores de peso, a situação tende a ser complicada. Vejamos o HTML. O **exemplo 2** mostra a página `index.html`. Basicamente, o código carrega um arquivo JavaScript, um arquivo CSS e mostra um título junto de alguns botões, os quais invocam ações em JavaScript.

Devemos nos concentrar especialmente no uso do atributo `id` em várias etiquetas HTML. Graças a esses `id`, poderemos manipulá-las através do JavaScript.

JavaScript por outra perspectiva

Muita gente tem uma má impressão do JavaScript. Essa linguagem é rara e, sem exagero, não muito útil. Ela permite modificar cores em páginas Web ou inserir faixas sem muito efeito. Isso para não falar das famosas janelas *pop-up*.

Isso fez com que ela ganhasse má fama, tanto que quase todos nós temos restrições em nossos navegadores em relação às ações que o JavaScript pode ou não realizar. O mais comum é que tenhamos um desses famosos bloqueadores de pop-ups. Mas o JavaScript se reintegrou à sociedade dos programadores pela porta da frente graças a um único objeto. Estamos nos referindo ao famoso `XMLHttpRequest`.

No código do **exemplo 3**, vemos uma linguagem que talvez nos lembre Java. Na verdade, não tem nada a ver, exceto pelo nome. Em nosso exemplo, vemos três funções:

- ◆ `Requisita()`
- ◆ `modificaConteudo()`
- ◆ `esvaziaConteudo()`

O JavaScript, além de muitas das características presentes em outras linguagens (e algumas ausentes), dispõe de uma coleção de objetos que lhe permitem realizar operações. Hoje em dia, as mais importantes são:

- ◆ Manipulação DOM
- ◆ Manipulação XML
- ◆ XMLHttpRequest

O DOM permite ao JavaScript manipular, em tempo real, o conteúdo da página Web. Pode acrescentar, modificar ou finalizar etiquetas e atributos, através dos quais podemos operar sobre o documento de todas as formas possíveis. O JavaScript pode manipular um arquivo XML da mesma forma que o DOM faz.

E a grande novidade: o JavaScript pode realizar conexões **assíncronas** com o servidor. **Assíncronas** está destacado em negrito porque é aí que está a chave. Isso significa que podemos fazer conexões, baixar documentos XML do servidor e realizar operações DOM ou de qualquer outro tipo, quando quisermos!

O usuário carrega sua página Web e, uma vez carregada, sem que haja necessidade de recarregá-la, pode modificá-la a seu gosto com base na informação que pede ao servidor, a qualquer momento.

De volta às nossas funções, a função `Requisita()` recebe uma URL, cria o objeto `XMLHttpRequest` e, depois de alguns

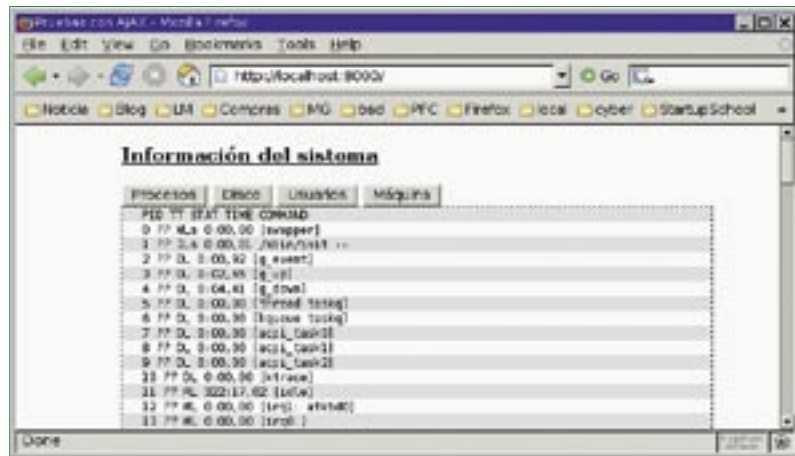


Figura 3 Nossa página mostra os resultados da consulta sem precisarmos recarregá-la.

testes, concede uma função para invocar quando for completamente descarregado o arquivo que essa URL especifica.

Isso significa que, enquanto lemos nossa Web, o JavaScript estará baixando um arquivo e, ao terminar, vai chamar a função `modificaConteudo`.

Mas o que essa função faz? Verifica o estado da solicitação (200 significa totalmente correto, e 404 é para “sentimos muito, mas o arquivo solicitado não está disponível”), e então obtém o documento XML do objeto que administra a conexão. Mas, antes, invoca `esvaziaConteudo()`, que localiza todas as etiquetas com as `ids linha0` ou `linha1` e as elimina da página. Fazemos isso porque em seguida voltaremos a introduzi-las na página, mas dessa vez com os dados frescos do servidor. Não queremos entrar em detalhes, já que, em teoria, este é um artigo sobre Python e não JavaScript, mas basicamente é isso que faz o arquivo `ajax.js`.

O arquivo `estilo.c`, que aparece no **exemplo 4**, simplesmente configura as cores e características de algumas das etiquetas, para que o aspecto fique melhor. Pronto, aqui temos nossa aplicação AJAX.

A **figura 3** mostra o resultado final. Quando pressionarmos qualquer um dos botões, será carregada a saída de texto da execução associada a cada um deles na tela, mas sem carregar a página. Se quisermos acrescentar um novo coman-

do, temos apenas que introduzir a linha correspondente em `acoes` em `server.py` e acrescentar um novo botão como os que já existem em `index.html`.

Conclusão

Esse negócio de AJAX é tão complicado assim? Mas é claro que não! Acontece que essa é uma palavra que está se transformando em um mito, mas não deixa de ser uma astuta combinação de programação no servidor e no cliente, além do uso intensivo de `HTMLHttpRequest`.

Pouco a pouco o AJAX está povoando todas as páginas Web e a maioria dos currículos. Quem sabe, dentro de dois anos essa palavra tenha tanto poder quanto outra de quatro letras: *JzEE*. ■

Exemplo 4: estilo.css

```
01 #documento{
02     margin-left: 100px;
03 }
04
05 #titulo{
06     text-decoration: underline;
07 }
08
09 #linha0 {
10     margin: 0px;
11     padding-left: 20px;
12     background: #e0e0e0;
13     font-family: monospace;
14 }
15
16 #linha1 {
17     margin: 0px;
18     padding-left: 20px;
19     font-family: monospace;
20 }
21
22 #container{
23     border-style: dashed;
24     border-width: 1px;
25     width: 600px;
26     border-color: black;
27 }
```

Mais Informações

- [1] AJAX, por Jesse James Garret: <http://www.adaptivepath.com/publications/essays/archives/000385.php>
- [2] AJAX na Wikipédia: <http://en.wikipedia.org/wiki/AJAX>
- [3] Sarissa: <http://sarissa.sourceforge.net/doc/>
- [4] Exemplos completos de código deste artigo, comentados: <http://www.linux-magazine.es/Magazine/Downloads/12>