AJAX Sem Mistérios, Uma Introdução ao Prototype

Anselmo Luiz Éden Battisti, Alexandre Semmer
Webgenium System
Cascavel - Paraná - Brasil
anselmobattisti@yahoo.com.br, alexandresemmer@hotmail.com

20 de julho de 2007

1 Prototype

A *Prototype* é uma ótima biblioteca Javascript escrita por Sam Stephenson. Esta coleção de funções foi muito bem escrita e bem pensada, utiliza técnicas de acordo com os padrões atuais e alivia o trabalho do programador na hora de produzir as páginas altamente interativas que caracterizam a chamada Web 2.0.

Na data em que esta apostila foi escrita a biblioteca está na versão 1.5.1.1. Ela pode ser baixada do web site http://Prototypejs.org/download, ela não possui nativamente uma versão compactada como muitas outras bibliotecas, por isto, sugerimos que a biblioteca seja compactada pois ela possui 97 Kb o que a torna pesada para algumas aplicações. O endereço http:bananascript.com/ oferece um ótimo compactador de arquivos Javascripts.

À medida que você ler os exemplos, caso você tenha familiaridade com a linguagem de programação Ruby, você notará, uma semelhança intencional entre as classes constituintes de Ruby e muitas das extensões implementadas pela *Prototype*.

1.1 Função $

A função de $() é equivalente ao método **document.getElementById()**, recebendo o ID de um elemento, ele o seleciona. Os objeto selecionado podemos executar alguns métodos da própria linguagem Javascript como por exemplo o value() ou da *Prototype* como hide(), show(), addClasseName(), etc.

Código 1: 'Função $'

```
1  <html>
2  <head>
3  <title> Test Page </title>
4  <script src="prototype.js"></script>
5  <script>
6     function test(){
7        var d = $('myDiv');
8        d.hide();
9        alert(d.innerHTML);
10       d.show();
```

```
11          d.addClassName('active');
12  }
13  </script>
14
15  <style type='text/css'>
16      .active {
17          color : #ff0000;
18      }
19  </style>
20
21  </head>
22  <body>
23
24      <div id="myDiv">
25          <p>This is a paragraph</p>
26      </div>
27
28      <div id="myOtherDiv">
29          <p>This is another paragraph</p>
30      </div>
31
32      <input type="button" value="Teste" onclick="test();"/><br/>
33
34  </body>
35  </html>
```

Código 2: 'Função $'

1.2 Função $$

A função $$() retorna uma coleção de elementos que pertençam a uma determinada classe CSS.

```
1   <html>
2   <head>
3       <script type="text/javascript" src="javascripts/prototype/prototype.js"></script>
4   <script>
5
6   function test$$(){
7
8           /*
9             Caso o CSS não seja o seu forte, a expressão abaixo quer
10            dizer 'ache todos os elementos INPUT que estão dentro
11            de um elemento cuja classe seja igual a field que esteja
12            dentro de um div cujo id seja loginForm'
13          */
14          var f = $$('div#loginForm .field input');
15
16          var s = '';
17
18          for(var i=0; i<f.length; i++){
19
20                  s += f[i].value + '/';
21
22          }
23
24          alert(s); // shows: "joedoe1/secret/"
25
26
27
28          //now passing more than one expression
29
```

```
30          f = $$('div#loginForm .field input', 'div#loginForm .fieldName');
31
32          s = '';
33
34          for(var i=0; i<f.length; i++){
35
36                  s += ( f[i].value ? f[i].value : f[i].innerHTML ) + '/';
37
38          }
39
40          alert(s); //shows: "joedoe1/secret/User name:/Password:/"
41
42  }
43  </script>
44
45
46  <div id='loginForm'>
47
48          <div class='field'>
49
50                  <span class='fieldName'>User name:</span>
51
52                  <input type='text' id='txtName' value='joedoe1'/>
53
54          </div>
55
56          <div class='field'>
57
58                  <span class='fieldName'>Password:</span>
59
60                  <input type='password' id='txtPass' value='secret' />
61
62          </div>
63
64          <input type='submit' value='login' onclick="test$$()"/>
65
66  </div>
67
68  <input type=button value='test $$()' onclick='test$$();' />
```

1.3 Função $A()

A função $A() cria um objeto *Enumerable* a partir de um argumento. Enumerable é um tipo de estrutura implementada pelo Prototype seguindo a linha de raciocínio da linguagem Ruby. Esta estrutura é muito poderosa e permite iterações simples sem a necessidade de laços *FOR*. Os enumerables baseiam seu funcionamento sobre o método *each*, neste método em cada iteração e lançado como parâmetro para a para a função o elemento atual da iteração.

Quando aplicamos a função $A() sobre vetores, esta função combinada com as extensões dos *enumerables*, torna mais fácil de converter ou copiar qualquer lista enumerável em um objeto de Ordem. Veja exemplo abaixo.

Código 3: 'Função $'

```
1  <script src="prototype.js"></script>
2  <script>
3     function showOptions(){
4        var someNodeList =             $('lstEmployees').getElementsByTagName('option');
5
```

```
6         var nodes = $A(someNodeList);
7         nodes.each(function(node){
8             alert(node.nodeName + ': ' + node.innerHTML);                    });
9     }
10 </script>
11
12
13 <select id="lstEmployees" size="10" >
14         <option value="5">Buchanan, Steven</option>
15         <option value="8">Callahan, Laura</option>
16         <option value="1">Davolio, Nancy</option>
17 </select>
18
19 <input type="button" value="Opções" onclick="showOptions();" />
```

Código 4: 'Função $'

1.4 Função $F

A função $F() devolve o valor de algum campo de um formulário recebendo como parâmetro o ID do elemento.

```
1 <script src="prototype.js"></script>
2 <script>
3     function test3(){
4         alert(  $F('userName')  );
5     }
6 </script>
7
8 <input type="text" id="userName" value="Joe Doe"><br/>
9 <input type="button" value="Test3" onclick="test3();">
```

2 HTML e DOM

HTML (acrônimo para a expressão inglesa HyperText Markup Language, que significa Linguagem de Marcação de Hipertexto) é uma linguagem de marcação utilizada para produzir páginas na Web. Documentos HTML podem ser interpretados por navegadores. A tecnologia é fruto do "casamento"dos padrões HyTime e SGML.

HyTime é um padrão para a representação estruturada de hipermédia e conteúdo baseado em tempo. Um documento é visto como um conjunto de eventos concorrentes dependentes de tempo (como áudio, vídeo, etc.), conectados por hiper-ligações. O padrão é independente de outros padrões de processamento de texto em geral.

SGML é um padrão de formatação de textos. Não foi desenvolvido para hipertexto, mas tornou-se conveniente para transformar documentos em hiper-objetos e para descrever as ligações.

Os documentos HTML podem ser manipulados através do seu DOM. O DOM ou Document Object Model (Modelo de Objeto de Documentos) possibilita que uma página web seja manipulada como um conjunto de nós (elementos da página), permitindo acesso direto a qualquer elemento da página por um *script* Javascript.

DOM representa os documentos HTML como uma hierarquia de nós que podem implementar interfaces especializadas. Por exemplo, o elemento TABLE está hierarquicamente acima dos elementos TR que está acima dos elementos TD de sua estrutura.

Com a API DOM podemos criar documentos, navegar pela estrutura de um documento qualquer, incluir, alterar e apagar nós do documento (o próprio nó ou seu conteúdo). Isso é válido tanto para páginas HTML quanto para documentos XML.

2.1 Easy DOM Creator

Quando se programa utilizando AJAX é muito comum a necessidade de criar objetos HTML em tempo de execução, a única saída é trabalhar com o DOM HTML através do Javascript. A manipulação do DOM utilizando Javascript é bastante trabalhosa sendo assim várias bibliotecas foram escritas para facilitar o uso do DOM, em geral o uso destas bibliotecas é aconselhável pois elas agrega funções não intrusivas.

O script Easy DOM Creator escrito por Michael Geary e pode ser baixado do seguinte endereço http://mg.to/2006/02/27/easy-dom-creation-for-jquery-and-Prototype , foi desenvolvido utilizando a biblioteca de Jquery, porém, funciona também muito bem com a Prototype tendo em vista que única função que o *script* usa da biblioteca é a função $ que funciona da mesma maneira em ambas as bibliotecas.

Veja um exemplo:

Código 5: 'Exemplo de uso do Easy Dom Creator'

```
1  var table = $.TABLE({Class:"MyTable", border: 1},
2     $.TBODY({},
3        $.TR({},
4           $.TD({}, 'Site' ),
5           $.TD({},'Link: ',
6                  $.A({ Class:'MyLink', href:'http://www.example.com'},'example.com'))
7               )
8           )
9  );
10
11 $('data').appendChild(table);
```

A variável table contém a tabela, o parâmetro $.TAG, onde TAG é o nome da TAG HTML indica a intenção de criar uma TAG do mesmo tipo. Ao final usa-se o *appendChild* para incluir o elemento criado em algum outro elemento já existente no DOM HTML, lembrando que o próprio elemento BODY pode ser utilizado como pai do elemento criado.

Para definir um evento em um elemento HTML criado dinamicamente podemos trabalhar de duas maneiras, a maneira tradicional é utilizar o método *setAttribute*, nele são passados o evento e a função que será realizada.

```
table.setAttribute('onClick',alert('teste');
```

A segunda maneira é utilizar o *observer* do *Prototype*, com ele podemos definir qual função será executada quando o evento acontecer sobre o elemento especificado.

```
Event.observe('ID', 'ACAO', FUNCAO);
```

Isto faz com que no momento em que o elemento com o ID realizar a ACAO a FUNCAO será disparada. A função pode ser criada em tempo de execução ou uma função já existente em outros Scripts. O nome da ação deve seguir as recomendações da W3C para o DOM Level 2 que pode ser encontrada no endereço http://www.w3.org/TR/DOM-Level-2-Events/events.html#Events-overview-terminology.

A função *observe* deve ser escrita após a criação do elemento que receberá a ação, para evitar isto podemos utilizar o seguinte script.

Exemplo :

```
Event.observe(window, 'load', function() {
   Event.observe('ID', 'ACAO', FUNCAO);
});
```

2.2 Orientação a Objeto utilizando o *Class*

A orientação a objetos é um recurso muito valioso, durante muito tempo foi restrito a sistemas Desktop. Para o desenvolvimento de sistemas Web o framework *Prototype* oferece o Class.create(), isto facilita a criação de classes utilizando o Javascript. Como conseqüência imediata do uso desta técnica temos uma significativa melhora na legibilidade do código fonte.

Código 6: 'Classes em Javascript'

```
1  <html>
2  <head>
3  <title> Test Page </title>
4  <script src="prototype.js"></script>
5  <script type='text/javascript'>
6      var Animal = Class.create();
7      Animal.prototype = {
8
9        initialize: function(name, sound) {
10         this.name  = name;
11         this.sound = sound;
12       },
13
14       speak: function() {
15         alert(this.name + " says: " + this.sound + "!");
16       }
17     };
18
19     var cobra = new Animal("Silvo", "Sheeeeeshee");
20     cobra.speak();
21
22     var gato= new Animal("Miado","Miauouo");
23     gato.speak();
24 </script>
25 </head>
26 <body>
27 </body>
```

No exemplo anterior foi criada a classe Animal, e dois objetos foram instanciados, o objeto COBRA e o objeto GATO, o método *initialize* é o método executado no momento da instanciação do objeto, seria comparado ao __*construct* do PHP.

3 AJAX

AJAX (acrónimo em língua inglesa de *Asyncronous Javascript And XML*) é o uso sistemático de *Javascript* e XML (e derivados) para tornar o navegador mais interativo com o usuário, utilizando-se de solicitações assíncronas de informações. AJAX não é somente um novo modelo, é também uma iniciativa na construção de aplicações *web* mais dinâmicas e criativas. AJAX não é uma tecnologia, são realmente várias tecnologias trabalhando juntas, cada uma fazendo sua parte, oferecendo novas funcionalidades. AJAX incorpora em seu modelo:

- Apresentação baseada em padrões, usando XHTML e CSS;
- Exposição e interação dinâmica usando o DOM;
- Intercâmbio e manipulação de dados usando XML e XSLT e JSON;
- Recuperação assíncrona de dados usando o objeto XMLHttpRequest;
- *Javascript* unindo todas elas em conjunto.

O modelo clássico de aplicação Web trabalha assim: A maioria das ações do usuário na interface dispara uma solicitação HTTP para o servidor Web. O servidor processa algo recuperando dados, mastigando números, conversando com vários sistemas legados e então retorna uma página HTML para o cliente. É um modelo adaptado do uso original da Web como um agente de hipertexto, porém o que faz a Web boa para hipertexto não necessariamente faz ela boa para aplicações de software.

Esta aproximação possui muito dos sentidos técnicos, mas não faz tudo que um usuário experiente poderia fazer. Enquanto o servidor está fazendo seu trabalho, o que o usuário estará fazendo? O que é certo, esperando. E a cada etapa em uma tarefa, o usuário aguarda mais uma vez.

Obviamente, se nós estivéssemos projetando a Web do nada para aplicações, não faríamos com que os usuários esperassem em vão. Uma vez que a interface está carregada, por que a interação do usuário deveria parar a cada vez que a aplicação precisasse de algo do servidor? Na realidade, por que o usuário deveria ver a aplicação ir ao servidor toda vez?

A maior vantagem das aplicações AJAX é que elas rodam no próprio navegador Web. Então, para estar hábil a executar aplicações AJAX, bastar possuir algum dos navegadores modernos, ou seja, lançados após 2001, são eles: Mozilla Firefox, Internet Explorer 5+, Opera, Konqueror e Safari.

3.1 Política de Origem

Para garantir a segurança dos usuários, os navegadores implementam a política de origem, nela não são permitidas que dados sejam enviados ou recebidos de um site diferente daquele que está sendo exibido pelo navegador.

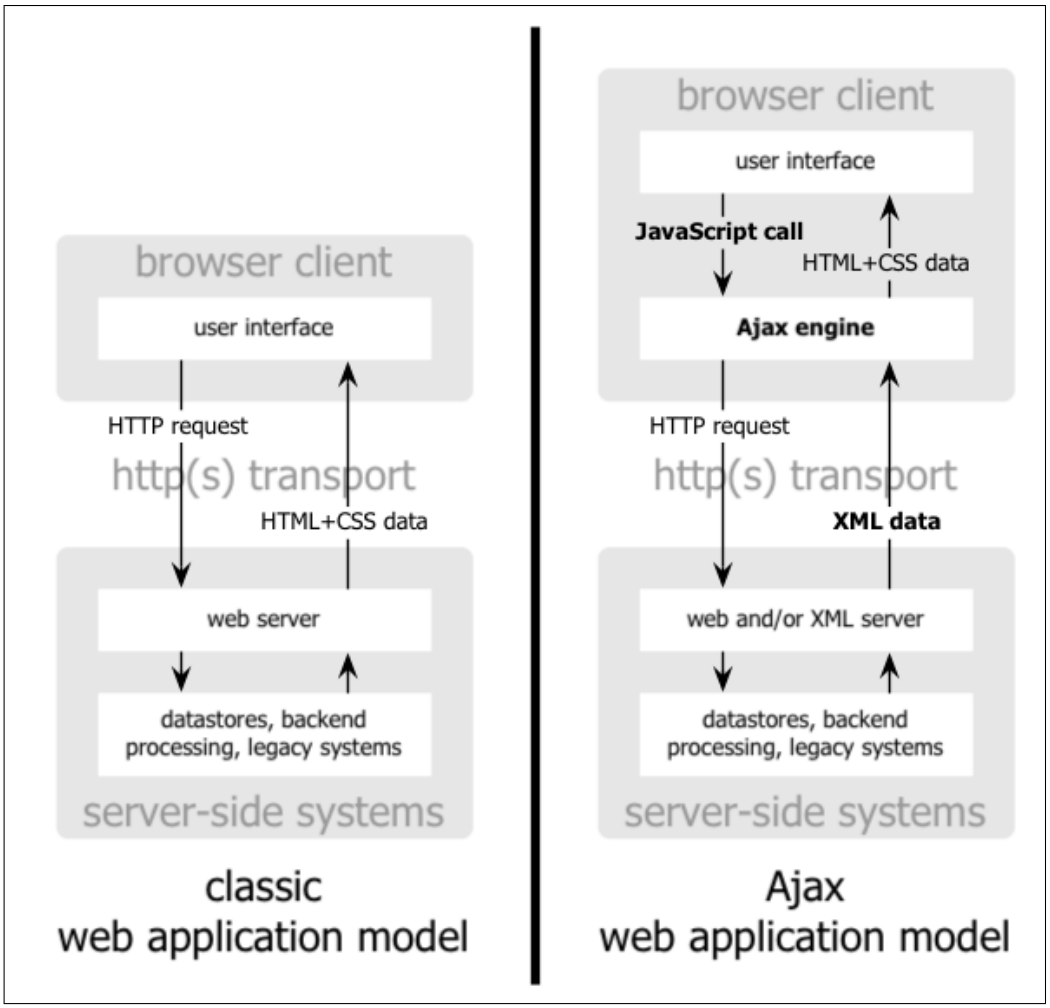

Figura 1: Diagrama de requisições ao servidor

A política de origem segue 3 parâmetros, protocolo, URL e a porta, caso algum destes parâmetros diferir dos da página atualmente exibido pelo navegador, então ocorrerá uma exceção e a chamada em AJAX é abortada.

Infelizmente esta política não é adotada por todos os navegadores, em particular temos o Internet Explorer da Microsoft que adota o conceito de região segura.

3.2 A Classe Assíncrona

Para se fazer um pedido HTTP ao servidor usando Javascript, você precisa de um objeto que disponibiliza essa funcionalidade. Tal classe foi primeiro introduzida no Internet Explorer sob a forma de um objeto ActiveX chamado XMLHTTP, então, o Mozilla, o Safari e outros navegadores também criaram seus objetos de conexão assíncrona, o objeto foi o XMLHttpRequest que suportava os métodos e as propriedades do objeto ActiveX original da Microsoft.

O código abaixo mostra a forma genérica e tradicional de instanciar um objeto que proverá os mecanismos para pedidos assíncronos.

Código 7: 'ajaxInit.js'

```
1  function ajaxInit() {
2
3    var xmlhttp;
4
5    try {
6       xmlhttp = new XMLHttpRequest();
7    } catch(ee) {
8       try {
9          xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
10      } catch(e) {
11         try {
12            xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
13         } catch(E) {
14            xmlhttp = false;
15         }
16      }
17   }
18
19   return xmlhttp;
20 }
```

4 AJAX com o Objeto AJAX do Prototype

O *Prototype* oferece três objetos para a manipulação de conexões assíncronas, todos os 3 possuem métodos comuns o que facilita o seu aprendizado.

Os métodos e opções descritos a seguir podem ser utilizados em qualquer um dos 3 tipos de objetos de conexão que o *Prototype* oferece.

- **asynchronous :** Determina se a chamada será assíncrona ou não, o manual não aconselha o uso de chamadas síncronas pois isto pode causar efeitos colaterais como por exemplo *deadlocks*, por padrão *true*;
- **contentType :** Define o tipo do cabeçalho que será enviado do servidor para o Javascript, por padrão application/x-www-form-urlencoded;
- **encoding :** Codificação dos dados que serão enviados do cliente para o servidor, por padrão UTF-8;
- **method :** O método de envio dos dados, pode ser post ou get;
- **parameters :** Os parâmetros que serão passados para o servidor, eles devem estar codificados no formato GET porém se o método for POST o Prototype codifica e insere no cabeçalho HTML os parâmetros. Exemplo '?a=1&b=2'.

O Ciclo de vida da chamada em AJAX possui várias fases, para cada uma delas o *Prototype* disponibiliza métodos específicos para o seu tratamento e interceptação das suas mensagens de controle. Estes são os estágios que uma conexão em AJAX pode passar durante a sua vida.

- Created;
- Initialized;
- Request sent;
- Response being received (Pode ocorrer várias vezes, a cada novo pacote HTTP chegar.)
- Response received, request complete

Para que possamos interceptar mensagens em cada uma das etapas do ciclo de vida da requisição o *Prototype* oferece métodos padrões para realizar a interceptação dos dados.

- **onCreate :** É chamada no logo após os parâmetros e a url ser processada e antes do objeto AJAX ser invocado;
- **onComplete :** Assim que o objeto AJAX retorna os dados ao *script*;
- **onFailute :** É chamado antes do onComplete assim que um erro é detectado.

4.1 Ajax.Updater

O Ajax.Updater é o método de conexão AJAX que podemos utilizar quando os dados que vem do servidor já estarão formatados em HTML. Para a primeira atividade utilizando AJAX vamos desenvolver uma calculadora.

Dois os arquivos serão desenvolvidos, o arquivo calc.php que realizará a conta e calculadora.html que enviará os dados e exibirá o resultado.

Código 8: 'Calculadora HTML'

```
1  <html>
2  <head>
3  <title> Test Page </title>
4  <script src="prototype.js"></script>
5  <script type='text/javascript'>
6     function calcular() {
7        var myAjax = new Ajax.Updater('resultado','calculadora.php',
8        {
9           method: 'get',
10          parameters: $(calculadora).serialize()
11       });
12    }
13 </script>
14 </head>
15 <body>
16 <form name="calculadora" id="calculadora">
17    <input type=" text " name="v1" id="v1" size="3"/> +
18    <input type=" text " name="v2" id="v2" size="3"/> =
19    <label id="resultado"></label> <br/>
20    <input type="button" value="Calc" onClick="calcular()"/>
21 </form>
22 </body>
```

Código 9: 'Calculadora PHP'

```
<?
   echo $_GET['v1'] + $_GET['v2'];
?>
```

O Ajax.Updater necessita de 3 parâmetros, no primeiro passamos o ID do elemento que irá exibir o resultado, o arquivo PHP que irá realizar o processamento e os parâmetros que serão passados para o servidor.

4.2 Ajax.Request

O método Ajax.Request é o mais utilizado e ele possui uma série de funções que o tornam ideal em diversas situações.

Código 10: 'Calculadora HTML'

```
<html>
<head>
<title> Test Page </title>
<script src="prototype.js"></script>
<script type='text/javascript'>

   function moveis() {
      new Ajax.Request('moveis.php', {
         onSuccess: function(transport,json) {
            alert(json.moveis[0]);
         }
      });
   }

</script>
</head>
<body>
   <input type='button' value='moveis' onClick='moveis()'/>
</body>
```

Código 11: 'Calculadora PHP'

```
<?
    include_once("json.php");

    $json = new json();

    # vetor de móveis
    $a[] = "Cama";
    $a[] = "Mesa";
    $a[] = "Cadeira";

    $b['moveis'] = $a;
    header("X-JSON: ".$json->encode($b));
?>
```

4.2.1 JSON

JSON (com a mesma pronuncia do nome "Jason"em inglês), um acrônimo para *Javascript Object Notation*, é um formato leve para intercâmbio de dados computacionais. JSON é um subconjunto da notação de objeto de Javascript, mas seu uso não requer Javascript

exclusivamente, por ser um formato muito simples e eficiente ele foi portado para diversas outras linguagens como C, Python e Java.

A simplicidade de JSON tem resultado na difusão do seu uso, especialmente como uma alternativa para XML em AJAX. Uma das vantagens do JSON sobre XML como um formato para intercâmbio de dados neste contexto é o fato de ser muito mais fácil escrever um analisador JSON. Em Javascript uma *string* em JSON podem ser analisado trivialmente usando a função eval(). Isto foi importante para a aceitação de JSON dentro da comunidade AJAX devido a presença deste recurso de Javascript em todos os navegadores Web atuais, XML foi escrito para humanos ler JSON foi escrito para as máquinas entender.

Na prática, os argumentos a respeito da facilidade de desenvolvimento e performance do analisador são raramente relevados devido aos interesses de segurança no uso de eval() e a crescente integração de processamento XML nos navegadores *web* modernos. Por esta razão JSON é tipicamente usado em ambientes onde o tamanho do fluxo de dados entre o cliente e o servidor é de suma importância (daí seu uso por Google, Yahoo, etc., os quais servem milhares de usuários simultaneamente). O JSON deve ser trabalhado em ambientes onde a fonte dos dados pode ser explicitamente confiável, e onde a perda dos recursos de processamento XSLT no lado cliente para manipulação de dados ou geração da interface, não é uma consideração.

Enquanto JSON é freqüentemente posicionado "em confronto"com XML, não é incomum ver tanto JSON como XML sendo usados na mesma aplicação. Por exemplo, uma aplicação no lado cliente a qual integra dados do Google Maps com dados atmosféricos através de SOAP, requer suporte para ambos formatos de dados.

A figura 2 ilustra em forma de um autômato finito a gramática da linguagem JSON. A *string* abaixo mostra um retorno das cidades do estado do Paraná.

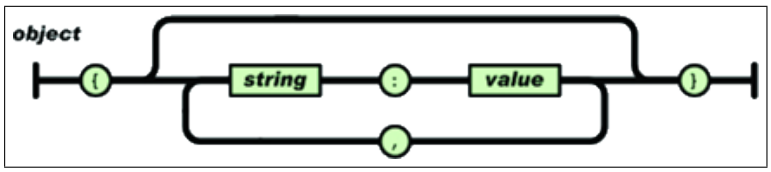

Figura 2: String básica Json

```
{
'cidades':[
{'cdg':'1','nm':'Tupãssi'},
{'cdg':'2','nm':'Toledo'},
{'cdg':'3','nm':'Cascavel'},
{'cdg':'4','nm':'Pato Branco'}
   ]
}
```

Usando a função *eval*[1] do *Javascript* podemos transformar esta *string* em uma estrutura de objetos facilmente manipuláveis.

Se usarmos a função *eval* no retorno das cidades poderiamos por exemplo acessar o código da cidade de Pato Branco através do seguinte comando:

```
json = eval("("+ajax.responseText+")");
json.cidades[3].cdg;
```

Na primeira linha é aplicada a função *eval* e seu resultado é armazenado na variável **Json**, na linha 2 é acessado o atributo cidades, que é um vetor, na posição 3 no atributo cdg.

Para facilitar o uso de JSON no PHP, a partir da versão 5.2.0 foi integrada a linguagem a função json_encode e o json_decode, estas duas funções transformam vetores em string JSON e decodificam string JSON em vetores PHP, caso você esteja trabalhando em um servidor que possua uma versão inferior do PHP então existem classes que fazer esta função.

4.2.2 Ajax.PeriodicalUpdater

Funciona da mesma forma como o Ajax.Request, a diferença é que este método permite que sejam feitas chamadas AJAX sem a intervenção do usuário, estas chamadas ocorrem em um determinado intervalo de segundos que é configurado. Este objeto é muito útil durante o desenvolvimento de aplicativos de chat ou mesmo em sistema gráficos de ações nas bolsas de valores.

Exemplo:

Código 12: 'Relógio PHP'

```
1  <script src="prototype.js"></script>
2  <script>
3        new Ajax.PeriodicalUpdater('data', 'data.php',
4        {
5           method: 'get',
6           frequency: 3
7        });
8  </script>
9
10 <div id='data'></div>
```

Código 13: 'Relógio PHP'

```
1  <?
2      echo time();
3  ?>
```

Isto irá fazer com que a cada 3 segundo o navegador chame o arquivo **data.php** e insira em seu conteúdo dentro do div cujo id é data.

[1] Introduzida na especificação 1.0 do *Javascript* esta função interpreta uma string como uma expressão, eval("fred=999; wilma=777; document.write(fred + wilma);") retorna 1776

5 Validação de Formulários Com AJAX

Quando desenvolvemos formulário em HTML temos que levar em conta vários fatores, um deles é a garantia que serão enviados para o banco de dados apenas dados válidos quanto a sua sintaxe, datas no formato correto, números onde devem ser números, campos obrigatórios preenchidos, em fim temos que nos certificar sobre a autenticidade dos dados.

Tradicionalmente a validação dos dados é feita usando Javascript, ela tem como vantagem a velocidade da validação, infelizmente ela pode ser facilmente eliminada apenas desabilitando o Javascript do navegador, ou ainda enviando os dados diretamente para a aplicação no servidor. A validação no lado do servidor por sua vez é mais segura pois o usuário não consegue burla-la, porém, ela é mais lenta e necessita de um *refresh* na página HTML. A solução então é utilizar o AJAX para fazer a validação dos dados do formulário.

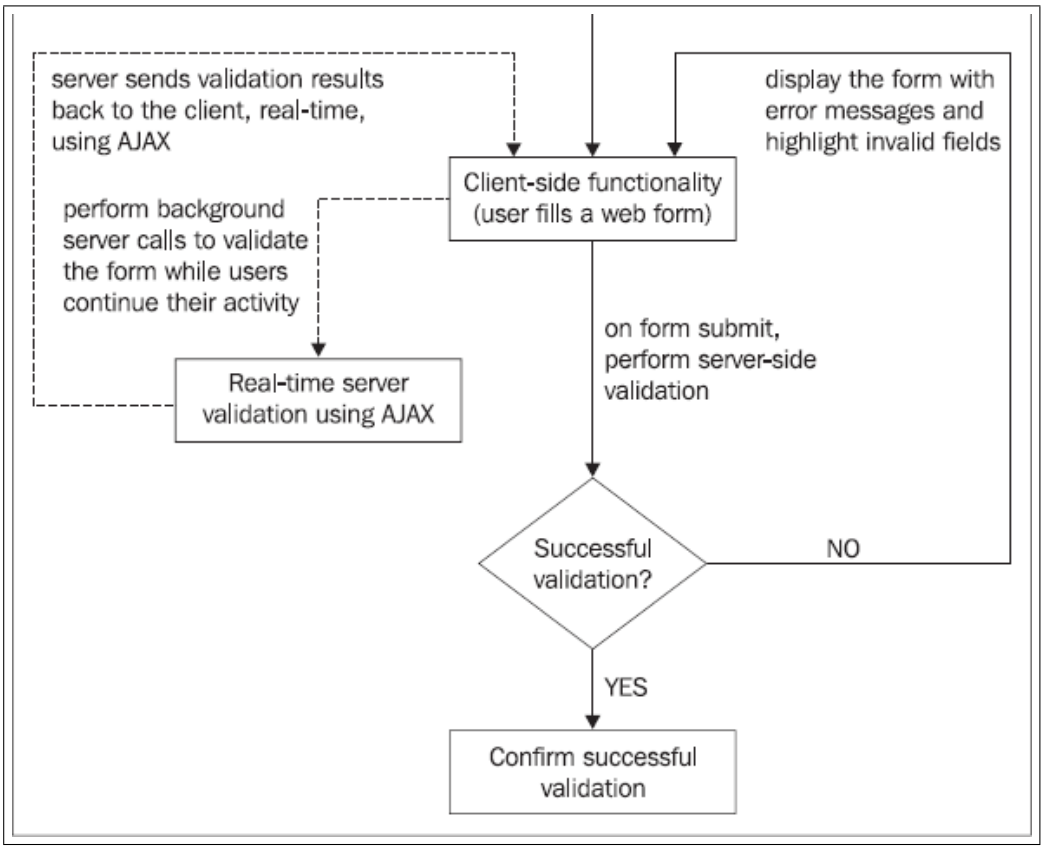

Figura 3: Validação de Formulários em Ajax

A figura 3 foi retirada do livro "AJAX and PHP", nela podemos ver claramente quais são as obrigações de cada um dos integrantes da validação, o navegador envia para o servidor os dados a serem validados, o servidor processa e retorna mensagem de erro caso haja algum problema e mensagem de sucesso caso não hajam problemas, e o navegador fica encarregado de exibir o resultado.

Código 14: 'Conexão com a Base de Dados'

```
// verificar se os dados que serão enviados ao servidor são válidos
validar:function(){
    new Ajax.Request(
        "ajax/validacao.php", {
            parameters : $('frmFuncionario').serialize(),
            method     : 'post',
            onSuccess  : function (t, json) {

                // coloca o estilo dos campos como normal
                $('nome').className    = "inputnormal";
                $('salario').className = "inputnormal";

                var msgErro = "";

                // mudar a cor dos campos que estão com problema
                for (i = 0 ; i < json.campo.length; i++) {
                    $(json.campo[i]).className = 'inputerro';
                }

                // mensagens de erro
                for (i = 0 ; i < json.erro.length; i++) {
                    msgErro += json.erro[i]+" <br />";
                }

                $('mensagem').className = "erro";
                $('mensagem').innerHTML = msgErro;
            }
        }
    )
},
```

6 Estudo de Caso

Para o final desta apostila iremos criar uma mini aplicação utilizando AJAX. Ela se consistem em um formulário e duas tabelas em uma base de dados, 4

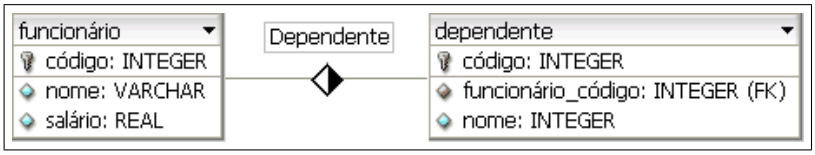

Figura 4: Base de Dados

A primeira coisa que vamos fazer é determinar a forma como trabalharemos com a base de dados, neste caso uma base Mysql acessada através da biblioteca de conexão nativa PDO.

Código 15: 'Conexão com a Base de Dados'

```
<?php
class conecta
{
    private function __construct(){}

    /**
```

```
7       * getConexao
8       *
9       * @abstract
10      * Retorna um objeto de conexão com o banco de daddos
11      **/
12      public static function getConexao() {
13          try {
14              $pdo = new PDO("mysql:host=localhost; dbname=anselmo", "anselmo", "321");
15              return $pdo;
16          } catch (Exception $e) {
17              die($e." erro ");
18          }
19      }
20  }
21  ?>
```

Um recurso muito importante no desenvolvimento de aplicativos em Ajax é a abstração de vetores em PHP atravez de nome padronizados.

Código 16: 'Conexão com a Base de Dados'

```
1   // adicionar um dependente ao funcionário
2   addDependente : function () {
3       var dep =
4           $.LABEL({},"Nome : ",
5           $.INPUT({
6               type : "text",
7               name : "nomedep[]"
8           }),$.BR({}));
9       $('dependentes').appendChild(dep);
10  },
```

Em nosso exemplo podemos ter vários dependentes para o mesmo funcionário, cada *input* deveria ter um nome único, porém isto é impraticável pois não sabemos sua quantidade, sendo assim podemos declarar seu nome como sendo nomedep[], desta forma ao ser recebido pelo PHP os dependentes 'automagicamente' se comportarão como um vetor.

7 Conclusão

AJAX é uma realidade que não deve ser utilizada em sistemas Web e Web sites de maneira indiscriminada, esta técnica aumenta a interação do usuário com o sistema pois elimina muitos *refresh* desnecessários.

Este curso não tem a pretenção de ser uma referência sobre AJAX, ele é um pontapé inicial para que você consiga iniciar no maravilho mundo do AJAX.

7.1 Para Saber Mais

- Artigo muito bom sobre AJAX
  http://adaptivepath.com/publications/essays/archives/000385.php

- Usando POST com AJAX
  http://www.dotpix.com.br/ wendel/projetos/ajax/postXMLHttpRequest/sumGet.html

- Não use innerHTML
  http://slayeroffice.com/articles/innerHTML_alternatives/#intro

- Página oficial do JSON
  http://www.json.org/

- DHTML
  http://www.quirksmode.org/js/cross_dhtml.html

- Exemplos em Ajax
  http://www.japs.etc.br/ajax/

- Material sobre Css
  http://www.maujor.com

- DOM XHTML e HTML
  http://www.amtechs.com/w3c/introduction.html