FRAMEWORK - CELEPAR

PADRÃO DE UTILIZAÇÃO DE AJAX - Asynchronous JavaScript and XML

Fevereiro – 2006

Sumário de Informações do Documento

Tipo do Documento: Relatório		
Título do Documento: PADRÃO DE UTILIZAÇÃO DE AJAX - Asynchronous JavaScript and XML		
Estado do Documento: Elaborado		
Responsáveis: Natasha Krassuski Fortes		
Palavras-Chaves: ajax, javascript, XML,DHTML		
Resumo:		
Número de páginas:		
Software utilizados:		
Versão	**Data**	**Mudanças**
1.0	22/02/2006	

SUMÁRIO

1.INTRODUÇÃO..... 5

1.1.Objetivo..... 5
1.2.Ajax..... 5
1.3.DHTML..... 5
1.4.Prototype..... 6
1.5.Venkman..... 6
1.6.Arquitetura do Framework com AJAX..... 6

2.UTILIZAÇÃO..... 7

2.1.Exemplo..... 7
2.2.Material de Apoio..... 9

3 PRÓS E CONTRAS DO AJAX..... 10

3.1.Prós..... 10
3.2.Contras..... 10

4.ANEXO..... 11

PROTOTYPE.JS..... 11
As funções utilitárias..... 11
Utilizando a função $()..... 11
Usando a função $F()..... 12
Usando a função $A()..... 13
Usando a função $H()..... 14
Usando a função $R()..... 14
Utilizando a função Try.these()..... 15
O objeto Ajax..... 15
Utilizando a classe Ajax.Request..... 15
Utilizando a classe Ajax.Updater..... 19
Referência da prototype.js..... 20
Extensões das classes JavaScript..... 20
Extensões da classe Object..... 21
Extensões da classe Number..... 21
Extensões da classe Function..... 22
Extensões da classe String..... 23
Extensões da classe Array..... 23
Extensões do objeto document do DOM..... 24
Extensões do objeto Event..... 25
Novos objetos e classes definidos em prototype.js..... 26
The PeriodicalExecuter object..... 26
The Prototype object..... 27
The Class object..... 27
The Ajax object..... 28
The Ajax.Base..... 28
The Ajax.Request..... 28
The options argument object..... 29
The Ajax.Updater..... 30
The Ajax.PeriodicalUpdater..... 31
The Element object..... 32
The Abstract object..... 33
The Insertion object..... 34

The Insertion.Before..... 34
The Insertion.Top.....34
The Insertion.Bottom..... 35
The Insertion.After.....36
The Field object..... 36
The Form object.....37
The Form.Element object.....37
The Form.Element.Serializers object.....38
The Abstract.TimedObserver.....38
The Form.Element.Observer.....39
The Form.Observer..... 39
The Abstract.EventObserver.....40
The Form.Element.EventObserver..... 41
The Form.EventObserver.....41
The Position object (preliminary documentation).....42

1.INTRODUÇÃO

1.1.Objetivo

O objetivo desse documento é fornecer as definições de padrões para utilização da tecnologia Ajax na arquitetura do Framework Pinhão. Esclarece também os prós e os contras existentes na utilização desse recurso. Com a utilização do Ajax de forma padronizada, tem-se a manutenção dos sistemas que utilizam o Framework Pinhão facilitada.

1.2.Ajax

Asynchronous JavaScript and XML, ou AJAX consiste em um método para se comunicar com um servidor web sem a necessidade da página ser recarregada. Em boa parte do tempo são utilizadas as mesmas técnicas da programação DHTML, o que difere a tecnologia AJAX da DHTML é o uso do objeto XMLHttpRequest. Este objeto permite a um código JavaScript enviar e receber dados de uma resposta de um servidor sem a necessidade de recarregar todo o conteúdo da página web. O termo Asynchronous (assíncrono) da definição de AJAX vem justamente disso, porque os dados podem ser carregados não necessariamente ao mesmo tempo em que a página é carregada, ou seja, não precisa haver sincronia do carregamento de dados com o carregamento da página web.

1.3.DHTML

O DHTML é a combinação de várias facilidades postas à disposição pela última geração de browsers, que com a conjugação da JavaScript/HTML, possibilita a construção de páginas web mais dinâmicas .

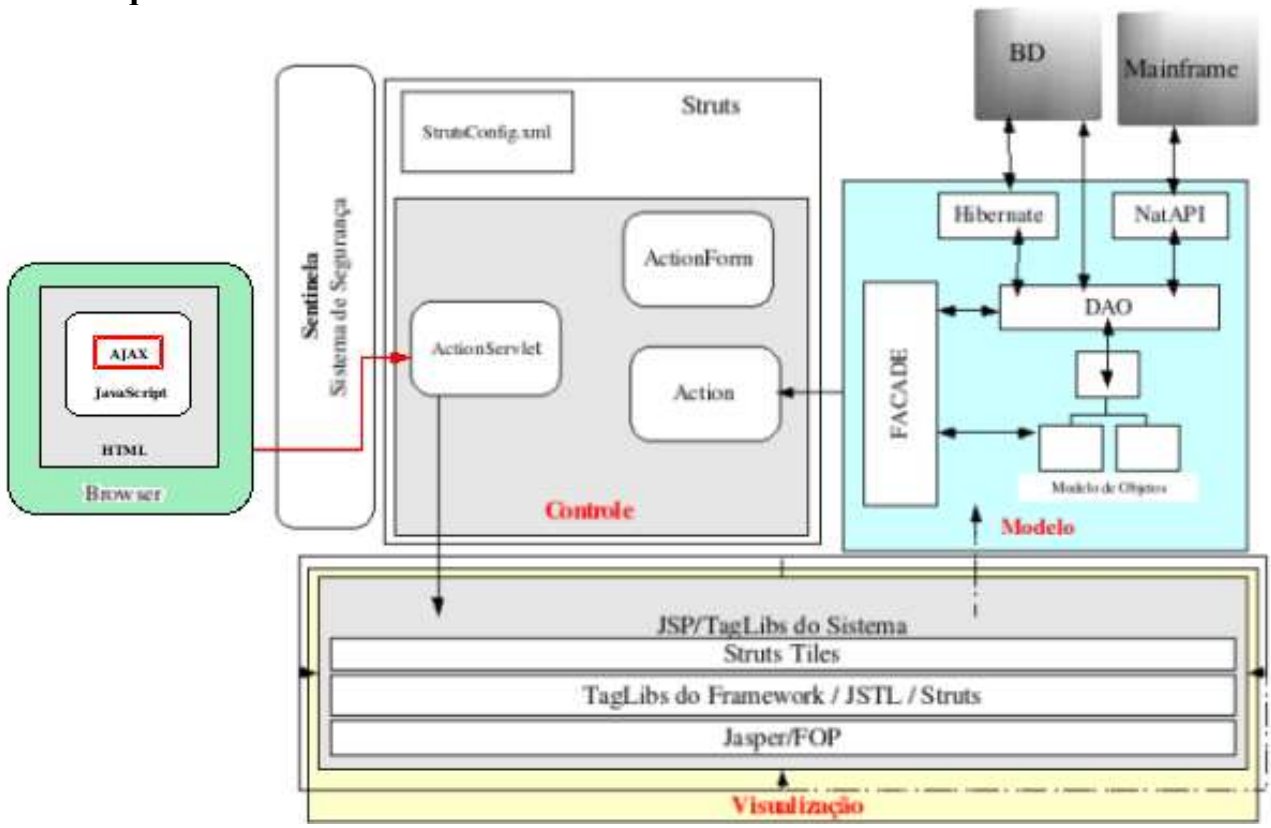
1.4.Prototype

Prototype é a biblioteca JavaScript/Ajax adotada como padrão do Framework Pinhão. Possui uma vasta coleção de funções que utilizam técnicas de acordo com os padrões atuais e reduzem o trabalho na hora de produzir as páginas altamente interativas.

1.5.Venkman

Venkman é um excelente projeto que cumpre o papel de JavaScript/Ajax Debugger. Porém, funciona apenas para browsers da família Mozilla. A página desse projeto é http://www/mozilla.org/projects/venkman/ .

1.6.Arquitetura do Framework com AJAX



2.UTILIZAÇÃO

2.1.Exemplo

Para poder exemplificar a utilização do Ajax dentro do Framework Pinhão foi utilizado um exemplo de aplicação de uma escola. Esse projeto trata de um cadastro de alunos de uma escola, possuindo dentro dele uma tela de pesquisa de alunos. A utilização de Ajax será exemplificada com base nessa tela de pesquisa contida no arquivo list_alunos.jsp.

Inicialmente deve-se ter a biblioteca javascript Prototype declarada no jsp que utilizará Ajax. A seguir na figura1 temos um exemplo de inclusão e configuração dessa biblioteca e de links :

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:url var="js_prototype" value="/js/prototype.js" />
<script language="JavaScript" type="text/JavaScript" src="${js_prototype}"></script>
<c:url var='link_ajax' value='/matricula.do' />
```

Figura 1 – Bibliotecas e links

Na figura 2 temos a implementação da função 'listar' do arquivo list_alunos.jsp. A implementação dessa função contém uma variável que recebe os parâmetros a serem requisitados para o servidor poder efetuar a busca. Nessa variável é definida qual a action irá tratar do processamento. Nela também pode-se observar a presença da função “$F()”, que retorna o valor de qualquer campo de um formulário (input control), tais como caixas de texto ou listas, substituindo o “document.getElementById()”

Na próxima linha temos a execução do Ajax propriamente dito, na qual recebe o nome do componente html div que irá mostrar o resultado, o link ajax que já foi definido como “/matricula.do”, o tipo de método no caso get, a função javascript para direcionar quando houver erro e os parâmetros configurados anteriormente.

```
<script language="JavaScript">
	function erro() {
		alert ("Ocorreu um erro ao buscar alunos!");
	}
	function listar() {
		var params = 'action=pesquisarAlunos&nomeAluno=' + $F
('nomeAluno');
		new Ajax.Updater(
				'listagem',
				'${link_ajax}',
				{
					method:'get',
					onFailure: erro,
					parameters: params
				});
	}
</script>
```

Figura 2 – Função Listar

A action pesquisarAlunos é requisitada e recebe como parâmetro o Nome do Aluno. Essa action segue com sua execução normal, sendo que o diferencial dela é o action forward, que direciona para um outro jsp que montará a lista de resposta.

```
actionForward = mapping.findForward("pgListAlunoAjax");
```

O padrão sugerido para nomenclatura desse jsp é _nomedoarquivoorigem.jsp, no exemplo fica _list_alunos.jsp. Esse novo jsp populado por meio da pesquisa efetuada na action será disponibilizado dentro do div listagem definido em list_alunos.jsp.

2.2.Material de Apoio

Além do uso demonstrado no tópico anterior, existe uma página web desenvolvida por Sergio Pereira que contém um excelente tutorial com outras possibilidades de uso do Ajax. Ele encontra-se anexado a esse documento e também pode ser acessado no seguinte endereço: http://www.sergiopereira.com/articles/prototype140.js.ptBR.html .

3 PRÓS E CONTRAS DO AJAX

3.1.Prós

- Menor utilização de banda da internet pois componentes estáticos como imagens, menus e cabeçalhos das páginas não precisam ser recarregados em cada requisição, sendo esses componentes os que mais consomem banda;
- A utilização do Ajax proporciona uma resposta mais rápida do sistema pois o conteúdo a ser buscado é apenas o de interesse do usuário;
- Os sistemas podem ser mais interativos pelo fato de Ajax usar poucos recursos;
- Evita o refresh das páginas que contém combos alinhadas.

3.2.Contras

- Sistemas feitos em Ajax aumentam a complexidade de desenvolvimento e manutenção;
- O debug dos códigos Ajax ainda não são simples de se realizar;
- Os desenvolvedores precisam saber JavaScript e DHTML muito bem;
- Em JavaScript o código-fonte precisa ser enviado para o computador do cliente ficando assim disponível para os usuários.

4.ANEXO

PROTOTYPE.JS

Anotações de um desenvolvedor sobre a prototype.js
abrange a versão 1.4.0

por Sergio Pereira

Ultima atualização : 28 de janeiro de 2006
O que é isso?

Caso você ainda não tenha tido a oportunidade de utilizá-la, prototype.js é uma ótima biblioteca javascript escrita por Sam Stephenson. Esta coleção de funções é impressionantemente bem escrita e bem pensada, **utiliza técnicas de acordo com os padrões** atuais e alivia seu trabalho na hora de produzir as páginas altamente interativas que caracterizam a chamada Web 2.0.

Se você andou tentando utilizar essa biblioteca recentemente, você provavelmente nottou que a documentação não pode ser considerada um de seus pontos fortes. Tal como muitos outros programadores fizeram, eu também só sonsegui entender como utilizar a prototype.js ao inspecionar seu código-fonte. Eu imaginei que poderia ser útil se eu fizesse algumas anotações enquanto eu aprendia e então compartilhasse com todo mundo.

Também estou incluindo aqui uma referência não-oficial para os objetos, classes, funções e extensões implementadas nessa biblioteca.

À medida que você ler os exemplos e a referência, caso você tenha familiaridade com a linguagem de progamação Ruby você notará uma semelhança proposital entre as classes constituintes de Ruby e muitas das extensões implementadas por esta biblioteca.

As funções utilitárias

A biblioteca vêm com diversos objetos pré-definidos e funções utilitárias. O objetivo claro dessas funções é evitar uma imensidão de digitação repetitiva e propensa a erros, que eu costumo comparar a musculação.

- **UTILIZANDO A FUNÇÃO `$()`**

A função `$()` é um conveniente atalho para as inúmeras ocorrências da chamada à função `document.getElementById()` do DOM. Tal como a função do DOM, esta retorna o elemento que é identificado pelo valor id passado como argumento.

No entanto, diferentemente da função do DOM, essa vai mais além. Você pode passar mais de um argumento e a função `$()` retornará um objeto `Array` com todos os elementos requisitados. O exemplo a seguir ilustra esse fato.

```
<HTML>
<HEAD>
<TITLE> Página de Teste </TITLE>
<script src="prototype-1.4.0.js"></script>
```

```
<script>
        function teste1()
        {
                var d = $('myDiv');
                alert(d.innerHTML);
        }

        function teste2()
        {
                var divs = $('myDiv','myOtherDiv');
                for(i=0; i<divs.length; i++)
                {
                        alert(divs[i].innerHTML);
                }
        }
</script>
</HEAD>

<BODY>
        <div id="myDiv">
                <p>Este é um parágrafo</p>
        </div>
        <div id="myOtherDiv">
                <p>Outro parágrafo</p>
        </div>

        <input type="button" value=Teste1 onclick="teste1();"><br>
        <input type="button" value=Teste2 onclick="teste2();"><br>

</BODY>
</HTML>
```

Outro ponto interessante dessa função é que você pode passar tanto o valor string do id de um elemento quanto o próprio objeto do elemento. Isso deixa a função mais flexível e bastante prática quando se deseja criar outras funções que tratam os argumentos da mesma forma.

- **USANDO A FUNÇÃO `$F()`**

A função `$F()` é outro atalho bem-vindo. Ela retorna o valor de qualquer campo de um formulário (input control) tais como caixas de texto ou listas. A função pode receber como parâmetro tanto o id do elemento como o próprio objeto do elemento.

```
<script>
```

```
        function teste3()
        {
                alert(  $F('nomeUsuario')  );
        }
</script>

<input type="text" id="nomeUsuario" value="João da Silva"><br>
<input type="button" value=Teste3 onclick="teste3();"><br>
```

- **USANDO A FUNÇÃO `$A()`**

A função `$A()` converte o único argumento que ela aceita em um objeto do tipo `Array`.

Esta função, combinada com as extensões da classe Array, torna mais fácil converter or copiar qualquer lista enumerável em um `Array`. Uma utilização sugerida é para converter objetos do tipo `NodeList` do DOM em arrays comuns, que podem ser percorridos mais eficientement. Veja o exemplo a seguir.

```
<script>

        function mostraOpcoes(){
                var minhaNodeList = $('lstEmpregados').
getElementsByTagName('option');
                var nodes = $A(minhaNodeList);

                nodes.each(function(node){
                                alert(node.nodeName + ': ' +
node.innerHTML);
                        });
        }
</script>

<select id="lstEmpregados" size="10" >
        <option value="5">Buchanan, Steven</option>
        <option value="8">Callahan, Laura</option>
        <option value="1">Davolio, Nancy</option>
</select>

<input type="button" value="Mostre items da lista" onclick="mostraOpcoes
();" >
```

- **USANDO A FUNÇÃO `$H()`**

A função `$H()` converte objetos em um Hash enumerável, que se assemelha a um arrays associativo (lista de pares chave/valor).

```
<script>
	function testarHash()
	{
		//criando o objeto
		var a = {
			primeiro: 10,
			segundo: 20,
			terceiro: 30
			};

		//transformando-o em um hash
		var h = $H(a);
		alert(h.toQueryString()); //mostra:
primeiro=10&segundo=20&terceiro=30
	}

</script>
```

- **USANDO A FUNÇÃO `$R()`**

A função `$R()` é um atalho para `new ObjectRange(limiteInferior, limiteSuperior, excluirLimites)`.

Siga para a documentação da classe ObjectRange onde você encontrará uma explicação mais abrangente sobre essa classe. Por momento, vamos examinar um exemplo simplificado que também demonstra o uso de iteradores por meio do método `each`. Maiores detalhes a respeito desse método podem ser encontrados na documentação da classe Enumerable.

```
<script>
	function demoDollar_R(){
		var range = $R(10, 20, false);
		range.each(function(value, index){
			alert(value);
		});
	}

</script>

<input type="button" value="Contagem" onclick="demoDollar_R();" >
```

- **UTILIZANDO A FUNÇÃO `TRY.THESE()`**

A função `Try.these()` facilita as coisas na hora que se torna necessário experimentar diferentes chamadas a funções até que uma funcione. Esta função aceita uma lista de chamadas a outras funções como argumento. Então cada chamada é executada na ordem dada, até que uma dê certo e então o resultado dessa chamada é retornado.

No exemplo a seguir, a propriedade `xmlNode.text` funciona em alguns browsers, e `xmlNode.textContent` funciona em outros browsers. Ao utilizarmos a função `Try.these()` nós podemos sempre retornar a chamada que funciona corretamente (sem erros).

```
<script>
function getXmlNodeValue(xmlNode){
        return Try.these(
                function() {return xmlNode.text;},
                function() {return xmlNode.textContent;)
                );
}
</script>
```

O objeto `Ajax`

Tá bem, as funções utilitárias mencionadas acima são legais mas convenhamos, elas também não são a coisa mais avançada que já se viu, não é? Você provavelmente poderia ter escrito essas funções sozinho ou talvez até você já tenha algumas funções semelhantes em seus próprios scripts. Só que essas funções são apenas a ponta do iceberg.

Eu tenho certeza que seu interesse pela prototype.js advém principalmente de suas capacidades relacionadas a AJAX. Então vamos explicar como a biblioteca facilita as coisas pra você quando é necessário implementar operações de AJAX.

O objeto `Ajax` é pré-definido na biblioteca, criado para encapsular e simplificar o código traiçoeiro que é necessário quando se implementam funcionalidades AJAX. Este objeto contém uma série de classes que disponibilizam lógica AJAX. Vamos dar uma olhada em algumas desas classes.

- **UTILIZANDO A CLASSE `AJAX.REQUEST`**

Se você não estiver utilizando nenhuma biblioteca auxiliar, você provavelmente está escrevendo uma montanha de código para criar um objeto `XMLHttpRequest`, acompanhar seu progresso assincronamente, e então extrair a resposta e processá-la. Considere-se um sortudo se você não precisa suportar mais de um tipo de browser.

Para auxiliar com operações AJAX, a biblioteca define a classe `Ajax.Request`.

Digamos que você tenha uma aplicação que pode se comunicar com o servidor através de uma url como *http://servidor/app/busca_vendas?empID=1234&ano=1998*, que retorna uma resposta em XML similar à seguinte.

```
<?xml version="1.0" encoding="utf-8" ?>
```

```
<ajax-response>
        <response type="object" id="productDetails">
                <monthly-sales>
                        <employee-sales>
                                <employee-id>1234</employee-id>
                                <year-month>1998-01</year-month>
                                <sales>$8,115.36</sales>
                        </employee-sales>
                        <employee-sales>
                                <employee-id>1234</employee-id>
                                <year-month>1998-02</year-month>
                                <sales>$11,147.51</sales>
                        </employee-sales>
                </monthly-sales>
        </response>
</ajax-response>
```

Contactar o servidor para buscar esse XML é bastante simples utilizando um objeto `Ajax.Request`. O exemplo abaixo mostra como isso pode ser feito.

```
<script>
        function buscaVendas()
        {
                var empID = $F('lstEmpregados');
                var y = $F('lstAnos');
                var url = 'http://servidor/app/busca_vendas';
                var pars = 'empID=' + empID + '&ano=' + y;


                var myAjax = new Ajax.Request(
                        url,
                        {
                                method: 'get',
                                parameters: pars,
                                onComplete: mostraResposta
                        });


        }

        function mostraResposta(requisicaoOriginal)
        {
                //copia o XML retornado para o textarea
```

```
                $('resultado').value = requisicaoOriginal.responseText;
        }
</script>

<select id="lstEmpregados" size="10" onchange="buscaVendas()">
        <option value="5">Buchanan, Steven</option>
        <option value="8">Callahan, Laura</option>
        <option value="1">Davolio, Nancy</option>
</select>
<select id="lstAnos" size="3" onchange="buscaVendas()">
        <option selected="selected" value="1996">1996</option>
        <option value="1997">1997</option>
        <option value="1998">1998</option>
</select>
<br><textarea id=resultado cols=60 rows=10 ></textarea>
```

Dá para você perceber o segundo parâmetro passado ao contrutor do objeto `Ajax.Request`? O parâmetro `{method: 'get', parameters: pars, onComplete: mostraResposta}` representa um objeto anônimo em notação literal (conhecida por JSON). O que ele significa é que estamos passando um objeto que tem uma propriedade chamada `method` que contém a string `'get'`, uma outra propriedade chamada `parameters` que contém a querystring da requisição HTTP, e uma propriedade/método `onComplete` contendo a função `mostraResposta`.

Ainda há mais algumas propriedades que você pode definir nesse objeto, tais como `asynchronous`, que pode ser `true` ou `false` e determina se a chamada AJAX ao servidor será feita de maneira assíncrona ou não (o valor default é `true`.)

Este parâmetro define as opções para a chamada AJAX. Em nosso exemplo estamos chamando a url do primeiro argumento através de um comando HTTP GET, passando a querystring contida na variável `pars`, e o objeto `Ajax.Request` irá chamar a função `mostraResposta` quando terminar de buscar receber a resposta.

Como se sabe, o objeto `XMLHttpRequest` anuncia o progresso durante a chamada HTTP. Este progresso pode representar quatro diferentes estágios:*Loading* (carregando), *Loaded* (carregado), *Interactive* (interativo), or *Complete* (completo). Você pode fazer com que o objeto `Ajax.Request` chame uma função sua em cada um desses estágios, sendo o *Complete* o mais comum. Para especificar a função para o objeto, basta utilizar propriedades/métodos chamados `onXXXXX` nas opções da chamada, tal como a propriedade `onComplete` em nosso exemplo anterior. A função especificada será chamada pelo objeto com um único argumento, que será o próprio objeto `XMLHttpRequest`. Você poderá então utilizar esse objeto para extrair os dados retornados e possivelmente checar a propriedade `status`, que informará o resultado (código) da chamada HTTP.

Duas outra opções interessantes podem ser utilizadas para processar os resultados. Nós podemos especificar a opção `onSuccess` como uma função a ser chamada quando a chamada AJAX executa sem erros. Analogamente, a opção `onFailure` pode ser espcificada

como uma função a ser chamada quando um erro ocorrer durante a chamada. Tal como as funções `onXXXXX`, essas duas funções também serão chamadas com um argumento que conterá o objeto `XMLHttpRequest` que serviu de veículo para a chamada AJAX.

Nosso exemplo não fez nenhum processamento interessante da resposta XML. Nós apenas ecoamos o XML no textarea. Uma utilizaçãomais típica da resposta iria provavelmente extrair a informação desejada de dentro do XML e atualizar alguns elementos da página, ou talvez mesmo utilizar uma transformação XSLT para criar HTML e inserir na página.

Na versão 1.4.0 da biblioteca uma nova forma de callback de evento foi introduzida. Se você possui código que deve sempre ser executado para um evento específico, não importando qual foi a chamada AJAX que causou o evento, então você pode utilizar o novo objeto Ajax.Responders.

Suponhamos que você necessita mostrar algum indicativo vivível de que uma chamada AJAX está em curso; algo como um GIF animado ou coisa do gênero. Você pode usar duas funções de callback, uma para mostrar o GIF quando a primeira chamada se iniciar e outra para escondê-lo quando a última chamada for concluída. Veja o exemplo abaixo.

```
<script>
        var callbacksGlobais = {
                onCreate: function(){
                        Element.show('chamadaEmCurso');
                },

                onComplete: function() {
                        if(Ajax.activeRequestCount == 0){
                                Element.hide('chamadaEmCurso');
                        }
                }
        };

        Ajax.Responders.register(callbacksGlobais);
</script>

<div id='chamadaEmCurso'><img src='macaco_pulando.gif'>Carregando...</div>
```

Para maiores detalhes, dê uma olhada na referência do Ajax.Request e na referência das opções.

- **UTILIZANDO A CLASSE `AJAX.UPDATER`**

Supondo que você tenha uma URL no seu servidor que possa retornar os dados já formatados em HTML, então a biblioteca facilita ainda mais sua vida com a classe `Ajax.Updater`. Com ela basta você informar qual o elemento que deve ser preenchido com o HTML que será retornado pela chamada AJAX. Um exemplo demonstra isso melhor do que eu conseguiria descrever.

```
<script>
        function buscaHTML()
        {
                var url = 'http://servidor/app/buscaHTML';
                var pars = 'algumParametro=ABC';


                var myAjax = new Ajax.Updater(
                        'resposta_aqui',
                        url,
                        {
                                method: 'get',
                                parameters: pars
                        });


        }
</script>

<input type=button value="Busca HTML" onclick="buscaHTML()">
<div id="resposta_aqui"></div>
```

Como você pode ver, o código é muito semelhante ao exemplo anterior, excluindo-se a função `onComplete` e passando-se o id do elemento no construtor. Vamos alterar um pouco o código para ilustrar como é possível tratar erros produzidos no servidor em seu código no cliente.

Vamos incluir mais opções na chamada, especificando uma função para capturar situações de erro. Isso se faz com o auxílio da opção `onFailure`. Vamos também especificar que o elemento de id `resposta_aqui` apenas será preenchido se a operação for concluída com sucesso. Para que isso seja possível, vamos mudar o primeiro parametro de um simples id de elemento para um objeto. O construtor de `Ajax.Updater` aceita também como primeiro parâmetro um objeto com duas propriedades, `success` (a ser usado quando tudo termina bem) e `failure` (a ser usado quando um erro ocorre na chamada). No nosso caso não precisaremos da propriedade `failure` pois estaremos usando uma função nossa para tratar o erro. A função `reportError` tratará o erro conforme especificado na opção `onFailure`.

```
<script>
        function buscaHTML()
        {
                var url = 'http://servidor/app/buscaHTML';
                var pars = 'algumParametro=ABC';
```

```
                var myAjax = new Ajax.Updater(
                                        {success: 'placeholder'},
                                        url,
                                        {
                                                method: 'get',
                                                parameters: pars,
                                                onFailure: mostraErro
                                        });


        }

        function mostraErro(request)
        {
                alert('Foi mal. Deu pau...');
        }
</script>

<input type=button value="Busca HTML" onclick="buscaHTML()">
<div id="resposta_aqui"></div>
```

Um outro caso interessante é se seu servidor retorna código em JavaScript ou invés de HTML. O objeto `Ajax.Updater` pode avaliar o código JavaScript. Para fazer o objeto tratar a resposta do servidor como JavaScript, basta que você use a opção `evalScripts: true;`.

Para mais detalhes sobre essa classe, veja e referência do Ajax.Updater e a referência das opções.

Referência da prototype.js

- **EXTENSÕES DAS CLASSES JAVASCRIPT**

Uma das formas que prototype.js adiciona funcionalidade é extendendo as classes JavaScript existentes.

- **EXTENSÕES DA CLASSE OBJECT**

Método	*Tipo*	*Argumentos*	*Descrição*
extend (destination, source)	estático	destination: qualquer objeto, source: qualquer objeto	Provém uma forma de se conseguir herança ao se copiar todas as propriedades e métodos de `source` para `destination`.

inspect (targetObj)	estático	targetObj: qualquer objeto	Retorna uma string legível que representa targetObj. Caso o objeto passado não implemente um método `inspect` então a função retornará `toString`.

- **EXTENSÕES DA CLASSE `NUMBER`**

Método	*Tipo*	*Argumentos*	*Descrição*
toColorPart()	de objeto	(nenhum)	Retorna a representação hexadecimal do número. Útil quando se quer converter valores RGB de uma cor para seu formato em HTML.
succ()	de objeto	(nenhum)	Retorna o próximo número. Esse método é comumente utilizado em contextos que envolvem iterações.
times (iterator)	de objeto	iterator: uma função com assinatura equivalente a Function (valor, indice)	Executa a função `iterator` repetidamente, passando os parâmetros `valor` e `indice` contendo o valor da vez na iteração e o índice da vez, respectivamente.

O exemplo a seguir mostrará mensagens com os números de 0 a 9.

```
<script>
        function usandoTimes(){
                var n = 10;
                n.times(function(valor, indice){
                        alert(indice);
                });
                /*************************
                 * Tambem daria certo assim:
                 *           (10).times( .... );
                 *************************/
        }

</script>

<input type=button value="Testar Number.times()" onclick="usandoTimes()">
```

- **EXTENSÕES DA CLASSE `FUNCTION`**

Método	*Tipo*	*Argumentos*	*Descrição*
bind(object)	de objeto	object: o objeto ao qual o método pertence	Retorna uma instância da função atrelada ao objeto que à contém. A nova função têm os mesmos argumentos que a original.
bindAsEventListener(object)	de objeto	object: o objeto ao qual o método pertence	Retorna uma instância da função atrelada ao objeto que à contém. A nova função terá o objeto `event` atual como argumento.

Vamos ver como se usam essas extensões.

```
<input type=checkbox id=myChk value=1> Test?
<script>
        //declarando a classe
        var CheckboxWatcher = Class.create();

        //definindo o resto da classe
        CheckboxWatcher.prototype = {

           initialize: function(chkBox, mensagem) {
                        this.chkBox = $(chkBox);
                        this.mensagem = mensagem;
                        //ligando nosso metodo ao evento


                        this.chkBox.onclick =
                           this.mostraMensagem.bindAsEventListener(this);


           },

           mostraMensagem: function(evt) {
                  alert(this.mensagem + ' (' + evt.type + ')');
           }
        };


        var watcher = new CheckboxWatcher('myChk', 'Mudou');
</script>
```

- **EXTENSÕES DA CLASSE `STRING`**

Método	*Tipo*	*Argumentos*	*Descrição*
stripTags()	de objeto	(nenhum)	Retorna a string subtraindo quaisquer tags HTML ou XML.
stripScripts()	de objeto	(nenhum)	Retorna a string, subtraindo quaisquer blocos `<script />`.
escapeHTML()	de objeto	(nenhum)	Retorna a string dando escape em quaisquer caracteres de marcação HTML.
unescapeHTML()	de objeto	(nenhum)	A operação reversa de `escapeHTML()`
extractScripts()	de objeto	(nenhum)	Retorna um objeto `Array` com mtodos os blocos `<script />` encontrados na string.
evalScripts()	de objeto	(nenhum)	Avalia todos os blocos `<script />` encontrados na string.
toQueryParams()	de objeto	(nenhum)	Separa a string em um array associativo indexado pelo nome de cada parametro (tal como um hash).
parseQuery()	de objeto	(nenhum)	Mesmo que `toQueryParams()`.
toArray()	de objeto	(nenhum)	Retrorna um `Array` com cada caractere da string.
camelize()	de objeto	(nenhum)	Converte um string-separada-por-hífens em uma stringFormatadaComCamelCase. Essa função pode ser útil quando se lida com propriedades de estilo, por exemplo.

EXTENSÕES DA CLASSE `ARRAY`

De cara, a classe `Array` extende `Enumerable`, então todos aqueles métodos úteis de `Enumerable` estão disponíveis. Alem destes, os métodos a seguir também estão incluídos.

Método	*Tipo*	*Argumentos*	*Descrição*
clear()	de objeto	(nenhum)	Esvazia o array e retorna-o.
first()	de objeto	(nenhum)	Retorna o primeiro elemento do array.
last()	de objeto	(nenhum)	Retorna o último elemento do array.
compact()	de objeto	(nenhum)	Retorna o array sem os elementos que sejam `null` ou `undefined`. Este método não altera o array original, apenas retorna um novo.

flatten()	de objeto	(nenhum)	Retorna uma versão unidimensional (achatada) do array. Esse achatamento ocorre porque cada elemento do array que também é um array terá seus elementos incluídos no array-pai. Esse processo ocorre de forma recursiva para cada elemento.
without (valor1 [, valor2 [, .. valorN]])	de objeto	valor1 ... valorN: valores a serem excluídos se encontrados no array.	Retorna o array subtraindo os elementos listados como argumentos.
indexOf (valor)	de objeto	valor: o valor procurado.	Retorna o índice, baseado em zero, do primeiro elemento encontrado com o `valor`. Retorna -1 se nenhuma ocorrência for encontrada.
reverse ([alterarOriginal])	de objeto	alterarOriginal: indica se o array original será revertido.	Retorna o array em ordem reversa. Se nenhum argumento for dado ou se o argumento for `true` então o array original também será revertido. Caso contrário ele permanecerá do jeito que está.
shift()	de objeto	(nenhum)	Retorna o primeiro elemento do array e retira-o do array, reduzindo o tamanho do array em 1 elemento.
inspect()	de objeto	(nenhum)	Retorna uma string com os elementos do array.

- **EXTENSÕES DO OBJETO `DOCUMENT` DO DOM**

Método	*Tipo*	*Argumentos*	*Descrição*
getElementsByClassName (nomeDaClasse [, elementoPai])	de objeto	nomeDaClasse: nome da clase CSS associada com os elementos, elementoPai: objeto ou id do elemento que contém os elementos a serem buscados.	Retorna todos os elementos com a classe CSS informada. Se o elemento-pai não for dado, então os elementos serão procurados dentro do elemento `body`.

- **EXTENSÕES DO OBJETO `EVENT`**

Propriedade	*Tipo*	*Descrição*
KEY_BACKSPACE	Number	8: Constant. Code for the Backspace key.
KEY_TAB	Number	9: Constant. Code for the Tab key.
KEY_RETURN	Number	13: Constant. Code for the Return key.
KEY_ESC	Number	27: Constant. Code for the Esc key.

KEY_LEFT	Number	37: Constant. Code for the Left arrow key.
KEY_UP	Number	38: Constant. Code for the Up arrow key.
KEY_RIGHT	Number	39: Constant. Code for the Right arrow key.
KEY_DOWN	Number	40: Constant. Code for the Down arrow key.
KEY_DELETE	Number	46: Constant. Code for the Delete key.
observers:	Array	List of cached observers. Part of the internal implementation details of the object.

Método	***Tipo***	***Argumentos***	***Descrição***
element(event)	estático	event: an Event object	Returns element that originated the event.
isLeftClick (event)	estático	event: an Event object	Returns `true` if the left mouse button was clicked.
pointerX(event)	estático	event: an Event object	Returns the x coordinate of the mouse pointer on the page.
pointerY(event)	estático	event: an Event object	Returns the y coordinate of the mouse pointer on the page.
stop(event)	estático	event: an Event object	Use this function to abort the default behavior of an event and to suspend its propagation.
findElement (event, tagName)	estático	event: an Event object, tagName: name of the desired tag.	Traverses the DOM tree upwards, searching for the first element with the given tag name, starting from the element that originated the event.
observe (element, name, observer, useCapture)	estático	element: object or id, name: event name (like 'click', 'load', etc), observer: function to handle the event, useCapture: if `true`, handles the event in the *capture* phase and if `false` in the *bubbling* phase.	Adds an event handler function to an event.
stopObserving (element, name, observer, useCapture)	estático	element: object or id, name: event name (like 'click'), observer: function that is handling the event, useCapture: if true handles the event in the *capture* phase and if false in the *bubbling* phase.	Removes an event handler from the event.

_observeAndCache(element, name, observer, useCapture)	estático		Private method, do not worry about it.
unloadCache()	estático	(nenhum)	Private method, do not worry about it. Clears all cached observers from memory.

Let's see how to use this object to add an event handler to the load event of the `window` object.

```
<script>
        Event.observe(window, 'load', showMessage, false);

        function showMessage() {
          alert('Page loaded.');
        }
</script>
```

- **NOVOS OBJETOS E CLASSES DEFINIDOS EM PROTOTYPE.JS**

Outra forma que a biblioteca ajuda é ao disponibilizar diversos objetos que implementam suporte a design orientado a objetos e também funcionalidades de uso geral.

- **THE `PERIODICALEXECUTER` OBJECT**

This object provides the logic for calling a given function repeatedly, at a given interval.

Método	*Tipo*	*Argumentos*	*Descrição*
[ctor] (callback, interval)	constructor	callback: a parameterless function, interval: number of seconds	Creates one instance of this object that will call the function repeatedly.

Propriedade	*Tipo*	*Descrição*
callback	Function()	The function to be called. No parameters will be passed to it.
frequency	Number	This is actually the interval in seconds
currentlyExecuting	Boolean	Indicates if the function call is in progress

THE `PROTOTYPE` OBJECT

The `Prototype` object does not have any important role, other than declaring the version of the library being used.

Propriedade	*Tipo*	*Descrição*

Version	String	The version of the library
emptyFunction	Function()	An empty function object

- **THE `CLASS` OBJECT**

The `Class` object is used when declaring the other classes in the library. Using this object when declaring a class causes the to new class to support an `initialize()` method, which serves as the constructor.

See the sample below.

```
//declaring the class
var MySampleClass = Class.create();

//defining the rest of the class implmentation
MySampleClass.prototype = {

   initialize: function(message) {
               this.message = message;
   },

   showMessage: function(ajaxResponse) {
      alert(this.message);
   }
};

//now, let's instantiate and use one object
var myTalker = new MySampleClass('hi there.');
myTalker.showMessage(); //displays alert
```

Método	*Tipo*	*Argumentos*	*Descrição*
create(*)	de objeto	(any)	Defines a constructor for a new class

THE `AJAX` OBJECT

This object serves as the root for many other classes that provide AJAX functionality.

Método	*Tipo*	*Argumentos*	*Descrição*
getTransport()	de objeto	(nenhum)	Returns a new `XMLHttpRequest` object

- **THE `AJAX.BASE`**

This class is used as the base class for the other classes defined in the `Ajax` object.

Método	*Tipo*	*Argumentos*	*Descrição*
setOptions(options)	de objeto	options: AJAX options	Sets the desired options for the AJAX operation
responseIsSuccess()	de objeto	(nenhum)	Returns `true` if the AJAX operation succeeded, `false` otherwise
responseIsFailure()	de objeto	(nenhum)	The opposite of `responseIsSuccess()`.

- **THE `AJAX.REQUEST`**

Inherits from `Ajax.Base`

Encapsulates AJAX operations

Propriedade	*Tipo*	*Tipo*	*Descrição*
Events	Array	estático	List of possible events/statuses reported during an AJAX operation. The list contains: 'Uninitialized', 'Loading', 'Loaded', 'Interactive', and 'Complete.'
transport	XMLHttpRequest	de objeto	The `XMLHttpRequest` object that carries the AJAX operation

Método	*Tipo*	*Argumentos*	*Descrição*
[ctor](url, options)	constructor	url: the url to be fetched, options: AJAX options	Creates one instance of this object that will call the given `url` using the given `options`. **Important:** It is worth noting that the chosen url is subject to the borwser's security settings. In many cases the browser will not fetch the url if it is not from the same host (domain) as the current page. You should ideally use only local urls to avoid having to configure or restrict the user's browser. (Thanks Clay).
request(url)	de objeto	url: url for the AJAX call	This method is typically not called externally. It is already called during the constructor call.
setRequestHeaders()	de objeto	(nenhum)	This method is typically not called externally. It is called by the object itself to assemble the HTTP header that will be sent during the HTTP request.
onStateChange()	de objeto	(nenhum)	This method is typically not called externally. It is called by the object itself when the AJAX call status changes.

respondToReadyState (readyState)	de objeto	readyState: state number (1 to 4)	This method is typically not called externally. It is called by the object itself when the AJAX call status changes.

- **THE `OPTIONS` ARGUMENT OBJECT**

An important part of the AJAX operations is the `options` argument. There's no `options` class per se. Any object can be passed, as long as it has the expected properties. It is common to create anonymous objects just for the AJAX calls.

Propriedade	***Tipo***	***Default***	***Descrição***
method	String	'post'	Method of the HTTP request
parameters	String	''	The url-formatted list of values passed to the request
asynchronous	Boolean	true	Indicates if the AJAX call will be made asynchronously
postBody	String	undefined	Content passed to in the request's body in case of a HTTP POST
requestHeaders	Array	undefined	List of HTTP headers to be passed with the request. This list must have an even number of items, any odd item is the name of a custom header, and the following even item is the string value of that header. Example: `['my-header1', 'this is the value', 'my-other-header', 'another value']`
onXXXXXXXX	Function (XMLHttpRequest)	undefined	Custom function to be called when the respective event/status is reached during the AJAX call. Example `var myOpts = {onComplete: showResponse, onLoaded: registerLoaded};`. The function used will receive one argument, containing the `XMLHttpRequest` object that is carrying the AJAX operation.
onSuccess	Function (XMLHttpRequest)	undefined	Custom function to be called when the AJAX call completes successfully. The function used will receive one argument, containing the `XMLHttpRequest` object that is carrying the AJAX operation.
onFailure	Function (XMLHttpRequest)	undefined	Custom function to be called when the AJAX call completes with error. The function used will receive one argument, containing the `XMLHttpRequest` object that is carrying the AJAX operation.
insertion	Function (Object, String)	null	Function to be called to inject the returned text into an element. The function will be called with two arguments, the element object to be updated and the response text Applies only to `Ajax.Updater` objects.

evalScripts	Boolean	undefined, false	Determines if script blocks will be evaluated when the response arrives. Applies only to `Ajax.Updater` objects.
decay	Number	undefined, 1	Determines the progressive slowdown in a `Ajax.PeriodicalUpdater` object refresh rate when the received response is the same as the last one. For example, if you use 2, after one of the refreshes produces the same result as the previous one, the object will wait twice as much time for the next refresh. If it repeats again, the object will wait four times as much, and so on. Leave it undefined or use 1 to avoid the slowdown.

- **THE `AJAX.UPDATER`**

Inherits from `Ajax.Request`

Used when the requested url returns HTML that you want to inject directly in a specific element of your page. You can also use this object when the url returns `<script>` blocks that will be evaluated upon arrival. Use the `evalScripts` option to work with scripts.

Propriedade	***Tipo***	***Tipo***	***Descrição***
ScriptFragment	String	estático	A regular expression to identify scripts
containers	Object	de objeto	This object contains two properties: `containers.success` will be used when the AJAX call succeeds, and `containers.failure` will be used otherwise.

Método	***Tipo***	***Argumentos***	***Descrição***
[ctor] (container, url, options)	constructor	container:this can be the id of an element, the element object itself, or an object with two properties - **object.success** element (or id) that will be used when the AJAX call succeeds, and **object.failure** element (or id) that will be used otherwise. url: the url to be fetched, options: `AJAX options`	Creates one instance of this object that will call the given `url` using the given `options`.

updateContent ()	de objeto	(nenhum)	This method is typically not called externally. It is called by the object itself when the response is received. It will update the appropriate element with the HTML or call the function passed in the `insertion` option. The function will be called with two arguments, the element to be updated and the response text.

- **THE `AJAX.PERIODICALUPDATER`**

Inherits from `Ajax.Base`

This class repeatedly instantiates and uses an `Ajax.Updater` object to refresh an element on the page, or to perform any of the other tasks the `Ajax.Updater` can perform. Check the Ajax.Updater reference for more information.

Propriedade	***Tipo***	***Tipo***	***Descrição***
container	Object	de objeto	This value will be passed straight to the `Ajax.Updater`'s constructor.
url	String	de objeto	This value will be passed straight to the `Ajax.Updater`'s constructor.
frequency	Number	de objeto	Interval (not frequency) between refreshes, in seconds. Defaults to 2 seconds. This number will be multiplied by the current `decay` when invoking the`Ajax.Updater` object
decay	Number	de objeto	Keeps the current decay level applied when re-executing the task
updater	Ajax.Updater	de objeto	The most recently used `Ajax.Updater` object
timer	Object	de objeto	The JavaScript timer being used to notify the object when it is time for the next refresh.

Método	***Tipo***	***Argumentos***	***Descrição***
[ctor](container, url, options)	constructor	container:this can be the id of an element, the element object itself, or an object with two properties - `object.success` element (or id) that will be used when the AJAX call succeeds, and `object.failure` element (or id) that will be used otherwise. url: the url to be fetched, options: `AJAX options`	Creates one instance of this object that will call the given `url` using the given `options`.

start()	de objeto	(nenhum)	This method is typically not called externally. It is called by the object itself to start performing its periodical tasks.
stop()	de objeto	(nenhum)	This method is typically not called externally. It is called by the object itself to stop performing its periodical tasks.
updateComplete()	de objeto	(nenhum)	This method is typically not called externally. It is called by the currently used `Ajax.Updater` after if completes the request. It is used to schedule the next refresh.
onTimerEvent()	de objeto	(nenhum)	This method is typically not called externally. It is called internally when it is time for the next update.

- **THE `ELEMENT` OBJECT**

This object provides some utility functions for manipulating elements in the DOM.

Método	*Tipo*	*Argumentos*	*Descrição*
toggle(elem1 [, elem2 [, elem3 [...]]])	de objeto	elemN: element object or id	Toggles the visibility of each passed element.
hide(elem1 [, elem2 [, elem3 [...]]])	de objeto	elemN: element object or id	Hides each element bu setting its `style.display` to `'none'`.
show(elem1 [, elem2 [, elem3 [...]]])	de objeto	elemN: element object or id	Shows each element bu resetting its `style.display` to `''`.
remove(element)	de objeto	element: element object or id	Removes the element from the document.
getHeight (element)	de objeto	element: element object or id	Returns the `offsetHeight` of the element
addClassName (element, className)	de objeto	element: element object or id, className: name of a CSS class	Adds the given class name to the element's class names.

hasClassName (element, className)	de objeto	element: element object or id, className: name of a CSS class	Returns `true` if the element has the given class name as one of its class names.
removeClassName (element, className)	de objeto	element: element object or id, className: name of a CSS class	Removes the given class name from the element's class names.
cleanWhitespace (element)	de objeto	element: element object or id	Removes any white space text node children of the element

- **THE `ABSTRACT` OBJECT**

This object serves as the root for other classes in the library. It does not have any properties or methods. The classes defined in this object are also treated as traditional abstract classes.

The `Abstract.Insertion`

This class is used as the base class for the other classes that will provide dynamic content insertion. This class is used like an abstract class.

Método	*Tipo*	*Argumentos*	*Descrição*
[ctor] (element, content)	constructor	element: element object or id, content: HTML to be inserted	Creates an object that will help with dynamic content insertion.

Propriedade	*Tipo*	*Tipo*	*Descrição*
adjacency	String	static, parameter	Parameter that specifies where the content will be placed relative to the given element. The possible values are: `'beforeBegin'`, `'afterBegin'`, `'beforeEnd'`, and `'afterEnd'`.
element	Object	de objeto	The element object that the insertion will be made relative to.
content	String	de objeto	The HTML that will be inserted.

- **THE `INSERTION` OBJECT**

This object serves as the root for other classes in the library. It does not have any properties or methods. The classes defined in this object are also treated as traditional abstract classes.

- **THE `INSERTION.BEFORE`**

Inherits from `Abstract.Insertion`

Inserts HTML before an element.

Método	*Tipo*	*Argumentos*	*Descrição*

[ctor] (element, content)	constructor	element: element object or id, content: HTML to be inserted	Inherited from Abstract.Insertion. Creates an object that will help with dynamic content insertion.

The following code

```
<br>Hello, <span id="person" style="color:red;">Wiggum. How's it
going?</span>

<script> new Insertion.Before('person', 'Chief '); </script>
```

Will change the HTML to

```
<br>Hello, Chief <span id="person" style="color:red;">Wiggum. How's it
going?</span>
```

- **THE `INSERTION.TOP`**

Inherits from Abstract.Insertion

Inserts HTML as the first child under an element. The content will be right after the opening tag of the element.

Método	*Tipo*	*Argumentos*	*Descrição*
[ctor] (element, content)	constructor	element: element object or id, content: HTML to be inserted	Inherited from Abstract.Insertion. Creates an object that will help with dynamic content insertion.

The following code

```
<br>Hello, <span id="person" style="color:red;">Wiggum. How's it
going?</span>

<script> new Insertion.Top('person', 'Mr. '); </script>
```

Will change the HTML to

```
<br>Hello, <span id="person" style="color:red;">Mr. Wiggum. How's it
going?</span>
```

- **THE `INSERTION.BOTTOM`**

Inherits from Abstract.Insertion

Inserts HTML as the last child under an element. The content will be right before the element's closing tag.

Método	*Tipo*	*Argumentos*	*Descrição*
[ctor] (element, content)	constructor	element: element object or id, content: HTML to be inserted	Inherited from `Abstract.Insertion`. Creates an object that will help with dynamic content insertion.

The following code

```
<br>Hello, <span id="person" style="color:red;">Wiggum. How's it
going?</span>

<script> new Insertion.Bottom('person', " What's up?"); </script>
```

Will change the HTML to

```
<br>Hello, <span id="person" style="color:red;">Wiggum. How's it going?
What's up?</span>
```

- **THE `INSERTION.AFTER`**

Inherits from `Abstract.Insertion`

Inserts HTML right after the element's closing tag.

Método	*Tipo*	*Argumentos*	*Descrição*
[ctor] (element, content)	constructor	element: element object or id, content: HTML to be inserted	Inherited from `Abstract.Insertion`. Creates an object that will help with dynamic content insertion.

The following code

```
<br>Hello, <span id="person" style="color:red;">Wiggum. How's it
going?</span>

<script> new Insertion.After('person', ' Are you there?'); </script>
```

Will change the HTML to

```
<br>Hello, <span id="person" style="color:red;">Wiggum. How's it
going?</span> Are you there?
```

- **THE `FIELD` OBJECT**

This object provides some utility functions for working with input fields in forms.

Método	*Tipo*	*Argumentos*	*Descrição*
clear(field1 [, field2 [, field3 [...]]])	de objeto	fieldN: field element object or id	Clears the value of each passed form field element.
present(field1 [, field2 [, field3 [...]]])	de objeto	fieldN: field element object or id	Returns `true` only if all forms fields contain non-empty values.
focus(field)	de objeto	field: field element object or id	Moves the input focus to the given form field.
select(field)	de objeto	field: field element object or id	Selects the value in fields that support text selection
activate(field)	de objeto	field: field element object or id	Move the focus and selects the value in fields that support text selection

- **THE `FORM` OBJECT**

This object provides some utility functions for working with data entry forms and their input fields.

Método	*Tipo*	*Argumentos*	*Descrição*
serialize(form)	de objeto	form: form element object or id	Returns a url-formatted list of field names and their values, like `'field1=value1&field2=value2&field3=value3'`
getElements (form)	de objeto	form: form element object or id	Returns an `Array` containing all the input fields in the form.
getInputs(form [, typeName [, name]])	de objeto	form: form element object or id, typeName: the type of the input element, name: the name of the input element.	Returns an `Array` containing all the `<input>` elements in the form. Optionally, the list can be filtered by the `type` or `name` attributes of the elements.
disable(form)	de objeto	form: form element object or id	Disables all the input fields in the form.
enable(form)	de objeto	form: form element object or id	Enables all the input fields in the form.
focusFirstElement(form)	de objeto	form: form element object or id	Activates the first visible, enabled input field in the form.
reset(form)	de objeto	form: form element object or id	Resets the form. The same as calling the `reset()` method of the form object.

- **THE `FORM.ELEMENT` OBJECT**

This object provides some utility functions for working with form elements, visible or not.

Método	*Tipo*	*Argumentos*	*Descrição*
serialize(element)	de objeto	element: element object or id	Returns the element's name=value pair, like `'elementName=elementValue'`
getValue(element)	de objeto	element: element object or id	Returns the value of the element.

- **THE `FORM.ELEMENT.SERIALIZERS` OBJECT**

This object provides some utility functions that are used internally in the library to assist extracting the current value of the form elements.

Método	*Tipo*	*Argumentos*	*Descrição*
inputSelector (element)	de objeto	element: object or id of a form element that has the *checked* property, like a radio button or checkbox.	Returns an `Array` with the element's name and value, like `['elementName', 'elementValue']`
textarea (element)	de objeto	element: object or id of a form element that has the *value* property, like a textbox, button or password field.	Returns an `Array` with the element's name and value, like `['elementName', 'elementValue']`
select (element)	de objeto	element: object or id of a <select> element	Returns an `Array` with the element's name and all selected options' values or texts, like `['elementName', 'selOpt1 selOpt4 selOpt9']`

- **THE `ABSTRACT.TIMEDOBSERVER`**

This class is used as the base class for the other classes that will monitor one element until its value (or whatever property the derived class defines) changes. This class is used like an abstract class.

Subclasses can be created to monitor things like the input value of an element, or one of the style properties, or number of rows in a table, or whatever else you may be interested in tracking changes to.

Método	*Tipo*	*Argumentos*	*Descrição*

[ctor] (element, frequency, callback)	constructor	element: element object or id, frequency: interval in seconds, callback: function to be called when the element changes	Creates an object that will monitor the element.
getValue()	instance, abstract	(nenhum)	Derived classes have to implement this method to determine what is the current value being monitored in the element.
registerCallback()	de objeto	(nenhum)	This method is typically not called externally. It is called by the object itself to start monitoring the element.
onTimerEvent()	de objeto	(nenhum)	This method is typically not called externally. It is called by the object itself periodically to check the element.

Propriedade	***Tipo***	***Descrição***
element	Object	The element object that is being monitored.
frequency	Number	This is actually the interval in seconds between checks.
callback	Function (Object, String)	The function to be called whenever the element changes. It will receive the element object and the new value.
lastValue	String	The last value verified in the element.

- **THE `FORM.ELEMENT.OBSERVER`**

Inherits from `Abstract.TimedObserver`

Implementation of an `Abstract.TimedObserver` that monitors the value of form input elements. Use this class when you want to monitor an element that does not expose an event that reports the value changes. In that case you can use the `Form.Element.EventObserver` class instead.

Método	***Tipo***	***Argumentos***	***Descrição***
[ctor] (element, frequency, callback)	constructor	element: element object or id, frequency: interval in seconds, callback: function to be called when the element changes	Inherited from `Abstract.TimedObserver`. Creates an object that will monitor the element's `value` property.
getValue()	de objeto	(nenhum)	Returns the element's value.

- **THE `FORM.OBSERVER`**

Inherits from `Abstract.TimedObserver`

Implementation of an `Abstract.TimedObserver` that monitors any changes to any data entry element's value in a form. Use this class when you want to monitor a form that contais a elements that do not expose an event that reports the value changes. In that case you can use the `Form.EventObserver` class instead.

Método	*Tipo*	*Argumentos*	*Descrição*
[ctor] (form, frequency, callback)	constructor	form: form object or id, frequency: interval in seconds, callback: function to be called when any data entry element in the form changes	Inherited from `Abstract.TimedObserver`. Creates an object that will monitor the form for changes.
getValue()	de objeto	(nenhum)	Returns the serialization of all form's data.

- **THE `ABSTRACT.EVENTOBSERVER`**

This class is used as the base class for the other classes that execute a callback: function whenever a value-changing event happens for an element.

Multiple objects of type `Abstract.EventObserver` can be bound to the same element, without one wiping out the other. The callbacks will be executed in the order they are assigned to the element.

The triggering event is `onclick` for radio buttons and checkboxes, and `onchange` for textboxes in general and listboxes/dropdowns.

Método	*Tipo*	*Argumentos*	*Descrição*
[ctor](element, callback)	constructor	element: element object or id, callback: function to be called when the event happens	Creates an object that will monitor the element.
getValue()	instance, abstract	(nenhum)	Derived classes have to implement this method to determine what is the current value being monitored in the element.
registerCallback()	de objeto	(nenhum)	This method is typically not called externally. It is called by the object to bind itself to the element's event.

registerFormCallbacks()	de objeto	(nenhum)	This method is typically not called externally. It is called by the object to bind itself to the events of each data entry element in the form.
onElementEvent()	de objeto	(nenhum)	This method is typically not called externally. It will be bound to the element's event.

Propriedade	***Tipo***	***Descrição***
element	Object	The element object that is being monitored.
callback	Function (Object, String)	The function to be called whenever the element changes. It will receive the element object and the new value.
lastValue	String	The last value verified in the element.

- **THE `FORM.ELEMENT.EVENTOBSERVER`**

Inherits from `Abstract.EventObserver`

Implementation of an `Abstract.EventObserver` that executes a callback function to the appropriate event of the form data entry element to detect value changes in the element. If the element does not expose any event that reports changes, then you can use the `Form.Element.Observer` class instead.

Método	***Tipo***	***Argumentos***	***Descrição***
[ctor] (element, callback)	constructor	element: element object or id, callback: function to be called when the event happens	Inherited from `Abstract.EventObserver`. Creates an object that will monitor the element's `value` property.
getValue()	de objeto	(nenhum)	Returns the element's value

- **THE `FORM.EVENTOBSERVER`**

Inherits from `Abstract.EventObserver`

Implementation of an `Abstract.EventObserver` that monitors any changes to any data entry element contained in a form, using the elements' events to detect when the value changes. If the form contains elements that do not expose any event that reports changes, then you can use the `Form.Observer` class instead.

Método	***Tipo***	***Argumentos***	***Descrição***

[ctor] (form, callback)	constructor	form: form object or id, callback: function to be called when any data entry element in the form changes	Inherited from `Abstract.EventObserver`. Creates an object that will monitor the form for changes.
getValue()	de objeto	(nenhum)	Returns the serialization of all form's data.

- **THE `POSITION` OBJECT (PRELIMINARY DOCUMENTATION)**

This object provides a host of functions that help when working with element positioning.

Método	*Tipo*	*Argumentos*	*Descrição*
prepare()	de objeto	(nenhum)	Adjusts the `deltaX` and `deltaY` properties to accomodate changes in the scroll position. Remember to call this method before any calls to `withinIncludingScrolloffset` after the page scrolls.
realOffset (element)	de objeto	element: object	Returns an `Array` with the correct scroll offsets of the element, including any scroll offsets that affect the element. The resulting array is similar to `[total_scroll_left, total_scroll_top]`
cumulativeOffset (element)	de objeto	element: object	Returns an `Array` with the correct positioning offsets of the element, including any offsets that are imposed by positioned parent elements. The resulting array is similar to `[total_offset_left, total_offset_top]`
within(element, x, y)	de objeto	element: object, x and y: coordinates of a point	Tests if the given point coordinates are inside the bounding rectangle of the given element
withinIncludingScrolloffsets (element, x, y)	de objeto	element: object, x and y: coordinates of a point	

overlap(mode, element)	de objeto	mode: 'vertical' or 'horizontal', element: object	`within()` needs to be called right before calling this method. This method will return a decimal number between 0.0 and 1.0 representing the fraction of the coordinate that overlaps on the element. As an example, if the element is a square DIV with a 100px side and positioned at (300, 300), then `within(divSquare, 330, 330); overlap('vertical', divSquare);` should return 0.10, meaning that the point is at the 10% (30px) mark from the top border of the DIV.
clone(source, target)	de objeto	source: element object or id, target: element object or id	Resizes and repositions the target element identically to the source element.