

## PROTOTYPE.JS 1.4 版开发者手册(强烈推荐)

### prototype.js开发者手册

对应版本 **1.4.0**

original article by sp('Sergio Pereira') Sergio Pereira

last update: March 30th 2006

中文版: THIN

最后更新: 2006-3-31

### prototype.js 是什么?

万一你没有使用过大名鼎鼎的prototype.js, 那么让我来告诉你, prototype.js是由Sam Stephenson写的一个javascript类库。这个构思奇妙, 而且兼容标准的类库, 能帮助你轻松建立有高度互动的web2.0 特性的富客户端页面。

如果你最近尝试使用它, 你大概了解到文档并不是作者的一个强项。和在我以前使用这个类库的不少开发者一样, 一开始, 我不得不一头扎进阅读prototype.js 的源代码和实验它的功能中。我想, 在我学习完它之后, 把我学到的东西分享给大家是件不错的事。

同时, 在本文中, 我也将提供一个关于这个类库提供的objects,classes,functions,extensions这对东东的非官方参考

在阅读这个文档时, 熟悉 Ruby 的开发者将会注意到 Ruby 的一些内建类和本类库扩展实现之间非常相似。

### 相关文章

Advanced JavaScript guide.

### 一些实用的函数

这个类库带有很多预定义的对象和实用函数, 这些东东的目的显然是把你从一些重复的打字中解放出来 。

### 使用\$()方法

\$() 方法是在 DOM 中使用过于频繁的 document.getElementById() 方法的一个便利的简写, 就像这个 DOM 方法一样, 这个方法返回参数传入的 id 的那个元素。

比起 DOM 中的方法, 这个更胜一筹。你可以传入多个 id 作为参数然后 \$() 返回一个带有所有要求的元素的一个 Array 对象。

```
<HTML>
<HEAD>
<TITLE> Test Page </TITLE>
<script src="prototype-1.3.1.js"></script>
<script>
function test1()
{
var d = $('myDiv');
alert(d.innerHTML);
}

function test2()
{
var divs = $('myDiv','myOtherDiv');
for(i=0; i<divs.length; i++)
{
alert(divs[i].innerHTML);
}
}
</script>
</HEAD>
<BODY>
<div id="myDiv">
<p>This is a paragraph</p>
</div>
<div id="myOtherDiv">
<p>This is another paragraph</p>
</div>
<input type="button" value=Test1 onclick="test1();"><br>
<input type="button" value=Test2 onclick="test2();"><br>
</BODY>
</HTML>
```

另外一个好处是，这个函数能传入用 **string** 表示的对象 ID，也可以传入对象本身，这样，在建立其它能传两种类型的参数的函数时非常有用。

## 使用\$F()函数

\$F()函数是另一个大受欢迎的“快捷键”，它能用于返回任何表单输入控件的值，比如 text box,drop-down list。这个方法也能用元素 id 或元素本身做为参数。

```
<script>

function test3()

{

alert(  $F('userName')  );

}

</script>

<input type="text" id="userName" value="Joe Doe"><br>

<input type="button" value=Test3 onclick="test3();"><br>
```

## 使用\$A()函数

\$A()函数能把它接收到的单个的参数转换成一个 Array 对象。

这个方法，结合被本类库扩展了的 Array 类，能方便的把任何的可枚举列表转换成或拷贝到一个 Array 对象。一个推荐的用法就是把 DOM Node Lists 转换成一个普通的 Array 对象，从而更有效率的进行遍历，请看下面的例子。

```
<script>

function showOptions() {

var someNodeList = $('lstEmployees').getElementsByTagName('option');

var nodes = $A(someNodeList);

nodes.each(function(node) {

alert(node.nodeName + '：' + node.innerHTML);

});

}

</script>

<select id="lstEmployees" 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="Show the options" onclick="showOptions();" >
```

## 使用 **\$H()** 函数

**\$H()**函数把一些对象转换成一个可枚举的和联合数组类似的 **Hash** 对象。

```
<script>

function testHash()
{
//let's create the object
var a = {
first: 10,
second: 20,
third: 30
};

//now transform it into a hash
var h = $H(a);

alert(h.toString()); //displays: first=10&second=20&third=30
}

</script>
```

## 使用**\$R()**函数

**\$R()**是 `new ObjectRange(lowBound,upperBound,excludeBounds)`的缩写。

跳到ObjectRange 类文档可以看到一个关于此类的完整描述。此时，我们还是先来看一个例子以展示这个缩写能代替哪些方法吧。其它相关的一些知识可以在Enumerable 对象文档中找到。

```
<script>

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

</script>
```

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

## 使用Try.these()函数

Try.these() 方法使得实现当你想调用不同的方法直到其中的一个成功正常的这种需求变得非常容易， 他把一系列的方法作为参数并且按顺序的一个一个的执行这些方法直到其中的一个成功执行，返回成功执行的那个方法的返回值。

在下面的例子中， `xmlNode.text` 在一些浏览器中好用，但是`xmlNode.textContent`在另一些浏览器中正常工作。 使用Try.these() 方法我们可以得到正常工作的那个方法的返回值。

```
<script>

function getXmlNodeValue(xmlNode){

    return Try.these(

        function() {return xmlNode.text;},

        function() {return xmlNode.textContent;}

    );

}

</script>
```

## Ajax对象

上面提到的共通方法非常好，但是面对它吧，它们不是最高级的那类东西。它们是吗？你很可能自己编写了这些甚至在你的脚本里面有类似功能的方法。但是这些方法只是冰山一角。

我很肯定你对 prototype.js 感兴趣的原因很可能是由于它的 AJAX 能力。所以让我们解释当你需要完成 AJAX 逻辑的时候，这个包如何让它更容易。

Ajax 对象是一个预定义对象，由这个包创建，为了封装和简化编写AJAX 功能涉及的狡猾的代码。 这个对象包含一系列的封装AJAX逻辑的类。我们来看看其中几个类。

## 使用Ajax.Request类

如果你不使用任何的帮助程序包，你很可能编写了整个大量的代码来创建XMLHttpRequest对象并且异步的跟踪它的进程， 然后解析出响应 然后处理它。当你不需要支持多于一种类型的浏览器时你会感到非常的幸运。

为了支持 AJAX 功能。这个包定义了 Ajax.Request 类。

假如你有一个应用程序可以通过url `http://yoursever/app/get_sales?empID=1234&year=1998`与服务器通信。它返回下面这样的XML 响应。

```

<?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>

```

用 Ajax.Request对象和服务端通信并且得到这段XML是非常简单的。下面的例子演示了它是如何完成的。

```

<script>
function searchSales()
{
var empID = $F('lstEmployees');
var y = $F('lstYears');
var url = 'http://yoursever/app/get_sales';
var pars = 'empID=' + empID + '&year=' + y;

var myAjax = new Ajax.Request(
url,
{
method: 'get',
parameters: pars,
onComplete: showResponse

```

```

});

}

function showResponse(originalRequest)
{
//put returned XML in the textarea
$('#result').value = originalRequest.responseText;
}
</script>
<select id="lstEmployees" size="10" onchange="searchSales()">
<option value="5">Buchanan, Steven</option>
<option value="8">Callahan, Laura</option>
<option value="1">Davolio, Nancy</option>
</select>
<select id="lstYears" size="3" onchange="searchSales()">
<option selected="selected" value="1996">1996</option>
<option value="1997">1997</option>
<option value="1998">1998</option>
</select>
<br><textarea id=result cols=60 rows=10 ></textarea>

```

你注意到传入 `Ajax.Request` 构造方法的第二个对象了吗？ 参数 `{method: 'get', parameters: pars, onComplete: showResponse}` 表示一个匿名对象的真实写法。他表示你传入的这个对象有一个名为 `method` 值为 `'get'` 的属性， 另一个属性名为 `parameters` 包含HTTP请求的查询字符串， 和一个 `onComplete` 属性/方法包含函数 `showResponse`。

还有一些其它的属性可以在这个对象里面定义和设置，如 `asynchronous`，可以为 `true` 或 `false` 来决定AJAX对服务器的调用是否是异步的（默认值是 `true`）。

这个参数定义AJAX调用的选项。在我们的例子中， 在第一个参数通过HTTP GET命令请求那个url， 传入了变量 `pars` 包含的查询字符串， `Ajax.Request` 对象在它完成接收响应的时候将调用 `showResponse` 方法。

也许你知道， `XMLHttpRequest` 在HTTP请求期间将报告进度情况。这个进度被描述为四个不同阶段：*Loading*, *Loaded*, *Interactive*, 或 *Complete*。你可以使 `Ajax.Request` 对象在任何阶段调用自定义方法， *Complete* 是最常用的一个。想调用自定义的方法只需要简单的在请求的选项参数中的名为 `onXXXXX` 属性/方法中提供自定义的方法对象。 就像我们例子中的 `onComplete`。你传入的方法将会被用一个参数调用，这个参数是 `XMLHttpRequest` 对象自己。你将会用这个对象去得到返回的数据并且或许检查包含有在这次调用中的HTTP结果代码的 `status` 属性。

还有另外两个有用的选项用来处理结果。我们可以在onSuccess 选项处传入一个方法，当AJAX无误的执行完后调用，相反的，也可以在onFailure 选项处传入一个方法，当服务器端出现错误时调用。正如onXXXXX 选项传入的方法一样，这两个在被调用的时候也传入一个带有AJAX请求的XMLHttpRequest对象。

我们的例子没有用任何有趣的方式处理这个 XML 响应， 我们只是把这段 XML 放进了一个文本域里面。对这个响应的一个典型的应用很可能就是找到其中的想要的信息，然后更新页面中的某些元素， 或者甚至可能做某些 XSLT 转换而在页面中产生一些 HTML。

在 1.4.0 版本中，一种新的事件回传外理被引入。如果你有一段代码总是要为一个特殊的事件执行，而不管是哪个AJAX调用引发它，那么你可以使用新的Ajax.Responders对象。

假设你想要在一个 AJAX 调用正在运行时，显示一些提示效果，像一个不断转动的图标之类的，你可以使用两个全局事件 Handler 来做到，其中一个在一个调用开始时显示图标，另一个在最后一个调用完成时隐藏图标。看下面的例子。

```
<script>

var myGlobalHandlers = {

onCreate: function() {

Element.show('systemWorking');

},

onComplete: function() {

if(Ajax.activeRequestCount == 0) {

Element.hide('systemWorking');

}

}

};

Ajax.Responders.register(myGlobalHandlers);

</script>

<div id='systemWorking'><img src='spinner.gif'>Loading...</div>
```

更完全的解释，请参照 Ajax.Request 参考 和 Ajax选项参考。

## 使用Ajax.Updater类

如果你的服务器的另一端返回的信息已经是HTML了，那么使用这个程序包中 Ajax.Updater 类将使你的生活变得更加得容易。用它你只需提供哪一个元素需要被AJAX请求返回的HTML填充就可以了，例子比我写说明的更清楚。

```
<script>
```



```

function getHTML()
{
var url = 'http://yourserver/app/getSomeHTML';
var pars = 'someParameter=ABC';

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

}
</script>

<input type=button value=GetHtml onclick="getHTML()">

<div id="placeholder"></div>

```

你可以看到，这段代码比前面的例子更加简洁，不包括 `onComplete` 方法，但是在构造方法中传入了一个元素id。我们来稍稍修改一下代码来描述如何在客户端处理服务器段错误成为可能。

我们将加入更多的选项，指定处理错误的一个方法。这个是用 `onFailure` 选项来完成的。我们也指定了一个 `placeholder` 只有在成功请求之后才会被填充。为了完成这个目的我们修改了第一个参数从一个简单的元素id到一个带有两个属性的对象，`success`（一切OK的时候被用到）和 `failure`（有地方出问题的时候被用到）在下面的例子中没有用到`failure`属性，而仅仅在 `onFailure` 处使用了 `reportError` 方法。

```

<script>

function getHTML()
{
var url = 'http://yourserver/app/getSomeHTML';
var pars = 'someParameter=ABC';

var myAjax = new Ajax.Updater(
{success: 'placeholder'},
url,

```

```

{
method: 'get',
parameters: pars,
onFailure: reportError
});

}

function reportError(request)
{
alert('Sorry. There was an error.');
```

如果你的服务器逻辑是连同HTML 标记返回JavaScript 代码， Ajax.Updater对象可以执行那段JavaScript代码。为了使这个对象对待响应为JavaScript，你只需在最后参数的对象构造方法中简单加入evalScripts: true属性。但是值得提醒的是，像这个选项名evalScripts暗示的，这些脚本会被执行，但是它们不会被加入到Page的脚本中。“有什么区别？”，可能你会这样问。我们假定请求地址返回的东东像这样：

```

<script language="javascript" type="text/javascript">
function sayHi() {
alert('Hi');
}
</script>
<input type=button value="Click Me" onclick="sayHi()">
```

如果你以前这样尝试过，你知道这些脚本不会如你所期望的那样工作，原因是这段脚本会被执行，但像上面这样的脚本执行并不会创建一个名叫 sayHi 的函数，它什么也不做。如果要创建一个函数，我们应当把代码改成下面这个样子：

```

<script language="javascript" type="text/javascript">
sayHi = function() {
alert('Hi');
};

</script>
```

```
<input type=button value="Click Me" onclick="sayHi()">
```

为什么我们在上面的代码中不使用 `var` 关键字来声明这个变量呢（指 `sayHi`），因为那样做创建出来的函数将只是当前脚本块的一个局部变量（至少在 IE 中是这样）。不写 `var` 关键字，创建出来的对象的作用域就是我们所期望的 `window`。

更多相关知识，请参看 [Ajax.Updater reference](#) 和 [options reference](#)。

## 枚举... 噢!噢!

你知道，我们都是这样来做循环的，建一个 `Array`，用 `elements` 组织它们，再建一个循环结构（例如 `for,foreach,while`）通过 `index` 数字来访问每一个 `element`，再用这个 `element` 做一些动作。

当你想到这时，你会发现几乎每次写循环代码你都会迟早用到一个 `Array`。那么，如果 `Array` 对象能够提供更多的功能给它们的迭代器使用不是很爽吗？

确实是这样，事实上很多的编程语言都在它们的 `Array` 或其它类似的结构中（如 `Collections,Lists`）提供一些这样的功能。

现在好了，`prototype.js`给了我们一个 `Enumerable`对象，它实现了很多和可迭代数据进行交互的窍门。和原有的JS对象相比`prototype.js`更上一层楼，它对`Array` 类s扩展了所有枚举要用的函数。

## 循环, Ruby 样式的

在标准的 `javascript` 中，如果你想把一个 `array` 中的所有 `elements` 显示出来，你可以像下面代码这样写得很好：

```
<script>

function showList() {

var simpsons = ['Homer', 'Marge', 'Lisa', 'Bart', 'Meg'];

    for(i=0;i<simpsons.length;i++) {

alert(simpsons[i]);

    }

}

</script>

<input type="button" value="Show List" onclick="showList();" >
```

使用我们新的最好的朋友，`prototype.js`，我们可以把它生写成这样

```
function showList() {

var simpsons = ['Homer', 'Marge', 'Lisa', 'Bart', 'Meg'];
```

```
        simpsons.each( function(familyMember) {  
alert(familyMember);  
});  
  
}
```

你可能会想“非常奇怪的方式...相对旧的，这种语法太怪异了”。哦，在上面的例子，确实什么也没有，在这个简单得要死例子中，也没有改变太多啊，尽管如此，请继续读下去。

在继续下面内容之前，你注意到那个被做为一个参数传递给 `each` 函数的函数？我们把它理解成迭代器函数。

## Your arrays on steroids

就如我们上面提到的，把你的 `Array` 中的 `elements` 当成相同的类型使用相同的属性和函数是很通用(Common,不知该翻译成通用还是庸俗)的。让我们看看怎么样利用我们新的马力强劲的 `Arrays` 的迭代功能吧。

依照标准找到一个 `element`。

```
<script>  
  
function findEmployeeById(emp_id) {  
var listBox = $('lstEmployees')  
var options = listBox.getElementsByTagName('option');  
options = $A(options);  
var opt = options.find( function(employee) {  
return (employee.value == emp_id);  
});  
alert(opt.innerHTML); //displays the employee name  
}  
  
</script>  
  
<select id="lstEmployees" 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="Find Laura" onclick="findEmployeeById(8);" >
```

现在我们在下一城，看看如何过滤一个 **Array** 中的元素，从每个元素中得到我们想要的成员。

```
<script>

function showLocalLinks(paragraph) {
  paragraph = $(paragraph);
  var links = $A(paragraph.getElementsByTagName('a'));
  //find links that do not start with 'http'
  var localLinks = links.findAll( function(link) {
    var start = link.href.substring(0,4);
    return start != 'http';
  });
  //now the link texts
  var texts = localLinks.pluck('innerHTML');
  //get them in a single string
  var result = texts.inspect();
  alert(result);
}

</script>

<p id="someText">

This <a href="http://othersite.com/page.html">text</a> has
a <a href="#localAnchor">lot</a> of
<a href="#otherAnchor">links</a>. Some are
<a href="http://wherever.com/page.html">external</a>
and some are <a href="#someAnchor">local</a>

</p>

<input type=button value="Find Local Links" onclick="showLocalLinks('someText')">
```

上面的代码仅仅是一点小小的实践让人爱上这种语法。请参看 [Enumerable](#)和[Array](#)的所有函数

# JavaScript 类扩展

prototype.js 类库实现强大功能的一种途径是扩展已有的 JavaScript 类。

## 对 Object 的扩展

Method	Kind	Arguments	Description
extend(destination, source)	static	destination: any object, source: any object	提供一种通过拷贝所有源以象属性和函数到目标函数实现继承的方法
inspect(targetObj)	static	targetObj: any object	返回可读性好关于目标对象的文字描述，如果对象实例没有定义一个 inspect 函数，默认返回 toString 函数的值。

## 对Number的扩展

Method	Kind	Arguments	Description
toColorPart()	instance	(none)	返回数字的十六进制表示形式。在把一个 RGB 数字转换成 HTML 表现形式时很有用。
succ()	instance	(none)	返回下一个数字，这个方法可用于迭代调用场景中。
times(iterator)	instance	iterator: a function object conforming to Function(index)	Calls the iterator function repeatedly passing the current index in the index argument. 反复调用iterator函数并传递当前index到iterator的index参数。

下面的例子用提示框显示 0-9。

```
<script>
function demoTimes() {
var n = 10;
n.times(function(index) {
alert(index);
});
/*****
* you could have also used:
*      (10).times( .... );
*****/
}
```

```
</script>
<input type=button value="Test Number.times()" onclick="demoTimes()">
```

对 **Function**扩展

Method	Kind	Arguments	Description
bind(object)	instance	object: the object that owns the method	返回 function 的实例，这个实例和源 function 的结构一样，但是它已被绑定给了参数中提供的 object，就是说，function 中的 this 指针指向参数 object。
bindAsEventListener(object)	instance	object: the object that owns the method	用法和上面的 bind 一样，区别在于用来绑定事件。

让我们看看如何运用这些扩展。

```
<input type=checkbox id=myChk value=1> Test?
<script>
//declaring the class
var CheckboxWatcher = Class.create();
//defining the rest of the class implementation
CheckboxWatcher.prototype = {
  initialize: function(chkBox, message) {
    this.chkBox = $(chkBox);
    this.message = message;
    //assigning our method to the event

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

  },
  showMessage: function(evt) {
    alert(this.message + ' (' + evt.type + ')');
  }
};
```

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

对String的扩展

Method	Kind	Arguments	Description
stripTags()	instance	(none)	返回一个把所有的 HTML 或 XML 标记都移除的字符串。
stripScripts()	instance	(none)	返回一个把所有的 script 都移除的字符串。
escapeHTML()	instance	(none)	返回一个把所有的 HTML 标记合适的转义掉的字符串。
unescapeHTML()	instance	(none)	escapeHTML()的反转。
extractScripts()	instance	(none)	返回一个包含在 string 中找到的所有<script>的数组。
evalScripts()	instance	(none)	执行在 string 中找到的所有<script>。
toQueryParams()	instance	(none)	把 querystring 分割成一个用 parameter name 做 index 的联合 Array，更像一个 hash。
parseQuery()	instance	(none)	和toQueryParams()一样。
toArray()	instance	(none)	把字符串转换成字符数组。
camelize()	instance	(none)	转换一个以连字符连接的字符串成一个骆驼法样式的字符串。比如，这个函数在写代码时，把它做为一个样式工具使用是很有用的。

对 Array的扩展

因为 array 扩展于 enumerable，所以所有 enumerable 对象的函数，array 都是可以使用的，除此之外，下面的这些也是已经实现了的。

Method	Kind	Arguments	Description
clear()	instance	(none)	清空。
compact()	instance	(none)	返回一个不包括源 array 中 null 或 undefined 元素的 array,此方法不改变源 array。
first()	instance	(none)	返回 array 的第一个对象。
flatten()	instance	(none)	通过递归组合 array 每个元素的子元素（如果该元素也是 array)来返回一个“扁平的”一维的 array。
indexOf(value)	instance	value: what you are looking for.	返回给出数字位置（从 0 算起）的元素，如果在该位置没有找到对象，返回-1。
inspect()	instance	(none)	重载 inspect(),返回更好格式的反映 Array 每个元素



			的字符描述。
last()	instance:	(none)	返回最后一个元素。
reverse([applyToSelf])	instance:	array itself should also be reversed.	反转 Array 中元素的顺序，如果没有给出参数，或参数为 true，则源 Array 中元素的顺序也反转，否则源 Array 保持不变。
shift()	instance:	(none)	返回 Array 的第一个元素并从 Array 中移除它，Array 的 Length-1。
without(value1 [, value2 [, ... valueN]])	instance:	value1 ... valueN: values to be excluded if present in the array.	返回一个把参数列表中包含的元素从源 Array 中排除的 Array。

document DOM扩展

Method	Kind	Arguments	Description
getElementsByClassName(className [, parentElement])	instance:	className: name of a CSS class associated with the elements, parentElement: object or id of the element that contains the elements being retrieved.	返回所有 CSS className 属性等于 className 参数的元素，如果没有给出 parentElement,那么将搜索 document body。(此处使用 document.body 我觉得不如使用 document,因为有时有的页面没有 body)

Event扩展

Property	Type	Description
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.

Method	Kind	Arguments	Description
element(event)	static	event: an Event object	返回事件源对象。
isLeftClick(event)	static	event: an Event object	如果点击了鼠标左键, 返回 true.
pointerX(event)	static	event: an Event object	返回鼠标的 X 坐标。
pointerY(event)	static	event: an Event object	返回鼠标的 Y 坐标。
stop(event)	static	event: an Event object	使用此函数来中断事件的默认行为并阻止传递（冒泡）。
findElement(event, tagName)	static	event: an Event object, tagName: name of the desired tag.	从事件源对象开始向上搜索 DOM 树, 直到找到第一个符合 tagName 的元素
observe(element, name, observer, useCapture)	static	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 <i>capture</i> phase and if false in the <i>bubbling</i> phase.	为对象的某个事件增加一个处理函数。
stopObserving(element, name, observer, useCapture)	static	element: object or id, name: event name (like 'click'), observer: function that is handling the event, useCapture: if true handles the event in the <i>capture</i> phase and if false in the <i>bubbling</i> phase.	和上面的函数相反。
_observeAndCache(element, name, observer, useCapture)	static		私有函数，别管它。
unloadCache()	static	(none)	私有函数，别管它。从内存中清除所有的 observers 缓存。

下面代码演示如何给 window 添加一个 load 事件处理函数。

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

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

在 **prototype.js** 中定义新的对象和类

另一个这个程序包帮助你的地方就是提供许多既支持面向对象设计理念又有共通功能的许多对象。

The **PeriodicalExecuter** object

这个对象提供一定间隔时间上重复调用一个方法的逻辑。

Method	Kind	Arguments	Description
[ctor](callback, interval)	constructor	callback: a parameterless function, interval: number of seconds	创建这个对象的实例将会重复调用给定的方法。

Property	Type	Description
callback	Function()	被调用的方法，该方法不能传入参数。
frequency	Number	以秒为单位的间隔。
currentlyExecuting	Boolean	表示这个方法是否正在执行。

The **Prototype** object

Prototype 没有太重要的作用，只是声明了该程序包的版本 。

Property	Type	Description
Version	String	版本。
emptyFunction	Function()	空函数。
K	Function(obj)	一个仅仅回传参数的函数。
ScriptFragment	String	识别 script 的正则式。

The **Enumerable** object

Enumerable 对象能够以更优雅的方式实现对列表样式的结构进行枚举。

很多其它的对象通过扩展自 Enumerable 对象来得到这些有用的接口。

Method	Kind	Arguments	Description
each(iterator)	instance	iterator: a function object conforming to Function(value, index)	把每个 element 做为第一个参数, element 的 index 作为第一个参数调用 iterator 函数。
all([iterator])	instance	iterator: a function object conforming to Function(value, index)	这个函数会用给出的 iterator 测试整个集合, 如果集合中任一元素在 iterator 函数测试中返回 false 或 null, 那么这个函数返回 false, 否则返回 true。如果没有给出 iterator, 那么就会测试所有的元素是不是不等于 false 和 null。你可以简单的把它看成是“检测每个元素都为非空非负”。
any(iterator)	instance	iterator: a function object conforming to Function(value, index), optional.	这个函数会用给出的 iterator 测试整个集合, 如果集合中任一元素在 iterator 函数测试中返回 true, 那么这个函数返回 true, 否则返回 false。如果没有给出 iterator, 那么就会测试所有的元素是不是有一个不等于 false 和 null。你可以简单的把它看成是“检测元素中是不是有非空非负的”。
collect(iterator)	instance	iterator: a function object conforming to Function(value, index)	调用 iterator 函数根据集合中每个元素返回一个结果, 然后按照原来集合中的顺序, 返回一个 Array。
detect(iterator)	instance	iterator: a function object conforming to Function(value, index)	集合中每个元素调用一次 Iterator, 返回第一个使 Iterator 返回 True 的元素, 如果最终都没有为 true 的调用, 那么返回 null。
entries()	instance	(none)	等于 toArray()。
find(iterator)	instance	iterator: a function object conforming to Function(value, index)	等于 detect()。
findAll(iterator)	instance	iterator: a function object conforming to Function(value, index)	集合中每个元素调用 Iterator, 返回一个由所有调用 Iterator 返回结果等于 true 的元素组成的数组。和 reject() 相反。
grep(pattern [, iterator])	instance	pattern: a RegExp object used to match the elements, iterator: a function object conforming to Function(value, index)	用 pattern 参数正则表达式测试集合中的每个元素, 返回一个包含所有匹配正则式的元素的 Array, 如果给出了 Iterator, 那个每个结果还要经过一下 Iterator 处理。

include(obj)	instance	obj: any object	判断集合中包不包含指定对象。
inject(initialValue, iterator)	instance	initialValue: any object to be used as the initial value, iterator: a function object conforming to Function(accumulator, value, index)	用 Iterator 联接所有集合中的元素。Iterator 在被调用时把上一次迭代的结果做为第一个参数传给 accumulator。第一次迭代时, accumulator 等于 initialValue,最后返回 accumulator 的值。
invoke(methodName [, arg1 [, arg2 [...]]])	instance	methodName: name of the method that will be called in each element, arg1..argN: arguments that will be passed in the method invocation.	集合中的每个元素调用指定的函数（查看源代码可以发现指定函数被调用时, this 指针被传成当前元素），并传入给出的参数，返回调用结果组成的 Array。
map(iterator)	instance	iterator: a function object conforming to Function(value, index)	同 collect()。
max([iterator])	instance	iterator: a function object conforming to Function(value, index)	返回集合中元素的最大值, 或调用 Iterator 后返回值的最大值(如果给出了 Iterator 的话)。
member(obj)	instance	obj: any object	同 include()。
min([iterator])	instance	iterator: a function object conforming to Function(value, index)	返回最小值, 参见 max()。
partition([iterator])	instance	iterator: a function object conforming to Function(value, index)	返回一个包含两个 Array 的 Array, 第一个 Array 包含所有调用 Iterator 返回 True 的元素, 第二个 Array 包含剩下的元素。如果 Iterator 没有给出, 那么就根据元素本身判断。
pluck(propertyName)	instance	propertyName name of the property that will be read from each element. This can also contain the index of the element	返回每个元素的指定属性名的属性的值组成的 Array。
reject(iterator)	instance	iterator: a function object conforming to Function(value,	和 findAll()相反（返回所有等于false的元素）。

		index)	
select(iterator)	instance	iterator: a function object conforming to Function(value, index)	同 findAll()。
sortBy(iterator)	instance	iterator: a function object conforming to Function(value, index)	根据每个元素调用 Iterator 返回的值进行排序返回一个 Array。
toArray()	instance	(none)	返回由集合所有元素组成的一个 Array。
zip(collection1[, collection2 [, ... collectionN [,transform]]])	instance	collection1 .. collectionN: enumerations that will be merged, transform: a function object conforming to Function(value, index)	合并每个给出的集合到当前集合。合并操作返回一个新的 array，这个 array 的元素个数和原集合的元素个数一样，这个 array 的每个元素又是一个子 array，它合并了所有集合中相同 index 的元素。如果 transform 函数被指定，那么 array 的每个元素还会调用 transform 函数先做处理。 举个例子: [1,2,3].zip([4,5,6], [7,8,9]).inspect() 返回 "[ [1,4,7],[2,5,8],[3,6,9] ]"

## The Hash object

Hash 对象实现一种 Hash 结构，也就是一个 Key:Value 对的集合。

Hash 中的每个 Item 是一个有两个元素的 array，前一个是 Key，后一个是 Value，每个 Item 也有两个不需加以说明的属性,key 和 value。

Method	Kind	Arguments	Description
keys()	instance	(none)	返回所有 Item 的 key 的集合的一个 array。
values()	instance	(none)	返回所有 Item 的 value 的集合的一个 array。
merge(otherHash)	instance	otherHash: Hash object	合并给出的 Hash，返回一个新 Hash。
toQueryString()	instance	(none)	以QueryString那样的样式返回hash中所有的item，例如： 'key1=value1&key2=value2&key3=value3'
inspect()	instance	(none)	用一种合适的方法显示 hash 中的 key:value 对。

## The ObjectRange class

继承自 *Enumerable*

用上、下边界描述一个对象区域。

Property	Type	Kind	Description
start	(any)	instance	range 的下边界
end	(any)	instance	range 的上边界
exclusive	Boolean	instance	决定边界自身是不是 range 的一部分。

Method	Kind	Arguments	Description
[ctor](start, end, exclusive)	constructor	start: the lower bound, end: the upper bound, exclusive: include the bounds in the range?	创建一个 range 对象，从 start 生成到 end,这里要注意的是，start 和 end 必段类型一致，而且该类型要有 succ()方法。
include(searchedValue)	instance	searchedValue: value that we are looking for	检查一个 value 是不是在 range 中。

## The Class object

在这个程序包中 Class 对象在声明其他的类时候被用到 。用这个对象声明类使得新类支持 initialize() 方法，他起构造方法的作用。

看下面的例子

```
//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
```

Method	Kind	Arguments	Description
create(*)	instance	(any)	定义新类的构造方法。

The Ajax object

这个对象被用作其他提供 AJAX 功能的类的根对象。

Property	Type	Kind	Description
activeRequestCount	Number	instance	正在处理中的 Ajax 请求的个数。

Method	Kind	Arguments	Description
getTransport()	instance	(none)	返回新的XMLHttpRequest 对象。

The Ajax.Responders object

继承自 Enumerable

这个对象维持一个在 Ajax 相关事件发生时将被调用的对象的列表。比如，你要设置一个全局钩子来处理 Ajax 操作异常，那么你就可以使用这个对象。

Property	Type	Kind	Description
responders	Array	instance	被注册到 Ajax 事件通知的对象列表。

Method	Kind	Arguments	Description
register(responderToAdd)	instance	responderToAdd: object with methods that will be called.	被传入参数的对象应包含名如 Ajax 事件的系列方法（如 onCreate,onComplete,onException）。通讯事件引发所有被注册的对象合适名称的函数被调用。
unregister(responderToRemove)	instance	responderToRemove: object to be removed from the list.	从列表中移除。
dispatch(callback, request, transport, json)	instance	callback: name of the AJAX event being reported, request: the Ajax.Request	遍历被注册的对象列表，找出有由 callback 参数决定的那个函数的对象。然后向这些函数传递其它的三个参数，如果 Ajax 响应中包含一个含有 JSON 内容的



		object responsible for the event, transport: the XMLHttpRequest object that carried (or is carrying) the AJAX call, json: the X-JSON header of the response (if present)	X-JSON HTTP 头,那么它会被热行并传入 json 参数。 如果事件是 onException,那么 transport 参数会被异常代替, json 不会传递。
--	--	--	--

The Ajax.Base class

这个类是其他在 Ajax 对象中定义的类的基类。

Method	Kind	Arguments	Description
setOptions(options)	instance	options: AJAX options	设定 AJAX 操作想要的选项。
responseIsSuccess()	instance	(none)	返回 true 如果AJAX操作成功,否则为 false 。
responseIsFailure()	instance	(none)	与 responseIsSuccess() 相反。

The Ajax.Request class

继承自 Ajax.Base

封装 AJAX 操作

Property	Type	Kind	Description
Events	Array	static	在AJAX操作中所有可能报告的事件/状态的列表。这个列表包括: 'Uninitialized', 'Loading', 'Loaded', 'Interactive', 和 'Complete'。
transport	XMLHttpRequest	instance	承载AJAX操作的 XMLHttpRequest 对象。
url	string	instance	请求的 URL。

Method	Kind	Arguments	Description
[ctor](url, options)	constructor	url: the url to be fetched, options: AJAX options	创建这个对象的一个实例，它将在给定的选项下请求url。 onCreate事件在调用constructor时被激发。 <b>重要：</b> 如果选择的url受到浏览器的安全设置，他会一点作用也不起。 很多情况下，浏览器不会请求与当前页面不同主机(域名)的url。你最好只使用本地url来避免限制用户配置他们的浏览器(谢

			谢Clay)
evalJSON()	instance	(none)	这个方法显然不会被外部调用。它在 Ajax 响应中含有 X-JSON HTTP 头时用于内部调用执行这些内容。
evalReponse()	instance	(none)	这也方法显然不会被外部调用，如果 Ajax 响应含有一个值为 text/javascript 的 Content-Type 头，那么这个方法就用被调用执行响应体。
header(name)	instance	name: HTTP header name	引用 Ajax 响应的头的内容，在 Ajax 访问结束后再调用这个方法。
onStateChange()	instance	(none)	这个方法通常不会被外部调用。 当 AJAX 请求状态改变的时候被这个对象自己调用。
request(url)	instance	url: url for the AJAX call	这个方法通常不会被外部调用。已经在构造方法中调用了。
respondToReadyState(readyState)	instance	readyState: state number (1 to 4)	这个方法通常不会被外部调用。 当 AJAX 请求状态改变的时候被这个对象自己调用。
setRequestHeaders()	instance	(none)	这个方法通常不会被外部调用。 被这个对象自己调用来配置在 HTTP 请求要发送的 HTTP 报头。

## 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.

Property	Type	Default	Description
method	String	'post'	HTTP 请求方式。
parameters	String	''	在 HTTP 请求中传入的 url 格式的值列表。
asynchronous	Boolean	true	指定是否做异步 AJAX 请求。
postBody	String	undefined	在 HTTP POST 的情况下，传入请求体中的内容。
requestHeaders	Array	undefined	和请求一起被传入的HTTP头部列表， 这个列表必须含有偶数个项目，任何奇数项目是自定义的头部的名称， 接下来的偶数项目使这个头部项目的字符串值。 例子: ['my-header1', 'this is the value', 'my-other-header', 'another value']

onXXXXXXX	Function(XMLHttpRequest, Object)	undefined	在AJAX请求中,当相应的事件/状态形成的时候调用的自定义方法。例如 var myOpts = {onComplete: showResponse, onLoaded: registerLoaded};. 这个方法将被传入一个参数,这个参数是承载AJAX操作的 XMLHttpRequest 对象,另一个是包含被执行X-JSON响应HTTP头。
onSuccess	Function(XMLHttpRequest, Object)	undefined	当AJAX请求成功完成的时候调用的自定义方法。这个方法将被传入一个参数,这个参数是承载AJAX操作的 XMLHttpRequest 对象,另一个是包含被执行X-JSON响应HTTP头。
onFailure	Function(XMLHttpRequest, Object)	undefined	当AJAX请求完成但出现错误的时候调用的自定义方法。这个方法将被传入一个参数,这个参数是承载AJAX操作的 XMLHttpRequest 对象,另一个是包含被执行X-JSON响应HTTP头。
onException	Function(Ajax.Request, exception)	undefined	当一个在客户端执行的 Ajax 发生像无效响应或无效参数这样的异常情况时被调用的自定义函数。它收到两个参数,包含异常 Ajax 操作的 Ajax.Request 对象和异常对象。
insertion	an Insertion class	undefined	一个能决定怎么样插入新内容的类,能 Insertion.Before, Insertion.Top, Insertion.Bottom, 或 Insertion.After. 只能应用于Ajax.Updater 对象.
evalScripts	Boolean	undefined, false	决定当响应到达的时候是否执行其中的脚本块,只在 Ajax.Updater 对象中应用。
decay	Number	undefined, 1	决定当最后一次响应和前一次响应相同时在 Ajax.PeriodicalUpdater 对象中的减慢访问的次数, 例如,如果设为 2,后来的刷新和之前的结果一样,这个对象将等待 2 个设定的时间间隔进行下一次刷新,如果又一次一样,那么将等待 4 次,等等。不设定这个只,或者设置为 1,将避免访问频率变慢。
frequency	Number	undefined, 2	用秒表示的刷新间的间隔,只能应用于 Ajax.PeriodicalUpdater 对象。

## The Ajax.Updater class

继承自 *Ajax.Request*

当请求的url返回一段HTML而你想把它直接放置到页面中一个特定的元素的时候被用到。如果url的返回<script> 的块并且想在接收到时就执行它的时候也可以使用该对象。含有脚本的时候使用 evalScripts 选项。

Property	Type	Kind	Description	
containers	Object	instance	这个对象包含两个属性:AJAX请求成功执行的时候用到 containers.success , 否则的话用到 containers.failure 。	
Method	Kind	Arguments	Description	
[ctor](container, url, options)	constructor	container: this can be the id of an element, the element object itself, or an object with two properties - <b>object.success</b> element (or id) that will be used when the AJAX call succeeds, and <b>object.failure</b> element (or id) that will be used otherwise. url: the url to be fetched, options: AJAX options	创建一个用给定的选项请求给定的 url 的一个实例。	
updateContent()	instance	(none)	这个方法通常不会被外部调用。当响应到达的时候, 被这个对象自己调用。 它 会用HTML更新适当的元素或者调用在 insertion 选项中传入的方法-这个方法将被传入两个参数, 被更新的元素和响应文本。	

## The Ajax.PeriodicalUpdater class

继承自Ajax.Base

这个类重复生成并使用 Ajax.Updater 对象来刷新页面中的一个元素。或者执行 Ajax.Updater 可以执行的其它任务。更多信息参照 Ajax.Updater 参考 。

Property	Type	Kind	Description	
container	Object	instance	这个值将直接传入Ajax.Updater的构造方法。	
url	String	instance	这个值将直接传入Ajax.Updater的构造方法。	
frequency	Number	instance	两次刷新之间的间隔（不是频率），以秒为单位。默认 2 秒。This 当调用 Ajax.Updater 对象的时候，这个数将和当前的 decay 相乘。	
decay	Number	instance	重负执行任务的时候保持的衰败水平。	
updater	Ajax.Updater	instance	最后一次使用的 Ajax.Updater 对象	
timer	Object	instance	通知对象该下一次更新时用到的 JavaScript 计时器。	

Method	Kind	Arguments	Description
[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	创建一个用给定的选项请求给定的 url 的一个实例。
start()	instance	(none)	这个方法通常不会被外部调用。对象为了开始周期性执行任务的时候调用的方法。
stop()	instance	(none)	使对象停止执行周期任务。停止后，如果有 onComplete 选项，那么引发 callback。
updateComplete()	instance	(none)	这个方法通常不会被外部调用。被当前的 Ajax.Updater 使用，当一次请求结束的时候，它被用作计划下一次请求。
onTimerEvent()	instance	(none)	这个方法通常不会被外部调用。当到下一次更新时被内部调用。

## The Element object

这个对象提供在操作 DOM 中元素时使用的功能性方法。

Method	Kind	Arguments	Description
addClassName(element, className)	instance	element: element object or id, className: name of a CSS class	将给出的 className 添加到对象的 className 属性中。
classNames(element)	instance	element: element object or id	返回一个 Element.ClassName 的对象表示 CSS 给出对象有的 class names。
cleanWhitespace(element)	instance	element: element object or id	清除对象子元素中所有空白的 text node。
empty(element)	instance	element: element object or id	返回一个布尔值指示对象为空或只有空白字符。

getDimensions(element)	instance	element: element object or id	返回对象的尺寸，返回值有两个属性，height 和 width。
getHeight(element)	instance	element: element object or id	返回元素的 offsetHeight 。
getStyle(element, cssProperty)	instance	element: element object or id, cssProperty name of a CSS property (either format 'prop-name' or 'propName' works).	返回给定对象的 CSS 属性值或没有指定 cssProperty 时返回 null。
hasClassName(element, className)	instance	element: element object or id, className: name of a CSS class	返回 true 如果元素的类名中含有给定的类名
hide(elem1 [, elem2 [, elem3 [...]]])	instance	elemN: element object or id	通过设定 style.display 为 'none' 来隐藏每个传入的元素。
makeClipping(element)	instance	element: element object or id	能过设定 overflow 的值设定内容溢出剪辑。
makePositioned(element)	instance	element: element object or id	更改对象的 style.position 为 'relative'。
remove(element)	instance	element: element object or id	从 document 对象中删除指定的元素。
removeClassName(element, className)	instance	element: element object or id, className: name of a CSS class	从元素的类名中删除给定的类名。
scrollTo(element)	instance	element: element object or id	滚动 window 到对象的位置。
setStyle(element, cssPropertyHash)	instance	element: element object or id, cssPropertyHash Hash object with the styles to be applied.	依照 cssPropertyHash 参数给对象设置 CSS 属性值。
show(elem1 [, elem2 [, elem3 [...]]])	instance	elemN: element object or id	用设定它的 style.display 为 '' 来显示每个传入的元素。
toggle(elem1 [, elem2 [, elem3 [...]]])	instance	elemN: element object or id	切换每一个传入元素的可视性。
undoClipping(element)	instance	element: element object or id	style.overflow 的值返回上一个设定值。
undoPositioned(element)	instance	element: element object or id	清除对象的 style.position 为 ''
update(element, html)	instance	element: element object or id, html: html content	用给出的 HTML 参数替换对象的 innerHTML, 如果 HTML 参数中包含 <script>, 那么它们不会被包含进去, 但是会执行。

visible(element)	instance	element: element object or id	返回一个布尔值指示对象可不可见。
------------------	----------	-------------------------------	------------------

## The Element.ClassNames class

继承自 *Enumerable*

在一个对象中表示 CSS class names 的集合。

Method	Kind	Arguments	Description
[ctor](element)	constructor	element: any DOM element object or id	创建一个对象，给出对象的 CSS class names 被表现在这个 ClassNames 对象中。
add(className)	instance	className: a CSS class name	把 CSS class name 包含进对象的 class names 列表。
remove(className)	instance	className: a CSS class name	从对象的 class names 列表中移除 className
set(className)	instance	className: a CSS class name	设定对象 CSS class name 为 className,移除其它 class names。

## The Abstract object

这个对象是这个程序包中其他类的根。它没有任何属性和方法。在这个对象中定义的类可以被视为传统的抽象类。

## The Abstract.Insertion class

这个类被用作其他提供动态内容插入功能的类的基类，它像一个抽象类一样被使用。

Method		Kind	Arguments	Description
[ctor](element, content)		constructor	element: element object or id, content: HTML to be inserted	创建一个可以帮助插入动态内容的对象。
contentFromAnonymousTable()		instance	(none)	对 content 通过匿名表格变成一个 Node 数组。
Property	Type	Kind	Description	
adjacency	String	static, parameter	这个参数指定相对于给定元素，内容将被放置的位置。可能的值是: 'beforeBegin', 'afterBegin', 'beforeEnd', 和 'afterEnd'。	
element	Object	instance	与插入物做参照元素对象。	
content	String	instance	被插入的 HTML。	

## The Insertion object

这个对象是其他类似功能的根。它没有任何属性和方法。在这个对象中定义的类仍然可以被视为传统的抽象类。

### The Insertion.Before class

继承自 Abstract.Insertion

在给定元素开始标记的前面插入 HTML。

Method	Kind	Arguments	Description
[ctor](element, content)	constructor	element: element object or id, content: HTML to be inserted	继承自 Abstract.Insertion. 创建一个可以帮助插入动态内容的对象。

下面的代码

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

将把 HTML 变为

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

### The Insertion.Top class

继承自 Abstract.Insertion

在给定元素第一个子节点位置插入 HTML。内容将位于元素的开始标记的紧后面。

Method	Kind	Arguments	Description
[ctor](element, content)	constructor	element: element object or id, content: HTML to be inserted	继承自 Abstract.Insertion. 创建一个可以帮助插入动态内容的对象。

下面的代码



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

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

将把 HTML 变为

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

## The Insertion.Bottom class

*Inherits from Abstract.Insertion*

在给定元素最后一个子节点位置插入 HTML。内容将位于元素的结束标记的紧前面。

Method	Kind	Arguments	Description
[ctor](element, content)	constructor	element: element object or id, content: HTML to be inserted	Inherited from <code>Abstract.Insertion</code> . 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 class

*Inherits from Abstract.Insertion*

在给定元素结束标记的后面插入 HTML。

Method	Kind	Arguments	Description
[ctor](element, content)	constructor	element: element object or id, content: HTML to be inserted	Inherited from <code>Abstract.Insertion</code> . 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

这个对象提供操作表单中的输入项目的功能性方法。

Method	Kind	Arguments	Description
clear(field1 [, field2 [, field3 [...]]])	instance	fieldN: field element object or id	清除传入表单中项目元素的值。
present(field1 [, field2 [, field3 [...]]])	instance	fieldN: field element object or id	只有在所有的表单项目都不为空时返回 true 。
focus(field)	instance	field: field element object or id	移动焦点到给定的表单项目。
select(field)	instance	field: field element object or id	选择支持项目值选择的表单项目的值。
activate(field)	instance	field: field element object or id	移动焦点并且选择支持项目值选择的表单项目的值。

## The Form object

这个对象提供操作表单和他们的输入元素的功能性方法。

Method	Kind	Arguments	Description
serialize(form)	instance	form: form element object or id	返回url参数格式的项目名和值的列表， 如 'field1=value1&field2=value2&field3=value3' 。
findFirstElement(form)	instance	form: form element object or id	返回 Form 中第一个 Enable 的对象。
getElements(form)	instance	form: form element object or id	返回包含所有在表单中输入项目的 Array 对象。

<code>getInputs(form [, typeName [, name]])</code>	<code>form: form instance element object or id, typeName: the type of the input element, name: the name of the input element.</code>	返回一个 Array 包含所有在表单中的 <input> 元素。另外， 这个列表可以对元素的类型或名字属性进行过滤。
<code>disable(form)</code>	<code>form: form instance element object or id</code>	使表单中的所有输入项目无效。
<code>enable(form)</code>	<code>form: form instance element object or id</code>	使表单中的所有输入项目有效。
<code>focusFirstElement(form)</code>	<code>form: form instance element object or id</code>	激活第一个表单中可视的，有效的输入项目。
<code>reset(form)</code>	<code>form: form instance element object or id</code>	重置表单。和调用表单对象的 <code>reset()</code> 方法一样。

## The Form.Element object

这个对象提供表单对象中的可视和非可视元素的功能性方法。

Method	Kind	Arguments	Description
<code>serialize(element)</code>	<code>instance</code>	<code>element: element object or id</code>	返回元素的 名称=值 对， 如 'elementName=elementValue'。
<code>getValue(element)</code>	<code>instance</code>	<code>element: element object or id</code>	返回元素的值。

## The Form.Element.Serializers object

这个对象提供了内部使用的用来协助解析出表单元素的当前值的一些有用的方法。

Method	Kind	Arguments	Description
inputSelector(element)	instance	element: object or id of a form element that has the <i>checked</i> property, like a radio button or checkbox.	返回带有元素名称和值的 Array , 如 ['elementName', 'elementValue']
textarea(element)	instance	element: object or id of a form element that has the <i>value</i> property, like a textbox, button or password field.	返回带有元素名称和值的 Array , 如 ['elementName', 'elementValue']
select(element)	instance	element: object of a <select> element	返回带有元素名称和所有被选择的选项的值或文本的 Array , 如 ['elementName', 'selOpt1 selOpt4 selOpt9']

## The Abstract.TimedObserver class

这个类是用于其它监听一个元素的值(或者任何类中涉及的属性)变化的类的基类，这个类像一个抽象类一样被使用。

子类可以被创建来监听如输入项目值，或 style 属性，或表格的行数，或者其他任何对跟踪变化相关的东西。

子类必须实现这个方法来决定什么才是被监听的元素的当前值。

Method	Kind	Arguments	Description
[ctor](element, frequency, callback)	constructor	element: element object or id, frequency: interval in seconds, callback: function to be called when the element changes	创建一个监听元素的对象。
getValue()	instance, abstract	(none)	子类必须实现这个方法以确定什么这个元素被监视的当前值。
registerCallback()	instance	(none)	这个方法通常不会被外部调用。 被这个对象自己调用来开始监听那个元素。
onTimerEvent()	instance	(none)	这个方法通常不会被外部调用。 被这个对象自己调用来周期性的检查那个元素。

Property	Type	Description
element	Object	被监听的元素对象。
frequency	Number	每次检查中的以秒为单位的时间间隔。
callback	Function(Object, String)	只要元素改变这个方法就会被调用。 会接收到元素对象和新值作为参数。
lastValue	String	元素被核实的最后一个值。

## The Form.Element.Observer class

继承自 `Abstract.TimedObserver`

`Abstract.TimedObserver` 的一个实现类用来监听表单输入项目的值的变化。当你想监听一个没有带报告值变化事件的元素的时候使用这个类。 否则的话使用 `Form.Element.EventObserver` 类代替。

Method	Kind	Arguments	Description
[ctor](element, frequency, callback)	constructor	element: element object or id, frequency: interval in seconds, callback: function to be called when the element changes	继承自 <code>Abstract.TimedObserver</code> . 创建一个监听元素值属性的对象。
getValue()	instance	(none)	返回元素的值。

## The Form.Observer class

继承自 `Abstract.TimedObserver`

`Abstract.TimedObserver` 的一个实现类用来监听表单中任何数据项的值的变化。当你想监听一个没有带报告值变化事件的元素的时候使用这个类。 否则的话使用类 `Form.EventObserver` 代替。

Method	Kind	Arguments	Description
[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	继承自 <code>Abstract.TimedObserver</code> . 创建一个监听表单变化的对象。
getValue()	instance	(none)	返回所有表单数据的一系列值。

## The Abstract.EventObserver class

这个类被用作其他一些类的基类，这些类具有在一个元素的值改变事件发生的时候执行一个回调方法这样的功能。

类 `Abstract.EventObserver` 的多个对象可以绑定到一个元素上，不是一个帮其他的擦出了，而是按照他们付给元素的顺序执行这些回调方法。

单选按钮和复选框的触发事件是 `onclick`，而文本框和下拉列表框/下拉列表框的是 `onchange`。

Method	Kind	Arguments	Description
[ctor](element, callback)	constructor	element: element object or id, callback: function to be called when the event happens	创建监听元素的对象。
getValue()	instance,abstract	(none)	子类必须实现这个方法以确定什么这个元素被监视的当前值。
registerCallback()	instance	(none)	这个方法通常不会被外部调用。被对象调用来把自己绑定到元素的事件上。
registerFormCallbacks()	instance	(none)	这个方法通常不会被外部调用。被对象调用来把自己绑定到表单中的每一个数据项元素的事件上。
onElementEvent()	instance	(none)	这个方法通常不会被外部调用。将被绑定到元素的事件上。

Property	Type	Description
element	Object	被监听的元素对象。
callback	Function(Object, String)	只要元素改变就调用的方法。会接收到元素对象和新值作为参数。
lastValue	String	元素被核实的最后一个值。

## The Form.Element.EventObserver class

继承自 `Abstract.EventObserver`

`Abstract.EventObserver` 的一个实现类，它在监测到表单中数据项元素的值改变的相应事件时候执行一个回调方法。如果元素没有任何报告变化的事件，那么你可以使用 `Form.Element.Observer` 类代替。

Method	Kind	Arguments	Description
[ctor](element, callback)	constructor	element: element object or id, callback: function to be called when the event	继承自 <code>Abstract.EventObserver</code> 。创建一个监听元素值属性的对象。

		happens	
getValue()	instance	(none)	返回元素的值。

The Form.EventObserver class

继承自 Abstract.EventObserver

Abstract.EventObserver 的一个实现类，监听表单对象中包含的任何对象的任何变化，用元素的事件检测值的变化。如果元素没有任何报告变化的事件， 那么你可以使用Form.Observer 类代替。

Method	Kind	Arguments	Description
[ctor](form, callback)	constructor	form: form object or id, callback: function to be called when any data entry element in the form changes	继承自 Abstract.EventObserver。 创建一个监听元素值属性的对象。
getValue()	instance	(none)	返回所有表单数据的一系列值。

Position 对象（预备文档）

这个对象提供许多和元素位置相关的方法。

Method	Kind	Arguments	Description
prepare()	instance	(none)	调整 deltaX 和 deltaY 属性来协调在滚动位置中的变化。 记得在页面滚动之后的任何调用的 withinIncludingScrolloffset 之前调用这个方法。
realOffset(element)	instance	element: object	返回这个元素的正确滚动偏差的 Array 对象， 包括所有影响元素的滚动偏差。结果数组类似 [total_scroll_left, total_scroll_top]
cumulativeOffset(element)	instance	element: object	回这个元素的正确滚动偏差的 Array 对象， 包含任何被放置的父元素强加偏差。结果数组类似 [total_offset_left, total_offset_top]
within(element, x, y)	instance	element: object, x and y: coordinates of a point	测试给定的点的坐标是否在给定的元素的外部矩形范围之内。

<code>withinIncludingScrolloffsets(element, x, y)</code>	<div>element: object, x and y: coordinates of a point</div>	测试给定的点的坐标是否在给定的元素的外部矩形范围之内(包含 scroll offsets)。
<code>overlap(mode, element)</code>	<div>mode: 'vertical' or 'horizontal', element: object</div>	在调用这个方法之前需要调用 <code>within()</code> 。这个方法返回 0.0 到 1.0 之间的数字，来表示坐标在元素重叠的分数。举个例子，如果元素是一个边长是 100px 的正方形的DIV，并且位于(300, 300)，然后 <code>within(divSquare, 330, 330)</code> ； <code>overlap('vertical', divSquare)</code> ；会返回 0.10，意思是那个点位于DIV顶部边框以下 10% (30px) 的位置上。
<code>clone(source, target)</code>	<div>source: element object instance or id, target: element object or id</div>	改变目标元素的大小尺寸和位置与源元素的相同。