prototype.js 开发笔记

Table of Contents

覆盖版本 1.3.1

Chapter 1. Programming Guide

1.1. Prototype 是什么?

或许你还没有用过它， prototype.js 是一个由 Sam Stephenson 写的 JavaScript 包。这个构思奇妙编写良好的一段**兼容标准**的一段代码将承担创造胖客户端， 高交互性 WEB 应用程序的重担。轻松加入 Web 2.0 特性。

如果你最近体验了这个程序包，你很可能会发现文档并不是它的强项之一。像所有在我之前的开发者一样，我只能一头扎进 prototype.js 的源代码中并且试验其中的每一个部分。 我想当我学习他的时候记写笔记然后分享给其他人将会很不错。

我也一起提供了这个包的对象，类，方法和扩展的 非官方参考 。

1.2. 关联文章

高级 JavaScript 指南

1.3. 通用性方法

这个程序包里面包含了许多预定义的对象和通用性方法。编写这些方法的明显的目的就是为了减少你大量的重复编码和惯用法。

1.3.1. 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。下面的例子会向你描述这些。

```
<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>
```

这个方法的另一个好处就是你可以传入 id 字符串或者元素对象自己，这使得在创建可以传入任何形式参数的方法的时候， 它变得非常有用。

1.3.2. 使用$F()方法

$F()方法是另一个非常受欢迎的简写。它可以返回任何输入表单控件的值，如文本框或下拉框。 这个方法可以传入元素的 id 或者元素自己。

```
<script>
    function test3()
    {
        alert(  $F('userName')  );
    }
</script>
```

```
<input type="text" id="userName" value="Joe Doe"><br>
<input type="button" value=Test3 onclick="test3();"><br>
```

1.3.3. 使用$A()方法

$A()方法把接收到的参数转换成一个 Array 对象。

这个方法加上对 Array 类的扩展，可以很容易的转换或者复制任意的列举列表到 Array 对象， 一个被推荐使用的用法就是转换 DOM 的 NodeLists 到一个普通的数组里，可以被更广泛高效的使用， 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();" >
```

1.3.4. 使用$H()方法

$H()方法把对象转化成可枚举的貌似联合数组 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.toQueryString()); //displays:
first=10&second=20&third=30
   }

</script>
```

1.3.5. 使用$R()方法

$R()方法是 new ObjectRange(lowerBound, upperBound, excludeBounds)的一个简单快捷的使用方式。

ObjectRange 类文档里面有完整的解释。 同时，我们来看看一个简单的例子， 来演示通过 each 方法遍历的用法。 那个方法的更多解释在 Enumerable 对象文档里面。

```
<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();" >
```

1.3.6. 使用 Try.these()方法

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>
```

1.4. 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。我们来看看它们的一些。

1.4.1. 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 对象和服务器通信并且得到这段 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>
```

你看到传入 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。

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

1.4.2. 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.');
   }
</script>

<input type=button value=GetHtml onclick="getHTML()">
<div id="placeholder"></div>
```

如果你的服务器逻辑是返回 JavaScript 代码而不是单纯的 HTML 标记，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。

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

Chapter 2. prototype.js 参考

2.1. JavaScript 类的扩展

prototype.js 包中加入功能的一种途径就是扩展已有的 JavaScript 类。

2.2. 对 Object 类的扩展

Table 2.1. Object 类的扩展

方法	类别	参数	描述
extend(destination, source)	static	destination: 任何对象, source: 任何对象	用从 source 到 destination 复制所有属性和方法的方式 来提供一种继承机制。
extend(object)	instance	任何对象	用从传入的 object 中复制所有属性和方法的方式来提供一种继承机制。

2.3. 对 Number 类的扩展

Table 2.2. Number 类的扩展

方法	类别	参数	描述
toColorPart()	instance	(none)	返回数字的十六进制描述， 当在 HTML 中转换为 RGB 颜色组件到 HTML 中使用的颜色。

2.4. 对 Function 类的扩展

Table 2.3. 对 Function 类的扩展

方法	类别	参数	描述
bind(object)	instance	object: 拥有这个方法的对象	返回预先绑定在拥有该函数(=方法)的对象上的函数实例， 返回的方法将和原来的方法具有相同的参数。
bindAsEventListener(object)	instance	object: 拥有这个方法的对象	返回预先绑定在拥有该函数(=方法)的对象上的函数实例， 返回的方法将把当前的事件对象作为它的参数。

让我们看看这些扩展的具体例子。

```
<input type=checkbox id=myChk value=1> Test?
<script>
   //declaring the class
   var CheckboxWatcher = Class.create();

   //defining the rest of the class implmentation
   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>
```

2.5. 对 String 类的扩展

Table 2.4. String 类的扩展

方法	类别	参数	描述
stripTags()	instance	(none)	返回一个把所有的 HTML 或 XML 标记都移除的字符串。
escapeHTML()	instance	(none)	返回一个把所有的 HTML 标记回避掉的字符串。

方法	类别	参数	描述
unescapeHTML()	instance	(none)	和 escapeHTML()相反。

2.6. 对 document DOM 对象的扩展

Table 2.5. document DOM 对象的扩展

方法	类别	参数	描述
getElementsByClassName(className)	instance	className: 关联在元素上的 CSS 类名	返回给定的具有相同的 CSS 类名的所有元素。

2.7. 对 Event 对象的扩展

Table 2.6. Event 对象的扩展

属性	类型	描述
KEY_BACKSPACE	Number	8: 常量，退格(Backspace)键的代码。
KEY_TAB	Number	9: 常量，Tab 键的代码。
KEY_RETURN	Number	13: 常量，回车键的代码。
KEY_ESC	Number	27: 常量， Esc 键的代码。
KEY_LEFT	Number	37: 常量，左箭头键的代码。
KEY_UP	Number	38: 常量，上箭头键的代码。
KEY_RIGHT	Number	39: 常量，右箭头键的代码。
KEY_DOWN	Number	40: 常量，下箭头键的代码。
KEY_DELETE	Number	46: 常量，删除(Delete)键的代码。
observers:	Array	*缓存的观察者的列表，这个对象内部具体实现的一部分。*

Table 2.7. Event 对象的扩展

方法	类别	参数	描述
element(event)	static	event: 事件对象	返回引发这个事件的元素。

方法	类别	参数	描述
isLeftClick(event)	static	event: 事件对象	如果鼠标左键单击返回true。
pointerX(event)	static	event: 事件对象	返回在页面上 x 坐标。
pointerY(event)	static	event: 事件对象	返回在页面上 y 坐标。
stop(event)	static	event: 事件对象	用这个方法来中止事件的默认行为来使事件的传播停止。
findElement(event, tagName)	static	event: 事件对象, tagName: 指定标记的名字	向 DOM 树的上位查找，找到第一个给定标记名称的元素，从这个元素开始触发事件。
observe(element, name, observer, useCapture)	static	element: 对象或者对象 id, name: 事件名 (如 'click', 'load', etc), observer: 处理这个事件的方法, useCapture: 如果 true, 在捕捉到事件的阶段处理事件 那么如果 false 在 *bubbling* 阶段处理。	加入一个处理事件的方法。
stopObserving(element, name, observer, useCapture)	static	element: 对象或者对象 id, name: 事件名 (如	删除一个处理实践的方

方法	类别	参数	描述
		'click', 'load', etc), observer: 处理这个事件的方法, useCapture: 如果 true, 在捕捉到事件的阶段处理事件 那么如果 false 在 *bubbling* 阶段处理。	法。
_observeAndCache(element, name, observer, useCapture)	static		私有方法，不用管它。
unloadCache()	static	(none)	私有方法，不用管它。清除内存中的多有观察着缓存。

让我们看看怎样用这个对象加入处理 window 对象的 load 事件的处理方法。

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

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

2.8. 在 prototype.js 中定义的新对象和类

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。

2.9. PeriodicalExecuter 对象

这个对象提供一定间隔时间上重复调用一个方法的逻辑。

Table 2.8. PeriodicalExecuter 对象

方法	类别	参数	描述
[ctor](callback, interval)	constructor	callback: 没有参数的方法, interval: 秒数	创建这个对象的实例将会重复调用给定的方法。

Table 2.9. PeriodicalExecuter 对象

属性	类型	描述
callback	Function()	被调用的方法，该方法不会被传入参数。
frequency	Number	以秒为单位的间隔。
currentlyExecuting	Boolean	表示这个方法是否正在执行。

2.10. Prototype 对象

Prototype 没有太重要的作用，只是声明了该程序包的版本 。

Table 2.10. The Prototype object

属性	类型	描述
Version	String	包的版本。
emptyFunction	Function()	空方法对象。

2.11. Class 对象

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
```

Table 2.11. Class 对象

方法	类别	参数	描述
create(*)	instance	(any)	定义新类的构造方法。

2.12. Ajax 对象

这个对象被用作其他提供 AJAX 功能的类的根对象。

Table 2.12. Ajax 对象

方法	类别	参数	描述
getTransport()	instance	(none)	返回新的 XMLHttpRequest 对象。

2.13. Ajax.Base 类

这个类是其他在 Ajax 对象中定义的类的基类。

Table 2.13. Ajax.Base 类

方法	类别	参数	描述
setOptions(options)	instance	options: AJAX 选项	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。

2.14. Ajax.Request 类

继承自 Ajax.Base

封装 AJAX 操作

Table 2.14. Ajax.Request 类

属性	类型	类别	描述
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	携带 AJAX 操作的 XMLHttpRequest 对象。

Table 2.15. Ajax.Request 类

方法	类别	参数	描述
[ctor](url, options)	constructor	url: 请求的 url, options: AJAX 选项	创建这个对象的一个实例，它将在给定的选项下请求 url。**重要:** 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)
request(url)	instance	url: AJAX 请求的 url	这个方法通常不会被外部调用。已经在构造方法

方法	类别	参数	描述
			中调用了。
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。
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。
respondToReadyState(readyState)	instance	readyState: 状态数字 (1 到 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。

2.15. options 参数对象

AJAX 操作中一个重要的部分就是 options 参数。 本质上没有 options 类。任何对象都可以被传入，只要带有需要的属性。通常会只为了 AJAX 调用创建匿名类。

Table 2.16. options 参数对象

属性	类型	Default	描述
method	Array	'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

属性	类型	Default	描述
			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']
onXXXXXXXX	Function(XMLHttpRequest)	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， 这个参数是携带 AJAX 操作的 XMLHttpRequest 对象。
onSuccess	Function(XMLHttpRequest)	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， 这个参数是携带 AJAX 操作的

属性	类型	Default	描述
			XMLHttpRequest 对象。
onFailure	Function(XMLHttpRequest)	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，这个参数是携带 AJAX 操作的 XMLHttpRequest 对象。
insertion	Function(Object, String)	null	为了把返回的文本注入到一个元素中而执行的方法。这个方法将被传入两个参数，要被更新的对象并且只应用于 Ajax.Updater 的响应文本 。
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 对象中的减漫访问的次数， 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。

2.16. Ajax.Updater 类

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。

Table 2.17. Ajax.Updater 类

属性	类型	类别	描述
ScriptFragment	String	static	可以判断是否为脚本的正则表达式。
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 。

Table 2.18. Ajax.Updater 类

方法	类别	参数	描述
[ctor](container, url, options)	constructor	container:可以是元素的 id, 也可以是元素自己, 或者可以是带有 2 个属性的对象 - **object.success** AJAX 请求成功的时候用到的元素(或者 id) 否则用到 **object.failure** 中设定的元素(或 id) ,url: 请求的 url, options: AJAX 选项	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

方法	类别	参数	描述
			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。

2.17. Ajax.PeriodicalUpdater 类

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 。

Table 2.19. Ajax.PeriodicalUpdater 类

属性	类型	类别	描述
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。

Table 2.20. Ajax.PeriodicalUpdater 类

方法	类别	参数	描述
[ctor](container, url, options)	constructor	container:可以是元素的 id，也可以是元素自己，或者可以是带有 2 个属性的对象 - **object.success** AJAX 请求成功的时候用到的元素(或者 id) 否则用到	创建一个用给定的选项请求给定的 url 的一个实例。

方法	类别	参数	描述
		object.failure 中设定的元素(或 id) ,url: 请求的 url, options: AJAX 选项	
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)	这个方法通常不会被外部调用。 对象为了停止周期性执行任务的时候调用的方法。
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。

2.18. Element 对象

这个对象提供在操作 DOM 中元素时使用的功能性方法。

Table 2.21. Element 对象

方法	类别	参数	描述

方法	类别	参数	描述
toggle(elem1 [, elem2 [, elem3 [...]]])	constructor	elemN: 元素对象或 id	切换每一个传入元素的可视性。
hide(elem1 [, elem2 [, elem3 [...]]])	instance	elemN: 元素对象或 id	用设定它的 style.display 为'none'来隐藏每个传入的元素。
show(elem1 [, elem2 [, Slem3 [...]]])	instance	elemN: 元素对象或 id	用设定它的 style.display 为''来显示每个传入的元素。
remove(element)	instance	element: 元素对象或 id	从 document 对象中删除指定的元素。
getHeight(element)	instance	element: 元素对象或 id	返回元素的 offsetHeight 。
addClassName(element, className)	instance	element: 元素对象或 id, className: CSS 类名	向元素的类名中加入给定的类名。
hasClassName(element, className)	instance	element: 元素对象或 id, className: CSS 类名	返回 true 如果元素的类名中含有给定的类名
removeClassName(element, className)	instance	element: 元素对象或 id, className: CSS 类名	从元素的类名中删除给定的类名。
cleanWhitespace(element)	instance	element: 元素对象或 id	删除该元素的所有只含有空格的子节点。

2.19. Abstract 对象

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。

2.20. Abstract.Insertion 类

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。

Table 2.22. Abstract.Insertion 类

方法	类别	参数	描述
[ctor](element, content)	constructor	element: 元素对象或 id, content: 被插入的 HTML	创建一个可以帮助插入动态内容的对象。

Table 2.23. Abstract.Insertion 类

属性	类型	类别	描述
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 。

2.21. Insertion 对象

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。

2.22. Insertion.Before 类

继承自 Abstract.Insertion

在给定元素开始标记的前面插入 HTML。

Table 2.24. Insertion.Before 类

方法	类别	参数	描述
[ctor](element, content)	constructor	element: 元素对象或 id,	继承自 Abstract.Insertion. 创建一个可以帮助插入动态内

方法	类别	参数	描述
		content: 被插入的 HTML	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>
```

2.23. Insertion.Top 类

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。

Table 2.25. Insertion.Top 类

方法	类别	参数	描述
[ctor](element, content)	constructor	element: 元素对象或 id, content: 被插入的 HTML	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>
```

2.24. Insertion.Bottom 类

继承自 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。

Table 2.26. Insertion.Bottom 类

方法	类别	参数	描述
[ctor](element, content)	constructor	element: 元素对象或 id, content: 被插入的 HTML	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.Bottom('person', " What's up?");
</script>
```

将把 HTML 变为

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

2.25. Insertion.After 类

继承自 Abstract.Insertion

在给定元素结束标记的后面插入 HTML。

Table 2.27. Insertion.After 类

方法	类别	参数	描述

方法	类别	参数	描述
[ctor](element, content)	constructor	element: 元素对象或 id, content: 被插入的 HTML	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.After('person', ' Are you there?');
</script>
```

将把 HTML 变为

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

2.26. Field 对象

这个对象提供操作表单中的输入项目的功能性方法。

Table 2.28. Field 对象

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

2.27. Form 对象

这个对象提供操作表单和他们的输入项目的功能性方法。

Table 2.29. Form 对象

方法	类别	参数	描述
serialize(form)	instance	form: 表单元素或 id	返回 url 参数格式的项目名和值的列表，如'field1=value1&field2=value2&field3=value3'。
getElements(form)	instance	form: 表单元素或 id	返回包含所有在表单中输入项目的 Array 对象。
getInputs(form [, typeName [, name]])	instance	form: 表单元素或 id, typeName: 输入项目的类型, name: 输入项目的名称	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。
disable(form)	instance	form: 表单元素或 id	使表单中的所有输入项目无效。
enable(form)	instance	form: 表单元素或 id	使表单中的所有输入项目有效。
focusFirstElement(form)	instance	form: 表单元素或 id	激活第一个表单中可视的，有效的输入项目。

方法	类别	参数	描述
reset(form)	instance	form: 表单元素或 id	重置表单。和调用表单对象的 reset() 方法一样。

2.28. Form.Element 对象

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。

Table 2.30. Form.Element 对象

方法	类别	参数	描述
serialize(element)	instance	element: 表单元素或 id	返回元素的 名称=值 对，如 'elementName=elementValue'。
getValue(element)	instance	element: 表单元素或 id	返回元素的值。

2.29. Form.Element.Serializers 对象

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功能性方法。

Table 2.31. Form.Element.Serializers 对象

方法	类别	参数	描述
inputSelector(element)	instance	element: 一个带有 checked 属性的表单元素或 id， 如 radio 或 checkbox。	返回带有元素名称和值的 Array，如 ['elementName', 'elementValue']
textarea(element)	instance	element: 一个带有 value 属性的表单元素或 id， 如 textbox, button 或 password 项	返回带有元素名称和值的 Array，如 ['elementName', 'elementValue']

方法	类别	参数	描述
		目。	
select(element)	instance	element: 一个 <select> 元素对象或 id。	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']

2.30. Abstract.TimedObserver 类

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。

Table 2.32. Abstract.TimedObserver 类

方法	类别	参数	描述
[ctor](element, frequency, callback)	constructor	element: 元素对象或 id, frequency: 以秒为单位的间隔, callback: 当元素改变的时候调用的方法。	创建一个监听元素的对象。
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

方法	类别	参数	描述
			调用来周期性的检查那个元素。

Table 2.33. Abstract.TimedObserver 类

属性	类型	描述
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。

2.31. Form.Element.Observer 类

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。

Table 2.34. Form.Element.Observer 类

方法	类别	参数	描述
[ctor](element, frequency, callback)	constructor	element: 元素对象或 id, frequency: 以秒为单位的间隔, callback: 当元素改变的时候调用的方法。	继承自 Abstract.TimedObserver. 创建一个监听元素值属性的对象。
getValue()	instance	(none)	返回元素的值。

2.32. Form.Observer 类

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。

Table 2.35. Form.Observer 类

方法	类别	参数	描述
[ctor](form, frequency, callback)	constructor	form: 表单对象或 id,	继承自 Abstract.TimedObserver. 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。

2.33. Abstract.EventObserver 类

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 。

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。

Table 2.36. Abstract.EventObserver 类

方法	类别	参数	描述
[ctor](element, callback)	constructor	element: 元素对象或 id, callback: 当事件发生的时候调用的方法。	创建监听元素的对象。
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

方法	类别	参数	描述
			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。

Table 2.37. Abstract.EventObserver 类

属性	类型	描述
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。

2.34. Form.Element.EventObserver 类

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。

Table 2.38. Form.Element.EventObserver 类

方法	类别	参数	描述
[ctor](element, callback)	constructor	element: 元素对象或 id, callback: 当事件发生的时候调用的方法。	继承自 Abstract.EventObserver。 创建一个监听元素值属性的对象。
getValue()	instance	(none)	返回元素的值。

2.35. Form.EventObserver 类

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。

Table 2.39. Form.Element.EventObserver 类

方法	类别	参数	描述
[ctor](form, callback)	constructor	form: 元素对象或 id, callback: 当表单中任何数据项改变的时候调用的方法	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。

2.36. 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。

Table 2.40. Position 对象 (预备文档)

方法	类别	参数	描述
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 withinIncludingScroll offset 之前调用这个方法。
realOffset(element)	instance	element: 对象或 id	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

方法	类别	参数	描述
			[total_scroll_left, total_scroll_top]
cumulativeOffset(element)	instance	element: 对象或 id	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: 对象, x 和 y: 一个点的坐标	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。
withinIncludingScrolloffsets(element, x, y)	instance	element: object, x and y: coordinates of a point	
overlap(mode, element)	instance	mode: 'vertical' 或 'horizontal', element: 对象	在调用这个方法之前需要调用 within() 。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 within(divSquare, 330, 330); overlap('vertical', divSquare); 会返回 0.10，意思是那个点位于

方法	类别	参数	描述
			DIV 顶部边框以下 10% (30px) 的位置上。
clone(source, target)	instance	source: 元素对象或 id, target: 元素对象或 id	改变目标元素的大小尺寸和位置与源元素的相同。

如果你发现错误，过失或信息不完全，或平淡无意义的东西，请 通知作者 ，作者会尽可能快的修正它，作者仍然在写其他没写的类，来保持一致。 任何翻译的过失，不足，词不达意，表达含糊等请通知我，我会再加斟酌，找到合适的描述。