用AJAX来控制书签和回退按钮

作者：Brad Neuberg

译者:boool

版权声明：任何获得Matrix授权的网站，转载时请务必以超链接形式标明文章原始出处和作者信息及本声明

作者:Brad Neuberg;boool

原文地址:http://www.onjava.com/pub/a/onjava/2005/10/26/ajax-handling-bookmarks-and-back-button.html

中文地址:http://www.matrix.org.cn/resource/article/43/43972\_AJAX.html

关键词： ajax;bookmarks;back button

这篇文章描述了一个支持AJAX应用书签和回退按钮的开源的javascript库。在这个指南的最后，开发者将会得出一个甚至不是Google Maps 或者 Gmail那样处理的AJAX的解决方案：健壮的，可用的书签和向前向后的动作能够象其他的web页面一样正确的工作。

AJAX：怎样去控制书签和回退按钮 这篇文章说明了一个重要的成果，AJAX应用目前面对着书签和回退按钮的应用，描述了非常简单的历史库（Really Simple History），一个开源的解决这类问题的框架，并提供了一些能够运行的例子。

这篇文章描述的主要问题是双重的，一是一个隐藏的html 表单被用作一个大而短生命周期的客户端信息的session缓存，这个缓存对在这个页面上前进回退是强壮的。二是一个锚连接和隐藏的iframes的组合用来截取和记录浏览器的历史事件，来实现前进和回退的按钮。这两个技术都被用一个简单的javascript库来封装，以利于开发者的使用。

存在的问题

书签和回退按钮在传统的多页面的web应用上能顺利的运行。当用户在网站上冲浪时，他们的浏览器地址栏能更新URL,这些URL可以被粘贴到的email或者添加到书签以备以后的使用。回退和前进按钮也可以正常运行，这可以使用户在他们访问的页面间移动。

AJAX应用是与众不同的，然而，他也是在单一web页面上成熟的程序。浏览器不是为AJAX而做的—AJAX他捕获过去的事件，当web应用在每个鼠标点击时刷新页面。

在象Gmail那样的AJAX软件里，浏览器的地址栏正确的停留就象用户在选择和改变应用的状态时，这使得作书签到特定的应用视图里变得不可能。此外，如果用户按下了他们的回退按钮去返回上一个操作，他们会惊奇的发现浏览器将完全离开原来他所在的应用的web页面。

解决方案

开源的Really Simply History(RSH)框架解决了这些问题，他带来了AJAX应用的作书签和控制前进后退按钮的功能。RSH目前还是beta版，在Firefox1.0上，Netscape7及以上,和IE6及以上运行。Safari现在还不支持（要得到更详细的说明，请看我的weblog中的文章Coding in Paradise: Safari: No DHTML History Possible).

目前存在的几个AJAX框架可以帮助我们做书签和发布历史，然而所有的框架都因为他们的实现而被几个重要的bug困扰（请看Coding in Paradise: AJAX History Libraries 得知详情）。此外，许多AJAX历史框架集成绑定到较大的库上，比如Backbase 和 Dojo，这些框架提供了与传统AJAX应用不同的编程模型，强迫开发者去采用一整套全新的方式去获得浏览器的历史相关的功能。

相应的，RSH是一个简单的模型，能被包含在已经存在的AJAX系统中。而且，Really Simple History库使用了一些技巧去避免影响到其他历史框架的bug.

Really Simple History框架由2个javascript类库组成，分别叫DhtmlHistory 和 HistoryStorage.

DhtmlHistory 类提供了一个对AJAX应用提取历史的功能。.AJAX页面add() 历史事件到浏览器里，指定新的地址和关联历史数据。DhtmlHistory 类用一个锚的hash表更新浏览器现在的URL，比如#new-location ，然后用这个新的URL关联历史数据。AJAX应用注册他们自己到历史监听器里，然后当用户用前进和后退按钮导航的时候，历史事件被激发，提供给浏览器新的地址和调用add()持续保留数据。

第二个类HistoryStorage，允许开发者存储任意大小的历史数据。一般的页面，当一个用户导航到一个新的网站，浏览器会卸载和清除所有这个页面的应用和javascript状态信息。如果用户用回退按钮返回过来了，所有的数据已经丢失了。HistoryStorage 类解决了这个问题，他有一个api 包含简单的hashtable方法比如put(),get(),hasKey()。这些方法允许开发者在离开web页面时存储任意大小的数据，当用户点了回退按钮返回时，数据可以通过HistoryStorage 类被访问。我们通过一个隐藏的表单域（a hidden form field），利用浏览器即使在用户离开web页面也会自动保存表单域值的这个特性，完成这个功能。

让我们立即进入一个简单的例子吧。

示例1

首先，任何一个想使用Really Simple History框架的页面必须包含（include）dhtmlHistory.js 脚本。

<!-- Load the Really Simple

History framework -->

<script type="text/javascript"

src="../../framework/dhtmlHistory.js">

</script>

DHTML History 应用也必须在和AJAX web页面相同的目录下包含一个叫blank.html 的指定文件，这个文件被Really Simple History框架绑定而且对IE来说是必需的。另一方面，RSH使用一个hidden iframe 来追踪和加入IE历史的改变，为了正确的执行功能，这个iframe需要指向一个真正的地址，不需要blank.html。

RSH框架创建了一个叫dhtmlHistory 的全局对象，作为操作浏览器历史的入口。使用dhtmlHistory 的第一步需要在页面加载后初始化这个对象。

window.onload = initialize;

function initialize() {

// initialize the DHTML History

// framework

dhtmlHistory.initialize();

然后，开发者使用dhtmlHistory.addListener()方法去订阅历史改变事件。这个方法获取一个javascript回调方法，当一个DHTML历史改变事件发生时他将收到2个自变量，新的页面地址，和任何可选的而且可以被关联到这个事件的历史数据。

indow.onload = initialize;

function initialize() {

// initialize the DHTML History

// framework

dhtmlHistory.initialize();

// subscribe to DHTML history change

// events

dhtmlHistory.addListener(historyChange);

historyChange()方法是简单易懂得，它是由一个用户导航到一个新地址后收到的新地址（newLocation）和一个关联到事件的可选的历史数据historyData 构成的。

/\*\* Our callback to receive history change

events. \*/

function historyChange(newLocation,

historyData) {

debug("A history change has occurred: "

+ "newLocation="+newLocation

+ ", historyData="+historyData,

true);

}

上面用到的debug()方法是例子代码中定义的一个工具函数，在完整的下载例子里有。debug()方法简单的在web页面上打一条消息，第2个Boolean变量，在代码里是true,控制一个新的debug消息打印前是否要清除以前存在的所有消息。

一个开发者使用add()方法加入历史事件。加入一个历史事件包括根据历史的改变指定一个新的地址，就像"edit:SomePage"标记, 还提供一个事件发生时可选的会被存储到历史数据historyData值.

window.onload = initialize;

function initialize() {

// initialize the DHTML History

// framework

dhtmlHistory.initialize();

// subscribe to DHTML history change

// events

dhtmlHistory.addListener(historyChange);

// if this is the first time we have

// loaded the page...

if (dhtmlHistory.isFirstLoad()) {

debug("Adding values to browser "

+ "history", false);

// start adding history

dhtmlHistory.add("helloworld",

"Hello World Data");

dhtmlHistory.add("foobar", 33);

dhtmlHistory.add("boobah", true);

var complexObject = new Object();

complexObject.value1 =

"This is the first value";

complexObject.value2 =

"This is the second data";

complexObject.value3 = new Array();

complexObject.value3[0] = "array 1";

complexObject.value3[1] = "array 2";

dhtmlHistory.add("complexObject",

complexObject);

在add()方法被调用后，新地址立刻被作为一个锚值显示在用户的浏览器的URL栏里。例如，一个AJAX web页面停留在http://codinginparadise.org/my\_ajax\_app，调用了dhtmlHistory.add("helloworld", "Hello World Data" 后，用户将在浏览器的URL栏里看到下面的地址

http://codinginparadise.org/my\_ajax\_app#helloworld

然后他们可以把这个页面做成书签，如果他们使用这个书签，你的AJAX应用可以读出#helloworld值然后使用她去初始化web页面。Hash里的地址值被Really Simple History框架显式的编码和解码（URL encoded and decoded） （这是为了解决字符的编码问题）

对当AJAX地址改变时保存更多的复杂的状态来说，historyData比一个更容易的匹配一个URL的东西更有用。他是一个可选的值，可以是任何javascript类型，比如Number, String, 或者 Object 类型。有一个例子是用这个在一个多文本编辑器（rich text editor）保存所有的文本，例如，如果用户从这个页面漂移（或者说从这个页面导航到其他页面，离开了这个页面）走。当一个用户再回到这个地址，浏览器会把这个对象返回给历史改变侦听器（history change listener）。

开发者可以提供一个完全的historyData 的javascript对象，用嵌套的对象objects和排列arrays来描绘复杂的状态。只要是JSON (JavaScript Object Notation)允许的那么在历史数据里就是允许的，包括简单数据类型和null型。DOM的对象和可编程的浏览器对象比如XMLHttpRequest ，不会被保存。注意historyData 不会被书签持久化，如果浏览器关掉，或者浏览器的缓存被清空，或者用户清除历史的时候，会消失掉。

使用dhtmlHistory 最后一步，是isFirstLoad() 方法。如果你导航到一个web页面，再跳到一个不同的页面，然后按下回退按钮返回起始的网站，第一页将完全重新装载，并激发onload事件。这样能产生破坏性，当代码在第一次装载时想要用某种方式初始化页面的时候，不会再刷新页面。isFirstLoad() 方法让区别是最开始第一次装载页面，还是相对的，在用户导航回到他自己的浏览器历史中记录的网页时激发load事件，成为可能。

在例子代码中，我们只想在第一次页面装载的时候加入历史事件，如果用户在第一次装载后，按回退按钮返回页面，我们就不想重新加入任何历史事件。

window.onload = initialize;

function initialize() {

// initialize the DHTML History

// framework

dhtmlHistory.initialize();

// subscribe to DHTML history change

// events

dhtmlHistory.addListener(historyChange);

// if this is the first time we have

// loaded the page...

if (dhtmlHistory.isFirstLoad()) {

debug("Adding values to browser "

+ "history", false);

// start adding history

dhtmlHistory.add("helloworld",

"Hello World Data");

dhtmlHistory.add("foobar", 33);

dhtmlHistory.add("boobah", true);

var complexObject = new Object();

complexObject.value1 =

"This is the first value";

complexObject.value2 =

"This is the second data";

complexObject.value3 = new Array();

complexObject.value3[0] = "array 1";

complexObject.value3[1] = "array 2";

dhtmlHistory.add("complexObject",

complexObject);

让我们继续使用historyStorage 类。类似dhtmlHistory ，historyStorage通过一个叫historyStorage的单一全局对象来显示他的功能，这个对象有几个方法来伪装成一个hash table, 象put(keyName, keyValue), get(keyName), and hasKey(keyName).键名必须是字符，同时键值可以是复杂的javascript对象或者甚至是xml格式的字符。在我们源码source code的例子中，我们put() 简单的XML到historyStorage 在页面第一次装载时。

window.onload = initialize;

function initialize() {

// initialize the DHTML History

// framework

dhtmlHistory.initialize();

// subscribe to DHTML history change

// events

dhtmlHistory.addListener(historyChange);

// if this is the first time we have

// loaded the page...

if (dhtmlHistory.isFirstLoad()) {

debug("Adding values to browser "

+ "history", false);

// start adding history

dhtmlHistory.add("helloworld",

"Hello World Data");

dhtmlHistory.add("foobar", 33);

dhtmlHistory.add("boobah", true);

var complexObject = new Object();

complexObject.value1 =

"This is the first value";

complexObject.value2 =

"This is the second data";

complexObject.value3 = new Array();

complexObject.value3[0] = "array 1";

complexObject.value3[1] = "array 2";

dhtmlHistory.add("complexObject",

complexObject);

// cache some values in the history

// storage

debug("Storing key 'fakeXML' into "

+ "history storage", false);

var fakeXML =

'<?xml version="1.0" '

+'encoding="ISO-8859-1"?>'

+'<foobar>'

+ '<foo-entry/>'

+'</foobar>';

historyStorage.put("fakeXML", fakeXML);

}

然后，如果用户从这个页面漂移走（导航走）又通过返回按钮返回了，我们可以用get（）提出我们存储的值或者用haskey（）检查他是否存在。

window.onload = initialize;

function initialize() {

// initialize the DHTML History

// framework

dhtmlHistory.initialize();

// subscribe to DHTML history change

// events

dhtmlHistory.addListener(historyChange);

// if this is the first time we have

// loaded the page...

if (dhtmlHistory.isFirstLoad()) {

debug("Adding values to browser "

+ "history", false);

// start adding history

dhtmlHistory.add("helloworld",

"Hello World Data");

dhtmlHistory.add("foobar", 33);

dhtmlHistory.add("boobah", true);

var complexObject = new Object();

complexObject.value1 =

"This is the first value";

complexObject.value2 =

"This is the second data";

complexObject.value3 = new Array();

complexObject.value3[0] = "array 1";

complexObject.value3[1] = "array 2";

dhtmlHistory.add("complexObject",

complexObject);

// cache some values in the history

// storage

debug("Storing key 'fakeXML' into "

+ "history storage", false);

var fakeXML =

'<?xml version="1.0" '

+'encoding="ISO-8859-1"?>'

+'<foobar>'

+ '<foo-entry/>'

+'</foobar>';

historyStorage.put("fakeXML", fakeXML);

}

// retrieve our values from the history

// storage

var savedXML =

historyStorage.get("fakeXML");

savedXML = prettyPrintXml(savedXML);

var hasKey =

historyStorage.hasKey("fakeXML");

var message =

"historyStorage.hasKey('fakeXML')="

+ hasKey + "<br>"

+ "historyStorage.get('fakeXML')=<br>"

+ savedXML;

debug(message, false);

}

prettyPrintXml() 是一个第一在例子源码full example source code中的工具方法。这个方法准备简单的xml显示在web page ，方便调试。