Quick Jump

Re-inventing XMLHttpRequest: Cross-browser implementation with sniffing capabilities

Published by Sergey Ilinsky on 16th October, 2007 (Last Updated on 17th December, 2007)

This article attempts to summarize some experience in working with the XMLHttpRequest object by showing how to:

- Implement a cross-browser XMLHttpRequest object (to replace the existing one)
- Enable interception of the XMLHttpRequest objects operations
- Solve multiple browser-specific XMLHttpRequest issues

The approach described in the article empowers the Backbase Enterprise Ajax 4 Debugger application enabling its Input/Output (Net) inspector.

Contents

1. Re-implementing XMLHttpRequest object

Most people are aware of how to make the Internet Explorer 6.0 browser compatible to the others with respect to its support for XMLHttpRequest object.

```
if (!window.XMLHttpRequest) {
 window.XMLHttpRequest = function() {
  return new ActiveXObject('Microsoft.XMLHTTP');
 }
}
```

Elegant, isn't it? Having this code executed you could have a global XMLHttpRequest object available, which is not that different from what you would get in other browsers. Unfortunately the instance of such a XMLHttpRequest object won't be a JavaScript object - the technique leaves you with a COM object! We are interested in a real JavaScript object representing instances, so we should use a different approach - wrapping.

Wrapping the native implementation into a cross-browser object

In order to create a real cross-browser implementation of the XMLHttpRequest object and to be able later to fix native XMLHttpRequest bugs along with implementation of sniffing we should re-implement XMLHttpRequest object from scratch by defining a new object that would incapsulate the native one. Let's define constructor, public properties and public methods. In the listing below there is only one property and one method implementation shown, still this should be enough to catch the idea.

```
// Keep the reference to earlier defined object implementation (if any)
var xXMLHttpRequest = window.XMLHttpRequest;

// Constructor
function XMLHttpRequest() {
 this.object = xXMLHttpRequest ? new xXMLHttpRequest : new window.ActiveXObject('Microsoft.XMLHTTP');
}

// Public properties
...
XMLHttpRequest.prototype.onreadystatechange = null;
...

// Public Methods
...

XMLHttpRequest.prototype.send = function(vData) {
 this.object.send(vData);
}

...
```

This way we have a prototyped implementation of the XMLHttpRequest object. Let's go further and see how we could enable sniffing.

Enabling object sniffing capabilities

The implementation should be capable of transparent sniffing of all operations done on all instances of the XMLHttpRequest object. We define custom static members on XMLHttpRequest global object (constructor) with names of public methods prefixed with "on" (onopen, onsend, onabort) plus onreadystatechange. Whenever a method will be invoked, we will first call a corresponding static member on the XMLHttpRequest object (if it is set) on behalf of the instance with arguments passed into the call. Take a look at the following piece of code.

```
// Static members
XMLHttpRequest.onsend = null;
...

// Public Methods
XMLHttpRequest.prototype.send = function (vData) {
 // Add method sniffer
 if (this.constructor.onsend)
  this.constructor.onsend.apply(this, arguments);

 // Invoke real method
 this.object.send(vData);
}
...
```

Now we have a cross-browser implementation of XMLHttpRequest object that also allows sniffing of its instances transparently.

Getting the XMLHttpRequest object working with FireBug

The FireBug extension for Firefox does some evil with XMLHttpRequest object by also wrapping it. Having another object wrapped for the second time actually breaks Firebug that in its own turn breaks the page display. The exception you would get has the following content: *"win.XMLHttpRequest.wrapped has no properties"* Fortunately there is a neat trick - copy the original pointer to the wrapped object.

```
if (xXMLHttpRequest.wrapped)
 XMLHttpRequest.wrapped = xXMLHttpRequest.wrapped;
```

Bringing XMLHttpRequest object up to the standards

So far we've only re-implemented functionality already available in native XMLHttpRequest objects (actually some implementations do have a more advanced API including experimental / custom members). Now, if we were to fit and match our new-baked object to the standard we would only see one missing set of API: static properties (UNSENT, OPENED, HEADERS_RECEIVED, LOADING, DONE). Thanks to the fact the object is now a real Javascript object, we can easily add them.

```
XMLHttpRequest.UNSENT  = 0;
XMLHttpRequest.OPENED  = 1;
XMLHttpRequest.HEADERS_RECEIVED = 2;
XMLHttpRequest.LOADING  = 3;
XMLHttpRequest.DONE  = 4;
```

Since we are using browser native XMLHttpRequest as the underlaying transport, our implementation is still affected by its bugs. In the following paragraphs we will see how all of these annoying bugs can be fixed.

2. Fixing native XMLHttpRequest bugs

Despite to the fact that most of native XMLHttpRequest object implementations do not differ much in their API, they still differ a lot in their behavior. Some implementations suffer from sever issues such as memory leak in Internet Explorer or missing readystatechange calls for synchronous requests in Gecko. There are other minor issues as well. In this paragraph I'd like to highlight issues considered.

Internet Explorer: memory leak in XMLHttpRequest (on-page)

Bug: The instance of XMLHttpRequest doesn't get garbage collected in case you have a reference to the instance or to an other COM object (for example: DOM Node etc.) in its onreadystatechange handler, thus producing runtime memory leaks.

This bug can easily be solved by cleaning the onreadystatechange property when the object readyState has changed to DONE.

```
if (self.readyState == self.constructor.DONE)
 self.object.onreadystatechange = new Function;
```

Internet Explorer: memory leak in XMLHttpRequest (inter-page)

Bug: The instance of XMLHttpRequest doesn't get garbage collected on page unload in case you have a reference to the instance or to an other COM object (for example: DOM Node etc.) in its onreadystatechange handler, thus producing inter-page memory leaks.

Solving this bug can be done by registering an onunload handler in which we would clean the onreadystatechange property of the request in progress.

Internet Explorer: cached document is not checked against modification date

Bug: Internet Explorer does not issue a validation request to the server if resource is available from its cache, no matter what expiration date this file was set to.

To work around the issue we will make an additional request to the server with proper "If-Modified-Since" header set in case the data was found in browser cache (can be verified by checking the presence of "Date" field in the response headers). If the server responds with status 304 (Not modified), we can safely use data loaded from cache.

Gecko: missing readystatechange calls in synchronous requests

Bug: When the request is synchronous, no readystatechange is fired.

Thanks to the fact that we have full control over XMLHttpRequest object now, we can easily simulate missing calls to the onreadystatechange handlers. Worth noting, the Firebug also affects behavior of the native XMLHttpRequest object. When Firebug is installed and is enabled it solves the issue partially by sending DONE readystatechange (while still missing the other state changes).

Gecko: unnecessary readystatechange DONE when request aborted

Bug: When the request is aborted, Gecko always fires readystatechange DONE.

To workaround this issue we can simply ignore the readystatechange call (in our implementation) made by Gecko if the abort method was called.

Gecko: Annoying <parsererror...>...</parsererror> documents

Bug: If the content retrieved by XMLHttpRequest was invalid with respect to XML well-formedness, Gecko returns a DOM document with an error report wrapped instead of null.

To solve that issue, we can simply check if the documentElement of the responseXML body is "parsererror" element, and if it is, reset the property to the value of null.

```
// Public Methods
XMLHttpRequest.prototype.open = function(sMethod, sUrl, bAsync, sUser, sPassword) {
...
 var self = this;
...
 this.object.onreadystatechange = function() {
 ...
  if (self.readyState == self.constructor.DONE) {
  ...
   // BUGFIX: Annoying <parsererror /> in invalid XML responses
   if (self.responseXML && self.responseXML.documentElement.tagName == "parsererror")
    self.responseXML = null;
  }
 ...
 }
}
```

Internet Explorer + Gecko: duplicate readystatechange OPEN triggered

Bug: Internet Explorer and Gecko web-browsers both issue an unnecessary OPEN readystatechange right after the send method is invoked.

Solution is simple - cache the readyState changes and only dispatch unique ones.

```
XMLHttpRequest.prototype.open = function(sMethod, sUrl, bAsync, sUser, sPassword) {
...
 var self = this;
 var nState = this.readyState;
...
 this.object.onreadystatechange = function() {
 ...
  if (nState != self.readyState)
   fReadyStateChange(self);

  nState = self.readyState;
 }
...
}
```

Internet Explorer: responseXML is not properly initialized for application/xxx+xml responses

Bug: Internet Explorer does not recognize retrieved XML document as such in case the later was served with application/soap+xml content-type.

Solving the issue is simple - let's parse the responseText into an XML document.

```
if (bIE && !oRequest.responseXML.documentElement && oRequest.getResponseHeader("Content-Type").match(/[^\/]+\/[^\+]+\+xml/)) {
 oRequest.responseXML = new ActiveXObject('Microsoft.XMLDOM');
 oRequest.responseXML.loadXML(oRequest.responseText);
}
```

Safari: fails sending document created/modified dynamically

Bug: Safari chokes on sending Document if it was modified dynamically, for example by setting some namespaced attribute.

The issue is fixed by manual serialization (with XMLSerializer) of Node passed into the send method along with setting appropriate Content-Type (application/xml).

Internet Explorer: custom Content-Type overridden when sending XML nodes

Bug: Internet Explorer overrides any custom Content-Type header with "text/xml" value when sending XML nodes.

The only way to approach solving this issue is following previous bugfix - sending serialized Node with proper Content-Type set manually.

Appendix A. Usage Examples

The examples given below demonstrate normal programming techniques used by developers when working with the XMLHttpRequest object. You can see that there is no debugging code put into the samples, however whenever the test is executed (by pressing 'click to run!' button) you can see XMLHttpRequest objects activities reported to the "XMLHttpRequest calling log" below. This is enabled by having class handlers set to XMLHttpRequest object.

```
function fReport(oSelf, sAction, oArguments) {
 var oElement = document.getElementById("log").appendChild(document.createElement("xmp"));
 oElement.innerHTML = new Date() + ' - Called "' + sAction + '" with arguments: (' + oArguments + ')';
}

XMLHttpRequest.onreadystatechange = function() {
 fReport(this, 'readystatechange [' + this.readyState + ']', []);
}
XMLHttpRequest.onopen = function(sMethod, sUrl, bAsync) {
 fReport(this, 'open', [sMethod, sUrl, bAsync]);
}
XMLHttpRequest.onsend = function(vData) {
 fReport(this, 'send', [vData]);
}
XMLHttpRequest.onabort = function() {
 fReport(this, 'abort', []);
}
```

Asynchronous GET XMLHttpRequest

This test demonstrates an asynchronous GET request call.

show/hide source code

click to run!

Synchronous POST XMLHttpRequest

This test demonstrates a synchronous POST request call.

show/hide source code

click to run!

Synchronous GET XMLHttpRequest

This test demonstrates a synchronous GET request call.

show/hide source code

click to run!

Asynchronous GET XMLHttpRequest with aborting

This test demonstrates an asynchronous GET request call **with abort**. See that readyState DONE is not fired.

show/hide source code

click to run!

XMLHttpRequest calling log

Please invoke the examples above and see here the log of operations.

Appendix B. XMLHttpRequest implementations compared

Browser	Availability	Standard API	Context	Leaks	Send execution sequence		Abort execution sequence (abort is called after send)
					Asynchronous	Synchronous	
XMLHttpRequest.js	available	Standard-compliant (*)	instance	no	O, R1, S, R2, R3, R4	O, R1, S, R2, R3, R4	O, R1, S, (R2, R3)?, A, (R4)?
Firefox 2.0 / 3.0	available	Missing Constants	handler	no	O, R1, S, R1, R2, R3, R4	O, S	O, R1, S, R1, (R2, R3)?, A, R4
Firefox X.0 (Firebug)	available	Missing Constants	handler	no	O, R1, S, R1, R2, R3, R4	O, S, R4	O, R1, S, R1, (R2, R3)?, A, R4

Internet Explorer 6.0-	missing (**)	Missing Constants	window	yes	O, R1, S, R1, R2, R3, R4	O, R1, S, R1, R2, R3, R4	O, R1, S, R1, (R2, R3)?, A, (R4)?
Internet Explorer 7.0	available	Missing Constants	window	yes	O, R1, S, R1, R2, R3, R4	O, R1, S, R1, R2, R3, R4	O, R1, S, R1, (R2, R3)?, A, (R4)?
Opera 9.2	available	Missing Constants	instance	no	O, R1, S, R2, R3, R4	O, R1, S, R2, R3, R4	O, R1, S, (R2, R3)?, A, (R4)?
Safari 3.0	available	Missing Constants	instance	no	O, R1, S, R2, R3, R4	O, R1, S, R2, R3, R4	O, R1, S, (R2, R3)?, A, (R4)?

(*) The implementation described doesn't throw exceptions on its properties setting/getting as it is defined by the standard
(**) In Internet Explorer 6.0 and older versions XMLHttpRequest object is accessible only via Automation Server

Appendix C. Source Code

The source code containing the implementation described above, can be obtained from XMLHttpRequest project.