Compare JavaScript frameworks
An overview of the frameworks that greatly enhance JavaScript development

Joe Lennon
February 02, 2010

Modern Web sites and Web applications tend to rely quite heavily on client-side JavaScript to provide rich interactivity, particularly through the advent of asynchronous HTTP requests that do not require page refreshes to return data or responses from a server-side script or database system. In this article, you will discover how JavaScript frameworks make it easier and faster to create highly interactive and responsive Web sites and Web applications.

Introduction

JavaScript is an object-oriented scripting language that has long been the client-side scripting interface of choice for Web browser applications. JavaScript lets Web developers programmatically work with objects on a Web page, providing a platform for manipulating these objects on-the-fly. When JavaScript was first introduced, it was commonly used to provide trivial features on Web pages such as clocks and scrolling text in the browser status bar. Another common feature was the "rollover link," where an image or text color of a link would be changed when the user rolled their mouse over it. In recent times, however, JavaScript has evolved to become far more useful, with the concept of Asynchronous JavaScript and XML (Ajax) bringing a whole new level of interactivity to Web-based programming. Prior to Ajax, any server-side processing or database access would require the entire page to be "refreshed" or a new page to be rendered by the browser. Not only is this slow and frustrating for the user, but it is also a waste of bandwidth and resources.

Ajax stands for Asynchronous JavaScript and XML, although the reference to XML is no longer valid as Ajax requests can return responses in several formats other than XML, for example JSON (JavaScript Object Notation). Ajax works by allowing JavaScript to asynchronously submit an HTTP request to the Web server, and to render its response without refreshing or rendering a new page. Instead, the developer would typically use DOM (Document Object Model) manipulation to modify part of the Web page to reflect the changes or data returned by the HTTP response.
What is a JavaScript framework?

JavaScript on its own is a very powerful language, and you do not need any additional frameworks to create Rich Internet Applications (RIA) that are powered by it. Using JavaScript, however, is not an easy task, primarily due to the complications that arise when trying to provide support for multiple Web browsers. Like HTML and CSS, different browsers have separate implementations of JavaScript, and it can be a nightmare to ensure that your JavaScript code is cross-browser compatible.

A JavaScript framework or library is a set of utilities and functions that make it much easier to produce cross-browser compatible JavaScript code. Each library is vigorously tested on modern versions of many popular Web browsers, and as such, you can be confident that by using one of these frameworks, that your JavaScript-based RIA will work similarly across different browsers and platforms.

In addition to taking care of cross-browser issues, JavaScript frameworks make it easier to write code that retrieves, traverses, and manipulates DOM elements. Not only do they provide shorthand functions to obtain a reference to a DOM element, but they also allow for the daisy-chaining of DOM traversal functions to find parent, child, or sibling elements of any depth. Finally, the frameworks provide a series of functions that make it easier to manipulate these objects, allowing the content itself to be changed, added, or removed; or the CSS styling of classes to be manipulated to impact the look of the element.

Another important feature of JavaScript frameworks is its improved support for event handling. Cross-browser event handling can be a nightmare due to the differing implementations between browsers, so JavaScript frameworks typically wrap browser events and provide a series of useful cross-browser compatible functions for dealing with them. Some frameworks also provide a series of standardized key codes that represent keyboard-based events (such as hitting the Escape key, Return key, cursor keys, and so forth).

All of these features are very useful, but there is one aspect to JavaScript frameworks that has been instrumental in their recent popularity—Ajax support. Like many other aspects of JavaScript, each Web browser tends to support Ajax in a different manner, making it tedious to work with Ajax in a way that is supported across all Web browsers. Almost all JavaScript frameworks include some form of Ajax library support that usually provides Ajax request and response objects, with helpers for evaluating responses, updating DOM elements, and polling a particularly request.

Typical features of a JavaScript framework

Now, let's take a look at some of the useful features common to most JavaScript frameworks. These features are:

- Selectors
- DOM traversal
- DOM manipulation
- Utility functions
- Event handling
• Ajax

In the explanations of each of these features, I will provide an example from one or more of the following JavaScript frameworks: Prototype, jQuery, YUI, ExtJS, and MooTools. Although each individual's implementation and syntax will vary, the concept is generally the same. Each framework has a detailed API reference that you can refer to in order to determine how to use these features in that particular library.

Selectors

Most of the JavaScript frameworks available implement some form of support for rapid element selection. Generally speaking, these selectors make the process of obtaining a reference to an HTML element much faster, and allows you to search for the element by ID, class name, element type, or by using a series of pseudo-selectors.

For example, using regular JavaScript, you might select a DOM element by its ID using the following code:

```javascript
var theElement = document.getElementById('the_element');
```

MooTools, like several other frameworks, provides a shorthand method of performing this action. In addition to selecting the element, MooTools also extends the element with a series of MooTools utility functions. The syntax for doing so is:

```javascript
var theElement = $('the_element');
```

The dollar function, as shown above, is available in several (but not all) of the popular JavaScript frameworks, and the syntax is generally similar. The Prototype library goes a step further, allowing multiple elements to be selected by ID at any one time, and returning an array of elements. Like MooTools, these elements are extended with Prototype utility functions. The syntax for selecting multiple elements at one time with Prototype is:

```javascript
var elementArray = $('element_one', 'element_two', 'element_three');
```

In the section Utility functions you will learn about some functions provided by JavaScript frameworks that make it easier to iterate over collections.

In the previous examples, you must provide the ID of the element you want to select. What if you wanted to selected multiple elements, however, for example, all images, or every table row with a particular CSS class name? MooTools (and some of the other libraries) provide a very simple method to do this—the dollar-dollar function. This works in a similar way to the dollar function, except it accepts CSS element names, class names, and pseudo selectors as an argument rather than an element ID. For example, to select all images on a Web page using MooTools, you would use the following code:

```javascript
var allImages = $$('img');
```
This returns an array of all images in the document, each extended by the dollar function to include MooTools utility functions.

Selecting elements by a tag name is useful, but what if you only want to select a subset of elements, based on their CSS class? This is also very simple. In the following example, MooTools will select every table row with the CSS class "odd," which could be useful for implementing row striping (alternating background colors between table rows):

```javascript
var allOddRows = $$('tr.odd');
```

In fact, MooTools provides an even better way of performing row striping. In the previous example, it was assumed that each odd row in the table was given the CSS class name "odd." The following code does not require any CSS class names to be defined on the table rows:

```javascript
var allOddRows = $$('tbody:odd');
```

This is an example of a pseudo-selector, which returns any objects that match the specification, in this case, every odd child element of the `tbody` (table body) elements on the page. Other examples of MooTools pseudo-selectors include:

- **checked**— select all elements that are checked (for example, checked checkboxes)
- **enabled**— select all elements that are enabled
- **contains**— select all elements that contain the text passed to the selector as an argument (for example, `contains('this text')`)

As mentioned earlier, not all JavaScript frameworks use the dollar function for DOM element selection. In version 3 of the YUI (Yahoo! User Interface) library, the following code is used to select an element by its ID (note that YUI 3 requires you to pass the ID selector character # before the ID itself):

```javascript
var theElement = Y.one('#the_element');
```

Similarly, rather than using the dollar-dollar function to retrieve elements by tag or class name, in YUI you use the `Y.all` function instead:

```javascript
var allOddRows = Y.all('tr.odd');
```

ExtJS works in a similar manner, with the following syntax for selecting an element by ID:

```javascript
var theElement = Ext.get('the_element');
```

And the following syntax for selecting by tag and class name:

```javascript
var allOddRows = Ext.select('tr.odd');
```
In the next section, you will see how JavaScript frameworks let you easily traverse the DOM, in other words, find elements that have a parent, child, or sibling relationship with a selected element. You will also learn about how the libraries let you manipulate the DOM to modify elements.

**DOM traversal**

Finding an element based on its ID, element type, or CSS class name is very useful, but what if you want to find an element based on its position in the DOM tree. In other words, say you have a given element and you want to find its parent element, one of its child elements, or its previous or next sibling element. For example, take the following HTML snippet shown in Listing 1.

**Listing 1. HTML snippet (an HTML Table)**

```
<table>
  <thead>
    <tr>
      <th>Name</th>
      <th>Email Address</th>
      <th>Actions</th>
    </tr>
  </thead>
  <tbody>
    <tr id="row-001">
      <td>Joe Lennon</td>
      <td>joe@joelennon.ie</td>
      <td><a href="#">Edit</a> &nbsp; <a href="#">Delete</a></td>
    </tr>
    <tr id="row-002">
      <td>Jill Mac Sweeney</td>
      <td>jill@example.com</td>
      <td><a href="#">Edit</a> &nbsp; <a href="#">Delete</a></td>
    </tr>
  </tbody>
</table>
```

Listing 1 uses indentation to illustrate where each element appears in the DOM node tree. In this instance, the `table` element is the root element, with two child nodes, `thead` and `tbody`. The `thead` element has a single child node, `tr`, with three children — all `th` elements. The `tbody` element has two children, both `tr` elements, each with three children. The third child in each of these rows contains two further children, both of which are `a` (anchor) tags.

As you know, you can use the selector function from a JavaScript framework to easily select an element by its ID. In this instance, you have two elements with IDs — both `tr` (table row) elements with the IDs row-001 and row-002, respectively. To select the first `tr` element using Prototype, you would use the following code:

```javascript
var theRow = $('row-001');
```

In the previous section, you also learned that you can use selectors to retrieve elements based on their type or CSS class. In this instance, you can select all `td` elements using the following syntax:

```javascript
var allCells = $$('td');
```
The problem with this code is that this returns each and every td element in the DOM. But, what if you only want to get those td elements that are in the row with ID row-001? This is where DOM traversal functions come into play. First, let's use Prototype to select all of the children of the tr element with ID row-001:

```javascript
var firstRowCells = theRow.childElements();
```

This returns an array of the child elements of the theRow variable (which you earlier set to the element with ID row-001).

Next, let's assume that you only want to get the first child element of the row, in this case, the td element with the content "Joe Lennon." To do this, you use the following statement:

```javascript
var firstRowFirstCell = theRow.down();
```

That's easy, huh? This particular usage is the equivalent of:

```javascript
var firstRowFirstCell = theRow.childElements()[0];
```

and can also be expressed as:

```javascript
var firstRowFirstCell = theRow.down(0);
```

JavaScript indexes begin with zero (0), so the above statement is essentially telling JavaScript to select the first child. To select the second child (the cell containing the e-mail address joe@joelennon.ie) you can use:

```javascript
var firstRowSecondCell = theRow.down(1);
```

Alternatively, you can navigate the DOM from sibling to sibling. In this case, the second cell is the next sibling to the first cell, so you can use the following statement:

```javascript
var firstRowSecondCell = firstRowFirstCell.next();
```

This works in the same way as the down() function, so to select the third cell, you would use:

```javascript
var firstRowThirdCell = firstRowFirstCell.next(1);
```

In addition to using indexes to find which particular node to find, Prototype also allows you to use the CSS selector syntax. Given the example in Listing 1, let's find the second link (the "Delete" link) for the row containing Jill Mac Sweeney's details:

```javascript
var secondRowSecondLink = $('row-002').down('a', 1);
```

In this instance, you are using the dollar function to find the element with ID row-002, and traversing down the DOM to the second next descendent a (anchor) element.
Some frameworks also allow you to "daisy-chain" traversal functions, meaning you can concatenate traversal commands to one another. Another way of expressing the previous example in Prototype would be:

```javascript
var secondRowSecondLink = $('row-002').down('a').next();
```

Consider the following example:

```javascript
var domTraversal = $('row-001').down().up().next().previous();
```

As you can see, daisy chaining allows you to concatenate several DOM traversal statements together. In fact, the above example actually ends up selecting the `tr` element `row-001`, so the daisy chain ended up right back where it started!

## DOM manipulation

Previously, you saw how selectors and DOM traversal make it easy to select particular elements using JavaScript frameworks. In order to change the appearance or content of a particular element on a Web page, however, you need to manipulate the DOM and apply that change. This can be quite cumbersome using pure JavaScript, but, thankfully, most JavaScript frameworks offer helpful functions that make it easy to do so.

Say you have a `div` element with the id `the-box`:

```html
<div id="the-box">Message goes here</div>
```

To change the text "Message goes here" using jQuery, you can simply use:

```javascript
$('#the-box').html('This is the new message!');
```

In fact, you can use HTML code inside this function and it will be parsed by the browser. For example:

```javascript
$('#the-box').html('This is the &lt;strong&gt;new&lt;/strong&gt; message!');
```

In this instance, the contents of the `div` element as it is represented in the DOM will look like:

```html
<div id="the-box">This is the <strong>new</strong> message!</div>
```

Of course, there may be instances in which you need to actually use special characters like the greater than or less than symbol. Rather than specifying the special HTML entity code, you can use jQuery's `text` function instead:

```javascript
$('#the-box').text('300 > 200');
```

This will update the `div` element to the following code:

```html
<div id="the-box">300 > 200</div>
```
In the examples above, the existing content is replaced by the new content. What if you wish to merely append the message to the text? Fortunately, jQuery offers the append function for this exact purpose:

```javascript
$('the-box').append(', here goes message');
```

Applying this to the original `div` element, the contents of the `div` element would now look like:

```html
<div id="the-box">Message goes here, here goes message</div>
```

In addition to appending, you can "prepend" content so that it is inserted before the existing content rather than after it. Additionally, jQuery offers functions for inserting the content outside of the given element, either before or after it. There are functions for replacing content, emptying content, removing elements from the DOM altogether, cloning elements, and more.

In addition to DOM manipulation functions, JavaScript frameworks usually contain several functions for manipulating the styles and CSS classes of an element programmatically. For example, let's say you have a table and you want to highlight a row when the mouse moves over it. You create a special CSS class named `hover` and you want to dynamically add this class to a row. In YUI you could use the following code to check if the row already has the `hover` class and remove it if it does and add it if it doesn't:

```javascript
if(row.hasClass('hover')) row.removeClass('hover'); else row.addClass('hover');
```

Again, most JavaScript frameworks have functions for CSS manipulation built-in.

**Utility functions**

Many JavaScript frameworks come with a myriad of utility functions that make developing applications using JavaScript easier. There are far too many to cover in this article, so I'll discuss one of the more noteworthy functions common to most frameworks.

If you have worked with arrays in JavaScript, you may be familiar with using `for` loops to iterate through the array to work with its values. For example, consider the code in Listing 2:

**Listing 2. Traditional method of looping through a JavaScript array**

```javascript
var fruit = ['apple', 'banana', 'orange'];
for(var i = 0; i < fruit.length; i++) {
    alert(fruit[i]);
}
```

The code in Listing 2 is fine, but it's a bit cumbersome. Most JavaScript frameworks include an `each` function, which calls a specified function for each element in an array. Using MooTools, you can perform the same action as in Listing 2 using the code in Listing 3.

**Listing 3. Using the `each` function in MooTools**

```javascript
['apple', 'banana', 'orange'].each(function(item) {
    alert(item);
});
```
The syntax in Listing 3 is identical to that used in Prototype and jQuery, while it varies slightly for the likes of YUI and ExtJS. The syntax is different across frameworks, however, when it is applied to a hash map or object rather than an array. In MooTools, for example, you might use the following code from Listing 4.

**Listing 4. Using each on a key/value pair-based object in MooTools**

```
var hash = new Hash({name: "Joe Lennon", email: "joe@joelennon.ie"});
hash.each(function(value, key) {
    alert(key + " : " + value);
});
```

In Prototype, however, this would look more like the code in Listing 5.

**Listing 5. Using each on a key/value pair-based object in Prototype**

```
var hash = $H({name: "Joe Lennon", email: "joe@joelennon.ie"});
hash.each(function(pair) {
    alert(pair.key + " : " + pair.value);
});
```

Each framework contains many more useful functions, usually divided up into **String** functions, **Number** functions, **Array** functions, **Hash** functions, **Date** functions, and so forth. For further information, please see the API reference for the relevant JavaScript framework.

**Event handling**

Each JavaScript framework implements cross-browser event handling support that encourages you to move from the old-style inline attaching of events to a streamlined method. Consider the jQuery example in Listing 6, where you highlight a **div** element based on the hover event.

**Listing 6. Attaching the hover Event using jQuery**

```
$('#the-box').hover(function() {
    $(this).addClass('highlight');
}, function() {
    $(this).removeClass('highlight');
});
```

This is a special event implemented by jQuery, and you’ll notice that it provides for two functions, the first is called when the **onMouseOver** event is triggered, and the second is called when the **onMouseOut** event is triggered. This is because there is no standard DOM event for **hover**. Let’s take a look at a more typical event, such as **click** (see Listing 7).

**Listing 7. Attaching the click Event using jQuery**

```
$('#the-button').click(function() {
    alert('You pushed the button!');
});
```

As you can see, in this instance there is a single function argument. Most JavaScript events are treated in this way by jQuery. When using event handlers in jQuery, the context variable refers to the element that fired the event. Some frameworks do not work in this manner. Take Prototype, for example, whose equivalent to Listing 7 looks like the code in Listing 8.
Listing 8. Attaching the click Event using Prototype

```javascript
$('the-button').observe('click', function(e) {
    alert('You pushed the button!');
});
```

The first thing you'll notice is that there is no `click` function, but rather an `observe` function that takes the event as an argument before the function reference itself. You'll also notice that the function takes an argument, `e`. This is the context variable that refers to the element that fired the event. To see how it works, let's rewrite the code from Listing 6 for Prototype (see Listing 9).

Listing 9. Attaching the hover Event using Prototype

```javascript
$('the-box').observe('mouseover', function(e) {
    var el = Event.element(e);
    el.addClassName('highlight');
});
$('the-box').observe('mouseout', function(e) {
    var el = Event.element(e);
    el.removeClassName('highlight');
});
```

Unlike jQuery, where you simply use the dollar function to get the context variable, in Prototype you need to use the `Event.element()` function. Also, you'll see that you need to separate the functions for `mouseover` and `mouseout`.

You may have noticed through the course of this article that the functions are created inline and are not named. This means they cannot be reused. This Prototype hover example provides an opportunity to show how you could use named functions instead. Listing 10 illustrates an example of this.

Listing 10. An improved hover example in Prototype

```javascript
function toggleClass(e) {
    var el = Event.element(e);
    if(el.hasClassName('highlight'))
        el.removeClassName('highlight');
    else
        el.addClassName('highlight');
}

$('the-box').observe('mouseover', toggleClass);
$('the-box').observe('mouseout', toggleClass);
```

You'll notice that this time you simply define a single function that is called by both the `mouseover` and `mouseout` events. This function then determines if the element already has the class name `highlight`, and removes or adds based on its findings. You'll notice that the argument `e` is passed implicitly. In other words, you do not explicitly pass the event as an argument in the `observe` function.

**Ajax**

One of the most compelling reasons to use a JavaScript framework is for standardized cross-browser Ajax requests. An Ajax request is an asynchronous HTTP request that is usually made...
to a server-side script, which returns a response in a format such as XML, JSON, HTML, or plain text. Most JavaScript frameworks have some form of Ajax object with a request method that accepts a series of options as an argument. These options generally include callback functions, which are called as soon as the script receives a response from the Web server. Let's look at what an Ajax request looks like in ExtJS, MooTools, and Prototype.

First off, take a look at a typical ExtJS Ajax request (see Listing 11).

**Listing 11. An ExtJS Ajax request**

```javascript
Ext.Ajax.request({
    url: 'server_script.php',
    params: {
        name1: 'value1',
        name2: 'value2'
    },
    method: 'POST',
    success: function(transport) {
        alert(transport.responseText);
    }
});
```

The request method in ExtJS takes a single argument, an object with various fields including `url`, `params`, `method`, and `success`. The `url` field contains the URL of the server-side script that will be called by the Ajax request. The `params` field is itself an object, consisting of key/value pairs that will be passed to the server-side script. The `method` field can be one of two values: `GET` or `POST`. It defaults to undefined, but will be treated as `POST` by default if there are `params` in the request. The final field, `success`, is a function that will be called when the Web server returns a successful response. In this instance, it is assumed that the server-side script returns plain text, and this text is shown to the user via an alert box.

Next, let's investigate what the same request would look like in MooTools (see Listing 12).

**Listing 12. A MooTools Ajax request**

```javascript
new Request({
    url: 'server-script.php',
    data: {
        name1: 'value1',
        name2: 'value2'
    },
    method: 'post',
    onComplete: function(response) {
        alert(response);
    }
}).send();
```

As you can see, MooTools is very similar to ExtJS. You'll notice that rather than `params`, variables are passed using the `data` field, and the method needs to be specified in lowercase. Also, rather than a success `callback` function, MooTools uses an `onComplete` function. Finally, unlike ExtJS, in MooTools you actually need to send the request by using the `send()` function of the Request object.

Finally, let's see whether Prototype is drastically different (see Listing 13).
Listing 13. A Prototype Ajax request

```javascript
new Ajax.Request('server-script.php', {
  params: {
    name1: 'value1',
    name2: 'value2'
  },
  method: 'post',
  onSuccess: function(transport) {
    alert(transport.responseText);
  }
});
```

Once again, Prototype works along the same lines, but with small syntactical differences. For starters, Prototype's `Request` object takes two parameters to its constructor. The first parameter is the URL the request is to be made to, and the second parameter is an object with the typical Ajax options, as seen in the previous two. Of course, with the URL now being passed as a separate parameter, it does not feature in the list of options. Also, it's worth noting that unlike MooTools, the Prototype Ajax `Request` object constructor implicitly sends the request, so you do not need to call any methods to actually initiate the HTTP request.

The Ajax support in most JavaScript frameworks goes even further than what has been illustrated here. Some notable enhancements include automatically updating an element with the response received, without requiring any special `onSuccess` functions. Some libraries even include pre-built auto-completion functions, as you might have seen on the Google search engine, where typical search queries are suggested as you type.

In the next section, you will learn about the User Experience (UX) enhancements that JavaScript frameworks offer to Web developers.

**UX enhancements**

Up until now, this article has focused entirely on the programmatic benefits of using JavaScript frameworks and how they can make coding interactive applications easier. With most frameworks, however, there is another aspect that makes their use an attractive prospect: User Interface (UI) components and User Experience enhancements that can be easily included and which would usually take a significant effort to build.

This section will explore the UX enhancements individually for each of the following frameworks: Prototype, jQuery, YUI, ExtJS, and MooTools.

**Prototype**

Prototype is one of the few JavaScript frameworks that does not include UI components or UX enhancements out-of-the-box. Instead, it defers all of this to its sister library, script.aculo.us (or Scripty2, as the latest version is known). Script.aculo.us adds support for a wide range of effects and behaviors in Prototype. These include effects like highlighting, morphing, folding, shaking, sliding, puffing, and more. Script.aculo.us also boasts drag-and-drop support, as well as sliders, in-place Ajax editors, and autocompleters. Unlike some frameworks, Script.aculo.us leaves the design of any controls (such as sliders and autocompleters) to the developer, providing no standard skin to work from.
jQuery

Unlike Prototype, jQuery includes some basic UX enhancements with the core library. These are similar to some of the basic effects that come with script.aculo.us, such as sliding and fading. For more advanced UX features, however, you need to download the jQuery UI library, which includes more effects than the jQuery core, as well as interactivity features such as drag and drop, resizing, and sorting. Unlike script.aculo.us, the jQuery UI also includes some widgets or components that make the development of attractive interfaces much easier. At present, these components include Accordion, Datepicker, Dialog, Progressbar, Slider, and Tabs. These widgets are completely themable, and jQuery UI includes a wide range of themes that can be used to fit the components to your own particular Web site or Web application. Figure 1 illustrates an example of a jQuery UI Datepicker widget, in the Cupertino theme.

Figure 1. jQuery UI Datepicker Widget

YUI

Whereas Prototype and jQuery do not include UI widgets out-of-the-box, the Yahoo! User Interface library (YUI) includes a bucketload. In addition to drag-and-drop support and resizing, version 2 of YUI includes autocompleters, calendar controls, carousel components, charting, dialogs, progress bars, rich text editors (or WYSIWYG text areas), sliders, tabs, trees, and more. At the time of this article, none of the aforementioned widgets have been included in version 3 of YUI. Figure 2 is a sample of a series of these components in action together.
ExtJS

Like YUI, ExtJS includes a plethora of components out-of-the-box, most notably some very powerful grid controls with support for inline editing, pagination, filtering, grouping, summaries, buffering, and data binding. ExtJS components have a professional look-and-feel and are completely themable. Some of the other widgets that come with ExtJS include tabs, charts, windows (dialogs), trees, layout managers, enhanced form controls, toolbars and menus, drag and drop, and direct remoting. This really only scratches the surface of just how much ExtJS has to offer, so if you are looking for an extensive set of RIA components, be sure to check out the ExtJS Website. Figure 3 shows a sample Web Desktop that has been developed in ExtJS, illustrating just how feature-rich the library is.
Figure 3. ExtJS Desktop Sample Application

MooTools

MooTools is aligned with Prototype and jQuery in that it does not include UI controls and widgets out-of-the-box. Like jQuery, it chooses to include a limited set of effects. It includes more advanced UX enhancements through the MooTools.More.js extension. This extension is not simply a UX/UI plug-in, however, and also includes some interesting add-ons to the Array, Date, Hash, and String classes, as well as some additional Element extensions. In terms of UX enhancements, MooTools.More.js includes drag-and-drop support and some additional effects. The add-on also boasts some UI controls such as accordion, sortable HTML tables, scrollers, tooltips, and Ajax spinner overlays. Like script.aculo.us, however, you are left to implement the design of these controls for yourself.

Framework comparison

Table 1 provides a detailed comparison of features available in the five frameworks discussed in this article.

Table 1. Framework feature comparison

<table>
<thead>
<tr>
<th></th>
<th>Prototype</th>
<th>jQuery</th>
<th>YUI</th>
<th>ExtJS</th>
<th>MooTools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latest Version</td>
<td>1.6.1</td>
<td>1.4.1</td>
<td>3.00</td>
<td>3.1</td>
<td>1.2.4</td>
</tr>
</tbody>
</table>
Other notable frameworks

For the purpose of brevity, this article examined five of the more popular JavaScript frameworks available. This does not mean that these are necessarily the most popular, the best, or the most feature-rich.

Some other JavaScript frameworks that deserve a mention and are worth a look include:

- Cappuccino
- Dojo
Summary

In this article, you have been introduced to the concept of a JavaScript framework and learned why it is beneficial in the development of Web sites and Web applications. You were given an overview of some of the common features of these frameworks, with some examples of how they might be implemented in the more popular libraries. You then learned about the UX and UI enhancements that come as part of or as a separate add-on to each of the five frameworks covered in this article. Finally, you were presented with a straight-talking feature comparison chart that highlights what options each framework does and does not have. With this information, you should be ready to further investigate these frameworks before making an informed decision on which one is right for you and your development team.
Related topics

- "Build apps using Asynchronous JavaScript with XML (AJAX)" (developerWorks, November 2005) explains how to build Ajax-based Web applications.
- Learn about Dojo.
- Read the article "Dojo concepts for Java developers" (developerWorks, October 2008) to bridge the gap from Java code to Dojo so that you can get up to speed quickly and use the toolkit when developing your applications.
- Download the Dojo toolkit. Dojo toolkit.
- Read "Call SOAP Web services with Ajax, Part 1: Build the Web services client" (developerWorks, October 2005) to learn how to implement a Web browser-based SOAP Web services client using the Ajax design pattern.
- Get the Prototype Javascript framework.
- Get jQuery.
- The jQuery User Interface provides abstractions for low-level interaction and animation, advanced effects and high-level, themeable widgets, built on top of the jQuery JavaScript Library.
- Get script.aculo.us.
- Get MooTools.
- Download and learn about the Yahoo! YUI Library.
- Get ExtJS.

© Copyright IBM Corporation 2010
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)