

Building efficient websites with AJAX

BREEZY BROWSING

Books were the original model for website design.

Navigation was similar to flipping the pages. Thanks to AJAX, many state-of-the-art websites now behave like desktop applications. **BY CARSTEN ZERBST**

The standards for an impressive Internet presentation have changed substantially since Tim Berners-Lee created the first web pages. Internet sites increasingly remind the surfer of interactive desktop applications rather than printed material. AJAX is a technology based on JavaScript that adds convenience, with pull-down menus, sortable tables, and interactive input pages. The main improvement is the absence of delays typically experienced while pages reloaded.

Long Way

Before rendering a website, the browser and web server go through a number of steps (Figure 1):

- The browser sends a page request to a web server.
- The server processes the request and serves up the HTML text and images. This might take a couple of seconds if the load is heavy. The network transmission speed decides how fast the content is delivered. The required time is still noticeable on fast intranets, however.
- Finally, the browser reads the response and displays the page. The same sequence is repeated for each image before the browser can render the final version of the page.

All told, these three steps typically take several seconds. In case of HTML pages without AJAX technology the steps are repeated for even the tiniest of changes.

In contrast to rich client applications, this considerably affects the user experience: Menus that drop down without a delay, point and click sorting in tables, or drag and drop are not easily implemented because of time-consuming page reloads. HTML pages that offer these kinds of features need to be autonomous, like local programs; that is, you should not need to rely on a server connection.

Removing Time-Consuming Requests

To improve the user experience, more and more web applications are starting to process user input directly browser-side and to do without time-consuming server requests.

Only two of the various techniques for browser-side data processing have achieved widespread success: JavaScript and Flash. Both are available on more than 90 percent of all computers. This means that web developers need not have any qualms about using them.

Other solutions, with the exception of the relatively widespread Java plugin, have been unable to achieve similar multi-

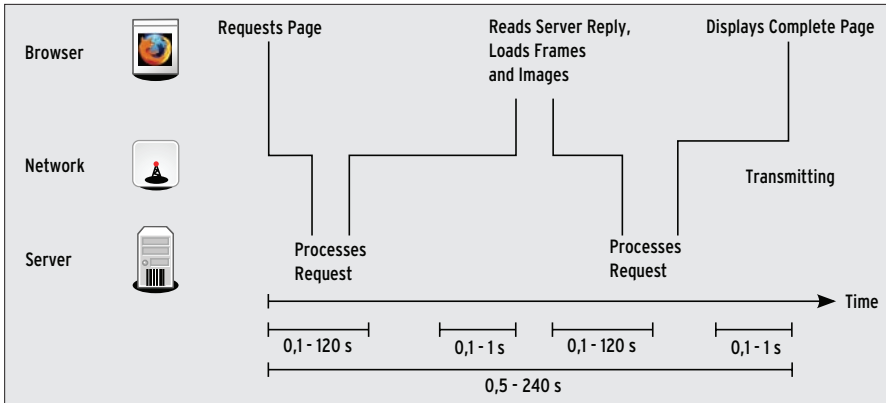


Figure 1: Pingpong: The exchange between browser and web server that occurs for each page change without AJAX typically takes several seconds.

platform success. Flash and JavaScript adopt completely different approaches.

Flash

The proprietary Flash plugin executes binary Flash applications in the browser. The plugin is embedded in the web page very much like a bitmap image, except that it offers the user an interactive interface. The Flash plugin has excellent graphics capabilities with virtually no restrictions on the developers' creativity. However, for lack of equivalent open source alternatives, there is virtually no alternative to the Adobe tools.

JavaScript

In contrast, JavaScript is not restricted to isolated areas of the web page. The interpreter that is integrated with the browser executes the program and converts the whole page into a dynamically modifiable interface. To do so, scripts create or modify the HTML code on the page, modify the cascading style sheet (CSS) styles, or even draw graphics.

The scripts themselves comprise uncompiled text. In contrast to Flash development, programming in JavaScript does not require any special tools. A simple text editor is fine for a start.

Good Tools

As always, good tools make programming easier. An editor with HTML, CSS, and JavaScript support is useful. It should also be able to handle source code that mixes all three [1][2]. Chris Pederick's Firefox plugin Web Developer [3] is the obvious choice for tracing and debugging programs. It reveals bugs in HTML, CSS, and JavaScript; investigates cookies; and displays the dynamically

modified HTML code, not just the original version delivered by the server.

Tables

An increasing number of applications, such as order processing and enterprise

resource planning (ERP) systems use web front ends: Parts lists or other list-type overviews are the main fare.

One important feature is the user's ability to sort these lists. If the web front end is based on static HTML, the server has to regenerate the page and serve up the modified version. Of course, client-side JavaScript-based data sorting makes for a smoother process.

Figure 2 gives an example of a directory listing implemented as an HTML table. Clicking the table header sorts the table by the current column. Figure 3 shows how to do this with JavaScript. An HTML table comprises nested `<tr>` and `<td>` elements, which can be addressed via DOM [4].

The script starts by dynamically removing all the `<tr>` elements (i.e., the rows in the table) from the page. The rows only exist as an array in JavaScript.

Listing 1: Sortable Table

```

01 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
02 <html>
03   <head>
04     <title>Sortable Table</title>
05     <meta http-equiv="Content-Type" content="text/html;
06       charset=UTF-8">
07     <link href="tabelle.css" rel="stylesheet" type="text/css" />
08     <script type="text/javascript" src="lib/MochiKit/MochiKit.
09       js"></script>
10     <script type="text/javascript" src="sortierbareTabelle.
11       js"></script>
12   </head>
13   <body>
14     <TABLE id="sortierbareTabelle" class="datagrid">
15       <THEAD>
16         <TH>Rights</TH> ... <TH mochi:format="int">Size</
17         TH><TH mochi:format="gdate">Change</TH>
18       </THEAD>
19       <TBODY>
20         <TR>
21           <TD>-rw-r--r--</TD><TD>1</TD><TD>root</
22           TD><TD>root</TD><TD>551</TD><TD>27.01.07</TD><TD>group</TD>
23         </TR>
24         ...
25       </TBODY>
26     </TABLE>
27   </body>
28 </html>

```

Rechte	Typ	Eigentümer	Gruppe	Größe	Änderung	Name
lrwxrwxrwx	1	root	root	18	11.02.08	printcap
-rw-r--r--	1	root	root	149	26.11.06	hosts.deny
-rw-r--r--	1	root	root	188	26.11.06	hosts.equiv
-rw-r--r--	1	root	root	191	26.11.06	hosts.lpd
-rw-r--r--	1	root	root	530	27.01.07	group.YaST2save
-rw-r--r--	1	root	root	536	27.01.07	group.old
-rw-r--r--	1	root	root	551	27.01.07	group
-rw-r--r--	1	root	root	677	19.12.06	hosts.YaST2save
-rw-r--r--	1	root	root	715	27.12.06	hosts
-rw-r--r--	1	root	root	1357	02.10.08	passwd
-rw-r--r--	1	root	root	2639	26.11.06	hosts.allow

Figure 2: New order: Clicking the table header tells the client-side JavaScript to sort the table on the requested column; there is no need to delay this process by talking to the server.

The JavaScript function then sorts the array on the requested column and writes the new sorting order between the empty `<table>` and `</table>` tags. Finally, the script draws an arrow, a Unicode arrow [5] character, before the label of the column on which the table is sorted.

Modular System

This method can be implemented with a few hundred lines of code; however, it is easier to re-sort to existing solutions. A dozen or so popular JavaScript libraries are on the Internet that include functions for frequently needed tasks, such as sorting tables, creating round corners in HTML, and drawing tree graphs.

The advantages of libraries compared with do-it-yourself solutions are in their enormous function scope and guaranteed compatibility with popular browsers. Although JavaScript conforms to the ECMA standard [6], the implementations in browsers on Linux, Mac OS X, and Windows differ in some major aspects. Most existing libraries abstract these differences to make life a little easier for developers.

Listing 1 shows the HTML code for a sortable table based on the free Mochikit [7] library. Besides the two includes in line 6 that bind Mochikit, the HTML source code is little different from a static table. The `table` tag needs a unique ID (`sortable_table`) as in the menu example here. The Mochikit-specific attributes `mochi:format = "int"` and `mochi:format = "gdate"` in the `<th>` tags help Mochikit sort numeric and date columns correctly.

Dynamic Data

The previous example stores the table content in the HTML source code; the following example takes things one step

further. The application picks up the table content without reloading the page from the server. Links, buttons, and menu items let the user fill the table with different values. Also, it is possible to display search results without page reloads.

Figure 4 illustrates the technology dis-

played here. The server delivers the data in JavaScript Object Notation (JSON), a text format that uses brackets and commas as separators.

JSON

XML is a popular alternative to JSON. JSON's advantage is the lower overhead compared with the unnecessarily verbose XML. The JavaScript `eval()` function converts the JSON code to normal JavaScript objects.

Listing 2 shows the HTML code. Instead of the table content, the code just has a `tbody` element as a placeholder. The JavaScript file referenced in the head of the table, `AjaxTabelle.js`, replaces the placeholder with new content with values from the JSON file (Listing 3).

If the lists are longer, it is faster to load just the first 25 entries. The user can then click a link or button to load the next 25. This approach reduces the wait time for the user as well as the server load.

Picture Perfect

The previous examples have been restricted to simple HTML elements such as lists and tables. Widgets and functions that HTML does not provide natively can be programmed in JavaScript in combination with cascading style sheets.

Querying the mouse position allows the developer to imple-

ment tool tips and even drag and drop. Most JavaScript libraries include implementations.

This does not exhaust the capabilities of dynamic HTML, including graphics that legacy HTML could only implement by embedding bitmaps.

Figure 5 shows a tree structure drawn by JavaScript. Client-side dynamic graphics have a number of advantages compared with bitmaps: The resolution is not fixed and can thus be modified to reflect the browser's preset font size.

User input can be visualized without server interaction, thus reducing the load on the server. Normally, much of a web server's bandwidth is consumed by serving up graphics.

HTML Extensions

Basically, there are two approaches to using JavaScript to create graphics. The greatest feature scope is offered by the XML-based graphics languages SVG [8], Canvas [9], and Vector Markup Language (VML) [10]. They can be embedded in HTML like normal graphics, but also can add typical graphical elements – such as lines, areas, or text at run time. They can thus respond interactively to user input like dynamic JavaScript-modified HTML.

Drawing programs such as Inkscape [11] or Karbon [12] will help you create a draft.

Unfortunately, multi-browser support is not guaranteed: None of the popular browsers support all three formats referred to here.

Firefox can handle SVG and Canvas, Safari just Canvas, and Internet Explorer just VML.

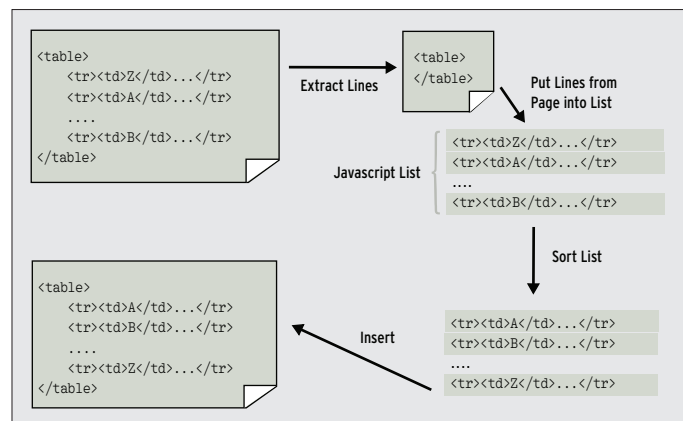


Figure 3: JavaScript sorts tables by dynamically removing the rows from the table, caching the results in an array, and reinserting the data in the new order.

Google JavaScript has two libraries for SVG [13] or Canvas [14]; however, support on the Windows platform is not guaranteed.

Tried and Trusted Tools

Many diagrams or graphs can be created with JavaScript and the standard HTML and CSS tools, such as the tree graph in Figure 5.

The boxes are made up of freely positioned CSS *div* elements. If *em* is used as a unit instead of *px* (pixel) for placing and sizing, the box height, width, and position are based on the size of the letter “m.”

The whole graph will then scale to reflect the text size without any effort on the developer’s part.

Users can also easily scale the graphic up and down in most browsers with the mouse wheel.

Connecting Lines

Drawing the connecting lines is slightly more difficult. Because JavaScript does not support graphical elements such as

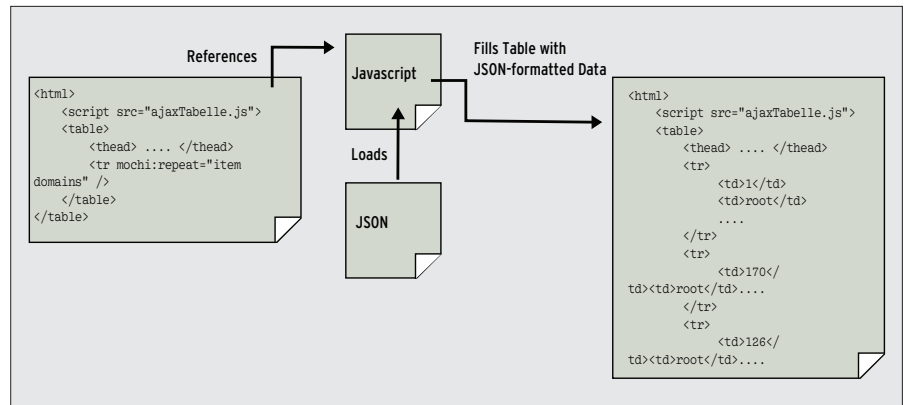


Figure 4: Even more dynamic: If the JavaScript picks up JSON-formatted data from the server, the data can be sorted and the table updated without reloading the page.

lines or circles, the trick here is to draw the graphical elements pixel by pixel as tiny *div* elements. Walter Zorn’s jsGraphics abstracts this complex process, giving developers basic shapes such as lines, circles, and polygons.

The *connect()* function relies on these to draw two boxes with lines. It ascertains the box positions at run time to allow the graphic to scale to the font size.

The *displayHierarchy()* function then redraws the connecting lines.

Lawrence Carvalhos’ TextResizeDetector [15] calls this function whenever the user changes the font size. The result is a loss-free, zoomable AJAX widget for tree structures that cannot be implemented using bitmaps.

Because it is entirely based on HTML elements, there is no need for extensions such as Canvas or VML that are not

Listing 2: Dynamically Generated Table

```

01 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
02 <html>
03   <head>
04     <link href="tabelle.css" rel="stylesheet" type="text/css" />
05     <script type="text/javascript" src="lib/MochiKit/MochiKit.js"></script>
06     <script type="text/javascript" src="AjaxTabelle.js"></script>
07   </head>
08   <body>
09     <a href="index.html">Examples</a>
10     <hr>
11     <h4>Ajax Table</h4>
12     <p>
13       <a href="top.json" mochi:dataformat="json">Reload Table 1</a><br>
14       <a href="top2.json" mochi:dataformat="json">Reload Table 2</a>
15     </p>
16     <table id="sortable_table" class="datagrid">
17       <thead>
18         <tr>
19           <th mochi:sortcolumn="PID int">PID</th>
20           <th mochi:sortcolumn="USER str">USER</th>
21           [...]
22           <th mochi:sortcolumn="COMMAND str">COMMAND</th>
23         </tr>
24       </thead>
25       <!-- replaced by the JSON file content -->
26       <tbody class="mochi-template">
27         <tr mochi:repeat="item domains">
28           <td mochi:content="item.PID"></td>
29           <td mochi:content="item.USER"></td>
30           [...]
31           <td mochi:content="item.COMMAND"></td>
32         </tr>
33       </tbody>
34     </table>
35   </body>
36 </html>

```

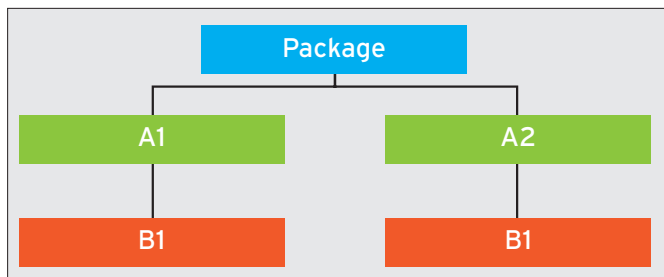


Figure 5: Better than bitmaps: The JavaScript graph dynamically reacts to font size changes while saving the bandwidth this would normally require.

available for some browsers. At least the box text is legible in a browser without JavaScript and CSS support.

Pros and Cons

Many sites benefit from JavaScript and AJAX with respect to usability. Short response times and the ability to do without page reloads are greeted enthusiastically by users. However, the use of AJAX can cause issues that do not occur with static pages. For example, users cannot simply click forward and back buttons to navigate.

Pages modified dynamically by JavaScript thwart user expectations. The problem even affects the simple dynamic menus referred to earlier. If users click to pop up a submenu, the back button will not take them to the previous page status; rather, it opens the page visited previously.

This might not be a big issue with a menu, but if the client-side script changes the page substantially, making it appear to be a new page from the user's viewpoint, confusion is likely.

capture the page status. If the website uses AJAX-based navigation, bookmarks will simply take the user to the front page.

The first thing to consider is whether quick response, or a working history, and the ability to bookmark subpages, are more useful to the surfer. Although the use of a client-side JavaScript that pulls down menus without reloading the page or that re-sorts tables is always going to be an elegant solution, the usability of a shop or catalog page with hundreds of subpages would be seriously affected when users lose the ability to navigate with forward and back buttons.

Remedies

Workarounds have been found for the history and bookmark problems. For example, Google Maps provides an alternative URL containing the GET parameters for the section of the map.

Users can't bookmark the page loaded in the browser. Instead, right-clicking the link shown on the page adds a bookmark.

The browser cannot detect the state changes that occur on an AJAX page because the URL remains the same. The client-side JavaScript logic makes the changes without reloading, which is why bookmarks are unable to cap-

Other workarounds use the HTML anchor that is normally used to store specific positions on a page in a URL. The anchor is the part of the URL that follows the hash sign (#).

Just like the URL itself, the anchor can be modified by means of JavaScript without reloading the page. If the browser fails to find an anchor tag for the string that follows the hash, the display remains unchanged. The anchor is thus perfect for caching status information: The anchor part is the only part of the URL that can be modified without reloading the page.

The capabilities of JavaScript graphics are fairly spartan compared with Flash: HTML and CSS can only draw squares and text. Libraries such as jsGraphics add further shapes, such as circles and polygons. SVG graphics or Canvas, an element of the future HTML 5 web standard, embedded in the web page add more abilities. Both are unsuitable for publicly accessible Internet sites because they are not available across the board for browsers. ■

Listing 3: JSON Data

```
01 {
02   "columns": [ "PID", "USER", "PR", "NI", "VIRT", "RES", "SHR", "S",
03     "CPU", "MEM", "TIME", "COMMAND"],
04   "rows": [
05     [ "6620", "cz", "15", "0", "166912", "57344", "40960",
06       "S", "11.6", "5.6", "0:02.40", "soffice.bin"],
07     [ "3701", "root", "15", "0", "241664", "225280", "17408",
08       "S", "4", "21.8", "4:23.50", "X"],
09     [ "4496", "cz", "15", "0", "63828", "20480", "15360", "R", "2", "2",
10       "0:16.02", "gnome-panel"],
11     [ "4506", "cz", "15", "0", "70480", "18432", "10240",
12       "R", "2", "1.8", "0:04.14", "gnome-terminal"],
13   ]
14 }
```

INFO

- [1] jEdit: <http://www.jedit.org>
- [2] NetBeans: <http://www.netbeans.org>
- [3] Web Developer Firefox plugin: <http://chrispederick.com/work/web-developer>
- [4] Document Object Model: <http://www.w3.org/DOM>
- [5] Unicode arrows: <http://www.alanwood.net/unicode/arrows.html>
- [6] ECMA standard: <http://www.ecma-international.org/publications/standards/Ecma-262.htm>
- [7] MochiKit: <http://www.mochikit.com>
- [8] SVG: <http://www.w3.org/Graphics/SVG>
- [9] HTML Canvas element: <http://www.w3.org/html/wg/html5/#the-canvas>
- [10] VML: <http://www.w3.org/TR/1998/NOTE-VML-19980513>
- [11] Inkscape: <http://www.inkscape.org>
- [12] Karbon: <http://www.koffice.org/karbon>
- [13] SVG2VML: <http://code.google.com/p/svg2vml>
- [14] ExplorerCanvas: <http://excanvas.sourceforge.net>
- [15] TextResizeDetector: <http://www.alistapart.com/articles/fontresizing>