Get it early: http://bit.ly/htmljspp

Dynamic HTML 5 using jQuery for Perl Devs

Pete Krawczyk

June 14, 2012

Intro

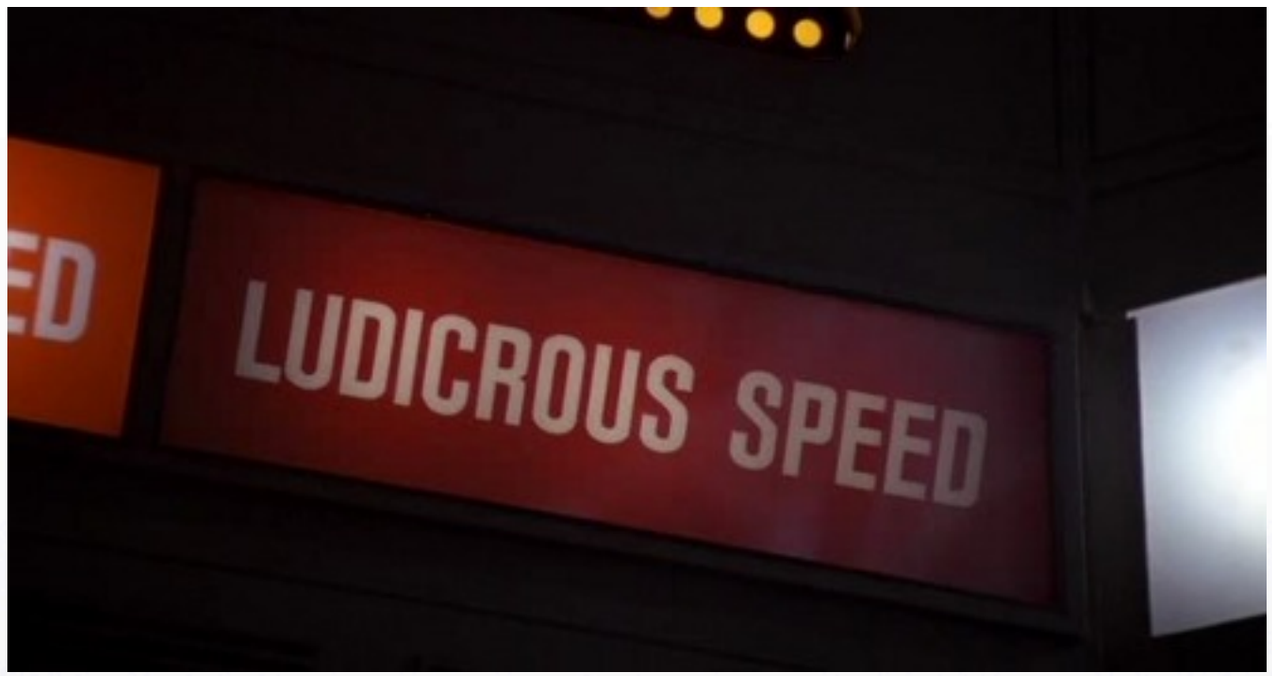
- My name: Pete Krawczyk
- My job: Developer for book wholesaler
- You know: Perl, some HTML and CSS
- You might also know basic JavaScript, but you haven't done Serious Programming with it

Get it early: http://bit.ly/htmljspp

Our Roadmap

- Perl-based To-Do list app
- Changes to HTML for HTML 5
- Introduce JavaScript and jQuery
- Create JSON backend for data transfer

Get it early: http://bit.ly/htmljspp



Go get it!

http://bit.ly/htmljspp

http://petekrawczyk.com/slides/jspp.tgz

(These links are the same)

I also have a thumb drive, if necessary

Notes on the code

- Standard demo app caveats
- Prerequisite check for executing code:
  - `perl prereq/check_prerequisites.pl`
  - You may also need to adjust your httpd.conf

Get it now: http://bit.ly/htmljspp

App Demo

http://localhost/orig/index.cgi

Let's talk HTML

- HTML5 simplifies HTML4 and XHTML
- Valid HTML more important than ever
  - Browsers act differently with invalid HTML
  - Especially don't duplicate: id="xyz"
- Not everything is available to everyone
- http://caniuse.com

Document Type

```
-<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
-    "http://www.w3.org/TR/html4/strict.dtd">

-<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
-    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

+<!DOCTYPE html>
```

Document header

```
-<html>
```

```
-<html xmlns="http://www.w3.org/1999/xhtml">
```

```
+<html lang="en">
```

META content tag

```
-<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">

+<meta charset="iso-8859-1">
```

- This declaration must be in the first 1K of the page
- If your webserver can do this, even better

Including CSS and JS

```
-<style type="text/css">

+<style>
```

```
-<script type="text/javascript">

+<script>
```

Presentational Markup

- HTML moving to separation of structure and presentation
- CSS now used instead of **`<font> <i> border=""...`**

- Tables, etc now styled through CSS as well
- No more frames! (but that doesn't stop you from using them)
- Browsers mostly support mixed versions, can do in steps

Styling content

```
-<b>
+<strong> <!-- style="font-weight: bold;" -->

-<i>
+<em> <!-- style="font-style: italic;" -->
```

```
-<font ...>
+<span style="font: ..."> <!-- CSS -->

-<table border="1" cellpadding="1" ...>
+<style>table { border-collapse: collapse; }
+  th,td { border: 1px solid black; padding: 1px }
+</style>
```

Eliminate self-close tags

```
-<img src="..." />
+<img src="...">

-<input type="..." />
+<input type="...">

-<meta ... />
+<meta ...>
```

CSS for presentation

- Use style sheets - separate from structure
- Be as specific and as general as you can
  - “cascading” is the key word here
- Degrade gracefully
- Don’t get cute
- CSS needs to be validated, too!

App Demo

http://localhost/html5/index.cgi

- Removed presentational HTML attributes
- Added CSS classes and a stylesheet
- Simplified tags like the DOCTYPE

Let's talk JavaScript

- Time to think event-driven programming
- Declared variables; no sigils; no interpolation
- **Everything** is an object!
- Scope is by function, not by block
- Learn by doing: get a good JavaScript console
  - Your browser may include one

References

- https://developer.mozilla.org/en/JavaScript/Reference
- http://msdn.microsoft.com/en-us/library/d1et7k7c(v=vs.94).aspx
- http://interglacial.com/hoj/hoj.html

Events

- Most JavaScript will be triggered by events
  - onClick, onFocus, onChange, onSubmit...
  - Return values are important
- This requires you to think asynchronously
  - You can't assume state or exclusivity
- Avoiding spaghetti is difficult but important

4 ways to include JS

- `<script src="javascript.js"></script>`
- `<script>alert('foo');</script>`
- `<button onClick="alert('foo');">`
- `<a href="javascript:alert('foo')">`

Inline or Include?

- Module vs. Script
- Page size vs. Browser Cache
  - May be helpful to version your filename
- My rules of thumb
  - Functions go in an include file, or in the head
  - Actions go where they're needed
  - Shift-reload as a habit

The document object

- Your page is represented with a DOM (document object model)
- Think of document as a dispatcher
- Object manipulation, not text
- Avoid the temptation of direct reference

Browser objects

- Your browser has objects, too
- window
- screen
- history

Return, Control, Fail

- Your function can only return one thing
- return true / false as control
- JavaScript's default failure mode: stop
- Knowing how to debug is a must

Declaring things

```
our $document = Document->new;
sub adjust_priority {
    my ($e) = @_;

    my $new_value = $e->{value};
    my $old_value = 1;
    my @all_selects = $document->getElementsByTagName('select');
    my $num_selects = scalar @all_selects;

function adjust_priority(e) {
    var new_value = e.value;
    var old_value = 1;
    var all_selects = document.getElementsByTagName('select');
    var num_selects = all_selects.length;
```

Control Loops

```
while (1) {
    for (my $i = 0; $i < $num_selects; $i++) {
        if ($all_selects[$i]->{value} == $old_value) {
            $old_value++;
            next;
        }
    }
    last;
}
```

```
while (1) {
    for (var i = 0; i < num_selects; i++) {
        if (all_selects[i].value == old_value) {
            old_value++;
            continue;
        }
    }
    break;
}
```

Loops/Comparisons

```
foreach my $select (@{$all_selects}) {
    if ($select->{id} != $e->{id} ) {
        if ($select->{value} >= $from && $select->{value} <= $to) {
            if ($dir eq 'i') { # increasing values
                $select->{value}++;
            }
            else {
...
```

```
...
for (select_index in all_selects) {
    var select = all_selects[select_index];
    if (select.id != e.id) {
        if (select.value >= from && select.value <= to) {
            if (dir == 'i') { // increasing values
                select.value++;
            }
            else {
...
```

Concept	Perl	JavaScript
Function	`sub foo {` `  my ($bar) = @_; }`	`function foo( bar ) {}`
Conditionals	`if / elsif / else` `for(;;) / foreach $a (@b)`	`if / else if / else` `for(;;) / for (idx in b)`
Loop Controls	`next` `last`	`continue` `break`
Arrays	`my @a = ($b, $c);`	`var a = new Array (b, c);` `var a = [ b, c ];`
Hashes*	`my %a = ('b' => $c);` `print $a{‘b’};`	`var a = { 'b': c };` `print(a['b']);`
Equality*	`$a == $b (integer)` `$a eq $b (string)`	`a == b (weak typing)` `a === b (strong typing)`
Increment	`$a++` `$a += 2`	`a++` `a += 2`
Comments	`# foo!` `=pod / =end`	`// foo!` `/* foo! */`

Concept	Perl	JavaScript
Concatenation*	`$a . $b`	`a + b` `a.toString() + b.toString()`
Print	`print()`	`document.write()` `// NOT print()`
Warn	`warn()`	`alert()` `console.warn() (log, error)`
Regex Match	`if ($a =~ /^b+c*$/)`	`var re = /^b+c*$/;` `if (re.test(a))`
Regex Extract	`($a) = $b =~ /^(c+)$/;`	`var re = /^(c+)$/;` `var a = re.exec(b)[1];`
eval BLOCK	`my $rc = eval { 1; }` `if (!$rc) {...}`	`try { 1; }` `catch(err) {...}`
Escape eval	`die "foo"`	`throw "foo"`
Object being called	`my $self = shift;` `$self->foo();`	`this.foo();`

App Demo

http://localhost/firstjs/index.cgi

- Added a function to adjust priorities
- Added an event: “onchange”

Let's talk jQuery

- Simplify common DHTML operations
- Handles cross-browser compatibility
- Easy to write set operations
- Extensions can be easily added dynamically
- (Really, all the frameworks give you these)
  - Prototype, GWT, YUI, others

Let's talk jQuery

- Everything revolves around an object: $
- Think set operations, like SQL
- Think callbacks and anonymous functions
- http://jquery.com/ is an awesome reference

Are you ready?

- You can't work on a page until it's rendered
- To this end, jQuery provides a helper: $(document).ready()
- Replaces <body onload="...">
- Can be called multiple times

jQuery ♥ CSS

- jQuery uses CSS selectors as targets
  - $("body").append(" Foo! ");
  - $("div.notes").hide();
  - $("table#taskrows th").addClass(...);
- This is why CSS validation is important

Do things on a click

```
<input type="submit" id="add_new" name="add_new" value="+">
<input type="checkbox" id="n1_c" name="n1_c" value="done">
<input type="text" id="n1_s" name="n1_s" value="" size="30">
<textarea id="n1_d" name="n1_d" rows="3" cols="50"></textarea>

$(document).ready( function() {
    var added_tasks = 1;
    $("#add_new").click( function(event) {
        event.preventDefault();
        added_tasks++;
```

Mid-action: prepare row data

```
// This stores the new task values and resets the "add" fields
var new_complete = $("#n1_c").is(':checked');
if (new_complete) {
    $("#n1_c").toggle();
}
 var new_summary = $("#n1_s").val();
$("#n1_s").val('');
var new_description = $("#n1_d").val();
$("#n1_d").val('');

// This adds a new priority to each existing one
var all_selects = document.getElementsByTagName('select');
var num_selects = all_selects.length;
var new_pri = num_selects + 1;
$("select.sel_pri").append('<option value="' + new_pri + '">' +
new_pri + '</option>');
```

Copy task into new row

```
// This duplicates an existing row and then modifies it
var new_task_row = $("tr.task_row :last").closest('tr').clone();
var pre = 'n' + added_tasks;
new_task_row.find('select').attr('id',pre+'_p').attr(
    'name',pre+'_p').val(new_pri);
new_task_row.find('.complete input').attr('id',pre+'_c').attr(
    'name',pre+'_c').prop('checked',new_complete);
new_task_row.find('.summary input').attr('id',pre+'_s').attr(
    'name',pre+'_s').val(new_summary);
new_task_row.find('.description textarea').attr('id',pre+'_d').attr(
    'name',pre+'_d').val(new_description);
new_task_row.change(function(){
    return adjust_priority(this);
});
$("tbody").append(new_task_row);
```

App Demo

http://localhost/addjq/index.cgi

- Adjusted Perl to support multiple new rows
- Added a class for jQuery selector
- Added a click handler for our “new task”

Let's talk data exchange

- JavaScript allows external data fetches
- This is what's known as "AJAX" - Asynchronous Javascript and XML
- Doesn't have to be XML
  - JSON: "JavaScript Object Notation"
  - If you've seen JSON, you've seen YAML (mostly)

AJAX calls

- In the old days: XMLHttpRequest
  - Catch your own errors, maintain state
  - Tracking asynchronous calls a pain
- These days: $.ajax() / $.getJSON()
  - Let your framework handle the details
  - Give a callback function when work completes

AJAX caveats

- Be prepared to offer a fallback
  - Your data service might be unavailable
  - Your request might not complete
- Be compatible with older browsers
  - Not just IE7: phones have old ones too
  - JSON: one additional include...

Make an AJAX service

```
use JSON;

$out = {
    status => 'OK',
    taskid => 1,
};

my $out_json = encode_json($out);
print <<"JSON";
Content-type: application/json

$out_json
JSON

exit(0);
```

Make an AJAX call

```
$.getJSON('new_task.cgi', {
    summary: new_summary,
    description: new_description,
    },
    function(json){
        if (json.status && json.status == 'OK') {
            add new task row(
                json.taskid,new_complete,new_summary,new_description
            );
        }
        else {
            alert("Error! Your new task may not have been saved.");
        }
    }
).error(function() {
    alert("Error! Asynchronous request could not be sent");
});
```

App Demo

http://localhost/ajax/index.cgi

- Added more Perl to handle incoming AJAX
- Added the extra JSON include for fallback
- Function to add task rows to our task table
- AJAX request/callback function for tasks
- Add a "template row" to copy

App Demo

http://localhost/ajax-nojs/index.cgi

- Removed script tags

Now what?

- Offer instant change sync support
- Add more data points?
- Support push of new tasks?
- See completed tasks?
- The rest is up to you...

Thank you!