JavaScript and Ajax

INFO 2310:
Topics in Web Design and Programming

Web 2.0

Most modern websites make extensive use of JavaScript and Ajax. How can we use them in Rails?

Baked in

Just as with testing, much of this is already wired right into Rails. In fact, two JS libraries are provided as part of Rails: Prototype and Scriptaculous.

To include these libraries, in the <head> tag of the corresponding view add

```
<%= javascript_include_tag :defaults %>
```

views/ application.html.erb

Includes

This will include 5 JS files:

- prototype.js
- effects.js
- dragdrop.js
- controls.js
- application.js

Can also specify just the files you want.

Rails helpers

Rather than needing to know the JS yourself, Rails has a number of built-in helper functions to write the JS with.

Example

As a simple start, add the following to the bottom of app/views/posts/index.html.erb.

```
<%= link_to_function "Annoying
  pop-up", "alert('Hi there!')"
  %>
```

HTML/JS

The underlying HTML/JS created is then:

```
<a href="#" onclick="alert('Hi
  there!'); return
  false;">Annoying pop-up</a>
```

There are some issues with this that we'll return to later.

More useful functions

We can also link to more useful functions from the Scriptaculous library.

For /views/posts/show.html.erb, modify the line

```
<%= render :partial => "/comments/comment",
  :collection => @post.comments %>
```

to

```
<div id="comments">
<%= render :partial => "/comments/comment",
  :collection => @post.comments %>
</div>
<%= link_to_function "Toggle comments" do |page|
  page.visual_effect(:toggle_blind, "comments")
  end %>
<br />
```

Ajax

link_to_remote

We can set up an Ajax call to take place on the clicking of a link via 'link_to_remote'.

Let's add an Ajax call to delete a comment via link_to_remote in app/views/comments/_comment.html.erb:

```
<%= link_to_remote 'Delete',
:url => post_comment_path (@post, comment),
:method => :delete,
:confirm => 'Are you sure?'  %>
```

You can then try deleting a comment.

Sure enough, when you refresh the page, it is gone, but that isn't what Web 2.0 is all about…

Routing Ajax

To get things to work the way we think they should, we need to route the Ajax call in the comments controller.

We need to tell the 'destroy' method what to do in case of a JS request.

In app/controllers/comments_controller.rb, we change the last line of the destroy method from

```
redirect_to post_url(@post)
```

to

```
respond_to do |format|
  format.html
  {redirect_to(post_comments_url(@post)) }
  format.js
end
```

destroy

```
def destroy
  @comment = @post.comments.find(params[:id])
  @post.comments.delete(@comment)
  respond_to do |format|
     format.html {
  redirect_to(post_comments_url(@post)) }
     format.js
  end
end
```

.js.rjs

Just like format.html by itself in respond_to would tell Rails to go render destroy.html.erb, format.js tells Rails to go render destroy.js.rjs.

RJS = 'Ruby JavaScript'.

destroy.js.rjs

In views/comments, we make a file destroy.js.rjs with the single line:

```
page["comment-
  #{@comment.id}"].remove
```

This looks for an element in the page with id comment-#, and removes it.

Adding the id

In order for this to work, we need to add the ids to the comments. Change the first line of the partial view/comments/_comment.html.erb from

```
<div class="comment">
```

to

```
<div class="comment" id="comment-
  <%= comment.id %>" >
```

Try it…

Restart the server and try it…

Take 2

It's a little unsatisfying to have it just disappear, no?

Let's add a little effect to make it fade away before disappearing.

Fade away…

Edit destroy.rjs.rjs to now read:

```
page.visual_effect :fade, "comment-
  #{@comment.id}", :duration => 1.0
page.delay 1 do
  page["comment-
  #{@comment.id}"].remove
end
```

Try it!

Other things

We can also do things like

```
page[id].hide
page[id].show
page[id].toggle
page[id].insert_html :before,
  html
page[id].replace_html html
```

Other things

We don't have to return RJS in response to an Ajax call; we can, for instance, return some HTML instead.

Let's do this for our login form…

Changing the link

In views/layouts/application.html.erb, change the login link from

```
<%= link_to('login', login_path) %>
```

to

```
<%= link_to_remote 'login', :url => login_path,
:update => 'login_id' %>
```

…

```
<%= link_to_unless_current('all posts',
posts_path) %>
<span id="login_id"></span>
```

The result of the Ajax call will be used to update the element of id 'login_id'. The Ajax call is made to the login_path, which in our routes file is defined to be the new method of sessions_controller.

Changing the controller

Now we need to tell the sessions controller what to do when it gets a Ajax call.

To the end of the new method in app/controllers/sessions_controller.rb, add

```
respond_to do |format|
  format.html
  format.js { render :action => 'new-tiny' }
end
```

This tells the controller to render the view in views/sessions/new-tiny.html.erb, which gets returned to the Ajax call.

new-tiny.html.erb

```
<% form_tag 'sessions' do -%>
    <label for="email">Email:</label>
    <%= text_field_tag :email %>
  <label
  for="password">Password:</label>
  <%= password_field_tag :password %>
  <%= submit_tag 'Login' %>
<% end -%>
```

Other Ajax calls

In addition to the `link_to_remote` helper, there are also:

- `remote_form_for, remote_form_tag`: Ajaxified forms; Ajax call made 'onsubmit'
- `observe_field, observe_form`: attaches Ajax calls to 'onchange' events of a particular form field, or all form fields
- `periodically_call_remote`: Makes Ajax call at specified intervals

Plugins and Ajax

There are plugins that give us more helper functions for using Ajax.

Let's try one that allows for 'in place' editing.

To get this one, either use

```
ruby script/plugin install
git://github.com/rails/in_place_editing
.git
```

or download the files from http://github.com/rails/in_place_editing/tree/master

and install in vendor/plugins/in_place_editing.

We'll allow for editing the titles of posts (we could do bodies too, but I'm not sure how that would interact with our markdown interpreter).

In the app/controllers/posts_controllers, we list the model, fields that we want to attach the in-place editors.

```
class PostsController <
  ApplicationController
  in_place_edit_for :post, :title
```

model field

Now we add the editor to our view. Remember that we only want to allow editing by authors.

In views/posts/show.html.erb, replace

```
<%=h @post.title %>
```

with

```
<% if
@post.editable_by?(current_user) %>
 <%= in_place_editor_field :post,
"title" %>
<% else %>
 <%=h @post.title %>
<% end %>
```

Try it!

Restart the server and give it a try…

Problems with Rails' JS

There are some issues with the JavaScript created by Rails' helpers functions.

Any guesses?

We want JavaScript that is:

- Unobtrusive (lives in a separate 'behavior' layer, just like CSS is a separate 'style' layer)
- Degrades gracefully (gives reasonable equivalents if JS turned off)

Obtrusive JS

Rails is currently pretty hopeless at the moment for unobtrusive JS if we use the standard helpers.

There are ways to write the JS directly in a separate JS file, but we won't go there.

Graceful degradation

We can help with graceful degradation a bit, though, by providing an href in the case that JS is disabled.

For instance, link_to_remote takes a third argument that gives the href:

```
link_to_remote 'login', { :url =>
  login_path, :update => 'login_id'
  }, :href => login_path
```

Other libraries

jQuery

It's possible to use other JS libraries such as jQuery with Rails – they just don't come bundled in.

For instance, the plugin jRails (http://ennerchi.com/projects/jrails) replaces all the underlying Prototype/Scriptaculous calls in link_to_remote, page.hide, etc. with jQuery calls, and allows for other jQuery effects.

Reminders…

Next week: A photo gallery in one lecture

Be sure to bring some digital images for your gallery. The galleries will have multiple albums, so bring images for more than one album.