RAD Lab
UC Berkeley

Ruby on Rails Short Course Part 5: AJAX & Testing

Armando Fox
UC Berkeley RAD Lab

Outline of the day

1. Web apps, MVC, SQL, Hello World
2. Just enough Ruby
3. Basic Rails

Lunch break

4. Advanced model relations
5. AJAX & intro to testing
6. Configure & deploy

Informal discussion: RoR and pedagogy

Outline of Session 5

- AJAX 101
  - XHTML DOM, JavaScript, prototype, script.aculo.us
  - Javascript integration with Rails
- Testing Basics
  - test infrastructure built right in
  - unit, functional, integration tests; fixtures
- Potpourri of miscellaneous cool stuff

Web 1.0 → Web 2.0

- Web 1.0 (“old world”) GUI: click → page reload
- Web 2.0: click → page updates in place
  - also timer-based interactions, drag-and-drop, animations, etc.

How is this done?

1. Document Object Model (c.1998, W3C) represents document as a hierarchy of elements
2. JavaScript (c.1995; now ECMAScript) makes DOM available programmatically
3. XMLHttpRequest (MSIE 5, c.2000; others, c.2002) allows async (callback semantics) HTTP transactions decoupled from page reload

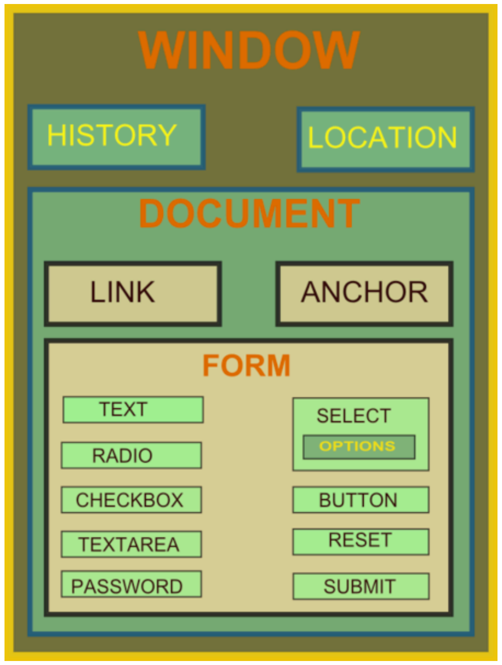
- Practical implication: server workloads denser & relatively more write-intensive

JavaScript

- A browser-side scripting language that
  - is dynamic
  - is weakly-typed (implicit conversion)
  - is prototype-based (vs. class-based)
  - has first-class functions, closures, H.O. functions
  - is embedded in most browsers since c.1998
  - keeps many security researchers' jobs safe
- Browser exposes some of its behaviors & attributes to JavaScript environment
  - eg, *window, document* objects
  - eg, *XmlHttpRequest* browser method

The DOM & JavaScript

- A platform-independent (?) hierarchical object model representing HTML or XML doc
  - part of a *separate* standards effort; in practice, implementations vary
- Exposed to JavaScript interpreter
  - Inspect DOM element value/attribs
  - Change value/attribs → redisplay



```
<input type="text" name="phone_number" id="phone_number"/>
<script type="text/javascript">
 var phone = document.getElementById('phone_number');
 phone.value='555-1212';
 phone.disabled=true;
 document.images[0].src="http://.../some_other_image.jpg";
</script>
```

JavaScript Libraries

- *prototype* provides functions and shortcuts for working with DOM & XmlHttpRequest

```
$("submit_btn").disabled = true;
var AjaxOpts = {
  method: "get",
  parameters: "id=3&user=" + $("usrname").value,
  onComplete: displayResponse };
var AjaxReq = new Ajax.Request (url,AjaxOpts);
function displayResponse() {...}
```

- *Handlers* allow associating JavaScript functions with events on DOM elements
  - e.g., `onClick, onMouseOver, onFocus`...

So: What’s AJAX?

- **A**synchronous **J**avaScript **A**nd **X**ML
  - Early showcase app: Google Maps
- Recipe (to a zeroth order):
  - attach JavaScript function callbacks to various events on browser objects
  - in callback, inspect/modify DOM elements and optionally do an asynchronous HTTP req. to server
  - on server response, pass result to yet another JavaScript function that will monkey with DOM again
- Rails integrates seamless Ajax support
  - *Prototype* to deal with cross-browser issues, common Ajax functionality, etc.
  - Script.aculo.us, a JavaScript library of visual effects

A Rails View of AJAX

- What events should be listened for?
  - Individual DOM element value changes?
  - Anything on a form changes?
  - Timeout?
- How should event be handled?
  - What controller & method should be called?
  - What DOM element value(s) should be marshalled & passed to it?
- What to do with the result?
  - Update DOM element *in place* with returned content?
  - Callbacks? (waiting, receiving, complete, error...)

Listening For Events

- Not surprisingly, Rails lets you listen at the *element* or *form* level

```
observe_field('student[last_name]',
  :url => {:controller=>'students',
             :action=>'lookup_by_lastname'},
  :update=>'lastname_completions')
```

- when `student[last_name]` field changes, call method `lookup_by_lastname` in *StudentController* with new field value
- returned text from controller method will *replace* the “inner contents” of element ID *lastname_completions*
  – typically using `render :partial` or `render :text`

Listening on a Whole Form

```
observe_form('student_form',
   :url => {:controller => 'students',
            :action => 'process_form'},
   :update => 'student_info_panel')
```

- When any element of *student_form* changes, call *process_form* method in *StudentsController,* marshalling all elements into `params[]`

Specifying Event Handlers

- Event handlers are just controller methods!
  - Rails wrappers around *prototype* library functions marshal arguments & do XHR call
- Controller method can use *render :partial* to produce a result
  - Typical example: table with collapsible entries
  - or *render :text* to send raw content back
- Method can tell how it was called by calling @*request.xhr?*

What to Do With Results

- Typically, results *replace* content of an HTML element
  - Remember you can “elementize” (almost) any arbitrary chunk using `<span>` or `<div>`
- Additional keyword-like arguments to `observe_field` and `observe_form` allow separate handling of other callback events
  - states: server contacted, waiting, receiving, done
  - different error codes for failures

Graceful Fallback to Web 1.0

- What if AJAX support not available in user's browser?
- Specifying a fallback in the AJAX tags
  - `:html` => *options*
  - how does view know whether to use it or not?
- How does the controller know what to do?
  - request. xhr?

Dressing it up with effects

- Script.aculo.us also wrapped in Ruby as part of standard Rails distro
- Effect.new(...)

Cool GUI Tasks as AJAX

- “Auto-completion” of a text field?
- “Update now” button?
- Periodically polling for updates?
- Cross-field validation in a form?
- Repopulate popup menus constrained to choices in other menus?

Remote Javascript templates

- What if the thing you want to return is not actually content, but JS code?
- Place it in an *.rjs (remote JS)* template!

```
page['student_menu'].value =
  page['other_menu'].value
```

  - "Rendering" *rjs* template wraps your code in `try {... } catch {`*...show alert...*`}`, among other things

The dark side of AJAX, RJS, etc.

- *Lots* of layers of code; can be hard to debug
- Browsers tend to fail silently when they choke on JS-related errors
  - Can open JS console log, but who does that?
- *On the plus side...*
  - eminently more maintainable
  - probably more robust and browser-neutral

Outline of Session 5

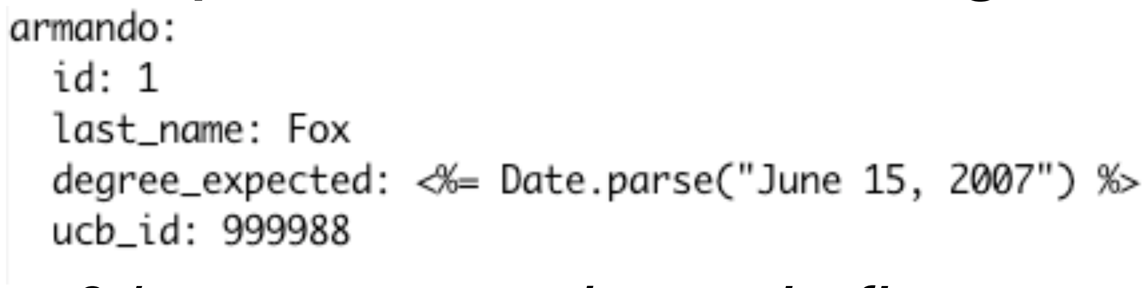
- AJAX 101
  - XHTML DOM, JavaScript, prototype, script.aculo.us
  - Javascript integration with Rails
- Testing Basics
  - test infrastructure built right in
  - unit, functional, integration tests; fixtures
- Potpourri of miscellaneous cool stuff

Test Infrastructure Built Right In

- Separate database for testing
  - Testing tasks automatically create its schema at beginning of test run
  - Automatically cleaned out and populated with *fixtures* before each individual test suite is run
- Test “scaffolds” created as by-product of creating app
  - when generate scaffold
  - when generate migration
  - etc.

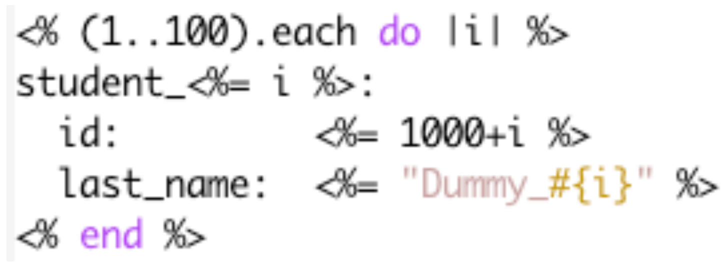
Test Fixtures

- Data preloaded into testing database



```
armando:
  id: 1
  last_name: Fox
  degree_expected: <%= Date.parse("June 15, 2007") %>
  ucb_id: 999988
```

  – &/or generate *dynamic fixtures* at test-run time



```
<% (1..100).each do |i| %>
student_<%= i %>:
  id:         <%= 1000+i %>
  last_name:  <%= "Dummy_#{i}" %>
<% end %>
```

A Simple Unit Test

- Note use of assertions throughout
- Only method names starting with test_ are run
- Run rake test:clone_structure to clone schema of development DB to test DB
- Run unit test(s) with rake test:units
  - rake test wraps all these tasks together
- Large library of assertions for checking tests

A Simple Functional Testcase

- Note examination of the flash to check that correct result was displayed to user

```
def test_000_failed_login
  post :login, :customer => {:login => customers(:tom).login,
    :password => 'BAD'}
  assert_nil session[:cid]
  assert_match /mistyped your password/i, flash[:notice]
  post :login, :customer => {:login => 'NOBODY', :password => 'BAD'}
  assert_match /can\'t find that email address/i, flash[:notice]
  assert_nil session[:cid]
  post :login, :customer => {}
  assert_match /please provide both/i, flash[:notice]
end
```

- Testing actions that fail & redirect

```
def test_003_non_admin_cant_view_cust_record
  simulate_login(customers(:tom))
  get :list
  assert_redirected_to :action => 'login'
end
```

Scanning the Output

- A more complicated example...
  - scan output for tags
  - submit XmlHttpRequests to trigger Ajax actions
  - use a helper function to “simulate” login (which is tested separately in another functional test)

Integration Testing

- Goal: navigate the site from a user's point of view
  - create a *session object* per dummy user
  - use same kinds of assertions but in the context of each user's session
  - can dynamically create many sessions (as with fixtures) to do directed random-interleaved testing

Outline of Session 5

- AJAX 101
  - XHTML DOM, JavaScript, prototype, script.aculo.us
  - Javascript integration with Rails
- Testing Basics
  - test infrastructure built right in
  - unit, functional, integration tests; fixtures
- Potpourri of miscellaneous cool stuff

Code Stats & Microbenchmarks

- *rake stats:* how much code did I write? ratio of lines of test code to lines of app code?
- *script/profiler:* method-level profiling tools
- *script/benchmarker:* sanity-check *μ*bench individual method calls
- (coming soon) `-rcoverage` option to Ruby when running tests
  - reports % coverage and which lines of code not covered by tests

Plug-Ins

- A separable extension to Rails framework
  - just copy a directory!
  - relies on Ruby classes being open, and on various mechanics of the mixin (Module) mechanism
- A plug-in...
  - defines additional classes and modules
  - provides one or more methods that result in the calling class “pulling in” plug-in
  - result: calling class(es) extended with plug-in methods

Example Plugins I Love

- Example 1: SslRequired

```
include SslRequirement
ssl_required :checkout, :place_order
ssl_allowed :index, :list_products
```

  - Inserts before-filters that check protocol of controller request, perform redirect if bad

- Example 2: ExceptionNotifiable

```
# in application.rb (toplevel controller)
include ExceptionNotification
# in environment.rb or environments/production.rb
config.after_initialize do
  ExceptionNotifier.exception_recipients =
  'fox@cs.berkeley.edu'
end
```

Other Cool Stuff (so you know what you don't know)

- View caching
- In-memory distributed session storage
- Slipping in another database
- Action Mailer
- `script/runner` for (e.g.) `cron(8)` actions
- REST & RXML
- API's to the rest of the world
  - Google Maps, Amazon, Facebook...
- ISP's that provide a Rails "virtual machine"

Yow! Questions?