Practical Functional JavaScript
Oliver Steele
Ajax Experience
Wednesday, 1 October 2008

Teasers

- AJAX is all about waiting for someone*, and remembering what you were going to do when they got back to you.
- Functions : interactions :: objects : domains
- You didn’t really want threads anyway. (Most of the time.)

* user, web server, other server, wall clock, plugin

About Me

<table>
<thead>
<tr>
<th>where?</th>
<th>what?</th>
<th>implementing languages</th>
<th>using languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oblong Industries</td>
<td>Ruby bindings for “Minority Report”-style interfaces</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Entrepreneurial &amp; Consulting</td>
<td>BrownGoods, StackBlur, Panoram Wall Calendar</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Lusak Systems</td>
<td>OpenLaszlo</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Apple Advanced Technology Group</td>
<td>Dylan (programming language)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Apple System Software</td>
<td>Ska (graphics library)</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

About You

Raise your hand if you know*:

- Closures
- Ruby / Smalltalk
- XHR / AJAX
- An AJAX framework (Prototype / jQuery / …)
- Threads

* none of these are required; this just helps me calibrate the talk

Agenda

- Some Context
  - Case Studies
  - MVC on the Client
- Fundamentals
  - Perspectives on Function
  - Closures (review)
  - Making Functions
  - Decorating Functions
  - Some Idioms
- Callbacks
  - Threaded Callbacks
  - Registered Callbacks
  - Order & Serializing
  - Retries, Guards, Timeouts
- Q&A

Non-Agenda

- Comet, Bayeux, Gears
- Frameworks*
- Theory (this isn’t your Monad or CPS fix)
- Security (standard practices apply)

* This talk should help you understand their implementation and use, but doesn’t explore their APIs in any depth
Case Studies

BrowseGoods

- Visually browse an Amazon department, graphically arranged
- Server: Ruby (layout, image processing, app server)
- Client: Prototype + Scriptaculous

BrowseGoods Capabilities

- Background prefetch of item maps
- Background fetch of saved items
- Operations on saved items are queued until initialization is complete

Style & Share
Style & Share Capabilities

- Retry with exponential backoff and failover
- Explicit queues to control serialization order
- Background prefetch for catalog items
- Multiple queues to prioritize user interaction

Fansnap

- Visualize available seats in a venue
- Seatmap is OpenLaszlo (compiled to Flash)
- Widgets are in jQuery

FanSnap Capabilities (Seatmap)

- Two-way asynchronous communication between the Flash plugin and the HTML
- Asynchronous communication between the Flash plugin and the server
- Again, initialization is particularly challenging

Webtop Calendar

Webtop Calendar Data

Data Layer

Calendar Client

Calendar Client Data Architecture

Range Cache

Calendar Model

Calendar Service

Calendar Collection

Event

Observer

View Layer

Event Cache

Serializer

Webtop Client Library

CalendarConnection

Event Report Cache

Webtop Server

Internet

Event Model
Webtop Calendar Capabilities (Data Model)

- Synchronizes local model with server model
- Local model is cache: some operations update it; others invalidate it
- Race conditions, where prefetch overlaps operations that invalidate the cache

The Problem: Web MVC

Waiting on the Server

Waiting on the User

Waiting on the Client

Function Fundamentals
What is a Function?

- Math: rule that maps inputs to outputs
- Computer science: abstracted computation with effects and outputs
- Software engineering: one of several units for documentation, testing, assertions, and analysis
- Source code: unit of source text with inputs and outputs
- Runtime: invocable parameterized behavior

Callbacks

```javascript
function callback(x) {
  log('received ' + x + '\n');
}
function request() {
  $.get('/request', callback);
} request();
```

Run

Making Functions

```javascript
function makeConst1() {
  return function() { return 1; }
}
function const1a() { return 1; }
var const1b = function() { return 1; }
var const1c = makeConst1();
log(const1a());
log(const1b());
log(const1c());
```

Closures

```javascript
var get, set;
function setAccessors() {
  var x = function() { return x; }
  var y = function() { x = y; }
  get = function() { return x; }
  set = function(y) { x = y; }
}
setAccessors();
log(get());
set(10);
log(get());
```

Closures

```javascript
function makeAccessors() {
  var gf;
  return {get: function() { return gf; },
           set: function(y) { gf = y; }}
}
var gf1 = makeAccessors();
var gf2 = makeAccessors();
gf1.set(10);
gf2.set(20);
log(gf1.get());
log(gf2.get());
```

Idioms

```javascript
// 'this' and 'arguments' are special
function f() {
  logArguments(this, arguments);
  f();
  f('a', 'b');
}
```

Results

- `f()`: 10
- `f('a', 'b')`: 10

Results

- `f()`: 10
- `f('a', 'b')`: 10
Summary

- Functions are values
- Functions can be arguments, return values, array elements, property values
- Functions can be created and "modified"
- Argument lists can be saved, modified, and replayed

Case Study: Callbacks

Callback Scenarios

- Chained Callbacks
- Queues and Priority
- Throttling
- Caching
- Timeouts
- Retry and Failover

- Conjunctive-Trigger Callbacks
- Conditional Callbacks
- Outdated Responses

Throttling

```javascript
for (var i = 0; i < 10; i++)
$.get('/services/time', log);
```

Results

- 21:35:36 success
- 21:35:36 success
- 21:35:36 success
- 21:35:36 success
- 21:35:36 success
- 21:35:36 success
- 21:35:36 success
- 21:35:36 success
- 21:35:36 success
- 21:35:36 success

Retry and Failover

```javascript
$.getWithRetry = function(url, k) {
  var countdown = 10;
  $.ajax({url: url, success: k, error: retry});
  function retry() {
    if (countdown > 0) {
      log('retry');
      $.ajax({url: url, success: k, error: retry});
    }
  }
};
$.getWithRetry('/services/error', log);
```

Caching

```javascript
var gRequestCache = {};
$.cachedGet = function(url, k) {
  if (url in gRequestCache)
    k(gRequestCache[url], 'success')
  else
    $.get(url, advizeBefore[k, function(x, status) {
      if (status == 'success')
        gRequestCache[url] = x;
    }]);
$.cachedGet('/services/random', log);
$.cachedGet('/services/random', log);
$.cachedGet('/services/echo/1', log);
$.cachedGet('/services/echo/2', log);
setTimeout(function() {
  $.cachedGet('/services/random', l
```
Summary

- Functions-as-objects allow separation of concerns
- Factor how, when, and whether from what
- Functions are to interaction patterns as objects are to domains

Q&A

Thanks!

– Oliver Steele

http://osteele.com