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

where?	what?	*implementing* graphics	*implementing* languages	*using* graphics	*using* languages
Oblong Industries	Ruby bindings for "Minority Report"-style interfaces			✔	✔
Entrepreneurial & Consulting	BrowseGoods Style&Share Fansnap Webtop Calendar			✔	✔
Laszlo Systems	OpenLaszlo		✔	✔	
Apple Advanced Technology Group	Dylan (programming language)		✔	✔	
Apple System Software	Skia (graphics library)	✔		✔	

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

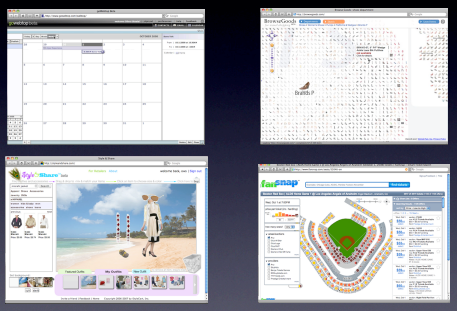
- Context
  - Example Applications
  - MVC on the Client
- Fundamentals
  - Closures (review)
  - *Making* Functions
  - *Decorating* Functions
  - Some Idioms
- Callbacks
  - Queueing & Throttling
  - Caching & Joining
  - Retries & Failover
- Conclusions
- Q&A

Non-Agenda

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

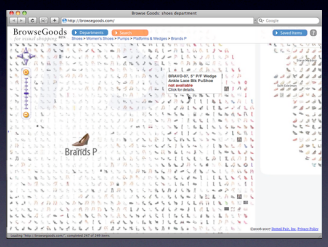
* This talk should help you understand their *implementation* and *use*, but doesn't explore their *APIs* in any depth

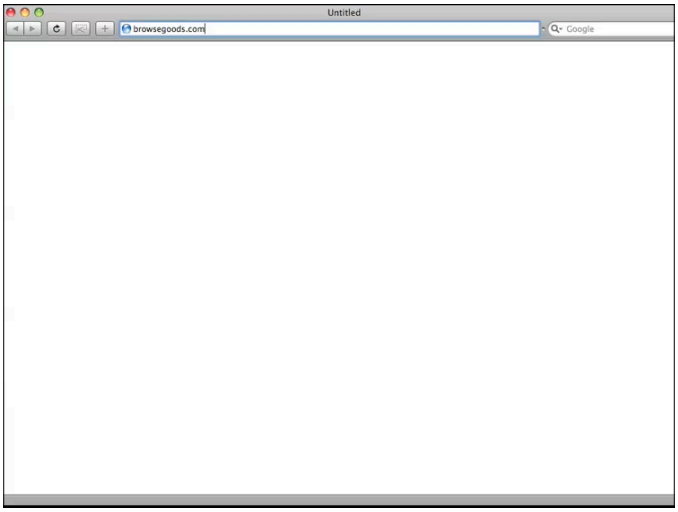
Example Applications



BrowseGoods

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

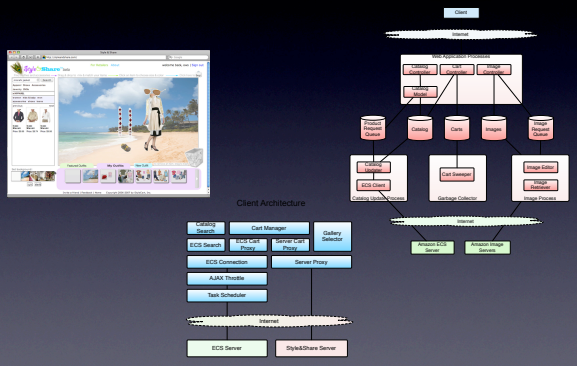


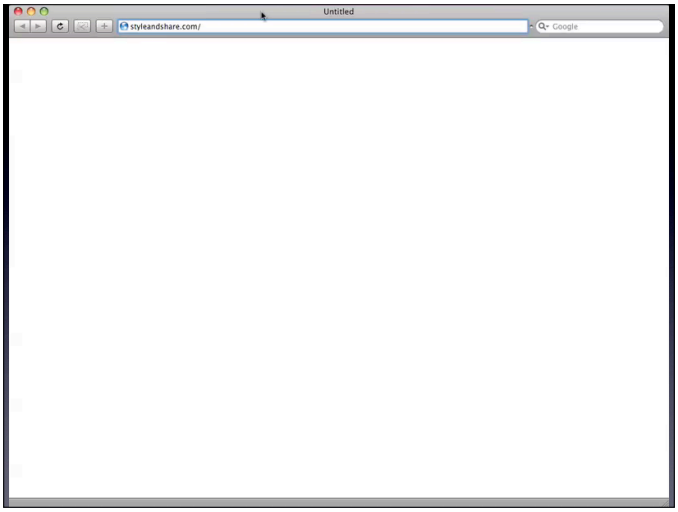


BrowseGoods Capabilities

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

Style & Share



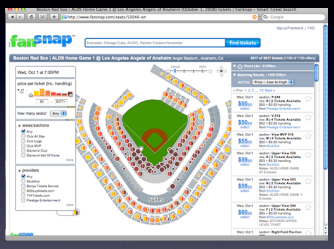


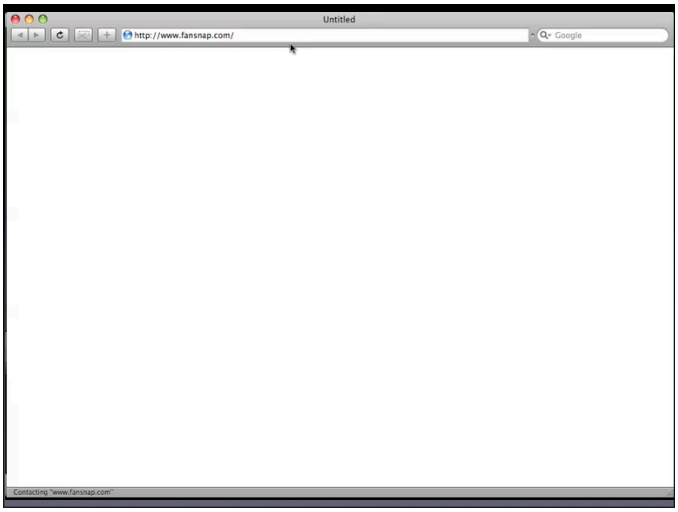
Style & Share Capabilities

- 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

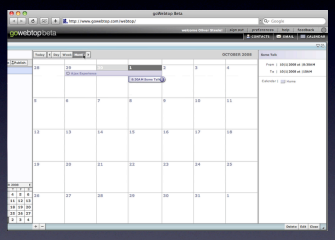


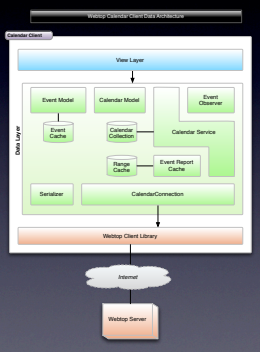


FanSnap Capabilities (Seatmap)

- 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

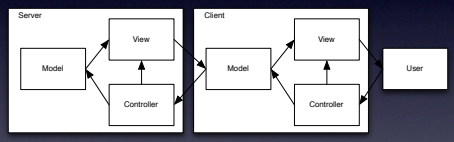




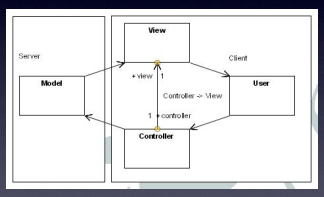
Webtop Calendar Capabilities (Data Model)

- 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

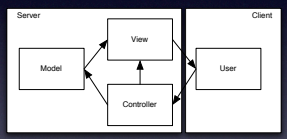
Motivating Problem: Web MVC



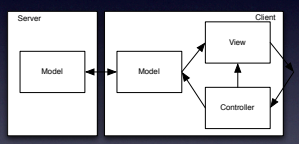
MVC Basics



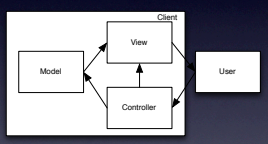
Server: Waiting on the Client



Client: Waiting on the Server



Client: Waiting on the User

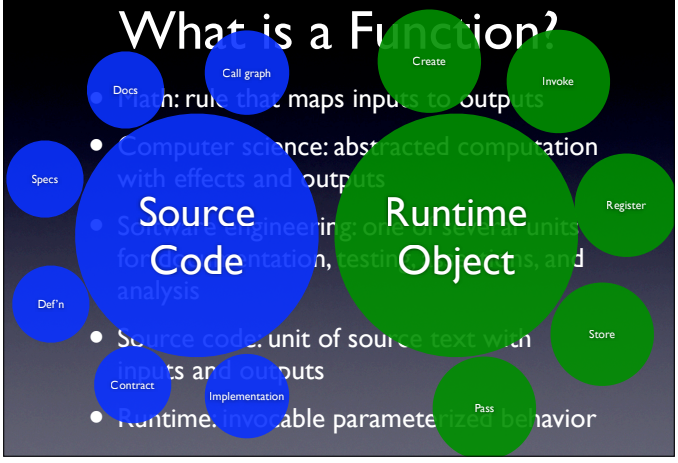


Lots of Waiting

- Client (server session)
- Server (XHR)
- User (UI event)
- Future (setTimeout)
- Concurrency
- State

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 analogous units for implementation, testing, documentation, and analysis
- Source code: unit of source text with inputs and outputs
- Runtime: invocable parameterized behavior

Callbacks

```
function callback(x) {
    log('received "' + x + '"');
}

function request() {
    $.get('/request', callback);
}

request();
```

Run

Making Functions

```
function makeConst1() {
    return function() { return 1; }
}

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

log(const1a());
log(const1b());
log(const1c());
```

Results

32:37s → 1
32:37s → 1
32:37s → 1
32:37s → 1

Closures

```
var get, set;
function setAccessors() {
    var x = 1;
    get = function() { return x; }
    set = function(y) { x = y; }
}

setAccessors();
log(get());
set(10);
log(get());
```

Results

32:46s → 1
32:46s → 10

Closures

```
function makeAccessors() {
    var x;
    return {get: function() { retu
            set: function(y) { x =
}

var gf1 = makeAccessors();
var gf2 = makeAccessors();
gf1.set(10);
gf2.set(20);
log(gf1.get());
log(gf2.get());
```

Results

32:37s → 10
32:37s → 20

Idioms

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

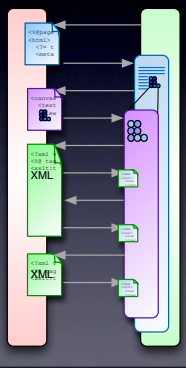
Results

32:37s → this = [object DOMWindow]
32:37s → arguments = []
32:37s → this = [object DOMWindow]
32:37s → arguments = [a]
32:37s → this = [object DOMWindow]
32:37s → arguments = [a, b]

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 Scenarii

- Chained Callbacks
- Queues and Priority
- Throttling
- Caching
- Timeouts
- Retry and Failover
- Conjunctive-Trigger Callbacks
- Conditional Callbacks
- Outdated Responses

Throttling

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

Results

33:06s → 33:06s success
33:06s → 33:06s success
33:06s → 33:06s success
33:06s → 33:06s success
33:06s → 33:06s success
33:06s → 33:06s success
33:06s → 33:06s success
33:06s → 33:06s success
33:06s → 33:06s success
33:06s → 33:06s success

Retry and Failover

```
$.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

```
var gRequestCache = {};
$.cachedGet = function(url, k) {
    if (url in gRequestCache)
        k(gRequestCache[url], 'success');
    else
        $.get(url, adviseBefore(k, function(result, stat
            if (status == 'success')
                gRequestCache[url] = result;
        }));
};
```

```
$.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*

What is FP?

- Functions are pure
- Functions are values

Q&A

Thanks!

– Oliver Steele

http://osteele.com