Developing & Maintaining Large Javascript Applications

Charlie Fiskeaux II
User Interface Engineer

OmniTI CIRCONUS

About Me & Circonus

- Lead User Interface Engineer for Circonus
- Industry-leading monitoring and analytics platform
- We deploy over 1 MB of Javascript
- Our customers: technical Chiefs of Operations

Phase 1: Getting Javascript to the Browser

- Concatenate Javascript files and CSS files
- Minify and obfuscate Javascript, minify CSS:
  YUI Compressor, JSMin, Packer, UglifyJS, Google's Closure Compiler
- Serve with gzip compression
- Insert `<script />` elements immediately before `</body>` tag

Phase 2: Choices

- “Framework A” versus “Framework B”
- “Technique 1” versus “Technique 2”
- Educate yourself and make your own decisions

Javascript in Circonus

- Do not build page content in the browser
- Building content on the server is 3-5 times faster: http://openmymind.net/2012/5/30/Client-Side-vs-Server-Side-Rendering/
- Use Javascript to add functionality on top of content

Start with Structure

- Frameworks are restricting
- Focus on your app, not your framework
- Circonus needs flexibility to handle a wide variety of content

Choose a Namespace

- Global “`circonus`” object
- Sub-objects grouped logically (e.g. “`circonus.graphs`”)
- Break logical groupings into their own files (e.g. “`circonus.graphs.js`”)
- Don’t get too complicated, this is just to help you
- Use an automated build script to minify, obfuscate, & concatenate the files back together when deploying

Basic Building Blocks

- There are many design patterns: module, composite, et al, but I don’t favor one in particular
- The most basic unit is the Function
- Functions can do it all:
  - They have their own variable scope
  - Allow class-like structuring using prototypal inheritance, allowing public & private properties / methods

Three Ways I Use Functions

1) Page initialization functions
2) Utility functions
3) Component functions

Page Initialization Functions

- Called at the bottom of the page:
  one of only two points where there is Javascript in our HTML
- Contains procedural code to setup bindings and other functionality
- In Perl templates:

  ```
  $request->page_js("circonus.graphs.initPage();")
  ```

  All JS is thus collected throughout the template and is output in a single `<script />` tag at the bottom of the page.

Utility Functions

- Keep them in their own namespace: `circonus.utilities`
- Contain snippets: repeated procedures
- Reduce overall Javascript file size

Component Functions

- Tied to page elements
- Look for repeated objects (patterns) in your page structure
- I have two classifications:
    1) Tracking functions (using closures)
    2) Constructor functions

Components: Tracking Functions

- Most useful when matched by backend components
- Example: a toolbar for graphs
  - `graph_date_tool.inc` calls `circonus.graphs.initDateTool()` (this is the second point where there is Javascript in our HTML)
  - Enables easy use of closures:

```
circonus.graphs.initDateTool = function() {
  var $tools = [];
  return function init() { /*do things here*/ }
}();
```

Components: Constructor Functions

- Called with the “`new`” keyword (e.g. “`new circonus.Graph()`”)
- Most useful for multiples of objects (great for encapsulating config data)
- Enables prototypal inheritance (saves memory)

```
circonus.Graph = function Graph(cfg) {
  this.destroy = destroy;
  function destroy() { /*destroy bindings here*/ }
  return this;
};
circonus.Graph.prototype.gotoView = function gotoView(){};
```

Be Wary of Whole Libraries

- Don’t add extra libraries without consideration
- Our goal: keep total Javascript size as small as possible
- Always ask:
  - Could I write this functionality?
    (specialized code is smaller than generic code)
  - Can I cut out parts?
    (leave documentation if you chop it up)
- Github forking makes it easy to maintain parallel branches of libraries

Documentation and White Space

- Why don’t we document? Bad habits…? Wasted time…?
- Spend 1 minute now to save 2 minutes later
- Documentation and white space are crucial for large applications
- Personal testimony: “future you” will thank yourself!

Continuous Deployment

- Deploy updates daily or weekly with Git
- Near-instant bug fixes
- Improve complex features over time (example: our new tags feature)

Feature Flags

- JSON config file detailing features to toggle

```
  { features:{ tags:{ default:0, force:0 } } }
```

- Hidden page with toggle switches
- A cookie tracks which features are enabled,
  with body classNames for CSS hooks
- Allows us to block unfinished features or
  release features as “beta” features to select customers

Dev Mode Feature

- Toggles between development code and built code (minified, obfuscated & concatenated)
- Allows debugging on live production server
- Debug with 20 development scripts instead of 1 production script

Message Center (PubSub)

- Simplifies and clarifies communication
- DOM events:
  - OK when document elements are involved
  - Too slow and decentralized for general communication (event bubbling)

How PubSub Works

- Only three methods:
  - `message_center.subscribe('namespace', callback)`
  - `message_center.unsubscribe('namespace', callback)`
  - `message_center.publish('namespace', 'message')`
- Subscriptions are namespaced ("`graphs`" versus "`graphs.toolbar`")
- No longer need a dozen custom event bindings

Multiple Concurrent Sessions

- Needed a way to track multiple tabs
- Each tab has its own set of filters (filtered by tags)
- When navigating in a tab, it should keep its own filters

Each Tab Needs an Identity

- Setting a window property (`window.id`) didn't stick
- `localStorage` is too global and persists beyond the session
- `sessionStorage` lasts for the entire session
  but is only available to the current tab

Talk to the Back-End

- A cookie tracks which tab is visible
- Tab "`focus`" event sets that tab as the visible tab in the cookie
- Data can be saved as pertaining to only the visible tab,
  then matched to the visible tab upon page load

Problem #1: Loading in the Background

- When a tab loads, it normally sets its ID as the visible tab
- Check for visibility at load time
- Hidden tabs shouldn’t set their IDs as visible tabs
- Check for visibility with the Page Visibility API

```
var prop_name = undefined !== document.mozHidden    ? ‘mozHidden’
              : undefined !== document.webkitHidden ? ‘webkitHidden’
              : undefined !== document.msHidden     ? ‘msHidden’
              : ‘hidden’,
    is_hidden = document[prop_name];
```

Problem #2: Refreshing in the Background

- If content is loaded in a background tab, the cookie will have the wrong ID
- Hidden tabs shouldn’t load or refresh content
- Check for visibility with the Page Visibility API

```
var prop_name = undefined !== document.mozHidden    ? ‘mozHidden’
              : undefined !== document.webkitHidden ? ‘webkitHidden’
              : undefined !== document.msHidden     ? ‘msHidden’
              : ‘hidden’,
    is_hidden = document[prop_name];
```

O'REILLY
Velocity
China 2013
Web性能与运维大会
Beijing, China
Aug 20-21, 2013
velocity.oreilly.com.cn