Building a JavaScript Library

John Resig (ejohn.org)
jQuery JavaScript Library / Mozilla Corporation

August 18th, 2007 - Google Tech Talk

jQuery

- Released Jan. 2006
- Focus on DOM Traversal
- Built in support for Events, Ajax, and Animations
- Succinct code, small file size
- Extensible via a plugin architecture

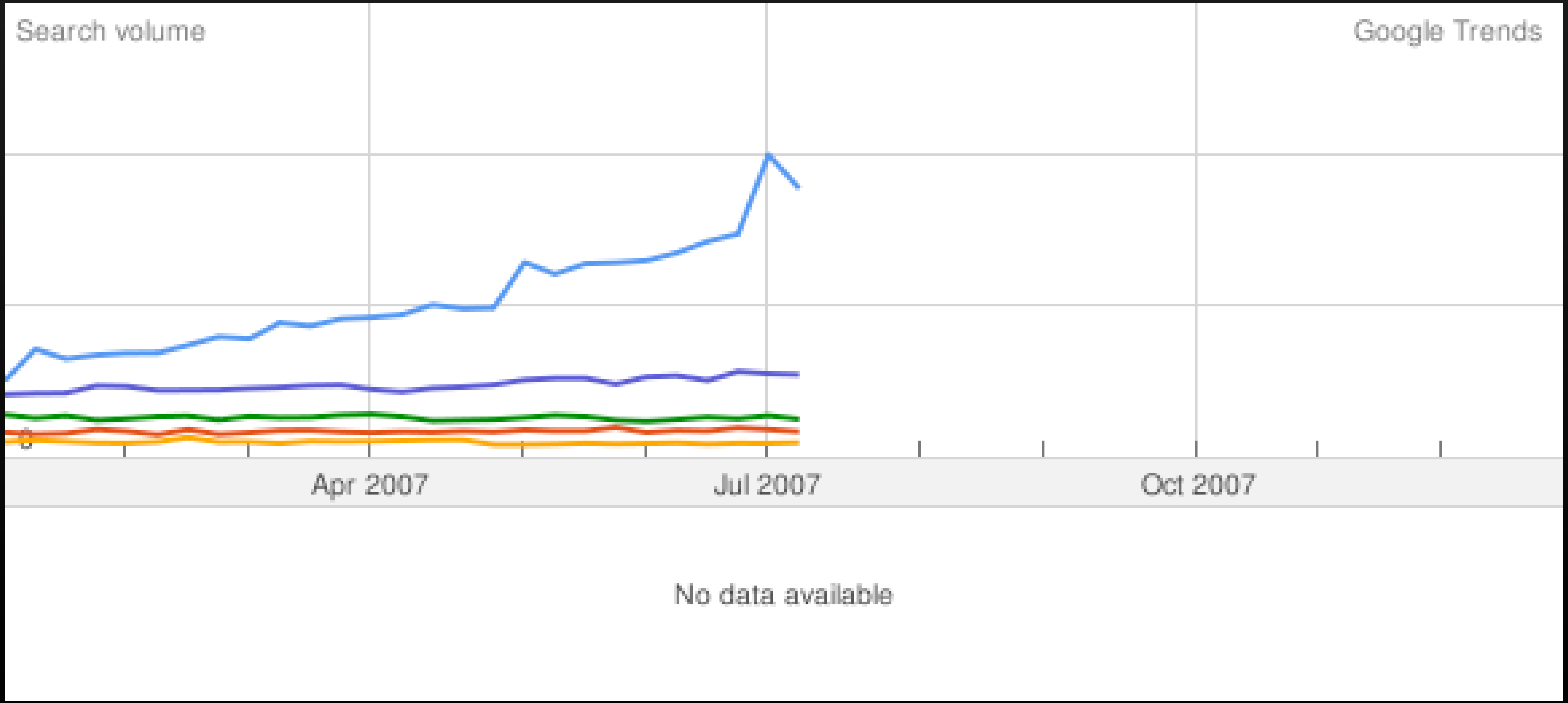
jQuery Samples

- `$(“#main div”).addClass(“test”);`
- `$(“#main”).slideDown(“slow”);`

-
  ```
  $(“ul > li”).click(function(){
      $(this).find(“ul”).toggle();
  });
  ```
- `$(“#contents”).load(“doc.html”);`

Doing something right?

- ~25 people on the jQuery Team
- 1/4 million visitors per month



- Fantastic Growth (via Google Trends)

FUEL

- To be included in Firefox 3
- A JavaScript library for extension developers
- Provides helpers for: Browser tabs, bookmarks, events, preferences
- Written in pure JavaScript, extensible

FUEL Samples

-
  ```
  Application.prefs.all.forEach(function(p){
    if ( p.modified )
      // do something
  });
  ```

-
  ```
  Application.events.addListener(“quit”, fn);
  ```

-
  ```
  Application.browser
      .open("http://google.com/").active = true;
  ```

-
  ```
  Application.bookmarks.all.forEach(function(cur){
      if ( cur.url.match(/google.com/) )
          cur.remove();
  });
  ```

Need to know:

- Writing a Solid API
- Implementation
- Complex Applications
- Browser Bugs
- Documentation
- Testing
- Maintenance

Writing A Solid API

Orthogonal

- Perform Universal Actions
- CRUD your methods! add/remove/delete/modify
- FUEL has the following on every object:
  - .all, .add(...), .remove(...), .get(...)
- jQuery was missing .removeAttr() for the first couple months (oops!)
- Made a grid for FUEL, filled in the blanks

Fear Adding Methods

- Methods can cause a support nightmare
- Avoid adding, if you can
- Defer to extensibility
- jQuery:
  - Added .top(), .left(), .background()
  - Duplicate of .css() - unnecessary

Embrace Removing Code

- Remove un-used code
- Reduces the size of your API
- Reduces your filesize
- Make your code more maintainable
- jQuery: Ran a poll to remove CSS 3 selectors.
- Reduced size of the API by 47% in 1.1

Provide an Upgrade Path

- Expect compatibility changes
- Provide transitional plugin
- Plugin for jQuery 1.1, for jQuery 1.0 users (+ documented in 3 languages)

Reduce to a common root

- Look for common patterns
- Reduce to its core and build up
- jQuery:
  - .eq(0), .gt(3), and .lt(2)
  - why not just implement slice?
    .slice(0,1) or .slice(1,4)

Consistency

- Users can "expect" good naming
- Pick a naming scheme and stick with it
  - .click() vs .onclick()
- Argument position
  - .method(options, arg2, ..., callback)
- Callback context

  -
    ```
    .method(function(){
        // this == DOMElement
    });
    ```

Implementation

Evolution of a JavaScript Coder

- “Everything is a reference!”
- “You can do OO code!”
- “Huh, so that’s how Object Prototypes work!”
- “Thank God for closures!”

Functional Programming

- Closures are essential
- Understand this snippet:

  -
    ```
    (function(){
      // your code...
    })();
    ```
  - Great for ‘local variables’
- Perfect for macros

  -
    ```
    var event = [‘click’,’focus’,’blur’,...];
    jQuery.each(event,function(i,name){
        jQuery.prototype[name] = function(fn){
            return this.bind(name,fn);
        };
    });
    ```

Quick Tip: Local Vars

-
  ```
  (function(){
    // your code...
    var test = false;

    this.method(function(){
      return test;
    });
  }).call(this);
  ```
- Locally-scoped variables
- Access to instance

Encapsulation

- Contain all of your code to a scope
- Hide your code and prevent it from leaking
- All of your code should be wrapped in a:

  -
    ```
    (function(){
      // your code...
    })();
    ```
- BONUS! Your code compresses really well with Dojo Compressor, et. al.

Namespacing

- Use as few global variables as feasible
- One namespace is optimal
  (see: Dojo, Yahoo UI, jQuery, MochiKit)
- Questions to ask:
  - Can my code coexist with other random code on the site?
  - Can my code coexist with other copies of my own library?
  - Can my code be embedded inside another namespace?

Don't Extend Native Objects

- Down this path lies great pain and suffering
- Impossible to get DOM extensions working cross-browser
- Object.prototype kills kittens
- JS 1.6 methods cause conflicts:
  - elem.getElementsByClassName
  - array.forEach, .map, .filter

Perform Type Checking

- Make your API more fault resistant
- Coerce values wherever possible
  - `.css(Number)` becomes:
    `.css(Number + “px”)`
  - `.map(String)` becomes:
    `.map(new Function(String))`
- Error messages
  - Quite useful
  - Byte-costly
  - Solution: Move to ‘debugging’ extension

Quick Tip for OO

- Tweak your Object constructor

-
  ```
  function jQuery(str, con){
      if ( window == this )
          return new jQuery(str, con);
      // ...
  }
  ```
- Make it easier for users:

  `new jQuery(“#foo”)` becomes:

  `jQuery(“#foo”)`

Quick Tip for Errors

- Never gobble errors
- Ignore the templation to `try{...}catch(e){}`
- Improves debug-ability for everyone

Complex Applications

Extensibility

- Your code should be easily extensible
  - Methods: `jQuery.fn.method = fn;`
    `$(...).method();`
  - Selectors: `jQuery.expr[‘:’].foo = “...”;`
    `$(“:foo”)`
  - Animations: `jQuery.easing.easeout = fn;`
    `.animate({height: 100}, “slow”, “easeout”);`
- Write less, defer to others
- Makes for cleaner code
- Foster community and growth

Pure OO Code

- Object-Oriented is only one answer
- JavaScript != Java
- Not a good solution
  - At least not until JavaScript 2
  - Classes, Packages, etc.

Custom Events

- The answer lies in custom events
- Dojo, jQuery, Yahoo UI - just added to Prototype
- Components trigger events and listen for others
  - .bind("drag",fn)
  - .trigger("refresh")

Browser Bugs

The Quirksmode Problem

- Fantastic resource
- Tells you where problems are
- Doesn’t tell you how to fix them
- Need to focus on problem sets
  - Events
  - Get Attribute
  - Get Element Style

Solving a Problem

- Some issues are solved in depth
  - DOM Events
  - DOM Traversal
- Many still require hard work:
  - getAttribute
  - getComputedStyle
- Permute your test cases, look for edge cases

When to run the fix

- Typical thought progression:
  - “I’ll just look at the useragent”
  - “I’ll just detect to see if an object exists”
  - “I’ll just think of an elegant solution to the problem”
  - “I’ll just look at the useragent”

Documentation

Structured

- Provide a clear format
- Users can build new views with it
  - No API view is perfect
- jQuery users built:
  - API browsers
  - Cheat sheets
  - Widgets
  - Translations
- An API for your API!

Users Want to Help

- Make barrier to helping very low
- Answer: Keep your docs in a wiki
- Only do this if you’ve already written all of your docs
  - Wiki != Documentation
- Use templates to maintain structure

Focus on Leverage

- Write the docs that will get the most benefit
- Rough Priority:
  - User-centric API
  - Plugin authoring
  - Docs on writing docs
  - Advanced plugin authoring

Write the Docs Yourself

- It isn’t glamorous, but it’s essential
- You must buckle-down and do it yourself
- Improves your longevity and uptake

Testing

1000% Essential

- Don’t trust any library that doesn’t have a test suite
    - Good: Prototype, MochiKit, jQuery, Yahoo UI
- Write your own suite (they’re pretty easy)
- Be sure to handle async tests
    - Ajax
    - Animations
- Pass in all supported browsers

Test-Driven Development

- Write test cases before you tackle bugs
- Find devs who love to write test cases
- Check for failures before commit
- Pure JS DOM (running in Rhino) can help spot obvious errors
  - pre_commit hook in SVN, if you can *cough* Google Code *cough*

Future of Testing

- Distributed Multi-Browser Testing
- How it works:
  - Report results back to central server
  - Server pushes new tests/code to clients
  - Repeat
- jQuery and Prototype are very interested in this (expect something soon)
- Test against browser nightlies
  - See: JS lib test cases in Mozilla trunk

Maintenance

Tackling New Bugs

- Create a rapid test environment
  - ./gen.sh bugnum (dom|ajax|selector|...)
- Your test suite must be passing
- Have good coverage
- Always test in all supported browsers to avoid regressions
- Check unsupported browsers before release
  - Don't want them to crash/error out

Use Your Community

- Use your community as a sounding board
  - Gauge the usefulness of features
- Users are great for finding weird edge cases
- Pay attention to repeat questions
  - Indicative of problem in API design
  - or lack of documentation

Maintain Focus

- Very, very, important
- Users generally choose libraries for ideological reasons
- Breaking your ideology will alienate your users
- jQuery: Small, concise, code. Small download, light core, easily extensible.
- Use plugins to divert functionality from your core

More Details

- Contact Me: jeresig@gmail.com
- More details:
  - http://ejohn.org/
  - http://jquery.com/
  - http://wiki.mozilla.org/FUEL

Fun

Fun With One Liners

- “I’m not suggesting that one-liners are the heart of all programming. But they can be the hook to get someone exploring. A single line of code simply shows that a language can be focused. And it lets a beginner get comfortable with the basic atoms.” - _why
- jQuery allows for fantastic one liners

One-Liners

- `$("div.section")`
- `$("div.section").removeClass("section").hide();`
-
```
 $("div.section")
  .find("dt")
    .addClass("section")
    .click(function(){
      $(this).next().toggle();
    })
  .end()
  .find("dd")
    .hide()
    .filter(":first")
      .show()
    .end()
  .end();
```

One-Liners

- Remove unsightly anonymous functions!
-
```
$("div.section")
  .find("dt")
    .addClass("section")
    .onclick()
      .next().toggle().end()
    .end()
  .end()
  .find("dd")
    .hide()
    .filter(":first")
      .show()
    .end()
  .end();
```

Domain-Specific Language

-
  ```
  $("div.section")
    find("dt")
      addClass("section")
      click(
        next()
          toggle()
      )
    end()
    find("dd")
    hide()
      filter(":first")
        show()
      end()
    end()
  ```

Domain-Specific Language

-
  ```
  $("div.section"
    find("dt"
      addClass("section")
      click(
        next(
          toggle())))
    find("dd"
    hide()
    filter(":first"
      show())))
  ```

Domain-Specific Language

- Lisp-y!

-
  ```
  ($ "div.section"
    (find "dt"
      (addClass "section")
      (click (next (toggle))))
    (find "dd"
      (hide)
      (filter ":first"
        (show)))))
  ```

Domain-Specific Language

- Python-y!

-
  ```
  div.section:
    dt:
      addClass "section"
      click
        next toggle
    dd:
      hide
      :first: show
  ```
- Give it a try:
  http://ejohn.org/apps/jquery2/
- Secret Sauce:
  `<script type=”text/jquery”>...</script>`