Javascript Performance in the Browser

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

Performance + Maintainability

Look to the Past

Behavioral Separation

- Web UI is in three layers:
  1) Content Layer – HTML
  2) Presentation Layer – CSS
  3) Behavior Layer – JS

- Gray (in-between) areas are ok

Content Layer

- Don’t use inline styles or event handlers

```
<a href=”/mag” style=”color:red;” onclick=”doIt();”>
```

- Having inline styles and event handlers mixes up your layers:
  - No context
  - No documentation
  - No high-level overview

Presentation Layer

- Easy to keep clean, but hard to keep in one place
- Let stylesheets do their job, don’t let Javascript take over

Behavior Layer (Don't Mix with Content)

- Don't build the content layer with Javascript (page templates, etc.)
- Building content in Javascript is 3-5 times slower than doing it on the server
  http://openmymind.net/2012/5/30/Client-Side-vs-Server-Side-Rendering/
- Don't mix HTML strings into your Javascript – they can't be obfuscated
- Minimize all strings in your JS (e.g. classNames)

```
var a_class   = "active",
    is_active = $link.hasClass(a_class);
$table.addClass(a_class);
```

Behavior Layer (Don’t Mix with Presentation)

- Libraries like jQuery make it easy to mix behavior and presentation, but DON’T DO IT
- Visual appearance is NOT the realm of Javascript
- Decouple (un-link) visual appearance from behavior controls

Behavior Layer (Working with Presentation)

- Javascript should only change the state of elements

```
$link.addClass(“active”);
```

- CSS will then look at the state and change the visual appearance

```
.link { color: black; }
.link.active { color: red; }
```

Keep Behavior Layer Clean (for the Future)

- It’s all about maintenance
- Don’t allow cruft to accumulate in your codebase
- Maintenance doesn’t make your application faster TODAY, but it does prevent it from slowing down TOMORROW

Now Back to the Future (for some practical tips)

Operating in the Browser

- Don’t worry about micro-performance tweaks
- Document == traffic jam
- Touch the document as seldom as possible

Save References to Everything

- Get element references as soon as possible (at load time)

```
var $table = $(“.table-one”),
    $form  = $(“#login-form”);
```

- Save attribute values & property values

```
var old_h  = $link.attr(‘href’),
    prefix = old_h.match(/^https/) ? “secure:” : “”;
$link.text(prefix + old_h);
```

Use Fast Selectors

- Don’t use modern query selector methods
```
querySelector()
querySelectorAll()
```
- Use older dedicated methods
```
getElementById()
getElementsByClassName()
```
- Even libraries like jQuery use these methods

Goodbye, Javascript Transitions

- Until recently, Javascript was our only option for transitions
- Anything is possible, but at a performance cost
- Not great for mobile – mobile Javascript is VERY slow: http://sealedabstract.com/rants/why-mobile-web-apps-are-slow/

Welcome, CSS Transitions!

- Widely compatible with modern browsers
  (Internet Explorer 7 - 9 are the exceptions)
- Not for cartoon animations, just to give polish to your interface
- Most numeric properties can be transitioned, including colors
- Still requires prefixes

```
-moz-transition: width 0.5s ease-out;
-webkit-transition: width 0.5s ease-out;
-o-transition: width 0.5s ease-out;
transition: width 0.5s ease-out;
```

TransitionEnd Event

- You can listen for when transitions are finished
- Be careful of multiple events
- Still requires prefixes

```
webkitTransitionEnd
oTransitionEnd
otransitionend
transitionend
```

Transitioning to “Auto”

- “`height: auto;`” cannot be transitioned to / from
- Use “`max-height`” with “`overflow: hidden;`” for clipping:

```
.menu {
  height: auto;
  max-height: 0;
  overflow: hidden;
  transition: max-height 0.5s ease-out;
}
.menu.active {
  max-height: 10em;
}
```

Pitfall #1: Memory Usage

- Most developers don't pay attention to memory usage (Garbage Collection is automatic, but computationally expensive)
- Plotting graphs on canvas elements with Flot:
  1000 x 300 px = 300 k px … x2 = 600 k px or 2.4 MB per graph
- Graphs are re-plotted every 5 minutes, for hours / days

O'REILLY®

Velocity

Web Performance and Operations

CHINA

Beijing, China

Dec 4–5, 2012

velocity.oreilly.com.cn

“Some people, when confronted with one problem, think
‘I know, I’ll use regular expressions.’

“Now they have two problems.”

Pitfall #2: Regular Expressions

- Rewriting a function which tokenized a formula, and decided to try Regular Expressions (RegExp)
- Pulling sets of letters out of a formula: `/\b[a-z]+\b(?!\()/`
- Tested beforehand to get a performance baseline
- With RegExp, Firefox was 4% slower
- With RegExp, Chrome was 250% slower!

Always test, never assume…
What happens when you assume?

assume = “ass” + u + me

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