Building Apps for the Open Web

Early Release

HTML5 Architecture

O'REILLY®

Wesley Hales

2

I ♥ The Mobile Web

It’s estimated that there will be one billion HTML5-capable phones sold in 2013. The ground swell of support for HTML5 over native is here and today’s developers are flipping their priorities to now put mobile development first — which is why this chapter comes first [note: it may actually be second technically speaking]. Whether you’re an HTML5, W3C Standards lovin’, Open Web expert or just coming fresh off HTML 1, this chapter will equip you with the latest code, trends, and market research to guide you on making the right decision for your next mobile web project.

The Mobile Web refers to browser-based applications created for mobile devices, such as smartphones or tablets, which are (at some point) connected wirelessly. Since 2008, the web has shifted towards focusing on mobile browsers, which are delivering an overall better quality of life for today’s web developers and users. However, this better quality of life is sometimes short lived after you go around testing your new mobile web app on the myriad of devices and browsers. There is a huge question of “What is supported and which HTML5 features should I build my app with?”.

This chapter tries to answer the tough questions we are all faced with as mobile web developers. Those questions which will have a profound impact on your new initiative for years to come. So what are you waiting for? Level up!

Mobile First

First, let’s get our priorities straight. Prioritizing mobile design and development over the desktop was once laughable. In just a few years, mobile first has taken over, giving web developers a breath of fresh air in terms of HTML5 based APIs towards hardware access on mobile devices.

Apart from the obvious, there are multiple reasons for thinking mobile first:

- Developing sites for constrained devices and resolutions will force you to create more fluid/flexible content.

- Device features, such as accelerometer and geolocation hardware, present new business opportunities with technologies like Augmented Reality.
- Overall, mobile first requires you to think in a code quality mindset. Today, it’s required for developers to worry about things like battery life when doing hardware accelerated animations with CSS; This quality of development not only brings better performing apps, but it also encourages you to focus on semantics.
- As you wean yourself off of desktop focused web development, mobile browsers give you a glimpse into the future. This allows you to stay on the bleeding edge and in touch with new specifications and features.

Unfortunately the Mobile Web isn’t write-once-run-anywhere yet. As specifications become final and features are implemented, interoperability will be achieved. However, in today’s world of mobile browsers, we don’t have a largely consistent implementation across all browsers. And even though new tablets and phones are constantly being released to achieve a consistent level of HTML5 implementation, we all know that we’re stuck supporting these older devices for a set amount of time. So, needless to say, devices like the iPhone 3G and any device which hasn’t upgraded past Android 4 will be the IE6’s of this mobile era.

At this point, you’re probably headed in one of three directions:

- You wish to write a Mobile Web only HTML5 based app.
- You’re looking to create a new web application for both mobile and desktop clients.
- You are converting an existing application to work on mobile devices.

We’ll address each of these scenarios in order, most fun to most painful. But first, we need to know what our target devices and browsers are capable of.

What’s Supported

As the mobile landscape exists today, we have multiple platforms and browsers to support and code for. By using core HTML5 API’s, you’re bound to what is supported by your target devices. So it’s critical to understand where the mobile browser scene is today, and where it’s headed.

Writing mobile web apps, which span all platforms and all browsers, can be a huge undertaking. Previously, web app developers didn’t care if your desktop computer had a camera or accelerometer attached to it. The web applications of yesterday were not tied to the operating system and the capabilities of your desktop hardware. Now, the mobile web adds another dimension of support to the apps we build and the fragmentation across browsers and devices is mind-blowing. We must now create applications to be compatible across browsers, platforms, AND devices.

For example, Android’s WebKit based browser may have supported Web Workers in version 2.1, but later disabled support in version 2.2, 3.0, and 4.0. Then it gets fixed and turned back on in 4.1! Confusing, right? This is what I mean by another dimension of support or “fragmentation”. You’re not only supporting browsers, but the operating system it’s tied to as well.

But, not to worry, next we’re going to look at the browsers, find out what is commonly supported per device, and identify a core set of features which we can build a solid enterprise mobile web app from.

For the latest matrix of HTML5 support across all rendering engines see: http://en.wikipedia.org/wiki/Comparison_of_layout_engines_(HTML5)

Mobile Web Browsers

Let's take a look at the various mobile browsers and their respective communities.

WebKit

WebKit (http://www.webkit.org/) is not just a web browser engine. It's a growing, open source project, with a welcoming and approachable community. WebKit is constantly pushing the HTML5 envelope, adapting to the latest W3C specifications as they're published. The recent explosion of interest in WebKit can be attributed to the fact that it powers many of the leading mobile platform browsers. This includes Android, Mobile Safari, PalmPre, Kindle, Nokia S60, and BlackBerry.

Figure 2-1 shows the source code revision (vertical) as the function of time (horizontal). Some icons are there to represent few products associated with WebKit, the position approximately resembles the era those products were made popular.

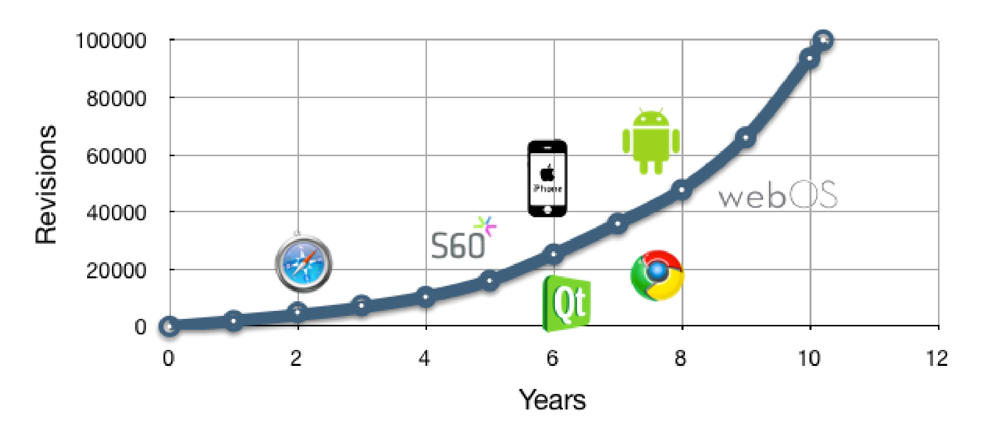


Figure 2-1. WebKit Revisions

Mobile Safari (iOS5)

Apples' adoption and implementation of early HTML5 specifications has been impressive. They have been an obvious force in pushing the web forward. With standard hardware and multi-core technology, iPhones and iPads have been a great platform for HTML5 development. But, it's not all ponies and rainbows in iOS land, as each mobile browser has it's share of quirks and bugs. Earlier iOS versions have suffered from a bug with JavaScript's innerHTML() and forced developers to think of new ways to insert dynamic content. We'll see the solution to this problem in the next chapter, as for now, we'll focus on the big picture.

Apple's community process around iOS progression and filing bugs is bound and limited to the way Apple chooses to do things. You can file bugs with their BugReporter (bugreport.apple.com) but you can only search through issues that you submit. Luckily, once again, the community has stepped up to give Apple a hand in allowing non customer-confidential data to be openly searched to see if your bug has already been filed (http://openradar.appspot.com/faq).

Android

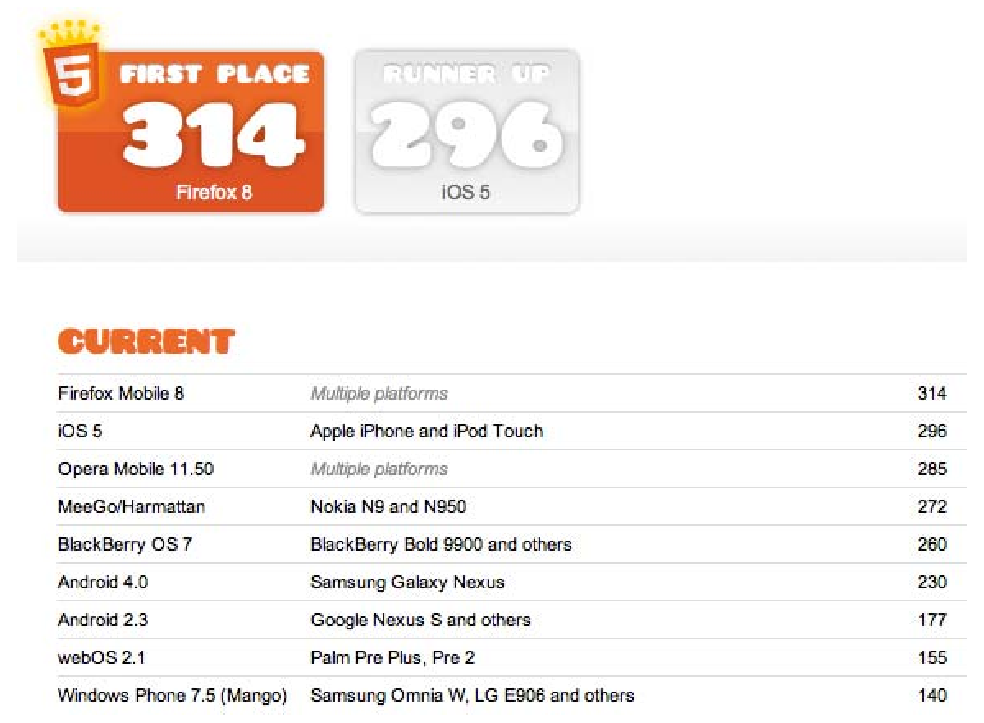
Even though the Android default browser is based off of WebKit, as of this writing, its implementation of HTML5 specifications is just starting to beef up in version 4. As Android evolves, we can rest assured that the coming HTML5 implementations will evolve with its community.

But for now, Android devices are horribly fragmented and HTML5 support varies on devices and OS versions.

http://source.android.com/community/

Mobile Firefox

Mozilla has been around for a while and is stronger than ever in focusing on community efforts and pushing the web forward. As of this writing, Mobile Firefox has trumped iOS’s Mobile Safari in terms of implemented HTML5 features.



CURRENT

Firefox Mobile 8	*Multiple platforms*	314
iOS 5	Apple iPhone and iPod Touch	296
Opera Mobile 11.50	*Multiple platforms*	285
MeeGo/Harmattan	Nokia N9 and N950	272
BlackBerry OS 7	BlackBerry Bold 9900 and others	260
Android 4.0	Samsung Galaxy Nexus	230
Android 2.3	Google Nexus S and others	177
webOS 2.1	Palm Pre Plus, Pre 2	155
Windows Phone 7.5 (Mango)	Samsung Omnia W, LG E906 and others	140

Figure 2-2. From http://html5test.com/results-mobile.html

This dethroning will continue to happen as the Mobile Web moves forward and evolves — and it’s a good thing. We want competition and standards progression. Mozilla is no stranger to the evolution of the Mobile Web with the ambitious new project called WebAPI (https://wiki.mozilla.org/WebAPI). The WebAPI project is a set of APIs for accessing device functionality usually accessible only for native applications. In summary, it’s an HTML, CSS, JavaScript based OS for mobile devices! It’s yet another effort to move the web forward enabling developers to write web applications once for all mobile OSes. Estimated delivery for the WebAPI project is mid 2012 through the Boot to Gecko project (B2G). Figure 2-3 shows a screenshot of B2G’s Gaia UI.

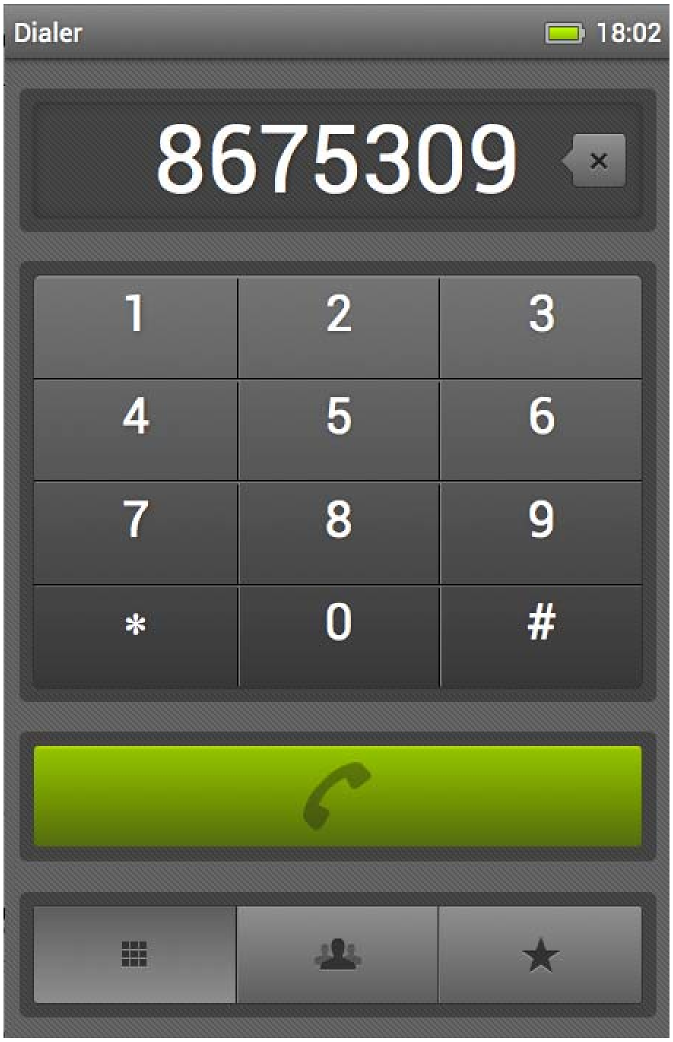


Figure 2-3. B2G’s Gaia UI

Opera Mobile

Opera has two separate browsers for mobile phones: Opera Mobile and Opera Mini.

In Opera Mini, the Opera Presto browser engine is located on a server. In Opera Mobile, it is installed on your phone. Currently, Opera Mini holds a large percentage of market share amongst other browsers, but for enterprise HTML5 applications, Opera Mobile supports the core specifications we need, such as WebStorage, WebWorkers, and GeoLocation.

Internet Explorer Mobile

Windows Phone 7.5 features a version of Internet Explorer Mobile with a rendering engine that is based on Internet Explorer 9. So the simplest way of answering the question of “What does Windows Phone support?” is to say that it supports what IE9 supports, including WebStorage and GeoLocation.

IE Mobile support may change in the near future after more specifications from the W3C are made final. But the message they’re sending to date is interoperability.

The supported specifications for IE9 Mobile can be found here: http://windowsteamblog.com/windows_phone/b/wpdev/archive/2011/09/22/ie9-mobile-developer-overview.aspx

Mobile Browser Market Share

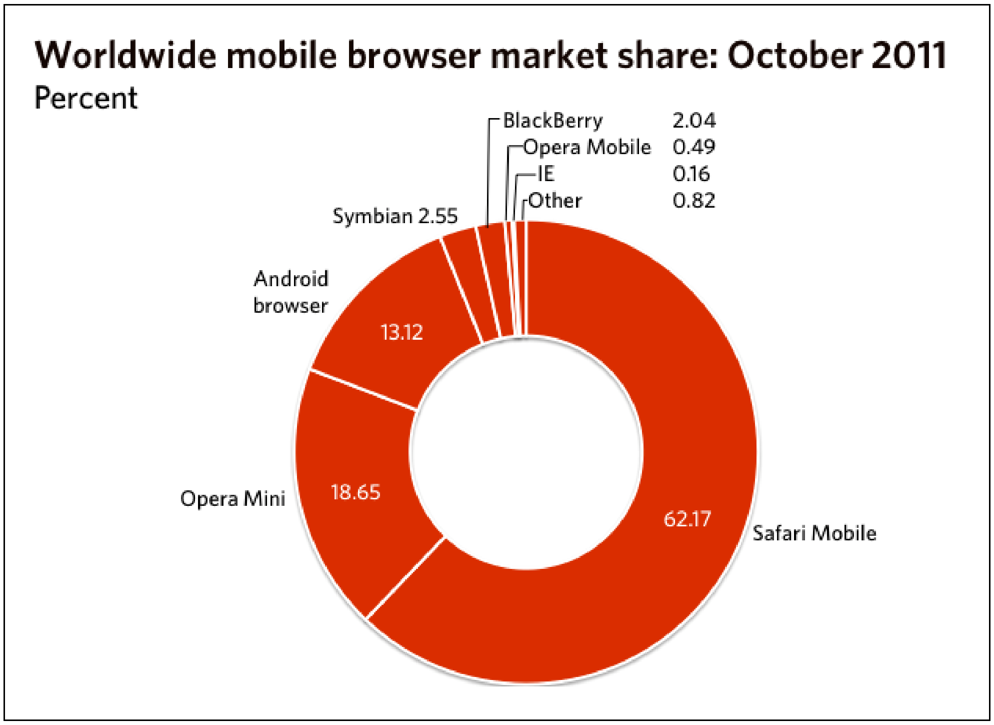


Figure 2-4. Worldwide Market Share – October 2011

[Note: Will need to update this chart before publish]

As of the latest worldwide report on browser market share, we see that WebKit based browsers are clearly in the lead with over 75% of the market. Right now, Android and iOS dominate, but as new operating systems emerge, like Mozilla’s HTML5 based mobile B2G project, we could see another shift of power in the ongoing “browser war”.

All of this information leads into the important topic of browser grading. Browser grading is a must for any mobile web project. It gives developers and QA a way to keep sane while developing and testing your application. It also sets forth a specific support schedule for your users and an overall target for what your mobile web app is capable of.

Table 2-1. Browser Grading Example
http://www.quirksmode.org/blog/archives/2010/08/first_serious_s.html

A : High Quality	A high quality browser with notable market share. A must-target for a mobile web developer.
B : Medium Quality	Either a lower quality browser with high market share or a high quality browser with low market share. Depending upon your capabilities you should work to support these browsers, as well.
C : Low Quality	Typically an extremely low quality browser with high market share. Generally not capable of running modern JavaScript or DOM code.
F : Failing	A barely-functioning browser. Even though it has some market share you should avoid developing for it completely.

HTML5 in the Enterprise

Now that we understand the mobile device and browser landscape, let's move on to the W3C specifications they support and how we can use them. In terms of enterprise development, there are certain HTML5 API's that are considered the advanced building blocks of today's mobile web applications. These are the specifications on last call from the W3C, or close to final, and are considered to be (somewhat) stable and adopted in today's mobile browsers.

Of course, there are many other specifications like the Media Capture API, allowing access to the device audio, video, and images, but we are looking at what is most widely supported across leading devices as of this writing.

Below, we have a subset of HTML5, or Open Web, specifications showing what is currently supported in the five leading and/or upcoming mobile platforms. From hereon, I will refer to the specifications and browsers in the following table as "HTML5 Enterprise" or "HTML5E".

HTML5 Enterprise (HTML5E)

Table 2-2. HTML5 Enterprise (HTML5E)

OS/API	Geolocation	WebSocket	WebStorage	Device Orientation	Web Workers
Mobile Safari	Yes	Yes	Yes	Yes	Yes
Android	Yes	No	Yes	Yes	No
Mobile IE	Yes	No	Yes	No	No
Opera Mobile	Yes	Mixed**	Yes	Mixed*	Yes
Mobile Firefox	Yes	Mixed**	Yes	Yes	Yes

*Opera Mobile for Android has experimental support

**Both Mozilla and Opera have temporarily disabled WebSockets due to security issues with the protocol.

Here we see Mobile Firefox and Safari are the clear winners with Opera Mobile coming in at a close third. Android still has some work to do, however version 4 is looking much better. Mobile IE, which is IE 9, is focusing on a "same markup" approach — which leaves us with little IE support for HTML5E.

All of these mobile browsers span the browser grading chart from the previous section and are considered grade "A", "B" and "C" browsers. This is a typical situation within most development shops; where you're asked/told to support the cool browsers but then there's that one customer who uses Mobile IE — ah yes… this reminds you of your old IE6 days, doesn't it?

So let's be realistic as we setup our demo application in the next chapter. We're going to focus our demo code on the "A" graded, WebKit based (Mobile Safari and Android) browsers. Most, if not all, mobile web initiatives start with developing for WebKit and then building out to support and test other platforms.

We now have a starting point; a decent view of which HTML5 API's are supported within mobile device browsers. In terms of the future, W3C, spec driven, device features are only guaranteed to get better as new device OS's are released and the specification themselves become final.

Note: For the latest Mobile HTML5 support information, check out http://caniuse.com/ and http://mobilehtml5.org/.

The Mobile Web Look and Feel

The Native vs. Mobile Web debate isn't about which programming model will win. It's about what can we build until HTML5-like technologies catch up. We have pure native approaches which are clearly winning today in terms of overall application responsiveness, then we have hybrid approaches and frameworks which try to bridge the gap of HTML5 and native, and finally we have the true, bleeding edge, mobile web frameworks which are trying to conquer the native feel with markup, JavaScript, and CSS.

Couple a fast and responsive mobile web app with your existing enterprise infrastructure and let the games begin. Web standards are quickly closing the gap on missing native features and device makers are catching up on implementing them. As of Android 3.1, it's possible to capture photos and videos due to the Media Capture API specification.

The W3C is a busy place these days, and developers are moving specifications and better use cases forward. Projects like jQuery are calling on the open source community to participate in these specifications and to submit their ideas for a better web.

It only makes sense that mobile developers are leaning in favor of writing once, and running their app anywhere. "Write once, run anywhere" or WORA received a lot of fanfare after Sun's JVM started to emerge in the enterprise. With HTML5, WORA basically means you can use standard JavaScript and CSS to access all of the device features that a native application can (the device GPS, camera, accelerometer, etc.). This approach has given new life to browsers and a language (HTML) that was once only used to serve up documents - not apps.

What are some of the requirements when trying to achieve a native look-and-feel? What should it look like?

The Mobile Web Look

To truly achieve that native look-and-feel, not only does our app need to respond quickly, but it must also look good. These days, the big secret to getting your native app listed in an App Store top 10 list is to have a good looking design. That's all it takes. If you have a killer data driven application using all the latest device bells and whistles, it will not make it very far without a good clean design.

IOS definitely has its own mobile web look-and-feel which mimics its native apps, but what about Android, Windows Mobile, Kindle, and all the other devices? Even if we could get our web app to respond like a native application, how do we conquer making it look like one? History and the data presented in the last section show us that we really only care about 3-4 of the leading platforms. So you could have 3 native skins for your target platforms and a default web look-and-feel for all the others.

Overall, the web has its own look-and-feel and everyone knows that. There isn't a default look that will make all your users happy. It's up to you and your design team to create an attractive user experience.

Theresa Neil does a great job of explaining UI Patterns for native apps in Mobile Design Pattern Gallery by O'Reilly Media. The website, shown in Figure 2-5, is a great resource for trending patterns in mobile design.

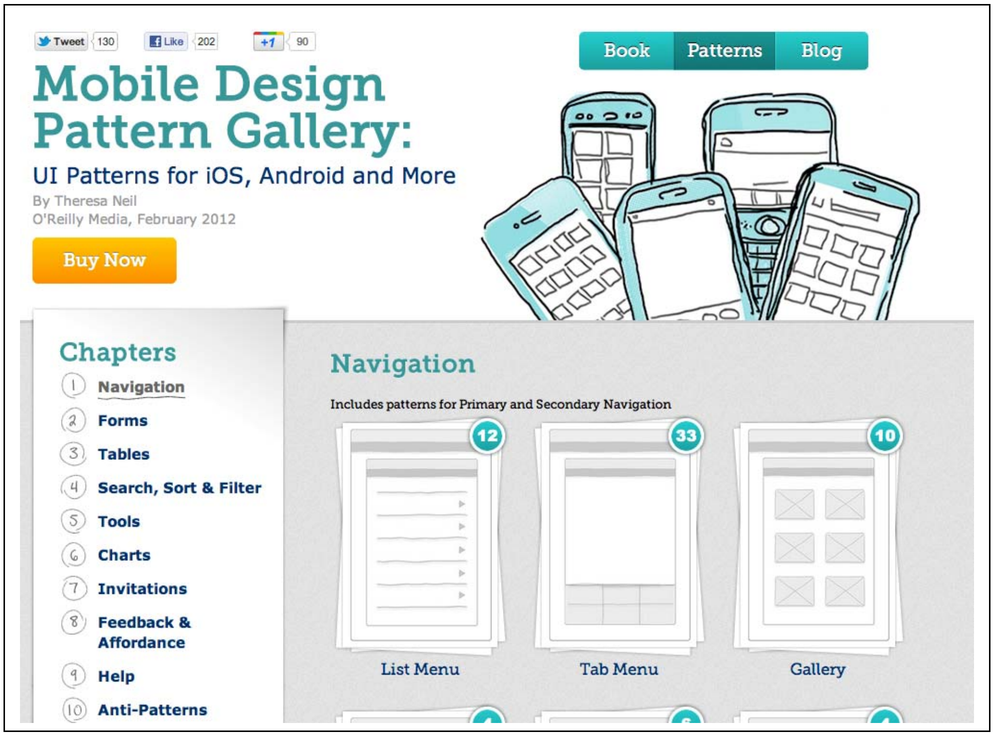


Figure 2-5. http://www.mobiledesignpatterngallery.com/mobile-patterns.php

The Mobile Web Feel

A choppy page transition or spinning refresh is unacceptable in today’s mobile web environment. The HTML driven web app not only needs to look good, but it must also respond well. In the next chapter, we’ll review advanced techniques for handling these scenarios. For now, we know that the overall mobile web experience is lacking in terms of responsive, zippy interfaces.

Frameworks and Approaches

Frameworks

It seems like there’s a new JavaScript based mobile framework popping up every day. You can literally spend days (or months) comparing frameworks and whipping up POCs, only to find out that you may not want or need a framework at all. In the majority of situations, either converting an existing app or starting from scratch, it’s better to start out writing your own CSS and DOM interactions. The harder you lean on a framework, the harder your app will fall when problems arise — knowing the basics and how to fix those problems “under the hood” are essential.

The DOM is the underlying infrastructure and API for all web apps. No matter how much you like or dislike the API, if you desire a mobile web app that screams at blazing fast speeds and gets “close to the metal,” you must understand how to work with it.

One commonly used programming model for the mobile web is called “Single Page”. This means you put your entire markup into a single HTML page, often enclosed by a <div> or some other sensible block element.

Let’s take a look at a single page web app structure:

[html]

```html
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <body>

    <div id="home-page">
      ...page content
    </div>

    <div id="contact-page">
      ...page content
    </div>

  </body>
</html>
```

Why put everything in one page? Primarily, It buys us native like transitions and fewer initial HTTP requests. We must use AJAX and CSS3 transitions to emulate the feel of a native application and load data dynamically. This single page approach also promotes including all your resources, like JavaScript and CSS, within the file. Again, this reduces additional HTTP requests to get the best performance possible from your mobile application.

Now that we have an understanding of the basics, let’s examine a few mobile focused JavaScript frameworks that try to take care of the heavy lifting on the UI. Most of today’s JavaScript frameworks have a specific browser or platform they’re targeting. Some are

WebKit only and others try to span all device browsers. There may be features you need, and ones you don't. So it's up to you to decide when to bring any framework into your current or existing project.

> Some mobile frameworks extend or build on older, bloated desktop browser frameworks. Be careful that whichever framework you choose does not check for older IE6 bugs or platforms you are not targeting. This bloat may seem minimal to some, but as you will see in the next chapter, every byte you can shave off the initial load time will greatly enhance the user experience.

Let's identify what we're looking for in a mobile JavaScript framework:

- Optimized for touch screen devices - This is a given, you want a framework that is using CSS3 transitions to handle animations.
- Cross Platform - We want our app to work consistently across all the major platform, or Grade "A" and "B", browsers.
- Uses (or wraps) the latest HTML5 and CSS3 standards.
- Open Source - Communities behind frameworks (or any project for that matter) are critical.
- Programming Model - Does our project require a dynamically generated UI through JavaScript? Or do we want to declare our markup beforehand (Single Page approach)?

Single Page

As previously mentioned, the single page approach forces us to put as much markup and resources as possible into a single HTML file. In the end, limiting HTTP requests for a better performing app.

Accessible, touch friendly, with built-in theming and native animations

jQuery Mobile

http://jquerymobile.com/

jQuery's Take on mobile interfaces. Strictly tied to the release schedule of the core jQuery library. Known for it's AJAX based navigation system, themeable ThemeRoller designs, and produced by the core jQuery project.

Table 2-3. jQuery Mobile

Platform Support	Android, bada, BlackBerry, iOS, MeeGo, Symbian, webOS, and Windows Phone (others are graded at different levels of support)
License	Dual license MIT or GPL 2
Programming Model	CSS and JavaScript - Declarative on the DOM itself. Markup with CSS and data-* attributes
Wrapped or Polyfilled HTML5 APIs	None

Page Setup:

[html]

```
<!DOCTYPE html>
<html>
    <head>
    <title>My Page</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="/jquery.mobile-1.0.min.css" />
    <script type="text/javascript" src="/jquery-1.6.4.min.js"></script>
    <script type="text/javascript" src=" /jquery.mobile-1.0.min.js"></script>
</head>
<body>

<div data-role="page">

    <div data-role="header">
         <h1>My Title</h1>
    </div><!-- /header -->

    <div data-role="content">
         <p>Hello world</p>
    </div><!-- /content -->

</div><!-- /page -->

</body>
</html>
```

Component Setup (Shown in Figure 2-6):

[html]

```
<ul data-role="listview" data-inset="true" data-filter="true">
    <li><a href="#">Acura</a></li>
    <li><a href="#">Audi</a></li>
    <li><a href="#">BMW</a></li>
    <li><a href="#">Cadillac</a></li>
    <li><a href="#">Ferrari</a></li>
</ul>
```

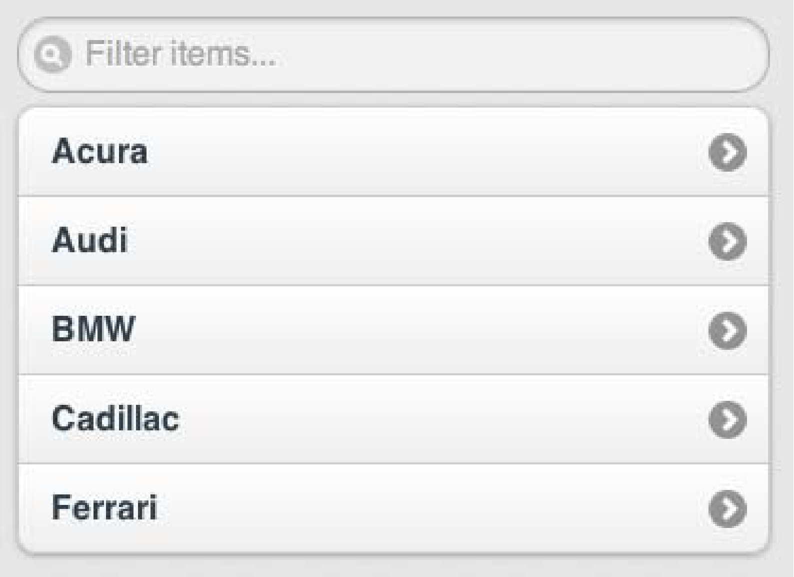


Figure 2-6. jQuery Mobile List View Component

jQTouch

http://jqtouch.com/

A Zepto/jQuery plugin. Good, simple framework to get started with quickly. Documentation is almost non-existent.

Table 2-4. jQTouch

Platform Support	Android and iOS only
License	MIT
Programming Model	Heavy CSS, light JavaScript - It uses CSS classes for detecting the appropriate animations and interactions. Extensions supported.
Wrapped or Polyfilled HTML5 APIs	None

Page Setup:

[html]

```
<html>
    <head>
        <Title>My App</title>
    </head>
    <body>
        <div id="home">
            <div class="toolbar">
                <H1>Hello World</h1>
            </div>
            <ul class="edgetoedge">
                <li class="arrow"><a href="#item1”>Item 1</a></li>
            </ul>
        </div>
    </body>
</html>
```

Component Setup: (Shown in Figure 2-7)

[html]

```
<ul class="edgetoedge">
        <li class="arrow"><a id="0" href="#date">Today</a></li>
        <li class="arrow"><a id="1" href="#date">Yesterday</a></li>
        <li class="arrow"><a id="2" href="#date">2 Days Ago</a></li>
        <li class="arrow"><a id="3" href="#date">3 Days Ago</a></li>
        <li class="arrow"><a id="4" href="#date">4 Days Ago</a></li>
        <li class="arrow"><a id="5" href="#date">5 Days Ago</a></li>
</ul>
```

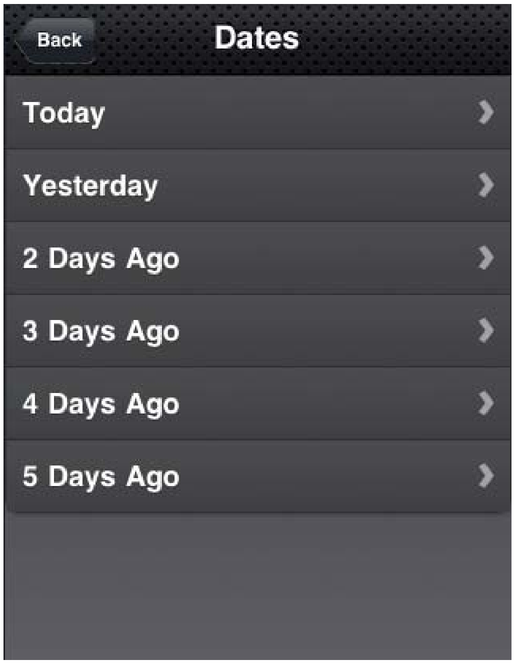


Figure 2-7. jQTouch List View Component

No Page Structure

The markup is flexible, lightweight, and not tied to a specific DOM structure (e.g. Single Page).

xui

http://xuijs.com/

xui (born from the PhoneGap framework) does not try and dictate a page structure or widget paradigm. Xui handles events, animations, transforms, and AJAX. It prides itself in being lightweight with the ability to add plugins for needed features.

Table 2-5. xuijs

Platform Support	WebKit, IE Mobile, BlackBerry
License	MIT
Programming Model	Clean, familiar (JQuery-like), chaining syntax. Plugins support.
Wrapped or Polyfilled HTML5 APIs	None

100% JavaScript Driven (Programmatically created UIs)

Sencha Touch

http://www.sencha.com/products/touch

HTML/CSS3/Javascript framework with a variety of native-style widgets, flexible theming via SASS/Compass, data feature like models, stores, and proxies. Enhanced touch events and a strong data model give Sencha Touch a bit of an enterprise edge without a ton of coverage across devices.

Table 2-6. Sencha Touch

Platform Support	Android, iOS, and BlackBerry (from Sencha 1.1)
License	GPLv3, Limited "Touch Commercial License"
Programming Model	In jQTouch or jQuery Mobile you write specially-structured HTML. When it loads the library reconfigures the page and turns your regular links into Ajax-based animated ones. With Sencha you basically don't write HTML at all, but instead create your UI and app by writing, subclassing, and instantiating JavaScript objects.
Wrapped or Polyfilled HTML5 APIs	Geolocation, WebStorage

Page Setup:

[html]

```
<!DOCTYPE html>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <meta name="viewport" content="width=device-width; initial-scale=1.0; maximum-
scale=1.0; minimum-scale=1.0; user-scalable=0;" />
    <link rel="stylesheet" href="/sencha-touch.css" type="text/css">
    <title>List</title>
    <script type="text/javascript" src="/sencha-touch.js"></script>
</head>
<body></body>
</html>
```

Component Setup and setup of entire app (Shown in Figure 2-8):

[javascript]

```
Ext.setup({
    tabletStartupScreen: 'tablet_startup.png',
    phoneStartupScreen: 'phone_startup.png',
    icon: 'icon.png',
    glossOnIcon: false,
    onReady : function() {
        Ext.regModel('Contact', {
            fields: ['firstName', 'lastName']
        });

        var groupingBase = {
            itemTpl: '<div class="contact2"><strong>{firstName}</strong>
{lastName}</div>',
            selModel: {
                mode: 'SINGLE',
                allowDeselect: true
            },
            grouped: true,
            indexBar: false,

            onItemDisclosure: {
                scope: 'test',
                handler: function(record, btn, index) {
                    alert('Disclose more info for ' + record.get('firstName'));
                }
            },

            store: new Ext.data.Store({
                model: 'Contact',
                sorters: 'firstName',

                getGroupString : function(record) {
                    return record.get('firstName')[0];
                },

                data: [
                    {firstName: ‘Hello’, lastName: ‘World’},
                ]
            })
        };


        if (!Ext.is.Phone) {
            new Ext.List(Ext.apply(groupingBase, {
                floating: true,
                width: 350,
                height: 370,
                centered: true,
                modal: true,
                hideOnMaskTap: false
            })).show();
        }
        else {
            new Ext.List(Ext.apply(groupingBase, {
                fullscreen: true
            }));
        }
        }
})
```

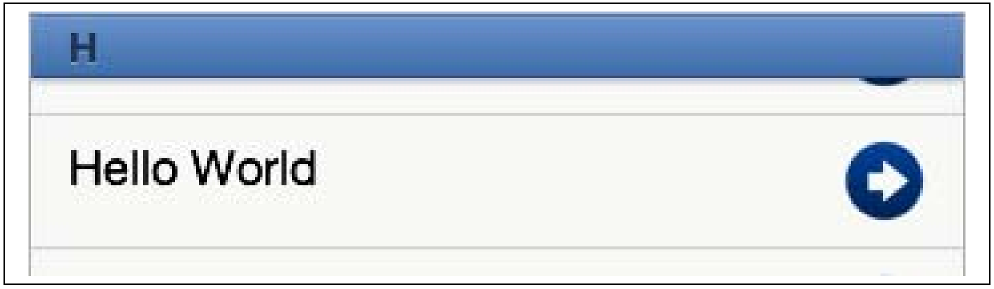


Figure 2-8. Sencha Touch List Component

Wink Toolkit

http://www.winktoolkit.org/

Wink's core offers all the basic functionalities a mobile developer would need from touch event handling to DOM manipulation objects or CSS transforms utilities. Additionally, it offers a wide range of UI components.

Table 2-7. Wink Toolkit

Platform Support	IOS, Android, BlackBerry, and Bada
License	Simplified BSD License
Programming Model	Javascript helpers to add standard mobile browser support. UI is created inside of JavaScript snippets. Project of the Dojo foundation.
Wrapped or Polyfilled HTML5 APIs	Accelerometer, Geolocation, WebStorage

Page Setup:

[html]

```
<html>
    <head>
         <link rel="stylesheet" href="wink.css" type="text/css" >
         <link rel="stylesheet" href="wink.default.css" type="text/css" >
         ...
         <script type="text/javascript" src="wink.min.js"></script>
         ...
    </head>
    <body onload="init()">
    <div class="w_box w_header w_bg_dark">
         <span id="title">accordion</span>
         <input type="button" value="home" class="w_button w_radius w_bg_light
w_right" onclick="window.location='..?theme='+theme"/>
    </div>

    <div class="w_bloc">
         click on the accordion section below to display the content.
    </div>
    <div id="output" style="width: 95%; margin: auto">
    </div>
</body>
</html>
```

Component Setup (Shown in Figure 2-9):

[javascript]

```
var accordion, section1, section2, section3;

init = function()
{
    accordion = new wink.ui.layout.Accordion();

    section1 = accordion.addSection('Section1', 'Hello World');
    section2 = accordion.addSection('section2', '...');
    section3 = accordion.addSection('section3', '...');

    $('output').appendChild(accordion.getDomNode());
}

deleteSection = function()
{
    accordion.deleteSection(section2);
```

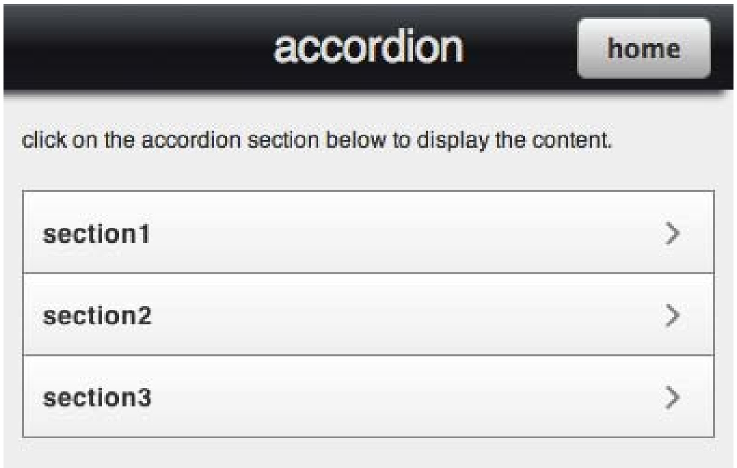

accordion
home
click on the accordion section below to display the content.
section1
section2
section3

Figure 2-9. Wink Accordion Component

The-M-Project

http://the-m-project.net/

The-M-Project is built on top of jQuery and jQuery Mobile. It uses concepts and parts from sproutcore and bases it's persistence handling on persistencejs.

Table 2-8. The-M-Project

Platform Support	IOS, Android, WebOS, BlackBerry
License	GPLv2 and MIT

Programming Model	Relies heavily on MVC pattern. Creates view components through JavaScript and addresses data binding.
Wrapped or Polyfilled HTML5 APIs	WebStorage (DataProvider for local and remote storage persistence)

Page Setup:

[javascript]

```
PageSwitchDemo.Page1 = M.PageView.design({
    childViews: 'header content',
    header: M.ToolbarView.design({
        value: 'Page 1'
    }),
    content: M.ScrollView.design({
        childViews: 'button',
        button: M.ButtonView.design({
            value: 'Goto Page 2',
            events: {
                tap: {
                    target: PageSwitchDemo.ApplicationController,
                    action: 'gotoPage2'
                }
            }
        })
    })

});
```

Component Setup (Shown in Figure 2-10):

[javascript]

```
M.SelectionListView.design({

    childViews: 'item1 item2 item3 item4',
    /* renders a selection view like radio buttons */
    selectionMode: M.SINGLE_SELECTION,

    item1: M.SelectionListItemView.design({
        value: 'item1',
        label: 'Item 1',
        isSelected: YES
    }),
    item2: M.SelectionListItemView.design({
        value: 'item2',
        label: 'Item 2'
    }),
    item3: M.SelectionListItemView.design({
        value: 'item3',
        label: 'Item 3'
    }),
    item4: M.SelectionListItemView.design
        value: 'item4',
        label: 'Item 4'
    })
});
```

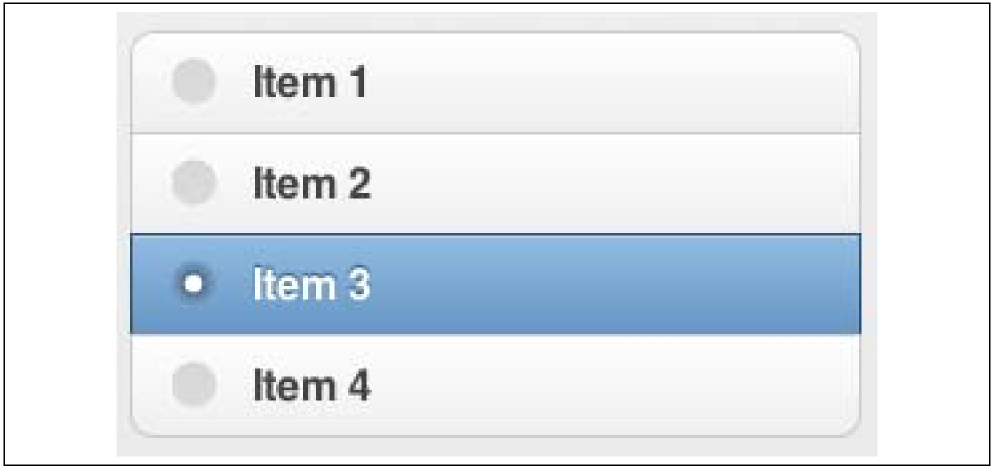


Figure 2-10. The-M-Project List Component

There are many other frameworks like SproutCore, Jo, Zepto, LungoJS and the list goes on. All of these frameworks contain useful features and building blocks for everyday programming of mobile web apps. Some even try to create a wrapper or proxy for spec driven features like WebStorage. But, it seems they all have a gaping hole in terms of the needs of enterprise developers and a consistent architecture across device browsers.

Approaches

How can we create a development environment that will service our enterprise project needs and give us an API that works and degrades gracefully across multiple mobile devices? Instead of focusing on touch interactions and animations, let’s review what it takes to build enterprise style APIs towards HTML5E (WebStorage, WebSockets, Geolocation, and WebWorkers).

> What is Graceful Degradation?
>
> As devices progress, new specifications will become available within the browser for accessing device hardware, leaving today’s devices to work their way out of your support lifecycle.

We’ll look at approaches to handling these issues and identify projects that could possibly provide an open source solution. In the next chapter, we’ll piece them together to form a reusable API for our demo project. But first, let’s identify the frameworks which might help us write a scalable HMTL5E mobile web app.

Web Storage

There are a few javascript frameworks which address Web Storage needs on mobile devices. When evaluating Web Storage frameworks, we’re looking for a nice consistent storage API that works across all devices. Of course, this is what the spec itself does through a simple JavaScript API, but until all devices support this specification, we need a helper framework. We’ll look at other frameworks which focus more on desktop browsers in Chapter 5, but for now, let’s evaluate which ones are focused on mobile.

LawnChair

http://westcoastlogic.com/lawnchair/

Lawnchair is designed with mobile in mind. It’s adaptive to the mobile environments we have identified in this chapter, and gives us a consistent API for accessing some form of localStorage. Extremely compact source and easy to read.

Store and query data on mobile devices without worrying about the underlying API. It’s also agnostic to any server side implementations, so this allows us to get started quickly with a simple, lightweight framework.

Platform Support	All major mobile browsers
License	MIT

Page Setup:

[html]

```
<!DOCTYPE html>
<html>
<head>
    <title>my app</theitle>
</head>
<body>
    <script src="lawnchair.js"></script>
</body>
</html>
```

Persist Data:

[javascript]

```
Lawnchair(function(){
    this.save({msg:'hooray!'})
})
```

persistencejs

http://persistencejs.org/

Asynchronous JavaScript object-relational mapper. Integration with node.js and server side MySQL databases. Recommended for server side use since using in-memory storage seems to slow down filtering and sorting — this means they don’t really recommend using localStorage because of the way their framework works. Download size is much heavier than Lawnchair.

Platform Support	All major mobile browsers
License	MIT

Page Setup:

[html]

```
<!DOCTYPE html>
<html>
<head>
    <title>my app</title>
</head>
```

```
<body>
    <script src="persistence.js" type="application/javascript"></script>
<script src="persistence.store.sql.js" type="application/javascript"></script>
<script src="persistence.store.websql.js" type="application/javascript"></script>
</body>
</html>
```

Persist Data:

[javascript]

```
if (window.openDatabase) {
    persistence.store.websql.config(persistence, "jquerymobile", 'database', 5 *
1024 * 1024);
} else {
    persistence.store.memory.config(persistence);
}

  persistence.define('Order', {
    shipping: "TEXT"
  });

  persistence.schemaSync();
```

Tracking for changes

Similar to JBoss' persistence framework, Hibernate, persistence.js uses a tracking mechanism to determine which object's changes have to be persisted to the database. All objects retrieved from the database are automatically tracked for changes. New entities can be tracked and persisted using the persistence.add function.

[javascript]

```
var c = new Category({name: "Main category"});
persistence.add(c);
```

All changes made to tracked objects can be flushed to the database by using persistence.flush, which takes a transaction object and callback function as arguments. A new transaction can be started using persistence.transaction:

[javascript]

```
persistence.transaction(function(tx) {
  persistence.flush(tx, function() {
    alert('Done flushing!');
  });
});
```

This gives us some of our data syncing needs, but how do we handle a merge of data when the user is disconnected then comes back online. We are dealing with mobile clients after all, who may not have a reliable connection to the internet. The data changes on the server after they go offline and it also changes on their device—who should win? This use case is covered in Chapter 5.

WebSocket

The WebSocket specification defines an API establishing "socket" connections between a web browser and a server. In plain words: There is an open connection between the client and the server and both parties can start sending data at any time.

There are just about as many comet, ajax push based, Web Socket frameworks and servers as there are mobile web frameworks. So it's essential to understand which ones are built for more lightweight mobile environments.

To provide realtime connectivity to every browser, we need a framework that will detect the most capable transport at runtime.

You may already be familiar with projects such as node.js, Ruby EventMachine, or Python Twisted. These projects use an event based API to allow you to create network aware applications in just a few lines of code. But what about enterprise-grade performance and concurrency?

vert.x

https://github.com/purplefox/vert.x

vert.x is a fully asynchronous general purpose application container for JVM languages. It is a framework which takes inspiration from event driven frameworks like node.js, combines it with a distributed event bus and sticks it all on the JVM - a runtime with *real* concurrency and unrivalled performance. Vert.x then exposes the API in Ruby and Java too.

vert.x supports TCP, HTTP, WebSocket, etc, and many more modules. You can think of it as "node.js for JVM languages".

vert.x recommends SockJS to provide a WebSocket-like object on the client. Under the hood SockJS tries to use native WebSockets first. If that fails it can use a variety of browser-specific transport protocols and presents them through WebSocket-like abstractions.

Platform Support	SockJS is intended to work for all modern browsers and in environments that don't support WebSocket protcol, for example behind restrictive corporate proxies. vert.x requires JDK 1.7.0. It uses open source projects such as Netty, JRuby, Mozilla Rhino, and Hazelcast
License	MIT and Apache 2.0

SockJS Page Setup:

[html]

```
<!DOCTYPE html>
<html>
<head>
    <title>my app</title>
</head>
<body>
   <script src="http://cdn.sockjs.org/sockjs-0.1.min.js"></script>
</body>
</html>
```

SockJS Usage:

[javascript]

```
var sock = new SockJS('http://mydomain.com/my_prefix');
   sock.onopen = function() {
       console.log('open');
```

```
    };
    sock.onmessage = function(e) {
        console.log('message', e.data);
    };
    sock.onclose = function() {
        console.log('close');
    };
```

socket.io

http://socket.io/

Specifically built for use with a Node.js server. Has the capability to be used with any backend after setting fallback capabilities by using Flash.

Platform Support	IOS, Android, WebOs WebKit
License	MIT

Page Setup:

[html]

```
<!DOCTYPE html>
<html>
<head>
    <title>my app</title>
</head>
<body>
   <script src="http://cdn.socket.io/stable/socket.io.js"></script>
</body>
</html>
```

Server Setup:

[javascript]

```
var io = require('socket.io').listen(80);

io.sockets.on('connection', function (socket) {
  socket.emit('news', { hello: 'world' });
  socket.on('my other event', function (data) {
    console.log(data);
  });
});
```

Client Setup:

[javascript]

```
var socket = io.connect('http://localhost');
  socket.on('news', function (data) {
    console.log(data);
    socket.emit('my other event', { my: 'data' });
  });
```

Atmosphere

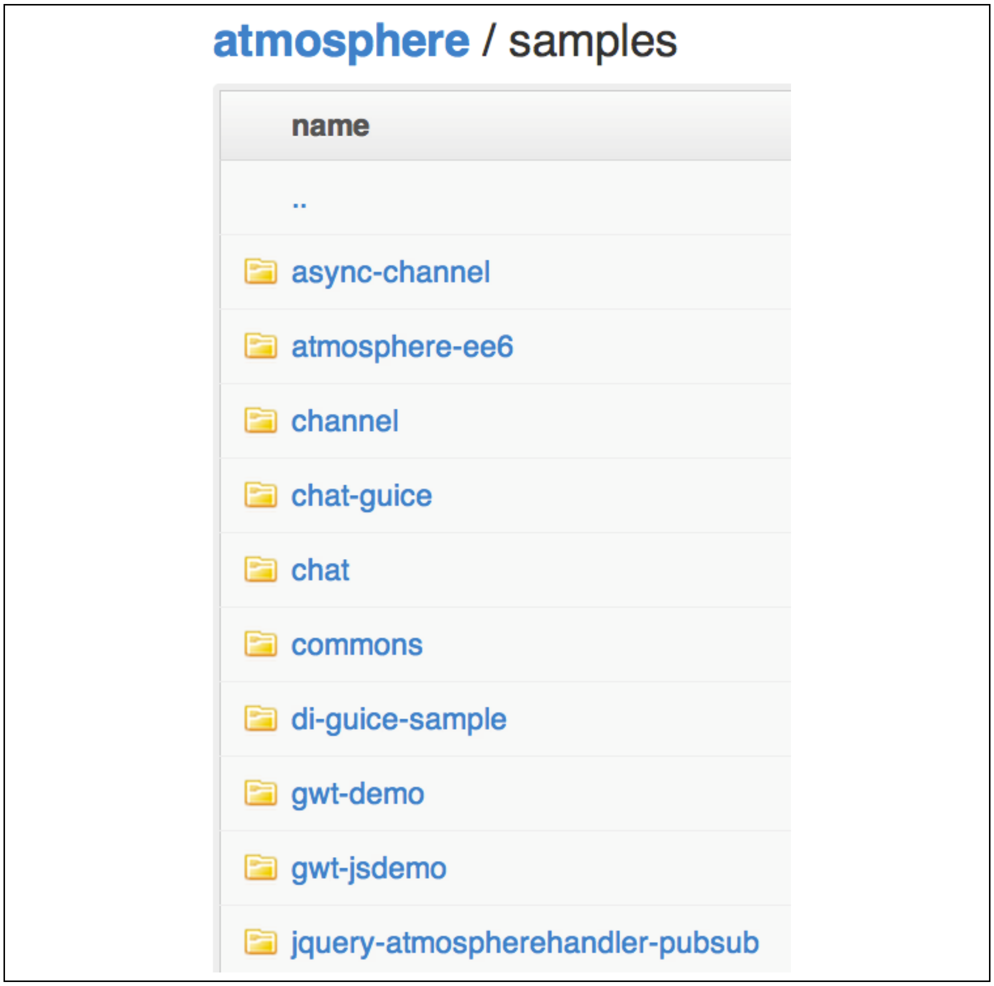
https://github.com/Atmosphere/

The only portable WebSocket/Comet Framework supporting Scala, Groovy and Java. Atmosphere can run on any Java based Web Server, including Tomcat, Jetty, GlassFish, Weblogic, Grizzly, JBoss, Resin, etc. The Atmosphere Framework has both client (Javascript, JQuery, GWT) and server components.

Setup:

There are many different examples on how to use Atmosphere in your project, as seen in Figure 2-11.

https://github.com/Atmosphere/atmosphere/tree/master/samples



atmosphere / samples

name

..

async-channel

atmosphere-ee6

channel

chat-guice

chat

commons

di-guice-sample

gwt-demo

gwt-jsdemo

jquery-atmospherehandler-pubsub

Figure 2-11. A few of many examples listed in Atmosphere's github repo

The main concern when using Web Sockets is graceful degradation, since most mobile browsers and servers have mixed support. All the frameworks mentioned (plus many more) support some kind of fallback when Web Sockets is not available within the browser. However, all of these fallbacks share one problem: they carry the overhead of HTTP, which doesn't make them well suited for low latency mobile applications.

Geolocation

The Geolocation API allows you to locate and track your users after they have given consent. This functionality would be used for something like geo-fencing, where companies target ads for users within a certain city or state. The API itself is device agnostic; it doesn't care how the browser determines location. The underlying mechanism to obtain the users actual location may be through wifi, GPS, or by the user actually entering a zip code into the device.

When working with the Geolocation API, one should detect and wrap available Geolocation mechanisms that are available across different mobile devices. For example, we could detect Google Gears, BlackBerry, and the default Geolocation API within one JavaScript init() method. But why try to code all this yourself, when we could just use a framework? As with all frameworks there may be things you don't want, so we must code to the devices we want to support.

The Geolocation JavaScript frameworks are relatively small in both size and selection.

geo-location-javascript

http://code.google.com/p/geo-location-javascript/

A mobile centric framework using non-standard Blackberry and WebOD tricks. It wraps the underlying platform specific implementation through a simple JavaScriptAPI that is aligned to the W3 Geolocation API Specification.

Platform Support	IOS, Android, Blackberry OS, Browsers with Google Gears support (Android, Windows Mobile), Nokia Web Run-Time (Nokia N97,…), webOS Application Platform (Palm Pre), Torch Mobile Iris Browser, Mozilla Geode
License	MIT

Setup and Usage:

[html]

```
<html>
<head>
        <title>Javascript geo sample</title>
        <script src="http://code.google.com/apis/gears/gears_init.js"
type="text/javascript" charset="utf-8"></script>
        <script src="js/geo.js" type="text/javascript" charset="utf-8"></script>
</head>
<body>
        <b>Javascript geo sample</b>
        <script>
                if(geo_position_js.init()){

geo_position_js.getCurrentPosition(success_callback,error_callback,
{enableHighAccuracy:true});
                }
                else{
                        alert("Functionality not available");
                }

                function success_callback(p)
                {
                        alert('lat='+p.coords.latitude.toFixed(2)+';
                        lon='+p.coords.longitude.toFixed(2));
                }

                function error_callback(p)
                {
                        alert('error='+p.message);
                }
        </script>
        </body>
</html>
```

Webshims lib

http://afarkas.github.com/webshim/demos/

Webshims is a framework, based on jQuery and Modernizr, which tries to handle many different polyfills and/or shims; Gelocation is one of them.

Platform Support	jQuery’s A graded browsers and latest Opera.
License	MIT

Setup and Usage:

[html]

```
<!DOCTYPE html>
<html lang="en">
<head>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js"></script>
    <script src="../js-webshim/minified/extras/modernizr-custom.js"></script>
    <script src="../js-webshim/minified/polyfiller.js"></script>
    <script>
         $.webshims.setOptions('geolocation', {
              confirmText: '{location} wants to know your position. It is Ok to
press Ok.'
         });
         //load all polyfill features
         //or load only a specific feature with $.webshims.polyfill('feature-
name');
         $.webshims.polyfill();
    </script>
```

There are a few other scripts which try to handle this polyfill:

- https://github.com/manuelbieh/Geolocation-API-Polyfill/blob/master/geolocation.js
- http://gist.github.com/366184

Currently, one of the greatest drawbacks to using the Geolocation API within a mobile web browser is not having the ability to run in the background after the browser has closed. For example, it gets extremely difficult to track the user in the background and allow them to switch to another app outside of their mobile browser. At this point, your browser must remain open as a background process for your Geolocation based app to work properly.

WebWorkers

JavaScript is a single-threaded environment, meaning multiple scripts cannot be run in parallel. The Web Workers specification defines an API for spawning background scripts in your web application. Web Workers allow you to do things like fire up long-running scripts to handle computationally intensive tasks, but without blocking the UI or other scripts to handle user interactions.

The only shim currently available for Web Workers makes use of Google Gears. If the core Web WorkersAPI is not supported on the mobile device, you can detect if they have Google Gears installed.

http://html5-shims.googlecode.com/svn/trunk/demo/workers.html

There are many scenarios where Web Workers could be put into action within your app. From preprocessing Wiki text as the user types then generating the HTML to visualizations and business graphs. Here are a few example applications:

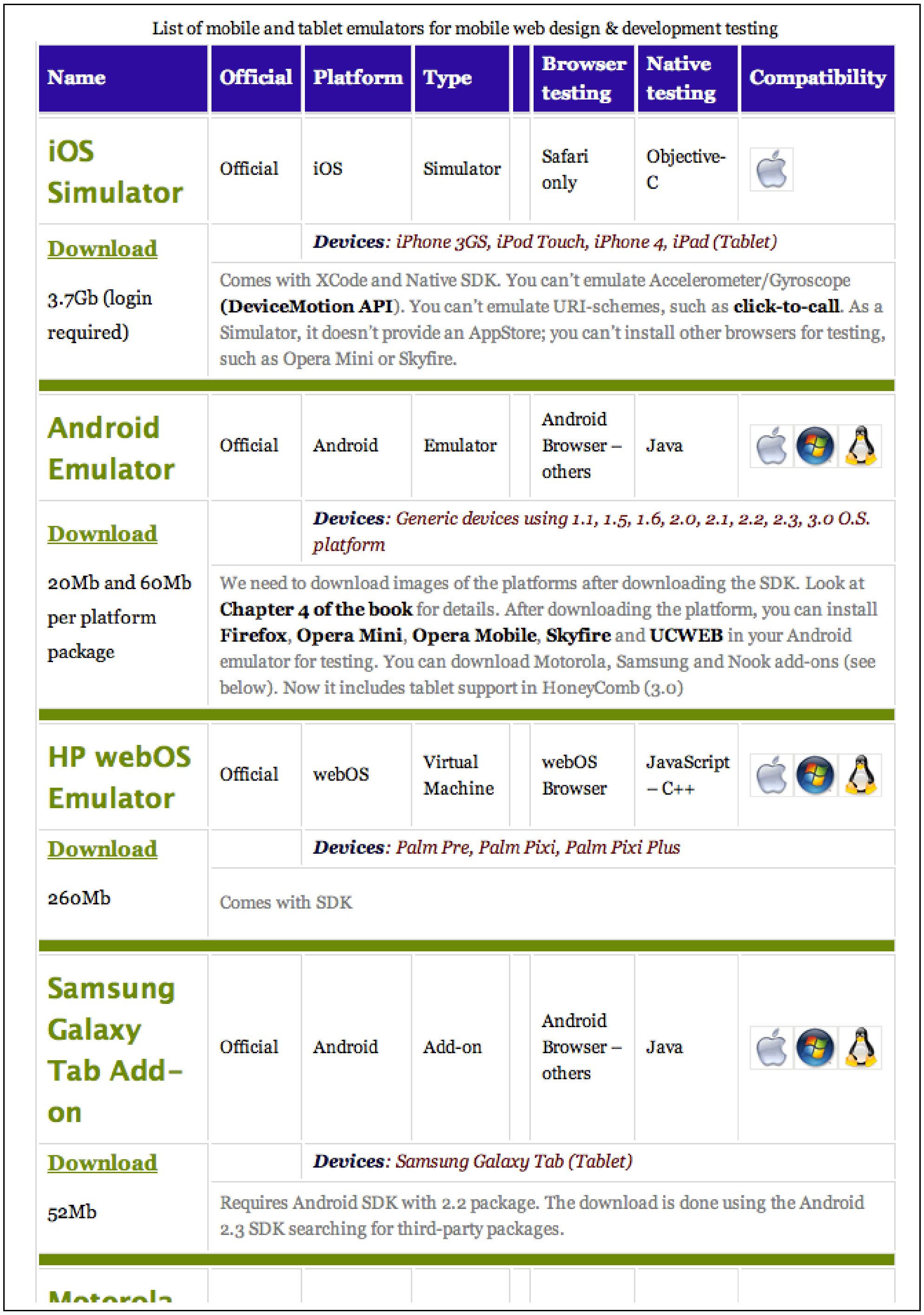
- Parsing Wiki Text
  - http://www.cach.me/blog/2011/01/javascript-web-workers-tutorial-parse-wiki-text-in-real-time/
- Visualization framework
  - https://github.com/samizdatco/arbor

QA and Device Testing

Once you've defined the browsers that should be supported for your new mobile web project, you'll need an easy way to develop and test across them. Enterprise development and QA cycles can get expensive depending on the scale of your project. So setting up the proper rapid development and testing environment is critical to the success of your project.

In the examples throughout this book, device testing will be fairly easy since we only care about "A" grade browsers, which are based on WebKit for our demo. However, this doesn't mean that everything will work properly across all WebKit based mobile browsers just because you tested your app on the desktop version of WebKit, Chrome, or Safari. This also does not mean that WebKit is the "mobile web." You should test across as many target platforms as possible based on W3C standards.

The best way to test your mobile HTML5 based application is to use the actual physical device you are targeting (or an emulator). Max Firtman has already done a great job of identifying available emulators and maintains an up to date list, as shown in Figure 2-12. This list can be found at http://www.mobilexweb.com/emulators.



List of mobile and tablet emulators for mobile web design & development testing

Name	Official	Platform	Type		Browser testing	Native testing	Compatibility
iOS Simulator	Official	iOS	Simulator		Safari only	Objective-C	🍎
Download 3.7Gb (login required)		***Devices:*** *iPhone 3GS, iPod Touch, iPhone 4, iPad (Tablet)*					
	Comes with XCode and Native SDK. You can't emulate Accelerometer/Gyroscope **(DeviceMotion API**). You can't emulate URI-schemes, such as **click-to-call**. As a Simulator, it doesn't provide an AppStore; you can't install other browsers for testing, such as Opera Mini or Skyfire.						
Android Emulator	Official	Android	Emulator		Android Browser – others	Java	🍎 ⊞ 🐧
Download 20Mb and 60Mb per platform package		***Devices:*** *Generic devices using 1.1, 1.5, 1.6, 2.0, 2.1, 2.2, 2.3, 3.0 O.S. platform*					
	We need to download images of the platforms after downloading the SDK. Look at **Chapter 4 of the book** for details. After downloading the platform, you can install **Firefox**, **Opera Mini**, **Opera Mobile**, **Skyfire** and **UCWEB** in your Android emulator for testing. You can download Motorola, Samsung and Nook add-ons (see below). Now it includes tablet support in HoneyComb (3.0)						
HP webOS Emulator	Official	webOS	Virtual Machine		webOS Browser	JavaScript – C++	🍎 ⊞ 🐧
Download 260Mb		***Devices:*** *Palm Pre, Palm Pixi, Palm Pixi Plus*					
	Comes with SDK						
Samsung Galaxy Tab Add-on	Official	Android	Add-on		Android Browser – others	Java	🍎 ⊞ 🐧
Download 52Mb		***Devices:*** *Samsung Galaxy Tab (Tablet)*					
	Requires Android SDK with 2.2 package. The download is done using the Android 2.3 SDK searching for third-party packages.						
Motorola							

Figure 2-12. http://www.mobilexweb.com/emulators

3

Mobile Performance Techniques

Spinning refreshes, choppy page transitions, and periodic delays in tap events are just a few of the headaches in today's mobile web environments. Developers are trying to get as close to native as they possibly can, but are often derailed by hacks, resets, and rigid frameworks.

In this chapter, we'll discuss how to create and tune your mobile web app to be more performant. There are two distinct divisions in the performance of a mobile web application; animations on the UI, and backend services which retrieve and send data via RESTful endpoints. To put it simply, your app is constrained by two ever changing speeds: the speed of the device CPU/GPU and the speed of the internet. The UI is handled by the GPU when doing native like animations and transitions through CSS, and your backend services are limited to the current internet connection speed of the mobile device.

We'll start with a low level explanation of hardware acceleration and how to make your app feel native when the user swipes his finger across the screen or taps on a link. From there, we'll look at the latest HTML5E specifications and use them to make our application much more performant.

Hardware Acceleration

Normally, GPUs handle detailed 3D modeling or CAD diagrams, but in this case, we want our primitive drawings (divs, backgrounds, text with drop shadows, images, etc...) to appear smooth and animate smoothly via the GPU. The unfortunate thing is that most front-end developers are dishing this animation process off to a third-party party framework without being concerned about the semantics, but should these core CSS3 features be masked? Let me give you a few reasons why caring about this stuff is important:

- Memory allocation and computational burden — If you go around compositing every element in the DOM just for the sake of hardware acceleration, the next person who works on your code may chase you down and beat you severely.

- Power Consumption / Battery Life — Obviously, when hardware kicks in, so does the battery. When developing for mobile, developers are forced to take the wide array of device constraints into consideration while writing mobile web apps. This will be even more prevalent as browser makers start to enable access to more and more device hardware. Luckily, we will soon have an API for checking the status of the device battery (http://www.w3.org/TR/2011/WD-battery-status-20111129/).
- Conflicts — You will encounter glitchy behavior when applying hardware acceleration to parts of the page that were already accelerated. So knowing if you have overlapping acceleration is **very** important.
- To make user interaction smooth and as close to native as possible, we must make the browser work for us. Ideally, we want the mobile device CPU to set up the initial animation, then have the GPU responsible for only compositing different layers during the animation process. This is what translate3d, scale3d and translateZ do — they give the animated elements their own layer, thus allowing the device to render everything together smoothly.

> Once downloaded, static images render faster than CSS effects and those effects can come at a cost on low-end devices. With great power comes great responsibility, and knowing when you should use an image instead of a CSS gradient will only improve the UI performance of your app. Hardware can easily handle and paint images onto the screen, but CSS requires software to calculate, render, and paint the image. Every situation is different, so it’s up to you to decide which route to take for your mobile web app.

Page Transitions

Let’s take a look at three of the most common user-interaction approaches when developing a mobile web app: slide, flip, and rotation effects.

You can view this code in action here http://html5e.org/slide-flip-rotate.html.

> This demo is built for a mobile device, so fire up an emulator, use your phone or tablet, or reduce the size of your browser window to 1024px or less.

Or, you can download the code here ([need to set this repository up]) and run it on your machine

First, we’ll dissect the slide, flip, and rotation transitions and how they’re accelerated. Notice how each animation only takes three or four lines of CSS and JavaScript. We’re not using any additional frameworks, only DOM and vendor prefixed APIs.

Sliding

The most common of the three transition approaches, sliding page transitions mimics the native feel of mobile applications. The slide transition is invoked to bring a new content area into the view port.

For the slide effect, first we declare our markup:

[html]

```
<div id="home-page" class="page">
```

```
  <h1>Home Page</h1>
</div>

<div id="products-page" class="page stage-right">
  <h1>Products Page</h1>
</div>

<div id="about-page" class="page stage-left">
  <h1>About Page</h1>
</div>
```

Notice how we have this concept of staging pages left or right. It could essentially be any direction, but this is most common.

We now have animation plus hardware acceleration with just a few lines of CSS. The actual animation happens when we swap classes on the page div elements.

[css]

```
.page {
  position: absolute;
  width: 100%;
  height: 100%;
  /*activate the GPU for compositing each page */
  -webkit-transform: translate3d(0, 0, 0);
}
```

translate3d(0,0,0) is known as the "silver bullet" approach.

> Hardware acceleration tricks do not provide any speed improvement under Android Froyo 2.2+. All composition is done within the software.

When the user clicks a navigation element, we execute the following JavaScript to swap the classes. No third-party frameworks are being used, this is straight up JavaScript!

[javascript]

```
function getElement(id) {
  if (document.querySelector){
      return document.querySelector('#' + id);
  }else{
      return document.getElementById(id);
  }
}

function slideTo(id) {
  //1.) the page we are bringing into focus dictates how
  // the current page will exit. So let's see what classes
  // our incoming page is using. We know it will have stage[right|left|etc...]
  var classes = getElement(id).className.split(' ');

  //2.) decide if the incoming page is assigned to right or left
  // (-1 if no match)
  var stageType = classes.indexOf('stage-left');

  //3.) on initial page load focusPage is null, so we need
  // to set the default page which we're currently seeing.
  if (FOCUS_PAGE == null) {
    // use home page
    FOCUS_PAGE = getElement('home-page');
  }

  //4.) decide how this focused page should exit.
  if (stageType > 0) {
    FOCUS_PAGE.className = 'page transition stage-right';
  } else {
    FOCUS_PAGE.className = 'page transition stage-left';
  }

  //5. refresh/set the global variable
  FOCUS_PAGE = getElement(id);

  //6. Bring in the new page.
  FOCUS_PAGE.className = 'page transition stage-center';
}
```

Stage-left or stage-right becomes stage-center and forces the page to slide into the center view port. We are completely depending on CSS3 to do the heavy lifting.

[css]

```
.stage-left {
  left: -100%;
}

.stage-right {
  left: 100%;
}

.stage-center {
  top: 0;
  left: 0;
}
```

Flipping

On mobile devices, flipping is known as actually swiping the page away. Here we use some simple JavaScript to handle this event on iOS and Android (WebKit-based) devices.

View it in action at http://html5e.org/slide-flip-rotate.html

When dealing with touch events and transitions, the first thing you'll want to get a handle on the current position of the element. See this doc for more information on WebKitCSSMatrix (http://developer.apple.com/library/safari/#documentation/AudioVideo/Reference/WebKitCSSMatrixClassReference/WebKitCSSMatrix/WebKitCSSMatrix.html).

[javascript]

```
function pageMove(event) {
  // get position after transform
  var curTransform = new
WebKitCSSMatrix(window.getComputedStyle(page).webkitTransform);
  var pagePosition = curTransform.m41;
}
```

Since we are using a CSS3 ease-out transition for the page flip, the usual element.offsetLeft will not work.

Next we want to figure out which direction the user is flipping and set a threshold for an event (page navigation) to take place.

[javascript]

```
if (pagePosition >= 0) {
 //moving current page to the right
 //so means we're flipping backwards
   if ((pagePosition > pageFlipThreshold) || (swipeTime < swipeThreshold)) {
     //user wants to go backward
     slideDirection = 'right';
   } else {
     slideDirection = null;
   }
} else {
  //current page is sliding to the left
  if ((swipeTime < swipeThreshold) || (pagePosition < pageFlipThreshold)) {
    //user wants to go forward
    slideDirection = 'left';
  } else {
    slideDirection = null;
  }
}
```

You'll also notice that we are measuring the swipeTime in milliseconds as well. This allows the navigation event to fire if the user quickly swipes the screen to turn a page.

To position the page and make the animations look native while a finger is touching the screen, we use CSS3 transitions after each event firing.

[javascript]

```
function positionPage(end) {
  page.style.webkitTransform = 'translate3d('+ currentPos + 'px, 0, 0)';
  if (end) {
    page.style.WebkitTransition = 'all .4s ease-out';
    //page.style.WebkitTransition = 'all .4s cubic-bezier(0,.58,.58,1)'
  } else {
    page.style.WebkitTransition = 'all .2s ease-out';
  }
```

I encourage you to play around with cubic-bezier to give the best native feel to the transitions, but ease-out did the trick for this demo.

Finally, to make the navigation happen, we must call our previously defined slideTo() methods that we used in the last example.

[javascript]

```
track.ontouchend = function(event) {
  pageMove(event);
  if (slideDirection == 'left') {
    slideTo('products-page');
  } else if (slideDirection == 'right') {
    slideTo('home-page');
  }
}
```

Rotating

Next, let's take a look at the rotate animation being used in this demo. At any time, you can rotate the page you're currently viewing 180 degrees to reveal the reverse side by tapping on the "Contact" menu option. Again, this only takes a few lines of CSS and some JavaScript to assign a transition class onclick.

> The rotate transition isn't rendered correctly on most versions of Android because it lacks 3D CSS transform capabilities. Unfortunately, instead of ignoring the flip, Android makes the page "cartwheel" away by rotating instead of flipping. I recommend using this transition sparingly until support improves.

The markup (basic concept of front and back):

[html]

```html
<div id="front" class="normal">
...
</div>
<div id="back" class="flipped">
    <div id="contact-page" class="page">
        <h1>Contact Page</h1>
    </div>
</div>
```

The JavaScript:

[javascript]

```javascript
function flip(id) {
  // get a handle on the flippable region
  var front = getElement('front');
  var back = getElement('back');

  // again, just a simple way to see what the state is
  var classes = front.className.split(' ');
  var flipped = classes.indexOf('flipped');

  if (flipped >= 0) {
    // already flipped, so return to original
    front.className = 'normal';
    back.className = 'flipped';
    FLIPPED = false;
  } else {
    // do the flip
    front.className = 'flipped';
    back.className = 'normal';
    FLIPPED = true;
  }
}
```

The CSS:

[css]

```css
#back,
#front {
  position: absolute;
  width: 100%;
  height: 100%;
  -webkit-backface-visibility: hidden;
  -webkit-transition-duration: .5s;
  -webkit-transform-style: preserve-3d;
}

.normal {
  -webkit-transform: rotateY(0deg);
}

.flipped {
  -webkit-user-select: element;
  -webkit-transform: rotateY(180deg);
}
```

Debugging Hardware Acceleration

Now that we have our basic transitions covered, let’s take a look at the mechanics of how they work and are composited. Here are a few tips to remember when using accelerated compositing:

- Reduce the quantity of layers
- Keep layers as small as possible
- Update layers infrequently
- Tailor layer compositing to purpose
- Trial and error; testing is important

To make this magical debugging session happen, let’s fire up a couple of browsers and your IDE of choice. First start Safari from the command line to make use of some debugging environment variables. I’m on Mac, so the commands might differ based on your OS. Open the Terminal and type the following:

```
$> export CA_COLOR_OPAQUE=1
$> export CA_LOG_MEMORY_USAGE=1
$> /Applications/Safari.app/Contents/MacOS/Safari
```

This starts Safari with a couple of debugging helpers. CA_COLOR_OPAQUE shows us which elements are actually composited or accelerated. CA_LOG_MEMORY_USAGE shows us how much memory we are using when sending our drawing operations to the backing store. This tells you exactly how much strain you are putting on the mobile device, and possibly give hints to how your GPU usage might be draining the target device’s battery.

You may also start Safari after running the following command. This gives us a full Debug menu with all available options as shown in Figure 3-13:

```
defaults write com.apple.Safari IncludeInternalDebugMenu 1
```

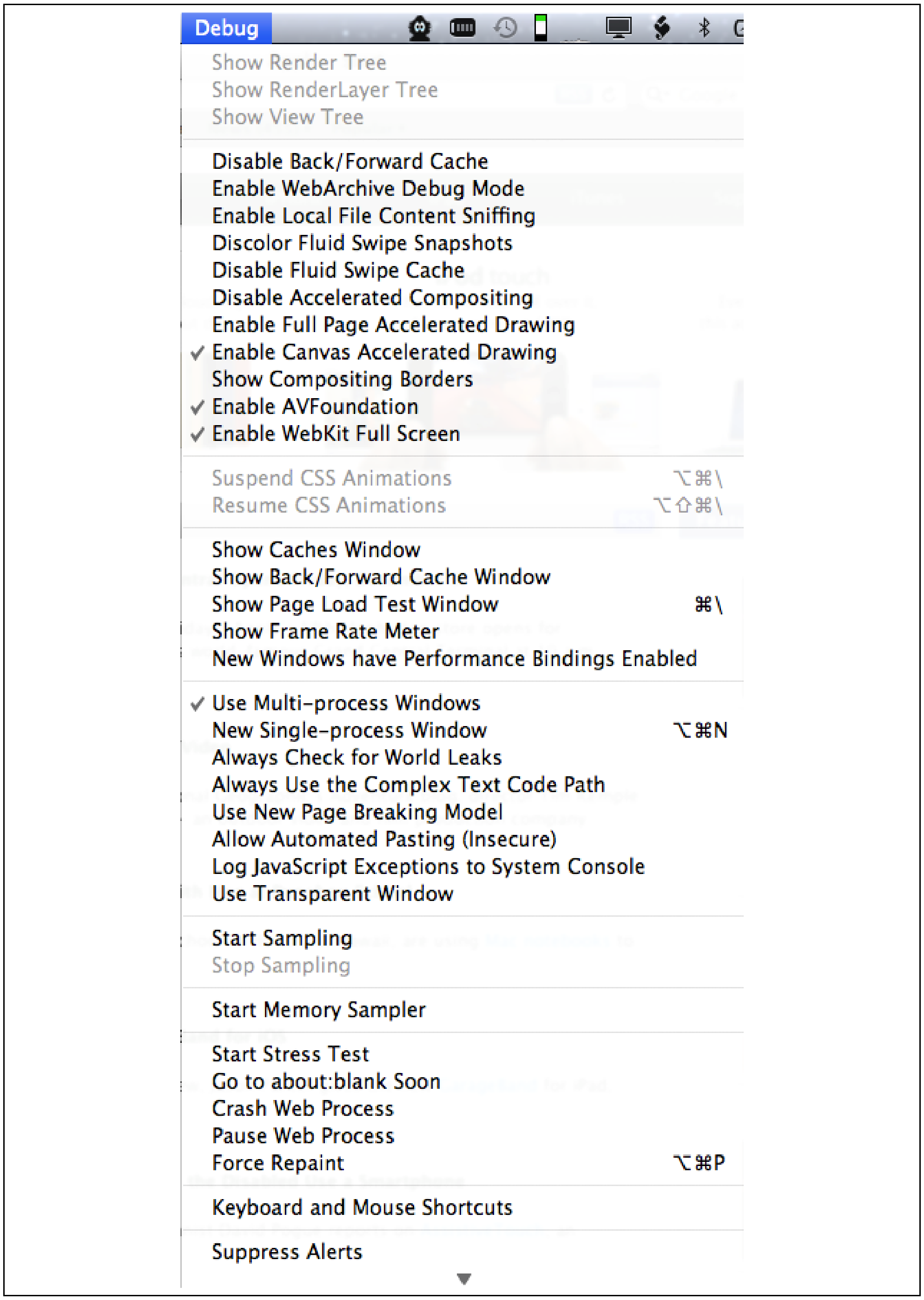

Debug
Show Render Tree
Show RenderLayer Tree
Show View Tree
Disable Back/Forward Cache
Enable WebArchive Debug Mode
Enable Local File Content Sniffing
Discolor Fluid Swipe Snapshots
Disable Fluid Swipe Cache
Disable Accelerated Compositing
Enable Full Page Accelerated Drawing
✓ Enable Canvas Accelerated Drawing
Show Compositing Borders
✓ Enable AVFoundation
✓ Enable WebKit Full Screen
Suspend CSS Animations ⌥⌘\
Resume CSS Animations ⌥⇧⌘\
Show Caches Window
Show Back/Forward Cache Window
Show Page Load Test Window ⌘\
Show Frame Rate Meter
New Windows have Performance Bindings Enabled
✓ Use Multi-process Windows
New Single-process Window ⌥⌘N
Always Check for World Leaks
Always Use the Complex Text Code Path
Use New Page Breaking Model
Allow Automated Pasting (Insecure)
Log JavaScript Exceptions to System Console
Use Transparent Window
Start Sampling
Stop Sampling
Start Memory Sampler
Start Stress Test
Go to about:blank Soon
Crash Web Process
Pause Web Process
Force Repaint ⌥⌘P
Keyboard and Mouse Shortcuts
Suppress Alerts

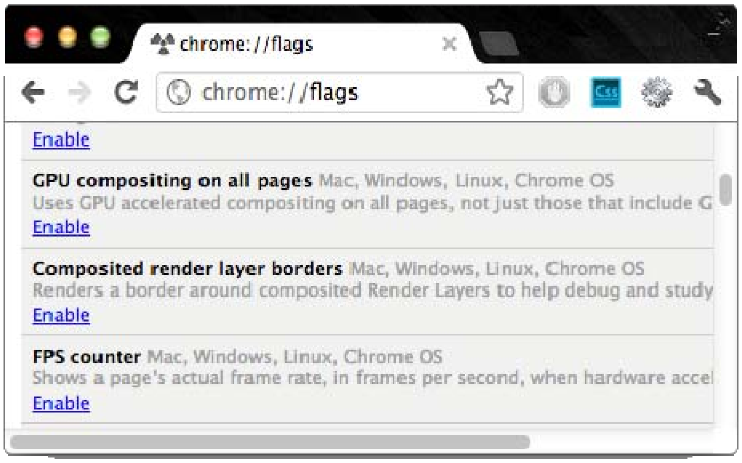
Figure 3-13. Safari Debug Menu

Now let's fire up Chrome so we can see some good frames per second (FPS) information:

Open the Google Chrome web browser.

In the URL bar, type about:flags.

Scroll down a few items and click on "Enable" for FPS Counter as shown in Figure 3-14.



Enable

GPU compositing on all pages Mac, Windows, Linux, Chrome OS
Uses GPU accelerated compositing on all pages, not just those that include G
Enable

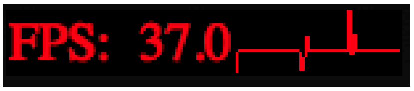
Composited render layer borders Mac, Windows, Linux, Chrome OS
Renders a border around composited Render Layers to help debug and study
Enable

FPS counter Mac, Windows, Linux, Chrome OS
Shows a page's actual frame rate, in frames per second, when hardware accel
Enable

Figure 3-14. Chrome about:flags

Do not enable the GPU compositing on all pages option. The FPS counter only appears in the left-hand corner if the browser detects compositing in your markup—and that is what we want in this case.

If you view this page in your souped up version of Chrome, you will see the red FPS counter in the top left hand corner, as shown in Figure 3-15.



FPS: 37.0

Figure 3-15. Chrome FPS meter

This is how we know hardware acceleration is turned on. It also gives us an idea on how the animation runs and if you have any leaks (continuous running animations that should be stopped).

Another way to actually visualize the hardware acceleration is if you open the same page in Safari (with the environment variables I mentioned above). Every accelerated DOM element have a red tint to it. This shows us exactly what is being composited by layer. Notice in Figure 3-16, the white navigation is not red because it is not accelerated.

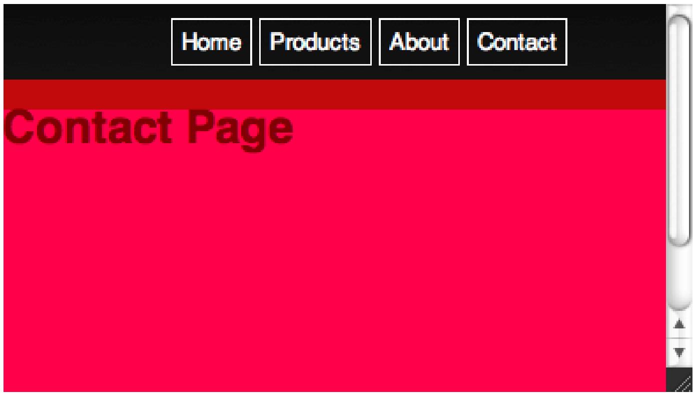


Figure 3-16. Demo App Debug Acceleration

A similar setting for Chrome is also available in the about:flags "Composited render layer borders".

Another great way to see the composited layers is to view the WebKit falling leaves demo (http://www.webkit.org/blog-files/leaves/) while this mod is applied. Shown in Figure 3-17.

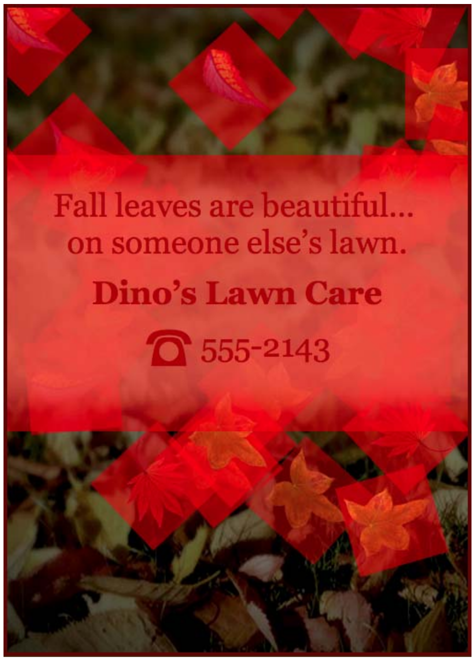
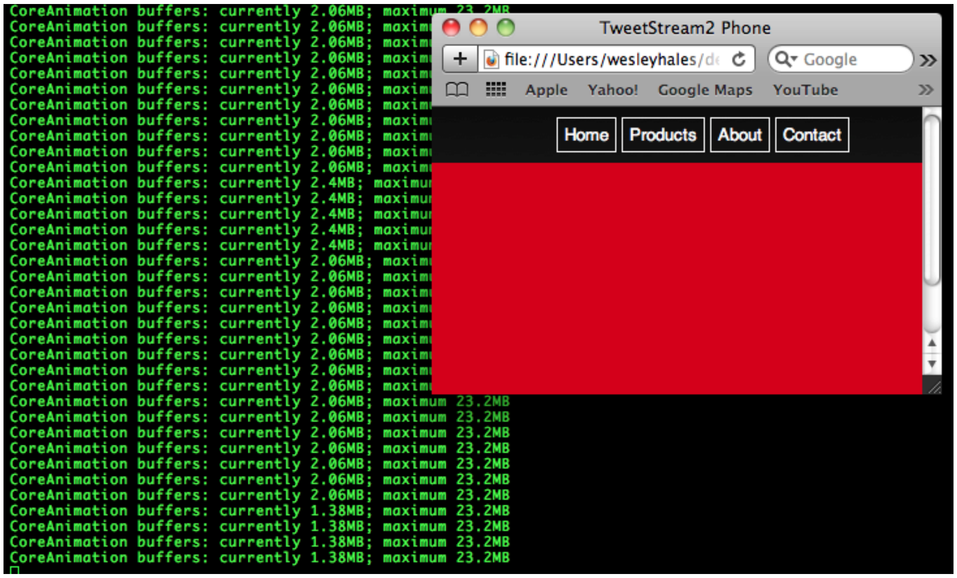


Figure 3-17. WebKit Falling Leaves Demo

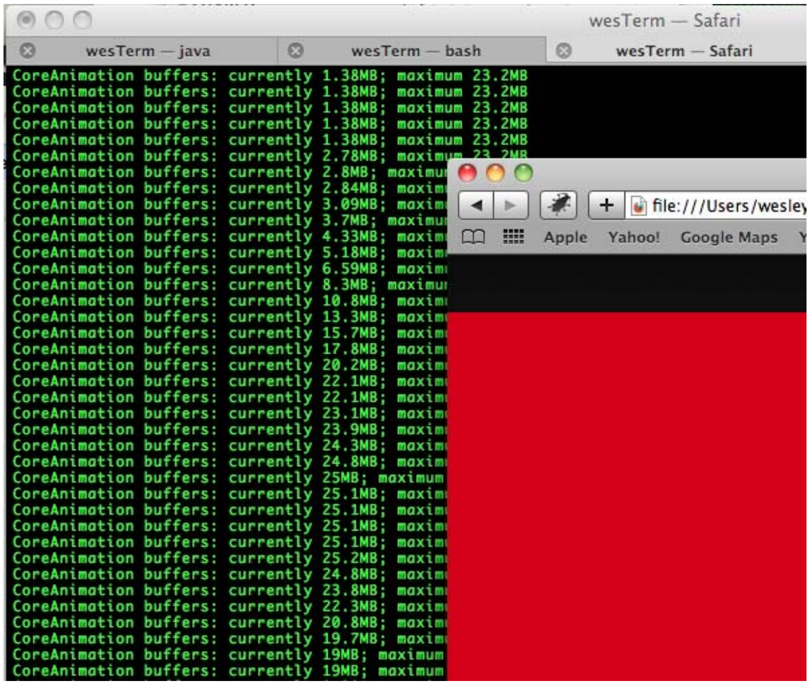
And finally, to truly understand the graphics hardware performance of our application, let's take a look at how memory is being consumed. Here we see that we are pushing 1.38MB of drawing instructions to the CoreAnimation buffers on Mac OS. The Core Animation memory buffers are shared between OpenGL ES and the GPU to create the final pixels you see on the screen in Figure 3-18.



```
CoreAnimation buffers: currently 2.06MB; maximum 23.2MB
CoreAnimation buffers: currently 2.06MB; maxim
CoreAnimation buffers: currently 2.06MB; maxim
CoreAnimation buffers: currently 2.06MB; maxim
CoreAnimation buffers: currently 2.06MB; maxim
CoreAnimation buffers: currently 2.06MB; maxim
CoreAnimation buffers: currently 2.06MB; maxim
CoreAnimation buffers: currently 2.06MB; maxim
CoreAnimation buffers: currently 2.06MB; maxim
CoreAnimation buffers: currently 2.06MB; maxim
CoreAnimation buffers: currently 2.06MB; maxim
CoreAnimation buffers: currently 2.4MB; maximu
CoreAnimation buffers: currently 2.4MB; maximu
CoreAnimation buffers: currently 2.4MB; maximu
CoreAnimation buffers: currently 2.4MB; maximu
CoreAnimation buffers: currently 2.4MB; maximu
CoreAnimation buffers: currently 2.06MB; maxim
CoreAnimation buffers: currently 2.06MB; maxim
CoreAnimation buffers: currently 2.06MB; maxim
CoreAnimation buffers: currently 2.06MB; maxim
CoreAnimation buffers: currently 2.06MB; maxim
CoreAnimation buffers: currently 2.06MB; maxim
CoreAnimation buffers: currently 2.06MB; maxim
CoreAnimation buffers: currently 2.06MB; maxim
CoreAnimation buffers: currently 2.06MB; maxim
CoreAnimation buffers: currently 2.06MB; maximum 23.2MB
CoreAnimation buffers: currently 2.06MB; maximum 23.2MB
CoreAnimation buffers: currently 2.06MB; maximum 23.2MB
CoreAnimation buffers: currently 2.06MB; maximum 23.2MB
CoreAnimation buffers: currently 2.06MB; maximum 23.2MB
CoreAnimation buffers: currently 2.06MB; maximum 23.2MB
CoreAnimation buffers: currently 2.06MB; maximum 23.2MB
CoreAnimation buffers: currently 1.38MB; maximum 23.2MB
CoreAnimation buffers: currently 1.38MB; maximum 23.2MB
CoreAnimation buffers: currently 1.38MB; maximum 23.2MB
CoreAnimation buffers: currently 1.38MB; maximum 23.2MB
```

Figure 3-18. CoreAnimation Debug Session

When we simply resize or maximize the browser window, we see the memory expand as well in Figure 3-19.



```
CoreAnimation buffers: currently 1.38MB; maximum 23.2MB
CoreAnimation buffers: currently 1.38MB; maximum 23.2MB
CoreAnimation buffers: currently 1.38MB; maximum 23.2MB
CoreAnimation buffers: currently 1.38MB; maximum 23.2MB
CoreAnimation buffers: currently 1.38MB; maximum 23.2MB
CoreAnimation buffers: currently 2.78MB; maximum 23.2MB
CoreAnimation buffers: currently 2.8MB; maximu
CoreAnimation buffers: currently 2.84MB; maxim
CoreAnimation buffers: currently 3.09MB; maxim
CoreAnimation buffers: currently 3.7MB; maximu
CoreAnimation buffers: currently 4.33MB; maxim
CoreAnimation buffers: currently 5.18MB; maxim
CoreAnimation buffers: currently 6.59MB; maxim
CoreAnimation buffers: currently 8.3MB; maximu
CoreAnimation buffers: currently 10.8MB; maxim
CoreAnimation buffers: currently 13.3MB; maxim
CoreAnimation buffers: currently 15.7MB; maxim
CoreAnimation buffers: currently 17.8MB; maxim
CoreAnimation buffers: currently 20.2MB; maxim
CoreAnimation buffers: currently 22.1MB; maxim
CoreAnimation buffers: currently 22.1MB; maxim
CoreAnimation buffers: currently 23.1MB; maxim
CoreAnimation buffers: currently 23.9MB; maxim
CoreAnimation buffers: currently 24.3MB; maxim
CoreAnimation buffers: currently 24.8MB; maxim
CoreAnimation buffers: currently 25MB; maximum
CoreAnimation buffers: currently 25.1MB; maxim
CoreAnimation buffers: currently 25.1MB; maxim
CoreAnimation buffers: currently 25.1MB; maxim
CoreAnimation buffers: currently 25.1MB; maxim
CoreAnimation buffers: currently 25.2MB; maxim
CoreAnimation buffers: currently 24.8MB; maxim
CoreAnimation buffers: currently 23.8MB; maxim
CoreAnimation buffers: currently 22.3MB; maxim
CoreAnimation buffers: currently 20.8MB; maxim
CoreAnimation buffers: currently 19.7MB; maxim
CoreAnimation buffers: currently 19MB; maximum
CoreAnimation buffers: currently 19MB; maximum
```

Figure 3-19. CoreAnimation Debug Session

This gives you an idea of how memory is being consumed on your mobile device only if you resize the browser to the correct dimensions. If you were debugging or testing for iPhone environments resize to 480px by 320px. We now understand exactly how hardware acceleration works and what it takes to debug. It’s one thing to read about it, but to actually see the GPU memory buffers working visually really brings things into perspective.

Fetching and Caching

Now it’s time to take our page and resource caching to the next level. Much like the approach that jQuery Mobile and similar frameworks use, we are going to pre-fetch and cache our pages with concurrent AJAX calls.

Let’s address a few core mobile web problems and the reasons why we need to do this:

- Fetching: Pre-fetching our pages allows users to take the app offline and also enables no waiting between navigation actions. Of course, we don’t want to choke the device’s bandwidth when the device comes online, so we need to use this feature sparingly.
- Caching: Next, we want a concurrent or asynchronous approach when fetching and caching these pages. We also need to use localStorage (since it’s well supported amongst devices), which unfortunately isn’t asynchronous.
- AJAX and parsing the response: Using innerHTML() to insert the AJAX response into the DOM is dangerous (and unreliable? http://martinkou.blogspot.com/2011/05/alternative-workaround-for-mobile.html). We instead use a reliable mechanism (http://community.jboss.org/people/wesleyhales/blog/2011/08/28/fixing-ajax-on-mobile-devices) for AJAX response insertion and handling concurrent calls. We also leverage some new features of HTML5 for parsing the xhr.responseText.

Building on the code from the Slide, Flip, and Rotate demos, we start out by adding some secondary pages and linking to them. We’ll then parse the links and create transitions on the fly.

[html]

```
<div id="home-page" class="page">
  <h1>Home Page</h1>
  <a href="demo2/home-detail.html" class="fetch">Find out more about the home
page!</a>
</div>
```

As you can see, we are leveraging semantic markup here. Just a link to another page. The child page follows the same node/class structure as its parent. We could take this a step further and use the data-* attribute for “page” nodes, etc. And here is the detail page (child) located in a separate html file (/demo2/home-detail.html), which will be loaded, cached, and set up for transition on app load.

[html]

```
<div id="home-page-detail" class="page">
   <h1>Home Page Details</h1>
   <p>Here are the details.</p>
</div>
```

Now lets take a look at the JavaScript. For simplicity sake, I'm leaving any helpers or optimizations out of the code. All we are doing here is looping through a specified array of DOM nodes to dig out links to fetch and cache.

[javascript]

```
var fetchAndCache = function() {
  // iterate through all nodes in this DOM to find all mobile pages we care about
  var pages = document.querySelectorAll('.page');

  for (var i = 0; i < pages.length; i++) {
    // find all links
    var pageLinks = pages[i].getElementsByTagName('a');

    for (var j = 0; j < pageLinks.length; j++) {
      var link = pageLinks[j];

      if (link.hasAttribute('href') &&
      //'#' in the href tells us that this page is already loaded in the DOM - and
      // that it links to a mobile transition/page
         !(/[\#]/g).test(link.href) &&
        //check for an explicit class name setting to fetch this link
        (link.className.indexOf('fetch') >= 0))  {
         //fetch each url concurrently
         var ai = new ajax(link,function(text,url){
              //insert the new mobile page into the DOM
             insertPages(text,url);
         });
         ai.doGet();
      }
    }
  }
};
```

We ensure proper asynchronous post-processing through the use of the "AJAX" object. In this example, you see the basic usage of caching on each request and providing the cached objects when the server returns anything but a successful (200) response.

[javascript]

```
function processRequest () {
  if (req.readyState == 4) {
    if (req.status == 200) {
      if (supports_local_storage()) {
        localStorage[url] = req.responseText;
      }
      if (callback) callback(req.responseText,url);
    } else {
      // There is an error of some kind, use our cached copy (if available).
      if (!!localStorage[url]) {
        // We have some data cached, return that to the callback.
        callback(localStorage[url],url);
        return;
      }
    }
  }
}
```

Unfortunately, since localStorage uses UTF-16 for character encoding, each single byte is stored as 2 bytes bringing our storage limit from 5MB to 2.6MB total. The whole reason for fetching and caching these pages/markup outside of the application cache scope is revealed in the next section.

With the recent advances in the iframe element with HTML5, we now have a simple and effective way to parse the responseText we get back from our AJAX call. There are plenty of 3000-line JavaScript parsers and regular expressions that remove script tags and so on. But why not let the browser do what it does best? In this example, we are going to write the responseText into a temporary hidden iframe. We are using the HTML5 "sandbox" attribute, which disables scripts and offers many security features.

> From the spec: The sandbox attribute, when specified, enables a set of extra restrictions on any content hosted by the iframe. Its value must be an unordered set of unique space-separated tokens that are ASCII case-insensitive. The allowed values are allow-forms, allow-same-origin, allow-scripts, and allow-top-navigation. When the attribute is set, the content is treated as being from a unique origin, forms and scripts are disabled, links are prevented from targeting other browsing contexts, and plugins are disabled. To limit the damage that can be caused by hostile HTML content, it should be served using the text/html-sandboxed MIME type.

[javascript]

```
var getFrame = function() {
    var frame = document.getElementById("temp-frame");

    if (!frame) {
        // create frame
        frame = document.createElement("iframe");
        frame.setAttribute("id", "temp-frame");
        frame.setAttribute("name", "temp-frame");
        frame.setAttribute("seamless", "");
        frame.setAttribute("sandbox", "");
        frame.style.display = 'none';
        document.documentElement.appendChild(frame);
    }
    // load a page
    return frame.contentDocument;
};

var insertPages = function(text, originalLink) {
  var frame = getFrame();
  //write the ajax response text to the frame and let
  //the browser do the work
  frame.write(text);

  //now we have a DOM to work with
  var incomingPages = frame.getElementsByClassName('page');

  var pageCount = incomingPages.length;
  for (var i = 0; i < pageCount; i++) {
    //the new page will always be at index 0 because
    //the last one just got popped off the stack with appendChild (below)
    var newPage = incomingPages[0];
    //stage the new pages to the left by default
    newPage.className = 'page stage-left';

    //find out where to insert
    var location = newPage.parentNode.id == 'back' ? 'back' : 'front';

    try {
      // mobile safari will not allow nodes to be transferred from one DOM to
another so
      // we must use adoptNode()
      document.getElementById(location).appendChild(document.adoptNode(newPage));
    } catch(e) {
      // todo graceful degradation?
    }
  }
};
```

Safari correctly refuses to implicitly move a node from one document to another. An error is raised if the new child node was created in a different document. So here we use adoptNode and all is well.

So why iframe? Why not just use innerHTML? Even though innerHTML is now part of the HTML5 spec, it is a dangerous practice to insert the response from a server (evil or good) into an unchecked area. InnerHTML has also been noted to fail intermittently on iOS (just Google “ios innerhtml” to see the latest results) so it’s best to have a good workaround when the time comes.

Here is the latest performance test from http://jsperf.com/ajax-response-handling-innerhtml-vs-sandboxed-iframe shown in Figure 3-20. It shows that this sandboxed iFrame approach is just as fast, if not faster than innerHTML on many of today’s top mobile browsers.

The count is operations per second, so higher is better.

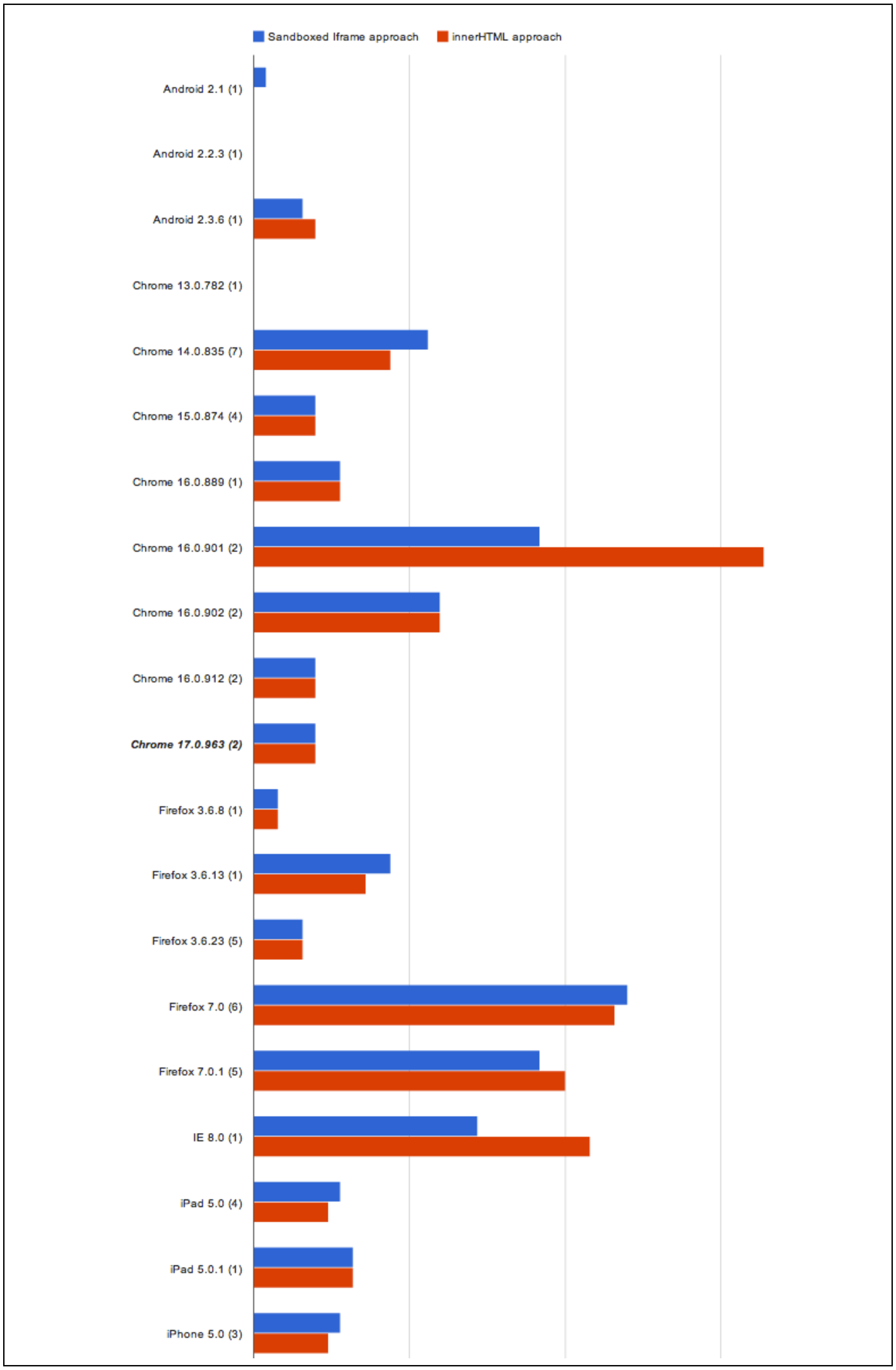


Figure 3-20. HTML5 iFrame vs. innerHTML()

Network Type Detection and Handling

Now that we have the ability to buffer (or predictive cache) our web app, we must provide the proper connection detection features that make our app smarter. This is where mobile app development gets extremely sensitive to online/offline modes and connection speed. Enter The Network Information API. Every time I show this feature in a presentation, someone in the audience raises their hand and asks “What would I use that for?”. So here is a possible way to setup an extremely smart mobile web app.

Boring common sense scenario first. While interacting with the Web from a mobile device on a high-speed train, the network may very well go away at various moments and different geographies might support different transmission speeds (e.g., HSPA or 3G might be available in some urban areas, but remote areas might support much slower 2G technologies). The following code addresses most of the connection scenarios and provides:

- Offline access through applicationCache.
- Detects if bookmarked and offline.
- Detects when switching from offline to online and vice versa.
- Detects slow connections and fetches content based on network type.

Again, all of these features require very little code. First we detect our events and loading scenarios:

[javascript]

```
window.addEventListener('load', function(e) {
 if (navigator.onLine) {
  // new page load
  processOnline();
 } else {
   // the app is probably already cached and (maybe) bookmarked...
   processOffline();
 }
}, false);

window.addEventListener("offline", function(e) {
  // we just lost our connection and entered offline mode, disable external link
  processOffline(e.type);
}, false);

window.addEventListener("online", function(e) {
  // just came back online, enable links
  processOnline(e.type);
}, false);
```

In the EventListeners above, we must tell our code if it is being called from an event or an actual page request or refresh. The main reason is because the body onload event won’t be fired when switching between the online and offline modes.

Next, we have a simple check for an ononline or onload event. This code resets disabled links when switching from offline to online, but if this app were more sophisticated, you might insert logic that would resume fetching content or handle the UX for intermittent connections.

[javascript]

```
function processOnline(eventType) {

  setupApp();
  checkAppCache();

  // reset our once disabled offline links
  if (eventType) {
    for (var i = 0; i < disabledLinks.length; i++) {
      disabledLinks[i].onclick = null;
    }
  }
}
```

The same goes for processOffline(). Here you would manipulate your app for offline mode and try to recover any transactions that were going on behind the scenes. The code below digs out all of our external links and disables them—trapping users in our offline app FOREVER muhahaha!

[javascript]

```
function processOffline() {
  setupApp();

  // disable external links until we come back - setting the bounds of app
  disabledLinks = getUnconvertedLinks(document);

  // helper for onlcick below
  var onclickHelper = function(e) {
    return function(f) {
      alert('This app is currently offline and cannot access the hotness');return
false;
    }
  };

  for (var i = 0; i < disabledLinks.length; i++) {
    if (disabledLinks[i].onclick == null) {
      //alert user we're not online
      disabledLinks[i].onclick = onclickHelper(disabledLinks[i].href);

    }
  }
}
```

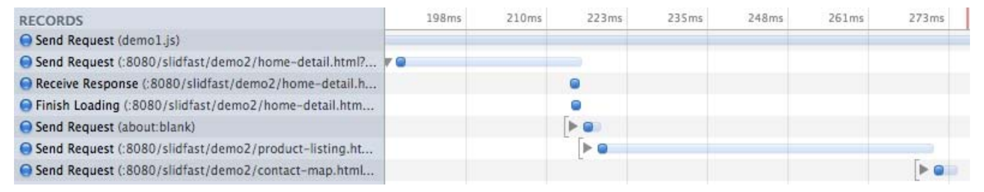
OK, so on to the good stuff. Now that our app knows what connected state it’s in, we can also check the type of connection when it’s online and adjust accordingly. I have listed typical North American providers download and latencies in the comments for each connection.

[javascript]

```
function setupApp(){
  // create a custom object if navigator.connection isn't available
  var connection = navigator.connection || {'type':'0'};
  if (connection.type == 2 || connection.type == 1) {
      //wifi/ethernet
      //Coffee Wifi latency: ~75ms-200ms
      //Home Wifi latency: ~25-35ms
      //Coffee Wifi DL speed: ~550kbps-650kbps
      //Home Wifi DL speed: ~1000kbps-2000kbps
      fetchAndCache(true);
  } else if (connection.type == 3) {
  //edge
      //ATT Edge latency: ~400-600ms
      //ATT Edge DL speed: ~2-10kbps
      fetchAndCache(false);
  } else if (connection.type == 2) {
      //3g
      //ATT 3G latency: ~400ms
      //Verizon 3G latency: ~150-250ms
      //ATT 3G DL speed: ~60-100kbps
      //Verizon 3G DL speed: ~20-70kbps
      fetchAndCache(false);
  } else {
  //unknown
      fetchAndCache(true);
  }
}
```

There are numerous adjustments we could make to our fetchAndCache process, but all I did here was tell it to fetch the resources asynchronous (true) or synchronous (false) for a given connection.

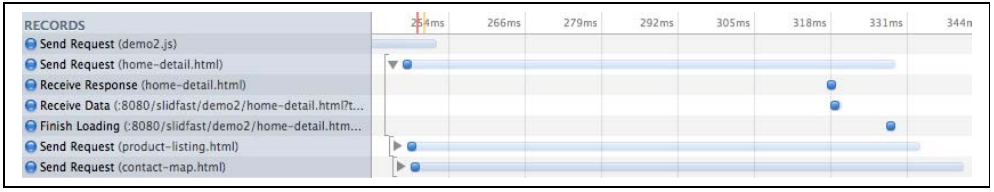
Edge (Synchronous) Request Timeline is shown in Figure 3-21.



RECORDS
Send Request (demo1.js)
Send Request (:8080/slidfast/demo2/home-detail.html?...
Receive Response (:8080/slidfast/demo2/home-detail.h...
Finish Loading (:8080/slidfast/demo2/home-detail.htm...
Send Request (about:blank)
Send Request (:8080/slidfast/demo2/product-listing.ht...
Send Request (:8080/slidfast/demo2/contact-map.html...

Figure 3-21. Synchronous Page Loading

WIFI (Asynchronous) Request Timeline is shown in Figure 3-22.



RECORDS
Send Request (demo2.js)
Send Request (home-detail.html)
Receive Response (home-detail.html)
Receive Data (:8080/slidfast/demo2/home-detail.html?t...
Finish Loading (:8080/slidfast/demo2/home-detail.htm...
Send Request (product-listing.html)
Send Request (contact-map.html)

Figure 3-22. Asynchronous Page Loading

This allows for at least some method of user experience adjustment based on slow or fast connections. This is by no means an end-all-be-all solution. Another todo would be to throw up a loading modal when a link is clicked (on slow connections) while the app still may be fetching that link's page in the background. The big point here is to cut down on latencies while leveraging the full capabilities of the user's connection with the latest and greatest HTML5 has to offer. View the network detection demo at http://html5e.org/network-detection.html.

Mobile Debugging

Weinre (http://phonegap.github.com/weinre/) is a debugger for web pages, like FireBug (for FireFox) and Web Inspector (for WebKit-based browsers), except it's designed to work remotely, and in particular, to allow you debug web pages on a mobile device such as a phone.

If you aren't familiar with FireBug or Web Inspector, weinre isn't going to make too much sense to you. weinre reuses the user interface code from the Web Inspector project at WebKit, so if you've used Safari's Web Inspector or Chrome's Developer Tools, weinre will be very familiar.

Supported platforms (debug client):

- weinre Mac application - Mac OS X 10.6 64-bit
- Google Chrome 8.x
- Apple Safari 5.x

Supported platforms (debug target):

- Android 2.2 Browser application
- Android 2.2 w/PhoneGap 0.9.2
- iOS 4.2.x Mobile Safari application
- BlackBerry v6.x simulator
- webOS 2.x (unspecified version)

4

HTML5 From the Server Side

To some, server-side UI frameworks, which automatically generate JavaScript, CSS and HTML, are the saviors of enterprise development. To others, server-side UI frameworks create a massive bottleneck, and tie you to stale ideas and structures.

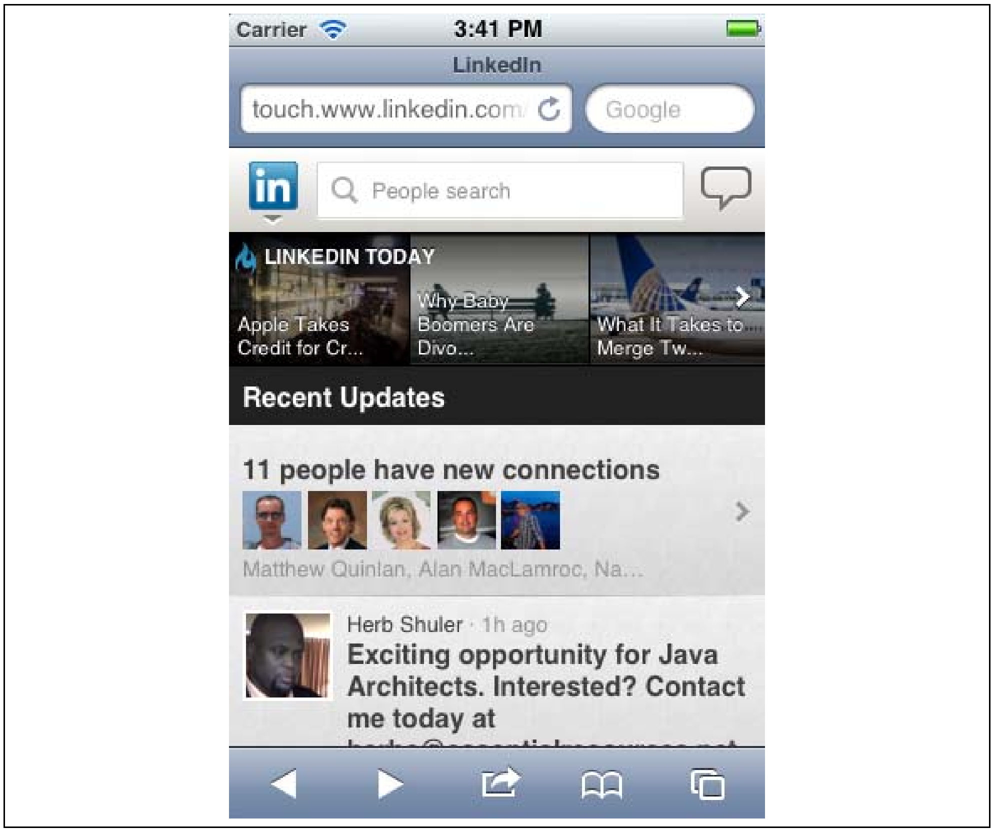
Today, developers are forced to look at web application architecture from a different perspective where the browser and JavaScript are taking just as much spotlight as server-side code (or in some cases, JavaScript is the server-side code).

Each developer has a preference on which server-side approach is "best" and I have personally listened to both sides of the fence on this topic. Veteran developers trust the JVM for scaling and thread management. They know what works, which languages are best at supporting the enterprise, and wouldn't think of using JavaScript as a server side language. But despite the naysayers, new server-based initiatives are getting started in the enterprise with JavaScript.

After LinkedIn launched their new HTML5 based mobile web app in late 2011 (Shown in Figure 4-23), their mobile development lead gave the following statement in an interview:

"The app is two to 10 times faster on the client side than its predecessor, and on the server side, it's using a fraction of the resources, thanks to a switch from Ruby on Rails to Node.js, a server-side JavaScript development technology that's barely a year old but already rapidly gaining traction."

http://venturebeat.com/2011/08/16/linkedin-node/


Carrier
3:41 PM
LinkedIn
touch.www.linkedin.com
Google
in
People search
LINKEDIN TODAY
Apple Takes Credit for Cr...
Why Baby Boomers Are Divo...
What It Takes to Merge Tw...
Recent Updates
11 people have new connections
Matthew Quinlan, Alan MacLamroc, Na...
Herb Shuler · 1h ago

Exciting opportunity for Java Architects. Interested? Contact me today at

Figure 4-23. LinkedIn Mobile Web

The world of web application development is ever-changing and will continue to change as years go by, but how do we build a solid HTML5 based solution in today's new world of server-side technologies? In this chapter, we're going to review everything it takes to setup the appropriate HTML5 infrastructure for your web app. Certain sections (such as WebSockets and WebStorage) will be given more emphasis and examples in latter chapters. This chapter will serve as our setup script for the rest of the book.

Device and Feature Detection

The first step in delivering new HTML5 based features to your user base is actually detecting what their browser supports. The need for communicating with the browser, to see what it supports before the page is even rendered, is here. In the past, we have been forced to detect and parse the, sometimes unreliable, browser User Agent (UA) string and then assume we have the correct device. Today, we have frameworks such as Modernizr, has.js, or just simple JavaScript, which help us detect client side capabilities at a much finer grained level.

Which detection method is best? Obviously, the User Agent string parsing approach is flawed and should be handled with caution. This is because, even at the time of this writing, browser vendors are still getting it wrong. The latest phone-based Firefox on Android User-Agent string reports itself as a tablet… not a phone.

In this section, we’ll see how we can use each approach effectively, by itself or to compliment the other.

Feature Detection

JavaScript based feature detection is often implemented by creating a DOM element to see if it behaves as expected.

For example:

[javascript]

```
detectCanvas() ? showGraph() : showTable();

function detectCanvas() {
  var canvas = document.createElement("canvas");
  return canvas.getContext ? true : false;
}
```

Here we are creating a canvas element and checking to see if it supports the getContext property. Checking a property of the created element is a must, because browsers will allow you to create any element in the DOM whether it’s supported or not.

This approach is one of many, and today we have open source, community backed frameworks that do the heavy lifting for us.

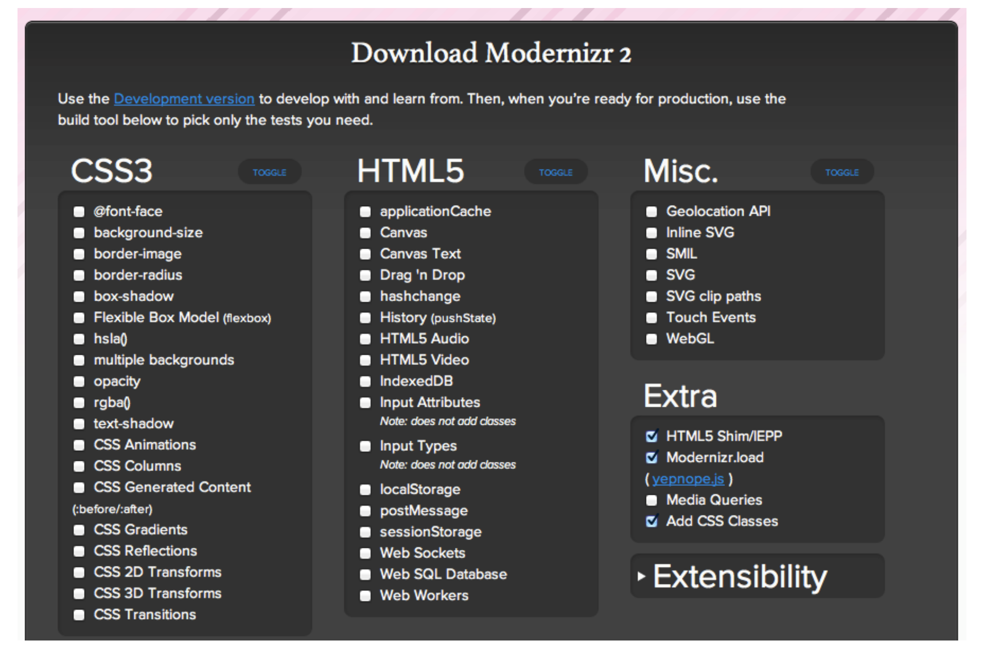
Here’s the same code as above, implemented with the Modernizr framework:

[javascript]

```
Modernizr.canvas ? showGraph() : showTable();
```

However, feature detection frameworks may come at a cost. As I stated in the opening paragraph, we are running a series of tests on the browser window before the page is rendered. This can get expensive; for example, running the full suite of Modernizr detections can take 30+ milliseconds to run per page load. One must consider the costs of computing values before DOM render and then modifying the DOM based on the frameworks findings. When you’re ready to take your app to production, make sure you’re not using the development version of your chosen feature detection library.

Frameworks like Modernizr give us a tool, which allows us to pick and choose the features our app must have, as shown in Figure 4-24.



Download Modernizr 2

Use the Development version to develop with and learn from. Then, when you're ready for production, use the build tool below to pick only the tests you need.

CSS3 TOGGLE

- □ @font-face
- □ background-size
- □ border-image
- □ border-radius
- □ box-shadow
- □ Flexible Box Model (flexbox)
- □ hsla()
- □ multiple backgrounds
- □ opacity
- □ rgba()
- □ text-shadow
- □ CSS Animations
- □ CSS Columns
- □ CSS Generated Content (:before/:after)
- □ CSS Gradients
- □ CSS Reflections
- □ CSS 2D Transforms
- □ CSS 3D Transforms
- □ CSS Transitions

HTML5 TOGGLE

- □ applicationCache
- □ Canvas
- □ Canvas Text
- □ Drag 'n Drop
- □ hashchange
- □ History (pushState)
- □ HTML5 Audio
- □ HTML5 Video
- □ IndexedDB
- □ Input Attributes
  Note: does not add classes
- □ Input Types
  Note: does not add classes
- □ localStorage
- □ postMessage
- □ sessionStorage
- □ Web Sockets
- □ Web SQL Database
- □ Web Workers

Misc. TOGGLE

- □ Geolocation API
- □ Inline SVG
- □ SMIL
- □ SVG
- □ SVG clip paths
- □ Touch Events
- □ WebGL

Extra

- ☑ HTML5 Shim/IEPP
- ☑ Modernizr.load (yepnope.js)
- □ Media Queries
- ☑ Add CSS Classes

▸ Extensibility

Figure 4-24. Modernizr Production Config

Feature detection performance also depends on what kind of devices and browsers you are targeting. For example, running a feature detection framework on an 1st generation smart phone or an old Blackberry may be the straw that breaks the camel's back. So, it's up to you to tweak feature detection to gain top performance on your target browsers.

Sometimes, you may need to go a step further and detect the actual form factor of the device. One other project worth mentioning is FormFactor.js — however it seems a bit inactive as of late. It helps you customize your web app for different form factors, e.g. when you make "the mobile version", "the TV version", etc.

FormFactor.js is a framework to manage conceptually distinct user interfaces. It doesn't eliminate the need for feature detection, but you use them in the context of a particular form factor's interface.

[javascript]

```
if(formfactor.is("tv")) {
  alert("Look ma, Im on tv!");
}

if(formfactor.isnt("tv")) {
  alert("The revolution will not be televised");
}
```

See https://github.com/PaulKinlan/formfactor for more examples.

UA Detection

There are times when we must detect the User Agent and parse it accordingly. Typically, we can determine the browser by inspecting JavaScript's window.navigator object or by using the User-Agent request header server side.

This approach may work for most browsers, but it’s not dependable — as noted in a recent bug report:

Issue Summary: When using the Firefox browser on an Android mobile phone, the MobileESP code library erroneously reports the device as an Android tablet. An Android tablet is correctly identified as an Android tablet. This issue only affects mobile phones and similar small-screen Android devices like MP3 players (such as the Samsung Galaxy Player).

Root Cause: Mozilla uses the exact same useragent string for both phones and tablets. The string has the word “Android” in both. According to Google guidelines, Mozilla should include the word “mobile” in the useragent string for mobile phones. Unfortunately, Mozilla is not compliant with Google’s guidelines. The omission of the word “mobile” is the reason why phones are erroneously identified as tablets.

So if User-Agent detection isn’t always dependable, why would we use it?

- You know, ahead of time, which platforms you are supporting and their UA strings report correctly. For example, if we only care about the environment (not its features) our application is running in, such as iOS, we could deliver a custom UI for that environment only.
- In combination with feature detection JavaScript which only calls the minimum functions to check the device. For example, you may not care about the discrepancy in the reported string since it’s unneeded information. You might only care that it reports “TV” and everything else is irrelevant. This also allows for “light” feature detection via JavaScript.
- When you don’t want all JavaScript based feature tests to be downloaded to every browser and executed when UA-sniffing based optimizations are available.

Another example of User-Agent detection at Yahoo!:

“At Yahoo we have a database full of around 10,000 mobile devices. Because user agent strings vary even on one device (because of locale, vendor, versioning, etc), this has resulted in well over a half a MILLION user agents. It’s become pretty crazy to maintain, but is necessary because there’s really no alternative for all these feature phones, which can’t even run JavaScript.”

Companies like Google use a JavaScript based (also ported to node.js) User Agent parser internally. It’s a wrapper for a ~7kb JSON file which can be used in other languages.

https://github.com/Tobie/Ua-parser

[javascript]

```
var uaParser = require('ua-parser');
var ua = uaParser.parse(navigator.userAgent);

console.log(ua.tostring());
// -> "Safari 5.0.1"

console.log(ua.toVersionString());
// -> "5.0.1"

console.log(ua.toFullString());
// -> "Safari 5.0.1/Mac OS X"
```

```
console.log(ua.family);
// -> "Safari"

console.log(ua.major);
// -> 5

console.log(ua.minor);
// -> 0

console.log(ua.patch);
// -> 1

console.log(ua.os);
// -> Mac OS X
```

Another platform detection library written in JavaScript is Platform.js. It’s used by jsperf.com for User Agent detection.

Platform.js has been tested in at least Adobe AIR 2.6, Chrome 5-15, Firefox 1.5-8, IE 6-10, Opera 9.25-11.52, Safari 2-5.1.1, Node.js 0.4.8-0.6.1, Narwhal 0.3.2, RingoJS 0.7-0.8, and Rhino 1.7RC3.

https://github.com/Bestiejs/Platform.js

[javascript]

```
// on IE10 x86 platform preview running in IE7 compatibility mode on Windows 7 64 bit edition
platform.name; // 'IE'
platform.version; // '10.0'
platform.layout; // 'Trident'
platform.os; // 'Windows Server 2008 R2 / 7 x64'
platform.description; // 'IE 10.0 x86 (platform preview; running in IE 7 mode) on Windows Server 2008 R2 / 7 x64'

// or on an iPad
platform.name; // 'Safari'
platform.version; // '5.1'
platform.product; // 'iPad'
platform.manufacturer; // 'Apple'
platform.layout; // 'WebKit'
platform.os; // 'iOS 5.0'
platform.description; // 'Safari 5.1 on Apple iPad (iOS 5.0)'

// or parsing a given UA string
var info = platform.parse('Mozilla/5.0 (Macintosh; Intel Mac OS X 10.7.2; en; rv:2.0) Gecko/20100101 Firefox/4.0 Opera 11.52');
info.name; // 'Opera'
info.version; // '11.52'
info.layout; // 'Presto'
info.os; // 'Mac OS X 10.7.2'
info.description; // 'Opera 11.52 (identifying as Firefox 4.0) on Mac OS X 10.7.2'
```

On the server-side, MobileESP (http://blog.mobileesp.com) is an open source framework, which detects the User-Agent header. This gives you the ability to direct the user to the appropriate page or allows developers to code to supported device features.

MobileESP is available in the following languages:

- PHP
- Java (server side)
- ASP.NET (C#)
- JavaScript
- Ruby
- Classic ASP (VBscript)

Java usage example:

```
[java]
userAgentStr = request.getHeader("user-agent");
httpAccept = request.getHeader("Accept");
uAgentTest = new UAgentInfo(userAgentStr, httpAccept);

If(uAgentTest.detectTierIphone()){
…//Perform redirect
}
```

PHP Usage example:

[php]

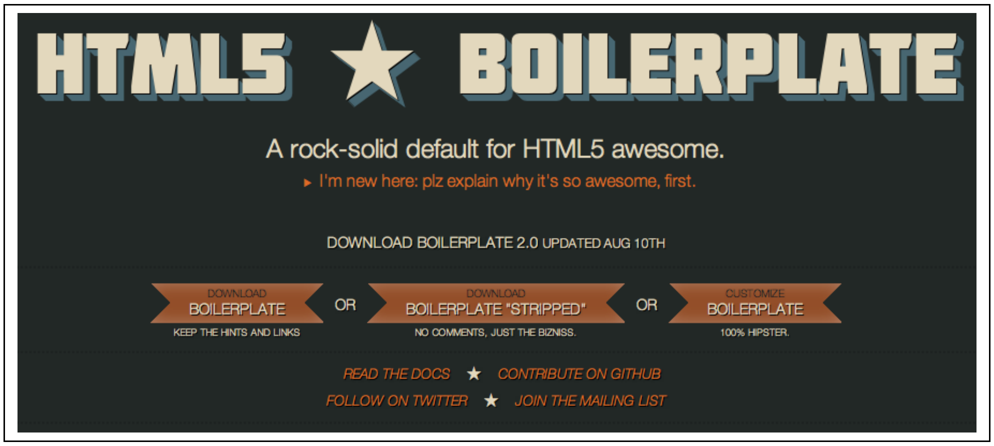
```
<?php

      //Load the Mobile Detection library
      include("code/mdetect.php");

      //In this simple example, we'll store the alternate home page file names.
      $iphoneTierHomePage = 'index-tier-iphone.htm';

      //Instantiate the object to do our testing with.
      $uagent_obj = new uagent_info();

      //In this simple example, we simply re-route depending on which type of
device it is.
      //Before we can call the function, we have to define it.
      function AutoRedirectToProperHomePage()
      {
          global $uagent_obj, $iphoneTierHomePage, $genericMobileDeviceHomePage,
$desktopHomePage;

          if ($uagent_obj->isTierIphone == $uagent_obj->true)
          //Perform redirect
      }
```

So there you have it, User Agent detection is unreliable and should be used with caution or for specific cases. In the Android/Firefox bug report mentioned above, you could still implement User Agent detection and then use feature detection to find the maximum screen size for Android-based mobile phones using CSS Media Queries. There’s always a workaround and problems such as these should not deter you from using the User Agent string.

Markup and Resources

Resource minification and compression are mandatory in today’s world of “Mobile First”. If you aren’t concerned with the size of your JavaScript and CSS libraries, you should be. GZIP compression is used to achieve a minimal transfer of bytes over a given web based connection and minification is the process of removing all unnecessary characters from source code, without changing its functionality. However, where you gain performance on the client side, it’s easy to forget about increased overhead on your server resources, so use caching where appropriate.

Thanks to the community at HTML5Boilerplate.com, we have a quick start for server optimizations and resource compression (Figure 4-25).



A rock-solid default for HTML5 awesome.

▸ I'm new here: plz explain why it's so awesome, first.

DOWNLOAD BOILERPLATE 2.0 UPDATED AUG 10TH

Figure 4-25. HTML5 Boilerplate

Compression

In Apache HTTP server, .htaccess (hypertext access) is the configuration file that allows for web server configuration. HTML5 Boilerplate team has identified a number of best practice server rules for making web pages fast and secure, these rules can be applied by configuring .htaccess file.

You'll want to have these modules enabled for optimum performance:

mod_setenvif.c (setenvif_module)

mod_headers.c (headers_module)

mod_deflate.c (deflate_module)

mod_filter.c (filter_module)

mod_expires.c (expires_module)

mod_rewrite.c (rewrite_module)

> You will need to locate the httpd.conf file, which is normally in the conf folder in the folder where you installed Apache (for example C:\apache\conf\httpd.conf). Open up this file in a text editor. Near the top (after a bunch of comments) you will see a long list of modules.

Check to make sure that the modules listed above are not commented out. If they are, go ahead and uncomment them and restart Apache.

Security

Do not turn off your ServerSignature (i.e., the Server: HTTP header). Serious attackers can use other kinds of fingerprinting methods to figure out the actual server and components running behind a port. Instead, as a site owner, you should keep track of what's listening on ports on hosts that you control. Run a periodic scanner to make sure nothing suspicious is running on a host you control, and use the ServerSignature to determine if this is the web server and version that you expect.

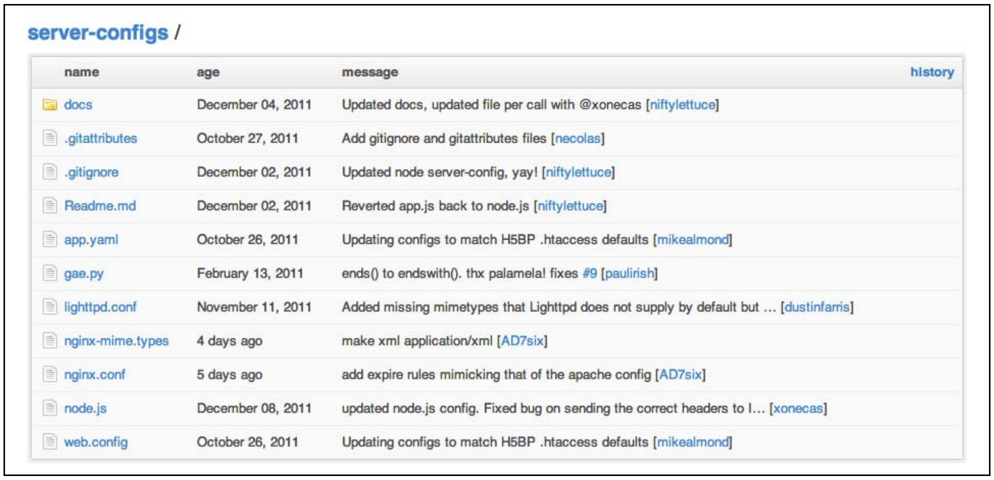
GZIP

Gzipping generally reduces the response size by about 70%. Approximately 90% of today's Internet traffic travels through browsers that claim to support gzip. If you use Apache, the module configuring gzip depends on your version: Apache 1.3 uses mod_gzip while Apache 2.x uses mod_deflate.

[xml]

```
<IfModule mod_deflate.c>
  AddOutputFilterByType DEFLATE text/html text/plain text/css text/javascript
application/javascript application/json
  AddOutputFilterByType DEFLATE text/xml application/xml text/x-component

  <FilesMatch "\.(ttf|otf|eot|svg)$" >
    SetOutputFilter DEFLATE
  </FilesMatch>
</IfModule>
```

There are many more server configuration examples at https://github.com/h5bp/server-configs/ shown in Figure 4-26.



server-configs /

name	age	message	history
docs	December 04, 2011	Updated docs, updated file per call with @xonecas [niftylettuce]	
.gitattributes	October 27, 2011	Add gitignore and gitattributes files [necolas]	
.gitignore	December 02, 2011	Updated node server-config, yay! [niftylettuce]	
Readme.md	December 02, 2011	Reverted app.js back to node.js [niftylettuce]	
app.yaml	October 26, 2011	Updating configs to match H5BP .htaccess defaults [mikealmond]	
gae.py	February 13, 2011	ends() to endswith(). thx palamela! fixes #9 [paulirish]	
lighttpd.conf	November 11, 2011	Added missing mimetypes that Lighttpd does not supply by default but … [dustinfarris]	
nginx-mime.types	4 days ago	make xml application/xml [AD7six]	
nginx.conf	5 days ago	add expire rules mimicking that of the apache config [AD7six]	
node.js	December 08, 2011	updated node.js config. Fixed bug on sending the correct headers to I… [xonecas]	
web.config	October 26, 2011	Updating configs to match H5BP .htaccess defaults [mikealmond]	

Figure 4-26. H5BP GitHub Examples

Other Server Side Proxies and Libraries

JAWR

http://jawr.java.net/

JAWR is a tunable packaging solution for Javascript and CSS, which allows for rapid development of resources in separate module files. Developers can work with a large set of split javascript files in development mode, then Jawr bundles all together into one or several files in a configurable way.

By using a tag library, Jawr allows you to use the same, unchanged pages for development and production. Jawr also minifies and compresses the files, resulting in reduced page load times.

Jawr is configured using a simple .properties descriptor, and aside of standard java web applications it can also be used with Facelets and Grails applications.

Ziproxy

http://ziproxy.sourceforge.net/

Ziproxy is a forwarding, non-caching, compressing HTTP proxy targeted for traffic optimization. It minifies and optimizes HTML, CSS, and JavaScript resources and, in addition, re-compresses pictures.

Basically, it squeezes images by converting them to lower quality JPEGs or JPEG 2000 and compresses (gzip) HTML and other text-like data.

It also provides other features such as: preemptive hostname resolution, transparent proxying, IP ToS marking (QoS), Ad-Blocker, detailed logging and more.

Ziproxy does not require a client software and provides acceleration for any web browser, any operational system.

JavaScript Minification

JavaScript and CSS resources may be minified, preserving their behavior while considerably reducing their file size. Some libraries also merge multiple script files into a single file for client download. This fosters a modular approach to development and limits HTTP requests.

Google has released their Closure Compiler, which provides minification as well as the ability to introduce more aggressive renaming, removing dead code, and providing function inlining.

In addition, certain online tools, such as Microsoft Ajax Minifier, the Yahoo! YUI Compressor or Pretty Diff, can compress CSS files.

JSMin

http://www.crockford.com/javascript/jsmin.html

JSMin is a filter that removes comments and unnecessary whitespace from JavaScript files. It typically reduces filesize by half, resulting in faster downloads. It also encourages a more expressive programming style because it eliminates the download cost of clean, literate self-documentation. It's recommended you use JSLint before minimizing your JavaScript with JSMin.

Packer

http://dean.edwards.name/packer/

Packer, for instance, can optionally Base64 compress the given source code in a manner that can be decompressed by regular web browsers, as well as shrink variable names that are typically 5–10 characters to single letters, which reduces the file size of the script and, therefore, makes it download faster.

JavaScript Build Tools

grunt

https://github.com/cowboy/grunt

Grunt is a task-based command line build tool for JavaScript projects. It gives you the ability to concatenate files, validate files with JSHint, and minify with UglifyJS. Grunt also allows your project to run headless QUnit tests with a PhantomJS instance.

JavaScript Frameworks and the Server

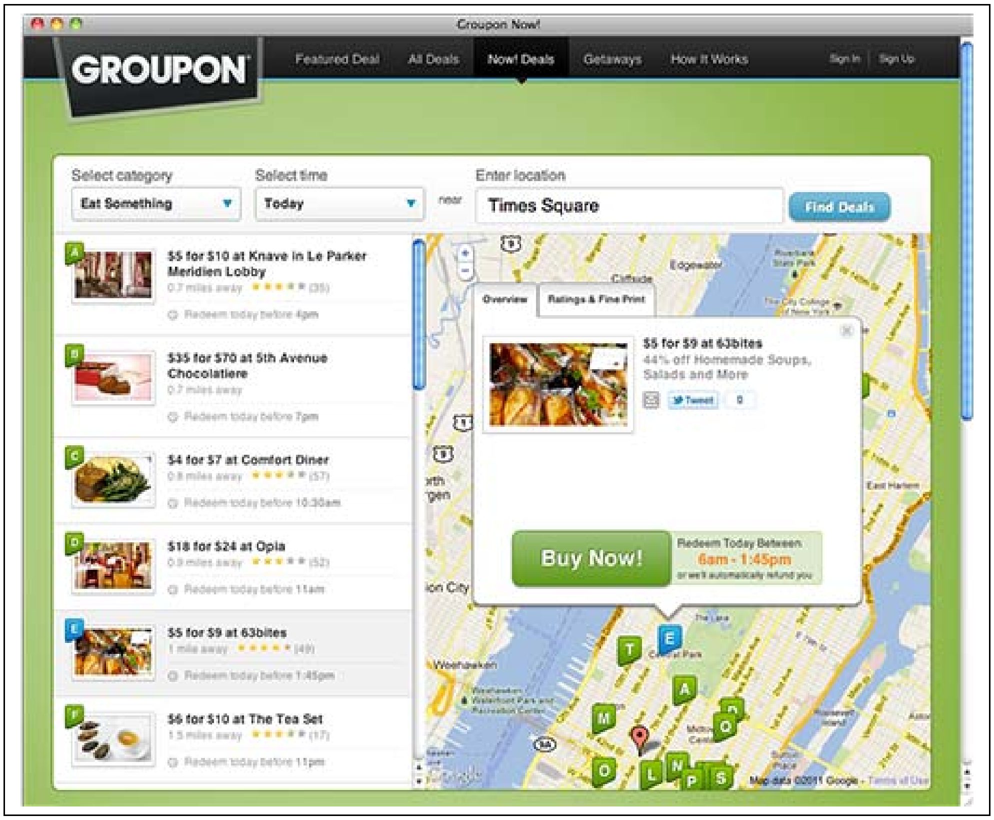
With the myriad of JavaScript MVC frameworks popping up over the past few years, it's important to get a high level view of what's available today and which ones support some form of server-side interaction. How each framework handles server-side collections of objects rendered to the DOM is important for a few reasons:

- Binding objects to the UI must be declarative and the view layer should auto-update as changes to the model occur.
- It's easy to create JavaScript heavy applications that end up as a tangled pile of jQuery, RESTful endpoint, callback mess. So a structured MVC approach makes code more maintainable and reusable.

Our goal for this section is to identify JavaScript frameworks which are server agnostic and use transports such as HTTP (for RESTful endpoints) and WebSockets.

Backbone.js

Backbone.js is today's framework of choice and for good reason; an impressive list of brands, such as Foursquare, Posterous, Groupon (Figure 4-27), and many others have built cutting-edge JavaScript applications with Backbone.js.



Select category: Eat Something

Select time: Today

Enter location: Times Square

A $5 for $10 at Knave in Le Parker Meridien Lobby
0.7 miles away (35)
Redeem today before 4pm

B $35 for $70 at 5th Avenue Chocolatiere
0.7 miles away
Redeem today before 7pm

C $4 for $7 at Comfort Diner
0.8 miles away (57)
Redeem today before 10:30am

D $18 for $24 at Opia
0.9 miles away (52)
Redeem today before 11am

E $5 for $9 at 63bites
1 mile away (49)
Redeem today before 1:45pm

F $6 for $10 at The Tea Set
1.5 miles away (17)
Redeem today before 11pm

$5 for $9 at 63bites
44% off Homemade Soups, Salads and More

Redeem Today Between
6am - 1:45pm
or we'll automatically refund you

Figure 4-27. Groupon uses Backbone!

Backbone.js uses Underscore.js heavily and gives developers the options of using jQuery or Zepto for the core DOM framework. It also boasts a healthy community and strong word of mouth.

With Backbone, you represent your data as Models, which can be created, validated, destroyed, and saved to the server. Whenever a UI action causes an attribute of a model to change, the model triggers a "change" event; all the Views that display the model's state can be notified of the change, so that they are able to respond accordingly, re-rendering themselves with the new information. In a finished Backbone app, you don't have to write the glue code that looks into the DOM to find an element with a specific id, and update the HTML manually — when the model changes, the views simply update themselves.

Server Synchronization

Backbone.sync is the function that Backbone calls every time it attempts to read or save a model to the server. By default, it uses (jQuery/Zepto).ajax to make a RESTful JSON request. You can override it in order to use a different persistence strategy, such as WebSockets, XML transport, or Local Storage.

With the default implementation, when Backbone.sync sends up a request to save a model, its attributes will be passed, serialized as JSON, and sent in the HTTP body with content-type application/json. When returning a JSON response, send down the attributes of the model that have been changed by the server, and need to be updated on the client. When responding to a "read" request from a collection (Collection#fetch), send down an array of model attribute objects.

The default sync handler maps CRUD to REST like so:

create → POST /collection

read → GET /collection[/id]

update → PUT /collection/id

delete → DELETE /collection/id

As an example, a Rails handler responding to an "update" call from Backbone might look like this: (In real code, never use update_attributes blindly, and always whitelist the attributes you allow to be changed.)

[ruby]

```
def update
  account = Account.find params[:id]
  account.update_attributes params
  render :json => account
End
```

One more tip for Rails integration is to disable the default namespacing for to_json calls on models by setting ActiveRecord::Base.include_root_in_json = false

emulateHTTPBackbone.emulateHTTP = true

If you want to work with a legacy web server that doesn't support Backbones's default REST/HTTP approach, you may choose to turn on Backbone.emulateHTTP. Setting this option will fake PUT and DELETE requests with a HTTP POST, setting the X-HTTP-Method-Override header with the true method. If emulateJSON is also on, the true method will be passed as an additional _method parameter.

[javascript]

```
Backbone.emulateHTTP = true;
model.save();  // POST to "/collection/id", with "_method=PUT" + header.
```

emulateJSONBackbone.emulateJSON = true

If you're working with a legacy web server that can't handle requests encoded as application/json, setting Backbone.emulateJSON = true; will cause the JSON to be serialized under a model parameter, and the request to be made with a application/x-www-form-urlencoded mime type, as if from an HTML form.

Since RESTful endpoints are server agnostic, we can choose any appserver backed by a database to save our Backbone.js models. In this example we’ll use a Java EE server coupled with a MySQL backend.

[In the process of writing code for this…almost done.]

https://github.com/ccoenraets/backbone-jax-cellar

http://coenraets.org/blog/2012/01/using-backbone-js-with-a-restful-java-back-end/

Ember.js

Ember.js (formerly Amber.js and SproutCore 2.0) is one of the newest contenders. It is an attempt to extricate the core features from SproutCore 2.0 into a more compact modular framework suited for the web. It's also well known for gracefully handling DOM updates.

Ember.js is what happened when SproutCore decided to be less Apple Cocoa and more jQuery. The result is a web framework that retains very important high-level concepts such as observers, bindings and state charts, while delivering a concise API. SproutCore started its life as the development framework behind an early client-side email application. Then, Apple used it to build MobileMe (and then iCloud), both of which include email clients. Needless to say, they've figured out that collections that update from the server are very important.

Server Synchronization

Ember Data is a library for loading models from a persistence layer (such as a JSON API), updating those models, then saving the changes. It provides many of the facilities you'd find in server-side ORMs like ActiveRecord, but is designed specifically for the unique environment of JavaScript in the browser.

As of this writing, Ember Data is definitely alpha-quality. The basics work, but there are edge cases that are not yet handled.

Here is a brief example of storing data with Ember:

[javascript]

```
// our model
App.Person = Ember.Object.extend();

App.people = Ember.ArrayController.create({
  content: [],
  save: function () {
    // assuming you are using jQuery, but could be other AJAX/DOM framework
    $.post({
      url: "/people",
      data: JSON.stringify( this.toArray() ),
      success: function ( data ) {
        // your data should already be rendered with latest changes
        // however, you might want to change status from something to "saved" etc.
      }
    });
  }
});
```

You'd then call App.people.save() at needed occasions.

Angular.js

Angular.js is a very nice framework developed by Googler's, and has some very interesting design choices — most namely Dependency Injection (or IOC) for JavaScript. Angular.js is well thought out with respect to template scoping and controller design. It supports a rich UI-Binding syntax to make things like filtering and transforming values a breeze.

Server Synchronization

The Angular “Model” is referenced from properties on angular scope objects. The data in your model could be Javascript objects, arrays, or primitives, it doesn't matter. What matters is that these are all referenced by the scope object.

Angular employs scopes to keep your data model and your UI in sync. Whenever something occurs to change the state of the model, angular immediately reflects that change in the UI, and vice versa.

Note: When building web applications your design needs to consider security threats from JSON Vulnerability and XSRF. Both server and the client must cooperate in order to eliminate these threats. Angular comes pre-configured with strategies that address these issues, but for this to work backend server cooperation is required.

[javascript]

```
// Define CreditCard class
var CreditCard = $resource('/user/:userId/card/:cardId',
 {userId:123, cardId:'@id'}, {
  charge: {method:'POST', params:{charge:true}}
 });

// We can retrieve a collection from the server
var cards = CreditCard.query();
// GET: /user/123/card
// server returns: [ {id:456, number:'1234', name:'Smith'} ];

var card = cards[0];
// each item is an instance of CreditCard
expect(card instanceof CreditCard).toEqual(true);
card.name = "J. Smith";
// non GET methods are mapped onto the instances
card.$save();
// POST: /user/123/card/456 {id:456, number:'1234', name:'J. Smith'}
// server returns: {id:456, number:'1234', name: 'J. Smith'};
```

For details see:

- http://docs.angularjs.org/#!/api/angular.service.$xhr
- http://docs.angularjs.org/#!/api/angular.service.$resource

Batman.js

Batman.js, created by Shopify, is another framework similar to Knockout and Angular. It has a nice UI binding system based on HTML attributes and is the only framework written in coffeescript. Batman.js is also tightly integrated with Node.js and even goes to the extent of having its own (optional) Node.js server.

Server Synchronization

A Batman.js model (http://batmanjs.org/documentation.html#models) object may have arbitrary properties set on it, just like any JS object. Only some of those properties are serialized and persisted to its storage backends.

Models have the ability to:

- persist to various storage backends

- only serialize a defined subset of their properties as JSON
- use a state machine to expose lifecycle events
- validate with synchronous or asynchronous operations

You define persisted attributes on a model with the encode macro:

[coffeescript]

```
class Article extends Batman.Model
    @encode 'body_html', 'title', 'author', 'summary_html', 'blog_id', 'id',
'user_id'
    @encode 'created_at', 'updated_at', 'published_at', Batman.Encoders.railsDate
    @encode 'tags',
      encode: (tagSet) -> tagSet.toArray().join(', ')
      decode: (tagString) -> new Batman.Set(tagString.split(', ')...)
```

Given one or more strings as arguments, @encode will register these properties as persisted attributes of the model, to be serialized in the model's toJSON() output and extracted in its fromJSON(). Properties that aren't specified with @encode will be ignored for both serialization and deserialization. If an optional coder object is provided as the last argument, its encode and decode functions will be used by the model for serialization and deserialization, respectively.

By default, a model's primary key (the unchanging property which uniquely indexes its instances) is its id property. If you want your model to have a different primary key, specify the name of the key on the primaryKey class property:

[coffeescript]

```
class User extends Batman.Model
  @primaryKey: 'handle'
  @encode 'handle', 'email'
```

To specify a storage adapter for persisting a model, use the @persist macro in its class definition:

[coffeescript]

```
class Product extends Batman.Model
  @persist Batman.LocalStorage
```

Now when you call save() or load() on a product, it will use the browser window's localStorage to retrieve or store the serialized data.

If you have a REST backend you want to connect to, Batman.RestStorage is a simple storage adapter which can be subclassed and extended to suit your needs. By default, it will assume your CamelCased-singular Product model is accessible at the underscored-pluralized "/products" path, with instances of the resource accessible at /products/:id. You can override these path defaults by assigning either a string or a function-returning-a-string to the url property of your model class (for the collection path) or to the prototype (for the member path). For example:

[coffeescript]

```
class Product extends Batman.Model
  @persist Batman.RestStorage
  @url = "/admin/products"
  url: -> "/admin/products/#{@id}"
```

Knockout.js

Knockout.js is built around three core features:

- Observables and dependency tracking
- Declarative bindings
- Templating

Knockout is designed to allow the use of arbitrary JavaScript objects as view models. As long as some of your view model's properties are observables, you can use KO to bind to them to your UI, and the UI will be updated automatically whenever the observable properties change.

Server Synchronization

Observables are declared on model properties. They allow automatic updates to the UI when the model changes:

[javascript]

```
var viewModel = {
    serverTime: ko.observable(),
    numUsers: ko.observable()
}
```

Since the server doesn't have any concept of observables, it will just supply a plain JavaScript object (usually serialized as JSON).

Manual Binding

You could bind this view model to some HTML elements as follows:

[html]

```
The time on the server is: <span data-bind='text: serverTime'></span>
and <span data-bind='text: numUsers'></span> user(s) are connected.
```

Since the view model properties are observable, KO will automatically update the HTML elements whenever those properties change.

Next, you want to fetch the latest data from the server. Every 5 seconds you might issue an Ajax request (e.g., using jQuery's $.getJSON or $.ajax functions):

[javascript]

```
var data = getDataUsingAjax();          // Gets the data from the server
```

The server might return JSON data similar to the following:

[javascript]

```
{
    serverTime: '2010-01-07',
    numUsers: 3
}
```

Finally, to update your view model using this data (without using the mapping plugin), you would write:

```
[javascript]
```

```
// Every time data is received from the server:
viewModel.serverTime(data.serverTime);
viewModel.numUsers(data.numUsers);
```

You would have to do this for every variable you want to display on your page. If your data structures become more complex (e.g. they contain children or contain arrays) this becomes very cumbersome to handle manually. What the mapping plugin allows you to do is create a mapping from the regular JavaScript object (or JSON structure) to an observable view model.

Mapping plugin

The mapping plugin gives you a straightforward way to map that plain JavaScript object into a view model with the appropriate observables. This is an alternative to manually writing your own JavaScript code that constructs a view model based on some data you've fetched from the server.

To create a view model via the mapping plugin, replace the creation of viewModel in the code above with the ko.mapping.fromJS function:

[javascript]

```
var viewModel = ko.mapping.fromJS(data);
```

This automatically creates observable properties for each of the properties on data. Then, every time you receive new data from the server, you can update all the properties on viewModel in one step by calling the ko.mapping.fromJS function again:

[javascript]

```
// Every time data is received from the server:
ko.mapping.fromJS(data, viewModel);
```

All properties of an object are converted into an observable. If an update would change the value, it will update the observable.

Arrays are converted into observable arrays. If an update would change the number of items, it will perform the appropriate add/remove actions. It will also try to keep the order the same as the original JavaScript array.

5

WebSockets

Every HTTP request sent from the browser includes headers, whether you want them or not. These are not small headers. Uncompressed request and response headers can vary in size from 200 bytes to over 2KB. Although, typical usage is somewhere between 700-900 bytes, those numbers will grow as User Agents expand features.

WebSockets give us minimal overhead and a much more efficient way of delivering data to the client and server with full duplex communication through a single socket. The WebSocket connection is made after a small HTTP handshake occurs between the client and the server, over the same underlying TCP/IP connection. This gives us an open connection between the client and the server and both parties can start sending data at any time.

A few advantages, among others, are:

- No HTTP headers
- No lag due to keep-alive issues
- Low latency, better throughput and responsiveness
- Easier on mobile device batteries

Building The Stack

To effectively develop any application with WebSockets, one must accept the idea of the “Real Time Web” and the programming model that comes along with it. Freemium services surrounding this space, such as Pusher (http://pusher.com/), are starting to emerge and companies like Kaazing, which offer the Kaazing Gateway, have been providing adapters for STOMP and Apache ActiveMQ for years. So you can choose to build a WebSocket stack yourself, as I show in the next section, or you can choose a service or project to manage the connections and graceful degradation for you.

There are plenty of wrapper frameworks around WebSockets which provide graceful degradation. From Socket.io to Cometd to whatever’s hot right now, graceful degradation is the process of falling back to use older technologies, such as Flash or long polling, within the browser if the WebSocket protocol is not supported. Comet, push technology, and long-polling in web apps are slow, inefficient, inelegant and have a higher potential magnitude for unreliability. For this book I am only covering the core WebSocket specification to avoid confusion and to keep things simple.

On the server - behind the scenes

Keeping a large number of connections open at the same time requires an architecture that permits other processing to continue before the transmission has finished. Such architectures are usually designed around threading or Asynchronous non-blocking IO (NIO). As for the debates between NIO and threading, some might say that NIO does not actually perform better than threading — only allowing you to write single-threaded event loops for multiple clients as with select on unix. Others argue that it depends on your expected workloads. If you have lots of long-term idle connections, NIO wins due to not having thousands of threads “blocking on a read” operation. Again, there are many debates over whether threads are faster or easier to write than event loops (or the opposite) so it all depends on the type of use case you are trying to handle. Don’t worry, I’ll show examples of both in this chapter.

Programming Models

With WebSockets, we have a new development model for server side applications; event based programming. Of course, most server side frameworks provide eventing mechanisms but there aren’t many which extend this eventing all the way through to the web browser.

For example, let’s say our server side framework is capable of sending an event and you have observers of this event in your code. WebSockets gives us the ability to extend that event so that it carries all the way from the server side into the connected client’s browser. A good example would be to notify all WebSocket connections that a user has registered on your site. We’ll jump into the code for this scenario at the in a minute, but for now let’s touch up on the basics.

There are 3 out-of-box events associated with WebSockets: onopen, onmessage, and onclose. For starters, we must wire up these three listeners to utilize the core functionality that the WebSocket specification gives us. The open event is fired when the WebSocket connection is opened successfully. The message event is fired when the server sends data. The close event is fired when the WebSocket connection is closed.

[javascript]

```
objWebSocket.onopen = function(evt)
{
   alert("WebSocket connection opened successfully");
};
objWebSocket.onmessage = function(evt)
{
   alert("Message : " + evt.data);
};
objWebSocket .onclose = function(evt)
{
   alert("WebSocket connection closed”);
};
```

Once the WebSocket connection is opened, the onmessage event is fired when the server sends the data to the client. If the client wants to send data to the server, it can do so as follows:

[javascript]

```
objWebSocket.send("Hello World");
```

But sending messages in the form of strings over raw WebSockets isn't very appealing when we want to develop enterprise-style web applications. Since current WebSocket implementations mostly deal with strings, we're going to be using JSON to transfer data to and from the server. But how do we propagate our server-side events that are fired on the server and have them bubble up on the client?

One approach is to relay the events. When a specific server-side event is fired, we have a listener or observer that translates the data to JSON and sends it to all connected clients.

Relaying Events From the Server to the Browser

Let’s start with the server. I'm using the JBoss AS7 application server (http://www.jboss.org/jbossas/downloads/) and embedding Jetty within the web application. The main reasoning behind this approach is to take advantage of a lightweight Java EE 6.0 [Full Profile] application server. There are a few other Java based options out there, such as GlassFish or running Jetty standalone, but the value of having Contexts and Dependency Injection (CDI), Distributed Transactions, Scalable JMS messaging and Data Grid support out-of-box is extremely valuable in cutting-edge enterprise initiatives and private cloud architectures.

The full deployable source code for this example is here: https://github.com/wesleyhales/HTML5-Mobile-WebSocket

A few things worth noting:

- Security: Since our WebSocket server is running on a different port (8081) than our JBoss AS7 server (8080), we must account for not having authentication cookies, etc. However, this problem can be handled with a reverse proxy as seen in the last section of this chapter.
- Proxies: As if existing proxy servers weren't already a huge problem for running WebSockets and HTTP over the same port, in this example we are now running them separately.
- Threading: Since we're observing/listening for CDI events, we must perform some same thread operations and connection sharing.

First, we setup the WebSocket server using Jetty's WebSocketHandler and embedding it inside a ServletContextListener. Here we're sharing a synchronized set of WebSocket connections across threads. Using the synchronized keyword, we ensure that only a single thread can execute a method or block at one time. To relay the CDI event to the browser, we must store all the WebSocket connections in an ConcurrentHashSet. As new connections come online we’ll write to the Set. At any time the set will be Read on a different thread so we know where to relay our CDI events to. The ChatWebSocketHandler contains a global Set of webSocket connections and adds each new connection as it's made within the Jetty server.

```
public class ChatWebSocketHandler extends WebSocketHandler {
```

[java]

```
private static Set<ChatWebSocket> websockets = new
ConcurrentHashSet<ChatWebSocket>();

    public WebSocket doWebSocketConnect(HttpServletRequest request,
            String protocol) {
        return new ChatWebSocket();
    }

    public class ChatWebSocket implements WebSocket.OnTextMessage {

        private Connection connection;

        public void onOpen(Connection connection) {
            // Client (Browser) WebSockets has opened a connection.
            // 1) Store the opened connection
            this.connection = connection;
            // 2) Add ChatWebSocket in the global list of ChatWebSocket
            // instances
            // instance.
            getWebsockets().add(this);
        }

        public void onMessage(String data) {
            // Loop for each instance of ChatWebSocket to send message server to
            // each client WebSockets.
            try {
                for (ChatWebSocket webSocket : getWebsockets()) {
                    // send a message to the current client WebSocket.
                    webSocket.connection.sendMessage(data);
                }
            } catch (IOException x) {
                // Error was detected, close the ChatWebSocket client side
                this.connection.disconnect();
            }

        }

        public void onClose(int closeCode, String message) {
            // Remove ChatWebSocket in the global list of ChatWebSocket
            // instance.
            getWebsockets().remove(this);
        }
    }

    public static synchronized Set<ChatWebSocket> getWebsockets() {
        return websockets;
    }

}
```

Next we embed the Jetty WebSocket capable server within our web application:

[java]

```
private Server server = null;
    /**
     * Start Embedding Jetty server when WEB Application is started.
     *
     */
    public void contextInitialized(ServletContextEvent event) {
        try {
            // 1) Create a Jetty server with the 8081 port.
            InetAddress addr = InetAddress.getLocalHost();
            this.server = new Server();
            Connector connector = new SelectChannelConnector();
            connector.setPort(8081);
            connector.setHost(addr.getHostAddress());

            server.addConnector(connector);

            // 2) Register ChatWebSocketHandler in the Jetty server instance.
            ChatWebSocketHandler chatWebSocketHandler = new ChatWebSocketHandler();
            chatWebSocketHandler.setHandler(new DefaultHandler());

            server.setHandler(chatWebSocketHandler);

            // 2) Start the Jetty server.
            server.start();
        } catch (Throwable e) {
            e.printStackTrace();
        }
    }

....
}
```

Now we'll create a method to observe CDI events and send the fired "Member" events to all active connections. This relays a very simple "cdievent" JavaScript object, which will be pushed to all connected clients and then evaluated on the browser through a JavaScript interpreter.

[java]

```
public void observeItemEvent(@Observes Member member) {
        try {
            for (ChatWebSocket webSocket : websockets) {

webSocket.connection.sendMessage("{\"cdievent\":{\"fire\":function(){" +
                    "eventObj.initEvent(\'memberEvent\', true, true);" +
                    "eventObj.name = '" +  member.getName() + "';\n" +
                    "document.dispatchEvent(eventObj);" +
                    "}}}");
            }
        } catch (IOException x) {
            //…
        }
    }
```

The above code will observe the following event when a new Member is registered through the web interface. As you can see below, memberEventSrc.fire(member) is fired when a user registers through our provided RESTful URL.

[java]

```
@POST
@Consumes(MediaType.APPLICATION_FORM_URLENCODED)
```

```
@Produces(MediaType.APPLICATION_JSON)
public Response createMember(@FormParam("name") String name, @FormParam("email")
String email, @FormParam("phoneNumber") String phone) {
     ...

      //Create a new member class from fields
      Member member = new Member();
      member.setName(name);
      member.setEmail(email);
      member.setPhoneNumber(phone);

      try {

         //Fire the CDI event
         memberEventSrc.fire(member);
```

Finally, we setup our WebSocket JavaScript client and safely avoid using the eval() method to execute the received JavaScript.

[javascript]

```
        ...
        var location = "ws://192.168.1.101:8081/"
        this._ws = new WebSocket(location);
        ....
        _onmessage : function(m) {
            if (m.data) {
                //check to see if this message is a CDI event
                if(m.data.indexOf('cdievent') > 0){
                    try{
                        //$('log').innerHTML = m.data;
                        //avoid use of eval...
                        var event = (m.data);
                        event = (new Function("return " + event))();
                        event.cdievent.fire();
                    }catch(e){
                        alert(e);
                    }
                }else{
                    //... append data in the DOM
                }
            }
        },
```

Here is the JavaScript code that listens for our CDI event, and executes the necessary client side code. (This is the alert popup seen in the video above.)

[javascript]

```
window.addEventListener('memberEvent', function(e) {
    alert(e.name + ' just registered!');
}, false);
```

As you can see, this is a very prototyped approach to achieve a running WebSocket server, but it's a step forward in adding a usable programming layer on top of the WebSocket protocol.

Binary Data over WebSockets with node.js

Another cool use of WebSockets is the ability to use binary data instead of just JSON strings.

[javascript]

```
objWebSocket.onopen = function(evt)
{
   var array = new Float32Array(5);
   for (var i = 0; i < array.length; ++i) array[i] = i / 2;
   ws.send(array, {binary: true});
};
```

Why send binary data? This allows us to stream audio to connected clients using the Web Audio API. Or you could give users the ability to collaborate with a realtime screen sharing application using canvas and avoid the need to base64 encode the images. The possibilities are limitless!

The following code sets up a node.js server to demo an example of sending audio over a WebSocket connection. See https://github.com/einaros/ws-audio-example for full example.

[javascript]

```
var express = require('express');
var WebSocketServer = require('ws').Server;
var app = express.createServer();

function getSoundBuffer(samples) {
  // header yanked from
  // http://html5-demos.appspot.com/static/html5-whats-new/template/index.html#30
  var header = new Buffer([
      0x52,0x49,0x46,0x46, // "RIFF"
      0, 0, 0, 0,          // put total size here
      0x57,0x41,0x56,0x45, // "WAVE"
      0x66,0x6d,0x74,0x20, // "fmt "
      16,0,0,0,            // size of the following
      1, 0,                // PCM format
      1, 0,                // Mono: 1 channel
      0x44,0xAC,0,0,       // 44,100 samples per second
      0x88,0x58,0x01,0,    // byte rate: two bytes per sample
      2, 0,                // aligned on every two bytes
      16, 0,               // 16 bits per sample
      0x64,0x61,0x74,0x61, // "data"
      0, 0, 0, 0           // put number of samples here
  ]);
  header.writeUInt32LE(36 + samples.length, 4, true);
  header.writeUInt32LE(samples.length, 40, true);
  var data = new Buffer(header.length + samples.length);
  header.copy(data);
  samples.copy(data, header.length);
  return data;
}

function makeSamples(frequency, duration) {
  var samplespercycle = 44100 / frequency;
  var samples = new Uint16Array(44100 * duration);
  var da = 2 * Math.PI / samplespercycle;
  for (var i = 0, a = 0; i < samples.length; i++, a += da) {
    samples[i] = Math.floor(Math.sin(a / 300000) * 32768);
  }
  return getSoundBuffer(new Buffer(Array.prototype.slice.call(samples, 0)));
}

app.use(express.static(__dirname + '/public'));
app.listen(8080);
var wss = new WebSocketServer({server: app, path: '/data'});

var samples = makeSamples(20000, 10);

wss.on('connection', function(ws) {
  ws.on('message', function(message) {
    ws.send('pong');
  });
  ws.send(samples, {binary: true});
});
```

Managing Proxies

With new technology comes a new set of problems. In the case of WebSockets, it's the compatibility with proxy servers that mediate HTTP connections in most company networks.

There's always going to be a firewall, proxy server, or switch that is the lynchpin of the enterprise. These devices and servers limit the kind of traffic we're allowed to send to and from the server.

The WebSocket protocol uses the HTTP upgrade system (which is normally used for HTTPS/SSL) to "upgrade" a HTTP connection to a WebSocket connection. Some proxy servers are not able to handle this handshake and will drop the connection. So, even if a given client uses the WebSocket protocol, it may not be possible to establish a connection.

> When using WebSocket Secure (wss://), the wire traffic is encrypted and intermediate transparent proxy servers may simply allow the encrypted traffic through, so there is a much better chance that the WebSocket connection will succeed. Using encryption is not free of resource cost, but often provides the highest success rate. We'll see an example of wss:// in ch x00.

Some proxy servers are harmless and work fine with WebSockets. Others will prevent WebSockets from working correctly, causing the connection to fail. In some cases additional proxy server configuration may be required, and certain proxy servers may need to be upgraded to support WebSocket connections.

If unencrypted WebSocket traffic flows through an explicit or a transparent proxy server on its way to the WebSocket server, then, whether or not the proxy server behaves as it should, the connection is almost certainly bound to fail. Therefore, unencrypted WebSocket connections should be used only in the simplest topologies. As WebSockets become more mainstream, proxy servers will become WebSocket aware.

If an encrypted WebSocket connection is used, then the use of Transport Layer Security (TLS) in the WebSocket Secure connection ensures that an HTTP CONNECT command is issued when the browser is configured to use an explicit proxy server. This sets up a tunnel, which provides low-level end-to-end TCP communication through the HTTP proxy, between the WebSocket Secure client and the WebSocket server. In the case of transparent proxy servers, the browser is unaware of the proxy server, so no HTTP CONNECT is sent. However, since the wire traffic is encrypted, intermediate transparent proxy servers may simply allow the encrypted traffic through, so there is a much better chance that the WebSocket connection will succeed if WebSocket Secure is used. Using encryption is not free of resource cost, but often provides the highest success rate.

> A mid-2010 draft (version hixie-76) broke compatibility with reverse-proxies and gateways by including 8 bytes of key data after the headers, but not advertising that data in a Content-Length: 8 header.[11] This data was not forwarded by all intermediates, which could lead to protocol failure. More recent drafts (e.g., hybi-09[12]) put the key data in a Sec-WebSocket-Key header, solving this problem.

Building your own

Things have changed since the days of fronting our servers with Apache for tasks like static resource serving. Apache configuration changes result in killing hundreds of active connections, which in turn, kills service availability.

With today's private cloud architectures, there is a high demand for throughput and availability. If we want our services like Apache or Tomcat to come up or go down at any time, then we simply have to put something in front of those services that can handle routing the traffic correctly, based on the cloud topology at the moment. One way to take down servers and bring new ones up without affecting service availability is to use a proxy. In most cases, HAProxy is the go-to-choice for high throughput and availability.

HAProxy is a lightweight proxy server that advertises obscenely high throughput. Companies such as GitHub, Fedora, Stack Overflow, and Twitter all use HAProxy for load balancing and scaling their infrastructure. Not only will HAProxy handle HTTP traffic, but it's a general-purpose TCP/IP proxy, and best of all, dead-simple to use.

Below, we've taken our example from the last section and added HAProxy. This gives us a reverse-proxy on the WebSocket port (8081), in the end allowing all traffic (HTTP and WS) to be sent across a common port — 8080 in this case.

Here is a simple reverse proxy from our example WebSocket server:

[config]

```
global
    maxconn     4096 # Total Max Connections. This is dependent on ulimit
    nbproc      1

defaults
    mode        http

frontend all 0.0.0.0:8080
    timeout client 86400000
    default_backend www_backend
    acl is_websocket hdr(Upgrade) -i WebSocket
    acl is_websocket hdr_beg(Host) -i ws
    use_backend socket_backend if is_websocket

backend www_backend
    balance roundrobin
    option forwardfor # This sets X-Forwarded-For
    timeout server 30000
    timeout connect 4000
    server apiserver 192.168.1.101:8080 weight 1 maxconn 4096 check

backend socket_backend
    balance roundrobin
    option forwardfor # This sets X-Forwarded-For
    timeout queue 5000
    timeout server 86400000
    timeout connect 86400000
    server apiserver 192.168.1.101:8081 weight 1 maxconn 4096 check
```

This approach is universal to any HTTP server which embeds a separate WebSocket server on a different port.