

Table of Contents (Summary)

	Intro	xix
1	Using Ajax: <i>Web Apps for a New Generation</i>	1
2	Designing Ajax Applications: <i>Thinking Ajaxian</i>	43
3	Javascript Events: <i>Reacting to your users</i>	93
4	Multiple Event Handlers: <i>Two's Company</i>	139
5	Asynchronous Applications: <i>It's Like Renewing Your Driver's License</i>	173
6	The Document Object Model: <i>Web Page Forestry</i>	229
7	Manipulating the DOM: <i>My Wish is Your Command</i>	283
8	Frameworks and Toolkits: <i>Trust No One</i>	329
9	XML Requests and Responses: <i>More Than Words Can Say</i>	343
10	JSON: <i>SON of JavaScript</i>	379
11	Forms and Validation: <i>Say What You Meant to Say</i>	407
12	Post Requests: <i>Paranoia: It's Your Friend</i>	445
i	Appendix i: <i>Top Five Topics We Didn't Cover</i>	471
ii	Appendix ii: <i>Utility Functions</i>	483

Table of Contents (the real thing)

Intro

Your brain on Ajax. Here you are trying to *learn* something, while here your *brain* is doing you a favor by making sure the learning doesn't *stick*. Your brain's thinking, "Better leave room for more important things, like which wild animals to avoid and whether naked snowboarding is a bad idea." So how *do* you trick your brain into thinking that your life depends on knowing Ajax?

Who is this book for?	xx
We know what you're thinking	xxi
Metacognition	xxiii
Bend your brain into submission	xxv
Read me	xxvi
The technical review team	xxviii
Acknowledgments	xxix

using ajax

1 Web Apps for a New Generation

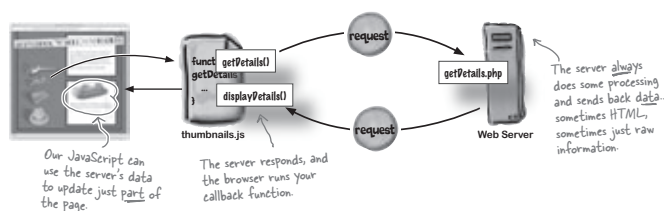
Tired of waiting around for your page to reload?

Frustrated by clunky web application interfaces? It's time to give your web apps that slick, responsive desktop feel. And how do you do that? With **Ajax**, your ticket to building Internet applications that are *more interactive*, *more responsive*, and *easier to use*. So skip your nap; it's time to put some polish on your web apps. It's time to get rid of unnecessary and slow full-page refreshes forever.

Web pages: the old-fashioned approach	2
Web pages reinvented	3
So what makes a page “Ajax” ?	5
Rob’s Rock ‘n’ Roll Memorabilia	6
Ajax and rock ‘n’ roll in 5 steps	12
Step 1: Modify the XHTML	14
Step 2: Initialize the JavaScript	16
Step 3: Create a request object	20
Step 4: Get the item’s details	22
Let’s write the code for requesting an item’s details	24
Always make sure you have a request object before working with it	25
The request object is just an object	26
Hey, server... will you call me back at displayDetails(), please?	27
Use send() to send your request	28
The server usually returns data to Ajax requests	30
Ajax is server-agnostic	31
Use a callback function to work with data the server returns	35
Get the server’s response from the request object’s responseText property	36
Goodbye traditional web apps...	38

I'm desperate... but I can't afford a more powerful server or a team of web experts.

Ajax pages only talk to the server when they have to... and only about what the server knows.



designing ajax applications

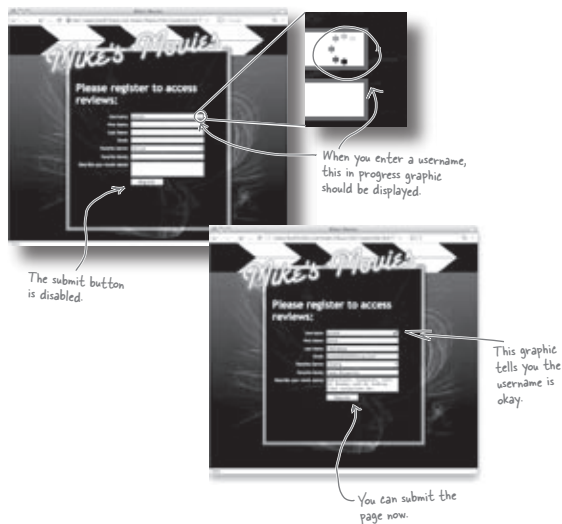
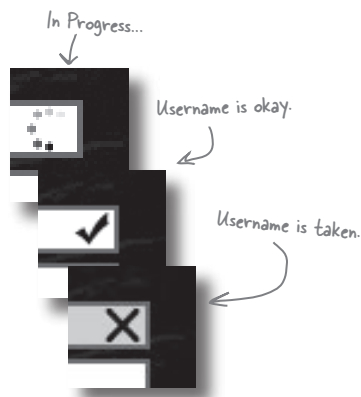
Thinking Ajaxian

2

Welcome to Ajax apps—it's a whole new web world.

So you've built your first Ajax app, and you're already thinking about how to change all your web apps to make requests asynchronously. But that's not all there is to Ajax programming. You've got to **think about your applications differently**. Just because you're making asynchronous requests, doesn't mean your application is user-friendly. It's up to you to help your users **avoid making mistakes**, and that means **rethinking** your entire application's **design**.

Mike's traditional web site sucks	44
Let's use Ajax to send registration requests ASYNCHRONOUSLY	46
Update the registration page	51
Event Handlers Exposed	53
Set the window.onload event handler... PROGRAMMATICALLY	54
Code in your JavaScript outside of functions runs when the script is read	56
What happens when...	57
And on the server...	58
Some parts of your Ajax designs will be the same... every time	60
createRequest() is always the same	61
Create a request object... on multiple browsers	64
Ajax app design involves both the web page AND the server-side program	66
The request object connects your code to the web browser	72
You talk to the browser, not the server	73
The browser calls back your function with the server's response	76
Show the Ajax registration page to Mike...	78
The web form has TWO ways to send requests to the server now	79
Let's create CSS classes for each state of the processing...	82
...and change the CSS class with our JavaScript	83
Changes? We don't need no stinkin' changes!	84
Only allow registration when it's appropriate	85



javascript events

Reacting to your users

3

Sometimes you need your code to react to other things going on in your web application... and that's where **events** come into the picture.

An event is *something that happens* on your page, in the browser, or even on a web server. But it's not enough to just know about events... sometimes you want to respond to them. By creating code, and registering it as an **event handler**, you can get the browser to run your handler every time a particular event occurs. Combine events and handlers, and you get **interactive web applications**.

It all started with a downward-facing dog...	94
Ajax apps are more than the sum of their parts	101
Here's Marcy's XHTML...	102
Events are the key to interactivity	104
Connect events on your web page to event handlers in your JavaScript	107
Use window.onload event to initialize the interactivity on a web page	108
Change those left-side images to be clickable	113
Use your XHTML's content and structure	114
Add the code for hideHint(), too	117
Tabs: an optical (and graphical) illusion	118
Use a for... loop to cycle through the images	119
CSS classes are the key (again)	120
Ummm... but the tabs aren't <a>'s!	121
This broke our JavaScript, too, didn't it?	122
Use a request object to fetch the class details from the server	127
Two functions changing the same part of a web page	128
When you need to change images in your script, think "change CSS classes"	133
Links in XHTML are represented by <a> elements	134
We need a function to show an active button and hide a button, too	135



multiple event handlers

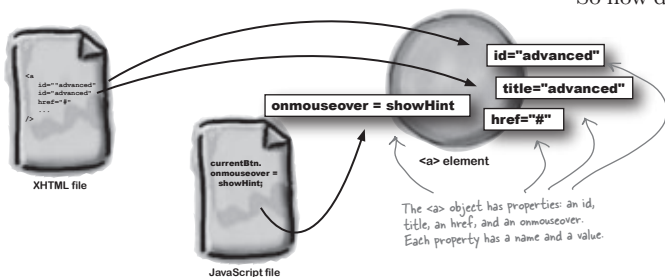
4

Two's company

A single event handler isn't always enough.

Sometimes you've got more than one event handler that needs to be called by an event. Maybe you've got some event-specific actions, as well as some generic code, and stuffing everything into a single event handler function won't cut it. Or maybe you're just trying to build **clean, reusable code**, and you've got **two bits of functionality** triggered by the **same event**. Fortunately, we can use some **DOM Level 2** methods to assign multiple handler functions to a **single event**.

An event can have only one event handler attached to it (or so it seems)	140
Event handlers are just properties	141
A property can have only ONE value	141
Assign multiple event handlers with <code>addEventListener()</code>	142
Objects can have multiple event handlers assigned to a single event in DOM Level 2	144
What's going on with Internet Explorer?	148
Internet Explorer uses a totally different event model	149
<code>attachEvent()</code> and <code>addEventListener()</code> are functionally equivalent	149
<code>addEventListener()</code> works for ALL apps, not just Marcy's yoga page	154
Let's update <code>initPage()</code> to use our new utility function	155
Use an <code>alert()</code> to troubleshoot	157
So what else could be going wrong?	157
Event handlers in IE are owned by IE's event framework	159
<code>attachEvent()</code> and <code>addEventListener()</code> supply another argument to our handlers	160
We need to name the Event argument, so our handlers can work with it	161
You say target, I say <code>srcElement</code> ...	162
So how do we actually GET the object that triggered the event?	166



asynchronous applications

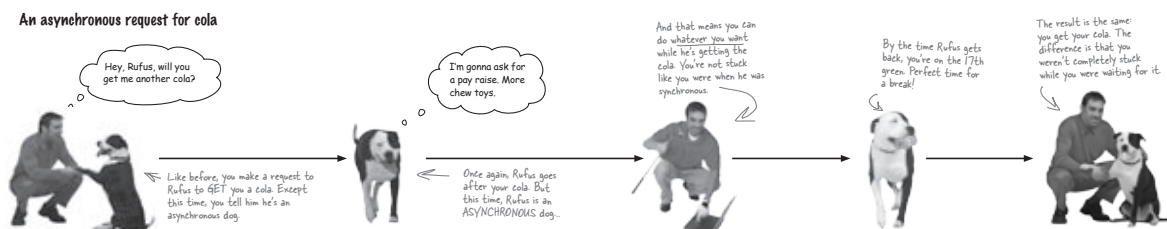
5

It's like renewing your driver's license

Are you tired of waiting around? Do you hate long delays?
You can do something about it with asynchrony!

You've already built a couple of pages that made asynchronous requests to the server to avoid making the user sit around waiting for a page refresh. In this chapter, we'll dive even deeper into the details of building asynchronous applications. You'll find out what **asynchronous really means**, learn how to use **multiple asynchronous requests**, and even build a **monitor function** to keep all that asynchrony from confusing you and your users.

What does asynchronous really mean?	174
You've been building asynchronous apps all along	176
But sometimes you barely even notice...	177
Speaking of more server-side processing..	178
(More) Asynchrony in 3 easy steps	181
We need two password fields and a <div> for the cover images	182
If you need new behavior, you probably need a new event handler function	187
With ONE request object, send and receive ONE asynchronous request	196
Asynchronous requests don't wait on anything.. including themselves!	197
If you're making TWO separate requests, use TWO separate request objects	198
You can't count on the ORDERING of your requests and responses	204
A monitor function monitors your application from OUTSIDE the action	209
You call a monitor function when action MIGHT need to be taken	210
Status variables let monitors know what's going on	212
And now for our last trick...	216
Synchronous requests block ALL YOUR CODE from doing anything	218
Use setInterval() to let JavaScript run your process, instead of your own code	221



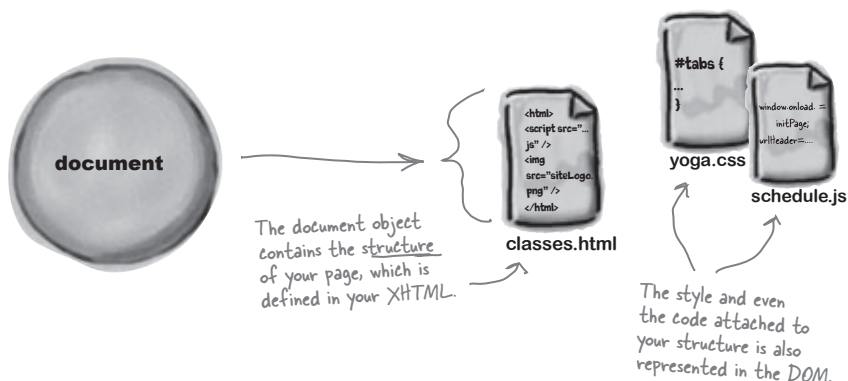
the document object model

6

Web Page Forestry

Wanted: easy-to-update web pages. It's time to take things into your own hands and start writing code that updates your web pages on the fly. Using the **Document Object Model**, your pages can take on new life, responding to users' actions, and you can ditch unnecessary page reloads forever. By the time you've finished this chapter, you'll be able to find, move, and update content virtually anywhere on your web page.

You can change the CONTENT of a page...	230
...or you can change the STRUCTURE of a page	231
Browsers use the Document Object Model to represent your page	232
Here's the XHTML that you write...	234
...and here's what your browser sees	235
Your page is a set of related objects	237
Let's use the DOM to build a dynamic app	244
You start with XHTML...	246
appendChild() adds a new child to a node	255
You can locate elements by name or by id	256
Interview with a new parent	259
Can I move the clicked tile?	260
You can move around a DOM tree using FAMILY relationships	262
A DOM tree has nodes for EVERYTHING in your web page	272
The nodeName of a text node is "#text"	274
Did I win? Did I win?	278
But seriously... did I win?	279

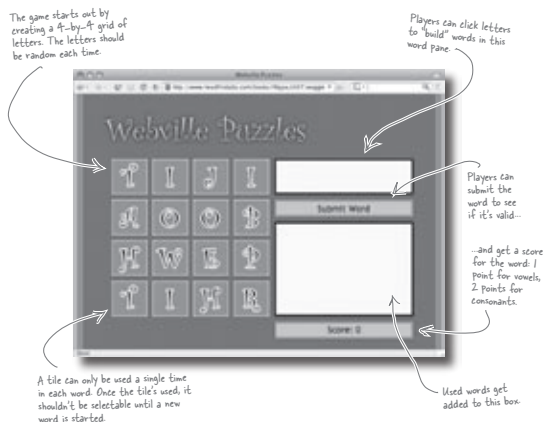


manipulating the DOM

My wish is your command

Sometimes you just need a little ^{DOM} ~~mind~~ control.

It's great to know that web browsers turn your XHTML into DOM trees. And you can do a lot by moving around within those trees. But real power is **taking control of a DOM tree** and making the tree look like *you* want it to. Sometimes what you really need is to **add a new element** and some text, or to **remove an element**, like an ``, from a page altogether. You can do all of that and more with the DOM, and along the way, **banish that troublesome innerHTML property** altogether. The result? Code that can do more to a page, *without* having to mix presentation and structure in with your JavaScript.



Webville Puzzles... the franchise	284
Woggle doesn't use table cells for the tiles	288
The tiles in the XHTML are CSS-positioned	289
"We don't want TOTALLY random letters..."	291
Our presentation is ALL in our CSS	293
We need a new event handler for handling tile clicks	295
Start building the event handler for each tile click	296
We can assign an event handler in our randomizeTiles() function	296
Property values are just strings in JavaScript	297
We need to add content AND structure to the "currentWord" div	300
Use the DOM to change a page's structure	300
Use createElement() to create a DOM element	301
You have to TELL the browser where to put any new DOM nodes you create	302
We need to disable each tile. That means changing the tile's CSS class...	310
...AND turning OFF the addLetter() event handler	310
Submitting a word is just (another) request	312
Our JavaScript doesn't care how the server figures out its response to our request	312
Usability check: WHEN can submitWord() get called?	317

frameworks and toolkits

8

Trust No One

So what's the *real* story behind all those Ajax frameworks?

If you've been in Webville awhile, you've probably run across at least one JavaScript or Ajax framework. Some frameworks give you **convenience methods for working with the DOM**. Others make **validation** and **sending requests** simple. Still others come with libraries of pre-packaged JavaScript **screen effects**. But which one should you use? And how do you know what's really going on inside that framework? It's time to do more than use other people's code... it's time to **take control of your applications**.

Reasons to use a framework

Reasons NOT to use a framework

So what frameworks ARE there?	335
Every framework uses a different syntax to do things	336
The syntax may change... but the JavaScript is still the same	337
To framework or not to framework?	340
The choice is up to you...	342

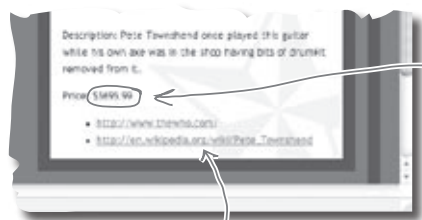
xml requests and responses

9

More Than Words Can Say

How will you describe yourself in 10 years? How about 20?

Sometimes you need **data that can change with your needs...** or the needs of your customers. Data you're using now might need to change in a few hours, or a few days, or a few months. With XML, the *extensible markup language*, your data can **describe itself**. That means your scripts won't be filled with ifs, elses, and switches. Instead, you can use the descriptions that XML provides about itself to figure out how to **use** the data the XML contains. The result: **more flexibility** and **easier data handling**.



Rob wants to add a price for each item.

Each item will have one or more URLs to find out more about the item.

Classic rock gets a 21st century makeover	344
How should a server send a MULTI-valued response?	347
innerHTML is only simple for the CLIENT side of a web app	353
You use the DOM to work with XML, just like you did with XHTML	359
XML is self-describing	366

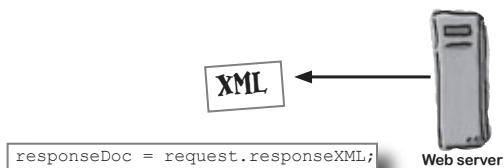
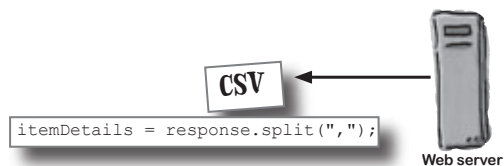
10

JSON

SON of JavaScript**JavaScript, objects, and notation, oh my!**

If you ever need to represent objects in your JavaScript, then you're going to love JSON, **JavaScript Standard Object Notation**. With JSON, you can **represent complex objects and mappings** with text and a few curly braces. Even better, you can **send and receive JSON** from other languages, like PHP, C#, Python, and Ruby.

JSON can be text AND an object	381
JSON data can be treated as a JavaScript object	382
So how do we get JSON data from the server's response?	383
JavaScript can evaluate textual data	385
Use <code>eval()</code> to manually evaluate text	385
Evaluating JSON data returns an object representation of that data	386
JavaScript objects are already dynamic because they're not compiled objects	392
Access an object's members, then get an object's values with those members	393
You need to PARSE the server's response, not just EVALUATE it	399

JSON can be text AND an object

11

forms and validation

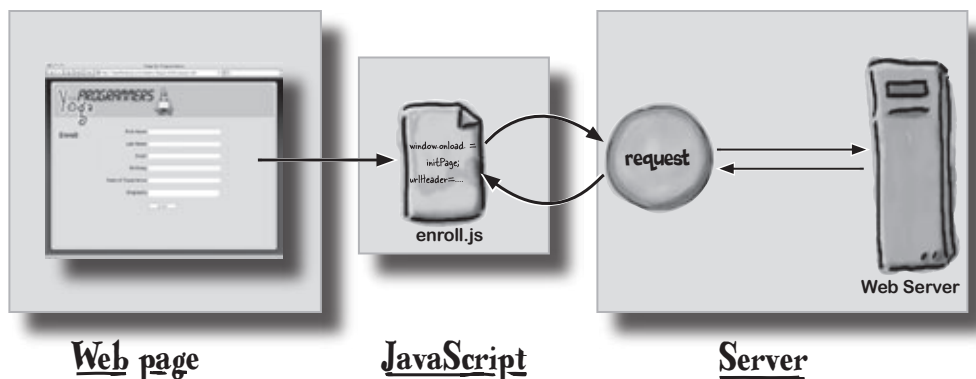
Say what you mean to say

Everyone makes mistakes from time to time.

Give a human being a chance to talk (or type) for a few minutes, and they'll probably make at least one or two **mistakes**. So how do your web apps **respond to those mistakes**?

You've got to **validate** your users' input and react when that input has problems. But who does what? What should your web page do? What should your JavaScript do? And what's the role of the server in **validation** and **data integrity**?

Validation should work from the web page BACK to the server	414
You can validate the FORMAT and CONTENT of data	420
Don't Repeat Yourself: DRY	423
Let's build some more event handlers	426
RETURN of SON of JavaScript	430
The value of a property can be another JavaScript object	430
Let's warn Marcy's customers when there's a problem with their entry	433
If you don't warn(), you have to unwarn()	437
IF there's a warning, get rid of it	437
Duplicate data is a SERVER problem	443



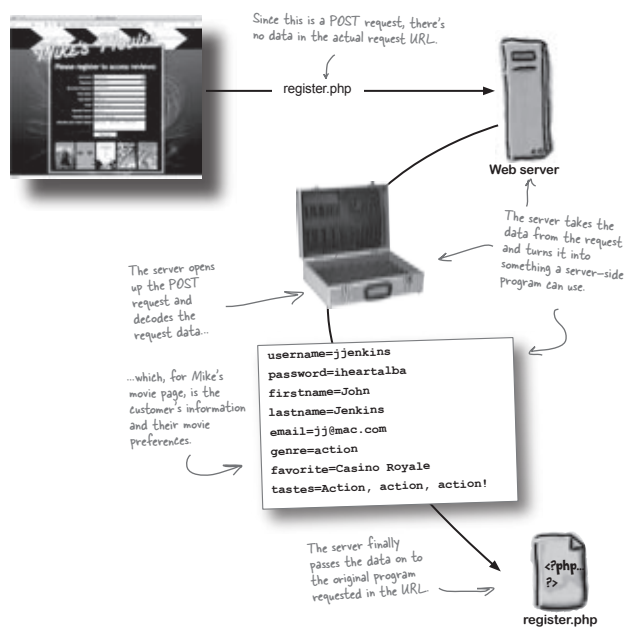
12

post requests

Paranoia: It's your friend**Someone's watching you. Right now. Seriously.**

Freedom of Information Act? Isn't that called the Internet? These days, anything a user types into a form or clicks on a web page is subject to **inspection**. Whether it's a network admin, a software company trying to learn about your trends, or a malicious hacker or spammer, your **information isn't safe unless you make it safe**. When it comes to web pages, you've got to **protect your users' data** when they click Submit.

GET requests send request parameters across the network as clear text	449
POST requests DON'T send clear text	450
The data in a POST request is ENCODED until it reaches the server	452
send() your request data in a POST request	454
Always check to make sure your request data was RECEIVED.	456
Why didn't the POST request work?	458
The server unencodes POST data	459
We need to TELL the server what we're sending	460
Set a request header using setRequestHeader() on your request object	462



leftovers



The Top Five Topics (we didn't cover)

It's been a long ride... and you're almost to the end.

We can barely stand to let you go, but before you do, there's still a few things left to cover. We can't possibly fit everything about Ajax into one 600-page book. So we threw out everything you didn't really need to know and kept the last few important bits in this appendix.



#1 Inspecting the DOM	472
#2 Graceful degradation	475
#3 script.aculo.us and the Yahoo UI libraries	476
#4 Using JSON libraries in your PHP code	478
#5 Ajax and ASP.NET	480

utility functions



Just Gimme the Code

Sometimes you just want everything in one place.

You've spent a lot of time using `utils.js`, our little utility class of Ajax, DOM, and event utility functions. Inside these pages, you'll get all those functions in one place, ready to put into your own utility scripts and applications.

<code>utils.js</code> : a work in progress	484
--------------------------------------------	-----