Join the discussion @ p2p.wrox.com

Wrox Programmer to Programmer™

Professional AngularJS

Valeri Karpov, Diego Netto

PROFESSIONAL ANGULARJS

PROFESSIONAL

AngularJS

PROFESSIONAL

AngularJS

Valeri Karpov
Diego Netto

wrox™
A Wiley Brand

Professional AngularJS

Published by
John Wiley & Sons, Inc.
10475 Crosspoint Boulevard
Indianapolis, IN 46256
www.wiley.com

Published simultaneously in Canada

ISBN: 978-1-118-83207-3

ISBN: 978-1-118-83209-7 (ebk)

ISBN: 978-1-118-83208-0 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Library of Congress Control Number: 2014951014

For my father, the elder Valeri Karpov,
who taught me to never settle for "good enough."

—Valeri Karpov

For my mother, Liliana, who showed me how to find
happiness by living each day like it might be your last.

—Diego Netto

ABOUT THE AUTHORS

VALERI KARPOV is a NodeJS Engineer at MongoDB, where he focuses on maintaining the popular Mongoose ODM and numerous other MongoDB-related NodeJS modules. In addition, he's a Hacker in Residence at BookaLokal, a blogger for StrongLoop, and the person who gave the MEAN stack its name. He has been running production AngularJS apps since AngularJS v0.9.4 in 2010. Most recently, he used AngularJS to build out BookaLokal's mobile website and a web client for MongoDB's internal continuous integration framework.

DIEGO NETTO is a software consultant and open source evangelist. He wears the many hats of a full stack engineer and entrepreneur. Owner of a development shop operating out of Los Angeles and Dallas, Diego creates web and mobile applications for both startups and enterprise companies. Maintainer of the IonicFramework Yeoman generator, he has most recently used AngularJS and the IonicFramework to build the Prop mobile app for `www.aboatapp.com`, and is using Famo.us/Angular to build the mobile app for `www.modelrevolt.com`.

ABOUT THE TECHNICAL EDITOR

STÉPHANE BÉGAUDEAU graduated from the Faculty of Sciences and Technology of Nantes and is currently working as a web technology specialist and Eclipse modeling consultant at Obeo in France. He has contributed to several open source projects in the Eclipse Foundation, and he is the leader of Acceleo. He also worked on Dart Designer, an open source tooling for the Dart programming language.

CREDITS

PROJECT EDITOR
Kelly Talbot

TECHNICAL EDITOR
Stéphane Bégaudeau

PRODUCTION EDITOR
Christine O'Connor

COPY EDITOR
Karen Gill

MANAGER OF CONTENT DEVELOPMENT & ASSEMBLY
Mary Beth Wakefield

MARKETING DIRECTOR
David Mayhew

MARKETING MANAGER
Carrie Sherrill

PROFESSIONAL TECHNOLOGY & STRATEGY DIRECTOR
Barry Pruett

BUSINESS MANAGER
Amy Knies

ASSOCIATE PUBLISHER
Jim Minatel

PROJECT COORDINATOR, COVER
Brent Savage

PROOFREADER
Nancy Carrasco

INDEXER
Johnna VanHoose Dinse

COVER DESIGNER
Wiley

COVER IMAGE
iStock.com/Manuel Faba Ortega

ACKNOWLEDGMENTS

SOCRATES WROTE THAT "Education is the kindling of a flame, not the filling of a vessel." In that vein, I'm thankful that I have had teachers and mentors who made programming my lifelong passion rather than just a career. In particular, I'd like to thank Professors Kernighan and Tarjan from Princeton University; and Dr. Nevard, Dr. Sankaran, Mr. Scarpone, and Mr. Nodarse from the Bergen County Academies. In addition, I'd like to thank Misko Hevery, my mentor when I interned at Google and original author of AngularJS, who taught me more about software engineering in 12 weeks than I had learned in my life leading up to that summer.

—Valeri Karpov

TO QUOTE THE SUCCESSFUL ENTREPRENEUR FELIX DENNIS, "Anyone not busy learning is busy dying." This is a prudent reminder, especially in the constantly evolving field of software engineering, that in order to remain relevant and sustain success we must commit to a lifelong pursuit of knowledge. I would like to thank Professors Tatar and Ribbens from Virginia Tech for enlightening me to this realization. I also want to thank Addy Osmani for helping me discover the importance of intelligent tooling and for inspiring me to contribute to the open source community. Special thanks to my friend and co-author Valeri Karpov for getting me involved with AngularJS during such an early stage.

—Diego Netto

CONTENTS

INTRODUCTION

IT'S AN EXCITING TIME to be a JavaScript developer. Between the meteoric rise of server-side JavaScript's open source community (100,000 packages on the NodeJS package manager as of October 2014—twice as many as in December 2013), the popularity of next-generation client-side frameworks like AngularJS, and the growing number of companies that build web tools based on full-stack JavaScript, JavaScript language skills are in high demand. Modern tools allow you to build sophisticated browser-based clients, highly concurrent servers, and even hybrid native mobile applications using a single language. AngularJS is quickly becoming the leading next-generation client-side web framework, enabling individuals, small teams, and large corporations to build and test phenomenally sophisticated browser-based applications.

WHAT IS ANGULARJS?

Within the rapidly growing JavaScript community, AngularJS burst onto the scene when it released version 1.0 in June 2012. Although a relatively new framework, its powerful features and elegant tools for structuring applications have made it the front-end framework of choice for many developers. AngularJS was originally developed at Google by testing engineer Misko Hevery, who found that existing tools, like jQuery, made it difficult to structure browser user interfaces (UIs) that needed to display large amounts of sophisticated data. Google now has a dedicated team developing and maintaining AngularJS and related tools. AngularJS also powers some active Google applications, ranging from the DoubleClick Digital Marketing Platform to the YouTube app on the PlayStation 3. AngularJS's popularity is growing rapidly: As of October 2014, it powers 143 of the Quantcast Top 10k websites and is rapidly outpacing its closest rivals, KnockoutJS, ReactJS, and EmberJS.

What makes AngularJS so special? One particularly pithy expression borrowed from the `https://angularjs.org/` website describes AngularJS as enabling you to "write less code, go have beer sooner." The heart of AngularJS is a concept called *two-way data binding*, which enables you to bind Hypertext Markup Language (HTML) and cascading style sheets (CSS) to the state of a JavaScript variable. Whenever the variable changes, AngularJS updates all HTML and CSS that references that JavaScript variable. For instance, in the following code:

```
<div ng-show="shouldShow">Hello</div>
```

If the `shouldShow` variable is changed to `false`, AngularJS automatically hides the `div` element for you. There is nothing special about the `shouldShow` variable: AngularJS doesn't require you to wrap your variables in special types; the `shouldShow` variable can be a plain old JavaScript Boolean value.

Although two-way data binding is the basis for what makes AngularJS so useful, it's only the tip of the iceberg. AngularJS provides an elegant framework for organizing your client-side JavaScript in a way to maximize reusability and testability. In addition, AngularJS has a rich set of testing tools, such as Karma, protractor, and ngScenario (see Chapter 9), which are optimized for use with AngularJS. AngularJS's focus on testable structures and rich testing tools makes it a natural

choice for mission-critical client-side JavaScript. Not only does it enable you to write sophisticated applications fast, it supplies tools and structure that make testing your application easy. As a matter of fact, Google's DoubleClick team cited AngularJS's "full testing story" as one of its six biggest reasons for porting its digital marketing platform to AngularJS. Here is a brief overview of some of the concepts that make AngularJS special.

Two-Way Data Binding

In many older client-side JavaScript libraries, like jQuery and Backbone, you are expected to manipulate the Document Object Model (DOM) yourself. In other words, if you want to change the HTML contents of a `div` element, you need to write imperative JavaScript. For example:

```
$('div').html('Hello, world!');
```

AngularJS inverts this paradigm and makes your HTML the definitive source for how your data is displayed. The primary purpose of two-way data binding is to bind an HTML or CSS property (for instance, the HTML contents or background color of a `div` element) to the value of a JavaScript variable. When the value of the JavaScript variable changes, the HTML or CSS property is updated to match. The opposite is also true: If the user types in an `input` field, the value of the bound JavaScript variable is updated to match what the user typed. For instance, the following HTML greets whoever's name is typed in the input field. You can find this example in this chapter's sample code as `data_binding.html`: Simply right-click on the file and open it in your browser—no web server or other dependencies required!

```
<input type="text" ng-model="user" placeholder="Your Name">
<h3>Hello, {{user}}!</h3>
```

No JavaScript is necessary! The `ngModel` directive and the {{}} shorthand syntax do all the work. There is limited benefit to using AngularJS in this simple example, but, as you'll see when you build a real application in Chapter 1, data binding greatly simplifies your JavaScript. It's not uncommon to see 800 lines of jQuery spaghetti code reduced to 40 lines of clean DOM-independent AngularJS code thanks to data binding.

Scopes in the DOM

DOM scopes are another powerful feature of AngularJS. As you might have guessed, there is no free lunch with data binding; code complexity has to go somewhere. However, AngularJS allows you to create scopes in the DOM that behave similarly to scopes in JavaScript and other programming languages. This permits you to break your HTML and JavaScript into independent and reusable pieces. For instance, here's the same greeting example from earlier, but with two separate scopes: one for greeting in English, the other in Spanish:

```
<div ng-controller="HelloController">
  <input type="text" ng-model="user" placeholder="Your Name">
  <h3>Hello, {{user}}!</h3>
</div>
```

```
<hr>
<div ng-controller="HelloController">
  <input type="text" ng-model="user" placeholder="Su Nombre">
  <h3>Hola, {{user}}!</h3>
</div>

<script type="text/javascript"
        src="angular.js">
</script>
<script type="text/javascript">
  function HelloController($scope) {}
</script>
```

The `ngController` directive is one way to create a new scope, enabling you to reuse the same code for two different purposes. Chapter 4 includes a thorough overview of two-way data binding and a discussion of internal implementation details.

Directives

Directives are a powerful tool for grouping HTML and JavaScript functionality into one easily reusable bundle. AngularJS has numerous built-in directives, like the `ngController` and `ngModel` directives you saw earlier, that enable you to access sophisticated JavaScript functionality from your HTML. You can write your own custom directives as well. In particular, AngularJS allows you to associate HTML with a directive, so you can use directives as a way of reusing HTML as well as a way of tying certain behavior into two-way data binding. Writing custom directives is beyond the scope of this introduction, but Chapter 5 includes a thorough discussion of the subject.

Templates

On top of two-way data binding, AngularJS lets you swap out entire portions of the page based on the state of a JavaScript variable. The `ngInclude` directive enables you to conditionally include *templates*, pieces of AngularJS-infused HTML, in the page based on the JavaScript state. The following example demonstrates a page with a `div` that contains different HTML based on the value of the `myTemplate` variable. You can find this example in `templates.html` in this chapter's sample code:

```
<div ng-controller="TemplateController">
  <div ng-include="myTemplate">
  </div>
  <br>
  <a  ng-click="myTemplate = 'template1';"
      style="cursor: pointer"
      ng-class="{'selected': myTemplate === 'template1' }">
    Display Template 1
  </a>
  <a  ng-click="myTemplate = 'template2';"
      style="cursor: pointer"
      ng-class="{'selected': myTemplate === 'template2' }">
    Display Template 2
  </a>
```

```
</div>

<script type="text/javascript"
        src="angular.js">
</script>
<script type="text/javascript">
  function TemplateController($scope) {
    $scope.myTemplate = 'template1';
  }
</script>
<script type="text/ng-template" id="template1">
  <h1>This is Template 1</h1>
</script>
<script type="text/ng-template" id="template2">
  <h1>This is Template 2</h1>
</script>
```

Chapter 6 includes a thorough discussion of AngularJS templates, including how to use them to structure single-page applications.

Testing and Workflow

Providing a framework for writing unit-testable code has been a core AngularJS goal from its first release. AngularJS includes an elegant and sophisticated dependency injector, and all AngularJS *components* (controllers, directives, services, and filters) are constructed using the dependency injector. This ensures that your code's dependencies are easy to stub out as necessary for your tests. Furthermore, the AngularJS team has developed numerous powerful testing tools, such as the Karma test runner and the protractor and ngScenario integration testing frameworks. These bring the sophisticated multibrowser testing infrastructure that was previously only feasible for large companies into the hands of the individual developer.

In addition, AngularJS's architecture and testing tools interface nicely with various open source JavaScript build and workflow tools, such as Gulp and Grunt. With these tools, you can execute your tests seamlessly, tie in tools like code coverage and linting into your test execution, and even scaffold entirely new applications from scratch. Core AngularJS is just a library, but the testing and workflow tools surrounding it make the AngularJS ecosystem as a whole an innovative new paradigm for building browser-based clients. Chapter 9 includes a more detailed discussion of the AngularJS testing ecosystem and the different types of testing strategies you can use for your AngularJS applications.

WHEN NOT TO USE ANGULARJS

Like any library, AngularJS is a perfect fit for some applications and a not-so-good fit for others. In the next section, you learn about several use cases in which AngularJS is a perfect fit. In this section, you learn about a few use cases in which AngularJS is not such a good fit and learn about some of AngularJS's limitations.