



LEARNING ANGULARJS



BRAD DAYLEY

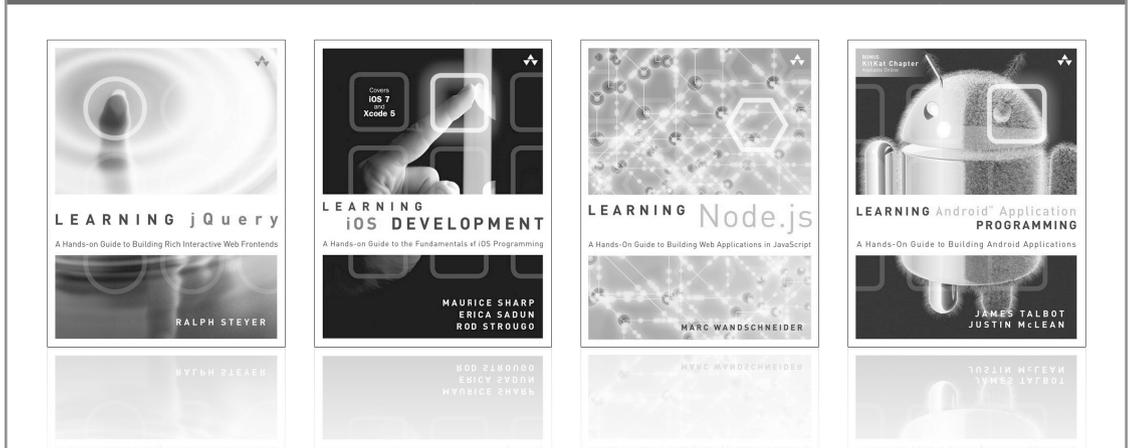
FREE SAMPLE CHAPTER



SHARE WITH OTHERS

Learning AngularJS

Addison-Wesley Learning Series



Visit informit.com/learningseries for a complete list of available publications.

The **Addison-Wesley Learning Series** is a collection of hands-on programming guides that help you quickly learn a new technology or language so you can apply what you've learned right away.

Each title comes with sample code for the application or applications built in the text. This code is fully annotated and can be reused in your own projects with no strings attached. Many chapters end with a series of exercises to encourage you to reexamine what you have just learned, and to tweak or adjust the code as a way of learning.

Titles in this series take a simple approach: they get you going right away and leave you with the ability to walk off and build your own application and apply the language or technology to whatever you are working on.

 Addison-Wesley

 **informIT**
The trusted technology learning source

 **Safari**
Books Online

Learning AngularJS

Brad Dayley

◆ Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Cape Town • Sydney • Tokyo • Singapore • Mexico City

Learning AngularJS

Copyright © 2015 by Pearson Education, Inc.

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

ISBN-13: 978-0-134-03454-6

ISBN-10: 0-134-03454-6

Library of Congress Control Number: 2014951593

Printed in the United States of America

First Printing: December 2014

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. The publisher cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an "as is" basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

Special Sales

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact international@pearsoned.com.

Acquisitions
Editor

Mark Taber

Managing Editor

Kristy Hart

Project Editor

Elaine Wiley

Copy Editor

Cheri Clark

Senior Indexer

Cheryl Lenser

Proofreader

Katie Matejka

Technical Editor

Jesse Smith

Editorial

Assistant

Vanessa Evans

Designer

Chuti

Prasertsith

Senior

Compositor

Gloria Schurick

Contents at a Glance

Introduction	1
1 Jumping Into JavaScript	5
2 Getting Started with AngularJS	35
3 Understanding AngularJS Application Dynamics	53
4 Implementing the Scope as a Data Model	65
5 Using AngularJS Templates to Create Views	77
6 Implementing Directives in AngularJS Views	99
7 Creating Your Own Custom Directives to Extend HTML	123
8 Using Events to Interact with Data in the Model	145
9 Implementing AngularJS Services in Web Applications	157
10 Creating Your Own Custom AngularJS Services	183
11 Creating Rich Web Application Components the AngularJS Way	199
A Testing AngularJS Applications	223
Index	233

Table of Contents

Introduction 1

- Who Should Read This Book 1
- Why You Should Read This Book 1
- What You Will Learn from This Book 2
- What Is AngularJS? 2
- How Is This Book Organized? 3
- Getting the Code Examples 4
- Finally 4

1 Jumping Into JavaScript 5

- Setting Up a JavaScript Development Environment Using Node.js 5
 - Setting Up Node.js 6
 - Using Node.js to Run JavaScript 7
 - Creating an Express Web Server Using Node.js 8
- Defining Variables 10
- Understanding JavaScript Data Types 11
- Using Operators 12
 - Arithmetic Operators 12
 - Assignment Operators 13
 - Applying Comparison and Conditional Operators 14
- Implementing Looping 16
 - while Loops 16
 - do/while Loops 17
 - for Loops 17
 - for/in Loops 18
 - Interrupting Loops 18
- Creating Functions 19
 - Defining Functions 19
 - Passing Variables to Functions 20
 - Returning Values from Functions 20
 - Using Anonymous Functions 21
- Understanding Variable Scope 22

Using JavaScript Objects	22
Using Object Syntax	23
Creating Custom Defined Objects	23
Using a Prototyping Object Pattern	24
Manipulating Strings	25
Combining Strings	27
Searching a String for a Substring	27
Replacing a Word in a String	27
Splitting a String into an Array	27
Working with Arrays	28
Combining Arrays	29
Iterating Through Arrays	30
Converting an Array into a String	30
Checking Whether an Array Contains an Item	30
Adding Items to and Removing Items from Arrays	30
Adding Error Handling	31
try/catch Blocks	31
Throwing Your Own Errors	32
Using Finally	32
Summary	33
2 Getting Started with AngularJS	35
Why AngularJS?	35
Understanding AngularJS	36
Modules	37
Scopes and the Data Model	37
Views with Templates and Directives	37
Expressions	38
Controllers	38
Data Binding	38
Services	38
Dependency Injection	38
Compiler	39
An Overview of the AngularJS Life Cycle	39
The Bootstrap Phase	39
The Compilation Phase	39
The Runtime Data Binding Phase	40

Separation of Responsibilities	40
Integrating AngularJS with Existing JavaScript and jQuery	40
Adding AngularJS to Your Environment	41
Bootstrapping AngularJS in an HTML Document	42
Using the Global APIs	42
Creating a Basic AngularJS Application	44
Loading the AngularJS Library and Your Main Module	45
Defining the AngularJS Application Root Element	45
Adding a Controller to the Template	46
Implementing the Scope Model	46
Using jQuery or jQuery Lite in AngularJS Applications	47
What Is jQuery Lite?	48
Accessing jQuery or jQuery Lite Directly	50
Accessing jQuery or jQuery Lite Directly	50
Summary	51
3 Understanding AngularJS Application Dynamics	53
Looking at Modules and Dependency Injection	53
Understanding Modules	53
Dependency Injection	54
Defining an AngularJS Module Object	54
Creating Providers in AngularJS Modules	55
Specialized AngularJS Object Providers	56
Service Providers	56
Implementing Providers and Dependency Injection	57
Injecting a Built-in Provider into a Controller	58
Implementing a Custom Provider and Injecting It into a Controller	59
Applying Configuration and Run Blocks to Modules	61
Adding Configuration Blocks	61
Adding Run Blocks	62
Implementing Configuration and Run Blocks	62
Summary	64
4 Implementing the Scope as a Data Model	65
Understanding Scopes	65
The Relationship Between the Root Scope and Applications	65
The Relationship Between Scopes and Controllers	66
The Relationship Between Scopes and Templates	68

The Relationship Between Scope and Back-End Server Data	71
The Scope Life Cycle	71
Implementing Scope Hierarchy	73
Summary	76
5 Using AngularJS Templates to Create Views	77
Understanding Templates	77
Using Expressions	78
Using Basic Expressions	79
Interacting with the Scope in Expressions	81
Using JavaScript in AngularJS Expressions	85
Using Filters	87
Using Built-in Filters	87
Using Filters to Implement Ordering and Filtering	91
Creating Custom Filters	94
Summary	97
6 Implementing Directives in AngularJS Views	99
Understanding Directives	99
Using Built-in Directives	99
Directives That Support AngularJS Functionality	100
Directives That Extend Form Elements	104
Directives That Bind the Model to Page Elements	109
Directives That Bind Page Events to Controllers	113
Summary	122
7 Creating Your Own Custom Directives to Extend HTML	123
Understanding Custom Directive Definitions	123
Defining the Directive View Template	125
Restricting Directive Behavior	126
Adding a Controller to a Directive	127
Configuring the Directive Scope	128
Transcluding Elements	130
Manipulating the DOM with a Link Function	130
Manipulating the DOM with a Compile Function	132
Implementing Custom Directives	133
Manipulating the DOM in Custom Directives	134

- Implementing Event Handlers in a Custom Directive 136
- Implementing Nested Directives 140
- Summary 144

8 Using Events to Interact with Data in the Model 145

- Browser Events 145
- User Interaction Events 145
- Adding `$watches` to Track Scope Change Events 146
 - Using `$watch` to Track a Scope Variable 146
 - Using `$watchGroup` to Track Multiple Scope Variables 147
 - Using `$watchCollection` to Track Changes to Properties of an Object in the Scope 147
- Implementing Watches in a Controller 148
- Emitting and Broadcasting Custom Events 150
 - Emitting a Custom Event to the Parent Scope Hierarchy 150
 - Broadcasting a Custom Event to the Child Scope Hierarchy 151
 - Handling Custom Events with a Listener 151
 - Implementing Custom Events in Nested Controllers 151
- Summary 155

9 Implementing AngularJS Services in Web Applications 157

- Understanding AngularJS Services 157
- Using the Built-in Services 158
 - Sending HTTP `GET` and `PUT` Requests with the `$http` Service 159
 - Using the `$cacheFactory` Service 165
 - Implementing Browser Alerts Using the `$window` Service 166
 - Interacting with Browser Cookies Using the `$cookieStore` Service 166
 - Implementing Timers with `$interval` and `$timeout` Services 168
 - Using the `$animate` Service 169
 - Using the `$location` Service 176
- Using the `$q` Service to Provide Deferred Responses 180
- Summary 181

10 Creating Your Own Custom AngularJS Services 183

- Understanding Custom AngularJS Services 183
 - Defining a `value` Service 183
 - Defining a `constant` Service 184

Using a Factory Provider to Build a <code>factory</code> Service	184
Using an Object to Define a <code>service</code> Service	184
Integrating Custom Services into Your AngularJS Applications	185
Implementing a Simple Application That Uses All Four Types of Services	185
Implementing Simple Time Service	188
Implementing a Database Access Service	192
Summary	198
11 Creating Rich Web Application Components the AngularJS Way	199
Building a Tabbed View	199
Implementing Draggable and Droppable Elements	204
Adding a Zoom View Field to Images	208
Implementing Expandable and Collapsible Elements	212
Adding Star Ratings to Elements	217
Summary	220
A Testing AngularJS Applications	223
Deciding on a Testing Platform	223
Understanding AngularJS Unit Tests	224
Dependencies and Unit Tests	224
Testing Controllers with User Input Bound to Scope Data	226
Testing Filters	227
Testing Simple Directives	228
Testing Custom Directives That Use Transclusion	229
Testing Directives that Use External Templates	230
Understanding AngularJS End-to-End Testing	230
Index	233

About the Author

Brad Dayley is a senior software engineer with over 20 years of experience developing enterprise applications and Web interfaces. He has a passion for new technologies, especially ones that really make a difference in the software industry. He has used JavaScript, jQuery, and AngularJS for years and is the author of *Node.js, MongoDB and AngularJS Web Development*, *jQuery and JavaScript Phrasebook*, and *Teach Yourself jQuery and JavaScript in 24 Hours*. He has designed and implemented a wide array of applications and services from application servers to complex 2.0 web interfaces. He is also the author of *Teach Yourself MongoDB in 24 Hours*, *Python Developer's Phrasebook*, and *Teach Yourself Django in 24 Hours*.

Dedication



For D!

A & F



Acknowledgments

I'd like to take this page to thank all those who made this title possible. First, I thank my wonderful wife for the inspiration, love, and support she gives me. I'd never make it far without you. I also want to thank my boys for the help they are when I am writing. Thanks to Mark Taber for getting this title rolling in the right direction, Cheri Clark and Katie Matejka for helping me turn technical ramblings into readable text, Jesse Smith for keeping me clear and technically accurate, and Elaine Wiley for managing the project and making sure the final book is the finest quality.

We Want to Hear from You!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

We welcome your comments. You can email or write directly to let us know what you did or didn't like about this book—as well as what we can do to make our books better.

Please note that we cannot help you with technical problems related to the topic of this book, and that due to the high volume of mail we receive, we might not be able to reply to every message.

When you write, please be sure to include this book's title and author, as well as your name and phone or email address.

Email: feedback@developers-library.info

Mail: Reader Feedback
Addison-Wesley Developer's Library
800 East 96th Street
Indianapolis, IN 46240 USA

Reader Services

Visit our website and register this book at www.informit.com/register for convenient access to any updates, downloads, or errata that might be available for this book.

Introduction

Welcome to *Learning AngularJS*. This book is designed to catapult you into the world of using AngularJS to build highly interactive yet well-structured web applications. The book covers the basics of the AngularJS framework and how to use it to build well-designed, reusable components for web applications. AngularJS is one of the most exciting and innovative technologies emerging in the world of web development.

This introduction covers the following:

- Who should read this book
- Why you should read this book
- What you will be able to achieve using this book
- What AngularJS is and why it is a great technology
- How this book is organized
- Where to find the code examples

Let's get started.

Who Should Read This Book

This book is aimed at readers who already have an understanding of the basics of HTML and have done some programming in a modern programming language. Having an understanding of JavaScript and jQuery will make this book easier to digest, but is not required as the basics of JavaScript are covered.

Why You Should Read This Book

This book will teach you how to create powerful, interactive web applications that have a well-structured, easy-to-reuse code base that will be easy to maintain. A great feature about

AngularJS is that it actually forces you to become a better web developer by adhering to the underlying structure and design.

The typical readers of this book want to master AngularJS for the purpose of building highly interactive web applications. The typical reader will also want to leverage the innovative MVC approach of AngularJS to implement well-designed and structured web pages and web applications. Overall, AngularJS provides an easy-to-implement, fully integrated web development platform enabling you to implement amazing Web 2.0 applications.

What You Will Learn from This Book

Reading this book will enable you to build real-world, dynamic websites and web applications. Websites are no longer simple static content that consist of HTML pages with integrated images and formatted text. Instead, websites have become much more dynamic, with a single page often serving as the entire site or application.

Using AngularJS technology enables you to build logic directly into your web page that binds the data model for the client web application to back-end services and databases. AngularJS also enables you to easily extend the capability of HTML so that the UI design logic can be expressed easily in an HTML template file. Following are just a few of the things you will learn while reading this book:

- How to quickly build AngularJS templates with built-in directives that enhance the user experience
- How to bind UI elements to the data model so that when the model changes the UI changes and vice versa
- How to bind mouse and keyboard events directly to the data model and back-end functionality to provide robust user interactions
- How to define your own custom AngularJS directives that extend the HTML language
- How to implement client-side services that can interact with the web server
- How to build dynamic browser views that provide rich user interaction
- How to create custom services that can easily be reused in other AngularJS applications
- How to implement rich UI components such as zoomable images and expandable lists as custom AngularJS directives

What Is AngularJS?

AngularJS is a client-side framework developed by Google. It is written in JavaScript with a reduced jQuery library called jQuery lite. The entire ideology behind AngularJS is to provide a framework that makes it easy to implement well-designed and well-structured web pages and applications using an MVC framework.

AngularJS provides all that functionality to handle user input in the browser, manipulate data on the client side, and control how elements are displayed in the browser view. Here are some of the benefits AngularJS provides:

- **Data Binding:** AngularJS has a very clean method to bind data to HTML elements using its powerful scope mechanism.
- **Extensibility:** The AngularJS architecture enables you to easily extend almost every aspect of the language to provide your own custom implementations.
- **Clean:** AngularJS forces you to write clean, logical code.
- **Reusable Code:** The combination of extensibility and clean code makes it very easy to write reusable code in AngularJS. In fact, the language often forces you to do so when you're creating custom services.
- **Support:** Google is investing a lot into this project, which gives it an advantage where other similar initiatives have failed.
- **Compatibility:** AngularJS is based on JavaScript and has a close relationship with jQuery. That makes it easier to begin integrating AngularJS into your environment and reuse pieces of your existing code within the structure of the AngularJS framework.

How Is This Book Organized?

This book is divided into 11 chapters and one appendix:

Chapter 1, "Jumping into JavaScript," provides sort of a JavaScript primer just in case you are not familiar with JavaScript. This chapter also walks you through the process of setting up a development environment with a Node.js server that you can use to follow along with some of the examples. You should at least check out the first few sections even if you are familiar with JavaScript so that you can create the development environment.

Chapter 2, "Getting Started with AngularJS," covers the basics of the AngularJS framework. You will learn how AngularJS is organized and how to design AngularJS applications.

Chapter 3, "Understanding AngularJS Application Dynamics," covers the basic structure of an AngularJS application. You will learn how to define modules and how dependency injection works in AngularJS.

Chapter 4, "Implementing the Scope as a Data Model," covers the relationship between the data model in AngularJS called the scope and other AngularJS components. You also will learn how scope hierarchy works.

Chapter 5, "Using AngularJS Templates to Create Views," covers the structure of AngularJS templates. You will learn how to add elements to the template that reflect data in the model and how to use filters to automatically format elements as they are rendered to the browser view.

Chapter 6, “Implementing Directives in AngularJS Views,” covers the built-in AngularJS directives. You will learn how to implement directives in various ways, from turning a simple JavaScript array into multiple HTML elements to binding elements on the web page directly to the scope model. You’ll also learn how to handle mouse and keyboard events in the controller.

Chapter 7, “Creating Your Own Custom Directives to Extend HTML,” covers creating custom AngularJS directives. You’ll learn how to build directives that can enhance the behavior of existing HTML elements as well as create completely new HTML elements that provide great interactions for users.

Chapter 8, “Using Events to Interact with Data in the Model,” covers the types of events you will encounter and how to manage them. You will learn how to create and handle your own custom events. This chapter also covers watching values in the scope model and taking action when they change.

Chapter 9, “Implementing AngularJS Services in Web Applications,” covers the built-in services that AngularJS provides. These services enable you to communicate with the web server using HTTP requests, interact with the browser, and implement animation of elements on the web page.

Chapter 10, “Creating Your Own Custom AngularJS Services,” covers the mechanics available in AngularJS to create your own custom services. Custom services are a great way to make functionality reusable because you can easily inject the functionality provided by custom services into multiple applications.

Chapter 11, “Creating Rich Web Application Components the AngularJS Way,” covers using AngularJS mechanisms to build richly interactive page elements. This chapter kind of acts as a review of all the others. You will learn about how to build expandable/collapsible elements, drag and drop functionality, zoomable images, tabbed panels, and star ratings using AngularJS.

Appendix A, “Testing AngularJS Applications,” discusses unit and end-to-end testing in AngularJS. This appendix provides some simple pointers for when you’re designing tests and also some links to additional resources.

Getting the Code Examples

Throughout this book you will find code examples contained in listing blocks. The titles for the listing blocks include a filename of the file that contains the source. You can access the source-code files and images used in the examples on GitHub.

Finally

I hope you enjoy this book and enjoy learning about AngularJS as much I did. It is a great, innovative technology that is really fun to use. Soon you’ll be able to join the many other web developers who use AngularJS to build interactive websites and web applications.

This page intentionally left blank

Getting Started with AngularJS

AngularJS is a JavaScript framework that provides a very structured method of creating websites and web applications. Essentially, AngularJS is a JavaScript library that is built on a lightweight version of jQuery—a combination that enables AngularJS to provide the best of JavaScript and jQuery and at the same time enforce a structured Model View Controller (MVC) framework.

AngularJS is a perfect client-side library for most web applications because it provides a very clean and structured approach. With a clean, structured front end, you will find that it is much easier to implement clean, well-structured server-side logic.

This chapter introduces you to AngularJS as well as the major components involved in an AngularJS application. Understanding these components is critical before you try to implement an AngularJS application because the framework is different from more traditional JavaScript web application programming.

After you have a good grasp of the components and the life cycle of an AngularJS application, you'll learn how to construct a basic AngularJS application, step-by-step. This should prepare you to jump into the following chapters, which provide much more detail on implementing AngularJS.

Why AngularJS?

AngularJS is an MVC framework that is built on top of JavaScript and a lightweight version of jQuery. MVC frameworks separate the business logic in code from the view and the model. Without this separation, JavaScript-based web applications can quickly get out of hand when you are trying to manage all three together and a complex maze of functions.

Everything that AngularJS provides, you could implement yourself by using JavaScript and jQuery, or you could even try using another MVC JavaScript framework. However, AngularJS has a lot of functionality, and the design of the AngularJS framework makes it easy to implement MVC in the correct manner. The following are some of the reasons to choose AngularJS:

- The AngularJS framework forces correct implementation of MVC and also makes it easy to implement MVC correctly.
- The declarative style of AngularJS HTML templates makes the intent of the HTML more intuitive and makes the HTML easier to maintain.
- The model portion of AngularJS is basic JavaScript objects, making it easy to manipulate, access, and implement.
- AngularJS uses a declarative approach to extend the functionality of HTML by having a direct link between the HTML declaratives and the JavaScript functionality behind them.
- AngularJS provides a very simple and flexible filter interface that enables you to easily format data as it passes from the model to the view.
- AngularJS applications tend to use a fraction of the code that traditional JavaScript applications use because you need to focus only on the logic and not all the little details, such as data binding.
- AngularJS requires a lot less Document Object Model (DOM) manipulation than traditional methods and guides you to put the manipulations in the correct locations in applications. It is easier to design applications based on presenting data than on DOM manipulation.
- AngularJS provides several built-in services and enables you to implement your own in a structured and reusable way. This makes your code more maintainable and easier to test.
- Due to the clean separation of responsibilities in the AngularJS framework, it is easy to test your applications and even develop them using a test-driven approach.

Understanding AngularJS

AngularJS provides a very structured framework based on an MVC (Model View Controller) model. This framework enables you to build structured applications that are robust and easily understood and maintained. If you are not familiar with the MVC model, the following paragraph provides a quick synopsis to help you understand the basics. It is by no means complete and only intended to give you enough reference to see how AngularJS applies MVC principles. The Wikipedia website is a great resource if you want additional information about MVC in general.

In MVC, there are three components: the Model is the data source, View is the rendered webpage, and the Controller handles the interaction between the two. A major purpose of MVC is to separate out responsibilities in your JavaScript code to keep it clean and easy to follow. AngularJS is one of the best MVC frameworks available because it makes it very easy to implement MVC.

To get started with AngularJS, you first need to understand the various components that you will be implementing and how they interact with each other. The following sections discuss the various components involved in an AngularJS application, their purpose, and what each is responsible for.

Modules

AngularJS introduces the concept of a module representing components in an application. The module provides a namespace that enables you to reference directives, scopes, and other components based on model name. This makes it easier to package and reuse parts of an application.

Each view or web page in AngularJS has a single module assigned to it via the `ng-app` directive. (Directives are discussed later in this chapter.) However, you can add other modules to the main module as dependencies, which provides a very structured and componentized application. The main AngularJS module acts similar to the root namespace in C# and Java.

Scopes and the Data Model

AngularJS introduces the concept of a scope. A scope is really just a JavaScript representation of data used to populate a view presented on a web page. The data can come from any source, such as a database, a remote web service, or the client-side AngularJS code, or it can be dynamically generated by the web server.

A great feature of scopes is that they are just plain JavaScript objects, which means you can manipulate them as needed in your AngularJS code with ease. Also, you can nest scopes to organize your data to match the context that they are being used in.

Views with Templates and Directives

HTML web pages are based on a DOM in which each HTML element is represented by a DOM object. A web browser reads the properties of a DOM object and knows how to render the HTML element on the web page, based on the DOM object's properties.

Most dynamic web applications use direct JavaScript or a JavaScript-based library such as jQuery to manipulate a DOM object to change the behavior and appearance of the rendered HTML element in the user view.

AngularJS introduces a new concept of combining templates that contain directives which extend the HTML tags and attributes directly with JavaScript code in the background to extend the capability of HTML. Directives have two parts. The first part is extra attributes, elements, and CSS classes that are added to an HTML template. The second part is JavaScript code that extends the normal behavior of the DOM.

The advantage of using directives is that the intended logic for visual elements is indicated by the HTML template such that it is easy to follow and is not hidden within a mass of JavaScript code. One of the best features of AngularJS is that the built-in AngularJS directives handle most of the necessary DOM manipulation functionality that you need in order to bind the data in the scope directly to the HTML elements in the view.

You can also create your own AngularJS directives to implement any necessary custom functionality you need in a web application. In fact, you should use your own custom directives to do any direct DOM manipulation that a web application needs.

Expressions

A great feature of AngularJS is the capability to add expressions inside the HTML template. AngularJS evaluates expressions and then dynamically adds the result to a web page. Because expressions are linked to the scope, you can have an expression that utilizes values in the scope, and as the model changes, so does the value of the expression.

Controllers

AngularJS completes the MVC framework through the implementation of controllers. Controllers augment the scope by setting up the initial state or values in the scope and by adding behavior to the scope. For example, you can add a function that sums values in a scope to provide a total such that if the model data behind the scope changes, the total value always changes.

You add controllers to HTML elements by using a directive and then implement them as JavaScript code in the background.

Data Binding

One of the best features of AngularJS is the built-in data binding. Data binding is the process of linking data from the model with what is displayed in a web page. AngularJS provides a very clean interface to link the model data to elements in a web page.

In AngularJS data binding is a two-way process: When data is changed on a web page, the model is updated, and when data is changed in the model, the web page is automatically updated. This way, the model is always the only source for data represented to the user, and the view is just a projection of the model.

Services

Services are the major workhorses in the AngularJS environment. Services are singleton objects that provide functionality for a web app. For example, a common task of web applications is to perform AJAX requests to a web server. AngularJS provides an HTTP service that houses all the functionality to access a web server.

The service functionality is completely independent of context or state, so it can be easily consumed from the components of an application. AngularJS provides a lot of built-in service components for basic uses, such as HTTP requests, logging, parsing, and animation. You can also create your own services and reuse them throughout your code.

Dependency Injection

Dependency injection is a process in which a code component defines dependencies on other components. When the code is initialized, the dependent component is made available for access within the component. AngularJS applications make heavy use of dependency injection.

A common use for dependency injection is consuming services. For example, if you are defining a module that requires access to the web server via HTTP requests, you can inject the HTTP service into the module, and the functionality is available in the module code. In addition, one AngularJS module consumes the functionality of another via dependency.

Compiler

AngularJS provides an HTML compiler that will discover directives in the AngularJS template and use the JavaScript directive code to build out extended HTML elements. The AngularJS compiler is loaded into the browser when the AngularJS library is bootstrapped. When loaded, the compiler will search through the HTML DOM in the browser and link in any back-end JavaScript code to the HTML elements, and then the final application view will be rendered to the user.

An Overview of the AngularJS Life Cycle

Now that you understand the components involved in an AngularJS application, you need to understand what happens during the life cycle, which has three phases: bootstrap, compilation, and runtime. Understanding the life cycle of an AngularJS application makes it easier to understand how to design and implement your code.

The three phases of the life cycle of an AngularJS application happen each time a web page is loaded in the browser. The following sections describe these phases of an AngularJS application.

The Bootstrap Phase

The first phase of the AngularJS life cycle is the bootstrap phase, which occurs when the AngularJS JavaScript library is downloaded to the browser. AngularJS initializes its own necessary components and then initializes your module, which the `ng-app` directive points to. The module is loaded, and any dependencies are injected into your module and made available to code within the module.

The Compilation Phase

The second phase of the AngularJS life cycle is the HTML compilation stage. Initially when a web page is loaded, a static form of the DOM is loaded in the browser. During the compilation phase, the static DOM is replaced with a dynamic DOM that represents the AngularJS view.

This phase involves two parts: traversing the static DOM and collecting all the directives and then linking the directives to the appropriate JavaScript functionality in the AngularJS built-in library or custom directive code. The directives are combined with a scope to produce the dynamic or live view.

The Runtime Data Binding Phase

The final phase of the AngularJS application is the runtime phase, which exists until the user reloads or navigates away from a web page. At that point, any changes in the scope are reflected in the view, and any changes in the view are directly updated in the scope, making the scope the single source of data for the view.

AngularJS behaves differently from traditional methods of binding data. Traditional methods combine a template with data received from the engine and then manipulate the DOM each time the data changes. AngularJS compiles the DOM only once and then links the compiled template as necessary, making it much more efficient than traditional methods.

Separation of Responsibilities

An extremely important part of designing AngularJS applications is the separation of responsibilities. The whole reason you choose a structured framework is to ensure that code is well implemented, easy to follow, maintainable, and testable. Angular provides a very structured framework to work from, but you still need to ensure that you implement AngularJS in the appropriate manner.

The following are a few rules to follow when implementing AngularJS:

- The view acts as the official presentation structure for the application. Indicate any presentation logic as directives in the HTML template of the view.
- If you need to perform any DOM manipulation, do it in a built-in or your own custom directive JavaScript code—and nowhere else.
- Implement any reusable tasks as services and add them to your modules by using dependency injection.
- Ensure that the scope reflects the current state of the model and is the single source for data consumed by the view.
- Ensure that the controller code only acts to augment the scope data and doesn't include any business logic.
- Define controllers within the module namespace and not globally. This ensures that your application can be packaged easily and prevents overwhelming the global namespace.

Integrating AngularJS with Existing JavaScript and jQuery

The fact that AngularJS is based on JavaScript and jQuery makes it tempting to simply try to add it to existing applications to provide data binding or other functionality. That approach will almost always end up in problem code that is difficult to maintain. However, using AngularJS doesn't mean that you need to simply toss out your existing code either. Often you can selectively take working JavaScript/jQuery components and convert them to either directives or services.

This also brings up another issue: when to use the full version of jQuery as opposed to the jQuery lite version that is provided with AngularJS? I know that many people have strong views in both directions. On one hand, you want to keep your implementation as clean and simple as possible. But on the other hand, there might be times when you need functionality that's available only in the full version of jQuery. My take, as always, is to use what makes sense. If I need functionality that is not provided with AngularJS jQuery lite, I will load the full library. I'll discuss the mechanics of loading jQuery as opposed to jQuery lite later in this chapter.

The following steps suggest a method to integrate AngularJS into your existing JavaScript and jQuery applications:

1. Write at least one small AngularJS application from the ground up that uses a model, custom HTML directives, services, and controllers. In other words, in this application, ensure that you have a practical comprehension of the AngularJS separation of responsibilities.
2. Identify the model portion of your code. Specifically, try to separate out the code that augments the model data in the model into controller functions and code that accesses the back-end model data into services.
3. Identify the code that manipulates DOM elements in the view. Try to separate out the DOM manipulation code into well-defined custom directive components and provide an HTML directive for them. Also identify any of the directives for which AngularJS already provides built-in support.
4. Identify other task-based functions and separate them out into services.
5. Isolate the directives and controllers into modules to organize your code.
6. Use dependency injection to link up your services and modules appropriately.
7. Update the HTML templates to use the new directives.

Obviously, in some instances it just doesn't make sense to use much if any of your existing code. However, by running through the preceding steps, you will get well into the design phase of implementing a project using AngularJS and can then make an informed decision.

Adding AngularJS to Your Environment

AngularJS is a client-side JavaScript library, which means the only thing you need to do to implement AngularJS in your environment is to provide a method for the client to get the `angular.js` library file by using a `<script>` tag in the HTML templates.

The simplest method of providing the `angular.js` library is to use the Content Delivery Network (CDN), which provides a URL for downloading the library from a third party. The downside of this method is that you must rely on a third party to serve the library, and if the client cannot connect to that third-party URL, your application will not work. For example, the following `<script>` tag loads the `angular.js` library from Google APIs CDN:

```
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.5/angular.min.js">
</script>
```

The other method of providing the `angular.js` library is to download it from the AngularJS website (<http://angularjs.org>) and use your own web server to serve the file to the client. This method takes more effort and also requires extra bandwidth on your web server; however, it might be a better option if you want more control over how the client obtains the library.

Bootstrapping AngularJS in an HTML Document

To implement AngularJS in your web pages, you need to bootstrap the HTML document. Bootstrapping involves two parts. The first part is to define the application module by using the `ng-app` directive, and the second is to load the `angular.js` library in a `<script>` tag.

The `ng-app` directive tells the AngularJS compiler to treat that element as the root of the compilation. The `ng-app` directive is typically loaded in the `<html>` tag to ensure that the entire web page is included; however, you could add it to another container element, and only elements inside that container would be included in the AngularJS compilation and consequently in the AngularJS application functionality.

When possible, you should include the `angular.js` library as one of the last tags, if not the last tag, inside the `<body>` of the HTML. When the `angular.js` script is loaded, the compiler kicks off and begins searching for directives. Loading `angular.js` last allows the web page to load faster.

The following is an example of implementing the `ng-app` and `angular.js` bootstrap in an HTML document:

```
<!doctype html>
<html ng-app="myApp">
  <body>
    <script src="http://code.angularjs.org/1.2.9/angular.min.js"></script>
    <script src="/lib/myApp.js"></script>
  </body>
</html>
```

Using the Global APIs

As you are implementing AngularJS applications, you will find that there are common JavaScript tasks that you need to perform regularly, such as comparing objects, deep copying, iterating through objects, and converting JSON data. AngularJS provides a lot of this basic functionality in the global APIs.

The global APIs are available when the `angular.js` library is loaded, and you can access them by using the `angular` object. For example, to create a deep copy of an object named `myObj`, you use the following syntax:

```
var myCopy = angular.copy(myObj);
```

The following code shows an example of iterating through an array of objects by using the `forEach()` global API:

```
var objArr = [{score: 95}, {score: 98}, {score: 92}];
var scores = [];
angular.forEach(objArr, function(value, key){
  this.push(key + '=' + value);
}, scores);
// scores == ['score=95', 'score=98', 'score=92']
```

Table 2.1 lists some of the most useful utilities provided in the global APIs. You will see these used in a number of examples in this book.

Table 2.1 Useful Global API Utilities Provided in AngularJS

Utility	Description
<code>copy(src, [dst])</code>	Creates a deep copy of the <code>src</code> object or array. If a <code>dst</code> parameter is supplied, it is completely overwritten by a deep copy of the source.
<code>element(element)</code>	Returns the DOM element specified as a jQuery element. If you have loaded jQuery before loading AngularJS, the object is a full jQuery object; otherwise, it is only a subset of a jQuery object, using the jQuery lite version built into AngularJS. Table 2.2 lists the jQuery lite methods available in AngularJS.
<code>equals(o1, o2)</code>	Compares <code>o1</code> with <code>o2</code> and returns <code>true</code> if they pass an <code>===</code> comparison.
<code>extend(dst, src)</code>	Copies all the properties from the <code>src</code> object to the <code>dst</code> object.
<code>forEach(obj, iterator, [context])</code>	<p>Iterates through each object in the <code>obj</code> collection, which can be an object or an array. The iterator specifies a function to call, using the following syntax:</p> <pre>function(value, key)</pre> <p>The <code>context</code> parameter specifies a JavaScript object that acts as the context, accessible via the <code>this</code> keyword, inside the <code>forEach</code> loop.</p>
<code>fromJson(json)</code>	Returns a JavaScript object from a JSON string.
<code>toJson(obj)</code>	Returns a JSON string form of the JavaScript object <code>obj</code> .
<code>isArray(value)</code>	Returns <code>true</code> if the <code>value</code> parameter passed in is an Array object.
<code>isDate(value)</code>	Returns <code>true</code> if the <code>value</code> parameter passed in is a Date object.
<code>isDefined(value)</code>	Returns <code>true</code> if the <code>value</code> parameter passed in is a defined object.
<code>isElement(value)</code>	Returns <code>true</code> if the <code>value</code> parameter passed in is a DOM element object or a jQuery element object.

Utility	Description
<code>isFunction(value)</code>	Returns <code>true</code> if the <code>value</code> parameter passed in is a JavaScript function.
<code>isNumber(value)</code>	Returns <code>true</code> if the <code>value</code> parameter passed in is a number.
<code>isObject(value)</code>	Returns <code>true</code> if the <code>value</code> parameter passed in is a JavaScript object.
<code>isString(value)</code>	Returns <code>true</code> if the <code>value</code> parameter passed in is a <code>String</code> object.
<code>isUndefined(value)</code>	Returns <code>true</code> if the <code>value</code> parameter passed in is not defined.
<code>lowercase(string)</code>	Returns a lowercase version of the <code>string</code> parameter.
<code>uppercase(string)</code>	Returns an uppercase version of the <code>string</code> parameter.

Creating a Basic AngularJS Application

Now that you understand the basic components in the AngularJS framework, the intent and design of the AngularJS framework, and how to bootstrap AngularJS, you are ready to get started implementing AngularJS code. This section walks you through a very basic AngularJS application that implements an HTML template, an AngularJS module, a controller, a scope, and an expression.

For this example it is expected that you have created a basic Node.js web server as described in Chapter 1, “Jumping Into JavaScript.” The folder structure for this example will be as follows. Future chapters will have a similar code structure for their examples with just the chapter folder changing:

- `./server.js`: Node.js server that serves up the static content.
- `./images`: Contains any images used in examples in all chapters.
- `./ch01`: Contains any HTML files used for the examples in this chapter.
- `./ch01/js`: Contains the necessary JavaScript for the examples in this chapter.
- `./ch01/css`: Contains the necessary CSS for the examples in this chapter.

After the `server.js` web server is running, the next step is to implement an AngularJS HTML template, such as `first.html` in Listing 2.1, and an AngularJS JavaScript module, such as `first.js` in Listing 2.2.

The following sections describe the important steps in implementing the AngularJS application and the code involved in each step. Each of these steps is described in much more detail in later chapters, so don’t get bogged down in them here. What is important at this point is that you understand the process of implementing the template, module, controller, and scope and generally how they interact with each other.

The web page defined by Listings 2.1 and 2.2 is a simple web form in which you type in first and last names and then click a button to display a message, as shown in Figure 2.1.

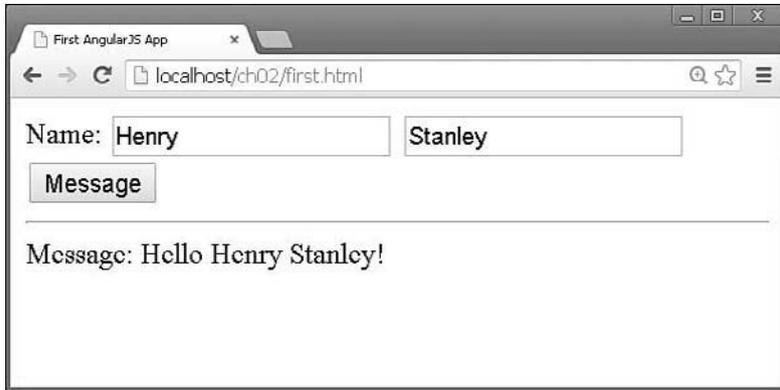


Figure 2.1 Implementing a basic AngularJS web application that uses inputs and a button to manipulate the model and consequently the view.

Loading the AngularJS Library and Your Main Module

Before you can implement an AngularJS application, you need to get the library loaded in an HTML template. The following lines in Listing 2.1 load the `angular.js` library and then load the `first.js` JavaScript custom module:

```
15 <script src="http://code.angularjs.org/1.2.9/angular.min.js"></script>
16 <script src="/js/first.js"></script>
```

Defining the AngularJS Application Root Element

The next step is to define the `ng-app` parameter in the root element so that AngularJS knows where to begin compiling the application. You should also define the module in your JavaScript code to provide a namespace to use when adding controllers, filters, and services.

Line 2 of Listing 2.1 defines the DOM root for an AngularJS module. Notice that `ng-app` is assigned the module name `firstApp`, which corresponds to the module in the JavaScript code:

```
02 <html ng-app="firstApp">
```

Line 1 in Listing 2.2 shows the `firstApp` module object being created in the JavaScript code:

```
01 var firstApp = angular.module('firstApp', []);
```

Adding a Controller to the Template

Next, you need to add a controller for HTML elements that you want the AngularJS module to control. You also need to define the controller in your module code.

Line 7 in Listing 2.1 assigns a controller named `FirstController` to a `<div>` element. This maps the element in the view to a specific controller, which contains a scope:

```
07 <div ng-controller="FirstController">
```

Line 2 in Listing 2.2 shows the `FirstController` code being added to the `firstApp` module:

```
02 firstApp.controller('FirstController', function($scope) {
```

Implementing the Scope Model

After the controller has been defined, you can implement the scope, which involves linking HTML elements to scope variables, initializing the variables in the scope, and providing functionality to handle changes to the scope values.

Lines 9 and 10 in Listing 2.1 are `<input>` elements that are assigned to the `first` and `last` values in the scope. These elements provide a method to update the scope from the browser. If the user types in the input, the scope is also updated:

```
09 <input type="text" ng-model="first">
10 <input type="text" ng-model="last">
```

Lines 3–5 in Listing 2.2 show the initial values of the scope being defined:

```
03 $scope.first = 'Some';
04 $scope.last = 'One';
05 $scope.heading = 'Message: ';
```

Line 11 in Listing 2.1 links a click handler to the `updateMessage()` function defined in the scope:

```
11 <button ng-click='updateMessage()'>Message</button>
```

Lines 6–8 in Listing 2.2 show the `updateMessage()` definition in the scope:

```
06 $scope.updateMessage = function() {
07     $scope.message = 'Hello ' + $scope.first + ' ' + $scope.last + '!';
08 };
```

Line 13 implements an expression that displays the value of the `heading` and `message` variables in the scope on the HTML page:

```
13 {{heading + message}}
```

Listing 2.1 first.html: A Simple AngularJS Template That Provides Two Input Elements and a Button to Interact with the Model

```
01 <!doctype html>
02 <html ng-app="firstApp">
03   <head>
04     <title>First AngularJS App</title>
05   </head>
06   <body>
07     <div ng-controller="FirstController">
08       <span>Name:</span>
09       <input type="text" ng-model="first">
10       <input type="text" ng-model="last">
11       <button ng-click='updateMessage()'>Message</button>
12       <hr>
13       {{heading + message}}
14     </div>
15     <script src="http://code.angularjs.org/1.2.9/angular.min.js"></script>
16     <script src="js/first.js"></script>
17   </body>
18 </html>
```

Listing 2.2 first.js: A Simple AngularJS Module That Implements a Controller to Support the Template in Listing 2.1

```
01 var firstApp = angular.module('firstApp', []);
02 firstApp.controller('FirstController', function($scope) {
03   $scope.first = 'Some';
04   $scope.last = 'One';
05   $scope.heading = 'Message: ';
06   $scope.updateMessage = function() {
07     $scope.message = 'Hello ' + $scope.first + ' ' + $scope.last + '!';
08   };
09 });
```

Using jQuery or jQuery Lite in AngularJS Applications

You will be using at least jQuery lite in your AngularJS applications, so it is important to understand the interactions between jQuery, jQuery lite and AngularJS. Even if you are not a jQuery developer, understanding these interactions will help you write better AngularJS applications. If you are a jQuery developer, understanding the interactions will enable you to leverage your jQuery knowledge in your AngularJS applications.

The following sections describe jQuery lite implementation as well as giving a brief introduction to the jQuery/jQuery lite interactions that you will be seeing in your AngularJS applications. The following chapters will expand on this topic as you see some practical examples that utilize jQuery objects in AngularJS applications.

What Is jQuery Lite?

jQuery lite is simply a stripped-down version of jQuery that is built directly into AngularJS. The intent is to provide all the useful features of jQuery and yet keep it constrained within the AngularJS separation of responsibilities paradigm.

Table 2.2 lists the jQuery methods available in jQuery lite along with any restrictions that might apply. The restrictions are necessary to enforce things like manipulating elements only within a custom directive, and so on.

Table 2.2 **jQuery Methods That Are Supported in jQuery Lite**

jQuery Method	Limitations, if any, in jQuery Lite
<code>addClass()</code>	
<code>after()</code>	
<code>append()</code>	
<code>attr()</code>	
<code>bind()</code>	Does not support namespaces, selectors, or eventData.
<code>children()</code>	Does not support selectors.
<code>clone()</code>	
<code>contents()</code>	
<code>css()</code>	
<code>data()</code>	
<code>detach()</code>	
<code>empty()</code>	
<code>eq()</code>	
<code>find()</code>	Limited to lookups by tag name.
<code>hasClass()</code>	
<code>html()</code>	
<code>text()</code>	Does not support selectors.
<code>on()</code>	Does not support namespaces, selectors, or eventData.
<code>off()</code>	Does not support namespaces or selectors.
<code>one()</code>	Does not support namespaces or selectors.

jQuery Method	Limitations, if any, in jQuery Lite
<code>parent()</code>	Does not support selectors.
<code>prepend()</code>	
<code>prop()</code>	
<code>ready()</code>	
<code>remove()</code>	
<code>removeAttr()</code>	
<code>removeClass()</code>	
<code>removeData()</code>	
<code>replaceWith()</code>	
<code>toggleClass()</code>	
<code>triggerHandler()</code>	Passes a dummy event object to handlers.
<code>unbind()</code>	Does not support namespaces.
<code>val()</code>	
<code>wrap()</code>	

Table 2.3 lists the additional events and methods that AngularJS adds to jQuery lite objects.

Table 2.3 Methods and Events Added to jQuery Lite Objects

Method/Event	Description
<code>\$destroy</code>	AngularJS intercepts all jQuery or jQuery lite DOM destruction calls and fires this event on all DOM nodes being removed. This can be used to clean up any third-party bindings to the DOM element before it is removed.
<code>controller(name)</code>	Returns the <code>controller</code> object of the current element or its parent. If no <code>name</code> is specified, the controller associated with the <code>ngController</code> directive is returned. If a <code>name</code> is provided as a directive name, the controller for this directive is returned.
<code>injector()</code>	Returns the <code>injector</code> object of the current element or its parent.
<code>scope()</code>	Returns the <code>scope</code> object of the current element or its parent.
<code>isolateScope()</code>	Returns an <code>isolate scope</code> object if one is attached directly to the current element. This works only on elements that contain a directive that starts a new isolate scope.
<code>inheritedData()</code>	Works the same as the jQuery <code>data()</code> method, but walks up the DOM until a value is found or the top parent element is reached.

Accessing jQuery or jQuery Lite Directly

For most AngularJS applications the jQuery lite library built into AngularJS is sufficient. However, if you need the additional functionality of the full version of jQuery, simply load the jQuery library before loading the AngularJS library. For example:

```
<script src="http://code.jquery.com/jquery-1.11.0.min.js"></script>
<script src="http://code.angularjs.org/1.2.9/angular.min.js"></script>
```

Regardless of whether jQuery lite or the full jQuery library is loaded, jQuery is accessed from the AngularJS code using the `element` attribute of the `angular` variable available when AngularJS is bootstrapped. Essentially, `angular.element` will be an alias for the jQuery variable that is normally used in jQuery applications. One of the best ways I've seen this relationship described is as follows:

```
angular.element() === jQuery() === $()
```

Accessing jQuery or jQuery Lite Directly

More often than not, you will be using the jQuery or jQuery lite functionality in jQuery objects that AngularJS creates for you. All element references in AngularJS are always wrapped as jQuery or jQuery lite objects; they are never raw DOM objects.

For example, when you create a directive in AngularJS as discussed later in this book, an `element` is passed to the `link` function. That `element`, as shown here, is a jQuery or jQuery lite object, and you can use the jQuery functionality accordingly:

```
angular.module('myApp', [])
  .directive('myDirective', function() {
    . . .
    link: function(scope, elem, attrs, photosControl) {
      //elem is a jQuery lite object
      elem.addClass(...);
    }
  });
```

Another example of accessing the jQuery functionality is from events that are triggered on AngularJS bindings. For example, consider the following code that uses the `ngClick` binding to bind a browser click event on a `<div>` element to a `clicked()` function in the AngularJS code:

```
<div ng-click="clicked($event)">Click Me</div>
You can access a jQuery version of the object using the following AngularJS code:
$scope.clicked = function(event){
  var jQueryElement = angular.element(event.target);
};
```

Note that it was necessary to use the `angular.element()` method to convert the `target` DOM object into a jQuery object.

Summary

AngularJS is a JavaScript library framework that provides a very structured method for creating websites and web applications. AngularJS structures a web application into a very clean MVC-styled approach. AngularJS scopes provide contextual binding to the data model for the application and are made up of basic JavaScript objects. AngularJS utilizes templates with directives that extend HTML capabilities, enabling you to implement totally customized HTML components.

In this chapter you looked at the different components in an AngularJS application and how they interact with each other. You also learned about the life cycle of an AngularJS application, which involves bootstrap, compilation, and runtime phases. At the end of this chapter, you walked through a step-by-step example of implementing a basic AngularJS application, including a template, module, controller, and scope.

This page intentionally left blank

Index

A

- absUrl() method, 176**
- accessing jQuery/jQuery lite directly, 49-50**
- addClass() method, 48**
- adding**
 - configuration blocks to modules, 61-62
 - controller to template, 46
 - items to arrays, 30-31
 - run blocks to modules, 62
- addition operator, 12**
- a directive, 104**
- after() method, 48**
- \$anchorScroll service, 158**
- And operator, 14**
- angular-animate.js library, 174**
- angular-cookies.js library, 166**
- AngularJS**
 - angular.js library file, providing, 41-42
 - applications. *See* applications
 - benefits, 3, 35-36
 - compiler, definition of, 39
 - controllers, definition of, 38
 - data binding, definition of, 38

- definition of, 2, 35
- dependency injection, definition of, 38-39
- directives, definition of, 37
- expressions
 - definition of, 38
 - JavaScript expressions versus, 79
- global APIs, list of, 42-43
- HTML documents, bootstrapping, 42
- integration with JavaScript and jQuery, 40-41
- life-cycle phases, 39-40
- modules, definition of, 37
- as MVC framework, 36
- scopes, definition of, 37
- separation of responsibilities, 40
- services, definition of, 38
- templates, 37
- website, 42
- angular.js library file**
 - bootstrapping HTML documents, 42
 - loading, 45
 - providing, 41-42
- angular.module() method, 54-55**
- animate() method, 171**
- \$animate service, 158, 169-172**
 - animation example, 172
 - directives for, 169
 - implementing
 - in CSS, 170-171
 - in JavaScript, 171-172
- animate.css listing, 174**
- animation**
 - directives for, 169-170
 - example, 172
 - implementing
 - in CSS, 170-171
 - in JavaScript, 171-172
- animation() method, 56**
- anonymous functions, 21**
- APIs, global, list of, 42-44**
- append() method, 48**
- appending elements in directives, 204-206**
- applications**
 - accessing jQuery directly, 49-50**
 - creating
 - adding controller, 46
 - defining root element, 45
 - folder structure, 44
 - implementing scope, 46
 - loading angular.js library, 45
 - drag-and-drop elements example, 204-206
 - expandable/collapsible elements example, 212-215
 - modules. *See* modules
 - root scope and, 65-66
 - star ratings example, 217-219
 - tabbed view example, 199-202
 - testing, 223
 - end-to-end testing, 230-231
 - evaluating platforms, 223
 - unit testing, 224-230
 - zoom view field example, 208-210
- \$apply() method, 72**
- arithmetic operators, list of, 12-13**
- array data type, 12**
- arrays**
 - adding/removing items, 30-31
 - combining, 29
 - converting to strings, 30
 - defining, 28
 - iterating through, 30
 - length of, 28
 - methods, 28

searching for items, 30
 splitting strings into, 27
assignment operators, list of, 13
attr() method, 48

B

back-end server data, scopes and, 71
bind() method, 48
binding data. See data binding
bindToController property, 123
blur events, 115
Boolean data type, 11
bootstrap phase (AngularJS life cycle), 39
bootstrapping HTML documents, 42
break keyword, 16, 18-19
\$broadcast() method, 151
broadcasting custom events, 151
browser alerts, implementing, 166
browser events, 145
browsers. See web browsers
built-in directives. See directives, built-in directives
built-in filters, 87-90
built-in services

- \$anchorScroll, 158
- \$animate, 158, 169-172
 - animation example, 172
 - directives for, 169
 - implementing in CSS, 170-171
 - implementing in JavaScript, 171-172
- \$cacheFactory, 158, 165
- \$compile, 158
- \$cookie, 166-167
- \$cookies, 158
- \$cookieStore, 166-167
- \$document, 158
- \$exceptionHandler, 158

\$http, 158-163

- configuring, 160
- HTTP server implementation and access, 161-163
- response callback functions, 161
- shortcut methods, 159

 \$interpolate, 158
 \$interval, 158, 168-169
 list of, 158
 \$locale, 158
 \$location, 158, 176-177
 \$log, 158
 \$parse, 158
 \$q, 158, 180-181
 \$resource, 158
 \$rootElement, 158
 \$rootScope, 158
 \$route, 158
 \$routeParams, 158
 \$sanitize, 158
 \$sce, 158
 \$swipe, 158
 \$templateCache, 158
 \$timeout, 158, 168-169
 usage with custom services, 192-194
 \$window, 158, 166

C

cache property, 160
\$cacheFactory service, 158, 165
case statements, 15-16
CDN (Content Delivery Network), 41
charAt() method, 26
charCodeAt() method, 26
child scopes

- broadcasting events to, 151
- controllers and, 66
- in scope hierarchy, 73

- children() method, 48**
- clone() method, 48**
- code listings. See listings**
- collapsible/expandable elements example application, 212-215**
- combining arrays, 29. See also concatenating strings**
- comparison operators, list of, 14-15**
- compilation phase (AngularJS life cycle), 39**
- compile() function, 132-133**
- compile property, 123**
- \$compile service, 158**
- compiler, definition of, 39**
- concat() method**
 - arrays, 28-29
 - strings, 26-27
- concatenating strings, 13, 27. See also combining arrays**
- conditional statements**
 - if statements, 15
 - switch statements, 15-16
- config_run_blocks.html listing, 63**
- config_run_blocks.js listing, 63**
- configuration blocks**
 - adding, 61-62
 - implementing, 62-63
- configuration phase (modules)**
 - adding configuration blocks, 61-62
 - implementing configuration blocks, 62-63
- configuring**
 - \$http service, 160
 - Node.js, 6-7
 - scopes for custom directives, 128-130
- constant() method, 56**
- constant service, 184-186**
- Content Delivery Network (CDN), 41**
- contents() method, 48**
- continue keyword, 19**
- controller() method, 49, 56, 66**
- controller property, 123, 127-128**
- controllerAs property, 123**
- controllers**
 - adding to directives, 127-128
 - adding to template, 46
 - binding events to, 113-120
 - definition of, 38
 - implementing watches in, 148
 - injecting providers into, 58-59
 - isolating scopes from, 212-215
 - nested controllers, implementing custom events, 151-152
 - scopes and, 66-67
 - unit testing, 226-227
- converting arrays to strings, 30**
- \$cookie service, 166-167**
- cookies, 166-167**
- \$cookies service, 158**
- \$cookieStore service, 166-167**
- copy() global API, 43**
- creation phase (scopes), 71-72**
- CSS, animation in, 170-171**
- css() method, 48**
- currency[:symbol] filter, 87**
- custom directives**
 - adding controller to, 127-128
 - configuring scope, 128-130
 - defining view template, 124-126
 - definition of, 123
 - directive() method, 123
 - DOM manipulation
 - with compile() function, 132-133
 - example, 134
 - with link() function, 130-132

- drag-and-drop elements example, 204-206
- event handlers, 136-137
- extending HTML, 208-210
- nested directives, 140-141, 199-202, 212-215
- properties, 123
- restricting behavior, 126-127
- transcluding elements, 130
- unit testing, 228
 - with external templates, 230
 - with transclusion, 229-230
- user interaction events, 145-146
- custom events, 150**
 - broadcasting, 151
 - emitting, 150-151
 - handling with listeners, 151
 - implementing in nested controllers, 151-152
- custom filters, 94-95**
 - unit testing, 227-228
- custom objects, defining, 23-24**
- custom services**
 - constant service, 184
 - database access service example, 192-194
 - factory service, 184
 - implementation example, 185-186
 - service service, 184-185
 - time service example, 188-189
 - types of, 183
 - value service, 183-184
- D**

- data binding**
 - definition of, 38
 - with directives, 109-112
 - events to controllers, 113-120
 - in runtime phase (AngularJS life cycle), 40
 - star ratings example application, 217-219
 - testing controllers, 226-227
- data() method, 48**
- data model, scopes as, 65**
 - back-end server data, 71
 - controllers and, 66-67
 - expressions and, 78-79, 81-82
 - life cycle phases, 71-72
 - root scope, 65-66
 - scope hierarchy, 73
 - templates and, 68-69
- data property, 160**
- data types, list of, 11-12**
- database access service example, 192-194**
- date[:format] filter, 87**
- decrement operator, 12**
- deferred responses, 180-181**
- defining**
 - arrays, 28
 - custom objects, 23-24
 - functions, 19-20
 - modules, 54-55
 - root element, 45
 - strings, 25
 - variables, 10-11
- dependency injection**
 - configuration phase (modules)
 - adding configuration blocks, 61-62
 - implementing configuration blocks, 62-63
 - defining modules, 54-55
 - definition of, 38-39, 54
 - filters and, 87
 - implementing, 57-58
 - injectors, 54

- providers, 54
 - injecting into controllers, 58-59
- run phase (modules)
 - adding run blocks, 62
 - implementing run blocks, 62-63
- services, 157
- unit testing and, 224-226
 - global lookup, 225
 - new operator, 224-225
 - passed parameters, 226
 - registry requests, 225-226
- \$destroy event, 49**
- \$destroy() method, 72**
- detach() method, 48**
- development environment, setup, 5-6**
- digest loop, 72**
- \$digest() method, 72**
- directive() method, 56, 123**
- directive_angular_include.html listing, 103**
- directive_angular_include.js listing, 103**
- directive_bind.html listing, 112**
- directive_bind.js listing, 112**
- directive_custom_dom.html listing, 135**
- directive_custom_dom.js listing, 134**
- directive_custom_photos.js listing, 141**
- directive_custom_zoom.html listing, 139**
- directive_custom_zoom.js listing, 137**
- directive_custom.html listing, 142**
- directive_focus_events.html listing, 116**
- directive_focus_events.js listing, 116**
- directive_form.html listing, 108**
- directive_form.js listing, 105**
- directive_keyboard_events.html listing, 118**
- directive_keyboard_events.js listing, 118**
- directive_mouse_events.html listing, 121**
- directive_mouse_events.js listing, 120**

directives

- for animation, 169-170
- built-in directives
 - a, 104
 - binding data to page elements, 109-112
 - binding events to controllers, 113-120
 - event, 113
 - extending form elements, 104-105
 - input, 104
 - input.checkbox, 104
 - input.date, 104
 - input.dateTimeLocal, 104
 - input.email, 104
 - input.month, 104
 - input.number, 104
 - input.radio, 104
 - input.text, 104
 - input.time, 104
 - input.url, 104
 - input.week, 104
 - ng-app, 37, 42, 45, 100
 - ng-bind, 109
 - ng-bind-html, 109
 - ng-bind-template, 109
 - ng-blur, 114-115
 - ng-change, 114
 - ng-checked, 114
 - ng-class, 109
 - ng-class-even, 109
 - ng-class-odd, 109
 - ng-click, 69, 114, 120
 - ng-cloak, 100
 - ng-controller, 66, 100
 - ng-copy, 114
 - ng-cut, 114

- ng-dblclick, 114
- ng-disabled, 109
- ng-focus, 114-115
- ng-form, 104
- ng-hide, 109
- ng-href, 100
- ng-if, 109
- ng-include, 100, 102
- ng-init, 109
- ng-keydown, 114, 117-118
- ng-keypress, 114
- ng-keyup, 114, 117-118
- ng-list, 100
- ng-model, 68, 109
- ng-mousedown, 114, 120
- ng-mouseenter, 114, 120
- ng-mouseleave, 114, 120
- ng-mousemove, 114, 120
- ng-mouseover, 114
- ng-mouseup, 114, 120
- ng-non-bindable, 100
- ng-open, 100
- ng-options, 104
- ng-paste, 114
- ng-pluralize, 100
- ng-readonly, 100
- ng-repeat, 109
- ng-required, 100
- ng-selected, 100
- ng-show, 109
- ng-src, 100
- ng-srcset, 100
- ng-style, 109
- ng-submit, 114
- ng-swipe-left, 114
- ng-swipe-right, 114
- ng-switch, 109
- ng-transclude, 100, 130
- ng-value, 109
- ng-view, 100
- script, 100
- select, 104
- support functionality, 100-102
- textarea, 104
- custom directives
 - adding controller to, 127-128
 - configuring scope, 128-130
 - defining view template, 124-126
 - definition of, 123
 - directive() method, 123
 - DOM manipulation example, 134
 - DOM manipulation with compile() function, 132-133
 - DOM manipulation with link() function, 130-132
 - drag-and-drop elements example, 204-206
 - event handlers, 136-137
 - extending HTML, 208-210
 - nested directives, 140-141, 199-202, 212-215
 - properties, 123
 - restricting behavior, 126-127
 - transcluding elements, 130
 - unit testing, 228-230
- definition of, 37, 77, 99
- user interaction events, 145-146
- division operator, 12**
- \$document service, 158**
- DOM manipulation, 37**
 - with compile() function, 132-133
 - with custom directives, 134
 - with link() function, 130-132
- do/while loops, 17**

drag-and-drop elements example application, 204-206

dragdrop.html listing, 206

dragdrop.js listing, 204

E

editors, purpose of, 5

element() global API, 43

else statements, 15

\$emit() method, 150-151

emitting custom events, 150-151

empty() method, 48

end-to-end testing, 230-231

eq() method, 48

equal to operator, 14

equals() global API, 43

error handling

finally keyword, 32-33

throwing errors, 32

try/catch blocks, 31-32

escape codes for strings, 25

evaluating testing platforms, 223

event directive, 113

event handlers

in custom directives, 136-137

for custom events, 151

positioning, 145

\$event keyword, 115

events

binding to controllers, 113-120

browser events, 145

custom events, 150

broadcasting, 151

emitting, 150-151

handling with listeners, 151

implementing in nested controllers,
151-152

definition of, 145

HTML5 drag and drop events,
204-206

scope change events, 146

implementing in controllers, 148

\$watch() method, 146-147

\$watchCollection() method, 147

\$watchGroup() method, 147

user interaction events, 145-146

example code listings. See listings

exception handling. See error handling

\$ExceptionHandler service, 158

expand_item.html listing, 214

expand_list.html listing, 214

expandable/collapsible elements example application, 212-215

expand.html listing, 215

expand.js listing, 213

Express web servers, building, 8-10

{{expression}} syntax, 69, 77-78

expressions

basic expressions, 79-80

in data model, 78-79

definition of, 38, 77

filters in, 87

JavaScript in, 85

scope interactions, 81-82

expressions_basic.html listing, 80

expressions_basic.js listing, 80

expressions_javascript.html listing, 85

expressions_javascript.js listing, 85

expressions_scope.html listing, 83

expressions_scope.js listing, 82

extend() global API, 43

extending

- form elements with directives, 104-105
- HTML, 208-210

external templates, testing custom directives, 230

F

factory() method, 56**factory service, 184, 185-186****filter() method, 56, 94-95****filter_customer.js listing, 95****filter_sort.html listing, 93****filter_sort.js listing, 92****filter:exp:compare filter, 87****filters**

- built-in filters, 87-90
- custom filters, 94-95
- definition of, 77
- sorting and ordering with, 91-92
- syntax, 87
- unit testing, 227-228

filters.html listing, 90**filters.js listing, 90****finally keyword, 32-33****find() method, 48****first.html listing, 47****first.js listing, 47****focus events, 115****folder structure, creating applications, 44****forEach() global API, 43****for/in loops, 18****for loops, 17-18****form elements, extending with directives, 104-105****fromCharCode() method, 26****fromJson() global API, 43****function keyword, 19****functions**

- anonymous functions, 21
- defining, 19-20
- definition of, 19
- in expressions, 81-82
- passing values to, 20
- returning values from, 20-21

G

global APIs, list of, 42-44**global lookup, 225****global variable scope, 22****greater than operator, 14****greater than or equal to operator, 14**

H

hasClass() method, 48**hash() method, 176****headers property, 160****hierarchy of scopes, 73****host() method, 176****HTML documents**

- bootstrapping, 42
- extending HTML, 208-210

html() method, 48**HTTP servers, implementing and accessing, 161-163****\$http service, 158-163**

- configuring, 160
- HTTP server implementation and access, 161-163
- response callback functions, 161
- shortcut methods, 159
- usage with custom services, 192-194

I

IDE (Integrated Development Environment), setup, 5-6
if statements, 15
images, zoom view field example application, 208-210
increment operator, 12
indexOf() method
 arrays, 28, 30
 strings, 26-27
inherited scope, 128
inheritedData() method, 49
\$inject property, 57-58
inject_builtIn.html listing, 58
inject_builtIn.js listing, 58
inject_custom.html listing, 60
inject_custom.js listing, 60
injection. See dependency injection
injector() method, 49
injectors, definition of, 54
input directive, 104
input.checkbox directive, 104
input.date directive, 104
input.dateTimeLocal directive, 104
input.email directive, 104
input.month directive, 104
input.number directive, 104
input.radio directive, 104
input.text directive, 104
input.time directive, 104
input.url directive, 104
input.week directive, 104
installing Node.js, 6-7
Integrated Development Environment (IDE), setup, 5-6
\$interpolate service, 158
interrupting loops, 18-19

\$interval service, 158, 168-169

isArray() global API, 43
isDate() global API, 43
isDefined() global API, 43
isElement() global API, 43
isFunction() global API, 43
isNumber() global API, 43
isObject() global API, 43
isolate scope, 129-130
isolateScope() method, 49
isString() global API, 43
isUndefined() global API, 43
iterating through arrays, 30

J

Jasmine, 223

JavaScript

in AngularJS expressions, 85
 animation in, 171-172
 arrays
 adding/removing items, 30
 combining, 29
 converting to strings, 30
 defining, 28
 iterating through, 30
 length of, 28
 methods, 28
 searching for items, 30
 data types, list of, 11-12
 development environment, setup, 5-6
 error handling
 finally keyword, 32-33
 throwing errors, 32
 try/catch blocks, 31-32
 expressions, AngularJS expressions versus, 79

functions

- anonymous functions, 21
- defining, 19-20
- definition of, 19
- passing values to, 20
- returning values from, 20-21

integration with AngularJS, 40-41**loops**

- definition of, 16
- do/while loops, 17
- for/in loops, 18
- for loops, 17-18
- interrupting, 18-19
- while loops, 16-17

Node.js

- building Express web server, 8-10
- running code on, 7
- setup, 6-7

objects

- defining custom, 23-24
- definition of, 22
- prototyping, 24-25
- syntax, 23

operators

- arithmetic operators, 12-13
- assignment operators, 13
- comparison operators, 14
- definition of, 12
- if statements, 15
- logical operators, 14
- switch statements, 15-16

strings

- concatenating, 27
- defining, 25
- escape codes, 25
- length of, 26
- methods, 26

- replacing words in, 27
- searching for substrings, 27
- splitting into arrays, 27
- testing. *See* applications, testing
- variables
 - defining, 10-11
 - scope, 22

join() method, 28, 30**jQuery**

- accessing directly, 49-50
- animation, 171-172
- full version usage example, 208-210
- integration with AngularJS, 40-41
- jQuery lite versus, 40-41

jQuery lite

- accessing directly, 49-50
- definition of, 48
- jQuery versus, 40-41
- methods, 48-49

json filter, 87

K

keyboard events, 117-118**keywords**

- break, 16, 18-19
- continue, 19
- finally, 32-33
- function, 19
- new, 23
- return, 20-21
- var, 10

L

large_title.html listing, 104**lastIndexOf() method**

- arrays, 28
- strings, 26

length

- of arrays, 28
- of strings, 26

less than operator, 14**less than or equal to operator, 14****life cycle phases**

- AngularJS, 39-40
- scopes, 71-72

limitTo:limit filter, 87**link() function, 130-132****link property, 123****listeners, handling custom events, 151****listings**

- animate.css, 174
- config_run_blocks.html, 63
- config_run_blocks.js, 63
- Defining global and local variables in JavaScript, 22
- directive_angular_include.html, 103
- directive_angular_include.js, 103
- directive_bind.html, 112
- directive_bind.js, 112
- directive_custom_dom.html, 135
- directive_custom_dom.js, 134
- directive_custom_photos.js, 141
- directive_custom_zoom.html, 139
- directive_custom_zoom.js, 137
- directive_custom.html, 142
- directive_focus_events.html, 116
- directive_focus_events.js, 116
- directive_form.html, 108
- directive_form.js, 105
- directive_keyboard_events.html, 118
- directive_keyboard_events.js, 118
- directive_mouse_events.html, 121
- directive_mouse_events.js, 120
- dragdrop.html, 206

dragdrop.js, 204

expand_item.html, 214

expand_list.html, 214

expand.html, 215

expand.js, 213

expressions_basic.html, 80

expressions_basic.js, 80

expressions_javascript.html, 85

expressions_javascript.js, 85

expressions_scope.html, 83

expressions_scope.js, 82

filter_customer.js, 95

filter_custom.html, 96

filter_sort.html, 93

filter_sort.js, 92

filters.html, 90

filters.js, 90

first.html, 47

first.js, 47

inject_builtin.html, 58

inject_builtin.js, 58

inject_custom.html, 60

inject_custom.js, 60

large_title.html, 104

my_photos.html, 143

pane.html, 201

rating.html, 219

rating.js, 218

scope_controller.html, 67

scope_controller.js, 67

scope_events.html, 153

scope_events.js, 152

scope_hierarchy.html, 74

scope_hierarchy.js, 74

scope_template.html, 70

scope_template.js, 69

scope_watch.html, 149

scope_watch.js, 148
 server.js, 9
 service_animate.html, 173
 service_animate.js, 172
 service_cache.js, 165
 service_cookie.html, 167
 service_cookie.js, 167
 service_custom_censor.html, 187
 service_custom_censor.js, 186
 service_custom_db_access.js, 194
 service_custom_db.html, 196
 service_custom_db.js, 195
 service_custom_time.html, 190
 service_custom_time.js, 189
 service_db_server.js, 192
 service_http.html, 164
 service_http.js, 163
 service_location.html, 178
 service_location.js, 177
 service_server.js, 162
 small_title.html, 104
 tabbable.html, 202
 tabbable.js, 200
 tabs.html, 201
 welcome.css, 9
 welcome.html, 9
 zooming.html, 210
 zooming.js, 209
 zoomit.html, 210

loading

- angular.js library file, 45
- modules, 45

local variable scope, 22

\$locale service, 158

\$location service, 158, 176-177

\$log service, 158

logical operators, list of, 14

loops

- definition of, 16
- do/while loops, 17
- for/in loops, 18
- for loops, 17-18
- interrupting, 18-19
- while loops, 16-17

lowercase filter, 87

lowercase() global API, 43

M

match() method, 26

message property, 32

method property, 160

methods

- for arrays, 28-29
- deferred responses, 180-181
- definition of, 23
- \$http service, 159
- in jQuery lite, 48-49
- \$location service, 176-177
- for strings, 26

model mutation phase (scopes), 72

modules

- adding as dependency, 57-58
- configuration phase
 - adding configuration blocks, 61-62
 - implementing configuration blocks, 62-63
- creating providers
 - service providers, 56-57
 - specialized providers, 56
- defining, 54-55
- definition of, 37, 53-54
- loading, 45

- run phase
 - adding run blocks, 62
 - implementing run blocks, 62-63
- modulo operator, 12
- mouse events, 120
- multiElement property, 123
- multiplication operator, 12
- mutation observation phase (scopes), 72
- MVC (Model View Controller) model, 36
- my_photos.html listing, 143

N

- name property, 32
- nested controllers, implementing custom events, 151-152
- nested directives, 140-141, 199-202, 212-215
- new keyword, 23
- new operator, 224-225
- ng-app directive, 37, 42, 45, 100
- ng-bind directive, 109
- ng-bind-html directive, 109
- ng-bind-template directive, 109
- ng-blur directive, 114-115
- ng-change directive, 114
- ng-checked directive, 114
- ng-class directive, 109, 169-170
- ng-class-even directive, 109
- ng-class-odd directive, 109
- ng-click directive, 69, 114, 120
- ng-cloak directive, 100
- ng-controller directive, 66, 100
- ng-copy directive, 114
- ng-cut directive, 114
- ng-dblclick directive, 114
- ng-disabled directive, 109
- ng-focus directive, 114-115
- ng-form directive, 104
- ng-hide directive, 109, 169
- ng-href directive, 100
- ng-if directive, 109, 169
- ng-include directive, 100, 102, 169
- ng-init directive, 109
- ng-keydown directive, 114, 117-118
- ng-keypress directive, 114
- ng-keyup directive, 114, 117-118
- ng-list directive, 100
- ng-model directive, 68, 109
- ng-mousedown directive, 114, 120
- ng-mouseenter directive, 114, 120
- ng-mouseleave directive, 114, 120
- ng-mousemove directive, 114, 120
- ng-mouseover directive, 114
- ng-mouseup directive, 114, 120
- ng-non-bindable directive, 100
- ng-open directive, 100
- ng-options directive, 104
- ng-paste directive, 114
- ng-pluralize directive, 100
- ng-readonly directive, 100
- ng-repeat directive, 109, 169
- ng-required directive, 100
- ng-selected directive, 100
- ng-show directive, 109, 169
- ng-src directive, 100
- ng-srcset directive, 100
- ng-style directive, 109
- ng-submit directive, 114
- ng-swipe-left directive, 114
- ng-swipe-right directive, 114
- ng-switch directive, 109, 169
- ng-transclude directive, 100, 130

ng-value directive, 109

ng-view directive, 100, 169

Node.js

building Express web server, 8-10

running JavaScript code on, 7

setup, 6-7

normalization, 78

not equal to operator, 14

Not operator, 14

\$notify() method, 180

null data type, 12

number data type, 11

number[:fraction] filter, 87

numbers in basic expressions, 79-80

O

object literal data type, 12

objects

arrays

adding/removing items, 30

combining, 29

converting to strings, 30

defining, 28

iterating through, 30

length of, 28

methods, 28

searching for items, 30

defining custom, 23-24

definition of, 22

DOM objects. *See* DOM manipulation

error handling

finally keyword, 32-33

throwing errors, 32

try/catch blocks, 31-32

modules, defining, 54-55

properties, tracking, 147

prototyping, 24-25

strings

concatenating, 27

defining, 25

escape codes, 25

length of, 26

methods, 26

replacing words in, 27

searching for substrings, 27

splitting into arrays, 27

syntax, 23

off() method, 48

one() method, 48

\$on() method, 151

on() method, 48

operators

arithmetic operators, 12-13

assignment operators, 13

comparison operators, 14

definition of, 12

if statements, 15

logical operators, 14

switch statements, 15-16

orderBy:exp:reverse filter, 87

ordering with filters, 91-92

Or operator, 14

P

pane.html listing, 201

parameters, passing dependencies as, 226

params property, 160

parent() method, 48

parent scopes, emitting events to, 150-151

\$parse service, 158

passing

dependencies as parameters, 226

variables to functions, 20

- path() method, 176
- pop() method, 28
- port() method, 176
- positioning event handlers, 145
- prepend() method, 48
- priority property, 123
- prop() method, 48
- properties
 - custom directives, 123-125
 - definition of, 23
 - \$http service, 160-161
 - tracking, 147
- protocol() method, 176
- prototyping objects, 24-25
- Protractor, 223, 230
- provider() method, 57
- providers
 - creating
 - service providers, 56-57
 - specialized providers, 56
 - definition of, 54
 - implementing, 57-58
 - injecting into controllers, 58-59
- push() method, 28

Q

- \$q service, 158, 180-181**
 - usage with custom services, 192-194

R

- rating.html listing, 219
- rating.js listing, 218
- ratings example application, 217-219
- ready() method, 48
- registry requests, 225-226

- \$reject() method, 180**
- remove() method, 48
- removeAttr() method, 48
- removeClass() method, 48
- removeData() method, 48
- removing items from arrays, 30
- replace() method, 26-27, 176
- replaceWith() method, 48
- replacing words in strings, 27
- require() method, 7
- require property, 123
- \$resolve() method, 180**
- \$resource service, 158**
- response callback functions, \$http service, 161
- responseType property, 160
- responsibilities, separation of, 40
- restrict property, 123, 126-127
- return keyword, 20-21
- returning variables from functions, 20-21
- reusability of custom services, 188-189
- reverse() method, 28
- root element, defining, 45
- root scope, applications and, 65-66
- \$rootElement service, 158**
- \$rootScope service, 158**
- \$route service, 158**
- \$routeParams service, 158**
- run blocks
 - adding, 62
 - implementing, 62-63
- run phase (modules)
 - adding run blocks, 62
 - implementing run blocks, 62-63
- running JavaScript on Node.js, 7
- runtime phase (AngularJS life cycle), 40

S

sample code listings. See listings

\$sanitize service, 158

\$sce service, 158

scope change events, 146

implementing in controllers, 148

\$watch() method, 146-147

\$watchCollection() method, 147

\$watchGroup() method, 147

scope destruction phase (scopes), 72

scope() method, 49

scope property, 123, 128-130

scope_controller.html listing, 67

scope_controller.js listing, 67

scope_events.html listing, 153

scope_events.js listing, 152

scope_hierarchy.html listing, 74

scope_hierarchy.js listing, 74

scope_template.html listing, 70

scope_template.js listing, 69

scope_watch.html listing, 149

scope_watch.js listing, 148

scopes

in AngularJS

definition of, 37

implementing, 46

configuring for custom directives,
128-130

custom events, 150

broadcasting, 151

emitting, 150-151

handling with listeners, 151

implementing in nested controllers,
151-152

as data model, 65

back-end server data, 71

controllers and, 66-67

expressions and, 78-79, 81-82

life cycle phases, 71-72

root scope, 65-66

scope hierarchy, 73

templates and, 68-69

object properties, tracking, 147

sharing, 212-215

testing controllers, 226-227

of variables, 22

tracking, 146-147

script directive, 100

search() method, 26, 176

searching

arrays for items, 30

strings for substrings, 27

select directive, 104

separation of responsibilities, 40

server data, scopes and, 71

server.js listing, 9

servers. See web servers

service() method, 56

service providers, creating, 56-57

service service, 184-185

implementation example, 185-186

service_animate.html listing, 173

service_animate.js listing, 172

service_cache.js listing, 165

service_cookie.html listing, 167

service_cookie.js listing, 167

service_custom_censor.html listing, 187

service_custom_censor.js listing, 186

service_custom_db_access.js listing, 194

service_custom_db.html listing, 196

service_custom_db.js listing, 195

service_custom_time.html listing, 190

service_custom_time.js listing, 189

service_db_server.js listing, 192

service_http.html listing, 164

service_http.js listing, 163

service_location.html listing, 178

service_location.js listing, 177

service_server.js listing, 162

services

built-in services

- \$anchorScroll, 158
- \$animate, 158, 169-172
- \$cacheFactory, 158, 165
- \$compile, 158
- \$cookie, 166-167
- \$cookies, 158
- \$cookieStore, 166-167
- \$document, 158
- \$exceptionHandler, 158
- \$http, 158-163
- \$interpolate, 158
- \$interval, 158, 168-169
- list of, 158
- \$locale, 158
- \$location, 158, 176-177
- \$log, 158
- \$parse, 158
- \$q, 158, 180-181
- \$resource, 158
- \$rootElement, 158
- \$rootScope, 158
- \$route, 158
- \$routeParams, 158
- \$sanitize, 158
- \$sce, 158
- \$swipe, 158
- \$templateCache, 158
- \$timeout, 158, 168-169
- usage with custom services, 192-194
- \$window, 158, 166

custom services

- constant service, 184
- database access service example, 192-194
- factory service, 184
- implementation example, 185-186
- service service, 184-185
- time service example, 188-189
- types of, 183
- value service, 183-184

definition of, 38, 157

sharing scopes, 212-215

shift() method, 28

slice() method

- arrays, 28
- strings, 26

small_title.html listing, 104

sort() method, 28

sorting with filters, 91-92

specialized providers, creating, 56

splice() method, 28

split() method, 26-27

splitting strings into arrays, 27

star ratings example application, 217-219

string data type, 11

strings

- in basic expressions, 79-80
- concatenating, 13, 27
- converting arrays to, 30
- defining, 25
- escape codes, 25
- length of, 26
- methods, 26
- replacing words in, 27
- searching for substrings, 27
- splitting into arrays, 27

substr() method, 26
substring() method, 26
substrings, searching for, 27
subtraction operator, 12
\$swipe service, 158
switch statements, 15-16

T

tabbable.html listing, 202
tabbable.js listing, 200
tabbed view example application, 199-202
tabs.html listing, 201
template property, 123-124
\$templateCache service, 158
templates, 37

- adding controller to, 46
- components of, 77-78
- defining with custom directives, 124-126
- definition of, 77
- directive support, 100-102
- expressions in, 78-79
 - basic expressions, 79-80
 - JavaScript in, 85
 - scope interactions, 81-82
- external templates, testing custom directives, 230
- filters in, 87
 - built-in filters, 87-90
 - custom filters, 94-95
 - sorting and ordering with, 91-92
- scopes and, 68-69
- star ratings example application, 217-219

templateUrl property, 123, 126
terminal property, 123

testing

- applications, 223
 - end-to-end testing, 230-231
 - evaluating platforms, 223
 - unit testing, 224-230
- JavaScript on Node.js, 7

text() method, 48
textarea directive, 104
throw statement, 32
throwing errors, 32
time service example, 188-189
timeout property, 160
\$timeout service, 158, 168-169
timers, 168-169
toggleClass() method, 48
toJson() global API, 43
toLowerCase() method, 26
toString() method, 28
toUpperCase() method, 26
transclude property, 123-124, 130
transclusion, testing custom directives, 229-230
transformRequest property, 160
transformResponse property, 160
triggerHandler() method, 48
try/catch blocks, 31-32
type property, 123

U

unbind() method, 48
unit testing, 224-230

- controllers, 226-227
- custom directives, 228
 - with external templates, 230
 - with transclusion, 229-230

- dependency injection and, 224-226
 - global lookup, 225
 - new operator, 224-225
 - passed parameters, 226
 - registry requests, 225-226
- filters, 227-228
- unshift() method, 28**
- uppercase filter, 87**
- uppercase() global API, 43**
- url() method, 176**
- url property, 160**
- user interaction events, 145-146**

V

- val() method, 48**
- value() method, 56**
- value service, 183-184**
 - implementation example, 185-186
- valueOf() method**
 - arrays, 28
 - strings, 26
- values**
 - passing to functions, 20
 - returning from functions, 20-21
- var keyword, 10**
- variables**
 - data types, list of, 11-12
 - defining, 10-11
 - in expressions, 81-82
 - passing to functions, 20
 - returning from functions, 20-21
 - scope, 22
 - tracking, 146-147

W

- \$watch() method, 72, 146-148**
- \$watchCollection() method, 147-148**
- watcher registration phase (scopes), 72**
- \$watchGroup() method, 147-148**
- web browsers**
 - alerts, implementing, 166
 - cookies, 166-167
 - events, 145
 - purpose of, 6
- web servers**
 - Express web servers, building, 8-10
 - purpose of, 6
- WebDriver, 230**
- websites, AngularJS, 42**
- welcome.css listing, 9**
- welcome.html listing, 9**
- while loops, 16-17**
- \$window service, 158, 166**
- withCredentials property, 160**
- words, replacing in strings, 27**
- wrap() method, 48**

X–Y

- xsrCookieName property, 160**
- xsrHeaderName property, 160**

Z

- zoom view field example application, 208-210**
- zooming.html listing, 210**
- zooming.js listing, 209**
- zoomit.html listing, 210**