AngularJS

Step by Logical Step

By Nicholas Johnson

version 0.1.4 - beta

Image credit: San Diego Air and Space Museum Archives

Welcome to Angular!

Angular gives you **super powers**. You will be able to see through walls, climb tall buildings with your hands and shoot laser beams from your eyes.

With Angular you will be **unstoppable**. You will be able to pull in live streams of data from across the Internet and mash them into your page like a **ninja**. You will be able to construct your page in modules that **download asynchronously**. You will be able to create tabs, dynamic forms, live comment streams and other features in a **fraction of the time** you might have spent using jQuery. You will be able to handle events with NO WIRING CODE. You will be able to AJAX in content and integrate it onto your page with **stealth and grace**.

So what exactly is Angular?

Angular is a front end JavaScript framework that helps you build single page web applications (SPAs). It comes packaged as a single JavaScript file which you include on your web page. It's purely front end, and says nothing about your server. Your server only needs to be able to ship out JSON to be able to talk to Angular.

Angular is flatfiles

This means you don't even need your own server to create an Angular app. You could use IMDb as a back end. You could use the Facebook graph API. You could use Parse or Firebase. Any API can be a back end for Angular. Your app becomes a series of flatfiles that in theory you can host anywhere you like.

This is a foundational principle of Angular. It was originally built to allow designers to add functionality to their page with little or no coding required.

Angular is an HTML5 compiler

Most other JavaScript frameworks give you functions you can call to transform your HTML. Angular instead gives you a compiler. Your HTML5 drops into the top of the compiler, is read top to bottom (depth first) and a web application falls out the bottom.

This means that Angular is **template driven**. Your HTML5 drives your app and tells your JAvaScript what to do. This is a radical inversion of control that may make some JavaScript developers rather cross. Angular is the polar opposite of unobtrusive JavaScript. Your HTML5 and your script are intimately connected and your HTML5 literally drives the app (via the Angular framework).

Giving up on unobtrusive JavaScript can seem like a big leap for some, stick with it, it'll be worth your while.

We extend the Angular compiler by writing compiler directives

You can extend the capabilities of the Angular compiler by writing directives. Directives are functions that match specific DOM elements. When the compiler encounters a DOM node that matches a directive, it calls the directive function and passes it the DOM node, the $scope (which is a data storage object), and a few other useful bits and pieces.

The directive can then modify the DOM, shell out to a service or other component, set up a new $scope, set up watch functions, and many other useful things.

Angular comes with a full compliment of built in directives. You will find that you can create many

simple apps with no JavaScript at all.

Angular is lightweight MVC with Data binding

Angular is a data binding framework using an MVC (Model View Controller) pattern. It allows you to store values in a **Model**, for example an item in a shopping cart. It then automatically keeps that value synchronised with the **View**, which in Angular is an HTML page. When your model updates, your view also updates as if by magic. The model can be pulled from anywhere on the Internet, it's just a JSON object.

A **Controller**, which itself is just a function, helps us hook the **Model** and **View** together.

Angular Wires your data into your template all by itself

Drop any data into $scope from any source, then write a template that assumes your data exists. Hey presto, it works! When your data changes your front end updates all by itself, no wiring code is required.

Of course, there is no magic in computer science, just extreme cleverness. Angular can use plain old JSON objects for data storage because of the digest cycle. More on this later.

What is this book?

This book is your guide to Angular. It's designed to complement my Angular course which you are very welcome to come on. Best of luck!

Who you are

You probably have some knowledge of JavaScript, perhaps a lot of knowledge! You have probably used jQuery for DOM manipulation. You might be starting to feel a little constrained by what JQuery can do.

You might have played with one of the other MV* frameworks like Backbone, but you've heard that Angular is something special. It is, you're going to like it.

About the author

Hi, I'm Nicholas Johnson. I'm a software developer and trainer working mainly with Ruby and JavaScript. I'm the developer behind Streetbank.com. I love JavaScript and have been teaching it for many years. Looking forward to working with you...

And now onwards to Angular!

Why Angular Rocks, and why you should care

1. Round trip data binding

Update your JSON model via AJAX, or in response to a user event and your web page updates automatically. This is the magic of the digest cycle. More on this later.

2. Your models are just JSON, Your controllers are just functions.

Unlike most other data binding frameworks, your models are just plain JSON objects. Manipulate them just like any other JSON object. No getters and setters are required. Likewise, your controllers are just functions, not special objects with special prototypes.

3. Dependency Injection with named function parameters

Angular magically extends the capabilities of JavaScript with named function parameters. Name your variables correctly, and Angular will automatically populate them for you with objects that you can use. You just have to write your functions in sensible modules. Angular will take care of ensuring your functions are available where you need them.

4. Built in Template language

Angular comes with it's own templating language that looks a lot like Handlebars. Your front end is just HTML with special extensions.

5. Built in AJAX

Pull in any JSON feed from anywhere on the Internet. Angular will automatically integrate it into your template for you. This just happens.

6. The Digest Cycle

Angular watches your data for you. Whenever your data changes, Angular figures out what has changed, executes any listeners, and recompiles the changed parts of the DOM. Angular does this automatically, so you don't need to write any wiring code.

Angular vs. JQuery

An overview of the philosopical and structural differences between jQuery and Angular.

Angular and JQuery adopt radically different ideologies. If you're coming from jQuery you may find some of the differences surprising. Angular will probably make you quite angry. Push through this, it'll be worth your time.

Semantic HTML vs. Angular Templates

With JQuery your HTML page should contain semantic meaningful content. If the JavaScript is turned off (by a user or search engine) your content remains accessible.

Angular treats your HTML page as a template. The template is compiled by the Angular app. With JavaScript turned off your template will not be compiled and your page may be almost completely blank.

Modern screen readers and Google will execute JAvaScript so this isn't as big a problem as it once was.

Unobtrusive JavaScript vs. JavaScript Application

JQuery is unobtrusive. Your JavaScript is linked in the header, and this is the only place it is mentioned. The JavaScript wraps round the DOM like a snail on a twig, making changes as required. Onclick attributes are very bad practice.

One of the first things your will notice about Angular is that it's highly obtrusive. Your HTML will be littered with ng attributes, which are essentially onClick attributes on steroids. These are directives, and are one of the ways in which the template is hooked to the model. Your HTML makes no sense without the JavaScript backing it because the meaning of the page resides in the model, rather than the view.

Separation of concerns vs. MVC

Front end Separation of concerns is a pattern that has grown up over many years of web development for a variety of reasons including SEO, accessibility and browser incompatibility. It looks like this:

1. HTML - Semantic meaning. The HTML should stand alone.
2. CSS - Styling, without the CSS the page is still readable.
3. JavaScript - Behaviour, without the script the content remains.

Angular pays no heed to this at all and goes its own way. Angular instead implements an MVC pattern.

1. Model - your models contains your semantic data. Models are usually JSON objects.
2. View - Your views are written in HTML. The view is usually not semantic because your data lives in the model.
3. Controller - Your controller is a JavaScript function which hooks the view to the model. Depending on your app, you may or may not need to create a controller. You can have many controllers on a page.

Plugins vs. Directives

Plugins extend jQuery. Angular Directives extend HTML.

In jQuery we define plugins by adding functions to the jQuery.prototype. We then hook these into the DOM by selecting elements and calling the plugin on the result. The idea is to extend the capabilities of JQuery.

In Angular, we define directives. A directive is a function which returns a JSON object. This object tells Angular how to change the DOM. Directives are hooked in to the template using either attributes or elements, which you invent. The idea is to extend the capabilities of HTML with new attributes and elements.

Closure vs. $scope

JQuery plugins are created in a closure. Privacy is maintained within that closure. It's up to you to correctly maintain your scope chain within that closure and ensure the data you need is accessible.

Angular has $scope objects. These are special objects created and maintained by Angular in which you store your model. Certain directives will spawn a new $scope, which can optionally inherit from its wrapping $scope. The $scope object is accessible in the controller and the view.

Manual DOM changes vs. Automatic Data Binding

In jQuery you make your DOM changes by hand. You construct new DOM elements programatically.

In Angular you can do this too, but you are encouraged to make use of data binding. Change your model and your DOM will automatically update, no intervention required. With practice you will find you lean into templates more and more and make hardly any DOM changes directly.

Further Reading

http://stackoverflow.com/questions/14994391/how-do-i-think-in-angularjs-if-i-have-a-jquery-background/23606512#23606512

How does Angular do MVC?

Learning objectives

1. A high level awareness of what an MVC pattern is.
2. Angular models are JSON objects.
3. Angular views are HTML5 files.
4. A controller hooks them together.
5. Controllers are just functions.
6. The scope object is an actual object provided by Angular in which we store models.

A reliable old pattern

MVC is an sturdy old design pattern in Computer science. The idea is to keep your data separate from your presentation. Your data resides in a model, which is usually an object of some type. Your view represents your front end and can be swapped out if you want to change your presentation. A controller wires the two together.

Wikipedia on MVC

The Angular Model

In angular a model can be anything, a JSON object, an array, or even just a variable. To make an ordinary JavaScript object into a model, you just need to assign it as an attribute of a variable called $scope.

The View

An angular view is an HTML5 file. It optionally contains special directives and Angular code which are compiled into more HTML by Angular. Angular treats your view as code, literally as instructions to the compiler.

In Angular the view drives the app. Angular will compile the view and pull in components as they are needed.

The Controller

In Angular, controllers are just functions. You turn them into controllers, simply by treating them as such. Angular will take care of providing them with the dependencies they need.

$scope

The $scope object is a special object provided by Angular which is shared by the controller and the view. You shouldn't confuse it with JavaScript scope, which is different but works in a similar way.

The $scope object is used to store models. The same $scope object is available in the view and the controller. You can use it to pass data between the two.

$Scope objects inherit from each other, so you might store an object in the global $scope object, or in a nested $scope object which only applies in a particular loop for example.

Crazy lightweight?

Angular's interpretation of MVC is ridiculously lightweight compared to say Backbone. In fact, when I first saw it I couldn't believe it could work like that, I thought there must be some mistake. My question was, where is the code that makes it all work? It's all in there, in the Angular framework.

Questions

Try to answer these questions for yourself before moving on:

1. What does MVC stand for?
2. What is a view in Angular?
3. What is an Angular model?
4. What is an Angular controller?
5. What do we store in the $scope object?

Answers

1. MVC is Model View Controller
2. A view in Angular is just an HTML5 file which contains special tags.
3. Usually just a JSON object.
4. A controller is just a function.
5. The $scope object is for storing models. We can also put functions in there which become available to our templates.

Templates (Views)

In this section we will see how to create a very simple view in Angular.

The front end of an Angular app is the view, or the template. In Angular a template is just an ordinary HTML5 file. Angular takes this HTML file, scans it for directives and template commands, and compiles it to produce your finished application.

Essentially **Angular Compiles your DOM.**

Treating the DOM as a template like this lets us do several things:

1. We can be radically productive. We just write HTML, and assume the presence of data.
2. We can include special attributes and elements which act as instructions to the compiler.
3. We can write Angular script right into the page.
4. We can use the nested structure of the DOM to provide variable scope. Models can be scoped to particular parts of the DOM. No Way!
5. We can include additional templates on the page really easily.

Downsides

Of course there are several downsides to this approach

1. Your HTML5 file is no longer semantic.
2. Without JavaScript your page will not work.
3. You will need to do some work to make your site work with search engines.

A Simple Template

An Angular template looks like this:

```
1   <!DOCTYPE html>
2   <html>
3     <head>
4       <title></title>
5       <script src="angular.js"></script>
6     </head>
7     <body ng-app>
8
9       {{"Hello" + "World"}}
10
11    </body>
12  </html>
```

There are three things to notice here:

Notice we're importing Angular, and no other scripts

We can import Angular in the head of our page, or at the bottom of the body. In these examples I'll import Angular in the head. If you're going all out for speed you should import it at the bottom of the body.

I'm not importing anything else here. I don't need any more code than this because the template itself is code.

Notice the ng-app attribute.

The ng-app attribute is a "directive". It tells Angular to do something. In this case it tells Angular to treat the contents of the body element as an AngularJS app and turn on the compiler. You can put the ng-app directive on any DOM element you like, and you can have multiple ng-app directives on the page. Typically you would put the ng-app directive on the html element or on the body.

We will see more directives shortly.

Notice HelloWorld

This curly brace syntax will be familiar to you if you have used Handlebars or Mustache.

The code between the braces is written in Angular Templating Language which looks quite a lot like JavaScript with a few extra features thrown in.

Anything we place between the curly braces will be evaluated.

Questions

1. What does the ng-app directive do?
2. What happens to content between double braces ?
3. Is the content between the double braces JavaScipt?

Answers

1. ng-app tells Angular that we are now inside an Angular app.
2. Content between is evaluated and added to the DOM
3. The content between looks a lot like JavaScript, but is in fact Angular templating language.

Exercise

1. Enter the code above to create a hello world application. Run it in your own browser.

Data Binding

Data binding is where we "bind" a model to the DOM. When the model updates the DOM will update. When the DOM updates (e.g. a form field is edited) the model will update.

Here's an example of data binding in action:

```
<!DOCTYPE html>
<html>
  <head>
    <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.15
  </head>
  <body ng-app>
    <h1>{{hello}}</h1>
    <input ng-model="hello" />
  </body>
</html>
```

This example will bind the input element to a model called "hello". When you type into the input element, the template updates itself.

But what? Where is all the code?

When I first saw this I actually felt a little bit scandalised. Where is all the code? It can't be that easy, surely. Well there is quite a lot going on under the hood here which Angular is taking care of for you.

Where is the model?

The model is the content of the "hello" variable. This might seem kind of crazy to you that a model can be just a string. It really is as easy as this though.

More commonly we use JSON objects as models, but it's possible to use any type of object or primititive. Even a String or number is acceptable.

The "hello" variable is actually an attribute of a special object called $scope. $scope is used as a data store and is available in your model, and also optionally in your controller.

Some directives create new scopes, so you get some privacy.

How can this work?

When Angular compiles the DOM it looks for angular template code between braces like this:

```
{{ }}
```

It then create a watcher on the current $scope object which can check to see if the value is dirty and execute a function if it is.

In the case above, the watcher will check to see if $scope.hello is dirty and execute a function to modify the DOM and insert the correct value if it is.

Whenever a value changes on a model, Angular cycles through all of the watchers on every $scope looking for dirty values. It will continue doing this until all the $scope objects are clean.

Exercise

1. Enter the above code and get it running in your browser.
2. Add a second input element, also bound to the "hello" model. Notice how these two are now synchronised.
3. Add a third input element. Bind it to a world variable. Output the world variable using the syntax.

Built in Directives

Learning objectives

1. A directive is a "helper function"
2. Directives extend HTML

What are directives?

Directives are the hooks that connect your template to your code. They are compiler directives, instructions to the DOM compiler that tells it how to behave.

When Angular compiles your DOM, it looks for directives and uses them to guide the compilation process. A directive might loop over a collection, output a value, bind a form input to a model, AJAX in some content, validate a form, manage an image carousel, any DOM related task you can conceive of.

You can think of them as a bit like helper functions.

Directives are hooked in to the template using html attributes, element names or sometimes class names. Angular comes with a bunch of built in directives which allow you to build most things. You can of course write your own directives.

In fact the original version of Angular did not have user definable directives at all. Everything was done using the built in directives.

ng-model directive

We've just seen the ng-model directive in action. It binds a form element to a model on the $scope.

```
1    <input ng-model="toast" />
2    {{toast}}
```

Directives and valid HTML5

If you prefer your code valid, you can express Angular directives as HTML5 data attributes. Just prefix data- to any angular attribute, like this:

```
1    <input data-ng-model="toast" />
```

Your template will now pass HTML5 validation.

ng-directive vs ngDirective

HTML is case insensitive, so all Angular directive attributes are lower case. JavaScript uses camel case for function names by convention, so in the angular codebase, the directives are referred to using camel case. You will find both forms talked about.

Hero Concepts - Dependency Injection (DI)

Angular talks a lot about DI, which is a scary term for a trivial concept. In fact, you're probably using DI already. It means passing an object its instance variables, rather than having it construct them itself.

Instead of doing this:

```
var myFunction = function() {
  var a = 12;
  alert(a);
}
myFunction();
```

We would do this:

```
var myFunction = function(a) {
  alert(a);
}
myFunction(12);
```

Code written this way is more modular, less prone to spaghettification, and is easier to test as you can pass in mock objects. This is DI. It's pretty simple. Angular however takes DI to a whole 'nother level...

DI in Angular is Magic Beans

Angular takes DI to a parallel level of magic and mystery. Your function headers are actually inspected, and dependencies provided to them automagically. Look at this example:

```
<!DOCTYPE html>
<html ng-app>
  <head>
    <title></title>
    <script src="../angular/angular.js"></script>
    <script>
      var MyController = function( $scope ) {
        $scope.test = "hello";
      }
    </script>
  </head>
  <body ng-controller='MyController'>
    {{test}}
  </body>
</html>
```

As you can see, we're passing in the $scope variable in to the controller. When we run the script, this variable becomes available to us. The template also has access to $scope and will pull

variables from it.

Let's say we want to do something else though, what if we want to write to the log. We can just adjust the code and tell MyController to receive a log, like so.

```
var MyController = function( $log ) {
  $log.log("hey there!");
}
```

HEY WHAT!!

We changed the variable name in the header, and Angular passed in an object capable of doing logging. This looks like named parameters, but JavaScript doesn't support named parameters. Something fishy is going on here...

Angular knows what to pass in by looking at the name of the variable. But how!

The magical curtain of toString

Angular is inspecting the parameters of our function, by calling

```
MyFunction.toString()
```

All JavaScript objects have a toString method which returns a string version of the object. In the case of a function object, toString returns a string containing the function code. Angular inspects this, works out the variable names, and calls a factory method (in this case called LogProvider) to get the dependency.

Passing in multiple dependencies

So what if we want to pass in more than one dependency? Well we can do that as well:

```
var MyController = function( $scope, $log ) {
  $scope.test = "hello";
  $log.log("hey there!");
}
```

As you might expect, we can reverse the parameters and still get the same effect:

```
var MyController = function( $log, $scope ) {
  $scope.test = "hello";
  $log.log("hey there!");
}
```

Further Reading

Dependency injection is a "$25 term for a ¢5 concept

www.jamesshore.com/Blog/Dependency-Injection-Demystified.html

What is Dependency Injection on Stack Overflow

stackoverflow.com/questions/130794/what-is-dependency-injection

The Magic behind AngularJS Dependency Injection

www.alexrothenberg.com/2013/02/11/the-magic-behind-angularjs-dependency-injection.html

Testing with Protractor

Protractor is an end to end test framework for Angular. It gives you a JavaScript enabled web browser (Selenium) and a simple JavaScript SDK that lets you drive it. The syntax looks a lot like RSpec or Jasmine which you may be familiar with. Protractor is the standard test harness used by the Angular core team. You should consider making use of it in your projects.

Protractor runs as a Node module, so you'll need Node installed. You'll also need the Java SDK.

You get access to the following objects:

Browser

Tell your browser to hit specific paths and URLs, like this:

```
1    browser.get('http://nicholasjohnson.com/');
```

Your browser will literally navigate to this address.

Element

Select elements on the page, for example form fields, buttons and links. Call methods to interact with them. For example:

```
1    element(by.id('gobutton')).click();
```

Tutorial

http://angular.github.io/protractor/#/tutorial