Angular.JS

Angular Overview

Angular.js is a highly opinionated, 5+ year old Javascript MV\* framework suited for rapid prototyping and single-page application UI development supported by Google. Java/ASP developers should feel most at home in Angular, as it borrows many object-oriented concepts from Java. Using Angular at scale is difficult because of a variety of performance issues unplanned for during its creation.

Angular 1.3 is the latest version which delivers many performance fixes, but ultimately cannot overcome the shortcomings of an expensive process called: 2-way data-binding, which in Angular is enabled everywhere by default. The process it uses to accomplish the data-binding is called dirty checking. Dirty checking tracks previous values of models used in a UI template. When a value changes, angular compares the previous value to the new value, and if the values are different (dirty), then it places the new value in the view asynchronously.

Angular is best for small prototype projects, and begins to show performance problems once 2000+ objects have been data bound on a page. Many advocates of Angular argue that most UI’s should stay away showing so many items on a page to keep the interface clean and simple anyway.[[1]](#footnote-1)

Before investing heavily in Angular, it is important to be aware that the Angular team at Google stated they would only support Angular 1.3 through the end of 2016 after announcing the development of Angular 2.0.[[2]](#footnote-2) There are a variety of alternatives worth considering, if you enjoy the style of Angular 1.x. One of the closest in style is called [Aurelia](http://www.aurelia.io/)[[3]](#footnote-3), which is being developed a former member of the Angular team now working on 2.0. There will likely be no supported migration path from version 1.x to Angular 2.0, which is scheduled to debut in 2016.

Angular emphasizes the power of declarative programming for UI. It tries to separate imperative programming from declarative syntax using the concepts of Models, Views, and Controllers. Views contain declarative code, while Models and Controllers contain imperative code. Angular stresses encapsulation of DOM manipulation into what it calls Directives (imperative wrappers for DOM manipulation). Angular also stresses UI testability using a dependency injection system, which brings typical server-side technologies, like view-dependent controllers to client-side web applications.

Models Overview

Angular models should contain the encapsulated definition of the structure of data which will be retrieved from a backend server/created and submitted to a persistent data-store.

To put is simply, the model for your Angular application lives on the server.

Angular merely provides gateways to access such a remote model. The alternative is to bring JS object model into the controller. (usually undesired for performance reasons.) The most commonly used gateways to server-side models are called factories and services. We will cover a brief implementation of a service:

angular.module(‘app’, []).service(‘ServiceName’,

function(dep1) {

return someService;

}, {$inject: [‘dep1’]});

All services are attached to an Angular module object, so that the dependency injection system can register and provide it when required. Business logic can (often should) be used to manipulate data in a service before handing it off to the View through a Controller.

Views Overview

It’s common in Angular to use templates to define Views, so that they can be shown on a single page without reloading. Templates are parsed and placed in the page. A template creating a list of authors would look something like this:

<ul>

<li ng-repeat=”author in authors”>

<input type=“checkbox” ng-model=“author.selected”>

</li>

</ul>

Anything prefaced with “ng” is usually a directive. The “ng-repeat” directive takes an object and repeats for as many items are available in an array of items, in this case “authors” contains the array. The “ng-model” in the input box, provides a declarative reference to the Javascript object the input to which it should be 2-way data-bound.

Angular comes with many built-in directives which provide much of the fundamental functionalities in Angular.

Controllers Overview

Controllers implement logic for views and templates, but are often used extensively for business logic. Writing a large amount of business logic in the controller is an anti-pattern in the framework, but sometimes seems unavoidable.

Declarative View Use:

<div ng-controller=“SomeController”>

Imperative Definition:

angular.module(‘app’, [])

.controller(‘ControllerName’, function($scope){

//Controller code goes here.

});

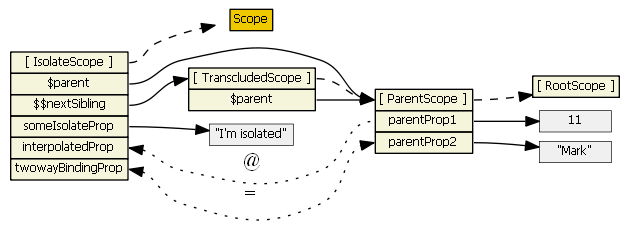
All controllers are attached to an Angular module object, so that the dependency injection system can register and provide it when required.

Controllers, properly implemented should only connect/wire-up a view/template to data objects. Most business logic should be written or referenced in the Models. Minimizing the number of objects/variables attached to the $scope is important for performance reasons.

Scope Overview

Scope is one of the trickiest concepts to master in Angular. Essentially, the scope is a special object that’s made available to a view template attached to a controller. Each controller has one scope for it’s corresponding view. This is easy to confuse with Javascript’s regular definition of scope, which is all the variables available within function scope.

Because your application will often have many controllers for a variety of views, there is also a parent scope.



Look at this wiki article for a much more detailed overview of scope’s relationship to controllers and views: https://github.com/angular/angular.js/wiki/Understanding-Scopes

Directives Overview

Directives are fundamentally extensions of HTML elements, and one of Angular’s core functional offerings. Whenever you manipulate the DOM, placing data, information, or interaction with you program logic on the view, a Directive will help.

A directive encapsulates a component of your site which can be reused over and over throughout your site. They are flexible, and have the ability to share the scope of their parent elements or contain their own scope. Here’s a simple example:

angular.module(‘app’, [])

.directive(‘DirectiveName’, function(){

//Directive code goes here.

return {

template: ‘<div a=“someOptions”></div>

}

});

Angular’s Github wiki has a great explaination and more examples of directives here: <https://github.com/angular/angular.js/wiki/Understanding-Directives>

Simple Hello World Example

<html xmlns:ng=<http://angularjs.org>>

<script src=<http://ajax.googleapis.com/ajax/libs/angularjs/1.3.7/angular.min.js>></script>

<body>

Your name: <input type=“text” name=”yourname” value= “World”/>

<hr/>

Hello {{yourname}}!

</body>

</html>

1. http://stackoverflow.com/questions/9682092/databinding-in-angularjs/9693933#9693933 [↑](#footnote-ref-1)
2. http://www.infoq.com/news/2014/10/angular-2-atscript [↑](#footnote-ref-2)
3. http://ilikekillnerds.com/2015/01/aurelia-vs-angularjs-round-one-fight/ [↑](#footnote-ref-3)