AngularJs Documentation

**AngularJs**

AngularJS is a **JavaScript framework**. AngularJS extends HTML attributes with **Directives**, and binds data to HTML with **Expressions**.

**AngularJS Expressions**

AngularJS binds data to HTML using Expressions.

* [Markup](https://docs.angularjs.org/api/ng/service/$interpolate) — The double curly brace notation {{ }} to bind expressions to elements is built-in AngularJS markup.

{{ expression }}

ng-bind="expression"

AngularJS expressions are like JavaScript expressions with the following differences:

* **Context:** JavaScript expressions are evaluated against the global window. In AngularJS, expressions are evaluated against a [scope](https://docs.angularjs.org/api/ng/type/$rootScope.Scope) object.
* **Forgiving:** In JavaScript, trying to evaluate undefined properties generates ReferenceError or TypeError. In AngularJS, expression evaluation is forgiving to undefined and null.
* **Filters:** You can use [filters](https://docs.angularjs.org/guide/filter) within expressions to format data before displaying it.

**AngularJS Modules**

You can think of a module as a container for the different parts of your app – controllers, services, filters, directives, etc.

There are several advantages to this approach:

* The declarative process is easier to understand.
* You can package code as reusable modules.
* The modules can be loaded in any order.

A module is created by using the AngularJS function angular.module

var app = angular.module("myApp", []);

**AngularJS Directives**

AngularJS lets you extend HTML with new attributes called Directives. AngularJS has a set of built-in directives which offers functionality to your applications. AngularJS also lets you define your own directives.

built-in directives, like ngBind, ngModel, and ngClass

**AngularJS ng-model Directive**

The ng-model directive binds the value of HTML controls (input, select, textarea) to application data. It supports Two-Way Binding.

**Data Binding**

Data binding in AngularJS is the synchronization between the model and the view. Any changes to the view are immediately reflected in the model, and any changes in the model are propagated to the view.



**HTML View**

The HTML container where the AngularJS application is displayed, is called the view.

**ng-bind** directive, which will bind the innerHTML of the element to the specified model property. Or display application data in the view.

**AngularJS Controllers**

AngularJS controllers control the data of AngularJS applications.

Controllers are "classes" or "constructor functions" that are responsible for providing the application behavior that supports the declarative markup in the template.

**AngularJS Scope**

Scope is an object that refers to the application model. The scope is the binding part between the HTML (view) and the JavaScript (controller).

**Scope characteristics**

Scopes provide APIs ($watch) to observe model mutations.

Scopes provide APIs ($apply) to propagate any model changes through the system into the view from outside of the "AngularJS realm" (controllers, services, AngularJS event handlers).

**$rootScope**

Every application has a single root scope. The rootScope is available in the entire application. All other scopes are descendant scopes of the root scope.

**AngularJS Filters**

Filters can be added in AngularJS to format data.

Ex. {{ lastName |filters }}, <li ng-repeat="x in names | orderBy:'country'">

**Dependency Injection** (<https://www.youtube.com/watch?v=NGrKfqhlSy4>)

Dependency Injection (DI) is a software design pattern that deals with how components get hold of their dependencies.

Drawbacks: Not flexible, Not good for testing

**Dependency Annotation**

AngularJS invokes certain functions (like service factories and controllers) via the injector. You need to annotate these functions so that the injector knows what services to inject into the function. There are three ways of annotating your code with service name information:

* Using the inline array annotation (preferred)

someModule.controller('MyController', ['$scope', 'greeter', function($scope, greeter) {

// ...

}]);

* Using the $inject property annotation

var MyController = function($scope, greeter) {

// ...

}

MyController.$inject = ['$scope', 'greeter'];

someModule.controller('MyController', MyController);

* Implicitly from the function parameter names (has caveats)

someModule.controller('MyController', function($scope, greeter) {

// ...

});

**AngularJS Services** (<https://www.youtube.com/watch?v=qhncBUvkRX4>)

In AngularJS, a service is a function, or object, that is available for, and limited to, your AngularJS application. You can use services to organize and share code across your app. Ex. $location, $http, $timeout.

To use an AngularJS service, you add it as a dependency for the component (controller, service, filter or directive) that depends on the service. AngularJS's [dependency injection](https://docs.angularjs.org/guide/di) subsystem takes care of the rest.

Why its needed:

For reusability

We can share data throughout the application

**Custom Services** (<https://www.youtube.com/watch?v=rlx1cf7qM0E&list=PL6n9fhu94yhWKHkcL7RJmmXyxkuFB3KSl&index=20>)

The service factory function generates the single object or function that represents the service to the rest of the application. The object or function returned by the service is injected into any component (controller, service, filter or directive) that specifies a dependency on the service.

myModule.factory('serviceId', function() {

var shinyNewServiceInstance;

// factory function body that constructs shinyNewServiceInstance

return shinyNewServiceInstance;

});

**$http** **$http** is an AngularJS service for reading data from remote servers.

**Providers**

Each web application you build is composed of objects that collaborate to get stuff done. In AngularJS apps most of these objects are instantiated and wired together automatically by the injector service.

It is similar as services and factory although they both come inside providers.

**$injector**

$injector is used to retrieve object instances as defined by provider, instantiate types, invoke methods, and load modules.

**Bootstrap**

This page explains the AngularJS initialization process and how you can manually initialize AngularJS if necessary.

**Automatic Initialization:** AngularJS initializes automatically upon DOMContentLoaded event or when the angular.js script is evaluated if at that time document.readyState is set to 'complete'.

 At this point AngularJS looks for the [ngApp](https://docs.angularjs.org/api/ng/directive/ngApp) directive which designates your application root. If the [ngApp](https://docs.angularjs.org/api/ng/directive/ngApp) directive is found then AngularJS will:

* load the [module](https://docs.angularjs.org/guide/module) associated with the directive.
* create the application [injector](https://docs.angularjs.org/api/auto/service/$injector)
* compile the DOM treating the [ngApp](https://docs.angularjs.org/api/ng/directive/ngApp) directive as the root of the compilation. This allows you to tell it to treat only a portion of the DOM as an AngularJS application.

**Manual Initialization**

If you need to have more control over the initialization process, you can use a manual bootstrapping method instead. Examples of when you'd need to do this include using script loaders or the need to perform an operation before AngularJS compiles a page.

Ex. angular.element(function() {

angular.bootstrap(document, ['myApp']);

});

**Interpolation**

Interpolation markup with embedded expressions is used by AngularJS to provide data-binding to text nodes and attribute values.

An example of interpolation is shown below:

<a ng-href="img/{{username}}.jpg">Hello {{username}}!</a>

**AngularJS Routing**

The ngRoute module helps your application to become a Single Page Application. If you want to navigate to different pages in your application with no page reloading, you can use the ngRoute module.

This application needs a container to put the content provided by the routing. This container is the **ng-view** directive.

**$routeProvider:** With the $routeProvider you can define what page to display when a user clicks a link.

**Application.config:** Work registered in the config method will be performed when the application is loading.

**Ng-include**

It inserts the separate files in the current web page by making Ajax request.

**$http.then vs $http.success**

The major difference between the 2 is that .then() call returns a promise (resolved with a value returned from a callback) while .success() is more traditional way of registering callbacks and doesn't return a promise.

Promise-based callbacks (.then()) make it easy to chain promises (do a call, interpret results and *then* do another call, interpret results, do yet another call etc.).

The .success() method is a streamlined, convenience method when you don't need to chain call nor work with the promise API (for example, in routing).

In short:

* .then() - full power of the promise API but slightly more verbose
* .success() - doesn't return a promise but offeres slightly more convienient syntax

**Promises:** A promise represents the eventual result of an operation. You can use a promise to specify what to do when an operation eventually succeeds or fails.

**MVC**

Model View Controller or MVC as it is popularly called, is a software design pattern for developing web applications. A Model View Controller pattern is made up of the following three parts:

Model − It is the lowest level of the pattern responsible for maintaining data.

View − It is responsible for displaying all or a portion of the data to the user.

Controller − It is a software Code that controls the interactions between the Model and View.