# What is AngularJS?

- AngularJS is a JavaScript framework. It is a library written in JavaScript.
- It provides **data-binding**, basic **templating directives**, **form validation**, **routing**, **deep-linking**, **reusable components**, **dependency injection**.
- It lets you use HTML as your template language and lets you extend HTML's syntax to express your application's components clearly.
- Angular's data binding and dependency injection eliminate much of the code you would otherwise have to write.
- AngularJS extends HTML attributes with Directives, and binds data to HTML with Expressions.
- Angular was built with the CRUD application in mind.
- Angular provides the following enhancement to HTML through directive:

  - ✓ Data binding, as in {{}}.
  - ✓ DOM control structures for repeating/hiding DOM fragments.
  - ✓ Support for forms and form validation.
  - ✓ Attaching new behavior to DOM elements, such as DOM event handling.
  - ✓ Grouping of HTML into reusable components.

# Angular Handles

- **Registering callbacks**: Registering callbacks clutters your code, making it hard to see the forest for the trees. It vastly reduces the amount of JavaScript coding you have to do, and it makes it easier to see what your application does.
- **Manipulating HTML DOM programmatically**: Manipulating HTML DOM is a cornerstone of AJAX applications, but it's cumbersome and error-prone. By declaratively describing how the UI should change as your application state changes, you are freed from low-level DOM manipulation tasks. Most applications written with Angular never have to programmatically manipulate the DOM, although you can if you want to.
- **Marshaling data to and from the UI**: CRUD operations make up the majority of AJAX applications' tasks. The flow of marshaling data from the server to an internal object to an HTML form, allowing users to

modify the form, validating the form, displaying validation errors, returning to an internal model, and then back to the server, creates a lot of boilerplate code. Angular eliminates almost all of this boilerplate, leaving code that describes the overall flow of the application rather than all of the implementation details.

- **Angular bootstraps your app**: Allows you to easily use services, which are auto-injected into your application in using dependency-injection.

## AngularJS History

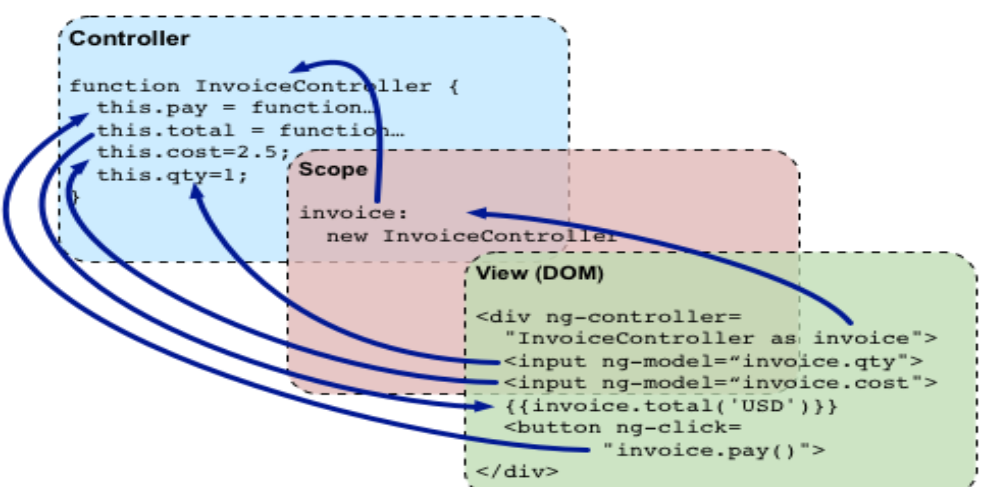
- AngularJS is quite new. Version 1.0 was released in **2012**.
- Misko Hevery, a Google employee, started to work on AngularJS in **2009**.
- The idea turned out very good, and the project is now officially backed by a Google development team.

**FirstPage.html**

```
<!DOCTYPE html>
<html>
<head>
    <title>First Angular App</title>
    <script src="angular.js"></script>
    <script>
        var module = angular.module("first", []);
        module.controller("FirstController",
             function ($scope)    {
              $scope.message = "My First AngularJS Page!";
              $scope.today = new Date().toString();
             }
        );
    </script>
</head>
<body ng-app="first">
    <h1 ng-controller="FirstController">
           {{ message }} <p/> {{today}}
    </h1>
</body>
</html>
```

# How is it executed?

- First the HTML document is loaded into the browser, and evaluated by the browser. At this time the AngularJS JavaScript file is loaded, the angular global object is created, and your JavaScript which registers controller functions is executed.
- Second, AngularJS scans through the HTML to look for AngularJS apps and views. When AngularJS finds a view it connects that view to the corresponding controller function.
- Third, AngularJS executes the controller functions and update (render) the views with data from the model populated by the controller. The page is now ready.
- Fourth, AngularJS listens for browser events (e.g. input fields being changed, buttons clicked, the mouse moved etc.). If any of these browser events require a view to change, AngularJS will update the view correspondingly. If the event requires that the corresponding controller function is called, AngularJS will do that too. Not all events require the controller function to be called, even if the view is updated as a result of the event.
- AngularJS views mix data from the model into an HTML template. You use AngularJS directives to tell AnguluarJS how to mix the data into the HTML template.



# Template

- Template is an HTML page.
- When Angular starts your application, it parses and processes this new markup from the template using the so-called "compiler".
- The loaded, transformed and rendered DOM is then called the "view".

# Directives

- The first kind of new markup are the so-called "directives".
- They apply special behavior to attributes or elements in the HTML.
- Angular also defines a directive for the input element that adds extra behavior to the element.
- The ng-model directive stores/updates the value of the input field into/from a variable.

| Directive | Description |
|---|---|
| ng-app | Defines the root element of an application. |
| ng-bind | Binds the innerHTML of HTML elements to application data. An alternative to {{ }} |
| ng-controller | Defines the controller object for an application. |
| ng-disabled | Binds application data to the HTML disabled attribute. |
| ng-hide | Hides or shows HTML elements. |
| ng-include | Includes HTML in an application. |
| ng-init | Defines initial values for an application. |
| ng-model | Binds the value of HTML controls to application data. |
| ng-repeat | Defines a template for each data in a collection. |
| ng-show | Shows or hides HTML elements. |
| ng-non-bindable | Tells Angular not to compile or bind the contents of the current DOM element. |
| ng-cloak | Prevents HTML template from being displayed in raw form while application is being loaded. |
| ng-href | Used to specify href for anchor elements. |
| ng-src | Used to specify source for image elements. |
| ng-if | Adds or removes the associated DOM tree. |
| ng-switch | Chooses one of the nested elements and makes it visible based on which element matches the value obtained from the evaluated expression |

```
<ANY ng-switch="expression">
  <ANY ng-switch-when="matchValue1">...</ANY>
  <ANY ng-switch-when="matchValue2">...</ANY>
  <ANY ng-switch-default>...</ANY>
</ANY>
```

## Directives ng-class, ng-class-even, ng-class-odd

The ngClass directive allows you to dynamically set CSS classes on an HTML element by data binding an expression that represents all classes to be added.

The directive operates in three different ways, depending on which of three types the expression evaluates to:

- If the expression evaluates to a string, the string should be one or more space-delimited class names.
- If the expression evaluates to an array, each element of the array should be a string that is one or more space-delimited class names.
- If the expression evaluates to an object, then for each key-value pair of the object with a truthy value the corresponding key is used as a class name.

```
<p ng-class="{strike: deleted, bold: important, red: error}">Map
Syntax Example</p>
<input type="checkbox" ng-model="deleted">deleted (apply "strike"
class)<br>
<input type="checkbox" ng-model="important"> important (apply "bold"
class)<br>
<input type="checkbox" ng-model="error"> error (apply "red"
class)<hr>
<p ng-class="style">Using String Syntax</p>
<input type="text" ng-model="style" placeholder="Type: bold strike
red">
```

The **ng-class-odd** and **ng-class-even** directives work exactly as **ng-class**, except they work in conjunction with **ng-repeat** and take effect only on odd (even) rows.

## Controller

- In Angular, a Controller is a JavaScript constructor function that is used to augment the Angular Scope.
- When a Controller is attached to the DOM via the **ng-controller** directive, Angular will instantiate a new Controller object, using the specified Controller's constructor function.
- A new child scope will be available as an injectable parameter to the Controller's constructor function as $scope.
- Controller is used to Set up the initial state of the $scope object and add behavior to the $scope object.
- Controller should contain only business logic.
- AngularJS controllers are regular JavaScript Objects
- A controller can also have methods (functions as object properties).

```
<div ng-app="" ng-controller="personController">
First Name: <input type="text" ng-model="firstName"><br/>
Last Name: <input type="text" ng-model="lastName"><br/>
Full Name: {{fullName()}}</div>
<script>
function personController($scope) {
    $scope.firstName = "Srikanth";
    $scope.lastName = "Pragada";
    $scope.fullName = function() {
        return $scope.firstName + " " + $scope.lastName;
    }
 }
</script>
```

## Modules

- A module is a container for the different parts of your app – controllers, services, filters, directives, etc.
- Modules make your application more readable, and keep the global namespace clean.
- Global values should be avoided in applications. They can easily be overwritten or destroyed by other scripts. So modules help us to keep our components away from global namespace.
- AngularJS recommend that you break your application to multiple modules like this:

  - ✓ A module for each feature
  - ✓ A module for each reusable component (especially directives and filters)
  - ✓ And an application level module which depends on the above modules and contains any initialization code.

## Scope

- The concept of a scope in Angular is crucial.
- A scope can be seen as the glue which allows the template, model and controller to work together.
- Angular uses scopes, along with the information contained in the template, data model, and controller, to keep models and views separate, but in sync.
- Any changes made to the model are reflected in the view; any changes that occur in the view are reflected in the model.

## $Scope Object

- Typically, when you create an application you need to set up the initial state for the Angular `$scope`.
- You set up the initial state of a scope by attaching properties to the `$scope` object. The properties contain the view model (the model that will be presented by the view).
- All the `$scope` properties will be available to the template at the point in the DOM where the Controller is registered.

- In order to react to events or execute computation in the view we must provide behavior to the scope. We add behavior to the scope by attaching methods to the `$scope` object. These methods are then available to be called from the template/view.

## What are Scopes?

Scope is an object that refers to the application model. It is an execution context for expressions. Scopes are arranged in hierarchical structure which mimic the DOM structure of the application. Scopes can watch expressions and propagate events.

## Scope characteristics

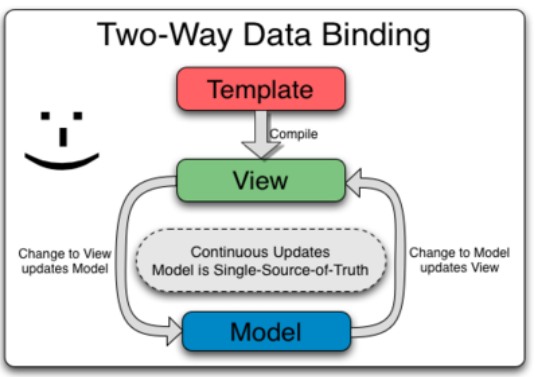
- Scopes provide APIs ($watch) to observe model mutations.
- Scopes provide APIs ($apply) to propagate any model changes through the system into the view from outside of the "Angular realm" (controllers, services, Angular event handlers).
- Scopes provide context against which expressions are evaluated. For example {{username}} expression is meaningless, unless it is evaluated against a specific scope which defines the username property.
- You can think of the scope and its properties as the data which is used to render the view. The scope is the single source-of-truth for all things view related.

## Scope Hierarchies

- Each Angular application has exactly one root scope, but may have several child scopes.
- The application can have multiple scopes, because some directives create new child scopes. When new scopes are created, they are added as children of their parent scope. This creates a tree structure which parallels the DOM where they're attached.
- When Angular evaluates {{name}}, it first looks at the scope associated with the given element for the name property. If no such property is found, it searches the parent scope and so on until the root scope is reached. In JavaScript this behavior is known as prototypical inheritance, and child scopes prototypically inherit from their parents.

# Data Binding

- Data-binding in Angular apps is the automatic synchronization of data between the model and view components.
- The view is a projection of the model at all times. When the model changes, the view reflects the change, and vice versa.
- Because the view is just a projection of the model, the controller is completely separated from the view and unaware of it.




# Templates

- In Angular, templates are written with HTML that contains Angular-specific elements and attributes.
- Angular combines the template with information from the model and controller to render the dynamic view that a user sees in the browser.
- These are the types of Angular elements and attributes you can use:

  - ✓ **Directive** — An attribute or element that augments an existing DOM element or represents a reusable DOM component.
  - ✓ **Markup** — The double curly brace notation `{{ }}` to bind expressions to elements is built-in Angular markup.
  - ✓ **Filter** — Formats data for display.
  - ✓ **Form controls** — Validates user input.

```
<html ng-app>
 <body ng-controller="MyController">
   <input ng-model="foo" value="bar">
   <button ng-click="changeFoo()">{{buttonText}}</button>
   <script src="angular.js">
 </body>
</html>
```

## AngularJS Expressions

- AngularJS expressions are written inside double braces {{ expression }}
- AngularJS expressions bind data to HTML the same way as the ng-bind directive.
- AngularJS will "output" data exactly where the expression is written.
- AngularJS expressions are much like JavaScript expressions: They can contain literals, operators, and variables.

```
{{ 5 + 5 }} or {{ firstName + " " + lastName }}
```

## AngularJS Filters

- Filters can be added to expressions and directives using a pipe character.
- It is possible to chain filters by simply putting more filters after each other in the filter section. When chaining filters, the output of one filter is used as input for the next filter in the chain.

```
{{ expression | filter:argument1:argument2:... }}
```

```
<p>The name is {{ lastName | uppercase }}</p>
```

| Filter | Description |
|---|---|
| currency | Format a number to a currency format. |
| filter | Select a subset of items from an array. |
| lowercase | Format a string to lower case. |
| uppercase | Format a string to upper case. |

| | |
|---|---|
| date | Formats the date according to the given format. |
| number | Formats the value as number |
| json | Converts the value to JSON string |

```
{{ course.Price | currency : 'Rs ' : 2}}
{{ course.StartDate | date :"dd MMM yyyy" }}
{{ course | json }}
```

## Array Filters

AngularJS also contains a set of array filters which filters or transforms arrays.

| Filter | Description |
|---|---|
| limitTo | Limits the array to the given size, beginning from some index in the array. The limitTo filter also works on strings. |
| filter | A general purpose filter. |
| orderBy | Sorts the array based on provided criteria. |

```
<div ng-repeat="n in names | orderBy :  '-' ">
       {{ n }}
</div>
```

```
<tr ng-repeat="c in courses | orderBy:'duration' : 'reverse'">
       <td> {{ c.name }}</td>
       <td> {{ c.duration }}</td>
</tr>
```

```
$scope.longCourses = function (c, idx) {
       return c.duration > 50;
};
```

```
<tr ng-repeat="c in courses | filter: longCourses | orderBy :
'duration' : 'reverse' ">
      <td> {{ c.name }}</td>
      <td> {{ c.duration }}</td>
</tr>
```

## Data Types

- AngularJS numbers are like JavaScript numbers.
- AngularJS strings are like JavaScript strings.
- AngularJS objects are like JavaScript objects.
- AngularJS arrays are like JavaScript arrays

```
<div ng-app="" ng-init="points=[1,15,19,2,40]">
  <p>The points are {{ points[2] }}</p>
</div>
```

## AngularJS Event Listener Directives

- You attach an event listener to an HTML element using one of the AngularJS event listener directives:

  - ✓ ng-click
  - ✓ ng-dbl-click
  - ✓ ng-mousedown
  - ✓ ng-mouseup
  - ✓ ng-mouseenter
  - ✓ ng-mouseleave
  - ✓ ng-mousemove
  - ✓ ng-mouseover
  - ✓ ng-keydown
  - ✓ ng-keyup
  - ✓ ng-keypress
  - ✓ ng-change

# Angular Forms

- A Form is a collection of controls for the purpose of grouping related controls together.
- Form and controls provide validation services, so that the user can be notified of invalid input.
- Attribute **novalidate** is used to disable browser's native form validation.
- The value of ngModel won't be set unless it passes validation for the input field. For example: inputs of type email must have a value in the form of user@domain.
- To allow styling of form as well as controls, ngModel adds these CSS classes:

  - ✓ ng-valid: the model is valid
  - ✓ ng-invalid: the model is invalid
  - ✓ ng-valid-[key]: for each valid key added by $setValidity
  - ✓ ng-invalid-[key]: for each invalid key added by $setValidity
  - ✓ ng-pristine: the control hasn't been interacted with yet
  - ✓ ng-dirty: the control has been interacted with
  - ✓ ng-touched: the control has been blurred
  - ✓ ng-untouched: the control hasn't been blurred
  - ✓ ng-pending: any $asyncValidators are unfulfilled

```
<input type="checkbox" ng-model="myForm.wantNewsletter"
ng-true-value="yes" ng-false-value="no">
<input type="radio" ng-model="myForm.whichNewsletter"
value="weeklyNews">
<input type="radio" ng-model="myForm.whichNewsletter"
value="monthlyNews">
```

## ng-options

Instead of using static HTML options you can have AngularJS create option elements based on data from the $scope object.

```
<select ng-model="myForm.car"
   ng-options="obj.id as obj.name for obj in myForm.options">
      <option value="">Please choose a car</option>
</select>
```

## Form Validation

- AngularJS has a set of form validation directives you can use.
- AngularJS validates form fields before copying their value into the $scope properties to which the form fields are bound.
- If a form field is invalid, its value is NOT copied into the $scope property it is bound to.
- Instead the corresponding $scope property is cleared. That is done to prevent the $scope properties from containing invalid values.
- The ng-minlength and ng-maxlength form validation directives can be used to validate the length of data entered in a form field.
- The ng-pattern directive can be used to validate the value of an input field against a regular expression.
- ng-required The ng-required directive checks if the value of the form field is empty or not. Actually, you just use the required attribute of HTML5, and AngularJS detects it automatically.

### Checking Field Validation State

If you give the <form> element a name attribute, then the form will be added to the $scope object as a property.

```
<form name="myFormNg" ng-submit="myForm.submitTheForm()" >
 ...
</form>
```

When you call a function on the $scope object (a function added to the $scope object by your controller function), you can access the ngFormController object via its name, like this:

```
$scope.myFormNg
```

Both ngFormController and ngModelController objects contain a set of properties that tell if the form or input field is valid. The properties are:

| Property | Description |
|---|---|
| $pristine | True if the form has not been changed (no form fields has changed), false if some fields have been changed. |
| $dirty | The reverse of $pristine - false if the form has not been changed - true if it has. |
| $valid | True if the form field (or the whole form = all form fields) is valid. False if not. |
| $invalid | The reverse of the $valid - false if the field (or all fields in the form) is valid, true if the field (or a single field in the for) is invalid. |

## Textbox Validation Directives

The following are directives related to validating a form input field.

```
<input
  [ng-required=""]
  [ng-minlength=""]
  [ng-maxlength=""]
  [ng-pattern=""]
  [ng-change=""]
  [ng-trim=""]>
...
</input>
```

| Param | Type | Details |
|---|---|---|
| ngModel | string | Assignable angular expression to data-bind to. |
| ngRequired (optional) | boolean | Sets required attribute if set to true. |
| ngMinlength (optional) | number | Sets **minlength** validation error key if the value is shorter than minlength. |
| ngMaxlength (optional) | number | Sets **maxlength** validation error key if the value is longer than maxlength. Setting the attribute to a negative or non-numeric value, allows view values of any length. |

| | | |
|---|---|---|
| ngPattern (optional) | string | Sets **pattern** validation error key if the value does not match the RegExp pattern expression. Expected value is **/regexp/** for inline patterns or regexp for patterns defined as scope expressions. |
| ngChange (optional) | string | Angular expression to be executed when input changes due to user interaction with the input element. |
| ngTrim (optional) | boolean | If set to false Angular will not automatically trim the input. This parameter is ignored for input[type=password] controls, which will never trim the input.(default: true) |

$**error** contains the key that is related to failed validation.

For example, if minlength validation failed for a field then the following for that field is true:

```
if (form.username.$error.minlength)
            // minlength validation for username failed
```

You can user this in HTML as follows:

```
<span class="error" ng-show="form.username.$error.minlength">
      Too short!</span>
```

```
<input type="checkbox"
       [ng-true-value=""]
       [ng-false-value=""]
       [ng-change=""]>

<input type="date"
       [min=""]
       [max=""]
       [ng-required=""]
       [ng-change=""]>
```

# AngularJS $http

- AngularJS $http is a core service for reading data from web servers using AJAX using browser's XMLHttpRequest object or via JSONP.
- The $http API is based on the deferred/promise APIs exposed by the $q service.
- Since the returned value of calling the $http function is a promise, you can also use the then method to register callbacks, and these callbacks will receive a single argument – an object representing the response.
- We can use either $http() or one of its shortcut methods:

  - ✓ $http.get
  - ✓ $http.head
  - ✓ $http.post
  - ✓ $http.put
  - ✓ $http.delete
  - ✓ $http.jsonp
  - ✓ $http.patch

```
$http.get('/someUrl').
  success(function(data, status, headers, config) {
  }).
  error(function(data, status, headers, config) {
  });
```

```
<table ng-controller=”customerController”>
  <tr ng-repeat="x in names">
    <td>{{ x.Name }}</td>
    <td>{{ x.Country }}</td>
  </tr>
</table>
<script>
function customersController($scope,$http){
  $http.get("url")
   .success(function(response) {$scope.names = response;});
}
</script>
```

```
$http(config)
```

For better control on Ajax request, use config object with $http service as follows:

```
var req = {
 method: 'POST',
 url: 'http://www.example.com',
 headers: {
   'Content-Type': undefined
 },
 data: { test: 'test'},
}

$http(req).success(function(){...}).error(function(){...});
```

Object describing the request to be made and how it should be processed. The object has following properties:

- **method** – {string} – HTTP method (e.g. 'GET', 'POST', etc)
- **url** – {string} – Absolute or relative URL of the resource that is being requested.
- **params** – {Object.<string|Object>} – Map of strings or objects which will be turned to?key1=value1&key2=value2 after the url. If the value is not a string, it will be JSONified.
- **data** – {string|Object} – Data to be sent as the request message data.
- **headers** – {Object} – Map of strings or functions which return strings representing HTTP headers to send to the server. If the return value of a function is null, the header will not be sent.
- **cache** – {boolean|Cache} – If true, a default $http cache will be used to cache the GET request, otherwise if a cache instance built with $cacheFactory, this cache will be used for caching.
- **timeout** – {number|Promise} – timeout in milliseconds, or promise that should abort the request when resolved.
- **withCredentials** - {boolean} - whether to set the withCredentials flag on the XHR object.
- **responseType** - {string}

### What it returns?

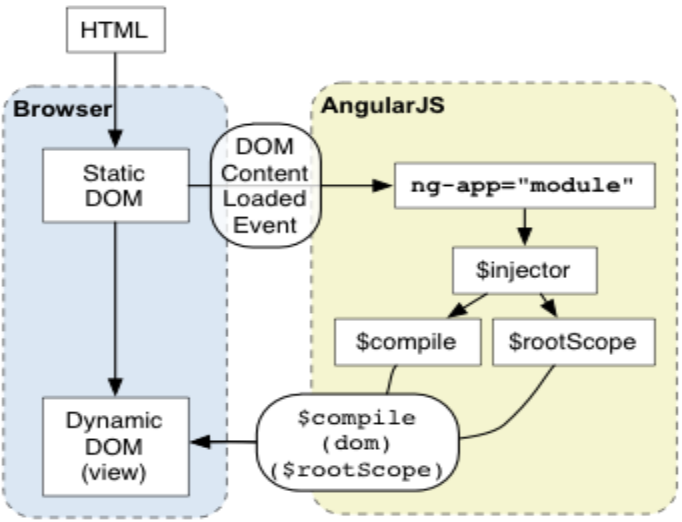
- It returns a promise object with the standard then method and two http specific methods: success and error.
- The **then** method takes two arguments a success and an error callback which will be called with a response object.
- The **success** and **error** methods take a single argument - a function that will be called when the request succeeds or fails respectively.
- The arguments passed into these functions are destructured representation of the response object passed into the then method.
- The response object has these properties:

  - ✓ **data** – {string|Object} – The response body transformed with the transform functions.
  - ✓ **status** – {number} – HTTP status code of the response.
  - ✓ **headers** – {function([headerName])} – Header getter function.
  - ✓ **config** – {Object} – The configuration object that was used to generate the request.
  - ✓ **statusText** – {string} – HTTP status text of the response.

## Bootstrapping AngularJS

- Place the script tag at the bottom of the page. Placing script tags at the end of the page improves app load time because the HTML loading is not blocked by loading of the angular.js script.
- Place **ng-app** to the root of your application, typically on the <html> tag if you want angular to auto-bootstrap your application

# Automatic Initialization




Angular 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 Angular looks for the **ng-app** directive which designates your application root. If the **ng-app** directive is found then Angular will:

- Load the module associated with the directive.
- Create the application injector
- Compile the DOM treating the ng-app directive as the root of the compilation. This allows you to tell it to treat only a portion of the DOM as an Angular application.

As a best practice, consider adding an **ng-strict-di** directive on the same element as **ng-app.**

```
<html ng-app="optionalModuleName" ng-strict-di>
```

# Dependency Injection

- Dependency Injection (DI) is a software design pattern that deals with how components get hold of their dependencies.
- The Angular injector subsystem is in charge of creating components, resolving their dependencies, and providing them to other components as requested.
- DI is pervasive throughout Angular. You can use it when defining components or when providing run and config blocks for a module.
- Components such as services, directives, filters, and animations are defined by an injectable factory method or constructor function. These components can be injected with "service" and "value" components as dependencies.
- Controllers are defined by a constructor function, which can be injected with any of the "service" and "value" components as dependencies, but they can also be provided with special dependencies.
- The run method accepts a function, which can be injected with "service", "value" and "constant" components as dependencies. Note that you cannot inject "providers" into run blocks.
- The config method accepts a function, which can be injected with "provider" and "constant" components as dependencies. Note that you cannot inject "service" or "value" components into configuration.

# Dependency Annotation

Angular 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)
- Using the $inject property annotation
- Implicitly from the function parameter names (has caveats)

## Inline Array Annotation

This is the preferred way to annotate application components. This is how the examples in the documentation are written.

```
someModule.controller
 ('MyController', ['$scope', 'greeter',
      function($scope, greeter) {
      // ...
      }]
);
```

- Here we pass an array whose elements consist of a list of strings (the names of the dependencies) followed by the function itself.
- When using this type of annotation, take care to keep the annotation array in sync with the parameters in the function declaration.

### $inject Property Annotation

To allow the minifiers to rename the function parameters and still be able to inject the right services, the function needs to be annotated with the $inject property. The $inject property is an array of service names to inject.

```
var MyController = function($scope, greeter) {
  // ...
}
MyController.$inject = ['$scope', 'greeter'];
someModule.controller('MyController', MyController);
```

In this scenario the ordering of the values in the $inject array must match the ordering of the parameters in MyController. Just like with the array annotation, you'll need to take care to keep the $inject in sync with the parameters in the function declaration.

### Implicit Annotation

The simplest way to get hold of the dependencies is to assume that the function parameter names are the names of the dependencies.

```
someModule.controller('MyController', function($scope, greeter) {
  // ...
});
```

- Given a function the injector can infer the names of the services to inject by examining the function declaration and extracting the parameter names. In the above example `$scope`, and `greeter` are two services which need to be injected into the function.
- One advantage of this approach is that there's no array of names to keep in sync with the function parameters. You can also freely reorder dependencies.
- However this method will not work with JavaScript minifiers/obfuscators because of how they rename parameters.

## Services

Angular services are substitutable objects that are wired together using dependency injection (DI). You can use services to organize and share code across your app. Angular services are:

- Lazily instantiated – Angular only instantiates a service when an application component depends on it.
- Singletons – Each component dependent on a service gets a reference to the single instance generated by the service factory.
- Like other core Angular identifiers, built-in services always start with $ (e.g. $http).
- To use an Angular service, you add it as a dependency for the component (controller, service, filter or directive) that depends on the service. Angular's dependency injection subsystem takes care of the rest.

## Service components in ng (Built-in Services)

The following are services predefined services provided by Angular.

| Name | Description |
|---|---|
| $anchorScroll | When called, it checks the current value of $location.hash() and scrolls to the related element, according to the rules specified in the Html5 spec. |
| $animate | The $animate service provides rudimentary DOM manipulation functions to insert, remove and move |

| | |
|---|---|
| | elements within the DOM, as well as adding and removing classes. This service is the core service used by the ngAnimate $animator service which provides high-level animation hooks for CSS and JavaScript. |
| $cacheFactory | Factory that constructs Cache objects and gives access to them. |
| $templateCache | The first time a template is used, it is loaded in the template cache for quick retrieval. You can load templates directly into the cache in a script tag, or by consuming the $templateCache service directly. |
| $compile | Compiles an HTML string or DOM into a template and produces a template function, which can then be used to linkscope and the template together. |
| $controller | $controller service is responsible for instantiating controllers. |
| $document | A jQuery or jqLite wrapper for the browser's window.document object. |
| $exceptionHandler | Any uncaught exception in angular expressions is delegated to this service. The default implementation simply delegates to $log.error which logs it into the browser console. |
| $filter | Filters are used for formatting data displayed to the user. |
| $http | The $http service is a core Angular service that facilitates communication with the remote HTTP servers via the browser's XMLHttpRequest object or via JSONP. |
| $httpBackend | HTTP backend used by the service that delegates to XMLHttpRequest object or JSONP and deals with browser incompatibilities. |
| $interpolate | Compiles a string with markup into an interpolation function. This service is used by the HTML $compile service for data binding. See $interpolateProvider for configuring the interpolation markup. |
| $interval | Angular's wrapper for window.setInterval. The fn function is executed every delay milliseconds. |

| | |
|---|---|
| $locale | $locale service provides localization rules for various Angular components. As of right now the only public api is: |
| $location | The $location service parses the URL in the browser address bar (based on the window.location) and makes the URL available to your application. Changes to the URL in the address bar are reflected into $location service and changes to $location are reflected into the browser address bar. |
| $log | Simple service for logging. Default implementation safely writes the message into the browser's console (if present). |
| $parse | Converts Angular expression into a function. |
| $q | A service that helps you run functions asynchronously, and use their return values (or exceptions) when they are done processing. |
| $rootElement | The root element of Angular application. This is either the element where ngApp was declared or the element passed into angular.bootstrap. The element represents the root element of application. It is also the location where the application's $injector service gets published, and can be retrieved using $rootElement.injector(). |
| $rootScope | Every application has a single root scope. All other scopes are descendant scopes of the root scope. Scopes provide separation between the model and the view, via a mechanism for watching the model for changes. They also provide an event emission/broadcast and subscription facility. See the developer guide on scopes. |
| $sceDelegate | $sceDelegate is a service that is used by the $sce service to provide Strict Contextual Escaping (SCE) services to AngularJS. |
| $sce | $sce is a service that provides Strict Contextual Escaping services to AngularJS. |

| | |
|---|---|
| $templateRequest | The $templateRequest service downloads the provided template using $http and, upon success, stores the contents inside of $templateCache. If the HTTP request fails or the response data of the HTTP request is empty, a$compile error will be thrown (the exception can be thwarted by setting the 2nd parameter of the function to true). |
| $timeout | Angular's wrapper for window.setTimeout.<br>The fn function is wrapped into a try/catch block and delegates any exceptions<br>to $exceptionHandler service. |
| $window | A reference to the browser's window object.<br>While window is globally available in JavaScript, it causes testability problems, because it is a global variable. In angular we always refer to it through the $window service, so it may be overridden, removed or mocked for testing. |

## Creating Services

- Application developers are free to define their own services by registering the service's name and service factory function, with an Angular module.
- 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.
- Typically you use the Module factory API to register a service.
- Note that you are not registering a service instance, but rather a factory function that will create this instance when called

**FaceBookService.js**

```
var app = angular.module('fbapp', ['ngResource']);

app.service('FacebookService',["$resource", function ($resource) {
      this.get_page_details = function (page) {
       FaceBook = $resource("http://graph.facebook.com/" + page, {});
       var details = FaceBook.get();
       return details;
      };
    }
  ]
);

app.controller("FacebookController",
  function ($scope, FacebookService)
  {
     $scope.showDetails = false;
     $scope.page = "srikanthtechnologies";
     $scope.getDetails = function () {
     $scope.details = FacebookService.get_page_details($scope.page);
             $scope.showDetails = true;
      };
  }

)
```

**FaceBookClient.html**

```
<!DOCTYPE html>
<html ng-app="fbapp">
<head>
    <title>Facebook Client</title>
    <script src="../angular.js"></script>
    <script src="../angular-resource.js"></script>
    <script src="FaceBookService.js"></script>

</head>
<body ng-controller="FacebookController">
    <h2>Facebook Client</h2>
```

```
    Facebook Page : <input type="text" ng-model="page" />
    <p/>
    <button ng-click="getDetails()">Get Details</button>
    <p/>
    <div ng-show="showDetails">
        <h3>{{ details.name }}</h3>
        <h4>{{ details.phone }}</h4>
    </div>
</body>
</html>
```

## Using Restful services

- The RESTful functionality is provided by Angular in the ngResource module, which is distributed separately from the core Angular framework.
- So download angular-resouce.js
- The ngResource module provides interaction support with RESTful services via the $resource service.

```
angular.module('app', ['ngResource']);
```

### $resource

- A factory which creates a resource object that lets you interact with RESTful server-side data sources.
- The returned resource object has action methods which provide high-level behaviors without the need to interact with the low level $http service.

```
$resource(url, [paramDefaults], [actions], options);
```

| Param | Type | Details |
|---|---|---|
| url | string | A parameterized URL template with parameters prefixed by : as in /user/:username. If you are using a URL with a port number (e.g. http://example.com:8080/api), it will be respected. |
| paramDefaults (optional) | Object | Default values for url parameters. These can be overridden in actions methods. If any of the parameter value is a function, it will be executed every time when a param value needs to be obtained for a request (unless the param was overridden).<br>Given a template /path/:verb and parameter {verb:'greet', salutation:'Hello '} results in URL /path/greet?salutation=Hello.<br><br>If the parameter value is prefixed with @ then the value for that parameter will be extracted from the corresponding property on the data object (provided when calling an action method). For example, if the defaultParam object is {someParam: '@someProp'} then the value of someParam will bedata.someProp. |

| | | |
|---|---|---|
| actions (optional) | Object.<Object>= | Hash with declaration of custom actions that should extend the default set of resource actions. The declaration should be created in the format of $http.config:<br>{action1: {method:?, params:?, isArray:?, headers:?, ...},<br>action2: {method:?, params:?, isArray:?, headers:?, ...},<br>...}<br>Where:<br>action – {string} – The name of action. This name becomes the name of the method on your resource object.<br>method – {string} – Case insensitive HTTP method<br>(e.g. GET, POST, PUT, DELETE, JSONP, etc).<br>params – {Object=} – Optional set of pre-bound parameters for this action. If any of the parameter value is a function, it will be executed every time when a param value needs to be obtained for a request (unless the param was overridden).<br>isArray – {boolean=} – If true then the returned object for this action is an array, seereturns section.<br>. |

It returns resource "class" object with methods for the default set of resource actions optionally extended with custom actions. The default set contains these actions:

```
{ 'get':    {method:'GET'},
  'save':   {method:'POST'},
  'query':  {method:'GET', isArray:true},
  'remove': {method:'DELETE'},
  'delete': {method:'DELETE'} };
```

```
var Github=$resource("https://api.github.com/users/srikanthpragada",
                     {});
$scope.user = Github.get();
```

## Routing

- Routing allows us to break the view into a layout and template views and only show the view we want to show based upon the URL the user is currently accessing.
- AngularJS allows us to do that by declaring routes on the $**routeProvider**, a provider of the $**route** service.
- We need to reference angular-route in our HTML after we reference Angular itself.

```
<script src="angular.js"></script>
<script src="angular-route.js"></script>
```

- We need to specify dependency on **ngRoute** module as follows:

```
angular.module('myApp', ['ngRoute']);
```

- The ng-view directive is a special directive that is included with the ngRoute module. Its specific responsibility is to stand as a placeholder for $route view content.
- The ngView directive follows these specific steps:
    - Any time the $routeChangeSuccess event is fired, the view will update
    - If there is a template associated with the current route:

- Create a new scope
- Remove the last view, which cleans up the last scope
- Link the new scope and the new template
- Link the controller to the scope, if specified in the routes
- Emits the $viewContentLoaded event
- Run the onload attribute function, if provided

- Use $routeProvider to configure which url is mapped to which route.
- To add a specific route, we can use the when method. This method takes two parameters (when(path, route)).

```
angular.module('myApp', []).
    config(['$routeProvider', function($routeProvider)
 {
    $routeProvider
    .when('/',
     { templateUrl: 'views/home.html',
       controller: 'HomeController'
     });
}]);
```

- The first parameter is the route path, which is matched against the $location.path, the path of the current URL. Trailing or double slashes will still work. We can store parameters in the URL by starting off the name with a colon (for instance, :name)
- The second parameter is the configuration object, which determines exactly what to do if the route in the first parameter is matched. The configuration object properties that we can set are controller, template, templateURL, resolve, redirectTo, and reloadOnSearch.

```
angular.module('myApp', []).
config(['$routeProvider', function($routeProvider) {
$routeProvider
.when('/', {
templateUrl: 'views/home.html',
controller: 'HomeController'
```

```
})
.when('/login', {
templateUrl: 'views/login.html',
controller: 'LoginController'
})
.when('/dashboard', {
templateUrl: 'views/dashboard.html',
controller: 'DashboardController'
})
.otherwise({
redirectTo: '/'
});
}]);
```

## $routeParams

- If we start a route param with a colon (:), AngularJS will parse it out and pass it into the $routeParams.
- Angular will populate the $routeParams with the key of :name, and the value of key will be populated with the value of the loaded URL.
- To get access to these variables in the controller, we need to inject the $routeParams in the controller:

```
app.controller('InboxController', function($scope, $routeParams) {
});
```

### Hashbang Mode

- Hashbang mode is a trick that AngularJS uses to provide deep-linking capabilities to your Angular apps. In hashbang mode (the fallback for html5 mode), URL paths take a prepended # character.
- They do not rewrite <a href=""></a> tags and do not require any server-side support. Hashbang mode is the default mode that AngularJS uses if it's not told otherwise. Ex: http://yoursite.com/#!/inbox/all

## HTML5 Mode

- The other routing mode that AngularJS supports is html5Mode. This mode makes your URLs look like regular URLs (except that in older browsers they will look like the hashbang URL). For instance,the same route above in HTML5 mode would look like: http://yoursite.com/inbox/all
- Inside AngularJS, the $location service uses HTML5's history API, allowing for our app to use the regular URL path. The $location service automatically falls back to using hashbang URLs if the browser doesn't support the HTML5 history API.

## HTML Compiler

Compiler is an Angular service which traverses the DOM looking for attributes. The compilation process happens in two phases.

- Compile: traverse the DOM and collect all of the directives. The result is a linking function.
- Link: combine the directives with a scope and produce a live view. Any changes in the scope model are reflected in the view, and any user interactions with the view are reflected in the scope model. This makes the scope model the single source of truth.
- Some directives such as ng-repeat clone DOM elements once for each item in a collection. Having a compile and link phase improves performance since the cloned template only needs to be compiled once, and then linked once for each clone instance.

## Matching Directives

A directive is a behavior which should be triggered when specific HTML constructs are encountered during the compilation process. The directives can be placed in element names, attributes, class names, as well as comments.

Angular normalizes an element's tag and attribute name to determine which elements match which directives. We typically refer to directives by

their case-sensitive camelCase normalized name (e.g. ngModel). However, since HTML is case-insensitive, we refer to directives in the DOM by lower-case forms, typically using dash-delimited attributes on DOM elements (e.g. ng-model).

**The normalization process is as follows :**

- Strip x- and data- from the front of the element/attributes.
- Convert the :, -, or _-delimited name to camelCase.

For example, the following forms are all equivalent and match the ngBind directive:

```
<span ng-bind="name"></span> <br/>
<span ng:bind="name"></span> <br/>
<span ng_bind="name"></span> <br/>
<span data-ng-bind="name"></span> <br/>
<span x-ng-bind="name"></span> <br/>
```

## Creating Directives

- Much like controllers, directives are registered on modules.
- To register a directive, you use the module.directive method, which
- takes the normalized directive name followed by a factory function.
- This factory function should return an object with the different options to tell $compile how the directive should behave when matched.
- The factory function is invoked only once when the compiler matches the directive for the first time.
- You can perform any initialization work here.
- The function is invoked using $injector.invoke which makes it injectable just like a controller.
- In order to avoid collisions with some future standard, it's best to prefix your own directive names. A two or three letter prefix (e.g.stTime) works well.

**app.js**

```
angular.module('app', [])
.controller('Controller', ['$scope', function($scope) {
  $scope.customer = {
    name: 'Srikanth',
    address: 'Srikanth Technologies, Vizag'
  };
}])
.directive('stCustomer', function() {
  return {
    template: 'Name: {{customer.name}} Address: {{customer.address}}'
  };
});
```

**test.html**

```
div ng-controller="Controller">
  <div st-customer></div>
</div>
```