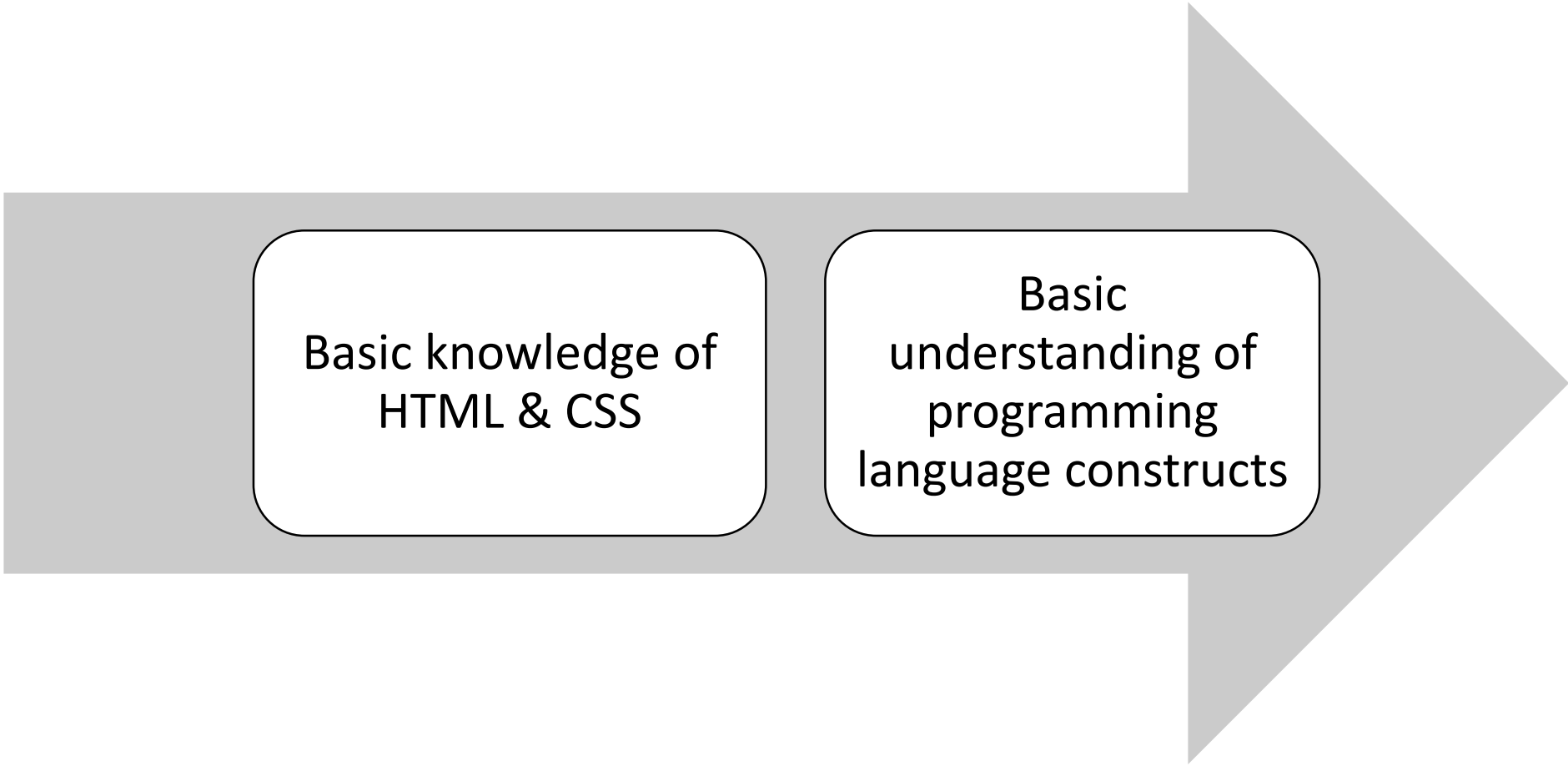
AngularJS

Superheroic framework

Training Agenda

- JavaScript : Overview
- MV* Frameworks
- AngularJS : Getting started
- Building Blocks
  - Controller
  - Directive
  - Filters
  - Service
  - Modules
- Forms
- Routing
- Testing

Prerequisites



JavaScript : Overview

Browser gets the HTML text of the page, parses it into DOM structure, lays out the content of the page and style the context before it gets displayed.

JS has become one of the most popular client-side scripting language on the web used to create dynamic views in web-apps

JS plays a major role in the usage of AJAX, user experience and RWD.

Objects in JavaScript

JS is Object-based language.

Object is unordered collection of properties and methods.

Objects can be created using :

- Literal Method
- Constructor Method
- Instance Method

Functions in JavaScript

JS treats functions as objects (first-class citizens)

In JS, functions can be instantiated, returned by other functions, stored as elements of array of assigned to variables.

A function with no name is called an *anonymous function.*

Closure is a function to which the variable of the surrounding context are bound by reference.

JS function acts as a constructor when we use it together with the *new* operator.

MV* Frameworks

Maintainability	MV* frameworks are designed to make our code easier to maintain and to improve the user experience.
Popular Patterns	MV* framework is nothing but the popular pattern like : • MVC • MVVM • MVP
Separation of Concerns	Idea of all pattern is to separate the Model, View and the Controller.
Some famous framework	AngularJS, BackboneJS, KnockoutJS, EmberJS, ExtJS are some famous framework libraries.

Model, View and Controller

Model

- Contains the data which we are using in our app

View

- Displays the data to the user and read the user input

Controller

- Format the data for views and handle application state

AngularJS : An Overview

Open-source	• AngularJS is an open source JS library , sponsored and maintained by Google.
Super-heroic Framework	• AngularJS termed described as *structural framework for dynamic web apps* by following MV* pattern.
Own Markups	• AngularJS lets you to extend HTML vocabulary for your application.
SPA	• Helps to create SPA apps easily.

AngularJS Features

Extending Markups	Two way binding	Templates	Testability	Super features
• Extending HTML to add dynamic nature	• Synchronize the data between model and view, view component gets updated when the model get change and vice-versa, no need for events to accomplish this.	• Template can be created using HTML itself.	• Testability is the primary consideration	• Supports routing, filtering, ajax calls, data-binding, caching, history and DOM manipulation

AngularJS Controller & Scope

- Controller primary responsibility is to create scope object ($scope).
- Scope communicate with the view in two way communication.
- Scope expose model to view. Model is nothing but the data present in the Scope.
- View can be bind to the functions on the scope.
- We can modify the model using methods available on scope.

AngularJS Model

The model is simply a plain old JS Object, does not use getter/setter methods or have any special framework needs.

Changes are immediately reflected in the view via the two-way binding feature.

All model stem from Scope object.

Typically model objects are initialized in controller code.

In the HTML template, model variable would be referenced in curly braces.

AngularJS View and Templates

The View in AngularJS is the compiled DOM

View is the product of $compile merging the HTML template with the $scope.

In AngularJS, templates are written with HTML that contains Angular specific elements and attributes.

AngularJS combines the template with information from the model and controller to render the dynamic view that a user sees in browser.

AngularJS Modules

Group Interrelated Code	• A module is the overall container used to group AngularJS code. It consists of compiled services, directives, views controllers etc.
App Wiring	• Module is like a main method that instantiates and wires together the different part of app.
Declarative	• Module declaratively specify how an app should be bootstrapped.
Module Creation	• Angular module API allows to declare module using *angular.module()* API method.
Module Definition	• When declaring the module, we need to pass two parameters to the method. First : Module_Name; Second : Injectables

AngularJS Expressions

- Expressions `{{ expression }}` are JavaScript like code snippets.
- Expressions are evaluated against a scope object.
- AngularJS let us to execute expressions directly within our HTML pages.
- Expressions can also hold computational code but can't directly use JavaScript syntax like conditionals, loops or exceptions inside it.

e.g.

```
{{ 3*3 }}
{{ “Foo” + “ ” + “Bar” }}
{{ [“Foo”, “Bar”, “Bam”, “Baz”] }}
```

$rootScope

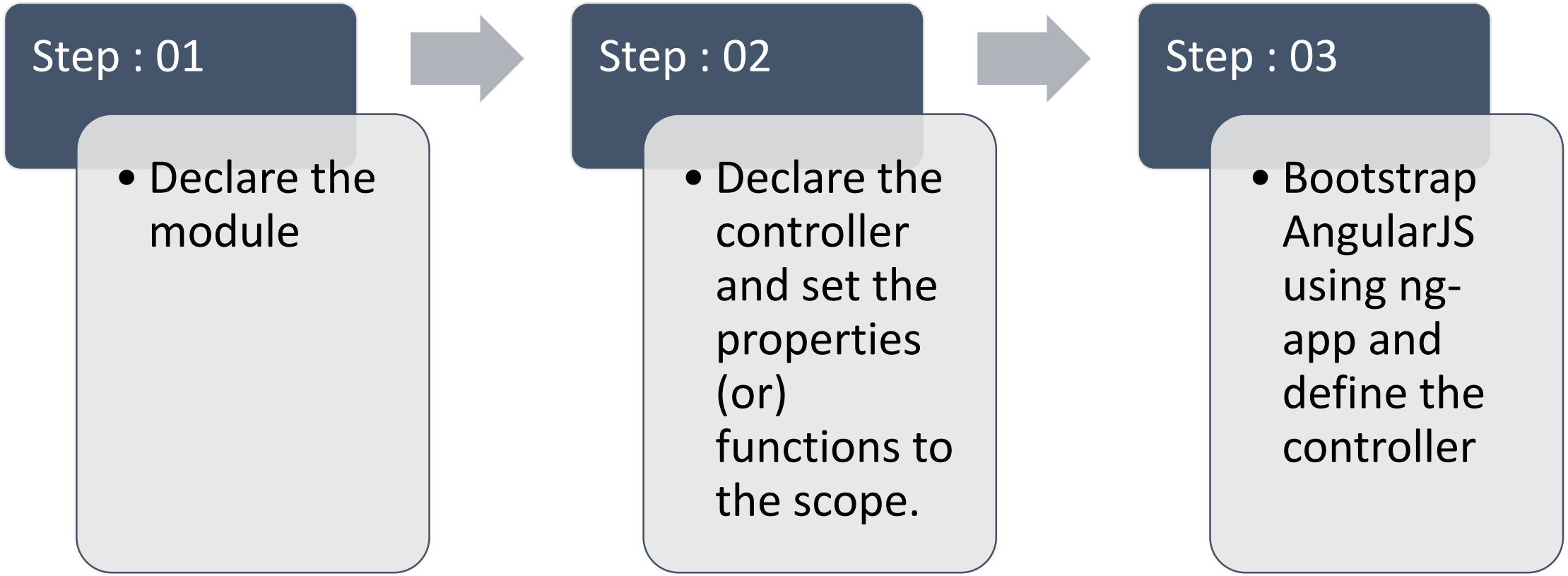
When AngularJS starts to run and generate the view, it will create a binding from the root ng-app element to the $rootScope.

$rootScope is the eventual parent of all $scope objects and it is set when the module initialize via run method.

The $rootScope object is the closest object we have in global context in Angular app.

Attaching too much logic here is a bad idea.

Steps for coding Hello World in AngularJS

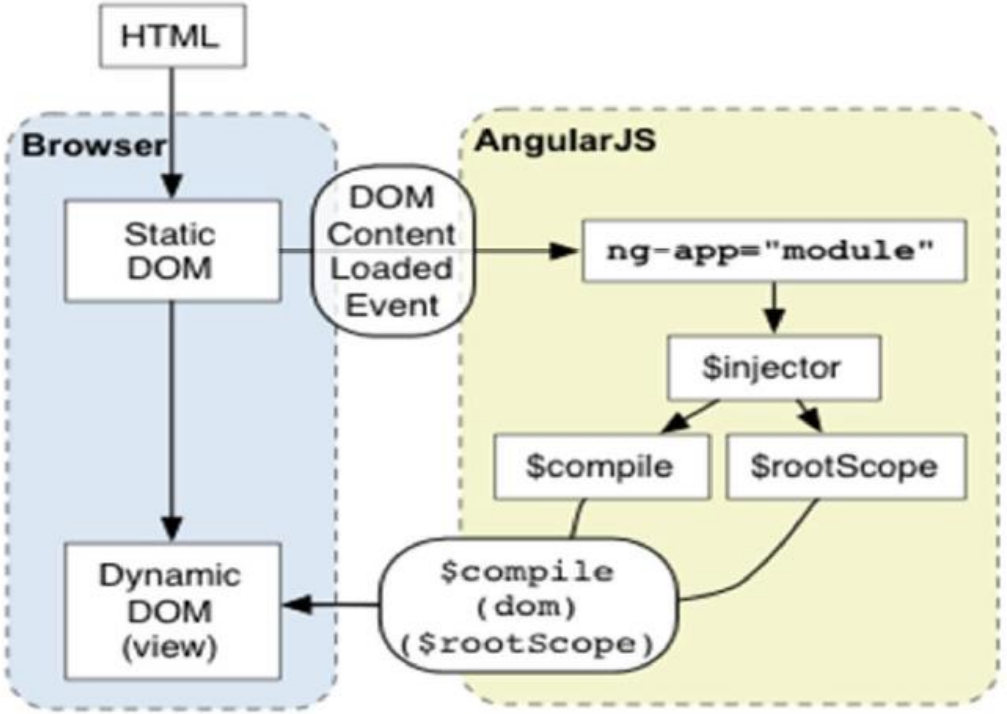


Config and Run Method

angular.module type has config () and run () method	
config (configFn)	We can use this method to register the work which needs to be performed on the module loading
	It will be very useful for configuring the routes and services.
run (initializationFn)	We can use this method to register the work which needs to be performed when the injector is done loading all modules.

How AngularJS Works

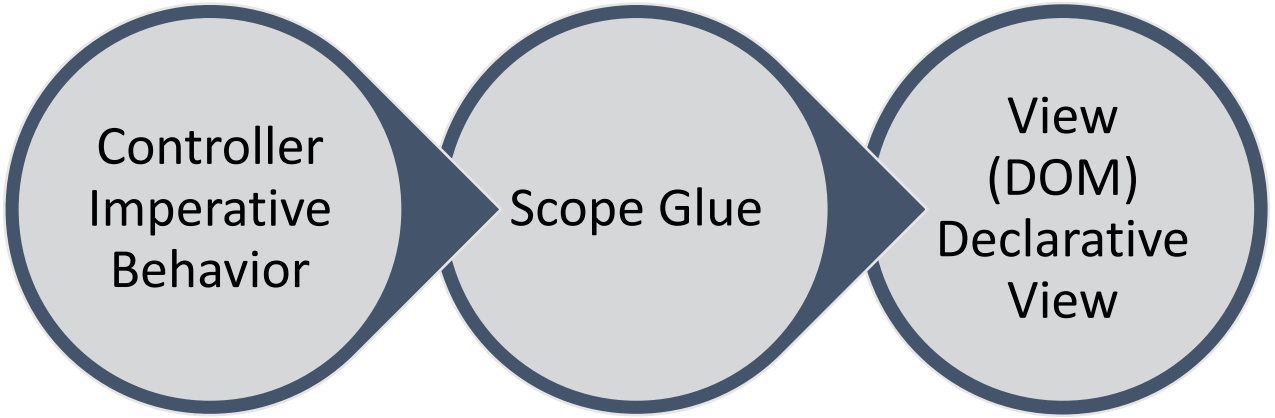
- $compile compiles DOM into a template function that can be used to link scope and the view together



Source : Angularjs.org

Controllers

- Controller is used to provide the business logic behind the view and construct and value the model
- The goal is to not manipulate the DOM at all in the controller, which we have done using other frameworks.



Controller – Best Practices

- Controllers should not know anything about the view they control.
- Controllers should be small and focused.
- Controllers should not talk to other controller.
- Controllers should not own the domain model.

AngularJS Directives

Directives are ways to transform the DOM through extending HTML and provides new functionality to it.

As a good practice DOM manipulation need to be done in directive.

Directive is simply a function that we run on a particular DOM element that tells Angular's HTML compiler to attach a specified behaviour or even transform the DOM element and its children.

Directives are actual defined with camelCase in the JavaScript, but applied with a dash to the HTML.

Angular comes with a set of built-in directives along with that we can create our own directives.

Directives

- Angular directive can be specified in 03 ways :
  - As a tag –
    - `<ng-form />`
  - As an attribute –
    - `<div ng-form> </div>`
  - As a class –
    - `<div class=“ng-form”> </div>`
- All the built-in directives in angular cant be specified with all the 03 ways. Some of them can be specified in 1 or 2 ways only.

Built-in Directives

AngularJS provides a suite of built-in directives.

ngApp

- Placing ngApp on any DOM element marks that element as the beginning of the $rootScope
- $rootScope is the beginning of the scope chain, and all directives nested under the ng-app in your HTML inherit from it.
- $rootScope can be accessed via the run method
- Using $rootScope is like using global scope hence it is not a best practice
- We can use ng-app once per document

ngController

- This directive is used to place a controller on a DOM element
- Instead of defining actions and models on $rootScope, use ng-controller

Built-in Directives

ngBind

- This attribute tells Angular to replace the text content of the specified HTML element with the value of a given expression and to update the text content when the value of that expression changes.
- It is preferable to use ngBind instead of {{ expression }}.

ngShow

- Shows or hide the given HTML element based in the expression provided to the ngShow attribute.

ngHide

- Shows or hide the given HTML element based in the expression provided to the ngHide attribute.

ngCloak

- Used to prevent the Angular HTML template from being briefly displayed by the browser in its raw form while your application is loading.

Built-in Directives

ngSrc

- AngularJS will tell the browser not to fetch the image via the given URL until all the expressions provided to ng-src have been interpolated

ngHref

- Angular waits for the interpolation to take place and then activates the link's behaviour

ngDisabled, ngChecked, ngReadonly & ngSelected

- Directives work with HTML Boolean attributes

ngIf

- Used to completely removed or re-create an element in the DOM based on an expression.
- Using ng-if when an element is removed from the DOM, its associated scope is destroyed. When it comes back into being, a new scope is created.

Built-in Directives

ngModel

- ngModel directive provides two way data-binding by synchronizing the model to the view, as well as view to the model.
- ngModel will try to bind the property given by evaluating the expression on the current scope. If the property does not already exist on this scope, it will be created implicitly and added to the scope.
- ngModel is responsible for :
  - Binding the view into the model.
  - Providing validation behaviour.
  - Keeping the state of the control.
  - Setting related CSS classes on the element
  - Registering the control with it’s parent form

Built-in Directives

ngInit

- Used to setup the state inside the scope of a directive when that directive is invoked

ngInclude

- Used to fetch compile and include an external HTML fragment into your current application.
- By default, the template URL is restricted to the same domain and protocol as the application document, unless white-listed or wrapped as a trusted-values.

ngRepeat

- Instantiates a template once per item from a collection. Each template instance gets its own scope, where the given loop variable is set to the current collection item, and $index is set to the item index or key.

Built-in Directives

ngStyle

- Allows you to set CSS style on an HTML element
- CSS class name and values must be quoted

ngClass

- Allows you to dynamically set CSS classes on an HTML element by databinding an expression that represent all classes to be added.

ngClassEven

- Works in conjuction with ngRepeat and take effect only on even row elements i.e 0,2,4,6 …

ngClassOdd

- Works in conjuction with ngRepeat and take effect only on odd row elements i.e 1,3,5,7 …

Built-in Event Directives

- When AngularJS parses the HTML, it look for directive and take action based on that, when it looks for event directives it register the event on the DOM object.

ngClick	ngDlbclick	ngMouseup
ngMousedown	ngMouseleave	ngMouseenter
ngMousemove	ngMouseover	ngChange

Custom Directive

Custom directive is defined using the .directive() method on app Angular module.

Directives can be implemented on the following ways:

Element
Class
Attribute
Comment

AngularJS recommends to use element and attribute directive and leave the CSS class and comment directive (unless absolutely necessary).

Custom Directive

Directive consist of three (or less) important things :

Link function is where DOM manipulation occurs

Controller is constructed during the pre-linking phase and receives the $scope for the element.

DDO tells the compiler how the directive to be assembled. Common properties include the link function, controller function, restrict, template and templateUrl etc.

Applying restrictions to directives

Custom directive is restricted to attribute by default.

In order to create directive that are triggered by element, class name & comment, we need to use the *restrict* option.

The restrict option is typically set to :

- A
- E
- C
- M

These restrictions can also be combined

'AEC' – matches either attribute or element or class name

Custom Directives - template

Template is an inline template specified using HTML as a string /function gets appended / replaced within the element where the directive was invoked.

Template has a scope that can be accessed using double curly markup, like {{ expression }}. Backslashes is used at the end of the each line to denote multi-line string.

When a template string must be wrapped in a parent element i.e a root DOM element must exist.

Custom Directives - templateUrl

templateUrl comes in handy when template becomes too big to inline.

We can specify the path of an HTML file / a function which return the path of an HTML file.

By default, the HTML file will be requested on demand via Ajax when the directive is invoked.

Custom Directives – compile() & link()

The compile() and link() function define how the directive is to modify the HTML that matched the directive.

When the directive is first compiled by AngularJS, the compile() function is called. The compile() function can do any one time configuration of the element needed.

The compile() function finishes by returning the link() function. The link() function is called everytime the element is to be bound to data in the $scope object.

We can even set only a link() function for the custom directive.

Custom Directives – compile() & link()

The template produced by a directive is meaningless unless it's compiled against the right scope.

By default, a directive does not get a new child scope. Rather, it gets the parent's scope. This means that if the controller is present inside a controller it will use that controller's scope. To utilize the scope, we can make use of a function called *link*.

The link function is mainly used for attaching event listeners to DOM elements, watching model properties for changes, and updating the DOM.

Link takes a function with the following signatures :

Scope
Element
Attributes

Custom Directive – Directive's Scope

By default, a directive get the parent's scope, so that they are free to modify the parent's controller scope properties.

If directive need to add properties and functions for internal use there is no need to add it to the parent's scope.

The scope can be configured with the scope property of the directive definition object.

A child scope – this scope prototypically inherits the parent's scope.

An isolated scope – a new scope does not inherit from the parent and exists on its own. These bindings are specified by the attributes defined in HTML and the definition of the scope property in the directive definition object.

Custom directives – Isolated Scope using '@ = &'

There are three types of binding options which are defined as prefixes in the scope property. The prefix is followed by the attribute name of HTML element	One way text binding (prefix : @)
	Two-way binding (prefix : =)
	Execute function in the parent scope (prefix : &)

Text binding are prefixed with @, and they are always strings. Whatever we write as attribute value, it will be parsed and return as strings. If the parent scope changes, the isolated scope will reflect those changes, but not the other way around.

Two way bindings are prefixed by = and can be of any type. Whenever the parent scope property changes, the corresponding isolated scope property also changes and vice-versa.

Custom Directive - Transclusion

Transclusion is a feature which lets us to wrap a Directive around the arbitrary content. We can extract later and compile it against the current scope, and finally place it at the specified position in the directive template.

Transclude allows to pass in an entire template, including its scope, to a directive. Doing so gives the opportunity to pass in arbitrary content and arbitrary scope to a directive. If the scope option is not set, then the scope available inside the directive will be applied to the template passed in.

Transclusion is most often used for creating reusable widgets.

ngTransclude directive mark the insertion point for the transcluded DOM of the nearest parent directive that uses transclusion.

- Any existing content of the element that this directive is placed on will be removed before the transcluded content is inserted.

Digest Cycle & $scope

Digest cycle can be considered as a loop, during which AngularJS checks if there are any changes occurred to the variable being watched.

AngularJS sets up a watcher on the scope model, which in turns update the view whenever the model changes.

$digest cycle fires the watchers.

When the $digest cycle starts, it fires each of the watchers. These watchers checks the current value of the scope model is different from the old value. If there is a change, then the corresponding listener function executes. As a result any expressions in the view gets updated.

Digest Cycle & $scope

$digest cycle starts as a result os a call to $scope.$digest().

AngularJS does not directly call $digest(). Instead it calls $scope.$apply(), which in turn calls $rootScope.$digest(). As a result of this, a digest cycle starts at the $rootScope, and subsequently visits all the child scopes calling the watchers along the way.

AngularJS wraps the function calls (which updates the model) from view within $scope.$apply()

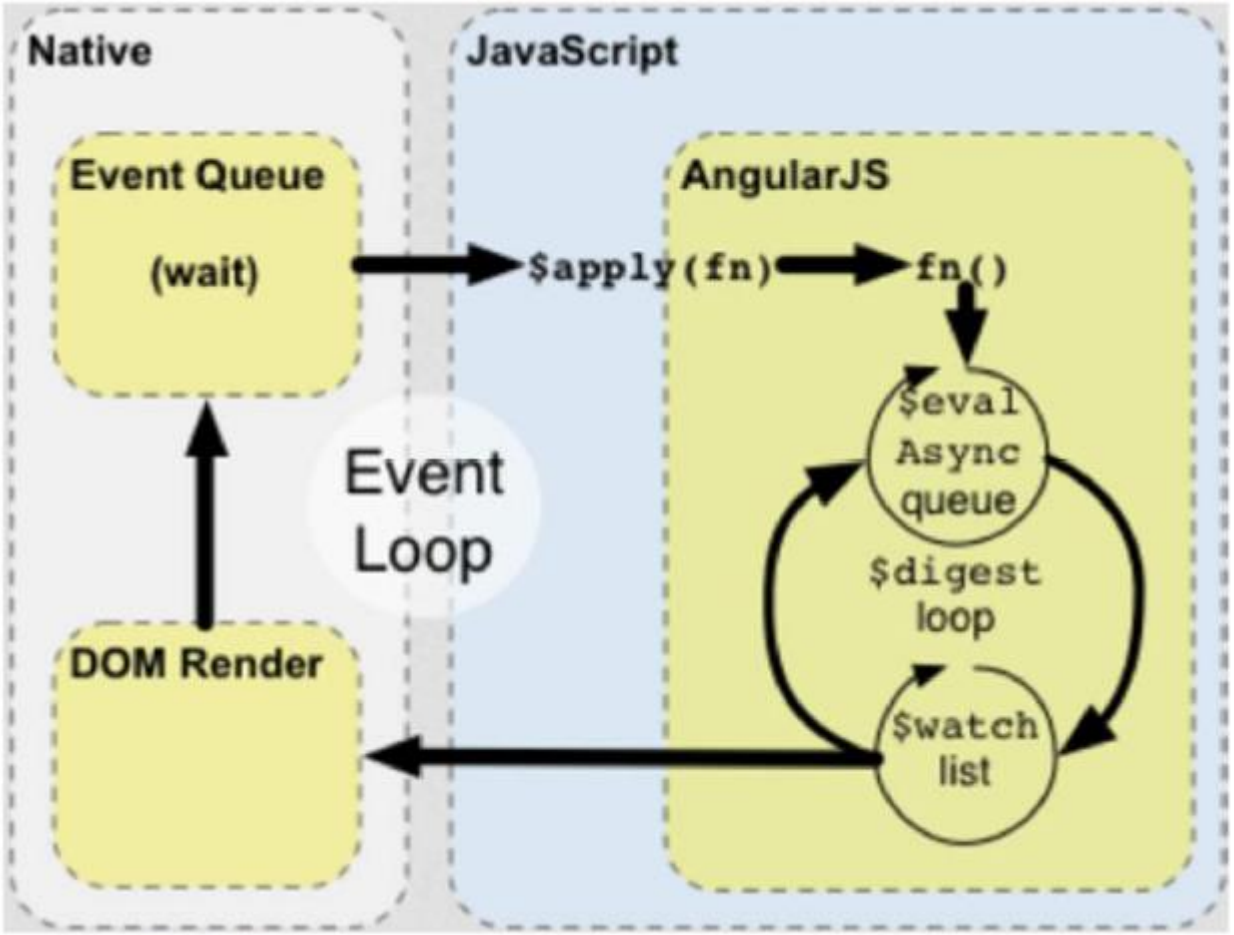
Digest Cycle and $scope

Built-in directives/services (like ng-repeat, ng-model, $timeout, $http etc) which changes models automatically trigger a $digest cycle i.e. it calls $apply() automatically and creates implicit watches to the model variables.

If we change any model outside of the angular context, then we need to inform angular of the changes made by calling $apply() manually. For instance, if setTimeout() function updates a scope model, Angular wont have any idea about the model change then it becomes our responsibility to call $apply() manually, which in turn triggers a digest cycle.

If a directive that sets up a DOM event listener and changes models inside handler function, we need to call $apply() to ensure the changes take effect.

Digest Cycle & $scope



Source : AngularJS.org

$watch

$watches can be used to watch any value, and trigger a function call when that value changes. A $watch can be setup from any $scope by calling $scope.$watch().

There are two ways to setup a watch (no difference between two) –

- By expression
- By function

$digest

$digest loops through all watchers on the scope on which it is called and its child scopes. It evaluates them and executing the handlers if any changes found.

To call $digest : `$scope.$digest();`

$apply

It’s a wrapper around $rootScope.$digest that evaluates any expression passed to it prior to calling $digest().

Use $digest/$apply in directives to let Angular know you've made changes after an asynchronous call, such as a DOM event.

Use $digest/$apply in services to let Angular know some asynchronous operation has returned, such as a service update, or an event from a 3rd party library.

DON'T use $digest/$apply in a controller.

Filters

A filter formats the value of an expression for display to the user.

Filters can be invoked in HTML with the | (pipe) character inside the template.

We can also use filters from within JavaScript by using $filter Service.

To pass an argument to a filter in the HTML form, we pass it with a colon after the filter name (for multiple arguments, we can simply append a colon after each argument)

AngularJS gives us several built-in filters as well as an easy way to create our own.

Built-in Filters

- Uppercase
- Lowercase
- Number
- Currency
- Json
- Date
- Filter
- limitTo
- orderBy

Service

Service is just a simple JavaScript object that does some sort of work.

It is typically stateless and encapsulates some sort of functionality.

As a best practice, we need to place business logic inside service instead of placing it in the controller.

It help us to adhere to the Single Responsibility Principle (SRP) and Dependency Injection Principle (DIP) as well make the service reusable.

Services provide a method for us to keep data around for the lifetime of the app and communicate across controllers in a consistent manner.

Services are singleton objects that are instantiated only once per app and lazy loaded.

Creating and Registering a Service

AngularJS comes with several built-in services along with that we can create our own services.

Angular compiler can reference service and load it as a dependency for runtime once it is registered.

We can create service using five different ways –

- Factory()
- Service()
- Provider()
- Constant()
- Value()

Registering Service using factory() function

The most common method for registering a service with our app is through the factory() method.

The factory() function takes two arguments :

- Name of the service which we want to register.
- Function which runs when Angular creates the service. It will be invoked once for the duration of the app lifecycle, as the service is a singleton object. It can return anything from a primitive value to a function to an object.

Registering Service using service() function

The service() function is used to register an instance of a service using a constructor function.

The service() function will instantiate the instance using the new keyword when creating the instance.

The service() function takes two arguments -

- Name of the service which we want to register
- The constructor function that we'll call to instantiate the instance

Registering Service using provider() function

A provider is an object with a $get() method. The $injector calls the $get() method to create a new instance of the service. The provider() method is responsible for registering the service in the $providerCache.

factory() function is shorthand for creating a service through the provider() method wherein we assume that the $get() function is the function passed.

$provide service is responsible for instantiating these providers at runtime.

We can also register service via the $provide service inside of a module's config function.

The provide() function takes two arguments.

- Name
- Object / Function / Array

Registering Service using constant() function

Constant() function is used to register an existing value as a service so that we can inject it into a module configuration function.

The constant() function returns a registered service instance.

The constant() function takes two arguments –

- The name with which to register the constant.
- The value to register as a constant

Registering Service using value() function

Value function is used to register the service, when the return value is just a constant.

The value () function returns a registered service instance.

We can’t inject a value into a config function.

The value() function takes two arguments –

- The name with which to register the value
- The injectable instance

Use value() to regsiter a service object or function and use constant() for configuration data.

Built-in Services

- AngularJS services are substitutable object that we wired together using Dependency Injection (DI). We can use those services to organize and share code across the application.
- AngularJS ships with lot of built-in services. Important services below:

$http	$q	$compile	$filter
$log	$window	$timeout	$locale

AngularJS Promise

A promise represents the eventual result of asynchronous operation.

A promise is an object with a then() method, accepts two functions as parameters :

- Function to be executed (onSuccess) when the promise is fulfilled
- Function to be executed (onFailure) when the promise is rejected

Both functions onSuccess and onFailure takes one parameter i.e. response outcome of an asynchronous service. For each promise only one of the functions can be called.

$q service in AngularJS provides deferred and promise implementations.

$http Service

Core AngularJS service that facilitates communication with the remote HTTP servers via the browser's XMLHttpRequest object or via JSONP.

$http service is a function which takes a single argument i.e. a configuration object which is used to generate an HTTP request and returns a promise with two $http specific methods : success and error

$compile Service

- $compile service compiles an HTML string or DOM into a template and produces a template function, which can then be used to link scope and template together.

```
<div ng-controller="ServiceController">
    <div id="target"></div>
    Markup : <input type="text" ng-model="markup"><br>
    <input type="button" ng-click="appendToDivElement(markup)" value="Append">
</div>

<script type="text/javascript">
    var app = angular.module("serviceApp", []);
    app.controller("ServiceController", function($scope, $compile){
        $scope.appendToDivElement = function(markup){
            return angular.element("#target").append($compile(markup)($scope));
        }
    });
</script>
```

$locale Service

- Provides localization rules for AngularJS components
- Locale ID formatted as languageId-countryId (e.g. en-us)
- Local script files are available under I18n folder, it needs to be refered in HTML.
  - Angular-locale_hi-in.js (hindi)
  - Angular.locale_tam-in.js (tamil)

```
<div ng-controller="LocaleController">
    <h2>{{ today | date : dateFormat }}</h2>
</div>
<script type="text/javascript">
    app.controller("LocaleController", function($scope, $locale){
        $scope.todayDate = new Date();
        $scope.dateFormat = $locale.DATETIME_FORMATS.fullDate;
    });
</script>
```

$timeout Service

- $timeout is a wrapper for window.setTimeout() function.
- The return value of registering a timeout() function is a promise, which will be resolved when the timeout is reached and the timeout function is executed.
- To cancel a timeout request, call $timeout.cancel(promise).

```
<div ng-controller="TimeoutController">
    <h2>Display Company Name in 5 sec(s) : {{ companyName }}</h2>
</div>
<script type="text/javascript">
    app.controller("TimeoutController", function($scope, $timeout, $interval){
        var timeoutPromise = $timeout(function(){
            $scope.companyName = "XYZ";
        }, 5000);
    });
</script>
```

Routing

It is important navigate from one page view to another in single page application.

We can break-out the view into a layout and template views and only show the view which we want to show based upon the URL the user is accessing.

Routing means loading sub-template depending upon the URL of page.

Routes are a way for multiple view to be used within a single HTML page. This enables your page to look more “app-like” because users are not seeing page reloads happen within the browser.

AngularJS Routes

AngularJS routes enable us to create different URLs for different content in our application. Having different URLs for different contents enables the user to bookmark URLs to specific content.

In AngularJS, each such bookmarkable URL is called a route.

A route is specified in the URL after the # sign

http://www.example.com/index.html#/training

Setting up page for routing

To setup the page for routing we need to follow the 4 steps given below-

- AngularJS requires the route service, which is not part of the default AngularJS library. We need to load angular-route.js as part of your script loading.
- We need to inject the route service in our app module.
- Use ngView directive in the HTML tag to display the given route.
- Configure $routeProvider in the module's config() function via calls to the when() and otherwise() functions.

Route Parameters

- AngularJS will parse a route param with a colon (:) and pass it to $routeParams.
- In below scenario, Angular will populate the $routeParams with the key of :id, and the value of key will be populated with the value of the loaded URL.

```
app.config(function($routeProvider){
    $routeProvider
        .when("/employees/:id", {
            redirectTo : function(routeParams, path, search){
                console.log(routeParams);
                console.log(path);
                console.log(search);
                return "/";
        });
});
```

References

Web

- https://angularjs.org/
- https://www.javatpoint.com/angularjs-tutorial
- http://tutorials.jenkov.com/angularjs/index.html
- https://www.youtube.com/watch?v=zKkUN-mJtPQ&list=PL6n9fhu94yhWKHkcL7RJmmXyxkuFB3KSl

Books

- Ng-book by Ari Lerner