List at least three ways to communicate between modules of your application using core AngularJS functionality.

Common ways to communicate between modules of your application using core AngularJS functionality include:

* Using services
* Using events
* By assigning models on $rootScope
* Directly between controllers, using $parent, $$childHead, $$nextSibling, etc.
* Directly between controllers, using ControllerAs, or other forms of inheritance

[Comment](https://www.toptal.com/angular-js/interview-questions)

Which means of communication between modules of your application are easily testable?

Using a service is definitely easy to test. Services are injected, and in a test either a real service can be used or it can be mocked.

Events can be tested. In unit testing controllers, they usually are instantiated. For testing events on $rootScope, it must be injected into the test.

Testing $rootScope against the existence of some arbitrary models is testable, but sharing data through $rootScope is not considered a good practice.

For testing direct communication between controllers, the expected results should probably be mocked. Otherwise, controllers would need to be manually instantiated to have the right context.

[Comment](https://www.toptal.com/angular-js/interview-questions)

The most popular e2e testing tool for AngularJS is Protractor. There are also others which rely on similar mechanisms. Describe how e2e testing of AngularJS applications work.

The e2e tests are executed against a running app, that is a fully initialized system. They most often spawn a browser instance and involve the actual input of commands through the user interface. The written code is evaluated by an automation program, such as a Selenium server (webdriver). That program sends commands to a browser instance, then evaluates the visible results and reports back to the user.

The assertions are handled by another library, for Protractor the default is Jasmine. Before Protractor, there was a module called Angular Scenarios, which usually was executed through Karma, and is now deprecated. Should you want to e2e test hybrid apps, you could use another Selenium server, called Appium.

Testing can be handled manually, or it can be delegated to continuous integration servers, either custom or ones provided by Travis, SauceLabs, and Codeship.

[Comment](https://www.toptal.com/angular-js/interview-questions)

This is a simple test written for Protractor, a slightly modified example from Protractor docs:

it('should find an element by text input model', function() {

browser.get('/some-url');

var login = element(by.model('username'));

login.clear();

login.sendKeys('Jane Doe');

var name = element(by.binding('username'));

expect(name.getText()).toEqual('Jane Doe');

// Point A

});

Explain if the code is synchronous or asynchronous and how it works.

The code is asynchronous, although it is written in a synchronous manner. What happens under the hood is that all those functions return promises on the control flow. There is even direct access, using “protractor.promise.controlFlow()”, and the two methods of the returned object, “.execute()” and “.await()”.

Other webdriver libraries, such as wd https://github.com/admc/wd, require the direct use of callbacks or promise chains.

[Comment](https://www.toptal.com/angular-js/interview-questions)

When a scope is terminated, two similar “destroy” events are fired. What are they used for, and why are there two?

The first one is an AngularJS event, “$destroy”, and the second one is a jqLite / jQuery event “$destroy”. The first one can be used by AngularJS scopes where they are accessible, such as in controllers or link functions.

Consider the two below happening in a directive’s postLink function. The AngularJS event:

scope.$on(‘$destroy’, function () {

// handle the destroy, i.e. clean up.

});

And

element.on(‘$destroy’, function () {

// respectful jQuery plugins already have this handler.

// angular.element(document.body).off(‘someCustomEvent’);

});

The jqLite / jQuery event is called whenever a node is removed, which may just happen without scope teardown.

[Comment](https://www.toptal.com/angular-js/interview-questions)

How do you reset a “$timeout”, and disable a “$watch()”?

The key to both is assigning the result of the function to a variable.

To cleanup the timeout, just “.cancel()” it:

var customTimeout = $timeout(function () {

// arbitrary code

}, 55);

$timeout.cancel(customTimeout);

The same applies to “$interval()”.

To disable a watch, just call it.

// .$watch() returns a deregistration function that we store to a variable

var deregisterWatchFn = $rootScope.$watch(‘someGloballyAvailableProperty’, function (newVal) {

if (newVal) {

// we invoke that deregistration function, to disable the watch

deregisterWatchFn();

...

}

});

[Comment](https://www.toptal.com/angular-js/interview-questions)

Name and describe the phases of a directive definition function execution, or describe how directives are instantiated.

The flow is as follows:

First, the “$compile()” function is executed which returns two link functions, preLink and postLink. That function is executed for every directive, starting from parent, then child, then grandchild.

Secondly, two functions are executed for every directive: the controller and the prelink function. The order of execution again starts with the parent element, then child, then grandchild, etc.

The last function postLink is executed in the inverse order. That is, it is first executed for grandchild, then child, then parent.

A great explanation of how directives are handled in AngularJS is available in the [AngularJS Tutorial: Demystifying Custom Directives](https://www.toptal.com/angular-js/angular-js-demystifying-directives) post on the Toptal blog.

[Comment](https://www.toptal.com/angular-js/interview-questions)

How does interpolation, e.g. “{{ someModel }}”, actually work?

It relies on $interpolation, a service which is called by the compiler. It evaluates text and markup which may contain AngularJS expressions. For every interpolated expression, a “watch()” is set. $interpolation returns a function, which has a single argument, “context”. By calling that function and providing a scope as context, the expressions are “$parse()”d against that scope.

[Comment](https://www.toptal.com/angular-js/interview-questions)

How does the digest phase work?

In a nutshell, on every digest cycle all scope models are compared against their previous values. That is *dirty checking*. If change is detected, the watches set on that model are fired. Then another digest cycle executes, and so on until all models are stable.

It is probably important to mention that there is no “.$digest()” polling. That means that every time it is being called deliberately. As long as core directives are used, we don’t need to worry, but when external code changes models the digest cycle needs to be called manually. Usually to do that, “.$apply()” or similar is used, and not “.$digest()” directly.

[Comment](https://www.toptal.com/angular-js/interview-questions)

List a few ways to improve performance in an AngularJS app.

The two officially recommended methods for production are disabling debug data and enabling strict DI mode.

The first one can be enabled through the $compileProvider:

myApp.config(function ($compileProvider) {

$compileProvider.debugInfoEnabled(false);

});

That tweak disables appending scope to elements, making scopes inaccessible from the console. The second one can be set as a directive:

<html ng-app=“myApp” ng-strict-di>

The performance gain lies in the fact that the injected modules are annotated explicitly, hence they don’t need to be discovered dynamically.

You don’t need to annotate yourself, just use some automated build tool and library for that.

Two other popular ways are:

* Using one-time binding where possible. Those bindings are set, e.g. in “{{ ::someModel }}” interpolations by prefixing the model with two colons. In such a case, no watch is set and the model is ignored during digest.
* Making $httpProvider use applyAsync:

myApp.config(function ($httpProvider) {

$httpProvider.useApplyAsync(true);

});

… which executes nearby digest calls just once, using a zero timeout.

[Comment](https://www.toptal.com/angular-js/interview-questions)

What is $rootScope and how does it relate to $scope?

$rootScope is the parent object of all $scope Angular objects created in a web page.

## AngularJS Interview Question #1

**What are the basic steps to unit test an AngularJS filter?**

(Question provided by[*Daniel Lamb*](https://www.codementor.io/daniellmb))

1. Inject the module that contains the filter.  
2. Provide any mocks that the filter relies on.  
3. Get an instance of the filter using $filter('yourFilterName').  
4. Assert your expectations.

Dependency injection is a powerful software design pattern that Angular employs to compose responsibilities through an intrinsic interface. However, for those new to the process, it can be puzzling where you need to configure and mock these dependencies when creating your isolated unit tests. The open-source project “Angular Test Patterns” is a free resource that is focused on dispelling such confusion through high-quality examples.

This question is useful since it can give you a feel for how familiar the candidate is with automated testing (TDD, BDD, E2E), as well as open up a conversation about approaches to code quality.

**Source:**  
<https://github.com/daniellmb/angular-test-patterns/blob/master/patterns/filter.md>  
<https://docs.angularjs.org/guide/unit-testing>

## AngularJS Interview Question ＃2

**What should be the maximum number of concurrent “watches”? Bonus: How would you keep an eye on that number?**

(Question provided by[*Daniel Lamb*](https://www.codementor.io/daniellmb))

TL;DR Summary: To reduce memory consumption and improve performance it is a good idea to limit the number of watches on a page to 2,000. A utility called ng-stats can help track your watch count and digest cycles.

Jank happens when your application cannot keep up with the screen refresh rate. To achieve 60 frames-per-second, you only have about 16 milliseconds for your code to execute. It is crucial that the scope digest cycles are as short as possible for your application to be responsive and smooth. Memory use and digest cycle performance are directly affected by the number of active watches. Therefore, it is best to keep the number of watches below 2,000. The open-source utility ng-stats gives developers insight into the number of watches Angular is managing, as well as the frequency and duration of digest cycles over time.

Caution: Be wary of relying on a “single magic metric” as the golden rule to follow. You must take the context of your application into account. The number of watches is simply a basic health signal. If you have many thousands of watches, or worse, if you see that number continue to grow as you interact with your page. Those are strong indications that you should look under the hood and review your code.

This question is valuable as it gives insight into how the candidate debugs runtime issues while creating a discussion about performance and optimization.

**Sources:**  
<https://github.com/kentcdodds/ng-stats>  
[http://jankfree.org](http://jankfree.org/)

## AngularJS Interview Question #3

**How do you share data between controllers?**

(Question provided by[*Tome Pejoski*](https://www.codementor.io/tomepejo))

Create an AngularJS service that will hold the data and inject it inside of the controllers.

Using a service is the cleanest, fastest and easiest way to test.  
However, there are couple of other ways to implement data sharing between controllers, like:  
– Using events  
– Using $parent, nextSibling, controllerAs, etc. to directly access the controllers  
– Using the $rootScope to add the data on (not a good practice)  
The methods above are all correct, but are not the most efficient and easy to test.  
Here is [a good video explanation on egghead.io](https://egghead.io/lessons/angularjs-sharing-data-between-controllers).

## AngularJS Interview Question #4

**What is the difference between**ng-show**/**ng-hide**and**ng-if**directives?**

(Question provided by[*Tome Pejoski*](https://www.codementor.io/tomepejo))

ng-show/ng-hide will always insert the DOM element, but will display/hide it based on the condition. ng-if will not insert the DOM element until the condition is not fulfilled.

ng-if is better when we needed the DOM to be loaded conditionally, as it will help load page bit faster compared to ng-show/ng-hide.

We only need to keep in mind what the difference between these directives is, so deciding which one to use totally depends on the task requirements.

## AngularJS Interview Question #5

**What is a digest cycle in AngularJS?**

(Question provided by[*Tome Pejoski*](https://www.codementor.io/tomepejo))

In each digest cycle Angular compares the old and the new version of the scope model values. The digest cycle is triggered automatically. We can also use $apply() if we want to trigger the digest cycle manually.

For more information, take a look in the ng-book explanation: [The Digest Loop and $apply](https://www.ng-book.com/p/The-Digest-Loop-and-apply/)

## Question #6

**Where should we implement the DOM manipulation in AngularJS?**

(Question provided by[*Tome Pejoski*](https://www.codementor.io/tomepejo))

In the directives. DOM Manipulations should not exist in controllers, services or anywhere else but in directives.

Here is a [detailed explanation](http://ng-learn.org/2014/01/Dom-Manipulations)

## Question #7

**Is it a good or bad practice to use AngularJS together with jQuery?**

(Question provided by[*Tome Pejoski*](https://www.codementor.io/tomepejo))

It is definitely a bad practice. We need to stay away from jQuery and try to realize the solution with an AngularJS approach. jQuery takes a traditional imperative approach to manipulating the DOM, and in an imperative approach, it is up to the programmer to express the individual steps leading up to the desired outcome.

AngularJS, however, takes a declarative approach to DOM manipulation. Here, instead of worrying about all of the step by step details regarding how to do the desired outcome, we are just declaring what we want and AngularJS worries about the rest, taking care of everything for us.  
Here is [a detailed explanation](https://www.quora.com/Is-AngularJS-a-good-replacement-for-jQuery)

## Question #8

**If you were to migrate from Angular 1.4 to Angular 1.5, what is the main thing that would need refactoring?**

(Question provided by[*Jad Salhani*](https://www.codementor.io/jadsalhani))

Changing .directive to .component to adapt to the new Angular 1.5 components

## Question #9

**How would you specify that a scope variable should have one-time binding only?**

(Question provided by[*Jad Salhani*](https://www.codementor.io/jadsalhani))

By using “::” in front of it. This allows the check if the candidate is aware of the available variable bindings in AngularJS.

## Question #10

**What is the difference between one-way binding and two-way binding?**

(Question provided by[*Jad Salhani*](https://www.codementor.io/jadsalhani))

– One way binding implies that the scope variable in the html will be set to the first value its model is bound to (i.e. assigned to)  
– Two way binding implies that the scope variable will change it’s value everytime its model is assigned to a different value

## Question #11

**Explain how**$scope.$apply()**works**

(Question provided by[*Jad Salhani*](https://www.codementor.io/jadsalhani))

$scope.$apply re-evaluates all the declared ng-models and applies the change to any that have been altered (i.e. assigned to a new value)  
Explanation: $scope.$apply() is one of the core angular functions that should never be used explicitly, it forces the angular engine to run on all the watched variables and all external variables and apply the changes on their values  
Source: [https://docs.angularjs.org/api/ng/type/$rootScope.Scope](https://docs.angularjs.org/api/ng/type/%24rootScope.Scope)

## Question #12

**What directive would you use to hide elements from the HTML DOM by removing them from that DOM not changing their styling?**

(Question provided by[*Jad Salhani*](https://www.codementor.io/jadsalhani))

The ngIf Directive, when applied to an element, will remove that element from the DOM if it’s condition is false.

## Question #13

**What makes the**angular.copy()**method so powerful?**

(Question provided by[*Jad Salhani*](https://www.codementor.io/jadsalhani))

It creates a deep copy of the variable.

A deep copy of a variable means it doesn’t point to the same memory reference as that variable. Usually assigning one variable to another creates a “shallow copy”, which makes the two variables point to the same memory reference. Therefore if we change one, the other changes as well

**Sources:**  
– <https://docs.angularjs.org/api/ng/function/angular.copy>  
– <https://en.wikipedia.org/wiki/Object_copying>

## Question #14

**How would you make an Angular service return a promise? Write a code snippet as an example**

(Question provided by[*Jad Salhani*](https://www.codementor.io/jadsalhani))

To add promise functionality to a service, we inject the “$q” dependency in the service, and then use it like so:

angular.factory('testService', function($q){

return {

getName: function(){

var deferred = $q.defer();

//API call here that returns data

testAPI.getName().then(function(name){

deferred.resolve(name)

})

return deferred.promise;

}

}

})

The $q library is a helper provider that implements promises and deferred objects to enable asynchronous functionality

**Source:** [https://docs.angularjs.org/api/ng/service/$q](https://docs.angularjs.org/api/ng/service/%24q)

## Quesion #15

**What is the role of services in AngularJS and name any services made available by default?**

(Question provided by[*Harikishore Tadigotla*](https://www.codementor.io/harikishoretadigotla))

– AngularJS Services are objects that provide separation of concerns to an AngularJS app.  
– AngularJS Services can be created using a factory method or a service method.  
– Services are singleton components. All components of the application (into which the service is injected) will work with single instance of the service.  
– An AngularJS service allows developing of business logic without depending on the View logic which will work with it.

Few of the inbuilt services in AngularJS are:  
– the $http service: 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  
– the $log service: Simple service for logging. Default implementation safely writes the message into the browser’s console  
– the $anchorScroll: it scrolls to the element related to the specified hash or (if omitted) to the current value of $location.hash()  
Why should one know about AngularJS Services, you may ask. Well, understanding the purpose of AngularJS Services helps bring modularity to AngularJS code.  
Services are the best way to evolve reusable API within and AngularJS app

**Overview:**

* AngularJS Services help create reusable components.
* A Service can be created either using the service() method or the factory() method.
* A typical service can be injected into another service or into an AngularJS Controller.

**Source:**  
– <https://docs.angularjs.org/guide/services>  
– <http://www.tutorialspoint.com/angularjs/angularjs_services.htm>

## Question #16

**When creating a directive, it can be used in several different ways in the view. Which ways for using a directive do you know? How do you define the way your directive will be used?**

(Question provided by[*Nuno Brites*](https://www.codementor.io/nmsb))

When you create a directive, it can be used as an attribute, element or class name. To define which way to use, you need to set the restrict option in your directive declaration.

The restrict option is typically set to:

‘A’ – only matches attribute name  
‘E’ – only matches element name  
‘C’ – only matches class name

These restrictions can all be combined as needed:

‘AEC’ – matches either attribute or element or class name

For more information, feel free to check out the [AngularJS documentation](https://docs.angularjs.org/guide/directive).

## Question #17

**When should you use an attribute versus an element?**

(Question provided by[*Nuno Brites*](https://www.codementor.io/nmsb))

Use an element when you are creating a component that is in control of the template. Use an attribute when you are decorating an existing element with new functionality.

This topic is important so developers can understand the several ways a directive can be used inside a view and when to use each way.

Sources: [https://docs.angularjs.org/api/ng/service/$compile#directive-definition-object](https://docs.angularjs.org/api/ng/service/%24compile#directive-definition-object)

## Question #18

**How do you reset a**$timeout**,**$interval()**, and disable a**$watch()**?**

(Question provided by[*Fouad Kada*](https://www.codementor.io/fouadk))

To reset a timeout and/or $interval, assign the result of the function to a variable and then call the .cancel() function.

var customTimeout = $timeout(function () {

// arbitrary code

}, 55);

$timeout.cancel(customTimeout);

to disable $watch(), we call its deregistration function. $watch() then returns a deregistration function that we store to a variable and that will be called for cleanup

var deregisterWatchFn = $scope.$on(‘$destroy’, function () {

// we invoke that deregistration function, to disable the watch

deregisterWatchFn();

});

## Question #19

**Explain what is a**$scope**in AngularJS**

(Question provided by[*Ashish Kumar*](https://www.codementor.io/ashish1dev))

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. Scopes are objects that refer to the model. They act as glue between controller and view.

This question is important as it will judge a persons knowledge about a $scope object, and it is one of the most important concepts in AngularJS. Scope acts like a bridge between view and model.

**Source:** <https://docs.angularjs.org/guide/scope>

## Question #20

**What are Directives?**

(Question provided by[*Ashish Kumar*](https://www.codementor.io/ashish1dev))

Directives are markers on a DOM element (such as an attribute, element name, comment or CSS class) that tell AngularJS’s HTML compiler ($compile) to attach a specified behavior to that DOM element (e.g. via event listeners), or even to transform the DOM element and its children. Angular comes with a set of these directives built-in, like ngBind, ngModel, and ngClass. Much like you create controllers and services, you can create your own directives for Angular to use. When Angular bootstraps your application, the HTML compiler traverses the DOM matching directives against the DOM elements.

This question is important because directives define the UI while defining a single page app. You need to be very clear about how to create a new custom directive or use the existing ones already pre-build in AngularJS.

**Source:** <https://docs.angularjs.org/guide/directive>

## Question #21

**What is DDO Directive Definition Object?**

(Question provided by[*Ashish Kumar*](https://www.codementor.io/ashish1dev))

“DDO is an object used while creating a custome directive. A standard DDO object has following parameters.

var directiveDefinitionObject = {

priority: 0,

template: '<div></div>', // or // function(tElement, tAttrs) { ... },

// or

// templateUrl: 'directive.html', // or // function(tElement, tAttrs) { ... },

transclude: false,

restrict: 'A',

templateNamespace: 'html',

scope: false,

controller: function($scope, $element, $attrs, $transclude, otherInjectables) { ... },

controllerAs: 'stringIdentifier',

bindToController: false,

require: 'siblingDirectiveName', // or // ['^parentDirectiveName', '?optionalDirectiveName', '?^optionalParent'],

compile: function compile(tElement, tAttrs, transclude) {

return {

pre: function preLink(scope, iElement, iAttrs, controller) { ... },

post: function postLink(scope, iElement, iAttrs, controller) { ... }

}

// or

// return function postLink( ... ) { ... }

},

// or

// link: {

// pre: function preLink(scope, iElement, iAttrs, controller) { ... },

// post: function postLink(scope, iElement, iAttrs, controller) { ... }

// }

// or

// link: function postLink( ... ) { ... }

};"

This question mainly judges whether candidate knows about creating custom directives.

Read more at <https://docs.angularjs.org/guide/directive>

## Question #22

(Question provided by[*Jon Oyanguren López*](https://www.codementor.io/jonoya))

**What is a singleton pattern and where we can find it in Angularjs?**

Is a great pattern that restricts the use of a class more than once. We can find singleton pattern in angular in dependency injection and in the services.

In a sense, if you do 2 times ‘new Object()‘ without this pattern, you will be alocating 2 pieces of memory for the same object. With singleton pattern, if the object exists, you reuse it.

**Source:** <http://joelhooks.com/blog/2013/05/01/when-is-a-singleton-not-a-singleton/>

## Question #23

(Question provided by[*Jon Oyanguren López*](https://www.codementor.io/jonoya))

**What is an interceptor? What are common uses of it?**

An interceptor is a middleware code where all the $http requests go through.

The interceptor is a factory that are registered in $httpProvider. You have 2 types of requests that go through the interceptor, request and response (with requestError and responseErrorrespectively). This piece of code is very useful for error handling, authentication or middleware in all the requests/responses.

**Source:** [https://docs.angularjs.org/api/ng/service/$http](https://docs.angularjs.org/api/ng/service/%24http)

## Question #24

**How would you programmatically change or adapt the template of a directive before it is executed and transformed?**

(Question provided by[*Johann de Swardt*](https://www.codementor.io/johannds))

You would use the compile function. The compile function gives you access to the directive’s template before transclusion occurs and templates are transformed, so changes can safely be made to DOM elements. This is very useful for cases where the DOM needs to be constructed based on runtime directive parameters.

Read more about it [here](https://docs.angularjs.org/api/ng/service/%24compile).

## Question #25

**How would you validate a text input field for a twitter username, including the @ symbol?**

(Question provided by[*Johann de Swardt*](https://www.codementor.io/johannds))

You would use the ngPattern directive to perform a regex match that matches Twitter usernames. The same principal can be applied to validating phone numbers, serial numbers, barcodes, zip codes and any other text input.

The official documentation can be found [here](https://docs.angularjs.org/api/ng/directive/ngPattern).

## Question #26

**How would you implement application-wide exception handling in your Angular app?**

(Question provided by[*Johann de Swardt*](https://www.codementor.io/johannds))

Angular has a built-in error handler service called $exceptionHandler which can easily be overriden as seen below:

myApp.factory('$exceptionHandler', function($log, ErrorService) {

return function(exception, cause) {

if (console) {

$log.error(exception);

$log.error(cause);

}

ErrorService.send(exception, cause);

};

});

This is very useful for sending errors to third party error logging services or helpdesk applications. Errors trapped inside of event callbacks are not propagated to this handler, but can manually be relayed to this handler by calling $exceptionHandler(e) from within a try catch block.

## Question #27

**How do you hide an HTML element via a button click in AngularJS?**

(Question provided by[*Nishant Kumar*](https://www.codementor.io/nihantanu))

You can do this by using the ng-hide directive in conjunction with a controller we can hide an HTML element on button click.

<div ng-controller="MyCtrl">

<button ng-click="hide()">Hide element</button>

<p ng-hide="isHide">Hello World!</p>

</div>

function MyCtrl($scope){

$scope.isHide = false;

$scope.hide = function(){

$scope.isHide = true;

}

}

## Question #28

**How would you react on model changes to trigger some further action? For instance, say you have an input text field called**email**and you want to trigger or execute some code as soon as a user starts to type in their email.**

(Question provided by[*Nishant Kumar*](https://www.codementor.io/nihantanu))

We can achieve this using $watch function in our controller.

function MyCtrl($scope) {

$scope.email = "";

$scope.$watch("email", function(newValue, oldValue) {

if ($scope.email.length > 0) {

console.log("User has started writing into email");

}

});

}

## Question #29

**How do you disable a button depending on a checkbox’s state?**

(Question provided by[*Nishant Kumar*](https://www.codementor.io/nihantanu))

We can use the ng-disabled directive and bind its condition to the checkbox’s state.

<body ng-app>

<label><input type="checkbox" ng-model="checked"/>Disable Button</label>

<button ng-disabled="checked">Select me</button>

</body>

# AngularJS and scope.$apply

If you’ve written a non-trivial amount of code in AngularJS, you may have come across the $scope.$apply() method. On the surface, it may seem like just a method you call to get your bindings to update. But why does it exist? And when do you need to use it?

To really understand **when** to use $apply, it’s good to know exactly **why** we need to use it, so let’s dive in!

## JavaScript is Turn Based

The JavaScript code we write doesn’t all run in one go, instead it executes in turns. Each of these turns runs uninterupted from start to finish, and when a turn is running, nothing else happens in our browser. No other JavaScript code runs, and our web page interface is completely frozen. This is why poorly coded JavaScript can freeze a web page.

Instead, whenever there is a task that takes some amount of time, such as an Ajax request, waiting for a click event, or setting a timeout, we set up a callback function and finish our current turn. Later, when the Ajax request completes, a click is detected, or the timer completes, a new JavaScript turn is created and the callback is run to completion.

Let’s look at an example JavaScript file:

var button = document.getElementById('clickMe');

function buttonClicked () {

alert('the button was clicked');

}

button.addEventListener('click', buttonClicked);

function timerComplete () {

alert('timer complete');

}

setTimeout(timerComplete, 2000);

When the JavaScript code is loaded, that is a single turn. It finds a button, adds a click listener, and sets a timeout. Then the turn is complete, and the browser will update the web page if necessary, and begin accepting user input.

If the browser detects a click on #clickMe, it creates a new turn, which executes the buttonClicked function. When that function returns, that turn is complete.

After 2000 milliseconds, the browser creates a new turn which calls timerComplete.

Our JavaScript code is run in turns, and in between the turns is when the page is repainted, and input is accepted.

## How do we update bindings?

So Angular lets us bind parts of our interface to data in our JavaScript code, but how does it know when data changes, and the page needs updating?

There are a few solutions. The code needs to know when a value has changed. Right now there is no way for our code to be directly notified of changes on an object [1](http://jimhoskins.com/2012/12/17/angularjs-and-apply.html#fn:1). Instead there are two main strategies.

One strategy is to use special objects, where data is set via methods, not property assignments. Then changes can then be noted, and the page can be updated. This has the downside in that we must extend some special object. Also, for assigning, we must use a more verbose form obj.set('key', 'value') instead of obj.key = 'value'. Frameworks like [EmberJS](http://emberjs.com/) and [KnockoutJS](http://knockoutjs.com/) use this strategy.

AngularJS takes a different approach: allow any value to be used as a binding target. Then at the end of any JavaScript code turn, check to see if the value has changed. This may seem inneficient at first, but there are some clever strategies to reduce the performance hit. The big benefit is we can use normal objects and update our data however we want, and the changes will be noticed and reflected in our bindings.

For this strategy to work, we need to know when data has possibly changed, and this is where $scope.$apply comes into play.

## $apply and $digest

That step that checks to see if any binding values have changed actually has a method, $scope.$digest(). That’s actually where the magic happens, but we almost never call it directly, instead we use $scope.$apply() which will call $scope.$digest()for you.

$scope.$apply() takes a function or an Angular expression string, and executes it, then calls $scope.$digest() to update any bindings or watchers.

So, when do you need to call $apply()? Very rarely, actually. AngularJS actually calls almost all of your code within an $apply call. Events like ng-click, controller initialization, $http callbacks are all wrapped in $scope.$apply(). So you don’t need to call it yourself, in fact you can’t. Calling $apply inside $apply will throw an error.

You do need to use it if you are going to run code in a new turn. And only if that turn isn’t being created from a method in the AngularJS library. Inside that new turn, you should wrap your code in $scope.$apply(). Here is an example. We are using setTimeout, which will execute a function in a new turn after a delay. Since Angular doesn’t know about that new turn, the update will not be reflected.

But, if we wrap the code for that turn in $scope.$apply(), the change will be noticed, and the page is updated.

As a convenience, AngularJS provides[*$timeout*](http://docs.angularjs.org/api/ng.$timeout), which is like*setTimeout*, but automatically wraps your code in $apply by default. Use that, not this

If you write any code that uses Ajax without $http, or listens for events without using Angular’s ng-\* listeners, or sets a timeout without $timeout, you should wrap your code in $scope.$apply

## $scope.$apply() vs $scope.$apply(fn)

Sometimes I see examples where data is updated, and then $scope.$apply() is called with no arguments. This achieves the desired result, but misses some opportunities.

If your code isn’t wrapped in a function passed to $apply, and it throws an error, that error is thrown outside of AngularJS, which means any error handling being used in your application is going to miss it. $apply not only runs your code, but it runs it in a try/catch so your error is always caught, and the $digest call is in a finally clause, meaning it will run regardless of an error being thrown. That’s pretty nice.

1. Can the angular application be initialized on only HTML element or ANY element?  
   **Ans**: ANY element
2. Can an HTML page have multiple “ng-app” directive for bootstrapping multiple AngularJS application?  
   **Ans**: Yes
3. With more than one ng-app in an HTML document (an HTML page), are they automatically initialized? Describe the angularJS application initialization process with multiple ng-app in an HTML document/page.  
   **Ans**: Only one AngularJS application can be auto-bootstrapped. The first ‘ng-app’ found in the document will be used to define the root element to auto-bootstrap as an application. To run multiple applications in an HTML document, one must manually bootstrap them using angular bootstrap service.
4. Describe the steps involved in bootstrapping an angular application?  
   **Ans**: Following steps are involved in bootstrapping the angular application:
   * Determine the element consisting of ng prefixes with “app”
   * Check whethar an angular app already exists on the element
   * Compile & link with the help of dependencies such as rootScope injected (used for linking).
5. Can angular applications (ng-app) be nested within each other?  
   **Ans**: No
6. What are various possible prefixes such as “ng-” using which Angular directives (for example, ng-app) can be defined?  
   **Ans**: “ng-“, “data-ng-“, “ng:”, “x-ng-“
7. What are various possible ways in which angular application can be initialized?  
   **Ans**: On an element, one could either put simply the attribute such as (ng-app, data-ng-app, ng:app, x-ng-app) or put the named attribute such as (ng-app=”demoApp”).
8. What angular function is used to manually start up an angular application?  
   **Ans**: angular.bootstrap
9. Can multiple angular applications be bootstrapped using same element?  
   **Ans**: No. An error is thrown such as “App Already Bootstrapped with this Element”
10. At framework level, how does Angular retrieve the matching elements for processing?  
    **Ans**: Makes use of jqLite(element) function. If jQuery is used, jQuery(element) is used by way of assigning jQuery to jqLite variable.>

What is AngularJS?

AngularJS is a framework to build large scale and high performance web application while keeping them as easy-to-maintain. Following are the features of AngularJS framework.

* AngularJS is a powerful JavaScript based development framework to create RICH Internet Application RIARIA.
* AngularJS provides developers options to write client side application usingJavaScriptusingJavaScript in a clean MVC ModelViewControllerModelViewController way.
* Application written in AngularJS is cross-browser compliant. AngularJS automatically handles JavaScript code suitable for each browser.
* AngularJS is open source, completely free, and used by thousands of developers around the world. It is licensed under the Apache License version 2.0.

What is data binding in AngularJS?

Data binding is the automatic synchronization of data between model and view components. ng-model directive is used in data binding.

What is scope in AngularJS?

Scopes are objects that refer to the model. They act as glue between controller and view.

What are the controllers in AngularJS?

Controllers are JavaScript functions that are bound to a particular scope. They are the prime actors in AngularJS framework and carry functions to operate on data and decide which view is to be updated to show the updated model based data.

What are the services in AngularJS?

AngularJS come with several built-in services. For example $https: service is used to make XMLHttpRequests (Ajax calls). Services are singleton objects which are instantiated only once in app.

What are the filters in AngularJS?

Filters select a subset of items from an array and return a new array. Filters are used to show filtered items from a list of items based on defined criteria.

Explain directives in AngularJS.

Directives are markers on DOM elements suchaselements,attributes,css,andmoresuchaselements,attributes,css,andmore. These can be used to create custom HTML tags that serve as new, custom widgets. AngularJS has built-in directives ng−bind,ng−model,etcng−bind,ng−model,etc to perform most of the task that developers have to do.

Explain templates in AngularJS.

Templates are the rendered view with information from the controller and model. These can be a single file likeindex.htmllikeindex.html or multiple views in one page using "partials".

What is routing in AngularJS?

It is concept of switching views. AngularJS based controller decides which view to render based on the business logic.

What is deep linking in AngularJS?

Deep linking allows you to encode the state of application in the URL so that it can be bookmarked. The application can then be restored from the URL to the same state.

What are the advantages of AngularJS?

Following are the advantages of AngularJS.

* AngularJS provides capability to create Single Page Application in a very clean and maintainable way.
* AngularJS provides data binding capability to HTML thus giving user a rich and responsive experience.
* AngularJS code is unit testable.
* AngularJS uses dependency injection and make use of separation of concerns.
* AngularJS provides reusable components.
* With AngularJS, developer writes less code and gets more functionality.
* In AngularJS, views are pure html pages, and controllers written in JavaScript do the business processing.
* AngularJS applications can run on all major browsers and smart phones including Android and iOS based phones/tablets.

What are the disadvantages of AngularJS?

Following are the disadvantages of AngularJS.

* **Not Secure** − Being JavaScript only framework, application written in AngularJS are not safe. Server side authentication and authorization is must to keep an application secure.
* **Not degradable** − If your application user disables JavaScript then user will just see the basic page and nothing more.

Which are the core directives of AngularJS?

Following are the three core directives of AngularJS.

* **ng-app** − This directive defines and links an AngularJS application to HTML.
* **ng-model** − This directive binds the values of AngularJS application data to HTML input controls.
* **ng-bind** − This directive binds the AngularJS Application data to HTML tags.

Explain AngularJS boot process.

When the page is loaded in the browser, following things happen:

* HTML document is loaded into the browser, and evaluated by the browser. AngularJS JavaScript file is loaded; the angular *global* object is created. Next, JavaScript which registers controller functions is executed.
* Next AngularJS scans through the HTML to look for AngularJS apps and views. Once view is located, it connects that view to the corresponding controller function.
* Next, AngularJS executes the controller functions. It then renders the views with data from the model populated by the controller. The page gets ready.

What is MVC?

**M**odel **V**iew **C**ontroller 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.

Explain ng-app directive.

ng-app directive defines and links an AngularJS application to HTML. It also indicates the start of the application.

Explain ng-model directive.

ng-model directive binds the values of AngularJS application data to HTML input controls. It creates a model variable which can be used with the html page and within the container controlforexample,divforexample,div having ng-app directive.

Explain ng-bind directive.

ng-bind directive binds the AngularJS Application data to HTML tags. ng-bind updates the model created by ng-model directive to be displayed in the html tag whenever user input something in the control or updates the html control's data when model data is updated by controller.

Explain ng-controller directive.

ng-controller directive tells AngularJS what controller to use with this view. AngularJS application mainly relies on controllers to control the flow of data in the application. A controller is a JavaScript object containing attributes/properties and functions. Each controller accepts $scope as a parameter which refers to the application/module that controller is to control.

How AngularJS integrates with HTML?

AngularJS being a pure javaScript based library integrates easily with HTML.

**Step 1** − Include angularjs javascript libray in the html page

<head>

<script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>

</head>

**Step 2** − Point to AngularJS app

Next we tell what part of the HTML contains the AngularJS app. This done by adding the *ng-app* attribute to the root HTML element of the AngularJS app. You can either add it to *html* element or *body* element as shown below:

<body ng-app = "myapp">

</body>

Explain ng-init directive.

ng-init directive initializes an AngularJS Application data. It is used to put values to the variables to be used in the application.

Explain ng-repeat directive.

ng-repeat directive repeats html elements for each item in a collection.

What are AngularJS expressions?

Expressions are used to bind application data to html. Expressions are written inside double braces like {{ expression}}. Expressions behave in same way as ng-bind directives. AngularJS application expressions are pure JavaScript expressions and outputs the data where they are used.

Explain uppercase filter.

Uppercase filter converts a text to upper case text.

In below example, we've added uppercase filter to an expression using pipe character. Here we've added uppercase filter to print student name in all capital letters.

Enter first name:<input type = "text" ng-model = "student.firstName">

Enter last name: <input type = "text" ng-model = "student.lastName">

Name in Upper Case: {{student.fullName() | uppercase}}

Explain lowercase filter.

Lowercase filter converts a text to lower case text.

In below example, we've added lowercase filter to an expression using pipe character. Here we've added lowercase filter to print student name in all lowercase letters.

Enter first name:<input type = "text" ng-model = "student.firstName">

Enter last name: <input type = "text" ng-model = "student.lastName">

Name in Upper Case: {{student.fullName() | lowercase}}

Explain currency filter.

Currency filter formats text in a currency format.

In below example, we've added currency filter to an expression returning number using pipe character. Here we've added currency filter to print fees using currency format.

Enter fees: <input type = "text" ng-model = "student.fees">

fees: {{student.fees | currency}}

Explain filter filter.

filter filter is used to filter the array to a subset of it based on provided criteria.

In below example, to display only required subjects, we've used subjectName as filter.

Enter subject: <input type = "text" ng-model = "subjectName">

Subject:

<ul>

<li ng-repeat = "subject in student.subjects | filter: subjectName">

{{ subject.name + ', marks:' + subject.marks }}

</li>

</ul>

Explain orderby filter.

orderby filter orders the array based on provided criteria.

In below example, to order subjects by marks, we've used orderBy marks.

Subject:

<ul>

<li ng-repeat = "subject in student.subjects | orderBy:'marks'">

{{ subject.name + ', marks:' + subject.marks }}

</li>

</ul>

Explain ng-disabled directive.

ng-disabled directive disables a given control.

In below example, we've added ng-disabled attribute to a HTML button and pass it a model. Then we've attached the model to an checkbox and can see the variation.

<input type = "checkbox" ng-model = "enableDisableButton">Disable Button

<button ng-disabled = "enableDisableButton">Click Me!</button>

Explain ng-show directive.

ng-show directive shows a given control.

In below example, we've added ng-show attribute to a HTML button and pass it a model. Then we've attached the model to a checkbox and can see the variation.

<input type = "checkbox" ng-model = "showHide1">Show Button

<button ng-show = "showHide1">Click Me!</button>

Explain ng-hide directive.

ng-hide directive hides a given control.

In below example, we've added ng-hide attribute to a HTML button and pass it a model. Then we've attached the model to a checkbox and can see the variation.

<input type = "checkbox" ng-model = "showHide2">Hide Button

<button ng-hide = "showHide2">Click Me!</button>

Explain ng-click directive.

ng-click directive represents a AngularJS click event.

In below example, we've added ng-click attribute to a HTML button and added an expression to updated a model. Then we can see the variation.

<p>Total click: {{ clickCounter }}</p></td>

<button ng-click = "clickCounter = clickCounter + 1">Click Me!</button>

l

How angular.module works?

angular.module is used to create AngularJS modules along with its dependent modules. Consider the following example:

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

Here we've declared an application **mainApp** module using angular.module function. We've passed an empty array to it. This array generally contains dependent modules declared earlier.

How to validate data in AngularJS?

AngularJS enriches form filling and validation. We can use dirtyanddirtyandinvalid flags to do the validations in seamless way. Use novalidate with a form declaration to disable any browser specific validation.

Following can be used to track error.

* **$dirty** − states that value has been changed.
* **$invalid** − states that value entered is invalid.
* **$error** − states the exact error.

Explain ng-include directive.

Using AngularJS, we can embed HTML pages within a HTML page using ng-include directive.

<div ng-app = "" ng-controller = "studentController">

<div ng-include = "'main.htm'"></div>

<div ng-include = "'subjects.htm'"></div>

</div>

How to make an ajax call using Angular JS?

AngularJS provides https:control which works as a servicetomakeajaxcalltoreaddatafromtheserver.Theservermakesadatabasecalltogetthedesiredrecords.AngularJSneedsdatainJSONformat.Oncethedataisready,https:controlwhichworksasaservicetomakeajaxcalltoreaddatafromtheserver.Theservermakesadatabasecalltogetthedesiredrecords.AngularJSneedsdatainJSONformat.Oncethedataisready,https: can be used to get the data from server in the following manner:

function studentController($scope,$https:) {

var url = "data.txt";

$http.get(url).success( function(response) {

$scope.students = response;

});

}

What is use of $routeProvider in AngularJS?

$routeProvider is the key service which set the configuration of urls, maps them with the corresponding html page or ng-template, and attaches a controller with the same.

What is $rootScope?

Scope is a special JavaScript object which plays the role of joining controller with the views. Scope contains the model data. In controllers, model data is accessed via scopeobject.scopeobject.rootScope is the parent of all of the scope variables.

What is scope hierarchy in AngularJS?

Scopes are controllers specific. If we define nested controllers then child controller will inherit the scope of its parent controller.

<script>

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

mainApp.controller("shapeController", function($scope) {

$scope.message = "In shape controller";

$scope.type = "Shape";

});

mainApp.controller("circleController", function($scope) {

$scope.message = "In circle controller";

});

</script>

Following are the important points to be considered in above example.

* We've set values to models in shapeController.
* We've overridden message in child controller circleController. When "message" is used within module of controller circleController, the overridden message will be used.

What is a service?

Services are JavaScript functions and are responsible to do specific tasks only. Each service is responsible for a specific task for example, https:isusedtomakeajaxcalltogettheserverdata.https:isusedtomakeajaxcalltogettheserverdata.route is used to define the routing information and so on. Inbuilt services are always prefixed with $ symbol.

What is service method?

Using service method, we define a service and then assign method to it. We've also injected an already available service to it.

mainApp.service('CalcService', function(MathService){

this.square = function(a) {

return MathService.multiply(a,a);

}

});

What is factory method?

Using factory method, we first define a factory and then assign method to it.

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

mainApp.factory('MathService', function() {

var factory = {};

factory.multiply = function(a, b) {

return a \* b

}

return factory;

});

What are the differences between service and factory methods?

factory method is used to define a factory which can later be used to create services as and when required whereas service method is used to create a service whose purpose is to do some defined task.

Which components can be injected as a dependency in AngularJS?

AngularJS provides a supreme Dependency Injection mechanism. It provides following core components which can be injected into each other as dependencies.

* value
* factory
* service
* provider
* constant

What is provider?

provider is used by AngularJS internally to create services, factory etc. during config phasephaseduringwhichAngularJSbootstrapsitselfphaseduringwhichAngularJSbootstrapsitself. Below mention script can be used to create MathService that we've created earlier. Provider is a special factory method with a method get which is used to return the value/service/factory.

//define a module

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

...

//create a service using provider which defines a method square to return square of a number.

mainApp.config(function($provide) {

$provide.provider('MathService', function() {

this.$get = function() {

var factory = {};

factory.multiply = function(a, b) {

return a \* b;

}

return factory;

};

});

});

What is constant?

constants are used to pass values at config phase considering the fact that value cannot be used to be passed during config phase.

mainApp.constant("configParam", "constant value");

Is AngularJS extensible?

Yes! In AngularJS we can create custom directive to extend AngularJS existing functionalities.

Custom directives are used in AngularJS to extend the functionality of HTML. Custom directives are defined using "directive" function. A custom directive simply replaces the element for which it is activated. AngularJS application during bootstrap finds the matching elements and do one time activity using its compile method of the custom directive then process the element using link method of the custom directive based on the scope of the directive.

On which types of component can we create a custom directive?

AngularJS provides support to create custom directives for following type of elements.

* **Element directives** − Directive activates when a matching element is encountered.
* **Attribute** − Directive activates when a matching attribute is encountered.
* **CSS** − Directive activates when a matching css style is encountered.
* **Comment** − Directive activates when a matching comment is encountered.

What is internationalization?

Internationalization is a way to show locale specific information on a website. For example, display content of a website in English language in United States and in Danish in France.

How to implement internationalization in AngularJS?

AngularJS supports inbuilt internationalization for three types of filters currency, date and numbers. We only need to incorporate corresponding js according to locale of the country. By default it handles the locale of the browser. For example, to use Danish locale, use following script

<script src = "https://code.angularjs.org/1.2.5/i18n/angular-locale\_da-dk.js"></script>

**Question: What is AngularJS?**   
It is javasScript framework which is written in javascript. It is Best for Single Page Applications. It extend the html with new attributes which makes it more useful for UI Developer.  
  
  
**Question: In which language, AngularJS is written?**  
javaScript  
  
  
**Question: When First AngularJS was released?**  
2009  
  
**Question: When latest AngularJS was released?**  
February 05, 2016  
  
  
**Question: What is latest version of AngularJS?**  
1.5.0   
  
  
**Question: Who created AngularJS?**  
Misko Hevery started to work on AngularJS in 2009. He was employee of Google.  
**Question: Is it opensource?**  
Yes, It is free to use.  
  
  
  
**Question: Explain what are the key features of Angular.js?**

1. Scope
2. Controller
3. Model
4. View
5. Services
6. Data Binding
7. Directives
8. Filters
9. Testable

**Question: From where we can download the AngularJS File?**

<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.26/angular.min.js"></script>

**Question: What is controller in AngularJS?**  
Controller is constructor function in Angular Controller.  
When a Controller is attached to the DOM with use the ng-controller, Angular will instantiate a new Controller object using constructor function.  
  
  
  
**Question: Explain what are directives?**  
Directives are used to add new attributes of HTML.  
  
  
  
**Question: What are the different types of Directive?**  
Different types of directives are

1. Element directives
2. Attribute directives
3. CSS class directives
4. Comment directives

**Question: Explain what is injector?**  
An injector is a service locator, used to retrieve object instances.  
  
  
  
**Question: Explain what are factory method in angularJs?**  
Factory method are used to create the directive. It is invoked only once, when compiler matches the directive for the first time.  
  
  
  
**Question: Does Angular use the jQuery library?**  
Ans. Yes, Angular can use jQuery if you have included the jQuery library.  
IF Not, Angular falls back to its own implementation of the subset of jQuery that we call jQLite.  
  
  
  
**Question: What is ng-app, ng-init and ng-model?**  
ng-app - To initialize the Angular Application.  
ng-init - To initialize the Angular Application data.  
ng-model - To bind the html tags (input, select, textarea) to Angular Application Data.  
  
  
  
**Question: What is Data Binding in Angular JS?**  
It is synchronization of data between the model(Angular Application variable) and view components (display with {{}}).  
  
  
  
**Question: Give an Example of Data-Binding in AngularJS?**

<div ng-app="" ng-init="quantity=10;cost=5">

<b>Total Cost: {{ quantity \* cost }}</b>

</div>

**Question: What is Looping in AngularJs and Give an Example?**  
It is used to display the data in loop same as foreach in PHP  
Example:

<div data-ng-app="" data-ng-init="names=['Web','Technology','Experts','Notes']">

<b>Loop Example:</b>

<br />

<ul>

<li data-ng-repeat="x in names">

{{ x }}

</li>

</ul>

</div>

**Question: How to Write Expression in AngularJS?**

<div ng-app="">

<b>Expression: {{ 15 + 55 }}</b>

</div>

**Question: How to initiate variable in AngularJS?**

<div ng-app="" ng-init="quantity=10;cost=5">

<b>Total Cost: {{ quantity \* cost }}</b>

</div>

OR

<div ng-app="" ng-init="quantity=1;cost=5">

<b>Total Cost: <span ng-bind="quantity \* cost"></span></b>

</div>

**Question: Example of AngularJS Strings?**

<div ng-app="" ng-init="Str1='Web';Str2='Technology'">

Full String is : <b>{{ Str1 + " " + Str2 }}</b>

</div>

**Question: Example of AngularJS Object?**

<div ng-app="" ng-init="myobj={Str1:'Web',Str2:'Technology'}">

String Display: <b>{{ myobj.Str2 }}</b></div>

**Question: What is Angular Controllers & give an Example?**  
Controller is constructor function in Angular Controller.  
When a Controller is attached to the DOM with use the ng-controller, Angular will instantiate a new Controller object using constructor function.  
Example:

<div ng-app="" ng-controller="StrControllerExample">

String 1: <input ng-model="str1" type="text" /><br />

String 2: <input ng-model="str2" type="text" /><br />

Full String <b> {{fullString()}}</b>

</div>

<script>

function StrControllerExample($scope) {

$scope.str1 = "Web",

$scope.str2 = "Technology",

$scope.fullString = function() {

return $scope.str1+ " " + $scope.str2;

}

}

</script>

## 1. Explain data binding in AngularJS.

According to AngularJS.org, “Data-binding in Angular apps is the automatic synchronization of data between the model and view components. The way that Angular implements data-binding lets you treat the model as the single-source-of-truth in your application. The view is a projection of the model at all times. When the model changes, the view reflects the change, and vice versa.”

There are two ways of data binding:

1. Data mining in classical template systems
2. Data binding in angular templates

## 2. Name the key features of AngularJS ?

The key features of AngularJS are:

* Scope
* Controller
* Model
* View
* Services
* Data Binding
* Directives
* Filters
* Testable

## 3. What are directives in AngularJS?

A core feature of AngularJS, directives are attributes that allow you to invent new HTML syntax, specific to your application. They are essentially functions that execute when the Angular compiler finds them in the DOM.  Some of the most commonly used directives are ng-app,ng-controller and ng-repeat.

The different types of directives are:

* Element directives
* Attribute directives
* CSS class directives
* Comment directives

## 4. What are Controllers in AngularJS?

Controllers are Javascript functions which provide data and logic to HTML UI. As the name suggests, they control how data flows from the server to HTML UI.

## 5. What is Angular Expression? How do you differentiate between Angular expressions and JavaScript expressions?

Angular expressions are code snippets that are usually placed in binding such as {{ expression }} similar to JavaScript.

The main differences between Angular expressions and JavaScript expressions are:

* **Context :** The expressions are evaluated against a scope object in Angular, while Javascript expressions are evaluated against the global window
* **Forgiving:** In Angular expression, the evaluation is forgiving to null and undefined whereas in JavaScript undefined properties generate TypeError or ReferenceError
* **No Control Flow Statements:** We cannot use loops, conditionals or exceptions in an Angular expression
* **Filters:** In Angular unlike JavaScript, we can use filters to format data before displaying it

## 6. What is the difference between link and compile in Angular.js?

* Compile function is used for template DOM Manipulation and to collect all the directives.
* Link function is used for registering DOM listeners as well as instance DOM manipulation and is executed once the template has been cloned.

## 7. What are the characteristics of ‘Scope’?

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. The characteristics of Scope are:

* 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 can be nested to limit access to the properties of application components while providing access to shared model properties. Nested scopes are either “child scopes” or “isolate scopes”. A “child scope” (prototypically) inherits properties from its parent scope. An “isolate scope” does not. See isolated scopes for more information.
* 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.

## 8. What are the advantages of using Angular.js framework?

Angular.js framework has the following advantages:

* Supports two way data-binding
* Supports MVC pattern
* Support static template and angular template
* Can add custom directive
* Supports REST full services
* Supports form validations
* Support both client and server communication
* Support dependency injection
* Applying Animations
* Event Handlers

## 9. What is the difference between AngularJS and backbone.js?

AngularJS combines the functionalities of most third party libraries and supports individual functionalities required to develop HTML5 Apps.  While Backbone.js does these jobs individually.

## 10. Explain what is injector in AngularJS?

An injector is a service locator, used to retrieve object instance as defined by provider, instantiate types, invoke methods, and load modules.

## 11. What is factory method in AngularJS?

Factory method is used for creating a directive.  It is invoked when the compiler matches the directive for the first time. We can invoke the factory method using $injector.invoke.

Syntax: module.factory( 'factoryName', function );  
Result: When declaring factoryName as an injectable argument you will be provided with the value that is returned by invoking the function reference passed to module.factory.

## 12. What is ng-app, ng-init and ng-model?

* ng-app : Initializes application.
* ng-model : Binds HTML controls to application data.
* ng-Controller : Attaches a controller class to view.
* ng-repeat : Bind repeated data HTML elements. Its like a for loop.
* ng-if : Bind HTML elements with condition.
* ng-show : Used to show the HTML elements.
* ng-hide : Used to hide the HTML elements.
* ng-class : Used to assign CSS class.
* ng-src : Used to pass the URL image etc.

## 13. Does Angular use the jQuery library?

Yes, Angular can use jQuery if it’s present in the app when the application is being bootstrapped. If jQuery is not present in the script path, Angular falls back to its own implementation of the subset of jQuery that we call jQLite.

## 14. Can AngularJS have multiple ng-app directives in a single page?

No. Only one AngularJS application can be auto-bootstrapped per HTML document. The first ngApp found in the document will be used to define the root element to auto-bootstrap as an application. If another ng-app directive has been placed then it will not be processed by AngularJS and we will need to manually bootstrap the second app, instead of using second ng-app directive.

## 15. Can angular applications (ng-app) be nested within each other?

No. AngularJS applications cannot be nested within each other.

## 16. What is internationalization and how to implement it in AngularJS?

Internationalization is a way in which you can show locale specific information on a website. AngularJS supports inbuilt internationalization for three types of filters: currency, date and numbers. To implement internalization, we only need to incorporate corresponding js according to locale of the country. By default it handles the locale of the browser.

## 17. On which types of component can we create a custom directive?

AngularJS provides support to create custom directives for the following:

* **Element directives** − Directive activates when a matching element is encountered.
* **Attribute** − Directive activates when a matching attribute is encountered.
* **CSS** − Directive activates when a matching css style is encountered.
* **Comment** − Directive activates when a matching comment is encountered.

## 18. What is $rootscope in AngularJS?

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 event emission/broadcast and subscription facility.

## 19. Can we have nested controllers in AngularJS?

Yes, we can create nested controllers in AngularJS. Nested controllers are defined in hierarchical manner while using in View.

## 20. What is bootstrapping in AngularJS?

Bootstrapping in AngularJS is nothing but initializing, or starting the Angular app. AngularJS supports automatic and manual bootstrapping.

* Automatic Bootstrapping: this is done by adding ng-app directive to the root of the application, typically on the tag or tag if you want angular to bootstrap your application automatically. When angularJS finds ng-app directive, it loads the module associated with it and then compiles the DOM.
* Manual Bootstrapping:Manual bootstrapping provides you more control on how and when to initialize your angular App. It is useful where you want to perform any other operation before Angular wakes up and compile the page.

## 21. What does SPA (Single Page Application) mean? How can we implement SPA with Angular?

Single Page Applications (SPAs) are web apps that load a single HTML page and dynamically update that page as the user interacts with the app. In an SPA the page never reloads, though parts of the page may refresh. This reduces the round trips to the server to a minimum.

It’s a concept where we create a single shell page or master page and load the webpages inside that master page instead of loading pages from the server by doing post backs. We can implement SPA with Angular using Angular routes. You can read up about SPAs [here](http://www.edureka.co/blog/spa-using-angularjs).

## 22. Why AngularJS?

AngularJS lets us extend HTML vocabulary for our application resulting in an expressive, readable, and quick to develop environment . Some of the advantages are:

* MVC implementation is done right.
* It extends HTML using directives, expression and data binding techniques to define a powerful HTML template.
* Two way data-binding, form validations, routing supports, inbuilt services.
* REST friendly.
* Dependency injection support.
* It helps you to structure and test your JavaScript code.

## 23. Is AngularJS compatible with all browsers?

Yes AngularJS is compatible with the following browsers: Safari, Chrome, Firefox, Opera 15, IE9 and mobile browsers (Android, Chrome Mobile, iOS Safari).

## 24. How to implement routing in AngularJS?

It is a five-step process:

* Step 1: – Add the “Angular-route.js” file to your view.
* Step 2: – Inject “ngroute” functionality while creating Angular app object.
* Step 3: – Configure the route provider.
* Step 4: – Define hyperlinks.
* Step 5: – Define sections where to load the view.

## 25. Explain $q service, deferred and promises.

* ‘Promises’ are post processing logics which are executed after some operation/action is completed whereas ‘deferred’ is used to control how and when those promise logics will execute.
* We can think about promises as “WHAT” we want to fire after an operation is completed while deferred controls “WHEN” and “HOW” those promises will execute.
* “$q” is the angular service which provides promises and deferred functionality.

**The big idea behind AngularJS**

AngularJS brings in a revolution and takes up challenges to create what is called a “Single Page Application”, wherein the additional content is loaded on to the same page as required without the user really making out the difference of being on the same page! The objective behind only loading the part that is needed is obvious, as this clearly adds on to the speed, which ultimately points towards the huge advantage of less code that AngularJS offers.

**AngularJS’s popularity from a business growth/application development standpoint**

AngularJS’s structure and built-in features promote the following, which helps earn its popularity from a business standpoint.

* Helps to create software while saving effort and time.
* Uses best programming practices.
* Helps to promote work and collaborate as a team.
* Helps in maintaining software
* Makes testability easy
* Helps in packaging and deploying the application to mobile devices and the web

**Angular’s popularity from a technology standpoint**

AngularJS extends HTML and uses JavaScript language. So for those already familiar with JavaScript language, understanding AngularJS wouldn’t be a big deal. Further, the fundamentals of AngularJS will surely come easy to those who have an understanding of the MVC-Model View Controller Architecture. The list below highlights the stand-out features of AngularJS that makes it powerful as well as easy to use.

**Client-side MVC**

AngularJS brings in the power of MVC framework on to the client-side programming. This is what empowers AngularJS with the entire idea of single-page application (SPA) development resting on the MVC framework. The programmer can now segregate client-side business logic from the UI by breaking the programme into specific components – model, view and controller –  each being categorized to perform specific tasks. This brings in all the popular benefits of an MVC architecture. The components are now more flexible and are loosely coupled, further leading to the ease of code development as well as ease of code maintenance.

The MVC framework works on the basic idea of breaking down the programme into three different components. The model represents the current state of the application and encapsulates the application’s data; the view is the HTML pages with the data to be presented as UI; and controller contains the logic for the link between the view and the model.  
On similar lines, in AngularJS, model components represent data in the form of a data structure or as JSON or from a database. This data is stored in something called as scope objects ($scope) which are used for temporary storage. Data is bound to these scope objects and made available to the view. The HTML templates, CSS and the AngularJS’s directives contribute as view components. The controller or view-model is a constructor function that takes scope objects as parameter to link the model and the view. It is written using JavaScript language, making it easy for the developers already familiar with the language. A view is generally linked to one or more controllers implementing different business logic. Also, the controllers are packed separately into modules which makes the code easy to maintain.

**A declarative UI**

The templates in the HTML page use directives form the basic building blocks in the AngularJS’s construct of view. They are like custom HTML tags and play a key role in the template. They are commands that tell AngularJS to do specific jobs. ng-app, ng-controller, ng- model, ng-view are a few main directives (ng is used by AngularJS to name spacing). The use of directives along with the HTML page offers the developers a declarative approach to UI. The use of directives give a more defined look to the HTML page and just by looking at the page it is easy to figure out where the data bindings and function calls are taking place.

**Document Object Model (DOM) manipulation**

DOM manipulation has originally been a part of the view, where it was with the view to modify the DOM to add behaviour or to update DOM to present the data. AngularJS offers the flexibility of putting all the DOM manipulation code into directives, thus easily separating that code out from the view and making it available as standalone reusable units. This helps the UI developer to concentrate fully on the user interface and the application developer to work separately on DOM manipulations and JQuery calls.

**Two-way data binding with MVVM**

In AngularJS, model / data is referenced using scope objects ($scope). Scope can be looked upon as temporary storage to put and get the data. Properties as well as functions can be attached to the scope to make them available to the view. The concept of “scope” in AngularJS is crucial as it helps the model, view and controller work in sync. It helps in updating corresponding changes in the model as per the changes in the view and vice versa by refreshing the appropriate binding points in the templates. The two-way data-binding and the nature of scope objects have led to the controller being referenced as view-model. Hence, the AngularJS architecture as MVVM (Model, View, View-Model) is popular.

So again, no explicit coding for wrapper classes etc. is required for checking event triggers and updates. Scope being the link between the model and the view, keeps both updated automatically with changes occurring in either. So the magic of two-way data binding greatly contributes in reduced code and time.

**‘Services’ by AngularJS**

With different model, view and controller / view-model components, programmers need to link or bind them to be able to execute the desired output. The link or binding of the components is taken care by AngularJS’s built-in services. Hence, separate code for binding the components together is not required as it is already taken care by AngularJS.

Apart from linking the MVC components, there are many useful built-in services that ease out the day for a programmer.

* $http – to make an Ajax call
* $routeProvider – The AngularJS’s design to execute single-page application works upon dividing the single page into multiple logical views for better management. We then require something to load the different parts as needed, which is where the routing service comes in. $routeProvider does routing and deep linking to organize the views and controllers.

**Dependency injection sub-system**

The basic programming concept of having dedicated functions to perform specific tasks and to have the required data for the functions to be passed as parameters into them instead of hardcoding it inside the function form the basis of the idea behind what we call dependencies in AngularJS.

Working on the same concept, AngularJS allows various components to use other components on which they are ‘dependent’ with the help of dependency injection sub-system. This is a design pattern that helps the component to locate its dependencies. For instance, a controller might be dependent on a service and hence by declaring its dependencies, AngularJS’s dependency injection system makes the service available for the controller’s use. Apart from making a programmer’s task easy, this software design pattern helps in keeping the components reusable, easy to maintain and ready for testing.

**Great for unit testing**

JavaScript that comes with almost no help from the compiler strongly needs to be tested and this set up in AngularJS, where we have different components dedicated to specific tasks, leaves the developer / tester with little scope for not testing it!

So to summarize, AngularJS with its powerful design pattern, saves big on reduced code and development time.

What is AngularJS ?

*“AngularJS is a JavaScript framework which simplifies binding JavaScript objects with HTML UI elements.”*

Let us try to understand the above definition with simple sample code.

Below is a simple “Customer” function with “CustomerName” property. We have also created an object called as “Cust” which is of “Customer” class type.

Hide   Copy Code

function Customer()

{

this.CustomerName = "AngularInterview";

}

var Cust = new Customer();

Now let us say the above customer object we want to bind to a HTML text box called as “TxtCustomerName”. In other words when we change something in the HTML text box the customer object should get updated and when something is changed internally in the customer object the UI should get updated.

Hide   Copy Code

<input type=text id="TxtCustomerName" onchange="UitoObject()"/>

So in order to achieve this communication between UI to object developers end up writing functions as shown below. “UitoObject” function takes data from UI and sets it to the object while the other function “ObjecttoUI” takes data from the object and sets it to UI.

Hide   Copy Code

function UitoObject()

{

Cust.CustomerName = $("#TxtCustomerName").val();

}

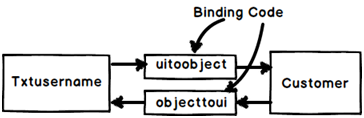
function ObjecttoUi()

{

$("#TxtCustomerName").val(Cust.CustomerName);

}

So if we analyze the above code visually it looks something as shown below. Your both functions are nothing but binding code logic which transfers data from UI to object and vice versa.



Now the same above code can be written in Angular as shown below. The javascript class is attached to a HTML parent div tag using “ng-controller” directive and the properties are binded directly to the text box using “ng-model” declarative.

So now whatever you type in the textbox updates the “Customer” object and when the “Customer” object gets updated it also updates the UI.

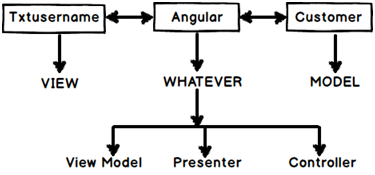
Hide   Copy Code

<div ng-controller="Customer">

<input type=text id="txtCustomerName" ng-model="CustomerName"/>

</div>

In short if you now analyze the above code visually you end up with something as shown in the below figure.You have the VIEW which is in HTML, your MODEL objects which are javascript functions and the binding code in Angular.



Now that binding code have different vocabularies.

* Some developers called it “ViewModel” because it connects the “Model” and the “View” .
* Some call it “Presenter” because this logic is nothing but presentation logic.
* Some term it has “Controller” because it controls how the view and the model will communicate.

To avoid this vocabulary confusion Angular team has termed this code as “Whatever”. It’s that “Whatever” code which binds the UI and the Model. That’s why you will hear lot of developers saying Angular implements “MVW” architecture.

Explain Directives in Angular?

Directives are attributes decorated on the HTML elements. All directives start with the word “ng”. As the name says directive it directs Angular what to do.

For example below is a simple “ng-model” directive which tells angular that the HTML textbox “txtCustomerName” has to be binded with the “CustomerName” property.

Hide   Copy Code

<input type=text id="txtCustomerName" ng-model="CustomerName"/>

Some of the most commonly used directives are ng-app,ng-controller and ng-repeat.

What are controllers and need of ng-controller and ng-model in Angular?

“Controllers” are simple javascript function which provides data and logic to HTML UI. As the name says controller they control how data flows from the server to HTML UI.

https://www.codeproject.com/KB/aspnet/891718/ajs.4.png

For example below is simple “Customer” controller which provides data via “CustomerName” and “CustomerCode” property and Add/ Update logic to save the data to database.

|  |
| --- |
| Note: - Do not worry too much about the $scope , we will discuss the same in the next question. |

Hide   Copy Code

function Customer($scope)

{

$scope.CustomerName = "Shiv";

$scope.CustomerCode = "1001";

$scope.Add = function () {

}

$scope.Update = function () {

}

}

“ng-controller” is a directive.Controllers are attached to the HTML UI by using the “ng-controller” directive tag and the properties of the controller are attached by using “ng-model” directive. For example below is a simple HTML UI which is attached to the “Customer” controller via the “ng-controller” directive and the properties are binded using “ng-model” directive.

Hide   Copy Code

<div ng-controller="Customer">

<input type=text id="CustomerName" ng-model="CustomerName"/><br />

<input type=text id="CustomerCode" ng-model="CustomerCode"/>

</div>

What are expressions in Angular?

Angular expressionsare unit of code which resolves to value. This code is written inside curly braces “{“.

Below are some examples of angular expressions:-

The below expression adds two constant values.

Hide   Copy Code

{{1+1}}

The below expression multiplies quantity and cost to get the total value.

Hide   Copy Code

The value total cost is {{ quantity \* cost }}

The below expression displays a controller scoped variable.

Hide   Copy Code

<div ng-controller="CustomerVM">

The value of Customer code is {{CustomerCode}}

</div>

The value of Customer code is {{CustomerCode}}

How can we initialize Angular application data?

We can use “ng-init” directive to achieve the same. You can see in the below example we have used “ng-init” directive to initialize the “pi” value.

Hide   Copy Code

<body ng-app="myApp" ng-init="pi=3.14">

The value of pi is {{pi}}

</body>

Explain $scope in Angular?

“$scope” is an object instance of a controller. “$scope” object instance get’s created when “ng-controller” directive is encountered.

For example in the below code snippet we have two controllers “Function1” and “Function2”. In both the controllers we have a “ControllerName” variable.

Hide   Copy Code

function Function1($scope)

{

$scope.ControllerName = "Function1";

}

function Function2($scope)

{

$scope.ControllerName = "Function2";

}

Now to attach the above controllers to HTML UI we need to use “ng-controller” directive. For instance you can see in the below code snippet how “ng-controller” directive attaches “function1” with “div1” tag and “function2” with “div2” tag.

Hide   Copy Code

<div id=”div1” ng-controller="Function1">

Instance of {{ControllerName}} created

</div>

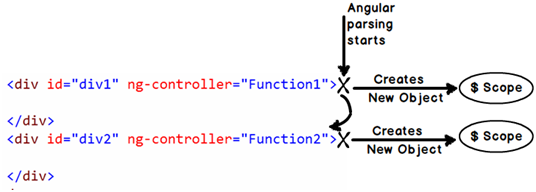
<div id=”div2” ng-controller="Function2">

Instance of {{ControllerName}} created

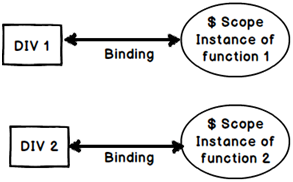
</div>

So this is what happens internally. Once the HTML DOM is created Angular parser starts running on the DOM and following are the sequence of events:-

* The parser first finds “ng-controller” directive which is pointing to “Function1”. He creates a new instance of “$scope” object and connects to the “div1” UI.
* The parser then starts moving ahead and encounters one more “ng-controller” directive which is pointing to “Function2”. He creates a new instance of “$scope” object and connects to the “div2” UI.

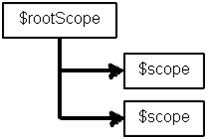


Now once the instances are created, below is a graphical representation of the same. So the “DIV1” HTML UI is binded with “function1” $scope instance and the “DIV2” HTML UI is binded with “function2” $scope instance. In other words now anything changes in the $scope object the UI will be updated and any change in the UI will update the respective $scope object.

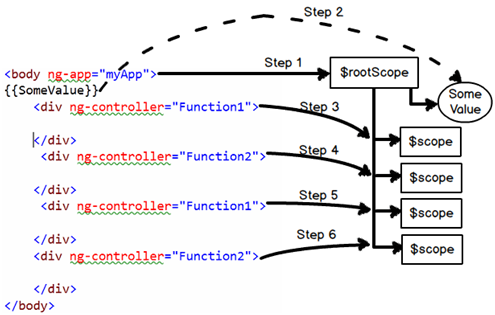


What is “$rootScope” and how is it related with “$scope”?

“$rootScope” is a parent object of all “$scope” angular objects created in a web page.



Let us understand how Angular does the same internally. Below is a simple Angular code which has multiple “DIV” tags and every tag is attached to a controller. So let us understand step by step how angular will parse this and how the “$rootScope” and “$scope” hierarchy is created.



The Browser first loads the above HTML page and creates a DOM (Document object model) and Angular runs over the DOM.Below are the steps how Angular creates the rootscope and scope objects.

* Step 1:- Angular parser first encounters the “ng-app” directive and creates a “$rootScope” object in memory.
* Step 2:- Angular parser moves ahead and finds the expression {{SomeValue}}. It creates a variable
* Step 3:- Parser then finds the first “DIV” tag with “ng-controller” directive which is pointing to “Function1” controller. Looking at the “ng-controller” directive it creates a “$scope” object instance for “Function1” controller. This object it then attaches to “$rootScope” object.
* Step 4:- Step 3 is then repeated by the parser every time it finds a “ng-controller” directive tag. Step 5 and Step 6 is the repetition of Step 3.

If you want to test the above fundamentals you can run the below sample Angular code. In the below sample code we have created controllers “Function1” and “Function2”. We have two counter variables one at the root scope level and other at the local controller level.

Hide   Copy Code

<script language="javascript">

function Function1($scope, $rootScope)

{

$rootScope.Counter = (($rootScope.Counter || 0) + 1);

$scope.Counter = $rootScope.Counter;

$scope.ControllerName = "Function1";

}

function Function2($scope, $rootScope)

{

$rootScope.Counter = (($rootScope.Counter || 0) + 1);

$scope.ControllerName = "Function2";

}

var app = angular.module("myApp", []); // creating a APP

app.controller("Function1", Function1); // Registering the VM

app.controller("Function2", Function2);

</script

Below is the HTML code for the same. You can we have attached “Function1” and “Function2” two times with “ng-controller” which means four instances will be created.

Hide   Copy Code

<body ng-app="myApp" id=1>

Global value is {{Counter}}<br />

<div ng-controller="Function1">

Child Instance of {{ControllerName}} created :- {{Counter}}

</div><br />

<div ng-controller="Function2">

Child Instance of {{ControllerName}} created :- {{Counter}}

</div><br />

<div ng-controller="Function1">

Child Instance of {{ControllerName}} created :- {{Counter}}

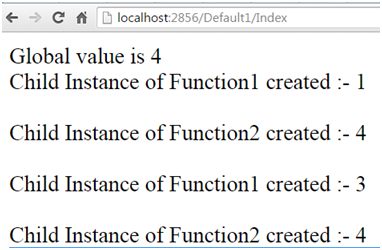
</div><br />

<div ng-controller="Function2">

Child Instance of {{ControllerName}} created :- {{Counter}}

</div><br />

</body>



Above is the output of the code you can see the global variable of root scope has be incremented four times because four instances of $scope have been created inside “$rootScope” object.

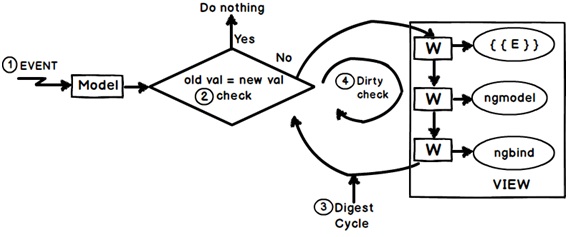
Explain the concept of digest cycle, watchers and dirty checking?

Angular is a MVW framework. It helps us to bind the model and the view. In other words when any change happens in the model the view gets updated. This updation of the model and the view is done by a loop called as digest cycle.

Digest cycle follows four important steps:-

1. Step 1:- Some kind of event is triggered by the end user like typing (onchange), button click etc and due to this activity model value changes.
2. Step 2:- Angular first checks if the new value and old values are same. If they are same he does not do anything. If they are not it then it invokes the digest cycle.
3. Step 3:- Digest cycle then runs through the scope objects to check which objects are getting affected because of this change. Every object in the scope have watchers. Watchers as the name says it listens whether the model has changed or not. Digest cycle informs the watchers about the model change and then watchers synchronize the view with the model data.
4. Step 4 :- In step 3 watchers update the view and due that update its very much possible that the model changes again. Now due to this model change we have to reevaulate the view again. So the digest loop runs once again to ensure that all things are synched up. This second loop which runs is termed as dirty check loop.

Below is the figure where in we have highlighted all the four steps.



So summarizing definitions for the above three concepts:-

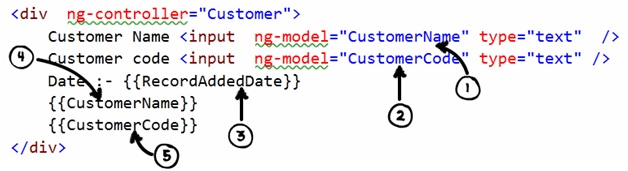
* Digest cycle: - It is a simple loop which updates the model and view.
* Watchers :- They are listeners which are attached to expression and angular directives and fire when the model data changes.
* Dirty check :- This is a extra digest loop which runs to check any cascading left over updates due to the first digest cycle.

What can be the performance implications of watchers and digest cycle ?

If there lot of unnecessary watchers then digest cycle has to work harder. As per AngularJS team having more than 2000 watchers on Angular screen is a bad practice.

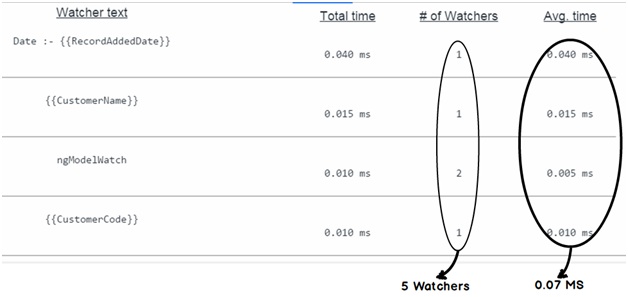
How can we measureno: of watchers & time spent on digest cycle?

Consider the below simple example where we have two ng-models and three expression. So in all we should have 5 watchers for the below screen



There are lot of great open source tools which help you to figure out the number of watchers , one such tool is the “batarang” tool. It’s a simple Google chrome extension which you can install separately.

Below is a simple snapshot where in we ran the above program , pressed f12 , enabled batarang and below are the results. You can see that he is showing 5 total watchers and for that digest cycle ran for 0.07 MS.



How can we decrease digest cycle time ?

To decrease digest cycle time you need to decrease the number of watchers. Below are some best practices you can follow to decrease number of watchers :-

* Remove unnecessary watchers.
* Use one time Angular binding. Especially if you see ng-repeat loop apply one time binding.
* Work in batches.
* Cache DOM
* Use Web worker

Can we force the digest cycle to run manually?

Yes , you can force it to run manually by calling the “$apply()” method.

Do I need Jquery for Angular?

No , you do not need Jquery for Angular. It’s independent of Jquery.

How is the data binding in Angular ?

Its two way binding. So whenever you make changes in one entity the other entity also gets updated.

Explain compile and link phase?

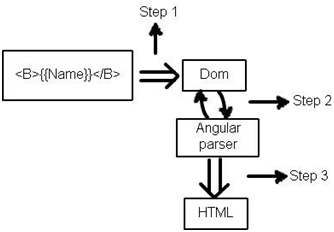
At the heart Angular framework is a parser. A parser which parses the Angular directives and render’s HTML output.

Angular parser works in 3 steps:-

Step 1:- HTML browser parses the HTML and creates a DOM (Document Object Model).

Step 2:- Angular framework runs over this DOM looks at the Angular directives and manipulates the DOM accordingly.

Step 3:- This manipulated is then rendered as HTML in the browser.

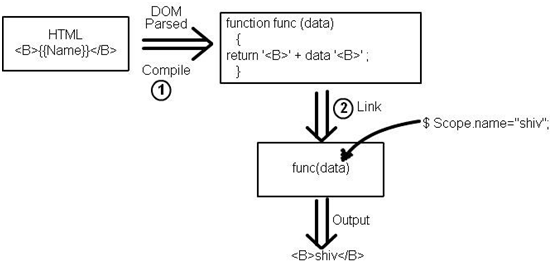


Now the above angular parsing is not so simple as it looks to be. It occurs in two phases “Compile” and “Link”. Firs the compile phase occurs then the link phase.

https://www.codeproject.com/KB/aspnet/891718/angular_parser_.2.png

In compile phase the angular parser starts parsing the DOM and whenever the parser encounters a directive it create a function. These functions are termed as template or compiled functions. In this phase we do not have access to the $scope data.

In the link phase the data i.e. ($scope) is attached to the template function and executed to get the final HTML output.



How do we make HTTP get and post calls in Angular?

To make HTTP calls we need to use the “$http” service of Angular. In order to use the http services you need to make provide the “$http” as a input in your function parameters as shown in the below code.

Hide   Copy Code

function CustomerController($scope,$http)

{

$scope.Add = function()

{

$http({ method: "GET", url: "http://localhost:8438/SomeMethod" }).success(function (data, status, headers, config)

{

// Here goes code after success

}

}

}

“$http” service API needs atleast three things:-

* First what is the kind of call “POST” or “GET”.
* Second the resource URL on which the action should happen.
* Third we need to define the “success” function which will be executed once we get the response from the server.

Hide   Copy Code

$http({ method: "GET", url: "http://localhost:8438/SomeMethod" }).success(function (data, status, headers, config)

{

// Here goes code after success

}

How do we pass data using HTTP POST in Angular ?

You need to pass data using the “data” keyword in the “$http” service API function. In the below code you can see we have created a javascript object “myData” with “CustomerName” property. This object is passed in the “$http” function using HTTP POST method.

Hide   Copy Code

Var myData = {};

myData.CustomerName = “Test”;

$http({ method: "POST",

data: myData,

url: "http://www.xyz.com"})

.success(function (data, status, headers, config)

{

// Here goes code after success

}

What is dependency injection and how does it work in Angular?

Dependency injection is a process where we inject the dependent objects rather than consumer creating the objects. DI is everywhere in Angular or we can go one step ahead and say Angular cannot work without DI.

For example in the below code “$scope” and “$http” objects are created and injected by the angular framework. The consumer i.e. “CustomerController” does not create these objects himself rather Angular injects these objects.

Hide   Copy Code

function CustomerController($scope,$http)

{

// your consumer would be using the scope and http objects

}

How does DI benefit in Angular?

There are two big benefits of DI: - Decoupling and Testing.

Let’s first start with Decoupling. Consider your application has a logger functionality which helps to log errors , warning etc in some central place. This central place can be a file, event viewer, database etc.

Hide   Copy Code

function FileLogger()

{

this.Log = function () {

alert("File logger");

};

}

function EventLogger()

{

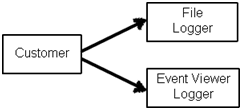
this.Log = function () {

alert("Event viewer logger");

};

}

Now let’s say you have a “Customer” class who wants to use the “Logger” classes. Now which “Logger” class to use depends on configuration.



So the code of “Customer” is something as shown below. So depending on the configuration “Customer” class either creates “FileLogger” or it creates “EventLogger” object.

Hide   Copy Code

function Customer($scope, Logger)

{

$scope.Logger = {};

if (config.Loggertype = "File")

{

$scope.Logger = new FileLogger();

}

else

{

$scope.Logger = new EventLogger();

}

}

But with DI our code becomes something as shown below. The “Customer” class says he is not worried from where the “Logger” object comes and which type of “Logger” objects are needed .He just wants to use the “Logger” object.

Hide   Copy Code

function Customer($scope,$http, Logger)

{

$scope.Logger = Logger;

}

With this approach when a new “Logger” object gets added the “Customer” class does not have to worry about the new changes because the dependent objects are injected by some other system.  
The second benefit of DI is testing. Let’s say you want to test the “Customer” class and you do not have internet connection. So your “$http” object method calls can throw errors. But now you can mock a fake “$http” object and run your customer class offline without errors.The fake object is injected using DI.

What are services in Angular?

Service helps to implement dependency injection. For instance let’s say we have the below “Customer” class who needs “Logger” object. Now “Logger” object can be of “FileLogger” type or “EventLogger” type.

Hide   Copy Code

function Customer($scope,$http, Logger)

{

$scope.Logger = Logger;

}

So you can use the “service” method of the application and tie up the “EventLogger” object with the “Logger” input parameter of the “Customer” class.

Hide   Copy Code

var app = angular.module("myApp", []); // creating a APP

app.controller("Customer", Customer); // Registering the VM

app.service("Logger", EventLogger); // Injects a global Event logger object

So when the controller object is created the “EventLogger” object is injected automatically in the controller class.

Are Service object instances global or local?

Angular Services create and inject global instances. For example below is a simple “HitCounter” class which has a “Hit” function and this function increments the variable count internally every time you call hit the button.

Hide   Copy Code

function HitCounter()

{

var i = 0;

this.Hit = function ()

{

i++;

alert(i);

};

}

This “HitCounter” class object is injected in “MyClass” class as shown in the below code.

Hide   Copy Code

function MyClass($scope, HitCounter)

{

$scope.HitCounter = HitCounter;

}

Below code advises the Angular framework to inject “HitCounter” class instance in the “MyClass” class. Read the last line of the below code specially which says to inject the inject the “HitCounter” instance.

Hide   Copy Code

var app = angular.module("myApp", []); // creating a APP

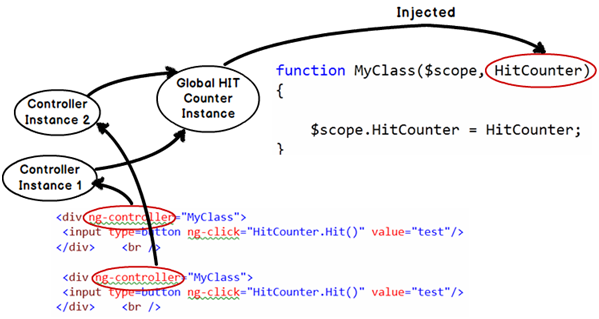
app.controller("MyClass", MyClass); // Registering the VM

app.service("HitCounter", HitCounter); // Injects the object

Now let’s say that the “Controller” “MyClass” is attached to twodiv tag’s as shown in the below figure.

So two instances of “MyClass” will be created. When the first instance of “MyClass” is created a “HitCounter” object instance is created and injected in to “MyClass” first instance.

When the second instance of “MyClass” is created the same “HitCounter” object instance is injected in to second instance of “MyClass”.  
Again I repeat the same instance is injected in to the second instance, new instances are not created.

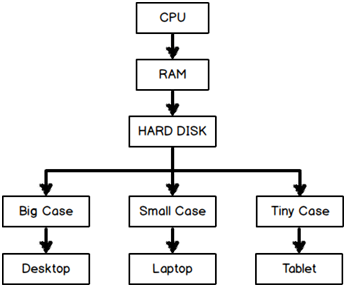


If you execute the above code you will see counter values getting incremented even if you are coming through different controller instances.

What is a Factory in Angular?

“Factory” in real world means a premise where products are manufactured. Let’s take an example of a computer manufacturing firm. Now the company produces different kinds and sizes of computers likelaptops,desktops, tablets etc.

Now the process of manufacturing the computer products are same with slight variation. To manufacture any computer we need processor, RAM and hard disk. But depending on what kind of final case packing is the final product shapes.



That’s what the use of Factory in Angular.

For example see the below code we have a “Customer”, “Phone” and “Address” class.

Hide   Copy Code

function Customer()

{

this.CustomerCode = "1001";

this.CustomerName = "Shiv";

}

function Phone()

{

this.PhoneNumber = "";

}

function Address()

{

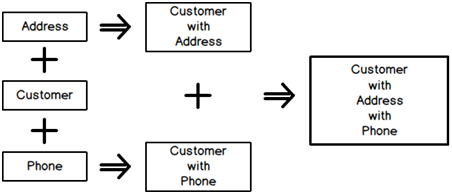
this.Address1 = "";

this.Address2 = "";

}

So now we would create different types of “Customer” object types using the combination of “Address” and “Phones” object.

* We would like to combine “Customer” with “Address” and create a “Customer” object which has “Address” collection inside it.
* Or must be we would like to create “Customer” object with “Phone” objects inside it.
* Or must be “Customer” object with both “Phone” and “Address” objects.



In other words we would like to have different permutation and combination to create different types of “Customer” objects.

So let’s start from bottom. Let’s create two factory function’s one which creates “Address” object and the other which creates “Phone” objects.

Hide   Copy Code

functionCreateAddress()

{

var add = new Address();

return add;

}

functionCreatePhone()

{

var phone = new Phone();

return phone;

}

Now let’s create a main factory function which uses the above two small factory functions and gives us all the necessary permutation and combination.

In the below factory you can see we have three functions:-

* “CreateWithAddress” which creates “Customer” with “Address” objects inside it.
* “CreateWithPhone” which creates “Customer” object with “Phone” objects inside it.
* “CreateWithPhoneAddress” which creates “Customer” object with aggregated “Phone” and “Address” objects.

Hide   Copy Code

function CreateCustomer() {

return {

CreateWithAddress: function () {

varcust = new Customer();

cust.Address = CreateAddress();

returncust;

},

CreateWithPhone: function () {

varcust = new Customer();

cust.Phone = {};

cust.Phone = CreatePhone();

returncust;

}

,

CreateWithPhoneAddress: function () {

debugger;

varcust = new Customer();

cust.Phone = CreatePhone();

cust.Address = CreateAddress();

returncust;

}

}

}

Below is a simple “CustomerController” which takes “CustomerFactory” as the input. Depending on “TypeOfCustomer” it creates with “Address” , “Phones” or both of them.

Hide   Copy Code

functionCustomerController($scope, Customerfactory)

{

$scope.Customer = {};

$scope.Init = function(TypeofCustomer)

{

if (TypeofCustomer == "1")

{

$scope.Customer = Customerfactory.CreateWithAddress();

}

if (TypeofCustomer == "2")

{

$scope.Customer = Customerfactory.CreateWithPhone();

}

if (TypeofCustomer == "3") {

$scope.Customer = Customerfactory.CreateWithPhoneAddress();

}

}

}

You also need to tell Angular that the “CreateCustomer” method needs to be passed in the input. For that we need to call the “Factory” method and map the “CreateCustomer” method with the input parameter “CustomerFactory” for dependency injection.

Hide   Copy Code

var app = angular.module("myApp", []); // creating a APP

app.controller("CustomerController", CustomerController); // Register the VM

app.factory("Customerfactory", CreateCustomer);

So if we consume the “CustomerController” in UI , depending on situation it creates different flavors of “Customer” object. You can in the below code we have three different “DIV” tags and depending on the “TypeofCustomer” we are displaying data.



What is the difference between Factory and Service?

“Factory” and “Service” are different ways of doing DI (Dependency injection) in angular. Please read the previous question to understand what is DI.

So when we define DI using “service” as shown in the code below. This creates a new GLOBAL instance of the “Logger” object and injects it in to the function.

Hide   Copy Code

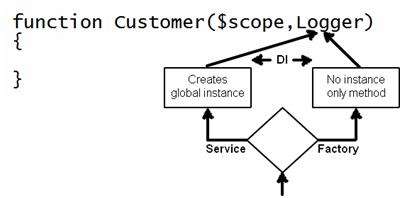
app.service("Logger", Logger); // Injects a global object

When you define DI using a “factory” it does not create a instance. It just passes the method and later the consumer internally has to make calls to the factory for object instances.

Hide   Copy Code

app.factory("Customerfactory", CreateCustomer);

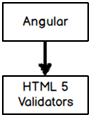
Below is a simple image which shows visually how DI process for “Service” is different than “Factory”.



|  |  |  |
| --- | --- | --- |
|  | **Factory** | **Service** |
| Usage | When we want to create different types of objects depending on scenarios. For example depending on scenario we want to create a simple “Customer” object , or “Customer” with “Address” object or “Customer” with “Phone” object. See the previous question for more detailed understanding. | When we have utility or shared functions to be injected like Utility , Logger , Error handler etc. |
| Instance | No Instance created. A method pointer is passed. | Global and Shared instance is created. |

How are validations implemented in Angular?

Angular leverages HTML 5 validations and new form element types to implement validation.



For instance below is a simple form which has two text boxes. We have used HTML 5 “required” validation attribute and a form element of type “email”.

Hide   Copy Code

<form name="frm1" id="frm1" >

Name :- <input type=text name="CustomerName" id="CustomerName" required /> Email :- <input type=email name="Email" id="Email" />

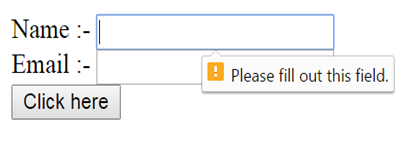
<input type=submit value="Click here"/>

</form>

Below are some example of new form elements introduced in HTML 5 and Angular works with almost all of them :-

* Color.
* Date
* Datetime-local
* Email
* Time
* Url
* Range
* Telephone
* Number
* Search

When you run the above HTML inside a browser which understands HTML 5 , you will see your validations and form types in actions as shown in the below browser screen shot.



Angular leverages HTML 5 validation attributes and new HTML 5 form elements. Now if we want Angular to handle validation we need first stop HTML 5 to do validation. So for that the first step is to specify “novalidate” attribute on the form tag.

Hide   Copy Code

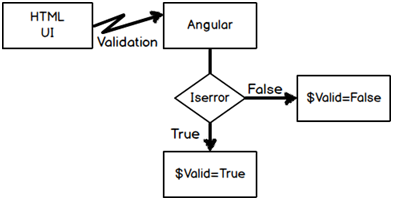
<form name="frm1" novalidate>

-----

</form>

So now the HTML will not fire those validations it will be routed to the Angular engine to further take actions.

In other words when end user fills data in the HTML UI , validation events are routed to Angular framework and depending on scenario Angular sets a field called as “$Valid”. So if the validations are fine it sets it to “True” or else its sets it to “False”.



So you can see in the below code we have attached the angular controller and models to the text boxes. Watch the code of the button it has “ng-disabled” attribute which is set via the “$Valid” property in a NEGATED fashion.

Negated fashion means when there is no error it should enable the button and when there are errors that means it’s false it should disable the button.

Hide   Copy Code

<form name="frm1" novalidate>

Name:-<input type=text ng-model="Customer.CustomerName" name="CustomerName" required />

Email :- <input type=email ng-model="Customer.Email" name="Email" />

<input type=submit value="Click here" ng-disabled="!(frm1.$valid)"/>

</form>

*Note :- “Name” is needed for the validations to work.*

How to check error validation for a specific field?

To check for a specific field you need to use the below DOM code.

Hide   Copy Code

!frm1.CustomerName.$valid

What does SPA (Single page application) mean?

SPA is a concept where rather loading pages from the server by doing post backs we create a single shell page or master page and load the webpages inside that master page.

How can we implement SPA with Angular?

By using Angular routes.

How to implement routing in Angular?

Implementing Angular route is a five step process: -

Step 1: - Add the “Angular-route.js” file to your view.

Hide   Copy Code

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

Step 2: - Inject “ngroute” functionality while creating Angular app object.

Hide   Copy Code

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

Step 3: - Configure the route provider.

In route provider we need to define which URL pattern will load which view. For instance in the below code we are saying “Home” loads “Yoursite/Home” view and “Search” loads “YourSite/Search” view.

Hide   Copy Code

app.config(['$routeProvider',

function ($routeProvider) {;

$routeProvider.

when('/Home, {

templateUrl: 'Yoursite/Home',

controller: 'HomeController'

}).

when('/Search', {

templateUrl: YourSite/Search',

controller: 'SearchController'

}).

otherwise({

redirectTo: '/'

});

}]);

Step 4: - Define hyperlinks.

Define hyper link with the “#” structure as shown below. So now when user clicks on the below anchor hyperlinks, these actions are forwarded to route provider and router provider loads the view accordingly.

Hide   Copy Code

<div>

<a href="#/Home">Home</a><br />

<a href="#/Search"> Search </a><br />

</div>

Step 5: - Define sections where to load the view.

Once the action comes to the router provider it needs a place holder to load views. That’s defined by using the “ng-view” tag on a HTML element. You can see in the below code we have created a “DIV” tag with a place holder. So the view will load in this section.

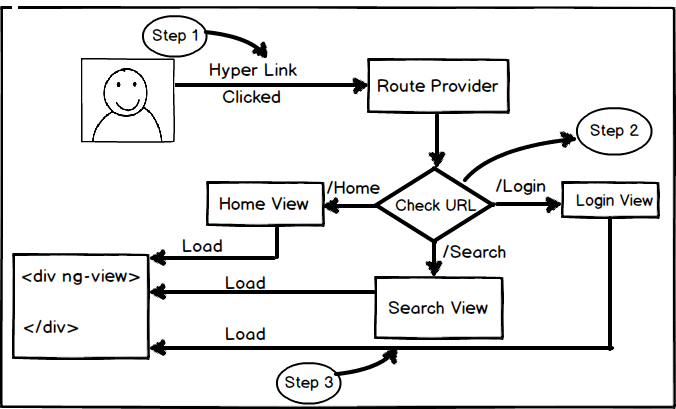
Hide   Copy Code

<div ng-view>

</div>

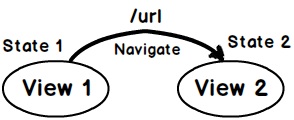
So if we summarize angular routing is a three step process (Below is a visual diagram for the same): -

* Step 1: - End user clicks on a hyperlink or button and generates action.
* Step 2: - This action is routed to the route provider.
* Step 3: - Router provider scans the URL and loads the view in the place holder defined by “ng-view” attribute.



How to implement SPA using angular-UI route?

Angular UI route helps to implement SPA concept using the concept of STATES. The main goal of SPA is navigating from one view to other view without reloading the main page. Angular UI route visualizes every view as a STATE. When you want to navigate from one view to other view you can either use the STATE names or use URL.



So let’s say we want to navigate from “Home.htm” view to About.htm” view so we can define two states “Home” and “About” and link them to the respective HTML page as shown below.

You can also specify URL by which you can move between these states by using “url” property as shown in the below code.

Hide   Copy Code

myApp.config(function ($stateProvider, $urlRouterProvider) {

$stateProvider

.state('Home', {

url: '/HomePage',

templateUrl: 'Home.htm'

})

.state('About', {

url: '/About',

templateUrl: 'About.htm'

})};

Now once the states are defined to we need to use “ui-sref” and if you want to navigate using url provide “url” value in the “href” of the anchor tag.

We also need to provide "<ui-view>" tag to define in which location we want to load the views.

Hide   Copy Code

<a ui-sref="About" href="#About">Home</a>

<a href="#Home">About</a>

<ui-view></ui-view>

Below is the complete code if HTML , please ensure you have also referenced of “Angular-UI” js file. You can also see “App.js” file , this file has code which defines the states.

Hide   Copy Code

<script src="Scripts/angular.js" type="text/javascript"></script>

<script src="Scripts/angular-ui-router.js" type="text/javascript"></script>

<script src="Scripts/App.js" type="text/javascript"></script>

<body ng-app="myApp">

<a ui-sref="About" href="#About">Home</a>

<a href="#Home">About</a>

<ui-view></ui-view>

</body>

</html>

Can we load HTML content rather than a full page ?

Yes, you can load simple HTML content by using “template” property as shown in the highlighted code below.

Hide   Copy Code

myApp.config(function ($stateProvider, $urlRouterProvider) {

$stateProvider

.state('About', {

url: '/About',

template: '<b>This is About us</b>'

})};

How can we create controllers and pass parameters in Angular UI route?

To create a controller we need to use “controller” property of the state provider. To specify parameters you can put the parameter name after the url. In the below code you can see ‘Id’ parameter after the url and also you can see how validations are applied on these parameters using regex.

Hide   Copy Code

myApp.config(function ($stateProvider, $urlRouterProvider) {

$stateProvider

.state('State1', {

url: '/SomeURL/{Id:[0-9]{4,4}}',

template: '<b>asdsd</b>',

controller: function ($scope, $stateParams) {

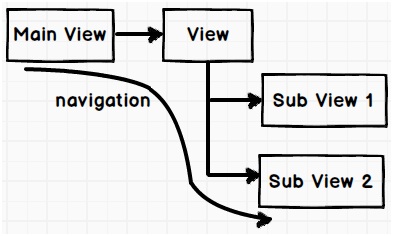
alert($stateParams.Id);

}

});

How to implement nested views using Angular UI route?

First let us understand the concept of nested views. We want to navigate as follows in SPA. From main view we want to navigate to some view and in that view we want to load some other view.



Angular UI Router helps to define nested states. Below is the code of “MainView” in which we have defined one more state “View” and in that we have two child states “View.SubView1” and “View.SubView2” which points to different views.

Hide   Copy Code

myApp.config(function ($stateProvider, $urlRouterProvider) {

$stateProvider

.state(“View”, {

templateUrl: 'View.htm'

})

.state('View.SubView1', {

template: '<b>Sub view 1</b>'

}).state('View.SubView2', {

template: '<b>Sub view 2</b>'

});

});

In the parte view we can now define navigation to child states i.e. “View.SubView1” and “View.SubView2”.

Hide   Copy Code

<a ui-sref="View.SubView1" href="#View.SubView1">Sub view 1</a>

<a ui-sref="View.SubView2" href="#View.SubView1 ">Sub view 2</a>

<div ui-view></div>

How can we create a custom directive in Angular?

Till now we have looked in to predefined Angular directives like “ng-controller”,”ng-model” and so on. But what if we want to create our own custom Angular directive and attach it with HTML elements as shown in the below code.

Hide   Copy Code

<div id=footercompany-copy-right></div>

To create a custom directive we need to use the “directive” function to register the directive with angular application. When we call the “register” method of “directive” we need to specify the function which will provide the logic for that directive.

For example in the below code we have created a copy right directive and it returns a copy right text.

*Please note “app” is an angular application object which has been explained in the previous sections.*

Hide   Copy Code

app.directive(*'companyCopyRight', function ()*

{

return

{

template: *'@CopyRight questpond.com '*

};

});

The above custom directive can be later used in elements as shown in below code.

Hide   Copy Code

<div ng-controller="CustomerViewModel">

<div company-copy-right></div>

</div>

What kind of naming conventions is used for custom directives?

For angular custom directive the best practice is to follow camel casing and that also with atleast two letter’s. In camel case naming convention we start with a small letter, followed by a capital letter for every word.

Some example of camel cases are “loopCounter” , “isValid” and so on.

So when you register a custom directive it should be with camel case format as shown in the below code “companyCopyRight”.

Hide   Copy Code

app.directive(*'companyCopyRight', function ()*

{

return

{

template: *'@CopyRight questpond.com '*

};

});

Later when this directive is consumed inside HTML before each capital letter of camel case we need to insert a “-“ as specified in the below code.

Hide   Copy Code

<div company-copy-right></div>



If you are making a one letter prefix like “copyright” it’s very much possible that tomorrow if HTML team creates a tag with the same name, it will clash with your custom directive. That’s why angular team recommends camel case which inserts a “-“ in between to avoid further collision with future HTML tag’s.

What are the different custom directive types in AngularJS?

There are different flavors of Angular directives depending till what level you want to restrict your custom directive.

In other words do you want your custom directive to be applied only on HTML element or only on an attribute or just to CSS etc.

So in all there are four different kinds of custom directives:-

* Element directives (E)
* Attribute directives (A)
* CSS class directives (C)
* Comment directives (M)

Below is a simple custom directive implementation at the element level.

Hide   Copy Code

myapp.directive(*'userinfo', function()*

{

var directive = {};

directive.restrict = *'E';*

directive.template = "User : {{user.firstName}} {{user.lastName}}";

return directie;

});

The “restrict” property is set to “E” which means that this directive can only be used at element level as shown in the code snippet below.

Hide   Copy Code

<userinfo></userinfo>

If you try to use it at an attribute level as shown in the below code it will not work.

Hide   Copy Code

<div userinfo></div>

So “E” for element, “A” for attribute, “C” for CSS and “M” for comments.

What if I want custom directives to be applied on element as well as attributes ?

Hide   Copy Code

directive.restrict = *'EA';*

Can I set an Angular directive template to a HTML web page?

Yes, you can set template to page directly by using “templateUrl” property of the directive as shown in the code snippet below.

Hide   Copy Code

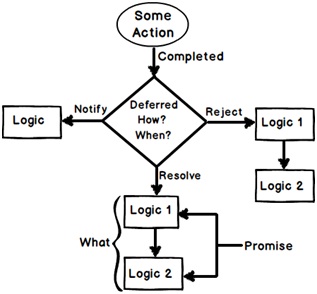
directive.templateUrl = "/templates/footer.html";

Explain $q service, deferred and promises?

Promises are POST PROCESSING LOGICS which you want to execute after some operation / action is completed. While deferred helps to control how and when those promise logics will execute.

We can think about promises as “WHAT” we want to fire after an operation is completed while deferred controls “WHEN” and “HOW” those promises will execute.

For example after an operation is complete you want to a send a mail, log in to log file and so on. So these operations you will define using promise. And these promise logics will be controlled by deferred.



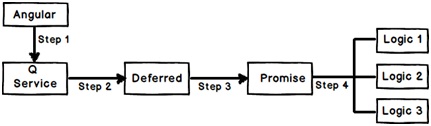
We are thankful to [www.stepbystepschools.net](http://www.stepbystepschools.net/) for the above image.

So once some action completes deferred gives a signal “Resolve”, “Reject” or “Notify” and depending on what kind of signal is sent the appropriate promise logic chain fires.

“$q” is the angular service which provides promises and deferred functionality.

Using promises, deferred and “q” service is a 4 step process:-

* Step 1:- Get the “q” service injected from Angular.
* Step 2 :- Get deferred object from “q” service object.
* Step 3 :- Get Promise object from deferred object.
* Step 4 :- Add logics to the promise object.



Below is the angular code for the above 4 steps.

Hide   Copy Code

*// Step 1 :- Get the "q" service*

function SomeClass($scope,$q) {

*// Step 2 :- get deferred from "q" service*

var defer = $q.defer();

*// step 3:- get promise from defer*

var promise = defer.promise;

*// step 4 :- add success and failure logics to promise object*

promise.then(function () {

alert("Logic1 success");

}, function () {

alert("Logic 1 failure");

});

promise.then(function () {

alert("Logic 2 success");

}, function () {

alert("Logic 2 failure");

});

}

So now depending on situations you can signal your promise logics via deferred to either fire the success events or the failure events.

Hide   Copy Code

*// This will execute success logics of promise*

defer.resolve();

Hide   Copy Code

*// This will execute failure logics of promise*

defer.reject();