ANGULAR JS

1.What is AngularJs ?
AngularJs is an MVW framework,that is,it can be either an MVC framework or MVVM framework.If the model and view are bound by the controller,then it is said to be an MVC framework.If the model and view are bound by the view model,then it is said to be an MVVM framework.

2.Why AngularJs ?
In addition to testing,AngularJs takes code quality seriously.They do this into two ways.First,AngularJs adheres to ES5's "strict mode" which restricts you to a cleaner subset of Javascript than what is normally available in the browser.Second,AngularJs makes use of JsHint for static analysis.JsHint ensures that code is consistent in its styling, and that many simple bugs you might find in projects not using JsHint have already been caught and cleaned up before they are committed.
Angular is TDD(Test Driven Development) that is we can write tests for all parts of our application - that is for model layer,service layer,view layer,etc.

3.Does AngularJs works in all the browsers ?
AngularJs works with IE8 and above.
Inorder to make AngularJs work with IE7 and earlier,we need to polyfill JSON.stringify.We can use JSON3 or JSON2 implementations.

4.What is an MVC framework ?
MVC architectural pattern divides an application into three distinct,modular parts:
Model - is a dynamic data,which you can get from a database like MySQL or from a static JSON file.
View - is display of model,that is,it displays the data.
Controller - gives control over model and view,that is,it controls how data is retrieved and displayed to the end-user.

5.What is a Single Page Application ?
In a Single Page Application,either all necessary code (that is,HTML,CSS and JavaScript)is retrieved with a single page load or the appropriate resources are dynamically loaded and added to the page as necessary usually in response to user actions.
In a Single Page Application,the page does not reload at any point in the process nor does control transfer to another page,although modern web technologies(such as HTML5) can provide the perception and navigability of separate logical pages in the application.
Interaction with the Single Page Application often involves dynamic communication with the web server behind the scenes.

6.Why do we prefer Single Page Application over the Older Application ?
In Older Application,the server used to manage the state of an application and there were disconnections between data the user was seeing and data synchronized on the server.
In Angular,browser manages the state of an application, and communicates the changes we want via Ajax using GET,POST,PUT and DELETE methods typically talking to a REST endpoint backend.
REST endpoints can be front-end independent,and the front-end can be back-end independent.We can serve the same endpoints to a mobile application which may have a different front-end to a web application.

7.What is a Module ?
In Angular,every Application is created using modules.A module can have dependencies of other modules or be a single module all by itself(that is, a module can have other modules as dependencies).The modules act as containers for different sections of an application,making code more reusable and testable.A module is considered as a

container for different parts of the application such as controllers,directives,services,filters,etc.
Modules are more modular,that is,Modules are easier to use in multiple applications,Modules are fully configurable,Modules are progressive enhancement.
angular.module (‘myApp’, [ ]) - is called as setter method - is used to create a moduleThe first parameter is the name of the module.
The second parameter is the list of dependencies or injectables.
Injector loads the list of dependencies before the module itself is loaded.
angular.module(‘myApp’) - is called as getter method - is used to reference our module
An application can contain one or more than one module.

8.What is two way data-binding ?
Two way data-binding provides synchronization between Model layer and View layer,that is,Model (Back-end)changes propagate to the View(Front-end) and View changes are instantly reflected back to the Model.This makes the Model the “ single source of truth” for maintaining the applications’ state.
To do the same two-way data-binding in JQuery,we need to write a logic in both of the layers,that is,Model and View.But in case of AngularJs,we have to bind the variable using $scope and Angular handles the rest.
ngModel directive provides the two-way data-binding by synchronizing the model to the view,as well as view to the model.

9.What is Dependency Injection ?
Dependency Injection is a software design pattern in which components are given their dependencies instead of hard coding them within the component.This relieves the component from locating the dependency and makes dependencies configurable.This helps in making components reusable,maintainable and testable.Angular provides some components which can be injected into each other as dependencies.The components are:
value - A value is a Javascript object ,and is used to pass values to controller during config phase.
factory -
service
provider
constant
We can use strict dependency injection by adding "ng-strict-di” directive on the same element as “ng-app”.If we use strict dependency injection then an error will be thrown,whenever a service tries to use “implicit annotations”.
Generally,dependency annotation is of three types:
(i) Inline array annotation - is preferred
(ii) $injector property annotation
(iii) Implicit annotation

10.Which type of components can be injected into .config( ) function ?
provider( ) method is one of the components.
provider( ) method can be injected into .config( ) function.

11.Which type of components can be injected into .run( ) function ?
service( ) method is one of the components.
service( ) method can be injected into .run( ) function.

12.What is an injector ?
The injector is a “service locator” that is responsible for construction and lookup of dependencies.

13.How do you define a parent scope and child scope ?
Parent scope is the root scope for all the descending child scopes.we use $rootScope for defining a parent scope and $scope for defining child scopes.
Instead of placing variables on the $rootScope,we can explicitly create a child $scope object using a controller.This controller can be attached to a DOM element using “ng-controller”.The “ng-controller” creates a new $scope object

for the DOM element and nests it in the containing $rootScope.

14.What is $rootScope ?
$rootScope is the top level $scope object from which all further scopes are created.Once Angular starts rendering your application,Angular creates a $rootScope object,and all further bindings and logic in your application create new $scope objects which then all become children of the $rootScope.

15.How do you know when is the $rootscope starting ?
whenever NgApp is mentioned from there $rootscope begins

16.What is $scope ?
$scope creates a binding between Javascript and DOM.We use $scope inside controllers,where we bind data from the controller to the view.
Scopes hold your Models (that is your data) ,Scopes cooperate with your Controllers,Scopes give the Views everything they need (that is what the user sees and interacts with)

17.What is the $scope Lifecycle ?
Creation : When we create a controller or directive,Angular creates a new scope with the $injector and passes this new scope for the controller or directive at runtime.
Linking : When the scope is linked to the view,all directives that create $scopes will register their watches on the parent scope.These watches watch for and propagate model changes from the view to the directive.
Updating : During the $digest cycle,which executes on the $rootScope,all of the children scopes will perform "dirty digest checking". All of the watching expressions are checked for any changes,and the scope calls the listener callback when they are changed.
Destructing : When a $scope is no longer needed,the child scope creator will need to call scope.$destroy( ) to clean up the child scope.Note that when a scope is destroyed,the $destroy event will be broadcasted.
Directives do not generally create their own $scopes,but ng-controller and ng-repeat directives create their own child scopes and attach them to the DOM element.

18.What can a $scope do ?
$scope provides “observers” to watch for model changes and propagates these model changes throughout the application as well as outside the system to other components.Scopes can be nested such that they can isolate functionality and model properties.

19.What is an Isolated scope ?
An Isolated scope is a scope that does not inherit from the parent and exist on its own.
Isolate scopes do not prototypically inherit from the parent scope, so it does not have access to any of the parent scope's properties (unless '@' or '=' or '&' are used). Using $parent is a way to still access the parent scope, but not via prototypical inheritance. Angular creates this special $parent property on scopes. Normally (i.e. best practice), it should not be used.
A scope created inside of a directive is called the isolate scope.With the exception of isolate scopes, all scopes are created with prototypal inheritance, meaning that they have access to their parent scopes. this behavior should look familiar.

When should we use Isolated scopes ?
To make reusable directives,we cannot rely on the parent scope and we must use the Isolated scopes.

20.Why do we need Isolated scope ?
In AngularJs by default directives get the parent controller’s scope.This is useful when we are exposing the parent controller’s scope to directives.But in some cases your directives may want to add several properties and functions

to the parent controller's scope.If we are doing this then we are populating parent controller's scope itself.So,to overcome this problem we can make use of Isolated scope directives.

21.How to create an Isolated scope ?
An Isolated scope can be created by setting the scope property to an empty object hash { } .

22.When Isolated scopes are preferred ?
Isolated scopes are preferred when we want to create reusable components.

23.How does Isolate scope communicate with Isolate directive ?
Isolate scope provides Local scope properties to communicate with Isolate directive.The Local scope properties are:
(i) with an attribute : using @
(ii)with a binding : using =
(iii)with an expression : using &

24.What is @ binding ?
@ binding is used for passing strings.These strings support {{ }} expressions for interpolated values.
@ binding is also called as Text binding or one-way binding.

25.What is = binding ?
= binding is used for two-way model binding.The model in parent scope is linked to the model in directive's isolated scope.Changes to one model affects the other,and vice versa.
= binding is also called as Direct model binding or two-way model binding.

26.What is & binding ?
& binding is used for passing a method into your directive's scope so that the method can be called within your directive.
& binding is also called as Behaviour binding or Method binding.

27.What is NgApp ?
Placing NgApp on any DOM element marks that element as the beginning of the $rootScope.
$rootScope is the beginning of the scope chain,and all directives nested under the ng-app in the HTML inherit from it.
NgApp creates the $rootScope of an Angular application,while NgController creates a child scope that prototypically inherits from either $rootScope or another NgController's $scope.

During AngularJs App bootstrapping,the following three things will happen :
The injector that will be used for dependency injection is created.
The injector will then create the "root scope" that will become the context for the model of our application.
Angular will then "compile" the DOM starting at the "NgApp" root element,processing any directives and bindings found along the way.

Once an application is bootstrapped,it will then wait for incoming browser events (such as mouse click,key press,or incoming HTTP response) that might change the model.Once such an event occurs,Angular detects if it caused any model changes and if changes are found,Angular will reflect them in the view by updating all of the affected bindings.

28.How many NgApp's can contain in one application ?
NgApp is used to auto bootstrap an AngularJs application.
NgApp can be placed on <body> tag or <html> tag.

In one HTML document,only one AngularJs application can be auto-bootstraped.
To run multiple AngularJs applications in one HTML document,we must manually bootstrap them using "angular.bootstrap"

29.What is Bindonce ?
Bindonce directive creates a single temporary watcher that is removed after the data becomes available.If the data is already available on the scope,then the watcher isn't created and the children are rendered.
To use Bindonce,we need to follow the following steps:
(i) bower install angular-bindonce
(ii) <script src = " scripts / vendor / bindonce.js > </script>
(iii) angular.module ('myApp', [pasvaz.bindonce']);

30.What are expressions in Angular ?
The { { } } notation for showing a variable attached to $scope is actually an expression.The expressions have the following properties:
All expressions are executed in the context of the scope and have access to local $scope variables.
An expression doesn't throw errors if it results in a TypeError or a ReferenceError.
An expression doesn't allow control flow functions such as "if else".
An expression can accept a filter or filter chains.

Angular evaluates an expression by an internal service called "$parse" service that has knowledge of the current scope.
In Javascript, we use eval( ) function to evaluate expressions but using eval( ) function for evaluating expressions arises security concerns.
Angular can evaluate an expression in three levels.They are:
$parse - Evaluates only single expressions.
$interpolate - Evaluates strings containing multiple expressions.$interpolate in turn calls $parse for evaluation.
$compile - It can turn html string with directives,interpolation expressions to live DOM.

31.What is $digest( ) method in Angular ?
$digest( ) method processes all of the watchers of the current scope and its children.
As a watcher's listener can change the model,the digest( ) method keeps calling the watchers until no more listeners are firing,that is,it is possible to get into an infinite loop.
digest( ) method throws "maximum iteration limit exceeded", if the number of iterations exceeds 10.
Usually,we don't call $digest( ) method directly in controllers or in directives,instead we call $apply( ) (in a directive) which will force $digest( ) method.
If we want to be notified whenever $digest( ) is called,we can register a "watchExpression" function with $watch( ) with no listener.

32.What is the difference between $watch and $digest?
We can write $watches and $digest will compile these $watches to observe the objects that are passed to $watch.
$digest is a lifecycle which will run or compile these $watches that we are setting onto the objects.

33.Is it good to have more $watches ?
No,if we have more $watches it will kill the performance.

34.What is the circumstance where we cannot $watch ?
The user is logged into an application and we want to track the user is changing any of his important credentials like Id,Password.If we give the access to the user that can be changed that is very important to be immediately updated.

35.What is $emit( ) method in Angular ?

$emit( ) method dispatches an event name upwards through the scope hierarchy notifying the registered $rootScope.Scope listeners.
$emit (name, args)
name - is the event name to emit
args - one or more arguments which will be passed onto the event listeners

36.What is $broadcast( ) method in Angular ?
broadcast( ) method dispatches an event name downwards to all child scopes notifying the registered $rootScope.Scope listeners.
$broadcast(name, args)
name - is the event name to emit
args - one or more arguments which will be passed onto the event listeners

37.What is $on( ) method in Angular ?
$on( ) method listens on events of a given type.
$on(name, listener)
name - is the event name to listen on
listener - function to call when the event is emitted.

38.What are the advantages of $apply( ) method over $digest( ) method ?
$digest( ) method is faster than $apply( ) method,since $apply( ) triggers watchers on the entire scope chain while $digest( ) triggers watchers on the current scope and its children (if it has)
Incase of $digest( ),when an error occurs in one of the watchers,$digest( ) can't handle the errors via "$exceptionHandler service",we have to handle the errors on our own.while incase of $apply( ) ,try catch block is present internally and if any error occurs in one of the watchers then it passes the errors to "$exceptionHandler service".

39.What is the difference between $timeout(callback) and $evalAsync(callback) ?
$timeout(callback) will wait for the current digest cycle to be done(that is,angular updates all model and the DOM),then it will execute its callback.
$evalAsync(callback) will add the callback to the current or next digest cycle,that is,if we are within a digest cycle(for instance in a function called from some ng-click directive) it will not wait for anything,and the code will be executed right away.If we are within an asynchronous call (for instance a setTimeout) it triggers a new digest cycle,that is,in this case $evalAsync(callback) acts as $timeout(callback).

40.What is $watch( ) ?
$watch( ) method on the $scope object sets up a "dirty check" on every call to "$digest" inside the Angular event loop.The $digest loop always returns,if it detects changes on the "expression".
$watch(watchExpression,listener,[objectEquality])
"watchExpression" is called on every call to $digest( ) and returns the value that will be watched.
"watchExpression" shouldn't change its value when executed multiple times with the same input,because it may be executed multiple times by $digest( ).
"listener" is called only when the value from the current "watchExpression" and the previous call to "watchExpression" are not equal.
when "objectEquality = = true" , inequality of the "watchExpression" is determined according to the "angular.equals" function.

$watchCollection(obj, listener)
$watchcollection allows us to set shallow watches for object properties or elements of an array and fire the "listener callback" whenever the properties change.
"obj" is observed via standard $watch operation and is examined on every call to $digest( ) to see if any items have been added,removed or moved.

“listener” is called whenever anything within “obj” has changed.

$watchGroup(watchExpression, listener) - It watches an array of “watchExpressions”.

$watch( ) returns a "deregistration function” for the “listener”.The “deregistration function” can be called to cancel a watch on the value.
We should never use $watch( ) in a controller because it makes difficult to test the controller.

41.What is the lifecycle of $apply( ) method in Angular ?
$apply( ) method is used to execute an expression in angular from outside of the angular framework,that is,from browser DOM events,setTimeout,XHR or third party libraries.

```
function $apply(expression) {
 try {
       return $eval (expression) ;
     }
 catch (expression) {
      $exceptionalHandler (expression) ;
     }
 finally {
    $root.$digest( ) ;
     }
}
```

$apply( ) method goes through the following stages :
(i) The expression is executed using the $eval( ) method.
(ii) Any exceptions from the execution of the expression are forwarded to the $exceptionHandler service.
(iii) The watch listeners are fired immediately after the expression was executed using the $digest( ) method.
expression - is an angular expression to be executed.
$applyAsync ([expression]) - schedules the invocation of $apply to occur at a later time.This can be used to queue up multiple expressions which need to be evaluated in the same digest.

42.In Angular,how did you create custom widgets ?
In Angular,I created custom widgets using “Custom Directives”. The steps I followed to creates a “Custom Widget” is :
I'll be defining "Custom Directive" for a particular module,in that function I’ll be returning an object which contains restrict as “E” for elements,that is for widgets.

43.Do you know about Angular material ?
Angular material is a companion suit to AngularJs framework.The added features in Angular material are Date,Grid,Event Binders,Google maps,Bootstrap.

44.How did you create a Custom directive ?
The steps I followed to create a “Custom directive” are : I’ll be defining "Custom Directive" for a particular module,in that function I’ll be returning an object which contains restrict as “E” for elements(that is widgets),”C” for class,”A” for an attribute,”M” for comments.

45.Have you herd about UI router ?
UI router is organized around the “states”(states are bound to named nested and parallel views which allows you to manage your application interface) which(UI router) is used in Angular,which may optionally have routes as well as other behavior.

46.Why UI router is preferred over AngularJs’ built-in router,that is,ng-route ?
With the built-in AngularJs router,that is,ng-route,only one view is allowed per page.This limitation can be

overcome using ng-include.
UI router supports multiple views and each view can have its own corresponding controller so that each of these views can be encapsulated and reused throughout the application if needed.

47.How do you achieve hiding and showing of an element in AngularJs ?
To achieve hiding and showing of an element in Angular,I made use of ng-show and ng-hide.If the expression provided to the "ng-show attribute" is true then the element will be shown,If the expression provided to the "ng-show-attribute" is false then the element will be hidden.Similarly,If the expression provided to the "ng-hide attribute" is true then the element will be hidden,If the expression provided to the "ng-hide attribute" is false then the element will be shown.

48.What is NgRepeat ?
NgRepeat is used to iterate over a collection and instantiate a new template for each item in the collection.Each item in the collection is given its own template and therefore its own scope.

49.How can you specify the sorting order in ng-repeat ?
I specified the sorting order in ng-repeat using orderBy.

50.What are the performance issues with ng-repeat ?
For every iteration in ng-repeat,$watch gets triggered which reduces the performance of an application.

51.How to overcome the issues caused by NgRepeat ?
The issues caused by NgRepeat can be overcome by using "limitTo" together with "infinite-scroll". "infinite-scroll" directive contains an expression which gets executed when the bottom of the element approaches the bottom of browser window.

52.What is an alternative to ng-repeat ?
The alternative to ng-repeat is,you can create a custom directive that writes your data without setting up unwanted $watches so that the performance of an application is improved.

53.What is TrackBy ?
By default,ngRepeat doesn't allow duplicate items in arrays.This is because when there are duplicates,it is not possible to maintain a "one-to-one" mapping between collection items and DOM elements.If you need to repeat duplicate items,you can substitute the default tracking behavior with your own using the "track by" expression. TrackBy allows you to specify your own key for ng-repeat to identify objects by,instead of just generating unique Id's.Here,the Id would be the same in both your original tasks and the updated ones from the server.Now,the ng-repeat will know not to recreate the DOM elements and reuse them.

54.What is ng-repeat-start ?
ng-repeat-start works the same as ng-repeat,but will repeat all the HTML code(including the tag it's defined on) up to and including the ending HTML tag where ng-repeat-end is placed.

55.Why NgOptions is preferred over NgRepeat though two work the same way ?
NgOptions provides some benefits such as reducing memory and increasing speed by not creating a new scope for each repeated instance.NgOptions should be used when the <select> model needs to be bound to a non-string value.This is because an <option> element can only be bound to string values at present.

56.What is NgOptions ?

The NgOptions attribute is used to dynamically generate a list of <option> elements for the <select> element using the array or object obtained by evaluating the NgOptions comprehension expression.

57.What is NgBind ?
NgBind tells Angular to replace the text content of the specified HTML element with the value of a given expression,and to update the text content when the value of that expression changes.

58.When is NgBind used instead of double curly markup ?
NgBind is preferable to use, if a template is momentarily displayed by the browser in its raw state before Angular compiles it.Since ngBind is an element attribute,it makes the binding invisible to the user while the page is loading.

59.What is NgController ?
NgController attaches a controller class to the view.

60.What is NgModel ?
NgModel directive binds an input,select,textarea to a property on the scope using NgModelController,which is created and exposed by its directive.

61.What is NgTransclude ?
If you want to add the text into the directive instead of hard coding, you can set transclude as true and finally add NgTransclude directive into your custom element

62.What is NgCloak ?
NgCloak is used to prevent the html template from being briefly displayed by the browser in its raw state while your application is loading.
NgCloak can be applied to the <body> element, but the preferred usage is to apply multiple ngCloak directives to small portions of the page to permit progressive rendering of the browser view.

63.What is NgOpen ?
HTML specification does not require browsers to preserve the values of boolean attributes such as “open".If we put an Angular interpolation expression into such an attribute then the binding information would be lost when the browser removes the attribute.The NgOpen directive solves this problem for the "open attribute”.
ng-open = “expression”

64.What is NgChange ?
NgChange : is used in conjunction with ng-model,when you’re dealing with “input”.
NgChange evaluates the given expression when the input changes.

65.What is NgClick ?
NgClick is used to specify a method or expression to run on the containing scope when the element is clicked.

65.What is NgSelect ?
NgSelect is used to bind data to an HTML <select> element.
NgSelect is used in conjunction with NgModel and NgOptions to provide sophisticated and highly performant dynamic forms.
NgOptions takes a comprehension expression for its attribute value,which is just a fancy way of saying it can take an array or an object and loop over its contents to provide the options available when using the “select” tag.It comes in the following forms:

-for array data sources:
label for "value in array"
select as label for "value in array”
label group by group for “value in array”
select as label group by group for “value in array track by trackexpr”

-for object data sources:
label for (key,value) in object
select as label for (key,value) in object
label group by group for (key,value) in object
select as label group by group for (key,value) in object

66.What is NgSubmit ?
NgSubmit is used to bind an expression to an “on submit” event.
NgSubmit also prevents the default action,(that is,sending the request and reloading the page),but only if the form does not contain an “action” attribute.

67.What is NgClass ?
NgClass is used "to dynamically set the class of an element by binding an expression, that represents all classes to be added”.
Duplicate classes will not be added.
When the expression changes,the previously added classes are removed and only then the new classes will be added.

68.What is NgAttr ?
When using NgAttr "the allOrNothing flag of “$interpolate” is used” so if any expression in the interpolated string results in “undefined”, the attribute is removed and not added to the element.This solution only works in AngularJs 1.3 and above.If you’re using AngularJs 1.2,you’re stuck with NgSwitch and NgIf.
There are some HTML tags and attributes that are processed by the browser with a “higher priority” than other attributes.This higher priority means that the browser will read "the AngularJs expression as a value" and not "the result of the expression itself”.NgAttr is added so that AngularJs can create attribute with "the expression value" during its normal digest cycle.

69.What is ng-bind-template ?
ng-bind-template is similar to ng-bind,except ng-bind-template is used to bind multiple expressions to the view.

70.What is NgInit ?
NgInit is used "to set up state” inside the scope of a directive,when the directive is invoked.

71.What is NgIf ?
ng-if is used to completely remove or recreate an element in the DOM based on an expression.If the expression assigned to ng-if evaluates to a false value,then the element is removed from the DOM,otherwise a “clone” of the element is reinserted into the DOM.
When an element is removed from the DOM using ng-if, its associated scope is destroyed.Furthermore,when it comes back into being,a new scope is created and inherits from its parent scope using prototypal inheritance.

72.What is the difference between NgIf and NgShow and NgHide ?
NgIf is used to completely remove or recreate an element in the DOM based on an expression,rather than just showing and hiding them via CSS.

73.What is NgSwitch ?

NgSwitch is used in conjunction with “ng-switch-when” and “on=propertyName” to switch which directives render in our view when the given “propertyName” changes.

74.How can you achieve accessibility in Angular ?
In Angular,accessibility can be achieved using NgAria.

75.What are the directives with which NgAria can interface ?
NgAria can interface with the following directives:
(i) NgModel
(ii) NgDisabled
(iii) NgShow
(iv) NgHide
(v) NgClick
(vi) NgDbClick
(vii) NgMessage

76.What are ARIA attributes ?
The ARIA attributes are:
aria-hidden
aria-checked
aria-disabled
aria-required
aria-invalid
aria-multiline
aria-valuenow
aria-valuemin
aria-valuemax
tabindex
If you want to turn off some of the ARIA attributes,we need to inject the $ariaProvider and use its "config method".

77.What is $interpolate ?
$interpolate service is used by the $compile service for data binding.

78.What is $interpolate service ?
$interpolate service returns an interpolation function that takes a context object against which the expressions are evaluated.
$interpolate service takes up three parameters,with only one required function.
text (string) - the text with markup to interpolate
mustHaveExpression (boolean) - If we set parameter to true,then the text will return null if there is no expression.
trustedContext (string) - Angular sends the result of the interpolation context through the $sce.getTrusted( ) method,which provides strict contextual escaping.

If it is desirable to use different beginning and ending symbols in our text,we can modify them by configuring the $InterpolateProvider.$InterpolateProvider has two methods startSymbol( ) method and endSymbol( ) method.
To modify the beginning string,we can set the starting symbol with the startSymbol( ) method.
To modify the ending string,we can set the ending symbol with the endSymbol( ) method.

79.What are the methods of $interpolate ?
$interpolateProvider.startSymbol - denotes the start of expression in the interpolated string.
$interpolateProvider.endSymbol - denotes the end of expression in the interpolated string.

80.What are the dependencies of $interpolate ?

$parse, $sce

81.What is $parse ?
$parse converts Angular expression into a function.
Syntax:
$parse(expression)

82.What is $sce ?
Strict Contextual Escaping is a mode in which AngularJs requires bindings in certain context to result in a value that is marked as safe to use for that context.

83.Can I disable SCE completely ?
Yes, you can.However this is strongly discouraged.If you take an SCE disabled application,either secure it on your own or enable SCE at a later stage.SCE can be disabled for cases where you have a lot of existing code that was written before SCE was introduced and you're migrating them a module at a time.

84.What are the methods of $sce ?
isEnabled( ) - returns true, if SCE is enabled

85.What is $interval ?
$interval service is used to refresh a DIV element or any element in specified time.$interval service works the same way as setInterval( ),but $interval service gives more control such as promise callback by which we can cancel out the timer.

86.What is $interval ?
The return value of registering an interval function is a promise.This promise will be notified upon each tick of the interval ,and will be resolved after count iterations, or run indefinitely if count is not defined.The value of the notification will be the number of iterations that have run.To cancel an interval,call $interval.cancel (promise).
In tests, we use $interval.flush(mills) to move forward by mills milliseconds and trigger any functions scheduled to run in that time.
Intervals created by $interval service must be explicitly destroyed when you are finished with them.In particular the intervals are not automatically destroyed when a controller's scope or a directive's element are destroyed.

87.What are the methods of $interval?
cancel(promise) - cancels a task associated with the promise.If the task was successfully cancelled true is returned.

88.What is $timeout ?
The return value of calling $timeout is a promise,which will be resolved when the delay has passed and the timeout function,if provided,is executed.
To cancel a timeout request,call $timeout.cancel (promise)
In tests,we use $timeout.flush( ) to synchronously flush the queue of deferred functions.

89.What are the methods of $timeout ?
cancel ([promise]) - cancels a task associated with the promise.As a result of this,the promise will be resolved with a rejection.

90.What is the difference between $interval and $timeout ?
$interval service is similar in function to the $timeout service,except it schedules a function for repeated execution

with a time interval in between.

91.What is $filter ?
$filter service is used for formatting data displayed to the user.
Syntax:
$filter (name)

92.What filters have you used in AngularJs ?
filters can be added to expressions and directives using a pipe character.
currency - Formats a number to a currency format
lowercase - Formats a string to lowercase
uppercase - Formats a string to uppercase
orderBy - orders an array by an expression
filter - selects a subset of items from an array
json - converts a JavaScript object to JSON string.json filter is used for debugging.When using the double curly markup notation,the binding is automatically converted to JSON.
limitTo - creates a new array containing only a specified number of elements.The elements are taken from either the beginning or the end of the source array as specified by the value and sign of limit.
date - formats date to a string
number - formats a number as text.If the input is null or undefined,it will just be returned.If the input is infinite, infinity symbol is returned.If the input is not a number an empty string is returned.

93.What are Services ?
Services provide a method to keep data around for the lifetime of the app and communicate across controllers in a consistent manner.
Services are singleton objects that are instantiated only once per app by the $injector and lazy-loaded(that is,Services are created only when necessary).
An example of an AngularJs service is $http,which provides low-level access to the browser's XMLHttpRequest object.Rather than needing to dirty the application with low-level calls to the XMLHttpRequest object,we can simply interact with the $http API.
To create a service,first we need to register a service.Once a service is registered,the Angular compiler can reference the service and load the service as a dependency for runtime use.

94.How many ways a service can be created ?
A service can be created in Five ways :
(i) factory( ) service - is a quick way to create and configure a service.As the "factory method" takes "function" as an argument,we can inject as many dependencies as needed in that function.

(ii) service( ) service - is used to register an instance of a service using a constructor function.service( ) function will instantiate the instance using the "new" keyword when creating the instance.
service( ) service - works the same as the factory( ) service.But instead of a "function",it receives a "Javascript class" or a "constructor function" as an argument.

(iii) constant( ) service - is used for declaring constants in our application and using the constants anywhere needed by just adding it as a dependency.

(iv) value( ) service - It is like a key value pair or like a variable that has some value.It just stores a single value.

(v) provider( ) service - is the core service and all other services are built on top of it.
The provider( ) service is mainly used when we want to expose API for application-wide configuration,which must be configured before starting the application.
The provider( ) service is defined as a custom and must implemented $get method.
The provider( ) service is the parent of all other services and it is almost complex but more configurable one.

95.What is the difference between creating an Angular Js service using a service( ) method and a factory( ) method ?
The service( ) method uses the function constructor to register an Angular Js service .It returns the function object.
The factory( ) method returns the value after execution of the service function.It can return any value like primitive value, function or object.

96.What is the difference between value( ) method and constant( ) method ?
The differences between value( ) method and constant( ) method are :
A “constant" can be injected into a "config function”,while a “value” cannot be injected.
A constant( ) method is used for configuration data,while a value( ) method is used to register a "service objet" or “function”.

97.Why do we prefer provider( ) service over factory( ) service ?
A provider( ) service is complex but it is more configurable one.

98.What is a provider ?
A provider is an object with a $get method.The injector calls $get method to create a new instance of a service.The provider can have additional methods which will allow for configuration of the provider.

99.How to create a custom provider ?
A custom provider can be created using the following six methods of $provide :
(i) service - A service is an injectable constructor.Incase of a service,we can specify the dependencies in a function.A service is a singleton.
(ii) factory - A factory is an injectable function.Incase of a factory,we can specify the dependencies in a function.A factory is a singleton.The difference between a factory and service is that a factory injects a plain function so Angular will call the function and a service injects a constructor,this constructor creates a new object so “new” is called on a service.
(iii) constant - A constant can be injected everywhere.A constant cannot be intercepted by a decorator,that is,the value of a constant can never be changed.
(iv) value - A value is a simple injectable value.A value can be a string,number or a function.A value differs from a constant in that a value cannot be injected into configurations,but it can be intercepted by a decorator.
(v) provider - A provider is a configurable factory.A provider accepts an object or a constructor.
(vi) decorator - A decorator can modify or encapsulate other providers except a “constant”(that is,a constant provider).A Decorator works on "the $get property of a provider" to alter their service.A constant's provider has no “$get" hence a constant cannot be decorated.A value’s provider has a $get hence a value can be decorated.

100.What is a Directives ?
A directive is a marker on a HTML tag that tells Angular to run or reference some JavaScript code.A directive method takes two-arguments one is “name”(The name of the directive as a “string" that we’ll refer to inside of our views) and the other is “factory function”(The factory function returns an “object" providing options that tell the $compile service how the directive should behave when it is invoked in the DOM.The factory function can also return a “function" instead an “object",but it is best practice to return an “object" because when a “function" is returned it is often referred to as the “postLink” function,which allows us to define the “link” function for the directive,that is,returning a “function" instead an “object" limits us to a narrow ability to customize our directive and thus is good for only "simple directives”).The “factory function” is invoked only “once” when the compiler matches the directive the first time.Just like the “ .controller function “,we invoke a “directive’s factory function” using “$injector.invoke”,then the directive’s lifecycle begins,starting with the "$compile method" and ending with the "link method”.

101.What if I want custom directives to be applied on element as well as attributes ?
directive.restrict = ‘EA’

102.What are the different types of custom directives available in Angular ?
Element directive
Attribute directive
CSS class directive
Comment directive

103.What are the possible options that we can provide inside a directive definition ?
(i) restrict : is an optional argument.It is responsible for telling Angular in which format our directive will be declared in the DOM.By default,Angular expects that we'll declare a "custom directive" as an "attribute",meaning the restrict option is set to "A".The available options are as follows:
E(an element)
A(an attribute)
C(a class)
M(a comment)
The above options can be used alone or in combination.
By default,Angular expects that we'll declare a "custom directive" as an "attribute",because "attributes" work across all the browsers including older versions of IE without having to register a "new tag" in the head of the document.

(ii) priority : priority option is set to 0 by default. "ngRepeat"sets the priority to 1000 so that it always gets invoked before other directives on the same element.
(iii) terminal : can be set to "true" or "false".We use "terminal " option to tell Angular to stop invoking any further directives on an element that have a higher priority.All directives with the same priority will be executed,however.
(iv)Template : is an optional argument."Template" can be set to either "a string of HTML" or "a function that takes two arguments - tElements and tAttrs
(v)TemplateUrl : is an optional argument.If provided,it must be set to either "the path to an HTML file as a string" or "a function that takes two arguments: tElement and tAttrs.The function must return the path to an HTML file as a string.
(vi)replace : is an optional argument.If provided,it must be set to true.It is set to false by default,that is the directive's template will be appended as a child node within the element where the directive was invoked.
(vii)scope : is an optional argument.It can be set to "true" or to "an object,{ } ".By default,it is set to false.When scope is set to true,a new object is created that prototypically inherits from its parent scope.When scope is set to { },an isolated scope is created.
(viii)transclude : is an optional argument.if provided,it must be set to true.By default,it is set to false.Transclusion is most often used for creating reusable widgets.A great example is "a modal box" or "a navbar".Transclusion allows us to pass in an entire template,including its scope,to a directive.Doing so gives us the opportunity to pass in arbitrary content and arbitrary scope to a directive.In order for scope to be passed in,the scope option must be isolated,{ } or set to "true".If the scope option is not set,then the scope available inside the directive will be applied to the template passed in.
(ix)controller : It can be set to either a string or a function.When it is set to a string,the name of the string is used to lookup "a controller constructor function" registered elsewhere in our application.A controller can be defined inline within a directive by setting "the controller function" as an "anonymous constructor function".
(x)controllerAs :
(xi)require : It can be set to either "a string" or "an array of strings".The string refers to the name of directives that reside in the current scope of the current directive.The string may optionally be prefixed with ?, ^ ,?^ which change the behavior when looking up a controller.If the string isn't prefixed,that means we tell the directive to locate the required controller on the "named directive provided" and throw an error if no controller is found.
(xii)link
(xiii)compile

104.What is a compile( ) function ?
Use the "compile( )" function to change the original DOM(template element) before AngularJs creates an instance of it and before a scope is created.

105.What is a link function?

The link function is used to create a directive that manipulates the DOM.
The link function is optional.If the compile function is defined,it returns the link function; therefore,the compile function will overwrite the link function when both are defined.
If you create a directive that only has a “link” function,AngularJs treats the function as a “post-link” function.

106.What are pre-link( ) and post-link( ) functions ?
A “pre-link” function of an element is guaranteed to run before any "pre-link” or “post-link” function of any of its child elements.
Use “pre-link” function to implement logic that runs when AngularJs has already compiled the child elements,but before any of the child element’s "post-link” functions have been called.
Use “post-link” function to execute logic,knowing that all child elements have been compiled and all “pre-link” and “post-link” functions of child elements have been executed.

107.What is $route ?
$route is used for deep-linking URLs to controllers and views.
$route watches $location.url( ) and tries to map the path to an existing route definition.
$route service is used in conjunction with ngView directive and $routeParams service.
Routes:
To declare all the application routes in Angular,we use “when" method and “otherwise” method.
To create a route on a specific module,we use “config” function,and to add a specific route we can use “when” method,the “when" method takes two parameters “path” and “route”.

108.What are the methods of $route service ?
The methods of $route service are:
reload( ) - It causes “$route” service to reload the current “route" even if “$location" hasn’t changed.As a result of that,ngView creates new scope and reinstantiates the controller.
updateParams(newParams) - It causes “$route”service to update the "current URL",replacing "current route parameters" with those specified in newParams.

109.What are the dependencies of $route service ?
The dependencies of $route service are :
$location
$routeParams

110.What does $location not do ?
$location doesn’t cause a full page reload when the browser URL is changed.To reload the page after changing the browser URL,use the lower-level API,that is,$window.location.href

111.What are the configuration modes of $location service ?
The two configuration modes of $location service are Hash bang mode(which is a default mode) and HTML5 mode.The configuration modes control the format of the URL in the browser address bar.

112.How to use $location outside of the scope life-cycle ?
$location knows about Angular’s scope life-cycle.When a URL changes in the browser,it updates the $location and calls $apply so that all $watches or $obervers are notified.
If you change $location inside the $digest phase,$location will propagate this change into the browser and will notify all the $watches or $observers.
If you change $location from outside Angular,you must call $apply to propagate the changes.

113.What is $routeParams ?

$routeParams service allows us to retrieve the current set of "route parameters".The route parameters are a combination of $location's search ( ) and path( ) .The "path" parameters are extracted when the "$route" path is matched.Otherwise,"path" parameters take precedence over "search" params.
$routeParams are only updated after a "route change" completes successfully.This means that you cannot rely on $routeParams begin correct in route resolve functions.Instead,you can use "$route.current.params" to access the new route's parameters.

114.How can you implement SPA(single page application) using AngularJs ?
Using $routeprovider,we can implement SPA using AngularJs.

115.What is the difference between a service and a factory ?
A service is a singleton object whereas a factory is not a singleton object.

116.What is scaffolding ?
Scaffolding is a framework that allows you to do basic CRUD operations against your database with little or no code.

117.What are the best features added in Angular 1.4 ?
A lot of animations have been introduced in Angular 1.4

118.What is $window ?
In JavaScript,window is a global variable and causes testability problems.In Angular,$window reference to the browser's window object,so it may be overridden,removed or mocked for testing.

119.What is NgForm ?
ngForm is used to group controls,ngForm is not a replacement for <form> tag.

120.What is angular.element ?
if jQuery is available,angular.element is an alias for the jQuery function.If jQuery is not available,angular.element delegates to Angular's built-in subset of jQuery,called "jqLite".
angular.element(element)

121.What is angular.equals ?
angular.equals determines if two objects or two values are equivalent.
angular.equals(o1,o2)

122.What is angular.extend ?
angular.extend extends the destination object "dst" by copying own enumerable properties from the "src" object to "dst" object.We can specify multiple "src" objects.
angular.extend(dst, src)
angular.extend doesn't support recursive merge(that is,deep copy) for this we use angular.merge

123.What is angular.fromJson ?
angular.fromJson deserializes a JSON string.
angular.fromJson(json)

124.What is angular.noop ?

angular.noop performs no operation.This function is useful when writing code in the functional style.

125.What is angular.toJson ?
angular.toJson serializes input into a JSON-formatted string.Properties with leading $$ characters will be stripped since angular uses this notation internally.
angular.toJson (obj, pretty)
pretty : if set to true,JSON output will contain newlines and whitespace.If set to an integer,JSON output will contain that many spaces per indentation.

126.What is Angular Caching ?
$cacheFactory is the service that generates cache objects for all Angular services.Internally,$cacheFactory creates a default cache object,even if we don't create one explicitly.
To create a cache object,we use $cacheFactory and create a cache by ID.
var cache = $cacheFactory('myCache');

The Cache object has the following methods:
info( ) - method returns the ID,size,and options of the cache object.
put( ) - method allows us to put a key(string) of any Javascript object value into the cache.
The put( ) method returns the value of the cache that we put in.
get( ) - method gives us access to the cache value for a "key".If the "key" is found,it returns the "value",otherwise it returns "undefined".
remove( ) - function removes a key-value pair from the cache,if it is found.If it is not found,then it returns "undefined".
removeAll( ) - function resets the cache and removes all cached values.
destroy( ) - method removes all references of this cache from the "$cacheFactory" cache registry.

127.Does Angular use JQuery library ?
Yes,Angular can use JQuery if it is present in your app when the application is being bootstrapped.If JQuery is not present in your script path,Angular falls back to its own implementation of the subset of JQuery that we call JQLite.

128.Does Angular supports jQuery ?
Angular 1.3 supports only jQuery 2.1
If you include jQuery in your page, however,AngularJs will use jQuery instead of jqLite when wrapping elements within your directives.
"angular.element( ) method" is actually an alias (that is,a direct reference) to the "jQuery( ) constructor function".This means you have access to the full suite of jQuery features without needing a reference to the jQuery or "$" global references.This means that "angular.element( ) method" contains all of the core jQuery functions,like "jQuery.contains( ) and jQuery.proxy( )".

129.What is angular.element ?
If jQuery is available,"angular.element" is an alias for the jQuery function.If jQuery is not available,angular.element delegates to Angular's built-in subset of jQuery called "jQuery lite" or "jqLite".
To use jQuery,simply ensure it is loaded before the angular.js file
angular.element(element) - element is an HTML string or DOMElement to be wrapped into jQuery.

130.What is jqLite ?
jqLite library doesn't cover all of the methods in the entire jQuery library,but it covers only those methods that Angular needs,so jqLite is said to be light weight.
The methods covered by jqLite are:
1.addClass( )
2.after( )
3.append( )

4.attr( )
5.bind( )
6.children( )
7.clone( )
8.contents( )
9.css( )
10.data( )
11.eq( )
12.find( )
13.hasClass( )
14.next( )
15.unbind( )
16.parent( )
17.prepend( )
18.prop( )
19.ready( )
20.remove( )
21.removeAttr( )
22.removeClass( )
23.removeData( )
24.replaceWith( )
25.text( )
26.toggleClass( )
27.triggerHandler( )
28.val( )
29.wrap( )

131.What is Batarang ?
Angular Batarang is a chrome extension developed by the Angular team at Google that integrates as a debugging tool for Angular apps.
we can peak into the performance of our application by using the “performance section of Batarang”.
we can inspect the dependency graph,that is,we can look at the dependencies of our application and view the different libraries of our application to see what they depend upon and track libraries that aren’t dependencies of the application at all.

132.What is the object you used for web services or web API calls ?
In Angular,I used $http and $resource.

133.What is Restangular ?
Restangular is an Angular service that is used to fetch data from the rest of the world.
$http and $resource are built into the AngularJs framework,but they have limitations.
Restangular takes a complete different approach to “XMLHttpRequests" and make it a pleasant experience.
The advantages of Restangular are:
(i)promises
(ii)supports all HTTP Methods
(iii)while $resource requires us to specify the URLs we want to fetch,Restangular doesn’t require us to know the URLs in advance,nor do we have to specify them all upfront,other than the base URL.
(iv)unlike $resource,we only need to create one instance of the Restangular resource object.
Restangular is the combination of both $q and $http.
$http and $resource doesn’t have promises
Incase of $http,we can cache requests if you set the property “cache" to true.Incase of Restangular,we can cache requests using the property “defaultHttpFields”.
Incase of $http,we can use $routeProvider.resolve( ).
Incase of Restangular,we can use $routeProvider.resolve( ) as every method in Restangular returns a promise.

134.What is $http ?
$http service makes hitting a URL easy,and also utilizes promises.

```
 $http({method: 'POST', url: '/contact', params: {name: "Joe", email: "joe@foob.ar"}).
    success(function(data, status, headers, config) {
      // do something with our response
    }).
    error(function(data, status, headers, config) {
      // process the error response
    });
```

135.What is $resource ?
$resource is built on the top of $http,$resource makes RESTful API interactions a lot thinner than if they were implemented with $http.

```
var Contact = $resource('/contact/:id', {id:'@id'});
 var joe = Contact.get({id:42}).$promise.
   then(function(data) {
     // process contact #42
   }).
   catch(function(response) {
     // process the error response
   })
```

136.What is Restangular ?
Restangular makes RESTful API interactions even simpler than $resource.

```
Restangular.one('contact', 42).
    then(function(data){...}).
    catch(function(response){…})
```

137.What are promises ?
A promise represents a one-time event typically the outcome of an asynchronous task like an AJAX call.At first,the promise will be in a pending state eventually either it is resolved or rejected.Once a promise is resolved or rejected,it will remain in the state forever and its callbacks will never fire again but you can attach more callbacks to the promise whenever you want and they will fire immediately.
Promises in Angular are implemented with $q.The $q implementation was inspired by "Kris Kowal's Q".
$q uses ".then( ) " to include a "success" function and "error" function,while $http uses "success" function and "error" function.
then( ) accepts two functions as parameters: a function to be executed when the promise is fulfilled,and a function to be executed when the promise is rejected.

138.What is $q service in Angular ?
$q service runs functions asynchronously and use the return values of the functions when they are done processing.
From the perspective of dealing with error handling, deferred API and promise API are to asynchronous programming while try,catch,and throw are to synchronous programming.
we can create a deferred object using "$q.defer( )" .The purpose of the deferred object is to expose the associated promise instance as well as APIs that are used for signaling the successful or unsuccessful completion,as well as the status of the task.

The methods of Deferred API are:
resolve(value) -resolves the derived promise with the value.
reject(reason) - rejects the derived promise with the reason.
notify(value) - provides updates on the status of the promise's execution.This may be called multiple times before the promise is either resolved or rejected.

we can create a promise object using “deferred.promise”.The purpose of the promise object is to allow access to the result of the deferred task when it completes.
The methods of Promise API are:
then(successCallback, errorCallback, notifyCallback) - regardless of when the promise will be resolved or rejected,”then” calls one of the success or error callbacks asynchronously as soon as the result is available.notify callback may be called zero or more times to provide a progress indication,before the promise is resolved or rejected.

catch(errorCallback) - is the shorthand for “promise.then(null, errorCallback)”
finally(callback, notifyCallback) - allows you to observe either the fulfillment or rejection of a promise.This is useful to release resources or do some clean-up that needs to be done whether the promise was rejected or resolved.

chaining promises:
Because calling “then” method of a promise returns a new derived promise,it is easy to create a chain of promises.

139.What is the dependency of $q ?
The dependency of $q is $rootScope.

140.What are the methods of $q ?
(i) defer( ) - creates a deferred object which represents a task which will finish in the future.
(ii) reject(reason) - creates a promise that is resolved as rejected with the specified reason.
(iii) when(value, [successCallback], [errorCallback], [progressCallback] - is useful when we are dealing with an object that might or might not be a promise,or if the promise comes from a source that can’t be trusted.
value - can be a value or a promise
(iv)resolve(value, [successCallback], [errorCallback], [progressCallback]) - is an alias of “when” to maintain naming consistency with ES6.
(v)all(promises) - combines multiple promises into a single promise that is resolved when all of the input promises are resolved.

141.What is JSONP ?
When using JSONP,we need to be aware of the potential security risks.We can only use JSONP to send GET requests,since we’re setting a GET request in the <script> tag.Aditionally,it is tough to manage errors on a script tag.Hence,we should JSONP sparingly and only with servers we trust and control.

142.What is Cross Origin Policy ?
Cross Origin Policy: is used to replace the JSONP hack in a standard way.
CORS is an extension to the standard XMLHttpRequest object that allows Javascript to make cross-domain XHR calls.It does so by “preflighting” a request to the server to effectively ask for permission to send the request,that is,the browser actually sends two requests : the “preflight” and the “request”.First,the browser issues a “preflight” request wherein the server requests permission to make the request.If the permissions have been granted,then the browser can make the actual request.

To use CORS within Angular :

```
angular.module (‘myApp’)
    .config ( function ($httpProvider) {
                $httpProvider.defaults.useXDomain = true;
                delete  $httpProvider.defaults.headers
                  .common [ ‘X-Requested-With ‘ ];
              } ) ;
```

143.How can you handle XSS using AngularJs ?
AngularJs can handle XSS using $sceProvider and $sanitizeProvider.

144.What is $SCE in AngularJs ?
SCE disabled application allows the user to render arbitrary HTML into the DIV,that is,HTML is an example of a context where "rendering user controlled input" creates security vulnerabilities.
For the case of HTML,you might use a library,either on the client-side,or on the server side,to sanitize unsafe HTML before binding to the value and rendering it in the document.

145.How can you ensure that this library sanitizes the HTML ?
To be secure by default,we want something to explicitly say that it is safe to use a value for binding in that context.
$sce.trustAsHtml explicitly says that it is safe to use a value for binding in SCE context.
In SCE contexts,directives and code will bind to the result of $sce.getTrusted (context,value) rather than to the value directly.
Directives use $sce.parseAs rather than $parse to watch attribute bindings,which performs $sce.getTrusted behind the scenes of non-constant literals.

SCE applies to only interpolation expressions.If your expressions are constant literals,they are automatically trusted and we don’t need to call $sce.trustAs on them.
Additionally, a[href] and img[src] automatically sanitize their URLs and don’t pass them through $sce.getTrusted.SCE doesn’t play a role here.

isEnabled( ) - returns true if SCE is enabled,false otherwise.

parseAs(type,expression) - converts an Angular expression into a function.This is like $parse and is identical when the expression is a literal constant.Otherwise,it wraps the expression in a call to $sce.getTrusted (type,result)
type - is the type of SCE context in which this result will be used.
The following are the types of SCE contexts :
$sce.HTML - For HTML that is safe to source into the application.
$sce.CSS - For CSS that is safe to source into the application
$sce.RESOURCE_URL - For URLs that are not only safe to follow as links,but whose contents are also safe to include in your application.
$sce.JS - For Javascript that is safe to execute in your application’s context.

result - is the return value from $sce.trustAs( ).If it is invalid,then it will throw an exception.

trustAs (type,value) - returns an object that is trusted by angular for use in specified strict contextual escaping
type - is the type of SCE context in which this result will be used.
value - The value that should be considered trusted

trustAs (type,value) - returns an object that can be passed to getTrusted(type, maybeTrusted)

trustAsHtml(value) - returns an objet that can be passed to $sce.getTrustedHtml(value)
trustAsUrl(value) - returns an object that can be passed to $sce.getTrustedUrl(value)
trustAsResourceUrl(value) - returns an object that can be passed to $sce.getTrustedResourceUrl(value)
trustAsJs(value) - returns an object that can be passed to $sce.getTrustedJs(value)

parseAsHtml(expression)
parseAsCss(expression)
parseAsUrl(expression)
parseAsResourceUrl(expression)
parseAsJs(expression)

146.What is dirty checking ?
Dirty checking is a relatively efficient approach to checking for changes on a model.

147.How to check error validations in Angular ?

Form and controls provide validation services,so the user can be notified of invalid input before submitting a form,hence this provides client-side validation but client-side
validation alone isn't enough because it can easily be circumvented and thus cannot be trusted so server-side validation is also necessary.

To allow styling of a Form as well as controls,ngModel adds the following CSS classes :
NgValid - the model is valid
NgInvalid - the model is invalid
NgValid-[key] - the model is valid for each valid key added by $setValidity
NgInvalid-[key] - the model is invalid for each invalid key added by $setValidity
NgPristine - the control hasn't been interacted with yet
NgDirty - the control has been interacted with
NgTouched - the control has been blurred

By default,any changes to the content will trigger a model update and Form validation.This behavior can be overridden using NgModelOptions to bind only to the specified
list of events,that is,ng-model-options = " { update : 'blur' } " will update the model and validate the Form only after the control loses focus.

148.What is NgView ?
NgView tells Angular where we wish to inject "HTML" based on the URL a user visits.

149.How can you make sure that your template is fetched from a trusted source ?
$getTrustResourceUrl is used to protect our template from being fetched from untrustworthy resources.

what is the difference between $location and window.location ?

150.How to establish communication between modules of your application ?
(i)Using services
(ii)Using events
(iii)By assigning models on $rootScope
(iv)Directly between controllers using $parent,$$childHead, $$nextSibling
(v)Directly between controllers,using ControllerAs, or other forms of inheritance

151.Explain how directives are instantiated ?
Firstly,$compile( ) function is executed which returns two link functions, preLink and postLink. $compile( ) function is executed for every directive,starting from the parent element,child element and then grandchild element.
Secondly,controller and preLink function are executed for every directive,starting from the parent element,child element and then grandchild element.
Thirdly,postLink function is executed for every directive,starting from the grandchild element,child element and then parent element.

152.what are the two ways to improve the performance of an AngularJS application ?
(i)Disabling debug data - can be achieved using $compileProvider
```
myApp.config(function ($compileProvider) {
  $compileProvider.debugInfoEnabled(false);
});
```
The above code disables appending scope to elements, making scopes inaccessible from the console.
(ii)Enabling strict DI mode
```
<html ng-app="myApp" ng-strict-di>
```

153.when a scope is terminated how many destroy events are fired ?
when a scope is terminated two destroy events are fired: the first one is AngularJS event, $destroy and the second one is jQuery event ,$destroy
The first one can be used by AngularJS scopes where they are accessible,such as in controllers or link functions.
The second one is called whenever a node is removed, which may just happen without scope teardown.

154.what should be the maximum number of concurrent watches ?
2000

155.How can you track the watch count and digest cycles ?
ng-stats is used to track the watch count and digest cycles.

156.where should we implement the DOM manipulations in AngularJS ?
In directives.
DOM manipulations should not exist in controllers, services or anywhere else but in directives.

157.How would you specify that a scope variable should have one-time binding only ?
By using : : infront of a scope.

158.what is the difference between one-way binding and two-way binding ?
One-way binding implies that the scope variable in the HTML will be set to the first value its model is assigned to.
Two-way binding implies that the scope variable will change its value every time its model is assigned to a different value.

159.what is angular.copy ?
angular.copy 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.

160.when should we use an attribute directive and an element directive ?
we should use element directive when we are creating a component that is in control of the template.
we should use attribute directive when we are decorating an existing element with new functionality.

161.How do you reset a $timeout, $interval( ) and disable a $watch( ) ?
we can reset a $timeout or $interval( ) by assigning the result of the function to a variable and then call the .cancel( ) function.
we can disable $watch( ) by calling its de-registration function.

162.what is DDO Directive Definition Object ?
DDO Directive Definition Object is used while creating a custom directive.

163.what is a singleton pattern and where we can find it in AngularJS ?
singleton pattern restricts the use of a class more than once.we can find singleton pattern in dependency injection and in services.

164.what is an interceptor ?
An interceptor is a factory registered in $httpProvider.The two types of request that go through an interceptor are request and response.

165.How can you implement application-wide exception handling in your AngularJS application ?
AngularJS has built-in error handler service called $exceptionHandler.

166.How can you change the template of a directive before it is executed and transformed ?
we use compile function.The compile function gives us access to directive's template before transclusion occurs and templates are transformed,so changes can be safely made to DOM elements.This scenario is useful where the DOM needs to be constructed based on runtime directive parameters.

167.What is the difference between AngularJS module pattern and AMD module pattern ?
AngularJS expects that all modules are loaded into a browser before an application can be bootstrapped
AMD modules are mostly concerned with scripts loading. The specification allows you to break down your application into several smaller scripts and load only the needed parts asynchronously, on-the fly, when needed.

You can use AMD modules to load AngularJS library, all the third-party dependencies, and the application's code before bootstrapping your application. The benefit here is that you can load all the scripts asynchronously and avoid blocking processing of the <script> tags. This might have a positive effect on the performance of your application (or not!) depending on the number of libraries to be loaded.
While working with AMD modules you can no longer use the ng-app directive to bootstrap AngularJS application. The reason is that AngularJS will start to process the DOM tree on the document-ready event. At this point asynchronous modules might not yet be loaded in a browser and AngularJS would try to bootstrap an application before all the required JavaScript les are downloaded. To cover this use case AngularJS provides a manual, JavaScript-based way of bootstrapping an application: angular.bootstrap.

169.How to get controllers and directives talk to each other ?
(i)The simplest way to get a "two-way communication" between a controller and a directive is to setup a shared object between the controller and a directive via " = " scope binding.Both the controller and directive can set-up watchers on the shared object to notify the changes on the controller and directive and can thus use it as a way to communicate with each other.

(ii)Another way to get controllers and directives to talk to each other is via services.However,the biggest difference is that services are singleton objects within a module.

(iii)Another way to get controllers and directives to talk to each other is using events.

What is the advantage of using events over shared objects to establish communication between a controller and a directive ?
The advantage of using events to establish communication between controller and directive is :
events let you communicate across scopes.
Using scope.$broadcast( ),the directive can communicate "upward" in the scope hierarchy,that is,any controller "above" the directive's scope can listen to the event and thus receive notifications from the directive.
Using scope.$emit( ),the directive can communicate "downward" in the scope hierarchy,that is,any controller "below" the directive's scope can listen to the event and thus receive notifications from the directive.

170.How to establish communication between modules of your application ?
using services
using events
by assigning models on $rootscope
Directly between controllers using $parent,$$childHead,$$nextSibling
Directly between controllers using ControllerAs or other forms of inheritance

171.what should be the maximum number of concurrent watches ?
2000

172.Explain how directives are instantiated ?
Firstly,$compile is executed which returns two functions pre-link and post-link functions.$compile is executed for every directive starting from parent element,child element and then grand child element
secondly,controller and pre-link function are executed for every directive starting from parent element,child element, and then grand child element.
Thirdly,post-link function is executed for every directive starting from grand child element, child element and then parent element.

173.What are the two ways to increase the performance of an AngularJS application ?
The two ways to increase the performance of an AngularJS application are :
Disabling Debug data - which can be achieved by using $compileProvider

```
myApp.config(function($compileProvider){
        $compileProvider.debugInfoEnable(false)
});
```

Enabling strict DI mode

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

174.when a scope is terminated how many destroy events will be fired ?
When a scope is terminated two destroy events will be fired.The first one is AngularJS event $destroy and the second one is jQuery event $destroy.
The first one is used by AngularJS scopes where they are accessible,such as in controllers and in link functions.
The second one is used when a child node is removed,which may just happen without scope teardown.

175.How can you track the watch count and digest cycle ?
ng-stats is used to track the watch count and digest cycle.

176.what is DDO ?
DDO Directive Definition Object is used while creating a custom directive

177.what is an Interceptor ?
An interceptor is a factory registered in $httpProvider.The two types of requests that go through an interceptor are request and response.

178.How can you implement application-wide exception handling in AngularJS ?
Angular has built-in error handling service called $exceptionHandling service.

179.How can you reset a $timeout or $interval ?
we can reset a $timeout or $interval by assigning the result of a function to a variable and then call the .cancel( )function.

180.How can you disable a $watch ?
we can disable a $watch by calling its deregistration function.

181.How can you specify that a scope should have one-time bonding only ?
By using :: infront of a scope,we can specify that a scope should have one-time binding only.

182.what is angular.copy ?
angular.copy is used to create a deep copy of a variable .The deep copy of a variable means that it doesn't have same memory reference as that variable.

183.what is the difference between one-way binding and two-way binding ?
one-way binding implies that the scope variable in the HTML will be set to the first value its model is assigned to.
two-way binding implies that the scope variable will change its value when its model is assigned to a different value.

184.what is a singleton pattern ?
A singleton pattern restricts the use of a class more than once.we can find the singleton pattern in dependency injection and in services

185.How can you change the template before it is executed and transformed ?
we use compile ( )function.The compile( )function gives us access to the directive's template before transclusion occurs and the template is transformed so the changes can be safely made to the DOM elements.This scenario is useful where the DOM needs to be constructed based on runtime directives parameters.

186.How to manually bootstrap an AngularJS application ?

Example: Manual Bootstrap

```
<!DOCTYPE html>
<html >
<head>
    <title>Angular Bootstrap</title>
    <script src="~/Scripts/angular.js"></script>
</head>
<body>
    <div>
         {{2/2}}
    </div>
    <div>
        {{5/2}}
        <div>
            {{10/2}}
        </div>
    </div>
<script>
     angular.element(document).ready(function () {
         angular.bootstrap(document);
     });
</script>
 </body>
 </html>
```

In the above example, we call angular.bootstrap() function and specify the root element, which is document object. This will initialize AngularJS and compile all the elements starting from root element i.e. the whole document in this example.

Note that multiple ng-app directives are NOT allowed in a single HTML document.

187.How to make GET and POST requests using $http ?

Example: $http()

```
<!DOCTYPE html>
<html >
<head>
    <script src="~/Scripts/angular.js"></script>
</head>
<body ng-app="myApp">
    <div ng-controller="myController">
            Response Data: {{data}} <br />
        Error: {{error}}

    </div>
    <script>
       var myApp = angular.module('myApp', []);

        myApp.controller("myController", function ($scope, $http) {

            var onSuccess = function (data, status, headers, config) {
                $scope.data = data;
            };

            var onError = function (data, status, headers, config) {
                $scope.error = status;
            }

           var getReq = {
                    method: 'GET',
                    url: '/demo/getdata'
                };

            $http(getReq).success(onSuccess).error(onError);

            var postReq = {
                    method: 'POST',
                    url: '/demo/submitData',
                    data: { myData: 'test data' }
                };

            $http(postReq).success(onSuccess).error(onError);

        });
    </script>

</body>
</html>
```

188.How to implement a calculator ?

```javascript
var CalculatorService = angular.module('CalculatorService', [])
.service('Calculator', function () {
    this.square = function (a) { return a*a};

});
```

```javascript
var myApp = angular.module('myApp', ['CalculatorService']);
myApp.controller('CalculatorController', function ($scope, Calculator) {

    $scope.findSquare = function () {
        $scope.answer = Calculator.square($scope.number);
    }
});
```

```
<div class="container">
        <div>
            <div ng-controller="CalculatorController">
                Enter a number:
                <input type="number" ng-model="number">
                <button  ng-click="findSquare()">Square</button>
                <div>{{answer}}</div>
            </div>
        </div>
    </div>
```

189.How to enter the first name and last name of a student and how to retrieve the first name and last name of a student ?

```javascript
var StudentService = angular.module('StudentService', [])
StudentService.factory('StudentDataOp', ['$http', function ($http) {

    var urlBase = 'http://localhost:2307/Service1.svc';
    var StudentDataOp = {};

    StudentDataOp.getStudents = function () {
        return $http.get(urlBase+'/GetStudents');
    };

    StudentDataOp.addStudent = function (stud) {
        return $http.post(urlBase + '/AddStudent', stud);
    };
    return StudentDataOp;

}]);
```

```javascript
var myApp = angular.module('myApp', ['StudentService']);

myApp.controller('StudentController', function ($scope, StudentDataOp) {
    $scope.status;
    $scope.students;
    getStudents();

    function getStudents() {
        StudentDataOp.getStudents()
            .success(function (studs) {
                $scope.students = studs;
            })
            .error(function (error) {
                $scope.status = 'Unable to load customer data: ' + error.message;
            });
    }
```

```
    $scope.addStudent = function () {

        var stud = {
            ID: 145,
            FirstName: $scope.fname,
            LastName: $scope.lname
        };
        StudentDataOp.addStudent(stud)
            .success(function () {
                $scope.status = 'Inserted Student! Refreshing Student list.';
                $scope.students.push(stud);
            }).
            error(function (error) {
                $scope.status = 'Unable to insert Student: ' + error.message;
            });
    };
});
```

```
<div ng-controller="StudentController">
            <form class="well">
                <input type="text" name="fname" ng-model="fname" placeholder="first name" /> <b
r/>
                <input type="text" name="lname" ng-model="lname" placeholder="last name" />

                <br /><br/>
                <input type="button" value="add student" ng-click="addStudent()" />
            </form>
            <br/>
            <table class="table">
                <tr ng-repeat="s in students">
                    <td>{{ s.FirstName }}</td>
                    <td>{{ s.LastName }}</td>
                </tr>
            </table>
        </div>
```

190.How many ways we create an AngularJS service ?

1. AngularJS Value

This is the simplest Angularjs service type that we can create and use. It is just like a key value pair. Or like a variable that has some value. **It just stores a single value**. Assume we have a service that displays user name. Let us see how we can create this service.

```
var app=angular.module("app",[]);
app.value("username","John");
```

Using Value:

We can use the service anywhere by injecting, for simplicity we are injecting in a controller and using it, here is how it looks like:

```
app.controller("MainController",function($scope, username){
   $scope.username=username;
});
```

In this example we have created a username value service and used it in MainController.

2. AngularJS Factory

As we have seen **Value** service, they are very easy to write but it lacks many important features. So next service type we will take a look at is **"Factory"** service. By creating factory service we can inject many other services in it, unlike Value service we cannot add any dependency in it. So here is how to create:

```
app.factory("username",function(){

   var name="John";
   return {
       name:name
   }
});
```

The above code shows how we can create factory service. As it takes a function, so we can inject as many dependencies as needed in that function. Also we can include any methods needed in this type of service. **And the function must return some object as in our example we have returned an object with property name**. So let us take a look at how we can use this service:

Using Factory:

We had returned an object from service which had a property name so we could access it and use it anywhere needed. Let's see how we can use it in controller:

```
app.controller("MainController",function($scope, username){
   $scope.username=username.name;
});
```

We are assigning the username from factory service to our scope username.

3. AngularJS Service

The **service** service works same as the factory. But instead of a function it receives a Javascript class/a constructor function as argument. So let us start with the example, suppose we have a function like this:

```
function FooBar(foo){
   this.variable="value";
}
```

Now we want to convert it into a service, first let's take a look at how can we do this with "Factory" method?

```
app.factory("FooBarService",["foo" ,function(foo){
   return new FooBar(foo);
}]);
```

So in this typical way we will create its new instance and return it. Also we have injected "foo" as a dependency in factory service.

So let us see how we can achieve this via **service** type:

```
app.service("FooBarService",["foo", FooBar]);
```

So we have called service method on module and provided name of the service, and provided the dependency and name of function in array.

So we have looked at the use of **service** type. Although it is also not too flexible but can be used in these types of scenarios.

4. AngularJS Provider

The next type of service we have is **provider**. It is the parent of all the service types in AngularJS except the constant which we will discuss next after provider. It is the core service type, and other services are built on top of it. It is a little complex but more configurable. It is defined as a custom and must implement $get method. The main use of provider service is when we want to expose API for application-wide configuration which must be configured before starting the application.

Let's take a look at the basic example:

```
app.provider('authentication', function() {

   var username = "John";

   return {

       set: function(newUserName) {
           username = newUserName;
       },

       $get: function() {
           function getUserName() {
               return username;
           }

           return {
               getUserName: getUserName
           };
       }

   };

});
```

In this example we have initialized a provider with name authentication. And it also implements a **$get** function, which returns a method **getUsername** which in turns return the private variable called username. This also has a setter we can set the username on application startup as follows:

```
app.config(["authenticationProvider", function(authenticationProvider) {
   authenticationProvider.set("David");
}]);
```

Notice that we have named "authenticationProvider" in the config method but our provider name was "authentication". So while configuring the provider we need to put "Provider" as a suffix in provider name. This is an indication to Angular we want to configure the provided provider. And then we have called the set method which we have wrote and assigned a new user name.

5. AngularJS Constant

As it's name depicts it provides a way for declaring constants in our application and we can use them anywhere needed by just adding it as a dependency. There are many areas of application where we want to use constants like some base urls, application name, some configurations, etc. So we just define them once and use them anywhere we need. This technique lets us define at one place, so that a later change the value doesn't have us change on all places, we simply just change the value of constant. So here is how we can define or create constants:

```
app.constant('applicationName', 'Service Tutorials');
```

So it is just as simple in creation; as for using we just add dependency where we want to use it.

191.How to establish the communication between two directives ?
(i)One of the ways to establish a communication between two directives is :By requiring a controller.

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

app.directive("server", function() {
  return {
    controller: function() {
      this.log = function(message) {
        console.log(message);
      };
    }
  };
});

app.directive("client", function() {
  return {
    require: "server",
    link: function($scope, $elem, $attrs, serverController) {
      serverController.log("Hello, this is client!");
    }
  };
});
```

-or-

This is very similar to the previous example. The only part we need to change is to tell Angular to look for the required directive among parent elements by prepending the require value with the ^ character.

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

app.directive("server", function() {
  return {
    controller: function() {
      this.log = function(message) {
        console.log(message);
      };
    }
  };
});

app.directive("client", function() {
  return {
    require: "^server",
    link: function($scope, $elem, $attrs, serverCtrl) {
```

```
    serverCtrl.log("Hello, this is the client!");
   }
  };
});
```

Registering a child controller :

Since we can't require a child element, we have to pass it to the parent directive. In the following example the parent's controller exposes a register function which accepts the child's controller as a parameter. To use it, the child directive must require the parent in the usual way (by require: "^client") and provide its controller in the link function. However, by default, the directive controller is not accessible from its link function. We can either require it (which IMHO feels a bit weird – to require a directive by itself) or use the controllerAs field to expose it in the scope.

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

app.directive("server", function() {
  return {
    // It is perfectly valid for the directive to require itself.
    // This allows to use the controller in the linking function.
    require: ["server", "^client"],
    controller: function() {
      this.log = function(message) {
        console.log(message);
      };
    },
    link: function($scope, $elem, $attrs, controllers) {
      // The require field contains an array, so the fourth parameter
      // of the linking function is also an array.
      var serverController = controllers[0];
      var clientController = controllers[1];
      clientController.register(serverController);
    }
  };
});

app.directive("client", function() {
  return {
    controller: function($scope) {
      this.register = function(serverController) {
        $scope.serverController = serverController;
      };
    },
    link: function($scope) {
      $scope.serverController.log("Hello, this is client!");
    }
  };
});
```

The above can also be written as :The serverDirective part using controllerAs syntax:

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

```
app.directive("server", function() {
  return {
    require: "^client",
    controller: function() {
      this.log = function(message) {
        console.log(message);
      };
    },
    // controllerAs publishes the controller in the scope
    controllerAs: "controller",
    link: function($scope, $elem, $attrs, clientController) {
      clientController.register($scope.controller);
    }
  };
});


app.directive("client", function() {
  return {
    controller: function($scope) {
      this.register = function(serverController) {
        $scope.serverController = serverController;
      };
    },
    link: function($scope) {
      $scope.serverController.log("Hello, this is client!");
    }
  };
});
```

(ii)The second way to establish a communication between two directives is : By using events

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

app.directive("server", function() {
  return {
    link: function($scope, $elem, $attrs) {
      $scope.$on("message", function (e, msg) {
        console.log(msg);
      });
    }
  };
});

app.directive("client", function() {
  return {
    link: function($scope, $elem, $attrs) {
      $scope.$broadcast("message", "Hello, this is the client!");
    }
  };
});
```

-or-

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

app.directive("server", function() {
  return {
    compile: function() {
      return {
        pre: function($scope, $elem, $attrs) {
          $scope.$on("message", function (e, msg) {
            console.log(msg);
          });
        }
      };
    }
  };
});

app.directive("client", function() {
  return {
    link: function($scope, $elem, $attrs) {
      $scope.$emit("message", "Hello, this is the client!");
    }
  };
});
```

The safest way to go in this case would be to $emit an event from the client directive. $broadcast could also work if the client did not create its own scope, but $emit will always be a correct solution.
Note that in this example I'm using the compile function returning the pre-link function instead of more common post-link function. This is because the client's post-link would fire before the server's one and emit an event which would not be caught. Server's pre-link, on the other hand, will be executed before client's post-link allowing the event to be handled appropriately.

If both directives create new scopes, neither $emit nor $broadcast will work ($emit goes up the scope hierarchy, $broadcast goes down; there's nothing that goes sideways ;)). Fortunately, we can utilize their common parent to broadcast an event. Accessing the parent scope is possible through the scope's $parent field:

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

app.directive("server", function() {
  return {
    scope: true,
    link: function($scope) {
      $scope.$on("message", function(e, msg) {
        console.log(msg);
      });
    }
  };
});

app.directive("client", function() {
```

```
  return {
    scope: true,
    link: function($scope) {
      $scope.$parent.$broadcast("message", "Hello, this is client!");
    }
  };
});
```

(iii)The third way to establish a communication between two directives is : By sharing attributes

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

app.directive("server", function() {
  return {
    link: function($scope, $elem, $attrs) {
      $attrs.$observe("message", function(message) {
        if (message) {
          console.log(message);
        }
      });
    }
  };
});

app.directive("client", function() {
  return {
    link: function($scope, $elem, $attrs) {
      $attrs.message = "Hello, this is the client!";
    }
  };
});
```

192.How to inspect a scope of any element in an Angular JS application or How to access the scope of an element in an Angular JS application ?

```
angular.element($0).scope( );
```

-or-

```
$($0).scope();
```

193.How to inspect an isolated directive scope of any element in an Angular JS application or How to access the isolated directive scope of an element in an Angular JS application ?

```
angular.element($0).isolatedScope( );
```

-or-

```
$($0).isolatedScope( );
```

194.what is $cacheFactory ?

The default $http cache can be particularly useful when our data doesn't change very often.

Here's how it would all work if our application has routing and AJAX calls, and we want to cache the AJAX calls to speed up the routes.

```
// Define an angular module for our app
var myApp = angular.module('myApp', ['ngRoute']);

// Define Routing for app
myApp.config(['$routeProvider',
  function($routeProvider) {
    $routeProvider.
      when('/main', {
        templateUrl: 'main.html',
        controller: 'MyMain'
      }).
      otherwise({
        redirectTo: '/main'
      });
    }
]);

// Caching the river...
myApp.factory('myCache', function($cacheFactory) {
 return $cacheFactory('myData');
});

// Displays a list of articles in the river
myApp.controller('MyMain', ['$scope', '$http', 'myCache',
 function ($scope, $http, myCache) {
   var cache = myCache.get('myData');
   if (cache) {
     $scope.variable = cache;
   }
   else {
     $http.get('http://www.example.com/path/to/api/endpoint')
       .success(function(data) {
         $scope.variable = data;

         myCache.put('myData', data);
       }
    );
  }
}

]);
```

This is a very elaborate version of the script that allows you finer control over your cache. However, you can easily eliminate most of the markup in the controller and just use $cacheFactory as an optional parameter in your AJAX request.

```
// Displays a list of articles in the river
myApp.controller('MyMain', ['$scope', '$http', 'myCache',
 function ($scope, $http, myCache) {
     $http.get('http://www.example.com/path/to/api/endpoint', {cache: myCache})
       .success(function(data) {
         $scope.variable = data;
       }
    );
  }
 }

]);
```

195.what to Cache ?
Individual items are easy to cache.
Collections are harder to cache,especially those that are the result of some query.You can cache Collections that are the result of some query but you may end up busting your cache often.

196.why Cache ?
To reduce the latency from 200ms to 1ms.
To load pages faster.
To eliminate extra AJAX requests.
To ease load on server.

197.what is the disadvantage of $cacheFactory ?
What if you want to cache data to improve user experience, but the data is liable to change once a day, or every few hours, etc. You would want to make sure your cached data gets cleared at least once a day, every ten minutes, etc. Unfortunately Angular's $cacheFactory does not provide any such capabilities.
You could hack something together using setInterval() or setTimeout(), but you don't want to do that. To solve this problem I created angular-cache, a drop-in Angular module that gives you access to $angularCacheFactory, which creates caches with more capabilities. With angular-cache your caches can periodically clear themselves with cacheFlushInterval. When adding to a cache you can specify a maxAge, which is the maximum amount of time that particular item will be in the cache before it is automatically removed. Or you can specify a maxAge for the whole cache which will be applied to every item added to the cache.

Create a cache

First, inject `$angularCacheFactory` then create as many caches as you so desire. Let's go:

```
app.service('myService', function ($angularCacheFactory) {

    // Create a new cache called "profileCache"
    var profileCache = $angularCacheFactory('profileCache');
});
```

Let's add some items to the cache:

```
profileCache.put('/profiles/34', {
    name: 'John',
    skills: ['programming', 'piano']
});

profileCache.put('/profiles/22', {
    name: 'Sally',
    skills: ['marketing', 'climbing', 'painting']
});
```

Right now, these items will stay in the cache indefinitely. What if, for the sake of the example, I know that users change their skills in their profile once a week. The profile information is subject to change, but we don't want to make tons of requests for profiles that probably haven't changed.

Let's have items which are added to `profileCache` expire after an hour:

```
profileCache.setOptions({
    maxAge: 3600000
});
```

Perfect. Say we also want the items removed from the cache when they expire:

```
profileCache.setOptions({
    deleteOnExpire: 'aggressive'
});
```

Let's say that when the items do expire, we want to refresh them with new values:

```
profileCache.setOptions({
    onExpire: function (key, value) {
        $http.get(key).success(function (data) {
            profileCache.put(key, data);
        });
    }
});
```

Sweet! Now we'd probably have configured our cache correctly when we created it:

```
var profileCache = $angularCacheFactory('profileCache', {
    maxAge: 3600000,
    deleteOnExpire: 'aggressive',
    onExpire: function (key, value) {
        $http.get(key).success(function (data) {
            profileCache.put(key, data);
        });
    }
});
```

Or say we want all of our caches to use that configuration as their default:

```
angular.module('app', ['jmdobry.angular-cache'])
    .config(function ($angularCacheFactoryProvider) {

        $angularCacheFactoryProvider.setCacheDefaults({
            maxAge: 3600000,
                deleteOnExpire: 'aggressive',
                onExpire: function (key, value) {
                    $http.get(key).success(function (data) {
                        profileCache.put(key, data);
                    });
                }
        });
    });
```

Working with a cache

We can retrieve items from a cache like so:

```
var profile = profileCache.get('/profiles/34');

profile.name; // 'John'
```

And get information about items in the cache:

```
var info = profileCache.info('/profiles/34');

info.isExpired; // false
// etc.
```

and information about the cache itself:

```
var info = profileCache.info();

info.size; // 2
info.maxAge; // 3600000
info.deleteOnExpire; // 'aggressive'
// etc.
```

Items are easily removed, and we can destroy out cache when we're done with it:

```
profileCache.remove('/profiles/34');

profileCache.get('/profiles/34'); // undefined

profileCache.destroy();

$angularCacheFactory.get('profileCache'); // undefined
```

See the API Documentation for more information on the available methods.

How to combine promises in AngularJS ?

```
$http({method: 'GET', url: '/api-one-url', cache: 'true'})
.then(function(apiOneData){ $http({method: 'GET', url: '/api-two-url', cache: 'true'})
.then(function(apiTwoData){ console.log(apiOneData, apiTwoData); }); });
```

The disadvantage of this approach is that,the second API will not be called until the first API has been completed.

How to combine promises in AngularJS ?

```
var promise1 = $http({method: 'GET', url: 'a/pi-one-url', cache: 'true'});
var promise2 = $http({method: 'GET', url: '/api-two-url', cache: 'true'});
$q.all([promise1, promise2]).then(function(data){ console.log(data[0], data[1]); });
```

To use the $q.all() method we simply pass it an array of promises we want to resolve, what this does is it takes the array of promises and returns us a single promise which will resolve once all the original promises have been resolved. The data passed to our .then() method when using $q.all() is an array of the returned values in the same order that we passed our original promises.

```
function giftControler ($scope, $http, $q) {
  var names = $http.get("names.json"),
     naughty = $http.get("naughty.json"),
     nice = $http.get("nice.json");
  $q.all([names, naughty,nice]).then(function(arrayOfResults) {
     ... This callback would be called when all promises would be resolved
    });
```

AngularJS Best Practices

What are AngularJS best practices ?
(i)Limit the number and size of watched instances:
we can limit the number and size of watched instances by minimizing the use of ng-repeat.
we can minimize the use of ng-repeat by :
retrieving only the number of records needed to be displayed at one time by using infinite scrolling or paging.
retrieving only the properties of a model needed for display.

(ii)Limit the use of directional binding
(iii)bind without a watch when possible
we can bind without a watch when possible by using bind-once syntax,that is,use bind-once syntax if value doesn’t need to change.
And, to use bind-once,simply place :: before a variable.bind-once works only in expressions.

(iv)End watches after work is done
we can end watches after work is done by calling the return value of $watch

(v)Consolidate watches
we can consolidate watches by :
using $watchGroup when two or more variables trigger the same watch.
using ng-switch / ng-switch-when instead of ng-show / ng-hide when same evaluation performed over and over.

(vi)Avoid unnecessary watch callbacks

(vii)Minimize work performed in watch callbacks
we can minimize work performed in watch callbacks by :
using Track by, so that the resolution can occur based on an identifier rather than the entire object.
using filters sparingly,
exiting filter methods early when there is no work to be done.
using $filter provider when more efficient to bind from Javascript.
not binding to functions especially in a repeater.Instead assigning results of a function to a variable and bind to that variable.

(viii)Limit triggers to the digest
we can limit triggers to the digest by :
using ngModelOptions. ngModelOptions reduces cycles caused by user input.
using "useApplyAsync" on http requests.

(ix)Avoid direct DOM manipulation (DOM Manipulation affects the performance).By design,AngularJS prevents the need for direct DOM manipulation

(x)Make scope light and independent
we can make scope light and independent by :
storing only what you need on $scope.
isolating scopes.
avoiding the use of $rootScope except when absolutely necessary.

(xi)Leverage caching,that is,when caching is enabled,http GET responses are saved to Angular's $cachefactory service as key-pair values.

(xii)Avoid repetitive work
we can avoid repetitive work by :
using compile method.Compile method occurs per declaration of a directive.
Compile method is a great place to put logic that doesn't need to be performed for every instance in a repeater.For an instance,validation and setting default values.

(xiii)Turn off Angular's debugging when outside of development environment.
(xiv)Use template cache.By default,Angular has built-in template cache that prevents multiple requests for the same HTML partial within a single page lifecycle.

198.what is the difference between .success()and .then() ?
Both .then( )and .success( )are used to register the callbacks and the major difference between .then() and .success() is that

.then() returns a promise whereas .success() doesn't return a promise.

.then( ) drives sequential operations,since each call returns a new promise.
$http.get('url').
then(function seqFun1(response){}).
then(function seqFunc2(response){});
The execution behind the scenes will be :
(i)First, $http.get() will be executed
(ii)Second,seqFunc1() will be executed
(iii)Third,seqFunc2() will be executed

.success() drives parallel operations,since handlers are chained on the same promise.
$http.get('url').
success(function parFunc1(data){}).
success(function parFunc2(data){});
The execution behind the scenes will be :
(i)First, $http.get() will be executed
(ii)Second, parFunc1(), parFunc2() will be executed in parallel

The difference between .then() and .success() :
Using .then() :

```
$http.get('/someURL')
.then(function(response) {
    // success handler
}, function(response) {
    // error handler
})
.then(function(response) {
    // success handler
}, function(response) {
    // error handler
})
.then(function(response) {
    // success handler
}, function(response) {
    // error handler
}).
```

Using .success() :

```
$http.get('/someURL')
.success(function(data, status, header, config) {
    // success handler
})
.error(function(data, status, header, config) {
    // error handler
});
```

If you chain .then(), the callbacks will run sequentially after each one is done, because it returns a new promise object on each chain.

If you chain .success() calls, the callbacks will run in parallel, because it returns the original promise object.

How to combine promises in AngularJS ?

```
$http({method: 'GET', url: '/api-one-url', cache: 'true'})
.then(function(apiOneData){ $http({method: 'GET', url: '/api-two-url', cache: 'true'})
.then(function(apiTwoData){ console.log(apiOneData, apiTwoData); }); });
```

The disadvantage of this approach is that,the second API will not be called until the first API has been completed.

How to combine promises in AngularJS ?

```
var promise1 = $http({method: 'GET', url: 'a/pi-one-url', cache: 'true'});
var promise2 = $http({method: 'GET', url: '/api-two-url', cache: 'true'});
$q.all([promise1, promise2]).then(function(data){ console.log(data[0], data[1]); });
```

To use the $q.all() method we simply pass it an array of promises we want to resolve, what this does is it takes the array of promises and returns us a single promise which will resolve once all the original promises have been resolved. The data passed to our .then() method when using $q.all() is an array of the returned values in the same order that we passed our original promises.

```
function giftControler ($scope, $http, $q) {
  var names = $http.get("names.json"),
      naughty = $http.get("naughty.json"),
      nice = $http.get("nice.json");
  $q.all([names, naughty,nice]).then(function(arrayOfResults) {
      ... This callback would be called when all promises would be resolved
    });
```

$watch()

The $scope.watch() function is used to observe changes in a variable on the $scope. It accepts three parameters : expression, listener and equality object where listener and equality object are optional parameters.

```
<!DOCTYPE html>
<html>
<head>
 <title>AngularJS Watch</title>
 <script src="lib/angular.js"></script>
 <script>
 var myapp = angular.module("myapp", []);
 var myController = myapp.controller("myController", function ($scope) {
 $scope.name = 'dotnet-tricks.com';
 $scope.counter = 0;
 $scope.$watch('name', function (newValue, oldValue) {
 $scope.counter = $scope.counter + 1;
 });
 });
 </script>

</head>
<body ng-app="myapp" ng-controller="myController">
 <input type="text" ng-model="name" />
 <br />
 Counter:
</body>
</html>
```

Angular Directives

AngularJS : A simple directive example

```
<div ng-app="app">
 <input type="text" ng-model="name">
 <div></div>
</div>
```

Invoking directive

AngularJS provides you four ways to invoke a directive from HTML. These are all equivalent in AngularJS.

01. Directive as an attribute

```
<div ng-directive></div>
```

02. Directive as a CSS class

```
<div class="ng-directive: expression;"></div>
```

03. Directive as an element

```
<ng-directive></ng-directive>
```

04. Directive as a comment

```
<!-- directive: ng-directive expression -->
```

Invoking directive with name prefixes

You can also invoke a directive with name prefix like ng-,ng:,ng_,x-ng-,data-ng-. These are all equivalent in AngularJS.

```
<input type="text" ng-model="directivename"/>
<div ng-bind="directivename"></div>
<div ng:bind="directivename"></div>
<div ng_bind="directivename"></div>
<div x-ng-bind="directivename"></div>
<div data-ng-bind="directivename"></div>
```

Note

> Use the data-prefixed version (e.g. `data-ng-bind for ngBind`) of directive, if you want to validate your HTML controls value.

Creating a custom directive

AngularJS also allow you to create your own directives based on your requirements. There is no built-in directive for comparing password in angular. So you can create your own custom directive for comparing password.

Creating Custom Directive Syntax

```
<script>
//creating custom directive syntax
myApp.directive("myDir", function () {
 return {
 restrict: "E", //define directive type like E = element, A = attribute, C = class, M = comment
 scope: { //create a new child scope or an isolate scope

 //@ reads the attribute value,
 //= provides two-way binding,
 //& works with functions

 title: '@'
 },
 template: "<div></div>",// define HTML markup
 templateUrl: 'mytemplate.html', //path to the template, used by the directive
 replace: true | false, // replace original markup with template yes/no
 transclude: true | false, // copy original HTML content yes/no
 controller: function (scope) { //define controller, associated with the directive template
 //TODO:
 },
 link: function (scope, element, attrs, controller) {//define function, used for DOM manipulation
 //TODO:
 }
 }
 });
</script>
```

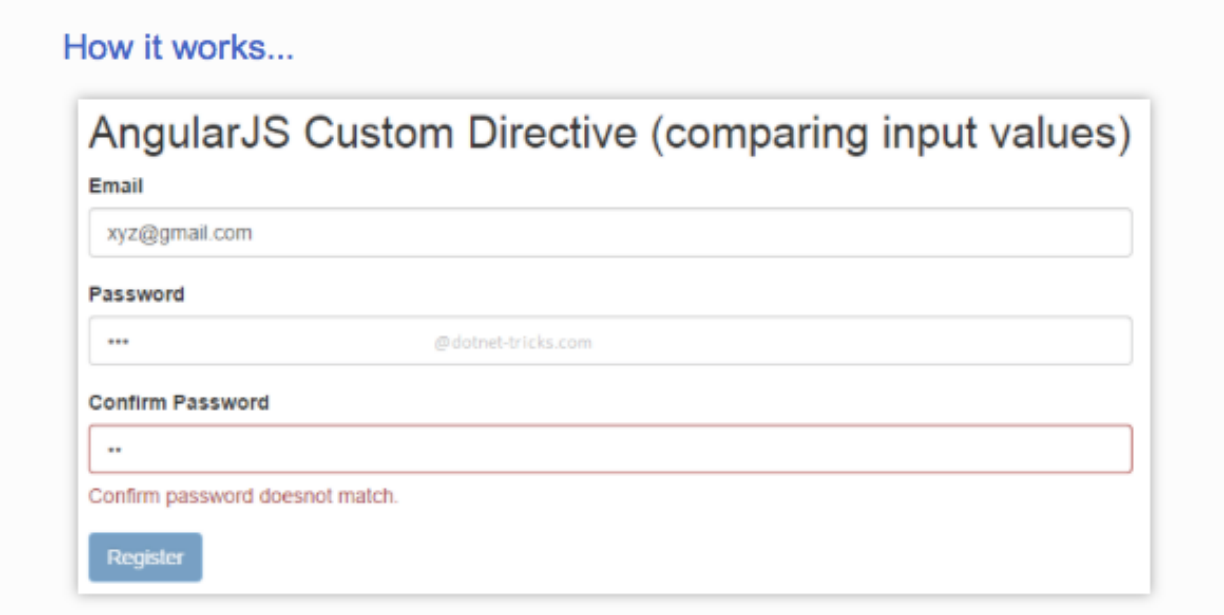
Creating custom directive for comparing textbox values

```
<script>
 //defining module
 var myapp = angular.module('myApp', []);

 //creating custom directive
 myapp.directive('ngCompare', [function () {
 return {
 require: 'ngModel',
 link: function (scope, elem, attrs, ctrl) {
 var firstfield = "#" + attrs.ngCompare;

 //second field key up
 elem.on('keyup', function () {
 scope.$apply(function () {
 var isMatch = elem.val() === $(firstfield).val();
 ctrl.$setValidity('valueMatch', isMatch);
 });
 });

 //first field key up
 $(firstfield).on('keyup', function () {
 scope.$apply(function () {
 var isMatch = elem.val() === $(firstfield).val();
 ctrl.$setValidity('valueMatch', isMatch);
 });
 });
 }
 }
 }]);
</script>
```

AngularJS : Creating custom directive with example

```
<!DOCTYPE html>
<html>
<head>
 <title>AngularJS Custom Directive</title>
 <link href="css/bootstrap.css" rel="stylesheet" />
 <link href="css/bootstrap-theme.css" rel="stylesheet" />
 <script src="lib/jquery-1.11.1.js"></script>
 <script src="lib/angular.js"></script>
</head>
<body>
 <div ng-app="myApp" ng-controller="mainController">
 <br />
 <div class="container">
 <form name="userForm" ng-submit="submitForm()" novalidate>
 <h2>AngularJS Custom Directive (comparing input values)</h2>

 <!-- EMAIL -->
 <div class="form-group" ng-class="{ 'has-error' : userForm.email.$invalid}">
 <label>Email</label>
 <input type="email" name="email" class="form-control" ng-model="user.email" pla
ceholder="Your Email Address" ng-required="true">
 <p ng-show="userForm.email.$error.required" class="help-block">Email is require
d.</p>
 <p ng-show="userForm.email.$error.email" class="help-block">Enter a valid email
.</p>
 </div>

 <!-- Password -->
 <div class="form-group" ng-class="{ 'has-error' : userForm.password.$invalid}">
 <label>Password</label>
 <input type="Password" name="password" id="password" class="form-control" ng-mo
del="user.password" placeholder="Your Password" ng-required="true">
 <p ng-show="userForm.password.$invalid" class="help-block">Your password is req
uired.</p>
 </div>

 <!-- Confirm Password -->
 <div class="form-group" ng-class="{ 'has-error' : userForm.confirmPassword.$inv
alid}">
 <label>Confirm Password</label>
 <input type="Password" name="confirmPassword" class="form-control" ng-model="us
er.confirmPassword" placeholder="Confirm Your Password" ng-compare="password" ng
-required="true">
 <p ng-show="userForm.confirmPassword.$error.required" class="help-block">Your c
onfirm password is required.</p>
 <p ng-show="userForm.confirmPassword.$error.valueMatch && !userForm.confirmPass
word.$error.required" class="help-block">Confirm password doesnot match.</p>
 </div>

 <button type="submit" class="btn btn-primary" ng-disabled="userForm.$invalid">R
egister</button>
 </form>
 </div>
 </div>

 <script>
```

```
44.
45.  <script>
46.  //defining module
47.  var myapp = angular.module('myApp', []);
48.
49.  //creating custom directive
50.  myapp.directive('ngCompare', [function () {
51.  return {
52.  require: 'ngModel',
53.  link: function (scope, elem, attrs, ctrl) {
54.  var firstfield = "#" + attrs.ngCompare;
55.
56.  //second field key up
57.  elem.on('keyup', function () {
58.  scope.$apply(function () {
59.  var isMatch = elem.val() === $(firstfield).val();
60.  ctrl.$setValidity('valueMatch', isMatch);
61.  });
62.  });
63.
64.  //first field key up
65.  $(firstfield).on('keyup', function () {
66.  scope.$apply(function () {
67.  var isMatch = elem.val() === $(firstfield).val();
68.  ctrl.$setValidity('valueMatch', isMatch);
69.  });
70.  });
71.  }
72.  }
73.  }]);
74.
75.  // create angular controller
76.  myapp.controller('mainController', function ($scope) {
77.
78.  // function to submit the form
79.  $scope.submitForm = function () {
80.
81.  // check to make sure the form is completely valid
82.  if ($scope.userForm.$valid) {
83.  alert('form is submitted');
84.  }
85.  };
86.  });
87.  </script>
88. </body>
89. </html>
```



Form Validations

AngularJS form validation enables you to develop a modern HTML5 form that is interactive and responsive. AngularJS provides you built-in validation directives to to validate form client side. This makes your life pretty easy to handle client-side form validations without adding a lot of extra effort. AngularJS form validation are based on the HTML5 form validators.

AngularJS directives for form validation

Here is a list of angularjs directive which can be apply on a input field to validate it's value.

```
<input type="text"
 ng-model="{ string }"
 [name="{ string }"]
 [ng-required="{ boolean }"]
 [ng-minlength="{ number }"]
 [ng-maxlength="{ number }"]
 [ng-pattern="{ string }"]
 [ng-change="{ string }"]>
</input>
```

The main advantage of using AngularJS form validation instead of HTML5 attributes based validation, is that AngularJS way allows to mantain the two way data binding between model and view.

Custom Validation

AngularJS allows you to crate your own custom validation directives. For example, you want to compare password and confirm password fields. To do this, you have to make a custom directive that will fire whenever the password or confirmpassword changes.

```
<script>
//defining module
var myapp = angular.module('myapp', []);

//creating custom directive
myapp.directive('ngCompare', function () {
 return {
 require: 'ngModel',
 link: function (scope, currentEl, attrs, ctrl) {
 var comparefield = document.getElementsByName(attrs.ngCompare)[0]; //getting fi
rst element
 compareEl = angular.element(comparefield);

 //current field key up
 currentEl.on('keyup', function () {
 if (compareEl.val() != "") {
 var isMatch = currentEl.val() === compareEl.val();
 ctrl.$setValidity('compare', isMatch);
 scope.$digest();
 }
 });

 //Element to compare field key up
 compareEl.on('keyup', function () {
 if (currentEl.val() != "") {
 var isMatch = currentEl.val() === compareEl.val();
 ctrl.$setValidity('compare', isMatch);
 scope.$digest();
 }
 });
 }
 }
});
</script>
```

Angular Form Properties

Angular provides properties on form which help you to get information about a form or it's inputs and to validate them.

01. $valid

It is a boolean property that tells whether the form or it's inputs are valid or not. If all containing form and controls are valid, then it will be true, otherwise it will be false.

Syntax

```
formName.$valid
formName.inputFieldName.$valid
```

02. $invalid

It is a boolean property that tells whether the form or it's inputs are invalid or not. If at least one containing form and control is invalid then it will be true, otherwise it will be false.

Syntax

```
formName.$invalid
formName.inputFieldName.$invalid
```

03. $pristine

It is a boolean property that tells whether the form or it's inputs are unmodified by the user or not. If the form or it's inputs are unmodified by the user, then it will be true, otherwise it will be false.

Syntax

```
formName.inputFieldName.$pristine
```

04. $dirty

It is a boolean property that is actually just reverse of pristine i.e. it tells whether the form or it's inputs are modified by the user or not. If the form or it's inputs are modified by the user, then it will be true, otherwise it will be false.

Syntax

```
formName.$dirty
formName.inputFieldName.$dirty
```

05. $error

This is an object hash which contains references to all invalid controls or forms. It has all errors as keys: where keys are validation tokens (such as required, url or email) and values are arrays of controls or forms that are invalid with given error. For a control, If a validation fails then it will be true, otherwise it will be false.

Syntax

```
formName.$error
formName.inputFieldName.$error
```

Angular Form Methods

Angular provides some methods to change a form state. These are available only for the form, not for individual controls.

01. $addControl()

This method registers a control with the form.

Syntax

```
formName.$addControl()
```

02. $removeControl()

This method remove a control from the form.

Syntax

```
formName.$removeControl()
```

03. $setDirty()

This method set the form to Dirty state.

Syntax

```
formName.$setDirty()
```

04. $setPristine()

This method set the form to its pristine state.

Syntax

```
formName.$setPristine()
```

05. $setValidity()

This method set the validity of a form control.

Syntax

```
formName.$setValidity()
```

Controlling Display of Error Messages

By default, angularjs validation error messages are shown as soon as the page is loaded. This doesn't seems pretty at all. The error messages will be shown, when you will click on submit button or when are interacting with the input. You can achieve this by checking button submit state or by checking form input field $dirty property. So you have to write the code as shown below:

```
<script >
//Other code has been removed for clarity
// function to submit the form after all validation has occurred
 $scope.submitForm = function () {

 // Set the 'submitted' flag to true
 $scope.submitted = true;
 //To DO
 };
});
</script >

 <!-- Control display of error messages for NAME input -->
 <div class="form-group" ng-class="{ 'has-error' : userForm.name.$invalid && (us
erForm.name.$dirty || submitted)}">
 <label>Name</label>
 <input type="text" name="name" class="form-control" ng-model="user.name" placeh
older="Your Name" ng-required="true">
 <p ng-show="userForm.name.$invalid && (userForm.name.$dirty || submitted)" clas
s="help-block">You name is required.</p>
 </div>
```

A simple registration form using Angularjs

index.html

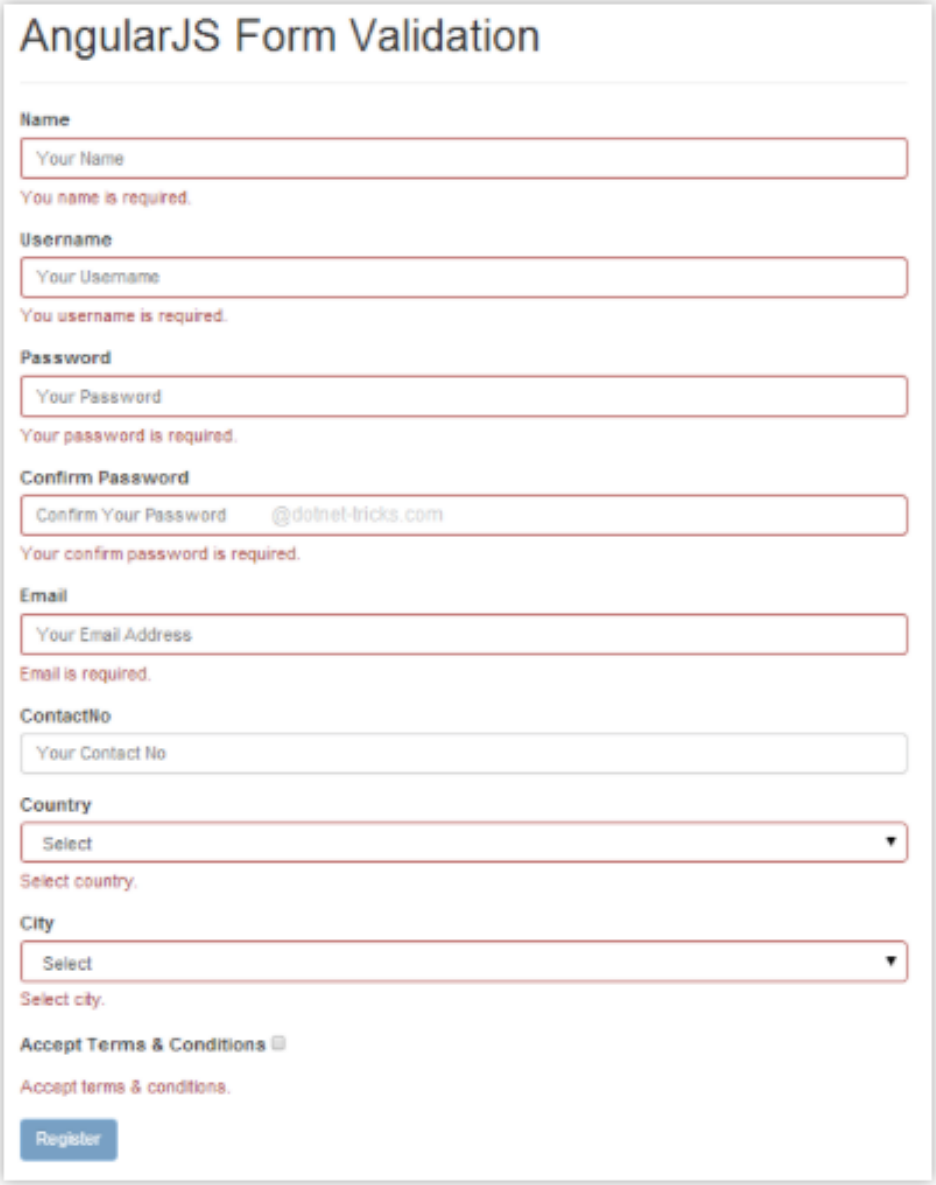
```
<!DOCTYPE html>
<html>
<head>
 <title>AngularJS & Bootstrap Form Validation</title>
 <link href="css/bootstrap.css" rel="stylesheet" />
 <link href="css/bootstrap-theme.css" rel="stylesheet" />
 <script src="lib/angular.js"></script>
<script >
//defining module
var myapp = angular.module('myapp', []);

//creating custom directive
myapp.directive('ngCompare', function () {
 return {
 require: 'ngModel',
 link: function (scope, currentEl, attrs, ctrl) {
 var comparefield = document.getElementsByName(attrs.ngCompare)[0]; //getting fi
rst element
 compareEl = angular.element(comparefield);

 //current field key up
 currentEl.on('keyup', function () {
 if (compareEl.val() != "") {
 var isMatch = currentEl.val() === compareEl.val();
 ctrl.$setValidity('compare', isMatch);
 scope.$digest();
 }
 });

 //Element to compare field key up
 compareEl.on('keyup', function () {
 if (currentEl.val() != "") {
 var isMatch = currentEl.val() === compareEl.val();
 ctrl.$setValidity('compare', isMatch);
 scope.$digest();
 }
 });
 }
 }
});

// create angular controller
myapp.controller('mainController', function ($scope) {

 $scope.countryList = [
 { CountryId: 1, Name: 'India' },
 { CountryId: 2, Name: 'USA' }
 ];

 $scope.cityList = [];

 $scope.$watch('user.country', function (newVal,oldVal) {

```

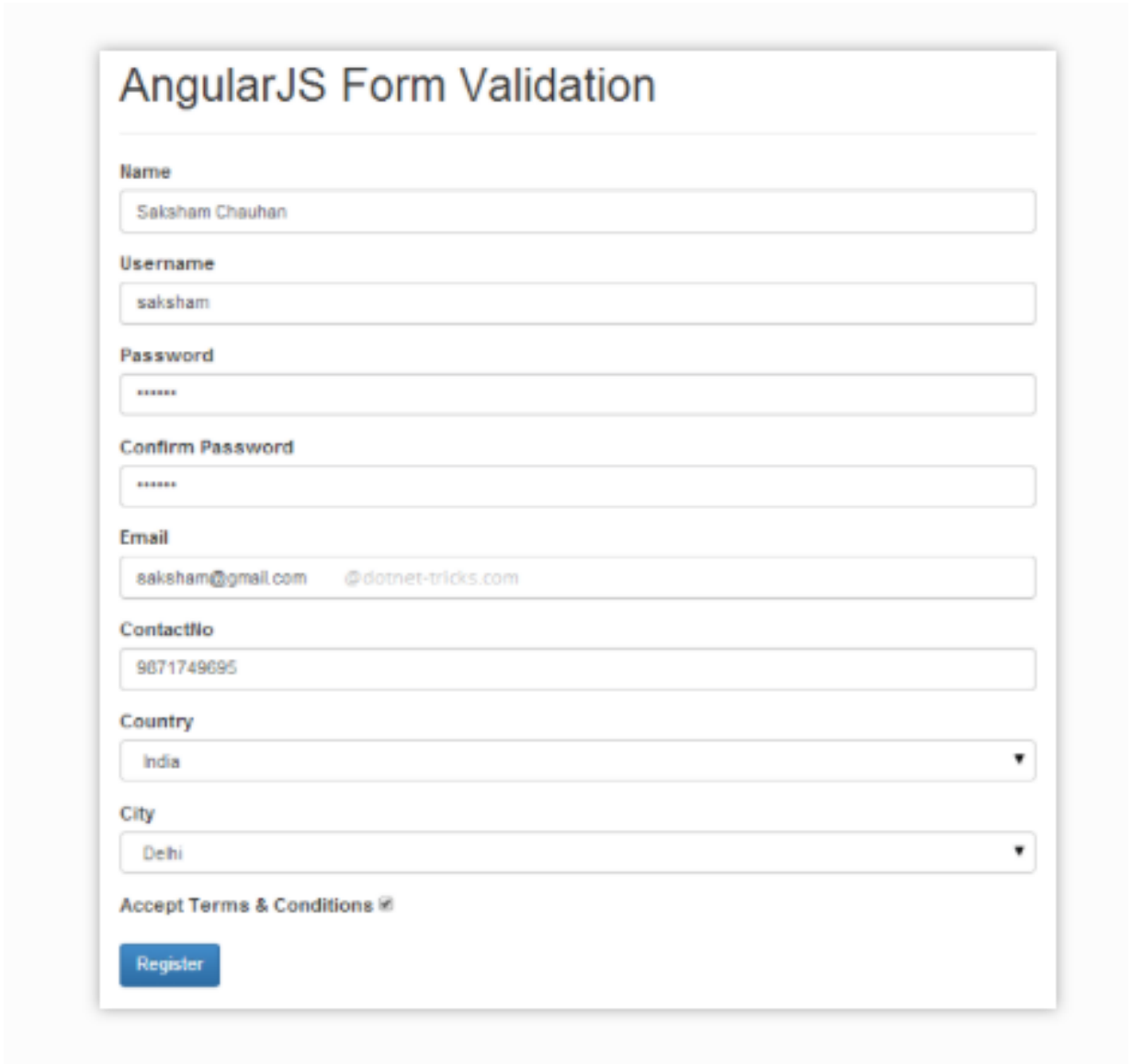
```
50.
51.  $scope.$watch('user.country', function (newVal,oldVal) {
52.
53.  if (newVal == 1)
54.  $scope.cityList = [
55.  { CountryId: 1, CityId: 1, Name: 'Noida' },
56.  { CountryId: 1, CityId: 2, Name: 'Delhi' }];
57.  else if (newVal == 2)
58.  $scope.cityList = [
59.  { CountryId: 2, CityId: 3, Name: 'Texas' },
60.  { CountryId: 2, CityId: 4, Name: 'NewYork' }];
61.  else
62.  $scope.cityList = [];
63.  });
64.
65.  // function to submit the form after all validation has occurred
66.  $scope.submitForm = function () {
67.
68.  // Set the 'submitted' flag to true
69.  $scope.submitted = true;
70.
71.  if ($scope.userForm.$valid) {
72.  alert("Form is valid!");
73.  }
74.  else {
75.  alert("Please correct errors!");
76.  }
77.  };
78. });
79.
80. </script>
81. </head>
82. <body ng-app="myapp" ng-controller="mainController">
83.  <div class="container">
84.  <div class="col-sm-8 col-sm-offset-2">
85.
86.  <!-- PAGE HEADER -->
87.  <div class="page-header"><h1>AngularJS Form Validation</h1></div>
88.
89.  <!-- FORM : YOU CAN DISABLE, HTML5 VALIDATION BY USING "novalidate" ATTRIBUTE--
    >
90.  <form name="userForm" ng-submit="submitForm()" novalidate>
91.
92.  <!-- NAME -->
93.  <div class="form-group" ng-class="{ 'has-error' : userForm.name.$invalid && (us
    erForm.name.$dirty || submitted)}">
94.  <label>Name</label>
95.  <input type="text" name="name" class="form-control" ng-model="user.name" placeh
    older="Your Name" ng-required="true">
96.  <p ng-show="userForm.name.$error.required && (userForm.name.$dirty || submitted
    )" class="help-block">You name is required.</p>
97.  </div>
98.
99.  <!-- USERNAME -->
100.  <div class="form-group" ng-class="{ 'has-error' : userForm.username.$invalid &&
    (userForm.username.$dirty || submitted)}">
101.  <label>Username</label>
102.  <input type="text" name="username" class="form-control" ng-model="user.username
    " placeholder="Your Username" ng-minlength="3" ng-maxlength="8" ng-required="tru
    e">
```

```
100.  <div class="form-group" ng-class="{ 'has-error' : userForm.username.$invalid &&
      (userForm.username.$dirty || submitted)}">
101.  <label>Username</label>
102.  <input type="text" name="username" class="form-control" ng-model="user.username
      " placeholder="Your Username" ng-minlength="3" ng-maxlength="8" ng-required="tru
      e">
103.  <p ng-show="userForm.username.$error.required && (userForm.username.$dirty || s
      ubmitted)" class="help-block">You username is required.</p>
104.  <p ng-show="userForm.username.$error.minlength && (userForm.username.$dirty ||
      submitted)" class="help-block">Username is too short.</p>
105.  <p ng-show="userForm.username.$error.maxlength && (userForm.username.$dirty ||
      submitted)" class="help-block">Username is too long.</p>
106.  </div>
107.
108.  <!-- PASSWORD -->
109.  <div class="form-group" ng-class="{ 'has-error' : userForm.password.$invalid &&
      (userForm.password.$dirty || submitted)}">
110.  <label>Password</label>
111.  <input type="Password" name="password" class="form-control" ng-model="user.pass
      word" placeholder="Your Password" ng-required="true">
112.  <p ng-show="userForm.password.$error.required && (userForm.password.$dirty || s
      ubmitted)" class="help-block">Your password is required.</p>
113.  </div>
114.
115.  <!-- CONFIRM PASSWORD -->
116.  <div class="form-group" ng-class="{ 'has-error' : userForm.confirmPassword.$inv
      alid && (userForm.confirmPassword.$dirty || submitted)}">
117.  <label>Confirm Password</label>
118.  <input type="Password" name="confirmPassword" class="form-control" ng-model="us
      er.confirmPassword" placeholder="Confirm Your Password" ng-compare="password" ng
      -required="true">
119.  <p ng-show="userForm.confirmPassword.$error.required && (userForm.confirmPasswo
      rd.$dirty || submitted)" class="help-block">Your confirm password is required.</
      p>
120.  <p ng-show="userForm.confirmPassword.$error.compare && (userForm.confirmPasswor
      d.$dirty || submitted)" class="help-block">Confirm password doesnot match.</p>
121.  </div>
122.
123.  <!-- EMAIL -->
124.  <div class="form-group" ng-class="{ 'has-error' : userForm.email.$invalid && (u
      serForm.email.$dirty || submitted)}">
125.  <label>Email</label>
126.  <input type="email" name="email" class="form-control" ng-model="user.email" pla
      ceholder="Your Email Address" ng-required="true">
127.  <p ng-show="userForm.email.$error.required && (userForm.email.$dirty || submitt
      ed)" class="help-block">Email is required.</p>
128.  <p ng-show="userForm.email.$error.email && (userForm.email.$dirty || submitted)
      " class="help-block">Enter a valid email.</p>
129.  </div>
130.
131.  <!-- CONTACTNO -->
132.  <div class="form-group" ng-class="{ 'has-error' : userForm.contactno.$invalid &
      & (userForm.contactno.$dirty || submitted) }">
133.  <label>ContactNo</label>
134.  <input type="text" name="contactno" class="form-control" ng-model="user.contact
      no" placeholder="Your Contact No" ng-pattern="/^[789]\d{9}$/" maxlength="10">
135.  <p ng-show="userForm.contactno.$error.pattern && (userForm.contactno.$dirty ||
      submitted)" class="help-block">Enter a valid contactno.</p>
136.  </div>
```

```
 <label>ContactNo</label>
 <input type="text" name="contactno" class="form-control" ng-model="user.contact
no" placeholder="Your Contact No" ng-pattern="/^[789]\d{9}$/" maxlength="10">
 <p ng-show="userForm.contactno.$error.pattern && (userForm.contactno.$dirty ||
submitted)" class="help-block">Enter a valid contactno.</p>
 </div>

 <!-- COUNTRY -->
 <div class="form-group" ng-class="{ 'has-error' : userForm.country.$invalid &&
(userForm.country.$dirty || submitted)}">
 <label>Country</label>
 <select name="country" class="form-control"
 ng-model="user.country"
 ng-options="country.CountryId as country.Name for country in countryList"
 ng-required="true">
 <option value=''>Select</option>
 </select>
 <p ng-show="userForm.country.$error.required && (userForm.country.$dirty || sub
mitted)" class="help-block">Select country.</p>
 </div>

 <!-- CITY -->
 <div class="form-group" ng-class="{ 'has-error' : userForm.city.$invalid && (us
erForm.city.$dirty || submitted)}">
 <label>City</label>
 <select name="city" class="form-control"
 ng-model="user.city"
 ng-options="city.CityId as city.Name for city in citylist"
 ng-required="true">
 <option value=''>Select</option>
 </select>
 <p ng-show="userForm.city.$error.required && (userForm.city.$dirty || submitted
)" class="help-block">Select city.</p>
 </div>

 <!-- TERMS & CONDITIONS -->
 <div class="form-group" ng-class="{ 'has-error' : userForm.terms.$invalid && (u
serForm.terms.$dirty || submitted)}">
 <label>Accept Terms & Conditions</label>
 <input type="checkbox" value="" name="terms" ng-model="user.terms" ng-required=
"true" />
 <p ng-show="userForm.terms.$error.required && (userForm.terms.$dirty || submitt
ed)" class="help-block">Accept terms & conditions.</p>
 </div>

 <!-- ng-disabled FOR ENABLING AND DISABLING SUBMIT BUTTON -->
 <!--<button type="submit" class="btn btn-primary" ng-disabled="userForm.$invali
d">Register</button>-->
 <button type="submit" class="btn btn-primary">Register</button>
 </form>
 </div>
 </div>
 <br />
</body>
</html>
```

How it works..


AngularJS Form Validation
Name
Your Name
You name is required.
Username
Your Username
You username is required.
Password
Your Password
Your password is required.
Confirm Password
Confirm Your Password
@dotnet-tricks.com
Your confirm password is required.
Email
Your Email Address
Email is required.
ContactNo
Your Contact No
Country
Select
Select country.
City
Select
Select city.
Accept Terms & Conditions
Accept terms & conditions.
Register

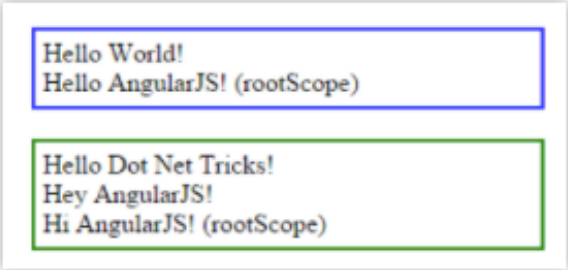

AngularJS Form Validation
Name
Saksham Chauhan
Username
saksham
Password
Confirm Password
Email
saksham@gmail.com
@dotnet-tricks.com
ContactNo
9871749695
Country
India
City
Delhi
Accept Terms & Conditions
Register

$rootScope

AngularJS : $rootScope and $scope with example

```
<!doctype html>
<html>
<body ng-app="myApp">
<div ng-controller="Ctrl1" style="border:2px solid blue; padding:5px">
 Hello !
 <br />
 Hello !
 (rootScope)
</div>
<br />
<div ng-controller="Ctrl2" style="border:2px solid green; padding:5px">
 Hello !
 <br />
 Hey !
 <br />
 Hi ! (rootScope)
</div>
<script src="lib/angular.js"></script>
<script>
 var app = angular.module('myApp', []);

 app.controller('Ctrl1', function ($scope, $rootScope) {
 $scope.msg = 'World';
 $rootScope.name = 'AngularJS';
 });

 app.controller('Ctrl2', function ($scope, $rootScope) {
 $scope.msg = 'Dot Net Tricks';
 $scope.myName = $rootScope.name;
 });
</script>
</body>
</html>
```

How it works...



Note

1. When you use ng-model with $rootScope objects then AngularJS updates those objects under a specific $scope of a controller but not at global level $rootScope.
2. Create a private $scope for each controller to bind it to the view.

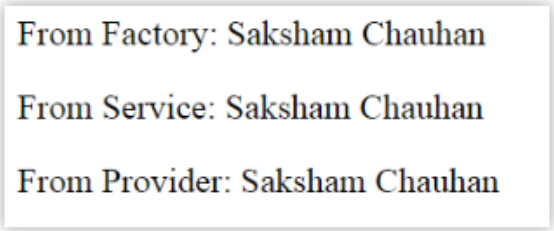
Factory,Service,Provider

AngularJS : Factory, Service and Provider with example

```
<html>
<head>
 <title>AngularJS Service Factory and Providers</title>
 <script src="lib/angular.js"></script>
</head>
<body>
 <div class="container" style="padding-top:20px;">
 <div ng-app="myApp" ng-controller="myController">
 <p>From Service: </p>
 <p>From Factory: </p>
 <p>From Provider: </p>
 </div>
 </div>
 <script>
 //defining module
 var app = angular.module('myApp', []);

 //defining service
 app.service('myService', function () {
 this.name = '';
 this.setName = function (newName) {
 this.name = newName;
 return this.name;
 };
 });

 //defining factory
 app.factory('myFactory', function () {
 var serviceObj = {};
 serviceObj.name = '';
 serviceObj.setName = function (newName) {
 serviceObj.name = newName;
 };
 return serviceObj;
 });

 //defining provider
 app.provider('configurable', function () {
 var privateName = '';
 this.setName = function (newName) {
 privateName = newName;
 };
 this.$get = function () {
 return {
 name: privateName
```

```
 name: privateName
 };
 };
 });

 //configuring provider
 app.config(function (configurableProvider) {
 configurableProvider.setName("Saksham Chauhan");
 });

 //defining controller
 app.controller('myController', function ($scope, myService, myFactory,
configurable) {
 $scope.serviceName = myService.setName("Saksham Chauhan");

 myFactory.setName("Saksham Chauhan");
 $scope.factoryName = myFactory.name;

 $scope.providerName = configurable.name;
 });
 </script>
</body>
</html>
```

How it works...



Angular Templates

In Angular, templates are the views with the HTML enriched by Angular elements like directive and attributes. Templates are used to display the information from the model and controller that a user sees in his browser. An angular templates can have Directive, HTML markup, CSS, Filters, Expressions and Form controls.

A simple Angular Template

```
<html ng-app>
<body ng-controller="MyController">
 <input ng-model="name" value="dotnet-tricks.com">
 <br/>
 Input Value is :
 <button ng-click="changeValue()">Click me!</button>
 <script src="angular.js">
 </body>
</html>
```

Types of Templates

There are two types of templates :

01. Static Templates

A static template is defined by using script tag. It must has an `id` attribute with a unique value and a `type` attribute with value `text/ng-template`. Also, a static template must be inside the scope of the ng-app directive otherwise it will be ignored by Angular.

```
<script type="text/ng-template" id="person.html">
 :
</script>
```

A static template can be rendered by using ng-include directive.

```
<div ng-include="'person.html'"></div>
```

Full Example of Dynamic Template

```
<!DOCTYPE html>
<html>
<head>
 <title>AngularJS Static Templates</title>
 <script src="lib/angular.js"></script>
 <script>
 //defining module
 var app = angular.module('app', []);

 app.controller("myController", function ($scope) {
 $scope.person = { name: "Deepak Chauhan", address: "Delhi" };
 });

 app.controller("homeController", function ($scope) {
 $scope.persons = [{ name: "Deepak Chauhan", address: "Delhi" }, { name:
"Shailendra Chauhan", address: "Noida" }, { name: "Kuldeep Chauhan",
address: "Gurgaon" }]
 });

 </script>

</head>
<body ng-app="app">
 <h1>AngularJS : Static Templates</h1>
 <!--It should be the part of ng-app directive-->
 <script type="text/ng-template" id="person.html">
  :
 </script>

 <div ng-controller="myController">
 <h1>myController</h1>
 <!--Please not the single quotes around person.html. The value of the n
g-include is an expression and person.html is a string value, so put sin
gle quotes around it.-->
 <div ng-include="'person.html'"></div>
 </div>

 <div ng-controller="homeController">
 <h1>homeController</h1>
 <div ng-repeat="person in persons" ng-include="'person.html'"></div>
 </div>
</body>
</html>
```

02. Dynamic Templates

A dynamic template is an html page which is compiled and rendered by Angular on demand. The above static template can be created as a HTML page within templates folder of your app like as:

person.html

```
:
```

A dynamic template can be rendered by using ng-include directive.

```
<div ng-include="'templates/person.html'"></div>
```

Full Example of Dynamic Template

```
<!DOCTYPE html>
<html>
<head>
 <title>AngularJS Dynamic Templates</title>
 <script src="lib/angular.js"></script>
 <script>
 //defining module
 var app = angular.module('app', []);

 app.controller("myController", function ($scope) {
 $scope.person = { name: "Deepak Chauhan", address: "Delhi" };
 });

 app.controller("homeController", function ($scope) {
 $scope.persons = [{ name: "Deepak Chauhan", address: "Delhi" }, { name: "Shailendra Chauhan", address: "Noida" }, { name: "Kuldeep Chauhan", address: "Gurgaon" }]
 });

 </script>

</head>
<body ng-app="app">
 <h1>AngularJS : Dynamic Templates</h1>
 <div ng-controller="myController">
 <h1>myController</h1>
 <!--Please not the single quotes around person.html. The value of the ng-include is an expression and person.html is a string value, so put single quotes around it.-->
 <div ng-include="'templates/person.html'"></div>
 </div>

 <div ng-controller="homeController">
 <h1>homeController</h1>
 <div ng-repeat="person in persons" ng-include="'templates/person.html'"></div>
 </div>
</body>
</html>
```

person.html

```
01.  :
```

Note

> Always put the single quotes around the name of template with ng-include directive, since it accept an expression. Hence to indicate to Angular that template (person.html) is a string value you have to put single quotes around it's name.

Ng-Repeat

The ng-repeat directive has a set of special variables which you are useful while iterating the collection. These variables are as follows:

01. $index
02. $first
03. $middle
04. $last

The $index contains the index of the element being iterated. The $first, $middle and $last returns a boolean value depending on whether the current item is the first, middle or last element in the collection being iterated.

```
<html>
<head>
 <title></title>
 <script src="lib/angular.js"></script>
 <script>
 var app = angular.module("app", []);

 app.controller("ctrl", function ($scope) {
 $scope.friends = [{ name: 'shailendra', gender: 'boy' },
 { name: 'kiran', gender: 'girl' },
 { name: 'deepak', gender: 'boy' },
 { name: 'pooja', gender: 'girl' }];
 });
 </script>
</head>
<body ng-app="app">
 <div ng-controller="ctrl">
 <ul class="example-animate-container">
 <li ng-repeat="friend in friends">
 <div>
 [1]  is a .

 <span ng-if="$first">
 <strong>(first element found)</strong>
 </span>
 <span ng-if="$middle">
 <strong>(middle element found)</strong>
 </span>
 <span ng-if="$last">
 <strong>(last element found)</strong>
 </span>
 </div>
 </li>
 </ul>
 </div>
</body>
</html>
```

How it works..

- [1] shailendra is a boy. **(first element found)**
- [2] kiran is a girl. **(middle element found)**
- [3] deepak is a boy. **(middle element found)**
- [4] pooja is a girl. **(last element found)**

Angular Routing

An Example of Routing

index.html

```
<!DOCTYPE html>
<html>
<head>
 <title>AngularJS Routing</title>
 <script src="lib/angular.js"></script>
 <script src="lib/angular-route.js"></script>

 <script>
 var app = angular.module("app", ['ngRoute']);

 app.controller("homeController", function ($scope) {

 $scope.message = "Welcome to Home Page!";
 });

 app.controller("aboutController", function ($scope) {

 $scope.message = "Welcome to About Page!";
 });

 app.controller("contactController", function ($scope) {

 $scope.message = "Welcome to Contact Page!";
 });

 app.config(['$routeProvider', function ($routeProvider) {
 $routeProvider.when("/", {
 templateUrl: "templates/pages/home.html",
 controller: "homeController"
 }).when("/about", {
 templateUrl: "templates/pages/about.html",
 controller: "aboutController"
 }).when("/contact", {
 templateUrl: "templates/pages/contact.html",
 controller: "contactController"
 }).otherwise({
 redirectTo: 'index.html'
 });
 }]);

 </script>
</head>
<body ng-app="app">
 <div>
 <a href="#">Home</a> |
 <a href="#contact">Contact</a> |
 <a href="#about">About</a>
 </div>
 <div ng-view></div>
</body>
</html>
```

home.html

```
<h1>Home Page</h1>
<h2></h2>
```

about.html

```
<h1>About Page</h1>
<h2></h2>
```

contact.html

```
<h1>Contact Page</h1>
<h2></h2>
```

How to manually bootstrap an AngularJS application ?

```
<html>
<head>
 <title>Multiple modules bootstrap</title>
 <script src="lib/angular.js"></script>
 <script>
 //module1
 var app1 = angular.module("module1", []);
 app1.controller("Controller1", function ($scope) {
 $scope.name = "Shailendra Chauhan";
 });

 //module2
 var app2 = angular.module("module2", []);
 app2.controller("Controller2", function ($scope) {
 $scope.name = "Deepak Chauhan";
 });

 //manual bootstrap process
 angular.element(document).ready(function () {
 var div1 = document.getElementById('div1');
 var div2 = document.getElementById('div2');

 //bootstrap div1 for module1 and module2
 angular.bootstrap(div1, ['module1', 'module2']);

 //bootstrap div2 only for module1
 angular.bootstrap(div2, ['module1']);
 });
 </script>
</head>
<body>
 <!--angularjs autobootstap process-->
 <div id="div1">
 <h1>Multiple modules bootstrap</h1>
 <div ng-controller="Controller1">

 </div>
 <div ng-controller="Controller2">

 </div>
 </div>

 <div id="div2">
 <div ng-controller="Controller1">

 </div>
 </div>
</body>
</html>
```

How to manually bootstrap an AngularJS application ?

```
<html>
<head>
 <title>Multiple modules bootstrap</title>
 <script src="lib/angular.js"></script>
 <script>
 //module1
 var app1 = angular.module("module1", []);
 app1.controller("Controller1", function ($scope) {
 $scope.name = "Shailendra Chauhan";
 });

 //module2
 var app2 = angular.module("module2", []);
 app2.controller("Controller2", function ($scope) {
 $scope.name = "Deepak Chauhan";
 });

 //manual bootstrap process
 angular.element(document).ready(function () {
 var div1 = document.getElementById('div1');
 var div2 = document.getElementById('div2');

 //bootstrap div1 for module1 and module2
 angular.bootstrap(div1, ['module1', 'module2']);

 //bootstrap div2 only for module1
 angular.bootstrap(div2, ['module1']);
 });
 </script>
</head>
<body>
 <!--angularjs autobootstap process-->
 <div id="div1">
 <h1>Multiple modules bootstrap</h1>
 <div ng-controller="Controller1">

 </div>
 <div ng-controller="Controller2">

 </div>
 </div>

 <div id="div2">
 <div ng-controller="Controller1">

 </div>
 </div>
</body>
</html>
```

How to Auto-Bootstrap an AngularJS application ?

```
<html>
<head>
 <title>Multiple modules bootstrap</title>
 <script src="lib/angular.js"></script>
 <script>
 //module1
 var app1 = angular.module("module1", []);
 app1.controller("Controller1", function ($scope) {
 $scope.name = "Shailendra Chauhan";
 });

 //module2
 var app2 = angular.module("module2", []);
 app2.controller("Controller2", function ($scope) {
 $scope.name = "Deepak Chauhan";
 });

 //module3 dependent on module1 & module2
 angular.module("app", ["module1", "module2"]);
 </script>
</head>
<body>
 <!--angularjs autobootstap process-->
 <div ng-app="app">
 <h1>Multiple modules bootstrap</h1>
 <div ng-controller="Controller2">

 </div>
 <div ng-controller="Controller1">

 </div>
 </div>
</body>
</html>
```

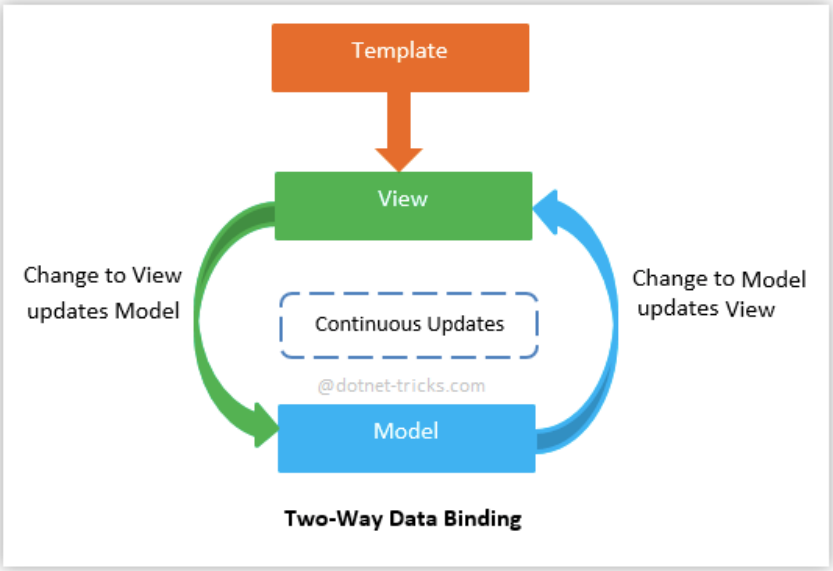
Two-way Data binding

How AngularJS handle data binding

AngularJS handle data-binding mechanism with the help of three powerful functions: $watch(), $digest() and $apply(). Most of the time AngularJS will call the $scope.$watch() and $scope.$digest() functions for you, but in some cases you may have to call these functions yourself to update new values.

Two-way data binding

It is used to synchronize the data between model and view. It means, any change in model will update the view and vice versa. ng-model directive is used for two-way data binding.



Issue with two-way data binding

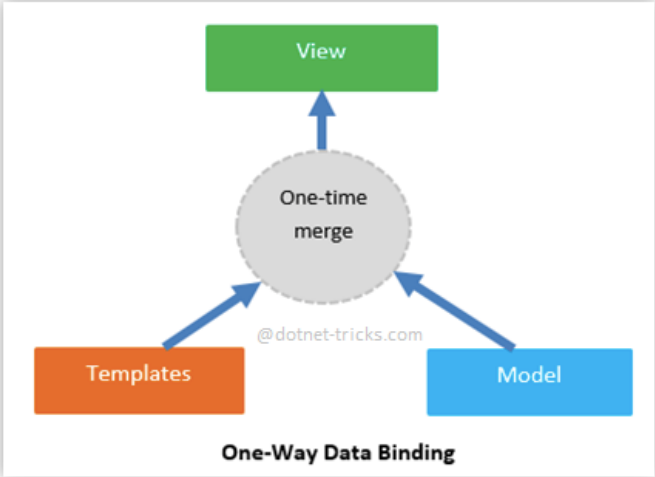
In order to make data-binding possible, Angular uses $watch APIs to observe model changes on the scope. Angular registered watchers for each variable on scope to observe the change in its value. If the value, of variable on scope is changed then the view gets updated automatically.

This automatic change happens because of $digest cycle is triggered. Hence, Angular processes all registered watchers on the current scope and its children and checks for model changes and calls dedicated watch listeners until the model is stabilized and no more listeners are fired. Once the $digest loop finishes the execution, the browser re-renders the DOM and reflects the changes.

By default, every variable on a scope is observed by the angular. In this way, unnecessary variable are also observed by the angular that is time consuming and as a result page is becoming slow. Hence to avoid unnecessary observing of variables on scope object, angular introduced one-way data binding.

One-way data binding

This binding is introduced in `Angular 1.3`. An expression that starts with `double colon (::)`, is considered a one-time expression i.e. one-way binding.



One-Way Data Binding

Two-Way and One-Way data binding Example

```
<div ng-controller="MyCtrl">
 <label>Name (two-way binding): <input type="text" ng-model="name" /></label>
 <strong>Your name (one-way binding):</strong> <br />
 <strong>Your name (normal binding):</strong>
</div>
<script>
 var app = angular.module('app', []);
 app.controller("MyCtrl", function ($scope) {
 $scope.name = "Shailendra Chauhan"
 })
</script>
```

$cookies

This service provides read/write access to browser's cookies. If you want to use existing cookie solution, say read/write cookies from your existing server session system then use $cookie.

```
<script>
var app=angular.module('cookiesExample', ['ngCookies']);

app.controller('ExampleController', ['$cookies', function ($cookies) {
 // Retrieving a cookie
 var favoriteCookie = $cookies.myFavorite;
 // Setting a cookie
 $cookies.myFavorite = 'oatmeal';
}]);
</script>
```

$cookiesStore

$cookieStore is a thin wrapper around $cookies. It provides a key-value (string-object) storage that is backed by session cookies. The objects which are put or retrieved from this storage are automatically serialized or deserialized by angular to JSON and vice versa.

If you are creating a new solution which will persist cookies based on key/value pairs, use $cookieStore.

```
<script>
var app=angular.module('cookieStoreExample', ['ngCookies']);

app.controller('ExampleController', ['$cookieStore', function ($cookieStore) {
 // Put cookie
 $cookieStore.put('myFavorite', 'oatmeal');
 // Get cookie
 var favoriteCookie = $cookieStore.get('myFavorite');
 // Removing a cookie
 $cookieStore.remove('myFavorite');
}]);
</script>
```

ng-if

```
<div ng-controller="MyCtrl">
 <div ng-if="data.isVisible">ng-if Visible</div>
</div>

<script>
 var app = angular.module("app", []);
 app.controller("MyCtrl", function ($scope) {
 $scope.data = {};
 $scope.data.isVisible = true;
 });
</script>
```

ng-switch

```
<div ng-controller="MyCtrl">
 <div ng-switch on="data.case">
 <div ng-switch-when="1">Shown when case is 1</div>
 <div ng-switch-when="2">Shown when case is 2</div>
 <div ng-switch-default>Shown when case is anything else than 1 and 2</div>
 </div>
</div>

<script>
 var app = angular.module("app", []);
 app.controller("MyCtrl", function ($scope) {
 $scope.data = {};
 $scope.data.case = true;
 });
</script>
```

ng-repeat

```
<div ng-controller="MyCtrl">
 <ul>
 <li ng-repeat="name in names">

 </li>
 </ul>
</div>

<script>
 var app = angular.module("app", []);
 app.controller("MyCtrl", function ($scope) {
 $scope.names = ['Shailendra', 'Deepak', 'Kamal'];
 });
</script>
```

$watch

```
<script>
scope.name = 'shailendra';
scope.counter = 0;

scope.$watch('name', function (newVal, oldVal) {
 scope.counter = scope.counter + 1;
});
</script>
```

$watchGroup

```
<script>
$scope.teamScore = 0;
$scope.time = 0;
$scope.$watchGroup(['teamScore', 'time'], function(newVal, oldVal) {
 if(newVal[0] > 20){
 $scope.matchStatus = 'win';
 }
 else if (newVal[1] > 60){
 $scope.matchStatus = 'times up';
 });
</script>
```

$watchCollection

```
<script>
$scope.names = ['shailendra', 'deepak', 'mohit', 'kapil'];
$scope.dataCount = 4;

$scope.$watchCollection('names', function (newVal, oldVal) {
 $scope.dataCount = newVal.length;
});
</script>
```

$broadcast

```
<!DOCTYPE html>
<html>
<head>
 <title>Broadcasting</title>
 <script src="lib/angular.js"></script>
 <script>
 var app = angular.module('app', []);

 app.controller("firstCtrl", function ($scope) {
 $scope.$on('eventName', function (event, args) {
 $scope.message = args.message;
 console.log($scope.message);
 });
 });

 app.controller("secondCtrl", function ($scope) {
 $scope.handleClick = function (msg) {
 $scope.$emit('eventName', { message: msg });
 };
 });

 </script>
</head>
<body ng-app="app">
 <div ng-controller="firstCtrl" style="border:2px solid #E75D5C; padding:5px;">
 <h1>Parent Controller</h1>
 <p>Emit Message : </p>
 <br />
 <div ng-controller="secondCtrl" style="border:2px solid #428bca;padding:5px;">
 <h1>Child Controller</h1>
 <input ng-model="msg">
 <button ng-click="handleClick(msg);">Emit</button>
 </div>
 </div>
</body>
</html>
```

How it works..

$emit

```
<!DOCTYPE html>
<html>
<head>
 <title>Broadcasting</title>
 <script src="lib/angular.js"></script>
 <script>
 var app = angular.module('app', []);

 app.controller("firstCtrl", function ($scope) {
 $scope.handleClick = function (msg) {
 $scope.$broadcast('eventName', { message: msg });
 };

 });

 app.controller("secondCtrl", function ($scope) {
 $scope.$on('eventName', function (event, args) {
 $scope.message = args.message;
 console.log($scope.message);
 });
 });

 </script>
</head>
<body ng-app="app">
 <div ng-controller="firstCtrl" style="border:2px solid #E75D5C; padding:5px;">
 <h1>Parent Controller</h1>
 <input ng-model="msg">
 <button ng-click="handleClick(msg);">Broadcast</button>
 <br /><br />
 <div ng-controller="secondCtrl" style="border:2px solid #428bca;padding:5px;">
 <h1>Child Controller</h1>
 <p>Broadcast Message : </p>
 </div>
 </div>
</body>
</html>
```

How it works..

Note

1. If there is no parent-child relation between your scopes you can inject $rootScope into the controller and broadcast the event to all child scopes but you cannot emit your event.
2. You can emit your event only when you have parent-child relation and event propagation is initiated by child. However, $emit can fire an event only for all $rootScope.$on listeners.