**angular filter -** allow us to format the value of expression before sending to the end user. It can be used in view templates, controller and services.

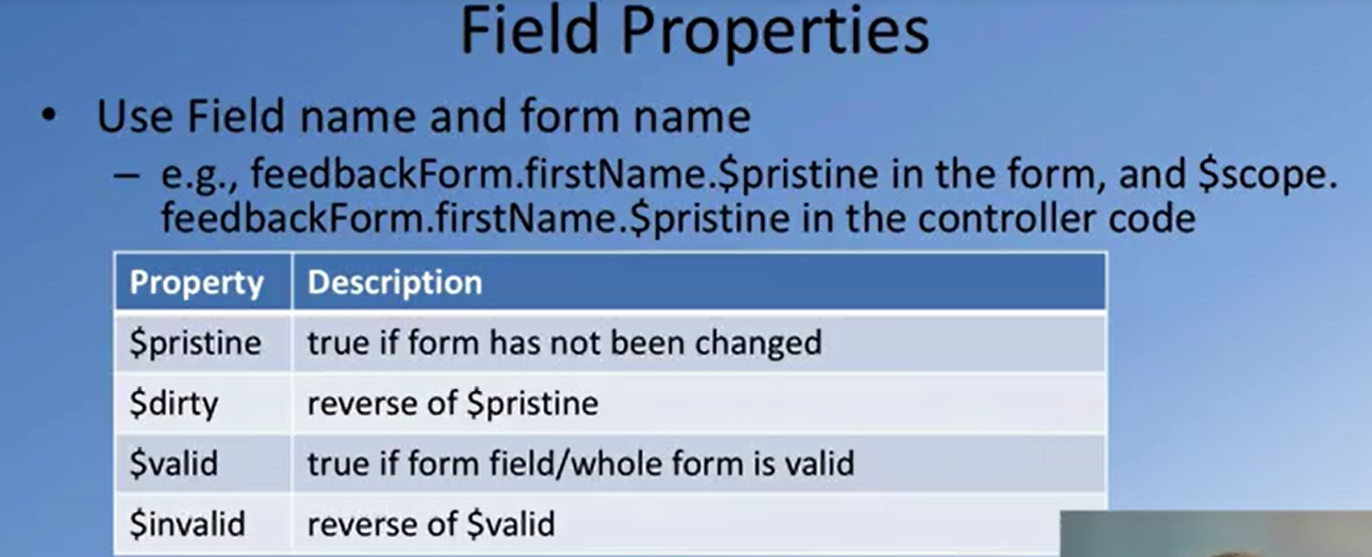
There few filter provided by angular like currency, uppercase, lowercase etc.

We can also defined customer filter at module level or inside controller.

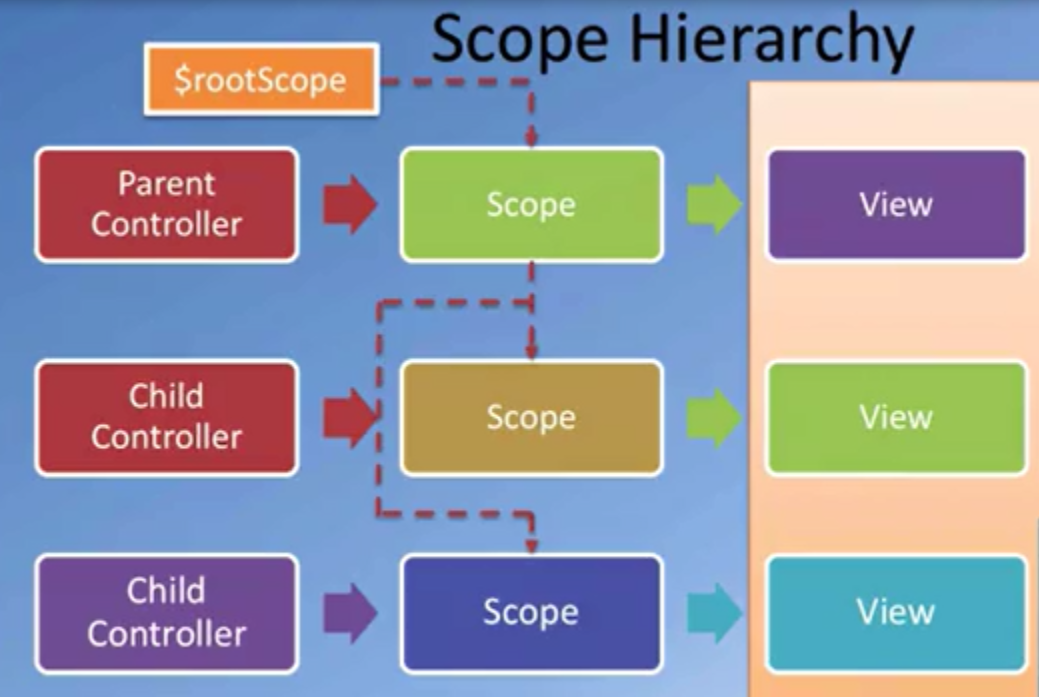
**Form Validation in AngularJS**

HTML5 validates form, to disable HTML5 validation use no-validate attribute in <form> tag.

These properties are available at form level and field level. There is one more property $error is used in validation.







**Angular Service**

* Substitutable object injected using DI.
* allow to share code across the module.
* It lazily instantiated.
* Singleton
* Start with $ sign

**Angular Factory**

* return a javascript object.

**Value Service -** It is like global variable which can be used in directive, controller and services.

Sometimes we need some data that is globally available but at the same time we do not want to pollute the global (window) namespace with the definition for this data.

 We can define any string, number, date-time, array or object as a value.

We can store session data, user information, application details etc.

|  |  |
| --- | --- |
| **Constant** | **Value** |
| constants are read only pieces of data. But when using objects with a constant they can be modified. | values are pieces of data that can change at any time, anywhere. |
| inject constants into module.config | but values cannot be injected into a config function call. |

**Provider:** It is configurable factory. It is allow consumer of the service to set value.

**e.g.** Angular service that can be used to retrieve weather data from a public API. Allow consumer of the service to set API.

**Code from angular.js**

function factory(name, factoryFn, enforce) {

return provider(name, {

$get: enforce !== false ? enforceReturnValue(name, factoryFn) : factoryFn

});

}

function service(name, constructor) {

return factory(name, ['$injector', function($injector) {

return $injector.instantiate(constructor);

}]);

}

function value(name, val) { return factory(name, valueFn(val), false); }

function constant(name, value) {

assertNotHasOwnProperty(name, 'constant');

providerCache[name] = value;

instanceCache[name] = value;

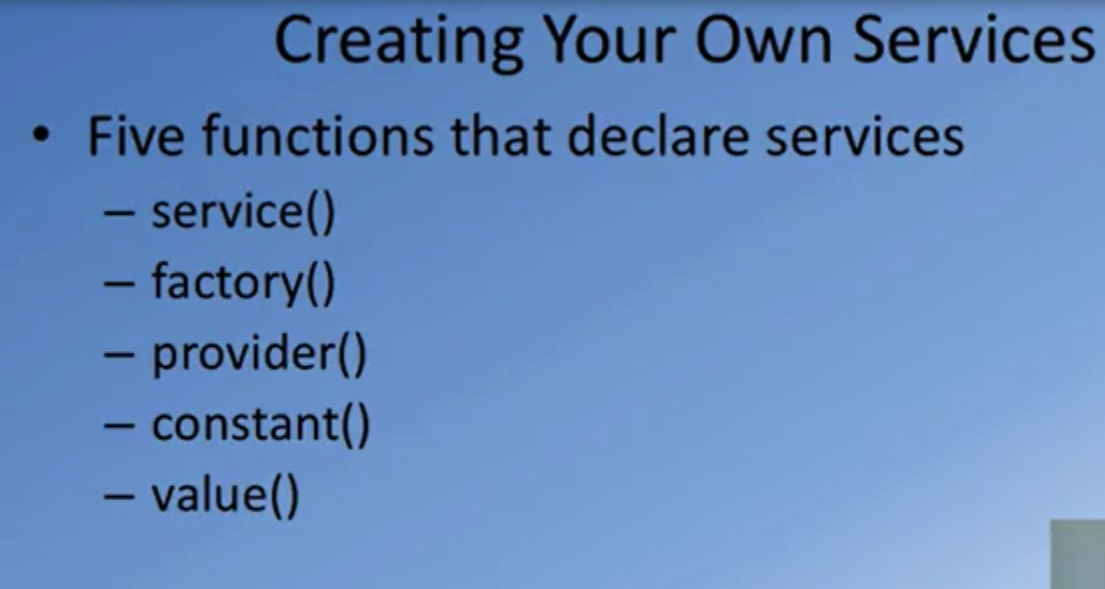
}

**Imp:**When the angular service is injected into controller that time it calls above service function from angular.js file where angular pass service name and function to service function, The service function create object of the passed function then it calls factory function which in turns calls provider function. The provider function create provider for function object and stores it in Provider Cache

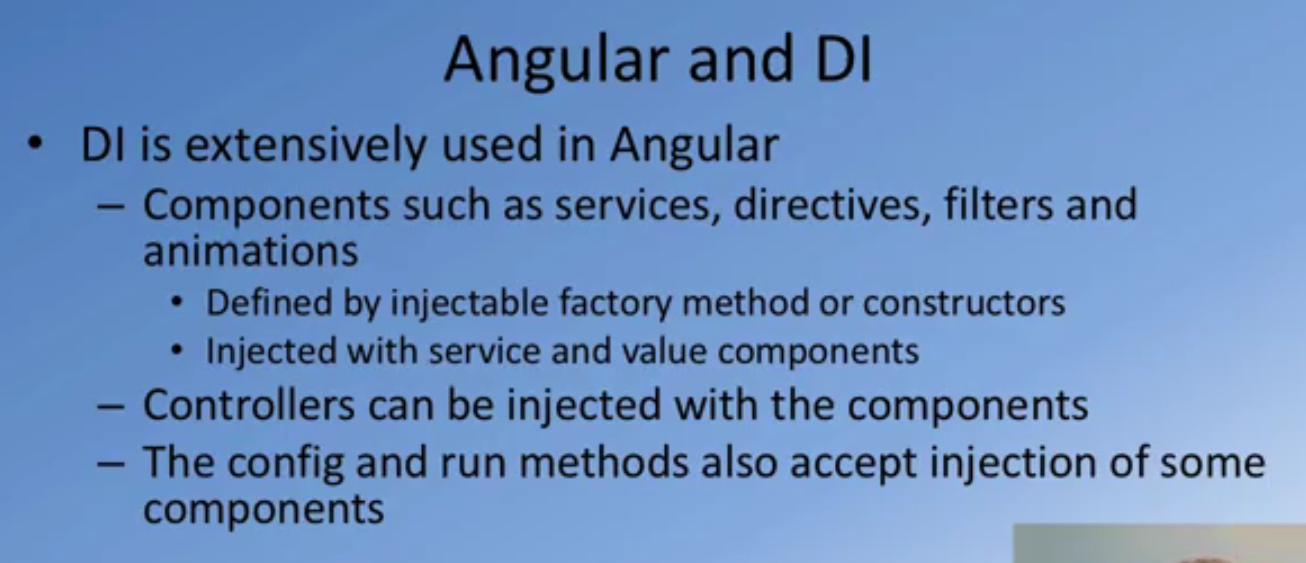
**config function:** The config() function is used to do configuration before instantiating services. It does not accept service, factory and value as parameter because those are instances of provider. It only accept provider and constant. If we want to access service in config function then pass it provider.

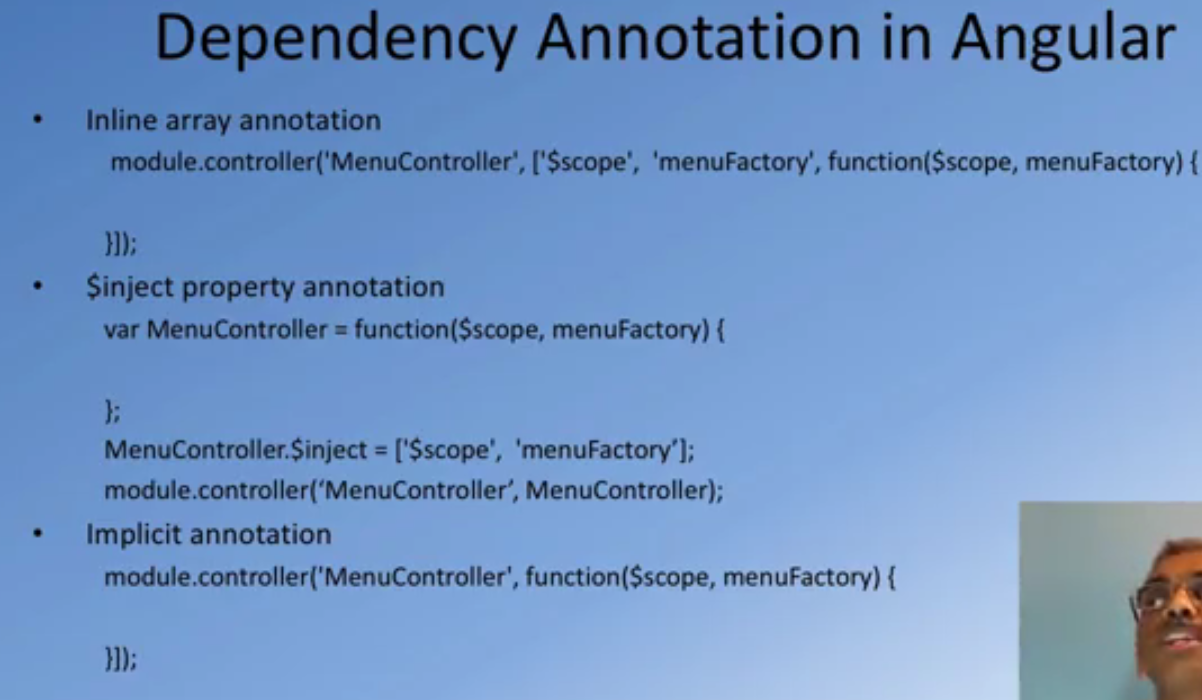
**run() function:** executed after the config() block and here we have facility with this block to inject any instance and constants in our application. This block is just like main() method in other language. This block is a great place to put any event handlers that we need to be executed at the root level for the application. For example, authentication handlers. - See more at: http://firstcrazydeveloper.com/Blogs/BlogView.html/46/importance-of-config-and-run-blocks-in-angularjs#sthash.6UVpFx5G.dpuf

.



**Dependency Injection in Angular**





**Angular Routing**

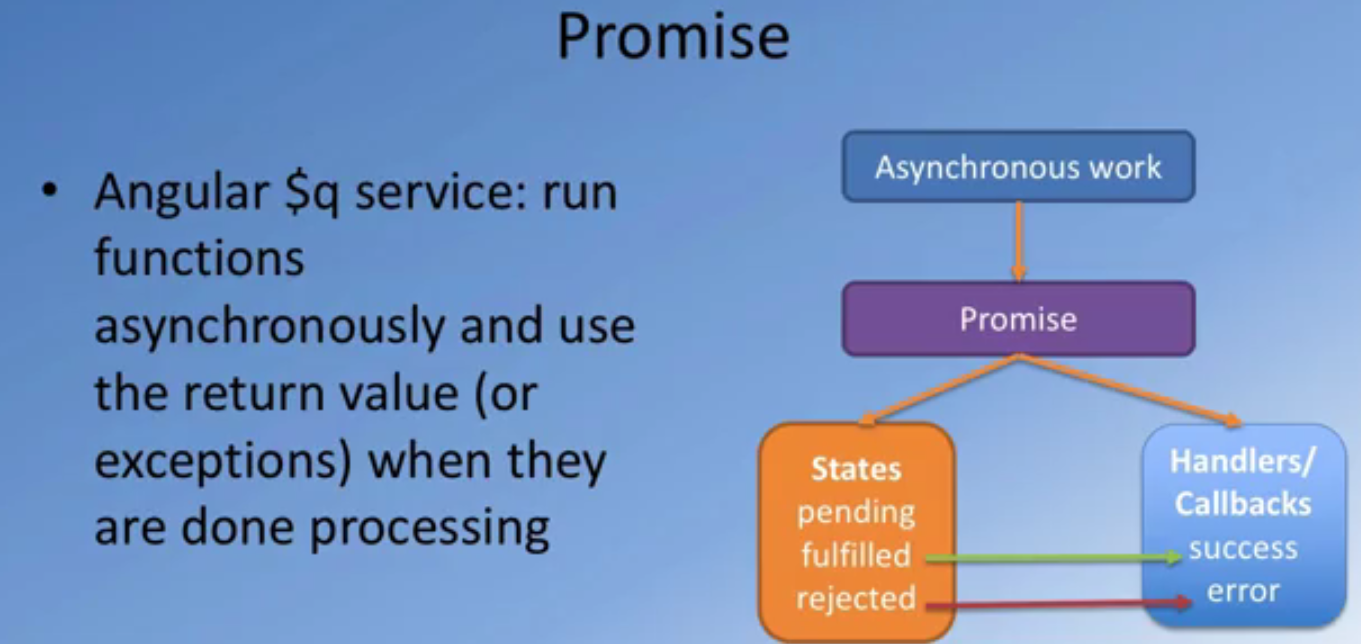
**$location service**  
  
watch and observe the URL  
Change the URL  
Allow to change the hash portion of URL

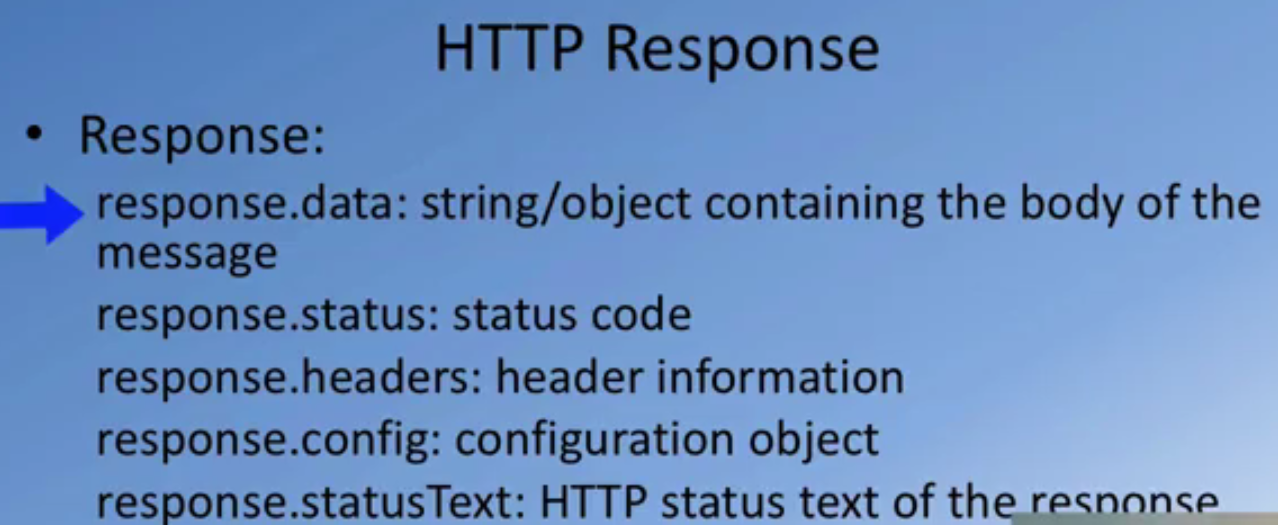
**Single Page application**  
  
Server supplies data and static HTML pages and other resources like images. Rendering of view is completely on client side. Client should support templating and routing.  
  
http://test.com#/menu/0  
Any change to the hash portion does not cause a page reload. Route is the hash portion of the URL in the context of the SPA.  
  
ngRoute uses location service and changes hash portion of URL if required.

|  |  |
| --- | --- |
| **UI-router** | **Ng-route** |
| view based on the state of the application. In this URL does not change. Changing the state of the application change the view. | View is based on the URL. View is tied to the URL. |
| It supports multiple view and nested view. | It does supports multiple view and nested view. |
| We can add multiple **ui-view**on the page. | We can have only one **ng-view** on the page. |
|  |  |

**Client Server communication**

Promise is service. When asynchronous operation is completed it come back to tell you that what happen.





**$resource**

A $resource is an abstraction on $http service.

When a resource object receives a response from the server it automatically deserializes the response into a JavaScript object, appending the following two methods to it in the process.

$save([params], [success], [error]) - POSTs the object back to the endpoint, saving changes

$remove([params], [success], [error]) - DELETEs the object

**Features**

* Custom request methods
* Default parameter values
* Custom (de)serialization logic
* Cache support

**$q Service**

Angular JS provides a service called $q which allows you to work with asynchronous functions and user their return values when the execution has been completed, and what its really cool about it is that it will let you write your custom promises as well (so you can resolve or reject a promise when appropriate).

A new instance of deferred is constructed by calling $q.defer().

The purpose of the deferred object is to expose the associated Promise instance as well as APIs that can be used for signaling the successful or unsuccessful completion, as well as the status of the task.

**Methods**

* resolve(value) – resolves the derived promise with the value. If the value is a rejection constructed via $q.reject, the promise will be rejected instead.
* reject(reason) – rejects the derived promise with the reason. This is equivalent to resolving it with a rejection constructed via $q.reject.
* 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.

**Properties**

* promise – {Promise} – promise object associated with this deferred.

all(promises) - wait for all operation to complete

* Combines multiple promises into a single promise that is resolved when all of the input promises are resolved.

**Scope in AngularJS**

**context where the model is stored so that controllers, directives and expressions can access it.**

When a Controller is attached to the DOM via the [ng-controller](https://docs.angularjs.org/api/ng/directive/ngController) directive, it creates new child scope.

We can see scope of current html element by typing $scope in console window.

Every module has rootScope and multiple child scopes.

Nesting of controller allow scope of child controller to access data of parent controller.

Angular add $watch function to variable and function which are added to $scope object.

We can also explicitly use $watch to watch an expression and take action when expression condition is true.

$digest() - calls all watches on current scope and their child scope.

$apply() - execute expression in angular from outside angular, It internally call $digest function. It is also calls internally when we call ng-click.

**Angular Expression**

**access variables and functions from the scope**

Angular expressions do not have access to global variables like window, document or location. This restriction is intentional. It prevents accidental access to the global state.

**One-time binding**

<span>{{::name}}</span>

The main purpose of one-time binding expression is to provide a way to create a binding that gets deregistered and frees up resources once the binding is stabilized. Reducing the number of expressions being watched makes the digest loop faster and allows more information to be displayed at the same time.