**Services**

* Services are javascript functions and are responsible to do a specific tasks
* Best way to implement reusable code
* Clean and maintainable code

**Creating services**

Angular provides 5 different ways by which you can create service. All of these function can be code on the built in **$provide service** and is also exposed in the **module object**.

* **provider()** : Most fundamental service creation function. Three of the remaining four functions all internally call the provider function. These functions are just short cuts which makes the code little simpler if you don't need to provide all the data required by the provider function.  
  + **factory()**
  + **service()**
  + **value()**
  + **constant() :** unique function of its own. It does not call provider internally unlike the rest.

**$provide**

Services are designed to be injected into the other components in your application.

Objects that know how to create injectable services are know as providers.

So before a new service can be created, first there must be a provider that knows how to create that service, it is the $provide service which is used to do that. (is a builtin service in angular)

**$provide service creates a provider which contains a function that is used to create a service.**

**provider()**

* As discussed **provider()** is a function in the **$provide service. ($provide.provider())**
* Most fundamental way to create a service. All the other methods are just wrappers around the provider function except constant().
* Has 2 parameters
  + Name of the provider
  + Function that will define the underlying provider
* my\_app.config(function($provide) {  
     
  $provide.provider('**user**', function() {  
   **this.$get** = function() {  
   return {  
   "name":"Arun",  
   "loc":"India",  
   "fn":function() {  
   return "Name = "+this.name+", location = "+this.loc  
   }  
   }  
   }  
   });  
    
  })
* So the provider name is user and the function passed to the provider should have a property this.$get
* The function assigned to **this.$get** is the function that will be called by angular to create your service.
* The service will be then represented by the return value of that function.
* Slightly complicated method compared to the rest of the methods.
* We call the config() which takes a function as a parameter which is the function that will execute to configure the module
* We inject the **$provide** service into config function
* Now let's use the service in your controller
* my\_app.controller('my\_cntrl', function($scope, user) {  
   $scope.btn\_label = user.name;  
  });
* One of the main benefit of using **provider()** is that, it is configurable at the module configuration phase. Let's modify our service a little.
* var my\_app = angular.module('my\_app', []);  
  **// creating provider via module object**  
    
  app.provider('data', function() {  
   this.$get = function() {  
   var namePrefix = "";  
   if(includePrefix) {  
   namePrefix = includePrefix;  
   }  
   return {  
   "firstname":"Arungopan",  
   "lastname":"Gopakumar",  
   "format\_name":function() {  
   return namePrefix+this.firstname+" "+this.lastname;  
   }  
   };  
   };  
     
   var includePrefix = false;  
   this.setPrefix = function(value) {  
   includePrefix = value;  
   };  
     
  });  
    
  app.config(function(dataProvider) {  
   dataProvider.setPrefix('Mr. ');  
  });  
    
    
  **// calling the injecting the provider into config function to do some configuration changes**  
  my\_app.config(function(userProvider) {  
   userProvider.setIncludePrefix(true);  
   //to access properties inside $get use the below statement  
   // userProvider.$get().name;  
  });  
    
  my\_app.controller('my\_cntrl', function($scope, user) {  
   $scope.btn\_label = user.name;  
  });
* In the code above we moved the **provider** out of **config()**
* Since the provider() is exposed in the module we can access it via module object
* From the above example, we can clearly see that the provider we created can be configured using the config function which is not available for any of the other methods.
* If you don't need to configure your service at the config level, we can go with other methods for creating a service

**Factory**

* Called using $provide.factory()
* Internally calls the provider function and assign the function you pass to the factory function as the value of the $get property on the provider.
* To use the factory function it has two parameters
  + 1st parameter is the name of the service that you create similar to provider()
  + 2nd parameter is a function that will return an object that represents the service instance
* If you don't need to configure the provider like we did in the last example, factory would be the simplest and more readable way of creating a service
* **Example for a service created using factory**
* var my\_app = angular.module('my\_app', []);  
    
  my\_app.factory('dataservice', dataService);  
    
  function dataService() {  
   return {  
   getAllUsers : getAllUsers,  
   getAllCars : getAllCars  
   }  
     
   function getAllUsers() {  
   return [  
   {  
   user\_id:1,  
   name:"Arun",  
   location:"India",  
   email:"arun.gopan@marlabs.com"  
   },  
   {  
   user\_id:2,  
   name:"Nanda",  
   location:"US",  
   email:"nanda@marlabs.com"  
   },  
   {  
   user\_id:3,  
   name:"Jacob",  
   location:"Canada",  
   email:"jacob@marlabs.com"  
   }  
   ];  
   }  
     
   function getAllCars() {  
   return [  
   {  
   car\_id:1,  
   name:"Volvo",  
   make:"2017"  
   },  
   {  
   car\_id:2,  
   name:"BMW",  
   make:"2015"  
   },  
   {  
   car\_id:3,  
   name:"Honda",  
   make:"2016"  
   }  
   ];  
   }  
     
  }  
    
    
  my\_app.controller('my\_cntrl', function($scope, dataservice) {  
   $scope.users = dataservice.getAllUsers();  
   $scope.cars = dataservice.getAllCars();  
  });

**Service**

* Called using $provide.service()
* Its again a wrapper around the factory function
* The function you passed to the service method will be treated as a constructor function and will called with the javascript new operator
* You would use the service method if you specifically need it your function to be treated as a constructor and called with the new operator
* So if you need to implement inheritance in your application, you can follow the service method.
* **Example for a service created using service()**
* var my\_app = angular.module('my\_app', []);  
    
  function human() {  
   this.txt = "I am a human";  
  }  
  human.prototype.sayHello = function() {  
   return "Hello everyone";  
  }  
    
  function user() {  
   human.call(this);  
   this.name = "Arun";  
   this.loc = "India";  
   this.display\_details = function() {  
   return "Name = "+this.name+", "+this.loc;  
   }  
  }  
    
  user.prototype = Object.create(human.prototype);  
    
  my\_app.service('dataService', user);  
    
    
  my\_app.controller('my\_cntrl', function($scope, dataService) {  
     
   $scope.service\_var = dataService.sayHello();  
     
  });

**Value and Constant Services**

Both are simplest and look lot alike

**Value Services**

* Shorthand for calling factory method with no parameters
* Cannot be injected into a module configuration function
* Can be overridden by decorators
* Services created with value are just wrappers around the factory function.
* You can use value instead of factory if you don't need to inject anything into the new service you are creating.
* Example for a service created using value()
* function rankUsers(rating) {  
   var rank = 0;  
   switch(true) {  
   case(rating > 90):  
   rank = "Good"  
   break;  
   case(rating > 95):  
   rank = "Excellant"  
   break;  
   case(rating < 90):  
   rank = "Average"  
   break;  
   }  
   return rank;  
  }  
    
  my\_app.value('valueService', {  
   rankUsers : rankUsers  
  });

**Constant services**

* Simply registers a service
* Can be injected into a module configuration function
* Cannot be overridden by decorators
* Example for a service created using constant function
* my\_app.constant('constants', {  
   APP\_TITLE:"My Application",  
   APP\_VERSION:"V 1.0.0.0"  
  });  
    
  app.config(function(constants) {  
   console.log(constants.APP\_TITLE);  
  });

**$http Services**

* Primary service available in angular for sending / receiving data from the server.
* **Configuration object**
  + method : get / post
  + url
  + params
  + data
  + headers
* **Response object**
  + Return a promise and the response object will be passed as a parameter to the callback function that process the promise.
  + **data** : contains the body of the response
  + **status :** http status code
  + **statusText** : http status text
  + **headers** : access http response headers
  + **config :** configuration object that was passed to $http service when the request was made
* **$http return a promise**
  + Example for a service using $http service
  + var app = angular.module('my\_app', []);  
      
    app.factory('dataService', ['$http', '$q', function($http, $q) {  
     return {  
     getData : function(){  
     return $http({  
     method:'GET',  
     url:'http://127.0.0.1:8080/data.json'  
     });  
     }  
     };  
      
    }]);  
      
    app.controller('my\_cntrl', ['$scope', 'dataService', function($scope, dataService, $sce) {  
     var service\_resp = dataService.getData()  
     service\_resp.then(function(success\_response) {  
     $scope.service\_var = success\_response.data;  
     },function(err\_response) {  
     $scope.service\_var = err\_response.data;  
     });  
    }]);
* var server\_req = {'method':'GET', 'url':'http://127.0.0.1:8080/profile.html', 'data':"name=Arun&jj=asdsa", headers: {'Content-Type': 'application/x-www-form-urlencoded;'}};  
   $http(server\_req).then(  
   function (response) { // success function  
   $scope.response\_via\_http\_service = $sce.trustAsHtml(response.data);  
   },  
   function (response) { // error function  
   $scope.response\_via\_http\_service = response.data;  
   }  
   );
* $**http shortcut methods**
  + **get()**
    - $http.get('<http://127.0.0.1:8080/data.json>');
  + **post()**
    - $http.post('http://127.0.0.1:8080/data.json', {"name":"ARun", "age":30});

**Promise**

* The Promise object is used for deferred or asynchronous computations. A Promise represents a value which may be available now, or in the future, or never.
* **Promises** are objects which represents the pending result of an asynchronous operation.
* The Promise object is used for **deferred or asynchronous computations**.
* A Promise represents a value which may be available now, or in the future, or never.

**Script without using promise**

function x() {  
   
 return {  
 'y': y,  
 'z':'Hello there!!!'  
 }  
   
 function y() {  
 setTimeout(function() {  
 return 'Hello World';  
 }, 2000);  
 }  
   
}  
  
var fn\_resp = x();  
  
console.log(fn\_resp.y());  
console.log(fn\_resp.z);

**Script using promise**

function x() {  
   
 return {  
 'y': y,  
 'z':'Hello there!!!'  
 }  
   
 function y() {  
 var promise = new Promise(function(resolve, reject) {  
 setTimeout(function() {  
 reject('Hello World');  
 }, 2000);  
 });  
 return promise;  
 }  
   
}  
  
var fn\_resp = x();  
  
fn\_resp.y().then(function(data) {  
 console.log('Data received = '+data);  
},  
function(data) {  
 console.log('Error happened = '+data)  
});  
  
console.log(fn\_resp.z);

In the above example both resolve and reject are methods which indicates whether the promise is resolved or rejected.

then(), has two function, first function for the handling resolved promise and second for rejected promise.

**Promises and $q service**

The **$q service** makes it easy to implement **asynchronous pattern** in angular application.

**$q service** in angular provides an api for working with promises and it also provides an **api for deferred objects** which returns promises to the calling code and signals them with result when the asynchronous operation is complete.

**Different steps for asynchronous operation**

1. Function initiates asynchronous call to service
2. When the service receives the call, it uses the $q service to create a new deferred object
3. Deferred object serves as a conduit or channel by which the service can communicate the the status of the asynchronous operation back to the function.
4. Deferred object will return a promise back to the function, which will prevent the service from blocking execution while the work is performed
5. Once the promise is received the function can use the promise api to configure callback function to execute when the work and the service is complete.
6. When the work is complete, either we call it as finished successfully or an error encountered.
7. The service uses the deferred api to signal the status of the work to the function.
8. At that point the function can execute the appropriate callback function based on result.

**Example for $q service**

var app = angular.module('myapp', []);  
  
app.factory('dataService', function($timeout, $q) {  
 return {  
 getAllUsers : getAllUsers,  
 getUserCompany:"Marlabs"  
 }  
   
 function getAllUsers() {  
 var deferred = $q.defer();  
 $timeout(function() {  
 deferred.resolve('Users array');  
 }, 2000);  
 return deferred.promise;  
 }  
   
});  
  
app.controller('my\_cntrl', function($scope, dataService) {  
 dataService.getAllUsers().then(function(success\_resp) {  
 console.log(success\_resp);  
 },  
 function(err\_resp) {  
 console.log(err\_resp);  
 });  
 console.log(dataService.getUserCompany);  
});