

AngularJS

AngularJS Services

Lesson Objectives

- **Creating and Registering Services**
- **Built-In Services**



Service Introduction

- **Service is just a simple JavaScript object that does some sort of work. It is typically stateless and encapsulates some sort of functionality.**
- **As a best practice we need to place the business logic into a service instead of placing it in the controller. It help us adhere to the Single Responsibility Principle(SRP) and Dependency Inversion Principle (DIP) as well as make the service reusable.**
- **Service can be used with controller, directive to construct model.**
- **Services provide a method for us to keep data around for the lifetime of the app and communicate across controllers in a consistent manner.**
- **Services are singleton objects that are instantiated only once per app (by the \$injector) and lazyloaded (created only when necessary).**

Creating and Registering a Service

- **Angular comes with several built-in services along with that we can create our own services.**
- **Angular compiler can reference service and load it as a dependency for runtime once it is registered.**
- **We can create service using five different ways**
 - `factory()`
 - `service()`
 - `provider()`
 - `constant()`
 - `value()`

Registering service using factory() function

- The most common method for registering a service with our Angular app is through the `factory()` method. This method is a quick way to create and configure a service.
- The `factory()` function takes two arguments.
 - Name of the service which we want to register
 - Function which runs when Angular creates the service. It will be invoked once for the duration of the app lifecycle, as the service is a singleton object. It can return anything from a primitive value to a function to an object.

```
var app = angular.module("serviceApp",[]);
app.factory("companyService",function(){
    return {
        company : {"Name":"IGATE", "Location":"India"}
    };
});
```

Registering service using service() function

- The `service()` function is used to register an instance of a service using a constructor function.
- The `service()` function will instantiate the instance using the `new` keyword when creating the instance.
- The `service()` function takes two arguments.
 - Name of the service which we want to register
 - The constructor function that we'll call to instantiate the instance.

```
var app = angular.module("serviceApp",[]);
/*constructor function*/
var Company = function(){
    this.getCompanyInfo = function(){
        return {"Name":"IGATE", "Location":"India"};
    };
};
app.service('companyService', Company); // service() function
```

Registering service using provider() function

- A provider is an object with a `$get()` method. The `$injector` calls the `$get` method to create a new instance of the service. The `provider()` method is responsible for registering services in the `$providerCache`.
- `factory()` function is shorthand for creating a service through the `provider()` method wherein we assume that the `$get()` function is the function passed
- `$provide service` is responsible for instantiating these providers at runtime.
- The `provide()` function takes two arguments. Name & (object/function/array)

```
var app = angular.module("serviceApp",[]);
app.provider("companyService", {
    $get : function(){
        return {
            company : {"Name":"IGATE", "Location":"India"}
        };
    }
});
```

Registering a Service with \$provide

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

```
<div ng-app="serviceApp" ng-controller="ServiceCntrl">
  <div>{{company.Name}} - {{company.Location}}</div>
</div>

<script>
var app = angular.module("serviceApp",[]);
app.config(function($provide){
  $provide.factory('companyService',function(){
    return {
      company : {"Name":"IGATE", "Location":"India"}
    };
  });
});

app.controller('ServiceCntrl',function($scope,companyService){
  $scope.company = companyService.company;
});
</script>
```


Registering service using constant() function

- **constant() function is used to register an existing value as a service so that we can inject it into a module configuration function**
- **The constant() function returns a registered service instance.**
- **The constant() function takes two arguments**
 - The name with which to register the constant.
 - The value to register as the constant

```
var app = angular.module("serviceApp",[]);  
app.constant('companyName','IGATE');  
app.config(function(companyName){  
    //We can resolve companyName here  
});
```

Registering service using value() function

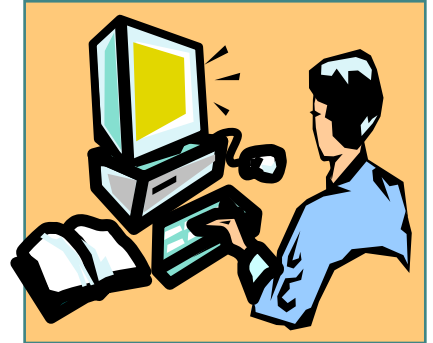
- **value() function is used to register the service, when the return value is just a constant.**
- **The value() function returns a registered service instance.**
- **we cannot inject a value into a config function.**
- **The value() function takes two arguments**
 - The name with which to register the value.
 - The injectable instance

```
var app = angular.module("serviceApp",[]);  
app.value('employee',{Id: 714709, Name:'Karthik'});
```

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

Demo

➤ CustomService



Built-In Services

- Angular services are substitutable objects that are wired together using dependency injection (DI). We can use those services to organize and share code across the application.
- Angular ships with lot of Built-In services. Some of important services are :

\$http

\$q

\$resource

\$anchorScroll

\$cacheFactory

\$compile

\$parse

\$locale

\$timeout

\$filter

\$exceptionHandler

\$log

AngularJS promise

- We used to implement asynchronous code is by using callback functions, but it is too complicated when we have to compose multiple asynchronous calls and make decisions depending upon the outcome of this composition.
- A promise represents the eventual result of an asynchronous operation.
- A promise is an object with a then method, then() function accepts 2 functions as parameters:
 - function to be executed (onSuccess) when the promise is fulfilled.
 - function to be executed (onFailure) when the promise is rejected.

promise.then(onSuccess,onFailure);

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

\$q Service

- **\$q service in AngularJS provides us deferred and promise implementations.**
- **Deferred represents a task that will finish in the future. We can get a new deferred object by calling the defer() function on the \$q service.**
 - `var deferred = $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.**
 - `resolve(...)` method of deferred object is used to signal that the task has succeeded.
 - `reject(...)` method is used to signal that the task has failed.
- **We can pass any type of information to resolve and reject methods which becomes the result of the task when it is called. The deferred object has a promise property which represents the promise of this task.**

\$q Service

```
<div ng-app="serviceApp" ng-controller="ServiceCntl">
  <div><input type="button" value="$q Service Test" ng-click="test()"/></div>
</div>
<script>
var app = angular.module("serviceApp",[]);
app.controller('ServiceCntl',function($scope,$q){
  $scope.test = function(){
    var deferred = $q.defer();
    var promise = deferred.promise;
    promise.then(function(response){
      alert('Success :'+response.message);
    },function(response){
      alert('Error :'+response.message);
    });
    /*uncomment to resolve the promise*/
    deferred.resolve({message:'Promise will gets fullfilled'});
    /*uncomment to reject the promise*/
    //deferred.reject({message:'Promise gets rejected'});
  }
});
</script>
```

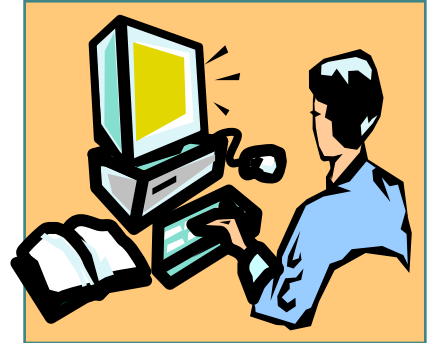
\$http Service

- **\$http service is a core Angular service that facilitates communication with the remote HTTP servers via the browser's XMLHttpRequest object or via JSONP.**
- **The \$http service is a function which takes a single argument i.e. a configuration object which is used to generate an HTTP request and returns a promise with two \$http specific methods: success and error.**

```
$http({method: 'GET', url: '/someUrl'}).  
  success(function(data, status, headers, config) {  
    // this callback will be called asynchronously when the response is available  
  }).  
  error(function(data, status, headers, config) {  
    // returns response with an error status.  
  });
```


Demo

- **httpService**
- **httpService-using-qService**



\$resource Service

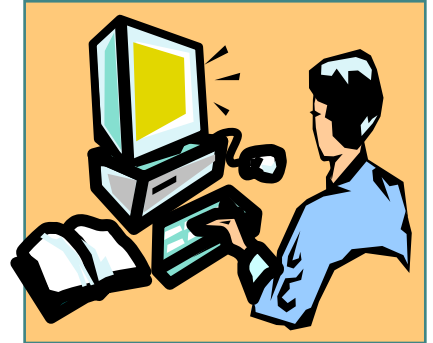
- **AngularJS's \$resource service allows us to create convenience methods for dealing with typical RESTful APIs.**
- **RESTful functionality is provided by Angular in the ngResource module, which is distributed separately from the core Angular framework so we need to refer angular-resource.js and define ngResource in our module.**

```
<script src="angular-resource.js"></script>  
var app = angular.module('myApp', ['ngResource']);
```

- **\$resource service returns resource object it has action methods which provides high-level behaviors so that we can interact with that methods without the need to interact with the low level \$http service.**

Demo

➤ resourceService



\$anchorScroll Service

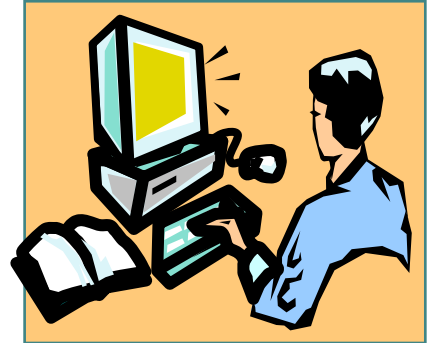
- `$anchorScrollService` checks current value of `$location.hash()` and scrolls to the related element.
- By calling `$anchorScrollProvider.disableAutoScrolling()` we can disable this feature.

```
<div id="scrollArea" ng-controller="ScrollController">  
  <a ng-click="gotoBottom()">Go to bottom</a>  
  ...  
  ...  
  <a id="bottom"></a> You're at the bottom!  
</div>
```

```
app.controller('ScrollController',function ($scope, $location, $anchorScroll) {  
  $scope.gotoBottom = function() {  
    $location.hash('bottom');  
    $anchorScroll();  
  };  
});
```

Demo

➤ anchorScrollService

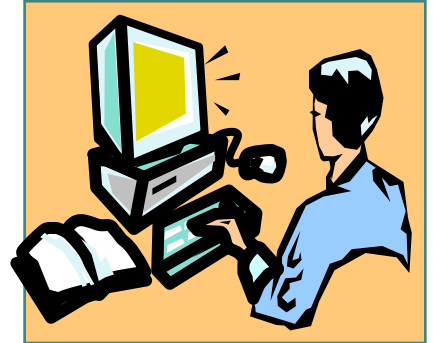


\$cacheFactory Service

- **We can cache the objects using \$cacheFactory service**
- **cache object has the following set of methods:**
 - {object} info() — Returns id, size, and options of cache.
 - {{*}} put({string} key, {*} value) — Puts a new key-value pair into the cache and returns it.
 - {{*}} get({string} key) — Returns cached value for key or undefined for cache miss.
 - {void} remove({string} key) — Removes a key-value pair from the cache.
 - {void} removeAll() — Removes all cached values.
 - {void} destroy() — Removes references to this cache from \$cacheFactory.
- **We can limit the number of items in the cache**
 - \$cacheFactory('customCache',{capacity:2})

Demo

➤ cacheFactoryService



\$compile Service

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

```

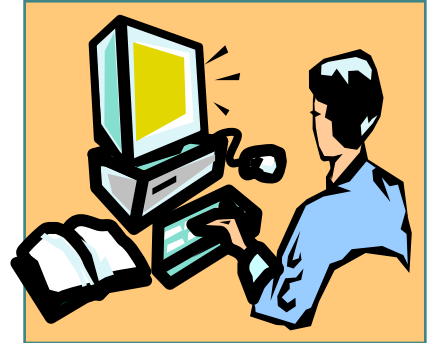
<div ng-controller="ServiceController">
    <div id="target"></div>
    Markup : <input type="text" ng-model="markup"/><br/>
    <input type="button" ng-click="appendToDivElement(markup)" value="Append"/>
</div>

var app = angular.module('serviceApp',[]);
app.controller("ServiceController",function($scope,$compile){
    $scope.appendToDivElement= function(markup){
        return
        angular.element('#target').append($compile(markup)($scope));
    }
});

```


Demo

➤ compileService



\$locale Service

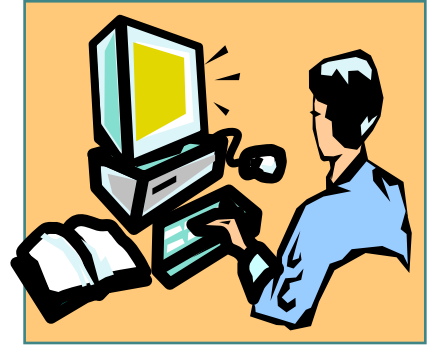
- **\$locale service provides localization rules for Angular components.**
- **locale id formatted as languageId-countryId (e.g. en-us)**
- **locale script files are available under i18n folder it needs to be referred in html.**
 - angular-locale_hi-in.js (hindi)
 - angular-locale_ta-in.js (tamil)

```
<div ng-app="serviceApp" ng-controller="ServiceController">  
  <h2>{{ todayDate | date:dateFormat }}</h2>  
</div>
```

```
var app = angular.module('serviceApp',[])  
app.controller("ServiceController",function($scope,$locale){  
  $scope.todayDate = new Date();  
  $scope.dateFormat = $locale.DATETIME_FORMATS.fullDate;  
});
```

Demo

➤ localeService



\$timeout Service

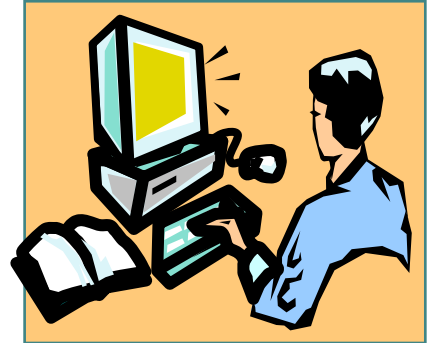
- **\$timeout is a wrapper for window.setTimeout function.**
- **The return value of registering a timeout function is a promise, which will be resolved when the timeout is reached and the timeout function is executed.**
- **To cancel a timeout request, call \$timeout.cancel(promise).**

```
<div ng-app="serviceApp" ng-controller="ServiceController">  
  <h2>Display Company Name in 5 sec(s): {{ companyName }}</h2>  
</div>
```

```
var app = angular.module('serviceApp',[])  
app.controller("ServiceController",function($scope,$timeout,$interval){  
  var timeoutPromise = $timeout(function(){  
    $scope.companyName = "IGATE";  
  },5000);  
});
```

Demo

➤ timeoutService



Summary

- Service is just a simple JavaScript object that does some sort of work.
- We can create service using five different ways
- Angular services are substitutable objects that are wired together using dependency injection (DI).
- A promise is an object with a then method, then() function accepts 2 functions as parameters which gets executed when the promise gets fulfilled and rejected respectively.
- AngularJS's \$resource service allows us to create convenience methods for dealing with typical RESTful APIs

