

创建服务组件

- 在AngularJS中创建一个服务组件很简单，只需要定义一个具有\$get方法的构造函数，然后使用模块的provider方法进行登记：

```
//定义构造函数
var myServiceProvider = function(){
    this.$get = function(){
        return ....
    };
};
//在模块中登记
angular.module("myModule", []).provider("myService",myServiceProvider);
```

可配置的服务

- 有时我们希望服务在不同的场景下可以有不同的行为，这意味着服务可以进行配置。

```
angular.module("myModule",[])  
  .config(["myServiceProvider",function(myServiceProvider){  
  
  }]);
```

“ 注意：服务提供者provider对象在注入器中的登记名称是：服务名+Provider。
例如：\$http的服务提供者实例名称是"\$httpProvider"。

语法糖

使用模块的provider方法定义服务组件，在有些场景下显得有些笨重。

AngularJS友好 地提供了一些简化的定义方法，这些方法通常只是对 provider方法的封装，分别适用于不同的应用场景：

- factory 使用一个对象工厂函数定义服务，调用该工厂函数将返回服务实例。
- service 使用一个类构造函数定义服务，通过new操作符将创建服务实例。
- value 使用一个值定义服务，这个值就是服务实例。
- constant 使用一个常量定义服务，这个常量就是服务实例。



factory方法

factory方法要求提供一个对象工厂，调用该类工厂将返回服务实例。

```
var myServiceFactory = function(){  
    return ...  
};  
angular.module("myModule", [])  
    .factory("myService", myServiceFactory);
```

“ AngularJS会将factory方法封装为provider，上面的示例等同于

```
var myServiceFactory = function(){  
    return ...  
};  
angular.module("myModule", []).provider("myService", function(){  
    this.$get = myServiceFactory;  
});
```

service方法

service方法要求提供一个构造函数，AngularJS使用这个构造函数创建服务实例：

```
var myServiceClass = function(){  
    this.method1 = function(){...}  
};  
angular.module("myModule", []).service("myService",myServiceClass);
```

“ AngularJS会将service方法封装为provider，上面的示例 等同于： `` var myServiceClass = function(){

```
}; angular.module("myModule",  
[]).provider("myService",function(){ this.$get = function(){  
    return new myServiceClass(); }; }); ``
```



value

有时我们需要在不同的组件之间共享一个变量，可以将这种情况视为一种服务：provider返回的总是变量的值。value方法提供了对这种情况的简化封装：

```
angular.module("myModule", [])  
  .value("myValueService", "19841230");
```

“ AngularJS会将value方法封装为provider，上面的示例 等同于：

```
angular.module("myModule", [])  
  .provider("myService", function() {  
    this.$get = function() {  
      return "19841230";  
    };  
  });
```

constant方法

有时我们需要在不同的组件之间共享一个常量，可以将这种情况视为一种服务：provider返回的总是常量的值。constant方法提供了对这种情况的简化封装：

```
angular.module("myModule", [])  
  .constant("myConstantService", "hello zfp");
```

和value方法不同，AngularJS并没有将constant方法封装成一个provider，而仅仅是在内部登记这个值。这使得常量在AngularJS的启动配置阶段就可以使用（创建任何服务之前）：你可以将常量注入到模块的config()方法中。



decorator

有时候我们希望给扩展一些angular自身的功能，又不能修改源码,此时
可以用decorator



拦截器

\$httpProvider 中有一个 interceptors 数组，而所谓拦截器只是一个简单的注册到了该数组中的常规服务工厂。下面的例子告诉你怎么创建一个拦截器：

```
module.factory('myInterceptor', ['$log', function($log) {  
    var myInterceptor = {};  
    return myInterceptor;  
}]);
```

然后通过它的名字添加到 \$httpProvider.interceptors 数组：

```
module.config(['$httpProvider', function($httpProvider) {  
    $httpProvider.interceptors.push('myInterceptor');  
}]);
```

拦截器功能

- 通过实现 **request** 方法拦截 **请求**：该方法会在 \$http 发送请求道后台之前执行，因此你可以修改配置或做其他的操作。
- 通过实现 **response** 方法拦截 **响应**：该方法会在 \$http 接收到从后台过来的响应之后执行，因此你可以修改响应或做其他操作
- 通过实现 **requestError** 方法拦截 **请求异常**：有时候一个请求发送失败或者被拦截器拒绝了
- 通过实现 **responseError** 方法拦截 **响应异常**：有时候我们后台调用失败了

ngresource

- ngResource模块是angular专门为RESTful架构而设计的一个模块,它提供了'\$resource'模块,\$resource模块是基于\$http的一个封装.

```
var HttpREST = angular.module('HttpREST',['ngResource']);  
HttpREST.factory('cardResource',function($resource){  
    return $resource('/card/user/:userID/:id',{userID:123,id:'@id'},{charge:{method:'  
}});
```

- 参数

\$resource(url,{url参数},{自定义方法})

url: 必填,资源的基础url - url中带有 ':' 项的是根据第二个参数来进行配置的.

url参数: 选填,配置url中的带有 ':' 项的参数

(' /card/user/:userID/:id',{userID:123,id:'@id'}),那么userID会被配置为123

自定义方法 给资源添加自定义的方法

\$resource()一共有以下5个方法:

- **get**: {method:'GET'} 一般用于获取某个资源:
- **query**: {method:'GET',isArray:true} 一般用于获取一整套的资源,以数组形式返回
- **save**: {method:'POST'} 一般用于保存某个资源,有可能是新建的资源,也有可能是更新现有的资源
- **remove**: {method:'DELETE'} 一般用于删除某个资源
- **delete**: {method:'DELETE'} 一般用于删除某个资源