Basic

概念概述(Conceptual Overview)

概念	说明
模板(Template)	带有Angular扩展标记的HTML
指令(Directive)	用于通过自定义属性和元素扩展HTML的行为
模型(Model)	用于显示给用户并且与用户互动的数据
作用域(Scope)	用来存储模型(Model)的语境(context)。模型放在这个语境中才能被控制器、指令和表达式等访问到
表达式(Expression)	模板中可以通过它来访问作用域（Scope）中的变量和函数
编译器(Compiler)	用来编译模板(Template)，并且对其中包含的指令(Directive)和表达式(Expression)进行实例化
过滤器(Filter)	负责格式化表达式(Expression)的值，以便呈现给用户
视图(View)	用户看到的内容（即DOM）
数据绑定(Data Binding)	自动同步模型(Model)中的数据和视图(View)表现
控制器(Controller)	视图(View)背后的业务逻辑
依赖注入(Dependency Injection)	负责创建和自动装载对象或函数
注入器(Injector)	用来实现依赖注入(Injection)的容器
模块(Module)	用来配置注入器
服务(Service)	独立于视图(View)的、可复用的业务逻辑

Template模板

Angular的模板是一个声明式的视图，它指定信息从模型、控制器变成用户在浏览器上可以看见的视图。 它把一个静态的DOM —— 只包含HTML，CSS以及Angular添加的标记和属性，然后引导Angular为其加上一些行为和格式转换器，最终变成一个动态的DOM。

在Angular中有以下元素属性可以直接在模板中使用:

- 指令(Directive) — 一个可扩展已有DOM元素或者代表可重复使用的DOM组件，用扩展属性(或者元素)标记。
- 表达式(Expressions) — 用双括号 {{ }} 给元素绑定表达式。
- 过滤器(Filter) — 格式化数据显示在界面上。
- 表单控件(Form Control) — 验证用户输入。

指令(Directive)

数据绑定ng-app、ng-init、ng-model、ng-bind

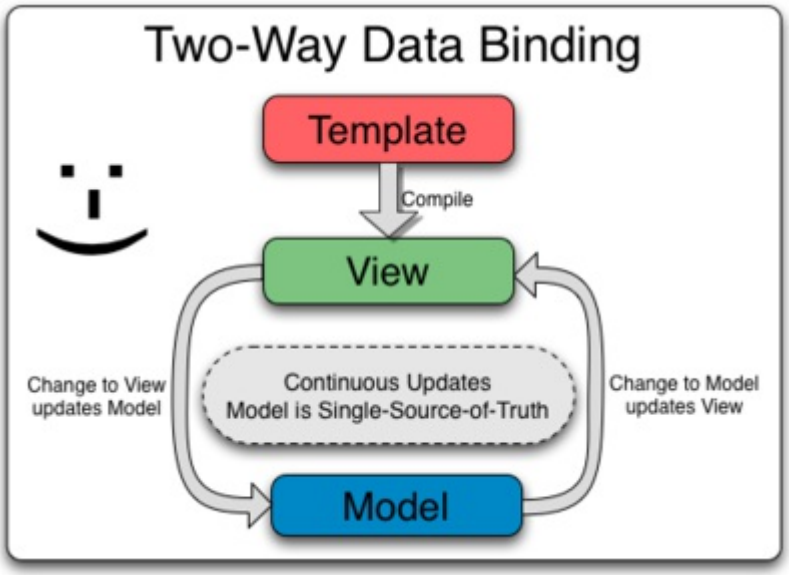
1. ng-app=" " 定义angularJS的使用范围；
2. ng-init="变量=值;变量='值'" 初始化变量的值，有多个变量时，中间用分号隔开；

3. ng-model="变量" 定义变量名；
4. ng-bind="变量" 绑定变量名，获取该变量的数据。这里的变量就是第3条的变量名。但是一般都用双重花括号来获取变量的值，比如：{{变量}}。

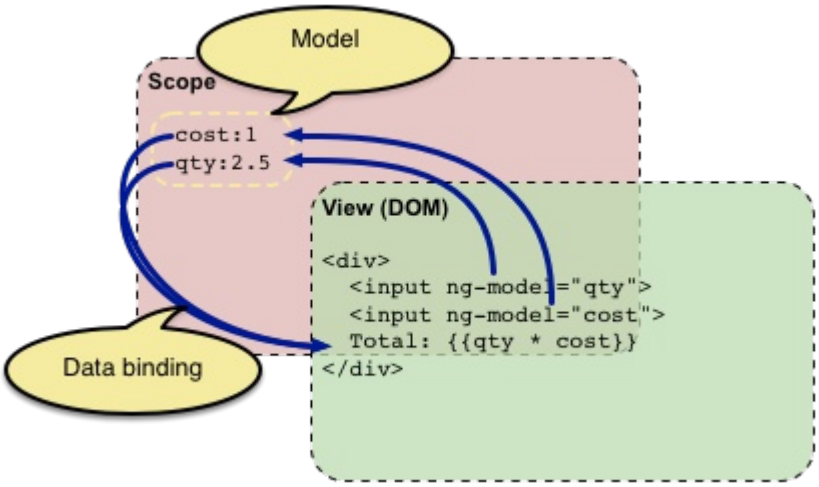
```
<!-- 定义ng变量firstName，初始值为John -->
<div ng-app ng-init="firstName='John'">
    <p>在输入框中尝试输入：</p>
    <p>姓名：<input type="text" ng-model="firstName"></p>
    <p>你输入的为： {{ firstName }} </p>
    <input type="text" value="{{firstName}}" disabled="disabled" />
</div>
```

ng-app 指令告诉 AngularJS， 元素是 AngularJS 应用程序 的"所有者"。

上面实例中的 {{ firstName }} 表达式是一个 AngularJS 数据绑定表达式。AngularJS 中的数据绑定，同步了 AngularJS 表达式与 AngularJS 数据。{{ firstName }} 是通过 ng-model="firstName" 进行同步。




```
<!--定义了两个初始值，数量qty为1，单价cost为5-->
<div ng-app="" ng-init="qty=1;cost=5">
    <h2>价格计算器</h2>
    <!--将两个变量绑定到两个文本框-->
    数量： <input type="number" ng-model="qty">
    价格： <input type="number" ng-model="cost">
    <p><b>总价：</b> {{ qty * cost }}</p>
</div>
```



PS:使用 ng-init 不是很常见。通常我们会使用控制器来代替它。

重复元素ng-repeat

```
<div ng-app ng-init="names=['Jani','Hege','Kai']">
    <p>使用 ng-repeat 来循环数组</p>
    <ul>
        <li ng-repeat="x in names">
            {{ x }}
        </li>
    </ul>
</div>
```

```
<div ng-app ng-init="names=[{name:'Jani',country:'Norway'},
{name:'Hege',country:'Sweden'},{name:'Kai',country:'Denmark'}]">
    <p>循环对象：</p>
    <ul>
        <li ng-repeat="x in names">
            {{ x.name + ', ' + x.country }}
        </li>
    </ul>
</div>
```

select ng-option

在 AngularJS 中我们可以使用 ng-option 指令来创建一个下拉列表，列表项通过对象和数组循环输出，如下实例:

```
<div ng-app="myApp" ng-controller="myCtrl">
    <!-- ng-options="a.brand for a in cars" 可以简单的设置为这样，或者使用键值对的
方式，如下for (key,value) in cars -->
    <select ng-model="selectedCar" ng-options="value.brand for (key,value) in
```

```
cars">
        <option value="">请选择购车品牌</option>
    </select>
    <h1>
        您选择的车是：{{selectedCar}}
    </h1>
</div>

<script>
    var app = angular.module('myApp', []);
    app.controller('myCtrl', function ($scope) {
        $scope.cars = [
            { brand: "Ford", model: "Mustang", color: "red" },
            { brand: "Fiat", model: "500", color: "white" },
            { brand: "Volvo", model: "XC90", color: "black" }
        ];
    });
</script>
```

状态指令

- ng-disabled 直接绑定到元素的disabled属性
- ng-show 显示
- ng-hide 隐藏

ng-include

可以用来包含html代码，其中的AngularJS代码也会被执行

ng-model模型

ng-model 指令用于绑定应用程序数据到 HTML 控制器(input, select, textarea)的值。

```
<!-- 定义了myApp模块，在加载时会先检测是否有名称为myApp的模块，如果没有则会报错 -->
<div ng-app="myApp" ng-controller="myCtrl">
    名字: <input ng-model="name">
</div>

<script>
    //定义一个模块，名称是myApp，和上面div的ng-app属性值对应
    var app = angular.module('myApp', []);

    //定义一个控制器模块，控制器的名称为myCtrl
    app.controller('myCtrl', function ($scope) {
        $scope.name = "John Doe";
    });
</script>
```

表单控件(Form Control) — 验证用户输入。

简单的表单

理解双向绑定的关键指令是ngModel. 指令ngModel通过维护"数据到视图"的同步以及"视图到数据"的同步实现了双向绑定。 另外，它还提供了API来让其他指令扩展其行为。

```
<div ng-app="myApp" ng-controller="formCtrl">
    <!-- novalidate 属性是在 HTML5 中新增的。禁用了使用浏览器的默认验证。 -->
    <form novalidate>
        First Name:<br>
        <input type="text" ng-model="user.firstName"><br>
        Last Name:<br>
        <input type="text" ng-model="user.lastName">
        <br><br>
        <button ng-click="reset()">RESET</button>
    </form>
    <p>form = {{user}}</p>
    <p>master = {{master}}</p>
</div>

<script>
    var app = angular.module('myApp', []);
    app.controller('formCtrl', function ($scope) {
        //设定一个默认值
        $scope.master = { firstName: "John", lastName: "Doe" };

        $scope.reset = function () {
            $scope.user = angular.copy($scope.master);
        };
        //init时设置当前model为默认值
        $scope.reset();
    });
</script>
```

表单验证

- 某个表单输入是否已填写，只要在输入字段元素上添加HTML5标记required即可： <input type="text" required />
- 验证表单输入的文本长度是否大于某个最小值，在输入字段上使用指令ng-minlength = "{number}"：<input type="text" ng-minlength="5" />
- 最大长度 验证表单输入的文本长度是否小于或等于某个最大值，在输入字段上使用指令ng-maxlength="{number}"：<input type="text" ng-maxlength="20" />
- 模式匹配 使用ng-pattern="/PATTERN/"来确保输入能够匹配指定的正则表达式：<input type="text" ng-pattern="/[a-zA-Z]/" />
- 邮箱验证 验证输入内容是否是邮箱地址，只要像下面这样将input的类型设置为email即可：<input type="email" name="email" ng-model="user.email" />

- 数字验证 验证输入内容是否是数字，将input的类型设置为number：<input type="number" name="age" ng-model="user.age" />
- URL验证 验证输入内容是否是URL，将input的类型设置为 url：<input type="url" name="homepage" ng-model="user.facebook_url" />

Angular还提供了内置的表单属性：

属性类	指令	描述
$valid	ng-valid	Boolean 告诉我们这一项当前基于你设定的规则是否验证通过
$invalid	ng-invalid	Boolean 告诉我们这一项当前基于你设定的规则是否验证未通过
$pristine	ng-pristine	Boolean 如果表单或者输入框没有使用则为True
$dirty	ng-dirty	Boolean 如果表单或者输入框有使用到则为True

```
<body ng-app="validationApp" ng-controller="mainController">
    <div class="container">
        <div class="row">

            <!--向右偏移两列-->
            <div class="col-md-8 col-md-offset-2">
                <div class="page-header"><h1>AngularJS 表单验证</h1></div>

                <!--novalidate 属性是在 HTML5 中新增的。禁用了使用浏览器的默认验
证。-->
                <form name="userForm" ng-submit="submitForm(userForm.$valid)"
novalidate>
                    <div class="form-group">
                        <label>用户名-</label>
                        <!--  input的name属性很重要，需要与验证属性对应  -->
                        <input class="form-control" type="text" name="name"
ng-model="name"
                        ng-minlength="3" ng-maxlength="5" required />
                        <!-- 当userForm.name.$invalid为true值，即验证不通过时，
ng-show为true，即显示当前段落 -->
                        <p class="bg-info" ng-show="userForm.name.$invalid">用
户名为必填项，长度在3~5之间</p>
                    </div>
                    <div class="form-group">
                        <label>邮箱-</label>
                        <!--  input的name属性很重要，需要与验证属性对应  -->
                        <input class="form-control" type="email" name="email"
ng-model="email" required />
                        <!-- 当userForm.email.$invalid为true值，即验证不通过时，
ng-show为true，即显示当前段落 -->
                        <p class="bg-info" ng-show="userForm.email.$invalid">请
输入正确邮箱地址</p>
                    </div>
                    <button type="submit" class="btn btn-
```

```
primary">Submit</button>
                </form>
            </div>
        </div>

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

        validationApp.controller('mainController', function ($scope) {

            //isValid 是表单是否验证通过的属性
            $scope.submitForm = function (isValid) {

                if (isValid) {
                    alert('验证通过');
                }
            };

        });
    </script>
</body>
```

Scope作用域

什么是Scope？ scope 是一个指向应用model的object。

- 是一个存储应用数据模型的对象
- 为 表达式 提供了一个执行上下文
- 作用域的层级结构对应于 DOM 树结构
- 作用域可以监听 表达式 的变化并传播事件

当在控制器中添加 $scope 对象时，视图 (HTML) 可以获取了这些属性。 视图中，你不需要添加 $scope 前缀, 只需要添加属性名即可，如： {{carname}}。

```
<div ng-app="myApp" ng-controller="myCtrl">
    <input ng-model="name">
    <h1>{{greeting}}</h1>
    <button ng-click='sayHello()'>点我</button>
</div>

<script>
    var app = angular.module('myApp', []);
    app.controller('myCtrl', function ($scope) {
        $scope.name = "Runoob";
        $scope.sayHello = function () {
            $scope.greeting = 'Hello ' + $scope.name + '!';
        };
    });
```

```
</script>
```

作用域有什么？

- 作用域提供了 ($watch) 方法监听数据模型的变化
- 作用域提供了 ($apply) 方法把不是由Angular触发的数据模型的改变引入Angular的控制范围内（如控制器，服务，及Angular事件处理器等）
- 作用域提供了基于原型链继承其父作用域属性的机制，就算是嵌套于独立的应用组件中的作用域也可以访问共享的数据模型（这个涉及到指令间嵌套时作用域的几种模式）
- 作用域提供了 表达式 的执行环境，比如像 {{username}} 这个表达式，必须得是在一个拥有属性这个属性的作用域中执行才会有意义，也就是说，作用域中可能会像这样 scope.username 或是 $scope.username，至于有没有 $ 符号，看你是在哪里访问作用域了

作用范围

$rootScope 全局对象的属性可在所有子作用域中访问，子作用域互相无法访问对方的私有变量，这一点与js的函数作用域完全一致。

```
<p>
    $rootScope 全局对象的属性可在所有子作用域中访问，子作用域互相无法访问对方的私有变量，这一点与js的函数作用域完全一致。
</p>
<form ng-app='myApp'>
    <div ng-controller="myCtrl1">
        {{name +' '+ globalName}}
    </div>
    <div ng-controller="myCtrl2">
        {{name +' '+ globalName}}
    </div>
</form>

<script>
    var app = angular.module('myApp', []);
    // 两个控制器
    app.controller('myCtrl1', ['$scope', '$rootScope', myCtrl1]);
    app.controller('myCtrl2', ['$scope', '$rootScope', myCtrl2]);

    function myCtrl1($scope, $rootScope) {
        $scope.name = 'jack';
        $rootScope.globalName = 'root';
    }

    function myCtrl2($scope, $rootScope) {
        $scope.name = 'white';
        $rootScope.globalName = 'global';
    }
</script>
```

表达式(Expressions) — 用双括号 {{ }} 给元素绑定表达式。

AngularJS 表达式写在双大括号内：{{ expression }}。

AngularJS 表达式 很像 JavaScript 表达式：它们可以包含文字、运算符和变量。

```
<div ng-app="" ng-init="quantity=1;cost=5">
    <!-- {{}}和ng-bind 都可以显示结果值-->
    <p>总价： {{ quantity * cost }}</p>
    <p><span ng-bind="quantity*cost"></span></p>
</div>
```

但需要注意的是：

1. AngularJS 表达式可以写在 HTML 中。
2. AngularJS 表达式不支持条件判断，循环及异常。
3. AngularJS 表达式支持过滤器。

Html编译（HTML Compiler）

> 注意：这篇文章是面向已经有一定 Angular 基础的开发者。 如果你仅仅只是刚上路，那么我们建议你看看 tutorial 先。如果你只是想创建几个自定义的指令，那可以去瞅瞅 directives guide。而如果你想要更深入地了解 Angular 的编译过程，那么你来对了，这就是你该看的。

Angular 的 HTML compiler 让开发者可以教浏览器一些新的语法技能。编译器允许你往现有的HTML元素或属性添加更多的操作逻辑，甚至可以让你自己创建新的带有自定义行为操作的HTML元素或属性。Angular 把这些操作扩展称之为 指令。

> http://www.angularjs.net.cn/tutorial/15.html

过滤器(Filter) — 格式化数据显示在界面上。

```
{{ 表达式 | 过滤器名 }}
{{ 表达式 | 过滤器1 | 过滤器2 |...}}
{{ 表达式 | 过滤器:参数1:参数2:...}}
```

过滤器可以使用一个管道字符（|）添加到表达式和指令中。AngularJS 过滤器可用于转换数据：

过滤器	描述
currency	格式化数字为货币格式。
filter	从数组项中选择一个子集。
lowercase	格式化字符串为小写。
orderBy	根据某个表达式排列数组。

uppercase　格式化字符串为大写。

```
<div ng-app="myApp" ng-controller="myAction">
    <h4>uppercase，lowercase 大小写转换</h4>
    <p>
        {{'small words' | uppercase}}
        <br />
        {{'THIS IS NEW SHIT!' | lowercase}}
    </p>
    <h4>date 格式化</h4>
    <p>
        {{1511797134424 | date:'yyyy-MM-dd HH:mm:ss'}}
        <br />
        {{ now.valueOf() | date:'yyyy-MM-dd HH:mm:ss'}}
    </p>
    <h4>number 格式化（保留小数）</h4>
    <p>
        {{149016.1945000 | number:2}}
    </p>
    <h4>currency货币格式化</h4>
    <p>
        {{ 250 | currency }}
        <br />
        {{ 250 | currency:"RMB ￥ " }}
    </p>
    <h4>filter查找</h4>
    <p>
        输入过滤器可以通过一个管道字符（|）和一个过滤器添加到指令中，该过滤器后跟一个冒
号和一个模型名称。
        filter 过滤器从数组中选择一个子集<br />
        {{ arrNames | filter:{'name':'iphone'} }}
    </p>
    <h4>limitTo 截取</h4>
    <p>
        {{"1234567890" | limitTo :6}} <!--从前面开始截取6位-->
        <br />
        {{"1234567890" | limitTo:-4}}  <!--从后面开始截取4位-->
    </p>
    <h4>orderBy 排序</h4>
    <p>
        降序：<br />
        {{ arrNames | orderBy:'age':true }}
        <br />升序<br />
        {{ arrNames | orderBy:'age' }}
    </p>
</div>

<script>
    var myApp = angular.module('myApp', []);
    myApp.controller('myAction', function ($scope) {
        $scope.now = new Date();
```

```
        $scope.arrNames = [
            { "age": 20, "id": 10, "name": "iphone" },
            { "age": 12, "id": 11, "name": "sunm xing" },
            { "age": 44, "id": 12, "name": "test abc" },
        ];

    });
</script>
```

控制器

AngularJS 应用程序被控制器控制。ng-controller 指令定义了应用程序控制器。控制器是 JavaScript 对象，由标准的 JavaScript 对象的构造函数 创建。

一般情况下，我们使用控制器做两件事：

- 初始化 $scope 对象
- 为 $scope 对象添加行为（方法）

```
<div ng-app="myApp" ng-controller="myCtrl">

    名: <input type="text" ng-model="firstName"><br/>
    姓: <input type="text" ng-model="lastName"><br/>
    <br/>
    姓名: {{firstName + " " + lastName}}

</div>

<script>
    var app = angular.module('myApp', []);
    app.controller('myCtrl', function ($scope) {
        $scope.firstName = "John";
        $scope.lastName = "Doe";
    });
</script>
```

略复杂的示例Adding UI Logic，商品单价及各币种金额之间的转换：

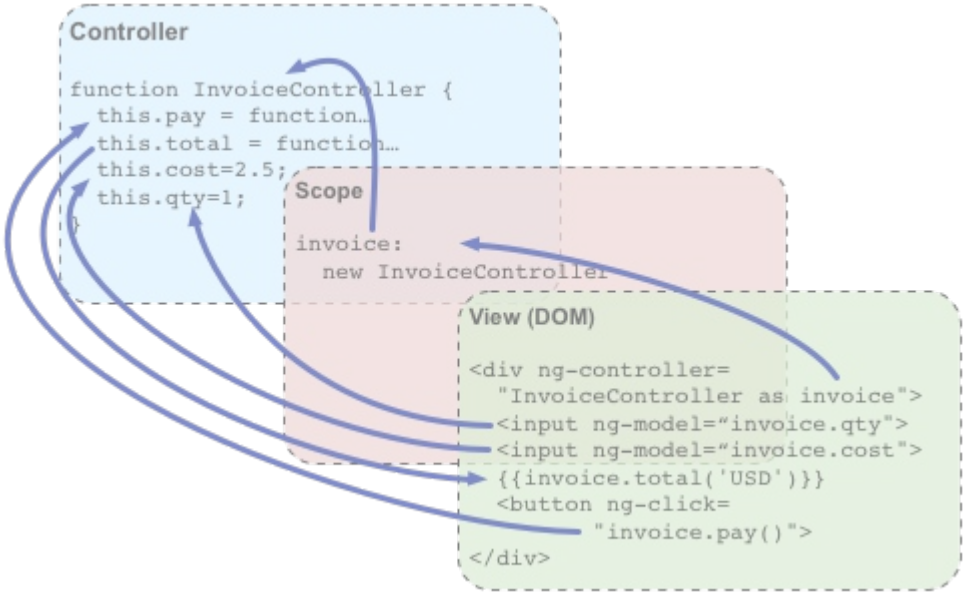
```
<body ng-app="myApp">
    <div ng-app="invoice1" ng-controller="InvoiceController as invoice">
        <b>Invoice:</b>
        <div>
            Quantity: <input type="number" min="0" ng-model="invoice.qty"
required>
        </div>
        <div>
            Costs: <input type="number" min="0" ng-model="invoice.cost"
```

```
required>
            <select ng-model="invoice.inCurr">
                <option ng-repeat="c in invoice.currencies">{{c}}</option>
            </select>
        </div>
        <div>
            <b>Total:</b>
            <span ng-repeat="c in invoice.currencies">
                {{invoice.total(c) | currency:c}}
            </span><br>
            <button class="btn" ng-click="invoice.pay()">Pay</button>
        </div>
    </div>

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

        myApp.controller('InvoiceController', function InvoiceController() {
            this.qty = 1;
            this.cost = 2;
            this.inCurr = 'EUR';
            this.currencies = ['USD', 'EUR', 'CNY'];
            this.usdToForeignRates = {
                USD: 1,
                EUR: 0.74,
                CNY: 6.09
            };

            this.total = function total(outCurr) {
                return this.convertCurrency(this.qty * this.cost, this.inCurr,
outCurr);
            };
            this.convertCurrency = function convertCurrency(amount, inCurr,
outCurr) {
                return amount * this.usdToForeignRates[outCurr] /
this.usdToForeignRates[inCurr];
            };
            this.pay = function pay() {
                window.alert('Thanks!');
            };
        });
    </script>
</body>
```



注意，下面的场合千万不要用控制器：

- 任何形式的DOM操作：控制器只应该包含业务逻辑。DOM操作则属于应用程序的表现层逻辑操作，向来以测试难度之高闻名于业界。把任何表现层的逻辑放到控制器中将会大大增加业务逻辑的测试难度。ng 提供数据绑定 （数据绑定） 来实现自动化的DOM操作。如果需要手动进行DOM操作，那么最好将表现层的逻辑封装在 指令 中
- 格式化输入：使用 angular表单控件 代替
- 过滤输出：使用 angular过滤器 代替
- 在控制器间复用有状态或无状态的代码：使用angular服务 代替
- 管理其它部件的生命周期（如手动创建 service 实例）

将控制器与 scope 对象关联

通过两种方法可以实现控制器和 scope 对象的关联：

- ngController指令 这个指令就会创建一个新的 scope
- $route路由服务

Scope 继承范例

我们常常会在不同层级的DOM结构中添加控制器。由于 ng-controller 指令会创建新的子级 scope ，这样我们就会获得一个与DOM层级结构相对应的的基于继承关系的 scope 层级结构。（译者注：由于 Js 是基于原型的继承，所以）底层（内层）控制器的 $scope 能够访问在高层控制器的 scope 中定义的属性和方法。详情参见理解“作用域” 。

> http://www.angularjs.net.cn/tutorial/2.html

控制器的单元测试

todo...

Service

在 AngularJS 中，服务是一个函数或对象，可在你的 AngularJS 应用中使用。 AngularJS 内建了30 多个服务。有个 $location 服务，它可以返回当前页面的 URL 地址。

```
<div ng-app="myApp" ng-controller="myAction">
    <h4>{{baseUrl}}</h4>
</div>

<script>
    var myApp = angular.module('myApp', []);
    myApp.controller('myAction', function ($scope, $location) {
        $scope.baseUrl = $location.absUrl();
    });
</script>
```

注意 $location 服务是作为一个参数传递到 controller 中。如果要使用它，需要在 controller 中定义。

$http 服务

$http 是 AngularJS 应用中最常用的服务。 服务向服务器发送请求，应用响应服务器传送过来的数据。

General usage

```
// Simple GET request example:
$http({
  method: 'GET',
  url: '/someUrl'
}).then(function successCallback(response) {
    // this callback will be called asynchronously
    // when the response is available
  }, function errorCallback(response) {
    // called asynchronously if an error occurs
    // or server returns response with an error status.
  });
```

The response object has these properties:

- data – {string|Object} – The response body transformed with the transform functions.
- status – {number} – HTTP status code of the response.
- headers – {function([headerName])} – Header getter function.
- config – {Object} – The configuration object that was used to generate the request.
- statusText – {string} – HTTP status text of the response.
- xhrStatus – {string} – Status of the XMLHttpRequest (complete, error, timeout or abort).

Shortcut methods:

```
$http.get('/someUrl', config).then(successCallback, errorCallback);
```

```
$http.post('/someUrl', data, config).then(successCallback, errorCallback);
```

在ASP.NET中$http的使用有点特殊，详见：

> https://stackoverflow.com/questions/19254029/angularjs-http-post-does-not-send-data

```
By default, jQuery transmits data using

Content-Type: x-www-form-urlencoded
and the familiar foo=bar&baz=moe serialization.

AngularJS, however, transmits data using

Content-Type: application/json
and { "foo": "bar", "baz": "moe" }
```

以上，需要做一个全局的设置，代码如下：

```
<div ng-app="myApp" ng-controller="myAction">
    <!--添加监听到OnHttp()-->
    <button ng-click="OnHttp()">请求</button>
    <div>
        姓名-
        <input type="text" ng-model="name" />
        <br />
        年龄-
        <input type="number" ng-model="age" />
    </div>
    <h2>
        $http:
        {{httpBack}}
    </h2>
    <h2>
        $http.get():
        {{getObj}}
    </h2>
    <h2>
        $http.post():
        {{postObj}}
    </h2>
</div>

<script>
    var myApp = angular.module('myApp', [], function ($httpProvider) {
        /*
        做一些全局性的设置，主要是content-type和param传参的改变
        */
```

```
        // Use x-www-form-urlencoded Content-Type
        $httpProvider.defaults.headers.post['Content-Type'] = 'application/x-
www-form-urlencoded;charset=utf-8';

        /**
            * The workhorse; converts an object to x-www-form-urlencoded
serialization.
            * @param {Object} obj
            * @return {String}
            */
        var param = function (obj) {
            var query = '',
                name, value, fullSubName, subName, subValue, innerObj, i;

            for (name in obj) {
                value = obj[name];

                if (value instanceof Array) {
                    for (i = 0; i < value.length; ++i) {
                        subValue = value[i];
                        fullSubName = name + '[' + i + ']';
                        innerObj = {};
                        innerObj[fullSubName] = subValue;
                        query += param(innerObj) + '&';
                    }
                } else if (value instanceof Object) {
                    for (subName in value) {
                        subValue = value[subName];
                        fullSubName = name + '[' + subName + ']';
                        innerObj = {};
                        innerObj[fullSubName] = subValue;
                        query += param(innerObj) + '&';
                    }
                } else if (value !== undefined && value !== null)
                    query += encodeURIComponent(name) + '=' +
encodeURIComponent(value) + '&';
            }

            return query.length ? query.substr(0, query.length - 1) : query;
        };

        // Override $http service's default transformRequest
        $httpProvider.defaults.transformRequest = [function (data) {
            return angular.isObject(data) && String(data) !== '[object File]'
? param(data) : data;
        }];
    });
    //var myApp = angular.module('myApp', []);
    myApp.controller('myAction', function ($scope, $http) {
        //设置初始值
        $scope.name = 'jack';
        $scope.age = 123;
```

```
        //实现OnHttp方法
        $scope.OnHttp = function () {
            $http({
                method: 'POST',
                url: '/HomeHandler.ashx',
                data: { name: $scope.name, age: $scope.age }
            }).then(function (resp) {
                $scope.httpBack = resp.data;
            }, function (err) {
                console.error(err);
            });

            //$http.get()
            var url = '/HomeHandler.ashx?name=' + $scope.name + '&age=' +
$scope.age;
            $http.get(url).then(function (resp) {
                //已转换为对象
                $scope.getObj = resp.data;
            }, function (err) {
                console.error(err.data);
            });

            //$http.post()
            $http.post('/HomeHandler.ashx', { name: $scope.name, age:
$scope.age }).then(function (resp) {
                $scope.postObj = resp.data;
            }, function (err) {
                console.error(err);
            });
        }
    });
</script>
```

table

搭配使用ng-repeat实现表格的自动填充，实现代码如下：

```
<style>
    /*适配奇数元素样式*/
    table tr:nth-child(odd) {
        background-color: lightblue;
    }
</style>
<div ng-app="myApp" ng-controller="myCtrl">
    <!--结合bootstrap样式-->
    <table class="table table-hover table-bordered">
        <tr>
            <th>Name</th>
            <th>City</th>
```

```
            <th>Country</th>
        </tr>
        <tr ng-repeat="x in records">
            <td>{{x.Name}}</td>
            <td>{{x.City}}</td>
            <td>{{x.Country}}</td>
        </tr>
    </table>
</div>

<script>
    var app = angular.module('myApp', []);
    app.controller('myCtrl', function ($scope) {
        $scope.records = [
            { "Name": "Alfreds Futterkiste", "City": "Berlin", "Country":
"Germany" },
            { "Name": "Ana Trujillo Emparedados y helados", "City": "México
D.F.", "Country": "Mexico" },
            { "Name": "Antonio Moreno Taquería", "City": "México D.F.",
"Country": "Mexico" },
            { "Name": "Around the Horn", "City": "London", "Country": "UK" },
            { "Name": "B's Beverages", "City": "London", "Country": "UK" },
            { "Name": "Berglunds snabbköp", "City": "Luleå", "Country":
"Sweden" },
            { "Name": "Blauer See Delikatessen", "City": "Mannheim",
"Country": "Germany" },
            { "Name": "Blondel père et fils", "City": "Strasbourg", "Country":
"France" },
            { "Name": "Bólido Comidas preparadas", "City": "Madrid",
"Country": "Spain" },
            { "Name": "Bon app'", "City": "Marseille", "Country": "France" },
            { "Name": "Bottom-Dollar Marketse", "City": "Tsawassen",
"Country": "Canada" },
            { "Name": "Cactus Comidas para llevar", "City": "Buenos Aires",
"Country": "Argentina" },
            { "Name": "Centro comercial Moctezuma", "City": "México D.F.",
"Country": "Mexico" },
            { "Name": "Chop-suey Chinese", "City": "Bern", "Country":
"Switzerland" },
            { "Name": "Comércio Mineiro", "City": "São Paulo", "Country":
"Brazil" }
        ];
    });
</script>
```

事件

```
<div ng-app="myApp" ng-controller="myCtrl">
    <button class="btn btn-primary" ng-click="toggle()">{{btnText}}</button>
```

```
    <h1 ng-hide="isHide">事件测试</h1>
</div>

<script>
    var app = angular.module('myApp', []);
    app.controller('myCtrl', function ($scope) {
        $scope.btnText = '隐藏';
        $scope.toggle = function () {
            $scope.isHide = !$scope.isHide;
            $scope.btnText = $scope.isHide ? '显示' : '隐藏';
        }
    });
</script>
```

AngularJS还提供了其他HTML事件的封装，有如下：

- ng-blur
- ng-change
- ng-click
- ng-copy
- ng-cut
- ng-dblclick
- ng-focus
- ng-keydown
- ng-keypress
- ng-keyup
- ng-mousedown
- ng-mouseenter
- ng-mouseleave
- ng-mousemove
- ng-mouseover
- ng-mouseup
- ng-paste

具体API说明和demo可参考：

> https://docs.angularjs.org/api/ng/directive/ngClick#overview

> https://segmentfault.com/a/1190000002634554

全局API

以下列出了一些通用的 API 函数：

API	描述
angular.lowercase()	转换字符串为小写
angular.uppercase()	转换字符串为大写

angular.isString()	判断给定的对象是否为字符串，如果是返回 true。
angular.isNumber()	判断给定的对象是否为数字，如果是返回 true。

依赖注入(Dependency Injection)

依赖注入（DI）是一种让代码管理其依赖关系的设计模式。 关于 DI 的更多讨论，请参见维基百科的 Dependency Injection 词条，以及 Martin Fowler 写的 Inversion of Control ，或查阅那些有关设计模式的书。

DI简介

对象或函数可以通过三种方式获得所依赖的对象（简称依赖）：

1. 创建依赖，通常是通过 new 操作符
2. 查找依赖，在一个全局的注册表中查阅它
3. 传入依赖，需要此依赖的地方等待被依赖对象注入进来

前两种方式：创建或是查找依赖都不是那么理想，因为它们都将依赖写死在对象或函数里了。问题在于，想要修改这两种方式获得依赖对象的逻辑是很困难的。尤其是在测试的时候，会遇到很多问题，因为测试时常常需要我们提供所依赖对象的替身(MOCK)。

第三种方式是最理想的，因为它免除了客户代码里定位相应的依赖这个负担，反过来，依赖总是能够很简单地被注入到需要它的组件中。

为了分离"创建依赖"的职责，每个 Angular 应用都有一个 injector对象。这个 injector 是一个服务定位器，负责创建和查找依赖。（译注：当你的app的某处声明需要用到某个依赖时，Angular 会调用这个依赖注入器去查找或是创建你所需要的依赖，然后返回来给你用） 下面是一个利用 injector 服务例子：(执行还有问题TODO)

```
<div ng-controller="MyController">
    <button ng-click="sayHello()">Hello</button>
</div>

<script>
    // Provide the wiring information in a module
    var myModule = angular.module('myModule', []);

    myModule.factory('greeter', function ($window) {
        return {
            greet: function (text) {
                $window.alert(text);
            }
        };
    });

    var injector = angular.injector(['ng', 'myModule']);
    var greeter = injector.get('greeter');

    function MyController($scope, greeter) {
        $scope.sayHello = function () {
            greeter.greet('Hello World');
```

```
        };
    }

    injector.instantiate(MyController);
</script>
```

Using Dependency Injection

Factory Methods

```
angular.module('myModule', [])
.factory('serviceId', ['depService', function(depService) {
  // ...
}])
.directive('directiveName', ['depService', function(depService) {
  // ...
}])
.filter('filterName', ['depService', function(depService) {
  // ...
}]);
```

Module Methods

```
angular.module('myModule', [])
.config(['depProvider', function(depProvider) {
  // ...
}])
.run(['depService', function(depService) {
  // ...
}]);
```

Controllers

```
someModule.controller('MyController', ['$scope', 'dep1', 'dep2',
function($scope, dep1, dep2) {
  //...
  $scope.aMethod = function() {
    //...
  }
  //...
}]);
```

依赖注解

那么，injector 是如何知道哪些服务需要被注入呢？

应用开发者需要提供 injector 需要使用的注解信息来解析依赖。Angular 之中，按照API文档说明，某些 API 方法需要通过 injector 调用。这样，injector 要知道得往这个方法中注入什么服务。下面是用服务名信息来进行注解的三种等价的方式，它们可以互用，按照你觉得适合的情况选用相应的方式。

推断依赖

最简单的获取依赖的方法是让你的函数的参数名直接使用依赖名。

```
function MyController($scope, greeter) {
    //...
}
```

给 injector 一个函数，它可以通过检查函数声明并抽取参数名可以推断需要注入的服务名。在上面的例子中，$scope 和 greeter 是两个需要被注入到函数中的服务。 虽然这种方式很直观明了，但是它对于压缩的 JavaScript 代码来说是不起作用的，因为压缩过后的 JavaScript 代码重命名了函数的参数名。这就让这种注解方式只对 pretotyping 和 demo级应用有用。

$inject 注解

为了让重命名了参数名的压缩版的 JavaScript 代码能够正确地注入相关的依赖服务。函数需要通过 $inject 属性进行标注，这个属性是一个存放需要注入的服务的数组。

```
var MyController = function (renamed$scope, renamedGreeter) {
    //...
}
MyController['$inject'] = ['$scope', 'greeter'];
```

在这种场合下，$inject 数组中的服务名顺序必须和函数参数名顺序一致。以上述代码段为例，$scope 将会被注入到 'renamed$scope'，而 greeter 则是注入到 'renamedGreeter'。需要注意 $inject 注解是和真实的函数声明中的参数保持同步的。 这种注解方法对于控制器声明很有用处，因为它是把注解信息赋给了函数。

行内注解

有时候用 $inject 注解的方式不方便，比如标注指令的时候（译注：这里标注指令可以理解为告诉指令需要加载哪些服务依赖的说明）。

```
//看下面的例子：
someModule.factory('greeter', function ($window) {
    //...
});

//由于需要一个临时的变量导致代码膨胀：
```

```
var greeterFactory = function (renamed$window) {
    //...
};

greeterFactory.$inject = ['$window'];

someModule.factory('greeter', greeterFactory);

//所以，第三种注解风格被引入，如下：
someModule.factory('greeter', ['$window', function (renamed$window) {
    //...
}]);
```

Providers Recipe供应者

你所构建的每个web应用都是由互相协作以达成特定目标的对象构成。为了让应用得以运行，这些对象还需要被实例化并绑定在一起。在基于Angular框架的应用里，这些对象大都是通过注入服务自动地实例化并绑定在一起。 注入器创建两类对象，服务和专用对象。 服务是对象，而这些对象的API是由编写服务的开发人员所决定的。 专用对象遵循Angular框架特定的API。这些对象包括控制器，指令，过滤器或动画。 注入器需要知道如何去创建这些对象。你应该通过注册一种"图纸"来告诉Angular如何创建你的对象。这里共有5种图纸。 最冗长同时又最复杂的图纸是Provider Recipe图纸，其余4种分别是 —— Value，Factory，Service和Constant，这4种都只是基于Provider之上的语法糖。 现在让我们看看通过不同图纸来创建和使用服务的场景。首先我们从最简单的例子开始 —— 你代码在很多地方都要使用同一个字符串，这个场景下，我们通过Value Recipe图纸来完成服务的创建。

> https://docs.angularjs.org/guide/providers

路由

AngularJS 路由允许我们通过不同的 URL 访问不同的内容。 通过 AngularJS 可以实现多视图的单页Web应用（single page web application，SPA）。

通常我们的URL形式为 localhost:8080/index.html，但在单页Web应用中 AngularJS 通过 # + 标记 实现，例如：

```
localhost:8080/#/first
localhost:8080/#/second
localhost:8080/#/third
```

当我们点击以上的任意一个链接时，向服务端请的地址都是一样的。 因为 # 号之后的内容在向服务端请求时会被浏览器忽略掉。 所以我们就需要在客户端实现 # 号后面内容的功能实现。

AngularJS 路由 就通过 # + 标记 帮助我们区分不同的逻辑页面并将不同的页面绑定到对应的控制器上。

需要注意的是从1.6+版本以后，$locationProvider.hashPrefix()默认值变为了!，在跳转链接处也需要特殊处理，有两种方案：

1. 使用#!代替旧版本中的#，如修改为 About
2. 针对$locationProvider配置恢复之前版本的hashPrefix

```
appModule.config(['$locationProvider', function($locationProvider) {
        $locationProvider.hashPrefix('');
}]);
```

路由设置对象：

```
$routeProvider.when(url, {
    template: string,//在 ng-view 中插入简单的 HTML 内容
    templateUrl: string,//在 ng-view 中插入 HTML 模板文件 eg:views/about.html
    controller: string, function 或 array,//在当前模板上执行的controller函数，对应处理的控制器名称
    controllerAs: string,//为controller指定别名。
    redirectTo: string, function,//重定向的地址。
    resolve: object<key, function>//指定当前controller所依赖的其他模块。
});
```

综合案例如下： index.html文件

```
<!DOCTYPE html>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title></title>
    <meta charset="utf-8" />
    <link rel="stylesheet"
href="//netdna.bootstrapcdn.com/bootstrap/3.0.0/css/bootstrap.min.css" />
    <link rel="stylesheet" href="//netdna.bootstrapcdn.com/font-
awesome/4.0.0/css/font-awesome.css" />
    <script src="//cdn.bootcss.com/angular.js/1.6.6/angular.min.js"></script>
    <script src="//cdn.bootcss.com/angular.js/1.6.6/angular-route.min.js">
</script>

    <!-- 官网的太慢了。。。 -->
    <!--
    <script src="//code.angularjs.org/1.6.6/angular.min.js"></script>
    <script src="//code.angularjs.org/1.6.6/angular-route.min.js"></script>
    -->
    <script src="app.js"></script>
</head>
<body ng-app="myApp">
    <header>
        <nav class="navbar navbar-default">
            <div class="container">
```

```
                <div class="navbar-header">
                    <a class="navbar-brand" href="/">AngularJS路由示例</a>
                </div>

                <ul class="nav navbar-nav navbar-right">
                    <li><a href="#"><i class="fa fa-home"></i> Home</a></li>
                    <li><a href="#/about"><i class="fa fa-shield"></i>
About</a></li>
                    <li><a href="#/contact/123/jackjones"><i class="fa fa-
comment"></i> Contact</a></li>
                </ul>
            </div>
        </nav>
    </header>

    <div class="container-fluid">
        <div ng-view></div>
        <!-- angular templating -->
        <!-- this is where content will be injected -->
    </div>
</body>
</html>
```

app.js文件

```
'use strict';
var myApp = angular.module('myApp', ['ngRoute']);

/*
AngularJS GitHub Pull #14202 Changed default hashPrefix to '!' 从 1.6+版本以
后。。。坑死了
stackoverflow https://stackoverflow.com/questions/41211875/angularjs-1-6-0-
latest-now-routes-not-working
或者采用此写法 href="#!/about"
*/
myApp.config(['$locationProvider', function ($locationProvider) {
    $locationProvider.hashPrefix('');
}]);

//配置路由
myApp.config(["$routeProvider", function ($routeProvider) {

    $routeProvider

      //home
      .when('/', {
          templateUrl: '/Template/home.html',
          controller: 'mainController'
      })
```

```
        //about
        .when('/about', {
            templateUrl: '/Template/about.html',
            controller: 'aboutController'
        })

        //contact
        .when('/contact', {
            templateUrl: '/Template/contact.html',
            controller: 'contactController'
        });

}]);
//main控制器
myApp.controller('mainController', ["$scope", function ($scope) {
    // create a message to display in our view
    $scope.message = 'Everyone come and see how good I look!';
}]);
//about控制器
myApp.controller('aboutController', ["$scope", function ($scope) {
    $scope.message = 'Look! I am an about page.';
}]);
//contact控制器
myApp.controller('contactController', ["$scope", "$routeParams", function
($scope, $routeParams) {
    $scope.id = $routeParams.id;
    $scope.name = $routeParams.name;
    $scope.message = 'Contact us! JK. This is just a demo.';
}]);
```

about.html文件

```
<div class="text-center" ng-controller="aboutController">
    <h1>About Page</h1>
    <p>{{ message }}</p>
</div>
```

contact.html文件

```
<div class="text-center" ng-controller="contactController">
    <h1>Contact Page</h1>

    <p>{{ message }}</p>
    <!-- 接收传递的参数 -->
    <p>ID-{{id}}</p>
    <p>NAME-{{name}}</p>
</div>
```

home.html文件

```
<div class="text-center" ng-controller="mainController">
    <h1>Home Page</h1>
    <p>{{ message }}</p>
</div>
```

modules模块

大多数应用程序都有个 main 函数来初始化、连接以及启动整个应用。ng 中虽然没有 main 函数，但它用模块来描述应用将如何启动。这种策略有如下几种优势：

- 整个过程是声明式的，更容易理解
- 在单元测试中，没有必要加载所有模块，这样有利于单元测试的代码书写
- 在场景测试中，额外的模块可以被加载进来，进而重写一些配置，这样有助于实现应用的端到端的测试
- 第三方代码可以很容易被打包成可重用的模块
- 模块可以用任意顺序或并行顺序加载（得益于模块执行的延迟性）

以自定义一个筛选器作为示例：

```
<div ng-app="myApp">
    <input type="text" ng-model="inputName" />
    <h1>
            {{ 'hello ' + inputName | myFilter}}
    </h1>
</div>

<script>
    var app = angular.module('myApp', []);
    app.filter('myFilter', function () {
        return function (name) {
            return name + '您好';
        }
    });
</script>
```

参考引用：

最推荐-官方原版教程

AngularJS 教程

菜鸟教程