新晨科技股份有限公司

AngularJS 介绍

AngularJS 学习文档

欧阳龙蛟

2014/1/15

[在此处键入文档的摘要。摘要通常是对文档内容的简短总结。在此处键入文档的摘要。摘要通常是对文档内容的简短总结。]

目录

1 前言

前端技术的发展是如此之快，各种优秀技术、优秀框架的出现简直让人目不暇接，紧跟时代潮流，学习掌握新知识自然是不敢怠慢。

AngularJS 是 google 在维护，其在国外已经十分火热，可是国内的使用情况却有不小的差距，参考文献/网络文章也很匮乏。这里便将我学习 AngularJS 写成文档，一方面作为自己学习路程上的记录，另一方面也给有兴趣的同学一些参考。

首先我自己也是一名学习者，会以学习者的角度来整理我的行文思路，这里可能只是些探索，有理解或是技术上的错误还请大家指出；其次我特别喜欢编写小例子来把一件事情说明白，故在文中会尽可能多的用示例加代码讲解，我相信这会是一种比较好的方式；最后，我深知 AngularJS 的使用方式跟 jquery 的使用方式有很大不同，在大家都有 jquery、ext 经验的条件下对于 angular 的学习会困难重重，不过我更相信在大家的坚持下，能够快速的学好 AngularJS，至少咱也能深入了解到 AngularJS 的基本思想，对咱们以后自己的插件开发、项目开发都会有很大的启示。

2 AngularJS 概述

2.1 AngularJS 是什么？

AngularJs（后面就简称 ng 了）是一个用于设计动态 web 应用的结构框架。首先，它是一个框架，不是类库，是像 EXT 一样提供一整套方案用于设计 web 应用。它不仅仅是一个 javascript 框架，因为它的核心其实是对 HTML 标签的增强。

何为 HTML 标签增强？其实就是使你能够用标签完成一部分页面逻辑，具体方式就是通过自定义标签、自定义属性等，这些 HTML 原生没有的标签/属性在 ng 中有一个名字：指令（directive）。后面会详细介绍。那么，什么又是动态 web 应用呢？与传统 web 系统相区别，web 应用能为用户提供丰富的操作，能够随用户操作不断更新视图而不进行 url 跳转。ng 官方也声明它更适用于开发 CRUD 应用，即数据操作比较多的应用，而非是游戏或图像处理类应用。

为了实现这些，ng 引入了一些非常棒的特性，包括模板机制、数据绑定、模块、指令、依赖注入、路由。通过数据与模板的绑定，能够让我们摆脱繁琐的 DOM 操作，而将注意力集中在业务逻辑上。

另外一个疑问，ng 是 MVC 框架吗？还是 MVVM 框架？官网有提到 ng 的设计采用了 MVC 的基本思想，而又不完全是 MVC，因为在书写代码时我们确实是在用 ng-controller 这个指令(起码从名字上看，是 MVC 吧)，但这个 controller 处理的业务基本上都是与 view 进行交互，这么看来又很接近 MVVM。让我们把目光移到官网那个非醒目的 title 上:“AngularJS — Superheroic JavaScript MVW Framework”。

2.2 AngularJS 简单介绍

AngularJS 重新定义了前端应用的开发方式。面对 HTML 和 JavaScript 之间的界线，它

非但不畏缩不前，反而正面出击，提出了有效的解决方案。

很多前端应用的开发框架，比如 Backbone、EmberJS 等，都要求开发者继承此框架特有的一些 JavaScript 对象。这种方式有其长处，但它不必要地污染了开发者自己代码的对象空间，还要求开发者去了解内存里那些抽象对象。尽管如此我们还是接受了这种方式，因为网络最初的设计无法提供 我们今天所需的交互性，于是我们需要框架，来帮我们填补 JavaScript 和 HTML 之间的鸿沟。而且有了它，你不用再“直接”操控 DOM，只要给你的 DOM 注上 metadata（即 AngularJS 里的 directive 们），然后让 AngularJS 来帮你操纵 DOM。同时，AngularJS 不依赖（也不妨碍）任何其他的框架。你甚至可以基于其它的框架来开发 AngularJS 应用。

API 地址：http://docs.angularjs.org/api/；

AngularJS 在 github 上的中文粗译版地址：https://github.com/basestyle/angularjs-cn。

2.3 什么时候该用 AngularJS

AngularJS 是一个 MV* 框架，最适于开发客户端的单页面应用。它不是个功能库，而是用来开发动态网页的框架。它专注于扩展 HTML 的功能，提供动态数据绑定（data binding），而且它能跟其它框架（如 jQuery）合作融洽。

如果你要开发的是单页应用，AngularJS 就是你的上上之选。Gmail、Google Docs、Twitter 和 Facebook 这样的应用，都很能发挥 AngularJS 的长处。但是像游戏开发之类对 DOM 进行大量操纵、又或者单纯需要 极高运行速度的应用，就不是 AngularJS 的用武之地了。

3 AugularJS 特性

AngularJS 是一个新出现的强大客户端技术，提供给大家的一种开发强大应用的方式。这种方式利用并且扩展 HTML，CSS 和 javascript，并且弥补了它们的一些非常明显的不足。本应该使用 HTML 来实现而现在由它开发的动态一些内容。

AngularJS 有五个最重要的功能和特性：

3.1 特性一：双向的数据绑定

数据绑定可能是 AngularJS 最酷最实用的特性。它能够帮助你避免书写大量的初始代码从而节约开发时间。一个典型的 web 应用可能包含了 80%的代码用来处理，查询和监听 DOM。数据绑定是的代码更少，你可以专注于你的应用。

我们想象一下 Model 是你的应用中的简单事实。你的 Model 是你用来读取或者更新的部分。数据绑定指令提供了你的 Model 投射到 view 的方法。这些投射可以无缝的，毫不影响的应用到 web 应用中。

传统来说，当 model 变化了。 开发人员需要手动处理 DOM 元素并且将属性反映到这些变化中。这个一个双向的过程。一方面，model 变化驱动了 DOM 中元素变化，另一方面，DOM 元素的变化也会影响到 Model。这个在用户互动中更加复杂，因为开发人员需要处理和解析

这些互动，然后融合到一个 model 中，并且更新 View。这是一个手动的复杂过程，当一个应用非常庞大的时候，将会是一件非常费劲的事情。

这里肯定有更好的解决方案!那就是 AngularJS 的双向数据绑定,能够同步 DOM 和 Model 等等。

这里有一个非常简单的例子,用来演示一个 input 输入框和<h1>元素的双向绑定(例 01)：

```
<!doctype html>
<html ng-app="demoApp">
  <head>
    <script src="./js/angular.min.js"></script>
  </head>
  <body>
    <div>
      <label>Name:</label>
      <input type="text" ng-model="user.name" placeholder="请输入名字">
      <hr>
      <h1>Hello, {{user.name}}!</h1>
    </div>
  </body>
</html>
```

说明：实际效果请大家看 AngularJS/demo/index.html

3.2 特性二：模板

在 AngularJS 中，一个模板就是一个 HTML 文件。但是 HTML 的内容扩展了，包含了很多帮助你映射 model 到 view 的内容。

HTML 模板将会被浏览器解析到 DOM 中。DOM 然后成为 AngularJS 编译器的输入。AngularJS 将会遍历 DOM 模板来生成一些指导，即，directive（指令）。所有的指令都负责针对 view 来设置数据绑定。

我们要理解 AuguarJS 并不把模板当做 String 来操作。输入 AngularJS 的是 DOM 而非 string。数据绑定是 DOM 变化，不是字符串的连接或者 innerHTML 变化。使用 DOM 作为输入，而不是字符串，是 AngularJS 区别于其它的框架的最大原因。使用 DOM 允许你扩展指令词汇并且可以创建你自己的指令，甚至开发可重用的组件。

最大的好处是为设计师和开发者创建了一个紧密的工作流。设计师可以像往常一样开发标签，然后开发者拿过来添加上功能，通过数据绑定将会使得这个过程非常简单。

这里有一个例子，我们使用 ng-repeat 指令来循环图片数组并且加入 img 模板，如下：

```
function AlbumCtrl($scope) {
    scope.images = [
        {"image":"img/image_01.png", "description":"Image 01 description"},
        {"image":"img/image_02.png", "description":"Image 02 description"},
        {"image":"img/image_03.png", "description":"Image 03 description"},
        {"image":"img/image_04.png", "description":"Image 04 description"},
        {"image":"img/image_05.png", "description":"Image 05 description"}
    ];
}
```

```
<div ng-controller="AlbumCtrl">
  <ul>
    <li ng-repeat="image in images">
      <img ng-src="{{image.thumbnail}}" alt="{{image.description}}">
    </li>
  </ul>
</div>
```

这里还有一件事值得提一句，AngularJS 并不强制你学习一个新的语法或者从你的应用中提出你的模板。

3.3 特性三：MVC

针对客户端应用开发 AngularJS 吸收了传统的 MVC 基本原则。MVC 或者 Model-View-Controll 设计模式针对不同的人可能意味不同的东西。AngularJS 并不执行传统意义上的 MVC，更接近于 MVVM（Moodel-View-ViewModel)。

Model

model 是应用中的简单数据。一般是简单的 javascript 对象。这里没有必要继承框架的 classes，使用 proxy 对象封装或者使用特别的 setter/getter 方法来访问。事实上我们处理 vanilla javascript 的方法就是一个非常好的特性，这种方法使得我们更少使用应用的原型。

ViewModel

viewmodel 是一个用来提供特别数据和方法从而维护指定 view 的对象。

viewmodel 是$scope 的对象，只存在于 AnguarJS 的应用中。$scope 只是一个简单的 js 对象，这个对象使用简单的 API 来侦测和广播状态变化。

Controller

controller 负责设置初始状态和参数化$scope 方法用以控制行为。需要指出的 controller 并不保存状态也不和远程服务互动。

View

view 是 AngularJS 解析后渲染和绑定后生成的 HTML 。这个部分帮助你创建 web 应用的架构。$scope 拥有一个针对数据的参考，controller 定义行为，view 处理布局和互动。

3.4 特性四：服务和依赖注入

AngularJS 服务其作用就是对外提供某个特定的功能。

AngularJS 拥有内建的依赖注入（DI）子系统，可以帮助开发人员更容易的开发，理解和测试应用。

DI 允许你请求你的依赖，而不是自己找寻它们。比如，我们需要一个东西，DI 负责找创建并且提供给我们。

为了而得到核心的 AngularJS 服务，只需要添加一个简单服务作为参数，AngularJS 会侦测并且提供给你：

```
function EditCtrl($scope, $location, $routeParams) {
    // Something clever here...
}
```

你也可以定义自己的服务并且让它们注入：

```
angular.module('MyServiceModule', []).
    factory('notify', ['$window', function (win) {
    return function (msg) {
        win.alert(msg);
    };
}]);
function myController(scope, notifyService) {
    scope.callNotify = function (msg) {
        notifyService(msg);
    };
}
myController.$inject = ['$scope', 'notify'];
```

3.5 特性五：指令（Directives）

指令是我个人最喜欢的特性。你是不是也希望浏览器可以做点儿有意思的事情？那么AngularJS 可以做到。

指令可以用来创建自定义的标签。它们可以用来装饰元素或者操作 DOM 属性。可以作为标签、属性、注释和类名使用。

这里是一个例子，它监听一个事件并且针对的更新它的$scope ，如下：

```
myModule.directive('myComponent', function(mySharedService) {
    return {
        restrict: 'E',
        controller: function($scope, $attrs, mySharedService) {
            $scope.$on('handleBroadcast', function() {
                $scope.message = 'Directive: ' + mySharedService.message;
            });
        },
        replace: true,
        template: '<input>'
    };
});
```

然后，你可以使用这个自定义的 directive 来使用：

```
<my-component ng-model="message"></my-component>
```

使用一系列的组件来创建你自己的应用将会让你更方便的添加，删除和更新功能。

4 功能介绍

4.1 数据绑定

AngularJS 的双向数据绑定，意味着你可以在 Mode(JS)中改变数据，而这些变动立刻就会自动出现在 View 上，反之亦然。即：一方面可以做到 model 变化驱动了 DOM 中元素变化，另一方面也可以做到 DOM 元素的变化也会影响到 Model。

在我们使用 jQuery 的时候，代码中会大量充斥类似这样的语句：var val = $(‘#id’).val(); $(‘#id’).html(str);等等，即频繁的 DOM 操作（读取和写入），其实我们的最终目的并不是要操作 DOM，而是要实现业务逻辑。ng 的绑定将让你摆脱 DOM 操作，只要模板与数据通过声明进行了绑定，两者将随时保持同步，最新的数据会实时显示在页面中，页面中用户修改的数据也会实时被记录在数据模型中。

从 View 到 Controller 再到 View 的数据交互（例 01）：

```
<html ng-app="demoApp">
……
<input type="text" ng-model="user.name" placeholder="请输入名称"/>
Hello, {{ user.name }}!
……
```

关键： ng-app 、 ng-model 和 { {user.name } }

首先： <html>元素的 ng-app 属性。标识这个 DOM 里面的内容将启用 AngularJS 应用。

其次：告诉 AngularJS，对页面上的“user.name” 这个 Model 进行双向数据绑定。

第三：告诉 AngularJS，在“{{ user.name}}”这个指令模版上显示“user.name”这个 Model 的数据。

从 Server 到 Controller 再到 View 的数据交互（例 02）：

```
<html ng-app="demoApp">
……
<div   ng-controller="demoController">
<input type="text" ng-model="user.name" disabled="disabled"/>
<a href="javascript:void(0);" ng-click="getAjaxUser()">AJAX 获取名字</a>
……
demoApp.controller("demoController", function($http, $scope){
    $scope. getAjaxUser = function(){
//          $http.get({url:"../xxx.action"}).success(function(data){
//                $scope.user= data;
//          });
          $scope.user = {"name":"从 JOSN 中获取的名称","age":22};
    };
});
```

改变$scope 中的 user，View 也会自动更新。

4.2 scopes、module、controller

4.2.1 scopes

$scope 是一个把 view（一个 DOM 元素）连结到 controller 上的对象。在我们的 MVC 结构里，这个 $scope 将成为 model，它提供一个绑定到 DOM 元素（以及其子元素）上的 excecution context。

尽管听起来有点复杂，但 $scope 实际上就是一个 JavaScript 对象，controller 和 view 都可以访问它，所以我们可以利用它在两者间传递信息。在这个 $scope 对象里，我们既存储数据，又存储将要运行在 view 上的函数。

每一个 Angular 应用都会有一个 $rootScope。这个 $rootScope 是最顶级的 scope，它对应着含有 ng-app 指令属性的那个 DOM 元素。

app.run(function($rootScope) { $rootScope.name = "张三"; });

如果页面上没有明确设定 $scope ，Angular 就会把数据和函数都绑定到这里， 第一部分中的例子就是靠这一点成功运行的。

这样，我们就可以在 view 的任何地方访问这个 name 属性，使用模版表达式{{}}，像这样：

{{ name }}

4.2.2 module

首先需要明确一下模板的概念。在我还不知道有模板这个东西的时候，曾经用 js 拼接出很长的 HTML 字符串，然后 append 到页面中，这种方式想想真是又土又笨。后来又看到可以把 HTML 代码包裹在一个<script>标签中当作模板，然后按需要取来使用。

在 ng 中，模板十分简单，它就是我们页面上的 HTML 代码，不需要附加任何额外的东西。在模板中可以使用各种指令来增强它的功能，这些指令可以让你把模板和数据巧妙的绑定起来。

在<html>标签上多了一个属性 ng-app=”MyApp”，它的作用就是用来指定 ng 的作用域是在<html>标签以内部分。在 js 中，我们调用 angular 对象的 module 方法来声明一个模块，模块的名字和 ng-app 的值对应。这样声明一下就可以让 ng 运行起来了。

示例：

```
<html ng-app="demoApp">
var demoApp = angular.module('demoApp', []);
```

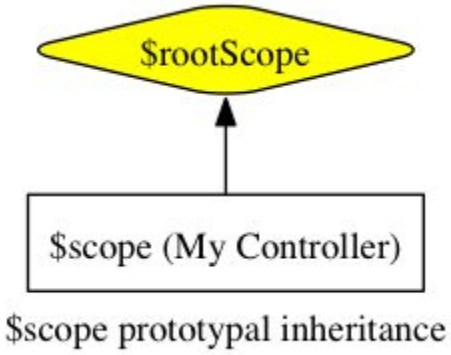
4.2.3 ng-controller

要明确创建一个$scope 对象，我们就要给 DOM 元素安上一个 controller 对象，使用的是 ng-controller 指令属性：

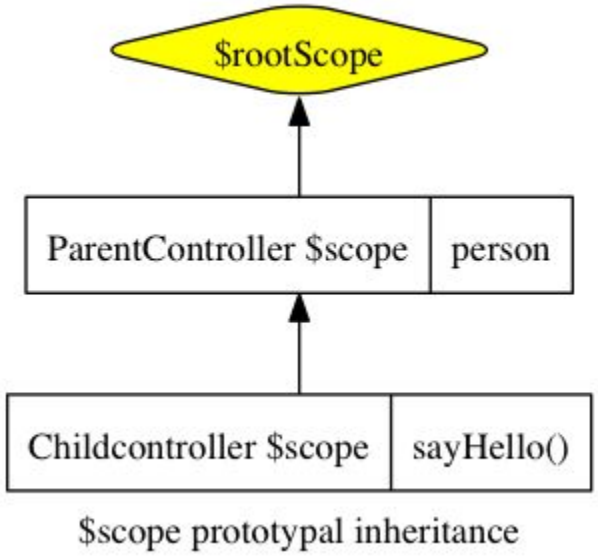
```
<div ng-controller="MyController"> {{ person.name }} </div>
```

ng-controller 指令给所在的 DOM 元素创建了一个新的$scope 对象，并将这个$scope 对象包含进外层 DOM 元素的$scope 对象里。在上面的例子里，这个外层 DOM 元素的$scope 对

象，就是$rootScope 对象。这个 scope 链是这样的：



所有 scope 都遵循原型继承（prototypal inheritance），这意味着它们都能访问父 scope 们。对任何属性和方法，如果 AngularJS 在当前 scope 上找不到，就会到父 scope 上去找，如果在父 scope 上也没找到，就会继续向上回溯，一直到$rootScope 上。即如果 controller 是多层嵌套的，就会从最里面一直往外找，这个 scope 链是这样的：



唯一的例外：有些指令属性可以选择性地创建一个独立的 scope，让这个 scope 不继承它的父 scope 们，这个会在指令详解中说明。

4.3 ajax

$http 服务是 AngularJS 的核心服务之一，它帮助我们通过 XMLHttpRequest 对象或 JSONP 与远程 HTTP 服务进行交流。

$http 服务是这样一个函数：它接受一个设置对象，其中指定了如何创建 HTTP 请求；它将返回一个承诺（*参考 JavaScript 异步编程的 promise 模式），其中提供两个方法：success 方法和 error 方法。

```
demoApp.controller("demoController", function($http, $scope){
    $scope. getAjaxUser = function(){
        $http.get({url:"../xxx.action"}).success(function(data){
            alert(data);
        }).error(function(){
```

```
            Alert("出错了！");
        });

    };
});
```

AngularJS 的 AJAX 与 jquery 等框架的 AJAX 基本一致，这里就不多说了。

4.4 表达式

ng 中的表达式与 javascript 表达式类似但是不可以划等号，它是 ng 自己定义的一套模式。表达式可以作为指令的值，如 ng-modle=”people.name”、ng-click=”showMe()”，看起来是如此像字符串，故而也叫字符串表达式。也可以在标记中使用表达式，如{{1+2}}，或者与过滤器一起使用{{1+2 | currency}}。在框架内部，字符串不会简单的使用 eval()来执行，而是有一个专门的$parse 服务来处理。在 ng 表达式中不可以使用循环语句、判断语句，事实上在模板中使用复杂的表达式也是一个不推荐的做法，这样视图与逻辑就混杂在一起了

我们在使用其他模板库时，一般都会有模板的循环输出、分支输出、逻辑判断等类似的控制。

要想理解指令属性的运作，我们必须先理解表达式。在之前的例子里我们已经见过表达式，例如 {{ user.name }}。

请查看例 03、例 04、例 05。

{{ 8 + 1 }}　　9

{{ person }}　　{"name":"Ari Lerner"}

{{ 10 * 3.3 | currency }}　　$33.00

表达式粗略来看有点像 eval(javascript) 的结果。它们会经过 Angular.js 的处理，从而拥有以下重要而独特的性质：

- 所有表达式都在 scope 这个 context 里被执行，因此可以使用所有本地 $scope 中的变量。
- 如果一个表达式的执行导致类型错误或引用错误，这些错误将不会被抛出。
- 表达式里不允许任何控制函数流程的功能（如 if/else 等条件语句）
- 表达式可接受一个或多个串联起来的过滤器。

4.5 过滤器

过滤器（filter）正如其名，作用就是接收一个输入，通过某个规则进行处理，然后返回处理后的结果。主要用在数据的格式化上，例如获取一个数组中的子集，对数组中的元素进行排序等。过滤器通常是伴随标记来使用的，将你 model 中的数据格式化为需要的格式。表单的控制功能主要涉及到数据验证以及表单控件的增强。ng 内置了一些过滤器，它们是：currency(货币)、date(日期)、filter(子串匹配)、json(格式化 json 对象)、limitTo(限制个数)、lowercase(小写)、uppercase(大写)、number(数字)、orderBy(排序)。

4.5.1 过滤器使用方式

总共九种。除此之外还可以自定义过滤器，这个就强大了，可以满足任何要求的数据处理。Filter 还是很简单的，需要明白的是内置的 filter 如何使用，以及自己如何定义一个 filter。filter 的两种使用方法：

1. 在模板中使用 filter

我们可以直接在{{}}中使用 filter，跟在表达式后面用 | 分割，语法如下：

```
{{ expression | filter }}
```

也可以多个 filter 连用，上一个 filter 的输出将作为下一个 filter 的输入：

```
{{ expression | filter1 | filter2 | ... }}
```

filter 可以接收参数，参数用 : 进行分割，如下：

```
{{ expression | filter:argument1:argument2:... }}
```

除了对{{}}中的数据进行格式化，我们还可以在指令中使用 filter，例如先对数组 array 进行过滤处理，然后再循环输出：

```
<span ng-repeat="a in array | filter ">
```

2. 在 controller 和 service 中使用 filter

我们的 js 代码中也可以使用过滤器，方式就是我们熟悉的依赖注入，例如我要在 controller 中使用 currency 过滤器，只需将它注入到该 controller 中即可，代码如下：

```
app.controller('testC',function($scope,currencyFilter){
    $scope.num = currencyFilter(123534);
}
```

在模板中使用{{num}}就可以直接输出$123,534.00 了！在服务中使用 filter 也是同样的道理。

如果你要在 controller 中使用多个 filter，并不需要一个一个注入吗，ng 提供了一个$filter 服务可以来调用所需的 filter，你只需注入一个$filter 就够了，使用方法如下：

```
app.controller('testC',function($scope,$filter){
    $scope.num = $filter('currency')(123534);
    $scope.date = $filter('date')(new Date());
}
```

可以达到同样的效果。好处是你可以方便使用不同的 filter 了。

4.5.2 ng 的内置过滤器

ng 内置了九种过滤器，使用方法都非常简单，看文档即懂。不过为了以后不去翻它的文档，我在这里还是做一个详细的记录。

currency(货币)、date(日期)、filter(子串匹配)、json(格式化 json 对象)、limitTo(限制个数)、lowercase(小写)、uppercase(大写)、number(数字)、orderBy(排序)

1. currency (货币处理)

使用 currency 可以将数字格式化为货币，默认是美元符号，你可以自己传入所需的符号，例如我传入人民币：

{{num | currency : '￥'}}

2. date (日期格式化)

原生的 js 对日期的格式化能力有限，ng 提供的 date 过滤器基本可以满足一般的格式化要求。用法如下：

{{date | date : 'yyyy-MM-dd hh:mm:ss EEEE'}}

参数用来指定所要的格式，y M d h m s E 分别表示 年 月 日 时 分 秒 星期，你可以自由组合它们。也可以使用不同的个数来限制格式化的位数。另外参数也可以使用特定的描述性字符串，例如“shortTime”将会把时间格式为 12:05 pm 这样的。ng 提供了八种描述性的字符串，个人觉得这些有点多余，我完全可以根据自己的意愿组合出想要的格式，不愿意去记这么多单词~

3. filter(匹配子串)

这个名叫 filter 的 filter。用来处理一个数组，然后可以过滤出含有某个子串的元素，作为一个子数组来返回。可以是字符串数组，也可以是对象数组。如果是对象数组，可以匹配属性的值。它接收一个参数，用来定义子串的匹配规则。下面举个例子说明一下参数的用法，我用现在特别火的几个孩子定义了一个数组：

```
$scope.childrenArray = [
        {name:'kimi',age:3},
        {name:'cindy',age:4},
        {name:'anglar',age:4},
        {name:'shitou',age:6},
        {name:'tiantian',age:5}
    ];
$scope.func = function(e){return e.age>4;}{{ childrenArray | filter : 'a' }} //匹配属性值中含有 a 的
{{ childrenArray | filter : 4 }}   //匹配属性值中含有 4 的
{{ childrenArray | filter : {name : 'i'} }} //参数是对象，匹配 name 属性中含有 i 的
{{childrenArray | filter : func }}   //参数是函数，指定返回 age>4 的
```

4. json(格式化 json 对象)

json 过滤器可以把一个 js 对象格式化为 json 字符串，没有参数。这东西有什么用呢，我一般也不会在页面上输出一个 json 串啊，官网说它可以用来进行调试，嗯，是个不错的选择。或者，也可以用在 js 中使用，作用就和我们熟悉的 JSON.stringify()一样。用法超级简单：

{{ jsonTest | json}}

5. limitTo(限制数组长度或字符串长度)

limitTo 过滤器用来截取数组或字符串，接收一个参数用来指定截取的长度，如果参数是负值，则从数组尾部开始截取。个人觉得这个 filter 有点鸡肋，首先只能从数组或字符串的开头/尾部进行截取，其次，js 原生的函数就可以代替它了，看看怎么用吧：

```
{{ childrenArray | limitTo : 2 }}  //将会显示数组中的前两项
```

6. lowercase(小写)

把数据转化为全部小写。太简单了，不多解释。同样是很鸡肋的一个 filter，没有参数，只能把整个字符串变为小写，不能指定字母。怎么用我都懒得写了。

7. uppercase(大写)

同上。

8. number(格式化数字)

number 过滤器可以为一个数字加上千位分割，像这样，123,456,789。同时接收一个参数，可以指定 float 类型保留几位小数：

```
{{ num | number : 2 }}
```

9. orderBy(排序)

orderBy 过滤器可以将一个数组中的元素进行排序，接收一个参数来指定排序规则，参数可以是一个字符串，表示以该属性名称进行排序。可以是一个函数，定义排序属性。还可以是一个数组，表示依次按数组中的属性值进行排序（若按第一项比较的值相等，再按第二项比较），还是拿上面的孩子数组举例：

```
<div>{{ childrenArray | orderBy : 'age' }}</div>          //按 age 属性值进行排序，若是-age，则倒序
<div>{{ childrenArray | orderBy : orderFunc }}</div>     //按照函数的返回值进行排序
<div>{{ childrenArray | orderBy : ['age','name'] }}</div>  //如果 age 相同，按照 name 进行排序
```

内置的过滤器介绍完了，写的我都快睡着了。。。正如你所看到的，ng 内置的过滤器也并不是万能的，事实上好多都比较鸡肋。更个性化的需求就需要我们来定义自己的过滤器了，下面来看看如何自定义过滤器。

4.5.3 自定义过滤器及示例

filter 的自定义方式也很简单，使用 module 的 filter 方法，返回一个函数，该函数接收

输入值，并返回处理后的结果。话不多说，我们来写一个看看。比如我需要一个过滤器，它可以返回一个数组中下标为奇数的元素，代码如下：

```
app.filter('odditems',function(){
    return function(inputArray){
        var array = [];
        for(var i=0;i<inputArray.length;i++){
            if(i%2!==0){
                array.push(inputArray[i]);
            }
        }
        return array;
    }
});
```

格式就是这样，你的处理逻辑就写在内部的那个闭包函数中。你也可以让自己的过滤器接收参数，参数就定义在 return 的那个函数中，作为第二个参数，或者更多个参数也可以。

自定义过滤器实例（例 04）：

```
/* View html */
First name：<input ng-model="user.firstName"/><br/>
Last  name：<input ng-model="user.lastName"/> <br/>
First name：{{user.firstName}}      Last  name：{{user.lastName}}
<br/>
Fullname：{{user | flFullname}}<br/>
Fullname：{{user | flFullname:"－"}}<br/>
Fullname：{{user | flFullname:"•" | uppercase }}
/* Controller js */
demoApp.filter("flFullname", function() {
    return function(user, sep) {
        sep = sep || " ";
        user = user || {};
        fullName = "";
        if(user.firstName){fullName += user.firstName;}
        if(user.lastName){fullName = fullName + sep + user.lastName;}
        if(fullName && fullName.length>0){return fullName;
        }else{return "";}
    };
});
```

4. 6 指令(directive)

通过使用模板，我们可以把 model 和 controller 中的数据组装起来呈现给浏览器，还可以通过数据绑定，实时更新视图，让我们的页面变成动态的。

模板中可以使用的东西包括以下四种：

1.指令(directive)：ng 提供的或者自定义的标签和属性，用来增强 HTML 表现力；

2.标记(markup)：即双大括号{{}}，可将数据单向绑定到 HTML 中；

3.过滤器(filter)：用来格式化输出数据；

4.表单控制：用来增强表单的验证功能。

其中，指令无疑是使用量最大的，ng 内置了很多指令用来控制模板，如 ng-repeat，ng-class，也有很多指令来帮你完成业务逻辑，如 ng-controller,ng-model。

指令的几种使用方式如下：

- 作为标签：<my-dir></my-dir>
- 作为属性：
- 作为注释：<!-- directive: my-dir exp -->
- 作为类名：

其实常用的就是作为标签和属性。

4.6.1 样式相关的指令

既然模板就是普通的 HTML，那我首要关心的就是样式的控制，元素的定位、字体、背景色等等如何可以灵活控制。下面来看看常用的样式控制指令。

1. ng-class

ng-class 用来给元素绑定类名，其表达式的返回值可以是以下三种：

- 类名字符串，可以用空格分割多个类名，如’redtext boldtext’；
- 类名数组，数组中的每一项都会层叠起来生效；
- 一个名值对应的 map，其键值为类名，值为 boolean 类型，当值为 true 时，该类会被加在元素上。

下面来看一个使用 map 的例子：

ng-class 测试

红色 加粗 删除线

map:{redtext:{{red}}, boldtext:{{bold}}, striketext:{{strike}}}

如果你想拼接一个类名出来，可以使用插值表达式，如：

字体样式测试

然后在 controller 中指定 style 的值：

$scope.style = ‘red’;

注意我用了 class 而不是 ng-class，这是不可以对换的，官方的文档也未做说明，姑且认为这是 ng 的语法规则吧。

与 ng-class 相近的，ng 还提供了 ng-class-odd、ng-class-even 两个指令，用来配合 ng-repeat 分别在奇数列和偶数列使用对应的类。这个用来在表格中实现隔行换色再方便不过了。

2. ng-style

ng-style 用来绑定元素的 css 样式，其表达式的返回值为一个 js 对象，键为 css 样式名，值为该样式对应的合法取值。用法比较简单：

```
<div ng-style="{color:'red'}">ng-style 测试</div>
<div ng-style="style">ng-style 测试</div>
$scope.style = {color:'red'};
```

3. ng-show，ng-hide

对于比较常用的元素显隐控制，ng 也做了封装，ng-show 和 ng-hide 的值为 boolean 类型的表达式，当值为 true 时，对应的 show 或 hide 生效。框架会用 display:block 和 display:none 来控制元素的显隐。

4.6.2 表单控件功能相关指令

对于常用的表单控件功能，ng 也做了封装，方便灵活控制。

ng-checked 控制 radio 和 checkbox 的选中状态

ng-selected 控制下拉框的选中状态

ng-disabled 控制失效状态

ng-multiple 控制多选

ng-readonly 控制只读状态

以上指令的取值均为 boolean 类型，当值为 true 时相关状态生效，道理比较简单就不多做解释。注意： 上面的这些只是单向绑定，即只是从数据到模板，不能反作用于数据。要双向绑定，还是要使用 ng-model 。

4.6.3 事件绑定相关指令

事件绑定是 javascrpt 中比较重要的一部分内容，ng 对此也做了详细的封装，正如我们之前使用过的 ng-click 一样，事件的指令如下：

ng-click

ng-change

ng-dblclick

ng-mousedown

ng-mouseenter

ng-mouseleave

ng-mousemove

ng-mouseover

ng-mouseup

ng-submit

事件绑定指令的取值为函数，并且需要加上括号，例如：

```
<select ng-change=”change($event)”></select>
```

然后在 controller 中定义如下：

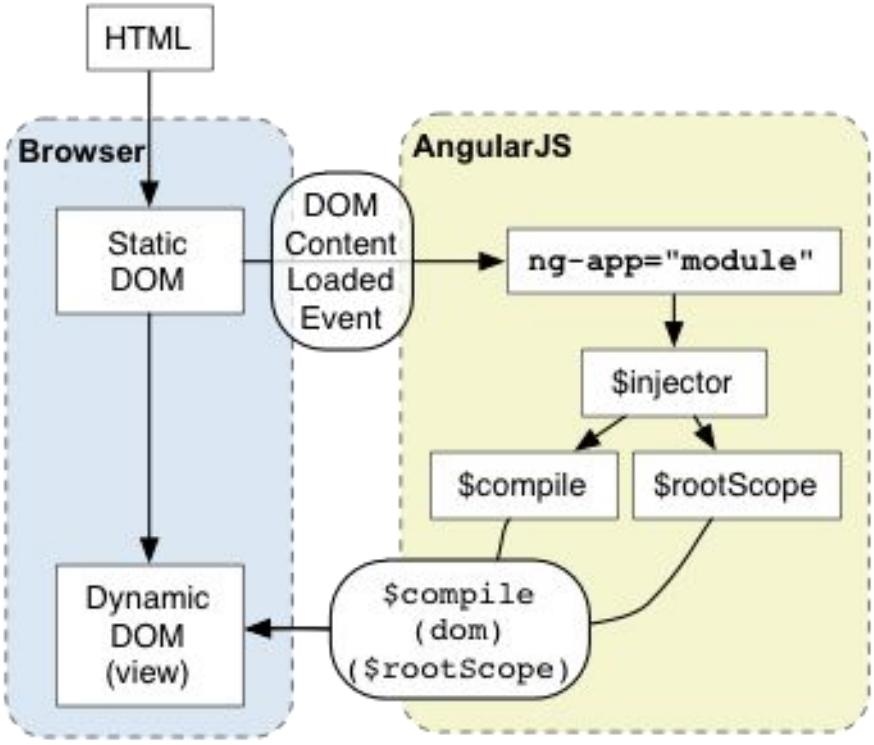
```
$scope.change = function($event){
        alert($event.target);
        //……
}
```

在模板中可以用变量$event 将事件对象传递到 controller 中。

对于 ng 的这种设计，一些人有所质疑，视图与事件绑定混在一起到底好不好？我们不是要讲究视图与逻辑分离吗？如此一来，把事件的绑定又变回了内联的，岂不是历史的倒退。我也一样对此表示不解，因为不写 onclick 已经很多年。。。但既然已经存在了，我们不妨往合理的方向上想一想，或许 ng 的设计者压根就不想让模板成为单纯的视图层，本来就是想增强 HTML，让它有一点业务能力。这么想的话似乎也能想通，好吧，先欺骗一下自己吧~

4.6.4 特殊的 ng-src 和 ng-href

在说明这两个指令的特殊之前，需要先了解一下 ng 的启动及执行过程，如下图：



1) 浏览器加载静态 HTML 文件并解析为 DOM；
2) 浏览器加载 angular.js 文件；
3) angular 监听 DOMContentLoaded 事件，监听到时开始启动；
4) angular 寻找 ng-app 指令，确定作用范围；
5) 找到 app 中定义的 Module 使用$injector 服务进行依赖注入；
6) 根据$injector 服务创建$compile 服务用于编译；
7) $compile 服务编译 DOM 中的指令、过滤器等；
8) 使用 ng-init 指令，将作用域中的变量进行替换；
9) 最后生成了我们在最终视图。

可以看到，ng 框架是在 DOMcontent 加载完毕后才开始发挥作用。假如我们模板中有一张图片如下：

那么在页面开始加载到 ng 编译完成之前，页面上会一直显示一张错误的图片，因为路径{{imgUrl}}还未被替换。

为了避免这种情况，我们使用 ng-src 指令，这样在路径被正确得到之前就不会显示找不到图片。同理，<a>标签的 href 属性也需要换成 ng-href，这样页面上就不会先出现一个地址错误的链接。

顺着这个思路再多想一点，我们在模板中使用{{}}显示数据时，在 ng 编译完成之前页面上岂不是会显示出大括号及里面的表达式？确实是这样。为了避免这个，ng 中有一个与{{}}等同的指令:ng-bind，同样用于单向绑定，在页面刚加载的时候就不会显示出对用户无用的数据了。尽管这样你可能不但没舒心反而更纠结了，{{}}那么好用易理解，还不能用了不成？好消息是我们依然可以使用。因为我编写的是单页面应用，页面只会在加载 index.html 的时

候出这个问题，只需在 index.html 中的模板中换成 ng-bind 就行。其他的模板是我们动态加载的，就可以放心使用{{}}了。

4.6.5 自定义指令示例

下面我们来解析下指令的例子（例 07）。

1.首先，我们定义一个名为 userInfo 的指令：

```
demoApp.directive('userInfo',function(){
    return {
        restrict : 'E',
        templateUrl : 'userInfoTemplate.html',
        replace : true,
        transclude : true,
        scope : {
            mytitle : '=etitle'
        },
        link : function(scope,element,attris){
            scope.showText = false;
            scope.toggleText = function(){
                scope.showText = ! scope.showText;
            }
        }
    };
})
```

Restrict 为'E'：用作标签；replace 为 true：用模板替换当前标签；transclude 为 true：将当前元素的内容转移到模板中；scope 为 {mytitle : '=etitle'}：定义一个名为 mytitle 的 MODEL，其值指向当前元素的 etitle 属性； templateUrl 为'userInfoTemplate.html'：模板内容为 ng-template 定义 ID 为 userInfoTemplate.html 的内容；link：指定所包含的行为。其具体的说明及其他参数，请参考：6.2 指令详解。

2. userInfoTemplate.html 模板为：

```
<script type="text/ng-template" id="userInfoTemplate.html">
    <div class="mybox">
        <div class="mytitle" style="cursor: pointer;" ng-click="toggleText()">
            { {mytitle} }
        </div>
        <div ng-transclude ng-show="showText">
        </div>
    </div>
</script>
```

将当前元素的内容添加到有 ng-transclude 属性的这个 DIV 下，默认是隐藏的。

3.Controller 信息：

```
demoApp.controller("test7Controller", function($scope){
    $scope.title = '个人简介';
    $scope.text = '大家好，我正在研究 AngularJs，欢迎大家与我交流。';
```

```
    $scope.updateInfo = function (){
        $scope.title = '个人信息';
        $scope.text = '大家好，今天天气真好！';
    }
});
```

4.指令使用方式（View 信息）为：

```
<user-info etitle="title">{ {text} }</user-info>
```

Etitle 指向 Controller 中的$scope.title。注意命名方式：指令名为 userInfo，对应的标签为 user-info。

4. 7 服务（service）

4.7.1 服务介绍

服务这个概念其实并不陌生，在其他语言中如 java 便有这样的概念，其作用就是对外提供某个特定的功能，如消息服务，文件压缩服务等，是一个独立的模块。ng 的服务是这样定义的：

Angular services are singletons objects or functions that carry out specific tasks common to web apps.

它是一个单例对象或函数，对外提供特定的功能。

首先是一个单例，即无论这个服务被注入到任何地方，对象始终只有一个实例。

其次这与我们自己定义一个 function 然后在其他地方调用不同，因为服务被定义在一个模块中，所以其使用范围是可以被我们管理的。ng 的避免全局变量污染意识非常强。

ng 提供了很多内置的服务，可以到 API 中查看 http://docs.angularjs.org/api/。知道了概念，我们来拉一个 service 出来溜溜，看看到底是个什么用法。

我们在 controller 中直接声明$location 服务，这依靠 ng 的依赖注入机制。$location 提供地址栏相关的服务，我们在此只是简单的获取当前的地址。

服务的使用是如此简单，我们可以把服务注入到 controller、指令或者是其他服务中。

4.7.2 自定义服务

如同指令一样，系统内置的服务以$开头，我们也可以自己定义一个服务。定义服务的方式有如下几种：

- 使用系统内置的$provide 服务；
- 使用 Module 的 factory 方法；
- 使用 Module 的 service 方法。

下面通过一个小例子来分别试验一下。我们定义一个名为 remoteData 服务，它可以从远程获取数据，这也是我们在程序中经常使用的功能。不过我这里没有远程服务器，就写死一点数据模拟一下。

```
//使用$provide 来定义
var app = angular.module('MyApp', [], function($provide) {
    $provide.factory('remoteData', function() {
```

```
        var data = {name:'n',value:'v'};
        return data;
    });
});
//使用 factory 方法
app.factory('remoteData',function(){
    var data = {name:'n',value:'v'};
    return data;
});
//使用 service 方法
app.service('remoteData',function(){
    this.name = 'n';
    this.value = 'v';
});
```

Module 的 factory 和$provide 的 factory 方法是一模一样的，从官网文档看它们其实就是一回事。至于 Module 内部是如何调用的，我此处并不打算深究，我只要知道怎么用就好了。

再看 Module 的 service 方法，它没有 return 任何东西，是因为 service 方法本身返回一个构造器，系统会自动使用 new 关键字来创建出一个对象。所以我们看到在构造器函数内可以使用 this，这样调用该服务的地方便可以直接通过 remoteData.name 来访问数据了。

4.7.3 管理服务的依赖关系

服务与服务中间可以有依赖关系，例如我们这里定义一个名为 validate 的服务，它的作用是验证数据是否合法，它需要依赖我们从远程获取数据的服务 remoteData。代码如下：

在 factory 的参数中，我们可以直接传入服务 remoteData，ng 的依赖注入机制便帮我们做好了其他工作。不过一定要保证这个参数的名称与服务名称一致，ng 是根据名称来识别的。若参数的名次与服务名称不一致，你就必须显示的声明一下，方式如下：

```
app.factory('validate',['remoteData',function(remoteDataService){
    return function(){
        if(remoteDataService.name=='n'){
            alert('验证通过');
        }
    };
}]);
```

我们在 controller 中注入服务也是同样的道理，使用的名称需要与服务名称一致才可以正确注入。否则，你必须使用$inject 来手动指定注入的服务。比如：

```
function testC(scope,rd){
    scope.getData = function(){
        alert('name：'+rd.name+'     value：'+rd.value);
    }
}
testC.$inject = ['$scope','remoteData'];
```

在 controller 中注入服务，也可以在定义 controller 时使用数组作为第二个参数，在此处

把服务注入进去，这样在函数体中使用不一致的服务名称也是可以的，不过要确保注入的顺序是一致的，如：

```
app.controller('testC',['$scope','remoteData',function($scope,rd){
    $scope.getData = function(){
        alert('name：'+rd.name+'    value：'+rd.value);
    }
}]);
```

4.7.4 自定义服务示例

接下来让我们看下例子（例 08 自定义服务）代码，自定义 userService 服务：

```
demoApp.factory('userService', ['$http', function($http) {
    var doGetUser = function(userId, path) {
        //return $http({
            //method: 'JSONP',
            //url: path
        //});
        /*手动指定数据*/
        var data = {userId:"woshishui",userName:"我是谁",userInfo:"我是谁！我是谁！"};;
        if(userId=='zhangsan'){
            data = {userId:"zhangsan",userName:"张三",userInfo:"我是张三，我为自己"};
        }else if(userId=='lisi'){
            data = {userId:"lisi",userName:"李四",userInfo:"我是李四，我为卿狂！"};
        }
        return data;
    }
    return {
        /*userService 对外暴露的函数，可有多个*/
        getUser: function(userId) {
            return doGetUser(userId, '../xxx/xxx.action');
        }
    };
}]);
```

我们创建了一个只有一个方法的 userService，getUser 为这个服务从后台获取用户信息的函数，并且对外暴露。当然，由于这是一个静态的例子，无法访问后台，那么我们便制定其返回的数据。

然后我们把这个服务添加到我们的 controller 中。我们建立一个 controller 并加载（或者注入）userService 作为运行时依赖，我们把 service 的名字作为参数传递给 controller 函数：

```
demoApp.controller("test8Controller", function($scope,userService){
        /*文章信息*/
        $scope.articles = [{
            title : "爱飞像风",
            userId : "zhangsan",
            userName : "张三"
```

```
    },{
        title : "无法停止的雨",
        userId : "lisi",
        userName : "李四"
    }];
    $scope.showUserInfo = false;//显示作者详细信息开关
    $scope.currentUser = {}; //当前选中的作者
    $scope.getUserInfo = function(userId){
        $scope.currentUser = userService.getUser(userId);
        //调用 userService的getUser函数
        $scope.showUserInfo = true;
        setTimeout(function(){//定时器：隐藏作者详细信息
            $scope.showUserInfo = false;
        },3000);
    }
});
```

我们的 userService 注入到我们的 test8Controller 后，我们就可以像使用其他服务（我们前面提到的$http 服务）一样的使用 userService 了。

相关的 HTML 代码如下：

```
/* View HTML*/
<tr ng-repeat="article_ in articles">
    <td>
        {{article_.title}}
    </td>
    <td>
        <a href="javascript:void(0);" ng-click="getUserInfo(article_.userId)">
        {{article_.userName}} </a>
    </td>
</tr>
.....
<div ng-show="showUserInfo">
    用户 ID：{{currentUser.userId}}<br/>
    用户名：{{currentUser.userName}}<br/>
    用户简介：{{currentUser.userInfo}}<br/>
</div>
```

4.8 依赖注入 DI

通过依赖注入，ng 想要推崇一种声明式的开发方式，即当我们需要使用某一模块或服务时，不需要关心此模块内部如何实现，只需声明一下就可以使用了。在多处使用只需进行多次声明，大大提高可复用性。

比如我们的 controller，在定义的时候用到一个$scope 参数。

```
app.controller('testC',function($scope){});
```

如果我们在此处还需操作其他的东西，比如与浏览器地址栏进行交互。我们只需再多添

一个参数$location 进去：

```
app.controller('testC',function($scope,$location){});
```

这样便可以通过$location 来与地址栏进行交互了，我们仅仅是声明了一下，所需的其他代码，框架已经帮我们注入了。我们很明显的感觉到了这个函数已经不是常规意义上的 javascript 函数了，在常规的函数中，把形参换一个名字照样可以运行，但在此处若是把$scope 换成别的名字，程序便不能运行了。因为这是已经定义好的服务名称。

这便是依赖注入机制。顺理成章的推断，我们可以自己定义模块和服务，然后在需要的地方进行声明，由框架来替我们注入。

来看下我们如何定义一个服务：

```
app.factory('tpls',function(){
    return ['tpl1','tpl2','tpl3','tpl4'];
});
```

看上去相当简单，是因为我在这里仅仅是直接返回一个数组。在实际应用中，这里应该是需要向服务器发起一个请求，来获取到这些模板们。服务的定义方式有好几种，包括使用 provider 方法、使用 factory 方法，使用 service 方法。它们之间的区别暂且不关心。我们现在只要能创建一个服务出来就可以了。我使用了 factory 方法。一个需要注意的地方是，框架提供的服务名字都是由$开头的，所以我们自己定义的最好不要用$开头，防止发生命名冲突。

定义好一个服务后，我们就可以在控制器中声明使用了，如下：

```
app.controller('testC',function($scope,tpls){
    $scope.question = questionModel;
    $scope.nowTime = new Date().valueOf();
    $scope.templates = tpls; //赋值到$scope 中
    $scope.addOption = function(){
        var o = {content:''};
        $scope.question.options.push(o);
    };
    $scope.delOption = function(index){
        $scope.question.options.splice(index,1);
    };
});
```

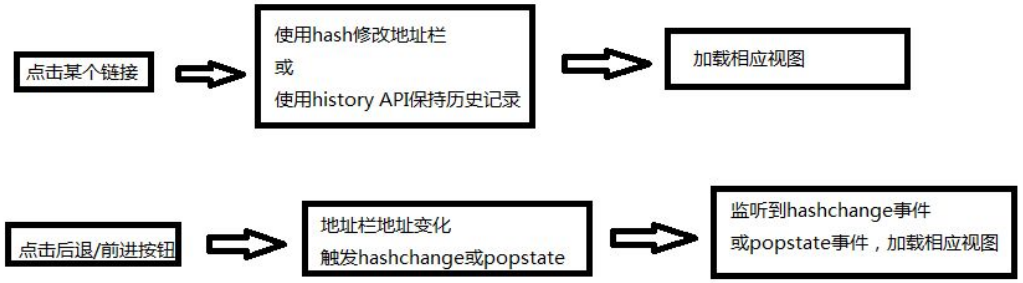
此时，若在模板中书写如下代码，我们便可以获取到服务 tpls 所提供的数据了：

模板:

```
<a href="javascript:void(0);" ng-repeat="t in templates">{{t}}  </a><br />
```

4.9 路由（route）

在谈路由机制前有必要先提一下现在比较流行的单页面应用，就是所谓的 single page APP。为了实现无刷新的视图切换，我们通常会用 ajax 请求从后台取数据，然后套上 HTML 模板渲染在页面上，然而 ajax 的一个致命缺点就是导致浏览器后退按钮失效，尽管我们可以在页面上放一个大大的返回按钮，让用户点击返回来导航，但总是无法避免用户习惯性的点后退。解决此问题的一个方法是使用 hash，监听 hashchange 事件来进行视图切换，另一个方法是用 HTML5 的 history API，通过 pushState()记录操作历史，监听 popstate 事件来进行视图切换，也有人把这叫 pjax 技术。基本流程如下：



如此一来，便形成了通过地址栏进行导航的深度链接（deeplinking ），也就是我们所需要的路由机制。通过路由机制，一个单页应用的各个视图就可以很好的组织起来了。

4.9.1 ngRoute 内容

ng 的路由机制是靠 ngRoute 提供的，通过 hash 和 history 两种方式实现了路由，可以检测浏览器是否支持 history 来灵活调用相应的方式。ng 的路由(ngRoute)是一个单独的模块，包含以下内容：

- 服务$routeProvider 用来定义一个路由表，即地址栏与视图模板的映射
- 服务$routeParams 保存了地址栏中的参数，例如{id : 1, name : 'tom'}
- 服务$route 完成路由匹配，并且提供路由相关的属性访问及事件，如访问当前路由对应的 controller
- 指令 ngView 用来在主视图中指定加载子视图的区域

以上内容再加上$location 服务，我们就可以实现一个单页面应用了。下面来看一下具体如何使用这些内容。

4.9.2 ng 的路由机制

第一步：引入文件和依赖

ngRoute 模块包含在一个单独的文件中，所以第一步需要在页面上引入这个文件，如下：

```
<script src="http://code.angularjs.org/1.2.8/angular.min.js"></script>
<script src="http://code.angularjs.org/1.2.8/angular-route.min.js"></script>
```

光引入还不够，我们还需在模块声明中注入对 ngRoute 的依赖，如下：

```
var app = angular.module('MyApp', ['ngRoute']);
```

完成了这些，我们就可以在模板或是 controller 中使用上面的服务和指令了。下面我们需要定义一个路由表。

第二步：定义路由表

$routeProvider 提供了定义路由表的服务，它有两个核心方法，when(path,route)和otherwise(params)，先看一下核心中的核心 when(path,route)方法。

when(path,route)方法接收两个参数，path 是一个 string 类型，表示该条路由规则所匹配的路径，它将与地址栏的内容($location.path)值进行匹配。如果需要匹配参数，可以在 path 中使用冒号加名称的方式，如：path 为/show/:name，如果地址栏是/show/tom，那么参数 name 和所对应的值 tom 便会被保存在$routeParams 中，像这样：{name : tom}。我们也可以用*进行模糊匹配，如：/show*/:name 将匹配/showInfo/tom。

route 参数是一个 object，用来指定当 path 匹配后所需的一系列配置项，包括以下内容：

- controller //function 或 string 类型。在当前模板上执行的 controller 函数，生成新的 scope；
- controllerAs //string 类型，为 controller 指定别名；
- template //string 或 function 类型，视图 z 所用的模板，这部分内容将被 ngView 引用；
- templateUrl //string 或 function 类型，当视图模板为单独的 html 文件或是使用了<script type="text/ng-template">定义模板时使用；
- resolve //指定当前 controller 所依赖的其他模块；
- redirectTo //重定向的地址。

最简单情况，我们定义一个 html 文件为模板，并初始化一个指定的 controller：

```
function emailRouteConfig($routeProvider){
    $routeProvider.when('/show', {
        controller: ShowController,
        templateUrl: 'show.html'
    }).
    when('/put/:name',{
        controller: PutController,
        templateUrl: 'put.html'
    });
};
```

otherwise(params)方法对应路径匹配不到时的情况，这时候我们可以配置一个 redirectTo 参数，让它重定向到 404 页面或者是首页。

第三步：在主视图模板中指定加载子视图的位置

我们的单页面程序都是局部刷新的，那这个“局部”是哪里呢，这就轮到 ngView 出马了，只需在模板中简单的使用此指令，在哪里用，哪里就是“局部”。例如：

```
<div ng-view></div>      或：<ng-view></ng-view>
```

我们的子视图将会在此处被引入进来。完成这三步后，你的程序的路由就配置好了。

4.9.3 路由示例

下面我们将用一个例子（例 09）来说明路由的使用方式及步骤：

1.为 demoApp 添加一个路由，代码如下：

```
demoApp.config(['$routeProvider',function($routeProvider) {
        $routeProvider.when('/list', {
            templateUrl: 'route/list.html',
                controller: 'routeListController'
        }).when('/list/:id', {
                templateUrl: 'route/detail.html',
                controller: 'routeDetailController'
            }).otherwise({
                redirectTo: '/list'
            });
}]);
```

/list 对应为：route/list.html 页面，显示用户列表；/list/:id 对应于 route/detail.html 页面，显示用户详细信息。

2.为 list.html 和 detail.html 分别声明 Controller：routeListController 和 routeDetailController。

```
demoApp.controller('routeListController',function($scope) {
    $scope.users = [{userId:"zhangsan",userName:"张三",userInfo:"我是张三，我为自己带盐！
"},
        {userId:"lisi",userName:"李四",userInfo:"我是李四，我为卿狂！"},
        {userId:"woshishui",userName:"我是谁",userInfo:"我是谁！我是谁！我是谁！"}];

});
demoApp.controller('routeDetailController',function($scope, $routeParams, userService) {
    $scope.userDetail = userService.getUser($routeParams.id);
});
```

routeDetailController 中如上面提到的一样，注入了 userService 服务，在这里直接拿来用。

3.创建 list.html 和 detail.html 页面，代码如下：

```
<hr/>
<h3>Route : List.html（用户列表页面）</h3>
<ul>
    <li ng-repeat="user in users">
        <a href="#/list/{{ user.userId }}"> {{ user.userName }}</a>
    </li>
</ul>
<hr/>

<h3>Route : detail.html（用户详细信息页面）</h3>
<h3>用户名：<span style="color: red;">{{userDetail.userName}}</span></h3>
<div>
    <span>用户 ID：{{userDetail.userId}}</span><span>用户名：{{userDetail.userName}}</span>
</div>
<div>
    用户简介：<span>{{userDetail.userInfo}}</span>
</div>
<div>
    <a href="#/list">返回</a>
</div>
```

4. 路由局部刷新位置：

```
<h1>AngularJS 路由（Route） 示例</h1>
<div ng-view></div>
```

4.10 NG 动画效果

4.10.1 NG 动画效果简介

NG 动画效果，现在可以通过 CSS3 或者是 JS 来实现，如果是通过 JS 来实现的话，需要其他 JS 库（比如 JQuery）来支持，实际上底层实现还是靠其他 JS 库，只是 NG 将其封装了，

使其更易使用。

NG 动画效果包含以下几种：

- enter：元素添加到 DOM 中时执行动画；
- leave：元素从 DOM 删除时执行动画；
- move：移动元素时执行动画；
- beforeAddClass：在给元素添加 CLASS 之前执行动画；
- addClass：在给元素添加 CLASS 时执行动画；
- beforeRemoveClass：在给元素删除 CLASS 之前执行动画；
- removeClass：在给元素删除 CLASS 时执行动画。

其相关参数为：

```
var ngModule = angular.module('YourApp', ['ngAnimate']);
  demoApp.animation('.my-crazy-animation', function() {
    return {
     enter: function(element, done) {
        //run the animation here and call done when the animation is complete
        return function(cancelled) {
          //this (optional) function will be called when the animation
          //completes or when the animation is cancelled (the cancelled
          //flag will be set to true if cancelled).
        };
      },
      leave: function(element, done) { },
      move: function(element, done) { },
      //animation that can be triggered before the class is added
      beforeAddClass: function(element, className, done) { },
      //animation that can be triggered after the class is added
      addClass: function(element, className, done) { },
      //animation that can be triggered before the class is removed
      beforeRemoveClass: function(element, className, done) { },
      //animation that can be triggered after the class is removed
      removeClass: function(element, className, done) { }
    };
  });
```

4.10.2 动画效果示例

下面我们来看下 DEMO 中的例子（例 10）。

1.首先，我们在 demoApp 下定义一个动画效果，匹配 CLASS：” .border-animation”

```
/*定义动画*/
demoApp.animation('.border-animation', function(){
    return{
        beforeAddClass : function (element, className, done) {
            $(element).stop().animate({
                'border-width':1
```

```
            },2000, function() {
                done();
            });
        },
        removeClass : function (element ,className ,done ) {
            $(element).stop().animate({
                'border-width':50
            },3000, function() {
                done();
            });
        }
    };
});
```

动画效果的含义就是：在匹配 CLASS 为 border-animation 的元素添加一个 CLASS 之前使其边框的宽度在 2 秒内变为 1PX；并在其移除一个 CLASS 时使其边框的宽度在 3 秒内变为 50PX。

2. 视图中的代码如下（主要，其他相关样式请查看例子代码）：

```
<div class="border-animation" ng-show="testShow"></div>
<a href="javascript:void(0);" ng-click="testShow=!testShow" >Change</a>
```

ng-show 为 false 时会为其加上“ng-hide“的 CLASS; ng-show 为 true 时会为其移除“ng-hide“的 CLASS，从而触发动画效果。

3.其他代码：

```
demoApp.controller("test10Controller", function($scope, $animate){
    $scope.testShow = true;
});
```

5 功能演示

略（详情请看 AngularJS/demo WEB 演示）

6 AngularJS 进阶

6.1 数据绑定原理研究

Angular 用户都想知道数据绑定是怎么实现的。你可能会看到各种各样的词汇：$watch、$apply、$digest、dirty-checking...它们是什么？它们是如何工作的呢？这里我想回答这些问题，其实它们在官方的文档里都已经回答了，但是我还是想把它们结合在一起来讲，但是我只是用一种简单的方法来讲解，如果要想了解技术细节，查看源代码。

6.1.1 AngularJS 扩展事件循环

我们的浏览器一直在等待事件，比如用户交互。假如你点击一个按钮或者在输入框里输入东西，事件的回调函数就会在 javascript 解释器里执行，然后你就可以做任何 DOM 操作，等回调函数执行完毕时，浏览器就会相应地对 DOM 做出变化。(记住，这是个重要的概念)，为了解释什么是 context 以及它如何工作，我们还需要解释更多的概念。

6.1.2 $watch 队列

每次你绑定一些东西到你的 DOM 上时你就会往$watch 队列里插入一条$watch。想象一下$watch 就是那个可以检测它监视的 model 里时候有变化的东西。例如你有如下的代码：

```
/*View    index.html */
User: <input type="text" ng-model="user" />
Password: <input type="password" ng-model="pass" />
```

在这里我们有个$scope.user，他被绑定在了第一个输入框上，还有个$scope.pass，它被绑定在了第二个输入框上，然后我们在$watch list 里面加入两个$watch。

再看下面的例子：

```
/*Controller    controllers.js */
app.controller('MainCtrl', function($scope) {
      $scope.foo = "Foo";
      $scope.world = "World";
});
/*View    index.html */
Hello, {{ World }}
```

这里，即便我们在$scope 上添加了两个东西，但是只有一个绑定在了 DOM 上，因此在这里只生成了一个$watch。

再看下面的例子：

```
/*Controller    controllers.js */
app.controller('MainCtrl', function($scope) {
  $scope.people = [...];
});
/*View    index.html */
<ul>
  <li ng-repeat="person in people">
        {{person.name}} - {{person.age}}
  </li>
</ul>
```

这里又生成了多少个$watch 呢？每个 person 有两个（一个 name，一个 age），然后 ng-repeat 又有一个，因此 10 个 person 一共是(2 * 10) +1,也就是说有 21 个$watch。

因此，每一个绑定到了 DOM 上的数据都会生成一个$watch。

那这写$watch 是什么时候生成的呢？

当我们的模版加载完毕时，也就是在 linking 阶段（Angular 分为 compile 阶段和 linking 阶段），Angular 解释器会寻找每个 directive，然后生成每个需要的$watch。

6.1.3 $digest 循环

还记得我前面提到的扩展的事件循环吗？当浏览器接收到可以被 angular context 处理的事件时，$digest 循环就会触发。这个循环是由两个更小的循环组合起来的。一个处理 evalAsync 队列，另一个处理$watch 队列。 这个是处理什么的呢？$digest 将会遍历我们的 $watch，然后询问：

- 嘿，$watch，你的值是什么？
  - 是 9。
- 好的，它改变过吗？
  - 没有，先生。
- （这个变量没变过，那下一个）
- 你呢，你的值是多少？
  - 报告，是 Foo。
- 刚才改变过没？
  - 改变过，刚才是 Bar。
- （很好，我们有 DOM 需要更新了）
- 继续询问直到$watch 队列都检查过。

这就是所谓的 dirty-checking。既然所有的$watch 都检查完了，那就要问了：有没有$watch 更新过？如果有至少一个更新过，这个循环就会再次触发，直到所有的$watch 都没有变化。这样就能够保证每个 model 都已经不会再变化。记住如果循环超过 10 次的话，它将会抛出一个异常，防止无限循环。当$digest 循环结束时，DOM 相应地变化。

例如：

```
/*Controller    controllers.js */
app.controller('MainCtrl', function() {
  $scope.name = "Foo";
  $scope.changeFoo = function() {
        $scope.name = "Bar";
  }
});
/*View    index.html */
{{ name }}
<button ng-click="changeFoo()">Change the name</button>
```

这里我们有一个$watch 因为 ng-click 不生成$watch（函数是不会变的）。

我们可以看出 ng 的处理流程：

- 我们按下按钮；
- 浏览器接收到一个事件，进入 angular context；
- $digest 循环开始执行，查询每个$watch 是否变化；
- 由于监视$scope.name 的$watch 报告了变化，它会强制再执行一次$digest 循环；
- 新的$digest 循环没有检测到变化；
- 浏览器拿回控制权，更新与$scope.name 新值相应部分的 DOM。

这里很重要的是每一个进入 angular context 的事件都会执行一个$digest 循环，也就是说每次我们输入一个字母循环都会检查整个页面的所有$watch。

6.1.4 如何进入 angular context

谁决定什么事件进入 angular context，而哪些又不进入呢？通过$apply！

如果当事件触发时，你调用$apply，它会进入 angular context，如果没有调用就不会进入。现在你可能会问：刚才的例子里我也没有调用$apply 啊，为什么？Angular 已经做了！因此你点击带有 ng-click 的元素时，时间就会被封装到一个$apply 调用。如果你有一个 ng-model="foo"的输入框，然后你敲一个 f，事件就会这样调用$apply("foo = 'f';")。

Angular 什么时候不会自动为我们$apply 呢？

这是 Angular 新手共同的痛处。为什么我的 jQuery 不会更新我绑定的东西呢？因为 jQuery 没有调用$apply，事件没有进入 angular context，$digest 循环永远没有执行。

我们来看一个有趣的例子：

假设我们有下面这个 directive 和 controller。

```
/*Controller    app.js */
app.directive('clickable', function() {
return {
   restrict: "E",
   scope: {
      foo: '=',
      bar: '='
   },
   template:                    '<ul                    style="background-color:
lightblue"><li>{{foo}}</li><li>{{bar}}</li></ul>',
   link: function(scope, element, attrs) {
      element.bind('click', function() {
         scope.foo++;
         scope.bar++;
      });
   }
}
});
app.controller('MainCtrl', function($scope) {
   $scope.foo = 0;
   $scope.bar = 0;
});
```

它将 foo 和 bar 从 controller 里绑定到一个 list 里面，每次点击这个元素的时候，foo 和 bar 都会自增 1。那我们点击元素的时候会发生什么呢？我们能看到更新吗？答案是否定的。因为点击事件是一个没有封装到$apply 里面的常见的事件，这意味着我们会失去我们的计数吗？不会。

真正的结果是：$scope 确实改变了，但是没有强制$digest 循环，监视 foo 和 bar 的$watch 没有执行。也就是说如果我们自己执行一次$apply 那么这些$watch 就会看见这些变化，然后根据需要更新 DOM。

执行$apply：

```
element.bind('click', function() {
```

```
    scope.foo++;
    scope.bar++;
    scope.$apply();
});
```

$apply 是我们的$scope（或者是 direcvie 里的 link 函数中的 scope）的一个函数，调用它会强制一次$digest 循环（除非当前正在执行循环，这种情况下会抛出一个异常，这是我们不需要在那里执行$apply 的标志）。

更好的使用$apply 的方法：

```
element.bind('click', function() {
    scope.$apply(function() {
        scope.foo++;
        scope.bar++;
    });
})
```

有什么不一样的？差别就是在第一个版本中，我们是在 angular context 的外面更新的数据，如果有发生错误，Angular 永远不知道。很明显在这个像个小玩具的例子里面不会出什么大错，但是想象一下我们如果有个 alert 框显示错误给用户，然后我们有个第三方的库进行一个网络调用然后失败了，如果我们不把它封装进$apply 里面，Angular 永远不会知道失败了，alert 框就永远不会弹出来了。

因此，如果你想使用一个 jQuery 插件，并且要执行$digest 循环来更新你的 DOM 的话，要确保你调用了$apply。

有时候我想多说一句的是有些人在不得不调用$apply 时会“感觉不妙”，因为他们会觉得他们做错了什么。其实不是这样的，Angular 不是什么魔术师，他也不知道第三方库想要更新绑定的数据。

6.1.5 使用$watch 来监视

你已经知道了我们设置的任何绑定都有一个它自己的$watch，当需要时更新 DOM，但是我们如果要自定义自己的 watches 呢？简单，来看个例子：

```
/*Controller    app.js */
app.controller('MainCtrl', function($scope) {
    $scope.name = "Angular";
    $scope.updated = -1;
    $scope.$watch('name', function() {
        $scope.updated++;
    });
});
/*View    index.html*/
<body ng-controller="MainCtrl">
    <input ng-model="name" />
    Name updated: {{updated}} times.
</body>
```

这就是我们创造一个新的$watch 的方法。第一个参数是一个字符串或者函数，在这里是只是一个字符串，就是我们要监视的变量的名字，在这里，$scope.name(注意我们只需要

用 name)。第二个参数是当$watch 说我监视的表达式发生变化后要执行的。我们要知道的第一件事就是当 controller 执行到这个$watch 时，它会立即执行一次，因此我们设置 updated 为-1。

例子 2：

```
/*Controller    app.js */
app.controller('MainCtrl', function($scope) {
  $scope.name = "Angular";
  $scope.updated = 0;
  $scope.$watch('name', function(newValue, oldValue) {
    if (newValue === oldValue) { return; } // AKA first run
    $scope.updated++;
  });
});
/*View    index.html*/
<body ng-controller="MainCtrl">
  <input ng-model="name" />
  Name updated: {{updated}} times.
</body>
```

watch 的第二个参数接受两个参数，新值和旧值。我们可以用他们来略过第一次的执行。通常你不需要略过第一次执行，但在这个例子里面你是需要的。

例子 3：

```
/*Controller    app.js */
app.controller('MainCtrl', function($scope) {
  $scope.user = { name: "Fox" };
  $scope.updated = 0;
  $scope.$watch('user', function(newValue, oldValue) {
    if (newValue === oldValue) { return; }
    $scope.updated++;
  });
});
/*View    index.html*/
<body ng-controller="MainCtrl">
  <input ng-model="user.name" />
  Name updated: {{updated}} times.
</body>
```

我们想要监视$scope.user 对象里的任何变化，和以前一样这里只是用一个对象来代替前面的字符串。

呃？没用，为啥？因为$watch 默认是比较两个对象所引用的是否相同，在例子 1 和 2 里面，每次更改$scope.name 都会创建一个新的基本变量，因此$watch 会执行，因为对这个变量的引用已经改变了。在上面的例子里，我们在监视$scope.user，当我们改变$scope.user.name 时，对$scope.user 的引用是不会改变的，我们只是每次创建了一个新的$scope.user.name，但是$scope.user 永远是一样的。

例子 4：

```
/*Controller    app.js */
```

```
app.controller('MainCtrl', function($scope) {
  $scope.user = { name: "Fox" };

  $scope.updated = 0;

  $scope.$watch('user', function(newValue, oldValue) {
    if (newValue === oldValue) { return; }
    $scope.updated++;
  }, true    );
});
/*View    index.html*/
<body ng-controller="MainCtrl">
  <input ng-model="user.name" />
  Name updated: {{updated}} times.
</body>
```

现在有用了吧！因为我们对$watch 加入了第三个参数，它是一个 bool 类型的参数，表示的是我们比较的是对象的值而不是引用。由于当我们更新$scope.user.name 时$scope.user 也会改变，所以能够正确触发。

6.1.6 总结

我希望你们已经学会了在 Angular 中数据绑定是如何工作的。我猜想你的第一印象是 dirty-checking 很慢，好吧，其实是不对的。它像闪电般快。但是，如果你在一个模版里有 2000-3000 个 watch，它会开始变慢。但是我觉得如果你达到这个数量级，就可以找个用户体验专家咨询一下了。

无论如何，随着 ECMAScript6 的到来，在 Angular 未来的版本里我们将会有 Object.observe 那样会极大改善$digest 循环的速度。

6.2 自定义指令详解

angular 的指令机制。angular 通过指令的方式实现了 HTML 的扩展，增强后的 HTML 不仅长相焕然一新，同时也获得了很多强大的技能。更厉害的是，你还可以自定义指令，这就意味着 HTML 标签的范围可以扩展到无穷大。angular 赋予了你造物主的能力。既然是作为 angular 的精华之一，相应的指令相关的知识也很多的。

6.2.1 指令的编译过程

在开始自定义指令之前，我们有必要了解一下指令在框架中的执行流程：

1.浏览器得到 HTML 字符串内容，解析得到 DOM 结构。

2.ng 引入，把 DOM 结构扔给 $compile 函数处理：

① 找出 DOM 结构中有变量占位符；

② 匹配找出 DOM 中包含的所有指令引用；

③ 把指令关联到 DOM；
④ 关联到 DOM 的多个指令按权重排列；
⑤ 执行指令中的 compile 函数（改变 DOM 结构，返回 link 函数）；
⑥ 得到的所有 link 函数组成一个列表作为 $compile 函数的返回。

3. 执行 link 函数（连接模板的 scope）。

这里注意区别一下$compile 和 compile，前者是 ng 内部的编译服务，后者是指令中的编译函数，两者发挥作用的范围不同。compile 和 link 函数息息相关又有所区别，这个在后面会讲。了解执行流程对后面的理解会有帮助。

在这里有些人可能会问，angular 不就是一个 js 框架吗，怎么还能跟编译扯上呢，又不是像 C++那样的高级语言。其实此编译非彼编译，ng 编译的工作是解析指令、绑定监听器、替换模板中的变量等。因为工作方式很像高级语言编辑中的递归、堆栈过程，所以起名为编译，不要疑惑。

6.2.2 指令的使用方式及命名方法

指令的几种使用方式如下：

- 作为标签：<my-dir></my-dir>
- 作为属性：
- 作为注释：<!-- directive: my-dir exp -->
- 作为类名：

其实常用的就是作为标签和属性，下面两种用法目前还没见过，感觉就是用来卖萌的，姑且留个印象。我们自定义的指令就是要支持这样的用法。

关于自定义指令的命名，你可以随便怎么起名字都行，官方是推荐用[命名空间-指令名称]这样的方式，像 ng-controller。不过你可千万不要用 ng-前缀了，防止与系统自带的指令重名。另外一个需知道的地方，指令命名时用驼峰规则，使用时用-分割各单词。如：定义 myDirective，使用时像这样：<my-directive>。

6.2.3 自定义指令的配置参数

下面是定义一个标准指令的示例，可配置的参数包括以下部分：

```
myModule.directive('namespaceDirectiveName', function factory(injectables) {
        var directiveDefinitionObject = {
            restrict: string,//指令的使用方式，包括标签，属性，类，注释
            priority: number,//指令执行的优先级
            template: string,//指令使用的模板，用 HTML 字符串的形式表示
            templateUrl: string,//从指定的 url 地址加载模板
            replace: bool,//是否用模板替换当前元素，若为 false，则 append 在当前元素上
            transclude: bool,//是否将当前元素的内容转移到模板中
            scope: bool or object,//指定指令的作用域
            controller: function controllerConstructor($scope, $element, $attrs,
    $transclude){...},//定义与其他指令进行交互的接口函数
            require: string,//指定需要依赖的其他指令
            link: function postLink(scope, iElement, iAttrs) {...},//以编程的方式操作 DOM，包
```

```
        括添加监听器等
            compile: function compile(tElement, tAttrs, transclude){
                return: {
                    pre: function preLink(scope, iElement, iAttrs, controller){...},
                    post: function postLink(scope, iElement, iAttrs, controller){...}
                }
            }//编程的方式修改 DOM 模板的副本，可以返回链接函数
        };
        return directiveDefinitionObject;
    });
```

看上去好复杂的样子，定义一个指令需要这么多步骤嘛？当然不是，你可以根据自己的需要来选择使用哪些参数。事实上 priority 和 compile 用的比较少，template 和 templateUrl 又是互斥的，两者选其一即可。所以不必紧张，接下来分别学习一下这些参数：

- 指令的表现配置参数：restrict、template、templateUrl、replace、transclude；
- 指令的行为配置参数：compile 和 link；
- 指令划分作用域配置参数：scope；
- 指令间通信配置参数：controller 和 require。

6.2.3 指令的表现参数 restrict 等

指令的表现配置参数：restrict、template、templateUrl、replace、transclude。

我将先从一个简单的例子开始。

例子的代码如下：

```
var app = angular.module('MyApp', [], function(){console.log('here')});
app.directive('sayHello',function(){
    return {
        restrict : 'E',
        template : '<div>hello</div>'
    };
})
```

然后在页面中，我们就可以使用这个名为 sayHello 的指令了，它的作用就是输出一个 hello 单词。像这样使用：

```
<say-hello></say-hello>
```

这样页面就会显示出 hello 了，看一下生成的代码：

```
<say-hello>
    <div>hello</div>
</say-hello>
```

稍稍解释一下我们用到的两个参数，restirct 用来指定指令的使用类型，其取值及含义如下：

取值	含义	使用示例
E	标签	<my-menu title=Products></my-menu>
A	属性	
C	类	<div class="my-menu":Products>
M	注释	<!--directive:my-menu Products-->

默认值是 A。也可以使用这些值的组合，如 EA，EC 等等。我们这里指定为 E，那么它就可以像标签一样使用了。如果指定为 A，我们使用起来应该像这样：

```
<div say-hello></div>
```

从生成的代码中，你也看到了 template 的作用，它就是描述你的指令长什么样子，这部分内容将出现在页面中，即该指令所在的模板中，既然是模板中，template 的内容中也可以使用 ng-modle 等其他指令，就像在模板中使用一样。

在上面生成的代码中，我们看到了 hello 外面还包着一层<say-hello>标签，如果我们不想要这一层多余的东西了，replace 就派上用场了，在配置中将 replace 赋值为 true，将得到如下结构：

```
<div>hello</div>
```

replace 的作用正如其名，将指令标签替换为了 temple 中定义的内容。不写的话默认为 false。

上面的 template 未免也太简单了，如果你的模板 HTML 较复杂，如自定义一个 ui 组件指令，难道要拼接老长的字符串？当然不需要，此时只需用 templateUrl 便可解决问题。你可以将指令的模板单独命名为一个 html 文件，然后在指令定义中使用 templateUrl 指定好文件的路径即可，如：

```
templateUrl : ‘helloTemplate.html’
```

系统会自动发一个 http 请求来获取到对应的模板内容。是不是很方便呢，你不用纠结于拼接字符串的烦恼了。如果你是一个追求完美的有考虑性能的工程师，可能会发问：那这样的话岂不是要牺牲一个 http 请求？这也不用担心，因为 ng 的模板还可以用另外一种方式定义，那就是使用<script>标签。使用起来如下：

```
<script type="text/ng-template" id="helloTemplate.html">
    <div>hello</div>
</script>
```

你可以把这段代码写在页面头部，这样就不必去请求它了。在实际项目中，你也可以将所有的模板内容集中在一个文件中，只加载一次，然后根据 id 来取用。

接下来我们来看另一个比较有用的配置：transclude，定义是否将当前元素的内容转移到模板中。看解释有点抽象，不过亲手试试就很清楚了，看下面的代码（例 06）：

```
app.directive('sayHello',function(){
    return {
        restrict : 'E',
        template : '<div>hello，<b ng-transclude></b>！</div>',
        replace : true,
        transclude : true
    };
})
```

指定了 transclude 为 true，并且 template 修改了一下，加了一个<b>标签，并在上面使用了 ng-transclude 指令，用来告诉指令把内容转移到的位置。那我们要转移的内容是什么呢？请看使用指令时的变化：

<say-hello>美女</say-hello>

内容是什么你也看到了哈~在运行的时候，美女将会被转移到<b>标签中，原来此配置的作用就是——乾坤大挪移！看效果：

hello, 美女！

这个还是很有用的，因为你定义的指令不可能老是那么简单，只有一个空标签。当你需要对指令中的内容进行处理时，此参数便大有可用。

6.2.4 指令的行为参数：compile 和 link

6.2.3 中简单介绍了自定义一个指令的几个简单参数，restrict、template、templateUrl、replace、transclude，这几个理解起来相对容易很多，因为它们只涉及到了表现，而没有涉及行为。我们继续学习 ng 自定义指令的几个重量级参数：compile 和 link

- 理解 compile 和 link

不知大家有没有这样的感觉，自己定义指令的时候跟写 jQuery 插件有几分相似之处，都是先预先定义好页面结构及监听函数，然后在某个元素上调用一下，该元素便拥有了特殊的功能。区别在于，jQuery 的侧重点是 DOM 操作，而 ng 的指令中除了可以进行 DOM 操作外，更注重的是数据和模板的绑定。jQuery 插件在调用的时候才开始初始化，而 ng 指令在页面加载进来的时候就被编译服务($compile)初始化好了。

在指令定义对象中，有 compile 和 link 两个参数，它们是做什么的呢？从字面意义上看，编译、链接，貌似太抽象了点。其实可大有内涵，为了在自定义指令的时候能正确使用它们，现在有必要了解一下 ng 是如何编译指令的。

- 指令的解析流程详解

我们知道 ng 框架会在页面载入完毕的时候，根据 ng-app 划定的作用域来调用$compile 服务进行编译，这个$compile 就像一个大总管一样，清点作用域内的 DOM 元素，看看哪些元素上使用了指令(如)，或者哪些元素本身就是个指令(如<mydierc></mydirec>)，或者使用了插值指令({{}}也是一种指令，叫 interpolation directive)，$compile 大总管会把清点好的财产做一个清单，然后根据这些指令的优先级(priority)排列一下，真是个细心的大总管哈~大总管还会根据指令中的配置参数(template，place，transclude 等)转换 DOM，让指令“初具人形”。

然后就开始按顺序执行各指令的 compile 函数，注意此处的 compile 可不是大总管$compile，人家带着$是土豪，此处执行的 compile 函数是我们指令中配置的，compile 函数中可以访问到 DOM 节点并进行操作，其主要职责就是进行 DOM 转换，每个 compile 函数执行完后都会返回一个 link 函数，这些 link 函数会被大总管汇合一下组合成一个合体后的 link 函数，为了好理解，我们可以把它想象成葫芦小金刚，就像是进行了这样的处理。

```
//合体后的 link 函数
function AB(){
  A(); //子 link 函数
  B(); //子 link 函数
}
```

接下来进入 link 阶段，合体后的 link 函数被执行。所谓的链接，就是把 view 和 scope 链接起来。链接成啥样呢？就是我们熟悉的数据绑定，通过在 DOM 上注册监听器来动态修改 scope 中的数据，或者是使用$watchs 监听 scope 中的变量来修改 DOM，从而建立双向绑定。由此也可以断定，葫芦小金刚可以访问到 scope 和 DOM 节点。

不要忘了我们在定义指令中还配置着一个 link 参数呢，这么多 link 千万别搞混了。那这

个 link 函数是干嘛的呢，我们不是有葫芦小金刚了嘛？那我告诉你，其实它是一个小三。此话怎讲？compile 函数执行后返回 link 函数，但若没有配置 compile 函数呢？葫芦小金刚自然就不存在了。

正房不在了，当然就轮到小三出马了，大总管$compile 就把这里的 link 函数拿来执行。这就意味着，配置的 link 函数也可以访问到 scope 以及 DOM 节点。值得注意的是，compile 函数通常是不会被配置的，因为我们定义一个指令的时候，大部分情况不会通过编程的方式进行 DOM 操作，而更多的是进行监听器的注册、数据的绑定。所以，小三名正言顺的被大总管宠爱。

听完了大总管、葫芦小金刚和小三的故事，你是不是对指令的解析过程比较清晰了呢？不过细细推敲，你可能还是会觉得情节生硬，有些细节似乎还是没有透彻的明白，所以还需要再理解下面的知识点：

- compile 和 link 的区别

其实在我看完官方文档后就一直有疑问，为什么监听器、数据绑定不能放在 compile 函数中，而偏偏要放在 link 函数中？为什么有了 compile 还需要 link？就跟你质疑我编的故事一样，为什么最后小三被宠爱了？所以我们有必要探究一下，compile 和 link 之间到底有什么区别。好，正房与小三的 PK 现在开始。

首先是性能。举个例子：

```
<ul>
  <li ng-repeat="a in array">
    <input ng-modle=”a.m” />
  </li>
</ul>
```

我们的观察目标是 ng-repeat 指令。假设一个前提是不存在 link。大总管$compile 在编译这段代码时，会查找到 ng-repeat，然后执行它的 compile 函数，compile 函数根据 array 的长度复制出 n 个<li>标签。而复制出的<li>节点中还有<input>节点并且使用了 ng-modle 指令，所以 compile 还要扫描它并匹配指令，然后绑定监听器。每次循环都做如此多的工作。而更加糟糕的一点是，我们会在程序中向 array 中添加元素，此时页面上会实时更新 DOM，每次有新元素进来，compile 函数都把上面的步骤再走一遍，岂不是要累死了，这样性能必然不行。

现在扔掉那个假设，在编译的时候 compile 就只管生成 DOM 的事，碰到需要绑定监听器的地方先存着，有几个存几个，最后把它们汇总成一个 link 函数，然后一并执行。这样就轻松多了，compile 只需要执行一次，性能自然提升。

另外一个区别是能力。

尽管 compile 和 link 所做的事情差不多，但它们的能力范围还是不一样的。比如正房能管你的存款，小三就不能。小三能给你初恋的感觉，正房却不能。

我们需要看一下 compile 函数和 link 函数的定义：

```
function compile(tElement, tAttrs, transclude) { ... }
function link(scope, iElement, iAttrs, controller) { ... }
```

这些参数都是通过依赖注入而得到的，可以按需声明使用。从名字也容易看出，两个函数各自的职责是什么，compile 可以拿到 transclude，允许你自己编程管理乾坤大挪移的行为。而 link 中可以拿到 scope 和 controller，可以与 scope 进行数据绑定，与其他指令进行通信。两者虽然都可以拿到 element，但是还是有区别的，看到各自的前缀了吧？compile 拿到的是编译前的，是从 template 里拿过来的，而 link 拿到的是编译后的，已经与作用域建立了

关联，这也正是 link 中可以进行数据绑定的原因。

我暂时只能理解到这个程度了。实在不想理解这些知识的话，只要简单记住一个原则就行了：如果指令只进行 DOM 的修改，不进行数据绑定，那么配置在 compile 函数中，如果指令要进行数据绑定，那么配置在 link 函数中。

6.2.5 指令的划分作用域参数：scope

我们在上面写了一个简单的<say-hello></say-hello>，能够跟美女打招呼。但是看看人家 ng 内置的指令，都是这么用的：ng-model=”m”，ng-repeat=”a in array”，不单单是作为属性，还可以赋值给它，与作用域中的一个变量绑定好，内容就可以动态变化了。假如我们的 sayHello 可以这样用：<say-hello speak=”content”>美女</say-hello>，把要对美女说的话写在一个变量 content 中，然后只要在 controller 中修改 content 的值，页面就可以显示对美女说的不同的话。这样就灵活多了，不至于见了美女只会说一句 hello，然后就没有然后。

为了实现这样的功能，我们需要使用 scope 参数，下面来介绍一下。

使用 scope 为指令划分作用域

顾名思义，scope 肯定是跟作用域有关的一个参数，它的作用是描述指令与父作用域的关系，这个父作用域是指什么呢？想象一下我们使用指令的场景，页面结构应该是这个样子：

```
<div ng-controller="testC">
    <say-hello speak="content">美女</say-hello>
</div>
```

外层肯定会有一个 controller，而在 controller 的定义中大体是这个样子：

```
var app = angular.module('MyApp', [], function(){console.log('here')});
    app.controller('testC',function($scope){
    $scope.content = '今天天气真好！ ';
});
```

所谓 sayHello 的父作用域就是这个名叫 testC 的控制器所管辖的范围，指令与父作用域的关系可以有如下取值：

取值	说明
false	默认值。使用父作用域作为自己的作用域
true	新建一个作用域，该作用域继承父作用域
javascript 对象	与父作用域隔离，并指定可以从父作用域访问的变量

乍一看取值为 false 和 true 好像没什么区别，因为取值为 true 时会继承父作用域，即父作用域中的任何变量都可以访问到，效果跟直接使用父作用域差不多。但细细一想还是有区别的，有了自己的作用域后就可以在里面定义自己的东西，与跟父作用域混在一起是有本质上的区别。好比是父亲的钱你想花多少花多少，可你自己挣的钱父亲能花多少就不好说了。你若想看这两个作用域的区别，可以在 link 函数中打印出来看看，还记得 link 函数中可以访问到 scope 吧。

最有用的还是取值为第三种，一个对象，可以用键值来显式的指明要从父作用域中使用属性的方式。当 scope 值为一个对象时，我们便建立了一个与父层隔离的作用域，不过也不是完全隔离，我们可以手工搭一座桥梁，并放行某些参数。我们要实现对美女说各种话就得靠这个。使用起来像这样：

```
scope: {
        attributeName1: 'BINDING_STRATEGY',
```

```
        attributeName2: 'BINDING_STRATEGY',...
}
```

键为属性名称，值为绑定策略。等等！啥叫绑定策略？最讨厌冒新名词却不解释的行为！别急，听我慢慢道来。

先说属性名称吧，你是不是认为这个 attributeName1 就是父作用域中的某个变量名称？错！其实这个属性名称是指令自己的模板中要使用的一个名称，并不对应父作用域中的变量，稍后的例子中我们来说明。再来看绑定策略，它的取值按照如下的规则：

符号	说明	举例
@	传递一个字符串作为属性的值	str : '@string'
=	使用父作用域中的一个属性，绑定数据到指令的属性中	name : '=username'
&	使用父作用域中的一个函数,可以在指令中调用	getName : '&getUserName'

总之就是用符号前缀来说明如何为指令传值。你肯定迫不及待要看例子了，我们结合例子看一下，小二，上栗子~

举例说明

我想要实现上面想像的跟美女多说点话的功能，即我们给 sayHello 指令加一个属性，通过给属性赋值来动态改变说话的内容 主要代码如下：

```
app.controller('testC',function($scope){
    $scope.content = '今天天气真好！ ';
});
app.directive('sayHello',function(){
    return {
        restrict : 'E',
        template: '<div>hello,<b ng-transclude></b>,{{ cont }}</div>',
        replace : true,
        transclude : true,
        scope : {
            cont : '=speak'
         }
    };
});
```

然后在模板中，我们如下使用指令：

```
<div ng-controller="testC">
    <say-hello speak=" content ">美女</say-hello>
</div>
```

看看运行效果：

美女今天天气真好！

执行的流程是这样的：

① 指令被编译的时候会扫描到 template 中的{ {cont} }，发现是一个表达式；

② 查找 scope 中的规则：通过 speak 与父作用域绑定，方式是传递父作用域中的属性；

③ speak 与父作用域中的 content 属性绑定，找到它的值“今天天气真好！”；

④ 将 content 的值显示在模板中。

这样我们说话的内容 content 就跟父作用域绑定到了一其，如果动态修改父作用域的 content 的值，页面上的内容就会跟着改变，正如你点击“换句话”所看到的一样。

这个例子也太小儿科了吧！简单虽简单，但可以让我们理解清楚，为了检验你是不是真的明白了，可以思考一下如何修改指令定义，能让 sayHello 以如下两种方式使用：

```
<span say-hello speak="content">美女</span>
<span say-hello="content" >美女</span>
```

答案我就不说了，简单的很。下面有更重要的事情要做，我们说好了要写一个真正能用的东西来着。接下来就结合所学到的东西来写一个折叠菜单，即点击可展开，再点击一次就收缩回去的菜单。

控制器及指令的代码如下（例 07）：

```
app.controller('testC',function($scope){
    $scope.title = '个人简介';
    $scope.text = '大家好，我是一名前端工程师，我正在研究 AngularJs，欢迎大家与我交流';
});
app.directive('expander',function(){
    return {
        restrict : 'E',
        templateUrl : 'expanderTemp.html',
        replace : true,
        transclude : true,
        scope : {
            mytitle : '=etitle'
        },
        link : function(scope,element,attris){
            scope.showText = false;
            scope.toggleText = function(){
                scope.showText = ! scope.showText;
            }
        }
    };
});
```

HTML 中的代码如下：

```
<script type="text/ng-template" id="expanderTemp.html">
    <div    class="mybox">
        <div class="mytitle" ng-click="toggleText()">
            {{mytitle}}
        </div>
        <div ng-transclude ng-show="showText">
        </div>
    </div>
</script>
<div ng-controller="testC">
    <expander etitle="title">{{text}}</expander>
```

还是比较容易看懂的，我只做一点必要的解释。首先我们定义模板的时候使用了 ng 的一种定义方式<script type=”text/ng-template”id="expanderTemp.html">，在指令中就可以用 templateUrl 根据这个 id 来找到模板。指令中的{{mytitle}}表达式由 scope 参数指定从 etitle 传递，etitle 指向了父作用域中的 title。为了实现点击标题能够展开收缩内容，我们把这部分逻辑放在了 link 函数中，link 函数可以访问到指令的作用域，我们定义 showText 属性来表示内容部分的显隐，定义 toggleText 函数来进行控制，然后在模板中绑定好。如果把 showText 和 toggleText 定义在 controller 中，作为$scope 的属性呢？显然是不行的，这就是隔离作用域的意义所在，父作用域中的东西除了 title 之外通通被屏蔽。

上面的例子中，scope 参数使用了=号来指定获取属性的类型为父作用域的属性，如果我们想在指令中使用父作用域中的函数，使用&符号即可，是同样的原理。

6.2.6 指令间通信参数：controller 和 require

使用指令来定义一个 ui 组件是个不错的想法，首先使用起来方便，只需要一个标签或者属性就可以了，其次是可复用性高，通过 controller 可以动态控制 ui 组件的内容，而且拥有双向绑定的能力。当我们想做的组件稍微复杂一点，就不是一个指令可以搞定的了，就需要指令与指令的协作才可以完成，这就需要进行指令间通信。

想一下我们进行模块化开发的时候的原理，一个模块暴露（exports）对外的接口，另外一个模块引用（require）它，便可以使用它所提供的服务了。ng 的指令间协作也是这个原理，这也正是自定义指令时 controller 参数和 require 参数的作用。

controller 参数用于定义指令对外提供的接口，它的写法如下：

```
controller: function controllerConstructor($scope, $element, $attrs, $transclude)
```

它是一个构造器函数，将来可以构造出一个实例传给引用它的指令。为什么叫 controller（控制器）呢？其实就是告诉引用它的指令，你可以控制我。至于可以控制那些东西呢，就需要在函数体中进行定义了。先看 controller 可以使用的参数，作用域、节点、节点的属性、节点内容的迁移，这些都可以通过依赖注入被传进来，所以你可以根据需要只写要用的参数。关于如何对外暴露接口，我们在下面的例子来说明。

require 参数便是用来指明需要依赖的其他指令，它的值是一个字符串，就是所依赖的指令的名字，这样框架就能按照你指定的名字来从对应的指令上面寻找定义好的 controller 了。不过还稍稍有点特别的地方，为了让框架寻找的时候更轻松些，我们可以在名字前面加个小小的前缀：^，表示从父节点上寻找，使用起来像这样：require： ‘^directiveName’，如果不加，$compile 服务只会从节点本身寻找。另外还可以使用前缀：？，此前缀将告诉 $compile 服务，如果所需的 controller 没找到，不要抛出异常。

所需要了解的知识点就这些，接下来是例子时间，依旧是从书上抄来的一个例子，我们要做的是一个手风琴菜单，就是多个折叠菜单并列在一起，此例子用来展示指令间的通信再合适不过。

首先我们需要定义外层的一个结构，起名为 accordion，代码如下：

```
app.directive('accordion',function(){
        return {
            restrict : 'E',
            template : '<div ng-transclude></div>',
            replace : true,
            transclude : true,
```

```
        controller :function(){
            var expanders = [];
            this.gotOpended = function(selectedExpander){
                angular.forEach(expanders,function(e){
                    if(selectedExpander != e){
                        e.showText = false;
                    }
                });
            }
            this.addExpander = function(e){
                expanders.push(e);
            }
        }
    }
});
```

需要解释的只有 controller 中的代码，我们定义了一个折叠菜单数组 expanders，并且通过 this 关键字来对外暴露接口，提供两个方法。gotOpended 接受一个 selectExpander 参数用来修改数组中对应 expander 的 showText 属性值，从而实现对各个子菜单的显隐控制。addExpander 方法对外提供向 expanders 数组增加元素的接口，这样在子菜单的指令中，便可以调用它把自身加入到 accordion 中。

看一下我们的 expander 需要做怎样的修改呢：

```
app.directive('expander',function(){
    return {
        restrict : 'E',
        templateUrl : 'expanderTemp.html',
        replace : true,
        transclude : true,
        require : '^?accordion',
        scope : {
            title : '=etitle'
        },
        link : function(scope,element,attris,accordionController){
            scope.showText = false;
            accordionController.addExpander(scope);
            scope.toggleText = function(){
                scope.showText = ! scope.showText;
                accordionController.gotOpended(scope);
            }
        }
    };
});
```

首先使用 require 参数引入所需的 accordion 指令，添加?^前缀表示从父节点查找并且失败后不抛出异常。然后便可以在 link 函数中使用已经注入好的 accordionController 了，调用 addExpander 方法将自己的作用域作为参数传入，以供 accordionController 访问其属性。然

后在 toggleText 方法中，除了要把自己的 showText 修改以外，还要调用 accordionController 的 gotOpended 方法通知父层指令把其他菜单给收缩起来。

指令定义好后，我们就可以使用了，使用起来如下：

```
<accordion>
    <expander ng-repeat="expander in expanders" etitle="expander.title">
        {{expander.text}}
    </expander>
</accordion>
```

外层使用了 accordion 指令，内层使用 expander 指令，并且在 expander 上用 ng-repeat 循环输出子菜单。请注意这里遍历的数组 expanders 可不是 accordion 中定义的那个 expanders，如果你这么认为了，说明还是对作用域不够了解。此 expanders 是 ng-repeat 的值，它是在外层 controller 中的，所以，在 testC 中，我们需要添加如下数据：

```
$scope.expanders = [
            {title: '个人简介',
             text: '大家好，我是一名前端工程师，我正在研究 AngularJs，欢迎大家与我交流'},
            {title: '我的爱好',
             text: 'LOL '},
            {title: '性格',
             text: ' 我的性格就是无性格'}
        ];
```

6.3 性能及调优

6.3.1 性能测试

AnglarJS 作为一款优秀的 Web 框架，可大大简化前端开发的负担。

AnglarJS 很棒，但当处理包含复杂数据结构的大型列表时，其运行速度就会非常慢。这是我们将核心管理页面迁移到 AngularJS 过程中遇到的问题。这些页面在显示 500 行数据时本应该工作顺畅，但首个方法的渲染时间竟花费了 7 秒，太可怕了。后来，我们发现了在实现过程中存在两个主要性能问题。一个与“ng-repeat ”指令有关，另一个与过滤器有关。

AngularJS 中的 ng-repeat 在处理大型列表时，速度为什么会变慢？

AngularJS 中的 ng-repeat 在处理 2500 个以上的双向数据绑定时速度会变慢。这是由于 AngularJS 通过“dirty checking”函数来检测变化。每次检测都会花费时间，所以包含复杂数据结构的大型列表将降低你应用的运行速度。

提高性能的先决条件

时间记录指令

为了测量一个列表渲染所花费的时间，我们写了一个简单的程序，通过使用“ng-repeat”的属性“$last”来记录时间。时间存放在 TimeTracker 服务中，这样时间记录就与服务器端的数据加载分开了。

```
// Post repeat directive for logging the rendering time
angular.module('siApp.services').directive('postRepeatDirective',
```

```
  ['$timeout', '$log',   'TimeTracker',
  function($timeout, $log, TimeTracker) {
    return function(scope, element, attrs) {
      if (scope.$last){
        $timeout(function(){
          var timeFinishedLoadingList = TimeTracker.reviewListLoaded();
          var ref = new Date(timeFinishedLoadingList);
          var end = new Date();
          $log.debug("## DOM rendering list took: " + (end - ref) + " ms");
        });
       }
    };
  }
]);
// Use in HTML:
<tr ng-repeat="item in items" post-repeat-directive>…</tr>
```

Chrome 开发者工具的时间轴（Timeline）属性

在 Chrome 开发者工具的时间轴标签中，你可以看见事件、每秒内浏览器帧数和内存分配。“memory”工具用来检测内存泄漏，及页面所需的内存。当帧速率每秒低于 30 帧时就会出现页面闪烁问题。“frames”工具可帮助了解渲染性能，还可显示出一个 JavaScript 任务所花费的 CPU 时间。

通过限制列表的大小进行基本的调优

缓解该问题，最好的办法是限制所显示列表的大小。可通过分页、添加无限滚动条来实现。

分页，我们可以使用 AngularJS 的“limitTo”过滤器（AngularJS1.1.4 版本以后）和“startFrom”过滤器。可以通过限制显示列表的大小来减少渲染时间。这是减少渲染时间最高效的方法。

6.3.2 七大调优法则

1.渲染没有数据绑定的列表

这是最明显的解决方案，因为数据绑定是性能问题最可能的根源。如果你只想显示一次列表，并不需要更新、改变数据，放弃数据绑定是绝佳的办法。不过可惜的是，你会失去对数据的控制权，但除了该法，我们别无选择。

2.不要使用内联方法计算数据

为了在控制器中直接过滤列表，不要使用可获得过滤链接的方法。“ng-repeat”会评估每个表达式。在我们的案例中，“filteredItems()”返回过滤链接。如果评估过程很慢，它将迅速降低整个应用的速度。

- <li ng-repeat="item in filteredItems()"> //这并不是一个好方法，因为要频繁地评估。
- <li ng-repeat="item in items"> //这是要采用的方法

3.使用两个列表（一个用来进行视图显示，一个作为数据源）

将要显示的列表与总的数据列表分开，是非常有用的模型。你可以对一些过滤进行预处理，并将存于缓存中的链接应用到视图上。下面案例展示了基本实现过程。filteredLists 变量保存着缓存中的链接，applyFilter 方法来处理映射。

```
/* Controller */
```

```
// Basic list
var items = [{name:"John", active:true }, {name:"Adam"}, {name:"Chris"}, {name:"Heather"}];
// Init displayedList
$scope.displayedItems = items;
// Filter Cache
var filteredLists['active'] = $filter('filter)(items, {"active" : true});
// Apply the filter
$scope.applyFilter = function(type) {
    if (filteredLists.hasOwnProperty(type){ // Check if filter is cached
        $scope.displayedItems = filteredLists[type];
    } else {
        /* Non cached filtering */
    }
}
// Reset filter
$scope.resetFilter = function() {
    $scope.displayedItems = items;
}
/* View */
<button ng-click="applyFilter('active')">Select active</button>
<ul><li ng-repeat="item in displayedItems">{{item.name}}<li></ul>
```

4.在其他模板中使用 ng-if 来代替 ng-show

如果你用指令、模板来渲染额外的信息，例如通过点击来显示列表项的详细信息，一定要使用 ng-if（AngularJSv. 1.1.5 以后）。ng-if 可阻止渲染（与 ng-show 相比）。所以其它 DOM 和数据绑定可根据需要进行评估。

```
<li ng-repeat="item in items">
    <p> {{ item.title }} </p>
    <button ng-click="item.showDetails = !item.showDetails">Show details</buttons>
    <div ng-if="item.showDetails">
        {{item.details}}
    </div>
</li>
```

5.不要使用 ng-mouseenter、ng-mouseleave 等指令

使用内部指令，像 ng-mouseenter，AngularJS 会使你的页面闪烁。浏览器的帧速率通常低于每秒 30 帧。使用 jQuery 创建动画、鼠标悬浮效果可以解决该问题。确保将鼠标事件放入 jQuery 的.live()函数中。

6.关于过滤的小提示：通过 ng-show 隐藏多余的元素

对于长列表，使用过滤同样会减低工作效率，因为每个过滤都会创建一个原始列表的子链接。在很多情况下，数据没有变化，过滤结果也会保持不变。所以对数据列表进行预过滤，并根据情况将它应用到视图中，会大大节约处理时间。

在 ng-repeat 指令中使用过滤器，每个过滤器会返回一个原始链接的子集。AngularJS 从 DOM 中移除多余元素（通过调用 $destroy），同时也会从$scope 中移除他们。当过滤器的输入发生改变时，子集也会随着变化，元素必须进行重新链接，或着再调用$destroy。

大部分情况下，这样做很好，但一旦用户经常过滤，或者列表非常巨大，不断的链接与

销毁将影响性能。为了加快过滤的速度，你可以使用 ng-show 和 ng-hide 指令。在控制器中，进行过滤，并为每项添加一个属性。依靠该属性来触发 ng-show。结果是，只为这些元素增加 ng-hide 类，来代替将它们移除子列表、$scope 和 DOM。
触发 ng-show 的方法之一是使用表达式语法。ng-show 的值由表达式语法来确定。可以看下面的例子：

```
<input ng-model="query"></input>
<li ng-repeat="item in items" ng-show="([item.name] | filter:query).length"> {{item.name}} </li>
<span style="font-size: 14px; line-height: 24px; font-family:; white-space: normal;"></span>
```

7.关于过滤的小提示：防抖动输入

解决第 6 点提出的持续过滤问题的另一个方法是防抖动用户输入。例如，如果用户输入一个搜索关键词，只当用户停止输入后，过滤器才会被激活。使用该防抖动服务的一个很好的解决方案请见： http://jsfiddle.net/Warspawn/6K7Kd/。将它应用到你的视图及控制器中，如下所示：

```
/* Controller */
// Watch the queryInput and debounce the filtering by 350 ms.
$scope.$watch('queryInput', function(newValue, oldValue) {
    if (newValue === oldValue) { return; }
    $debounce(applyQuery, 350);
});
var applyQuery = function() {
    $scope.filter.query = $scope.query;
};
/* View */
<input ng-model="queryInput"/>
<li ng-repeat= item in items | filter:filter.query>{{ item.title }} </li>
```

7 总结

angular 上手比较难，初学者（特别是习惯了使用 JQuery 的人）可能不太适应其语法以及思想。随着对 ng 探索的一步步深入，也确实感觉到了这一点，尤其是框架内部的某些执行机制。

7.1 页面效果

ng-show ng-hide 无动画效果问题

7.2 委派事件（代理事件）

7.2.1 NG 循环及事件绑定

```
<ul>
```

```
  <li ng-repeat="a in array">
    <input ng-modle=”a.m” />
  </li>
</ul>
```

Ng 会根据 array 的长度复制出 n 个<li>标签。而复制出的<li>节点中还有<input>节点并且使用了 ng-modle 指令，所以 ng 会对所有的<input>绑定监听器（事件）。如果 array 很大，就会绑定太多的事件，性能出现问题。

7.2.2 jQuery 委派事件

从 jQuery1.7 开始，提供了.on()附加事件处理程序。

.on(events [, selector] [, data], handler(eventObject))

参数 Selector 为一个选择器字符串，用于过滤出被选中的元素中能触发事件的后代元素。如果选择器是 null 或者忽略了该选择器，那么被选中的元素总是能触发事件。

如果省略 selector 或者是 null，那么事件处理程序被称为直接事件 或者 直接绑定事件 。每次选中的元素触发事件时，就会执行处理程序，不管它直接绑定在元素上，还是从后代（内部）元素冒泡到该元素的。

当提供 selector 参数时，事件处理程序是指为委派事件（代理事件）。事件不会在直接绑定的元素上触发，但当 selector 参数选择器匹配到后代（内部元素）的时候，事件处理函数才会被触发。jQuery 会从 event target 开始向上层元素(例如，由最内层元素到最外层元素)开始冒泡，并且在传播路径上所有绑定了相同事件的元素若满足匹配的选择器，那么这些元素上的事件也会被触发。

委托事件有两个优势：他们能在后代元素添加到文档后，可以处理这些事件；代理事件的另一个好处就是，当需要监视很多元素的时候，代理事件的开销更小。

例如，在一个表格的 tbody 中含有 1,000 行，下面这个例子会为这 1,000 元素绑定事

```
$("#dataTable tbody tr").on("click", function(event){ alert($(this).text());});
```

委派事件的方法只有一个元素的事件处理程序，tbody，并且事件只会向上冒泡一层（从被点击的 tr 到 tbody ）：

```
$("#dataTable tbody").on("click", "tr", function(event){    alert($(this).text());});
```

许多委派的事件处理程序绑定到 document 树的顶层附近，可以降低性能。每次发生事件时，jQuery 需要比较从 event target（目标元素） 开始到文档顶部的路径中每一个元素上所有该类型的事件。为了获得更好的性能，在绑定代理事件时，绑定的元素最好尽可能的靠近目标元素。避免在大型文档中，过多的在 document 或 document.body 上添加代理事件。