ANGULARJS by Google 中文社区

使用超动感 HTML&JS 设计WEB应用！

www.angularjs.cn

AngularJS 入门教程

严清　李诗剑　译

TURING 图灵电子书
技术改变世界
阅读塑造人生
iturin g.com.cn

版权说明

《AngularJS入门教程》由北京图灵文化发展有限公司全球范围内出版。《AngularJS入门教程》为免费电子书，转载或引用本书内容应注明出自图灵社区。

版权所有，侵权必究。

书 名 《AngularJS入门教程》
译 严清 李诗剑
版本信息 2013年5月
定 价 0.00元

阅读、书评、交流、勘误、意见或建议，欢迎您来到图灵社区！
http://www.ituring.com.cn/
反侵权热线 （010）51095186

AngularJS入门教程目录

AngularjsTutorial_cn

AngularJS入门教程——AngularJS中文社区提供

AngularJS中文社区是一个专业的AngularJS中文开源技术社区，致力于AngularJS的技术学习、交流和研究。

我们非常欢迎更多对AngularJS感兴趣的朋友加入！

AngularJS入门教程目录

AngularJS快速开始

Hello World!

开始学习AngularJS的一个好方法是创建经典应用程序“Hello World!”：

1. 使用您喜爱的文本编辑器，创建一个 HTML 文件，例如：helloworld.html。
2. 将下面的源代码复制到您的 HTML 文件。
3. 在 web 浏览器中打开这个 HTML 文件。

源代码

```
<!doctype html>
<html ng-app>
    <head>
        <script src="http://code.angularjs.org/angular-1.0.1.min.js"></script>
    </head>
    <body>
        Hello {{'World'}}!
    </body>
</html>
```

请在您的浏览器中运行以上代码查看效果。

现在让我们仔细看看代码，看看到底怎么回事。 当加载该页时，标记`ng-app`告诉AngularJS处理整个HTML页并引导应用：

```
<html ng-app>
```

这行载入AngularJS脚本：

```
<script src="http://code.angularjs.org/angular-1.0.1.min.js"></script>
```

（想了解AngularJS处理整个HTML页的细节，请看Bootstrap。）

最后，标签中的正文是应用的模板，在UI中显示我们的问候语：

```
Hello {{'World'}}!
```

注意，使用双大括号标记`{{}}`的内容是问候语中绑定的表达式，这个表达式是一个简单的字符串‘World’。

下面，让我们看一个更有趣的例子：使用AngularJS对我们的问候语文本绑定一个动态表达式。

Hello AngularJS World!

本示例演示AngularJS的双向数据绑定（bi-directional data binding）：

1. 编辑前面创建的 helloworld.html 文档。
2. 将下面的源代码复制到您的 HTML 文件。
3. 刷新浏览器窗口。

源代码

```
<!doctype html>
<html ng-app>
    <head>
        <script src="http://code.angularjs.org/angular-1.0.1.min.js"></script>
    </head>
    <body>
        Your name: <input type="text" ng-model="yourname" placeholder="World">
        <hr>
        Hello {{yourname || 'World'}}!
    </body>
</html>
```

请在您的浏览器中运行以上代码查看效果。

该示例有一下几点重要的注意事项：

- 文本输入指令`<input ng-model="yourname" />`绑定到一个叫 `yourname` 的模型变量。
- 双大括号标记将 `yourname` 模型变量添加到问候语文本。
- 你不需要为该应用另外注册一个事件侦听器或添加事件处理程序！

现在试着在输入框中键入您的名称，您键入的名称将立即更新显示在问候语中。 这就是AngularJS双向数据绑定的概念。 输入框的任何更改会立即反映到模型变量（一个方向），模型变量的任何更改都会立即反映到问候语文本中（另一方向）。

AngularJS应用的解析

本节描述AngularJS应用程序的三个组成部分，并解释它们如何映射到模型-视图-控制器设计模式：

模板（Templates）

模板是您用HTML和CSS编写的文件，展现应用的视图。 您可给HTML添加新的元素、属性标记，作为AngularJS编译器的指令。 AngularJS编译器是完全可扩展的，这意味着通过AngularJS您可以在HTML中构建您自己的HTML标记！

应用程序逻辑（Logic）和行为（Behavior）

应用程序逻辑和行为是您用JavaScript定义的控制器。AngularJS与标准AJAX应用程序不同，您不需要另外编写侦听器或DOM控制器，因为它们已经内置到AngularJS中了。这些功能使您的应用程序逻辑很容易编写、测试、维护和理解。

模型数据（Data）

模型是从AngularJS作用域对象的属性引申的。模型中的数据可能是Javascript对象、数组或基本类型，这都不重要，重要的是，他们都属于AngularJS作用域对象。

AngularJS通过作用域来保持数据模型与视图界面UI的双向同步。一旦模型状态发生改变，AngularJS会立即刷新反映在视图界面中，反之亦然。

此外，AngularJS还提供了一些非常有用的服务特性：

1. 底层服务包括依赖注入，XHR、缓存、URL 路由和浏览器抽象服务。
2. 您还可以扩展和添加自己特定的应用服务。
3. 这些服务可以让您非常方便的编写 WEB 应用。

AngularJS入门教程：导言和准备

学习AngularJS的一个好方法是逐步完成本教程，它将引导您构建一个完整的AngularJS web应用程序。 该web应用是一个Android设备清单的目录列表，您可以筛选列表以便查看您感兴趣的设备，然后查看设备的详细信息。

本教程将向您展示AngularJS怎样使得web应用更智能更灵活，而且不需要各种扩展程序或插件。 通过本教程的学习，您将：

1. 阅读示例学习怎样使用 AngularJS 的客户端数据绑定和依赖注入功能来建立可立即响应用户操作的动态数据视图。
2. 学习如何使用 AngularJS 创建数据侦听器，且不需要进行 DOM 操作。
3. 学习一种更好、更简单的方法来测试您的 web 应用程序。
4. 学习如何使用 AngularJS 创建常见的 web 任务，例如更方便的将数据引入应用程序。

而且这一切可在任何一个浏览器实现，无需配置浏览器！

当你完成了本教程后，您将学会：

1. 创建一个可在任何浏览器中的工作的动态应用。
2. 了解 AngularJS 与其它 JavaScript 框架之间的区别。
3. 了解 AngularJS 如何实现数据绑定。
4. 利用 AngularJS 的种子项目快速创建自己的项目。
5. 创建和运行测试。
6. 学习更多 AngularJS 标识资源(API)。

本教程将指导您完成一个简单的应用程序创建过程，包括编写和运行单元测试、不断地测试应用。 教程的每个步骤为您提供建议以了解更多有关AngularJS和您创建的web应用程序。 您可能会在短时间内快速读完本教程，也可能需要花大量时间深入研究本教程。 如果想看一个简短的AngularJS介绍文档，请查看[快速开始][Getting Started]文档。

搭建学习环境

无论是Mac、Linux或Windows环境中，您均可遵循本教程学习编程。您可以使用源代码管理版本控制系统Git获取本教程项目的源代码文件，或直接从网上下载本教程项目源代码文件的镜像归档压缩包。

1. 您需要安装Node.js和Testacular来运行本项目，请到Node.js官方网站下载并安装最新版，然后把node可执行程序路径添加到系统环境变量PATH中，完成后在命令行中运行一下命令可以查看是否安装成功：

```
node -version
```

然后安装Testacular单元测试程序，请运行如下命令：

```
npm install -g testacular
```

3. 安装Git工具，然后用以下命令从Github复制本教程项目的源代码文件：

```
git clone git://github.com/angular/angular-phonecat.git
```

您也可以直接从网上下载本教程项目源代码的镜像归档压缩包。这个命令会在您当前文件夹中建立新文件夹angular-phonecat。

5. 最后一件事要做的就是确保您的计算机安装了web浏览器和文本编辑器。
6. 进入教程源代码文件包angular-phonecat，运行服务器后台程序，开始学习AngularJS！

```
cd angular-phonecat
node scripts/web-server.js
```

AngularJS入门教程00：引导程序

我们现在开始准备编写AngularJS应用——**phonecat**。这一步骤（步骤0），您将会熟悉重要的源代码文件，学习启动包含AngularJS种子项目的开发环境，并在浏览器端运行应用。

1. 进入angular-phonecat目录，运行如下命令：

```
2.  git checkout -f step-0
```

该命令将重置phonecat项目的工作目录，建议您在每一学习步骤运行此命令，将命令中的数字改成您学习步骤对应的数字，该命令将清除您在工作目录内做的任何更改。

3. 运行以下命令：

```
4.  node scripts/web-server.js
```

来启动服务器，启动后命令行终端将会提示`Http Server running at http://localhost:8000`，请不要关闭该终端，关闭该终端即关闭了服务器。在浏览器中输入http://localhost:8000/app/index.html来访问我们的**phonecat**应用。

现在，在浏览器中您应该已经看到了我们的初始应用，很简单，但说明我们的项目已经可以运行了。

应用中显示的“Nothing here yet!”是由如下HTML代码构建而成，代码中包含了AngularJS的关键元素，正是我们需要学习的。

app/index.html

```
<!doctype html>
<html lang="en" ng-app>
<head>
    <meta charset="utf-8">
    <title>My HTML File</title>
    <link rel="stylesheet" href="css/app.css">
    <link rel="stylesheet" href="css/bootstrap.css">
    <script src="lib/angular/angular.js"></script>
</head>
<body>
<p>Nothing here {{'yet' + '!'}}</p>
</body>
</html>
```

代码在做什么呢?

ng-app指令：

```
<html lang="en" ng-app>
```

ng-app指令标记了AngularJS脚本的作用域，在<html>中添加ng-app属性即说明整个<html>都是AngularJS脚本作用域。开发者也可以在局部使用ng-app指令，如<div ng-app>，则AngularJS脚本仅在该 中运行。

AngularJS脚本标签：

```
<script src="lib/angular/angular.js"></script>
```

这行代码载入angular.js脚本，当浏览器将整个HTML页面载入完毕后将会执行该angular.js脚本，angular.js脚本运行后将会寻找含有ng-app指令的HTML标签，该标签即定义了AngularJS应用的作用域。

双大括号绑定的表达式：

```
<p>Nothing here {{'yet' + '!'}}</p>
```

这行代码演示了AngularJS模板的核心功能——绑定，这个绑定由双大括号`{{}}`和表达式`'yet' + '!'`组成。

这个绑定告诉AngularJS需要运算其中的表达式并将结果插入DOM中，接下来的步骤我们将看到，DOM可以随着表达式运算结果的改变而实时更新。

AngularJS表达式Angular expression是一种类似于JavaScript的代码片段，AngularJS表达式仅在AngularJS的作用域中运行，而不是在整个DOM中运行。

引导AngularJS应用

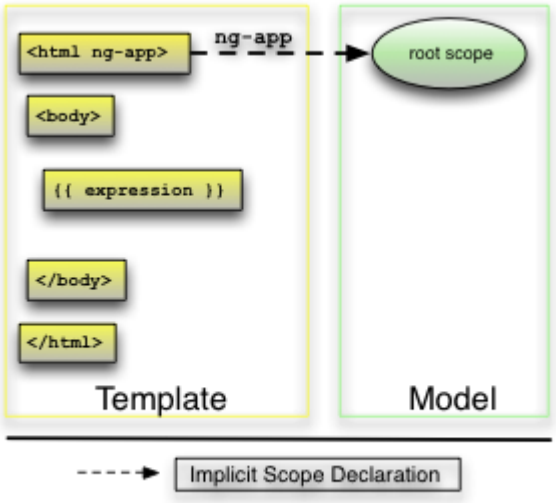
通过ngApp指令来自动引导AngularJS应用是一种简洁的方式，适合大多数情况。在高级开发中，例如使用脚本装载应用，您也可以使用bootstrap手动引导AngularJS应用。

AngularJS应用引导过程有3个重要点：

1. 注入器(injector)将用于创建此应用程序的依赖注入(dependency injection)；
2. 注入器将会创建根作用域作为我们应用模型的范围；
3. AngularJS 将会链接根作用域中的 DOM，从用 ngApp 标记的 HTML 标签开始，逐步处理 DOM 中指令和绑定。

一旦AngularJS应用引导完毕，它将继续侦听浏览器的HTML触发事件，如鼠标点击事件、按键事件、HTTP传入响应等改变DOM模型的事件。这类事件一旦发生，AngularJS将会自动检测变化，并作出相应的处理及更新。

上面这个应用的结构非常简单。该模板包仅含一个指令和一个静态绑定，其中的模型也是空的。下一步我们尝试稍复杂的应用！



我工作目录中这些文件是干什么的?

上面的应用来自于AngularJS种子项目，我们通常可以使用AngularJS种子项目来创建新项目。种子项目包括最新的AngularJS代码库、测试库、脚本和一个简单的应用程序示例，它包含了开发一个典型的web应用程序所需的基本配置。

对于本教程，我们对AngularJS种子项目进行了下列更改：

1. 删除示例应用程序；
2. 添加手机图像到 app/img/phones/；
3. 添加手机数据文件(JSON)到 app/phones/；
4. 添加 Twitter Bootstrap 文件到 app/css/ 和 app/img/。

练习

试试把关于数学运算的新表达式添加到index.html：

```
<p>1 + 2 = {{ 1 + 2 }}</p>
```

总结

现在让我们转到步骤1，将一些内容添加到web应用程序。

AngularJS入门教程01：静态模板

为了说明angularJS如何增强了标准HTML，我们先将创建一个静态HTML页面模板，然后把这个静态HTML页面模板转换成能动态显示的AngularJS模板。

在本步骤中，我们往HTML页面中添加两个手机的基本信息，用以下命令将工作目录重置到步骤1。

```
git checkout -f step-1
```

请编辑app/index.html文件，将下面的代码添加到index.html文件中，然后运行该应用查看效果。

app/index.html

```
<ul>
    <li>
        <span>Nexus S</span>
        <p>
        Fast just got faster with Nexus S.
        </p>
    </li>
    <li>
        <span>Motorola XOOM™ with Wi-Fi</span>
        <p>
        The Next, Next Generation tablet.
        </p>
    </li>
</ul>
```

练习

尝试添加多个静态HTML代码到index.html， 例如：

```
<p>Total number of phones: 2</p>
```

总结

本步骤往应用中添加了静态HTML手机列表， 现在让我们转到步骤2以了解如何使用AngularJS动态生成相同的列表。

AngularJS入门教程02：AngularJS 模板

是时候给这些网页来点动态特性了——用AngularJS！我们这里为后面要加入的控制器添加了一个测试。

一个应用的代码架构有很多种。对于AngularJS应用，我们鼓励使用模型-视图-控制器（MVC）模式解耦代码和分离关注点。考虑到这一点，我们用AngularJS来为我们的应用添加一些模型、视图和控制器。

请重置工作目录：

```
git checkout -f step-2
```

我们的应用现在有了一个包含三部手机的列表。

步骤1和步骤2之间最重要的不同在下面列出。，你可以到GitHub去看完整的差别。

视图和模板

在AngularJS中，一个**视图**是模型通过HTML**模板**渲染之后的映射。这意味着，不论模型什么时候发生变化，AngularJS会实时更新结合点，随之更新视图。

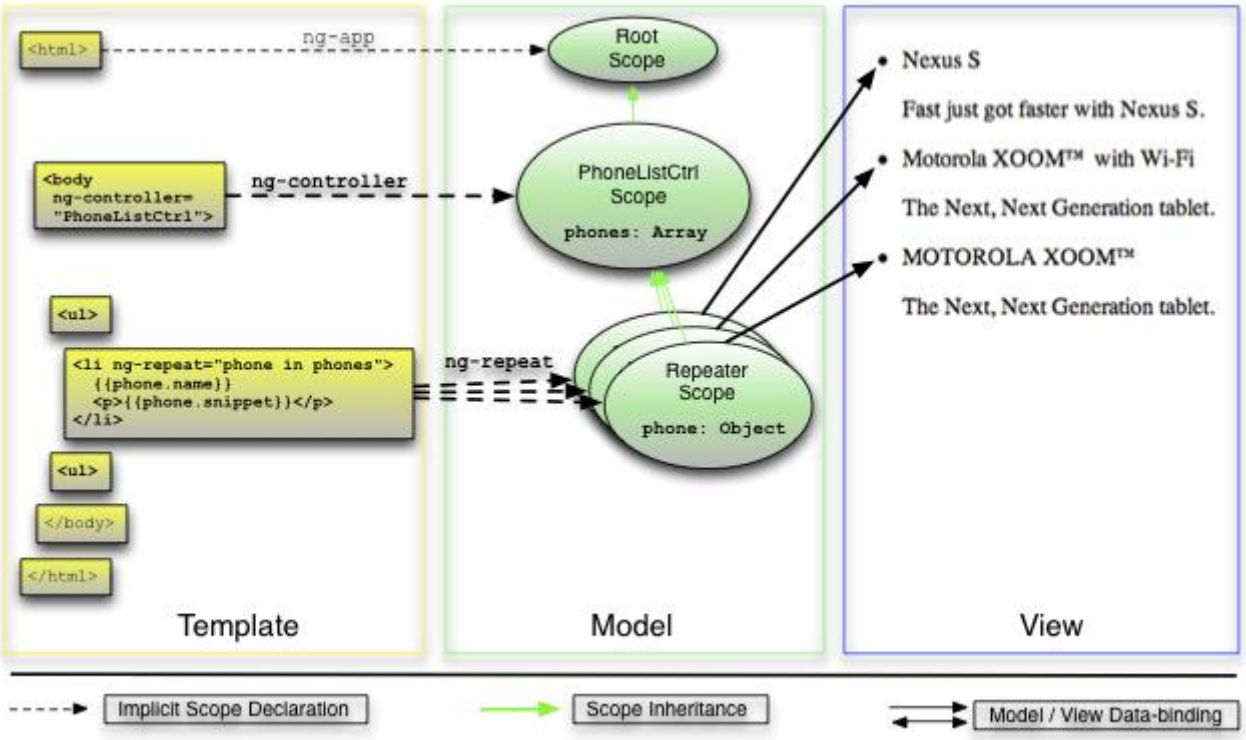
比如，视图组件被AngularJS用下面这个模板构建出来：

```
<html ng-app>
<head>
  ...
  <script src="lib/angular/angular.js"></script>
  <script src="js/controllers.js"></script>
</head>
<body ng-controller="PhoneListCtrl">

  <ul>
    <li ng-repeat="phone in phones">
      {{phone.name}}
    <p>{{phone.snippet}}</p>
    </li>
  </ul>
</body>
</html>
```

我们刚刚把静态编码的手机列表替换掉了，因为这里我们使用ngRepeat指令和两个用花括号包裹起来的AngularJS表达式——`{{phone.name}}`和`{{phone.snippet}}`——能达到同样的效果。

- 在`<li>`标签里面的 `ng-repeat="phone in phones"`语句是一个 AngularJS 迭代器。这个迭代器告诉 AngularJS 用第一个`<li>`标签作为模板为列表中的每一部手机创建一个`<li>`元素。
- 正如我们在第 0 步时学到的，包裹在 `phone.name` 和 `phone.snippet` 周围的花括号标识着数据绑定。和常量计算不同的是，这里的表达式实际上是我们应用的一个数据模型引用，这些我们在 `PhoneListCtrl` 控制器里面都设置好了。



模型和控制器

在`PhoneListCtrl`**控制器**里面初始化了**数据模型**（这里只不过是一个包含了数组的函数，数组中存储的对象是手机数据列表）：

app/js/controller.js:

```
function PhoneListCtrl($scope) {
  $scope.phones = [
    {"name": "Nexus S",
     "snippet": "Fast just got faster with Nexus S."},
    {"name": "Motorola XOOM™ with Wi-Fi",
     "snippet": "The Next, Next Generation tablet."},
    {"name": "MOTOROLA XOOM™",
     "snippet": "The Next, Next Generation tablet."}
  ];
}
```

尽管控制器看起来并没有起到什么控制的作用，但是它在这里起到了至关重要的作用。通过给定我们数据模型的语境，控制器允许我们建立模型和视图之间的数据绑定。我们是这样把表现层，数据和逻辑部件联系在一起的：

- `PhoneListCtrl`——控制器方法的名字（在 JS 文件 `controllers.js` 中）和`<body>`标签里面的 ngController 指令的值相匹配。
- 手机的数据此时与注入到我们控制器函数的**作用域**（`$scope`）相关联。当应用启动之后，会有一个根作用域被创建出来，而控制器的作用域是根作用域的一个典型后继。这个控制器的作用域对所有`<body ng-controller="PhoneListCtrl">`标记内部的数据绑定有效。

AngularJS的作用域理论非常重要：一个作用域可以视作模板、模型和控制器协同工作的粘接器。AngularJS使用作用域，同时还有模板中的信息，数据模型和控制器。这些可以帮助模型和视图分离，但是他们两者确实是同步的！任何对于模型的更改都会即时反映在视图上；任何在视图上的更改都会被立刻体现在模型中。

想要更加深入理解AngularJS的作用域，请参看AngularJS作用域文档。

测试

“AngularJS方式”让开发时代码测试变得十分简单。让我们来瞅一眼下面这个为控制器新添加的单元测试：

test/unit/controllersSpec.js:

```
describe('PhoneCat controllers', function() {

  describe('PhoneListCtrl', function(){

    it('should create "phones" model with 3 phones', function() {
      var scope = {},
      ctrl = new PhoneListCtrl(scope);

      expect(scope.phones.length).toBe(3);
    });
  });
});
```

这个测试验证了我们的手机数组里面有三条记录（暂时无需弄明白这个测试脚本）。这个例子显示出为AngularJS的代码创建一个单元测试是多么的容易。正因为测试在软件开发中是必不可少的环节，所以我们使得在AngularJS可以轻易地构建测试，来鼓励开发者多写它们。

在写测试的时候，AngularJS的开发者倾向于使用Jasmine行为驱动开发（BBD）框架中的语法。尽管AngularJS没有强迫你使用Jasmine，但是我们在教程里面所有的测试都使用Jasmine编写。你可以在Jasmine的官方主页或者Jasmine Wiki上获得相关知识。

基于AngularJS的项目被预先配置为使用JsTestDriver来运行单元测试。你可以像下面这样运行测试：

1. 在一个单独的终端上，进入到 `angular-phonecat` 目录并且运行 `./scripts/test-server.sh` 来启动测试（Windows 命令行下请输入 `.\scripts\test-server.bat` 来运行脚本，后面脚本命令运行方式类似）；
2. 打开一个新的浏览器窗口，并且转到 http://localhost:9876 ；
3. 选择“Capture this browser in strict mode”。

这个时候，你可以抛开你的窗口不管然后把这事忘了。JsTestDriver会自己把测试跑完并且把结果输出在你的终端里。

4. 运行 `./scripts/test.sh` 进行测试 。

你应当看到类似于如下的结果：

```
Chrome: Runner reset.
.
Total 1 tests (Passed: 1; Fails: 0; Errors: 0) (2.00 ms)
  Chrome 19.0.1084.36 Mac OS: Run 1 tests (Passed: 1; Fails: 0; Errors 0) (2.00 ms)
```

耶！测试通过了！或者没有... 注意：如果在你运行测试之后发生了错误，关闭浏览器然后回到终端关了脚本，然后在重新来一边上面的步骤。

练习

- 为 `index.html` 添加另一个数据绑定。例如：
- `<p>Total number of phones: {{phones.length}}</p>`
- 创建一个新的数据模型属性，并且把它绑定到模板上。例如：
- `$scope.hello = "Hello, World!"`

更新你的浏览器，确保显示出来“Hello, World!”

- 用一个迭代器创建一个简单的表：

```
<table>
    <tr><th>row number</th></tr>
    <tr ng-repeat="i in [0, 1, 2, 3, 4, 5, 6, 7]"><td>{{i}}</td></tr>
</table>
```

现在让数据模型表达式的`i`增加1：

```
<table>
    <tr><th>row number</th></tr>
    <tr ng-repeat="i in [0, 1, 2, 3, 4, 5, 6, 7]"><td>{{i+1}}</td></tr>
</table>
```

- 确定把`toBe(3)`改成`toBe(4)`之后单元测试失败，然后重新跑一遍`./scripts/test.sh`脚本

总结

你现在拥有一个模型，视图，控制器分离的动态应用了，并且你随时进行了测试。现在，你可以进入到步骤3来为应用加入全文检索功能了。

AngularJS入门教程03：迭代器过滤

我们在上一步做了很多基础性的训练，所以现在我们可以来做一些简单的事情喽。我们要加入全文检索功能（没错，这个真的非常简单！）。同时，我们也会写一个端到端测试，因为一个好的端到端测试可以帮上很大忙。它监视着你的应用，并且在发生回归的时候迅速报告。

请重置工作目录：

```
git checkout -f step-3
```

我们的应用现在有了一个搜索框。注意到页面上的手机列表随着用户在搜索框中的输入而变化。

步骤2和步骤3之间最重要的不同在下面列出。你可以在GitHub里看到完整的差别。

控制器

我们对控制器不做任何修改。

模板

app/index.html

```
<div class="container-fluid">
  <div class="row-fluid">
    <div class="span2">
      <!--Sidebar content-->

      Search: <input ng-model="query">

    </div>
    <div class="span10">
      <!--Body content-->

      <ul class="phones">
        <li ng-repeat="phone in phones | filter:query">
          {{phone.name}}
        <p>{{phone.snippet}}</p>
        </li>
      </ul>

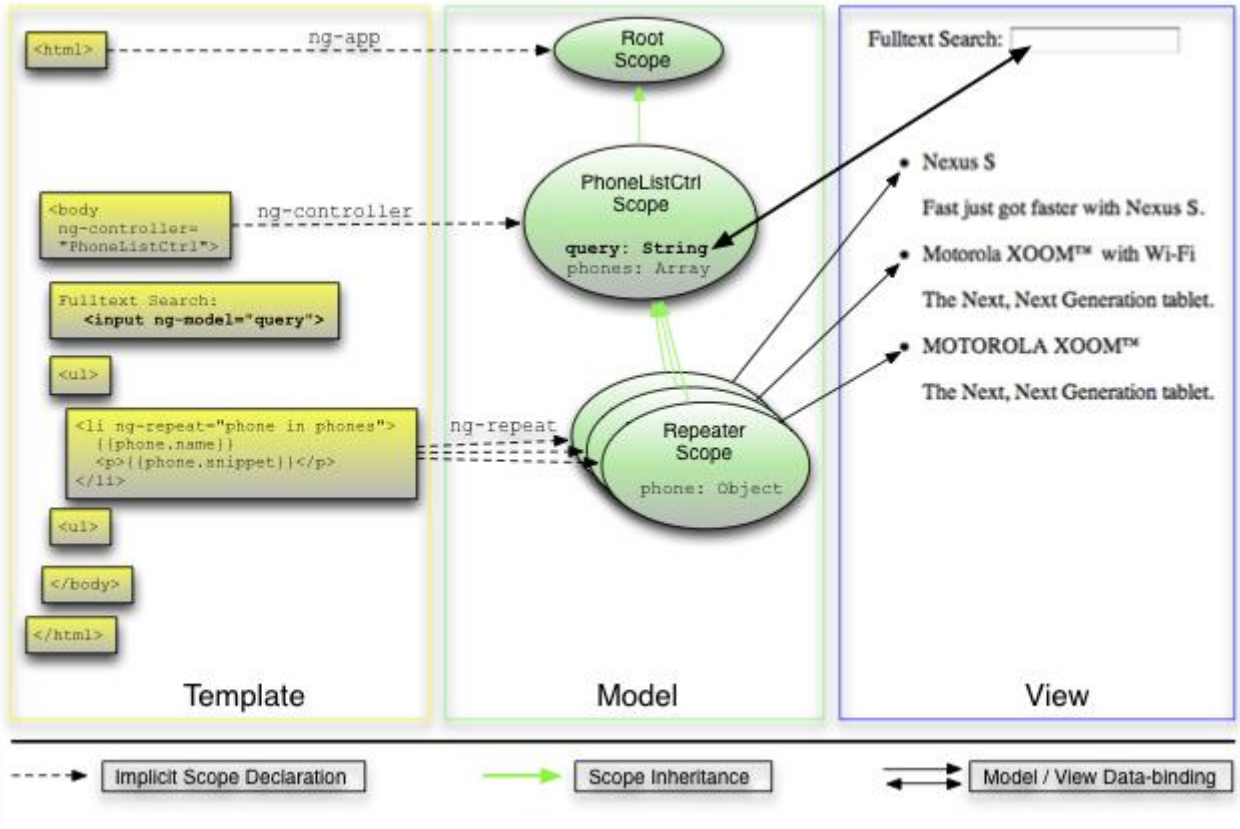
    </div>
  </div>
</div>
```

我们现在添加了一个`<input>`标签，并且使用AngularJS的$filter函数来处理ngRepeat指令的输入。

这样允许用户输入一个搜索条件，立刻就能看到对电话列表的搜索结果。我们来解释一下新的代码：

- 数据绑定： 这是AngularJS的一个核心特性。当页面加载的时候，AngularJS会根据输入框的属性值名字，将其与数据模型中相同名字的变量绑定在一起，以确保两者的同步性。

在这段代码中，用户在输入框中输入的数据名字称作`query`，会立刻作为列表迭代器（`phone in phones | filter:query`）其过滤器的输入。当数据模型引起迭代器输入变化的时候，迭代器可以高效得更新DOM将数据模型最新的状态反映出来。



- 使用`filter`过滤器：[filter](#)函数使用`query`的值来创建一个只包含匹配`query`记录的新数组。

`ngRepeat`会根据`filter`过滤器生成的手机记录数据数组来自动更新视图。整个过程对于开发者来说都是透明的。

测试

在步骤2，我们学习了编写和运行一个测试的方法。单元测试用来测试我们用js编写的控制器和其他组件都非常方便，但是不能方便的对DOM操作和应用集成进行测试。对于这些来说，端到端测试是一个更好的选择。

搜索特性是完全通过模板和数据绑定实现的，所以我们的第一个端到端测试就来验证这些特性是否符合我们的预期。

test/e2e/scenarios.js：

```
describe('PhoneCat App', function() {

  describe('Phone list view', function() {

    beforeEach(function() {
      browser().navigateTo('../../app/index.html');
    });


    it('should filter the phone list as user types into the search box', function() {
      expect(repeater('.phones li').count()).toBe(3);

      input('query').enter('nexus');
      expect(repeater('.phones li').count()).toBe(1);

      input('query').enter('motorola');
      expect(repeater('.phones li').count()).toBe(2);
    });
  });
});
```

尽管这段测试代码的语法看起来和我们之前用Jasmine写的单元测试非常像，但是端到端测试使用的是AngularJS端到端测试器提供的接口。

运行一个端到端测试，在浏览器新标签页中打开下面任意一个：

- node.js 用户：http://localhost:8000/test/e2e/runner.html
- 使用其他 http 服务器的用户：`http://localhost:[port-number]/[context-path]/test/e2e/runner.html`
- 访客：http://angular.github.com/angular-phonecat/step-3/test/e2e/runner.html

这个测试验证了搜索框和迭代器被正确地集成起来。你可以发现，在AngularJS里写一个端到端测试多么的简单。尽管这个例子仅仅是一个简单的测试，但是用它来构建任何一个复杂、可读的端到端测试都很容易。

练习

- 在`index.html`模板中添加一个`{{query}}`绑定来实时显示`query`模型的当前值，然后观察他们是如何根据输入框中的值而变化。
- 现在我们来看一下我们怎么让`query`模型的值出现在HTML的页面标题上。

你或许认为像下面这样在`title`标签上加上一个绑定就行了：

```
<title>Google Phone Gallery: {{query}}</title>
```

但是，当你重载页面的时候，你根本没办法得到期望的结果。这是因为`query`模型仅仅在`body`元素定义的作用域内才有效。

```
<body ng-controller="PhoneListCtrl">
```

如果你想让`<title>`元素绑定上`query`模型，你必须把`ngController`声明**移动**到`HTML`元素上，因为它是`title`和`body`元素的共同祖先。

```
<html ng-app ng-controller="PhoneListCtrl">
```

一定要注意把`body`元素上的`ng-controller`声明给删了。

当绑定两个花括号在`title`元素上可以实现我们的目标，但是你或许发现了，页面正加载的时候它们已经显示给用户看了。一个更好的解决方案是使用ngBind或者ngBindTemplate指令，它们在页面加载时对用户是不可见的：

```
<title ng-bind-template="Google Phone Gallery: {{query}}">Google Phone Gallery</title>
```

- 在`test/e2e/scenarios.js`的`describe`块中加入下面这些端到端测试代码：

```
it('should display the current filter value within an element with id "status"',
    function() {
        expect(element('#status').text()).toMatch(/Current filter: \s*$/);
        input('query').enter('nexus');
        expect(element('#status').text()).toMatch(/Current filter: nexus\s*$/);
        //alternative version of the last assertion that tests just the value of the binding
        using('#status').expect(binding('query')).toBe('nexus');
});
```

刷新浏览器，端到端测试器会报告测试失败。为了让测试通过，编辑`index.html`，添加一个`id`为“`status`”的`div`或者`p`元素，内容是一个`query`绑定，再加上`Current filter:`前缀。例如：

```
<div id="status">Current filter: {{query}}</div>
```

- 在端到端测试里面加一条`pause();`语句，重新跑一遍。你将发现测试器暂停了！这样允许你有机会在测试运行过程中查看你应用的状态。测试应用是实时的！你可以更换搜索内容来证明。稍有经验你就会知道，这对于在端到端测试中迅速找到问题是多么的关键。

总结

我们现在添加了全文搜索功能，并且完成一个测试证明了搜索是对的！现在让我们继续到步骤4来看看给我们的手机应用增加排序功能。

AngularJS入门教程04：双向绑定

在这一步你会增加一个让用户控制手机列表显示顺序的特性。动态排序可以这样实现，添加一个新的模型属性，把它和迭代器集成起来，然后让数据绑定完成剩下的事情。

请重置工作目录：

```
git checkout -f step-4
```

你应该发现除了搜索框之外，你的应用多了一个下来菜单，它可以允许控制电话排列的顺序。

步骤3和步骤4之间最重要的不同在下面列出。你可以在GitHub里看到完整的差别。

模板

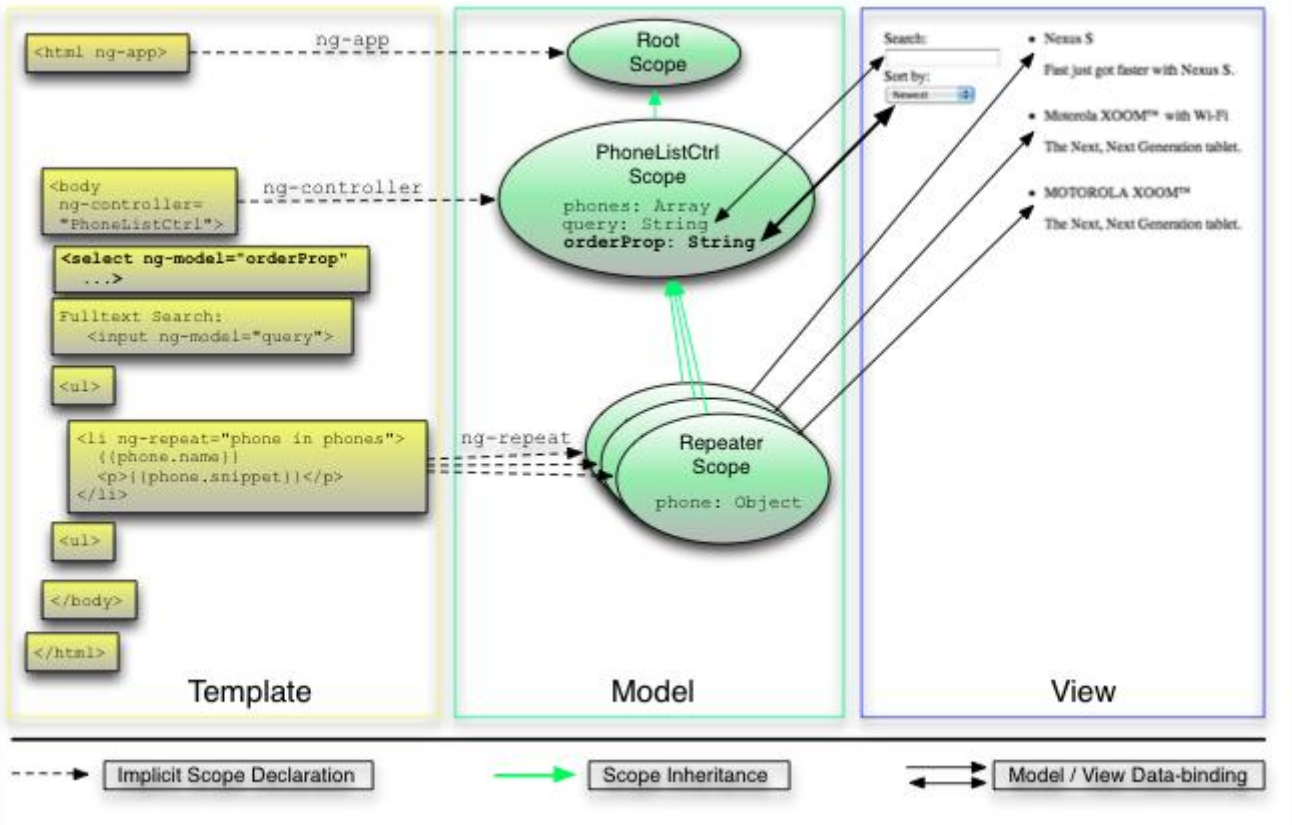
app/index.html

```
Search: <input ng-model="query">
Sort by:
<select ng-model="orderProp">
  <option value="name">Alphabetical</option>
  <option value="age">Newest</option>
</select>


<ul class="phones">
  <li ng-repeat="phone in phones | filter:query | orderBy:orderProp">
    {{phone.name}}
    <p>{{phone.snippet}}</p>
  </li>
</ul>
```

我们在index.html中做了如下更改：

- 首先，我们增加了一个叫做 `orderProp` 的`<select>`标签，这样我们的用户就可以选择我们提供的两种排序方法。



- 然后，在 `filter` 过滤器后面添加一个 orderBy 过滤器用其来处理进入迭代器的数据。`orderBy` 过滤器以一个数组作为输入，复制一份副本，然后把副本重排序再输出到迭代器。

AngularJS在`select`元素和`orderProp`模型之间创建了一个双向绑定。而后，`orderProp`会被用作`orderBy`过滤器的输入。

正如我们在步骤3中讨论数据绑定和迭代器的时候所说的一样，无论什么时候数据模型发生了改变（比如用户在下拉菜单中选了不同的顺序），AngularJS的数据绑定会让视图自动更新。没有任何笨拙的DOM操作！

控制器

app/js/controllers.js：

```
function PhoneListCtrl($scope) {
  $scope.phones = [
    {"name": "Nexus S",
     "snippet": "Fast just got faster with Nexus S.",
     "age": 0},
    {"name": "Motorola XOOM™ with Wi-Fi",
     "snippet": "The Next, Next Generation tablet.",
     "age": 1},
    {"name": "MOTOROLA XOOM™",
     "snippet": "The Next, Next Generation tablet.",
     "age": 2}
  ];

  $scope.orderProp = 'age';
}
```

- 我们修改了 `phones` 模型—— 手机的数组 ——为每一个手机记录其增加了一个 `age` 属性。我们会根据 `age` 属性来对手机进行排序。
- 我们在控制器代码里加了一行让`orderProp`的默认值为`age`。如果我们不设置默认值，这个模型会在我们的用户在下拉菜单选择一个顺序之前一直处于未初始化状态。

现在我们该好好谈谈双向数据绑定了。注意到当应用在浏览器中加载时，“Newest”在下拉菜单中被选中。这是因为我们在控制器中把orderProp设置成了‘age’。所以绑定在从我们模型到用户界面的方向上起作用——即数据从模型到视图的绑定。现在当你在下拉菜单中选择“Alphabetically”，数据模型会被同时更新，并且手机列表数组会被重新排序。这个时候数据绑定从另一个方向产生了作用——即数据从视图到模型的绑定。

测试

我们所做的更改可以通过一个单元测试或者一个端到端测试来验证正确性。我们首先来看看单元测试：

test/unit/controllersSpec.js：

```
describe('PhoneCat controllers', function() {

  describe('PhoneListCtrl', function(){
    var scope, ctrl;

    beforeEach(function() {
      scope = {},
      ctrl = new PhoneListCtrl(scope);
    });

    it('should create "phones" model with 3 phones', function() {
      expect(scope.phones.length).toBe(3);
    });

    it('should set the default value of orderProp model', function() {
      expect(scope.orderProp).toBe('age');
    });
  });
});
```

单元测试现在验证了默认值被正确设置。

我们使用Jasmine的接口把PhoneListCtrl控制器提取到一个beforeEach块中，这个块会被所有的父块describe中的所有测试所共享。

运行这些单元测试，跟以前一样，执行./scripts/test.sh脚本，你应该会看到如下输出（**注意：**要在浏览器打开http://localhost:9876并进入严格模式，测试才会运行！）：

```
Chrome: Runner reset.
..
Total 2 tests (Passed: 2; Fails: 0; Errors: 0) (3.00 ms)
  Chrome 19.0.1084.36 Mac OS: Run 2 tests (Passed: 2; Fails: 0; Errors 0) (3.00 ms)
```

现在我们把注意力转移到端到端测试上来。

test/e2e/scenarios.js：

```
...
    it('should be possible to control phone order via the drop down select box',
        function() {
      //let's narrow the dataset to make the test assertions shorter
      input('query').enter('tablet');

      expect(repeater('.phones li', 'Phone List').column('phone.name')).
          toEqual(["Motorola XOOM\u2122 with Wi-Fi",
                   "MOTOROLA XOOM\u2122"]);

      select('orderProp').option('Alphabetical');

      expect(repeater('.phones li', 'Phone List').column('phone.name')).
          toEqual(["MOTOROLA XOOM\u2122",
                   "Motorola XOOM\u2122 with Wi-Fi"]);
    });
...
```

端到端测试验证了选项框的排序机制是正确的。

你现在可以刷新你的浏览器，然后重新跑一遍端到端测试，或者你可以在AngularJS的服务器上运行一下。

练习

- 在 `PhoneListCtrl` 控制器中，把设置 `orderProp` 那条语句删掉，你会看到 AngularJS 会在下拉菜单中临时添加一个**空白**的选项，并且排序顺序是默认排序（即未排序）。
- 在 `index.html` 模板里面添加一个`{{orderProp}}绑定来实时显示它的值。

总结

现在你已经为你的应用提供了搜索功能，并且完整的进行了测试。步骤5我们将学习AngularJS的服务以及AngularJS如何使用依赖注入。

AngularJS入门教程05：XHR和依赖注入

到现在为止，我们使用是硬编码的三条手机记录数据集。现在我们使用AngularJS一个内置服务$http来获取一个更大的手机记录数据集。我们将使用AngularJS的依赖注入（dependency injection (DI)）功能来为PhoneListCtrl控制器提供这个AngularJS服务。

请重置工作目录：

```
git checkout -f step-5
```

刷新浏览器，你现在应该能看到一个20部手机的列表。

步骤4和步骤5之间最重要的不同在下面列出。你可以在GitHub里看到完整的差别。

数据

你项目当中的app/phones/phones.json文件是一个数据集，它以JSON格式存储了一张更大的手机列表。

下面是这个文件的一个样例：

```
[
 {
  "age": 13,
  "id": "motorola-defy-with-motoblur",
  "name": "Motorola DEFY\u2122 with MOTOBLUR\u2122",
  "snippet": "Are you ready for everything life throws your way?"
  ...
 },
...
]
```

控制器

我们在控制器中使用AngularJS服务$http向你的Web服务器发起一个HTTP请求，以此从app/phones/phones.json文件中获取数据。$http仅仅是AngularJS众多内建服务中之一，这些服务可以处理一些Web应用的通用操作。AngularJS能将这些服务注入到任何你需要它们的地方。

服务是通过AngularJS的依赖注入DI子系统来管理的。依赖注入服务可以使你的Web应用良好构建（比如分离表现层、数据和控制三者的部件）并且松耦合（一个部件自己不需要解决部件之间的依赖问题，它们都被DI子系统所处理）。

app/js/controllers.js

```
function PhoneListCtrl($scope, $http) {
  $http.get('phones/phones.json').success(function(data) {
    $scope.phones = data;
  });

  $scope.orderProp = 'age';
}

//PhoneListCtrl.$inject = ['$scope', '$http'];
```

$http向Web服务器发起一个HTTP GET请求，索取phone/phones.json（注意，url是相对于我们的index.html文件的）。服务器用json文件中的数据作为响应。（这个响应或许是实时从后端服务器动态产生的。但是对于浏览器来说，它们看起来都是一样的。为了简单起见，我们在教程里面简单地使用了一个json文件。）

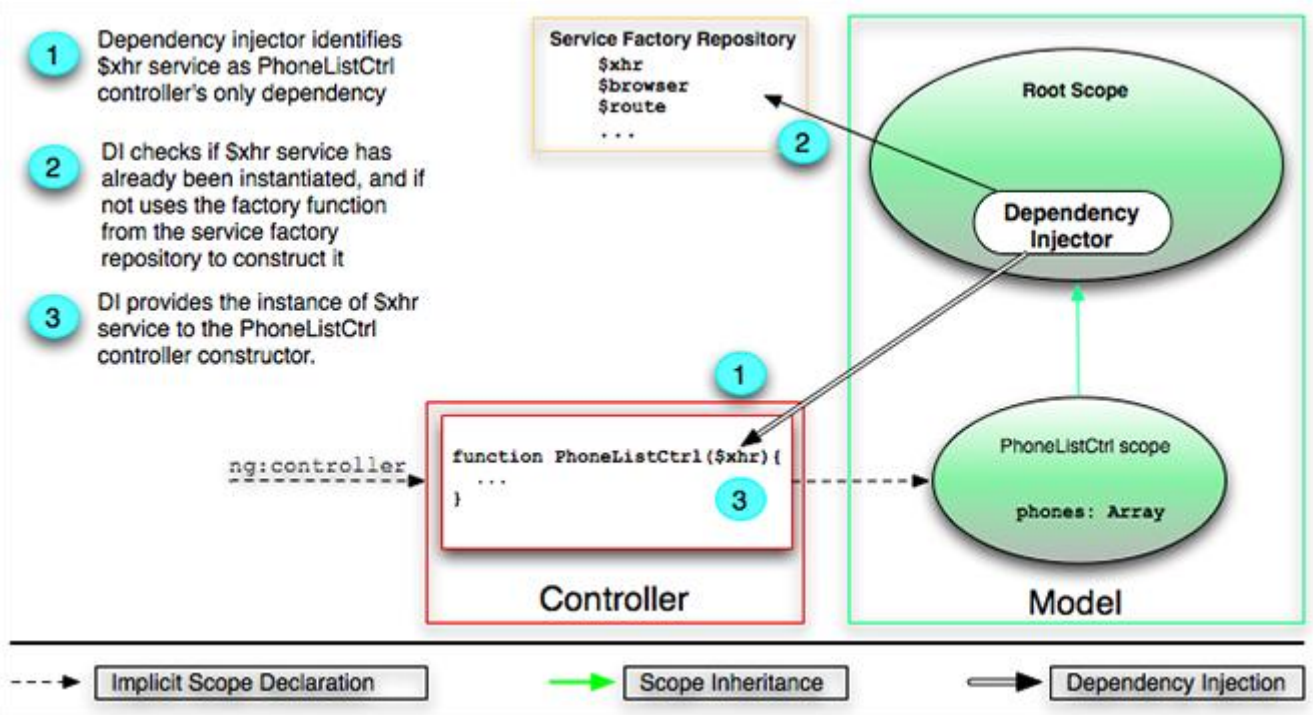
`$http`服务用`success`返回[对象应答][ng.$q]。当异步响应到达时，用这个对象应答函数来处理服务器响应的数据，并且把数据赋值给作用域的`phones`数据模型。注意到AngularJS会自动检测到这个json应答，并且已经为我们解析出来了！

为了使用AngularJS的服务，你只需要在控制器的构造函数里面作为参数声明出所需服务的名字，就像这样：

```
function PhoneListCtrl($scope, $http) {...}
```

当控制器构造的时候，AngularJS的依赖注入器会将这些服务注入到你的控制器中。当然，依赖注入器也会处理所需服务可能存在的任何传递性依赖（一个服务通常会依赖于其他的服务）。

注意到参数名字非常重要，因为注入器会用他们去寻找相应的依赖。



'$'前缀命名习惯

你可以创建自己的服务，实际上我们在步骤11就会学习到它。作为一个命名习惯，AngularJS内建服务，作用域方法，以及一些其他的AngularJS API都在名字前面使用一个‘$’前缀。不要使用‘$’前缀来命名你自己的服务和模型，否则可能会产生名字冲突。

关于 JS 压缩

由于AngularJS是通过控制器构造函数的参数名字来推断依赖服务名称的。所以如果你要压缩`PhoneListCtrl`控制器的JS代码，它所有的参数也同时会被压缩，这时候依赖注入系统就不能正确的识别出服务了。

为了克服压缩引起的问题，只要在控制器函数里面给`$inject`属性赋值一个依赖服务标识符的数组，就像被注释掉那段最后一行那样：

```
PhoneListCtrl.$inject = ['$scope', '$http'];
```

另一种方法也可以用来指定依赖列表并且避免压缩问题——使用Javascript数组方式构造控制器：把要注入的服务放到一个字符串数组（代表依赖的名字）里，数组最后一个元素是控制器的方法函数：

```
var PhoneListCtrl = ['$scope', '$http', function($scope, $http) { /* constructor body */ }];
```

上面提到的两种方法都能和AngularJS可注入的任何函数完美协作，要选哪一种方式完全取决于你们项目的编程风格，建议使用数组方式。

测试

test/unit/controllerSpec.js：

由于我们现在开始使用依赖注入，并且我们的控制器也含有了许多依赖服务，所以为我们的控制器构造测试就有一点小小的复杂了。我们需要使用`new`操作并且提供给构造器包括`$http`的一些伪实现。然而，我们推荐的方法（而且更加简单噢）是在测试环境下创建一个控制器，使用的方法和AngularJS在产品代码于下面的场景下做的一样：

```
describe('PhoneCat controllers', function() {

  describe('PhoneListCtrl', function(){
    var scope, ctrl, $httpBackend;

    beforeEach(inject(function(_$httpBackend_, $rootScope, $controller) {
      $httpBackend = _$httpBackend_;
      $httpBackend.expectGET('phones/phones.json').
          respond([{name: 'Nexus S'}, {name: 'Motorola DROID'}]);

      scope = $rootScope.$new();
      ctrl = $controller(PhoneListCtrl, {$scope: scope});
    }));
```

注意：因为我们在测试环境中加载了**Jasmine**和**angular-mock.js**，我们有了两个辅助方法，`module`和`inject`，来帮助我们获得和配置注入器。

用如下方法，我们在测试环境中创建一个控制器：

- 我们使用`inject`方法将`$rootScope`，`$controller`和`$httpBackend`服务实例注入到Jasmine的`beforeEach`函数里。这些实例都来自一个注入器，但是这个注入器在每一个测试内部都会被重新创建。这样保证了每一个测试都从一个周知的起始点开始，并且每一个测试都和其他测试相互独立。
- 调用`$rootScope.$new()`来为我们的控制器创建一个新的作用域。
- `PhoneListCtrl`函数和刚创建的作用域作为参数，传递给已注入的`$controller`函数。

由于我们现在的代码在创建`PhoneListCtrl`子作用域之前，于控制器中使用`$http`服务获取了手机列表数据，我们需要告诉测试套件等待一个从控制器来的请求。我们可以这样做：

- 将请求服务`$httpBackend`注入到我们的`beforeEach`函数中。这是这个服务的一个伪版本，这样做在产品环境中有助于处理所有的XHR和JSONP请求。服务的伪版本允许你不用考虑原生API和全局状态——随便一个都能构成测试的噩梦——就可以写测试。
- 使用`$httpBackend.expectGET`方法来告诉`$httpBackend`服务来等待一个HTTP请求，并且告诉它如何对其进行响应。注意到，当我们调用`$httpBackend.flush`方法之前，响应是不会被发出的。

现在，

```
it('should create "phones" model with 2 phones fetched from xhr', function() {
  expect(scope.phones).toBeUndefined();
  $httpBackend.flush();

  expect(scope.phones).toEqual([{name: 'Nexus S'},
                                {name: 'Motorola DROID'}]);
});
```

- 在浏览器里，我们调用`$httpBackend.flush()`来清空（flush）请求队列。这样会使得`$http`服务返回的promise（什么是promise请参见这里）能够被解释成规范的应答。

- 我们设置一些断言，来验证手机数据模型已经在作用域里了。

最终，我们验证`orderProp`的默认值被正确设置：

```
it('should set the default value of orderProp model', function() {
  expect(scope.orderProp).toBe('age');
});
;
```

执行`./scripts/test.sh`脚本来运行测试，你应该会看到如下输出：

```
Chrome: Runner reset.
..
Total 2 tests (Passed: 2; Fails: 0; Errors: 0) (3.00 ms)
  Chrome 19.0.1084.36 Mac OS: Run 2 tests (Passed: 2; Fails: 0; Errors 0) (3.00 ms)
```

练习

- 在**index.html**末尾添加一个`{{phones | json}}`绑定，观察json格式的手机列表。
- 在`PhoneListCtrl`控制器中，把HTTP应答预处理一下，使得只显示手机列表的前五个。在`$http`回调函数里面使用如下代码：
- ```
  $scope.phones = data.splice(0, 5);
  ```

总结

现在你应该感觉得到使用AngularJS的服务是多么的容易（这都要归功于AngularJS服务的依赖注入机制），转到步骤6，你会为手机添加缩略图和链接。
  ```

AngularJS入门教程06：链接与图片模板

这一步，你会为手机列表的手机添加缩略图以及一些链接，不过这些链接还不会起作用。接下来你会使用这些链接来分类显示手机的额外信息。

请重置工作目录：

```
git checkout -f step-6
```

现在你应该能够看到列表里面手机的图片和链接了。

步骤5和步骤6之间最重要的不同在下面列出。你可以在GitHub里看到完整的差别。

数据

注意到现在phones.json文件包含了唯一标识符和每一部手机的图像链接。这些url现在指向app/img/phones/目录。

app/phones/phones.json（样例片段）

```
[
 {
  ...
  "id": "motorola-defy-with-motoblur",
  "imageUrl": "img/phones/motorola-defy-with-motoblur.0.jpg",
  "name": "Motorola DEFY\u2122 with MOTOBLUR\u2122",
  ...
 },
...
]
```

模板

app/index.html

```
...
        <ul class="phones">
          <li ng-repeat="phone in phones | filter:query | orderBy:orderProp" class="thumbnail">
            <a href="#/phones/{{phone.id}}" class="thumb"><img ng-src="{{phone.imageUrl}}"></a>
            <a href="#/phones/{{phone.id}}">{{phone.name}}</a>
            <p>{{phone.snippet}}</p>
          </li>
        </ul>
...
```

这些链接将来会指向每一部电话的详细信息页。不过现在为了产生这些链接，我们在href属性里面使用我们早已熟悉的双括号数据绑定。在步骤2，我们添加了{{phone.name}}绑定作为元素内容。在这一步，我们在元素属性中使用{{phone.id}}绑定。

我们同样为每条记录添加手机图片，只需要使用ngSrc指令代替<img>的src属性标签就可以了。如果我们仅仅用一个正常src属性来进行绑定（<img class="diagram" src="{{phone.imageUrl}}">），浏览器会把AngularJS的{{ 表达式 }}标记直接进行字面解释，并且发起一个向非法urlhttp://localhost:8000/app/{{phone.imageUrl}}的请求。因为浏览器载入页面时，同时也会请求载入图片，AngularJS在页面载入完毕时才开始编译——浏览器请求载入图片时{{phone.imageUrl}}还没得到编译！有了这个ngSrc指令会避免产生这种情况，使用ngSrc指令防止浏览器产生一个指向非法地址的请求。

测试

test/e2e/scenarios.js

```
...
    it('should render phone specific links', function() {
      input('query').enter('nexus');
      element('.phones li a').click();
      expect(browser().location().url()).toBe('/phones/nexus-s');
    });
...
```

我们添加了一个新的端到端测试来验证应用为手机视图产生了正确的链接，上面就是我们的实现。

你现在可以刷新你的浏览器，并且用端到端测试器来观察测试的运行，或者你可以在AngularJS服务器上运行它们。

练习

将`ng-src`指令换成普通的`src`属性。用像Firebug，Chrome Web Inspector这样的工具，或者直接去看服务器的访问日志，你会发现你的应用向`/app/%7B%7Bphone.imageUrl%7D%7D`（或者`/app/{{phone.imageUrl}}`）发送了一个非法请求。

这个问题是由于浏览器会在遇到`img`标签的时候立刻向还未得到编译的URL地址发送一个请求，AngularJS只有在页面载入完毕后才开始编译表达式从而得到正确的图片URL地址。

总结

如今你已经添加了手机图片和链接，转到步骤7，我们将学习AngularJS的布局模板以及AngularJS是如何轻易地为应用提供多重视图。

AngularJS入门教程07：路由与多视图

在这一步，你将学习如何创建一个布局模板并且通过路由功能来构建一个具有多个视图的应用。

请重置工作目录：

```
git checkout -f step-7
```

注意到现在当你转到app/index.html时，你会被重定向到app/index.html#/phones并且相同的手机列表在浏览器中显示了出来。当你点击一个手机链接时，一个手机详细信息列表也被显示了出来。

步骤6和步骤7之间最重要的不同在下面列出。你可以在GitHub里看到完整的差别。

多视图，路由和布局模板

我们的应用正慢慢发展起来并且变得逐渐复杂。在步骤7之前，应用只给我们的用户提供了一个简单的界面（一张所有手机的列表），并且所有的模板代码位于index.html文件中。下一步是增加一个能够显示我们列表中每一部手机详细信息的页面。

为了增加详细信息视图，我们可以拓展index.html来同时包含两个视图的模板代码，但是这样会很快给我们带来巨大的麻烦。相反，我们要把index.html模板转变成**“布局模板”**。这是我们应用所有视图的通用模板。其他的“局部布局模板”随后根据当前的**“路由”**被充填入，从而形成一个完整视图展示给用户。

AngularJS中应用的路由通过$routeProvider来声明，它是$route服务的提供者。这项服务使得控制器、视图模板与当前浏览器的URL可以轻易集成。应用这个特性我们就可以实现深链接，它允许我们使用浏览器的历史(回退或者前进导航)和书签。

关于依赖注入（DI），注入器（Injector）和服务提供者（Providers）

正如从前面你学到的，依赖注入是AngularJS的核心特性，所以你必须要知道一点这家伙是怎么工作的。

当应用引导时，AngularJS会创建一个注入器，我们应用后面所有依赖注入的服务都会需要它。这个注入器自己并不知道$http和$route是干什么的，实际上除非它在模块定义的时候被配置过，否则它根本都不知道这些服务的存在。注入器唯一的职责是载入指定的服务模块，在这些模块中注册所有定义的服务提供者，并且当需要时给一个指定的函数注入依赖（服务）。这些依赖通过它们的提供者“懒惰式”（需要时才加载）实例化。

提供者是提供（创建）服务实例并且对外提供API接口的对象，它可以被用来控制一个服务的创建和运行时行为。对于$route服务来说，$routeProvider对外提供了API接口，通过API接口允许你为你的应用定义路由规则。

AngularJS模块解决了从应用中删除全局状态和提供方法来配置注入器这两个问题。和AMD或者require.js这两个模块（非AngularJS的两个库）不同的是，AngularJS模块并没有试图去解决脚本加载顺序以及懒惰式脚本加载这样的问题。这些目标和AngularJS要解决的问题毫无关联，所以这些模块完全可以共存来实现各自的目标。

App 模块

app/js/app.js

```
angular.module('phonecat', []).
  config(['$routeProvider', function($routeProvider) {
  $routeProvider.
      when('/phones', {templateUrl: 'partials/phone-list.html',   controller: PhoneListCtrl}).
```

```
      when('/phones/:phoneId', {templateUrl: 'partials/phone-detail.html', controller: PhoneDetailCtrl}).
      otherwise({redirectTo: '/phones'});
}]);
```

为了给我们的应用配置路由，我们需要给应用创建一个模块。我们管这个模块叫做`phonecat`，并且通过使用`config`API，我们请求把`$routeProvider`注入到我们的配置函数并且使用`$routeProvider.when`API来定义我们的路由规则。

注意到在注入器配置阶段，提供者也可以同时被注入，但是一旦注入器被创建并且开始创建服务实例的时候，他们就不再会被外界所获取到。

我们的路由规则定义如下

- 当 URL 映射段为`/phones`时，手机列表视图会被显示出来。为了构造这个视图，AngularJS 会使用`phone-list.html`模板和`PhoneListCtrl`控制器。
- 当 URL 映射段为`/phone/:phoneId`时，手机详细信息视图被显示出来。这里`:phoneId`是 URL 的变量部分。为了构造手机详细视图，AngularJS 会使用`phone-detail.html`模板和`PhoneDetailCtrl`控制器。

我们重用之前创造过的`PhoneListCtrl`控制器，同时我们为手机详细视图添加一个新的`PhoneDetailCtrl`控制器，把它存放在`app/js/controllers.js`文件里。

`$route.otherwise({redirectTo: '/phones'})`语句使得当浏览器地址不能匹配我们任何一个路由规则时，触发重定向到`/phones`。

注意到在第二条路由声明中`:phoneId`参数的使用。`$route`服务使用路由声明`/phones/:phoneId`作为一个匹配当前URL的模板。所有以`:`符号声明的变量（此处变量为`phones`）都会被提取，然后存放在`$routeParams`对象中。

为了让我们的应用引导我们新创建的模块，我们同时需要在ngApp指令的值上指明模块的名字：

app/index.html

```
<!doctype html>
<html lang="en" ng-app="phonecat">
...
```

控制器

app/js/controllers.js

```
...
function PhoneDetailCtrl($scope, $routeParams) {
  $scope.phoneId = $routeParams.phoneId;
}

//PhoneDetailCtrl.$inject = ['$scope', '$routeParams'];
```

模板

`$route`服务通常和ngView指令一起使用。`ngView`指令的角色是为当前路由把对应的视图模板载入到布局模板中。

app/index.html

```
<html lang="en" ng-app="phonecat">
<head>
...
  <script src="lib/angular/angular.js"></script>
  <script src="js/app.js"></script>
  <script src="js/controllers.js"></script>
</head>
<body>
```

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

</body>
</html>
```

注意，我们把index.html模板里面大部分代码移除，我们只放置了一个 容器，这个 具有ng-view属性。我们删除掉的代码现在被放置在phone-list.html模板中：

app/partials/phone-list.html

```
<div class="container-fluid">
  <div class="row-fluid">
    <div class="span2">
      <!--Sidebar content-->

      Search: <input ng-model="query">
      Sort by:
      <select ng-model="orderProp">
        <option value="name">Alphabetical</option>
        <option value="age">Newest</option>
      </select>

    </div>
    <div class="span10">
      <!--Body content-->

      <ul class="phones">
        <li ng-repeat="phone in phones | filter:query | orderBy:orderProp" class="thumbnail">
          <a href="#/phones/{{phone.id}}" class="thumb"><img ng-src="{{phone.imageUrl}}"></a>
          <a href="#/phones/{{phone.id}}">{{phone.name}}</a>
          <p>{{phone.snippet}}</p>
        </li>
      </ul>

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

同时我们为手机详细信息视图添加一个占位模板。

app/partials/phone-detail.html

```
TBD: detail view for {{phoneId}}
```

注意到我们的布局模板中没再添加PhoneListCtrl或PhoneDetailCtrl控制器属性！

测试

为了自动验证所有的东西都良好地集成起来，我们需要写一些端到端测试，导航到不同的URL上然后验证正确地视图被渲染出来。

```
...
  it('should redirect index.html to index.html#/phones', function() {
    browser().navigateTo('../../app/index.html');
    expect(browser().location().url()).toBe('/phones');
  });
...

 describe('Phone detail view', function() {

    beforeEach(function() {
      browser().navigateTo('../../app/index.html#/phones/nexus-s');
    });


    it('should display placeholder page with phoneId', function() {
      expect(binding('phoneId')).toBe('nexus-s');
    });
```

```
});
```

你现在可以刷新你的浏览器，然后重新跑一遍端到端测试，或者你可以在AngularJS的服务器上运行一下。

练习

试着在`index.html`上增加一个`{{orderProp}}`绑定，当你在手机列表视图上时什么也没变。这是因为`orderProp`模型仅仅在`PhoneListCtrl`管理的作用域下才是可见的，这与`<div ng-view>`元素相关。如果你在`phone-list.html`模板中加入同样的绑定，那么这个绑定会按你设想的那样被渲染出来。

总结

设置路由并实现手机列表视图之后，我们已经可以进入步骤8来实现手机详细信息视图了。

AngularJS入门教程08：更多模板

在这一步，你将实现手机详细信息视图，这个视图会在用户点击手机列表中的一部手机时被显示出来。

请重置工作目录：

```
git checkout -f step-8
```

现在当你点击列表中的一部手机之后，这部手机的详细信息页面就会被显示出来。

为了实现手机详细信息视图我们将会使用$http来获取数据，同时我们也要增添一个phone-detail.html视图模板。

步骤7和步骤8之间最重要的不同在下面列出。你可以在GitHub里看到完整的差别。

数据

除了phones.json，app/phones/目录也包含了每一部手机信息的json文件。

app/phones/nexus-s.json（样例片段）

```
{
  "additionalFeatures": "Contour Display, Near Field Communications (NFC),...",
  "android": {
      "os": "Android 2.3",
      "ui": "Android"
  },
  ...
  "images": [
      "img/phones/nexus-s.0.jpg",
      "img/phones/nexus-s.1.jpg",
      "img/phones/nexus-s.2.jpg",
      "img/phones/nexus-s.3.jpg"
  ],
  "storage": {
      "flash": "16384MB",
      "ram": "512MB"
  }
}
```

这些文件中的每一个都用相同的数据结构描述了一部手机的不同属性。我们会在手机详细信息视图中显示这些数据。

控制器

我们使用$http服务获取数据，以此来拓展我们的PhoneListCtrl。这和之前的手机列表控制器的工作方式是一样的。

app/js/controllers.js

```
function PhoneDetailCtrl($scope, $routeParams, $http) {
  $http.get('phones/' + $routeParams.phoneId + '.json').success(function(data) {
    $scope.phone = data;
  });
}

//PhoneDetailCtrl.$inject = ['$scope', '$routeParams', '$http'];
```

为了构造HTTP请求的URL，我们需要$route服务提供的当前路由中抽取$routeParams.phoneId。

模板

phone-detail.html文件中原有的TBD占位行已经被列表和构成手机详细信息的绑定替换掉了。注意到，这里我们使用AngularJS的{{表达式}}标记和ngRepeat来在视图中渲染数据模型。

app/partials/phone-detail.html

```
<img ng-src="{{phone.images[0]}}" class="phone">

<h1>{{phone.name}}</h1>

<p>{{phone.description}}</p>

<ul class="phone-thumbs">
  <li ng-repeat="img in phone.images">
    <img ng-src="{{img}}">
  </li>
</ul>

<ul class="specs">
  <li>
    <span>Availability and Networks</span>
    <dl>
      <dt>Availability</dt>
      <dd ng-repeat="availability in phone.availability">{{availability}}</dd>
    </dl>
  </li>
    ...
  </li>
    <span>Additional Features</span>
    <dd>{{phone.additionalFeatures}}</dd>
  </li>
</ul>
```

测试

我们来写一个新的单元测试，这个测试和我们在步骤5中为PhoneListCtrl写的那个很像。

test/unit/controllersSpec.js

```
...
  describe('PhoneDetailCtrl', function(){
    var scope, $httpBackend, ctrl;

    beforeEach(inject(function(_$httpBackend_, $rootScope, $routeParams, $controller) {
      $httpBackend = _$httpBackend_;
      $httpBackend.expectGET('phones/xyz.json').respond({name:'phone xyz'});

      $routeParams.phoneId = 'xyz';
      scope = $rootScope.$new();
      ctrl = $controller(PhoneDetailCtrl, {$scope: scope});
    }));


    it('should fetch phone detail', function() {
      expect(scope.phone).toBeUndefined();
      $httpBackend.flush();

      expect(scope.phone).toEqual({name:'phone xyz'});
    });
  });
...
```

执行./scripts/test.sh脚本来执行测试，你应该会看到如下输出：

```
Chrome: Runner reset.
...
Total 3 tests (Passed: 3; Fails: 0; Errors: 0) (5.00 ms)
  Chrome 19.0.1084.36 Mac OS: Run 3 tests (Passed: 3; Fails: 0; Errors 0) (5.00 ms)
```

同时，我们也添加一个端到端测试，指向Nexus S手机详细信息页面并且验证页面的头部是“Nexus S”。

test/e2e/scenarios.js

```
...
  describe('Phone detail view', function() {

    beforeEach(function() {
      browser().navigateTo('../../app/index.html#/phones/nexus-s');
    });


    it('should display nexus-s page', function() {
      expect(binding('phone.name')).toBe('Nexus S');
    });
  });
...
```

你现在可以刷新你的浏览器，然后重新跑一遍端到端测试，或者你可以在AngularJS的服务器上运行一下。

练习

使用AngularJS端到端测试API写一个测试，用它来验证我们在Nexus S详细信息页面显示的四个缩略图。

总结

现在手机详细页面已经就绪了，在步骤9中我们将学习如何写一个显示过滤器。

AngularJS入门教程09：过滤器

在这一步你将学习到如何创建自己的显示过滤器。

请重置工作目录：

```
git checkout -f step-9
```

现在转到一个手机详细信息页面。在上一步，手机详细页面显示“true”或者“false”来说明某个手机是否具有特定的特性。现在我们使用一个定制的过滤器来把那些文本串图形化：√作为“true”；以及×作为“false”。来让我们看看过滤器代码长什么样子。

步骤8和步骤9之间最重要的不同在下面列出。你可以在GitHub里看到完整的差别。

定制过滤器

为了创建一个新的过滤器，先创建一个`phonecatFilters`模块，并且将定制的过滤器注册给这个模块。

app/js/filters.js

```
angular.module('phonecatFilters', []).filter('checkmark', function() {
  return function(input) {
    return input ? '\u2713' : '\u2718';
  };
});
```

我们的过滤器命名为**checkmark**。它的输入要么是`true`，要么是`false`，并且我们返回两个表示true或false的unicode字符（`\u2713`和`\u2718`）。

现在我们的过滤器准备好了，我们需要将我们的`phonecatFilters`模块作为一个依赖注册到我们的主模块`phonecat`上。

app/js/app/js

```
...
angular.module('phonecat', ['phonecatFilters']).
...
```

模板

由于我们的模板代码写在`app/js/filter.js`文件中，所以我们需要在布局模板中引入这个文件。

app/index.html

```
...
 <script src="js/controllers.js"></script>
 <script src="js/filters.js"></script>
...
```

在AngularJS模板中使用过滤器的语法是：

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

我们把过滤器应用到手机详细信息模板中：

app/partials/phone-detail.html

```
...
    <dl>
      <dt>Infrared</dt>
      <dd>{{phone.connectivity.infrared | checkmark}}</dd>
      <dt>GPS</dt>
      <dd>{{phone.connectivity.gps | checkmark}}</dd>
    </dl>
...
```

测试

过滤器和其他组件一样，应该被测试，并且这些测试实际上很容易完成。

test/unit/filtersSpec.js

```
describe('filter', function() {

  beforeEach(module('phonecatFilters'));


  describe('checkmark', function() {

    it('should convert boolean values to unicode checkmark or cross',
        inject(function(checkmarkFilter) {
      expect(checkmarkFilter(true)).toBe('\u2713');
      expect(checkmarkFilter(false)).toBe('\u2718');
    }));
  });
});
```

注意在执行任何过滤器测试之前，你需要为`phonecatFilters`模块配置我们的测试注入器。

执行`./scripts/test/sh`运行测试，你应该会看到如下的输出：

```
Chrome: Runner reset.
....
Total 4 tests (Passed: 4; Fails: 0; Errors: 0) (3.00 ms)
  Chrome 19.0.1084.36 Mac OS: Run 4 tests (Passed: 4; Fails: 0; Errors 0) (3.00 ms)
```

练习

- 现在让我们来练习一下AngularJS内置过滤器，在index.html中加入如下绑定：
  - `{{ "lower cap string" | uppercase }}`
  - `{{ {foo: "bar", baz: 23} | json }}`
  - `{{ 1304375948024 | date }}`
  - `{{ 1304375948024 | date:"MM/dd/yyyy @ h:mma" }}`
- 我们也可以用一个输入框来创建一个模型，并且将之与一个过滤后的绑定结合在一起。在index.html中加入如下代码：

```
<input ng-model="userInput"> Uppercased: {{ userInput | uppercase }}
```

总结

现在你已经知道了如何编写和测试一个定制化插件，在步骤10中我们会学习如何用AngularJS继续丰富我们的手机详细信息页面。

AngularJS入门教程10：事件处理器

在这一步，你会在手机详细信息页面让手机图片可以点击。

请重置工作目录：

```
git checkout -f step-10
```

手机详细信息视图展示了一幅当前手机的大号图片，以及几个小一点的缩略图。如果用户点击缩略图就能把那张大的替换成自己那就更好了。现在我们来看看如何用AngularJS来实现它。

步骤9和步骤10之间最重要的不同在下面列出。你可以在GitHub里看到完整的差别。

控制器

app/js/controllers.js

```
...
function PhoneDetailCtrl($scope, $routeParams, $http) {
  $http.get('phones/' + $routeParams.phoneId + '.json').success(function(data) {
    $scope.phone = data;
    $scope.mainImageUrl = data.images[0];
  });

 $scope.setImage = function(imageUrl) {
    $scope.mainImageUrl = imageUrl;
  }
}

//PhoneDetailCtrl.$inject = ['$scope', '$routeParams', '$http'];
```

在`PhoneDetailCtrl`控制器中，我们创建了`mainImageUrl`模型属性，并且把它的默认值设为第一个手机图片的URL。

模板

app/partials/phone-detail.html

```
<img ng-src="{{mainImageUrl}}" class="phone">

...

<ul class="phone-thumbs">
  <li ng-repeat="img in phone.images">
    <img ng-src="{{img}}" ng-click="setImage(img)">
  </li>
</ul>
...
```

我们把大图片的`ngSrc`指令绑定到`mainImageUrl`属性上。

同时我们注册一个ngClick处理器到缩略图上。当一个用户点击缩略图的任意一个时，这个处理器会使用`setImage`事件处理函数来把`mainImageUrl`属性设置成选定缩略图的URL。

测试

为了验证这个新特性，我们添加了两个端到端测试。一个验证主图片被默认设置成第一个手机图片。第二个测试点击几个缩略图并且验证主图片随之合理的变化。

test/e2e/scenarios.js

```
...
  describe('Phone detail view', function() {

...

    it('should display the first phone image as the main phone image', function() {
      expect(element('img.phone').attr('src')).toBe('img/phones/nexus-s.0.jpg');
    });

    it('should swap main image if a thumbnail image is clicked on', function() {
      element('.phone-thumbs li:nth-child(3) img').click();
      expect(element('img.phone').attr('src')).toBe('img/phones/nexus-s.2.jpg');

      element('.phone-thumbs li:nth-child(1) img').click();
      expect(element('img.phone').attr('src')).toBe('img/phones/nexus-s.0.jpg');
    });
  });
});
```

你现在可以刷新你的浏览器，然后重新跑一遍端到端测试，或者你可以在AngularJS的服务器上运行一下。

练习

为`PhoneDetailCtrl`添加一个新的控制器方法：

```
    $scope.hello = function(name) {
        alert('Hello ' + (name || 'world') + '!');
    }
```

并且添加：

```
    <button ng-click="hello('Elmo')">Hello</button>
```

到phone-details.html模板。

总结

现在图片浏览器已经做好了，我们已经为步骤11（最后一步啦！）做好了准备，我们会学习用一种更加优雅的方式来获取数据。

AngularJS入门教程11：REST和定制服务

在这一步中，我们会改进我们APP获取数据的方式。

请重置工作目录：

```
git checkout -f step-11
```

对我们应用所做的最后一个改进就是定义一个代表RESTful客户端的定制服务。有了这个客户端我们可以用一种更简单的方式来发送XHR请求，而不用去关心更底层的$http服务（API、HTTP方法和URL）。

步骤9和步骤10之间最重要的不同在下面列出。你可以在GitHub里看到完整的差别。

模板

定制的服务被定义在app/js/services，所以我们需要在布局模板中引入这个文件。另外，我们也要加载angularjs-resource.js这个文件，它包含了ngResource模块以及其中的$resource服务，我们一会就会用到它们：

app/index.html

```
...
  <script src="js/services.js"></script>
  <script src="lib/angular/angular-resource.js"></script>
...
```

服务

app/js/services.js

```
angular.module('phonecatServices', ['ngResource']).
    factory('Phone', function($resource){
      return $resource('phones/:phoneId.json', {}, {
        query: {method:'GET', params:{phoneId:'phones'}, isArray:true}
      });
    });
```

我们使用模块API通过一个工厂方法注册了一个定制服务。我们传入服务的名字Phone和工厂函数。工厂函数和控制器构造函数差不多，它们都通过函数参数声明依赖服务。Phone服务声明了它依赖于$resource服务。

$resource服务使得用短短的几行代码就可以创建一个RESTful客户端。我们的应用使用这个客户端来代替底层的$http服务。

app/js/app.js

```
...
angular.module('phonecat', ['phonecatFilters', 'phonecatServices']).
...
```

我们需要把phonecatServices添加到phonecat的依赖数组里。

控制器

通过重构掉底层的$http服务，把它放在一个新的服务Phone中，我们可以大大简化子控制器（PhoneListCtrl和PhoneDetailCtrl）。AngularJS的$resource相比于$http更加适合于与RESTful数据源交互。而且现在我们更容易理解控制器这些代码在干什么了。

app/js/controllers.js

```
...

function PhoneListCtrl($scope, Phone) {
  $scope.phones = Phone.query();
  $scope.orderProp = 'age';
}

//PhoneListCtrl.$inject = ['$scope', 'Phone'];



function PhoneDetailCtrl($scope, $routeParams, Phone) {
  $scope.phone = Phone.get({phoneId: $routeParams.phoneId}, function(phone) {
    $scope.mainImageUrl = phone.images[0];
  });

  $scope.setImage = function(imageUrl) {
    $scope.mainImageUrl = imageUrl;
  }
}

//PhoneDetailCtrl.$inject = ['$scope', '$routeParams', 'Phone'];
```

注意到，在`PhoneListCtrl`里我们把：

```
$http.get('phones/phones.json').success(function(data) {
  $scope.phones = data;
});
```

换成了：

```
$scope.phones = Phone.query();
```

我们通过这条简单的语句来查询所有的手机。

另一个非常需要注意的是，在上面的代码里面，当调用Phone服务的方法是我们并没有传递任何回调函数。尽管这看起来结果是同步返回的，其实根本就不是。被同步返回的是一个“future”——一个对象，当XHR相应返回的时候会填充进数据。鉴于AngularJS的数据绑定，我们可以使用future并且把它绑定到我们的模板上。然后，当数据到达时，我们的视图会自动更新。

有的时候，单单依赖future对象和数据绑定不足以满足我们的需求，所以在这些情况下，我们需要添加一个回调函数来处理服务器的响应。`PhoneDetailCtrl`控制器通过在一个回调函数中设置`mainImageUrl`就是一个解释。

测试

修改我们的单元测试来验证我们新的服务会发起HTTP请求并且按照预期地处理它们。测试同时也检查了我们的控制器是否与服务正确协作。

$resource服务通过添加更新和删除资源的方法来增强响应得到的对象。如果我们打算使用`toEqual`匹配器，我们的测试会失败，因为测试值并不会和响应完全等同。为了解决这个问题，我们需要使用一个最近定义的`toEqualData`Jasmine匹配器。当`toEqualData`匹配器比较两个对象的时候，它只考虑对象的属性而忽略掉所有的方法。

test/unit/controllersSpec.js：

```
describe('PhoneCat controllers', function() {

  beforeEach(function(){
    this.addMatchers({
      toEqualData: function(expected) {
        return angular.equals(this.actual, expected);
      }
```

```
    });
  });

 beforeEach(module('phonecatServices'));

  describe('PhoneListCtrl', function(){
    var scope, ctrl, $httpBackend;

    beforeEach(inject(function(_$httpBackend_, $rootScope, $controller) {
      $httpBackend = _$httpBackend_;
      $httpBackend.expectGET('phones/phones.json').
          respond([{name: 'Nexus S'}, {name: 'Motorola DROID'}]);

      scope = $rootScope.$new();
      ctrl = $controller(PhoneListCtrl, {$scope: scope});
    }));

    it('should create "phones" model with 2 phones fetched from xhr', function() {
      expect(scope.phones).toEqual([]);
      $httpBackend.flush();

      expect(scope.phones).toEqualData(
          [{name: 'Nexus S'}, {name: 'Motorola DROID'}]);
    });

    it('should set the default value of orderProp model', function() {
      expect(scope.orderProp).toBe('age');
    });
  });

  describe('PhoneDetailCtrl', function(){
    var scope, $httpBackend, ctrl,
        xyzPhoneData = function() {
          return {
            name: 'phone xyz',
            images: ['image/url1.png', 'image/url2.png']
          }
        };

    beforeEach(inject(function(_$httpBackend_, $rootScope, $routeParams, $controller) {
      $httpBackend = _$httpBackend_;
      $httpBackend.expectGET('phones/xyz.json').respond(xyzPhoneData());

      $routeParams.phoneId = 'xyz';
      scope = $rootScope.$new();
      ctrl = $controller(PhoneDetailCtrl, {$scope: scope});
    }));

    it('should fetch phone detail', function() {
      expect(scope.phone).toEqualData({});
      $httpBackend.flush();

      expect(scope.phone).toEqualData(xyzPhoneData());
    });
  });
});
```

执行`./scripts/test.sh`运行测试，你应该会看到如下的输出：

```
Chrome: Runner reset.
....
Total 4 tests (Passed: 4; Fails: 0; Errors: 0) (3.00 ms)
  Chrome 19.0.1084.36 Mac OS: Run 4 tests (Passed: 4; Fails: 0; Errors 0) (3.00 ms)
```

总结

完工！你在相当短的时间内已经创建了一个Web应用。在完结篇里面我们会提起接下来应该干什么。

AngularJS入门教程：完结篇

我们的应用现在完成了。你可以随意练习这些代码，用`git checkout`或者`goto_step.sh`命令切换到之前的步骤。

对于更多我们在教程部分提及的细节以及AngularJS理论的例子，你可以在开发指南中找到。

一些更多的例子，请参照Cookbook。

当你准备好使用AngularJS创建一个新项目时，我们推荐使用AngularJS种子项目来引导你的开发。

我们希望这篇教程对你有用，让你对AngularJS有了足够的了解，并且愿意对其进行更加深入的学习。我们特别期待着你能够开发出自己的AngularJS应用，或者让你对AngularJS贡献代码产生兴趣。

如果你有什么问题，反馈，或是想跟我们打个招呼，可在社区论坛交流，或者直接向https://groups.google.com/forum/#!forum/angular发消息吧！