angular-phonecat

xdsnet

Published
with GitBook

Table of Contents

PhoneCat 入门教程 App 导言

（基于AngularJS1.3.0-rc.1 由xdsnet at gmail 翻译）

本教程是一个介绍利用AngularJS开发的极好入门材料，它给出了一个完整的AngularJS web app开发的过程。这个app会构建一个Android设备的目录显示服务，可以通过过滤来定位显示个别关注的设备以显示更多细节信息。

下面这个教程会展示Angular是如何在没有扩展程序或插件的支持下使得浏览器应用更智能，你将：

- 通过例子了解如何使用客户端数据绑定技术实现能快速响应应用户活动操作的动态视图应用。
- 了解Angular是如何在没有专门的DOM操作中实现视图与数据同步的。
- 学习如何利用Karma和Protractor技术来更好、更方便的测试应用。
- 学习如何使用依赖注入和服务来使常见的web任务更容易：例如在应用中获得数据。

当你完成这个入门教程，你将能够：

- 创建一个能在所有现代浏览器使用的动态应用程序
- 通过数据绑定连接你的数据模型与视图
- 利用Karma技术创建和实施单元测试
- 利用Protractor技术创建和实施 端到端（e2e）的测试
- 把应用逻辑从template(模板)中移出到Controllers(控制器)中
- 利用Angular服务从服务器获取数据
- 利用ngAnimate技术在应用中应用动画
- 知道学习更多的AngularJS需要哪方面资源

这个教程通过完整的开发一个简单应用的过程指导你学习AngularJS，这包括如何写并且运行单元测试和端到端测试。每一步还额外提供了建议，供您了解更多的AngularJS和您正在构建的应用程序。

你可以在几个小时内完成整个教程，当然你也可以深度学习来度过愉快的一天。如果你正在寻找一个更短

的AngularJS介绍，请看入门文档。

开始

这部门内容展示如何在你本地计算机上设置来进行开发AngularJS相关程序。如果你仅仅想阅读入门教程，你可以直接跳到步骤0——准备

使用代码

在你的计算机上，你完全可以一直跟随教程使用代码，也可以进行随意改动，实际上改动代码可以让你更深入体验如何撰写AngularJS程序以及利用推荐的测试工具工作。

这个教程依赖于Git版本控制工具，但使用本教程你并不需要知道太多Git使用知识,只需要会用教程中用到的很少几个命令（到时也会给出了完整命令）。

安装Git

你可以从 http://git-scm.com/download下载安装Git。安装后你应该可以访问到git命令行工具，在本教程中你仅需要用到两个git命令:

- `git clone ...` 从远处版本仓库克隆到本地计算机
- `git checkout ...` 在本地计算机上查看（取出）特定版本（标签）的代码

下载angular-phonecat

从 GitHuB上的angular-phonecat repository克隆需要用到下面的命令

```
git clone --depth=14 https://github.com/angular/angular-phonecat.git
```

这个命令会在你计算机的当前目录下建立 `angular-phonecat` 目录：

> `--depth=14` 参数选项是告诉Git获取最近的14次提交，这让下载的东西更少，也就更快了。

改变当前目录到 `angular-phonecat`

```
cd angular-phonecat
```

本教程说明：从现在开始，都假定你的所有命令都是在 `angular-phonecat` 目录下执行。

安装Node.js

如果你想运行预定义的本地web服务来测试，你需要有 Node.js v0.10.27+。你可以从http://nodejs.org/download/下载安装你操作系统合适的版本。确认你的Node.js是符合要求的版本，需要运行：

```
node --version
```

在基于Debian的发行版，因为有另外一个工具命名为 `node`，所以提供的解决方案是安装 `nodejs-legacy` apt包，但把 `node` 改名为 `nodejs`：

```
apt-get install nodejs-legacy
nodejs --version
```

> 如果你需要在你的本地环境中运行不同版本的node.js，可以考虑安装 Node Version Manager (nvm)。

一旦在你的计算机中安了Node.js，你就可以考虑安装下面的工具来自动解决依赖问题：

```
npm install
```

这个命令会下载下面的工具到 `node_modules` 目录：

- Bower -客户端代码包管理
- Http-Server - 简单的本地静态web服务
- Karma -单元测试工具
- Protractor - 端到端(E2E)测试工具

运行 `npm install` 会自动下载Angular框架到 `app/bower_components` 目录

> 注意angular-phonecat项目通过npm脚本自动安装运行各个单元，这意味着你不需要单独为了这个教程在系统（全局）中安装相应内容。更多信息请在下面的安装辅助工具中了解。 项目中预定义了几个npm辅助脚本来使得你更容易运行常规任务，这些在整个开发中都是有效的：

`npm start`*:* 打开本地开发*web*服务 `npm test`: 运行Karma单元测试 `npm run protractor`*:* 运行*Protractor*端到端*(E2E)*测试 `npm run update-webdriver`: 安装Protractor需要的驱动程序

安装辅助工具（可选）

Bower, Http-Server, Karma 和 Protractor 模块都是单独可执行的，你可以全局安装，然后在一个终端/命令行中运行。在这个教程中你不一定需要下面的工作，如果你决定立即执行，你可以在全局安装:

```
sudo npm install -g ...
```

例如要安装Bower的命令是:

```
sudo npm install -g bower
```

*(*注意在*windows*上运行时需要省略那个*sudo)* 然后你就可以运行bower工具了，例如：

```
bower install
```

运行开发**Web**服务

整个Angular应用是纯粹的客户端代码，所以它可以用浏览器在本地文件系统打开，这也可能比通过HTTP服务打开更好。但是，为了安全现在很多浏览器是不允许JavaScript直接从本地文件系统加载文件的。所以angular-phonecat项目定义了一个简单的静态web服务来支持开发。打开运行这个服务需要输入:

```
npm start
```

这会建立一个本地web服务，它监听者端口8000.你可以用浏览器打开

```
http://localhost:8000/app/index.html
```

来访问

> 这个web服务也可以绑定到不同的ip或者端口，你需要编辑package.json中的"start"脚本部分，你可以用-a 来设置ip地址，用-p来设置端口。

运行单元测试

单元测试可以确认JavaScript脚本会执行正确的操作，单元测试关注程序中局部的功能实现。所有的单元测试都在`test/unit`目录下。

本项目使用Karma来定义单元测试，执行这些测试需要运行

```
npm test
```

这将运行Karma单元测试。 Karma读取`test/karma.conf.js`中的配置,它指示Karma执行: 打开一个*Chrome*浏览器，并且连接到*Karma* 在浏览器中执行所有的单元测试 在终端/命令行中报告测试结果 监控项目JavaScript文件的改变并再次执行测试

确保所有测试项都是正确的才算完成相应的编码工作，因为单元测试可以检测你的改动是否符合你编码需要，并及时反馈。

运行端到端（E2E）测试

我们使用端到端（End to End——E2E） 的测试确保应用的整体操作。端到端测试被设计来测试完整客户端应用，包括显示的视图和操作交互对应的行为是否正确。它模拟真实的用户在真实浏览器中访问应用的场景。

端到端测试放置在`test/e2e`目录下。

angular-phonecat项目配置使用Protractor来执行端到端测试。Protractor依赖一套驱动使它能与浏览器交互。你可以通过下面的命令进行安装:

```
npm run update-webdriver
```

(你只需要执行它一次.)

因为Protractor工作需要与程序交互，所以需要先启动我们的web服务：:

```
npm start
```

然后在一个特定的终端/命令行窗口中执行Protractor测试脚本：:

```
npm run protractor
```

Protractor会读取在`test/protractor-conf.js`中的配置，这些配置告诉Protractor执行:

打开一个*Chrome*浏览器并且连接到程序 在一个浏览器中执行端到端测试 在终端/命令行窗口中报告测试结果 关闭/退出浏览器

无论你对程作了何种修改都运行端到端测试来确认修改仍然使得程序的整体操作是正确的，在把修改提交到远程仓库前完整运行端到端测试的行为是很常见的。

步骤0（step-0）——准备

你已经准备开始建立AngularJS phonecat程序了。在这一步中，你将熟悉最重要的源代码文件，了解如何开始与angular-seed绑定在一起的开发服务，并且在浏览器中运行程序。

在 `angular-phonecat` 目录中运行：

```
git checkout -f step-0
```

这将把你的工作区切换到教程程序的 step-0分支中。完成教程学习中你会重复这个步骤（只是后面的数字会不同）以切换到不同步骤分支中。注意的是这将丢失你在原有工作区中做的所有改动。

如果你不确定你已经安装了所有的依赖，需要运行一次：

```
npm install
```

为了在浏览器中运行程序，打开终端/命令行窗口，并运行 `npm start` 来开始web服务。现在，打开浏览器窗口，在地址栏输入 `[http://localhost:8000/app/](http://localhost:8000/app/)` 就可以访问到程序了。

如果你在浏览器中看见个页面（不是报错信息），则表示工作是正常的，当然现在显示的内容不让人兴奋，但这还是很好的。

这个HTML页面会显示 "Nothing here yet!"，它是由如下的HTML代码定义的，这些代码包括了我们后面需要进一步利用的Angular关键元素。 `app/index.html`：

```
<!doctype html>
<html lang="en" ng-app>
<head>
  <meta charset="utf-8">
  <title>My HTML File</title>
  <link rel="stylesheet" href="bower_components/bootstrap/dist/css/bootstrap.css">
  <link rel="stylesheet" href="css/app.css">
  <script src="bower_components/angular/angular.js"></script>
</head>
<body>

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

</body>
</html>
```

这些代码做了什么？

`ng-app` 指令：

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

这个 `ng-app` 属性代表一个Angular指令 `ngApp` (在Angular约定 `spinal-case` 暨连词线拼接词用在定制属性中， `camelCase` 暨驼峰拼接词用在指令中，并提供一致的效果）。这个指令标志这个html元素会被Angular

用作应用程序的根（root）元素。这将告诉Angular是整个html页面还是部分元素作为Angular程序。

AngularJS脚本标签:

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

这段代码将让浏览器下载angular.js脚本，并注册一个在浏览器完整下载HTML并初始化后会执行的回调。当回调执行了，Angular将搜索 `ngApp` 指令，如果找到了，就将以ngApp指令定义的DOM作为程序根元素来启动程序。

双大括号（Double-curly）绑定表达式:

```
Nothing here {{'yet'+'!'}}
```

这一行展示了Angular模板应用的两个核心功能：

- 一个绑定需要被双大括号（Double-curly） `{{ }}` 括起来
- 简单表达式 `'yet'+'!'` 可以用于绑定

绑定告诉Angular需要进行对表达式求值，并把结果插入放置在绑定的DOM内。注意的是这种插入不是一次性的，在接下来的步骤中你会更多的了解体验到，它会自动感知表达式结果值发生的变化，并及时更新。

Angular表达式(`Angular expression`)是类似JavaScript的代码，其在当前的Angular环境数据模型空间（依上下文）中求值，这不同于全局数据空间（window DOM）。

正如预期，这个模板由Angular处理后，在html页面会显示文字: "Nothing here yet!"

AngularJS程序的启动

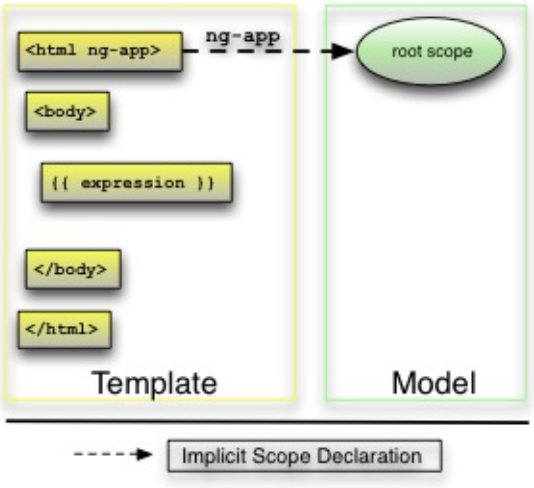
使用 `ngApp` 指令来自动启动AngularJS程序简单且适用于大多数情景。在高级例子中，例如利用脚本加载器，你可以使用 指令来手动控制（参见开发指南的 `imperative / manual way`` 部分）如何启动程序。

在程序启动过程中其实有3件重要的事情发生：

1. `injector` 用于创建了一个依赖注入
2. `injector` 创建了一个根作用范围(`root scope`)，这将用作程序的数据模型上下文环境
3. Angular会"编译(compile)" `ngApp` 为根开始的DOM元素，并在其下执行指令和发现的绑定

一旦程序启动完成，他将等待传入的浏览器事件（例如鼠标点击、键盘输入或者HTTP响应），这都意味着（数据）模型的改变。一旦模型改变发生了，Angular会检测模型改变（改变被找到），Angular会通过更新视图在所有受到影响的绑定中反映出变化。

当前我们的程序结构还十分简单。这个模板只包括了一个指令，还是一个静态的绑定，我们的数据模型其实还是空的，不过马上就会改变！



在我们的工作目录有哪些文件？

工作区大部分的文件是来自于 `angular-seed`，它们通常用于开始一个新的Angular项目。原子项目通常预定义安装了angular框架（通过 `bower` 安装到 `app/bower_components/` 目录下）和一些开发常见App会用到的工具（通过 `npm`)。

具体到本教程中，我们编辑改变了 `angular-seed` 这些部分：

- 移除了其所有例子app
- 添加一些手机图片到 `app/img/phones/`
- 添加手机数据文件(JSON格式)到 `app/phones/`
- 在 `bower.json` 文件中添加了对Bootstrap的依赖

尝试

尝试添加一个新的表达式到 `index.html` 文件中，它可以执行一些数学计算：

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

小结

让我们进入到下个步骤，添加更多内容到web app中

步骤1——静态模板

这里将举例展示Angular是如何增强标准HTML的。你将创建一个纯粹的静态HTML页面，下面的步骤你会看到我们如何把HTML代码转为Angular模板，通过数据来动态显示出相同的结果。

在这一步中，我们将添加两部手机的基本信息到一个HTML页面中。*页面现在显示了两个手机的一个列表信息

工作区切换到步骤1

让工作区切换到合适的分支(注意是在`angular-phonecat`目录中执行下面的操作，以后各步都会执行类似的操作，只是最后的数字不同，所以将不再详细说明和介绍，只是提示切换到步骤多少)：

> 切换到步骤1
>
> ```
> git checkout -f step-1
> ```

你也可以直接用浏览器访问步骤1在线演示

大多数的重要改变都会列在下面，不过你也可以在GitHub看到完整的差异。

介绍

`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>
```

尝试

试着再添加一些静态HTML，例如：

```
<p>总共手机数量：2</p>
```

小结

这里添加了一个静态HTML来显示一个列表，让我们到步骤2去学习如何利用AngularJS来动态生成相同的列表。

步骤2——Angular模板

现在我们让利用AngularJS让页面动态化。我们还将添加一些测试，这将检验我们添加的控制器代码会执行预期的工作。

其实有很多方法来构建应用程序，在Angular中，我们推荐采用Model-View-Controller（MVC——模型-视图-控制器）的设计模式来解耦代码与单独的问题。利用这点，我们在model、view和controller中用到尽量少的Angular和Javascript。

- 程序将展示利用数据动态的列出3部手机列表。

工作区切换到步骤2

直接用浏览器访问步骤2在线演示

大多数的重要改变都会列在下面，不过你也可以在GitHub看到完整的差异。

视图（view）和模板（template）

在Angular中，视图是数据模型通过HTML模板的投影，这意味着任何数据模型的变化，Angular都会针对绑定点刷新视图。

这里的视图组件是由Angular从下面的模板构建的：

`app/index.html`:

```
<html ng-app="phonecatApp">
<head>
  ...
  <script src="bower_components/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>
```

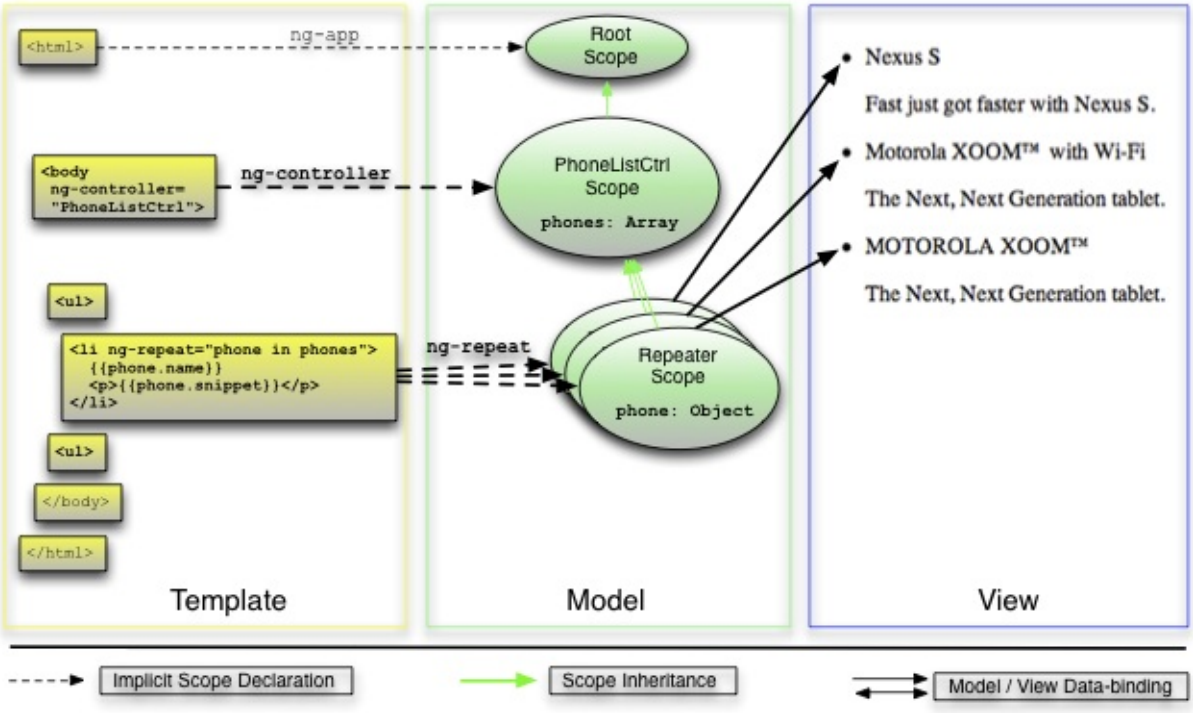
这里我们用`ngRepeat指令`和两个`Angular表达式(expression)`取代了硬编码的电话列表:

- `<li>`的`ng-repeat="phone in phones"`属性就是一个Angular重复指令，它告诉angular在列表中为每个电话信息重复创建`<li>`元素，这里的`<li>`标签就作为了模板。
- 在双大括号中的表达式`{{phone.name}}`和`{{phone.snippet}}`会用实际的值替代。

这里，我们还添加了新的指令，叫做`ng-controller`，它为`<body>`标签指定了一个`PhoneListCtrl`的控制器，对于这点

- 在双大括号中的表达式 {{phone.name}} 和 {{phone.snippet}} 指定的绑定，将参考我们程序中的数据模型，这些都是在我们的 PhoneListCtrl 的控制器进行设置的。

> 注意：我们已经通过加载使用 ng-app="phonecatApp" 来指定了一个Angular数据模型，这里 phonecatApp 是我们数据模型的名字，这个模型被包含在 PhoneListCtrl 中。



数据模型（Model）和控制器（Controller）

这里的数据模型（**Model**）（一个简单的电话信息数值）是由 PhoneListCtrl 控制器（**Controller**）实例化的。这个控制器只是简单的一个构造函数，它操作了一个$scope参数。

app/js/controllers.js：

```
var phonecatApp = angular.module('phonecatApp', []);

phonecatApp.controller('PhoneListCtrl', function ($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.'}
  ];
});
```

这里，我们定义了一个叫做 PhoneListCtrl 的控制器，并且注册它到一个叫做 phonecatApp 的AngularJS数据模型中。注意，我们在 <html> 中的 ng-app 指令现在指定了一个 phonecatApp 的数据模型，这将在启动Angular过程中加载。

尽管现在看来控制器并没有做什么特别的事，但它是至关重要的。通过上下文数据模型,控制器允许我们建立模型和视图之间的数据绑定。我们由此连接起二者间的描述、数据和逻辑，它们是如下工作的：

- `ngController` 指令定位到 `<body>` 标签，引用指定名称的控制器， `PhoneListCtrl` 在JavaScript文件 `controllers.js` 中设定。
- `PhoneListCtrl` 控制器把数据链接到 `$scope` 来注入我们的控制器函数。这个*scope*是一个在应用定义时预定义（指定）的根作用范围*(root scope)*，然后这个控制器通过 `<body ng-controller="PhoneListCtrl">` 标签绑定。

作用范围

在Angular中作用范围的概念至关重要。作用范围被视作把模型、视图和控制器粘合起来协同工作的胶水。Angular利用作用范围来整合模板中的信息、数据模型和控制器，保持模型和视图之间的独立与同步。任何在模型中的变化都被反映到视图中，在视图中的操作变化也反映到数据模型。

要了解更多的关于Angular作用范围的内容，请参考**Angular**作用范围文档主题部分。

测试

Angular分离了控制器和视图，这使得更容易实施测试驱动的开发。如果我们的控制器可以在全局命名空间起效,那么我们可以在一个模拟的范围内实例化它：

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

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

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

});
```

这个对于 `PhoneListCtrl` 的测试实例检验了手机数组中是否有3条记录。这个例子简单的示范了如何为Angular项目创建一个单元测试。因为测试是软件开发的关键部分，我们可以让这一过程更简单，从而鼓励开发者写测试。

测试非全局控制器

实际上，你不一定想你的控制器工作在全局名称空间中，反而你更愿意把控制器通过匿名构造函数注册为 `phonecatApp` 的模块。在这种情况下Angular提供了一种服务(`$controller`)让你可以通过名字查询到控制器。下面就是利用 `$controller` 进行相同的测试: `test/unit/controllersSpec.js` :

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

  beforeEach(module('phonecatApp'));

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

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

});
```

- 在每个测试前告诉Angular要加载 `phonecatApp` 数据模型
- 请求Angular向我们的测试函数注入 `$controller` 服务
- 我们利用 `$controller` 来创建 `PhoneListCtrl` 的实例 *通过这个实例，我们检测范围内的电话数组是否有预期的3条记录

编写并运行测试

Angular开发者一般更喜欢采用Jasmine的行为驱动开发(Behavior-driven Development——BDD)框架来写测试。但是Angular并不限定你只能使用Jasmine。我们（原文作者）的为教程采用Jasmine V1.3编写过所有测试。如果你打算学习下Jasmine，可以到Jasmine主页和Jasmine文档去访问以学到更多。

angular-seed 预配采用Kama进行单元测试，你需要确保Karma的支持组件安装好了。对此你只需要简单的执行 `npm install`

为了运行测试，或者在文件修改后检测，你需要: `npm test`

- Karma会自动打开一个新的Chrome实例，只是忽略它而运行在后台。Karma会使用浏览器来运行测试。
- 你会在控制台看到类似下面的输出：

  ```
  info: Karma server started at http://localhost:9876/
  info (launcher): Starting  browser "Chrome"
  info (Chrome 22.0): Connected on socket id tPUm9DXcLHtZTKbAEO-n
  Chrome 22.0: Executed 1 of 1 SUCCESS (0.093 secs / 0.004 secs)
  ```

  这代表了测试通过或者没有通过
- 在测试没有主动关闭前，在对源码或者test.js有任何修改后都会自动运行测试。Karma会注意到变化并执行测试返回结果给你。难道这不是很好？

  > 确认你没有最小化karma打开的浏览器。在一些操作系统中，最小化浏览器会导致内存访问受限，这使得karma测试运行的特别慢。

尝试

为 `index.html` 添加其他的绑定，例如：

```
<p>Total number of phones:{{phones.length}}</p>
```

在控制器中创建一个新的模型属性，并绑定到模板。例如：

```
$scope.name = "World";
```

为 index.html 添加新的绑定：

```
<p>Hello,{{name}}!</p>
```

刷新你的浏览器来检测是否会显示"Hello, World!"。 在 ./test/unit/controllersSpec.js 中为控制器更新单元测试来反映前面的修改。例如添加：

```
expect(scope.name).toBe('World');
```

利用 repeater 指令在 index.html 中构建一个简单的表格：

```
<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>
```

现在把列表改成对i都增加1的绑定：

```
<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>
```

额外还可以尝试通过使用附加的ngrepeat实现8×8的表格。

改变测试项让单元测试失败:如把 expect(scope.phones.length).toBe(3) 改成 toBe(4) 。

小结

你现在已经有了一个动态程序，有了分离的数据模型、视图和控制组件，你还测试了它们按预期工作，现在让我们进入步骤3去学习如何为程序添加文字搜索。

步骤3——过滤转换器

我们在上一步我们做了大量的工作，为后面打下了良好基础。所以现在我们可以简单的扩展就添加一个全文搜索的功能（是的，这很简单！）。我们也会写一个端到端（end-to-end,E2E）测试,因为良好的端到端测试是应用的好朋友，就像有一双眼睛时刻看着应用，快速定位故障。

- 程序会有一个搜索框。你需要注意在搜索框输入时电话列表的变化。

工作区切换到步骤3

直接用浏览器访问步骤3在线演示

大多数的重要改变都会列在下面，不过你也可以在GitHub看到完整的差异。

控制器

这里我们没有修改控制器

模板

`app/index.html`:

```
<div class="container-fluid">
  <div class="row">
    <div class="col-md-2">
      <!--工具条内容-->

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

    </div>
    <div class="col-md-10">
      <!--主体内容-->

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

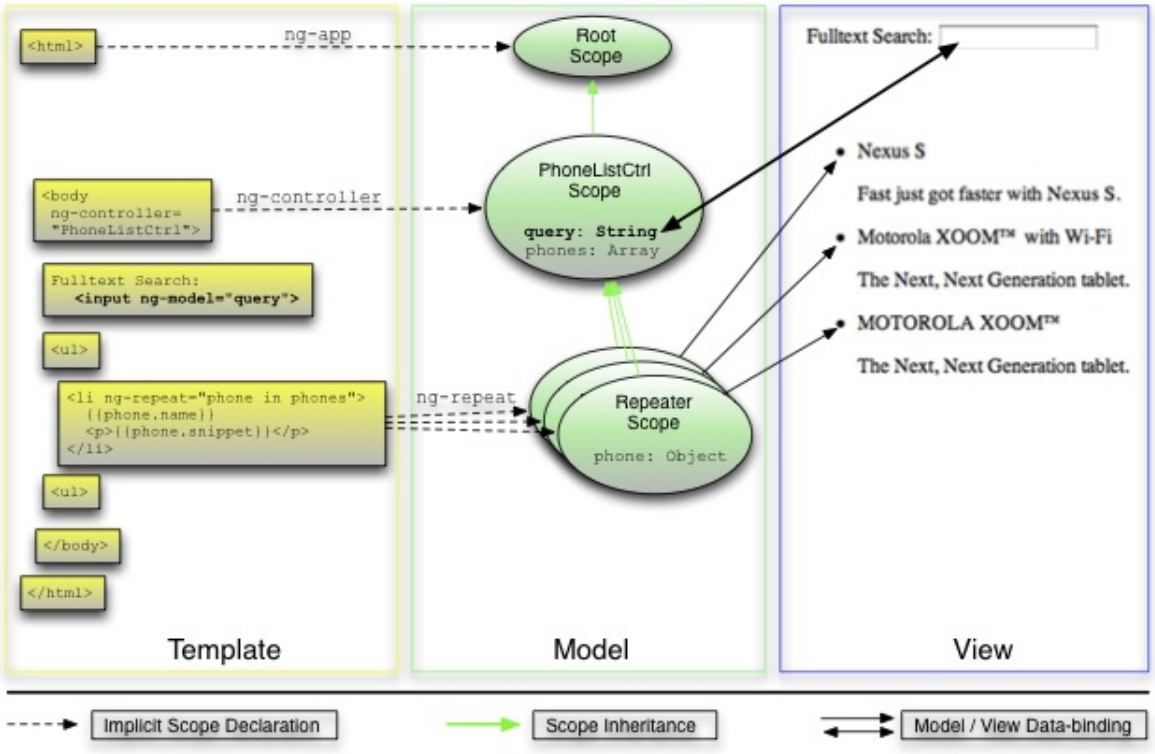
    </div>
  </div>
</div>
```

我们添加了一个标准的HTML `<input>` 标签，并对ngRepeat指令应用了Angular的filter功能来处理 `<input>` 输入。

就是这些改变让用户输入搜索条件后马上可以在手机列表中看到对应的变化。这些新代码展示了：

- 数据绑定：这一Angular核心特性。当页面加载后，Angular把input标签定义的输入框的值绑定到一个命名变量中（页面中命名为query的变量），这个变量还以相同的名字存在于数据模型中，二者是完全同步的。在这个代码中输入框键入的内容（绑定到 `query` ）会立即作为列表输出时的过滤条件 (通

过 `phone in phones | filter:query` 这一句)。数据模型的变化导致转换器的输入改变,转换器通过更新DOM来反映模型的当前状态。



- 使用 `filter` 过滤器： `filter` 功能使用 `query` 来创建一个新的数组（只有那些记录中匹配了 `query` 对应值）， `ngRepeat` 自动根据附加了 `filter` 变动的手机列表数据。这在开发过程中是完全透明的。

测试

在步骤2中，我们学习了如何写单元测试。单元测试可以测试控制器和其他由JavaScript写的组件，但不能方便测试DOM操作或者我们写的程序本身。对此，端到端测试就是更好的选择。

这个搜索功能完全是通过模板和数据绑定来实施的，所以我们写出第1个端到端测试来验证这个工作。 `test/e2e/scenarios.js` :

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

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

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


    it('根据用户在search输入框中的输入过滤手机列表', function() {

      var phoneList = element.all(by.repeater('phone in phones'));
      var query = element(by.model('query'));

      expect(phoneList.count()).toBe(3);

      query.sendKeys('nexus');
      expect(phoneList.count()).toBe(1);

      query.clear();
      query.sendKeys('motorola');
      expect(phoneList.count()).toBe(2);
    });
  });
});
```

这个测试验证了搜索框和转换器很好的连接在一起工作。你可以看出在Angular中是多么容易的写出端到端测试啊。虽然这个例子展示的是一个简单测试，但它却是是十分容易设置功能的、且可读的端到端测试。

利用Protractor运行端到端测试

尽管这些测试语句很像我们前面用Jasmine写的对控制器测试的语法，但端到端测试是使用的Protractor提供的APIs。要想了解关于这些Protractor APIs的更多信息，可以阅读 https://github.com/angular/protractor/blob/master/docs/api.md

我们用Protractor来完成端到端测试与采用Karma进行单元测试过程类似，尝试运行 `npm run protractor`。端到端测试比单元测试运行的速度要慢些。Protractor会在运行了一次端到端测试后自动退出，而不会自动的在测试区内文件变化时运行，为此，你必须在有变化后手动启动端到端测试，即再次运行 `npm run protractor`。

> 注意：你必须确认你已经打开了一个web服务来供protractor测试使用，你可以采用 `npm start` （译者注：这时你有两个终端/命令行在运行，其中一个运行着 `npm start` 以提供web服务，另外一个执行着 `npm run protractor` 来进行端到端测试）。你还需要确认你已经安装了protractor 并且升级了webdriver使之符合运行 `npm run protractor` 的条件，这需要你之前执行过 `npm install` 和 `npm run update-webdriver`

尝试

显示当前的查询条件

即在 index.html 中添加用 `{{query}}` 来显示当前 `query` 模型中当前值的模板设计，这可以观察到当输入框键入信息改变时最直接的变化。

在标题中显示查询

让我们看看如何把`query`模型的值赋予HTML页面标题：

- 添加一个端到端测试到`test/e2e/scenarios.js`文件中`describe`部分：

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

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

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

    var phoneList = element.all(by.repeater('phone in phones'));
    var query = element(by.model('query'));

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

      query.sendKeys('nexus');
      expect(phoneList.count()).toBe(1);

      query.clear();
      query.sendKeys('motorola');
      expect(phoneList.count()).toBe(2);
    });

    it('should display the current filter value in the title bar', function() {
      query.clear();
      expect(browser.getTitle()).toMatch(/Google Phone Gallery:\s*$/);

      query.sendKeys('nexus');
      expect(browser.getTitle()).toMatch(/Google Phone Gallery: nexus$/);
    });
  });
});
```

现在运行protractor（`npm run protractor`)肯定会看到失败信息。

- 你想该如何添加`{{query}}`到title标签，是不是应该像下面：

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

然而你直接刷新页面将不能看到期望的结果，这是因为"query"模型只存在于一定的作用范围(scope)内，这里是利用`ng-controller="PhoneListCtrl"命令定义在body元素内的：

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

这意味着只有在body标签起效的区域（其本身和包括在内的子标签内）是query有效作用范围，之外则不是，而title标签是在这之外的。所以为了让这个数据模型在`<title>`内也有效，我们需要移动/改变`ngController`使得对整个HTML元素都有效（只有这个标签是同时包括了body和title的）：

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

并且确认已经把 `ng-controller` 从body标签中移除了。

- 再运行 `npm run protractor`，这时你将看到测试通过的信息。
- 可能你注意到了因为angular的加载和初始化是需要时间的，你在浏览器标题栏中会在一瞬间看到如 `Google Phone Gallery:{{query}}` 这样的信息，但其实这不是你希望的（这时angular的数据绑定还没有起作用，就造成了前面的原始数据显示，和后面起作用后的正确显示），对此一个更好的处理方法是使用 `ngBind` 或者 `ngBindTemplate` 命令，这使得在加载器无关的信息对用户来说是不可见的：

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

小结

我们已经实现了文本搜索功能，还为该功能提供了测试，现在让我们进入步骤4，以给这个程序增添排序能力。

步骤4——双向数据绑定

在这一步骤中，我们将为程序添加一个特性使得可以对手机的显示排序。这还需要我们为手机数据模型中添加一些新信息作为排序的依据，并以此写出转换器的处理让数据起作用使得程序达到预期效果。

- 程序现在除了有一个搜索框还有一个下拉选择框，它允许选择数据排序的要求。

工作区切换到步骤4

直接用浏览器访问步骤4在线演示

大多数的重要改变都会列在下面，不过你也可以在GitHub看到完整的差异。

模板

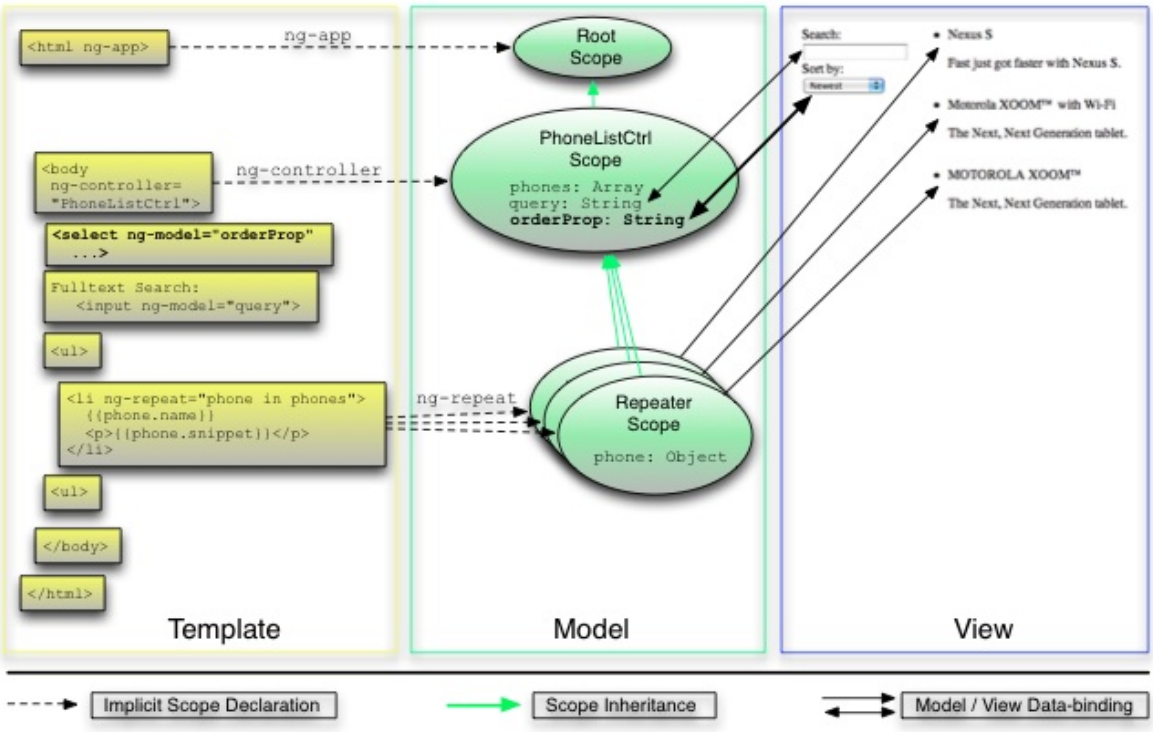
`app/index.html`：

```
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">
    <span>{{phone.name}}</span>
    <p>{{phone.snippet}}</p>
  </li>
</ul>
```

这里只列出了在`index.html`模板中发生变化的部分，主要有：

- 首先，我们添加了一个`<select>`的html元素，并命名为`orderProp`，这样我们就有了两个排序依据选择了。



- 然后我们链接了经过 filter 转换器过滤后的数据作为 orderBy 转换器处理的输入。 orderBy 转换器会依据规则对输入的数据（数组）排序，即输出一个排序后的新数据（数组形式）。

在这里，Angular在 select 元素和 orderProp 数据模型间创建了一个双向数据绑定，然后orderProp被用于 orderBy 转换器（过滤器）。

正如我们在步骤3中看到的，因为这样的数据绑定使得任何的数据变化都可以及时的反映到输出结果中（这里是通过下拉菜单对排序条件的改变），而且这一过程中没有特别指定复杂的DOM操作处理（没有专门写这方面的代码）就自动实现了效果，这就是Angular数据绑定的威力所在。

控制器

app/js/controllers.js :

```
var phonecatApp = angular.module('phonecatApp', []);

phonecatApp.controller('PhoneListCtrl', function ($scope) {
  $scope.phones = [
    {'name': 'Nexus S',
     'snippet': 'Fast just got faster with Nexus S.',
     'age': 1},
    {'name': 'Motorola XOOM™ with Wi-Fi',
     'snippet': 'The Next, Next Generation tablet.',
     'age': 2},
    {'name': 'MOTOROLA XOOM™',
     'snippet': 'The Next, Next Generation tablet.',
     'age': 3}
  ];

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

- 我们编辑了 phones 的数据模型，即手机数组，在这个结构对每条记录中增加了 age 元素项，其可用于

按手机推出时间排序。

- 我们在控制器中设置了默认的值作为排序依据，即设置了 `orderProp` 的值为 `age`。如果不在这里设置这个值，则 `orderBy` 转换器（过滤器）是没有初始化的，直到我们在页面下拉菜单中进行了选择为止。

现在可以好好来谈谈双向数据绑定了。注意，当浏览器加载完成程序后下拉菜单的 `Newest` 项目是被选中的，这就是因为我们在控制器中设置了 `orderProp` 的排序依据是 `age`，所以发生了数据模型向UI的绑定，现在我们在下拉菜单选择 `Alphabetically` 项，则数据模型马上会自动更新，让手机列表的排序依据要求发生变化，这时的数据绑定方向与前面恰好相反，是UI向数据模型的。

测试

这里的改变需要单元测试和端到端测试两个方面内容，先来看看单元测试。 `test/unit/controllersSpec.js`：

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

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

    beforeEach(module('phonecatApp'));

    beforeEach(inject(function($controller) {
      scope = {};
      ctrl = $controller('PhoneListCtrl', {$scope: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');
    });
  });
});
```

这个单元测试现在验证了默认排序的设置。

我们在 `beforeEach` 块内使用Jasmine的API来提取控制器内容，测试中这将在父级 `describe` 块内共享相关内容。

你现在应该能看到Karma输出类似下面信息：

```
Chrome 22.0: Executed 2 of 2 SUCCESS (0.021 secs / 0.001 secs)
```

再看看端到端测试。 `test/e2e/scenarios.js`：

```
...
    it('should be possible to control phone order via the drop down select box', function() {

      var phoneNameColumn = element.all(by.repeater('phone in phones').column('{{phone.name}}'));
      var query = element(by.model('query'));

      function getNames() {
        return phoneNameColumn.map(function(elm) {
          return elm.getText();
        });
      }

      query.sendKeys('tablet'); //let's narrow the dataset to make the test assertions shorter

      expect(getNames()).toEqual([
        "Motorola XOOM\u2122 with Wi-Fi",
        "MOTOROLA XOOM\u2122"
      ]);

      element(by.model('orderProp')).element(by.css('option[value="name"]')).click();

      expect(getNames()).toEqual([
        "MOTOROLA XOOM\u2122",
        "Motorola XOOM\u2122 with Wi-Fi"
      ]);
    });...
```

这里端到端测试验证下拉菜单的选择是否正确。我们现在可以运行 `npm run protractor` 来看看运行情况。

尝试

在 `PhoneListCtrl` 控制器中移除掉 `orderProp` 值的设置，你就会在Angular模板加载时看到下拉菜单会有一个"unknown"（显示为空白）项目，而且列表也会以“未排序/原始定义顺序”来排列。

在 `index.html` 模板添加一个`{{orderProp}}的绑定来显示当前的值。

反转排序只需要在排序值前面添加一个 `-` 号：

```
<option value="-age">Oldest</option>
```

小结

这一步我们为程序添加了排序功能，让我们进入步骤5，继续学习Angular的服务和依赖注入。

步骤5——XHRs与依赖注入

显然，一个足够强大的程序仅仅依靠硬编码的3条手机数据设置是不行的。这里我们将利用Angular内置的叫做 `$http` 的服务功能，从web服务器获取一个巨大的数据表。我们还将使用到Angular的依赖注入(dependency injection)向 `PhoneListCtrl` 控制器提供服务。

- 现在有20条手机信息了，而且这些信息是通过服务器加载的。

工作区切换到步骤5

直接用浏览器访问步骤5在线演示

大多数的重要改变都会列在下面，不过你也可以在GitHub看到完整的差异。

数据

在项目 `app/phones/phones.json` 文件中有一个巨大的列表，，是采用JSON格式组织的手机信息数据。文件中，它们大概是如下的形式:

```
[
 {
  "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

这里的控制器中，我们使用了Angular的 `$http` 服务，利用一个HTTP请求，从服务器中获取到 `app/phones/phones.json` 文件数据。 `$http` 是Angular内建web程序通用服务（功能）中的一个，Angular会在程序需要时自动注入这些服务功能。

这些服务由Angular依赖注入子系统进行管理。依赖注入帮助你的程序在好的结构（例如独立的数据、控制和表现/展示)和松耦合（组件间解耦，组件之间的依赖关系不由组件自身确定，而由依赖管理子系统协调）。 `app/js/controllers.js` :

```
var phonecatApp = angular.module('phonecatApp', []);

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

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

`$http`发起一个HTTP GET请求，其内容是请求web服务器端的`phones/phones.json`文件（这里的URL是相对于`index.html`的相对路径）。服务器端响应这个请求，提供了json文件的内容（响应其实是一个后台服务中动态处理的反馈结果，这对于浏览器或者我们的程序来说这看起来是相同的/透明的。这个教程中为了简单起便，直接是一个json数据文件了。）

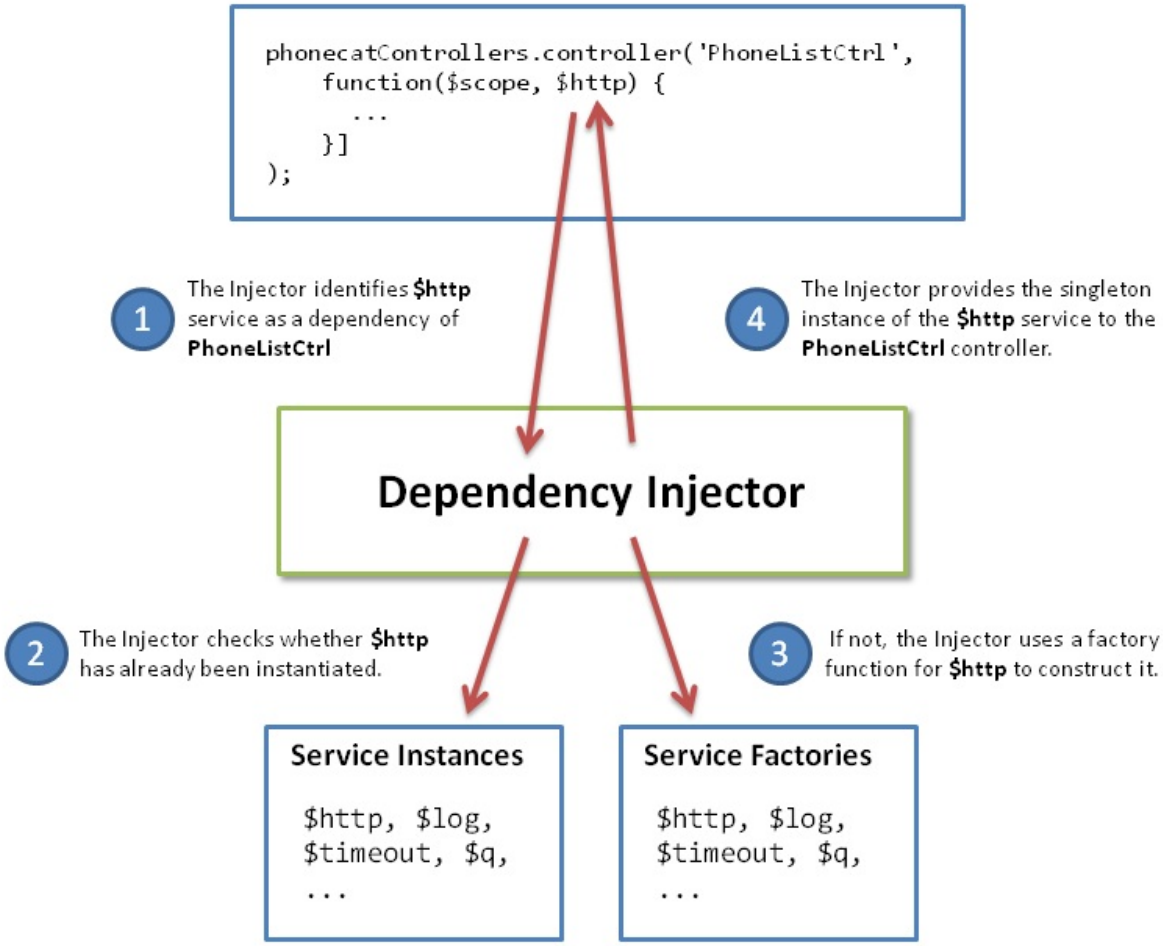
`$http`服务得到了一个有`success`方法的`promise object`，然后我们就可以调用方法来处理异步响应和分配手机数据来构建我们控制器中作用范围中的`phones`数据了。注意，这里Angular自动检测了json类型响应，并分析结构化了数据。

为了使用Angular服务，你只用在控制器中需要的地方简单把调用名字作为构造函数的参数，例如:

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

Angular的依赖注入管理会在控制器初始化时自动的提供声明的功能，而且依赖注入管理还自动的处理相应的层次依赖关系（通常一个服务功能还取决于其他服务功能，这些问题Angular都会自动处理）。

注意参数的名字很重要（不能随意变动），因为依赖管理会用到这些名字进行查找来解决依赖关系并进行注入。



$ 前缀名的约定

你可以创建自己的服务，事实上我们在步骤11中就会这样做。作为一个名称方面的约定，Angular内置的服务，作用范围方法和一些其它的Angular的API有一个`$`前缀。

这样有 `$` 前缀的都被用于了Angular预定服务，为了防止名称方面的冲突，你在定义服务和数据模型时最好都不要用 `$` 作为前缀。

在检查代码时你可能会注意到有些作用范围内的数据定义是 `$$` 来开始的，这意味这这些内容是私人的（该受保护的），你不该在外部进行访问或修改。

在压缩代码时需要关注的地方

因为Angular的依赖控制采用了以名字作为构造函数的参数传入机制运行，所以如果你想压缩你的 `PhoneListCtrl` 控制器部分JavaScript代码就需要注意一些细节，否则自动机制下所有的参数名会自动压缩而导致依赖注入功能出错。

对于这样的问题就是提供一个禁止压缩的依赖名称列表，这样列表中的名称在压缩时不会进行缩减替换，这样就能保证压缩后的代码能够正常工作了，对此有两个方法：

- 在控制器构造函数上创建一个 `inject` (注入)字符串数组，数组中每个字符串都是需要注入的服务名。在我们的例子中就是这样写：

```
function PhoneListCtrl($scope, $http) {...}
PhoneListCtrl.$inject = ['$scope', '$http'];
phonecatApp.controller('PhoneListCtrl', PhoneListCtrl);
```

- 使用内联注解语句，函数不是仅仅提供功能要求，还包括一个功能名的字符串数组（内联注解数组），例如:

```
function PhoneListCtrl($scope, $http) {...}
phonecatApp.controller('PhoneListCtrl', ['$scope', '$http', PhoneListCtrl]);
```

这两种方法都可以被Angular正常识别进行注入，所以你只需要依据你项目风格选择一种即可。

当采用第二种方式，通常定义一个内联的匿名函数供注册器实施注入:

```
phonecatApp.controller('PhoneListCtrl', ['$scope', '$http', function($scope, $http) {...}]);
```

从现在开始，教程中我们将采用内联注解方式（第二种方法）进行处理，使得代码支持压缩。让我们为 `PhoneListCtrl` 添加一个内联注解： `app/js/controllers.js` :

```
var phonecatApp = angular.module('phonecatApp', []);

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

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

测试

test/unit/controllersSpec.js：因为我们应用了依赖注入使得控制器有了依赖关系，所以构建控制器的单元测试变动有点复杂了。我们可以使用 new 运算符在构造中提供 $http 的模拟实现。然而Angular提供了用于单元测试的 $http 模拟，我们需要配置 $httpBackend 实现“模拟”服务器响应来完成单元测试：

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

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

    // Load our app module definition before each test.
    beforeEach(module('phonecatApp'));

    // The injector ignores leading and trailing underscores here (i.e. _$httpBackend_).
    // This allows us to inject a service but then attach it to a variable
    // with the same name as the service in order to avoid a name conflict.
    beforeEach(inject(function(_$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：因为需要加载Jasmine和 angular-mocks.js 到测试环境，所以我们有两个辅助方法 module 和 inject 用于访问和配置注入器。

我们在测试环境中创建控制器：

- 我们使用 inject 辅助方法向Jasmine的 $beforeEach 函数中注入 $rootScope 、 $controller``$httpBackend 等功能实例。这些注入的功能实例用来从头到尾创建每一个测试，这可以保证每一个测试都是从共用的起点开始，但有互相隔离的。 *我们在我们的控制器中新建了一个作用范围叫做: $rootScope.$new()
- 我们通过 $controller 调用 PhonelistCtrl （其由名字进行了注入，并且有指定的作用范围）

因为我们的代码使用 $http 服务获取手机列表数据到控制器中，所以之前我们需要在 PhoneListCtrl 中创建子作用范围，我们要告诉测试需要用预期的输入请求（返回结果）替代数据，对此我们需要:

- 在 beforEach 函数中注入 $httpBackend 的请求服务，这个模拟服务功能将模拟生产环境中所有的XHR和JSONP请求。这样的模拟服务允许您编写测试时无需处理本地api和与它们相关的全局状态（服务器端响应结果）——这些都是测试的噩梦

- 使用 $httpBackend.expectGET 这样的 $httpBackend 服务方法链可以模拟预期的HTTP请求和响应。注意：最终结果的返回是在调用了 $httpBackend.flush 方法后

现在我们可以验证 phones 数据模型通过请求构建完成前，是否已经有 scope 了:

```
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'}]);
});
```

- 我们利用 `$httpBackend.flush()` 激活浏览器请求序列，这导致 `$http` 服务返回预期的响应。
- 我们验证手机数据模型是否已经存在于作用范围内。

最后，我们验证默认的 `orderProp` 值是否设置正确:

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

这时你应该看到类似如下信息的Karma输出：

```
Chrome 22.0: Executed 2 of 2 SUCCESS (0.028 secs / 0.007 secs)
```

尝试

在 `index.html` 的底部添加`

```
{{phones | json }}
```

的数据绑定输出来显示json格式的手机数据。

在 `PhoneListCtrl` 控制器中，利用手机数字序号预处理http的响应结果来限定数据值的范围，可以把下面的代码加入 `$http` 回调中：

```
$scope.phones = data.splice(0, 5);
```

小结

现在你了解学习了使用Angular服务是多么简单（依靠Angular依赖注入机制），下面跳到步骤6，将为手机添加图片缩略图和一些链接。

步骤6——模板链接与图形

在这一步，会为手机列表添加缩略图以及对应的购买信息链接。后续步骤中，你将通过链接显示目录中手机的更多附加（详细）信息。

- 现在就为手机列表添加缩略图和链接。

工作区切换到步骤6

直接用浏览器访问步骤6在线演示

大多数的重要改变都会列在下面，不过你也可以在GitHub看到完整的差异。

数据

注意到在 `phones.json` 中包含了每个手机独一无二的id和图片链接地址。这些链接地址都指向 `app/img/phones/` 目录。 `app/phones/phones.json` （一段手机数据的快照）

```
[
  {
    ...
    "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>
...
```

为了动态处理链接（这在后面将导向手机的详细介绍页面），我们给 `href` 属性的赋值中用到了由双大括号括起来的数据绑定。在步骤2中，我们把 `{{phone.name}}` 绑定到一个html元素的内容中，这一步 `{{phone.id}}` 将用来绑定到元素属性中。

我们还为每款手机添加了图片，这里用到了 `ngSrc` 命令，它会阻止浏览器在Angular还没有初始化并能够正常展开绑定时发出诸如下面一样的无效URL请求: `http://localhost:8000/app/{{phone.imageUrl}}` ，而是仅仅描述当前元素需要一个 `src` 属性，这个属性（ `<img src="{{phone.imageUrl}}">` ，会在Angular初始化

好后进行绑定），使用 ngSrc 指令能始终阻止浏览器发出明显是无效的http请求。

测试

test/e2e/scenarios.js :

```
...
    it('should render phone specific links', function() {
      var query = element(by.model('query'));
      query.sendKeys('nexus');
      element.all(by.css('.phones li a')).first().click();
      browser.getLocationAbsUrl().then(function(url) {
        expect(url.split('#')[1]).toBe('/phones/nexus-s');
      });
    });
...
```

我们添加一个新的端到端测试来验证程序会处理正确的链接，对于这个手机显示程序来说，正是我们要实现的功能。

你现在可以运行 npm run protractor 来看测试情况。

尝试

把 ng-src 指令替换为原来的 src 属性，使用诸如Firebug之类的工具或者Chrome浏览器的Web开发组件观察web访问日志，可以观察到如 /app/%7B%7Bphone.imageUrl%7D%7D 或者 /app/{{phone.imageUrl}} 这样的无效请求，这些无效请求就是浏览器在Angular还没有正确初始化和进行数据绑定（注入）时，构建 img 标签时发出的，这使得图片不能正确显示。

小结

我们已经给手机添加了图片和链接，让我们进入步骤7，学习如何对模板进行布局，以及Angular如何容易的创建一个多视图的程序。

步骤7——路由与多视图

这一步，将学习到如何创建布局模板，以及利用Angular路由模块(ngRoute)实现的多视图应用。 当浏览器导航到 app/index.html 时，会重定向到 app/index.html#/phones 来访问呈现一个手机列表。 当点击手机链接时，URL会改变指向到一款具体的手机，并且显示其详细的信息页面。

工作区切换到步骤7

直接用浏览器访问步骤7在线演示

大多数的重要改变都会列在下面，不过你也可以在GitHub看到完整的差异。

依赖

这一步添加的路由功能是由Angular的 ngRoute 模块提供，这是一个独立于Angular核心框架的模块。

我们使用 Bower 来安装客户端依赖。这一步骤中我们会更新 bower.json 配置文件来包含新的依赖关系:

```
{
  "name": "angular-seed",
  "description": "A starter project for AngularJS",
  "version": "0.0.0",
  "homepage": "https://github.com/angular/angular-seed",
  "license": "MIT",
  "private": true,
  "dependencies": {
    "angular": "1.2.x",
    "angular-mocks": "~1.2.x",
    "bootstrap": "~3.1.1",
    "angular-route": "~1.2.x"
  }
}
```

新的依赖关系 "angular-route": "~1.2.x" 告诉 bower 安装版本1.2.x的 angular-route 组件。我们必须明确告诉 bower 下载安装这个依赖。

如果你是在全局环境中使用bower安装，你可能会用直接到 bower install 指令，但是在这个项目中我们已经预配使用 npm 来启动 bower install ，所以你只需要: npm install

多视图、路由与布局模板

我们的程序逐渐强大，也变得更加复杂。在步骤7（本步骤）之前，我们只有唯一的视图来（用来显示手机列表），并且所有的模板代码都放置在 index.html 中。新的步骤中会添加一个视图来显示列表中每个设备详细的信息（详细说明视图）。

为了添加详细说明视图，我们扩展 index.html 模板文件来包含两个视图，但这将很快引起混乱，为了替代，我们尝试把 index.html 转换到我们称为布局模板（ layout template ）其中有模板（布局模板）在所有视图中通用，其他则是局部模板（ partial templates ），局部模板只包括当前路由 route ——视图当前显示需要的部分。

在Angular中，程序的路由通过 $routeProvider 声明，它是 $route 服务的提供者。这个服务能容易的把控

制器、视图模板和浏览器当前的地址栏信息连接起来。使用这个特性，我们可以实现深层链接(deep linking)，它可以让你可以利用浏览器历史（后退和前进）以及收藏标签。

关于依赖注入（DI）的提醒：注入器（Injector）和提供者（Providers）

是否注意到，依赖注入（DI）是AngularJS的核心，所以了解一下其是如何实现的是十分重要的。

当程序启动时，Angular创建一个注入器用于查找和注入程序需要的服务。注入器自身是不知道任何关于 `$http` 或者 `$route` 服务能做什么的，事实上注入器都不知道这些服务是否存在（除非服务配置在适当的模块定义中）。

注入器只执行以下步骤：

- 加载描述在程序中的模块定义
- 注册所有模块定义的提供者
- 当有实际的请求时，注入器检测具体的功能要求及对应的依赖（服务），再通过其提供者（延时）实例化来完成功能的提供

提供者是一种可以提供（创建）服务实例的对象，其还可以通过配置的API暴露调用接口，用于控制的建立和运行时行为功能。实际上这 `$route` 服务、 `$routeProvider` 暴露的API允许为你的程序定义路由。

> 注意：提供者只能注入到 `config` 功能，因此你不能把 `$routeProvider` 注册入 `$PhoneListCtrl`

Angular模块从应用程序中解决（去除了）全局状态的问题,并提供一个配置注入器的方法。与AMD或者 `require.js` 模块截然相反，Angular模块不会尝试处理脚本的加载和延时获取等问题。这些目标是完全独立的模块系统，它们可以并列存在和实现他们的目标。

要深入理解Angular的依赖注入，观看Understanding Dependency Injection

模板

`$route` 服务通常与 `ngView` 指令联合使用。 `ngView` 指令规包含了视图模板如何路由到的布局视图。这使得 `index.html` 模板显得更加完美。

> 注意:从AngularJS版本1.2开始 `ngRoute` 是独立的模块，需要单独加载 `angular-route.js` 文件，这可以通过 `bower` 下载到。

`app/index.html`

```
<!doctype html>
<html lang="en" ng-app="phonecatApp">
<head>
...
  <script src="bower_components/angular/angular.js"></script>
  <script src="bower_components/angular-route/angular-route.js"></script>
  <script src="js/app.js"></script>
  <script src="js/controllers.js"></script>
</head>
<body>

  <div ng-view></div>

</body>
</html>
```

我们添加了2个新的 `<script>` 标签以在我们的index文件中加载额外的JavaScript文件：

- `angular-route.js` ：定义了Angular的 `ngRoute` 模块，用于提供路由功能
- `app.js` :这个文件在程序中提供一个root模块。

注意我们从index.html模板文件中移除了大部分代码，取而代之的是一个定义了 `ng-view` 属性的 `div` 标签。移除的代码被放置到 `phone-list.html` 模板文件中： `app/partials/phone-list.html` :

```
<div class="container-fluid">
  <div class="row">
    <div class="col-md-2">
      <!--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="col-md-10">
      <!--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>
```

我们还增加了一个包含占位符的手机详细说明视图的模板: `app/partials/phone-detail.html` :

```
TBD:detail view for <span>{{phoneId}}</span>
```

注意现在我们使用了 phoneId 表达式，它是定义在 PhoneDetailCtrl 控制器中的。

程序模块

为了改善程序的组织，我们使用Angular的 ngRoute·模块，并且把控制器移动到自己的模块 phonecatControllers`（下面会看见）。

我们添加 angular-route.js 到 index.html 中，并添加了一个新的 phonecatControllers 模块到 controllers.js 中。这些不是所有我们会用到的代码，我们还要添加一些我们程序依赖的模块。下面两个模块都是 phonecatApp 需要的，我们可用指令和服务提供。

app/js/app.js :

```
var phonecatApp = angular.module('phonecatApp', [
  'ngRoute',
  'phonecatControllers'
]);

...
```

注意传递给 angular.module 的第2个参数 ['ngRoute','phonecatControllers']， 这个数组列出了 phonecatApp 依赖的模块。

```
...

phonecatApp.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'
      });
  }]);
```

通过 phonecatApp.config() 方法，我们请求 $routeProvider 注入自己配置的函数，并且使用 $routeProvider.when() 方法提供了自定义路由规则。

我们的程序定义了如下的路由:

- 匹配 /phones ：当URL结尾片段是 /phones 时显示一个手机列表。为了构造这个输出视图，Angular会使用 phone-list.html 模板和 PhoneListCtrl 控制器
- 匹配 /phones/:phoneId : 当URL结尾片段匹配 /phones/:phoneId 时会显示对应手机的详细说明视图。这里 :phoneId 是一个URL变量区块。为了生成这个手机详细说明视图，Angular使用 phone-detail.html 模板和 PhoneDetailCtrl 控制器。
- 其他的匹配（重定向到 /phones ）:当浏览器地址栏信息不能匹配到有效路由（自定义的路由设置没有对应项目）时，尝试重定向到 /phones

我们再次使用了上一步骤的 PhoneListCtrl 控制器，然后添加了一个新的（但是是空的） PhoneDetailCtrl 控制器到 app/js/controllers.js 文件中来为手机详细说明视图提供数据。

注意在第2个路由声明中的 :phoneId 参数。 $route 服务使用路由声明—— /phones/:phoneId ——作为一个模板来匹配当前的URL。所有的这些变量定义都会在 $routeParams 对象中展开。

控制器

app/js/controllers.js :

```
var phonecatControllers = angular.module('phonecatControllers', []);

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

    $scope.orderProp = 'age';
  }]);

phonecatControllers.controller('PhoneDetailCtrl', ['$scope', '$routeParams',
  function($scope, $routeParams) {
    $scope.phoneId = $routeParams.phoneId;
  }]);
```

再次提醒，我们是创建了一个新的模块叫做 phonecatControolers 。对于很多小的AngularJS程序，共同创造是一个模块为所有控制器(如果有几个)公用。随着程序的增强，很容易为程序添加更多的公用模块。对于大的程序，你需要为每个主要的程序功能创建特定的模块。

因为我们的例子程序是相对比较小的程序，所以我们把所有的控制器都添加到 phoecatControllers 模块中。

测试

为了自动验证所有的可能，我们写端到端测试来导航到各式URL，并且验证是否能显示正确：

```
...
   it('should redirect index.html to index.html#/phones', function() {
    browser.get('app/index.html');
    browser.getLocationAbsUrl().then(function(url) {
       expect(url.split('#')[1]).toBe('/phones');
     });
  });

  describe('Phone list view', function() {
    beforeEach(function() {
      browser.get('app/index.html#/phones');
    });
...

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

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


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

你可以用`npm run protractor`来运行并观察测试结果。

尝试

试着在`index.html`中添加一个`{{orderProp"}}`的绑定观察点，你会发现当手机列表正常显示时，你看不到任何相关信息，这是因为现在`orderProp`数据模型只在`PhoneListCtrl`模型中的作用范围内有效（可见）。为了关联

元素，你需要添加相同绑定到`phone-list.html`模板中，这个绑定将正常工作。

小结

这一步，我们建立了路由，并实现了手机列表视图，接下来的步骤8中，我们将实现手机详细说明视图。

步骤8——更多模板

这一步我们会实现手机详细说明视图，它将在我们点击手机列表中的手机链接后显示。 *当点击手机列表中一个手机的链接，指定手机的详细说明页面将显示出对应手机的详细信息。

工作区切换到步骤8

直接用浏览器访问步骤8在线演示

大多数的重要改变都会列在下面，不过你也可以在GitHub看到完整的差异。

数据

添加 `phones.json` ,在 `app/phones/` 目录中也包括每个手机的json文件: `app/phones/nexus-s.json` :(一个简单的例子)

```
{
 "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"
 }
}
```

每个文件都有相同的属性数据结构，我们会把这些数据显示在手机详细说明视图中。

控制器

我们使用 `$http` 服务请求json文件功能来增强 `PhoneDetailCtrl` 。它们采用手机列表控制器相同的方式工作。 `app/js/controllers.js` :

```
var phonecatControllers = angular.module('phonecatControllers',[]);

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

为了构建HTTP的URL请求，我们从定义在 `$route` 功能中的当前路由中扩展出 `$routeParams.phoneId` 。

模板

前面用占位符粗略定义的模板被手机详细说明各部分的列表和数据绑定替换。注意我们使用了Angular `{{表达式}}` 标签和 `ngRepeat` 来从我们数据模型中提取手机数据供显示。 `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

我们写了一个类似我们在步骤5中 `PhoneListCtrl` 控制器的新单元测试。 `test/unit/controllersSpec.js` :

```
  beforeEach(module('phonecatApp'));

  ...

  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'});
    });
  });
...
```

你应该可以看到类似下面的Karma输出：

```
Chrome 22.0: Executed 3 of 3 SUCCESS (0.039 secs / 0.012 secs)
```

我们也添加一个新的端到端测试导航到Nexus S的详细说明页面验证页面头部是否有"Nexus S"。`test/e2e/scenarios.js`：

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

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


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

你现在可以运行`npm run protractor`来运行端到端测试了。

尝试

使用Protractor API，写一个测试在Nexus S详细说明页面会显示4个缩略图的端到端测试项目。

小结

现在，我们的手机详细说明视图也已经建立好了，进入步骤9，学习如何写自定义的显示过滤器。

步骤9——转换器

在这一步骤中，将学习到如何定制一个显示转换器 *在前一步，详细说明页面直接用"true"和"false"来显示某个手机特性是否被支持，在这里，我们将定制一个转换器，实现用符号:✓ 对应 "true", 以及 ✘ 对应 "false"，来进行显示。让我们看看转换器的代码是怎么样的。

工作区切换到步骤9

直接用浏览器访问步骤9在线演示

大多数的重要改变都会列在下面，不过你也可以在GitHub看到完整的差异。

定制转换器

为了创建一个新的转换器，你需要创建一个 `phonecatFilters` 模块并把定制的转换器注册到这个模块中。 `app/js/filters.js` :

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

我们要定制的转换器叫做"checkmark"。其输入(`input`)是 `true` 或者 `false` ，返回的输出结果根据输入是 `true` (\u2713 -> ✓) 或者 `false` (\u2718 -> ✘)。

现在我们的转换器已经准备好了，需要注册到 `phonecatFilters` 模块中，并成为我们主模块 `phonecatApp` 的一个依赖。

`app/js/app.js` :

```
...
angular.module('phonecatApp', ['ngRoute','phonecatControllers','phonecatFilters']);
...
```

模板

因为转换器的代码存在于 `app/js/filters.js` 文件中，所以我们需要把它添加到布局模板中。 `app/index.html` :

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

在Angular模板中采用如下的语法来应用转换器:

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

让我们把这个转换器应用到手机详细说明模板中: 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

转换器就像其它组件一样，也需要被测试，而且其测试也很容易写。 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');
    }));
  });
});
我们在进行任何转换器测试前要保证调用了`beforeEach(module('phonecatFilters'))`，通过这个调用来加载`phonec

注意我们还调用了辅助功能函数`inject(function(checkmarkFilter) { ... })`来访问我们打算测试的转换器，具体函数

注意注入后，一个后缀"Filter"会被添加到转换器名称后。请查看“Filter Guide ”的相关部分来了解更多。

你现在可以看到类似如下的Karma输出:
```cmd
Chrome 22.0: Executed 4 of 4 SUCCESS (0.034 secs / 0.012 secs)
```

尝试

让我们尝试一些Angular内建的转换器，把这些添加绑定到index.html : {{"lower cap string" | uppercase }} {{foo : "bar",baz :23 | json }} {{1304375948024 | date }} {{ 1304375948024 | date:"MM/dd/yyyy @ h:mma" }}

我们也创建一个输入模块，来包括这些转换器绑定，添加如下代码到index.html :

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

小结

现在我们已经学习了如何写一个定制转换器，并知道如何进行测试，让我们进入步骤10学习如何使用
```

Angular来增强手机详细说明页面。

步骤10——行为处理

在这一步中，我们会对手机图片添加点击支持，点击后会在手机详细说明页面显示新的图形。

- 手机详细说明显示当前手机的一个巨大的图片和一些缩略图。我们点击缩略图会显示对应的大图片，这个效果很令人兴奋。让我们看看如何利用Angular来实现。

工作区切换到步骤10

直接用浏览器访问步骤10在线演示

大多数的重要改变都会列在下面，不过你也可以在GitHub看到完整的差异。

控制器

`app/js/controllers.js`：

```
...
var phonecatControllers = angular.module('phonecatControllers',[]);

phonecatControllers.controller('PhoneDetailCtrl', ['$scope', '$routeParams', '$http',
  function($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;
    }
  }]);
```

在`PhoneDetailCtrl`控制器中，我们创建了一个`mainImageUrl`模型属性，并且设置默认值为第一个手机图片的URL。

我们也创建可一个`setImage`的行为处理函数，来改变`mainImageUrl`。

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 ngSrc 指令通过 mainImageUrl 绑定到大图片的URL。

我们也为缩略图注册了 ngClick 处理，当用户点击任何缩略图的时候进行处理，这个处理会使用 setImage 行为处理函数依据点击的缩略图来改变 mainImageUrl 的值，从而改变大图显示内容。

测试

为了检测新特性，我们需要写2个端到端测试。其中一个验证手机详细说明页面开始时，主要图（大图）是否是对应显示的第1个手机图片（默认值）。第二个测试检查点击一些缩略图是否会正确进行图片切换。
test/e2e/secnarios.js：

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

...

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


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

     element(by.css('.phone-thumbs li:nth-child(1) img')).click();
     expect(element(by.css('img.phone')).getAttribute('src')).toMatch(/img\/phones\/nexus-s.0.jpg/);
   });
 });
```

我们现在可以运行 npm run protractor 来完成测试了。

你也可以重构单元测试，因为这一步添加了 mainImageUrl 模型到 PhoneDetailCtrl 控制器中。下面，我们创建一个叫 xyzPhoneData 的函数，它会返回 images 属性的json数据来完成测试。
test/unit/controllersSpec.js：

```
...
  beforeEach(module('phonecatApp'));

...

 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).toBeUndefined();
      $httpBackend.flush();

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

现在单元测试能通过了。

尝试

让我们给 `PhoneDetailCtrl` 添加新的方法:

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

并把

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

添加到`phone-detail.html`模板中。

小结

我们的手机图片已经可以在指定位置进行切换了，我们准备进入步骤11来学习一个更好的获取数据的方

法。

步骤11——REST与定制服务

在这一步，我们将改变数据获取方法：

- 我们定义定制服务来代表一个RESTful客户端。使用这样的客户端，我们能够用更简单的方法向服务器请求数据，而不需要处理低层次的`$http` API，HTTP方法以及URL。

工作区切换到步骤11

直接用浏览器访问步骤11在线演示

大多数的重要改变都会列在下面，不过你也可以在GitHub看到完整的差异。

依赖

RESTful功能是由Angular中的`ngResource`模块提供，这个模块不属于Angular框架核心模块，所以我们必须要单独的安装和引入。

我们可以使用Bower来安装客户端依赖，这一步将更新`bower.json`配置以安装:

```
{
 "name": "angular-seed",
 "description": "A starter project for AngularJS",
 "version": "0.0.0",
 "homepage": "https://github.com/angular/angular-seed",
 "license": "MIT",
 "private": true,
 "dependencies": {
  "angular": "1.2.x",
  "angular-mocks": "~1.2.x",
  "bootstrap": "~3.1.1",
  "angular-route": "~1.2.x",
  "angular-resource": "~1.2.x"
 }
}
```

新的依赖描述`"angular-resource": "~1.2.x"`告诉bower要安装的angular-resource组件兼容版本是1.2.x。我们必须命令bower下载并安装这个依赖，这里我们运行下:

```
npm install
```

> 如果你是利用bower全局安装，你可能要单独运行`bower install`，但在这个项目中只需要运行`npm install`即可调用bower

模板

我们在`app/js/services.js`中定制服务，所以需要在布局模板中引入这个文件。额外，我们也需要引入`angular-resource.js`文件，它提供了`ngResource`模块： `app/index.html`:

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

服务

我们创建自己的服务提供从服务器访问手机数据： `app/js/services.js`

```
var phonecatServices = angular.module('phonecatServices', ['ngResource']);

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

我们使用模块API来注册了一个定制服务（作为工厂函数）。我们用 `Phone` 来代表这个服务(调用工厂函数)。工厂函数类似于一个控制器的构造函数,通过函数参数可以声明依赖注入。这个 `Phone` 服务描述了对 `$resource` 服务功能的依赖。

这个 `$resource` 服务功能只需要很少几行代码就简单的创建一个RESTful客户端。这个客户端可以被用于我们的程序以替代低层次的 `$http` 服务。 `app/js/app.js`

```
...
angular.module('phonecatApp', ['ngRoute', 'phonecatControllers','phonecatFilters', 'phonecatServices']).
...
```

我们也需要把 `phonecatServices` 模块添加进 `phonecatApp` 的模块依赖数组中。

控制器

通过使用 `Phone` 服务代理替换低层次的 `$http` 服务，我们可以让我们的子控制器 (`PhoneListCtrl` 和 `PhoneDetailCtrl`)进一步简化。Angular的 `$resource` 服务利用RESTful资源模型也提供了比 `$http` 更好的资源交互（展示），让我们更容易理解控制器中的代码如何工作。

`app/js/controllers.js` :

```
var phonecatControllers = angular.module('phonecatControllers', []);

...

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

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

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

注意现在 PhoneListCtrl 中我们把

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

替换成

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

这是一个简单的声明。我们可以查询到所有手机。

注意在上面的代码中一个重要的事情发生了，就是我们在调用 Phone 服务的方法时，没有定义任何的回调函数。虽然这像可以获得同步的返回果，但其实并不是。我们是在"未来"获得一个同步返回结果——一个由XHR响应返回数据填充的对象。因为Angular的数据绑定，我们可以使用这个"未来"结果，并绑定到我们的模板中。到时，当数据获取到后，视图会自动更新。

当然有时这样的"未来"对象以及相关数据绑定并不能满足我们所有的工作要求，所以在具体实现中，我们有时也需要添加回调来处理服务响应。例如 PhoneDetailCtrl 控制器中就包括了设置 mainImageUrl 这样的回调函数。

测试

因为我们现在使用了 ngResource 模块，所以我们要更新Karma配置来完成测试。 test/karma.conf.js :

```
  files : [
    'app/bower_components/angular/angular.js',
    'app/bower_components/angular-route/angular-route.js',
    'app/bower_components/angular-resource/angular-resource.js',
    'app/bower_components/angular-mocks/angular-mocks.js',
    'app/js/**/*.js',
    'test/unit/**/*.js'
  ],
```

我们编辑单元测试来验证新服务是否能够像预期一样正确发出HTTP请求和处理响应。这些测试也检验我们的控制器能够与服务正常协作。

`$resource` 服务利用响应对象（作为参数）方法来更新和移除资源。如果我们使用标准的 `toEqual` 检测将因为不确定（恰当）的响应（比如包括一些额外方法定义等）而肯定失败，为了处理这样的问题，我们使用新定义一个Jasmine matcher式 `toEqualData` 完成检测。使用 `toEqualData` 比较两个对象，将之考虑对象属性而忽略方法。 `test/unit/controllersSpec.js` :

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

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

  beforeEach(module('phonecatApp'));
  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).toEqualData([]);
      $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());
    });
  });
});
```

我们现在可以看到类似如下的Karma输出了:

```
Chrome 22.0: Executed 4 of 4 SUCCESS (0.038 secs / 0.01 secs)
```

小结

我们准备在步骤12（最后一步）中为手机图片替换应用上动画效果。

步骤12——应用动画

在这个最终步骤中，我们通过在以前代码基础上附加CSS和JavaScript实现动画效果来增强手机展示web程序。

- 用 ngAnimater 使得在程序中可以应用动画
- 公用的 ng 指令自动利用钩子(hook)来触发动画
- 当标准的DOM操作被赋予了动画时，对应的元素将会应用指定的动画效果，这些操作包括了利用 ngRepeat 插入或者移除DOM节点，以及利用 ngClass 对元素指定或者去除类型设定。

工作区切换到步骤12

直接用浏览器访问步骤12在线演示

大多数的重要改变都会列在下面，不过你也可以在GitHub看到完整的差异。

依赖

动画功能由Angular的 ngAnimate 模块提供，它不属于Angular框架核心。此外我们使用 JQuery 来提供额外的JavaScript动画支持。 我们使用 Bower 来安装客户端依赖。这一步骤中我们会更新 bower.json 配置文件来包含新的依赖关系:

```
{
 "name": "angular-seed",
 "description": "A starter project for AngularJS",
 "version": "0.0.0",
 "homepage": "https://github.com/angular/angular-seed",
 "license": "MIT",
 "private": true,
 "dependencies": {
  "angular": "1.2.x",
  "angular-mocks": "~1.2.x",
  "bootstrap": "~3.1.1",
  "angular-route": "~1.2.x",
  "angular-resource": "~1.2.x",
  "jquery": "1.10.2",
  "angular-animate": "~1.2.x"
 }
}
```

- "angular-animate": "~1.2.x" 告诉bower需要安装1.2.x版本的angular-animate兼容组件。
- "jquery": "1.10.2" 告诉bower需要1.10.2版本的JQuery，注意，JQuery不属于Angular库，它是一个标准的JQuery库。我们可以用bower安装第三方的库。

我们必须执行bower来下载安装，这里运行 npm install

> 如果你是在全局环境中使用bower安装，你可能会用直接到 bower install 指令，但是在这个项目中我们已经预配使用 npm 来启动 bower install ，所以你只需要: npm install

如何利用 ngAnimate 实现动画

如何利用AngularJS实现动画，请首先阅读 AngularJS Animation Guide (AngularJS动画指南)。

模板

需要在HTML模板代码中进行一些改动以链接诸如 angular-animate.js 等定义动画所需的资源文件。动画模块被称为 ngAnimate，其在 angular-animate 中定义，并包含让程序应用动画的必要代码。

这里，我们在index文件中进行必要改动。 app/index.html

```
...
  <!-- for CSS Transitions and/or Keyframe Animations -->
  <link rel="stylesheet" href="css/animations.css">

  ...

  <!-- jQuery is used for JavaScript animations (include this before angular.js) -->
  <script src="bower_components/jquery/jquery.js"></script>

  ...

  <!-- required module to enable animation support in AngularJS -->
  <script src="bower_components/angular-animate/angular-animate.js"></script>

  <!-- for JavaScript Animations -->
  <script src="js/animations.js"></script>

...
```

> 重要提醒:在Angular 1.3中必须要使用jQuery 2.1以上版本，而jQuery1.x版本是不被正式支持的。一定要在所有AngularJS脚本之前加载jQuery，否则AngularJS可能不能正确检测到jQuery而造成动画不按预想工作。

可以创建动画中的CSS代码(animations.css)创建和JavaScript代码(animations.js)。不过在此之前我们创建一个新依赖ngAnimate的模块，就像前面的 ngResource 服务一样。

模块与动画

app/js/animations.js :

```
angular.module('phonecatAnimations', ['ngAnimate']);
  // ...
  // this module will later be used to define animations
  // .
```

然后把这个模块加入到我们的程序中... app/js/app.js :

```
// ...
angular.module('phonecatApp', [
  'ngRoute',

  'phonecatAnimations',
  'phonecatControllers',
  'phonecatFilters',
  'phonecatServices',
]);
// ...
```

现在这个手机展示已经准备好动画环境了，让我们创作一些动画。

在动态 ngRepeat 中使用CSS的过渡动画

让我们在 phone-list.html 中添加CSS过渡动画到 ngRepeat 指令处理过程中。首先，我们给重复的元素添加一个额外CSS类，这样我们就可以以此钩住（hook）CSS过渡动画。 app/partials/phone-list.html

```
<!--
  Let's change the repeater HTML to include a new CSS class
  which we will later use for animations:
-->
<ul class="phones">
  <li ng-repeat="phone in phones | filter:query | orderBy:orderProp"
      class="thumbnail phone-listing">
    <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>
```

注意我们是怎么添加 phone-listing CSS类的？这些就是我们获得动画效果所有必须的HTML代码。

现在为CSS过渡动画实现代码

```
.phone-listing.ng-enter,
.phone-listing.ng-leave,
.phone-listing.ng-move {
  -webkit-transition: 0.5s linear all;
  -moz-transition: 0.5s linear all;
  -o-transition: 0.5s linear all;
  transition: 0.5s linear all;
}

.phone-listing.ng-enter,
.phone-listing.ng-move {
  opacity: 0;
  height: 0;
  overflow: hidden;
}

.phone-listing.ng-move.ng-move-active,
.phone-listing.ng-enter.ng-enter-active {
  opacity: 1;
  height: 120px;
}

.phone-listing.ng-leave {
  opacity: 1;
  overflow: hidden;
}

.phone-listing.ng-leave.ng-leave-active {
  opacity: 0;
  height: 0;
  padding-top: 0;
  padding-bottom: 0;
}
```

正如所见， `phone-listing` CSS类通过向对象添加或者去除（CSS类）而使得动画钩子触发。

- `ng-enter` 类被赋予一个新加入并渲染展示在页面中的手机元素。
- `ng-move` 类赋予在列表中移动的手机元素
- `ng-leave` 类赋予从列表中去除的元素

手机列表根据 `ng-repeat` 属性添加或者删除元素。例如如果转换器输出数据改变， 导致在列表中动态的添加或者放置元素。

这点尤为需要注意，在动画工作时，有两套CSS类可被加入元素：

1. 一套是"starting"类，这代表了动画的开始形式
2. 一套"active"类，代表了动画结束的形式

这个命名为 `starting` 的类会触发一些有 `ng-` 前缀的行为(例如 `enter` , `move` 或者 `leve`)，像 `enter` 就会触发 `ng-enter` 。

而 `active` 的类像 `starting` 类一样也会触发一些行为，只是这些行为是有 `-active` 后缀。这两套CSS类的名称可以在开发中指定动画的实现，是开始还是结束。

在我们的例子中，在列表中添加或者移动时，元素会从高度0扩展到高度120像素，在去除时逐渐消隐掉（缩小），同时还有一个淡入淡出效果。所有这些操作都是通过CSS过渡声明在前面的例子代码中。

尽管许多现代浏览器都能很好的支持CSS过渡和CSS动画，但IE9及以前版本是不支持的。如果你想对老的浏览器提供兼容的动画效果，可以考虑使用基于JavaScript的动画，这将在后面描述。

通过CSS提供 ngView 关键帧动画

接着，让我们为 ngView 中的内容切换提供动画。

首先，还是要添加一个新的CSS类到我们的HTML中，就像上面示例一样。这次，不是插入到 ng-repeat 所在的元素中，而是加入到应用 ng-view 命令所在的元素。为此，我们需要对HTML进行一些小改动，所以我们可以在视图变化时有更多的控制动画。 app/index.html

```
<div class="view-container">
  <div ng-view class="view-frame"></div>
</div>
```

通过这点改变，嵌入在 view-container CSS类元素下的 ng-view 指令所在元素有了 view-frame CSS类属性。而 view-container CSS类元素下的各个子元素有了 position: relative 样式设定，使得 ng-view 是相对于父级元素进行定位实现动画转换。

在这里，我们添加一些CSS到 animations.css 中实现转换动画: app/css/animations.css

```
.view-container {
  position: relative;
}

.view-frame.ng-enter, .view-frame.ng-leave {
  background: white;
  position: absolute;
  top: 0;
  left: 0;
  right: 0;
}

.view-frame.ng-enter {
  -webkit-animation: 0.5s fade-in;
  -moz-animation: 0.5s fade-in;
  -o-animation: 0.5s fade-in;
  animation: 0.5s fade-in;
  z-index: 100;
}

.view-frame.ng-leave {
  -webkit-animation: 0.5s fade-out;
  -moz-animation: 0.5s fade-out;
  -o-animation: 0.5s fade-out;
  animation: 0.5s fade-out;
  z-index:99;
}

@keyframes fade-in {
  from { opacity: 0; }
  to { opacity: 1; }
}
@-moz-keyframes fade-in {
  from { opacity: 0; }
  to { opacity: 1; }
}
@-webkit-keyframes fade-in {
  from { opacity: 0; }
  to { opacity: 1; }
}

@keyframes fade-out {
  from { opacity: 1; }
  to { opacity: 0; }
}
@-moz-keyframes fade-out {
  from { opacity: 1; }
  to { opacity: 0; }
}
@-webkit-keyframes fade-out {
  from { opacity: 1; }
  to { opacity: 0; }
}

/* don't forget about the vendor-prefixes! */
```

不是太疯狂！只是一个简单的划入划出效果就出现在页面中。这里的唯一不寻常的是,我们使用绝对定位来前一个页面(其有 ng-leave 类型设定),使其在下一个页面位置(通过 ng-enter 确认)的顶部。前一个页面将被移除，所以会淡出，新的页面将淡入并移动到原来的顶部。

在页面渲染时，一旦离开动画完成（动画对应元素在显示中被移除），或者一旦进入动画完成(被赋予了 ng-enter 和 ng-enter-active CSS 类的元素)，都会移除附加的预订CSS类（即动画相关的CSS类，例如 ng-enter 、 ng-enter-active 和 ng-leve 等）。这显得十分自然流畅,页面处理流程没有任何跳来跳去。

这些CSS类利用 ng-repeat 赋予所有需要的元素（开始和结束类）。每次页面加载 ng-view 都会创建一个它的副本，下载模板并且添加内容。这将确保所有的内容在一个单页HTML的元素中来允许简单的动画控制。

要了解更多CSS动画，请看WEB平台文档。

利用JavaScript实现 ngClass 动画

我们将在我们的程序中添加其他动画，转换到 phone-detail.html 页面，我们已经有了一个不错的缩略图切换效果：点击缩略图，将显示对应的手机不同角度图片，但我们能否为它添加动画呢？

让我们先考虑一下这一过程：当你点击缩略图,会改变显示不同角度图像（状态）来反映新点选的缩略图。在HTML中要改变描述状态最好是使用类。所以像前面一样，我们可以通过指定CSS类来描述一个动画，这次每当CSS类本身发生变化时会显示动画。

每当一个新手机缩略图被选中， .active CSS类就会添加到匹配的图片上，并播放动画。

首先，我们开始调整 phone-detail.html 页面中的HTML代码:

app/partials/phone-detail.html

```
<!-- We're only changing the top of the file -->
<div class="phone-images">
  <img ng-src="{{img}}"
       class="phone"
       ng-repeat="img in phone.images"
       ng-class="{active:mainImageUrl==img}">
</div>

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

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

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

就像缩略图，我们使用一个转换器来显示所有的概要文件列表图片，但是我们没有动画，也没有重复关联动画。反而，我们把视线投向了 ng-class 指令，因为无论何时，只有 active 类被激活，它就被加入到元素，并作为可视元素渲染显示出来。另外，其他不同角度的图片是隐藏的。在这个例子中，同一时间总有一个元素（图片）是 active 而被显示在屏幕上的。

当 active 类被加入到元素，在 active-add 和 active-add-active 类被添加前AngularJS可以触发一个动画。当元素移除时， active-remove 和 active-remove-active 类被赋予到元素，这也可以触发另一个动画。

为了确保手机图片中页面第一次加载时正确显示，我们也要处理一下页面CSS样式: app/css/app.css :

```
.phone-images {
  background-color: white;
  width: 450px;
  height: 450px;
  overflow: hidden;
  position: relative;
  float: left;
}

...

img.phone {
  float: left;
  margin-right: 3em;
  margin-bottom: 2em;
  background-color: white;
  padding: 2em;
  height: 400px;
  width: 400px;
  display: none;
}

img.phone:first-child {
  display: block;
  }
```

你可能想我们是不是要创建另外的CSS动画？事实上，我们不这样做，这里我们将借此机会学习如何利用 `animation()` 方法创建基于JavaScript的动画： `app/js/animations.js`

```
var phonecatAnimations = angular.module('phonecatAnimations', ['ngAnimate']);

phonecatAnimations.animation('.phone', function() {

  var animateUp = function(element, className, done) {
    if(className != 'active') {
      return;
    }
    element.css({
      position: 'absolute',
      top: 500,
      left: 0,
      display: 'block'
    });

    jQuery(element).animate({
      top: 0
    }, done);

    return function(cancel) {
      if(cancel) {
        element.stop();
      }
    };
  }

  var animateDown = function(element, className, done) {
    if(className != 'active') {
      return;
    }
    element.css({
      position: 'absolute',
      left: 0,
      top: 0
    });

    jQuery(element).animate({
      top: -500
    }, done);

    return function(cancel) {
      if(cancel) {
        element.stop();
      }
    };
  }

  return {
    addClass: animateUp,
    removeClass: animateDown
  };
});
```

注意，这里我们使用了jQuery来实现动画。在AngularJ中实现动画时jQuery并不是必须的，但是直接用JavaScript实现动画其实已经超出了本教程的范围。为了更多了解 jQuery.animat 请查看jQuery文档。

我们注册了 addClass 和 removeClass 回调函数，其会在 .phone 内部的任何元素添加或是移除类属性时调用。当 .active 类型被添加到元素(通过 ng-class 指令)，则JavaScript的 addClass 回调被调用，而且所对应的 element 作为参数传入。最后一个参数 done 是一个回调函数。事实上 done 是为了通知Angular知道

JavaScript动画已经完成，从而调用以完成后续工作。

`removeClass` 回调也基于同样的机制工作，不过是插入到一个类从元素中移除时。

通过JavaScript的回调，我们创建了对DOM的手动操作，在代码内，这些就是 `element.css()` 和 `element.animate()` 被执行。回调定位了下一个元素在500像素下，并移动前后两张图片，并动态同步移动以实现动画。这就是一个传送带动画，在 `animate` 功能完成后，调用 `done`。

注意: `addClass` 和 `removeClass` 每次返回一个函数，这是一个可选项，以实现函数调用链（可以让一个动画结束后进入下一个动画——或者功能），一个布尔值被传递，让开发者知道动画是否被调用。这通常被用于在动画结束后执行一些清理工作。

小结

你已经完成了它！我们在一个相对短的时间内就创建了一个web程序。最后（下一页），我们将指出如何进一步学习。

结束

我们的程序现在已经完成了，你可以任意利用，`git checkout`命令跳到某个步骤继续体验代码。

为了更多体验了解本教程例子信息，可进一步观看开发指南。

当你准备开始开发一个使用Angular的程序时，我们建议你在`angular-seed`项目的基础上扩展开发。

我们希望这个教程对你有用，让你学到关于Angular足够多的内容（比你预期更好）。我们特别希望你能开发出你自己的Angular程序，并以你感兴趣的合适方式为Angular发展提供灵感。

如果你有问题或者仅仅是想说声"hi"，都欢迎你在(https://groups.google.com/forum/#!forum/angular)发消息。