**简介**

  AngularJS是为了克服HTML在构建应用上的不足而设计的。HTML是一门很好的为静态文本展示设计的声明式语言，但要构建WEB应用的话它就显得乏力了。所以我做了一些工作（你也可以觉得是小花招）来让浏览器做我想要的事。

  通常，我们是通过以下技术来解决静态网页技术在构建动态应用上的不足：

  类库 - 类库是一些函数的集合，它能帮助你写WEB应用。起主导作用的是你的代码，由你来决定何时使用类库。类库有：jQuery等

  框架 - 框架是一种特殊的、已经实现了的WEB应用，你只需要对它填充具体的业务逻辑。这里框架是起主导作用的，由它来根据具体的应用逻辑来调用你的代码。框架有：knockout、sproutcore等。

  AngularJS使用了不同的方法，它尝试去补足HTML本身在构建应用方面的缺陷。AngularJS通过使用我们称为标识符(directives)的结构，让浏览器能够识别新的语法。例如：

  使用双大括号{{}}语法进行数据绑定；

  使用DOM控制结构来实现迭代或者隐藏DOM片段；

  支持表单和表单的验证；

  能将逻辑代码关联到相关的DOM元素上；

  能将HTML分组成可重用的组件。

**端对端的解决方案**

  AngularJS试图成为WEB应用中的一种端对端的解决方案。这意味着它不只是你的WEB应用中的一个小 部分，而是一个完整的端对端的解决方案。这会让AngularJS在构建一个CRUD（增加Create、查询Retrieve、更新Update、删除 Delete）的应用时显得很“固执”（原文为 opinionated,意指没有太多的其他方式）。但是，尽管它很“固执”，它仍然能确保它的“固执”只是在你构建应用的起点，并且你仍能灵活变动。 AngularJS的一些出众之处如下：

  构建一个CRUD应用可能用到的全部内容包括：数据绑定、基本模板标识符、表单验证、路由、深度链接、组件重用、依赖注入。

  测试方面包括：单元测试、端对端测试、模拟和自动化测试框架。

  具有目录布局和测试脚本的种子应用作为起点。

**AngularJS的可爱之处**

  AngularJS通过为开发者呈现一个更高层次的抽象来简化应用的开发。如同其他的抽象技术一样，这也会损 失一部分灵活性。换句话说，并不是所有的应用都适合用AngularJS来做。AngularJS主要考虑的是构建CRUD应用。幸运的是，至少90%的 WEB应用都是CRUD应用。但是要了解什么适合用AngularJS构建，就得了解什么不适合用AngularJS构建。

  如游戏，图形界面编辑器，这种DOM操作很频繁也很复杂的应用，和CRUD应用就有很大的不同，它们不适合用AngularJS来构建。像这种情况用一些更轻量、简单的技术如jQuery可能会更好。

**实例**

**AngularJS下载地址**

<http://angularjs.org>

**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。

**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：

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。

1. 运行以下命令：

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](http://localhost:8000/app/index.html" \t "_blank)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<div>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](http://angularjs.cn/A00s" \t "_blank)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**

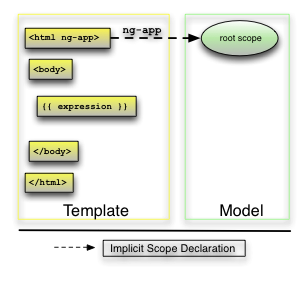
通过ngApp指令来自动引导AngularJS应用是一种简洁的方式，适合大多数情况。在高级开发中，例如使用脚本装载应用，您也可以使用[bootstrap](http://angularjs.cn/A00o" \t "_blank)手动引导AngularJS应用。

**AngularJS应用引导过程有3个重要点：**

1. [注入器(injector)](http://docs.angularjs.org/api/AUTO.%24injector)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](https://github.com/angular/angular-seed)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](http://twitter.github.com/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>

**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>

**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](http://en.wikipedia.org/wiki/Model%E2%80%93View%E2%80%93Controller" \t "_blank)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](https://github.com/angular/angular-phonecat/compare/step-1...step-2)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>

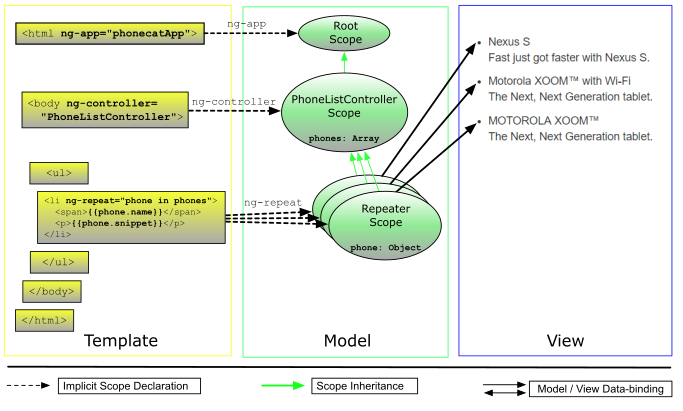
</ul>

</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](http://docs.angularjs.org/api/ng.directive%3angRepeat" \t "_blank)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](http://docs.angularjs.org/api/ng.directive%3angController" \t "_blank)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](http://docs.angularjs.org/api/ng.%24rootScope.Scope" \t "_blank)。

**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](http://pivotal.github.com/jasmine/" \t "_blank)或者[Jasmine Wiki](https://github.com/pivotal/jasmine/wiki)上获得相关知识。

基于AngularJS的项目被预先配置为使用[JsTestDriver](http://code.google.com/p/js-test-driver/" \t "_blank)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。

1. 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

**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](https://github.com/angular/angular-phonecat/compare/step-2...step-3)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>

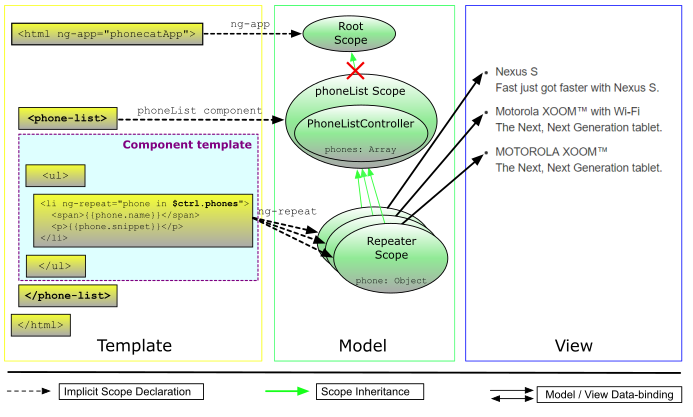
</div>

我们现在添加了一个<input>标签，并且使用AngularJS的[$filter](http://docs.angularjs.org/api/ng.filter%3afilter" \t "_blank)函数来处理[ngRepeat](http://docs.angularjs.org/api/ng.directive%3angRepeat" \t "_blank)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](http://docs.angularjs.org/api/ng.filter%3afilter" \t "_blank)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](http://code.angularjs.org/1.1.0/docs/guide/dev_guide.e2e-testing" \t "_blank)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](http://localhost:8000/test/e2e/runner.html" \t "_blank)
* 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](http://angular.github.com/angular-phonecat/step-3/test/e2e/runner.html" \t "_blank)

这个测试验证了搜素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](http://docs.angularjs.org/api/ng.directive%3angBind" \t "_blank)或者[ngBindTemplate](http://docs.angularjs.org/api/ng.directive%3angBindTemplate" \t "_blank)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。

**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](https://github.com/angular/angular-phonecat/compare/step-3...step-4)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](http://docs.angularjs.org/api/ng.filter%3aorderBy" \t "_blank)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。