

目 录

致谢

简介

环境的准备

项目的创建和配置

AngularJS的第一步

学习和使用AngularJS

- 基本表达式

- AngularJS初始化 ng-app

- 控制器 ng-controller

- 数据绑定 data-binding

- 条件判断 ng-if / ng-show / ng-hide

- 重复语句 ng-repeat

- 过滤器 filter

- 样式选择器 ng-class/ng-style

- 下拉列表选项 ng-options

- 引入ng-include和模板ng-template

- 本章总结

深入学习AngularJS - Directive

- 制作一个自定义的Directive

- Directive的命名和使用规则

- 让Directive支持传入数据

- 使用templateUrl获取模板

- 让Directive动起来link()

- 把Directive变为一个容器transclude

- Directive之间互相通讯

- 本章总结

致谢

当前文档《学习AngularJS 1.x》由 进击的皇虫 使用 书栈(BookStack.CN) 进行构建，生成于 2018-04-18。

书栈(BookStack.CN) 仅提供文档编写、整理、归类等功能，以及对文档内容的生成和导出工具。

文档内容由网友们编写和整理，书栈(BookStack.CN) 难以确认文档内容知识点是否错漏。如果您在阅读文档获取知识的时候，发现文档内容有不恰当的地方，请向我们反馈，让我们共同携手，将知识准确、高效且有效地传递给每一个人。

同时，如果您在日常生活、工作和学习中遇到有价值有营养的知识文档，欢迎分享到 书栈(BookStack.CN) ，为知识的传承献上您的一份力量！

如果当前文档生成时间太久，请到 书栈(BookStack.CN) 获取最新的文档，以跟上知识更新换代的步伐。

文档地

址：http://www.bookstack.cn/books/learning_angular

书栈官网：<http://www.bookstack.cn>

书栈开源：<https://github.com/TruthHun>

分享，让知识传承更久远！ 感谢知识的创造者，感谢知识的分享者，也感谢每一位阅读到此处的读者，因为我们都将成为知识的传承者。

简介

- [学习AngularJS 1.x](#)
 - [Learning AngularJS 1.x](#)
 - [本书地址](#)
 - [本书的读者](#)
 - [我应用AngularJS的方法](#)
 - [这样做的好处](#)
 - [为什么选择AngularJS 1.x](#)
 - [我选择AngularJS的历程](#)
 - [为什么写这本书](#)
 - [另：为什么没有选择Angular 2](#)
 - [版权声明 LICENSE](#)
 - [来源\(书栈小编注\)](#)

学习AngularJS 1.x

Learning AngularJS 1.x

本书是我在学习和应用AngularJS 1.x 的过程中的资料梳理。希望能对大家学习AngularJS有一定帮助。

1. 如果您在阅读过程中，有任何疑问或者发现错误，可联系：
2. 作者： [Harry](mailto:harry@andtoo.net)<harry@andtoo.net>
3. 微信： hharry

本书地址

本书使用GitBook和GitHub托管。

GitBook地址: [hairui219/learning_angular](#)

GitHub地址: [hairui219/learning_angular](#)

本书的读者

本书会介绍如何应用AngularJS, 但是本书不会涉及到JavaScript语法以及HTML和CSS的布局模式。因此, 本书对读者有一定的前置技术要求:

1. 您需要知道和理解HTML和CSS布局的方法
2. 您需要知道JavaScript的基本语法
3. (推荐) 您知道一些Node.js的使用方法
4. (推荐) 您知道控制台命令如何使用

如果您需要对以上某方面内容入门, 我向您推荐[w3School网站](#)。您可以在这个网站上快速的了解相关的知识。

另外, 在JavaScript方面, 如果您有一定的PHP经验, 我向您推荐这本书 (如果您没有PHP经验, 这本书也可以阅读):

- JavaScript for PHP Developers (中文版) - Stoyan Stefanov 著 - 李强 译

这本书只有141页 (定价28元, 网站还有折扣), 阅读起来非常轻松, 如果您有编程经验, 一个下午就可掌握JavaScript的基本语法和用法 (同时也可能学会PHP的语法)。

京东特价优惠时我购买了几十本与JavaScript相关的书籍, 这本是我认为最靠谱的入门书。

我应用AngularJS的方法

在进行前端开发的工作之前, 我担任过几年的移动互联网产品经理, 设

计了几款应用（参与了一小部分的开发工作）。之后我改做应用和手机游戏的后端开发（主要使用PHP）。

在这几年的工作经验中，我形成了一套app的构建思路。因此，在制作网站前端时，我也希望采用类似的方法（我不喜欢直接在php代码中直接嵌入html模板的方案，那样做感觉上比较混乱，难以管理。

因此，我将AngularJS作为一个类似于app的载体，当网站代码在客户端载入完成后，再通过api请求获取数据。

这样做的好处

这样实现后，我认为主要有以下几个方面的好处：

1. 数据通过API(https)获取，过程并不向用户开放，起到了隐蔽后端服务器的效果
2. 可以对API的访问限制和安全性进行更完善的设计实现
3. 前端网站托管在阿里云的OSS上（以静态网站的方式部署），这样页面部分不再占用服务器的流量和空间
4. 便于之后的扩展，前端网站可以使用阿里云的CDN直接进行访问加速；后端在使用API模式通讯的情况下，本来就可以极大的提高负载能力（网络带宽优化），如果需要扩展，可以提高机器配置或者增加机器数量。

幸运的是，**AngularJS**推荐这么做。

为什么选择AngularJS 1.x

对于选择AngularJS，业(zhi)界(hu)其实有一个调侃的说法：

写Java的写不来JavaScript的用AngularJS

虽然这个说法比较武断，但是其中也体现出来一个明显的信息，如果你

之前有**Java**或其他后端语言的编程经验，**AngularJS**是让你快速上手**Web**前端开发的很好的选择。至少对我而言是如此。

我选择AngularJS的历程

以下内容部分读者可能会感到有些偏激，但是这是对我（一个拥有一些其他编程经验的前端入门者）的心路记录。

2015年初，在准备通过完全的Web方式实现一个B/S模式的企业服务网站之后，我开始进行技术方面的准备。在此之前，我只有Android客户端和PHP服务端的工作经验。另外，我对web的基本开发技能使用过一些，掌握了HTML+CSS的一些应用技巧。JavaScript的语法和基本特性方面也通过Node.js在工作中的应用熟悉了。

但是，在网站前端技术准备过程中，我发现我对DOM的操作一窍不通。即使是购买了若干本入门的书籍，通过阅读书籍，我知道了如何通过`document.getElementById('xxx')`或者jQuery的`$('#xxx')`来获取元素和填入代码之类的工作。但学会基本的语法之后，我仍然是对于具体如何操作DOM来实现具体的功能完全没有头绪。这是一种什么感觉呢？就像是在读大学时学习C/C++/Java语言，我们可以手工的去实现一些基本的算法或数据结构。但是学完了之后，我们可以直接使用C/C++或Java制作基于Windows的客户端程序了吗？一点也不。学习DOM入门给我的感觉与之一模一样。

因为当时并不知道还有JavaScript框架这种东西存在，我去网上搜索教程时也是一头雾水。因此，我想了一个笨办法，直接在一个国外的网站上购买了一套后台管理系统界面的模板。浏览了下源代码之后，我发现源码提供了两套版本，一套完全基于jQuery+BootStrap的版本，和一套基于AngularJS的版本。两个版本的界面布局一模一样，但是基于AngularJS的版本提供了一些如单页应用（在同一个界面直接刷新部分界面而不是整页刷新）的特性。

然后便是一系列的在网上搜索了解AngularJS的过程，看完它基本的Tutorial (PhoneCat) 之后，我马上被它的特性震惊了。因为，

1. AngularJS的整个结构体系非常符合我的思路
2. 双向绑定的特性实在是太和我胃口了，这让我完全不用再操心DOM

因为这两点特性，我义无反顾的加入了使用AngularJS的大军。

题外话：

国外网站购买的这种模板，都是由专业的前端人员开发，提供的功能都非常丰富完整。另外，他们对管理工具的使用也很正规（使用**bower**或其他工具来管理第三方库等），代码结构和注释也非常完善。

对我而言，这笔投资绝对物有所值（当时花费了我19美金）。

接下来我所做的事情，就是拿着这个模板的代码，修改html和增加controller/service，调整ui-router的配置，最终完成了网站的第一个版本。花的时间大约是2个月出头（前端+后端+部署调试，开发工作全部由我一人完成）。

从今天来看，我的做法完全不符合AngularJS的最佳实践。虽然我未在网站中使用jQuery等类库，但是我也同样没有使用AngularJS推荐的directive方式（完全没有使用，这也是我准备在更深入的学习AngularJS和重新构建网站的第二版的主要原因）。

但是，不管怎么说，我把东西做出来了，而且整个网站是可用的，从程序结构上也是可维护和修改的。

为什么写这本书

由于业务需要，我准备重新制作网站的v2版本。在这个版本中，会有很多新的功能需求（功能扩大非常多）。因此，完全重构整个网站是可以接受的。另外，我之前只是误打误撞的制作出来一个“可用”的网站，为

了使用更地道的方式制作这个新版网站，我现在正在重新学习AngularJS。

写这本书，一方面是为了记录学到的AngularJS的技术，便于我日后查询；另一方面也是为了给自己一点压力，让自己沉下心来掌握这门技术。

我在AngularJS方面也是一名菜鸟，虽然我会尽力保证信息的正确性，但也还请您在阅读的过程中批判的接收信息。如果有何问题，请通过邮件或微信联系我（联系地址在此页面上方）。

另：为什么没有选择Angular 2

其实在重构时，我优先的选择是Angular 2 & TypeScript。因为它引入了很多新的特性，比如应用了最新的ES标准，大量优化了Angular工作的效率等等。

但是，下面3个原因导致了我放弃了Angular 2作为新版本网站的技术选择。

1. 我自认为是一个比较追求新技术的人，但是当我测试Angular 2时，发现我电脑上安装的Chrome v44版本无法运行，当我下载更新到v47后才能正常运行（官网介绍v46以上才支持es6）。
2. 新版本Angular 2抛弃了controller/service，完全用directive，这点与我的技术构想不符。
3. 即使在我评估Angular 2时已经是beta版本，但是官网仍然推荐不要将其用于生产环境。

我的网站运行的是对公业务，且对合作方使用的浏览器有一定的影响能力，但是我没有信心让所有客户都安装上Chrome的最新版本。

与我技术构想不符则是对我本人来说更重要的一个原因，因为我无法认

同Angular 2的优化方向。同时，根据我的工作经验来看，这样革命性的优化通常是内部一群理念不合的人另起炉灶，结果可能喜忧参半，不一定会最终成功。

TypeScript的引入我不好评价，但是强类型定义和模型化并不太符合我的业务模式（api一次性获取界面需要的所有数据，这些数据已经在服务端进行好了类型定义和完整性处理，本地主要是实现对CRUD的调用）。

综上，我放弃了在新版本的网站中应用Angular 2技术，而选择继续使用AngularJS 1.x。

版权声明 LICENSE

1. 署名-非商业性使用 4.0 国际
2. 版权所有 (c) 2016 Harry<harry@andtoo.net>
3. 本作品采用知识共享 署名-非商业性使用 4.0 国际 许可协议进行许可。访问
4. <http://creativecommons.org/licenses/by-nc/4.0/> 查看该许可协议。
- 5.
6. Attribution-NonCommercial 4.0 International (CC BY-NC 4.0)
7. Copyright (c) 2016 Harry<harry@andtoo.net>
8. This work is licensed under the Creative Commons Attribution-NonCommercial 4.0
9. International License. To view a copy of this license, visit
10. <http://creativecommons.org/licenses/by-nc/4.0/>.

来源(书栈小编注)

https://github.com/hairui219/learning_angular

环境的准备

- 环境的准备
 - [WebStorm](#)
 - [Chrome](#)
 - [Node.js](#)
 - [其他工具](#)
 - [其他工具](#)

环境的准备

开发Web网站，使用合适的工具和环境会极大的提升开发的效率。本章将讲述如何准备用于AngularJS项目开发的环境。

本书将会使用如下工具和库：

- [WebStorm](#) - 前端开发
- [Chrome](#) - Google Chrome浏览器
- [AngularJS](#) - 前端JS框架
- [Angular Material](#) - 前端界面框架
- [bower](#) - 获取包括AngularJS在内的各种开发库

为了使用bower，您可能还需要安装如下工具：

- [cnpm](#) - 国内用户推荐，淘宝的npm加速服务
- [npm](#) - Node.js的包管理软件
- [Node.js](#) - 基于Chrome V8引擎的本地/服务端JS运行环境

1. 以上内容除去[WebStorm](#)外，都可以免费获得。[WebStorm](#)可以获取免费试用30天的版本。
2. 因此，您在学习本书的过程中，并不需要花费购买任何软件。

WebStorm

在过去的开发中，WebStorm一直是我对编辑器的首选。WebStorm是商业化的产品，如果长期使用，需要花钱购买（目前已经支持支付宝购买）。价格的话，如果你是一名专职的前端工作人员，可以自行购买或者要求公司购买，个人购买的费用是第一年\$59美金，我个人认为还是非常超值的（对比工作效率的提升而言）。另外，出于个人习惯，我使用的是英文版的WebStorm，目前网络上也存在有汉化包，如果希望使用汉化界面的朋友可以自行搜索尝试。

在学习本书的过程中，您可以使用WebStorm的30天尝试版本（直接下载安装即可）。30天的时间对于学习AngularJS和进行一些初级的开发尝试完全足够了。

本书写作时使用的WebStorm版本是11.0.3。

*WebStorm*自8.x版本起就可以正确的开发*AngularJS*网站（再往前的版本没有评估）。使用最新的11.0.3版本是在准备本书内容过程中，评估了*Angular 2*。而*WebStorm*是从11版本才开始部分支持*Angular 2*的*TypeScript*开发。

1. 另：使用哪种编辑器更多的是个人的偏好。如果您有其他喜欢的编辑器，可以自行选择。

Chrome

Chrome浏览器的开发者工具可以极大的方便开发时的调试工作，目前国内大量浏览器也是基于Chrome的开源内核开发，在通用性上也有保证。另外，Chrome上有支持WebStorm的插件“JetBrains IDE Support”。

由于国内网络原因，Chrome浏览器必须要连接至国外路由才能下载。另外推荐您点击以下链接进行下载，这样可以直接下载完整的安装包（直接在Google搜索Chrome的官网页面下载的是一个小型下载器，在

国内无法正确安装)。

Chrome完整版本下载链接

```
1. https://www.google.com/chrome/browser/desktop/index.html?standalone=1
```

本书写作时使用的Chrome版本是47.0.2526.106 m。

Chrome的版本对于应用AngularJS并不关键，本书特意安装目前的最新版本，是因为要同时评估Angular 2，而Angular 2必须要在非常新的浏览器上才能正确运行(v 46+)。

Node.js

Node.js在本书中主要的使用用途是运行bower工具，用以安装AngularJS极其相关的库文件。通过官网可以直接下载Node.js的安装包(Windows/OS X都有对应的安装包)。

本书使用的Node.js版本是5.4.1(目前最新的Stable稳定版本)。推荐安装较新的稳定版本。

其他工具

在安装好Node.js后，可以通过Node.js的npm命令安装最新版本的npm/cnpm/bower(如果已经存在，自动安装最新版本)。

使用命令如下：

Windows上打开cmd，OS X上打开Terminal。

```
1. //Windows
2. > npm install -g npm --registry=https://registry.npm.taobao.org
3. > npm install -g cnpm --registry=https://registry.npm.taobao.org
4. > npm install -g bower --registry=https://registry.npm.taobao.org
```

```

5.
6. //可选, 将npm默认设置从淘宝服务器上获取数据
7. > npm config set registry "https://registry.npm.taobao.org"
8.
9.
10.
11. //OS X (会需要您输入本机的密码)
12. $ sudo npm install -g npm --
    registry=https://registry.npm.taobao.org
13. $ sudo npm install -g cnpm --
    registry=https://registry.npm.taobao.org
14. $ sudo npm install -g bower --
    registry=https://registry.npm.taobao.org
15.
16. //可选, 将npm默认设置从淘宝服务器上获取数据
17. $ npm config set registry "https://registry.npm.taobao.org"

```

所有命令后面都跟上了一个—

`registry=https://registry.npm.taobao.org`标志, 这是使用了淘宝提供的npm镜像服务来安装所需的软件, 这样访问的速度会加快非常多。如果您优先运行了后面的可选的命令, 那么之前三个命令的此标志项都可以去除。

OS X的命令和Windows类似, 但是前面都加了一个sudo, 用于提权后把这些工具安装到Node.js的公共库中。

安装完成后, 请手动命令确认这几个工具的版本。(如果您安装时不成功, 请先确认Node.js工具的版本是否是最新的)

安装好后, 您的几个工具的版本应该如下(或者更高):

```

1. $ npm -v
2. 3.5.3
3.
4. $ cnpm -v
5. 3.4.0

```

```
6.  
7. $ bower -v  
8. 1.7.2
```

其他工具

其他的工具和库，我们将不再需要从网站上下载安装，直接通过bower在项目内进行下载使用。

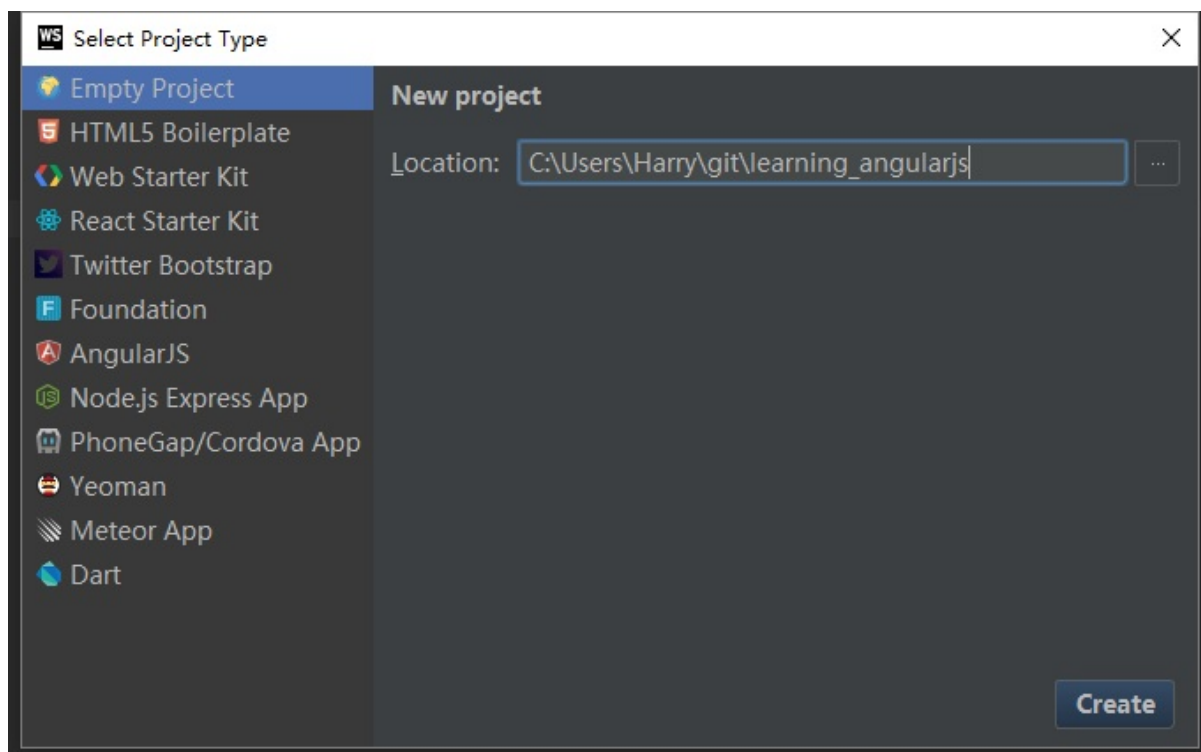
项目的创建和配置

- 项目的创建和配置
 - 创建基本的文件结构
 - `public` 目录设置的意义
 - 配置并初始化bower
 - 打开命令行工具
 - 初始化bower
 - `.bowerrc` 配置文件
 - 配置完成

项目的创建和配置

使用WebStorm可以直接创建AngularJS项目，且会自动帮助你配置好项目并自动下载AngularJS等库。但是我们这里将创建一个新的空项目（Empty Project），然后一步一步的将各个需要的内容填充进来，这样我们会对使用AngularJS开发有一个更好的了解。

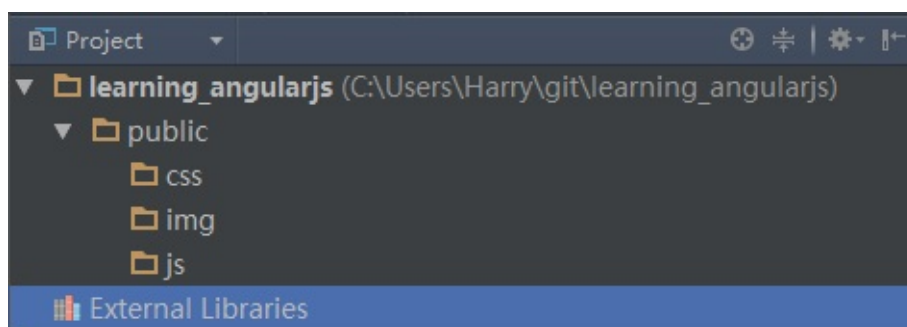
在WebStorm中，选择 `File > New Project`，然后选择Empty Project，在右边的Location设置好项目的位置，然后点击界面右下角的Create即可。



下面我们将一步一步的完善整个项目的结构。

创建基本的文件结构

首先建立一个 `public` 目录，并在 `public` 目录中建立 `js`、`css`、`img` 三个目录。如下图所示：



大家对于 `js/css/img` 三个目录的设置应该比较熟悉，他们分别用于存放对应的文件，`img` 用于存放图片文件。

但是，为什么要把他们放置到 `public` 目录中呢？

`public` 目录设置的意义

从字面意思即可理解，`public` 目录是用来存放供外部用户访问的内容的根目录。非 `public` 目录下的内容，既是我们不期望用户通过网络链接直接可以访问到的内容。会有哪些内容呢？

- 产品的文档
- 一些项目配置文件（如 `bower` 的配置文件）
- 测试文档

这些文件我们是绝对不希望用户可以直接访问到的，通过设置一个 `public` 目录，并在部署时将网站的根目录直接指向到 `public` 目录，即可保证目录外的内容不被暴露到网络当中。

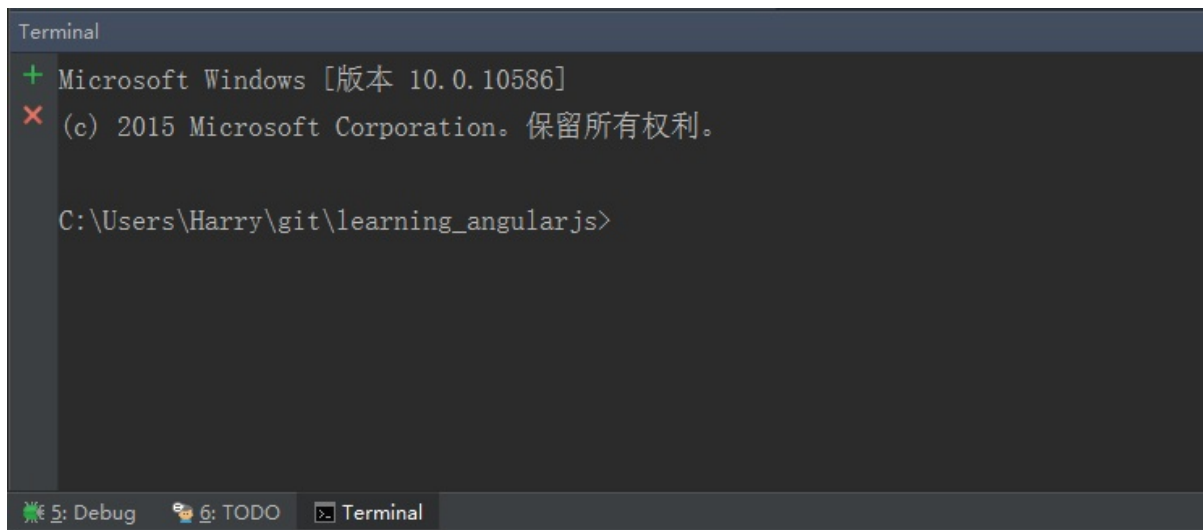
另外，这样设置的主要原因是为了使用 `git` 将整个项目都管理起来。通过 `git` 版本控制的方式来保证项目代码的完整性和安全性。具体的 `git` 的操作方法就不在本文中叙述了。

配置并初始化 `bower`

`bower` 的配置可以通过手动创建文件或者命令行的方法来进行。我推荐使用命令行的方式来进行创建，这样可以更好的理解配置生成的文件的内容。如果不想通过命令行创建，也可以跳过下面命令行创建的部分，直接在下方生成的文件解析的部分，将文档内容拷贝过去。

打开命令行工具

WebStorm 内置了命令行工具（调用系统的命令行功能），在左下角点击 `Terminal` 即可启用。

A screenshot of a Windows Terminal window. The title bar says "Terminal". The content shows a green plus icon followed by "Microsoft Windows [版本 10.0.10586]", a red X icon followed by "(c) 2015 Microsoft Corporation. 保留所有权利。", and the command prompt path "C:\Users\Harry\git\learning_angularjs>". The bottom taskbar shows icons for "5: Debug", "6: TODO", and "Terminal".

```
Terminal
+ Microsoft Windows [版本 10.0.10586]
X (c) 2015 Microsoft Corporation. 保留所有权利。

C:\Users\Harry\git\learning_angularjs>
```

初始化bower

在命令行下运行 `bower init`，你将会看到如下的若干选项，并会自动的在项目的根目录生成一个 `bower.json`。

注：以下的汉字部分都是额外加入的注释。

```
1. //运行命令
2. >bower init
3.
4. //第一次运行的时候会弹出，是否愿意提交匿名的统计信息。随意选择
5. ? May bower anonymously report usage statistics to improve the tool
   over time? Yes
6. //项目名称
7. ? name learning_angularjs
8. //项目说明
9. ? description
10. //主文件
11. ? main file
12. //项目的类型
13. ? what types of modules does this package expose?
14. //项目的关键字
15. ? keywords
16. //作者和联系方式
17. ? authors Harry <harry@andtoo.net>
```

```
18. //授权方式, 如果您期望这是一个私人项目, 可以输入No License
19. ? license MIT
20. //项目主页
21. ? homepage
22. //是否要把当前已经安装的模块设置为项目的依赖项?
23. ? set currently installed components as dependencies? Yes
24. //将常用的忽略文件项添加到列表?
25. ? add commonly ignored files to ignore list? Yes
26. //将项目设置为私有, 防止其被误发布到网络上?
27. ? would you like to mark this package as private which prevents it
    from being accidentally published to the registry? Yes
28.
29. //以下是生成的配置文件的预览
30. {
31.   name: 'learning_angularjs',
32.   authors: [
33.     'Harry <harry@andtoo.net>'
34.   ],
35.   description: '',
36.   main: '',
37.   moduleType: [
38.   ],
39.   license: 'MIT',
40.   homepage: '',
41.   private: true,
42.   ignore: [
43.     '**/*.*',
44.     'node_modules',
45.     'bower_components',
46.     'test',
47.     'tests'
48.   ]
49. }
50.
51. ? Looks good? Yes
```

命令运行完毕后, 会在项目的根目录生成一个 `bower.json` 文件, 里面的内容如下:

```

1. //bower.json
2. {
3.   "name": "learning_angularjs",
4.   "authors": [
5.     "Harry <harry@andtoo.net>"
6.   ],
7.   "description": "",
8.   "main": "",
9.   "moduleType": [
10.  ],
11.   "license": "MIT",
12.   "homepage": "",
13.   "private": true,
14.   "ignore": [
15.     "**/*.\"",
16.     "node_modules",
17.     "bower_components",
18.     "test",
19.     "tests"
20.   ]
21. }

```

如果您不希望bower配置文件这么复杂，那么可以手工最简化的创建 `bower.json`。

```

1. //最简化bower.json
2. {
3.   "name": "learning_angularjs"
4. }

```

只需要一个 `name` 字段即可让bower好好工作了。

`.bowerrc` 配置文件

由于我们建立了 `public` 目录，并且项目的根目录与网站根目录不同，

因此，我们需要新建一个额外的 `.bowerrc` 文件，告诉 `bower` 将组件库下载到特定的目录。

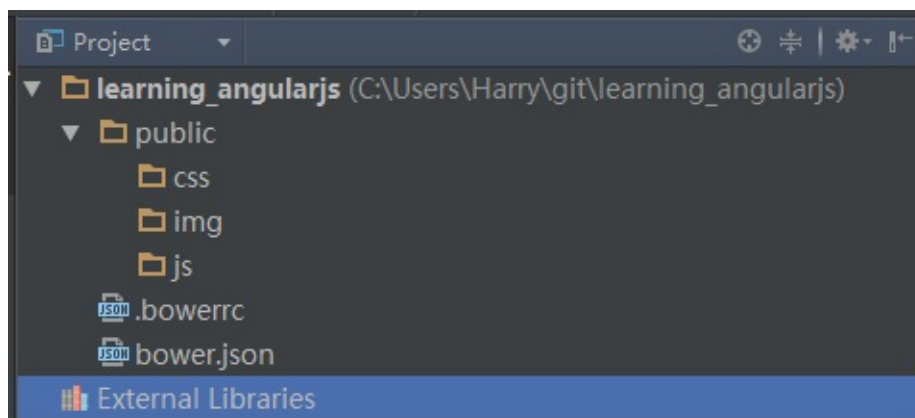
在项目根目录创建 `.bowerrc` 文件，并在其中加入如下内容：

```
1. ///.bowerrc  
2. {  
3.   "directory": "public/components"  
4. }
```

以上配置文件会告诉 `bower` 将文档下载到 `./public/components` 目录中。

配置完成

至此，我们对 `bower` 的配置已经全部结束！现在项目看起来应该长这个样子：



AngularJS的第一步

- AngularJS的第一步
 - 在项目中安装AngularJS的基本库
 - 建立 `index.html` 文件
 - 引入AngularJS的库文件
 - 第一个程序Hello World
 - 运行Hello World

AngularJS的第一步

在前面的章节，我们已经创建好了项目，并配置好了bower工具。本章开始，我们将进入正式的学习使用AngularJS的过程。首先，我们将从安装AngularJS开始。

在项目中安装AngularJS的基本库

AngularJS官网提供了通过Bower安装的命令行，我们需要做的，就是在WebStorm的命令行工具中，运行如下命令：

```
1. $ bower install angular#1.5.0-rc.0 --save
```

结果如下所示：

```
1. >bower install angular#1.5.0-rc.0 --save
2.
3. bower angular#1.5.0-rc.0          cached
   git://github.com/angular/bower-angular.git#1.5.0-rc.0
4. bower angular#1.5.0-rc.0          validate 1.5.0-rc.0 against
   git://github.com/angular/bower-angular.git#1.5.0-rc.0
5. bower angular#1.5.0-rc.0          cached
   git://github.com/angular/bower-angular.git#1.5.0-rc.0
```

```

6. bower angular#1.5.0-rc.0      validate 1.5.0-rc.0 against
   git://github.com/angular/bower-angular.git#1.5.0-rc.0
7. bower angular#1.5.0-rc.0      install angular#1.5.0-rc.0
8.
9. angular#1.5.0-rc.0 public\components\angular

```

命令解释

这行命令告诉 `bower` 在这个项目中安装 `angular#1.5.0-rc.0`，也即是 AngularJS 的 1.5.0-rc.0 版本（当前的最新版本）。

`--save` 标志 这个额外的标志，是告诉 `bower` 把我们的安装记录放置入 `bower.json` 文件。这样，我们以后可以直接通过 `bower` 对此项目使用的 AngularJS 或其他库进行更新。

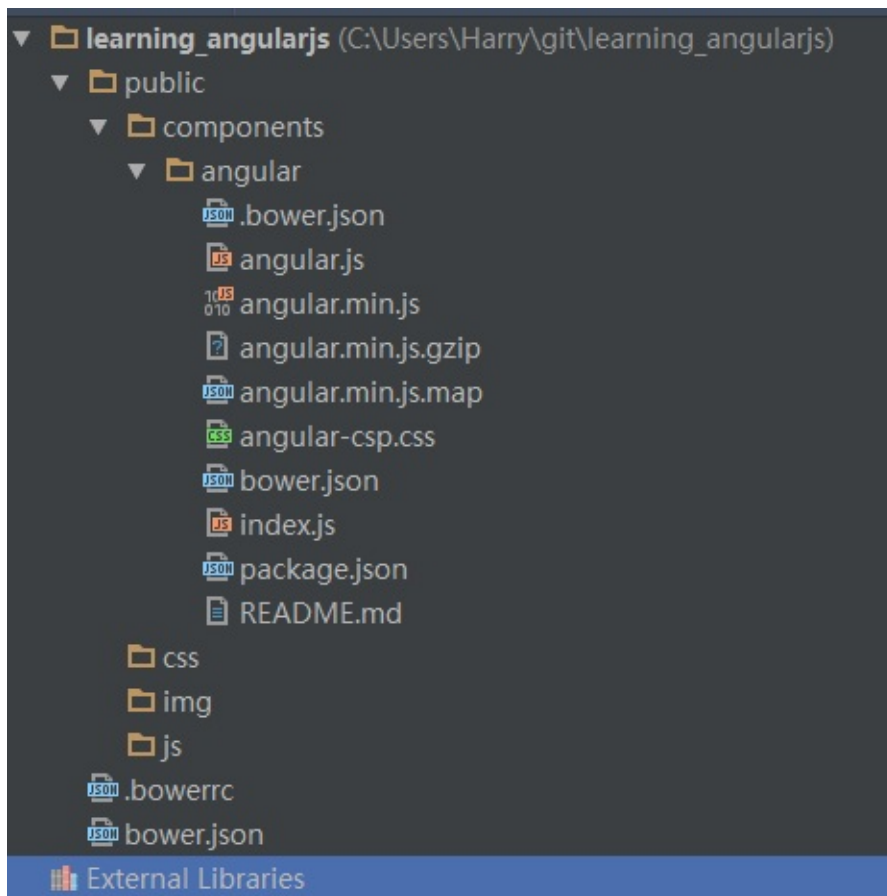
此时，当我们打开 `bower.json`，我们会发现文档中的内容变多了，如下所示：

```

1. //bower.json
2. {
3.   "name": "learning_angularjs",
4.   "dependencies": {
5.     "angular": "1.5.0-rc.0"
6.   }
7. }

```

同时，项目中会多出 `./public/components/angular` 目录，所有的 AngularJS 的文件都在这个目录中存放。



建立 `index.html` 文件

在 `./public` 目录下建立 `index.html` 文件（右键 `public` 目录，`New > HTML File`，然后输入 `index`，点击 `OK`）。WebStorm 会自动帮助我们加入基本的HTML内容。

```
1. <!DOCTYPE html>
2. <html lang="zh">
3. <head>
4.     <meta charset="UTF-8">
5.     <title>学习AngularJS 1.x</title>
6. </head>
7. <body>
8.
9. </body>
10. </html>
```

我将 `lang` 从 `en` 改为了 `zh`，表明此网站是简体中文的。同时调整了 `title`。HTML的基础并不属于本书的范围，因此不在此细述。

引入AngularJS的库文件

引入AngularJS库文件很简单，一行HTML语言加入HTML的head部分即可：

```
1. <script type="text/JavaScript" src="components/angular/angular.js">
    </script>
```

注：这里引入的`angular.js`是完整的版本（1M大小），如果在运行环境中，您应该将`angular.js`替换为`angular.min.js`（148KB）。

小贴士：将`js`文件放在`head`部分和`body`部分有何区别？>放在`head`部分的`JavaScript`文件，会在`body`渲染完毕后才开始执行。从`AngularJS`工作的特性来看，推荐所有的`JavaScript`文件都放在`body`部分引入。

第一个程序Hello World

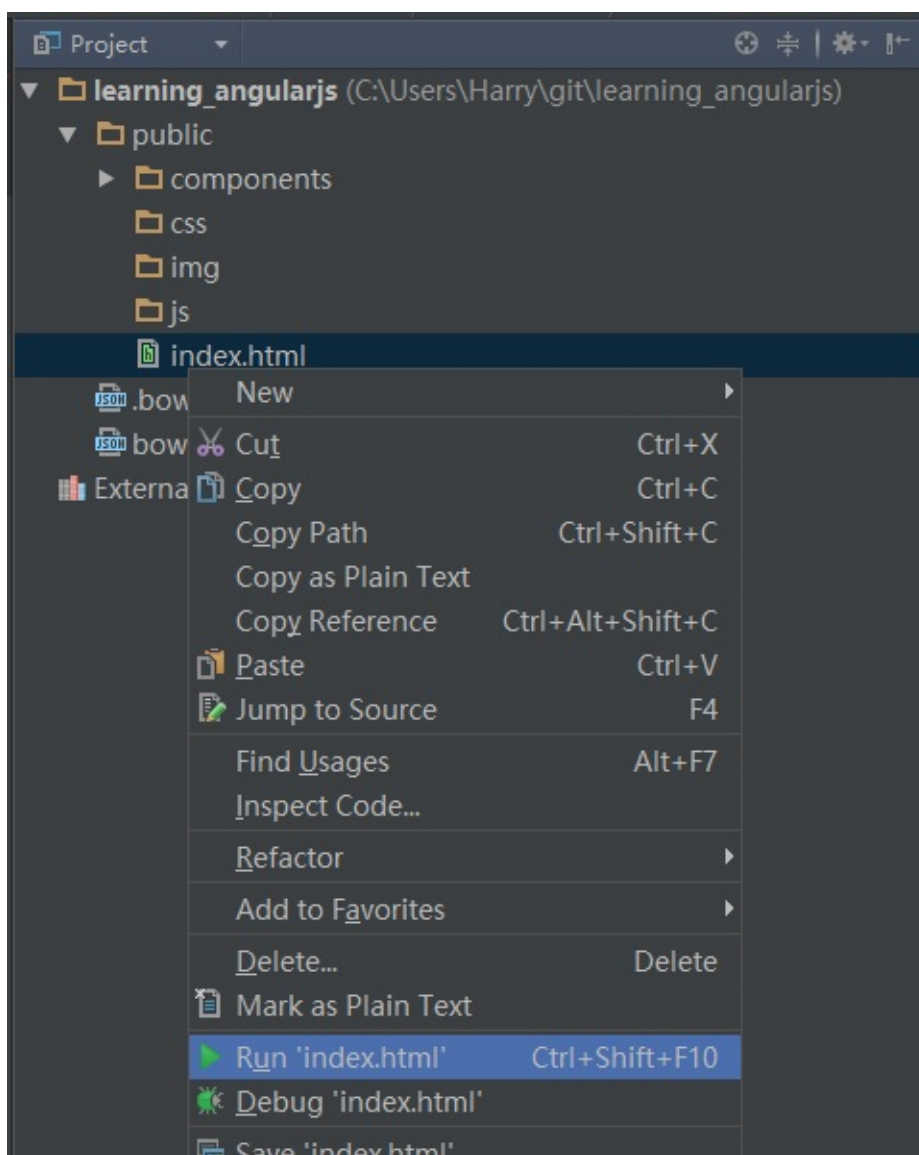
是时候来测试下我们是否成功的引入了AngularJS了。我们在元素中加入一个`ng-app=""`，然后加入一行代码 `{{"Hello World!"}}`。如下所示：

```
1. <!DOCTYPE html>
2. <html lang="zh">
3. <head>
4.     <meta charset="UTF-8">
5.     <title>学习AngularJS 1.x</title>
6. </head>
7. <body ng-app="">
8.     <h1>{{"Hello World!"}}</h1>
9.     <script type="text/JavaScript"
    src="components/angular/angular.js"></script>
```

```
10. </body>
11. </html>
```

运行Hello World

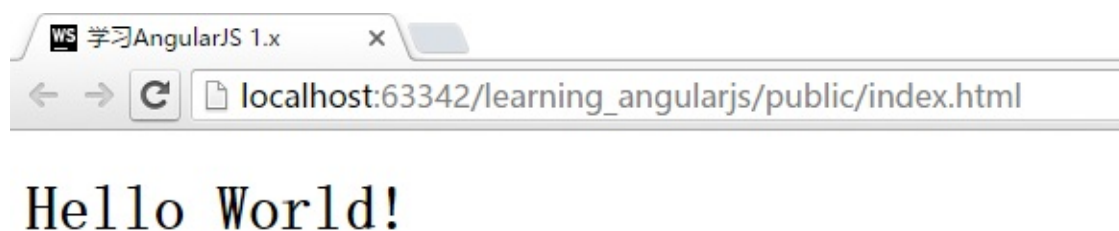
右键选择 `index.html` ，然后选择 `run "index.html"` ，然后你就可以在新打开的浏览器中看到运行的效果。



如果浏览器没有自动打开？

我在Windows电脑上遇到了相同的问题，请将WebStorm使用管理员权限打开。

运行效果如下图所示：



看不到 `{{` 和 `}}`，就说明AngularJS已经成功运行起来了！

如果AngularJS没有成功运行，那么您看到的应该是如下内容：

```
1.  {{"Hello World!"}}
```

如果出现以上结果，请您按书的前面内容仔细检查您的代码。

Hello World到此结束，下面我们将进入AngularJS的世界，学习它，理解它，使用它！

学习和使用AngularJS

- 学习AngularJS的规划

学习AngularJS的规划

我们已经完成了基本的准备工作，从本章开始，我们将专注于学习和应用AngularJS。

作为一个功能完整的框架，AngularJS提供了一套开发理念和方法，我们只需要掌握这套理念和方法即可明确如何实现我们需要的功能。

根据我个人的认知，我设计了如下的学习路线图：

第四章 基本语法

1. 基本表达式
2. AngularJS初始化 `ng-app`
3. 控制器 `ng-controller`
4. 数据绑定 `data-binding`
5. 条件判断语句 `ng-if` / `ng-show` / `ng-hide`
6. 重复语句 `ng-repeat`
7. 过滤器 `filter`
8. 样式选择器 `ng-class` / `ng-style`
9. 下拉列表选项 `ng-options`
0. 引入 `ng-include` 和模板 `ng-template`

深入学习

1. `Directive`
2. 数据获取\$http
3. 如何使用第三方的AngularJS扩充库

4. ui-route
5. 页面模板获取和植入
6. 全局事件监听
7. 如何调试AngularJS代码
8. 如何使用自动化测试工具

表单、数据验证

界面库的引入

1. Angular-Material

案例实践

1. 登陆、注册
2. 权限控制
3. ...

基本表达式

- 基本表达式

基本表达式

在第三章的结尾，我们制作了一个基本的Hello World应用。在其中，我们使用了如下的语法：

1. `<h1>{{"Hello World!"}}</h1>`
- 2.
3. `//上方的“{{”和“}}”既是AngularJS的基本表达式`
- 4.
5. `//如果AngularJS被成功的引入，那么最终的页面将不会显示双括号，而是直接显示Hello World.`

我们可以尝试下将表达式中的内容替换为如下内容，并观察运行的结果：

1. `{{ 1+1 }}` `//网页会显示2`
- 2.
3. `{{ 'a' + 'bc' }}` `//网页会显示abc`

从上我们可以看出，双括号 `{{` 和 `}}` 内的内容，其实是一个JavaScript表达式，并将表达式进行计算的结果显示在此处。这也是AngularJS最吸引人的特性，因为它还支持将JavaScript中的数据显示在此处（我们将在后面的内容中介绍如何操作）。

并且，如果此处输出的是JavaScript中的变量，此处的显示会自动的随JavaScript变量的变化而变化。

AngularJS初始化 ng-app

- AngularJS初始化 `ng-app`
 - 创建一个独立的JavaScript文件，并在HTML中引用
 - AngularJS的作用域
 - 引入文件的顺序

AngularJS初始化 `ng-app`

本节我们将学习基本的 `ng-app` 初始化。

`ng-app` 可以将网页自动初始化为一个AngularJS应用，这样你才能在网页中使用各种AngularJS提供的功能（比如前一节介绍的基本表达式功能）。

在现在阶段，我们并不会用到 `ng-app` 的扩展功能，只需要在独立的JavaScript文件中将 `ng-app` 声明好，再引用入HTML页面即可。

创建一个独立的JavaScript文件，并在HTML中引用

在 `./public/js` 目录中新建一个 `app.js` 文件，并填入如下代码：

```
1. var App = angular.module("App", []);
```

在 `index.html` 中引入 `app.js` 文件，并将 `ng-app` 配置进去即可。

```
1. <!DOCTYPE html>
2. <html lang="zh" ng-app="App">
3. <head>
4.     <meta charset="UTF-8">
5.     <title>学习AngularJS 1.x</title>
```

```

6.  </head>
7.  <body>
8.      <h1>{{"Hello World!"}}</h1>
9.      <script type="text/javascript"
      src="components/angular/angular.js"></script>
10.     <script type="text/javascript" src="js/app.js"></script>
11.     <!-- 这里我们将app.js引入了进来 -->
12. </body>
13. </html>

```

以上代码有两个问题需要注意：

AngularJS的作用域

`ng-app` 标签可以放置在 `<html>` 标签或者 `<body>` 标签上，也可以放置在HTML页面的任何一个标签上。

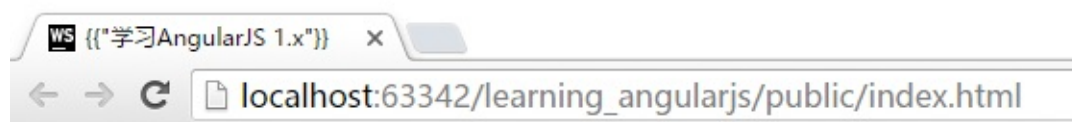
这里，我们就需要注意AngularJS对于作用域的定义。我们先通过如下的例子来看看作用域的具体表现：

```

1.  <!DOCTYPE html>
2.  <html lang="zh">
3.  <head>
4.      <meta charset="UTF-8">
5.      <!-- 网页的标题，我们在这里使用了AngularJS的基本表达式 -->
6.      <title>{{"学习AngularJS 1.x"}}</title>
7.  </head>
8.  <body ng-app="App"> <!-- ng-app被放置在了这里 -->
9.      <h1>{{"Hello World!"}}</h1>
10.     <script type="text/javascript"
      src="components/angular/angular.js"></script>
11.     <script type="text/javascript" src="js/app.js"></script>
12. </body>
13. </html>

```

以上代码的运行结果如下：



Hello World!

我们可以看到，网页标题中的AngularJS表达式并没有执行，这是因为网页的 `<head>` 标签中的内容，并不在AngularJS的管理之下。

如果我们将 `ng-app` 声明放置在 `<body>` 元素中，那么AngularJS只会针对 `<body>` 元素中的内容进行处理。这也是**AngularJS**的核心特性之一，它让我们的JavaScript代码有了作用域的概念，降低了代码之间不期望的一些互相影响。

这个特性我们在后面会大量的使用，将网页分为多个部分，并分别交于不同的JavaScript代码进行管理，各个部分之间互相独立，这样即可在网页中实现逻辑复杂的功能。

引入文件的顺序

引入JavaScript文件的顺序是有差异的，如果我们将上面代码的 `angular.js` 和 `app.js` 文件呼唤，那么网页将不能正常的展示。并且我们可在Chrome的“开发者工具”中看到报错信息。

控制器 ng-controller

- 控制器 `ng-controller`
 - `ng-controller` 详解
 - JavaScript部分
 - `function($scope){}` 详解
 - HTML中调用 `ng-controller` 中的数据
 - 为什么要额外封装一层

控制器 `ng-controller`

控制器 `ng-controller` 是使用AngularJS的核心功能之一。在前一节我们已经了解了作用域的概念，`ng-controller` 则是真正应用作用域来制作功能的核心部分。

应用 `ng-controller` 和应用 `ng-app` 类似，下面我们来尝试创建一个控制器吧！

还是在app.js中，我们创建一个控制器，代码如下：

```
1. //app.js
2. var App = angular.module("App", []);
3.
4. App.controller("FirstCtrl", function($scope){
5.     $scope.data = {
6.         message : "Hello"
7.     };
8. });
```

同时，我们在 `index.html` 中进行一些代码修改，最终代码如下：

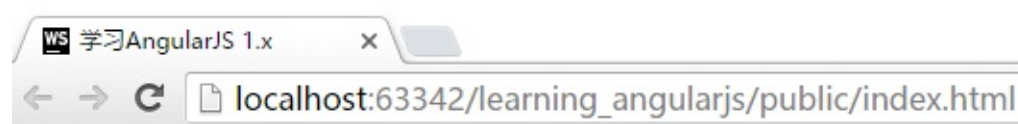
```
1. <!DOCTYPE html>
2. <html lang="zh" ng-app="App">
```

```

3. <head>
4.   <meta charset="UTF-8">
5.   <!-- 网页的标题，我们在这里使用了AngularJS的基本表达式 -->
6.   <title>{{"学习AngularJS 1.x"}}</title>
7. </head>
8. <body>
9.   <!-- 将FirstCtrl绑定到这个div标签上，这个标签中的内容将可以使用
      FirstCtrl中的数据-->
10.  <div ng-controller="FirstCtrl">
11.    <h1>{{data.message + " World!"}}</h1>
12.  </div>
13.
14.  <!-- 以下表达式不会输出任何内容，因为它在FirstCtrl之外-->
15.  <p>下面的内容不会显示</p>
16.  <p>{{data.message}}</p>
17.
18.  <script type="text/javascript"
      src="components/angular/angular.js"></script>
19.  <script type="text/javascript" src="js/app.js"></script>
20. </body>
21. </html>

```

刷新页面，我们可以看到运行的效果：



Hello World!

下面的内容不会显示

下面，我们再来分析下 `ng-controller` 的具体形式：

`ng-controller` **详解**

JavaScript部分

我们先分析 `ng-controller` 的JavaScript编码部分

```

1. //原有的ng-app声明部分
2. var App = angular.module("App", []);
3.
4. /**
5.  * App.controller 声明ng-controller的方法
6.  * "FirstCtrl" 这个ng-controller的名称
7.  * function($scope){} 这个ng-controller的实体，并注入$scope（下文详解）
8.  */
9. App.controller("FirstCtrl", function($scope){
10.     $scope.data = {
11.         message : "Hello"
12.     };
13. });

```

`function($scope){}` 详解

`function(){}` 封装的函数，会被绑定到 `FirstCtrl` 上。这个概念相对容易理解，我们需要注意的，是我们在 `function` 中传入的参数 `$scope`。

与一般的函数声明时的参数不同，此处的参数是不可随意命名的，AngularJS会解析参数的名称，并转化为对应的对象传入。

这里使用的 `$scope`，用于将 `ng-controller` 中的数据和HTML代码绑定起来，传入 `$scope` 的数据，可以直接在HTML代码中调用。在上面的例子中，我们对 `$scope` 传入了 `{data:{message:"Hello"}}` 对象，并在HTML代码中直接使用了 `data.message` 来调用。

`data` 命名并不是固定用法，我们也可以使用 `$scope.shuju = {m:"hello"}`。（这里只是为了表明变量命名的约束，如果可能，请不要

使用拼音命名的变量)。

值得注意的是，`$scope` 之下除了可以传入数据外，还可以传入其他函数，比如我们声明一个 `onClick` 函数传入 `$scope` 之后，可在HTML页面中调用这个功能，实现比如按钮点击触发功能的效果。

HTML中调用 `ng-controller` 中的数据

```
1. <h1>{{data.message + " World!"}}</h1>
```

在 `$scope` 中传入数据后，通过表达式可以直接调用。

为什么要额外封装一层

在AngularJS中，最简单传入数据的方法其实可以更简单，但是不推荐这样做。

最简单的做法

```
1. //JavaScript
2. $scope.message = "Hello";
```

```
1. //HTML
2. <div ng-controller="FirstCtrl">
3.   {{message}}
4. </div>
```

为什么不推荐这样做呢？因为在我们后续会学习应用 `filter` 或 `directive` 等功能时，或者将数据在多个 `ng-controller` 之间共享时，如果不对数据进行二次封装，可能会导致数据互相访问不了的情况。在使用AngularJS的时候，养成数据二次封装的习惯，可以避免很多这样的问题。

数据绑定 data-binding

- 数据绑定data-binding
 - 功能的绑定
 - 更多地方的绑定

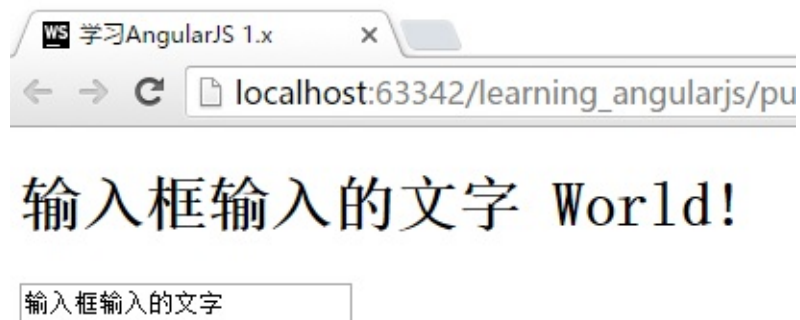
数据绑定data-binding

上一节我们介绍了如何将 `ng-controller` 的数据显示在HTML当中，但是我们如何在页面中修改这些数据呢？这一步操作也很简单。

我们先直接看看代码吧！

```
1. <!DOCTYPE html>
2. <html lang="zh" ng-app="App">
3. <head>
4.     <meta charset="UTF-8">
5.     <!-- 网页的标题，我们在这里使用了AngularJS的基本表达式 -->
6.     <title>{{"学习AngularJS 1.x"}}</title>
7. </head>
8. <body>
9.     <!-- 将FirstCtrl绑定到这个div标签上，这个标签中的内容将可以使用
        FirstCtrl中的数据-->
10.    <div ng-controller="FirstCtrl">
11.        <h1>{{data.message + " World!"}}</h1>
12.        <!-- 这里增加了一个input输入框，并使用ng-model绑定了data.message-
            -->
13.        <input type="text" ng-model="data.message">
14.    </div>
15.
16.    <script type="text/javascript"
        src="components/angular/angular.js"></script>
17.    <script type="text/javascript" src="js/app.js"></script>
18. </body>
19. </html>
```

运行结果



如果您实际运行代码，会发现，您每输入/删除一个文字，改动会马上在上方显示出来。这就是数据绑定的魅力！

功能的绑定

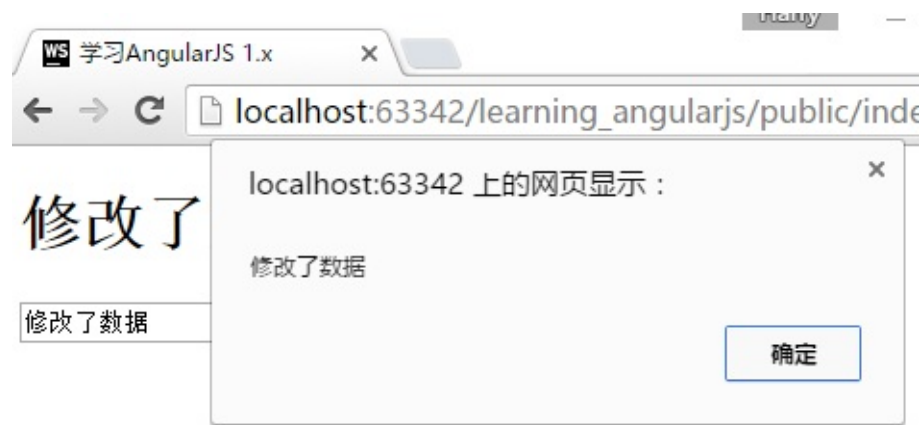
界面操作里面还有一个重要的功能，就是界面上按钮的与控制器中的函数进行绑定。这个也很容易实现：

```
1. App.controller("FirstCtrl", function($scope){
2.     $scope.data = {
3.         message : "Hello"
4.     };
5.
6.     //在$scope上绑定一个函数
7.     $scope.onClick = function(){
8.         alert($scope.data.message);
9.     }
10. });
```

```
1. <div ng-controller="FirstCtrl">
2.     <h1>{{data.message + " World!"}}</h1>
3.     <input type="text" ng-model="data.message">
4.     <!-- 这里我们放置一个按钮，并使用ng-click绑定了$scope.onClick事件-->
5.     <input type="button" value="按钮" ng-click="onClick()">
```

6. `</div>`

点击按钮，我们即可看到数据通过弹出框显示了出来（`onClick` 函数中的逻辑）。



更多地方的绑定

除了用于显示信息外，数据绑定还可用于其他地方。比如，下面的例子用于根据输入来调整样式：

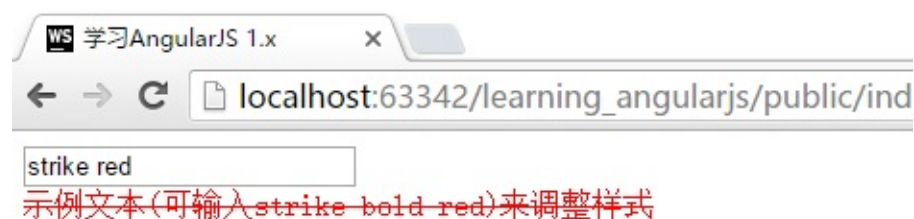
```

1. <!DOCTYPE html>
2. <html lang="zh" ng-app="App">
3. <head>
4.     <meta charset="UTF-8">
5.     <title>{{"学习AngularJS 1.x"}}</title>
6.     <style type="text/css">
7.         /* 删除线 */
8.         .strike {
9.             text-decoration: line-through;
10.        }
11.
12.        /* 粗体 */
13.        .bold {
14.            font-weight: bold;
15.        }
16.    </style>

```

```
17.      /* 红色 */
18.      .red {
19.          color: red;
20.      }
21.  </style>
22. </head>
23. <body>
24.
25. <input type="text" ng-model="data.style">
26. <div class="{{data.style}}">示例文本(可输入strike bold red)来调整样
    式</div>
27.
28. <script type="text/javascript" src="components/angular/angular.js">
    </script>
29. <script type="text/javascript" src="js/app.js"></script>
30. </body>
31. </html>
```

运行效果如下：



条件判断 ng-if / ng-show / ng-hide

- 条件判断 `ng-if` / `ng-show` / `ng-hide`
 - `ng-if` 与 `ng-show` / `ng-hide` 的区别
 - 冒号中的表达式

条件判断 `ng-if` / `ng-show` / `ng-hide`

本节我们将学习的是如何通过变量来控制HTML是否显示。

这三个语句具体的用法如下：

在 `$scope.data` 中添加 `flag` 变量，设置默认值为 `true`

```
1. //app.js
2. $scope.data = {
3.     message: "Hello",
4.     flag: true
5. };
```

将以下代码放置在FirstCtrl中

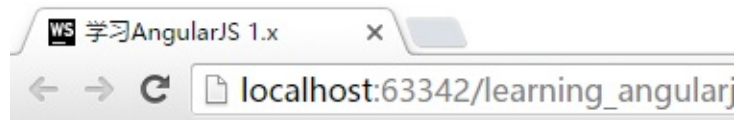
```
1. <input type="checkbox" ng-model="data.flag">通过复选框来控制文字是否显示
2.
3. <!-- 如果data.flag == true,则显示此段文字-->
4. <div ng-if="data.flag">
5.     <p>ng-if中的文字</p>
6. </div>
7.
8. <!-- 如果data.flag == true,则显示此段文字-->
9. <div ng-show="data.flag">
10.     <p>ng-show中的文字</p>
11. </div>
12.
13. <!-- 如果data.flag == false,则显示此段文字-->
```

```

14. <div ng-hide="data.flag">
15.     <p>ng-hide中的文字</p>
16. </div>

```

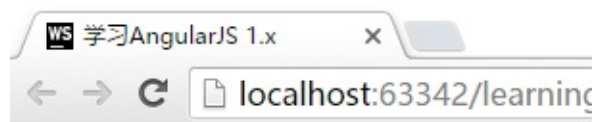
运行结果



☒ 通过复选框来控制文字是否显示

ng-if中的文字

ng-show中的文字



☐ 通过复选框来控制文字是否显示

ng-hide中的文字

从上面的例子可以看出，如果复选框打勾，则界面只显示了前两行文字；而取消复选框的打勾，则显示了最后的一行文字。这个特性可以用于展示界面上的某些信息或者按钮。

ng-if 与 ng-show / ng-hide 的区别

虽然效果看起来类似，但 `ng-if` 的工作模式与 `ng-show` / `ng-hide` 不一样。

如果使用 `ng-if` 来控制元素是否显示，则在不显示的情况下，`ng-if` 中包含的内容，会被全部从HTML中移除掉。

而如果使用 `ng-show` / `ng-hide`，AngularJS只是使用CSS控制将内

容隐藏起来。

这两者可以应用于不同的场景，如果内容较多，且之后不会使用到，那么可以使用 `ng-if`；如果之后还可能会显示出来，那么可以使用 `ng-show` / `ng-hide`。

冒号中的表达式

例子中使用了 `data.flag` 直接作为判断依据，但是以上三个标签都支持传入表达式。

比如以下的表达式都可以作为冒号中的表达方式：

```
1. data.flag >= 1
2. data.flag == true
```

但是请注意，在 `ng-model`，`ng-if` 等标签中传入参数时，是不需要双括号 `{{` 与 `}}` 将参数包裹起来的。

重复语句 ng-repeat

- 重复语句 `ng-repeat`
 - `$index`

重复语句 `ng-repeat`

本节将讲述如何使用 `ng-repeat` 对一个列表的数据进行遍历并显示出来。

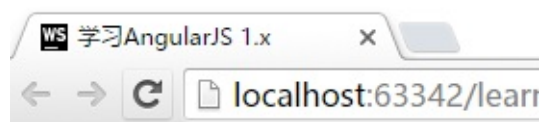
首先，我们准备如下的数据。这是一个包括三个 `object` 的 `array`。

```
1. $scope.list = [  
2.     {  
3.         name: "Harry"  
4.     },  
5.     {  
6.         name: "Tom"  
7.     },  
8.     {  
9.         name: "Jerry"  
10.    }  
11. ];
```

下面，我们将这些数据显示在HTML的一个表格中：

```
1. <table>  
2.     <tr ng-repeat="x in list">  
3.         <td>{{x.name}}</td>  
4.     </tr>  
5. </table>
```

运行结果



Harry
Tom
Jerry

为了更好的查看具体的运行效果，我们可以查看下最终生成的HTML代码。

```
▼ <table>
  ▼ <tbody>
    <!-- ngRepeat: x in list -->
    ▼ <tr ng-repeat="x in list" class="ng-scope">
      <td class="ng-binding">Harry</td>
    </tr>
    <!-- end ngRepeat: x in list -->
    ▼ <tr ng-repeat="x in list" class="ng-scope">
      <td class="ng-binding">Tom</td>
    </tr>
    <!-- end ngRepeat: x in list -->
    ▼ <tr ng-repeat="x in list" class="ng-scope">
      <td class="ng-binding">Jerry</td>
    </tr>
    <!-- end ngRepeat: x in list -->
  </tbody>
</table>
```

从源代码可以看出来，`ng-repeat` 直接将其所在的 `<tr>` 元素根据 `list` 的长度复制了3次。并将相应的数据填充了进去。

\$index

在实际的使用场景中，如果我们使用一个表格来管理信息，那么可能表格的每一行都会有一些对应的操作功能（如编辑、删除）等。那么，我们如何在 `ng-repeat` 中知道是哪一行被点击了呢？

AngularJS提供了 `$index` 这个字段让我们实现这个功能。

下面，我们来看看使用 `$index` 的例子：

在 `app.js` 中，对 `onClick` 函数进行一些改造，让他能够获取传入的数据：

```
1. $scope.onClick = function (index) {
2.     alert("点击了第"+index+"行的按钮");
3. };
```

将刚才的表格也进行一些改造：

```
1. <table>
2.     <tr ng-repeat="x in list">
3.         <td>{{x.name}}</td>
4.         <td><input type="button" value="我是第{{$index}}行的按钮"
5.             ng-click="onClick($index)"></td>
6.     </tr>
7. </table>
```

刷新页面后，让我们点击第一个按钮，效果如下：



这样，我们就可以明确的知道用户点击了哪一行了！

另外，值得注意的是，`$index` 是从 `0` 开始计算的哟！

过滤器 filter

- 过滤器 `filter`
 - 多个filter同时应用
 - 创建自己的过滤器
 - 通过 `filter` 进行搜索
 - 一些值得注意的用法

过滤器 `filter`

过滤器是AngularJS的另一项强大的功能，如果能使用好它，能够帮助我们极大的节省工作量。

以下是几个应用 `filter` 的例子：

```
1. {{ 1234 | number:2 }}
2. //显示两位小数, 结果 1,234.00
3.
4. {{ 1234.56 | currency:"人民币¥":0 }}
5. //转化为货币后输出(保留0位小数, 四舍五入), 结果为
6. //人民币¥1,234.00
7.
8. {{ list | json }}
9. //将对象转化为json文本输出, 结果为
10. //[ { "name": "Harry" }, { "name": "Tom" }, { "name": "Jerry" } ]
11.
12. <tr ng-repeat="x in list | orderBy:'name'">
13. //对显示的数据列表按照name进行排序
14. //结果为显示顺序Harry, Jerry, Tom
```

以上都是AngularJS的常用用法，具体的系统自带的 `filter` 的列表，我们可以从[官方网站](#)上获取。这里对官网提供的功能进行一个简要列表：

Filter名称	示例用法	说明
filter	-	传入自定义的函数作为过滤器
currency	currency / currency:"人民币 ¥":0	转化为货币后输出。可选货币单位和保留小数位数。
number	number / number:2	将数字转化为文本，自动加逗号。可选设置小数位数。
date	data : format : timezone	将时间转化到对应的格式和时区
json	json	将对象转化为Json格式内容输出
lowercase	lowercase	将文本转化为小写
uppercase	uppercase	将文本转化为大写
limitTo	limitTo : limit : begin	截取array从begin位置开始的limit个元素
orderBy	orderBy : expression : reverse	根据expression的条件对list进行排序，reverse可选，设置为true则反过来排

多个filter同时应用

AngularJS支持多个filter同时应用，比如以下的例子：

1. `{{ list | orderBy:'name' | json }}`
2. `//对list的内容进行排序后输出成json文本，结果为`
3. `//[{ "name": "Harry" }, { "name": "Jerry" }, { "name": "Tom" }]`

创建自己的过滤器

自己创建自定义的过滤器也很简单，我们下面尝试自己制作一个将文字全部翻转过来的过滤器。

在 `app.js` 中增加如下代码：

1. `//app.js`
2. `App.filter("reverse", function(){`
3. `return function(text){`

```

4.         return text.split("").reverse().join("");
5.     }
6. });

```

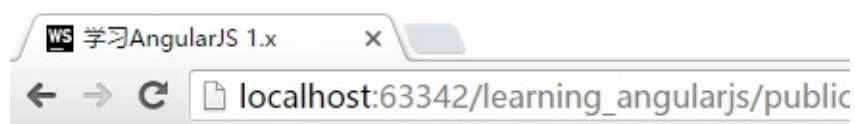
同时，我们利用最早的Hello World的例子，将我们定义的 `reverse` 这个过滤器应用上去，代码如下：

```

1. <div ng-controller="FirstCtrl">
2.     <h1>{{data.message | reverse}}</h1>
3.     <input type="text" ng-model="data.message">
4. </div>

```

运行效果：



果效的后字文他其入输

输入其他文字后的效果

如果您期望界面显示的内容进行一些通用的处理，但是又不希望对原本的数据进行改动，那么可以考虑自己制作过滤器！

通过 `filter` 进行搜索

AngularJS提供了通过filter的搜索功能。当然，这个搜索功能并不是非常常用，因为搜索工作现在一般在服务端完成。如果数据量非常小（几百行以内），可以考虑使用本功能来筛选结果。

示例如下(使用上一节的例子)：

```

1. App.controller("FirstCtrl", function ($scope) {

```

```

2.     $scope.searchText = '';
3.
4.     $scope.list = [
5.         {
6.             name: "Harry"
7.         },
8.         {
9.             name: "Tom"
10.        },
11.        {
12.            name: "Jerry"
13.        }
14.    ];
15. });

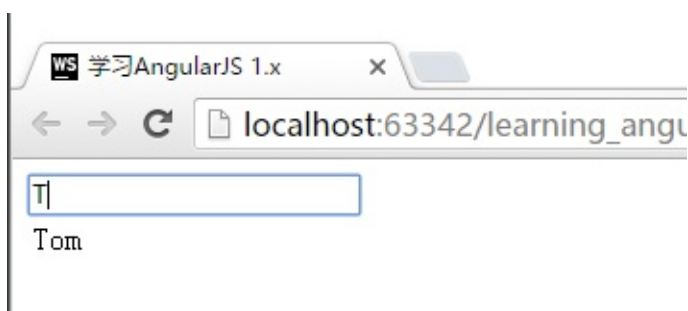
```

```

1. <div ng-controller="FirstCtrl">
2.     <input type="text" ng-model="searchText">
3.     <table>
4.         <tr ng-repeat="x in list | filter:searchText">
5.             <td>{{x.name}}</td>
6.         </tr>
7.     </table>
8. </div>

```

如果我们在输入框中输入T，则列表中只会显示包含T的项目。



一些值得注意的用法

用法	效果
	搜索所有字段

	搜索所有字段
searchText = {name:"T"}	只搜索 <input type="text" value="name"/> 字段包含 <input type="text" value="T"/> 的项目
searchText = {name:"T", last:"H"}	搜索 <input type="text" value="name"/> 字段包含 <input type="text" value="T"/> 且 <input type="text" value="last"/> 字段包含 <input type="text" value="H"/> 的项目

对于最后一项，我们可以采取如下输入方法来应用：

```

1. <div ng-controller="FirstCtrl">
2.     <input type="text" ng-model="searchText.name">
3.     <input type="text" ng-model="searchText.last">
4.     <table>
5.         <tr ng-repeat="x in list | filter:searchText">
6.             <td>{{x.name}}</td>
7.             <td>{{x.last}}</td>
8.         </tr>
9.     </table>
10. </div>

```

样式选择器 ng-class/ng-style

- 样式选择器 `ng-class` / `ng-style`
 - `ng-class`
 - 动态化的样式输入
 - 结合两种模式的应用示例
 - CSS动画效果应用
 - `ng-style`

样式选择器 `ng-class` / `ng-style`

`ng-class`

通过 `ng-class`，我们可以对界面元素的css样式进行控制。下面，让我们通过示例来看看功能如何实现：

此示例来源于官网的[ngClass介绍界面](#)，我进行了一些加工。

首先，我们先创建一个 `style.css` 文件。

```
1.  /* 删除线 */
2.  .strike {
3.      text-decoration: line-through;
4.  }
5.
6.  /* 粗体 */
7.  .bold {
8.      font-weight: bold;
9.  }
10.
11. /* 红色 */
12. .red {
13.     color: red;
14. }
```



```

15.
16.  /* 错误 */
17.  .has-error {
18.      color: red;
19.      background-color: yellow;
20.  }
21.
22.  /* 橙色 */
23.  .orange {
24.      color: orange;
25.  }

```

修改 `FirstCtrl` 为如下代码：

```

1.  //app.js
2.  App.controller("FirstCtrl", function ($scope) {
3.      $scope.data = {
4.          deleted:false,
5.          important:false,
6.          error:false
7.      };
8.  });

```

并在 `index.html` 的 `<head>` 部分将css文件引入，并加入对应的代码。
全部代码如下：

```

1.  <!DOCTYPE html>
2.  <html lang="zh" ng-app="App">
3.  <head>
4.      <meta charset="UTF-8">
5.      <title>{{"学习AngularJS 1.x"}}</title>
6.
7.      <!-- 此处引入style.css样式文件-->
8.      <link type="text/css" rel="stylesheet" href="css/style.css">
9.
10. </head>
11. <body>

```

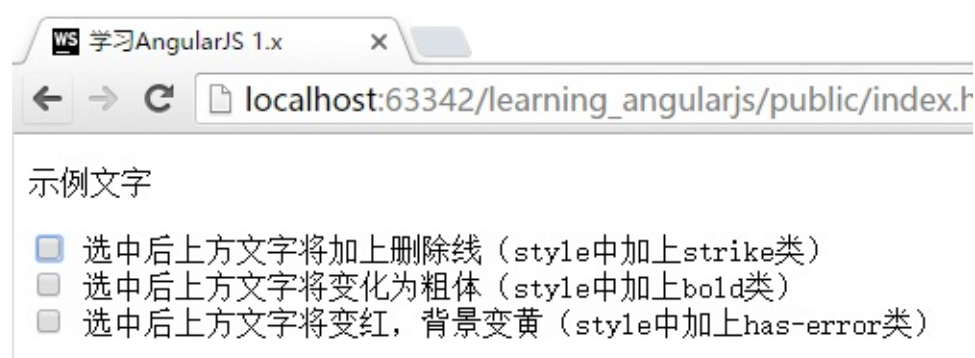
```

12. <div ng-controller="FirstCtrl">
13.
14.     <!-- 此处放置了ng-class, 并设定了每个样式激活时的条件（对应下方3个复选
        框）-->
15.     <p ng-class="{strike: data.deleted, bold: data.important, 'has-
        error': data.error}">示例文字</p>
16.     <input type="checkbox" ng-model="data.deleted">
17.         选中后上方文字将加上删除线（style中加上strike类） <br>
18.     <input type="checkbox" ng-model="data.important">
19.         选中后上方文字将变化为粗体（style中加上bold类） <br>
20.     <input type="checkbox" ng-model="data.error">
21.         选中后上方文字将变红，背景变黄（style中加上has-error类）
22. </div>
23.
24. <script type="text/javascript" src="components/angular/angular.js">
    </script>
25. <script type="text/javascript" src="js/app.js"></script>
26. </body>
27. </html>

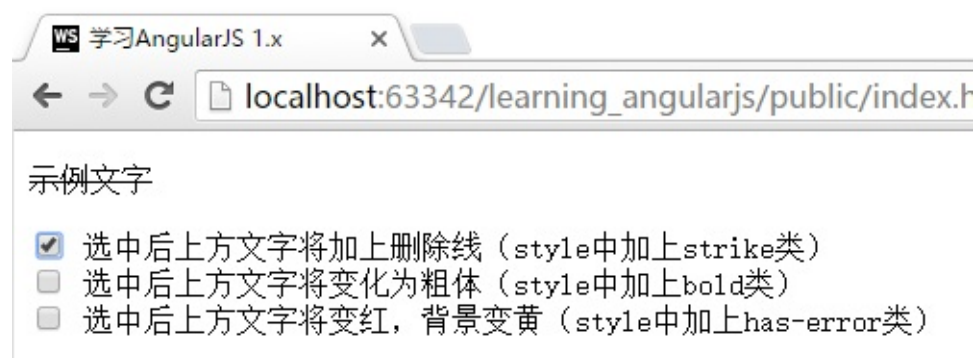
```

运行效果：

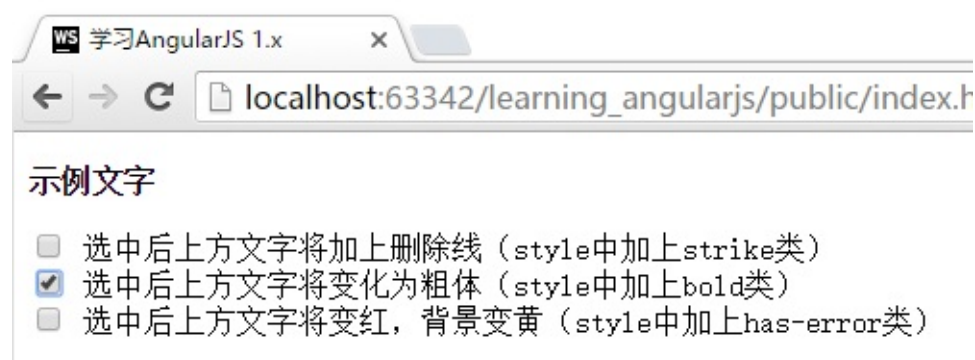
未选中时效果



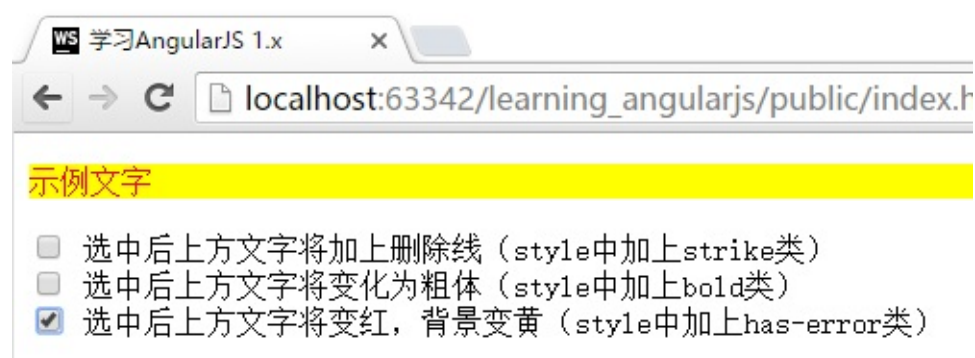
选中第一个的效果



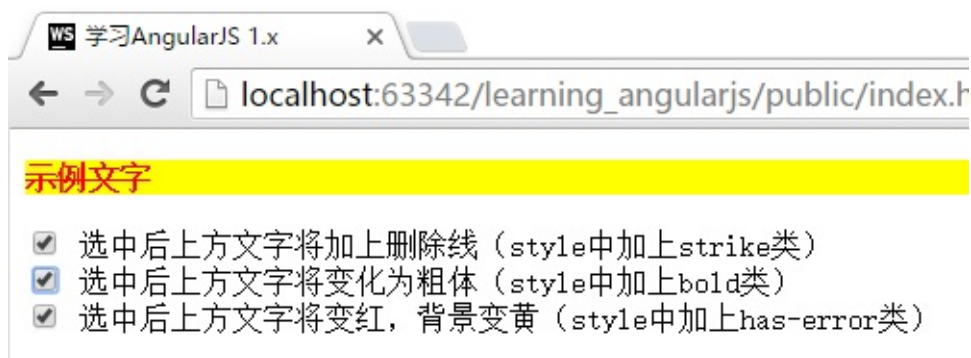
选中第二个的效果



选中第三个的效果



全部选中的效果



动态化的样式输入

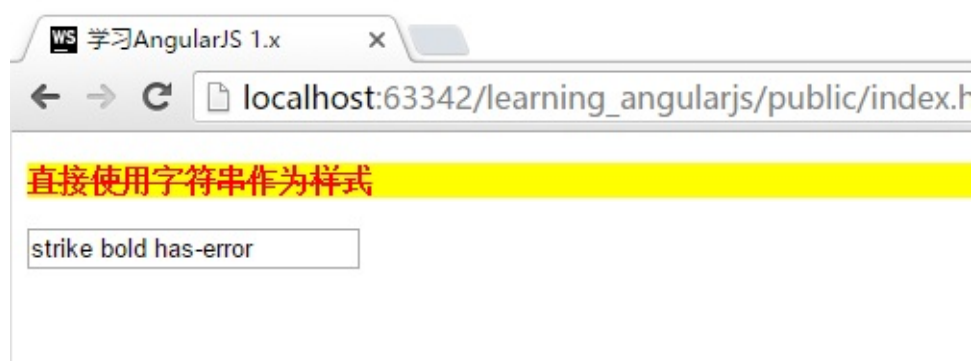
在上方的示例中，我们是对提前设定好的样式进行是否生效的判断。除了这种使用方式，ng-class还支持直接传入字符串的方式进行样式调整。

比如如下示例：

```
1. App.controller("FirstCtrl", function ($scope) {
2.     $scope.data = {
3.         style: ""
4.     };
5. });
```

```
1. <div ng-controller="FirstCtrl">
2.     <p ng-class="data.style">直接使用字符串作为样式</p>
3.     <input type="text" ng-model="data.style">
4. </div>
```

运行效果



结合两种模式的应用示例

这两种模式也可以结合使用，示例如下：

```
1. <p ng-class="[data.style, {orange: warning}]">同时应用两种样式</p>
```

此示例可举一反三，比如加入多个文本输入（对应多个来源），以及多个设定好的样式开关。

此示例就不进行具体的运行效果展示了，请读者自行测试效果。

CSS动画效果应用

我们先在`style.css`中加入css的动画效果代码：

```
1. .base-class {
2.     transition:all cubic-bezier(0.250, 0.460, 0.450, 0.940) 0.5s;
3. }
4.
5. .base-class.animate {
6.     color: red;
7.     font-size:3em;
8. }
```

修改HTML代码如下：

```
1. <div ng-controller="FirstCtrl">
```

```

2.     <input type="button" value="开始动画" ng-
      click="data.style='animate'">
3.     <br>
4.     <input type="button" value="恢复原始" ng-click="data.style='' ">
5.     <br>
6.     <span class="base-class" ng-class="data.style">示例文本</span>
7. </div>

```

运行之后，点击上面的按钮，则文字放大变红。点击第二个按钮，则文本变回原来的样子。

由于动画效果无法通过截图表示，还请读者自行测试。

ng-style

`ng-style` 提供的功能比 `ng-class` 要少一些，只支持样式的传入。我们可以使用以下两种模式：

```

1. <span ng-style="{ 'background-color': data.colorInput }">示例文本
   </span>
2. //colorInput为$scope中的对象，传入文本即可
3.
4. <span ng-style="data.styleText">示例文本</span>
5. //styleText为样式为 '{ 'color': red }' 类型的文本

```

通过样式传入，我们可以直接向元素传入对应的样式，实现样式动态化的效果。

一个比较主要的用途是向元素传入动态的背景图片，例子如下（以下两个示例来自于[StackOverflow](#)）：

```

1. data-ng-style="{ 'background-
  image': 'url(img/products/{{product.img}})' }"

```

也可以传入一个函数（主要用于解决IE11中背景图片不显示的问题）：

```

1. <div ng-style="getBackgroundStyle(imagepath)"></div>
2.
3. <script type="text/javascript">
4.     //代码放置在ng-controller中
5.     $scope.getBackgroundStyle = function (imagepath) {
6.         return {
7.             'background-image': 'url(' + imagepath + ')'
8.         }
9.     }
10. </script>

```

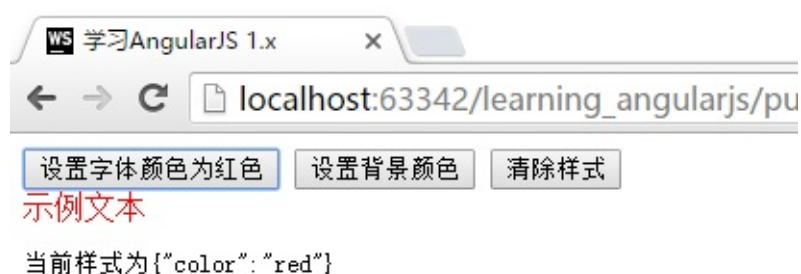
下面，我们通过官网的[示例](#)来看看如何传入文本：

```

1. <input type="button" value="设置字体颜色为红色" ng-
   click="data.myStyle={color:'red'}">
2. <input type="button" value="设置背景颜色" ng-click="data.myStyle=
   {'background-color':'blue'}">
3. <input type="button" value="清除样式" ng-click="data.myStyle={}">
4. <br>
5. <span ng-style="data.myStyle">示例文本</span>
6. <pre>当前样式为{{data.myStyle}}</pre>

```

运行效果为：



在下一节中，我们还将看到约束为只设置背景颜色的示例。

下拉列表选项 ng-options

- 下拉列表选项 `ng-options`
 - 增加未选中的选项
 - 按组排列 `group by`
 - 禁用某些选项 `disable when`
 - 将对象作为参数传入

下拉列表选项 `ng-options`

在学习了 `ng-repeat` 过后，我们其实已经可以用循环的方式实现下拉列表的选项。但是，AngularJS提供了 `ng-options` 的方法，让我们能够更轻松的完成这项工作。

官网提供了一个详尽的[示例](#)进行演示(点击打开后滑动到页面最下部进行效果测试)，本节中的内容，是将官网的内容进行梳理后进行的讲解。

`ng-options` 提供了很多功能用来梳理或筛选下拉列表的选项。我们将分别学习它们。

首先，让我们学习下如何使用 `ng-options`。这里，我们期望实现的是，使用下拉列表，让页面上的一个方块的颜色对应变化：

我们首先配置一个颜色的列表，并且在 `$scope` 中存储一个用于保存选中状态的变量，将它的默认值设置为颜色列表的第一个：

```
1. App.controller("FirstCtrl", function ($scope) {
2.     $scope.colors = [
3.         {name: '黑色', color:'black' },
4.         {name: '白色', color:'white' },
5.         {name: '红色', color:'red' },
```

```

6.         {name: '蓝色', color:'blue' },
7.         {name: '黄色', color:'yellow'}
8.     ];
9.
10.    //保存选中的状态, 默认颜色设置为黑色
11.    $scope.colorChosen = $scope.colors[0];
12.
13. });

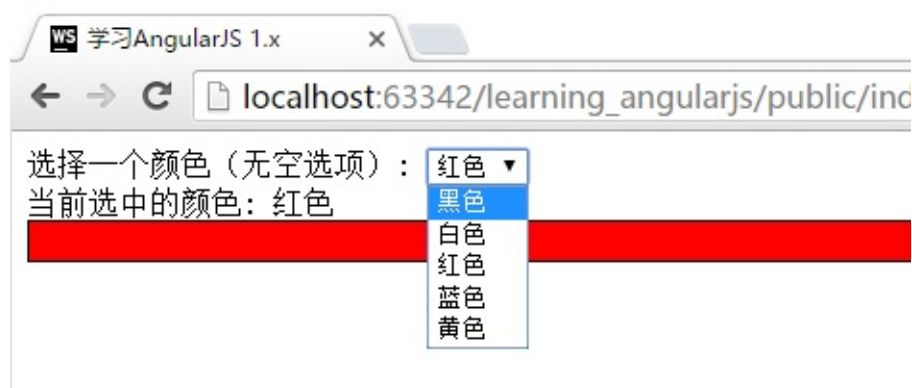
```

```

1. <div ng-controller="FirstCtrl">
2.
3.     <label>选择一个颜色（无空选项）:
4.         <select ng-model="colorChosen" ng-options="color.name for
           color in colors">
5.
6.         </select>
7.     </label>
8.     <br>
9.
10.    当前选中的颜色: {{ colorChosen.name }}
11.    <div style="border:solid 1px black; height:20px"
12.        ng-style="{ 'background-color':colorChosen.color}">
13.    </div>
14. </div>

```

运行页面，我们可以看到下拉列表中有列表中的五种颜色，并且选中不同颜色后，下方方框内的颜色，会根据选择变化。



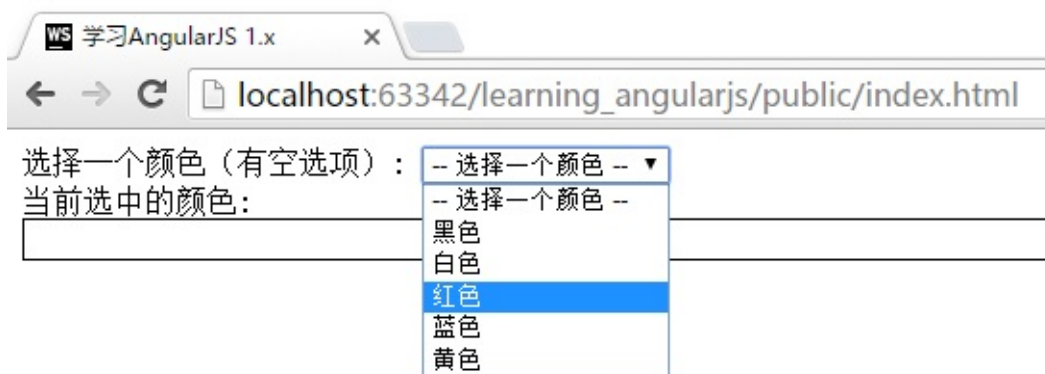
颜色根据选中状态变化，利用到了 `ng-model` 和 `ng-style` 的特性。下面，我们将仔细讲解 `ng-options` 中的语法：

- `color.name`
 - 用于显示在下拉框中的名称
- `for color in colors`
 - 类似于`ng-repeat`中的用法，将`colors`遍历，每次遍历的对象命名为`color`

增加未选中的选项

在很多时候，我们期望能有一个没选中的选项。我们可以通过手动的方式添加这个选项：

```
1. <select ng-model="colorChosen" ng-options="color.name for color in
   colors">
2.     <option value="">-- 选择一个颜色 --</option>
3. </select>
```



按组排列 `group by`

`ng-options` 也支持按组排列数据，使用 `group by` 语法，下面我们看

看例子：

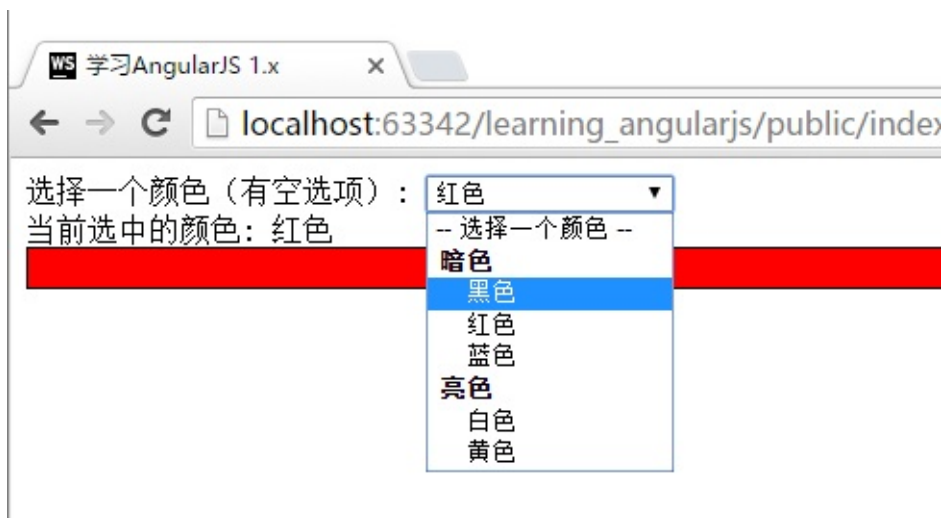
首先，我们在列表中增加类型字段：

```
1. $scope.colors = [
2.     {name: '黑色', color: 'black', type: "暗色"},
3.     {name: '白色', color: 'white', type: "亮色"},
4.     {name: '红色', color: 'red', type: "暗色"},
5.     {name: '蓝色', color: 'blue', type: "暗色"},
6.     {name: '黄色', color: 'yellow', type: "亮色"}
7. ];
```

然后修改ng-options的语法：

```
1. <select ng-model="colorChosen" ng-options="color.name group by
    color.type for color in colors">
```

运行效果：



禁用某些选项

disable when

ng-options 还可以通过 disable when 语法来设置选项是否可选，同样的，我们需要在列表中先新增字段：

注意，以下第一列数据并未加上 `disabled` 属性，这是为了表明如果没有这项数据，默认的 `ng-options` 的操作（可以选择）。

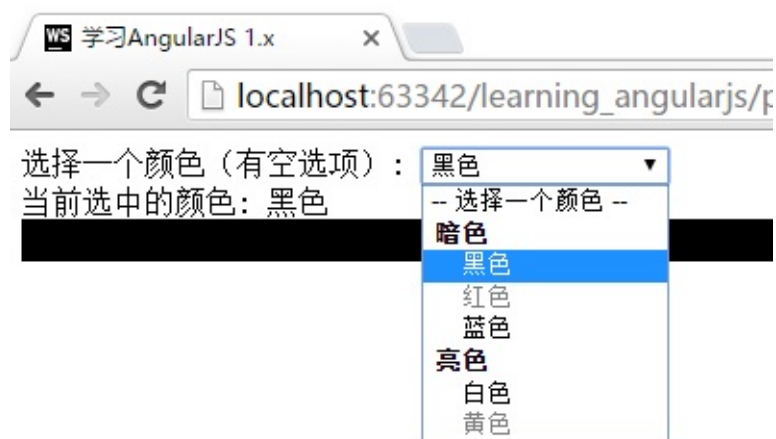
```
1. $scope.colors = [
2.     {name: '黑色', color: 'black', type: "暗色"},
3.     {name: '白色', color: 'white', type: "亮色", disabled: false},
4.     {name: '红色', color: 'red', type: "暗色", disabled: true},
5.     {name: '蓝色', color: 'blue', type: "暗色", disabled: false},
6.     {name: '黄色', color: 'yellow', type: "亮色", disabled: true}
7. ];
```

然后修改 `ng-options` 的语法：

```
1. <select ng-model="colorChosen"
2.         ng-options="color.name group by color.type
3.         disable when color.disabled for color in colors">
```

运行结果：

可以看到，图中的“红色”和“黄色”变为了不可选择的状态。



将对象作为参数传入

在上面的例子中，我们传入的是一个列表 `array`。 `ng-options` 也支持

以对象 `object` 的方式传入数据：

使用 `object` 的方式传入数据，一般是为了简化配置，比如我们采取如下的配置：

```
1. $scope.countries = {
2.     CN: '中国China',
3.     US: '美国United States',
4.     UK: '英国United Kingdom',
5.     GR: '德国Germany'
6. };
7.
8. $scope.country = 'CN';
```

注意，以下代码中，for后面有一个空格。如果没有，则无法成功运行！这是因为传入 `ng-options` 的其实是一串文本，而AngularJS需要解析这段文本，因此提出了对格式的要求。

```
1. <div ng-controller="FirstCtrl">
2.
3.     <label>选择一个国家（有空选项）：
4.         <select ng-model="country" ng-options="k as v for (k, v) in
5.             countries">
6.             <option value="">-- 选择一个国家 --</option>
7.         </select>
8.     </label>
9.     <br>
10.    当前选中的国家：{{ country }}
11. </div>
```

运行结果，请注意列表中显示的值和选中时变量的值的关系。



引入ng-include和模板ng-template

- 引入 `ng-include` 和模板 `ng-template`
 - 引入 `ng-include`
 - 其他属性
 - 模板 `ng-template`

引入 `ng-include` 和模板 `ng-template`

引入 `ng-include` 和模板 `ng-template` 是定义和使用HTML代码碎片的功能。用于将HTML切碎分别存储，并根据需求再去获取对应的代码块，达到加速访问和代码复用的效果。

下面，我们将分别介绍 `ng-include` 和 `ng-template` 。

引入 `ng-include`

当HTML代码过于复杂，或者期望建立单页应用(Single-page Application - SPA)时，需要将部分HTML打包成独立的文件。这时候，我们在引入这个独立HTML文件时，可以使用 `ng-include` 功能。

使用方法如下：

1. `<!-- 直接传入一个网页的地址，请注意这里的使用用法 -->`
2. `<!-- 'view/part.html' 外部有单引号 -->`
3. `<div ng-include="'views/part.html'"></div>`
- 4.
5. `<!-- 假设$scope中有template:{url:"http://..."}这个对象 -->`
6. `<div ng-include="template.url"></div>`
- 7.
8. `<div ng-include="getUrl()"></div>`

从上面的例子可以看出，`ng-include` 支持直接传入静态文本、传入变量、传入函数（返回网页地址）的方式来进行调用。

另外，`ng-include` 的用法也可以直接作为标签名使用，如：

```
1. <div data-ng-include="'views/part.html'"></div>
2.
3. <ng-include src="'views/part.html'"></ng-include>
```

这些用法的效果都是一样的。

其他属性

`ng-include` 还有 `onload` 和 `autoscroll` 的属性。

但是我目前不清楚具体的使用方法和效果，如果有读者清楚，可与我联系以便更新上此段内容。

模板 `ng-template`

`ng-template` 用于将多个HTML片段存放于一个HTML文件中。并且可以根据需求分别调用其中的某一个片段。

`ng-template` 的用法如下：

```
1. <script type="text/ng-template" id="html_part.html">
2.   <!-- HTML片段的实际内容 -->
3. </script>
```

`ng-template` 需要将 `<script>` 的 `type` 设置为 `text/ng-template`，然后给他配置一个 `id`。这个 `id` 就是这段HTML代码被引用时的名称。

使用 `ng-template` 代码片段的方法就是上述的 `ng-include` 方法，例如：

```
1. <div ng-include="'html_part.html'"></div>
```

本章总结

- [本章总结](#)

本章总结

在本章中，我们学习了AngularJS的基本用法。掌握了这些用法，相信大家对于AngularJS已经有了基本的了解。

我建议您在阅读后续内容之前，对本章内容进行一次回顾，并将印象不深的地方多浏览几遍。如果可能，也请尽量的实际的运行一下示例，修改修改代码，观察具体的运行情况。

深入学习AngularJS - Directive

- 深入学习AngularJS - Directive
 - Directive在系统中的使用
 - 学习Directive的路程

深入学习AngularJS - Directive

在上一章中，我们学习了AngularJS的基本用法。从本章开始，我们将学习“深入”一些的部分。

本章将介绍AngularJS的Directive。

有若干AngularJS的中文译文将Directive翻译为“指令”，但是我感觉此翻译很难让读者明确其具体的含义和用法，因此，我在本书中直接应用了英文名。

AngularJS的Directive，从实际用途的理解，可以称之为“自定义HTML标签”。举个例子，AngularJS可以让我们进行如下的HTML编码：

```
1. <!-- 直接作为标签名 -->
2. <people>    </people>
3.
4. <!-- 直接作为标签属性 -->
5. <div people="Harry"> </div>
6.
7. <!-- 作为类属性 -->
8. <div class="people:Harry"> </div>
```

还有一种比较特殊的放置在注释中生效的表达方式，但是我目前没有理解其实现意义，就不在这里介绍了。

如果我们预先定义好了针对这些标签的处理方式，那么AngularJS将可以把这些标签自动的转化成HTML显示代码。

Directive在系统中的使用

其实，Directive作为AngularJS的基本特性，我们已经在前面大量的学习和使用了它。

在第四章中我们学习的 `ng-app` , `ng-controller` , `ng-model` , `ng-if`等使用方法。如果您现在再仔细看下它们的使用方法，就会发现它们无一例外的都是Directive！

学习Directive的路程

本章我们将从最基本的自定义Directive开始，逐渐深入的学习Directive的特性和高级使用方法。由于Directive的特性主要针对展示界面的操作，目的是对界面操作的抽象与解耦。因此，可能像我一样对前端经验不太足的读者们，可能会对Directive的学习或者使用价值感到困难。因此，学习Directive可能会多花您一些时间，但是相信我，这些付出是非常有价值的！

制作一个自定义的Directive

- 制作一个自定义的Directive
 - 代码分析
 - `template`
 - 用替换而不是插入的方式应用Directive
 - `restrict`

制作一个自定义的Directive

下面我们将制作我们的第一个自定义Directive。让我们对 `app.js` 和 `index.html` 进行一些修改：

```

1. //app.js
2. var App = angular.module("App", []);
3.
4. App.directive("people", function(){
5.     return {
6.         restrict: "E",
7.         template : "<p>姓名:{{data.name}}</p><p>性别: {{data.sex}}</p>"
8.     }
9. });
10.
11. App.controller("FirstCtrl", function ($scope) {
12.     $scope.data = {
13.         name: "Harry",
14.         sex : "男"
15.     };
16. });

```

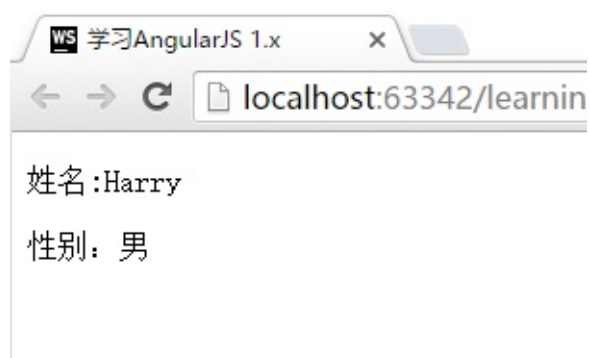
```

1. <!DOCTYPE html>
2. <html lang="zh" ng-app="App">
3. <head>

```

```
4.     <meta charset="UTF-8">
5.     <title>{{"学习AngularJS 1.x"}}</title>
6.     <link type="text/css" rel="stylesheet" href="css/style.css">
7. </head>
8. <body>
9. <div ng-controller="FirstCtrl">
10.     <!-- 注意这里只加入了一个people的标签 -->
11.     <people></people>
12. </div>
13.
14. <script type="text/javascript" src="components/angular/angular.js">
    </script>
15. <script type="text/javascript" src="js/app.js"></script>
16. </body>
17. </html>
```

运行结果：



代码分析

以下这段代码用于声明一个Directive：

```
1. App.directive("people", function(){
2.     return {
3.         restrict : "E",
4.         template : "<p>姓名:{{data.name}}</p><p>性别: {{data.sex}}
    </p>"
5.     }
```

```
6. });
```

我们将这段代码分拆开逐步讲解：

首先，声明一个Directive的基本结构如下，我们调用了 `directive()` 函数来告诉AngularJS加入一个新的Directive：

```
1. App.directive();
```

调用这个函数，我们需要传入2个参数，第一个参数是Directive的命名（这里是 `people` ），第二个参数是这个Directive的功能。

```
1. App.directive("people", function({}));
```

在实例中，我们直接在第二个参数的函数中返回了一个对象：

```
1. return {
2.     restrict: "E",
3.     template : "<p>姓名:{{data.name}}</p><p>性别:{{data.sex}}</p>"
4. }
```

这个对象中有两个元素， `restrict` 和 `template` 。

`template`

`template` 相对比较容易理解，在运行网页时，HTML对应的标签，将被替换成对应的内容。我们这里看看替换后实际的HTML代码如何：

```
<div ng-controller="FirstCtrl" class="ng-scope">
▼<people>
  <p class="ng-binding">姓名:Harry</p>
  <p class="ng-binding">性别: 男</p>
</people>
</div>
```

可以看到，AngularJS在 `<people></people>` 中间加入了template中的内容。

用替换而不是插入的方式应用Directive

如果在配置Directive时，加入 `replace :`
`true`（与 `restrict` 和 `template` 同级别），则可以让AngularJS用替换的模式应用Directive。

```
1. App.directive("people", function(){
2.     return {
3.         restrict : "E",
4.         replace : true,
5.         template : "<p>姓名:{{data.name}}</p><p>性别:{{data.sex}}</p>"
6.     }
7. });
```

具体的效果，是会去除掉 `<people> </people>` 这对标签。

`restrict`

`restrict` 是告诉AngularJS，这个Directive应该如何使用。

下面这个表格，总结了restrict可能有的值，具体的每种应用方案，我们将在下一节详解。

值	对应类型	使用方法
E	element	<code><people> </people></code>
A	attribute	<code><div people> </div></code>
C	class	<code><div class="people"> </div></code>
EAC	-	以上三种都可使用

如果在restrict中设定了使用方法，而在HTML代码中却未按照对应的方法使用，那么代码将不会生效！

Directive的命名和使用规则

- Directive的命名和使用规则
 - 命名规则
 - 使用规则
 - 为什么会有这种差异
 - 怎么实现的？
 - 连接符
 - 为什么要先去除 `data-/x-` 部分

Directive的命名和使用规则

在前一节中，我们创建了一个名为“people”的Directive。并且通过 `<people> </people>` 使用了它。

但是，在实际应用场景中，我们的命名通常不止1个单词，这时候我们就需要注意Directive的命名和使用规则。

对Directive的命名，AngularJS是有特殊的规则需求的。并且，在JavaScript中的命名，与在HTML对应使用时的名称不一样！

命名规则

AngularJS要求Directive的命名使用驼峰式语法，也就是从第二个单词开始，每个单词的首字母大写，并且不使用连接符号。

驼峰式命名的例子：

- `people`
- `peopleList`
- `peopleListArray`

使用规则

在HTML代码中，使用的是连接符的形式，下面我们对比看看命名和使用的对应字符串：

命名	使用
people	people
peopleList	people-list
peopleListArray	people-list-array

实际使用举例

```
1. <people-list> </people-list>
2.
3. <div people-list-array> </div>
```

为什么会有这种差异

命名和用法不同的核心原因，是因为**HTML**对大小写不敏感，而**JavaScript**对大小写敏感。

为了保证HTML和JavaScript都能按原有模式正常工作，AngularJS提出了这套解决方法。

怎么实现的？

AngularJS在解析HTML时，会将名称取出（如 `people-list-array`），并进行一下两个方面的处理：

1. 去除字段的 `x-` 或 `data-` 头。（`people-list-array`）
2. 将字段中的连接符号去除，并将第二个单词开始改为首字母大写，其他字母小写。（`[people, List, Array]`）
3. 然后合并起来。（`peopleListArray`）

连接符

在前面的讲解中，我们讲解示例时，使用的连接符全部都为减号。但实际上，AngularJS支持的连接符有：

符号	示例
减号(-)	people-list
冒号(:)	people:list
下划线(_)	people_list

但是实际使用中，推荐使用减号作为连接符。其他的两种符号，只是因为历史原因提供了支持，但是并不推荐使用。

为什么要先去除 data-/x- 部分

data-/x- 存在的原因是需要符合HTML5标准。如果你使用[HTML5验证器](#)来验证我们之前的代码，你可能会看到如下结果(黄色背景部分代表不符合标准)：

Document checking completed.

Source

```

1. <!DOCTYPE html>↵
2. <html lang="zh" ng-app="App">↵
3. <head>↵
4.   <meta charset="UTF-8">↵
5.   <title>{{"学习"AngularJS 1.x}}</title>↵
6.   <link type="text/css" rel="stylesheet" href="css/style.css">↵
7. </head>↵
8. <body>↵
9.   <div ng-controller="FirstCtrl">↵
10.     <people></people>↵
11.   </div>↵
12.   ↵
13.   <script type="text/javascript" src="components/angular/angular.js"></script>↵
14.   <script type="text/javascript" src="js/app.js"></script>↵
15. </body>↵
16. </html>

```

Total execution time 4 milliseconds.

而如果我们在ng-app和ng-controller前加上data-前缀，则可以

通过HTML5的验证。

Document checking completed.

Source

```

1.  <!DOCTYPE html>↵
2.  <html lang="zh" data-ng-app="App">↵
3.  <head>↵
4.    <meta charset="UTF-8">↵
5.    <title>{{"学习"}AngularJS 1.x}}</title>↵
6.    <link type="text/css" rel="stylesheet" href="css/style.css">↵
7.  </head>↵
8.  <body>↵
9.    <div data-ng-controller="FirstCtrl">↵
10.     <people></people>↵
11.  </div>↵
12.  ↵
13.  <script type="text/javascript" src="components/angular/angular.js"></script>↵
14.  <script type="text/javascript" src="js/app.js"></script>↵
15.  </body>↵
16.  </html>

```

Total execution time 2 milliseconds.

所以，如果你的项目需要使用HTML5的验证工具，那么就需要在字段前加上data-前缀。

另：x-的存在，可能是针对XHTML的支持。

注意：，因为这个条件存在，因此，请不要给你的Directive起data开头的名字！

让Directive支持传入数据

- 让Directive支持传入数据
 - Directive的 `scope`
 - `scope` 中的配置
 - 在一个 `ng-ontroller` 中放入多个相同的Directive
 - 通过 `ng-repeat` 和directive一起显示数据
 - 在Directive中修改控制器中的数据
 - 以只读的方式传入数据
 - 在Directive中进行函数回调

让Directive支持传入数据

在第一节中，我们使用的Directive，可以直接获取并显示控制器中的数据（人的名称和性别）。但是，如果我们有多个人的信息需要显示怎么处理？这个问题其实非常常见，因为Directive通常是将需要在界面中重复使用的部分抽象出来，便于一次修改，多处地方生效（如博客的评论的列表显示）。

如果我们按照现在的代码结构（详见第一节），期望放置不同的人的信息，只能采取重复多个控制器的方式，可能的代码如下：

```

1. <!DOCTYPE html>
2. <html lang="zh" ng-app="App">
3. <head>
4.     <meta charset="UTF-8">
5.     <title>{{"学习AngularJS 1.x"}}</title>
6.     <link type="text/css" rel="stylesheet" href="css/style.css">
7. </head>
8. <body>
9. <div ng-controller="FirstCtrl">
10.     <!-- 第一个人的信息 -->

```

```

11.     <people></people>
12. </div>
13.
14. <div ng-controller="SecondCtrl">
15.     <!-- 第二个人的信息 -->
16.     <people></people>
17. </div>
18.
19.
20. <script type="text/javascript" src="components/angular/angular.js">
    </script>
21. <script type="text/javascript" src="js/app.js"></script>
22. </body>
23. </html>

```

这样做，无法实现动态化的列表，是无法实现我们期望的功能的。我们理想中的情况应该是什么样子呢？当然是能够并列放置这些 Directive，通过传入不同的数据来让Directive展示不同的内容。

```

1. <div ng-controller="FirstCtrl">
2.     <!-- 第一个人的信息 -->
3.     <div people='peopleOneInfo'></div>
4.
5.     <!-- 第二个人的信息 -->
6.     <div people='peopleTwoInfo'></div>
7. </div>

```

这样的功能，可以通过配置Directive的 `scope` 定义实现。

Directive的 `scope`

在之前学习控制器 `ng-controller` 的使用过程中，我们使用了 `$scope` 功能。 `$scope` 用于提供对接HTML和JavaScript对应模块的功能。

而Directive在默认情况下，是没有自动绑定一个 `$scope` 的。也就是说，在默认情况下，Directive无法在JavaScript中接收传入的数据（因为缺少一个存储信息的载体），形成我们期望的效果。但是，Directive提供了非常简单的定义一个scope的功能：

```

1. App.directive("people", function(){
2.     return {
3.         restrict: "A",
4.         scope:{
5.             info: "="
6.         },
7.         template : "<p>姓名:{{info.name}}</p><p>性别: {{info.sex}}</p>"
8.     }
9. });
10.
11. App.controller("FirstCtrl", function ($scope) {
12.     $scope.harry = {
13.         name: "Harry",
14.         sex : "男"
15.     };
16. });

```

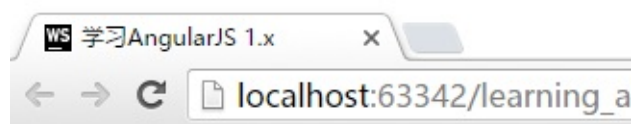
注意，这里我将restrict从"E"(element元素)改变成为了"A"(attribute 属性)，这样它的使用方法有了一些变化：

```

1. <div ng-controller="FirstCtrl">
2.     <div people info="harry"></div>
3. </div>

```

在HTML代码里，我们为div元素配置了一个people的属性和一个info属性；并将FirstCtrl的\$scope.harry传入给了info。最终的显示效果如下：



姓名:Harry

性别: 男

scope 中的配置

可以看到，在上方的JavaScript文件中，我们对scope的定义使用了如下结构：

```
1. scope:{
2.     info: "="
3. }
```

首先，`scope: {}` 是告诉这个Directive它需要自己存储信息（类似于建立一个基于这个Directive的 `$scope`）。

`info: "="` 这段配置，告诉Directive从HTML标签中，获取名为 `info` 的属性，并将它的值存储在 `scope.info` 中。这样，我们就达到了存储数据的效果。

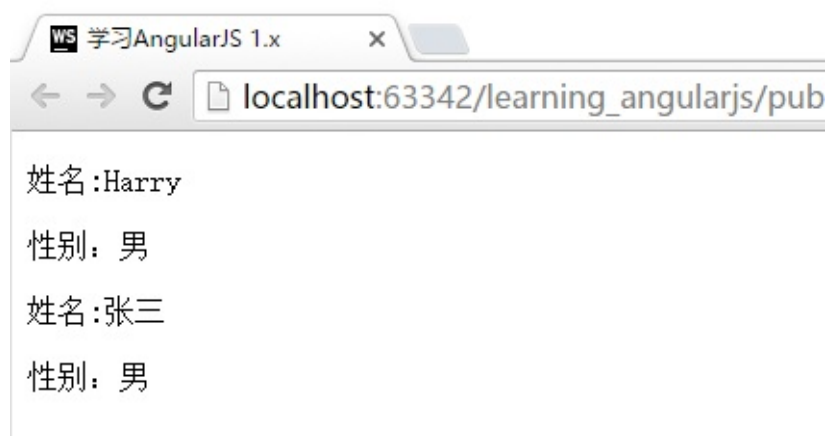
在一个 `ng-oncontroller` 中放入多个相同的Directive

下面，我们在 `FirstCtrl` 中增加几个人的数据，并将它们通过Directive显示出来：

```
1. //在FirstCtrl中加入如下代码
2. $scope.anotherPerson = {
3.     name : "张三",
4.     sex : "男"
5. };
```

```
1. <div ng-controller="FirstCtrl">
2.     <div people info="harry"></div>
3.     <div people info="anotherPerson"></div>
4. </div>
```

运行效果如下：



通过 `ng-repeat` 和 `directive` 一起显示数据

知道了如何传入数据，那么我们就可以将Directive的使用和 `ng-repeat` 结合起来，实现列表显示数据的效果。

我们先将 `FirstCtrl` 的数据变化为一个 `array`：

```
1. App.controller("FirstCtrl", function ($scope) {
2.     $scope.people = [
3.         {
4.             name: "Harry",
5.             sex: "男"
6.         },
7.         {
8.             name: "张三",
9.             sex: "男"
10.        }
11.    ];
12. });
```

```
12. });
```

```
1. <div ng-controller="FirstCtrl">
2.   <span ng-repeat="person in peopleList">
3.     <div people info="person"></div>
4.   </span>
5. </div>
```

实现的效果与上一张图片一样。（具体的页面HTML代码会有差异，请您自行测试查看）

在Directive中修改控制器中的数据

以上我们看到的示例只是将数据显示了出来，如果我们期望在Directive中修改这些数据如何处理呢？

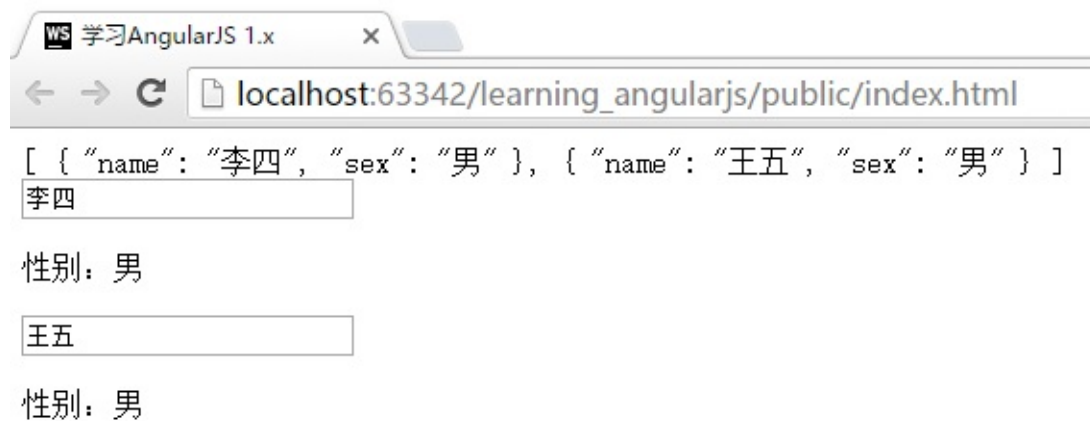
其实很简单，将 `template` 中原先显示的数据的部分，替换为 `input` 即可。

```
1. App.directive("people", function () {
2.   return {
3.     restrict: "A",
4.     scope: {
5.       info: "="
6.     },
7.     template: "<input type='text' ng-model='info.name'><p>性别：
      {{info.sex}}</p>"
8.   }
9. });
```

```
1. <div ng-controller="FirstCtrl">
2.   {{ peopleList | json}}
3.   <span ng-repeat="person in peopleList">
4.     <div people info="person"></div>
5.   </span>
```

6. `</div>`

刷新界面后，我们可以在输入框中尝试修改。效果如下：



以只读的方式传入数据

除了以等号 `=` 直接传入对象之外，Directive也支持直接传入文本，使用 `@` 符号。

```

1. App.directive("people", function () {
2.     return {
3.         restrict: "A",
4.         scope: {
5.             name: "@",
6.             sex : "@"
7.         },
8.         template: "<input type='text' ng-model='name'><p>性别 :
           {{sex}}</p>"
9.     }
10. });

```

```

1. <div ng-controller="FirstCtrl">
2.     {{ peopleList | json}}
3.     <span ng-repeat="person in peopleList">
4.         <!-- 注意此处的数据传入方法 -->

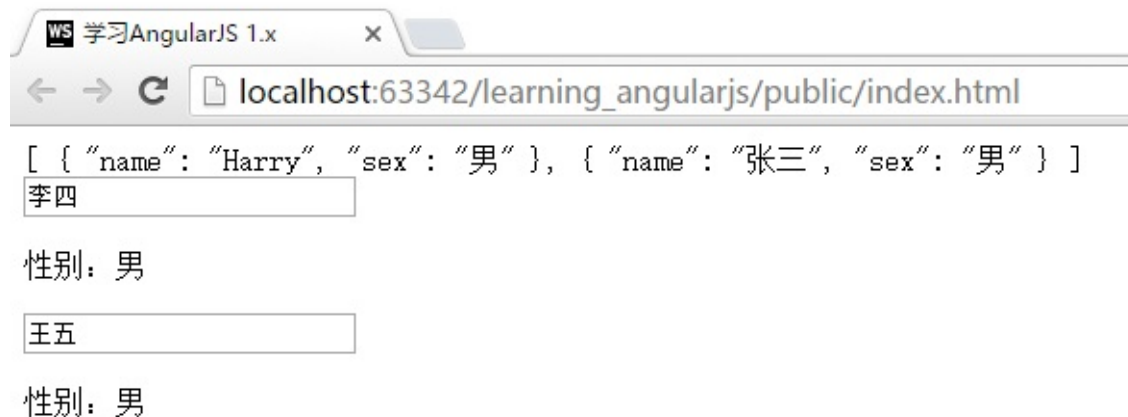
```

```

5.         <div people name="{{person.name}}" sex="{{person.sex}}">
           </div>
6.     </span>
7. </div>

```

运行效果：



可以看到，我们在Directive中传入的数据进行的数据修改，并未反馈到FirstCtrl中。

在Directive中进行函数回调

上面我们介绍了等号 `=` 和 `@` 符号的使用方法，它们分别对应传入对象和文本。但是，如果我们期望传入一个回调函数呢？这样我们就可以实现如封装一个按钮为一个Directive，然后让它在点击后实现我们期望的功能的效果。

这就需要用到 `&` 符号，下面我们来看看实际的例子（这个例子比较复杂，请仔细分析研读）：

```

1. var App = angular.module("App", []);
2.
3. App.directive("formDirective", function () {
4.     return {

```

```

5.         restrict: "A",
6.         scope: {
7.             //这里使用&符号来接受传入的函数
8.             btnClick: "&"
9.             //注意:这里没有加入下方的value模型
10.        },
11.        template:
12.        //一个用于输入文字的输入框, 绑定到value上
13.        "<input type='text' ng-model='value'><br>" +
14.        //提交的按钮, 绑定上方scope的btnClick方法
15.        //注意传入参数的方式和HTML中具体使用的方式
16.        "<input type='button' value='提交' ng-
17.        click='btnClick({message:value})'>"
18.    });
19.
20. App.controller("FirstCtrl", function ($scope) {
21.     $scope.clickBtnCallback = function (msg) {
22.         alert("点击了按钮!信息是:" + msg);
23.     }
24. });

```

对应的HTML代码:

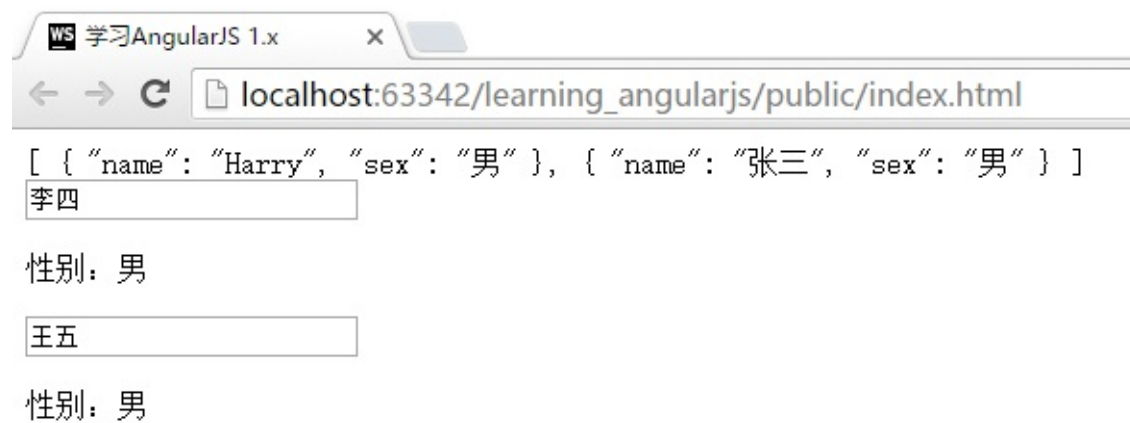
```

1. <!DOCTYPE html>
2. <html lang="zh" ng-app="App">
3. <head>
4.     <meta charset="UTF-8">
5.     <title>{{"学习AngularJS 1.x"}}</title>
6.     <link type="text/css" rel="stylesheet" href="css/style.css">
7. </head>
8. <body>
9. <div ng-controller="FirstCtrl">
10.     <!-- 注意这里绑定btn-click/btnClick中传入的参数命名 -->
11.     <div form-directive btn-click="clickBtnCallback(message)">
12.     </div>
13. </div>

```

```
14. <script type="text/javascript" src="components/angular/angular.js">
    </script>
15. <script type="text/javascript" src="js/app.js"></script>
16. </body>
17. </html>
```

运行结果：



使用templateUrl获取模板

- 使用 `templateUrl` 获取模板
 - `ng-template`
 - 使用函数获取templateUrl

使用 `templateUrl` 获取模板

有些时候，Directive中的模板 `template` 会变得很大，如果仍然放置在定义中，那么可能会造成阅读和修改不方便的情况。

针对这种情况，我们可以将 `template` 替换为 `templateUrl`，通过引入外部文件的形式来调用布局。

例如：

```
1. var App = angular.module("App", []);
2.
3. App.directive("formDirective", function () {
4.     return {
5.         restrict: "A",
6.         scope: {
7.         },
8.         templateUrl:"part.html"
9.     }
10. });
11.
12. App.controller("FirstCtrl", function ($scope) {
13.
14. });
```

```
1. <!DOCTYPE html>
2. <html lang="zh" ng-app="App">
3. <head>
```

```

4.     <meta charset="UTF-8">
5.     <title>{{"学习AngularJS 1.x"}}</title>
6.     <link href="css/style.css" rel="stylesheet">
7. </head>
8. <body>
9.
10. <div ng-controller="FirstCtrl">
11.     <div form-directive></div>
12. </div>
13.
14.
15. <script type="text/javascript" src="components/angular/angular.js">
    </script>
16. <script type="text/javascript" src="js/app.js"></script>
17. </body>
18. </html>

```

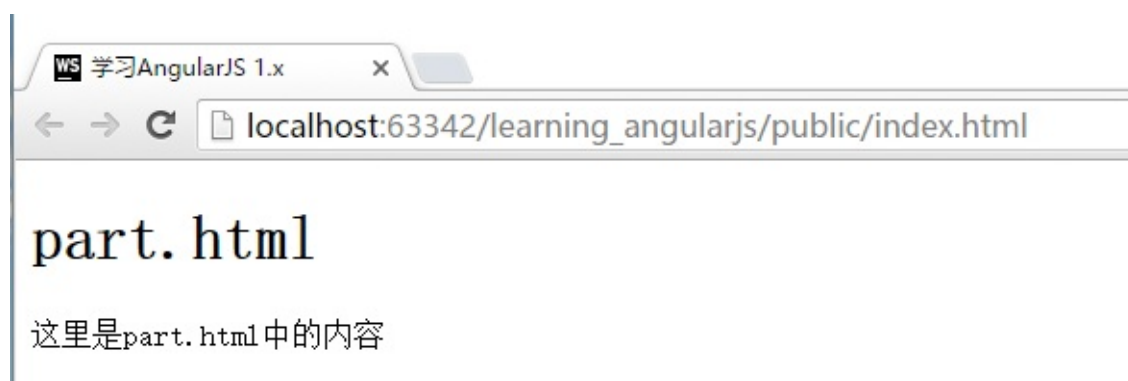
同时，我们还要加入一个新的html文档。为了演示，我们将新建的文档放置在和index.html同一个目录，命名为part.html：

```

1. <h1>part.html</h1>
2. <p>这里是part.html中的内容</p>

```

运行效果：



除了直接将HTML部件存储为独立的文件，我们也可以直接使用

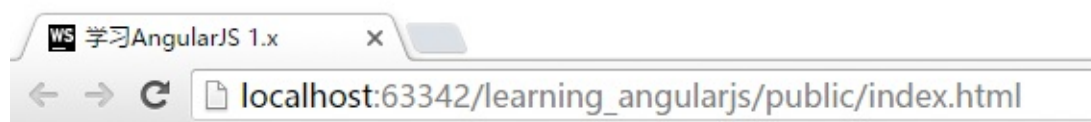
AngularJS提供的ng-template功能。这点在第四章中也有提到，这里是一个新的示例，帮助您对比物理文件和 `ng-template` 文件的优先级。

例如：

```
1. <!DOCTYPE html>
2. <html lang="zh" ng-app="App">
3. <head>
4.     <meta charset="UTF-8">
5.     <title>{{"学习AngularJS 1.x"}}</title>
6.     <link href="css/style.css" rel="stylesheet">
7. </head>
8. <body>
9.
10. <!-- 代码片段开始 -->
11. <script type="text/ng-template" id="part.html">
12.     <h1>通过ng-template封装的part.html</h1>
13.     <p>这里是part.html中的内容</p>
14. </script>
15. <!-- 代码片段结束 -->
16.
17. <div ng-controller="FirstCtrl">
18.     <div form-directive></div>
19. </div>
20.
21. <script type="text/javascript" src="components/angular/angular.js">
22.     </script>
23. <script type="text/javascript" src="js/app.js"></script>
24. </body>
25. </html>
```

这里，我们保留了上个例子中的所有文件（包括独立的part.html），对JavaScript也未进行任何修改。

运行效果如下：



通过ng-template封装的part.html

这里是part.html中的内容

要使用这个功能，我们需要在 `<script>` 标签中，首先声明 `type="text/ng-template"`，并给这段代码进行命名，示例代码中的 `id="part.html"` 即是这部分HTML代码片段的命名。使用时，直接使用 `part.html` 即可。

这样进行封装，与函数封装调用的概念类似。当默认HTML进行展示时，是不会显示这段代码的。而通过 `id` 调用后，又可以直接展示出来。

注意：通过示例我们可以看到，文档中的 `part.html` 的优先级高于独立的HTML文件。

使用函数获取templateUrl

templateUrl的特性与我们在第四章学习的ng-include类似，也支持通过函数来获取最终的url地址。例如：

```
1. App.directive("formDirective", function () {
2.     return {
3.         restrict: "A",
4.         scope: {},
5.         templateUrl: function () {
6.             return "part.html";
7.         }
8.     }
9. });
```

运行效果与上面的图片一致，就不重复展示了。

让Directive动起来link()

- 让Directive动起来 `link()`
 - `element`
 - `element` 与jQuery的关系
 - 在AngularJS中使用jQuery
 - 实际体验 `element` 的功能
 - 在 `element` 上绑定鼠标移入移出时的变化效果

让Directive动起来 `link()`

在前面几节中，我们都是在学习如何配置Directive。下面，我们将开始学习Directive最灵活的部分，`link()` 函数。

`link()` 函数的基本使用方法如下：

```

1. App.directive("formDirective", function () {
2.     return {
3.         restrict: "A",
4.         scope:{
5.             a:"",
6.             b:"",
7.             c=""
8.         },
9.         link: function(scope, element, attrs){
10.             console.log(scope);
11.             console.log(JSON.stringify(element));
12.             console.log(JSON.stringify(attrs));
13.         }
14.     }
15. });

```

`link()` 带有三个参数，`scope`，`element`，`attrs`。为了更好的了

解link函数三个参数的意义，函数中加上了日志输出的 `console.log` 的命令。

`scope` 未加上 `JSON.stringify` 转变为json格式输出，是因为 `scope` 本身涉及循环引用，因此无法转化为json。

对应的HTML代码：

```
1. <div form-directive a="1" b="2" c="3"></div>
```

在Chrome的“开发者工具”的控制台中，我们可以看到如下结果：

```
1. Scope
2. {$id: 3, $$childTail: null, $$childHead: null, $$prevSibling: null,
   $$nextSibling: null...}
```

```
1. //element
2. {"0":{"ng339":7},"length":1}
3.
4. //attrs
5. {
6.   "$attr":{
7.     "formDirective":"form-directive",
8.     "a":"a",
9.     "b":"b",
10.    "c":"c"
11.   },
12.   "$$element":[{"ng339":7},1],
13.   "formDirective":"",
14.   "a":"1",
15.   "b":"2",
16.   "c":"3"
17. }
```

在这里，我们最主要的是观察最下方的 `attrs` 部分。我们可以看到，在这个Directive元素中传入的数据，都可以通过 `attrs` 获取到，这

也是我们在 `link()` 中获取数据的主要方法。

`element`

从 `element` 被转义成json的文本中很难获取任何实质的信息，但是，这里才是整个Directive中的核心部分。

`element` 与jQuery的关系

在Directive中，我们不免需要对页面元素进行操作。为了提供这项功能，AngularJS几乎原版搬运了jQuery操作元素的功能，他们称之为“jQuery Lite”(jqLite)。

AngularJS通过jqLite的调用方法 `angular.element`，实现了jQuery中的大部分常用功能。也就是，我们可以在获取了 `element` 参数后，调用jQuery常用的语法，如 `bind()`，`addClass()`，`removeClass()` 等来直接对元素进行操作，实现我们期望的功能。

同时，如果你希望在AngularJS中直接使用完整的jQuery也是非常容易的。只需要安装jQuery，并在 `index.html` 中，保证在 `angular.js` 引入之前引入 `jquery.js`。AngularJS会自动将 `angular.element` 绑定到jQuery上。也就是，在Directive中，会自动使用jQuery来解析界面元素，我们获取的 `element` 会自动变为jQuery对象。

在AngularJS中使用jQuery

安装和使用jQuery共有两步：

- 通过 `bower` 安装jQuery

```
1. bower install jquery --save
```


- 在 `index.html` 中引入 `jquery.js`

```
1. <!-- 保证在angular.js之前引入jquery.js -->
2. <script type="text/javascript"
   src="components/jquery/dist/jquery.js"></script>
3. <script type="text/javascript" src="components/angular/angular.js">
   </script>
```

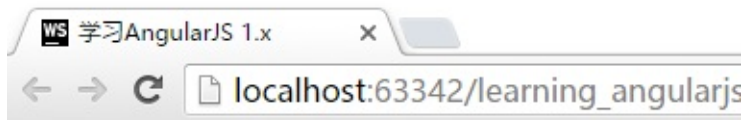
实际体验 `element` 的功能

我们在引入jQuery后，可以通过如下代码查看效果：

这里需要注意的是，使用*jqLite*的方法和以下代码中使用的方法是不一样的，因为*jqLite*不支持通过标签方式获取子元素。如果您有兴趣学习*jqLite*的使用方法（并且拥有一定的英文阅读能力），可以在[这里](#)查看官方文档。

```
1. App.directive("formDirective", function () {
2.     return {
3.         restrict: "A",
4.         template: "<h1>标题</h1><p>这里是段落文字</p>",
5.         link: function(scope, element, attrs){
6.             element.children("h1").addClass("strike");
7.         }
8.     }
9. });
```

```
1. <div ng-controller="FirstCtrl">
2.     <div form-directive></div>
3. </div>
```



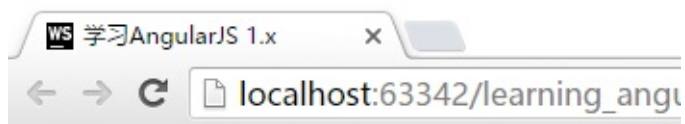
标题

这里是段落文字

在 `element` 上绑定鼠标移入移出时的变化效果

```
1. App.directive("formDirective", function () {
2.     return {
3.         restrict: "A",
4.         template: "<h1>标题</h1><p>这里是段落文字</p>",
5.         link: function(scope, element, attrs){
6.             element.children("h1").bind("mouseenter", function(){
7.                 element.children("h1").addClass("strike");
8.                 element.children("h1").text("鼠标移过来了");
9.             });
10.
11.             element.children("h1").bind("mouseleave", function(){
12.                 element.children("h1").removeClass("strike");
13.                 element.children("h1").text("鼠标移开了");
14.             })
15.         }
16.     }
17. });
```

运行时，当鼠标移动到标题上，则标题文字会变化成“鼠标移过来了”，并加上删除线效果；当鼠标移开，则文字会变为“鼠标移开了”。



~~鼠标移过来了~~

这里是段落文字

因为主要是jQuery的用法，更多的实际应用就不在本书中详述了。如果有兴趣学习jQuery，您可以通过上网搜索或者购买书籍的方式来学习。

把Directive变为一个容器transclude

- 把Directive变为一个容器 `transclude`

把Directive变为一个容器 `transclude`

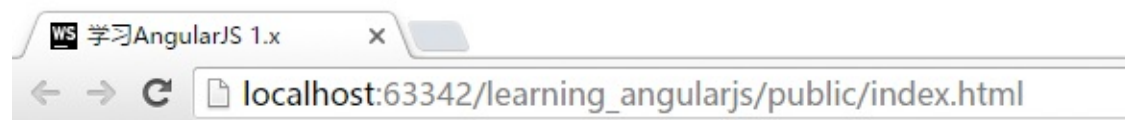
在前面我们使用到的Directive，都会将包含有Directive的元素整体替换为 `template` 中的内容。这样，就让Directive的用途缩减为只能封装最低级别的元素。

但是我们使用的 `ng-app`，`ng-controller` 等，也同样都是Directive，而我们可以在这些元素中，直接填入HTML代码。这是如何实现的呢？这就要应用到Directive的 `transclude` 属性。

```
1. App.directive("formDirective", function () {
2.     return {
3.         restrict: "A",
4.         //通过transclude标签将Directive变为一个容器
5.         transclude: true,
6.         //注意template中的ng-transclude，这里是放置原有代码的地方。
7.         template: "<h1>标题</h1><p>这里是段落文字</p><div ng-
      transclude></div>"
8.     }
9. });
```

```
1. <div ng-controller="FirstCtrl">
2.     <div form-directive>
3.         <p>这段文字是放置在Directive中间的。</p>
4.     </div>
5. </div>
```

运行结果：



标题

这里是段落文字

这段文字是放置在Directive中间的。

查看HTML代码如下：

```
▼ <body>
  ▼ <div ng-controller="FirstCtrl" class="ng-scope">
    ▼ <div form-directive>
      ▼ <div>
        <h1>标题</h1>
        <p>这里是段落文字</p>
        ▼ <div ng-transclude>
          <p class="ng-scope">这段文字是放置在Directive中间的。</p>
        </div>
      </div>
    </div>
  </div>
```

Directive之间互相通讯

- [Directive之间互相通讯](#)

Directive之间互相通讯

本节是Directive的高级使用方法之一，通过赋予Directive之间互相通讯的功能，我们可以将部件的抽象化提升到一个更高的层次。

本节将通过官方网站中一个较复杂的[例子](#)来讲解具体的使用方法，通过Directive的配置，将HTML中的代码自动抽取为一个Tab列表，点击Tab列表中的标题，则可自动的显示Tab中包含的内容。

在JavaScript代码中，我们声明了2个

Directive， `gqTabContainer` 和 `gqTabContent` 。

`gqTabContainer` 中加入了 `controller` 这项配置，并封装了 `panes` 用于存储数据， `$scope.select` 用于接收界面点击事件，以及一个 `addPane` 方法用于接收 `gqTabContent` 的调用。

`gqTabContent` 中加入了 `require` 这项配置，获取的对象作为第四个参数传入 `link()` 函数。

```

1. var App = angular.module("App", []);
2.
3. App.directive("gqTabContainer", function () {
4.     return {
5.         restrict: 'E',
6.         transclude: true,
7.         scope: {},
8.         //注意这里为tabContainer增加了一个controller，并引入了$scope
9.         controller: ['$scope', function ($scope) {
10.             var panes = $scope.panes = [];
11.

```

```

12.          //tab列表中项目被选中（点击）的处理函数
13.          $scope.select = function (pane) {
14.              angular.forEach(panes, function (pane) {
15.                  pane.selected = false;
16.              });
17.              pane.selected = true;
18.          };
19.
20.          //初始化页面时，供其他Directive调用的注册函数
21.          this.addPane = function (pane) {
22.              if (panes.length === 0) {
23.                  $scope.select(pane);
24.              }
25.              panes.push(pane);
26.          };
27.      }],
28.      //注意templateUrl的命名
29.      templateUrl: "gqTabList"
30.  };
31. });
32.
33. App.directive('gqTabContent', function () {
34.     return {
35.         //获取gqTabContainer这个Directive
36.         require: '^gqTabContainer',
37.         restrict: 'E',
38.         transclude: true,
39.         scope: {
40.             title: '@'
41.         },
42.         //第四个参数是获取到的Directive
43.         link: function (scope, element, attrs, tabContainer) {
44.             //调用了上方gqTabContainer的addPane()方法
45.             //注意参数命名，不需要和上方Directive一致
46.             tabContainer.addPane(scope);
47.         },
48.         //注意templateUrl的命名
49.         templateUrl: "gqTabContent"

```

```
50.     };
51. });
```

在HTML代码中，我们直接使用了 `gq-tab-container` 和 `gq-tab-content`。

```
1.  <!DOCTYPE html>
2.  <html lang="zh" ng-app="App">
3.  <head>
4.      <meta charset="UTF-8">
5.      <title>{{"学习AngularJS 1.x"}}</title>
6.      <link href="css/style.css" rel="stylesheet">
7.  </head>
8.  <body>
9.
10. <script type="text/javascript"
    src="components/jquery/dist/jquery.js"></script>
11. <script type="text/javascript" src="components/angular/angular.js">
    </script>
12. <script type="text/javascript" src="js/app.js"></script>
13.
14. <gq-tab-container>
15.     <gq-tab-content title="标签1">
16.         <h4>标题1</h4>
17.         <p>这是第一个标签下的内容</p>
18.     </gq-tab-content>
19.     <gq-tab-content title="标签2">
20.         <h4>标题2</h4>
21.         <p>这是第二个标签下的内容</p>
22.     </gq-tab-content>
23. </gq-tab-container>
24.
25. <!--这里为了将文件整理在一起，使用了ng-template的方式-->
26. <script type="text/ng-template" id="gqTabList">
27.
28.     <ul>
29.         <li ng-repeat="pane in panes" ng-class="
```



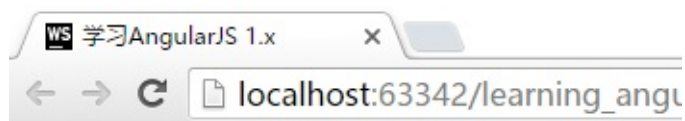
```

    {active:pane.selected}">
30.         <a href="" ng-click="select(pane)">{{pane.title}}</a>
31.     </li>
32. </ul>
33. <div ng-transclude></div>
34.
35. </script>
36.
37. <script type="text/ng-template" id="gqTabContent">
38.
39.     <div ng-show="selected" ng-transclude>
40.     </div>
41.
42. </script>
43.
44. </body>
45. </html>

```

运行效果：

- 当点击“标签1”或“标签2”时，下方会自动展示对应的内容。



- [标签1](#)
- [标签2](#)

标题1

这是第一个标签下的内容

本章总结

- [本章总结](#)

本章总结

本章讲述的内容是AngularJS的核心内容之一，并且对于之前前端开发经验不足的读者可能会造成一些学习上的困难。但是我建议各位读者好好的掌握本章的内容，这样在利用AngularJS构建网站时，思路更清晰更有条理。