

需要整本电子书

联系我QQ: 2667271557



图灵程序设计丛书



ng-book 2

The Complete Book on Angular 2

angular 权威教程

[美] Ari Lerner [巴西] Felipe Coury

[美] Nate Murray [巴西] Carlos Taborda 著

Nice Angular社区 译

强力的IDE支持+完善的生态圈+一套代码、多种平台+.....

= Angular

资深全栈开发工程师**经验总结**

雪狼组织的Nice Angular社区主力**倾情翻译**

Google**推荐阅读**

需要整本电子书，联系我QQ: 2667271557

需要整本电子书，联系我QQ：2667271557



图灵程序设计丛书



ng-book 2
The Complete Book on Angular 2

angular

权威教程

[美] Ari Lerner [巴西] Felipe Coury 著
[美] Nate Murray [巴西] Carlos Taborda
Nice Angular社区 译

人民邮电出版社
北 京

需要整本电子书，联系我QQ：2667271557

需要整本电子书，联系我QQ：2667271557

图书在版编目（CIP）数据

Angular权威教程 / (美) 阿里·勒纳 (Ari Lerner)
等著；Nice Angular社区译. -- 2版. -- 北京：人民
邮电出版社，2017.4
(图灵程序设计丛书)
ISBN 978-7-115-45158-3

I. ①A… II. ①阿… ②N… III. ①超文本标记语言
—程序设计—教材 IV. ①TP312.8

中国版本图书馆CIP数据核字(2017)第051224号

内 容 提 要

本书堪称 Angular 领域的里程碑式著作，涵盖了关于 Angular 的几乎所有内容。对于没有经验的人，本书平实、通俗的讲解，递进、严密的组织，可以让人毫无压力地登堂入室，迅速领悟新一代 Web 应用开发的精髓。如果你有相关经验，那本书对 Angular 概念和技术细节的全面剖析，以及引人入胜、切中肯綮的讲解，将帮助你彻底掌握这个框架，在自己职业技术修炼之路上更进一步。

本书的读者对象为所有想要理解和学习 Angular 的前端开发人员。

-
- ◆ 著 [美] Ari Lerner [巴西] Felipe Coury
[美] Nate Murray [巴西] Carlos Taborda
译 Nice Angular社区
责任编辑 杨琳
责任印制 彭志环
- ◆ 人民邮电出版社出版发行 北京市丰台区成寿寺路11号
邮编 100164 电子邮件 315@ptpress.com.cn
网址 <http://www.ptpress.com.cn>
北京印刷
- ◆ 开本：800×1000 1/16
印张：32
字数：756千字 2017年4月第2版
印数：14 301 - 18 300册 2017年4月北京第1次印刷
著作权合同登记号 图字：01-2017-0675号
-

定价：109.00元

读者服务热线：(010)51095186转600 印装质量热线：(010)81055316

反盗版热线：(010)81055315

广告经营许可证：京东工商广字第 8052 号

需要整本电子书，联系我QQ：2667271557

推 荐 序

很高兴这本《Angular权威教程》成为Angular中文资源的一部分，希望它能广受欢迎，给中国的Angular社区提供一份令人愉悦的学习资源，也希望它帮助更多工程师开始使用下一代Angular框架来开发应用。

我认识雪狼和他所属的Nice Angular社区是在2016年。那时候，他们开始了对Angular官方网站卓越的本地化工作。现在，这份中文官网已经部署在了angular.cn上。

本书及其翻译工作充分体现了中国开源软件开发者的热情和共享精神。感谢雪狼等来自Nice Angular社区的志愿者们对此作出的贡献。愿本书帮助你开始试用Angular！祝你成功！

Naomi Black，Google Angular项目经理兼主管

作为一项开源技术和前沿Web开发框架，Angular持续吸引着中国区开发人员的关注。作为雪狼及其所属Nice Angular社区的集体工作成果，这本书是开源力量的又一次证明，证明这种热情、这种志愿精神确实可以帮助业界享受到全球最新的开发技术。我谨代表Google开发技术推广部向这本书的出版表示祝贺。

栾跃，Google开发技术推广部大中华区主管

译者序

简介

以笔者之所见，《Angular权威教程》大概是目前除了Angular官方文档之外最全面的学习资料了，这从其英文版多达600多页的篇幅就可见一斑。相应地，它面对的对象涵盖了从入门级到高级的读者，是一本可以陪伴你成长的好书。

在内容安排上，本书具有大量的例子以保障其足够浅显，但也穿插着一些原理分析以保障其足够深入。除此之外，本书还给出了很多外部参考资料，让富有探险精神的你可以向专家级进发。

翻译说明

未来的版本号及发布计划

Angular就要出4.0了！是的，过一阵子还有Angular 5/6/7/8……这本书会很快过时吗？答案是“不会”。Angular开发组对于未来的版本号及发布计划有一个正式的说明，大意是：

我们要兼顾向后兼容和向前演进，因此以后我们将严格遵循SemVer语义化版本规范，并力求让版本升级变得可预测，以便使用者可以提前安排。在大版本号之间会出现少量破坏性变更，但是不用担心，相邻的大版本号之间只会把一些API标记为废弃的。也就是说，理想情况下，4的程序是可以直接迁移到5的，只是会收到一些API废弃提示，到6中才会彻底移除。同时，官方会在文档中给出详细的升级指南，帮助开发者升级。

因此，请放心，Angular以后绝不会出现像从1升级到2这么大的变化。事实上，NodeJS现在采用的就是类似的版本策略，提高发布的可预测性对于工程化开发是很有价值的。

另外，这里为什么没有3？简单点说就是因为路由模块比其他模块多发布过一次，因此当你使用core模块的2.0时，和它配套的router模块却是3.0的，这容易让开发人员困惑，跳过3，可以让所有模块的编号重新对齐。

对框架名称的说明

Angular开发组正式确定了新的命名策略：用AngularJS来代表1.x版本，而Angular代表2.x、4.x、5.x等很多后续版本，因为Angular 2+将支持TypeScript/JavaScript/Dart，而不再是JavaScript。这些变化已经在官方文档中体现出来了，而本书也将同样遵循这样的命名策略。

名词：装饰器与注解

@Component等语法元素在TypeScript中被称为装饰器（decorator），但在本书中，作者统一称其为注解（annotation）。这两种提法都是正确的。在语法层面，@Component确实是装饰器，这是TypeScript的标准叫法；但是在语义层面，Angular中是把它作为注解使用的。两者的区别是，装饰器直接改变被装饰者的行为，而注解则提供元数据，供框架去根据这些元数据做不同的处理。在Angular目前的版本中，@Component确实只是提供了元数据。

我们在跟原作者讨论之后，决定还是跟随作者的提法来翻译。不过在日常工作中，还是建议你遵循TypeScript的提法，将其称为装饰器。

支持与勘误

如果对本书中的一些概念不太理解，请参阅Angular官方中文站angular.cn。这里有来自官方开发组的权威资料。

如果对本书有任何疑问或发现问题，请到<https://github.com/nice-angular/ng-book-2>提交issue。

同时，对于一些经过确认的issue，我们也会更新在勘误区。

关于我们

参与本次翻译的一共有7位成员，都是AngularJS领域的专家和Angular领域的先行者。稍后会有我们的简短介绍。

本书各章的译者和校对者如下：

	翻译	一校	二校
第1章	雪狼、叶志敏	郑丰彧	郑丰彧
第2章	破狼	破狼	雪狼
第3章	张旋	张旋	雪狼
第4章	郑丰彧	郑丰彧	雪狼
第5章	破狼	破狼	雪狼
第6章	王子实	王子实	雪狼

(续)

	翻译	一校	二校
第7章	叶志敏	叶志敏	叶志敏
第8章	雪狼	雪狼	雪狼
第9章	郑丰彧	郑丰彧	雪狼
第10章	郑丰彧	郑丰彧	Hantsy
第11章	郑丰彧	郑丰彧	Hantsy
第12章	郑丰彧	郑丰彧	雪狼
第13章	郑丰彧	郑丰彧	雪狼
第14章	郑丰彧	郑丰彧	雪狼
第15章	Hantsy	Hantsy	叶志敏
第16章	雪狼	雪狼	张旋

除此之外，雪狼还承担了项目管理和中文统稿工作；破狼负责全书的技术准确性把关；叶志敏负责与作者沟通，并在英文理解方面进行把关。

我们的感恩

本书得以发行，首先要感谢Angular开发组及其项目经理Naomi Black。正是由于她的支持和牵线搭桥，才有了我们和图灵的这次合作。

我们还要感谢Google开发技术推广部及其大中华区主管栾跃和项目经理程路，正是由于他们的努力，让Angular在中国的推广普及工作有了正规军的加入，而本书的出版正是推广计划中的一小部分。

我们还要感谢图灵的编辑朱巍和杨琳，在整个翻译过程中，她们给了我们许多专业的指导和帮助。本书得以在迅速出版的同时保证高质量，她们的经验和把关居功甚伟。

最后，要感谢Angular中文社区。我所指的并不是由我们几个创建并管理的这些QQ群、微信群等，而是指广义的中文社区。无论你在北京还是上海，也无论你在国内还是海外；无论你是高手还是新兵，也无论你是否像我们一样是Angular的忠实粉丝，你们都是广义Angular中文社区中的一员。在我们的心中，只有一个Angular中文社区，她不被任何人拥有，也被每一个人拥有，因为她就是我们每个人。

固然，我们这几位译者都是推广Angular的志愿者与先行者，但我们真正希望看到的是一个繁荣、开放、互通的中文社区，是全球Angular社区的一部分，我们希望看到Angular的技术社区遍地开花。因此，如果你有自己的组织或影响力，请联系我们，我们愿与你携手共进，分享各种知识、渠道与资源，共同制定与推进社区发展计划。要知道，无论你将来是求职还是创业，一个繁荣的社区都会给你带来强力的支持。

一旦有了共同的愿景和开放、包容的文化，我们就能无视时空的阻隔，在天南海北守望相助，共同面对新技术的挑战与机遇。纷繁的世界、冰冷的技术与温暖的社区，共同构成了本书的出版背景。

雪狼的感恩

汪志成，网名雪狼。ThoughtWorker & Google开发者专家（GDE），拥有18年软件开发经验，崇尚简单、专业、分享，“好为人师，好为人师”；合著有《AngularJS深度剖析与最佳实践》。

首先，我要感谢我的家人，特别是我的妻子春娜。为了翻译官方文档和这本书，我失去了很多陪伴他们的时间，没有他们的支持，故事将无从开始。

其次，我要感谢ThoughtWorks，没有这样一个平台，我就无法安心钻研技术，更没有大量把新技术应用于工程实践中的机会。

最后，要特别感谢我刚刚出生的女儿，你是激励我前进的动力。闺女，看到了吗？这是老爹给你的迎新礼物。

破狼的感恩

格茸扎西，网名破狼。ThoughtWorks一线码农、架构师、咨询师；爱好读书和旅游，也常涂鸦一些技术博文；合著有《AngularJS深度剖析与最佳实践》；国内Angular最早布道者，Nice Angular社区“狼主”。

首先，要感谢我的妻子和父亲。因为他们的鼓励，我才能顺利完成本书相应章节的翻译。

其次，要感谢ThoughtWorks这个大家庭。因为在这个自组织和黑客文化环境的熏陶下，我才能潜心钻研这些技术。

最后，要感谢图灵出版社的朱巍编辑、本书的作者以及其他译者。

叶志敏的感恩

叶志敏，虽留英多年、远漂他乡、四处奔波，一颗热爱软件开发的心却依旧如初。多年前曾与雪狼共事，合作愉快，因此成为好朋友。由雪狼推荐进入Angular世界，使用Angular和.NET平台开发软件多年。从Alpha阶段开始使用Angular。与雪狼合作，翻译Angular官方文档站，并经过Angular团队的推荐，承接翻译本书的重任。

首先感谢我的妻子。从怀孕到照顾女儿健康成长，她一直对我的工作非常理解和支持，从无怨言。其次，感谢我母亲和岳母的慈爱与帮助。最后，希望女儿能健康成长，平安一生。

Hantsy 的感恩

Hantsy，拥有15年软件工程经验。2012年曾受JBoss（RedHat子公司）邀请前往波士顿参加JBoss用户和开发人员年度大会，并获得JBoss Community Recognition Awards。现为自由职业者，远程工作多年。

感谢Angular中文团队和图灵的支持，非常荣幸参与本书中文版的翻译。感谢Angular团队的努力，为我们带来如此优秀的工具框架。

张旋的感恩

张旋，PMP、ACP、NPDP，中科院计算所烟台分所集成应用中心主任。1982年生人，1996年起接触编程。正式从事软件工作行业11年。擅长项目管理、团队管理、技术体系建设。非常喜欢研究和对比各种新技术，生成适合工程使用的技术栈，并灌输到整个团队中去。

十分荣幸能成为Nice Angular社区的一员，感谢雪狼和破狼。感谢本书原作者为我们提供了一本这么好的Angular教程，也感谢本书的所有翻译者，从你们身上我确实学到了很多。感谢我的老婆莉莉，照看乐乐辛苦了，谢谢你给我时间让我做自己喜欢的事。最后感谢图灵出版社的朱巍编辑，本次合作非常愉快，期待下次更好的机会！

郑丰彧的感恩

郑丰彧，网名Z，现就职于大商集团天狗网，Angular爱好者，喜欢函数式编程、WebGL。

首先，我要感谢雪狼，一次很偶然的机会受到雪狼的邀请，让我受宠若惊，也为我开启了这次Angular翻译之旅。

其次，我要感谢我的家人，尤其是有孕在身的老婆和孕育中的宝宝。为了翻译这本书，我牺牲了很多原本用来陪伴你们的时间。

最后，我想对即将出世的女儿柚柚说句话：我们全家人对你的期待正如我们Nice Angular社区对此书的期待。所以，赶快“问世”吧！

王子实的感恩

王子实，现任光辉城市全栈工程师。1992年生，自学生时代便喜好编程，一直以来对各种新技术非常着迷，乐于对其进行研究与探索，并将成果在团队中进行推广，以提

升整体效率。

非常感谢雪狼能够给我这次机会参与到本书的翻译中来，能够让我对Angular社区尽一点点自己的绵薄之力。

同时也要感谢其他参与翻译的译者们，让我有了这次非常宝贵的经验。尤其是在翻译过程中遇到一些技术问题以及对原书内容有一些疑惑时，大家探究与实践的精神让我印象深刻。

还要感谢我的妻子默默给予我支持与理解。

最后，就是要感谢Google带给我们Angular这个强大而又好用的框架。希望它也能越来越好，不断进步！

目 录

第 1 章 编写你的第一个 Angular Web 应用..... 1

1.1 仿制 Reddit 网站.....	1
1.2 起步	3
1.2.1 TypeScript.....	3
1.2.2 angular-cli	3
1.2.3 示例项目	4
1.3 运行应用.....	7
1.3.1 制作 Component.....	8
1.3.2 导入依赖	9
1.3.3 Component 注解.....	10
1.3.4 用 templateUrl 添加模板	11
1.3.5 添加 template.....	11
1.3.6 用 styleUrls 添加 CSS 样式.....	12
1.3.7 加载组件	12
1.4 把数据添加到组件中.....	13
1.5 使用数组	15
1.6 使用 UserItemComponent 组件	18
1.6.1 渲染 UserItemComponent	18
1.6.2 接收输入	19
1.6.3 传入 Input 值.....	20
1.7 “启动”速成班	21
1.8 扩展你的应用	22
1.8.1 添加 CSS.....	24
1.8.2 应用程序组件.....	24
1.8.3 添加互动	26
1.8.4 添加文章组件.....	29
1.9 渲染多行.....	36
1.9.1 创建 Article 类.....	36
1.9.2 存储多篇文章.....	40

1.9.3 使用 inputs 配置

ArticleComponent.....	41
-----------------------	----

1.9.4 渲染文章列表

1.10 添加新文章.....

1.11 最后的修整.....

1.11.1 显示文章所属的域名

1.11.2 基于分数重新排序.....

1.12 全部代码.....

1.13 总结

1.14 获得帮助.....

第 2 章 TypeScript.....

2.1 Angular 是用 TypeScript 构建的

2.2 TypeScript 提供了哪些特性.....

2.3 类型

2.4 内置类型.....

2.4.1 字符串

2.4.2 数字.....

2.4.3 布尔类型

2.4.4 数组.....

2.4.5 枚举.....

2.4.6 任意类型

2.4.7 “无”类型

2.5 类.....

2.5.1 属性.....

2.5.2 方法.....

2.5.3 构造函数.....

2.5.4 继承.....

2.6 工具

2.6.1 胖箭头函数

2.6.2 模板字符串

2.7 总结

第 3 章 Angular 的工作原理	60	4.3 ngSwitch	92
3.1 应用	60	4.4 ngStyle	93
3.1.1 主导航组件	61	4.5 ngClass	95
3.1.2 面包屑导航组件	61	4.6 ngFor	98
3.1.3 产品列表组件	62	4.7 ngNonBindable	102
3.2 产品数据模型	64	4.8 总结	102
3.3 组件	64	第 5 章 Angular 中的表单	103
3.4 组件注解	66	5.1 表单——既重要，又复杂	103
3.4.1 组件 selector	66	5.2 FormControl 和 FormGroup	103
3.4.2 组件 template	67	5.2.1 FormControl	104
3.4.3 添加产品	67	5.2.2 FormGroup	104
3.4.4 用模板绑定来查看产品	68	5.3 我们的第一个表单	105
3.4.5 添加更多产品	69	5.3.1 加载 FormsModule	106
3.4.6 选择一个产品	70	5.3.2 简易 SKU 表单：@Component	107
3.4.7 用<products-list>列出产品	70	注解	107
3.5 产品列表组件	73	5.3.3 简易 SKU 表单：template	107
3.5.1 设置 ProductsList 的	73	5.3.4 简易 SKU 表单：组件定义类	110
@Component 配置项	73	5.3.5 试试看	110
3.5.2 组件的输入	74	5.4 使用 FormBuilder	111
3.5.3 组件的输出	77	5.5 响应式表单 FormBuilder	112
3.5.4 触发自定义事件	78	5.5.1 使用 FormBuilder	112
3.5.5 编写 ProductsList 的	79	5.5.2 在视图中使用 myForm	113
控制器类	79	5.5.3 试试看	114
3.5.6 编写 ProductsList 的视图模板	80	5.6 添加验证	115
3.5.7 完整的 ProductsList 组件	81	5.6.1 显式地把 sku 设置为实例	116
3.6 产品条目组件	83	变量	116
3.6.1 产品条目的组件配置	83	5.6.2 自定义验证器	120
3.6.2 产品条目组件的定义类	84	5.7 监听变化	121
3.6.3 产品条目组件的 template	84	5.8 ngModel	122
3.6.4 完整的 ProductRow 代码清单	85	5.9 总结	124
3.7 产品图片组件	85	第 6 章 HTTP	125
3.8 价格展示组件	86	6.1 简介	125
3.9 产品分类组件	87	6.2 使用 @angular/http	126
3.10 创建 NgModule 并启动应用	88	6.3 基本请求	127
3.11 完整的项目	89	6.3.1 构建 SimpleHTTPComponent 的	127
3.12 关于数据架构的一点说明	90	@Component	127
第 4 章 内置指令	91	6.3.2 构建 SimpleHTTPComponent 的	128
4.1 简介	91	template	128
4.2 ngIf	91		

6.3.3 构建 SimpleHTTPComponent 控制器	128	7.10 音乐搜索应用	168
6.3.4 完整的 SimpleHTTP- Component	130	7.10.1 首要步骤	169
6.4 编写 YouTubeSearchComponent	130	7.10.2 SpotifyService	170
6.4.1 编写 SearchResult	132	7.10.3 SearchComponent	171
6.4.2 编写 YouTubeService	132	7.10.4 尝试搜索	179
6.4.3 编写 SearchBox	140	7.10.5 TrackComponent	180
6.4.4 编写 SearchResult- Component	145	7.10.6 音乐搜索应用小结	182
6.4.5 编写 YouTubeSearch- Component	147	7.11 路由器钩子	182
6.5 @angular/http API	150	7.11.1 AuthService	183
6.5.1 发起一个 POST 请求	150	7.11.2 LoginComponent	184
6.5.2 PUT/PATCH/DELETE/HEAD	150	7.11.3 ProtectedComponent 组件 和路由守卫	186
6.5.3 RequestOptions	151	7.12 嵌套路由	190
6.6 总结	151	7.12.1 配置路由	191
第 7 章 路由	152	7.12.2 ProductsComponent 组件	191
7.1 为什么需要路由	152	7.13 总结	194
7.2 客户端路由的工作原理	153	第 8 章 依赖注入	195
7.2.1 初级阶段：使用锚标记	153	8.1 注入示例：PriceService	196
7.2.2 进化：HTML5 客户端路由	154	8.2 “别打给我们……”	197
7.3 编写第一个路由配置	155	8.3 依赖注入的部件	199
7.4 Angular 路由的组成部件	155	8.4 尝试注入器	200
7.4.1 导入	155	8.5 用 NgModule 提供依赖	201
7.4.2 路由配置	155	8.6 提供者	202
7.4.3 安装路由配置	156	8.6.1 使用类	202
7.4.4 使用<router-outlet>调用 RouterOutlet 指令	157	8.6.2 使用工厂	203
7.4.5 使用[routerLink]调用 routerLink 指令	158	8.6.3 使用值	205
7.5 整合	159	8.6.4 使用别名	205
7.5.1 创建组件	160	8.7 应用中的依赖注入	205
7.5.2 应用程序组件	161	8.8 使用注入器	207
7.5.3 配置路由	163	8.9 替换值	211
7.6 路由策略	164	8.10 NgModule	215
7.7 路径定位策略	165	8.10.1 NgModule 与 JavaScript 模块	215
7.8 运行应用程序	165	8.10.2 编译器与组件	215
7.9 路由参数	167	8.10.3 依赖注入与提供者	216
		8.10.4 组件可见性	217
		8.10.5 指定提供者	218
		8.11 总结	219
		第 9 章 Angular 数据架构	220

第 10 章 使用可观察对象的数据架构，

第 1 部分：服务 222

10.1 可观察对象和 RxJS 222

10.1.1 注意：一些必备的 RxJS

相关知识 222

10.1.2 学习响应式编程和 RxJS 223

10.2 聊天应用概览 224

10.2.1 组件 225

10.2.2 数据模型 226

10.2.3 服务 226

10.2.4 总结 226

10.3 实现数据模型 227

10.3.1 User 227

10.3.2 Thread 227

10.3.3 Message 228

10.4 实现 UserService 228

10.4.1 currentUser 流 229

10.4.2 设置新用户 230

10.4.3 UserService.ts 231

10.5 MessagesService 231

10.5.1 newMessages 流 231

10.5.2 messages 流 233

10.5.3 操作流模式 233

10.5.4 共享流 234

10.5.5 把 Message 对象添加到

messages 流中 235

10.5.6 完整的 MessagesService 238

10.5.7 试用 MessagesService 241

10.6 ThreadsService 242

10.6.1 当前一组 Thread 的映射

(threads 流) 242

10.6.2 按时间逆序排列的 Thread

列表 (orderedthreads

流) 246

10.6.3 当前已选的 Thread

(currentThread 流) 246

10.6.4 当前已选 Thread 的 Message

列表 (currentThread-

Messages 流) 248

10.6.5 完整的 ThreadsService 250

10.7 总结 251

第 11 章 使用可观察对象的数据架构，

第 2 部分：视图组件 252

11.1 构建视图：顶层组件 ChatApp 252

11.2 ChatThreads 组件 254

11.2.1 ChatThreads 控制器 255

11.2.2 ChatThreads 的 template 255

11.3 单个 ChatThread 组件 256

11.3.1 ChatThread 控制器和

ngOnInit 257

11.3.2 ChatThread 的 template 258

11.3.3 ChatThread 的完整代码 258

11.4 ChatWindow 组件 259

11.4.1 ChatWindow 组件类属性 260

11.4.2 ChatWindow 的 ngOnInit 261

11.4.3 ChatWindow 的 send-

Message 261

11.4.4 ChatWindow 的 onEnter 262

11.4.5 ChatWindow 的 scrollTo-

Bottom 262

11.4.6 ChatWindow 的 template 263

11.4.7 处理键盘动作 264

11.4.8 使用 ngModel 264

11.4.9 点击 Send 按钮 265

11.4.10 完整的 ChatWindow 组件 265

11.5 ChatMessage 组件 267

11.5.1 设置 incoming 属性 268

11.5.2 ChatMessage 的 template 268

11.5.3 完整的 ChatMessage 代码

清单 270

11.6 ChatNavBar 组件 273

11.6.1 ChatNavBar 的 @Component 273

11.6.2 ChatNavBar 控制器 273

11.6.3 ChatNavBar 的 template 274

11.6.4 完整的 ChatNavBar 组件 275

11.7 总结 276

11.8 更进一步 277

第 12 章 基于 TypeScript 的 Redux	
简介	278
12.1 Redux	279
12.2 Redux 核心概念	280
12.2.1 reducer 是什么	280
12.2.2 定义 Action 和 Reducer 的接口	281
12.2.3 创建第一个 Reducer	281
12.2.4 运行第一个 Reducer	282
12.2.5 使用 action 调整计数器	283
12.2.6 reducer 的 switch	284
12.2.7 action 的“参数”	285
12.3 保存 state	286
12.3.1 使用 store	287
12.3.2 使用 subscribe 进行通知	287
12.3.3 Redux 核心	290
12.4 消息应用	291
12.4.1 消息应用的 state	291
12.4.2 消息应用的 action	292
12.4.3 消息应用的 reducer	292
12.4.4 试用 action	295
12.4.5 action creator	296
12.4.6 使用真正的 Redux	297
12.5 在 Angular 中使用 Redux	299
12.6 规划应用	299
12.7 组建 Redux	300
12.7.1 定义应用的 state	300
12.7.2 定义 reducer	301
12.7.3 定义 action creator	301
12.7.4 创建 store	302
12.8 CounterApp 组件	303
12.9 提供 store	304
12.10 启动应用	305
12.11 CounterComponent	306
12.11.1 import	306
12.11.2 模板	306
12.11.3 constructor	307
12.11.4 整合	308
12.12 更进一步	310
12.13 参考资源	310
第 13 章 在 Angular 中引入 Redux	312
13.1 阅读背景	312
13.2 聊天应用概览	313
13.2.1 组件	313
13.2.2 数据模型	314
13.2.3 reducer	315
13.2.4 总结	315
13.3 实现数据模型	315
13.3.1 User	315
13.3.2 Thread	316
13.3.3 Message	316
13.4 应用的 state	316
13.4.1 关于代码布局	317
13.4.2 根 reducer	317
13.4.3 UserState	318
13.4.4 ThreadsState	318
13.4.5 可视化 AppState	319
13.5 构建 reducer (和 action creator)	321
13.5.1 设置当前用户的 action creator	321
13.5.2 UsersReducer: 设置当前用户	321
13.5.3 会话和消息概览	322
13.5.4 添加新会话的 action creator	322
13.5.5 添加新会话的 reducer	323
13.5.6 添加新消息的 action creator	324
13.5.7 添加新消息的 reducer	325
13.5.8 选择会话的 action creator	326
13.5.9 选择会话的 reducer	327
13.5.10 reducer 总结	328
13.6 构建 Angular 聊天应用	328
13.6.1 顶层组件 ChatApp	330
13.6.2 ChatPage	330
13.6.3 容器型组件与展示型组件	331
13.7 构建 ChatNavBar	332

13.7.1	Redux 选择器	334	14.4	查询相邻的指令：编写标签页	366
13.7.2	会话选择器	334	14.4.1	Tab 组件	367
13.7.3	未读消息总数选择器	336	14.4.2	Tabset 组件	367
13.8	构建 ChatThreads 组件	336	14.4.3	使用 Tabset	369
13.8.1	ChatThreads 控制器	337	14.5	生命周期钩子	370
13.8.2	ChatThreads 的 template	338	14.5.1	OnInit 和 OnDestroy	371
13.9	单个 ChatThread 组件	338	14.5.2	OnChanges	374
13.10	构建 ChatWindow 组件	340	14.5.3	DoCheck	378
13.10.1	ChatWindow 的 update- State()	341	14.5.4	AfterContentInit、 AfterViewInit、 AfterContentChecked 和 AfterViewChecked	386
13.10.2	ChatWindow 的 scrollToBottom()	342	14.6	高级模板	391
13.10.3	ChatWindow 的 sendMessage	342	14.6.1	重写 ngIf: ngBookIf	392
13.10.4	ChatWindow 的 onEnter	343	14.6.2	重写 ngFor: ngBook- Repeat	394
13.10.5	ChatWindow 的 template	343	14.7	变更检测	398
13.10.6	处理键盘动作	345	14.7.1	自定义变更检测	401
13.10.7	使用 ngModel	345	14.7.2	Zones	405
13.10.8	点击 Send 按钮	345	14.7.3	可观察对象和 OnPush	406
13.11	ChatMessage 组件	345	14.8	总结	409
13.11.1	设置 incoming 属性	346	第 15 章	测试	410
13.11.2	ChatMessage 的 template	346	15.1	测试驱动?	410
13.12	总结	347	15.2	端对端测试与单元测试	411
第 14 章	高级组件	349	15.3	测试工具	411
14.1	样式	349	15.3.1	Jasmine	411
14.1.1	视图 (样式) 封装	351	15.3.2	Karma	412
14.1.2	Shadow DOM 封装	354	15.4	编写单元测试	412
14.1.3	不使用封装	355	15.5	Angular 单元测试框架	412
14.2	创建 popup 指令：引用并修改宿主 元素	357	15.6	测试前准备	413
14.2.1	popup 指令的结构	357	15.7	测试服务类和 HTTP	415
14.2.2	使用 ElementRef	359	15.7.1	HTTP 要点	416
14.2.3	绑定到 host 属性	360	15.7.2	伪装	417
14.2.4	添加按钮并使用 exportAs	362	15.7.3	模拟	417
14.3	使用内容投影创建消息面板	363	15.7.4	Http MockBackend	418
14.3.1	改变 host 属性的 CSS 类	364	15.7.5	TestBed.configureTes- tingModule 和提供者	418
14.3.2	使用 ng-content	364	15.7.6	测试 getTrack 方法	419

15.8	测试组件间的路由	424	16.5.6	AngularJS: HomeComponent- ller 模板	461
15.8.1	为测试创建路由器	424	16.5.7	AngularJS: pin 指令	462
15.8.2	模拟依赖	427	16.5.8	AngularJS: pin 指令模板	462
15.8.3	探子	427	16.5.9	AngularJS: AddContro- ller	463
15.9	回到测试代码	429	16.5.10	AngularJS: AddContro- ller 模板	465
15.9.1	fakeAsync 和 advance	431	16.5.11	AngularJS: 总结	467
15.9.2	inject	432	16.6	构建混合式应用	468
15.9.3	测试 ArtistComponent 组件 初始化	432	16.6.1	混合式应用的结构	468
15.9.4	测试 ArtistComponent 方法	433	16.6.2	引导混合式应用	471
15.9.5	测试 ArtistComponent DOM 模板值	434	16.6.3	我们要升级什么	473
15.10	测试表单	436	16.6.4	插一小段内容: 类型文件	475
15.10.1	创建一个 ConsoleSpy	438	16.6.5	写 Angular 的 PinControls- Component	479
15.10.2	安装 ConsoleSpy	439	16.6.6	使用 Angular 的 PinCon- trolsComponent	481
15.10.3	配置测试模块	439	16.6.7	把 Angular 的 PinControls- Component 降级到 AngularJS	482
15.10.4	测试表单	440	16.6.8	用 Angular 添加图钉	483
15.10.5	重构表单测试	441	16.6.9	把 AngularJS 的 PinsSer- vice 和 \$state 升级到 Angular	484
15.11	测试 HTTP 请求	444	16.6.10	写 Angular 版的 AddPin- Component	485
15.11.1	测试 POST 方法	445	16.6.11	使用 AddPinComponent	490
15.11.2	测试 DELETE 方法	446	16.6.12	把 Angular 的服务暴露给 AngularJS	490
15.11.3	测试 HTTP 头	447	16.6.13	实现 AnalyticsService	491
15.11.4	测试 YouTubeService	448	16.6.14	把 Angular 的 Analytics- Service 降级到 AngularJS	491
15.12	总结	452	16.6.15	在 AngularJS 中使用 AnalyticsService	492
第 16 章	把 AngularJS 应用升级到 Angular	453	16.7	总结	493
16.1	周边概念	453	16.8	参考资源	493
16.2	我们要构建什么	454			
16.3	把 AngularJS 映射到 Angular	455			
16.4	关于互操作性的需求	456			
16.5	AngularJS 应用	456			
16.5.1	AngularJS 应用的 HTML	458			
16.5.2	代码概览	458			
16.5.3	AngularJS: PinsService	459			
16.5.4	AngularJS: 配置路由	460			
16.5.5	AngularJS: HomeComponent- ller	461			

需要整本电子书，联系我QQ：2667271557

需要整本电子书，联系我QQ：2667271557

第 1 章

编写你的第一个Angular Web应用

1.1 仿制 Reddit 网站

在本章中，我们将构建一个应用，它能让用户发表推荐文章（包括标题和URL）并对每篇文章投票。

你可以把该应用看作类似于Reddit^①或Product Hunt^②的起步版网站。

这个简单的应用将涵盖Angular中的大部分基本要素，包括：

- ❑ 构建自定义组件；
- ❑ 从表单中接收用户输入；
- ❑ 把对象列表渲染到视图中；
- ❑ 拦截用户的点击操作，并据此作出反应。

读完本章之后，你将掌握如何构建基本的Angular应用。

图1-1展示了该应用最终完成后的界面截图。

首先，用户将提交一个新的链接。之后，其他用户可以对每篇文章投票：“顶”或“踩”。每个链接都有一个最终得票数，我们可以对自己认为有用的链接投票（如图1-2所示）。

① <http://reddit.com>

② <http://producthunt.com>

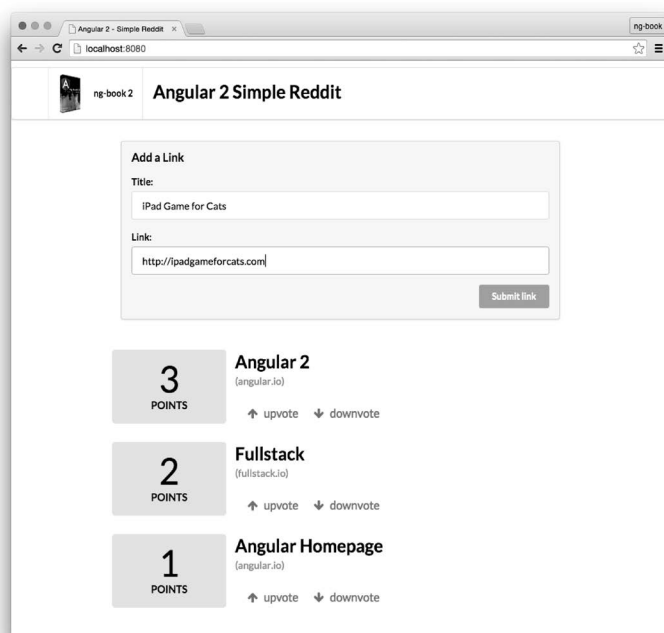


图1-1 完成后的应用

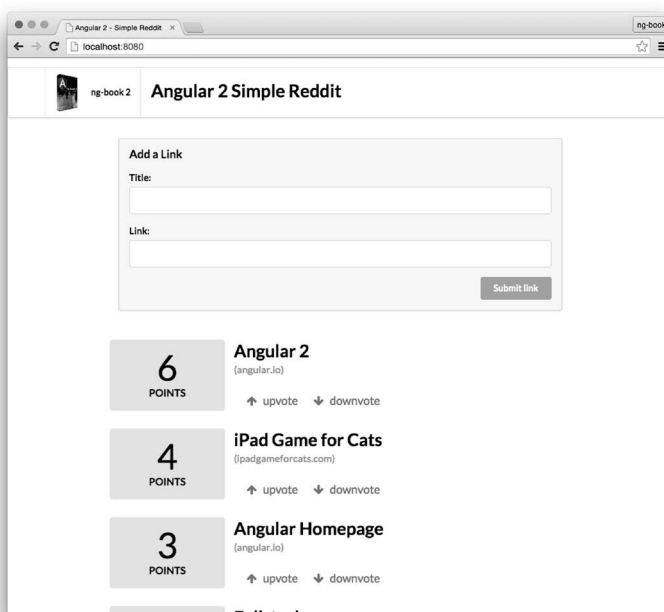


图1-2 包含新文章的应用

在本项目和整本书中，我们都将使用TypeScript。TypeScript是JavaScript ES6版的一个超集，增加了类型支持。本章不会深入讲解TypeScript；如果你熟悉ES5（“普通”的JavaScript）或ES6（ES2015），那么在后续的学习过程中应该不会有什问题。

在第2章中，我们将更深入地学习TypeScript。因此，即使你对某些新语法不太熟悉，也不必担心。

1.2 起步

1.2.1 TypeScript

要开始使用TypeScript，首先需要安装Node.js。安装方式很多，请参见Node.js官方网站（<https://nodejs.org/download/>）了解详情。



我必须用TypeScript吗？并非如此！要使用Angular，TypeScript并不是必需的，但它可能是最好的选择。Angular也有一套ES5 API，但Angular本身就是用TypeScript写成的，所以人们一般也会选用它。本书也将使用TypeScript，因为它确实很棒，能让Angular写起来更简单。当然，并不是非它不可。

安装完Node.js，接着就要安装TypeScript了。请确保安装1.7或更高的版本。要想安装它，请运行下列npm命令：

```
$ npm install -g typescript
```



通常，npm是Node.js的一部分。如果你的系统中没有npm命令，请确认你安装的Node.js是包含它的版本。



Windows用户：我们将在全书中使用Linux/Mac风格的命令行。强烈建议你安装Cygwin^①。借助它，你就能直接运行本书中的这些命令了。

1.2.2 angular-cli

Angular提供了一个命令行工具angular-cli，它能让用户通过命令行创建和管理项目。它自动化了一系列任务，比如创建项目、添加新的控制器等。多数情况下，选用angular-cli都是明

^① <https://www.cygwin.com/>

智的决定。当你创建和维护应用时，它能帮你遵循很多常用模式。

要想安装angular-cli，只要运行下列命令即可：

```
$ npm install -g angular-cli@1.0.0-beta.18
```

安装完毕之后，你就可以在命令行中用ng命令运行它了。运行ng命令时，你会看到一大堆输出，不过不用管它；往回滚屏，你会看到如下内容：

```
$ ng
Could not start watchman; falling back to NodeWatcher for file system events.
Visit http://ember-cli.com/user-guide/#watchman for more info.
Usage: ng <command> (Default: help)>
```

之所以得到这一大堆输出，是因为当我们不带参数运行ng命令时，它就会执行默认的help命令。help命令会解释如何使用本工具。

如果你在OS X或Linux上运行，可能还会在输出中看到这一行：

```
Could not start watchman; falling back to NodeWatcher for file system events.
```

这意味着我们没有安装过一个名叫watchman的工具。此工具能帮助angular-cli监听文件系统的变化。如果你在OS X上运行，建议使用Homebrew工具安装它，命令如下：

```
$ brew install watchman
```



如果你是OS X用户并且运行这个brew命令时出现错误，那么表示你尚未正确安装Homebrew工具。请参阅<http://brew.sh/>来安装它，然后再试一次。

如果你是Linux用户，可以参阅<https://ember-cli.com/user-guide/#watchman>来学习如何安装watchman。

如果你是Windows用户，那么不必安装任何东西，angular-cli将使用原生的Node.js文件监视器。

现在你应该已经装好angular-cli及其依赖了。在本章中，我们就用它来创建第一个应用。

1.2.3 示例项目

现在，环境已经准备好了，我们这就来编写第一个Angular应用吧！

打开终端窗口并且运行ng new命令，快速创建一个新的项目：

```
$ ng new angular2_hello_world
```

运行之后，你将看到下列输出：

```
installing ng 2
create .editorconfig
create README.md
```

```
create src/app/app.component.css
create src/app/app.component.html
create src/app/app.component.spec.ts
create src/app/app.component.ts
create src/app/app.module.ts
create src/app/index.ts
create src/app/shared/index.ts
create src/assets/.gitkeep
create src/assets/.npmignore
create src/environments/environment.dev.ts
create src/environments/environment.prod.ts
create src/environments/environment.ts
create src/favicon.ico
create src/index.html
create src/main.ts
create src/polyfills.ts
create src/styles.css
create src/test.ts
create src/tsconfig.json
create src/typings.d.ts
create angular-cli.json
create e2e/app.e2e-spec.ts
create e2e/app.po.ts
create e2e/tsconfig.json
create .gitignore
create karma.conf.js
create package.json
create protractor.conf.js
create tslint.json
Successfully initialized git.
□ Installing packages for tooling via npm
```

它将运行一段时间，进行npm依赖的安装。一旦安装结束，我们会看到一条成功信息：

```
Installed packages for tooling via npm.
```

这里生成了很多文件！现在不用关心它们都是什么。我们会在本书中讲解每一个文件的含义和用途。不过现在，我们先把注意力集中在如何用Angular代码开始工作上。

进入ng命令创建的angular2_hello_world目录，来看看它里面都有什么：

```
$ cd angular2_hello_world
$ tree -F -L 1
.
├── README.md           // an useful README
├── angular-cli.json    // angular-cli configuration file
├── e2e/                // end to end tests
├── karma.conf.js       // unit test configuration
├── node_modules/       // installed dependencies
├── package.json        // npm configuration
├── protractor.conf.js  // e2e test configuration
├── src/                // application source
└── tslint.json         // linter config file
```

6 第1章 编写你的第一个 Angular Web 应用

3 directories, 6 files

我们目前关注的目录是src，应用代码就在里面。下面看看我们在那里创建了什么：

```
$ cd src
$ tree -F
.
|-- app/
|   |-- app.component.css
|   |-- app.component.html
|   |-- app.component.spec.ts
|   |-- app.component.ts
|   |-- app.module.ts
|   |-- index.ts
|   `-- shared/
|       `-- index.ts
|-- assets/
|-- environments/
|   |-- environment.dev.ts
|   |-- environment.prod.ts
|   `-- environment.ts
|-- favicon.ico
|-- index.html
|-- main.ts
|-- polyfills.ts
|-- styles.css
|-- test.ts
|-- tsconfig.json
`-- typings.d.ts
```

4 directories, 18 files

用你惯用的文本编辑器打开index.html，应该会看到如下代码。

code/first_app/angular2_hello_world/src/index.html

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Angular2HelloWorld</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root>Loading...</app-root>
</body>
</html>
```

我们把它分解一下。

code/first_app/angular2_hello_world/src/index.html

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Angular2HelloWorld</title>
  <base href="/">
```

如果你熟悉HTML，这第一部分就很平淡无奇了。我们在这里声明了页面的字符集（charset）、标题（title）和基础URL（base href）。

code/first_app/angular2_hello_world/src/index.html

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

如果你继续深入模板主体（body），就会看到下列代码。

code/first_app/angular2_hello_world/src/index.html

```
<app-root>Loading...</app-root>
</body>
</html>
```

我们的应用将会在app-root标签处进行渲染，稍后剖析源代码的其他部分时还会看到它。文本Loading...是一个占位符，在应用代码加载之前会显示它。我们可以借助此技巧来通知用户该应用正在加载，可以像这里一样显示一条消息，也可以显示一个加载动画或其他形式的进度通知。

之后就可以编写应用代码了。

1.3 运行应用

在开始修改之前，我们先把这个自动生成的初始应用加载到浏览器中。angular-cli有一个内建的HTTP服务器，我们可以用它来启动应用。回到终端，进入应用的根目录（在本应用中是./angular2_hello_world目录）并运行命令。

```
$ ng serve
** NG Live Development Server is running on http://localhost:4200. **
// a bunch of debug messages

Build successful - 1342ms.
```

我们的应用正在localhost的4200端口上运行。打开浏览器并访问http://localhost:4200，结果如图1-3所示。



注意，如果4200端口由于某种原因被占用了，也可以在其他端口号上启动。仔细阅读你电脑上的输出信息，找出开发服务器的实际URL。

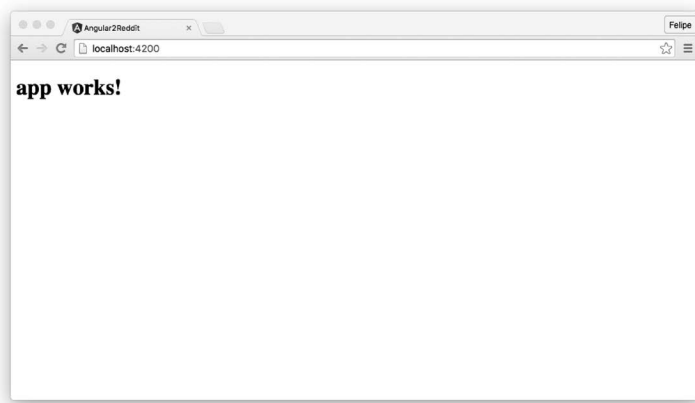


图1-3 运行中的应用

好，现在我们设置好了应用，而且知道了该如何运行它，可以开始写代码了。

1.3.1 制作 Component

Angular背后的指导思想之一就是组件化。

在Angular应用中，我们写HTML标记并把它变成可交互的应用。不过浏览器只认识一部分标签，比如<select>、<form>和<video>等，它们的功能都是由浏览器的开发者预先定义好的。

如果我们想教浏览器认识一些新标签，该怎么办呢？比如我们想要一个<weather>标签，用来显示天气；又比如想要一个<login>标签，用来创建一个登录面板。

这就是组件化背后的基本思想：我们要教浏览器认识一些拥有自定义功能的新标签。



如果你用过AngularJS，那么可以把组件当作新版本的指令。

让我们来创建第一个组件。写完该组件之后，就能在HTML文档中使用它了，就像这样：

```
<app-hello-world></app-hello-world>
```

要使用angular-cli来创建新组件，可以使用generate（生成）命令。

要生成hello-world组件，我们需要运行下列命令：

```
$ ng generate component hello-world
installing component
  create src/app/hello-world/hello-world.component.css
  create src/app/hello-world/hello-world.component.html
  create src/app/hello-world/hello-world.component.spec.ts
  create src/app/hello-world/hello-world.component.ts
```

那该怎么定义一个新组件呢？最基本的组件包括两部分：

(1) Component注解

(2) 组件定义类

下面来看看组件的代码，然后逐一讲解。打开第一个TypeScript文件：src/app/hello-world/hello-world.component.ts。

code/first_app/angular2_hello_world/src/app/hello-world/hello-world.component.ts

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-hello-world',
  templateUrl: './hello-world.component.html',
  styleUrls: ['./hello-world.component.css']
})
export class HelloWorldComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

}
```



注意，TypeScript文件的后缀是.ts而不是.js。问题在于浏览器并不知道该如何解释TypeScript文件。为了解决这个问题，ng serve命令会自动把.ts文件编译为.js文件。

这个代码片段乍一看可能有点恐怖，但别担心，我们接下来就会一步步讲解它。

1.3.2 导入依赖

import语句定义了我们写代码时要用到的那些模块。这里我们导入了两样东西：Component和OnInit。

我们从"@angular/core"模块中导入了组件（import Component）。"@angular/core"部分告诉程序到哪里查找所需的这些依赖。这个例子中，我们告诉编译器：“@angular/core”定义并导出了两个JavaScript/TypeScript对象，名字分别是Component和OnInit。

同样，我们还从这个模块中导入了OnInit（import OnInit）。稍后你就会知道，OnInit能帮我们在组件的初始化阶段运行某些代码。不过现在先不用管它。

注意这个import语句的结构是import { things } from wherever格式。我们把{ things }

这部分的写法叫作解构。解构是由ES6和TypeScript提供的一项特性，下一章会深入讲解。

import的用法很像Java中的import或Ruby中的require：从另一个模块中拉取这些依赖，并且让这些依赖在当前文件中可用。

1.3.3 Component 注解

导入依赖后，我们还要声明该组件。

```
code/first_app/angular2_hello_world/src/app/hello-world/hello-world.component.ts
```

```
@Component({
  selector: 'app-hello-world',
  templateUrl: './hello-world.component.html',
  styleUrls: ['./hello-world.component.css']
})
```

如果你习惯用JavaScript编程，那么下面这段代码可能看起来有点怪异：

```
@Component({
  // ...
})
```

这是什么？如果你有Java开发背景，应该会很熟悉：它们是注解。



AngularJS的依赖注入技术在幕后使用了注解的概念。也许你还不熟悉它们，但注解其实是让编译器为代码添加功能的途径之一。

我们可以把注解看作添加到代码上的元数据。当在HelloWorld类上使用@Component时，就把HelloWorld“装饰”（decorate）成了一个Component。

这个<app-hello-world>标签表示我们希望在HTML中使用该组件。要实现它，就得配置@Component并把selector指定为app-hello-world。

```
@Component({
  selector: 'app-hello-world'
  // ... more here
})
```

有很多种方式来配置选择器（selector），类似于CSS选择器、XPath或JQuery选择器。Angular组件对选择器的混用方式添加了一些特有的限制，稍后会谈到。现在，只要记住我们正在定义一个新的HTML标签就可以了。

这里的selector属性用来指出该组件将使用哪个DOM元素。如果模板中有<app-hello-world></app-hello-world>标签，就用该Component类及其组件定义信息对其进行编译。

1.3.4 用 templateUrl 添加模板

1

在这个组件中，我们把templateUrl指定为./hello-world.component.html。这意味着我们将从与该组件同目录的hello-world.component.html文件中加载模板。下面来看看这个文件。

code/first_app/angular2_hello_world/src/app/hello-world/hello-world.component.html

```
<p>
  hello-world works!
</p>
```

这里定义了一个p标签，其中包含了一些简单的文本。当Angular加载该组件时，就会读取此文件的内容作为组件的模板。

1.3.5 添加 template

我们有两种定义模板的方式：使用@Component对象中的template属性；指定templateUrl属性。

我们可以通过传入template选项来为@Component添加一个模板：

```
@Component({
  selector: 'app-hello-world',
  template: `
    <p>
      hello-world works inline!
    </p>
  `
})
```

注意，我们在反引号中（`...`）定义了template字符串。这是ES6中的一个新特性（而且很棒），允许使用多行字符串。使用反引号定义多行字符串，可以让我们更轻松地把模板放到代码文件中。



你真的应该把模板放进代码文件中吗？ 答案是：视情况而定。在很长一段时间里，大家都觉得最好把代码和模板分开。这对于一些开发团队来说确实更容易，不过在某些项目中会增加成本，因为你将不得不在一大堆文件之间切换。个人观点：如果模板行数短于一页，我更倾向于把模板和代码放在一起（也就是.ts文件中）。这样就能同时看到逻辑和视图部分，同时也便于理解它们之间如何互动。把视图和代码内联在一起的最大缺点是，很多编辑器仍然不支持对内部HTML字符串进行语法高亮。我们期待能尽快看到有更多编辑器支持对模板字符串内嵌HTML的语法高亮。

1.3.6 用 styleUrls 添加 CSS 样式

注意styleUrls属性：

```
styleUrls: ['./hello-world.component.css']
```

这段代码的意思是，我们要使用hello-world.component.css文件中的CSS作为该组件的样式。Angular使用一项叫作样式封装（style-encapsulation）的技术，它意味着在特定组件中指定的样式只会应用于该组件本身。14.1节会深入讨论它。

目前还用不到任何“组件局部样式”，你只要先知道它就行了（或整体删除此属性）。



你可能注意到了该属性与template有个不同点：它接收一个**数组**型参数。这是因为我们可以为同一个组件加载多个样式表。

1.3.7 加载组件

现在，我们已经写完了第一个组件的代码，那该如何把它加载到页面中呢？

如果再次在浏览器中访问此应用，我们会看到一切照旧。这是因为我们仅仅创建了该组件，但还没有使用它。

为了解决这一点，需要把该组件的标签添加到一个将要渲染的模板中去。打开文件first_app/angular2_hello_world/src/app/app.component.html。

记住，因为我们为HelloWorldComponent配置了app-hello-world选择器，所以要在模板中使用<app-hello-world></app-hello-world>。让我们把<app-hello-world>标签添加到app.component.html中。

code/first_app/angular2_hello_world/src/app/app.component.html

```
<h1>
  {{title}}

  <app-hello-world></app-hello-world>
</h1>
```

现在，刷新该页面就会看到如图1-4所示结果。

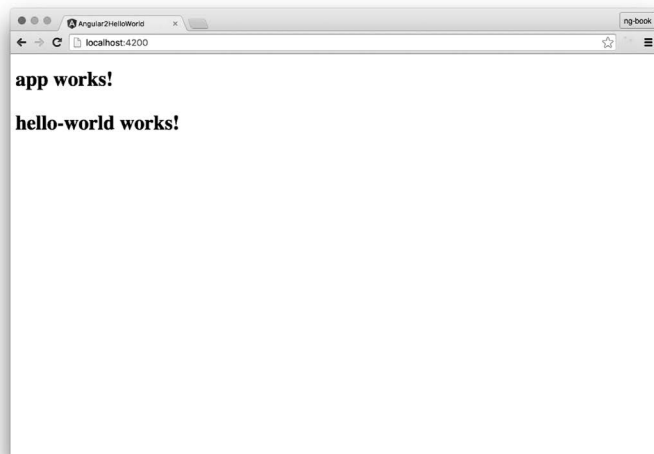


图1-4 “Hello world” 一切正常

工作正常！

1.4 把数据添加到组件中

现在，该组件渲染了一个静态模板。这表示我们的组件还不够有趣。

设想有一个应用会显示一个用户列表，并且我们想在其中显示用户的名字。在渲染整个列表之前，需要先渲染一个单独的用户。因此，我们来创建一个新的组件，它将显示用户的名字。

再次使用ng generate命令：

```
ng generate component user-item
```

记住，想看到我们创建好的组件，就要把它添加到一个模板中。

让我们把app-user-item标签添加到app.component.html中，以便看到所作的改动。把app.component.html修改成下面这样。

code/first_app/angular2_hello_world/src/app/app.component.html

```
<h1>
  {{title}}

  <app-hello-world></app-hello-world>

  <app-user-item></app-user-item>
</h1>
```

然后刷新页面，并确认你在该页看到文本user-item works!。

我们希望UserItemComponent显示一个指定用户的名字。

因此，引入name并声明为组件的一个新属性。有了name属性，我们就在不同的用户之间复用该组件了（但要求页面脚本、逻辑和样式相同）。

为了添加名字，我们要在UserItemComponent类上引入一个属性，来声明该组件有一个名叫name的局部变量。

code/first_app/angular2_hello_world/src/app/user-item/user-item.component.ts

```
export class UserItemComponent implements OnInit {
  name: string; // <-- added name property

  constructor() {
    this.name = 'Felipe'; // set the name
  }

  ngOnInit() {
  }
}
```

注意，我们改变了以下两点。

1. name属性

我们往UserItemComponent类添加了一个属性。注意，这相对于ES5 JavaScript来说是个新语法。在name:string;中，name是我们想设置的属性名，而string是该属性的类型。

为name指定类型是TypeScript中的特性，用来确保它的值必须是string。这些代码在UserItemComponent类的实例中设置了一个名为name的属性，并且编译器会确保name是一个string。

2. 构造函数

在UserItemComponent类中，我们定义了一个构造函数。这个函数会在创建这个类的实例时自动调用。

在我们的构造函数中，可以通过this.name来设置name属性。

如果这样写：

code/first_app/angular2_hello_world/src/app/user-item/user-item.component.ts

```
constructor() {
  this.name = 'Felipe'; // set the name
}
```

就表示当一个新的UserItemComponent组件被创建时，把name设置为'Felipe'。

● 渲染模板

填好这个值之后，我们可以用模板语法（也就是双花括号语法{{ }}）在模板中显示该变量

的值。

```
code/first_app/angular2_hello_world/src/app/user-item/user-item.component.html
```

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

注意，我们在template中引入了一个新的语法：{{ name }}。这些括号叫作“模板标签”（也叫“小胡子标签”）。模板标签中间的任何东西都会被当作一个表达式来展开。这里，因为template是绑定到组件上的，所以name将会被展开为this.name的值，也就是'Felipe'。

3. 试试看

进行这些修改之后，重新加载页面，页面上应该显示Hello Felipe，如图1-5所示。

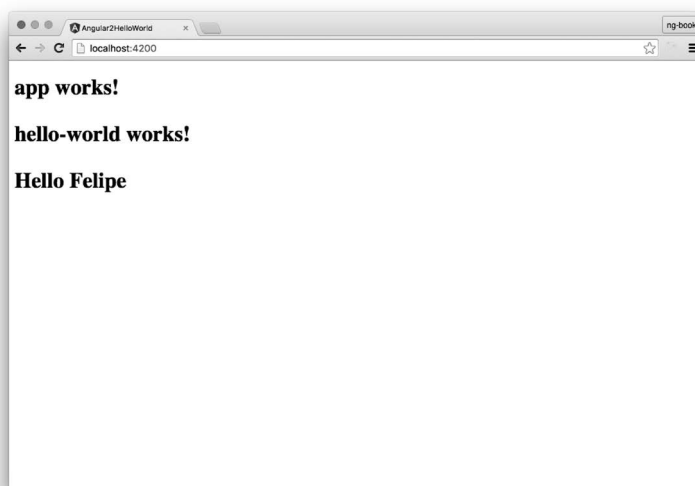


图1-5 带有数据的应用

1.5 使用数组

现在，我们可以对一个单独的名字问好了，但是如果想对一组名字问好呢？

如果你以前用过AngularJS，那么可能用过ng-repeat指令。在Angular中，NgFor是类似的指令（我们在模板标记中通过*ngFor语法来使用它，稍后会讲到）。它们在语法上略有不同，但作用是一样的：为一组对象反复渲染同样的页面脚本。

下面创建一个会渲染用户列表的新组件。我们还是从生成一个新组件开始：

```
ng generate component user-list
```

接着，把app.component.html文件中的<app-user-item>替换为<app-user-list>。

code/first_app/angular2_hello_world/src/app/app.component.html

```
<h1>
  {{title}}

  <app-hello-world></app-hello-world>

  <app-user-list></app-user-list>
</h1>
```

就像给UserItemComponent添加了name属性一样，我们也给UserListComponent添加names属性。

不过，不再设置该属性只存储一个字符串，而是存储一个字符串数组。数组的语法就是在类型后面紧跟一对方括号[]，如下所示。

code/first_app/angular2_hello_world/src/app/user-list/user-list.component.ts

```
export class UserListComponent implements OnInit {
  names: string[];

  constructor() {
    this.names = ['Ari', 'Carlos', 'Felipe', 'Nate'];
  }

  ngOnInit() {
  }
}
```

要留意的第一处变化是在UserListComponent类中添加了一个新的string[]属性。这种语法表示names的类型是string构成的数组。它的另一种写法是Array<string>。

我们还修改了构造函数，让它将this.names的值设置为['Ari', 'Carlos', 'Felipe', 'Nate']。

现在就可以更新模板，渲染出这个名字列表了。这时我们要用到*ngFor，它会在一个列表上进行迭代，为列表中的每一个条目生成一个新标签。新模板如下所示。

code/first_app/angular2_hello_world/src/app/user-list/user-list.component.html

```
<ul>
  <li *ngFor="let name of names">Hello {{ name }}</li>
</ul>
```

我们用一个ul和一个添加了*ngFor="let name of names"属性的li元素来更新模板。这个*字符和let语法可能会让你摸不着头脑，我们把它们拆开来解释。

*ngFor语法是说我们想在这个属性上使用NgFor指令。你可以把NgFor理解成一个类似于for的循环，其目的是为集中的每个条目都新建一个DOM元素。

它的值是"let name of names"。names是我们在HelloWorld对象中定义的名数组。let name叫作引用。"let name of names"的意思是，循环处理names中的每一个元素并将其逐个赋值给一个名叫name的局部变量。

NgFor指令将为数组names中的每一个条目都渲染出一个li标签，并声明一个本地变量name来持有当前迭代的条目。然后，这个新变量将被插值到Hello {{ name }}代码片段里。



并不是必须把这个引用变量命名为name。我们也可以这样写：

```
<li *ngFor="let foobar of names">Hello {{ foobar }}</li>
```

但把顺序反过来行吗？来个小测验吧！如果写成下面这样会如何？

```
<li *ngFor="let name of foobar">Hello {{ name }}</li>
```

当然会出错！因为foobar并不是该组件上的属性。



NgFor会重复渲染ngFor所在的元素。也就是说，我们应该把它放到li标签上而不是ul标签上，因为我们希望重复的是列表元素(li)而非列表本身(ul)。



如果你想进一步探索，可以直接阅读Angular源代码来学习Angular核心团队是如何编写组件的。比如，你能在https://github.com/angular/angular/blob/master/modules/%40angular/common/src/directives/ng_for.ts找到NgFor指令的源代码。

现在刷新页面，就会看到此数组中的每个字符串都有了对应的li，如图1-6所示。

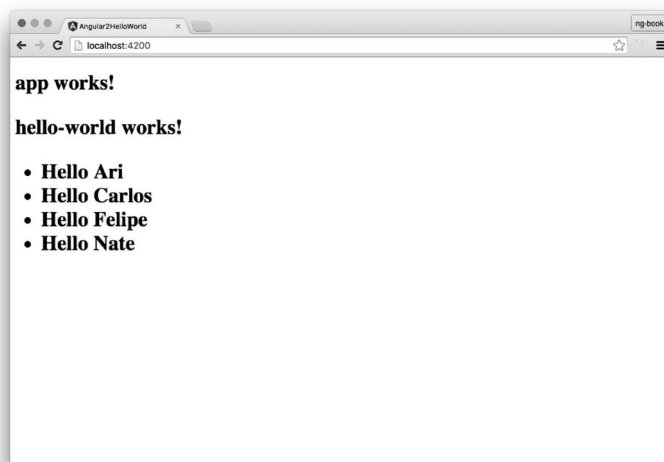


图1-6 带有数据的应用



在这个例子中，如果我们忘了这项限制，并且在`app.ts`（Angular）中调用`angular.module('interestApp', [])`，就会意外地覆盖现有的`interestApp`模块。你的应用将无法正常工作。千万要小心！

我们调用`.directive`并创建了一个名叫`'pinControls'`的指令。这是一种标准的AngularJS实践。它的第二个参数是指令定义对象（directive definition object, DDO），我们不会手动创建DDO，而是调用`upgradeAdapter.downgradeNg2Component`。

`downgradeNg2Component`会把我们的`PinControlsComponent`转换成与AngularJS兼容的指令。干净！漂亮！

刷新一下，你会发现收藏功能仍然正常工作（如图16-8所示），但我们已经把目前的实现方式改成在AngularJS中嵌入Angular了！



图16-8 收藏功能仍然很棒

16.6.8 用 Angular 添加图钉

接下来要用Angular组件对添加图钉的页面进行升级（如图16-9所示）。

The screenshot shows a web browser window with the title 'ng-book 2: Interest'. The address bar shows 'localhost:8080/#/add'. The page content is titled 'Interest what you're interested in' and has links for 'Home' and 'Add'. Below these links is a form with four input fields: 'Title' (containing 'Steampunk Cat'), 'Description' (containing 'A cat wearing goggles'), 'Link URL' (containing 'http://cats.com'), and 'Image URL' (containing '/images/pins/cat.jpg'). A 'Submit' button is located below the 'Image URL' field.

图16-9 新增图钉的表单

回想一下，这个页面一共做了三件事：

- (1) 为用户提供一个用来描述这个图钉的表单；
- (2) 借助PinsService把新的图钉添加到图钉列表中；
- (3) 把用户重定向到首页。

我们来看看该如何在Angular中做到这些。

Angular提供了一个强力的表单库，所以这没什么难度。那我们就来写一个正統的Angular表单吧。

不过，PinsService仍然来自AngularJS。通常，我们会有很多来自AngularJS的既有服务，但又没那么多时间把它们都改写成Angular的。因此在这个例子中，我们仍然把PinsService保留为AngularJS对象，并把它注入到Angular中。

与之类似，我们把来自AngularJS的ui-router作为路由系统。要想在ui-router中进行页面跳转，就得使用\$state服务，它是一个AngularJS服务。

那么，在这里要做的就是将PinsService和\$state服务从AngularJS升级到Angular，这已经是最简易的方式了。

16.6.9 把 AngularJS 的 PinsService 和\$state 升级到 Angular

要想升级AngularJS的服务，我们可以调用upgradeAdapter.upgradeNg1Provider。

code/conversion/hybrid/ts/app.ts

```
/*  
 * Expose our AngularJS content to Angular  
 */  
upgradeAdapter.upgradeNg1Provider('PinsService');  
upgradeAdapter.upgradeNg1Provider('$state');
```

这样就足够了。现在我们可以把AngularJS的服务注入（@Inject）到Angular的组件中，就像这样：

```
class AddPinComponent {  
  constructor(@Inject('PinsService') public pinsService: PinsService,  
              @Inject('$state') public uiState: IStateService) {  
  }  
  // ...  
  // now you can use this.pinsService  
  // or this.uiState  
  // ...  
}
```

在这个构造函数中，有几点需要注意。

@Inject注解的意思是，我们要把参数中指定的可注入对象解析出来，赋值给紧随其后的变量。比如这里的pinsService将被赋值为我们在AngularJS中定义的服务PinsService。

在TypeScript语法中，在constructor中使用public关键字其实是一种简写形式，用来把该变量赋值给this。也就是说，当我们写public pinsService时，其实是在做两件事：(1) 在该类上定义一个实例属性pinsService；(2) 把构造函数的参数pinsService赋值给this.pinsService。

最终的效果是我们可以在这个类中访问this.pinsService了。

最后，我们定义了所注入的两个服务的类型：PinsService和IStateService。

PinsService来自我们以前定义过的app.d.ts。

code/conversion/hybrid/js/app.d.ts

```
export interface PinsService {  
  pins(): Promise<Pin[]>;  
  addPin(pin: Pin): Promise<any>;  
}
```

IStateService来自ui-router的类型文件，它是我们以前用typings工具安装的。

通过把这些服务的类型信息告诉TypeScript，我们在写代码时就可以享受类型检查带来的好处了。

下面来写完AddPinComponent的剩余部分。

16.6.10 写 Angular 版的 AddPinComponent

我们先从导入所需的类型信息开始。

code/conversion/hybrid/ts/components/AddPinComponent.ts

```
/*  
 * AddPinComponent: a component that controls the "add pin" page  
 */  
import {  
  Component,  
  Inject  
} from '@angular/core';  
import { Pin, PinsService } from 'interestAppNg1';  
import { IStateService } from 'angular-ui-router';
```

注意，我们导入了自定义类型Pin和PinsService，还从angular-ui-router中导入了IStateService。

1. AddPinComponent的@Component

这个@Component注解非常简明。

code/conversion/hybrid/ts/components/AddPinComponent.ts

```
@Component({
  selector: 'add-pin',
  templateUrl: '/templates/add-Angular.html'
})
```

2. AddPinComponent模板

我们使用`templateUrl`来加载模板。在该模板中，我们的表单和AngularJS中的表单非常像，但所用的是Angular的表单指令集。



我们在这里不准备深入讲解`ngModel/ngSubmit`。如果你想深入了解Angular表单的工作原理，请阅读第5章，我们在那里对表单进行了详细讲解。

code/conversion/hybrid/templates/add-Angular.html

```
<div class="container">
  <div class="row">

    <form (ngSubmit)="onSubmit()"
      class="form-horizontal">

      <div class="form-group">
        <label for="title"
          class="col-sm-2 control-label">Title</label>
        <div class="col-sm-10">
          <input type="text"
            class="form-control"
            id="title"
            name="title"
            placeholder="Title"
            [(ngModel)]="newPin.title">
        </div>
      </div>
    </div>
  </div>
```

这里使用了两个指令：`ngSubmit`和`ngModel`。

我们在表单上使用了`(ngSubmit)`。这样当表单被提交时，就会调用`onSubmit`函数。（我们会在稍后的`AddPinComponent`控制器中定义`onSubmit`函数。）

我们使用`[(ngModel)]`来把`title`输入框的值绑定到控制器中`newPin.title`的值。

下面是完整的模板代码。

code/conversion/hybrid/templates/add-Angular.html

```
<div class="container">
  <div class="row">

    <form (ngSubmit)="onSubmit()"
      class="form-horizontal">
```

```
<div class="form-group">
  <label for="title"
    class="col-sm-2 control-label">Title</label>
  <div class="col-sm-10">
    <input type="text"
      class="form-control"
      id="title"
      name="title"
      placeholder="Title"
      [(ngModel)]="newPin.title">
  </div>
</div>

<div class="form-group">
  <label for="description"
    class="col-sm-2 control-label">Description</label>
  <div class="col-sm-10">
    <input type="text"
      class="form-control"
      id="description"
      name="description"
      placeholder="Description"
      [(ngModel)]="newPin.description">
  </div>
</div>

<div class="form-group">
  <label for="url"
    class="col-sm-2 control-label">Link URL</label>
  <div class="col-sm-10">
    <input type="text"
      class="form-control"
      id="url"
      name="url"
      placeholder="Link URL"
      [(ngModel)]="newPin.url">
  </div>
</div>

<div class="form-group">
  <label for="url"
    class="col-sm-2 control-label">Image URL</label>
  <div class="col-sm-10">
    <input type="text"
      class="form-control"
      id="url"
      name="url"
      placeholder="Image URL"
      [(ngModel)]="newPin.src">
  </div>
</div>

<div class="form-group">
  <div class="col-sm-offset-2 col-sm-10">
```



```
<button type="submit"
        class="btn btn-default"
        >Submit</button>
</div>
</div>
<div *ngIf="saving">
    Saving...
</div>
</form>
```

3. AddPinComponent控制器

现在我们就可以定义AddPinComponent了。先从两个实例变量开始。

code/conversion/hybrid/ts/components/AddPinComponent.ts

```
export class AddPinComponent {
    saving: boolean = false;
    newPin: Pin;
```

saving会告诉用户我们正在进行保存，而newPin用于存储我们正在使用的Pin对象。

code/conversion/hybrid/ts/components/AddPinComponent.ts

```
constructor(@Inject('PinsService') private pinsService: PinsService,
            @Inject('$state') private uiState: IStateService) {
    this.newPin = this.makeNewPin();
}
```

如前所述，我们用Inject在constructor中注入了这些服务，并且把this.newPin的值设置成了makeNewPin，也就是下面这个函数。

code/conversion/hybrid/ts/components/AddPinComponent.ts

```
makeNewPin(): Pin {
    return {
        title: 'Steampunk Cat',
        description: 'A cat wearing goggles',
        user_name: 'me',
        avatar_src: 'images/avatars/me.jpg',
        src: '/images/pins/cat.jpg',
        url: 'http://cats.com',
        faved: false,
        id: Math.floor(Math.random() * 10000).toString()
    };
}
```

这看起来很像AngularJS中的定义方式，不过这种方式现在的优点在于它是带类型信息的。

当用户提交表单时，我们调用onSubmit，其定义如下所示。

code/conversion/hybrid/ts/components/AddPinComponent.ts

```
onSubmit(): void {
    this.saving = true;
```

```

console.log('submitted', this.newPin);
setTimeout(() => {
  this.pinsService.addPin(this.newPin).then(() => {
    this.newPin = this.makeNewPin();
    this.saving = false;
    this.uiState.go('home');
  });
}, 2000);
}

```

我们再次使用超时（timeout）技术来模拟通过向服务器发起请求来保存图钉的效果。这里我们使用的是`setTimeout`。下面对比一下在AngularJS中实现同样功能的写法。

code/conversion/AngularJS/js/app.js

```

ctrl.submitPin = function() {
  ctrl.saving = true;
  $timeout(function() {
    PinsService.addPin(ctrl.newPin).then(function() {
      ctrl.newPin = makeNewPin();
      ctrl.saving = false;
      $state.go('home');
    });
  }, 2000);
}

```

注意，我们在AngularJS中必须使用`$timeout`服务。为什么呢？这是因为AngularJS是基于摘要循环（digest loop）的。如果你在AngularJS中直接使用`setTimeout`，那么当调用回调函数时，它会处于Angular的控制范围之外。因此改动造成的影响不会扩散出来，除非某些代码触发了摘要循环（比如使用`$scope.apply`）。

然而在Angular中，你可以直接使用`setTimeout`，因为Angular中的变更检测使用的是Zones，所以更加自动化。你再也不用担心摘要循环了，这太好了！

在`onSubmit`中，我们通过下列代码调用了`PinsService`：

```

this.pinsService.addPin(this.newPin).then(() => {
  // ...

```

`PinsService`可以通过`this.pinsService`来访问，因为我们定义`constructor`时使用了特殊写法。编译器没有报错，这是因为我们已经在`app.d.ts`中声明过`addPin`接收一个`Pin`对象作为第一个参数。

code/conversion/hybrid/js/app.d.ts

```

export interface PinsService {
  pins(): Promise<Pin[]>;
  addPin(pin: Pin): Promise<any>;
}

```

我们还把`this.newPin`定义成了一个`Pin`对象。

addPin解析完成后，我们把this.newPin重置为this.makeNewPin()的结果，并设置this.saving = false。

要返回首页，就要使用ui-router的\$state服务。我们已经通过依赖注入把它存储到了this.uiState属性中，所以可以直接调用this.uiState.go('home')来变更状态。

16.6.11 使用 AddPinComponent

我们现在就来使用AddPinComponent。

1. 降级Angular的AddPinComponent

要想使用AddPinComponent，就得先把它降级。

code/conversion/hybrid/ts/app.ts

```
angular.module('interestApp')
  .directive('pinControls',
    upgradeAdapter.downgradeNg2Component(PinControlsComponent))
  .directive('addPin',
    upgradeAdapter.downgradeNg2Component(AddPinComponent));
```

这会在AngularJS中创建一个addPin指令，它会匹配<add-pin>标签。

2. 路由到add-pin

为了使用这个新的AddPinComponent页，就要把它放进AngularJS应用中的某个地方。这很简单，只要让路由器拿到这个add状态，并把<add-pin>指令放到模板中就可以了。

code/conversion/hybrid/js/app.js

```
.state('add', {
  template: "<add-pin></add-pin>",
  url: '/add',
  resolve: {
    'pins': function(PinsService) {
      return PinsService.pins();
    }
  }
})
})
```

16.6.12 把 Angular 的服务暴露给 AngularJS

目前，我们已经降级了Angular的组件使其能用在AngularJS中，还升级了AngularJS的服务使其能用在Angular中。但是当我们的应用开始升级到Angular时，可能会需要用TypeScript/Angular写一些服务，并把它暴露给AngularJS的代码。

那么我们就在Angular中创建一个简单的“分析”（analytics）服务，用来记录事件。

我们的想法是：在应用中有一个AnalyticsService，我们将调用它的recordEvent方法。在

具体实现上，我们只会调用`console.log`来记录该事件，并把它存到一个数组中。这样做是为了把精力集中在最重要的事情上：描述如何把Angular的服务共享给AngularJS。

16.6.13 实现 `AnalyticsService`

我们先来看看`AnalyticsService`的实现。

code/conversion/hybrid/ts/services/AnalyticsService.ts

```
import { Injectable } from '@angular/core';

/**
 * Analytics Service records metrics about what the user is doing
 */
@Injectable()
export class AnalyticsService {
  events: string[] = [];

  public recordEvent(event: string): void {
    console.log(`Event: ${event}`);
    this.events.push(event);
  }
}

export var analyticsServiceInjectables: Array<any> = [
  { provide: AnalyticsService, useClass: AnalyticsService }
];
```

这里需要注意两点：`recordEvent`和`Injectable`。

`recordEvent`很简明：我们接收一个`event: string`参数，输出它的日志，并且把它保存到`events`中。在现实世界的应用中，你可能会把它发给某个外部服务，比如Google分析或Mixpanel。

要让该服务可注入，我们得做两件事：(1) 为该类添加`@Injectable`注解；(2) 把`AnalyticsService`这个令牌bind到该类。

现在，Angular将会管理该服务的单例对象，而我们可以把它注入到任何需要它的地方了。

16.6.14 把 Angular 的 `AnalyticsService` 降级到 AngularJS

在AngularJS中使用`AnalyticsService`服务之前，我们需要把它降级。

把Angular服务降级到AngularJS的过程和指令的降级过程很相似，只不过多出了一个额外的步骤：得先确保`AnayticsService`出现在了我们的`NgModule`的`providers`列表中。

code/conversion/hybrid/ts/app.ts

```
@NgModule({
  declarations: [
```

```
PinControlsComponent,  
  AddPinComponent  
],  
imports: [  
  CommonModule,  
  BrowserModule,  
  FormsModule  
],  
providers: [  
  AnalyticsService,  
]  
})  
class InterestAppModule { }
```

然后就可以使用`downgradeNg2Provider`了。

code/conversion/hybrid/ts/app.ts

```
angular.module('interestApp')  
  .factory('AnalyticsService',  
    upgradeAdapter.downgradeNg2Provider(AnalyticsService));
```

我们先调用`angular.module('interestApp')`来取得AngularJS的模块，然后像在AngularJS中一样调用`.factory`。要想降级该服务，要调用`upgradeAdapter.downgradeNg2Provider(AnalyticsService)`。它会把我们的`AnalyticsService`包装到一个函数中，而该函数会把它适配成一个AngularJS的工厂（`factory`）。

16.6.15 在 AngularJS 中使用 AnalyticsService

现在就可以把Angular的`AnalyticsService`注入到AngularJS中去了。假如我们想记录`HomeController`是什么时候被访问的，就可以像下面这样来记录此事件。

code/conversion/hybrid/js/app.js

```
.controller('HomeController', function(pins, AnalyticsService) {  
  AnalyticsService.recordEvent('HomeControllerVisited');  
  this.pins = pins;  
})
```

这里注入了`AnalyticsService`，就像它是AngularJS中的普通服务一样，然后调用`recordEvent`。真棒！

我们可以在AngularJS中任何能使用依赖注入的地方使用该服务。比如，我们也可以把`AnalyticsService`注入到AngularJS的`pin`指令中。

code/conversion/hybrid/js/app.js

```
.directive('pin', function(AnalyticsService) {  
  return {  
    restrict: 'E',  
    templateUrl: '/templates/pin.html',
```

```
scope: {
  'pin': "=item"
},
link: function(scope, elem, attrs) {
  scope.toggleFav = function() {
    AnalyticsService.recordEvent('PinFaved');
    scope.pin.faved = !scope.pin.faved;
  }
}
})
})
```

16.7 总结

现在我们掌握了把AngularJS应用升级到AngularJS/Angular混合式应用时所需的工具。AngularJS和Angular之间也有非常好的互操作性，这是因为Angular开发组付出了很多努力来对其进行简化。

AngularJS和Angular的指令与服务之间能够互通，让应用升级变得非常容易。当然，我们不可能一夜之间就把AngularJS的应用升级到Angular，不过UpgradeAdapter能让我们不必把那些老代码扔掉就开始使用Angular。

16.8 参考资源

如果你想了解关于混合式Angular应用的更多知识，可以参阅下列资源。

- ❑ 官方的Angular升级指南（中文版）：<https://angular.cn/docs/ts/latest/guide/upgrade.html>
- ❑ Angular 升级模块的单元测试：https://github.com/angular/angular/blob/master/modules/angular2/test/upgrade/upgrade_spec.ts
- ❑ Angular中DowngradeNg2ComponentAdapter的源代码：https://github.com/angular/angular/blob/master/modules/angular2/src/upgrade/downgrade_Angular_adapter.ts

需要整本电子书，联系我QQ：2667271557

ng-book 2 The Complete Book on Angular 2

“很高兴这本《Angular权威教程》成为Angular中文资源的一部分，希望它能广受欢迎，给中国的Angular社区提供一份令人愉悦的学习资源，也希望它帮助更多工程师开始使用下一代Angular框架来开发应用。”

——Naomi Black, Google Angular项目经理兼主管

“作为一项开源技术和前沿Web开发框架，Angular持续吸引着中国区开发人员的关注。作为雪狼及其所属Nice Angular社区的集体工作成果，这本书是开源力量的又一次证明，证明这种热情、这种志愿精神确实可以帮助业界享受到全球最新的开发技术。”

——栾跃, Google开发技术推广部大中华区主管

“作者们太棒了！如果没有这本书，真不知道我该怎么学习Angular。你们让学习并跟进Angular变得更简单了。再次感谢！”

——Jacob Cheriathundam, AccountsPRO公司CTO、高级开发工程师兼开发架构师

“我刚刚读完这本书，认为它是目前学习Angular的最佳材料。”

——Jegor Uglov, BlaBlaBlogger产品主管

“如果你和我一样是一名经验丰富的开发者，并且在积极寻找关于Angular最新信息的高效来源，那就别再找了！这本书就是目前最棒的参考资料，简洁易懂、结构合理。”

——Frederic Filiatrault, TEKsystem公司高级软件工程师

“我在书中获取了大量有价值的信息，而这是在其他网络资源中无法做到的。在我深入这些前沿工具和主题的时候，这本书给了我极大帮助。”

——Sean McGill, Anexinet公司高级顾问

图灵社区：iTuring.cn

热线：(010)51095186转600

分类建议 计算机/Web开发/Angular

人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-45158-3



ISBN 978-7-115-45158-3

定价：109.00元

需要整本电子书，联系我QQ：2667271557