



疯狂

Kotlin 讲义

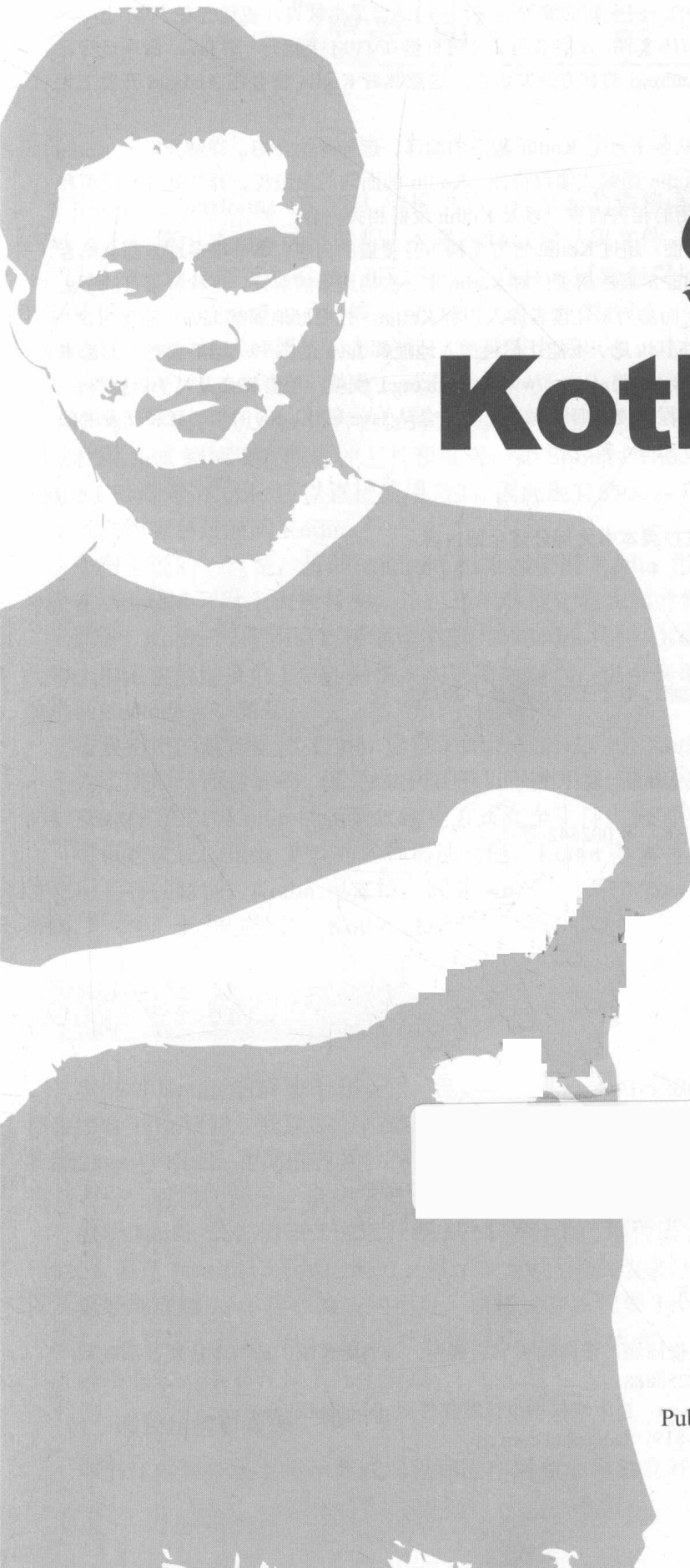
李刚 编著

疯狂源自梦想

技术成就辉煌

疯狂源自梦想

技术成就辉煌



疯狂

Kotlin讲义

李刚 编著

电子工业出版社

Publishing House of Electronics Industry

北京·BEIJING

内 容 简 介

Kotlin 是 JetBrains 在 2011 年推出的一门全新的编程语言，这门语言最早被设计成运行在 JVM 上——使用 Kotlin 编写的程序会被编译成字节码文件，该字节码文件可直接在 JVM 上运行（用 java 命令运行）。目前 Google 已推荐使用 Kotlin 作为 Android 的官方开发语言，这意味着 Kotlin 将会在 Android 开发上大放异彩。

本书全面介绍了 Kotlin 的语法。从各平台上 Kotlin 程序的编译、运行开始介绍，详细介绍了 Kotlin 的基本语法，Kotlin 的数组和集合，Kotlin 函数式编程特征，Kotlin 的面向对象编程、异常处理、泛型和注解，还介绍了 Kotlin 与 Java 混合调用的相关内容，以及 Kotlin 反射相关内容。

本书对 Kotlin 的解读十分系统、全面，超过 Kotlin 官方文档本身覆盖的内容。本书很多地方都会结合 Java 字节码进行深度解读，比如对 Kotlin 扩展的解读，对 Kotlin 主、次构造器的解读，这种解读目的不止于教会读者简单地掌握 Kotlin 的用法，而是力求让读者深入理解 Kotlin，且更好地理解 Java。简单来说，本书不仅是一本 Kotlin 的学习图书，而且也是一本能让你更深入地理解 Java 的图书。如果读者在阅读本书时遇到了技术问题，可以登录疯狂 Java 联盟 (<http://www.crazyit.org>) 发帖，笔者将会及时予以解答。

本书为所有打算深入掌握 Kotlin 编程的读者而编写，尤其适合从 Java 转 Kotlin 的学习者和开发者阅读，也适合作为大学教育、培训机构的 Kotlin 教材。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有，侵权必究。

图书在版编目 (CIP) 数据

疯狂 Kotlin 讲义 / 李刚编著. —北京: 电子工业出版社, 2018.2
ISBN 978-7-121-33459-7

I. ①疯… II. ①李… III. ①JAVA 语言—程序设计 IV. ①TP312.8

中国版本图书馆 CIP 数据核字 (2018) 第 002842 号

策划编辑: 张月萍

责任编辑: 葛 娜

印 刷: 三河市良远印务有限公司

装 订: 三河市良远印务有限公司

出版发行: 电子工业出版社

北京市海淀区万寿路 173 信箱

邮编: 100036

开 本: 787×1092 1/16 印张: 20.25

字数: 571 千字

版 次: 2018 年 2 月第 1 版

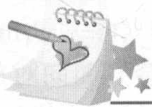
印 次: 2018 年 2 月第 1 次印刷

印 数: 3500 册 定价: 69.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话: (010) 88254888, 88258888。

质量投诉请发邮件至 zltz@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式: 010-51260888-819, faq@phei.com.cn。



前言

Kotlin 是 JetBrains 在 2011 年推出的一门全新的编程语言，这门语言最早被设计成运行在 JVM 上——使用 Kotlin 编写的程序会被编译成字节码文件，该字节码文件可直接在 JVM 上运行（用 java 命令运行即可）。Kotlin 可以与现有的 Java 语言包保持完全兼容，而且 Kotlin 代码比 Java 代码更简洁。Kotlin 增加了扩展、对象表达式、对象声明、委托等 Java 原本不支持的功能，它们都是现代编程语言广泛支持的功能，并且完全可以在 JVM 上运行。

简单来说，Kotlin 既可利用 Java 的优势，又比 Java 更简洁。

Kotlin 与现有的 Java 语言包保持完全兼容，这意味着 Kotlin 不是一门简单的语言，它完全可以利用 Java 领域现有的各种工具和框架，如 Spring、Hibernate、MyBatis、Lucene、Hadoop、Spring Cloud 等。Kotlin 可以直接使用它们，因此现有的 Java 项目完全可以采用 Kotlin 开发，Java 开发者也很容易过渡到 Kotlin。

不得不说的有一点是，目前 Android 已推荐使用 Kotlin 作为官方开发语言，这意味着 Kotlin 将会在 Android 开发上大放异彩，这也是笔者决定向大家介绍这门语言的重要原因之一。

此外，Kotlin 程序还可直接编译生成 JavaScript 代码，Kotlin 程序既可编译成前端 JavaScript 代码，用于实现网页的 DOM 操作，实现前端编程；也可编译成后端 JavaScript 代码，与服务端技术（如 Node.js）交互。

需要指出的是，虽然 Kotlin 提供了简洁的语法，但 Kotlin 的功能并不简单，Kotlin 从来就不是为了更简单而设计的，而是为了更强大而设计的。Kotlin 既支持函数式编程方式，也支持面向对象编程方式。Kotlin 的函数式编程方式完全支持主流的函数和闭包，语法功能非常丰富。

可以这么说：Java 支持的各种语法功能，Kotlin 基本都支持；Java 不支持的很多现代编程语言所具有的特征，Kotlin 也支持，因此 Kotlin 绝不比 Java 更简单。如果读者相信网络上某些所谓“大神”肤浅的结论：Kotlin 很简单，那么我建议你放弃阅读这本书。

本书有什么特点



本书对 Kotlin 的解读十分全面、深入，并非一本简单介绍 Kotlin 语法的图书，在很多地方都会结合 Java 语法、底层字节码进行讲解。如果读者有较好的 Java 功底，阅读本书能更清晰地看清 Java 与 Kotlin 之间的差异，便于快速上手 Kotlin；对于没有 Java 功底的读者，可选择忽略将二者进行对比的部分，直接学习本书也可掌握 Kotlin 语言的编程。

由于 Kotlin 最先被设计成运行在 JVM 平台上的编程语言，因此 Kotlin 具有和 Java 天然的相似性，但在 Java 设计不足的地方又做了大量的补充、改进，所以本书也能让你更好地理解 Java，以及更好地理解 Java 存在的一些不足。这样说并不代表 Java 不优秀，“知其雄，守其雌，为天下谿”，只有深入理解 Java，才能更好地感悟 Java 的优秀。

总结起来，本书有如下几个特点。

1. 逻辑结构更合理

本书在内容体系上将函数式编程和面向对象编程独立开来，先介绍函数式编程部分，再介

绍面向对象编程，更符合 Kotlin 语言本身的知识体系。而不像某些资料一会儿函数，一会儿面向对象，搅得读者晕头转向。实际上，无论是经典的图书如《C++ Primer》，还是 Swift 官方文档（甚至 Kotlin 官方文档），几乎都没见过一会儿函数、一会儿面向对象这样介绍的。

在介绍知识时，本书会先详细讲解各种知识点的理论，然后再通过示例演示 Kotlin 各理论的使用法，将知识点融合在示例中，符合读者的认知、学习规律。

另外，整本书的知识具有和《疯狂 Java 讲义》大致相同的脉络，所以《疯狂 Java 讲义》的读者会很容易上手。

2. 讲解深入本质

Kotlin 是 JVM 语言，所以其很多东西其实是受到 Java 的影响的。书中会对一些看似奇怪的语法从字节码文件层次进行剖析，让读者更好地理解 Kotlin 与 Java 的对应关系。

比如，主构造器和次构造器到底是什么？Java 构造器并不区分主次，为什么 Kotlin 搞出这两个东西？主、次构造器为何要委托父类构造器？委托父类构造器时为什么存在区别？主、次构造器生成字节码之后到底对应 Java 的哪个部分？这些知识在本书 7.5 节有深入讲解。

再比如，Java 本身不支持扩展，那么 Kotlin 的扩展是如何在 JVM 上运行的？难道 Kotlin 改造了 JVM 吗？Java 本身不支持扩展，那么 Java 是否可以调用 Kotlin 扩展的成员？这些问题需要从字节码层次进行剖析，本书在 8.1 节有深入讲解。

还有，Java 泛型的上限、下限的本质是什么？Kotlin 泛型的声明处型变和使用处型变的本质是什么？与 Java 的对应关系是怎样的？Java 本身并不支持声明处型变，那为何 JVM 能支持 Kotlin 的声明处型变？这些问题可以在本书第 10 章中找到答案。

3. 知识内容更全面

本书内容超过 Kotlin 官方文档本身所覆盖的知识，比如介绍反射的章节就超过了 Kotlin 官方文档内容。本书反射部分不仅更详细地介绍如何获得类、函数、属性的引用，而且真正从 API 级别介绍 KClass、KCallable、KFunction、KProperty，KProperty0、KProperty1、KProperty2 的用法，以及它们的内在关联，并实实在在地教读者掌握如何用 Kotlin 反射动态创建对象、动态调用方法。

本书写给谁看



本书为所有打算深入掌握 Kotlin 编程的读者而编写，尤其适合从 Java 转 Kotlin 的学习者和开发者阅读，也适合作为大学教育、培训机构的 Kotlin 教材。

本书程序文件请从 www.broadview.com.cn/33459 下载。

2017-12-3

目 录 CONTENTS

第 1 章 Kotlin 语言与开发环境.....	1	2.11 本章小结.....	38
1.1 Kotlin 语言简介.....	2	第 3 章 运算符和表达式.....	39
1.1.1 服务端的 Kotlin.....	2	3.1 与 Java 相同的运算符.....	40
1.1.2 使用 Kotlin 开发 Android 应用.....	2	3.1.1 单目前缀运算符.....	40
1.1.3 Kotlin 用于 JavaScript.....	3	3.1.2 自加和自减运算符.....	41
1.2 使用命令行编译、运行 Kotlin.....	3	3.1.3 双目算术运算符.....	41
1.2.1 下载和安装 Kotlin 的 SDK.....	3	3.1.4 in 和 in 运算符.....	42
1.2.2 第一个 Kotlin 程序.....	4	3.1.5 索引访问运算符.....	43
1.2.3 编译、运行 Kotlin 程序.....	5	3.1.6 调用运算符.....	43
1.3 使用 IntelliJ IDEA 编译、运行 Kotlin.....	6	3.1.7 广义赋值运算符.....	44
1.4 使用 Eclipse 编译、运行 Kotlin.....	8	3.1.8 相等与不等运算符.....	44
1.5 本章小结.....	10	3.1.9 比较运算符.....	45
第 2 章 Kotlin 的基础类型.....	11	3.2 位运算符.....	46
2.1 注释.....	12	3.3 区间运算符.....	48
2.1.1 单行注释和多行注释.....	12	3.3.1 闭区间运算符.....	48
2.1.2 文档注释.....	12	3.3.2 半开区间运算符.....	49
2.2 变量.....	14	3.3.3 反向区间.....	49
2.2.1 分隔符.....	15	3.3.4 区间步长.....	49
2.2.2 标识符规则.....	16	3.4 运算符重载.....	50
2.2.3 Kotlin 的关键字.....	17	3.4.1 重载单目前缀运算符.....	50
2.2.4 声明变量.....	19	3.4.2 重载自加和自减运算符.....	51
2.3 整型.....	21	3.4.3 重载双目算术运算符.....	51
2.4 浮点型.....	23	3.5 本章小结.....	52
2.5 字符型.....	24	第 4 章 流程控制.....	53
2.6 数值型之间的类型转换.....	25	4.1 顺序结构.....	54
2.6.1 整型之间的转换.....	25	4.2 分支结构.....	54
2.6.2 浮点型与整型之间的转换.....	28	4.2.1 if 分支.....	54
2.6.3 表达式类型的自动提升.....	28	4.2.2 if 表达式.....	58
2.7 Boolean 类型.....	30	4.2.3 when 分支语句.....	58
2.8 null 安全.....	31	4.2.4 when 表达式.....	61
2.8.1 非空类型和可空类型.....	31	4.2.5 when 分支处理范围.....	62
2.8.2 先判断后使用.....	32	4.2.6 when 分支处理类型.....	62
2.8.3 安全调用.....	32	4.2.7 when 条件分支.....	63
2.8.4 Elvis 运算.....	33	4.3 循环结构.....	63
2.8.5 强制调用.....	34	4.3.1 while 循环.....	64
2.9 字符串.....	34	4.3.2 do while 循环.....	65
2.9.1 字符串类型.....	34	4.3.3 for-in 循环.....	66
2.9.2 字符串模板.....	35	4.3.4 嵌套循环.....	66
2.9.3 Kotlin 字符串的方法.....	36	4.4 控制循环结构.....	68
2.10 类型别名.....	37	4.4.1 使用 break 结束循环.....	68

4.4.2	使用 continue 忽略本次循环的 剩下语句	69	6.5.2	使用函数类型作为形参类型	113
4.4.3	使用 return 结束方法	70	6.5.3	使用函数类型作为返回值类型	114
4.5	本章小结	71	6.6	局部函数与 Lambda 表达式	115
第 5 章	数组和集合	72	6.6.1	回顾局部函数	116
5.1	数组	73	6.6.2	使用 Lambda 表达式代替局部函数	116
5.1.1	创建数组	73	6.6.3	Lambda 表达式的脱离	117
5.1.2	使用数组	75	6.7	Lambda 表达式	117
5.1.3	使用 for-in 循环遍历数组	76	6.7.1	调用 Lambda 表达式	118
5.1.4	使用数组索引	76	6.7.2	利用上下文推断类型	118
5.1.5	数组的常用方法	77	6.7.3	省略形参名	119
5.1.6	多维数组	80	6.7.4	调用 Lambda 表达式的约定	120
5.1.7	数组的应用举例	82	6.7.5	个数可变的参数和 Lambda 参数	120
5.2	Kotlin 集合概述	85	6.8	匿名函数	121
5.3	Set 集合	88	6.8.1	匿名函数的用法	121
5.3.1	声明和创建 Set 集合	88	6.8.2	匿名函数和 Lambda 表达式 的 return	122
5.3.2	使用 Set 的方法	90	6.9	捕获上下文中的变量和常量	123
5.3.3	遍历 Set	91	6.10	内联函数	125
5.3.4	可变的 Set	92	6.10.1	内联函数的使用	125
5.4	List 集合	93	6.10.2	部分禁止内联	126
5.4.1	声明和创建 List 集合	93	6.10.3	非局部返回	127
5.4.2	使用 List 的方法	94	6.11	本章小结	128
5.4.3	可变的 List	95	第 7 章	面向对象 (上)	129
5.5	Map 集合	95	7.1	类和对象	130
5.5.1	声明和创建 Map 集合	95	7.1.1	定义类	130
5.5.2	使用 Map 的方法	97	7.1.2	对象的产生和使用	132
5.5.3	遍历 Map	98	7.1.3	对象的 this 引用	133
5.5.4	可变的 Map	98	7.2	方法详解	136
5.6	本章小结	99	7.2.1	方法与函数的关系	136
第 6 章	函数和 Lambda 表达式	100	7.2.2	中缀表示法	137
6.1	函数入门	101	7.2.3	componentN 方法与解构	138
6.1.1	定义和调用函数	101	7.2.4	数据类和返回多个值的函数	140
6.1.2	函数返回值和 Unit	102	7.2.5	在 Lambda 表达式中解构	141
6.1.3	递归函数	103	7.3	属性和字段	142
6.1.4	单表达式函数	104	7.3.1	读写属性和只读属性	142
6.2	函数的形参	105	7.3.2	自定义 getter 和 setter	144
6.2.1	命名参数	105	7.3.3	幕后字段	147
6.2.2	形参默认值	106	7.3.4	幕后属性	148
6.2.3	尾递归函数	108	7.3.5	延迟初始化属性	149
6.2.4	个数可变的形参	109	7.3.6	内联属性	150
6.3	函数重载	110	7.4	隐藏和封装	151
6.4	局部函数	111	7.4.1	包和导包	151
6.5	高阶函数	112	7.4.2	Kotlin 的默认导入	153
6.5.1	使用函数类型	112	7.4.3	使用访问控制符	153
			7.5	深入构造器	155

7.5.1	主构造器和初始化块	156	8.6	对象表达式和对象声明	212
7.5.2	次构造器和构造器重载	158	8.6.1	对象表达式	212
7.5.3	主构造器声明属性	161	8.6.2	对象声明和单例模式	215
7.6	类的继承	161	8.6.3	伴生对象和静态成员	217
7.6.1	继承的语法	161	8.6.4	伴生对象的扩展	218
7.6.2	重写父类的方法	164	8.7	枚举类	219
7.6.3	重写父类的属性	166	8.7.1	枚举类入门	219
7.6.4	super 限定	167	8.7.2	枚举类的属性、方法和构造器	221
7.6.5	强制重写	168	8.7.3	实现接口的枚举类	222
7.7	多态	169	8.7.4	包含抽象方法的抽象枚举类	222
7.7.1	多态性	169	8.8	类委托和属性委托	223
7.7.2	使用 is 检查类型	170	8.8.1	类委托	224
7.7.3	使用 as 运算符转型	172	8.8.2	属性委托	225
7.8	本章小结	174	8.8.3	延迟属性	227
			8.8.4	属性监听	228
			8.8.5	使用 Map 存储属性值	230
			8.8.6	局部属性委托	231
			8.8.7	委托工厂	233
			8.9	本章小结	234
第 8 章	面向对象 (下)	175	第 9 章	异常处理	236
8.1	扩展	176	9.1	异常处理机制	237
8.1.1	扩展方法	176	9.1.1	使用 try...catch 捕获异常	237
8.1.2	扩展的实现机制	179	9.1.2	异常类的继承体系	240
8.1.3	为可空类型扩展方法	182	9.1.3	访问异常信息	242
8.1.4	扩展属性	182	9.1.4	异常处理嵌套	243
8.1.5	以成员方式定义扩展	183	9.1.5	try 语句是表达式	243
8.1.6	带接收者的匿名函数	184	9.2	使用 throw 抛出异常	243
8.1.7	何时使用扩展	186	9.2.1	抛出异常	243
8.2	final 和 open 修饰符	187	9.2.2	自定义异常类	244
8.2.1	可执行“宏替换”的常量	187	9.2.3	catch 和 throw 同时使用	245
8.2.2	final 属性	188	9.2.4	异常链	246
8.2.3	final 方法	189	9.2.5	throw 语句是表达式	247
8.2.4	final 类	190	9.3	异常的跟踪栈	248
8.2.5	不可变类	190	9.4	本章小结	250
8.3	抽象类	192	第 10 章	泛型	251
8.3.1	抽象成员和抽象类	192	10.1	泛型入门	252
8.3.2	抽象类的作用	195	10.1.1	定义泛型接口、类	252
8.3.3	密封类	196	10.1.2	从泛型类派生子类	253
8.4	接口	198	10.2	型变	254
8.4.1	接口的定义	198	10.2.1	泛型型变的需要	254
8.4.2	接口的继承	199	10.2.2	声明处型变	256
8.4.3	使用接口	200	10.2.3	使用处型变: 类型投影	258
8.4.4	接口和抽象类	202	10.2.4	星号投影	260
8.5	嵌套类和内部类	202	10.3	泛型函数	261
8.5.1	内部类	204			
8.5.2	嵌套类	207			
8.5.3	在外部类以外使用内部类	209			
8.5.4	在外部类以外使用嵌套类	209			
8.5.5	局部嵌套类	210			
8.5.6	匿名内部类	211			

10.3.1	泛型函数的使用	261	12.1.6	调用参数个数可变的方法	288
10.3.2	具体化类型参数	262	12.1.7	checked 异常	289
10.4	设定类型形参的上限	263	12.1.8	Object 的处理	289
10.5	本章小结	264	12.1.9	访问静态成员	290
第 11 章	注解	266	12.1.10	SAM 转换	290
11.1	Kotlin 注解入门	267	12.1.11	在 Kotlin 中使用 JNI	291
11.1.1	定义注解	267	12.2	Java 调用 Kotlin	291
11.1.2	注解的属性和构造器	268	12.2.1	属性	291
11.2	元注解	270	12.2.2	包级函数	292
11.2.1	使用@Retention	270	12.2.3	实例变量	294
11.2.2	使用@Target	271	12.2.4	类变量	294
11.2.3	使用@MustBeDocumented	272	12.2.5	类方法	296
11.2.4	使用@Repeatable 标记可重复注解	273	12.2.6	访问控制符的对应关系	297
11.3	使用注解	273	12.2.7	获取 KClass	298
11.3.1	提取注解信息	273	12.2.8	使用@JvmName 解决签名冲突	298
11.3.2	使用注解的示例	274	12.2.9	生成重载	299
11.4	Java 注解与 Kotlin 的兼容性	279	12.2.10	checked 异常	300
11.4.1	指定注解的作用目标	279	12.2.11	泛型的型变	300
11.4.2	使用 Java 注解	281	12.3	Kotlin 反射	302
11.5	本章小结	282	12.3.1	类引用	302
第 12 章	Kotlin 与 Java 互相调用	283	12.3.2	从 KClass 获取类信息	303
12.1	Kotlin 调用 Java	284	12.3.3	创建对象	306
12.1.1	属性	284	12.3.4	构造器引用	306
12.1.2	void 和调用名为关键字的成员	285	12.3.5	调用方法	307
12.1.3	Kotlin 的已映射类型	286	12.3.6	函数引用	308
12.1.4	Kotlin 对 Java 泛型的转换	287	12.3.7	访问属性值	309
12.1.5	对 Java 数组的处理	287	12.3.8	属性引用	311
			12.3.9	绑定的方法与属性引用	313
			12.4	本章小结	313

第 1 章

Kotlin 语言与开发环境

本章要点

- ✎ Kotlin 语言简介
- ✎ 服务端 Kotlin 与客户端 Kotlin
- ✎ 下载和安装 Kotlin 的 SDK
- ✎ 编写第一个 Kotlin 程序
- ✎ 编译、运行 Kotlin 程序
- ✎ 使用 IntelliJ IDEA 编译、运行 Kotlin
- ✎ 使用 Eclipse 编译、运行 Kotlin

Kotlin 是 JetBrains 在 2011 年推出的一门全新的编程语言，这门语言最早被设计成运行在 JVM（Java 虚拟机）上——使用 Kotlin 编写的程序会被编译成字节码文件，该字节码文件可直接在 JVM 上运行（用 java 命令运行）。Kotlin 可以与现有的 Java 语言包保持 100% 的兼容性，而且 Kotlin 代码比 Java 代码更简洁、更富有表现力。简单来说，一句话：Kotlin 既可利用 Java 的优势，又比 Java 更简洁。

此外，Kotlin 程序还可直接编译生成 JavaScript 代码——Kotlin 程序既可编译成前端 JavaScript 代码，用于实现网页的 DOM 操作，实现前端编程；也可编译成后端 JavaScript 代码，与服务端技术（如 Node.js）交互。

不得不说的有一点是，目前 Android 已推荐使用 Kotlin 作为官方开发语言，这意味着 Kotlin 将会在 Android 开发中大放异彩，这也是笔者决定向读者介绍这门语言的重要原因之一。

1.1 Kotlin 语言简介

1.1.1 服务端的 Kotlin

Kotlin 程序可以编译成 Java 字节码文件，字节码文件可以直接在 JVM 上运行，因此 Kotlin 非常适合开发后端应用程序。Kotlin 与现有的 Java 语言包能保持完全兼容，这意味着 Kotlin 不是一门简单的语言，它完全可以利用 Java 领域现有的各种技术框架，如 Spring、Hibernate、MyBatis、Lucene 等，因此 Java 开发者非常容易过渡到使用 Kotlin。

总结来看，Kotlin 的显著优势有如下几点。

- ▶ **简洁性**：这是我们选择使用 Kotlin 的最大动力。Kotlin 具有大量现代编程语言的简洁性和便捷性，因此 Kotlin 被誉为 Android 平台的 Swift。
- ▶ **兼容性**：Kotlin 完全兼容 Java，因此 Kotlin 既是一门新的语言，也不是一门“全新”的语言，Kotlin 可以自由使用 Java 领域的无数框架和库。因此，开发者既可保持熟悉的技术栈，又可获得现代编程语言的优势。
- ▶ **迁移性**：Kotlin 支持大型项目从 Java 向 Kotlin 逐步迁移——项目主体部分继续使用 Java，新开发的功能可使用 Kotlin 编写，也可逐步使用 Kotlin 代替部分老旧的 Java 代码。

短期内，Kotlin 不会对 Java 造成巨大的冲击，但 Kotlin 简洁、优雅的语法可以对 Java 形成良好的补充，开发者可根据需要自由选择 Java 或 Kotlin，最终都会生成字节码文件，运行于 JVM 平台上。

1.1.2 使用 Kotlin 开发 Android 应用

Google 官方推荐使用 Kotlin 作为 Android 开发语言，证明了 Kotlin 非常适合开发 Android 应用。使用 Kotlin 开发 Android 应用可充分利用 Kotlin 的简洁性和便捷性。

Kotlin 完全兼容 JDK 1.6，因此保证了基于 Kotlin 开发的 Android 应用完全可以在较老的 Android 设备上运行。

对于广大的 Android 应用开发者而言，大部分时候都是与 Android 应用程序框架层交互的，调用 Android 应用程序框架层的 API，而 Kotlin 可以自由调用 Java 的各种类库，因此使用 Kotlin 调用 Android 应用程序框架层来开发应用程序甚至无须额外学习，开发者可以无缝地过渡为使用 Kotlin 开发。

对于性能方面来说，Kotlin 编译的字节码与 Java 原生字节码极为相似。随着 Kotlin 对内联函数的支持，使用 Lambda 表达式的代码通常比用 Java 写的代码运行得更快。

正是基于以上两点主要优势，Google 官方推荐使用 Kotlin 作为 Android 开发语言。

1.1.3 Kotlin 用于 JavaScript

Kotlin 程序还可以编译成 JavaScript 代码，Kotlin 程序会生成遵守 ECMAScript 规范的 JavaScript 代码。当选择生成 JavaScript 目标时，不仅会包括开发者自己写的 Kotlin 代码，也会包括 Kotlin 附带的标准库，它们都会转换为 JavaScript。

Kotlin 既可生成前端使用的 JavaScript 代码，也可生成后端使用的 JavaScript 代码。

当生成前端 JavaScript 代码时，Kotlin 可实现如下功能。

- Kotlin 提供了大量 API 来操作 DOM（文档对象模型），允许通过 DOM API 来动态创建和更新页面。
- Kotlin 也提供了支持 WebGL 的 API，因此可以在网页上用 WebGL 创建图形元素。
- Kotlin 也可使用现有的前端库和框架，如 jQuery 和 ReactJS 等。

当生成后端 JavaScript 代码时，Kotlin 完全可以与后端 JavaScript（如 Node.js）进行交互。

本书重点介绍 Kotlin 生成 JVM 字节码，这种 Kotlin 程序完全可以兼容 Java 程序，这也是 Kotlin 开发 Android 应用的基础。

1.2 使用命令行编译、运行 Kotlin

下面先介绍使用命令行环境编译、运行 Kotlin。与其他编程语言类似，Kotlin 同样需要下载并安装对应的 SDK，接下来即可使用该 SDK 提供的命令来编译、运行 Kotlin 程序。

1.2.1 下载和安装 Kotlin 的 SDK

下载和安装 Kotlin SDK 请按如下步骤进行。

① 登录 <https://github.com/JetBrains/kotlin/releases>，看到 Kotlin SDK 的最新发布版本，即可看到如图 1.1 所示的页面。本书成书之时，Kotlin SDK 的最新版本是 1.1.4-2，本书所有的案例也是基于该版本 SDK 的。



图 1.1 下载 Kotlin SDK 的页面

② 单击如图 1.1 所示页面中的链接，即可下载得到一个 kotlin-compiler-1.1.4-2.zip 压缩包文件。

③ 将该压缩包文件解压缩到任意目录下，即可看到如下文件路径。

- **bin**: 该路径下存放了 Kotlin SDK 的各种工具命令，常用的 `kotlinc`、`kotlin` 等命令就放在该路径下。
- **lib**: 该目录下包含了 Kotlin 的各种工具 JAR 包。
- **license**: 存放与 Kotlin 项目相关的各种授权文档。

在上面路径中，`bin` 是一个非常有用的路径，在这个路径下包含了编译和运行 Kotlin 程序的 `kotlinc` 和 `kotlin` 两个命令。除此之外，还包含了 `kotlinc-jvm`、`kotlinc-js` 等工具命令。其中 `kotlinc` 和 `kotlinc-jvm` 是一样的，都用于将 Kotlin 程序编译成适用于 JVM 的字节码文件，而 `kotlinc-js` 则用于将 Kotlin 程序编译成 JavaScript 代码。

④ 将解压缩路径下的 `bin` 目录添加到系统的 `PATH` 环境变量中，这样即可保证操作系统能通过 `PATH` 环境变量找到 `kotlinc`、`kotlin` 等工具命令。

⑤ 将解压缩路径下的 `lib` 目录中的 `kotlin-stdlib.jar` 文件（该文件与 `kotlin-runtime.jar` 文件相同，都代表了 Kotlin 的运行时环境库）添加到 `CLASSPATH` 环境变量中，这样即可保证运行 Kotlin 程序时能正常加载 Kotlin 的运行时环境库。



提示：

本书假设读者已有最基本的 Java 编程基础，因此书中不会逐步介绍环境变量配置这些基础内容。如果读者需要学习环境变量配置这些基础知识，则建议参考《疯狂 Java 讲义》一书。

➤➤ 1.2.2 第一个 Kotlin 程序

编辑 Kotlin 源代码可以使用任何无格式的文本编辑器，在 Windows 操作系统上可使用记事本 (NotePad)、EditPlus 等程序，在 Linux 平台上可使用 VI 工具等。



提示：

编写 Kotlin 程序不要使用写字板，更不可使用 Word 等文档编辑器。因为写字板、Word 等工具是有格式的编辑器，当使用它们编辑一个文档时，这个文档中会包含一些隐藏的格式化字符，这些隐藏字符会导致程序无法正常编译、运行。

在记事本中新建一个文本文件，并在该文件中输入如下代码。

程序清单：codes\01\1.2\helloWorld.kt

```
fun main(args: Array<String>) {
    println("Hello, World!")
}
```

编辑上面的 Kotlin 文件时，注意程序中粗体字标识的单词，Kotlin 程序严格区分大小写。将上面文本文件保存为 `helloWorld.kt`，该文件就是 Kotlin 程序的源程序，Kotlin 源程序的文件名要求以 `.kt` 结尾。

Kotlin 程序与 Java 程序不同，Kotlin 支持函数式编程，因此 Kotlin 程序只需要一个 `main()` 函数作为程序入口，不需要将该 `main()` 函数放在任何类中。

**提示:**

仔细看一下上面的 main() 函数, 不难发现这个 main 函数其实就是 Java 主类中的 main() 方法的变体。只不过在 Kotlin 语言中, 函数也是一等公民, 因此函数可以独立存在, 而 Java 主类中的 main() 方法必须放在类中声明。Java 主类中的 main() 方法需要声明一个 String[] args 形参, 而 Kotlin 的 main() 函数同样声明了 args: Array<String> 形参——它们都是一个字符串数组类型的形参, 这表明它们的本质是一样的, 只是形式不同而已。至于 Kotlin 主函数前面的 fun 关键字, 它专门用于声明函数; Swift 语言还使用 func 关键字声明函数呢, 其实没有什么特别的, 习惯不同而已。

编写好 Kotlin 程序的源代码后, 接下来就应该编译该 Kotlin 源文件来生成字节码文件了。

1.2.3 编译、运行 Kotlin 程序

编译 Kotlin 程序需要使用 kotlinc 或 kotlinc-jvm 命令 (两个命令完全一样), 因为前面已经把 kotlinc 命令所在的路径添加到了系统的 PATH 环境变量中, 因此现在可以使用 kotlinc 命令来编译 Kotlin 程序。

对于初学者而言, 先掌握 kotlinc 命令的如下用法:

```
kotlinc -d destdir srcFile
```

在上面命令中, -d destdir 是 kotlinc 命令的选项, 用以指定编译生成的字节码文件的存放路径, destdir 只需是本地磁盘上的一个有效路径即可; 而 srcFile 是 Kotlin 源文件所在的位置, 这个位置既可以是绝对路径, 也可以是相对路径。

通常, 总是将所生成的字节码文件放在当前路径下, 当前路径可以用一点 (.) 来表示。在命令行窗口进入 HelloWorld.kt 文件所在的路径, 在该路径下输入如下命令:

```
kotlinc -d . HelloWorld.kt
```

运行该命令后, 在该路径下生成一个 HelloWorldKt.class 文件。

与 javac 编译器类似, 使用 kotlinc 编译文件只需要指定存放目标文件的位置即可, 无须指定字节码文件的文件名。因为使用 kotlinc 编译后生成的字节码文件有默认的文件名: 如果 Kotlin 源程序中包含一个或多个函数, kotlinc 会额外生成一个文件, 该生成文件的文件名是源文件名首字母大写并添加 Kt 后缀, 以 .class 作为扩展名; 如果 Kotlin 源程序中包含一个或多个类, kotlinc 则会为每个类生成一个字节码文件, 文件名以源文件所定义各个类的类名作为主文件名, 以 .class 作为扩展名。

事实上, 指定目标文件存放位置的 -d 选项也是可以省略的, 如果省略该选项, 则意味着将生成的字节码文件放在当前路径下。

可能有读者会感觉到: 这不就是《疯狂 Java 讲义》所介绍的内容吗? kotlinc 的用法与 javac 的用法何其相似啊! 实际上这也是 Kotlin 设计者希望实现的效果: Java 是一门如此深入人心、影响深远的语言, 拥有广泛的程序员基础, 而 Kotlin 不仅借助于 JVM 作为运行平台, 而且保持和 Java 相似的编译、运行方式, 这样 Kotlin 也更容易被广大程序员所接受。

运行 Kotlin 程序可使用 kotlin 或 java 命令 (你没看错, 可以用 java 命令), 启动命令行窗口, 进入 HelloWorldKt.class 所在的位置, 在命令行输入如下命令:

```
java HelloWorldKt
```

运行上面命令，将看到如下输出：

```
Hello World!
```

这表明 Kotlin 程序运行成功。

当然，Kotlin 也专门提供了一个 `kotlin` 命令来运行 Kotlin 程序，因此在命令行输入如下命令：

```
kotlin HelloWorldKt
```

运行上面命令，同样可以看到如下输出：

```
Hello World!
```

1.3 使用 IntelliJ IDEA 编译、运行 Kotlin

Kotlin 本身就是 JetBrains 开发的，而 IntelliJ IDEA 则是 JetBrains 自家开发的 IDE 工具，因此 IntelliJ IDEA 必然支持 Kotlin 语言。

IntelliJ IDEA 是一个商业版软件，使用该软件的商业版是要收费的，但 JetBrains 还为 IntelliJ IDEA 提供了一个免费的社区版，因此本书会基于免费的社区版 IntelliJ IDEA 进行介绍。

登录 <https://www.jetbrains.com>，下载并安装最新的社区版 IntelliJ IDEA。

通过 IntelliJ IDEA 新建一个支持 Kotlin 的 Java 项目，请按如下步骤进行。

① 单击 IntelliJ IDEA 的“File”→“New”→“Project...”新建一个 Java 项目，并为新建的项目勾选“Kotlin/JVM”复选框，如图 1.2 所示。

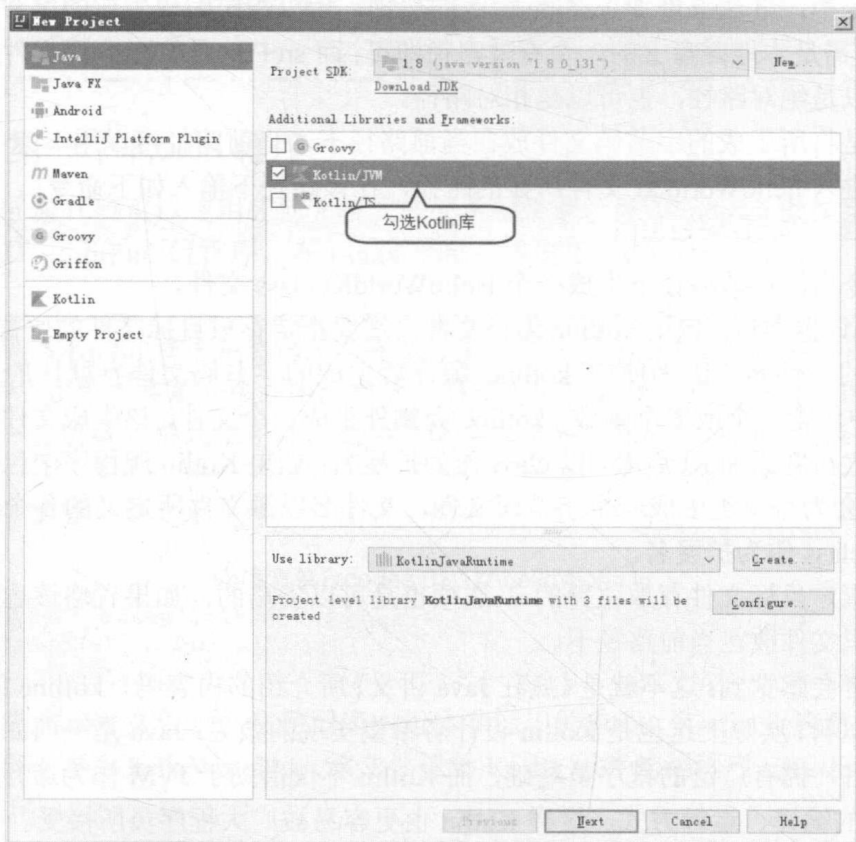


图 1.2 新建支持 Kotlin 的 Java 项目

② 让 Java 项目支持 Kotlin 的关键就是勾选“Kotlin/JVM”复选框，项目创建完成后可看到如图 1.3 所示的项目结构。

其实让 Java 项目支持 Kotlin 也很简单，无非就是需要两个条件：

- 该项目“知道”使用 kotlinc 编译器来编译 Kotlin 程序。
- 该项目包含了 Kotlin 的运行时环境。

正如从图 1.3 所看到的，上面项目中包含了 kotlin-stdlib.jar，这就是 Kotlin 的运行时环境库（即本章 1.2 节让大家添加到 CLASSPATH 环境变量中的 JAR 包），而 kotlin-reflect.jar 是 Kotlin 反射才需要的 JAR 包，kotlin-stdlib-jre7.jar、kotlin-stdlib-jre8.jar 是 Kotlin 为支持 Java 7、Java 8 提供的运行时环境库。此外，由于 IntelliJ IDEA 本身就是 JetBrains 自家的 IDE 工具，让它集成一个 kotlinc 编译器是顺理成章的事——这就是该项目能支持 Kotlin 程序的根本所在。

③ 右键单击图 1.3 所示项目结构中的 src 目录，在弹出的快捷菜单中选择“New”→“Kotlin File/Class”菜单项，如图 1.4 所示。

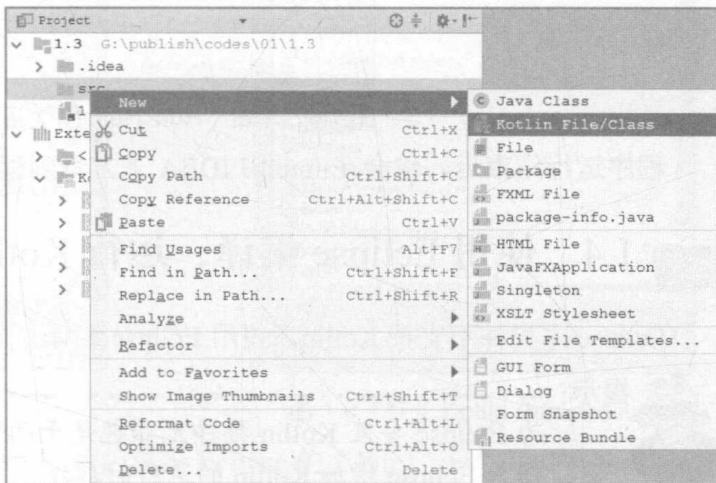
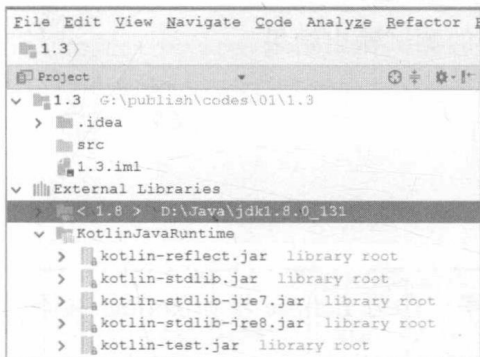


图 1.3 支持 Kotlin 的 Java 项目

图 1.4 新建 Kotlin 文件

④ 系统弹出如图 1.5 所示的对话框，在该对话框中可以选择 Kotlin 文件的类型。

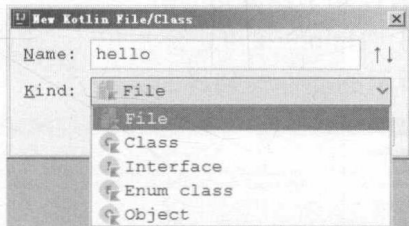


图 1.5 选择 Kotlin 文件的类型

对于熟悉 Java 的读者来说，对类、接口和枚举这些文件类型太熟悉了，它们都是 Java 程序员的老朋友。但在这里不打算选择这些老朋友，而是直接选择 File 类型——这表明将会新建一个普通的 Kotlin 文件。正如前面所介绍的，Kotlin 支持函数式编程，函数也是 Kotlin 的一等公民，因此此处将直接在 Kotlin 程序中新建主函数作为程序入口。

⑤ 在 hello.kt 文件中添加一个主函数，IntelliJ IDEA 提供了一个快速完成此操作的模板，只需输入 main，然后按 Tab 键，即可在编辑界面中看到添加了 main() 函数。

在 main() 函数中添加一行简单的输出语句用于测试。

⑥ 在 hello.kt 的编辑界面中单击鼠标右键，系统弹出如图 1.6 所示的快捷菜单，单击该菜单中的“Run 'HelloKt'”菜单项即可运行该程序。

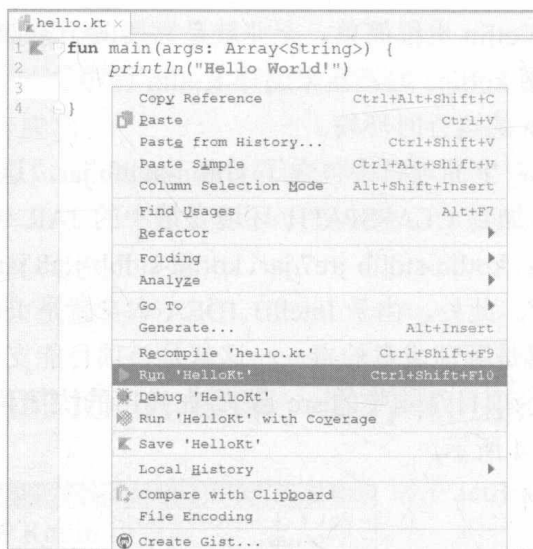


图 1.6 单击“Run 'HelloKt'”运行 Kotlin 程序

程序运行结束后，将会在 IntelliJ IDEA 下方看到程序的测试输出结果。

1.4 使用 Eclipse 编译、运行 Kotlin

Eclipse 本身并不支持 Kotlin，使用 Eclipse 编译、运行 Kotlin 需要安装插件。



提示：

为 Eclipse 安装 Kotlin 插件无非也是干两件事：①为 Eclipse 增加 kotlinc 编译器；②为 Eclipse 增加 Kotlin 的运行环境。

由于本书假设读者已有一定的 Java 基础，因此关于 Eclipse 的安装步骤本书就不介绍了。为 Eclipse 安装 Kotlin 插件请按如下步骤进行。

① 单击 Eclipse 主菜单中的“Help”菜单，然后单击“Eclipse Marketplace”菜单项，即可看到 Eclipse 弹出的如图 1.7 所示的插件安装界面。

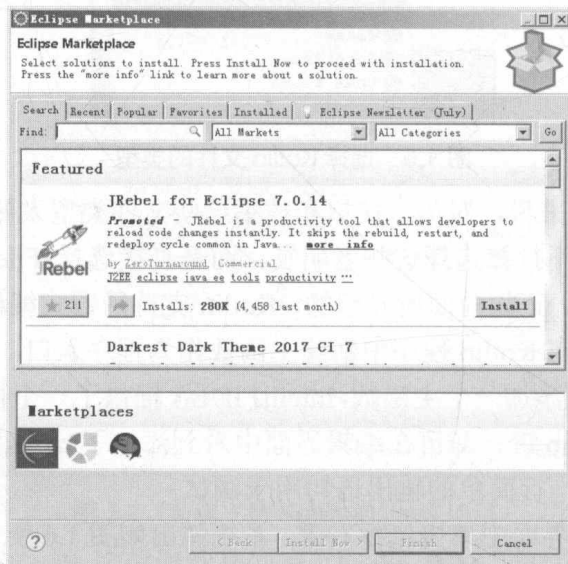


图 1.7 Eclipse 插件安装界面