移动开发人才培养系列丛书

基于Kotlin的Android应用程序开发

Building Android App Using Kotlin

薛岗 编著

- 从 **Kotlin 语言**介绍开始，由**浅入深**讨论 **Android 移动应用**开发过程中涉及的**关键技术**和**核心方法**
- **案例驱动方式**介绍相关程序的实现方法和过程，并通过**示例程序版本迭代**的方式，逐渐**引入新概念**
- 在技术讨论的同时，**拓展介绍**与**主题密切**相关的**背景知识**和**概念**

中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

移动开发人才培养系列丛书

基于Kotlin的Android应用程序开发

Building Android App Using Kotlin

薛岗 编著

人民邮电出版社

北京

图书在版编目（CIP）数据

基于Kotlin的Android应用程序开发 / 薛岗编著. --
北京 : 人民邮电出版社, 2019.4
（移动开发人才培养系列丛书）
ISBN 978-7-115-50098-4

Ⅰ. ①基… Ⅱ. ①薛… Ⅲ. ①移动终端－应用程序－
程序设计 Ⅳ. ①TN929.53

中国版本图书馆CIP数据核字(2018)第279971号

内 容 提 要

本书在云南大学软件学院“嵌入式应用开发实践”（本科）课程讲义的基础上，结合作者多年的教学和工程实践经验，并参考了国内外有关技术材料编写而成。全书内容分为 11 个章节，分别为：Kotlin 语言基础、Android 应用开发概述、多窗体应用、布局与界面交互组件、窗体类运行时的生命周期、列表与适配器、碎片技术、菜单与导航抽屉式界面、基于 SQLite 的数据持久化、应用服务和传感器。书中章节以案例驱动方式分析、讨论与 Android 应用开发相关的技术概念和实现手段；而且，全书以示例程序版本迭代的方式，逐渐深入介绍相关的核心技术和工程方法。

全书内容丰富，案例详实，讲解通俗易懂，并配有一定量的练习题。本书可作为高等院校信息技术相关专业移动应用开发类课程的教材，也可作为 Android 移动应用开发有关工程技术人员的学习或参考用书。

◆ 编　　著　薛　岗
责任编辑　刘　博
责任印制　陈　犇

◆ 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
北京隆昌伟业印刷有限公司印刷

◆ 开本：787×1092　1/16
印张：13.5　　2019 年 4 月第 1 版
字数：351 千字　　2019 年 4 月北京第 1 次印刷

定价：49.80 元

读者服务热线：(010)81055256　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京东工商广登字 20170147 号

前　言

Android 是 Google 公司开发并维护的一个移动操作系统平台。该平台以 Linux 内核为基础进行构建，系统构成中包含大量的开源软件工具。2007 年至今，Android 系统已经发布了多个版本；2018 年发布的版本 9（API 28）是当前最新的稳定版本。Android 开源项目（Android Open Source Project，AOSP）主要以 Apache 开源软件许可为基础，提供 Android 系统的核心程序，并能支持和帮助用户实现系统的定制和扩展。目前，Android 系统可以在智能手机、平板电脑、智能手表和电视、车载应用等环境中运行，而且，相关应用已经延伸到个人计算机、游戏主机、数码相机等领域。因此，Android 应用程序开发也是应用开发领域中较为热门的一个方向。

Android 应用开发一直以 Java 和 XML 为主要的开发语言。2017 年 Google 公司正式宣布 Kotlin 为官方支持的开发语言，这也为 Android 应用开发提供了另一种以 Kotlin 和 XML 为主的程序实现模式。Kotlin 语言具有简洁、安全、支持跨语言互操作等技术特征。Kotlin 语言的引入，提高了程序开发的效率，并在一定程度上简化了应用的实现过程。然而，新兴技术的引入也会促使工程人员改变他们的工作方式，这意味着，工程人员需要首先了解并掌握有关的技术和方法，才能在工程实践中合理使用该新兴技术。鉴于此，本书以 Kotlin 为基础，分析讨论 Android 应用程序开发相关的方法和技巧，并期望通过有关内容展示 Kotlin 的技术特点，总结基于 Kotlin 构建 Android 应用程序的技术方法。

本书所涉及的技术包含 Kotlin 语言、Android 界面实现、多线程编程、数据持久化、应用服务和传感器技术等，主要内容被组织成 11 个章节。其中，薛岗老师主要完成第 1 章至第 10 章的内容，周维老师负责完成第 11 章的内容。云南大学软件学院硕士研究生武丽雯、王佳伟、刘峻松、刘惠剑负责完成本书附录、章节练习题的设计和习题参考答案的整理工作；其中，刘惠剑完成附录 A、附录 B、第 1 章至第 2 章的习题设计与参考答案的整理工作；王佳伟完成第 3 章至第 6 章，及第 11 章的习题设计与参考答案的整理工作；刘峻松完成第 7 章至第 10 章的习题设计与参考答案的整理工作；武丽雯负责完成全书所有章节练习题的审查与参考答案的校对工作。另外，杨亦昆同学参与了本书的校稿工作。

本书在编写和修订过程中，得到了云南大学软件学院姚绍文教授、刘璟老师的指导和关注；同时，本书的出版得到了人民邮电出版社高等教育出版分社刘博等老师的大力支持和帮助。在此向他们表示诚挚的感谢。

本书由云南大学“工业 4.0 及云南的对策研究”（编号：KS161006）项目支持出版。由于作者水平和经验有限，书中缺点、不足在所难免，恳请读者批评指正。

编　者

2018 年 4 月

目 录

第1章 Kotlin 语言基础

Kotlin 是一种静态类型编程语言（它由 JetBrains 公司的一个开发小组实现，该语言的命名来源于俄罗斯圣彼得堡附近 Kotlin 岛的名称），该语言可在 Java 虚拟机（JVM）上运行，并可被编译成 JavaScript 源程序[1]。Kotlin 语言在使用时可直接调用 Java 类库，并具有与 Java 程序进行互操作的能力。除了能作为通用程序开发的工具，Kotlin 语言在开发 Android 应用程序方面具有得天独厚的优势。2017 年 Google I/O 大会上，Kotlin 语言被认定为官方支持的 Android 应用程序开发语言之一。

Kotlin 语言具有简洁、安全、支持跨语言互操作等技术特征。与传统开发语言相比较，Kotlin 语言的使用可在一定程度上提高程序开发的效率；同时，通过该语言实现的程序可避免出现诸如“空指针”等技术错误。本章的后续内容主要介绍 Kotlin 语言，相关内容组织为 4 个部分，分别为：①基本语法；②方法与 Lambda 表达式；③类与对象；④泛型、对象表达式和代理。

本章所讨论的程序均使用 IntelliJ IDEA 环境创建、运行。IntelliJ IDEA 是一个集成开发环境，该软件由 JetBrains 公司开发并维护。IntelliJ IDEA 有两个发布版本：社区版（Community Edition）和商业版（Commercial Edition）；其中，社区版遵循 Apache 开源协议（版本 2），可免费下载和使用。

1.1 基本语法

Kotlin 语言支持较为自由的程序编写风格，程序可采用面向对象或面向过程的方式进行编写。在程序编写过程中，程序文件的名称可根据实际情况任意指定，同时，程序文件的扩展名为 kt。Kotlin 程序运行的起点为 main 方法（或称为 main 函数）。

以下示例是一个简单的 Kotlin 程序。该程序运行时可通过打印语句在输出窗口中显示一个“Hello World!”字符串。

```
1  package niltok.demos
2
3  fun main(args: Array<String>){
4      println("Hello World!")
5  }
```

上述程序包含两个部分：包（程序第 1 行）及 main 方法声明（程序第 3 行至第 5 行）。其中，包声明使用 package 命令实现，而 main 方法声明实质是 main 方法的一个定义。在 main 方法声明中，fun 为方法（或函数）声明关键字，main 为方法的名称，args 为 main 方法的输入参数。在输入参数方面，“args: Array<String>”语句说明 args 是一个字符串数组，数据类型为 Array<String>。示例程序 main 方法中的 println 语句是一个打印语句，该语句可在输出窗口中打印一个指定的字符串。

有的情况下，main 方法的输入参数 args 可被忽略，但该参数可用于传输与程序运行有关的多个数值。在程序运行前，参数 args 中数值的输入需借助命令行工具，以手工输入方式来指定。而在程序运行时，main 方法中的程序可访问并使用 args 所包含的数值。以下为一个访问 args 参数的简单示例：

```
1   package niltok.demos
2
3   fun main(args: Array<String>){
4       println("Inputs:") //显示提示信息
5       println(args.size) //打印显示 args 的长度
6       for (i in args){ //访问 args
7           println(i)
8       }
9   }
```

上述示例中，程序第 5 行是打印显示 args 参数的长度，第 6 行至第 8 行则使用 for 语句遍历 args 参数中的元素，并将每个元素进行打印显示。

在 IntelliJ IDEA 中，若要指定 args 中的数值，可在系统菜单中单击“Run”项，并在显示的菜单中选择“Run”（也可在集成开发环境中直接使用快捷键“Alt+Shift+F10”）项；之后，开发环境会显示一个对话框，在对话框中选择“Edit Configurations…”项，系统显示一个标题为“Run”的配置向导（对话框）。向导的“Configuration”（配置）标签页中，可在“Program arguments:”项中指定程序所需参数（即设置 args 所包含的多个数值）。输入参数设置时，参数间使用空格作为分割，例如，若想在 args 中填写两个参数“123”和“456”，所填写的内容为 123 345。参数填写完毕，单击向导中的“Run”按钮，程序开始运行。以输入参数为“123”和“456”为例，示例程序运行的结果如下。

```
1   inputs:
2   2
3   123
4   345
```

在语法方面，Kotlin 程序的每个语句不使用结尾符（这与传统语言不同）。Kotlin 程序中，只读变量的使用场景相对广泛。只读变量在声明时使用 val 关键字进行说明，一旦某个变量被指定为只读变量，则在程序运行过程中，该变量的值是不允许被修改的。与只读变量不同，可变更变量（即普通变量）的值可以在程序运行过程中根据需要而改变。可变更变量在声明时使用 var 关键字进行说明。

Kotlin 程序中，变量声明使用的格式为**变量名: 变量类型**。例如，声明一个整型只读变量 i

时，声明语句为 val i: Int；当声明一个整型变量（普通变量）j 时，声明语句为 var j: Int。

Kotlin 程序中的程序注释基本格式与 C 语言或 Java 语言相同，即使用符号//和/*…*/。其中，//符号用于实现单行注释，而/*…*/则可实现多行注释。

1.1.1　基本数据类型

Kotlin 语言支持的基本数据类型包含[2]数字型、布尔型、字符和数组。

（1）数字

Kotlin 内置的数字类型如表 1.1 所示，具体包含双精小数（Double）、单精小数（Float）、长整型（Long）、整型（Int）、短整型（Short）和字节（Byte）。

表 1.1　　Kotlin 内置数字类型及物理存储长度

类型	Double	Float	Long	Int	Short	Byte
长度（位数）	64	32	64	32	16	8

在未特别说明的情况下，Kotlin 程序中的小数数值默认为 Double 类型。若需要指定 Float 类型的小数数值时，可使用格式“**小数数值 f**”或“**小数数值 F**”。例如，当需要指定 123.4 为单精小数时，程序中可使用 123.4F。对于整数，Kotlin 暂不支持八进制整数，其他的整型数字按以下规则表示。

- 对于普通十进制整数，使用普遍格式，例如：321；
- 对于长整型（十进制），使用格式为**数字 L**，例如：321L；
- 对于十六进制整数，格式为 **0x 数字**，例如：0xFF；
- 对于二进制整数，格式为 **0b 数字**，例如：0b001。

另外，Kotlin 支持以下画线来分割数字，如：1_2345_6789。

（2）类型转换

Kotlin 程序中，位数短的数据类型数据不能直接转换成为位数长的数据类型数据（这个技术特点与 Java 语言的特点不同），例如，下列程序将无法运行。

```
1   val i: Int = 100 //只读变量 i，类型为整型
2   val l: Long = i //本句无法运行，因为系统中 Int 类型数据比 Long 类型数据所占的位数短
```

在程序编写过程中，对于不同类型的数字，它们之间的转换可以显式方式实现。具体的转换可借助以下函数[2]：

- 转换为字节型（Byte），使用 toByte()，例如：10.toByte()；
- 转换为短整型（Short），使用 toShort()，例如：(12.34).toShort()；
- 转换为整型（Int），使用 toInt()，例如：(12.23).toInt()；
- 转换为长整型（Long），使用 toLong()，例如：(1234.56).toLong()；
- 转换为单精小数（Float），使用 toFloat()，例如：123.toFloat()；
- 转换为双精小数（Double），使用 toDouble()，例如：123.toDouble()；
- 转换为字符（Char），使用 toChar()，例如：123.toChar()。

（3）数学运算

Kotlin 语言能实现多种数学运算，其中，基本运算包含：+（加）、–（减）、*（乘）、/（除）、

%（求模）。Kotlin 语言中的*运算符还可作为传值符号，支持将一个数组赋值给一个包含可变长输入参数的方法。

Kotlin 语言还为常见数学运算提供了实现方法，这些方法可被调用，并能完成相应计算任务。例如，位运算可通过以下方式实现[2]。

- shl(bits)，类似 Java 的<<运算，是带符号位左移运算；
- shr(bits)，类似 Java 的>>运算，是带符号位右移运算；
- ushr(bits)，类似 Java 的>>>运算，是无符号位右移运算；
- ushr(bits)，类似 Java 的<<<运算，是无符号位左移运算；
- and(bits)，位上的 and（和）运算；
- or(bits)，位上的 or（或）运算；
- xor(bits)，位上的 xor（异或）运算；
- inv()，位上取反。

（4）字符

Kotlin 语言中，字符使用类型声明符 Char 进行说明，字符数据必须使用单引号来表示；例如，将字符'a'赋值给一个变量 *c*，实现语句为：var c: Char = 'a'。区别于 Java 语言，Kotlin 语言中的一个字符不能被当作数字来直接使用，但可使用 toInt()方法实现从字符到整型的转换。对于特殊字符，可使用转义符：\；例如：\t（制表）、\b（退格）、\n（换行）、\r（回车）、\'（单引号）、\"（双引号）、\\（斜杠） 和 \$（美元符号） 等；此外，还可使用 Unicode 转义语法，例如：'\uFFFF'。

（5）布尔型数据

Kotlin 语言中的布尔型数据类型为 Boolean，基本的取值为：true（真）和 false（假）。对于布尔型数据的运算，Kotlin 语言包含：||（或运算）、&&（与运算）、!（否运算）等。

（6）数组

Kotlin 语言中的数组基于 Array 类实现，Array 类中常用的操作包含[2]：size（数组元素个数）、set（设值）、get（取值）等。创建数组使用 arrayOf 或 arrayOfNulls 方法。例如，创建一个字符串数组，且数组包含的元素有：{"this", "is", "an", "array", "of", "Strings"}，则使用 arrayOf("this", "is", "an", "array", "of", "Strings")语句创建字符串数组；另外，当需要定义一个空的字符串数组时，可使用 arrayOfNulls<String>语句。在实际程序中，这些方法的使用如下列示例所示。

```
1  package niltok.demos
2
3  fun main(args: Array<String>){
4      var strs1: Array<String> = arrayOf("this", "is", "an", "array", "of", "Strings")
5      var strs2: Array<String?> = arrayOfNulls<String>(2)
6      println(strs1[0]) //显示第一个字符串的第一个元素
7      println(strs2.get(0)) //显示第二个字符串的第一个元素
8  }
```

运行时，上述程序初始化了两个字符串数组 strs1 和 strs2（程序中的 var 为变量定义说明符）。其中，strs2 是一个长度为 2 的空字符串数组。程序第 6 行将打印显示 strs1 中的第 0 位元素（即“this”），而程序第 7 行将打印显示 strs2 中的第 0 位元素，实际的结果为空（“null”）。程序中的数组可使用“[]”操作符来实现基于位置的元素访问，例如，strs1[0]表示获取 strs1 数组中的第 0 号元素。

需要特别注意的是，Kotlin 程序在声明变量时，如果变量类型后使用了符号“?”，则表示该变量可为空；但若变量类型后未使用“?”符号，则该变量不能被赋空值（null）。

Kotlin 语言的类库中还为基础数据类型定义了特定的数组类，如 ByteArray（字节型数组）、ShortArray（短整型数组）、IntArray（整型数组）等。这些类与 Array 类的使用方法类似。

数组初始化可基于“工厂函数”实现。例如，下列示例程序中的“{i->i+1}”为一个工厂函数。在程序第 1 行中，Array(5, {i->i+1})语句第 1 个参数用于指定数组的长度，程序中使用了数值 5（数组中，实际元素的位置索引则为 0，1，2，3，4）；而 Array(5, {i->i+1})语句中的第 2 个参数{i->i+1}可实现这样的功能：将元素索引值加 1，并将值设置为数组中对应元素的值，即 i 的取值范围为 0 至 4，而数组中对应元素的数值为 i+1。

```
1   var ins = Array(5, {i->i+1})
2   for (i in ins){
3       println(i)
4   }
```

（7）字符串

Kotlin 程序中的字符串为 String 类型，字符串为不可变更的数据类型。字符串中的字符可通过字符元素的位置进行访问；字符串中可使用转义字符；另外，可使用 3 个双引号（“”“…”””）来表示一个自由格式的字符串，如多行字符串。

Kotlin 程序中的字符串可使用模板表达式，基本格式为**$标识名**。下列程序展示了模板表达式的使用：

```
1   fun main(args: Array<String>){
2       val i = 9
3       val s = "$i is in the string" //$i 为一个模板表达式
4       val s1 = "\'$s\'.length is ${s.length}" //$s 和${s.length}为模板表达式
5       println(s)
6       println(s1)
7   }
```

上述程序中除了$i 和$s 为基本的模板元素，${s.length}是基于模板来显示一个操作结果。${s.length}中，s.length 是一个运算操作，含义是获得字符串 s 的长度，而${s.length}则将实际的结果组织到字符串 s1 中。上述程序运行结束，$i 位置显示 9，$s 位置显示字符串“9 is in the string”，${s.length}为 s 字符串的长度 18。

（8）空值

程序中可使用空值 **null**。当变量、常量、参数或返回值中可包含空值时，在声明时必须使用符号“?”。例如，var a: Int? 语句说明变量 a 是可为空的整型变量。需要特别说明的是，在程序中，当变量、参数或返回值声明中未使用?号，则表示该变量、参数或返回值不能为空，否则相关程序语句为非法语句。空值的检查可使用比较符==，该比较符所计算的结果为布尔值，如 if (a == null){…}。

（9）数据类型的检查与转换

程序中，数据类型检查使用操作符 is（是）或 !is（不是），其中，!is 是 is 的否操作。例如，当检查变量 a 是否为一个整型时，使用 if (a is Int){…}。针对空值 null，is 或!is 操作无效，可使用

==完成相关的比较操作。

类型转换可使用操作符 as。若类型转换过程中可能会发生违例的情况，则这样的类型转换被称为不安全转换；例如，当变量 *a* 为 null（空值）时，var b: Int = a as Int 为不安全转换，可使用 var b: Int? = a as Int? 进行控制。另外，可使用 as? 操作符号进行安全转换。

Kotlin 程序中的数据类型会根据具体情况进行类型的智能转换。智能转换主要指无须直接使用类型转换操作符的情况，例如，println(1)语句中，整型数据被智能转换为字符串。

1.1.2 包

Kotlin 中关于“包”的概念与 Java 中的“包”相似。在程序中，package 命令是用来声明程序包的信息，而 import 则是用来加载程序包的命令。

在应用程序中，“包”技术主要用于建立程序的名称空间，并避免在不同范围内的同名概念之间可能会产生的冲突。例如，A 组织在开发程序时定义了 Person 类，B 组织开发程序时也使用 Person 作为类名；若不使用名称空间，当 A 组织的 Person 类和 B 组织的 Person 类在同一个程序中工作时，这两个同名类会发生概念冲突；因为，程序执行工具无法正确区分两者之间的区别。面对这样的问题，可使用名称空间技术来解决问题。例如，A 组织的程序定义包 a，B 组织定义包 b，当两个类在同一个程序中相遇时，实际上它们被解释为 a.Person 和 b.Person；这样，两个类在同一个程序中可正常工作，也有效地解决了同名概念所引起的冲突问题。

包在声明时，建议基于程序编制单位的网址来进行命名，例如，假设程序员隶属于 A 组织，网址为 a.test，而当前正在开发的项目名称为 DEMO，则包的命名可以为 test.a.demo。

Kotlin 平台本身存在大量预定义的程序包；另外，由于 Kotlin 可与 Java 程序相互协作，编写程序时，可加载技术环境中可用的 Java 程序包。Kotlin 应用程序在编写和运行时，系统预加载包包含 java.lang.*、kotlin.jvm.*、kotlin.js.*等（其中，符号*表示“所有类库包”），而预加载的开发类库包含[2]：

- kotlin.*
- kotlin.annotation.*
- kotlin.collections.*
- kotlin.comparisons.*
- kotlin.io.*
- kotlin.ranges.*
- kotlin.sequences.*
- kotlin.text.*

1.1.3 程序的控制结构

Kotlin 程序中常用的控制结构包含 if 结构、when 结构、for 循环、while 循环。其中，if 和 when 可作为表达式直接使用。

（1）if 结构

if 结构的基本格式如下：

```
if (条件){
    程序语句 1
    ...
```

```
}else{
    程序语句 2
    …
}
```

if 结构执行时，首先对条件部分进行判断；若条件判断为真，则从“程序语句 1”开始执行，当程序执行至第 1 个“}”时结束；若条件判断为假，则从“程序语句 2”开始执行，当程序执行至第 2 个“}”时结束。Kotlin 程序可将 if 结构作为表达式，放在赋值语句的右侧，例如，下列语句将结构运算结果直接赋值给变量 value：

```
var value = if (a>b) {a} else {b}
```

（2）when 结构

when 结构类似 Java 中的 switch 语句，基本格式为：

```
when (变量){
    值 1 -> 语句 1
    值 2 -> 语句 2
    …
    else -> 语句 n
}
```

上述结构在运行时，基本的工作过程为：程序首先对“when (变量)”部分中的“变量”进行计算判别，并将结果与结构中的分支条件（即结构中箭头的左侧部分）进行匹配，若某分支条件匹配成功，则该分支所对应的程序语句执行（箭头右侧代码）。when 结构在执行过程中，只要一个分支条件被执行，则其他分支条件将不再参与匹配运算；另外，when 结构可指定默认执行程序，默认执行程序的分支判断条件为 else，也就是说，若其他任何一个分支条件都不满足时，else 条件满足，所对应的程序开始工作。

when 结构中的分支条件可以使用逗号进行组合，当使用逗号时，两个分支条件之间的关系类似于条件间的“或”关系；另外，分支条件中可以使用表达式，或者 in、!in、is、!is 等操作符（其中，!is 是 is 的否操作，!in 是 in 的否操作）。例如，分支条件可以是“条件 1，条件 2”“in 范围”“!in 范围”“is 类型”“!is 类型”等。

（3）for 循环

for 循环用于控制程序代码段的多重循环，最常见的应用场景为数据或对象集合的遍历。Kotlin 中，for 循环的常见结构为：

```
for (变量 in 集合){
    关于变量的执行语句
}
```

注意，传统 for(…; …; …)结构在 Kotlin 中已不被支持。

上述结构中，程序段循环的次数基于“集合”中的元素个数确定。需要注意，上述结构可以运行的基本条件为，集合对象必须提供内置的迭代器（iterator）。对于一个数组，若想通过数组索引（元素位置）来访问数组，可使用数组实例的 indices 属性来获得索引集合，例如：

```
for (idx in numbers.indices){
    //执行程序
}
```

另外，可使用数组实例的 withIndex 方法获取数组中的键值对，例如：for ((k, v) in strs.withIndex()){...}。

（4）while 循环

Kotlin 中可使用两种 while 循环，基本结构为：

```
while (判断条件){ //while 循环结构 1
    执行语句
    ...
}

do{ //while 循环结构 2
    执行语句
    ...
} while (判断条件)
```

上述结构中，第 1 个结构的工作原理为：程序首先判断 while 的判断条件；当条件满足时，运行结构中的程序语句；当语句执行结束，程序再次进行条件判断，若条件满足，则继续执行结构中的程序语句，直到判断条件不满足为止；最后，while 循环结束。第 2 个结构的工作原理为：程序首先执行结构中的程序语句（即 do{...}中的所有语句），语句执行完毕，对 while 条件进行判断，若条件满足则再次执行结构中的程序语句；这样的过程一直到 while 判断条件不被满足为止。

1.1.4 返回值与循环结构的跳转

当方法或函数需要返回值时，程序语句中需要使用 return 命令，例如：return 123。

循环结构的跳转主要包含两个命令，即 break 和 continue。其中，break 命令是终止当前循环；continue 是跳出当前循环，继续后续循环。下列示例程序展示了 break 和 continue 的工作原理：

```
1   fun main(args: Array<String>){
2       for (i in 1..10){
3           println("index: " + i.toString())
4           if(i == 2)
5               continue
6           println("after continue")
7           if(i == 4)
8               break
9           println("after break")
10      }
11  }
```

上述程序使用 for 结构遍历 1 至 10 之间的数字。程序在变量 i 为 2 时，由于使用了 continue 命令，该语句之后的语句都不会执行；随后 i 为 3，程序继续执行其他语句；当 i 为 4 时，程序第 8 行使用了 break 语句，则该语句的后续语句不会被执行，且循环被终止。程序运行的结果为：

```
1   index:1
2   after continue
3   after break
4   index: 2
5   index: 3
```

```
6    after continue
7    after break
8    index: 4
9    after continue
```

1.1.5　集合类型

除了数组结构外，Kotlin 中的集合类型包含列表（List）、集合（Set）、字典（Map）等；Kotlin 中，集合类型分为可修改和只读两种[2]。

列表结构类似于数组，但与数组相比较，列表的长度大小可在程序运行时被动态调整。列表中的元素必须为相同类型，而且，在一个列表中可以存在多个值相同的元素。与列表相比，集合（Set）类型是多个相同类型元素的一个集合，但集合中的元素不允许重复。字典类型的结构相对复杂，该类型中的元素按“键-值”对方式进行组织；每个元素具有“键”值和“值”项两个部分，其中，该键值用于标识一个元素，而“值”项则用于存储该元素的具体数值。

Kotlin 中，只读列表基于 List<T>接口定义，可修改列表基于 MutableList<T>定义；类似，Set<T>为只读集合，MutableSet<T>为可修改集合；Map<K, V>为只读字典，MutableMap<K, V>为可修改字典。

初始化集合类型时，推荐直接调用系统提供的标准方法：listOf、mutableListOf、setOf、mutableSetOf、mapOf、mutableMapOf 等。复制一个集合类型的数据，可使用的方法为 toMap、toList、toSet 等。

以字典为例，若想创建、访问并扩展一个具有 3 个元素的字典，相关程序如下：

```
1    fun main(args: Array<String>){
2        val m = mutableMapOf("k1" to "v1", "k2" to "v2", "k3" to "v3")
3        println(m.get("k2"))
4        m.put("k4", "new value")
5        println(m.get("k4"))
6    }
```

上述程序中，第 2 行使用 mutableMapOf 创建一个字典，该数据结构初始化时具有数据{“k1: v1”, “k2: v2”, “k3: v3”}；mutableMapOf 中，一个键值对按“键 to 值”方式进行声明；第 3 行，程序访问字典（结构）中键为“k2”的值，并进行打印显示；第 4 行，程序在结构中增加一个数据项“k4: new value”；第 5 行，程序访问字典（结构）中键为“k4”的值，并进行打印显示。程序运行的结果为：

```
1    v2
2    new value
```

1.1.6　数值范围

Kotlin 可以直接使用数值范围表达式：..（两个点）。例如，1..10 表示范围 1 至 10（整数）。在 for 循环中使用范围时需要注意，for (i in 1..10)是可工作的，但 for (i in 10..1)是不可工作的。当范围起始值大于终止值时，可使用类似于 for (i in 10 downto 1)的语句来进行程序控制。在循环语句中使用范围表达式还可控制变量访问的步长，例如 for (i in 1..10 step 2)，表示从 1 开始每次前进 2 步，至 10 终止。另外，当不需要使用某个范围的终止值时，可使用关键字 until，例如 for (i in 1 until 10)，表示数值范围是从 1 开始，并至 9 终止。

1.1.7 等式

Kotlin 可使用两种等式运算符：===和==；其中，==用于值或结构相等关系的判断（!=为对应的不相等关系的判断）；===用于应用对象相等关系的判断（!==为对应的不相等关系的判断），例如，在下列语句中，===被用于对象直接的比较判定：

```
1   var o = MyClass()
2   var oo = o
3   oo === o //本语句为真
```

1.1.8 操作符

Kotlin 基本的操作符号包含以下几种。

- 一元前缀操作符：+（正）、–（负）、!（非）；
- 递增、递减：++和--，例如：a++或 a--；
- 数学操作符：+（加号）、–（减号）、*（乘号）、/（除号）、%（取模）、..（范围）等；
- 在范围中进行查询或遍历的 in 操作符：in 和!in；
- 数组基于位置索引的访问符：[]，例如：a[i]、b[i, j]、c[i_2]等；
- 扩展赋值符：+=（累加）、–=（累减）、*=（累乘）、/=（累除）、%=（累积取模），例如：a += b 与 a = a+b 相同；
- 比较操作符：==（等）、!=（不等）、<（小于）、>（大于）、<=（小于等于）、>=（大于等于）。

1.1.9 其他操作符

Elvis 操作符格式为：**被判断对象 ?: 返回值**。例如：n ?: “nothing”语句与 if(n!=null) n else “nothing”等价；再例如：o?.length ?: 0 语句与 if(o!=null) o.length else 0 的含义相同。

另外，!!操作符会对被操作对象进行检查，如果该对象为空值时，操作符会掷出违例，例如：s!!.length 语句中，如果 s 为空，则程序会产生违例。

1.1.10 违例处理

在应用程序开发、运行过程中，违例是在所难免的。所谓违例是指程序运行过程中可能会发生的错误。违例产生原因有：①程序语句使用错误；②运行过程中，程序运行的外部条件不能满足程序运行的需求而引发的执行错误等。

Kotlin 程序中的所有违例类从 Throwable 类继承。程序运行时，一个违例对象包含了关于违例的描述信息，具体包含：错误、程序堆栈信息和错误原因。在编写程序时，违例的掷出需要使用 throw 命令，例如：throw MyException(“messages”)语句在执行时会产生一个 MyException 类型的违例。程序中，throw 命令为特殊类型 Nothing；如果某方法在定义时，只实现了 throw 语句，则该方法的返回值可使用类型 Nothing。

Kotlin 中，违例处理的结构与 Java 语言中的违例处理类似，基本结构为：

```
try{
    操作语句
    ...
```