Pearson

[美] G．布莱克·梅克（G．Blake Meike） 著
师蓉 译

Android 并发开发

Android Concurrency

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

Pearson

[美] G．布莱克·梅克（G．Blake Meike） 著
师蓉 译

Android 并发开发

Android Concurrency

人民邮电出版社
北京

图书在版编目（CIP）数据

Android并发开发 / （美）G.布莱克·梅克（G. Blake Meike）著 ; 师蓉译. -- 北京 : 人民邮电出版社, 2018.10
ISBN 978-7-115-48961-6

Ⅰ. ①A… Ⅱ. ①G… ②师… 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169661号

版权声明

◆ 著　[美] G. 布莱克·梅克（G. Blake Meike）
译　师　蓉
责任编辑　吴晋瑜
责任印制　焦志炜

◆ 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
固安县铭成印刷有限公司印刷

◆ 开本：800×1000　1/16
印张：12.75
字数：278 千字　2018 年 10 月第 1 版
印数：1－2 400 册　2018 年 10 月河北第 1 次印刷

著作权合同登记号　图字：01-2016-7582 号

定价：49.00 元

读者服务热线：(010)81055410　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

内容提要

本书共 8 章。第 1 章介绍一个非典型的并发模型，为后文的阐释做好铺垫。第 2 章和第 3 章分别介绍 Java 并发和 Android 应用程序模型，主要介绍 Java 线程、同步、并发包、生命周期和组件、Android 进程等基本概念。第 4 章介绍 AsyncTask 和 Loader。第 5 章～第 7 章是本书的核心内容，深入探讨 Android 操作系统的细节，如 Looper/Handler、Service、Binder、定时任务等。第 8 章介绍并发工具，如静态分析、注解、断言等。

本书适合有一定 Android 开发经验的读者阅读。如果你是一名新手，建议你在掌握相关入门知识的基础上阅读本书，以达到更好的学习效果。本书给出了多段代码，旨在让读者亲自实践后更好地掌握 Android 并发开发的相关内容。

前 言

我有幸在这个行业工作多年，看到了很多背景下的并发。我在学校读书时，并发只是一个论文主题。作为一名熟练的开发人员，我看到分布式后端系统中的并发代码几乎都是用 Java 编写的。最近，我有机会亲身体验 Erlang 和 Scala 这样的语言，希望能使并发代码的设计和编写更容易。

在我职业生涯早期，一位非常支持我的面试官指导我重新设计了双重检查锁。我还记得自己在一年后发现双重检查锁模式不安全，并在不久后遇到了第一个不正确的字节码时的那种心情。然而，最令人惊讶的也许是我最近从 2015 年编写的代码中删除了双重检查锁模式的那次实践。

在此期间，不变的是这个话题的神秘性和争议。必须使用并发代码时，即便是能够完美胜任工作的老手也会突然犯很幼稚的错误。开发人员有时会就某段并发代码的正确性产生争议（偶尔会很强烈）。他们可能会争论好几个小时，最后不可避免地终结于脆弱的细节，谁赢谁输却是无关紧要的。

必须承认的是，在那场面试中解决“双重检查锁”时，我感到很开心。最近，在听某些演讲者讲述一些快速而松散的并发技巧（通常是完全错误的）时，我想我在他们和听众脸上看到了同样的兴奋。

找到一个非常聪明的算法和一种非常快乐的感觉是很美妙的事情。如果我们能摒弃并发编程的神秘和魔法，就可以编写出更好的代码。如果并发代码的正确性是两个开发者（甚至是两个具有不同兴趣的开发者）都同意的东西，那就太好了！

读者对象

本书是为那些具有 Android 开发经验的开发者准备的。

如果你是一名新手开发人员，你可能会发现一些不熟悉的术语和概念。如果你是正在开发自己第一个 Android 应用程序的开发人员，你可能更关心如何简单地熟悉 Android 框架。

如果你属于上述类型中的一种，本书暂时不适合你！如果真是这样，我建议你先把本书放在一边，去学习一些基础的东西，当你完成了一个完整的 Android 应用程序后再阅读本书。

我写这本书是为了让读者阅读的，虽然这听起来有点像是废话。它既不是一本教科书，也不是一本参考手册。我鼓励你自己试一下示例代码。这些示例只是沙箱实验，你可以试着扩展它们。你可以借此进行新实验，以形成自己对 Android OS 细节的理解。然而，我不希望你把这本书放在笔记本旁边来从中获取价值。

我有一段时间着迷于 Perl。回想一下，我认为主要的原因是发现那本“骆驼书”《Learning GrassMudHorse Programming Language》太有意思了（不是粉色的那本，是蓝色的那本——第 2 版）。我回想起很多远离键盘的快乐时光，脑子里没有具体的应用程序，只是单纯地阅读那本书。

我不是要拿自己和 Larry Wall 比较，也不是要用什么方式比较 Android 和 Perl。我只是希望你能喜欢阅读这本书。我希望这本书会是你坐飞机或者长途通勤路上的好伙伴。

本书内容结构

本书的前 3 章为你提供了一次复习机会。

我建议你至少不要跳过第 1 章。我在这一章介绍了一个非典型的并发模型，这个并发模型是后文讨论的基础。

我故意将第 2 章和第 3 章设计得非常短。这两章是对一些基本概念的复习，并给你提供了一个重新认识一些常见用法的机会。有经验的开发者可以选择跳过这两章。

第 4 章是一个警示故事。

本书的核心是第 5 章～第 7 章。这 3 章深入探讨了 Android 操作系统的细节。

第 8 章是“餐后甜点”—— 一些并发工具的指南。

示例代码

本书示例中的大部分代码可以在 GitHub 上找到。如果你在实验这些代码的过程中发现了一些有趣的东西，一定要和其他人分享。

为方便下载、更新和修正，我建议你在 informit 官网上注册自己所购买的这本书。要开始注册过程，请先登录或者创建一个账户。输入 ISBN 9780134177434 并提交。一旦流程完成，你就可以在“已注册产品”中找到可用的奖励内容。

本书体例

本书中使用的印刷规范如下。

- 黑体字表示新名词、URL、E-mail 地址、文件名和文件扩展名。
- 等宽字体用于程序清单和段落中引用的程序元素，例如变量或函数名、数据结构、数据类型、环境变量、语句和关键字。
- 等宽黑体字表示注释或者应该由用户输入的其他文本。
- 加框的内容表示一个小窍门、建议或者一般注意事项。

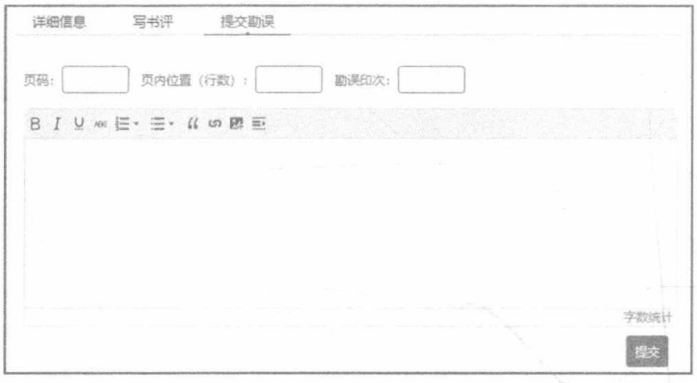
资源与支持

本书由异步社区出品，社区（https://www.epubit.com/）为您提供相关资源和后续服务。

提交勘误

作者和编辑尽最大努力来确保书中内容的准确性，但难免会存在疏漏。欢迎您将发现的问题反馈给我们，帮助我们提升图书的质量。

如果您发现错误，请登录异步社区，搜索到本书页面，单击“提交勘误”，输入相关信息，单击“提交”按钮即可。本书的作者和编辑会对您提交的勘误进行审核，确认并接受后，赠送给您异步社区的 100 积分（积分可用于在异步社区兑换优惠券，或者用于兑换样书或奖品）。



扫码关注本书

扫描下方二维码，您将会在异步社区微信服务号中看到本书信息及相关的服务提示。

与我们联系

我们的联系邮箱是 contact@epubit.com.cn。

如果您对本书有任何疑问或建议，请您发邮件给我们，并请在邮件标题中注明本书书名，以便我们更高效地做出反馈。

如果您有兴趣出版图书、录制教学视频，或者参与图书翻译、技术审校等工作，可以发邮件给我们，有意出版图书的作者也可以到异步社区在线提交投稿（直接访问 www.epubit.com/ selfpublish/submission 即可）。

如果您是学校、培训机构或企业，想批量购买本书或异步社区出版的其他图书，也可以发邮件给我们。

如果您在网上发现有针对异步社区出品图书的各种形式的盗版行为，包括对图书全部或部分内容的非授权传播，请您将怀疑有侵权行为的链接发邮件给我们。您的这一举动是对作者权利的保护，也是我们持续为您提供有价值的内容的动力之源。

关于异步社区和异步图书

“**异步社区**” 是人民邮电出版社旗下 IT 专业图书社区，致力于出版精品 IT 技术图书和相关学习产品，为作译者提供优质出版服务。异步社区创办于 2015 年 8 月，提供大量精品 IT 技术图书和电子书，以及高品质技术文章和视频课程。更多详情请访问异步社区官网 https://www.epubit.com。

“**异步图书**” 是由异步社区编辑团队策划出版的精品 IT 专业图书的品牌，依托于人民邮电出版社近 30 年的计算机图书出版积累和专业编辑团队，相关图书在封面上印有异步图书的 LOGO。异步图书的出版领域包括软件开发、大数据、AI、测试、前端、网络技术等。

异步社区

微信服务号

致　谢

本书能够出版要感谢很多人。我的老同事 Zigurd Mednieks 提议我写这样一本书。Pearson Technology Group 出版社的主编 Laura Lewin 给它开了绿灯，Jelen Publishing 的文稿代理人 Carol Jelen 促成了这本书的出版。

Pearson 的制作人员灵巧地将手稿变成了一本书。插图画家 Jenny Huckleberry 修整了本书的图形，开发编辑、文字编辑 Abigail Manheim Bass 和执行编辑 Lori Lyons 修改了手稿。编辑助理 Olivia Basegio 和项目经理 Ellora Sengupta 和我们一起完成了这个过程。在此致以我诚挚的谢意。

非常感谢主编 Laura Lewin，谢谢她耐心地鼓励我坚持下去。

虽然我是本书的唯一作者，但我一度想把技术编辑 Joe Bowbeer、Thomas Kountis 和 Zigurd Mednieks 列为合著者，因为他们给了我太多的帮助。他们就是技术审查团队的完美成员。作为这个领域公认的专家，他们花时间仔细阅读、理解，然后建议我进行大大小小的修改。如果本书还有错误，这些错误都是我造成的，而表达的清楚和准确都要归功于他们。我非常有幸能得到他们的帮助。

最后要感谢我的妻子 Catherine，感谢你容忍我又在沙发上戴着耳机度过了一个周末。我非常爱你，宝贝！

目 录

第 1 章
了解并发

我们建议用延迟 τ 作为可信赖的同步设备各部分功能的绝对时间单位。

约翰·冯·诺依曼

要构建正确的并发 Android 程序，开发人员需要有一个好的并发进程模型，那么这些模型是如何工作的？它们的用途是什么？对于大多数人来说，并发并不是什么大问题。对于任何多细胞动物（甚至是对病毒）来说，并发只是正常的存在。也只有我们这些沉迷于计算机的人，才会再三考虑一边走路一边嚼口香糖的想法。

1.1 并发很难

一边走路一边嚼口香糖在约翰·冯·诺依曼博士的世界里并非易事。1945 年，他在论文《The First Draft Report on the EDVAC》（冯·诺依曼，1945）里描述了最早的电子数字计算机的体系结构。70 多年来，这个体系结构几乎没有多大变化。大致来讲，纵观它们的历史，数字计算机这个巨大的状态球随着时间的推移被一系列精确定义的操作所转换。时间和顺序是机器定义的内在组成部分。

大多数计算机科学一直在讨论将一种机器状态转换成另一种更理想状态的巧妙操作序列。由于现代机器通常有超过 10^{14} 种可能的状态，因此很难对所有状态进行管理。如果转换发生的顺序可以改变，讨论必然会扩大到包括所有可能状态的所有可能组合，这将是完全不可能的。顺序执行是王道。

1.1.1 软件中的并发

当然，计算机语言是为人类编写的。它们的目的是帮助人们高效、正确，甚至以

未来读者能够理解的方式来表达一种算法（转换机器状态的指令序列）。

早期的编程语言本质上是硬件的扩展。甚至在今天，很多编程语言最初设计只是用于控制机器体系结构的映像。它们绝大多数是**过程化**的，由用于修改（**突变**）内存状态的指令列表组成。由于很难对内存的所有状态进行推理，因此随着时间的推移，语言对允许开发人员表达的状态变化日渐严格。查看编程语言设计历史的一种方法是找出一种允许开发人员轻松表达正确算法的系统，而不是表达不正确的算法。

最早的语言是机器语言，即将计算机指令一对一地翻译的代码。机器语言不可取的原因有两个：首先，甚至表达一个非常简单的想法可能都需要几十行代码；其次，很容易表达错误。

随着时间的推移，为了限制和管理程序修改状态的方式，语言缩小了选择范围。例如，大多数语言都从限制程序执行指令之间的任意跳转变为使用现在熟悉的条件、循环和过程调用。模块和 OOP（面向对象的编程）遵循的方式是将程序分解成小的、容易理解的块，然后限制这些块的交互方式。这种模块化、积木式的方式让现代语言更抽象且更富有表现力。有的语言甚至有完善的类型系统——有助于防止错误。然而，修改机器状态的指令列表仍然是必需的。

1. 函数式编程

虽然大多数计算机研究者和开发者关注的都是在基于冯·诺依曼体系结构的硬件上做更复杂的事情（这些硬件体积更大、速度更快），但一个小而执着的团队在追求一种完全不同的想法——**函数式编程**。

纯函数式编程和过程化编程的区别在于它没有可变状态。函数语言不是对机器状态的连续变化进行推理的，而是求解给定参数的函数值。这是一种相当激进的想法，需要思考才能理解它是如何工作的。然而，如果可以从并发角度看，它有一些非常吸引人的方面，尤其是如果没有可变状态，就不存在隐式时间或顺序。如果没有隐式顺序，并发就只是一个无趣的同义反复。

在创建第一个广为大众所接受的过程语言 Fortran 仅一两年后，John McCarthy 于 1958 年引入了第一个函数语言 Lisp。从那时起，Lisp 等函数语言（Scheme、ML、Haskel、Erlang 等）就被视为卓越但不切实际的教育工具或者时髦开发人员的口头禅。由于摩尔定律（摩尔，1965）更容易预测一个芯片上的处理器数（而不是单个处理器的速度），因此人们不再对函数语言不屑一顾［到 1975 年，摩尔修正了最初的想法，让这一概念正式化，他说集成电路（IC）的数量每两年翻一番］。