Broadview®
www.broadview.com.cn

objc.io | ObjC 中国

Core Data

core data

[德] Florian Kugler　Daniel Eggert　著

徐　涛　钱世家　王　巍　译

中国工信出版集团

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

Core Data

core data

[德] Florian Kugler　Daniel Eggert　著

徐　涛　钱世家　王　巍　译

電子工業出版社
Publishing House of Electronics Industry
北京·BEIJING

内容简介

本书向读者介绍使用 Core Data 时需要特别注意的事项，这将帮助读者避开使用 Core Data 这个十分灵活且异常强大的框架时的一些陷阱。我们从一个简单的应用例子开始，逐步将其扩展为包含关系、高级数据类型、并发、同步以及其他很多特性的完整例子，并在这个过程中对所有这些主题进行讲解。在本书后半部分，我们还会超出这个基本应用所需要涉及的范围，将知识点深入扩展到 Core Data 幕后的工作原理上。我们会学习如何获取高性能、不同 Core Data 设置之间的权衡，以及如何对 Core Data 代码进行调试和性能测试。

本书所有的代码都使用 Swift 编写，我们也展示了如何将 Swift 的语言特性融入 Core Data 中，并写出优雅和安全的代码。我们希望读者在阅读本书的时候有一定的 Swift 和 iOS 开发基础，不过相信不论是新人还是富有经验的开发者，都能从本书中找到实用的信息和设计模式。

图书在版编目（CIP）数据

Core Data /（德）佛罗莱恩 · 库格勒（Florian Kugler），（德）丹尼尔 · 埃格特（Daniel Eggert）著；徐涛，钱世家，王巍译.—北京：电子工业出版社，2016.9
ISBN 978-7-121-29459-4
I. ① C…II. ① 佛… ② 丹… ③ 徐… ④ 钱… ⑤ 王…III. ① 移动终端－应用程序－程序设计 IV. ① TN929.53

中国版本图书馆 CIP 数据核字（2016）第 170663 号

策划编辑：张春雨
责任编辑：王　静
印　　刷：三河市双峰印刷装订有限公司
装　　订：三河市双峰印刷装订有限公司
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：787×980　1/16　　印张：15.75　　字数：328 千字
版　　次：2016 年 9 月第 1 版
印　　次：2016 年 9 月第 1 次印刷
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 zlts@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。

译序

在 20 世纪 60 年代，导航式数据库的概念随着磁盘直接存取而发展起来；从 70 年代开始，关系型数据库登上历史舞台，它的概念一直延续至今。我们无法想象现代的计算机程序中离开了数据库会是怎样的景象，数据库技术已经成了这个世界方方面面的基石。

在数据管理和数据库相关的方面，Apple 给出的选择是 Core Data。正如在简介中所提到的那样，Core Data 其实并不是一个传统意义上的数据库，而是一套对象图管理系统。这套系统默认使用 SQLite 作为底层存储，通过由低向高地将相关的管理组件构建为一个栈，来提供缓存和对象管理机制。这让我们对于数据对象的存储和访问都能够高效而有序地进行。从这一点上来说，Core Data 与单纯的数据库相比，实在是强大得多。

但是能力越大，责任也越大。如果使用不当，那么 Core Data 不但不能为你提供良好的数据存储和访问的性能，甚至会连最基本的操作都难以保证。在这种情况下，Core Data 将不再是你开发的助力，反而会成为掣肘。不幸的是，Core Data 本身学习曲线比较陡峭，而涉及的概念又非常多，所以真正想要精通 Core Data 并完全发挥它的效能并不是很容易的事情。

Apple 在 iOS 的很多原生应用中大量使用了 Core Data，比如照片、音乐和 iBooks 等，并且事实证明它们都出色地完成了任务。在国外，也有很多开发者使用 Core Data 作为应用程序的数据层和持久化的选择。相比其他第三方的解决方案，Core Data 不需要引入额外的框架，也相对稳定可靠。但是在国内，现在使用这项技术的开发者较少，大家对 Core Data 的研究也普遍没有国外深入，这导致了提到 Core Data 很多人会不自觉地抗拒和躲避。将 Core Data 的使用方法和最佳实践以更容易理解的方式带给国内开发者，促进大家接触 Core Data 的架构和思想，这正是我们选择翻译本书的目的。

本书的结构和阅读方法在前言中会有所说明，这里就不再赘述了。需要补充的是，本书里提供了大量的例子和相应的代码，它们大多是需要进行权衡的选择，并对应了不同的场景。只有在你充分理解这些例子的含义后，你才可能在实际使用时做出正确的判断。另外，Core Data 的灵活性是一把双刃剑，当你选择了更多的上下文以及协调器时，也意味着你为项目引入了更多的复杂度。尽可能在能够满足需求的前提下，选择最简单的 Core Data 栈设置，是高效、正确使用 Core Data 的关键。

本书原著的两位作者有着多年的 Core Data 使用经验。Florian Kugler[1] 是 objc.io 的联合创始人，曾经为 objc.io 撰写了很多 Core Data 相关的文章，深受读者喜爱。Daniel Eggert[2] 曾供职于 Apple，帮助 Apple 将照片应用迁移到 Core Data 框架内。他们的努力让 Core Data 这个看起来有些“可怕”的框架变得平易近人，借此我们可以一窥 Core Data 的究竟。不过不论是原作者还是译者，其实和各位读者一样，都只不过是普通开发者中的一员，所以本书出现谬漏可能在所难免。如果你在阅读时发现了问题，可以通过出版社联系我们，我们将及时研究并加以改进。

最后，祝你阅读愉快。

徐　涛
钱世家
王　巍

[1] *https://twitter.com/floriankugler*
[2] *https://twitter.com/danielboedewadt?lang=en*

前言

Core Data 是 Apple 为 iOS、OS X、watchOS 和 tvOS 而设计的对象图管理 (object graph management) 和数据持久化框架。如果你的 App 需要存储结构化的数据，那么 Core Data 是一个显而易见的方案：它是现成的，Apple 仍然在积极地维护它，而且它已经存在超过 10 年了。Core Data 是一个成熟、经过实践检验的代码库。

然而 Core Data 最初会让人有一些困惑：它非常灵活，但是 API 的最佳实践却并非显而易见。换句话说，本书的目标是帮助读者快速入门 Core Data。我们希望提供给读者一系列包括从简单到高级的使用场景中的最佳实践，这样你可以充分利用 Core Data 的能力而又不会迷失在一些不必要的复杂性中。

比如，Core Data 经常被诟病难以在多线程环境中使用。其实 Core Data 的并发模型非常明确和一致。如果正确使用，那么它可以帮助你避免许多并发编程中一些固有的陷阱。其他的复杂性并不是由 Core Data 引入的，它们的根源其实是并发本身。我们会在第 9 章中对其进行深入研究，另外我们还会实际演示一个后台同步方案的例子。

除此之外，Core Data 也经常被吐槽性能糟糕。如果你像使用关系型数据库那样来使用 Core Data，那么你会发现与直接使用类似 SQLite 这样的数据库相比，Core Data 的性能开销会很高。但如果把 Core Data 当成一个对象图管理系统来正确使用，那么得益于内建的缓存和对象管理机制，它在很多方面实际上反而更快。此外，抽象级别更高的 API 可以让你专注于优化 App 里关键部分的性能，而不是从头开始来实现如何持久化。在本书中，我们会介绍保持 Core Data 高性能的最佳实践，并在专门讲性能以及性能分析的章节中探讨如何解决 Core Data 的性能问题。

本书使用 Core Data 的方式

本书展示了如何在实际例子中使用 Core Data，而不仅仅是简单地对 API 手册进行一些扩展。我们有意专注于完整例子的最佳实践。根据我们的经验，正确地组合使用 Core Data 的各个部分往往是最大的挑战。

此外，本书还深入解释了 Core Data 内部的运作原理。了解 Core Data 这个灵活框架可以帮助你做出正确的决定，同时能让你的代码保持简单易懂。特别是当遇到并发和性能问题时，这一点尤为重要。

示例代码

你可以在 GitHub 上[1]找到一个完整的示例程序的源代码。我们在本书中很多地方都将用这个示例程序来演示 Core Data 在较大的项目中面临的挑战和相应的解决方案。

请注意该示例程序代码有时会和本书前面的一些章节中的示例程序有所不同。因为示例项目是最终形态的完整的代码，而本书前面章节中描述的是该示例程序早期、简单阶段的代码。

结构

在本书的第一部分，我们会创建一个简单版本的应用程序，来演示如何使用 Core Data 以及 Core Data 的基本工作原理。即使早期的示例对读者来说可能相当容易，但我们仍然建议读者浏览本书的这些部分，因为后面更复杂的例子是建立在前面介绍的最佳实践和技术基础之上的。我们还想告诉你的是，即便在简单的应用场景中，Core Data 也会非常有用。

第二部分则着重深入介绍 Core Data 各个部分是如何一起协作的。我们会仔细探讨当以不同方式访问数据时会发生什么，我们也会对插入或者操作数据时发生的情况进行研究。这部分所覆盖的内容会比写一个简单的 Core Data 应用程序所必要得多，这些方面的知识在处理更大或更复杂的情况时可以派上用场。在此基础上，我们将以性能方面的考量来对这个部分进行总结。

第三部分从描述一个用来保持本地数据与网络服务一致的通用同步架构开始，然后我们会深入探讨如何在 Core Data 中同时使用多个托管对象上下文 (managed object context)。我们提出设置 Core Data 栈的不同方案，并讨论了它们的优缺点。在第 9 章里，介绍了如何应对同时使用多个上下文带来的额外复杂性。

第四部分涉及一些高级的主题，比如高级的谓词 (predicate)、搜索和文本排序、如何在不同的数据模型版本之间迁移数据，以及分析 Core Data 栈的性能时所需要的工具和技术等。这部分中有一章是从 Core Data 视角介绍有关关系数据库和 SQL 查询语言的基本知识的。如果你不熟悉这些内容，那么这些章节能对你有所帮助，特别是可以让你理解 Core Data 潜在的性能问题，以及解决这些问题所需要的分析技术。

[1] *https://github.com/objcio/core-data*

关于 Swift 的一些说明

贯穿本书，我们所有的示例都使用 Swift 编写。我们拥抱 Swift 的语言特性——比如泛型、协议以及扩展——它们能让我们更优雅、简单、安全地使用 Core Data 的 API。

用 Swift 表示的最佳实践和设计模式同样也适用于 Objective-C 的代码。在实现上，由于语言上的不同，或许在某些方面会稍有不同，但是底层的原则是相通的。

可选值的约定

Swift 提供了 `Optional` 数据类型，这迫使我们显式地思考和处理没有值的情况。我们非常喜欢这个功能，所以我们在所有的例子里都使用了它。

因此我们尽量避免使用 Swift 的 ! 操作符来强制解包 (包括用它来定义隐式解包类型的用法)，在我们看来这是一种坏代码的味道，因为它破坏了我们使用可选值类型所带来的类型安全。

唯一的例外是那些必须设置但又无法在初始化时设置的属性。比如 Interface Builder 的 outlets 或必要的代理 (delegate) 属性等。在这些情况下，使用隐式解包的可选值符合“尽早崩溃”原则：我们会立刻知晓这些必须要设置而又没有正确设置的属性。

错误处理的约定

Core Data 中许多方法会抛出错误。基于它们是不同类型的错误这一基本事实，我们可以分类处理这些错误。我们将区分逻辑错误和其他错误。

逻辑错误是指程序员犯错的结果。它们应该从代码层面上修复而不应该尝试动态恢复程序的运行。

举一个例子，当你尝试读取应用程序包里的一个文件时，因为应用程序包是只读的，那么一个文件要么存在，要么不存在，而且它的内容永远不会变。所以如果我们无法打开或者解析应用程序包里的文件，那么这就是一个逻辑错误。

对于这些类型的错误，我们使用 Swift 的 `try!` 或 `fatalError()` 来尽可能早地让应用程序崩溃。

同样的思想可以适用于 `as!` 操作符的强制类型转换: 如果我们知道一个对象必须是某种类型，转换失败的唯一原因会是逻辑错误，那么在这种时候我们实际上是希望应用程序崩溃的。

很多时候我们用 Swift 的 guard 关键字来更好地表达哪些地方出错了。例如 fetched results controller 返回的类型是 NSManagedObject 的对象，我们知道它必须是一个特定的子类，我们使用 guard 来保证向下转换，并在出错的时候使用 fatal error 来中止程序:

```
func objectAtIndexPath(indexPath: NSIndexPath) -> Object {
    guard let result = fetchedResultsController.objectAtIndexPath(indexPath)
        as? Object else
    {
        fatalError("Unexpected object at \(indexPath)")
    }
    return result
}
```

对于可恢复的非逻辑性错误，我们使用 Swift 的错误传递方法：抛出 (throw) 或者重新抛出 (rethrow) 这些错误。

Florian
Daniel

目录

I Core Data 基础 1

第 1 章 初探 Core Data 2

1.1 Core Data 架构 2

1.2 数据建模 4

实体和属性 5

托管对象子类 6

1.3 设置 Core Data 栈 7

1.4 显示数据 9

获取请求 11

Fetched Results Controller 13

1.5 操作数据 19

插入对象 19

删除对象 22

1.6 总结 26

重点 26

第 2 章 关系 27

2.1 添加 Country 和 Continent 实体 27

子实体 31

2.2 创建关系 33

其他类型的关系 35
建立关系 36
关系和删除 41
2.3 适配用户界面 43
2.4 总结 48
重点 48
第 3 章 数据类型 49
3.1 标准数据类型 49
数值类型 49
日期 50
二进制数据 50
字符串 51
3.2 原始属性和临时属性 51
原始属性 51
临时属性 52
3.3 自定义数据类型 52
自定义值转换器 52
自定义存取方法 56
3.4 默认值和可选值 59
3.5 总结 60
重点 60
II 理解 Core Data 61
第 4 章 访问数据 62
4.1 获取请求 62
对象情值 64
获取请求的结果类型 67

批量获取 69
异步获取请求 70
4.2 关系 70
4.3 其他取回托管对象的方法 71
4.4 内存考量 72
托管对象及其上下文 72
关系的循环引用 73
4.5 总结 74
重点 74
第 5 章 更改和保存数据 76
5.1 变更追踪 76
5.2 保存更改 78
验证 80
保存冲突 82
5.3 批量更新 82
5.4 总结 84
重点 84
第 6 章 性能 86
6.1 Core Data 栈的性能特质 86
详解性能 87
6.2 避免获取请求 89
关系 89
搜索特定的对象 91
类似单例的对象 93
小数据集 96
6.3 优化获取请求 96
对象排序 96

避免多个、连续的惰值 97
批量获取 98
Fetched Results Controller 99
关系预加载 99
索引 100
6.4 插入和修改对象 102
6.5 如何构建高效的数据模型 103
6.6 字符串和文本 106
6.7 独家秘诀的可调参数 106
6.8 总结 107
III 并行和同步 109
第 7 章 与网络服务同步 110
7.1 组织和设置 110
项目结构 111
7.2 同步架构 112
7.3 上下文属主 113
线程、队列和上下文 113
7.4 响应本地更改 115
7.5 响应远程更改 119
7.6 更改处理器 119
上传 Moods 120
7.7 删除本地对象 123
7.8 分组和保存更改 123
7.9 扩展同步架构 125
跟踪每个属性的更改 125
链接更改处理器 125

自定义网络代码 126

第 8 章 使用多个上下文 128

8.1 Core Data 和并发 128

在不同的上下文之间传递对象 130

合并更改 132

8.2 Core Data 栈 134

两个上下文，一个协调器 134

两个协调器 136

嵌套上下文的设置 137

8.3 总结 144

重点 145

第 9 章 使用多个上下文的问题 146

9.1 保存冲突 146

预定义的合并策略 147

自定义合并策略 148

9.2 删除对象 153

两步删除法 154

传播删除 156

9.3 唯一性约束 157

9.4 总结 159

IV 进阶话题 161

第 10 章 谓词 162

10.1 一个简单的例子 162

使用谓词 163

10.2 用代码来创建谓词 164

10.3 格式字符串 165

比较 166
可选类型值 167
日期 168
10.4 合并多个谓词 168
常量谓词 170
10.5 遍历关系 171
子查询 171
10.6 匹配对象和对象 ID 172
10.7 匹配字符串 173
字符串和索引 175
10.8 可转换的值 175
10.9 性能和排序表达式 176
10.10 总结 177
第 11 章 文本 178
11.1 一些例子 178
11.2 搜索 179
字符串标准化 180
高效搜索 182
11.3 排序 183
一种简单的方法 183
更新一个已排序的数组 184
持久化一个已排序的数组 188
11.4 总结 189
重点 189
第 12 章 数据模型版本以及迁移数据 190
12.1 数据模型版本 190
12.2 数据迁移的过程 192

自动数据迁移 193
手动数据迁移 194
12.3 推断的映射模型 201
12.4 自定义映射模型 202
自定义实体映射策略 204
12.5 数据迁移和用户界面 206
12.6 测试数据迁移 209
调试数据迁移时的输出 210
12.7 总结 210
重点 211
第 13 章 性能分析 212
13.1 SQL 调试输出 212
获取请求 213
填充惰值 217
保存数据 218
13.2 Core Data Instruments 219
13.3 线程保护 222
13.4 总结 222
第 14 章 关系型数据库基础和 SQL 223
14.1 一个嵌入式数据库 223
14.2 数据表、列以及行 224
14.3 数据库系统的结构 225
查询处理器 225
存储管理器 226
事务管理器 226
数据和元数据 226
14.4 数据库语言 SQL 227

排序 228
14.5 关系 229
一对一关系 229
一对多关系 230
多对多关系 230
14.6 事务 231
14.7 索引 232
14.8 日志 232
14.9 总结 233