HZ BOOKS
华章科技

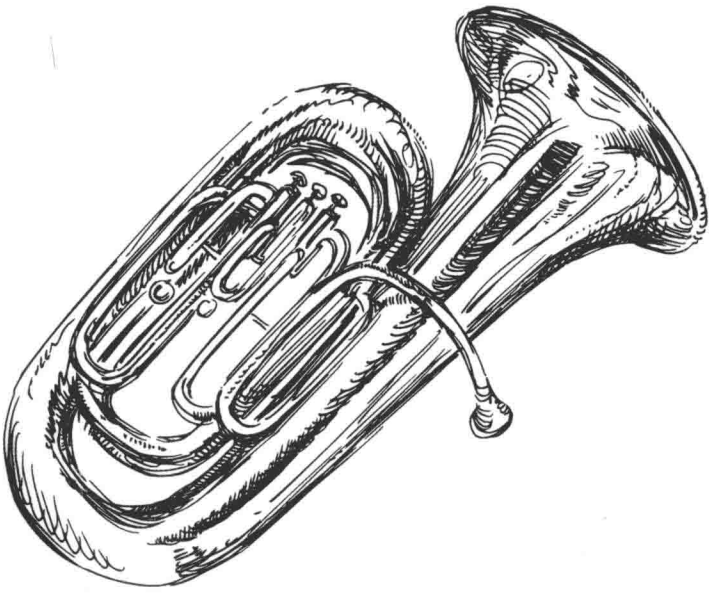
EFFECTIVE
系列丛书

从基本原则、惯用法、语法、内存管理、设计、实现、设计模式、兼容性和性能优化等方面深入探讨编写高质量Objective-C代码的技巧、禁忌和最佳实践

Writing Solid Objective-C Code
61 Suggestions to Improve Your Objective-C Program

编写高质量代码
改善Objective-C程序的61个建议

刘一道 著

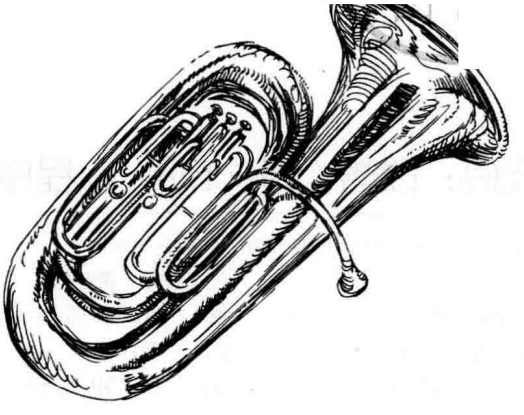


机械工业出版社
China Machine Press

Writing Solid Objective-C Code
61 Suggestions to Improve Your Objective-C Program

编写高质量代码
改善Objective-C程序的61个建议

刘一道 著



机械工业出版社

图书在版编目（CIP）数据

编写高质量代码：改善 Objective-C 程序的 61 个建议 / 刘一道著 . —北京：机械工业出版社，2015.8
（Effective 系列丛书）

ISBN 978-7-111-51463-3

I. 编… II. 刘… III. C 语言 – 程序设计 IV. TP312

中国版本图书馆 CIP 数据核字（2015）第 213529 号

编写高质量代码：改善 Objective-C 程序的 61 个建议

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码：100037）
责任编辑：孙海亮 责任校对：董纪丽
印 刷：北京市荣盛彩色印刷有限公司 版 次：2015 年 9 月第 1 版第 1 次印刷
开 本：186mm × 240mm 1/16 印 张：14
书 号：ISBN 978-7-111-51463-3 定 价：59.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：(010) 88379426 88361066 投稿热线：(010) 88379604
购书热线：(010) 68326294 88379649 68995259 读者信箱：hzit@hzbook.com

Preface 前 言

如何写出高质量的代码?

我一直在思考，如何才能编写出高质量、优秀的代码，我也在不停地探寻，希望找出类似于武侠小说中所说的武功秘籍，在编写代码一途可以帮助大家走“捷径”从而达到事半功倍的效果。

《道德经》第四十八章中说“为学者日益，为道者日损。损之有损，以至于无为。无为而不为”。这句话是说，治学上，不要过于追求外在的经验知识，否则经验知识越积累增多，就会越僵化臃肿。要学会透过直观体悟把握事物未分化时的状态或者内索自身虚静，洞悉其内在的道化真谛，从而简之再简。这些也就是我们现在说的“大道至简”。

治学如此，写代码更是如此。在程序员写代码的职业生涯中，前 5 年，他看到的只是一行一行的代码，他会为自己洋洋洒洒写成的代码而陶醉；5 年之后，就不是单纯地写代码了，而是在做一件艺术品，此时的程序员就像雕刻家一样，在刻下每一刀之前，都需纵观全局，细细揣摩，落刀如有神，一气呵成。故此，写出优秀的高质量代码，需要像唐僧西天取经一样，踏踏实实，用平常心闯过一关又一关，如此，写出高质量的代码自然就是水到渠成的事了。写代码时切忌心态浮躁，急功近利。

本书适合哪些读者

本书不是一本介绍“ Objective-C”代码如何编写的入门级的书籍。故此，如果你只想初步了解一下“Objective-C”开发，而不想做深入研究的话，那么本书就不适合你了。

本书主要面向专业从事 Objective-C 开发或者想转向“ Objective-C”开发的研究人员，帮助其编写便于维护、执行迅速且不易出错的代码。如果你是“ Objective-C”开发技术大

咖，翻阅本书，对你来说可能会有些浪费时间，故此也请你一瞥而过！

本书主要适合如下读者：

- 对软件开发，特别是对 Objective-C 开发有兴趣的人。
- 想成为一名专职的软件开发人员的人。
- 想进一步提高自己“Objective-C”技术水平的在校学生。
- 开设相关专业课程的大专院校的师生。

你该如何阅读本书

本书共 9 章，从内容上可以分 3 部分[⊖]。大家可以根据自身状况，选择性跳读，翻阅自己最感兴趣或与当前工作相关的章节来读。下面把这几章简单归类评述，为你进行选择性跳读时提供参考：

第一部分（第 1 ~ 5 章）：主要围绕如何写出高质量代码提出一些建议。这些建议一部分来源于苹果每年举行的一些开发大会，主要是针对 Objective-C 语法、性能与功能改进进行的阐述和解惑；另一部分来自于一线开发者平时工作中的点点滴滴的感受。作为笔者，我不过是把这些点滴像串珍珠一样把它们串起来，形成一个实用的、具有整体性的建议。

第 1 章 作为本书的首章，我感觉唠唠“Objective-C 的那些事儿”很有必要。希望通过这一章的唠叨能让你有所收获。

第 2 章 主讲数据类型、集合和控制语句。《道德经》第五十二章中有言“天下有始，以为天下母。既得其母，以知其子；既知其子，复守其母。”掌握了基本的开发语言，就可以“知其子”——程序；反过来，通过开发程序，又可以“复守其母”——开发语言，即通过开发程序可以进一步巩固、加深对开发语言的理解。在 Objective-C 中，构建语言的基本单元——值和集合，更是重中之重。本章将围绕这些构建开发语言的“基石”从不同角度加以阐述。

第 3 章 主讲内存管理。内存管理曾经是程序员的噩梦，特别是在面向过程开发的过程中。虽然，在纯面向对象的开发语言中，例如 C# 和 Java 等开发语言，有了自动内存垃圾回收的机制，但依赖这种自动内存垃圾回收机制的代价往往很高，容易造成程序性能的耗损，进而容易产生难以突破的“性能劲瓶”。因此，在编写高性能的应用程序，特别是服务器程序时，不得不依赖 C、Pascal 和 C++ 等非纯面向对象的语言来完成。现在，众多的 Android 手机或者平板上遇到的 App 莫名其妙崩溃的现象，在很大程度上与 Android 底层的内存回收处理不当有很大的关系。

⊖ 这只是逻辑上的划分，为了不影响读者选择性跳读，正文中未明确体现。

在本书定稿之时，虽然，Objective-C 已经具有了“垃圾自动回收”的机制，但洞悉其内存的管理机制，写出高质量的代码，还是至关重要的。

第 4 章 主讲设计与声明。《庄子 · 齐物论》中曰：“昔者庄周梦为蝴蝶，……不知周之梦为蝴蝶与？蝴蝶之梦为周与？”意思是说，庄子梦见自己变成蝴蝶，醒来后发现自己是庄周。庄子分不清自己是梦中变成蝴蝶，还是蝴蝶梦中变成庄子了。在 Objective-C 中，类与对象的关系，就有些类似庄子和蝴蝶的关系，从一定的角度来看，类就是对象，而从另外一个角度来看，对象又是类。本章就以庄周梦蝶做引子，来谈谈设计和声明。

第 5 章 众所周知，在面向对象的设计中，对于一个类，完成其定义，要包含类名、实例变量及方法的定义和声明，但这最多也只能算是完成 20% 的工作，其余 80% 的工作主要在于其实现。如何把类的实现写好，很考验一个人的功底。尽可能利用开发语言本身一些特性，是实现类的一个比较有效的途径。本章就结合 Objective-C 的特性，来介绍在类的实现中一些可利用的做法。

第二部分（第 6 ~ 8 章），相对于第一部分，本部分更多关注一些“习以为常”的理念，结合 Objective-C 语言，进行一番新的解读。这几章里面，很多东西你也许早已经知道，但一些理念如何融入自己的代码和设计中，却是更高的要求。正如巴菲特的理财理念，很多人都能知道，且能背诵得滚瓜烂熟，而在实际操作上，结果却大相径庭，其主要原因是，很多“理念”自己不过是记住罢了，并未已融入自己的血液和骨髓里，在实际操作上，仍旧沉溺于“旧习”。

第 6 章 主讲继承与面向对象的设计。随着这几年面向对象编程（OOP）越来越盛行，在编程领域，现在几乎是面向对象编程语言的天下了。继承，作为面向对象编程的三大特征（继承、多态和封装）之一，起着核心的作用。但不同面向对象的编程语言实现的方式，有着比较大的差异性。本章从纵向和横向两个维度，来阐述“继承”在 Objective-C 开发语言中的多面性，使你娴熟掌握“继承”在 Objective-C 开发语言中的“有所为而有所不为”。

第 7 章 主讲设计模式与 Cocoa 编程。设计模式是一种设计模板，用于解决在特定环境中反复出现的一般性问题。它是一种抽象工具，在架构、工程和软件开发领域相当有用。本章将介绍什么是设计模式，解释为什么设计模式在面向对象的设计中非常重要，并讨论一个设计模式的实例。

第 8 章 主讲定制 init... 和 dealloc。在 Objective-C 中，没有 new 和 delete 这两个关键字，但存在着 alloc 和 init...，它们承担着与 new 类似的功能，前者用于处理内存的分配，后者用于在分配的内存中进行数据的初始化；与 delete 类似的是 dealloc。掌握 init...，对于写出高性能的应用是至关重要的。

第三部分（第 9 章），在本书将近定稿时，恰遇到 Swift 的新版本推出，为了解决开发者在混用 Objective-C 和 Swift 过程中遇到的一些疑难点，故此添加了本部分。如果对

Swift“不感冒”，那么本部分你就没有阅读的必要了。

第 9 章 主讲 Objective-C 和 Swift 的兼容性。Swift 是苹果公司在 WWDC 2014 新发布的一门编程语言，用来撰写 OS X 和 iOS 应用程序，但是 Swift 的推出，使很多原先很熟练使用 Objective-C 的码农产生了一定程度的“惊惶”：担忧苹果公司放弃对 Objective-C 的支持，不得不花费一定的时间和精力来学习一门新的开发语言——Swift。Xcode 6 或者更高的版本支持二者的相互兼容。也就是说，在 Objective-C 文件中可以使用 Swift 编写的代码，也可以在 Swift 文件中使用 Objective-C 编写的代码，但其相互兼容性是有条件的。本章就来讲解二者的兼容性问题。

本书没有采取循序渐进的方式。故此，读者可以根据“兴趣”选择自己喜欢的章节来阅读，这是最有效的读书方式，也是笔者推崇的读书方式。

勘误和支持

除封面署名外，参加本书编写工作的还有孙振江、陈连增、边伟、郭合苍、郑军、吴景峰、杨珍民、王文朝、崔少闯、韦闪雷、刘红娇、王洁、于雪龙、孔琴。由于笔者的水平有限，编写时间仓促，书中难免会出现一些错误或者不准确的地方，恳请读者批评指正。

书中的不足之处，还望各位读者多提意见，欢迎发送邮件至邮箱 yifeilang@aliyun.com，期待能够得到你们的真挚反馈。

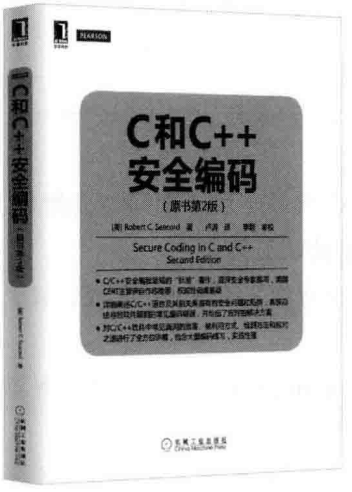
感恩致谢

本书之所以能出版，多亏华章公司的编辑孙海亮先生多次催促，不断地给予我鼓励。期间华章群一些网友的鼓励，也使我受益匪浅，在此向他们表示感谢。最后，要感谢的就是我亲爱的读者，感谢你拿起这本书，你的认可，就是我的最大的快乐。

刘一道

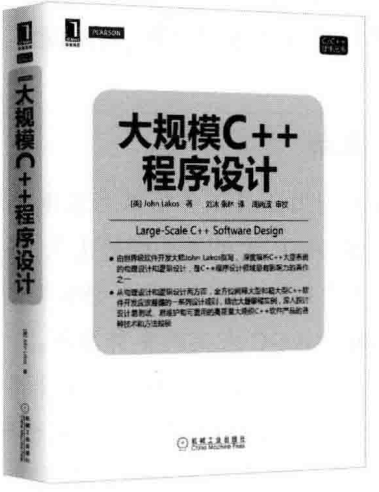
于北京

推荐阅读



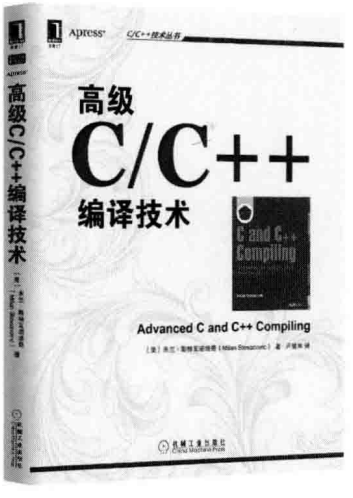
C和C++安全编码（原书第2版）

作者：Robert C. Seacord ISBN：978-7-111-44279-0 定价：79.00元



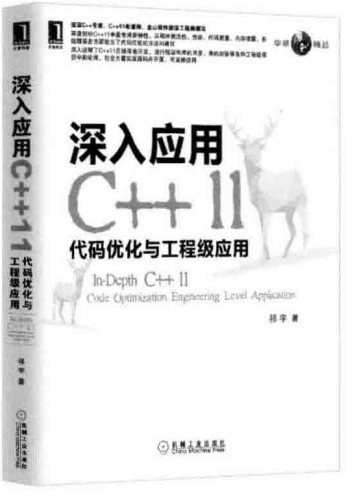
大规模C++程序设计

作者：John Lakos ISBN：978-7-111-47425-8 定价：129.00元



高级C/C++编译技术

作者：米兰·斯特瓦诺维奇 ISBN：978-7-111-49618-2 定价：69.00元



深入应用C++11：代码优化与工程级应用

作者：祁宇 ISBN：978-7-111-50069-8 定价：79.00元

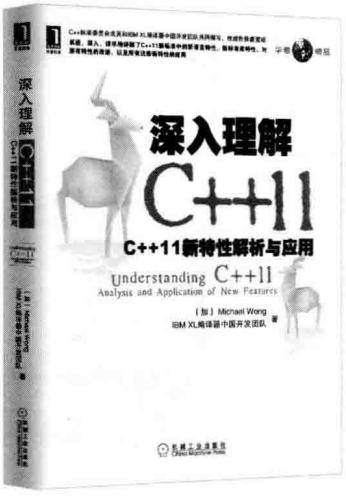
推荐阅读

深入实践Boost：Boost程序库开发的94个秘笈

作者：Antony Polukhin ISBN：978-7-111-46242-2 定价：59.00元

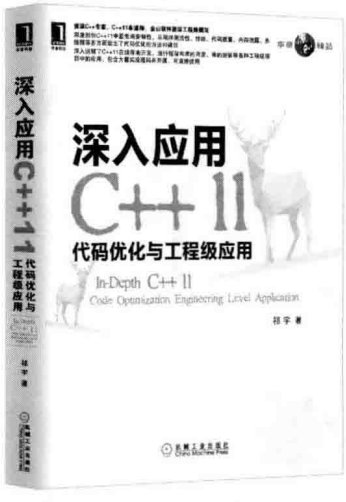
大规模C++程序设计

作者：John Lakos ISBN：978-7-111-47425-8 定价：129.00元



深入理解C++11：C++11新特性解析与应用

作者：Michael Wong IBM XL编译器中国开发团队 ISBN：978-7-111-42660-8 定价：69.00元



深入应用C++11：代码优化与工程级应用

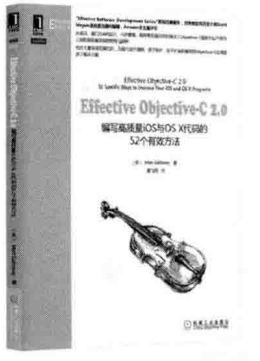
作者：祁宇 ISBN：978-7-111-50069-8 定价：79.00元

推荐阅读

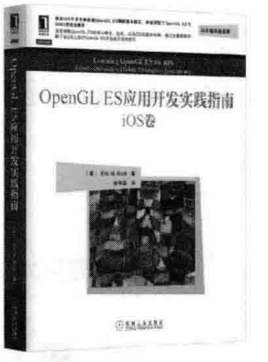

Swift iOS
应用开发实战
Swift iOS in Action

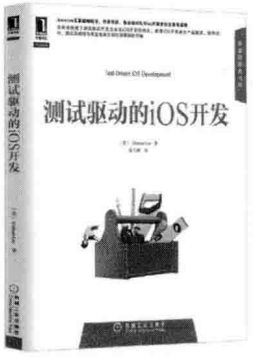

iOS 7
iPhone/iPad
应用开发技术详解
iOS 7
iPhone & iPad Application Development

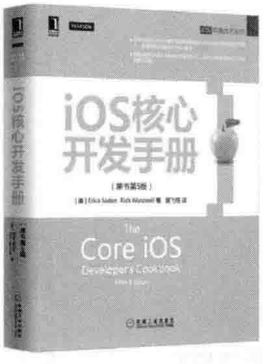

iOS
应用逆向工程
iOS App Reverse Engineering
Second Edition

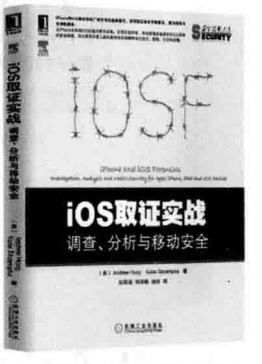

Effective Objective-C 2.0
Effective Objective-C 2.0
编写高质量iOS与OS X代码的
52个有效方法


Objective-C 2.0 Mac
和iOS开发实践指南


OpenGL ES应用开发实践指南
iOS卷


测试驱动的iOS开发
Test-Driven iOS Development


iOS核心
开发手册
The
Core iOS
Developer's Cookbook


iOS取证实战
调查、分析与移动安全

Contents 目 录

第 1 章 Chapter 1

让自己习惯 Objective-C

听说有本书叫《明朝那些事儿》，很多人喜欢，但我没有看过。我这人有些“老帽儿”，对一些时髦的东西不是很感兴趣。我看的都是那些经过千百年“浪沙冲洗”沉淀下来的书籍（技术书籍除外）。作为本书的首章，我感觉聊 一些“ Objective-C 的那些事儿”很有必要，希望读者能有所收获。

建议 1：视 Objective-C 为一门动态语言

Objective-C，是美国人布莱德·确斯（Brad Cox，见图 1-1）于 1980 年年初发明的一种程序设计语言，其与同时代的 C++ 一样，都是在 C 的基础上加入面向对象特性扩充而成的。C++ 如日中天，红火已有 30 年之久，而 Objective-C 到 2010 年才被人注意，并逐渐红火起来。造成这种结果的原因跟其版权为苹果独自拥有和苹果自我封闭性、不作为性有很大的关系。这也是 Objective-C 在语法上跟其他语言（如 C++、Pascal、Java 和 C#）等相比有些不足的原因。可喜的是，苹果公司于 2010 年起，在每年的苹果年度发布大会上都会针对 Objective-C 语言的语法发布新的改良版本。

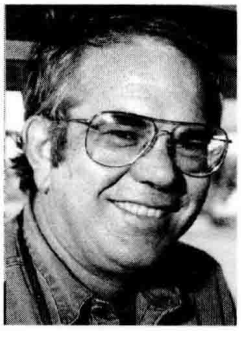


图 1-1 布莱德·确斯

虽然 Objective-C 和 C++ 都是在 C 的基础上加入面向对象特性扩充而成的程序设计语言，但二者实现的机制差异很大。Objective-C 基于动态运行时类型，而 C++ 基于静态类型。也就是说，用 Objective-C 编写的程序不能直接编译成可令机器读懂

的机器语言（二进制编码），而是在程序运行时，通过运行时（Runtime）把程序转译成可令机器读懂的机器语言；用 C++ 编写的程序，编译时就直接编译成了可令机器读懂的机器语言。这也就是为什么把 Objective-C 视为一门动态开发语言的原因。

从原理上来讲，目前常用的 Java 和 C# 等程序开发语言都是动态开发语言，只不过 Java 用的是虚拟机 JVM（Java Virtual Machine），如图 1-2 所示，C# 用的是公共语言运行时 CLR（Common Language Runtime）。与此相对，C++ 可视为静态开发语言。

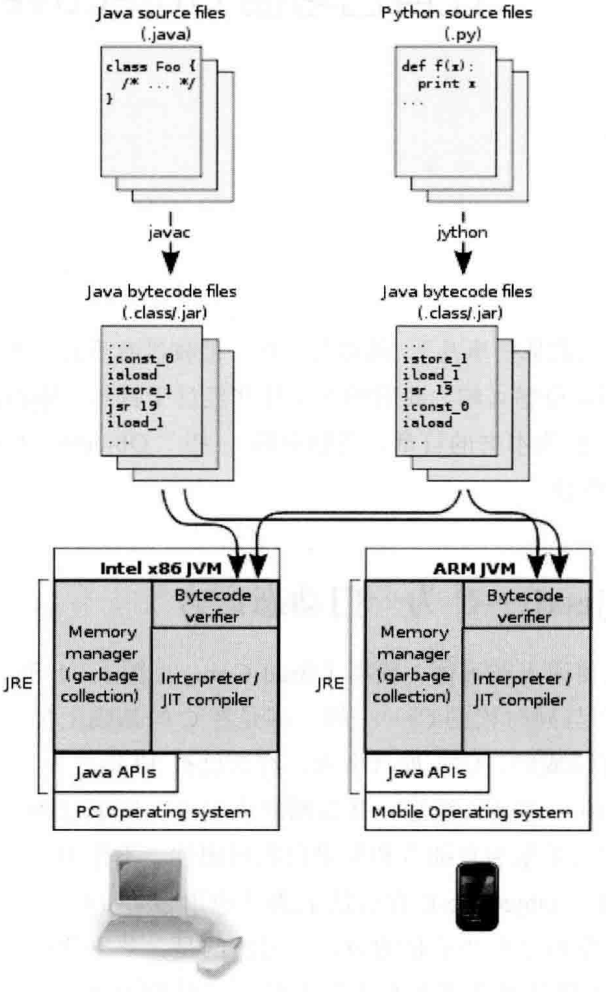


图 1-2　Java 虚拟机实现机制

Objective-C 作为一门动态开发语言，会尽可能地将编译和链接时要做的事情推迟到运行时。只要有可能，Objective-C 总是使用动态的方式来解决问题。这意味着 Objective-C 语言不仅需要一个编译环境，同时也需要一个运行时系统来执行编译好的代码。

而运行时（Runtime）正是扮演着这样的角色。在一定程度上，运行时系统类似于Objective-C 语言的操作系统，Objective-C 就基于该系统来工作。因此，运行时系统好比Objective-C 的灵魂，很多东西都是在这个基础上出现的。很多开发人员常常对运行时系统充满疑惑，而恰恰这是一个非常重要的概念。可以这么问："如果让你实现（设计）一个计算机语言，你会如何下手？"很少程序员这么思考过。但是这么一问，就会强迫你从更高层次来思考以前的问题了。

要想理解"Objective-C 是一门动态开发语言"，就不得不理解开发语言的三个不同层次。

（1）传统的面向过程的语言开发。例如 C 语言，实现 C 语言编译器很简单，只要按照语法规则实现一个 LALR 语法分析器就可以了。编译器优化是非常难的，不在本节讨论的范围内。这里实现了编译器中最基础和原始的目标之一，就是把一份代码里的函数名称转化成一个相对内存地址，把调用这个函数的语句转换成一个跳转指令。在程序开始运行时，调用语句可以正确跳转到对应的函数地址。这样很好，也很直白，但是太死板了。

（2）改进的开发面向对象的语言。相对面向过程的语言来说，面向对象的语言更加灵活了。例如 C++，C++ 在 C 的基础上增加了类的部分。但这到底意味着什么呢？再写它的编译器要如何考虑呢？其实，就是让编译器多绕个弯，在严格的 C 编译器上增加一层类处理的机制，把一个函数限制在它处在的类环境里，每次请求一个函数调用，先找到它的对象，其类型、返回值、参数等确定后再跳转到需要的函数。这样很多程序增加了灵活性，同样一个函数调用会根据请求参数和类的环境返回完全不同的结果。增加类机制后，就模拟了现实世界的抽象模式，不同的对象有不同的属性和方法。同样的方法，不同的类有不同的行为！这里大家就可以看到作为一个编译器开发者都做了哪些进一步的思考。虽然面相对象的语言有所改进，但还是死板，故此仍称 C++ 是静态语言。

（3）动态开发语言。希望更加灵活，于是完全把上面那个类的实现部分抽象出来，做成一套完整运行阶段的检测环境，形成动态开发语言。这次再写编译器甚至保留部分代码里的 sytax 名称，名称错误检测，运行时系统环境注册所有全局的类、函数、变量等信息，可以无限地为这个层增加必要的功能。调用函数时，会先从这个运行时系统环境里检测所有可能的参数再做 jmp 跳转。这就是运行时系统。编译器开发起来比上面更加绕弯，但是这个层极大地增加了程序的灵活性。例如，当调用一个函数时，前两种语言很有可能一个跳到了一个非法地址导致程序崩溃，但是在这个层次里面，运行时系统过滤掉了这些可能性。 这就是动态开发语言更加强壮的原因，因为编译器和运行时系统环境开发人员已经帮你处理了这些问题。

好了，上面说了这么多，再返回来看 Objective-C 的这些语句：

```
id obj=self;
if ([obj respondsToSelector:@selector(function1:)) {
```

```
}
if ([obj isKindOfClass:[NSArray class]] ) {
}
if ([obj conformsToProtocol:@protocol(myProtocol)]) {
}
if ([[obj class] isSubclassOfClass:[NSArray class]]) {
}
[obj someNonExistFunction];
```

看似很简单的语句，但是为了让语言实现这个能力，语言开发者要付出很多努力来实现运行时系统环境。这里运行时系统环境处理了弱类型及函数存在检查工作。运行时系统会检测注册列表里是否存在对应的函数，类型是否正确，最后确定正确的函数地址，再进行保存寄存器状态、压栈、函数调用等实际的操作。

```
id knife=[Knife grateKnife];
NSArray *monsterList=[NSArray array];
[monsterList makeObjectsPerformSelector:@selector(killMonster:)
withObject:knife];
```

用 C 和 C++ 完成这个功能还是比较麻烦的，但是动态语言处理却非常简单，并且这些语句让 Objective-C 语言更加直观。

在 Objective-C 中，针对对象的函数调用将不再是普通的函数调用：

```
[obj function1With:var1];
```

这样的函数调用将被运行时系统环境转换成：

```
objc_msgSend(target,@selector(function1With:),var1);
```

Objective-C 的运行时系统环境是开源的，这里可以进行简单介绍，可以看到 objc_msgSend 由汇编语言实现，甚至不必阅读代码，只需查看注释就可以了解。运行时系统环境在函数调用前做了比较全面的安全检查，已确保动态语言函数调用不会导致程序崩溃。对于希望深入学习的朋友可以自行下载 Objective-C 的运行时系统源代码来阅读，进行更深入的了解。

```
/********************************************************************
* id        objc_msgSend(id self,
*           SEL op,
*           ...)
*
* a1 是消息接收者，a2 是选择器
********************************************************************/
     ENTRY objc_msgSend
# 检查接收者是否空
    teq     a1, #0
    moveq   a2, #0
    bxeq    lr

# 保存寄存器（对象），通过在缓存里查找，来加载接收者（对象）类
```