Computer Science and Application 计算机科学与应用, 2017, 7(3), 192-198
Published Online March 2017 in Hans. http://www.hanspub.org/journal/csa
https://doi.org/10.12677/csa.2017.73024

Hans汉斯

Dynamic Loading Technique Runtime System Based on Objective-C

Tian Xia[1], Zhengzhi Li[2], Pengfei Xue[3]

College of Computer Science, Sichuan University, Chengdu Sichuan
Email: scuxiatian@foxmail.com

Received: Mar. 3rd, 2017; accepted: Mar. 20th, 2017; published: Mar. 23rd, 2017

Abstract

From the beginning of the 21st century, with the rapid development of mobile Internet, the number of global Internet users increased rapidly. In order to meet the needs of different users in different scenarios, mobile applications continue to emerge; while during its continuous improvement, people also have a dependence on mobile applications. So, in order to adapt to the ever-changing market demand and the use of scenarios, various service providers have to quickly iterate products. Conversely, because the product is too fast iterated, users need to constantly update and download applications, resulting in poor user experience. In this paper, we put forward a feasible solution on iOS and OS X platform by studying the mechanism of Objective-C loading Class, thus achieving the function that users do not have to reinstall APK to upgrade the application.

Keywords

Dynamic Loading, App Update, Update, Mobile Internet

基于Objective-C运行时系统的动态加载技术

夏　添[1]，李政志[2]，薛鹏飞[3]

四川大学计算机学院，四川　成都
Email: scuxiatian@foxmail.com

收稿日期：2017年3月3日；录用日期：2017年3月20日；发布日期：2017年3月23日

摘　要

21世纪，移动互联网飞跃式的发展，全球互联网用户数量急速上升。为了满足不同用户在不同使用场景下的需求，移动应用不断推陈出新，在其不断完善的同时，人们对移动应用也产生了依赖性。所以，为了适应市场不断变化的需求和使用场景，各类服务提供商不得不快速的迭代产品。反之，由于产品的过于快速的迭代，使用户需要不断的更新和下载应用，造成用户体验较差。本文通过对Objective-C加载Class的机制的研究，采用包bundle机制

文章引用: 夏添, 李政志, 薛鹏飞. 基于 Objective-C 运行时系统的动态加载技术[J]. 计算机科学与应用, 2017, 7(3): 192-198. https://doi.org/10.12677/csa.2017.73024

在iOS和OS X平台上实现了动态加载。从而实现了让用户不用重新安装APK就实现应用升级更新功能。

关键词

动态加载，应用更新，更新，移动互联网

1. 研究背景

在如今，智能手机与互联网高速发展，也是未来的方向。而 iOS 系统手机作为高端智能手机的代表，将引领全球进入智能时代。Objective-C 作为 iOS 系统的主要开发语言，拥有相当多的动态特性，这些特性在运行程序时发挥作用，而不是在编译或链接代码时发挥作用。Objective-C 运行时系统实现了这些特性，而这些功能为 Objective-C 语言提供了非常多的强大功能和灵活性。开发人员使用它们能够以实时方式促进程序的开发和更新，而无需重新编译和重新部署软件。在运行时，Objective-C 语言会执行其他语言在程序编译或链接时会执行的许多常规操作，如确定类型和方法解析。这些操作还可以提供 API，使编写的程序能够执行额外的运行时操作，如动态内省和以动态方式创建和加载代码[1]。

在早期的 iOS 市场上发布应用，如果应用中含有广告，可能会审核不通过，那么一些个人开发者会在服务器端配置一个开关，审核时关闭开关，应用就不显示广告了，通过审核后，再通过服务器端把广告开关打开，这样就可以很好的规避应用市场的审核。之后，应用市场通过扫描 APK 内的 manifest 甚至 dex 文件，以此来确定开发者是否在 APK 包里植入广告代码。在服务器端配置开关参数的方法不行了，很多开发者就另辟蹊径。在应用的原生 APK 代码内部写入广告代码，在用户下载安装运行后，再从服务器下载广告代码，运行，实现广告的功能。该方法是可行的，这就是动态加载[2]。

那么接下来我们将具体来说明动态加载。1、应用能够通过在本地加载一些不存在的文件来实现特定的功能；2、这些可执行文件具有可替换性；3、动态加载不包括静态资源(如：启动图，主题，广告在服务器端的控制参数开关等) [3]。对 Objective-C 语言的动态特性的研究不仅能提高编程的灵活性，还能大大提升系统的运行效率，对于 iOS 系统的发展甚至智能手机的发展都有着深远的影响。

2. 传统的动态加载技术

传统 PC 端，动态加载技术被广泛使用，比如有些输入法，在初次安装的时候没有截图功能，在用户第一次使用的时候，会自行从服务器端下载安装，这样就能使用截图功能了[4]。此外，DLL 文件(Dynamic Link Library)在许多软件的安装目录中大量存在，一些特定的功能就是 PC 软件通过调用这些 DLL 里的代码执行的，这些技术就是一种动态加载[4]。在 JAVA 中，JAR 作为其可执行文件，运行于虚拟机 JVM 上，虚拟机则通过 ClassLoader 加载 JAR 文件，并执行其中的代码。所以在 JAVA 中的动态加载，也是通过动态调用 JAR 文件来实现的[5]。

3. 基于 Objective-C 运行时系统的动态加载技术的优势

Objective-C 程序通过动态加载功能可以根据加载可执行代码和源代码，而无需在启动程序时就加载

程序的所有组件。可执行代码(在加载前就链接好的)可以含有新的类，并使这些新类在运行程序时整合到整个程序中[6]。这种程序代码和数据资源的延迟加载方式可以提高程序的整体性能，因为它降低了对系统内存的需求。该方法还提高了程序可拓展性，因为它能够使新软件在不更改已存在程序的情况下，以动态方式将新增代码添加到程序中[7]。

4. Objective-C 动态协议类型

运行时系统通过动态类型功能可以在运行时程序时决定对象的类型，因而可以使运行时因素能够在程序中指定哪种对象[8]。Objective-C 通过 id 类型支持动态类型。id 数据类型是一种 Objective-C 独有的数据类型。其变量可以存储任何数据类型的 Objective-C 对象，而不论该对象是哪种类的实例。以下是静态类型和动态类型的使用：

```
// 声明为静态类型
Atom *atom1 = [[Atom alloc] init];
// 声明为动态类型
id atom2 = [[Atom alloc] init];
```

由于 Objective-C 既支持静态类型又支持动态类型，所以可在方法声明中使用不同等级的类型信息：

```
// 输入参数可以接收任何类的实例
-(NSInteger) computeValue1:(id)parameter;
// 输入参数可以接收任何遵守Writer协议的对象
-(NSInteger) computeValue2:(id<Writer>)parameter;
// 输入参数可以接收任何类型为NSNumber的对象
-(NSInteger) computeValue3:(NSNumber *)parameter;
// 输入参数可以接收任何类型为NSNumber且遵守Writer协议的对象
-(NSInteger) computeValue4:(NSNumber<Writer> *)parameter;
```

5. Objective-C 动态绑定

动态绑定指在运行程序时(而不是在编译时)将消息与方法对应起来的处理过程。因为许多接收器对象可能会实现相同的方法，调用方法的方式会动态变化。因此，动态绑定实现了 OPP 的多态性，可以在不影响既有代码的情况下，将新对象和代码连接或添加到系统中，从而降低对象之间的耦合度。同时通过消除用于处理多选情景的条件逻辑，动态绑定还能够降低程序的复杂程度[9]。以下面代码段为例(Hydrogen 类为 Atom 类的子类，而 logInfo 方法定义在 Atom 类中)：

```
id atom = [[Hydrogen alloc] initWithNeutrons:1];
[atom logInfo];
```

执行这段代码时，运行时系统会通过动态绑定确定变量 atom 的实际类型，然后使用消息选择器将该消息与接收器的实例方法对应起来。在本例中，atom 的类型被设置为 Hydrogen*，因此运行时系统会搜索 Hydrogen 类的实例方法 logInfo，如果没有找到，就会在 Hydrogen 类的父类中寻找相应的实例方法。运行时系统会一直在类层次结果中寻找该实例方法，直到找到它为止。

动态绑定是 Objective-C 的一种继承特性，它不需要任何 API。使用动态绑定甚至可以将消息选择器设置为在运行程序时确定的变量[10]。

6. Objective-C 动态加载的方法和过程

Objective-C 程序通过动态加载功能可以根据需要加载可执行代码和源代码，而无需在启动程序时就加载程序的所有组件。该方式不仅降低了对系统内存的需求，还提高了程序的可扩展性，因为它能够使新软件在不更改已存在程序的情况下，以动态方式将新增代码添加到程序中。苹果公司提供了以动态方式加载软件的包 bundle 机制[11]。

包是一种软件交付机制。它由具有标准层次结构的目录以及该目录中的可执行代码和源代码构成。

包可以含有可执行代码、图像、音频文件、和其他类型的代码与资源整合。它还含有一个运行时配置文件，即信息属性列表 info.plist [12]。包可以分为 3 类：

1) 应用程序包；

2) 框架包(如 Foundation 框架)；

3) 可选加载包(也称为插件，用于动态加载的自定义包)。

可以使用 Foundation 框架中的 NSBundle 类管理包。一个 NSBundle 对象就代表文件系统中的一个存储位置，该位置存储着可在程序中使用的代码和数据资源。

下面是使用 NSbundle API 动态加载自己编写的框架包的示例，共需要创建两个工程，一个命令行程序和一个可选包。

创建一个协议和一个遵守该协议的类用于测试：

Greeter 协议：

```
#import <Foundation/Foundation.h>

@protocol Greeter <NSObject>

-(NSString *)greeting:(NSString *)salutation;

@end
```

BasicGreeter 类：

```
#import <Foundation/Foundation.h>
#import "Greeter.h"

@interface BasicGreeter : NSObject <Greeter>

@end
```

```
#import "BasicGreeter.h"

@implementation BasicGreeter

-(NSString *) greeting:(NSString *)salutation{
    return [NSString stringWithFormat:@"%@,World!",salutation];
}

@end
```

创建可选包

新建工程中选择 OS X->Framework&Library->Bundle：

先将 Greeter.h 添加到可选包中。再创建一个遵守 Greeter 协议的类

CustomGreeter 类：

```
#import <Foundation/Foundation.h>
#import "Greeter.h"

@interface CustomGreeter : NSObject <Greeter>

@end
```

```
#import "CustomGreeter.h"

@implementation CustomGreeter

-(NSString *)greeting:(NSString *)salutation{
    return [NSString stringWithFormat:@"%@,Universe!", salutation];
}

@end
```

传入包路径

接下需要将可选包的路径作为参数传入命令行程序的 main 函数中：

选择工程中的 bundle 文件(记得先编译一次，不然 bundle 文件是红色的，无法使用)，然后将 bundle 文件完整路径复制下来。

回到命令行程序，点击 DynaLoader，选择 Edit Scheme，在弹出窗口的 run->Arguments->Arguments Passed On Launch 位置粘贴刚刚复制的 bundle 文件路径。

这样就完成了参数的传入。

最后就可在 main.m 中使用可选包：

```
#import <Foundation/Foundation.h>
#import "BasicGreeter.h"

int main(int argc, const char * argv[]) {
    @autoreleasepool {
        // 声明一个类型为id并且遵循Greeter协议的变量并进行测试
        id<Greeter> greeter = [[BasicGreeter alloc] init];
        NSLog(@"%@",[greeter greeting:@"Hello"]);

        // 在指定路径（通过输入参数获得）创建一个包
        NSString *bundlePath;
        // argc设置了程序的参数数量，而argv[]数组存储了参数值，数组第一个参数存储了程序的名称，
        // 因此argc大于或等于1，这里传递了一个参数（包的路径），所以argc应该等于2.
        if (argc != 2) {
            // 没有获得包路径，退出
            NSLog(@"Please provide a path for the bundle");
        }
        else{
            // 获取包的路径并创建一个NSBundle对象
            bundlePath = [NSString stringWithUTF8String:argv[1]];
            NSBundle *greeterBundle = [NSBundle bundleWithPath:bundlePath];
            if (greeterBundle == nil) {
                NSLog(@"Bundle not found at path");
            }
            else{
                // 以动态方式加载包
                NSError *error;
                BOOL isLoaded = [greeterBundle loadAndReturnError:&error];
                if (!isLoaded) {
                    NSLog(@"Error = %@",[error localizedDescription]);
                }
                else{
                    // 加载包后，使用该包创建一个对象，并向这个对象发送一条消息
                    Class greeterClass = [greeterBundle classNamed:@"CustomGreeter"];
                    greeter = [[greeterClass alloc] init];
                    NSLog(@"%@",[greeter greeting:@"Hello"]);

                    // 使用完以动态方式加载的包后进行卸载
                    // 先释放所有用包中的类创建的对象！
                    greeter = nil;
                    BOOL isUnloaded = [greeterBundle unload];
                    if (!isUnloaded) {
                        NSLog(@"Couldn't unload bundle");
```

```
                    }
                }
            }
        }
    }
    return 0;
}
```

7. 动态加载技术的作用与代价

凡事都有两面性，特别是这种非官方支持的非常规开发方式，在采用前一定要权衡清楚其作用与代价。如果决定了要采用动态加载技术，个人推荐可以现在实际项目的一些比较独立的模块使用这种框架，把遇到的一些问题解决之后，再慢慢引进到项目的核心模块；如果遇到了一些无法跨越的问题，要有能够迅速投入生产的替代方案。

作用：

1、规避 APK 覆盖安装升级的局限性，提高了用户体验，而且也能规避一些安卓市场的限制；

2、动态修复一些比较紧急的 bug；

3、提高启动速度，因为插件模块可以在需要时才初始化，叫做懒加载；

4、主项目和插件项目可以并行开发，特别是应用体积较大时，可以把一些模块改为动态加载，以插件的形式，这样也可以减少主项目的体积，提高项目的编译速度；

5、减少主项目的 DEX 方法数量，彻底解决 65535 问题；

6、从项目管理的角度上来说，分割模块的方式，也做到了代码分离，大大降低了模块之间的耦合度，利于代码管理和 bug 走查；

7、在 Android 应用的推广其他应用时，可以利用动态加载技术，让用户在不用下载新的 APK 的情况下体验新应用的功能。

代价：

1、开发方式繁琐，和常规开发不同；

2、非常规的开发方式，存在兼容性的风险，特别是在一些老旧机型上；

3、随着动态加载框架复杂程度的加深，项目的构建过程也变得复杂，有可能要主项目和插件项目分别构建，再整；

4、由于插件项目可能是独立开发，可能遇到主项目和插件项目的运行环境的不同，代码逻辑容易出现 bug，而且在主项目中调试插件十分繁琐；

5、兼容性问题，也是采用动态加载的产检在使用系统资源时经常发生的。

8. 结束语

本文详细阐述了移动互联网时代下 iOS 应用的动态加载技术。从 Objective-C 的 NSBundle 类入手，提出了可实行的解决方案，一定程度上的解决了应用更新快用户反复安装的烦恼。也提出了动态加载技术的优势和劣势。系统中用到的技术在应用快速迭代上的性能和用户体验的提高上具有重要的价值和广阔的应用前景。

参考文献 (References)

[1] 吴吉义, 李文娟, 黄剑平, 等. 移动互联网研究综述[J]. 中国科学: 信息科学, 2015, 45(1): 45-69.

[2] Feng, Q.-C., Wen, Q.-Y. and Fan, Y.-J. (2011) A Systemic Code-Protecting Methodology for the Dex File on Android

Platform. *IEEE International Conference on Intelligent Computing and Intelligent Systems* (*ICIS*), Vol. 2, IEEE Beijing Section, China, Guangdong University of Technology, China, 4.

[3] 曹森, 苏贵斌. 软件开发中的设计原则[J]. 软件导刊, 2012(1).

[4] 黄金国, 罗震. 手机应用程序开发架构的研究[J]. 计算机工程与科学, 2010(11).

[5] Zhang, X.L., Breitinger, F. and Baggil, I. (2016) Rapid Android Parser for Investigating DEX Files (RAPID). *Digital Investigation*, **17**, 28-39.

[6] 张峰, 李基亮. 校园私有云存储方案的探索[J]. 华东师范大学学报(自然科学版), 2015(B03): 139-145.

[7] Phillips, B. and Hardy, B. (2014) Android Programming: The Big Nerd Ranch Guide. Big Nerd Ranch.

[8] 詹新林. 基于架构的软件设计方法[D]: [硕士学位论文]. 上海: 上海师范大学, 2009.

[9] 王良, 王伟平, 孟丹. FVS k-匿名: 一种基于 k-匿名的隐私保护方法[J]. 高技术通讯, 2015(3): 228-233.

[10] Weiss, M.A. (2015) Data Structures and Algorithm Analysis in Java: International Edition, 3/E. *Journal of the American Chemical Society*, **130**, 2156-2157.

[11] Horstmann, C.S. and Cormell, G. (2015) Core Java Volume I: Fundamentals. 9th Edition, by Cay S. Horstmann and Gary Cornell. *ACM Sigsoft Software Engineering Notes*, **38**, 33.

[12] Goetz, B., Peierls, T., Bloch, J. and Bowbeer, J. (2016) Java Concurrency in Practice. China Machine Press, Addison-Wesley, 1171-1177.