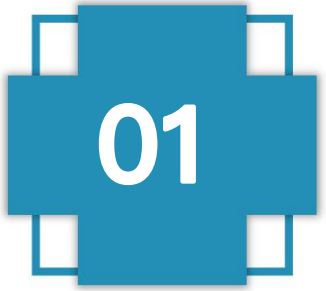
APP插件化架构探索

目录



背景

01 背景

2014年苹果在iOS上开放动态库，带给我们许多的探索机会。我们都知道，动态库是一种资源打包方式，可以将源代码、头文件、资源文件、文档等捆绑在一起，方便开发者使用。动态库在编译的时候不会拷贝到程序的可执行文件中，而是等到程序运行时，动态库才加载，不需要重新编译代码。正是有这样一个特性，才使我们可以随时替换库，这样我们就可以做很多的事情，比如插件化以及动态更新。

应用插件化：按需加载某些应用功能，在用户使用某个功能的时候让其从网络中下载，然后动态加载应用，实现功能的插件化。

应用动态更新：当应用需要更新某个功能时候，从服务器下载新版本即可。类似于小程序更新。

组件化、模块化、插件化介绍

2.1 组件化

组件化：组件化是基于可复用的目的，将一个大的系统按照“点”的形式，拆分成多个独立的组件。目的是为了解决全局工程中有很多重复代码的问题。组件化的另一个目的是为了解耦，把系统拆分成多个组件，分离组件边界和责任，便于独立升级和维护。例如：弹窗组件，网络组件、路由组件等。

2.2 模块化

模块化：模块化的目的在于将一个程序按照其业务功能做拆分，分成相互独立的模块，以便于每个模块只包含与其业务功能相关的内容，模块之间通过接口调用。将一个大的系统模块化之后，每个模块都可以被高度复用。 模块可以由许许多多个组件构成。

注意点：模块化不要求可复用，但要求高内聚；模块可以转化为组件；组件可以转化为模块，两则并没有那么泾渭分明。

组件讲究纵向分离，模块讲究横向分离

2.3 插件化

插件化：将某个功能独立出来，独立开发，独立测试，然后再插入到主工程中，动态加载。简单说：寄宿在宿主的应用（组件），具有动态加载特性。表现形式是“拔插式”，实现方式是面向接口编程。例如：动态下载应用、热修复、热更新。

组件

特点：可复用，功能相对单一或者独立，无统一接口；
表现形式：独立单元（通用能力下沉）；
实现方式常用pod库。

插件

特点：有统一接口；
表现形式：“拔插式”；
实现方式：统一接口，面向接口或协议编程。

模块

特点：高内聚，松耦合，功能相对复杂，有多个统一接口；

表现形式：独立业务模块；

实现方式：按业务逻辑独立业务功能。

“关注点”不同

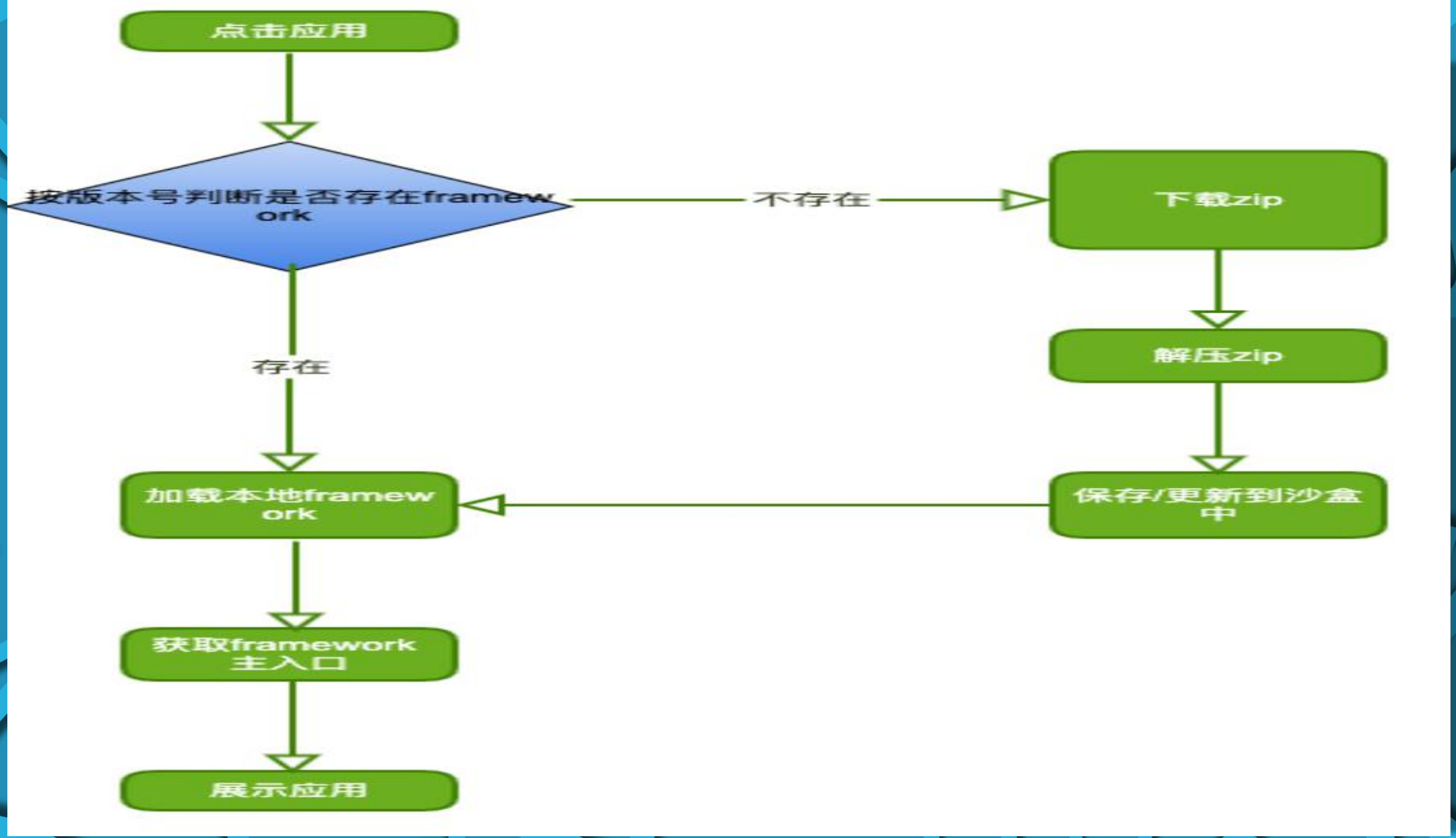
插件化实现思路

3.1 插件化实现思路

将某个模块编译成动态framework，在用户想使用某个功能的时候，从服务器上下载动态库，然后加载动态库，显示模块应用。

源码实现：

https://github.com/wequick/Small/blob/master/iOS/Small


点击应用
按版本号判断是否存在framework
不存在
下载zip
解压zip
保存/更新到沙盒中
存在
加载本地framework
获取framework主入口
展示应用

04

注意事项与总结

注意事项与总结

系统在加载动态库时，会检查Framework的签名，
签名中必须包含TeamIdentifier，并且Framework和主App的TeamIdentifier必须一致

总结，很遗憾，目前仅仅适用于对于不需要上架的企业级应用

思考：框架与架构的区别？