

# DynamicCocoa：滴滴 iOS 动态化方案的诞生与起航

2016-12-19 孙源 iOS开发by唐巧

## 推荐序

---

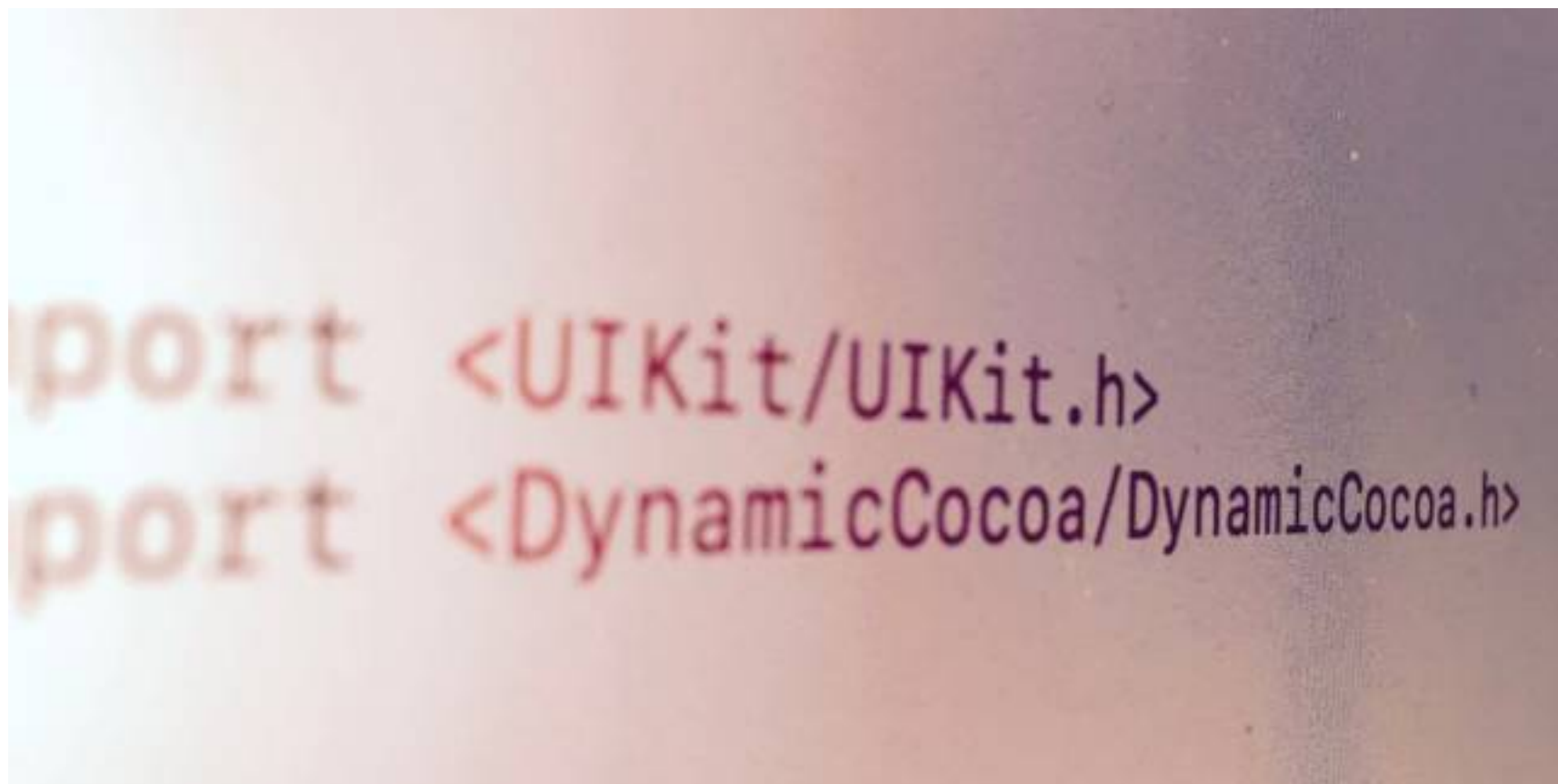
我和本文的作者孙源早就认识，我参加过孙源组织的好几次线下分享活动。孙源是一个对技术喜欢刨根问底的人，热爱分享和开源，同时特别喜欢狗，除了他的头像外，他在百度时的开源组织都叫 forkingdog。

孙源后来去了滴滴 App 架构组，近期一直在潜心研究编译器相关的东西，如果你关注过他今年在 MDCC 和 iDev 大会上的分享的话，你就会发现他的分享内容都与编译器有关。其实，他分享这些是有原因的，因为他们团队在做一个很牛逼的动态化方案，可以直接把 Objective-C 代码经过编译，转换成 JS 代码后下发给客户端。

我一直不知道这里面的技术细节，直到上周末他找到我，希望在这里发表他们的 iOS 动态化方案，于是我就有幸成为了本文的最早一批读者。

读完之后，我意识到，他们团队的这篇文章将会极大促进业界内对于 iOS 动态化方案的探索，也将会对其它动态化方案，例如 React Native, Weex, LuaView, 甚至 JSPatch 造成深远影响。

这篇文章你一定得看，感谢滴滴 App 架构组授权发表。



# 方案诞生

---

动态化一直是 App 开发梦寐以求的能力，而在 iOS 环境下，Apple 禁止了在 Main Bundle 外加载和执行的自己的动态库，所以像 Android 一样下发原生代码的方案被堵死。

后来像 ReactNative、Weex 这样的基于 Web 标准的跨端方案出现，各大公司都有对其进行尝试，但对于滴滴现状，也许并不适合：

- 滴滴 App 强交互、以地图为主体、端特异性高
- 客户端人员充足，跨技术栈学习和开发有较大成本
- 大量固化 Native 代码，重写成本高

所以我们思考，能不能做一套保持 iOS 原生技术栈、不重写代码就神奇的拥有动态化能力的方案呢？

于是，我们设计和实现了一个具有里程碑意义的 iOS 专属动态化方案：**DynamicCocoa**

## DynamicCocoa 初识

---

**DynamicCocoa** 可以让现有的 **Objective-C** 代码转换生成中间代码（JS），下发后动态执行，相比其他动态化方案，优势在于：

- 使用原生技术栈：使用者完全不用接触到 JS 或任何中间代码，保持原生的 Objective-C 开发、调试方式不变
- 无需重写已有代码：已有 native 模块能很方便的变成动态化插件
- 语法支持完备性高：支持绝大多数日常开发中用到的语法，不用担心这不支持那不支持
- 支持 HotPatch：改完 bug 后直接从源码打出 patch，一站式解决动态化和热修复需求

不论是动态化还是 HotPatch，我们都能让开发者：“Write Cocoa, Run Dynamically”



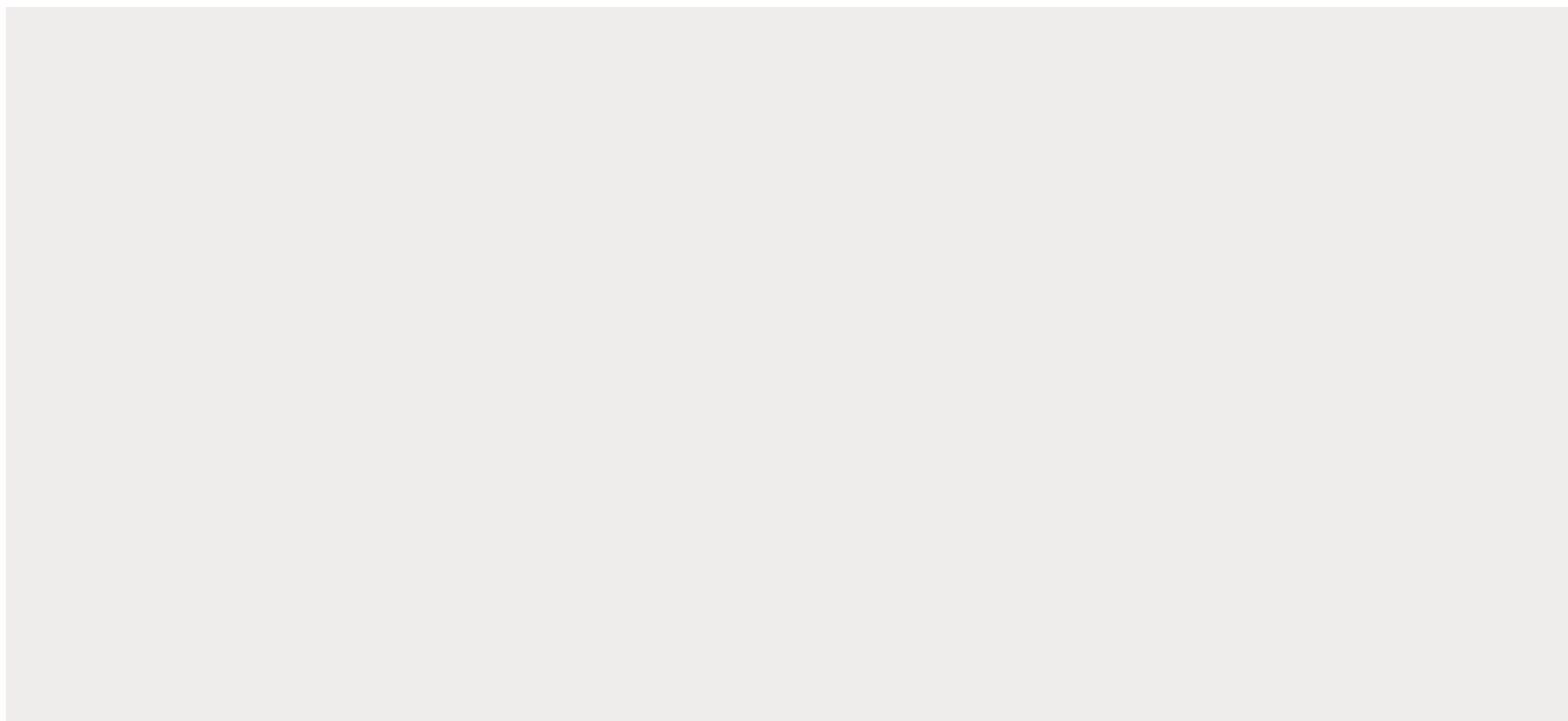
# 语法支持

---

DynamicCocoa 能支持绝大部分日常使用的 Objective-C / C 语法，挑几个特殊的：

- **完整的 Class 定义**：interface、category、class extension、method、property，最重要的是支持完备的 ivar 定义，保持和 native 完全一致的实例内存结构
- **ARC**：可以正确处理 strong、weak、unsafe\_unretained 等对象的引用计数，对象的 ivar 也可以正确的释放
- **C 函数**：支持 C 函数的定义与 C 函数的调用、内联函数的调用
- **可变参数**：支持 C 与 OC 的可变参数方法的调用，如 NSLog
- **struct**：支持任意结构体的使用，无需额外处理
- **block**：支持创建和调用任意参数类型的 block
- **其他 OC 特性**：如 @selector、@protocol、@encode、for..in 等
- **其他 C 特性**：支持使用宏、static 变量、全局变量，取地址等

举个栗子，你可以放心的使用下面的写法，并能被正确的动态执行：



# 资源支持

---

一个功能模块，除了代码外，资源也是必不可少的，DynamicCocoa 的动态 bundle 支持：

- xib 和 storyboard
- xcassets
- 不放在 xcassets 里的图片资源
- 其他资源文件

对于习惯于使用 IB 来开发 UI 的人来说，这将是一个很好的开发体验。

## 工具链支持

---

我们使用 ruby 开发了一套命令行工具（ 类比为 xcodebuild ），大幅简化了配置开发环境、OC 代码转换、资源处理、打包的复杂度，它可以：

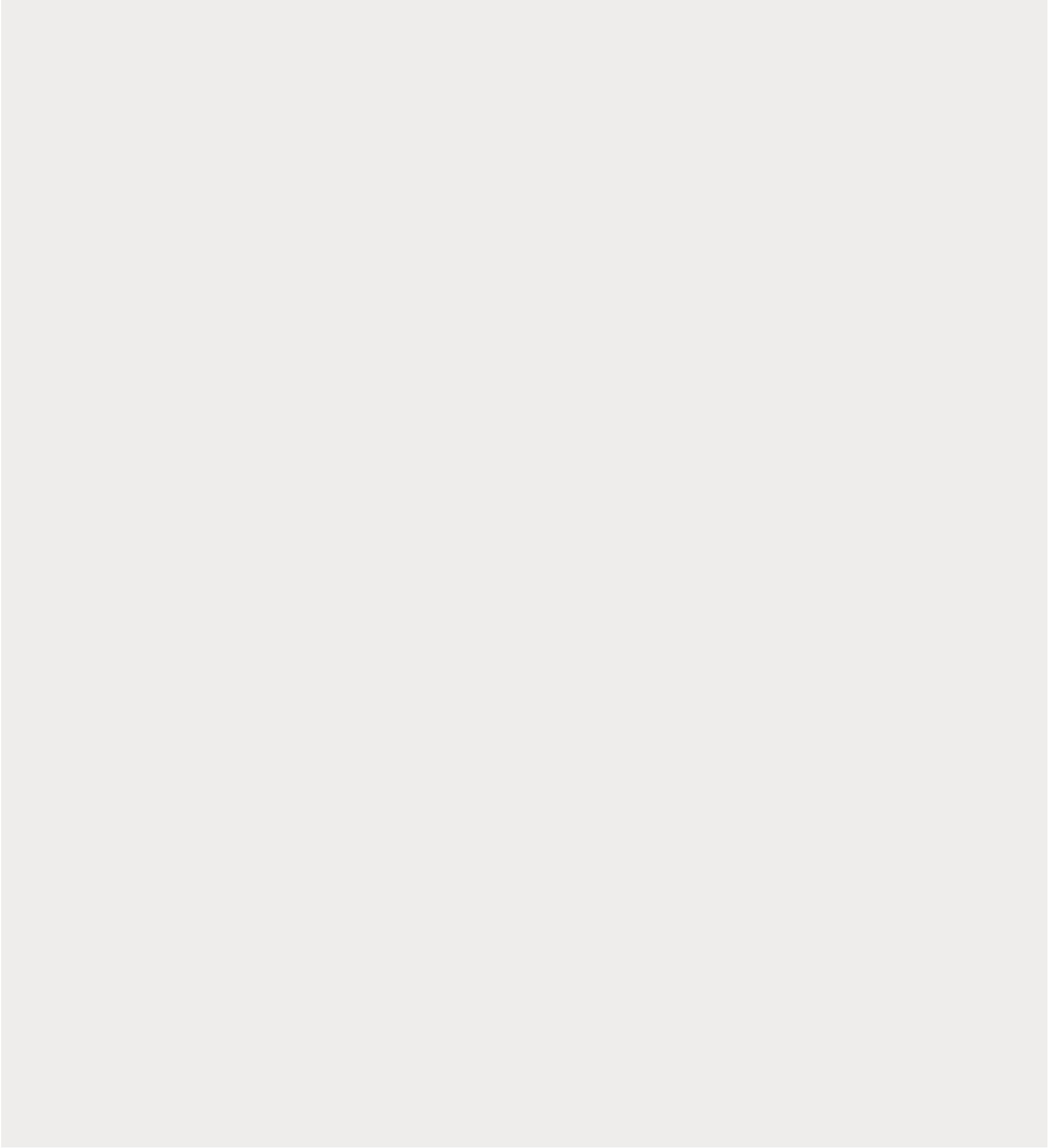
- 解析 Xcode Project：读取工程编译选项，保持和 native 编译参数一致
- 增量编译：缓存 JS 转换结果，只重新转换修改过的文件，大幅提高 build 速度
- 链接：分析类依赖，将多个 JS 按依赖顺序合并，提高文件读取速度
- 资源编译：编译用到的 xib、storyboard 和 xcassets
- 打包：将 JS、资源等打包成 bundle

对于开发者来说，就像 pod 命令一样，所有操作都可以通过这个命令完成。

## 动态插件开发流程

---

首先 App 中需要集成 DynamicCocoa Engine SDK，用来执行下发的 bundle 开发到发布的流程如下图所示：



当然，DynamicCocoa 只提供命令行工具和 Engine SDK，可以完成本地打包、运行和测试，而线上发布后台、服务端、CDN 等需要自行解决。

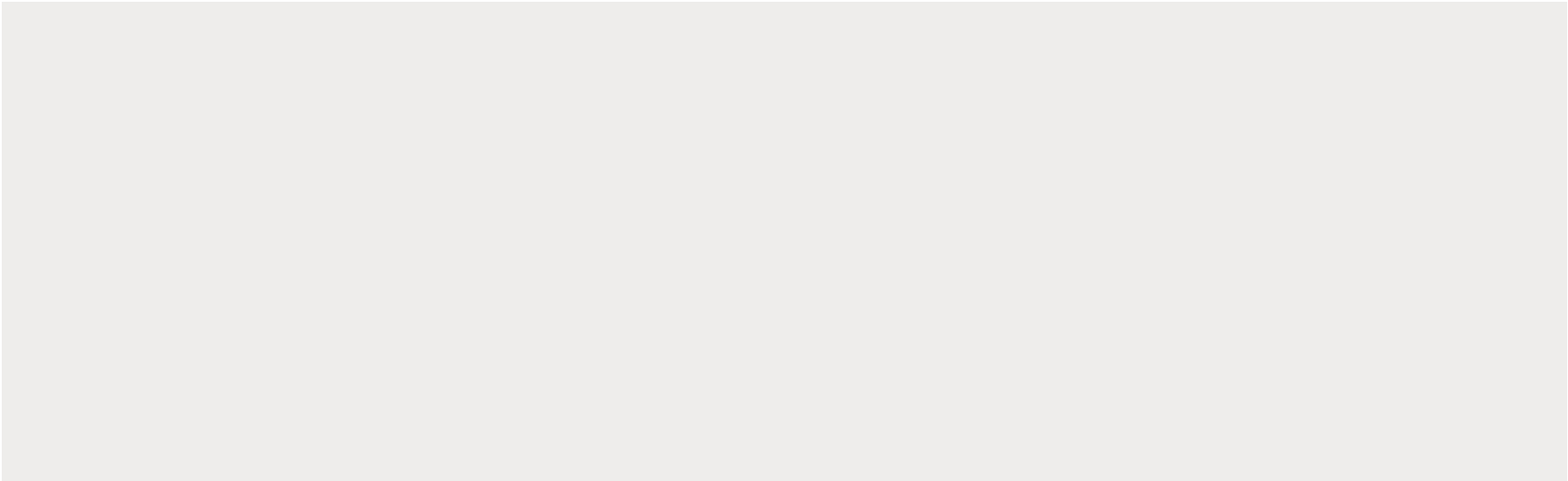
在滴滴内部，我们构建了开发、Review、线上回归测试、灰度、发布、回滚、统计的闭环系统，以服务的形式给内部接入。

## HotPatch 过程

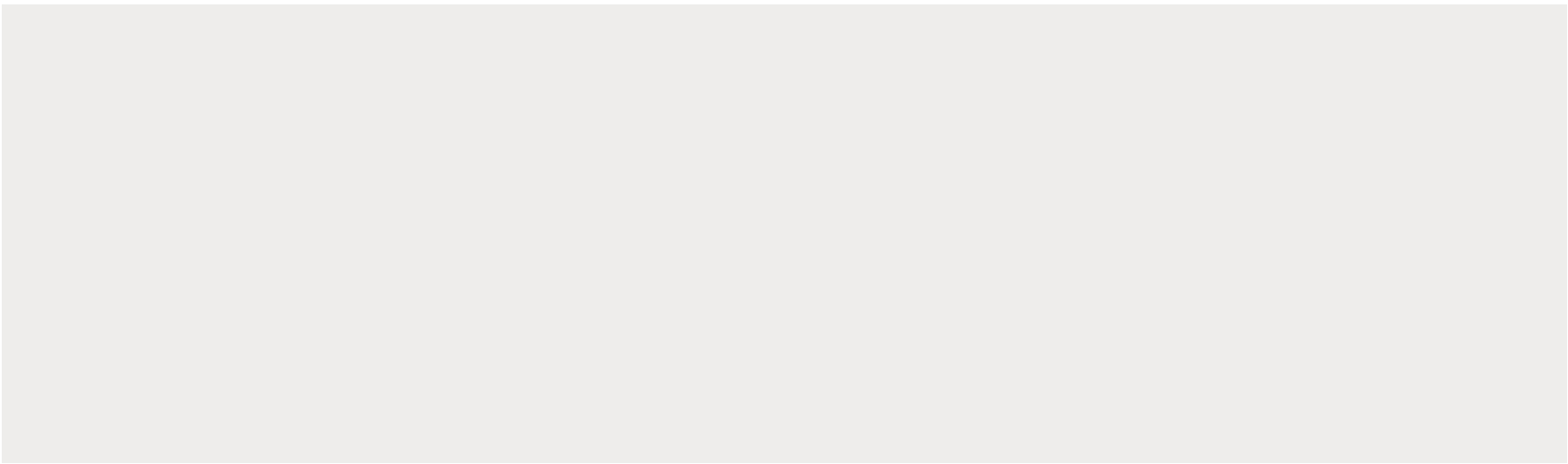
---

**HotPatch** 本质上是方法粒度上的动态化，所以在整个框架搭建起来后，HotPatch 也不难实现，使用 DynamicCocoa 做热修复的最大优势是开发者依然只对源码负责，修改完 bug 后，打个 patch 包，修复成功后把源码改动直接 push 到代码仓库就行了。

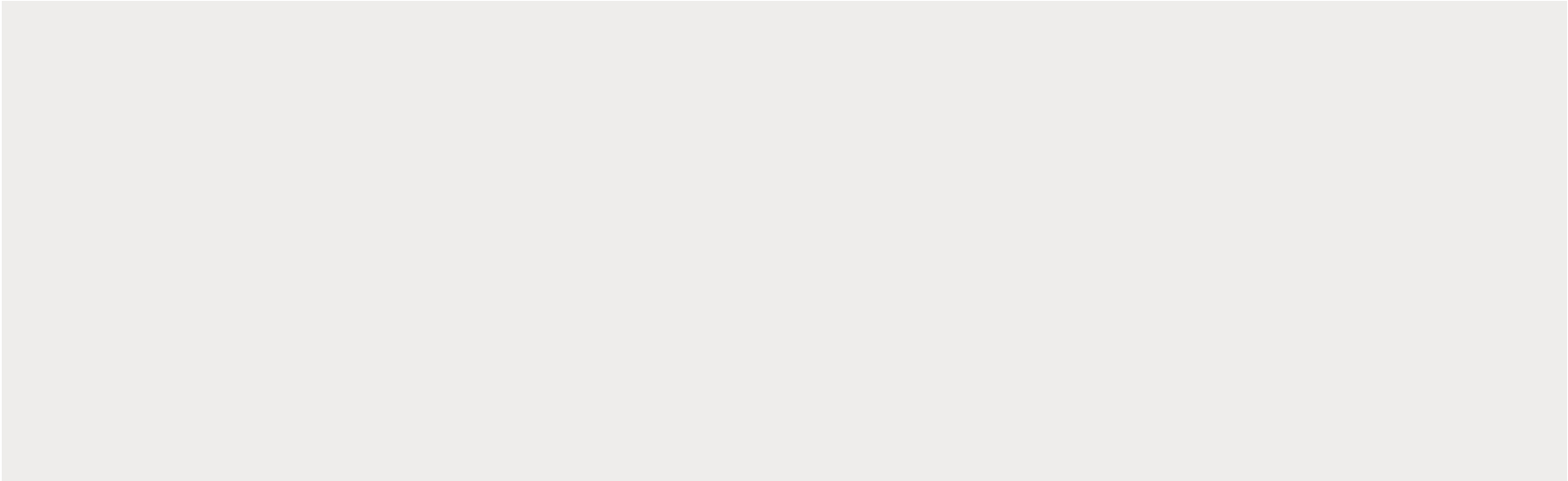
假设我们发现了下面的 bug：



然后在 native 进行修复并自测：



自测完成后，在这个方法后面添加一个神奇的 **Annotation**：



使用命令行工具在 patch 模式下进行打包，就能把所有标记了的 method 提取出来，分别转换成 JS 表示，打到一起进行发布。

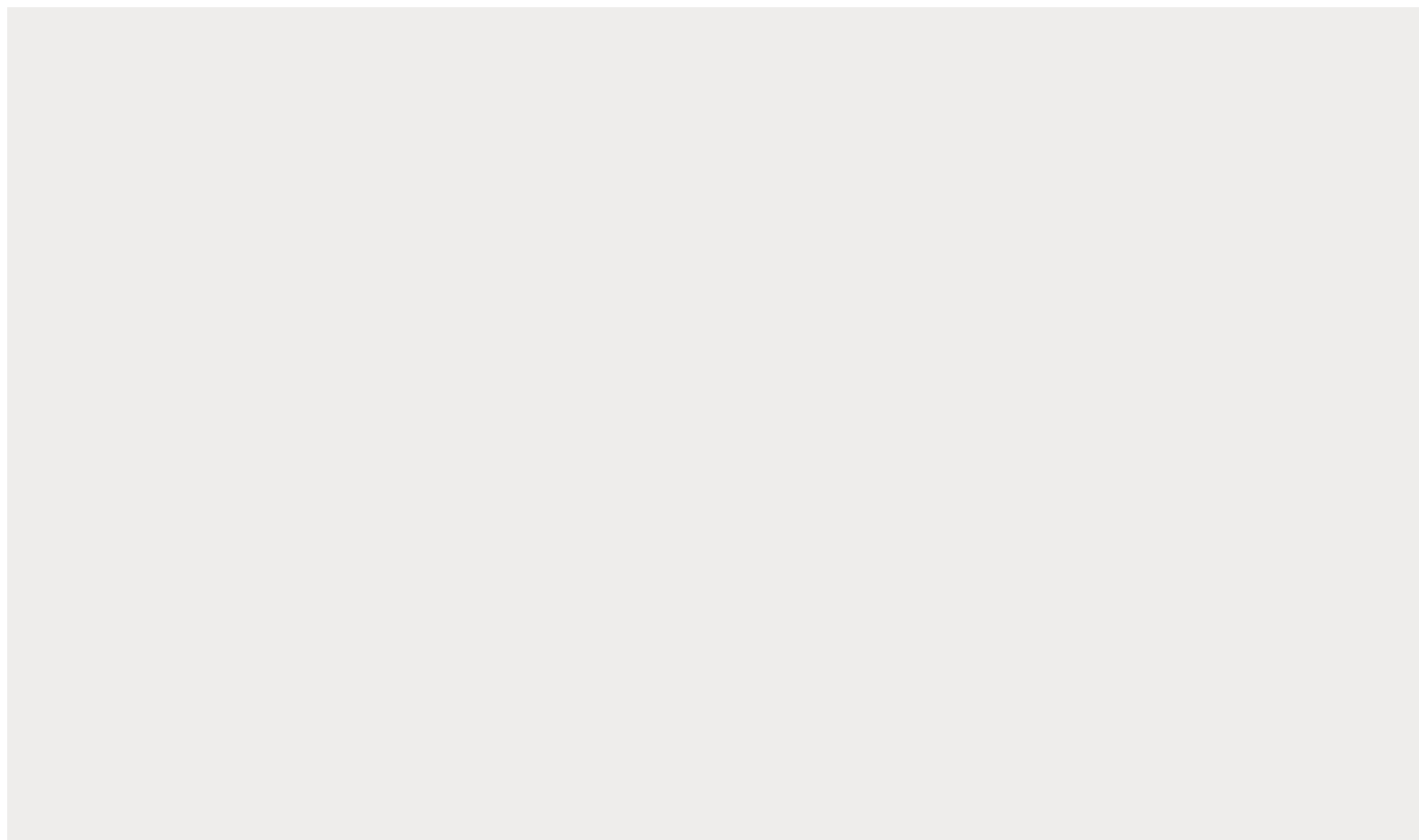
除了修改一个方法外，patch 模式还支持：

- 调用原方法
- 新增一个方法
- 新增一个 property 来辅助修复 bug
- 新增一个 Class

最后，开发者可以安心的把修改后的代码（甚至可以保留 Annotation）git push，完成热修复工作。

## 打开黑箱

---



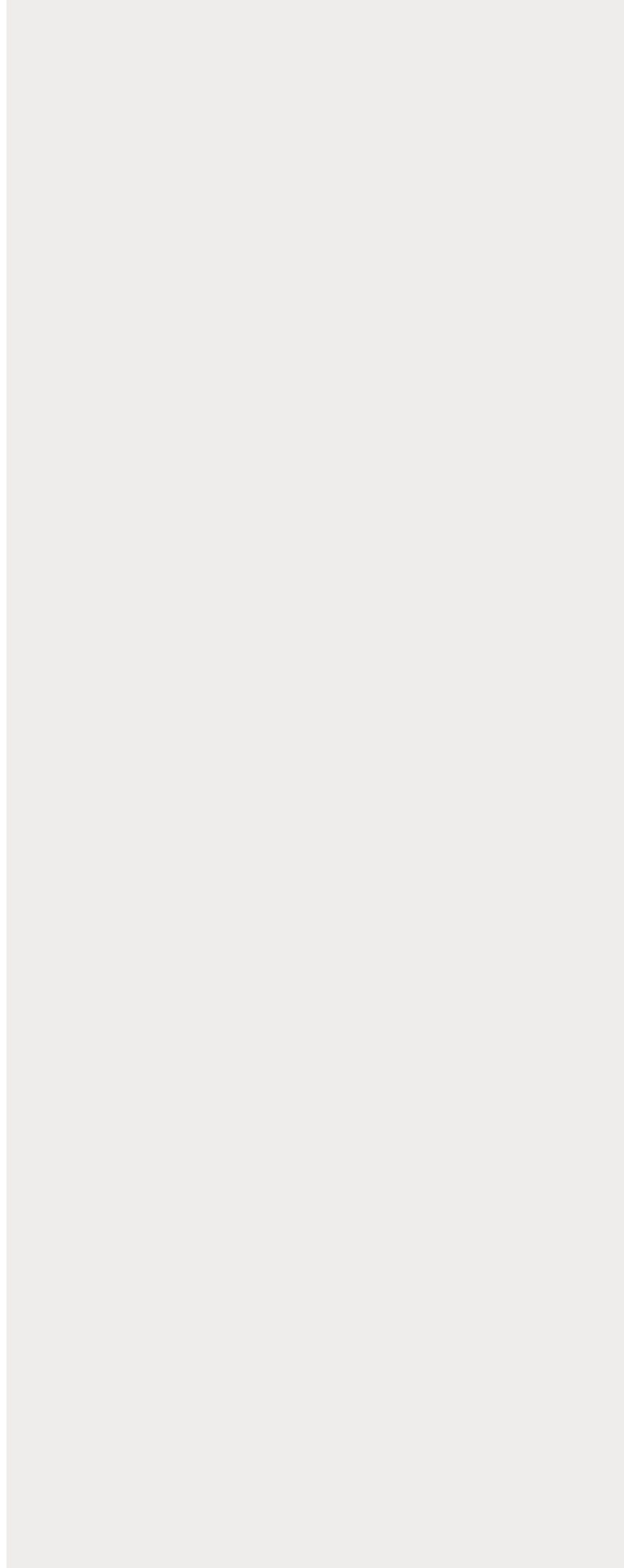
就像 Objective-C 是由 Clang 编译器和 Objective-C Runtime 共同实现一样，DynamicCocoa 也是由对应的两部分构成：

- 在 Clang 的基础上，实现了一个 OC 源码到 JS 代码的转换器
- 实现 OC-JS 互调引擎的 DynamicCocoa SDK

我们知道，Clang-LLVM 的标准编译流程是从源代码经过预处理、词法解析、语法解析生成语法树，CodeGen 生成 LLVM-IR，进入编译器后端进行优化和汇编，最终生成目标文件 (Mach-O)

而我们既希望 Clang 帮助完成源码处理的步骤，又希望生成结果是 JS 表示形式，于是在 Clang 生成抽象语法树（AST）后，我们进行接管，实现了一个 OC2JS CodeGen，遍历各个特定语法节点输出 JS 表示：

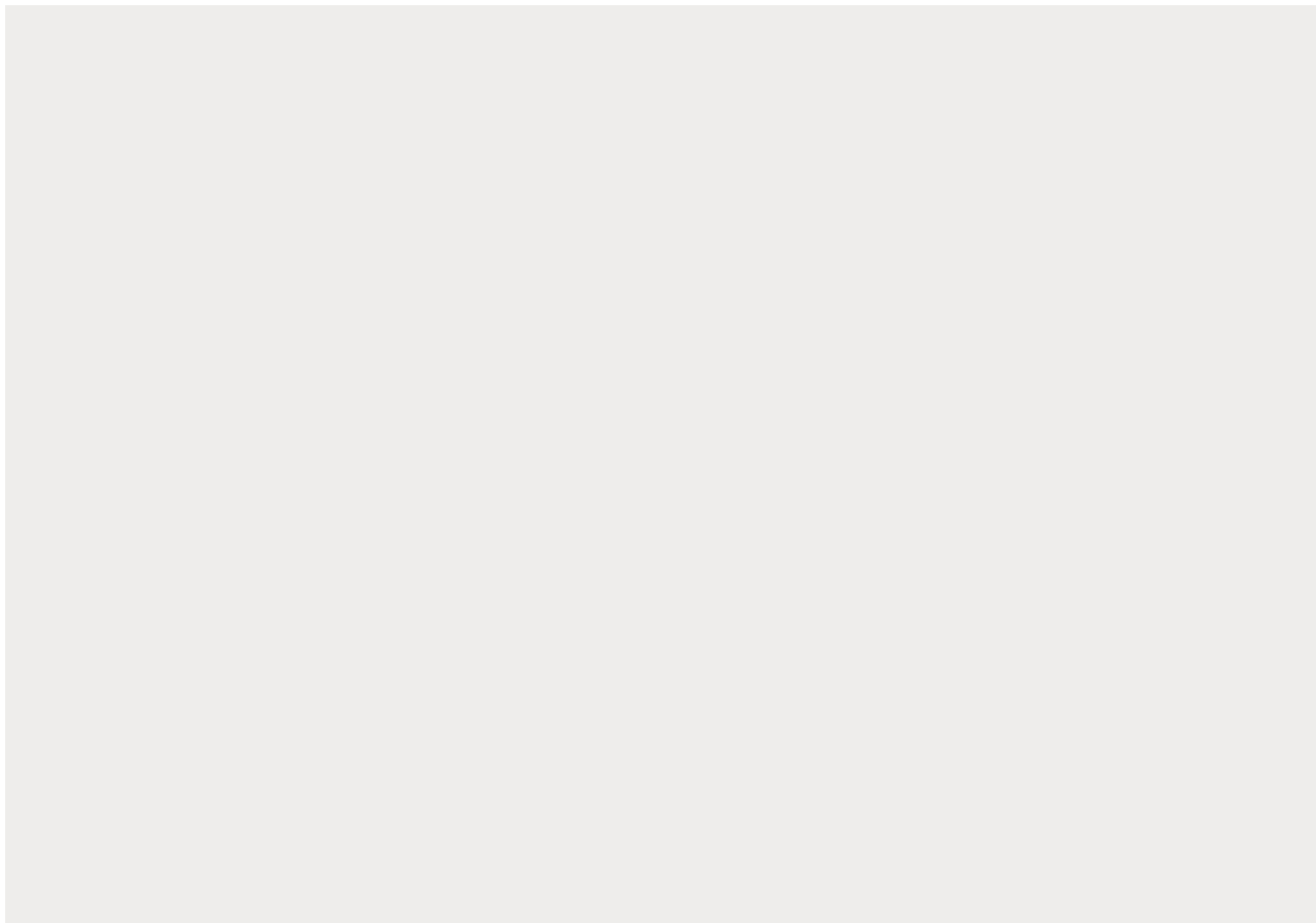




由于转换器和 Clang 前端标准编译流程相同，所以只要 native 代码能 build，转换器就能 build，这也是 DynamicCocoa 能让动态包和 native 保持严格一致的先决条件。

注：转换器是基于 Clang 开发的独立命令行工具，它的使用并不会对原有的 Xcode 工程产生任何影响。

另一部分是要集成进 App 的 DynamicCocoa SDK，它的职责是为 JS 中间代码提供 Runtime 环境，实现 OC-JS 的互调引擎，能够加载动态 bundle，提供便捷的 API，整体架构如下：



其中一些有趣的点：

- 底层使用 libffi 来处理各个架构下的 calling conventions，实现 caller 调用栈的构建和 callee 调用栈的解析，用于实现 OC / C 函数调用、动态 imp、block 等。
- 由于 JS 的弱类型，数值变量在做计算时很容易丢失类型信息，比如 `int a = 1 / 2;` 在 OC 中表示整除，结果为 0，但进入 JS 就都会按照 double 计算，结果为 0.5，造成了不一致。所以 DynamicCocoa 接管了 JS 中的类型信息，强转或运算符都需要特殊处理。
- 为了实现 block，我们构造了和 native block 一致的内存结构，不论是 JS 创建的 block 还是 native 传进 JS 的 block，都可以无差别的调用。
- 虽然 runtime 提供了动态创建 OC Class 的 API，但只能创建 MRC 的 Class，导致 ARC 下 ivar 并不会乖乖释放，我们深入到 Class 和实例真实内存结构中，给动态创建的类增加了 ARC 能力，并按照 Non-Fragile ABI 模拟真实 ivar 内存布局 and ivar layout 编码，如果你重写了 dealloc 方法，DynamicCocoa 甚至能够像 native 一样自动调用 super。

## DynamicCocoa 带来的改变

DynamicCocoa 动态化技术给 App 开发带来了很大的想象空间：

- 低成本的动态化：无需额外学习，无需重写代码，可以快速的将已有模块动态化
- 协作方式：对于大团队，发布版本不必再彼此牵制
- 功能快速迭代：无需经过审核和 App Store 发版，像 h5 一样随发随上
- App 瘦身：native 只需要留好插件入口，实现由网络下发，减少 App 体积
- AB Test：不必局限于 native 埋进去的 AB 功能 Test，发版后能动态下发各种 Test

相比跨端方案，也带来了一个新思路：iOS 和 Android 都保留 native 开发模式，用各自的方式将 native 代码直接动态化，保持各平台的差异性。

## Q&A

---

### 与 JSPatch 有什么区别？

两者思路都是实现 JS 和 OC 的互调：DynamicCocoa 的重点是动态化能力，优势在于完全不用写 JS 和更多的语法特性支持；对于 HotPatch 来说 JSPatch 是更加小巧、轻量的解决方案。

### 这套框架在滴滴 App 有上线使用么？

有，在滴滴 App 已经上线并使用了好几个版本，如滴滴小巴、专车接送机都有过 10k 级别的动态化模块上线。

### 动态包运行的性能是否有很大下降？

动态 JS 代码的运行要经过频繁的 JSCore 和 OC 间的切换，性能相比 native 必定会有损耗，但经过优化，现在已经达到了无感知的程度：在我们的实际使用中，若不在页面上添加特定标志，开发者和 QA 都无法分辨出当前页面运行的是 native 还是动态包... 后续会有详细的性能分析和大家分享。

### 动态包大小如何？

与资源大小和 native 源码量有很大关系，不考虑资源的情况下，量级大概在 10000 行代码 100kb 的动态包。

### 是否支持多线程？

现在简单的支持 GCD 来处理多线程，可以使用 `dispatch_async` 将一个 block 放到另一个

queue 中执行。

## 如何定位动态包的 crash?

动态 JS 代码运行在 JSCore 中，并没有直接获取调用栈的方式，我们提供了 stack trace 功能，将最近调用栈中每个 JS 到 OC / C 的互调都记录下来，在发生 crash 时便可以取出来作为附加信息随 crash 日志上报给统计平台，方便问题的定位。

## 会不会过不了苹果审核?

市面上很多动态化、HotPatch 方案都基于 JS 的下发，运行在原生 JSCore 上，相信只要不在审核期间下发动态功能，Apple 是不太会拒绝的。

## 有没有可能支持 Swift 直接动态化?

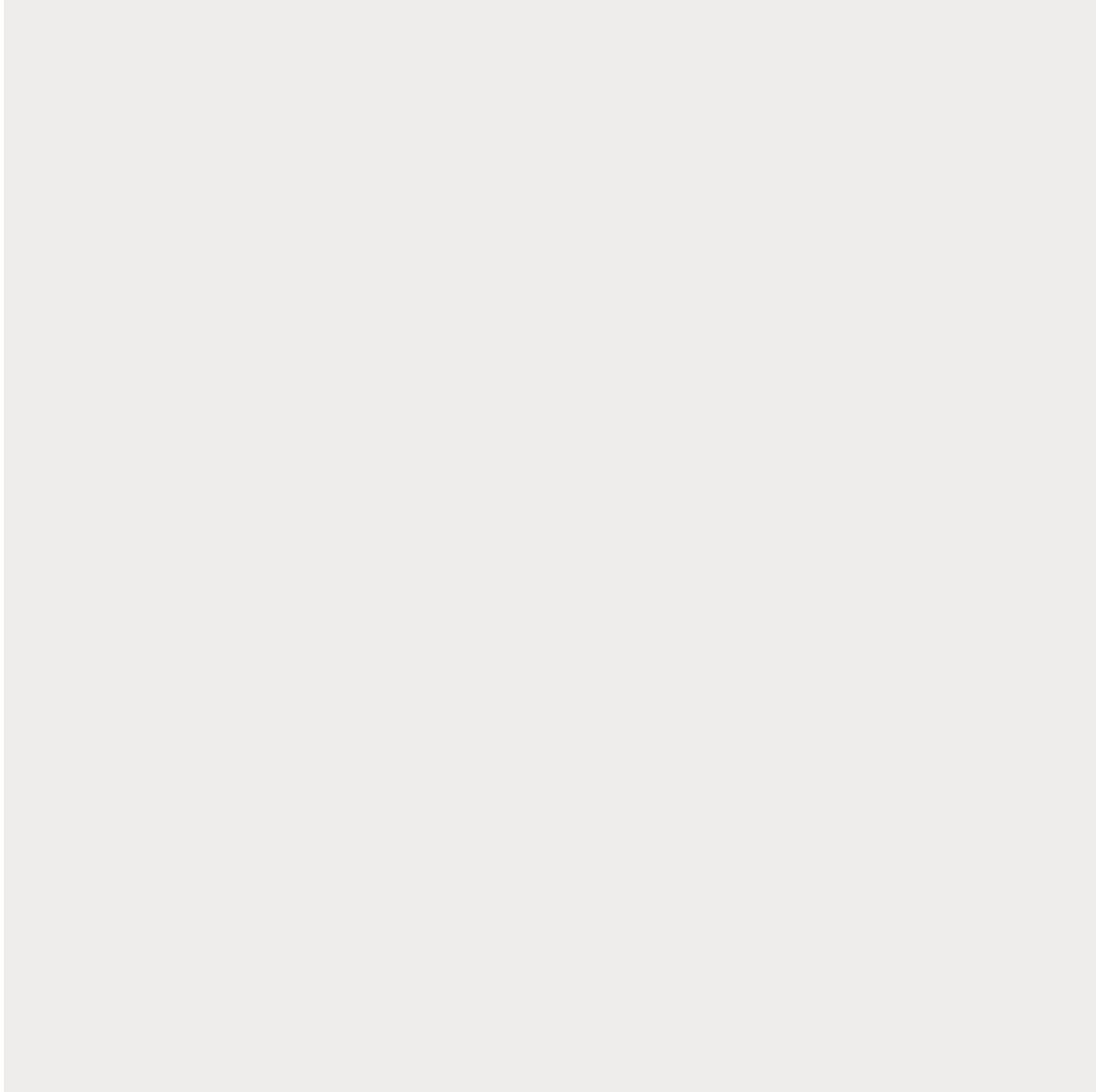
相比 OC，Swift 的动态化和 HotPatch 更加有难度，但我们已经有了可行的方案，是可以做到的，只是对于当前滴滴的现状（绝大多数都在用 OC 开发），紧急程度并不高，后面再考虑支持。

## 是否有开源计划?

有，我们正在积极的准备相关事项，于 2017 年初考虑开源。

## 该从哪里关注后续进展?

请关注滴滴 App 开发技术微信公众号 **DDApp**，我们会在上面发布 DynamicCocoa 的最新的进展，也将会把滴滴 iOS 和 Android 开发的干货技术文章分享给大家：



以上。

---