微信 Android 客户端架构演进之路

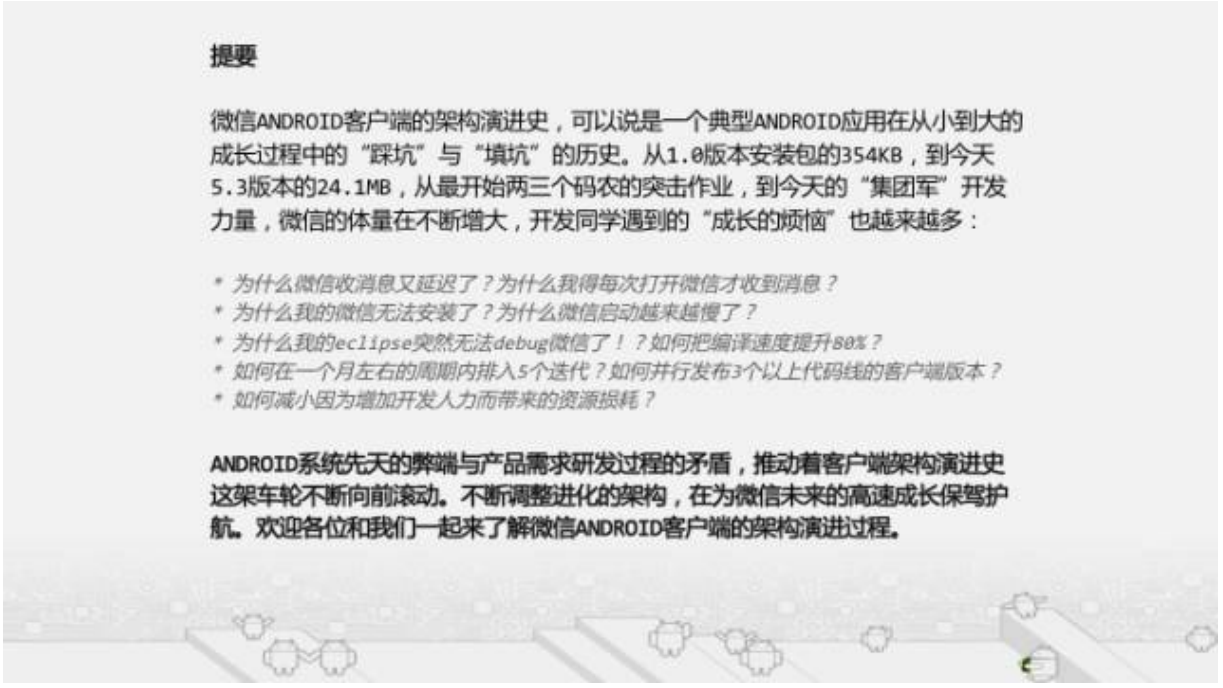

童仲毅 学生开发者 @ 复旦大学

这是一个典型的 Android 应用在从小到大的成长过程中的 “踩坑” 与“填坑”的历史。互联网的变化速度如此之快，1 年的时间里，可以发生翻天覆地的变化。今天在这里， 重新和大家回顾微信客户端架构的演进过程，以及其背后的开发团队、流程的变化与思考。来自『移动开发前线』公众号。

原文 mp.weixin.qq.com

本文来自微信公众号「移动开发前线」

笔者在InfoQ举办的ArchSummit深圳2014的架构师峰会上，分享了微信Android客户端的架构演进史。可以说，这是一个典型的Android应用在从小到大的成长过程中的“踩坑”与“填坑”的历史。互联网的变化速度如此之快，1年的时间里，可以发生翻天覆地的变化。今天在这里， 重新和大家回顾微信客户端架构的演进过程，以及其背后的开发团队、流程的变化与思考。



拓荒

微信1.0 for Android的测试版本于2011年1月发布。这是微信Android客户端的第一个版本，软件架构采用早期标准的Android系统应用设计。

第一个版本是两个人用了一个多月的时间开发出来的，其中一个还是刚刚毕业没多久的实习生。这个时期团队一穷二白，资源有限、经验不够，主导思想是，复杂的事情尽量交出去做，保持最精简的客户端代码。得益于Android应用开发简单快速，从结构上看，这个时候其实还没有到需要特别设计的阶段，是最原始、简单的Android应用。

当然，再简单的软件也要考虑基本的设计思路。分层设计思想从这最早的版本开始引入一直到今天。回顾当时的设计，更像是MVP结合事件通知机制。从最上面由Activity组件组成的UI层（VIEW），往下到由NetScene组成的表现层（Presenter），再往下Network负责网络长短连接与数据库的通信与Storage组成的存储层。

NetScene是一个网络或者本地任务的基本单元，包括操作网络做数据收发、协议编解码，操作数据库做各种联系人、消息模块的读写。典型的例子如发送一条消息NetSceneSendMsg、做一次收信同步操作NetSceneSync。1.0版本的微信整个UI的activity可能不超过五个。

这个阶段，不需要做什么“减法”，我们的安装包也只有354k。

成长

微信的快速增长，从2.0版本开始第一次爆发。从语音版，到附近的人、漂流瓶，再到摇一摇。这个阶段的我们，似乎将全部的时间和精力都放在新功能拓展上。新的Activity、新的NetScene加上新的Storage，在看似成熟的框架里，一个功能就这样完成了。简单、快速、暴力。随之而来的，是一些之前完全没有想到会可能出现的问题，让最开始接触Android开发的我们措手不及。

早期版本因为经验的问题，产品上很多功能不去想也不敢去想，版本开发的时间跨度也比较长。随着开发经验的积累和对产品方向的理解，3.0之后的每一个小版本都处在一到两周的高速迭代过程中。

追求更好的用户体验，更多丰富的功能是产品经理们永远不会放弃的事情，尤其是在新功能为微信的新增用户带来了一次次爆发式的增长之后，更是无法控制。功能的试错频率大大加速，机型覆盖量上升后的兼容性问题也逐渐暴露。代码量、内存占用、安装包体积迅速膨胀。

而同样处于发展中的Android系统，也给我们埋下了很多坑，需要开发自己来实现、修复和优化。放在今天webview组件不再是什么问题，但在2.3之前的系统里面都会存在严重的内存泄露。内存问题为微信客户端架构的第一次进化埋下了伏笔。

变革

在微信1.0时代的时候，我们的关注点更偏向功能，随着用户增长，性能和稳定性问题逐渐浮上水面。2.0版本后，用户反馈中微信消息推送不及时的比例在上升，作为一款目标替代短信的即时通讯应用，无法及时收取别人发来的消息，这一点是非常致命的。

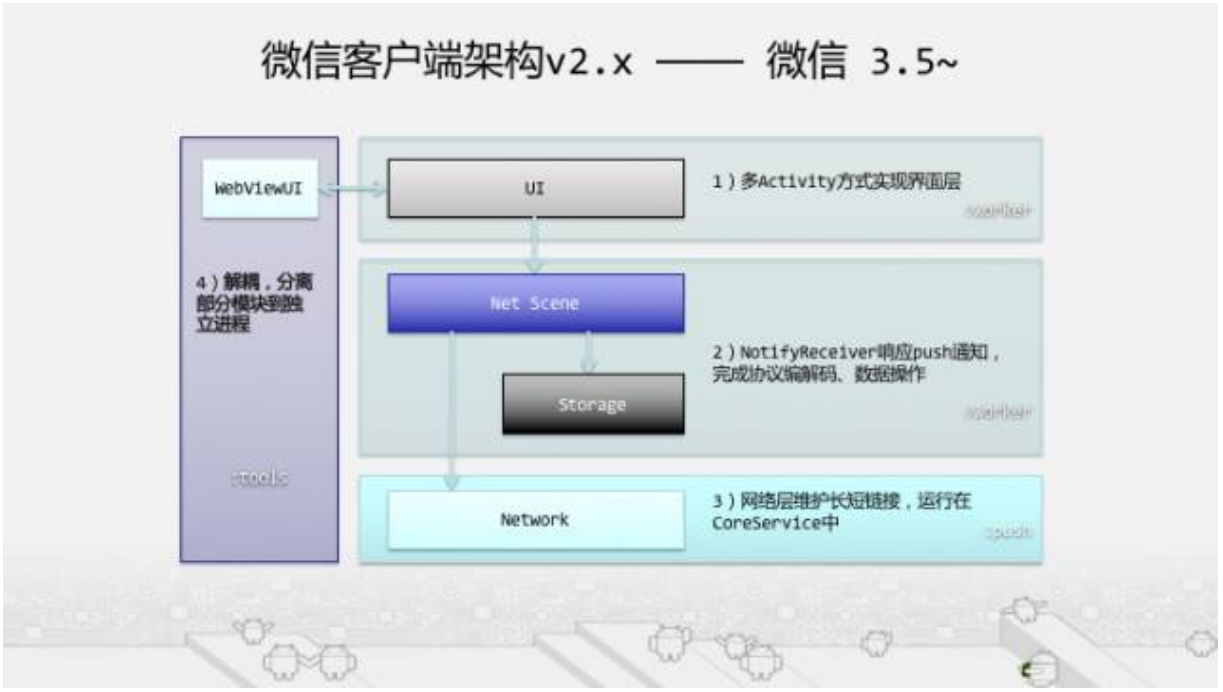
Android 1.5、1.6、2.1在当时是主要的版本，那个时候是没有今天的GCM的，甚至连C2DM也是2.2系统之后才会有的。而谷歌服务在国内被屏蔽，在相当长一段时间内，无论是C2DM还是GCM都无法正常进行推送。没有像APNS一样稳定的推送通道可供Android平台的应用使用，怎么办？自己实现。

国内网络的特殊性，使得我们再实现微信推送机制时，需要维持准确的心跳周期。如果一段时间没有活动，运营商便会将长连接断开以回收资源，这时服务器发消息给客户端就接收不到了。进一步研究发现，运营商网络的时间限制各个地区不同，有的地区有两分钟，有的地区有半个小时，这种情况是不可接受的。我们的解决方案是缩短心跳间隔，在网络运营商把客户端到接入点之间的连接断开之前，我再发送一次心跳，主动维持住这个长连接的活性。这个我们称之为长连接的保活。关于长连接的保活策略，微信也做过多次优化，这里另文介绍，不再赘述。

还记得前面说微信的膨胀吗？代码、内存、apk大小都在膨胀，这其中，内存对消息收发的影响很关键。Android运行时的择优置换机制，会选取占用资源最多的程序结束掉，除了微信自己功能膨胀导致内存占用加大之外，前面说的不省心的webview，还会给我们在内存问题上挖坑。而结果就是在用户手机运行APP比较多的时候，微信会被系统杀掉回收资源，消息收取不及时的问题就出来了。

如何解决？方法其实不少，微信选择的，是轻重分离的思路。通过在微信3.5版本

时候做的架构重构，实现了不受功能增长、系统缺陷影响的稳定推送方案。



对比v1.x版本的微信客户端架构图，我们将右下角Network的部分用轻重进程分离的思想，独立到一个单独的进程（:push）中，而上面两个层级依然跑在微信的主进程（:worker）中。而对于有内存泄露问题的webview或者其他不频繁使用的功能，再把其分离到独立的工具进程（:tools）中。通过分离进程，微信第一次重构解决了系统因为微信资源消耗，主动干掉微信服务的困境。

分离后的push进程内存占用以及被系统kill回收的几率大幅降低，而对于worker和tools进程，我们不再要求其一定存在，只在用户收到消息，或者进入h5相关功能界面时存在即可。这个版本的架构变更基本达成了我们设定的目标，无论是电量还是平均待机内存消耗上都大幅度下降，从内存上来看下降了70%，电量的话也比竞品和我们前一个版本有好转。

当然任何事物都有两面性。这一次架构的改变存在的问题逐渐在我们后面的开发当中暴露出来。比如进程每一次都要重新加载，里面所有的Cache、图片、界面全部要重新去执行一遍同样的代码，每一次加载内存都需要重新消耗时间。而启动速度变慢，则是最明显，用户最能感知的问题。“地球出现频率高了”是我们在这一时期经常听到的声音。而系统资源的消耗实际上比原来单进程的时候会更多，每一个进程都需要额外多占用一份虚拟机部分的内存。

这些缺点在3.5版本时代是可以接受的。从监测结果上看，启动速度变慢将微信的启动速度延长到了秒级，从原来的300-500毫秒到现在800-1000毫秒的级别。主要的图片缓存失效，则通过异步加载、解码、展示解决。拉长来看，微信的主进程资源会被自动回收，平均内存占用相比之前还是下降的。

即便在今天来回顾，依然可以看到，轻重进程拆分的思路是正确的选择。即便系统层面各种各样的bug逐渐减少，但应用的迭代使得功能一定不会减少。为了保

证图片、资源类在速度上的体验，内存的消耗也只会更大，是空间换时间的思路。而轻重分离，保证了核心服务在设备资源发生竞争时最大概率存活的同时，不造成对设备过多的资源占用。典型的场景就是用户开启游戏、视频录制通话等大型应用，作为常驻应用，不应该抢占额外的有限资源，需要做到“该放手的时候就放手”。

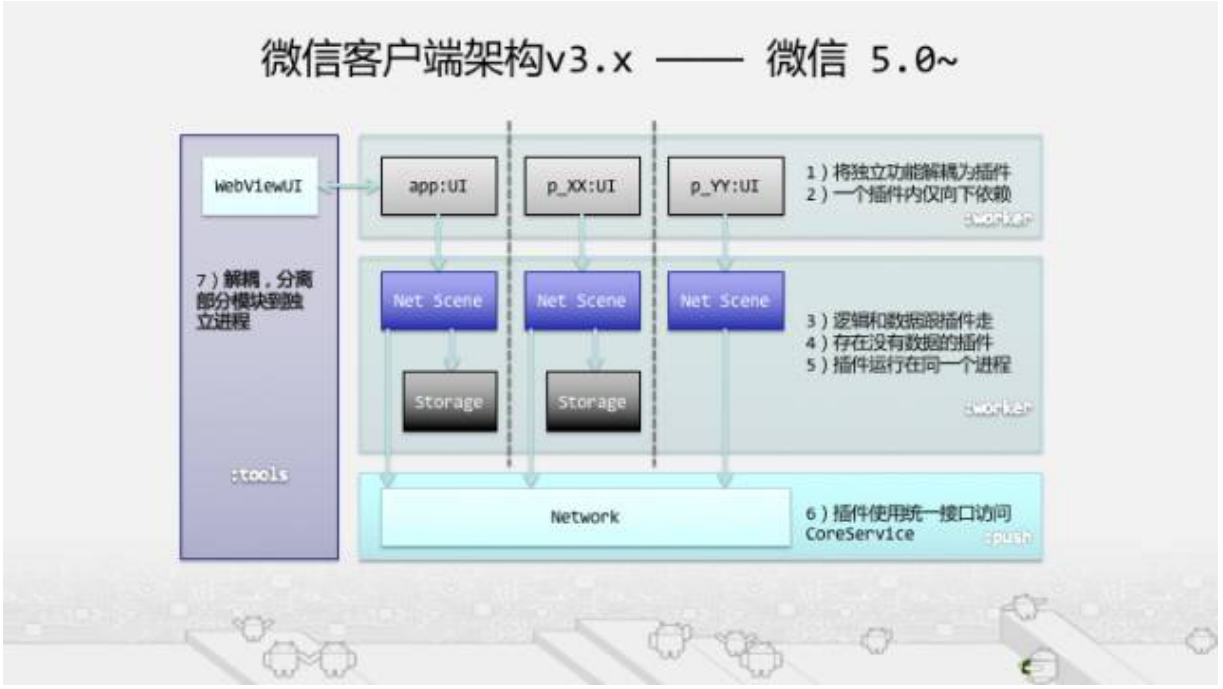
进化

很快，微信的架构演进进入了第三个阶段（v3.x）。微信4.5版本的开发过程中，出现了无法安装在一部分Android 2.3以下系统的机器上，当时2.3的市占率还在50%以上。试想一下，市场上一半用户的手机不能安装使用微信，这对我们是一个致命打击。放在今天看，2.3早已被淘汰，但在当时，我们不可能停下来等待，无论从开发和产品来说都是不可接受的。技术分析的直接原因就是，微信的发展速度太快，触发到Android虚拟机机制的设计缺陷。

两个问题，一是单dex 65535方法数限制，二是线性内存分配器（LinearAlloc）限制。今天的Android开发者看到这两个限制都不会陌生。前者是因为Android的早期设计中，对dex文件中方法id用16位整型标记，单个dex文件中的方法数无法超过65535，eclipse环境中生成不了未做过proguard的debug apk。后者则是dalvik虚拟机用来加载类的堆内存大小被硬编码了，2.3以下是5M，2.3以上是8M，微信无法安装的原因就是因为这个堆内存被耗尽导致dexopt失败。

今天来看，Google已经给出了一些可靠的解决方案，辅以更加先进的gradle + Android Studio，开发者们可能根本不会再遇到这两个经典问题，。官方的MultiDex分dex机制解决了方法数限制的问题，其中main dex最小化原则，结合dalvik LinearAlloc heap size调整（修改到了16M），使得dexopt的失败几率大幅下降。而art的出现彻底不再存在LinearAlloc这样的限制。回过来再看，那个时代里微信是如何通过软件架构调整解决这些问题的。

微信在高速发展过程当中，到5.0的时候已经有很多功能，而其中一些功能，随着用户群体、产品设计等因素变化，用户使用的频率在改变。之前试错的一些功能，也大量存留在微信版本中。这些不常使用的功能不应该始终占用程序资源，从架构上进行纵向分离，保证主要场景的体验，是这一时期的主要设计思路。



要做的第一步就是解耦。微信这类社交型应用，在用户数据、关系、消息等结构上存在着各种各样复杂的依赖，这些依赖相比工具型的软件来说，调用频率更高，性能要求也更高。想做到完美的拆分不是一件容易的事，但是不完美的拆分却是可以达成的。

轻重分离的思想再一次被应用，这一次是在代码模块的使用和组织上。保证主app功能的快速和稳定，将附属的新功能分离在独立的插件工程（p_XX）中，每个插件有独立的UI界面逻辑和资源、存储及网络协议编解码处理逻辑，通过共用统一的基础库接口访问网络服务。

将微信功能解耦为插件，一个插件内仅向下依赖。插件最后编译出来会是一个jar包，其内包括的实际内容是对应的dex。这里需要注意的是，插件并不需要有独立的进程空间，而是根据该插件实际的场景决定其运行的实际进程，绝大多数情况下，插件是和主app功能共享多进程载体的。

v3.x架构的改造工作量对当时的我们来说很大，从最开始4.3版本发现dex limit和LinearAlloc limit到5.0版本成型做第一次的验证，我们花了8个月时间，解耦出来的工程项目有60个以上。4.5版本将附近的人分离出去是作为一次试验，为5.0这一大版本填完了坑。5.0版本是微信历史上非常重要的一环，从这个版本开始引入了游戏、支付和更加完善的公众账号体系。

这种设计思路不是微信首创，现在回顾也并不复杂。如果你的产品历史功能不多，迭代不是很快，可以全部人停下来1到2个月集中一次重构搞定。但对于微信来说，但这一版的架构变更，更像是在给天上飞着的飞机换发动机，由一支10来个人组成的团队完成。互联网的快速、敏捷给处在“创业”阶段的我们新的挑战。如何做到呢？要保证不给处于高度需求压力下的开发人员增加架构变动的额外负担，首先要做的就是不要让他们重复修改代码，无缝迁移到新的架构。

一、创建必要的工具和规范。微信在4.3发现问题之前，一直坚持着非常好的开发效率优化思想，代码自动生成起到了很大的帮助。团队内部使用的自研的代码生成工具autogen，通过简单的xml定义，即可生成所需要的存储、协议编解码、事件机制代码。这使得我们具备了比较轻松解耦的前提。

二、新的架构要求开发者在做新功能时，使用独立插件子工程，好的工程模板可以事半功倍。早期传承下来的分层设计，也使得开发人员在前后两种开发模式下的学习成本降到最低。对应的编译和开发调试工具。

三、对于历史实现的功能特性，尽量通过反射等一些技巧，来保证不需要大规模重写代码，“先抗住，再优化”。不要一开始就追求完美，先活下来。直到5.1、5.2版本，我们才基本上全部完成这一次程序架构调整。

四、人。架构调整是必须要做的事，但是作为发起者，也不能只从理论角度去强硬推动。减少开发者的工作量，而不是增加，站在开发者的角度想问题，往往会得到非常积极的响应。

不同的客户端架构时期，背后的团队和开发模式也会有所不同。对比三个版本的客户端架构，v1.x和v2.x的时候比较适合小型团队、没有特别复杂或者反复需求的客户端进行快速开发。单trunk主线开发即可满足。每次发布后，拉出来一个对应的release branch，如果发现有bug或者小优化需要修改，直接在这个release branch上修改、测试、发布上线。这一时期release branch通常保持在两个以内，当前版本和前一版本。当前版本是为了线上问题的快速发布，而前版本则是为了修复一些厂商渠道预装的问题。

v3.x就比较适合中到大型团队，解耦之后，可以支持多个团队的并行开发，也可以满足多个版本的同时开发和发布。微信的产品经理和客户端开发人员的比例大概是1.5+，也就是说产品经理会比开发人员多50%以上。开发人员会面对产品各

式各样的需求需要实现，很多需求处于原型或者是设计阶段，而有些即便开发完成，也需要和老大体验修改。

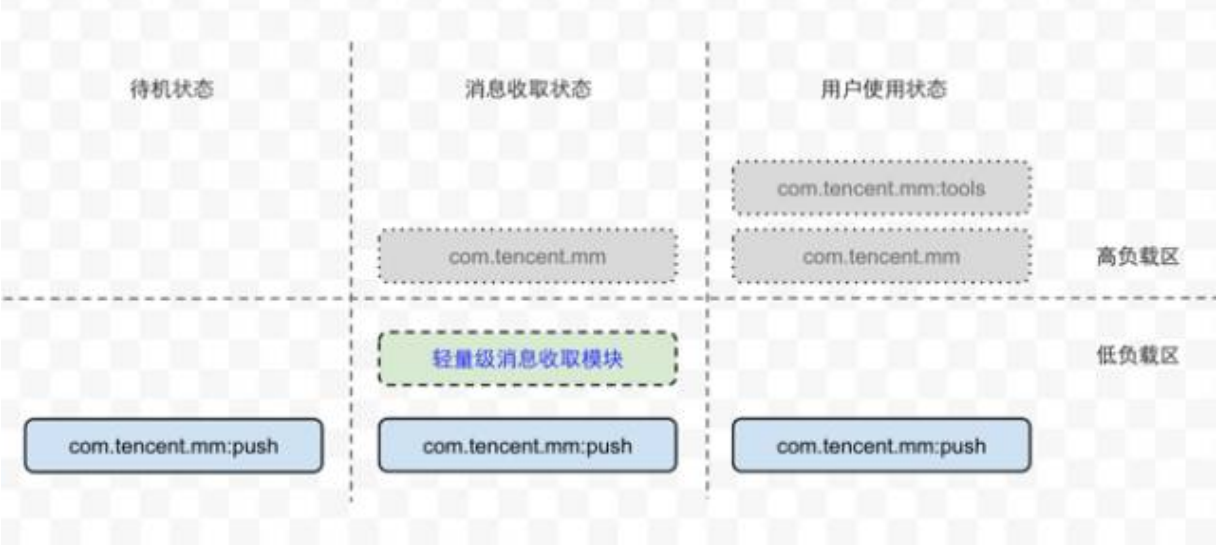
这种不稳定状态的需求，往往是今天说最终确定，但2个小时后就可能要修改，甚至临上线前都可能说这个功能被砍掉，不要发布了。不稳定的需求会带来不稳定的代码实现，质量也没法控制。通过多迭代多版本多分支并行开发，只有所有人体验过最终确认没问题可以上线，才会合入发布版本上线。

这个过程像什么？像开源项目的开发。

开放

进入到2015年后，微信在软件架构上逐渐趋于平稳。在v3.x原有插件加载基础上，研究了更多行业内Android应用的技术架构。结合官方MultiDex的实现，增加动态热补丁功能，通过终端的运营系统，实现了微信客户端补丁版本更新48小时90%+覆盖率。编译系统也从buck+修改为微信自研的builder构建，支持LinearAlloc和methods/fields count的实时计算，以及融合了MultiDex与微信插件模式的dex自动分包。

在v2.x架构轻重分离的多进程思路基础上，进一步优化实现了push的在收信条件下的“lightpush”运行模式。在仅消耗push进程低内存的条件下，实时收取新消息通知，避免对进行中的游戏进行资源抢占的同时，又可以及时收取消息。



更重要的是，我们开始将目光转移到开源的开发模式上。v3.x的并行开发模式，在svn下已不再适应。2015年上半年开始微信Android客户端团队开始转向git，充分发挥git在多团队并行开发下的优势。内部也放弃了沿用许久的ant + eclipse，全面转向gradle + Android Studio的分布式构建思想。通过内部开源，微信内的公共组件已经可以通过maven在不同的开发团队中共享并随时使用。或许某一天，我们也可以在github上看到”WeChat Mobile Dev”的身影。

（全文完）

Android 微信 腾讯 移动开发

相关热门文章

配置你的 Android Studio

Android 学习第六弹之 Touch 事件的处理

Android 去除烦人的闪退 Dialog

fir.im Weekly - 论个人技术影响力是如何炼成的