京东移动技术能力提升之路

作者：谭丁强、刘卫欢、姚醒、王孝满、陈吉、赵云霄、何福永、范超

我们相信，移动技术能力是一个体系，是一个整体，任何方面的短板都会造成系统服务能力的降低，造成转化率的降低，进而造成真正的损失。无线技术部通过整体提升、改造、有机互补，实现了移动整体技术能力的提升，支撑移动端用户量、转化率、订单占比的持续提升。

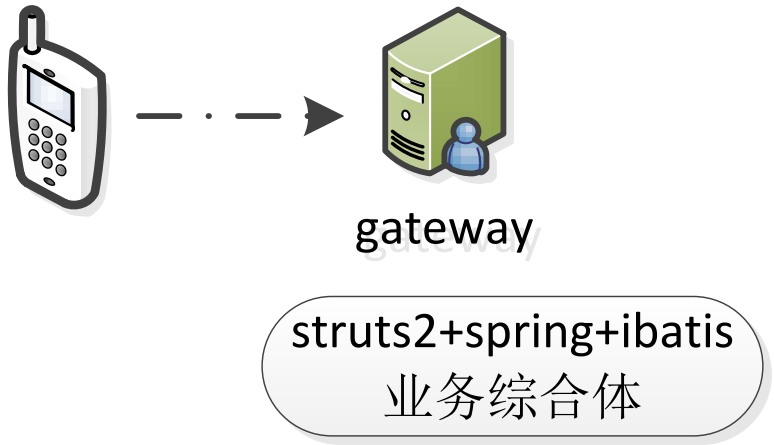
本文提炼了几个实践过程中总结的移动技术能力关键点，分别阐述这些关键点的突破之路。

一、 移动网关架构变迁

随着移动业务的兴起，京东移动网关的架构，经历了几个阶段的演变。

1. 从 0 到 1

第一阶段，服务端为响应业务的需求， 诞生了一个单一的服务端应用-网关，纯粹为了支持业务需求。 从分类、搜索、商品详情、购物车到下单， 一整套业务流程都在该应用上，以便最大程度的满足新增业务的发布和修改。这个阶段， 服务端采用的是 MVC 式的结构。



目标：快速搭建京东 app 后端业务系统，解决从无到有的问题。

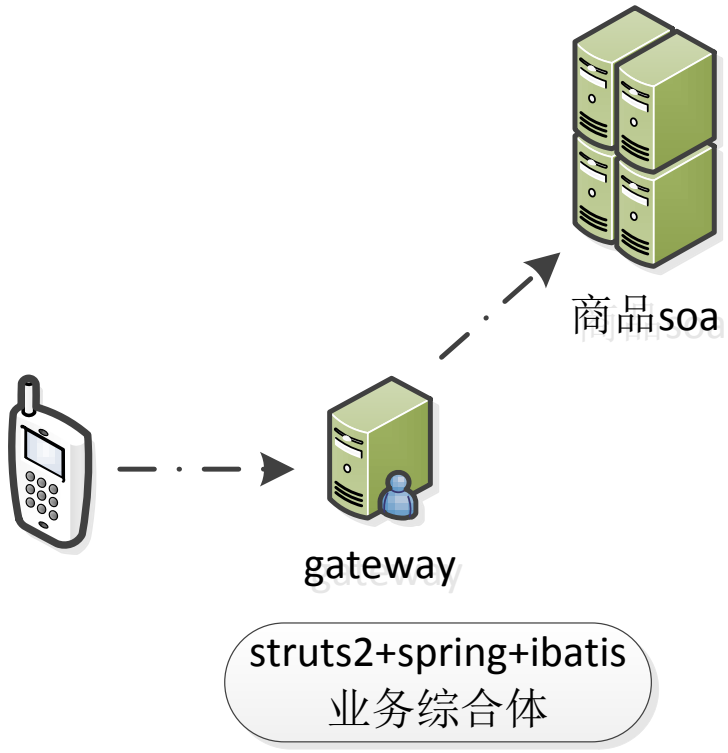
技术关键词：ssi

遗留问题：随着业务的拓展，一个系统的模式已经不再适用，不同业务团队的开发会相互影响。某个业务的宕机也会影响整个 app 后台的可用性。

2. SOA 小试牛刀

第二阶段，DAU 的快速增长以及移动业务的膨胀式发展， 尤其是双 11 和 618 两个大促，任何一个业务高并发量的访问， 都可能拖跨整个应用。 根据对业务数据的统计分析，搜索和商品详请两个业务访问量最大， 因此，在 2013 年的 618 大促来临之前， 我们开启

了第一轮的架构升级， 我们将搜索和商品详情的业务拆分出来， 成为第一个单独的 SOA 应用-商品 SOA。关于搜索和商品详情的用户请求，将从网关上转发至商品 SOA 进行处理，事实证明， 在当年的 618 大促中， 服务端平稳度过。



目标：将访问量较大的业务独立，将整个 app 后端初步分层，初步达到不同业务系统相互不影响。

技术关键词：servlet，后端系统性能至上，没有用户界面，故而抛弃 struts,采用简单的 servlet 作为 http 入口。

遗留问题：过渡产品，拆分不够，还有一些系统遗留在原来的网关中，老的网关又有业务又有转发功能，不同业务系统的划分不够合理。

3. 彻底剥离业务

第三阶段，随便 DAU 进一步增长， 移动业务需求越来越多， 业务都开发在网关上，引出的问题日益彰显，不论是开发的效率还是服务的性能质量， 都面临着巨大的挑战。 因此，在第二阶段的成功经验上， 展开了第二次的架构升级。以业务线条进行划分， 诞生诸多的 SOA 系统， 经由网关统一转发。

目标：将所有业务从网关中剥离，后端业务系统彻底下沉 soa 化，网关负责转发。

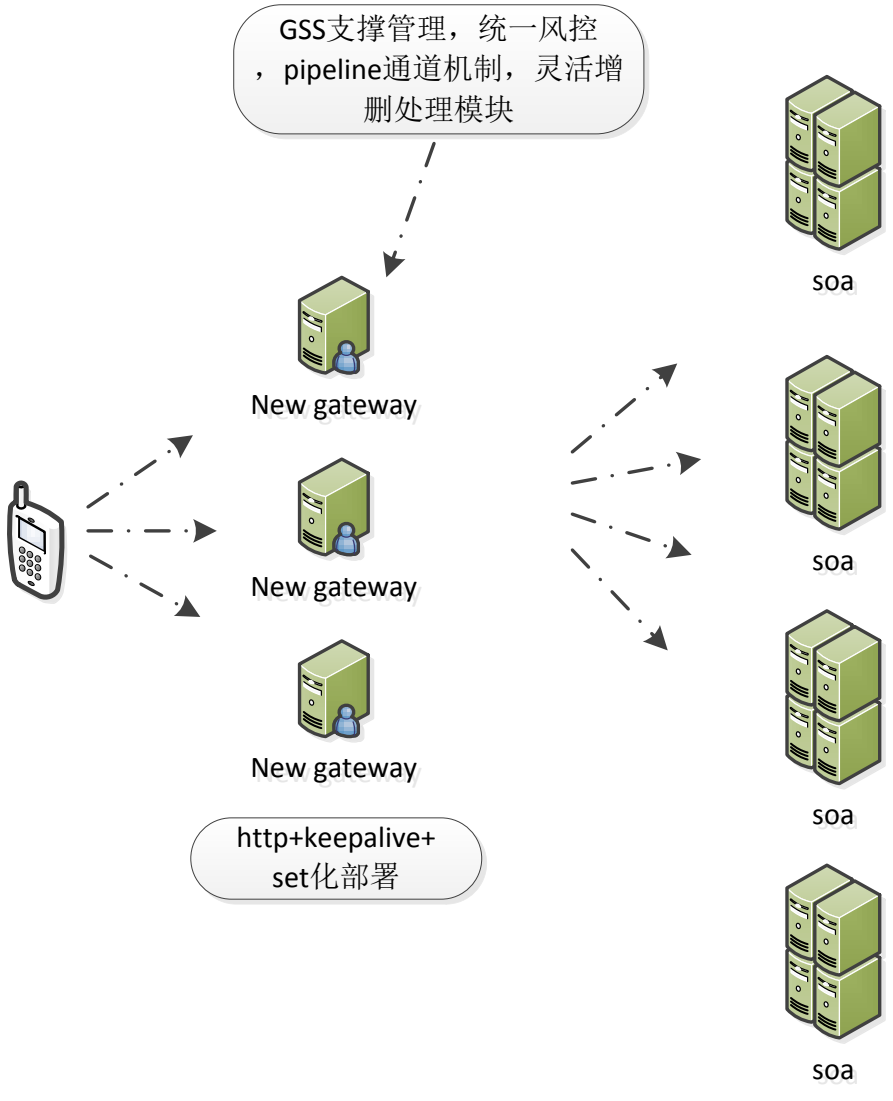
遗留问题：老网关实际上是最初的移动后端系统，很多业务代码冗余其中，虽然业务系统都拆分完成，但由于时间和人力的问题，相对独立的风控和登录依然在老的网关中。同时，网关作为移动流量的入口，没有一个独立功能强大的管理系统，网关功能单一。

4. 功能升级

第四阶段，用户正式流量、恶意流量越来越大， 越来越的业务要接入。当前的网关，性能、扩展性等指标都极差，而且对后端业务系统的数据格式限制极大。 我们想做流控、风控、业务容灾， 抱歉，除了纯转发， 它什么也做不了。因此， 我们开始考虑， 什么才是一个像样的网关，我们又应该如何去做。

针对这些问题，2014 年 618 过后， 我们启动了网关架构的升级， 并于双 11 前投入生产， 称为新网关。新网关只是针对各 SOA 业务系统的一个通道， 只要机器的内存网卡够用， 则不存在调用上限和性能问题， 主要有以下特点：

1. set 化部署， 更好地容灾。
2. 与 SOA 系统去耦合， 各 SOA 系统可灵活定义自身数据格式。
3. 采用模块化设计、pipeline 机制，分流、限流、转发、数据统计等模块灵活添加与删除。
4. 统一风控接入， 各 SOA 系统可专注于业务的实现。
5. 支持长、短连接两种模式与后端 SOA 交互。
6. 配套的管理支撑系统 GSS(gateway support system)。



京东 app 自诞生起，它的服务后端架构就在不断的改进中，这些改进都是我们在不断遇到问题不断思考解决的过程中总结出来的，任何的架构升级都是来源于实际的需求，没有一蹴而就的“好架构”，只有不断改进不断适应不断调整，才能让我们的系统保持在一个较高的水平。

二、 移动客户端焦点问题改进

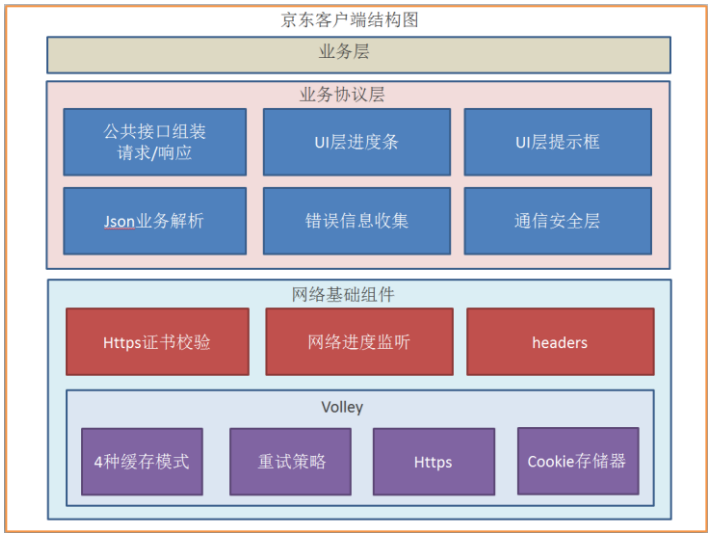
随着移动客户端的装机量突破 2 亿，日活突破千万，内嵌功能上百个，移动客户端也遇到了各种问题，其中突出的问题有网络收发问题、图片显示问题、内存溢出问题、崩溃率问题，以下分别从几个方面说明一下解决方法。

1. 网络基础组件

为了让诸多功能共用网络资源，创建了京东客户端网络层，单独提取组件，与协议层完全分离。

京东网络组件（基于 Volley 框架二次开发 ）除了具有原有框架的多个优点外，还增如下功能：

1) 4 种缓存模式：只用网络、只用缓存、先用缓存再用网络、缓存网络共用。功能满足客户端多种缓存需要，提高客户端响应速度，同时减少与服务端的数据请求频率。并提供缓存规则检查接口，方便具有个性化的接口数据不保存缓存。
2) 重试策略：使 wifi 环境与 2g/3g 环境重试时间分开，将用户等待时间降到最低。具有 gzip 降级重试逻辑，保证稳定压缩数据，提交响应速度。
3) 客户端与服务端的会话保持：多数采用 Cookie 通信，所以框架中加入了 Cookie 存储器，方便网络通道传输用户会话数据。
4) Header 参数：有些特殊接口需要向服务端传输 header 信息，我们加入 Header 参数，用于传输自定义头信息。
5) https：支持 https 提高网络通信安全性。并加入证书校验，防止中间人攻击。
6) 网络响应监听：为了能及时监听到网络真实请求，我们在网络响应监听中加入 onStart 方法。



2. 图片基础组件

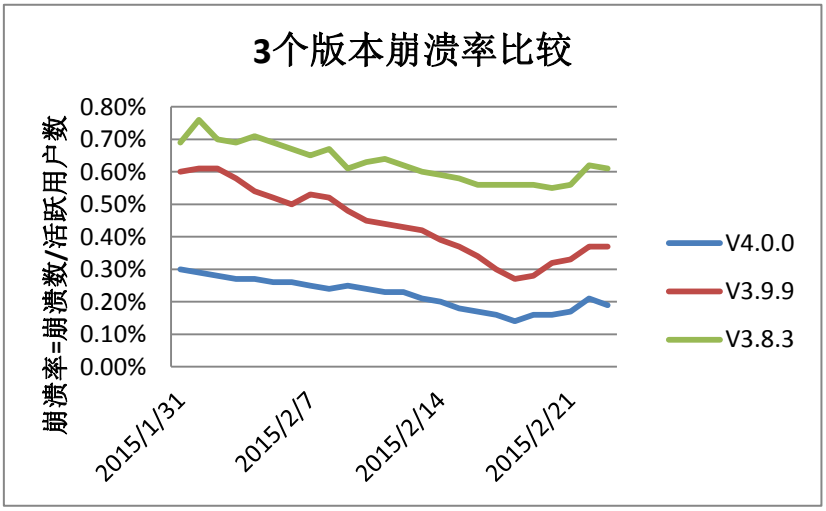
京东客户端内的图片量非常大，对客户端的图片展示非常有挑战性。低内存溢出、UI界面流畅，是图片处理的最佳指标。我们基于 Universal-Image-Loader 框架二次开发

1) 将图片加载业务跟主客户端的逻辑完全解耦。
2) 加入图片缓存的初始化统一入口，防止程序多套缓存问题，方便控制。
3) 通过图片任务管理器管理图片加载顺序，控制同一时间图片加载的数量。
4) 随时取消无效的加载任务等。
5) 解决 AbsListView 快速滚动过程中图片加载影响滚动的问题。

3. 崩溃问题处理方法

android 系统有碎片化严重，手机机型多、适配难度大的特点。在这样的背景下，即使经过严格测试，发布的 android 客户端依旧有崩溃发生。崩溃现象严重影响用户体现，所以降低崩溃率成为开发团队的重要任务，也做为考验产品质量的重要指标。

Android 版本经过不断努力，将崩溃率由 0.8%降低到 0.2%左右。



从 3.8.3 到 4.0.0，通过逐步修改崩溃问题，崩溃率降低明显。

1) 崩溃数据收集

我们采用 Thread.setDefaultUncaughtExceptionHandler 注册异常堆栈监听，将崩溃异常堆栈、出错页面信息、出错前页面路径、意见反馈信息提交服务器。

2) 数据分析

服务端将数据收集后对异常堆栈过滤，提取出如下重要信息：

提取字段	举例说明
代码出错定位	com.jingdong.app.mall.basic.JDFragment.onStart(87) 包名、类名和行数
异常名称	java.lang.NullPointerException 空指针错误
崩溃时停留页面	com.jingdong.app.mall.product.ProductDetailInfoActivity 商品页面

3) 问题修复

根据系统设定参数，可迅速精确定位问题的所属模块、开发负责人、错误行数和错误类型。对数据分析得到每个模块崩溃问题数量和占比，及时通知到开发人员，准确修复崩溃问题。

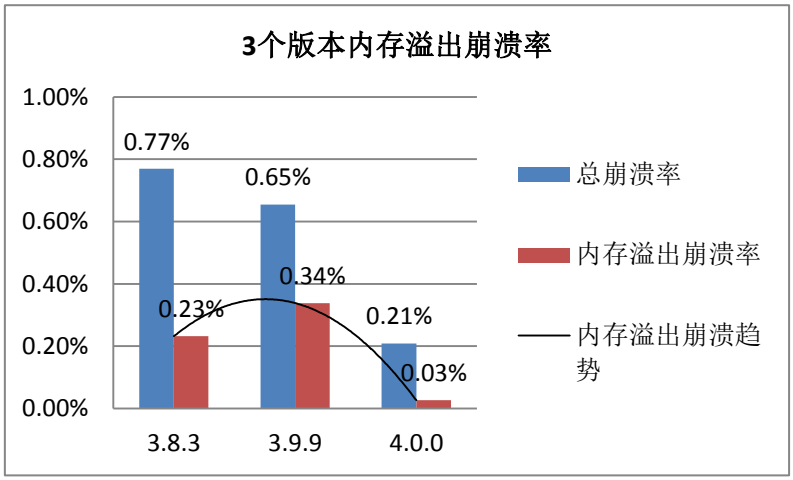
4) 灰度发布应用

灰度发布与崩溃问题修复是相互结合，持续一星期将崩溃率降到最低，再正式上官网。

采用如下流程：5%灰度用户发布->1 到 2 天崩溃问题分析与修复->新的 10%用户发布->1 到 2 天崩溃问题分析与修复。

4. 内存问题处理方法

分析崩溃数据发现一半为内存溢出问所导致,所以在4.0.0版本重点对内存溢出做了修复。



对内存溢出数据做充分分析，并对症下药将问题解决：

1) 界面循环打开，如从商详页进搜索页，再进商详页，因页面未释放导致内内占用较大,一半以上内存溢出都是这个原因导致。针对该问题,客户端开发页面堆栈控制，控制客户端打开页面总数为 10 个并能够根据机型动态下发，有循环嵌套的页面控制最大打开实例数，很好地降低了该问题发生的概率。
2) 界面打开占用内存过大，该类问题多见于引导图过大并未释放，采用 ViewStub 按需加载图片修复；activity 层级太深，通过修改设计来修复问题。
3) 修复未释放内存的页面，有些页即使关闭，也未能释放掉，使用 MemoryAnalyzer 工具，找到内存占用的对象和引用位置，能定位到问题根源。解决比较多的模块有首页、商品列表页、商品详情页等。

三、 移动首屏加载速度提升

1. 模块化

将首页所有数据合并成一个大的接口，加载网络数据，渲染时分为各个独立楼层模块，动态的按照滑动显示区域加载。

2. 内存优化

图片内存优化，当楼层可见的时候才会去加载楼层里的图片，第一次先加载图片缓存内存以及硬盘。缓存硬盘 App 关闭后不需要重新下载图片；缓存内存并动态按照显示区域去加载图片到内存可以节省内存；限制图片最多有 N 张在内存中，内存总量不允许太大，当有内存告警的时候，立即清理图片内存。

楼层内存优化，楼层当滑动到屏幕以外，不可见区域，则立刻回收对应楼层的内存，楼层基础内存是复用的。

3. 加载流程优化

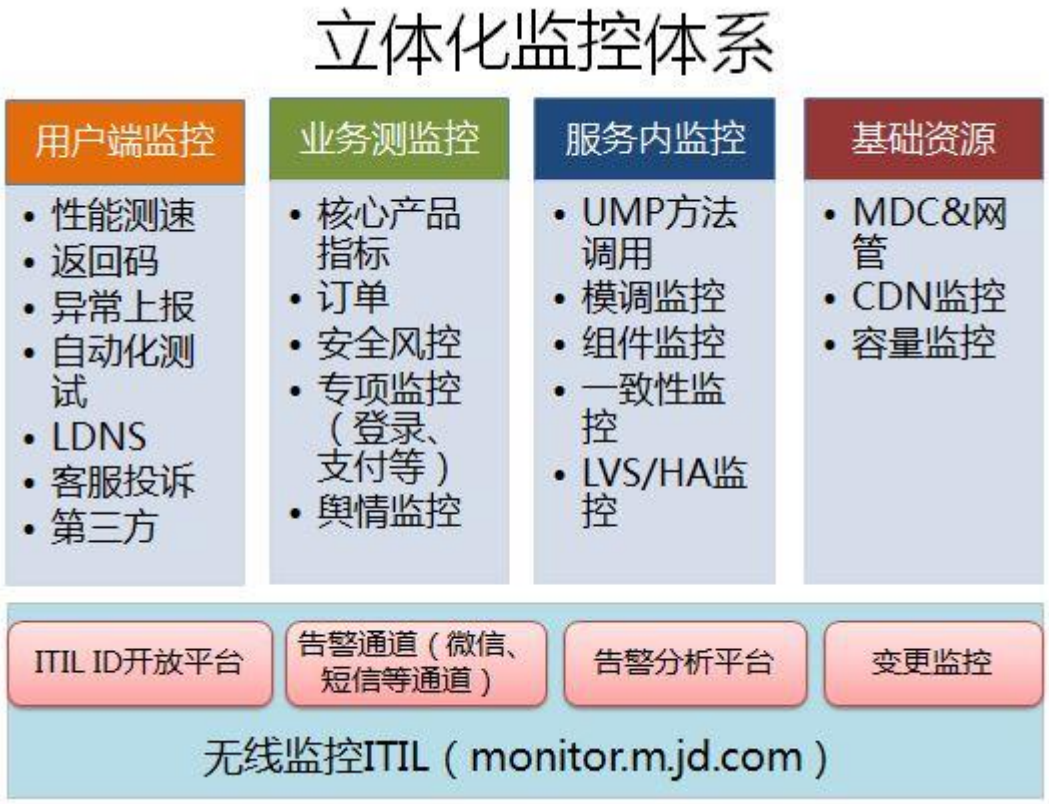
在进入程序的时候，先预加载获取首页的接口数据，比如在启动广告图，或者引导图界面即完成了首页数据的下载，这样在首页绘制之前就直接获取了数据，减少界面空白等待数据的时间。

4. 渲染效率优化

首页各个楼层都对应有一个数据模型，我们在刷新首页的时候，如果上一次缓存的某个楼层的数据模型与下一次新下载的数据模型完全一致，则不需要重复渲染，节省了渲染浪费的资源，减少了楼层渲染次数。

四、 移动立体监控

随着公司业务体量的快速增长，对于移动监控也提出更高的要求；监控也不单单要做到发现问题，需要具备多维度细分和高效定位的能力。一方面团队结合现有监控资源做深度定制，同时也借鉴了腾讯海量监控的方法论与多部门开展深度合作。



1. 用户端监控

以用户体验维度的监控视角出发，收集用户真实数据。性能测速覆盖用户核心路径页面，可按区域、运营商网络等多维度查询；异常上报采集了客户端上报的异常信息，为快速定向排查问题提供了数据支撑；自动化测试轮询测试用例，模拟外网用户行为；返回码监控则是利用自研采集器收集用户请求的 HTTP 返回码。

2. 业务指标监控：

订单、登录、注册、SOA 访问量等方面的实时趋势；安全攻击风控防刷实时数据。

3. 服务内监控：

各服务组件及服务间调用的监控方式。模调以服务--方法--服务为主线，采集串联主被调的调用量、耗时、成功失败率等数据，可对服务运行质量进行实时监控； 各类业务组件及基础组件监控。

4. 基础资源：

结合集团和自研解决方案，提供网络和服务器的基础运行数据，按照业务模块构建的容

量监控方式。

5. ITIL：

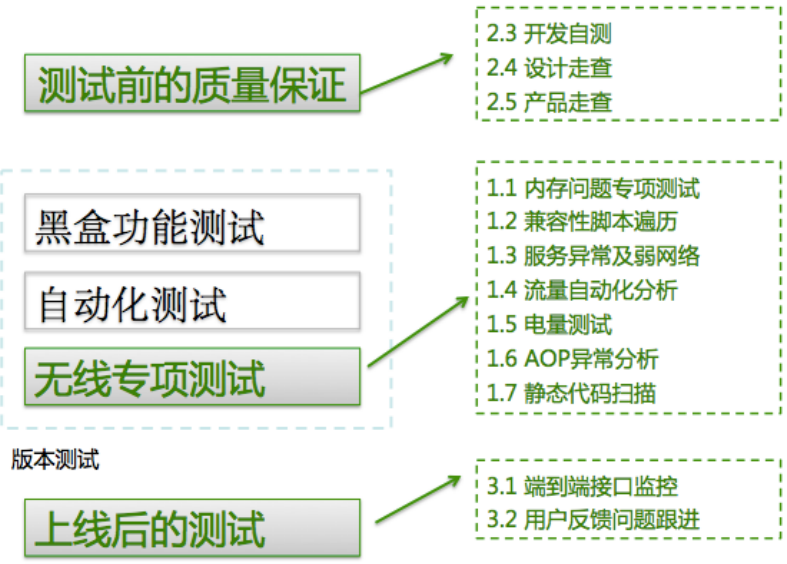
监控 ITIL 提供多维配置管理、数据管控与加工、告警通道管理、监控开放平台、智能分析、定制化视图策略等等能力。便于不同角色不同关注方向的用户汇总监控数据和有效管理。流程上也设计与其他平台互联打通。

以上监控体系规划，结合了集团现有的资源以及基于业务特性研发的自研。也是我们无线团队近半年的一些积累与尝试，也还在持续完善中。

五、 移动测试新思路

通常谈起测试，大家第一时间想到的是功能测试，对于手机 App 测试，大家想到的场景估计就是测试人员桌上多种型号的手机，然后不停的手工测试各种功能。

刚开始开展移动测试的时候，也许是这样的场景，不过经过不断的技术积累和探索，无线事业部有了很多已经在落地执行的新思路，可以用下面的一张图来概况。



1. 不只是功能测试和自动化测试

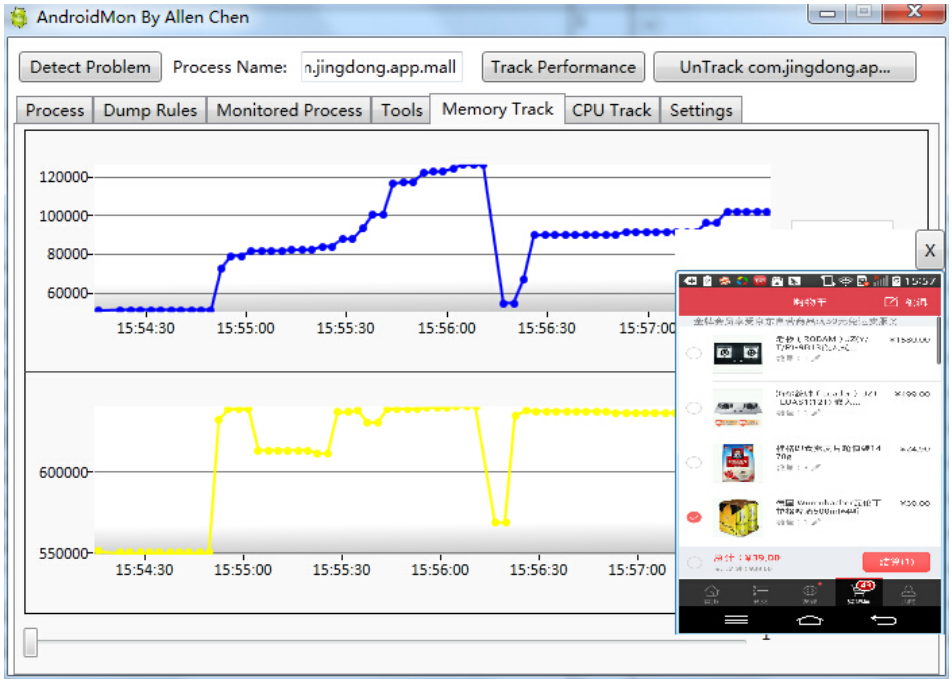
手工的功能测试之外，大家最容易想到的是自动化，我们开展了基于 App UI 的自动化测试，基于 Appium 开发了一套完善的自动化测试框架并编写了用例用于部分功能的回归。接口方面，我们以 JMeter 为基础编写了大量的接口自动化用例，用于回归后台接口逻辑。不过这里我们要讨论的不是自动化的开展，而是针对 App 各个方面的专项测试的开展。包括代码静态扫描、兼容性测试、内存专项测试、流量测试、电量测试、稳定性测试、代码覆盖率分析和针对异常的 AOP 测试等方面。这些专项测试对于提升 App 的性能、稳定性，以及减少流量电量的消耗都非常的有帮助。

限于篇幅，这里介绍其中两种。

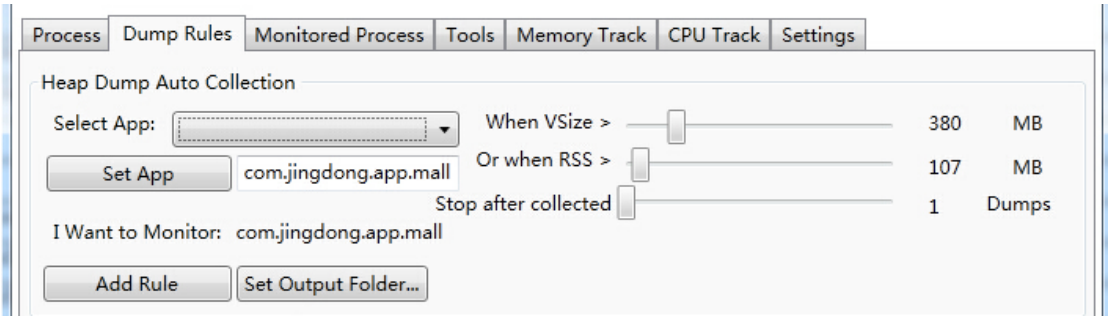
1) 安卓内存专项测试

内存问题对 App 来说非常关键，一方面影响性能，另一方面有非常多的 crash 问题分析下来是内存占用过高导致的。针对内存问题我们开展了一些专项的测试，这里介绍下安卓方面的做法。

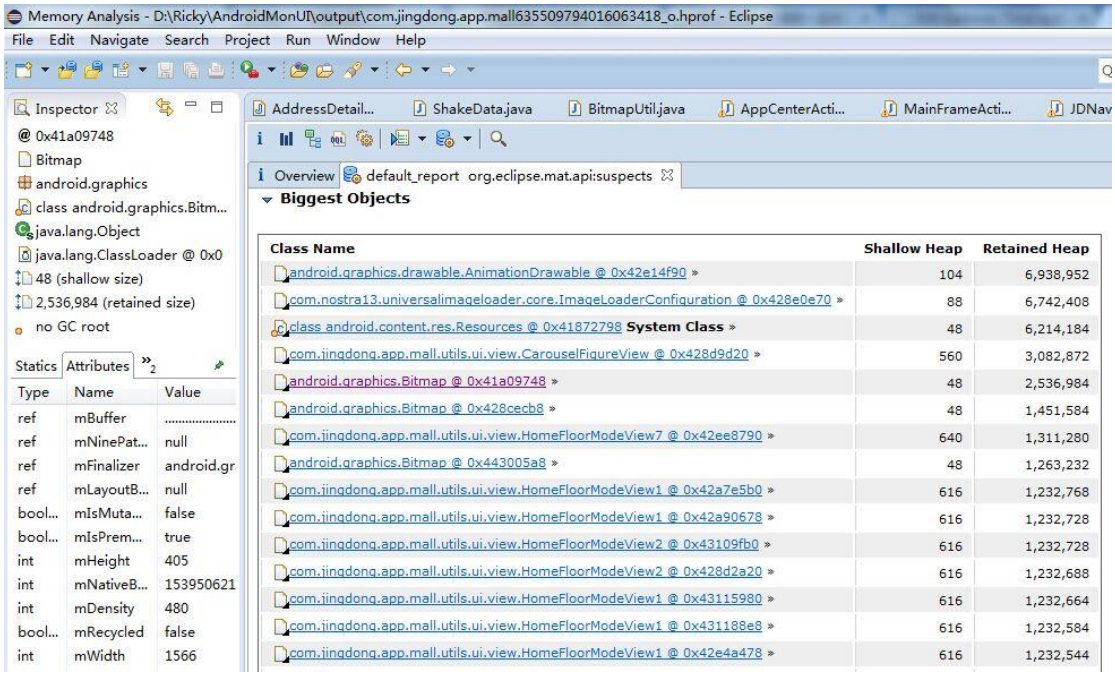
首先我们开发了一个 PC 端的工具，手机通过 USB 连接到 PC 上，就可以实时记录我们 App 的内存消耗，并有对应的截图功能，可以知道内存消耗较高时在做哪些操作。



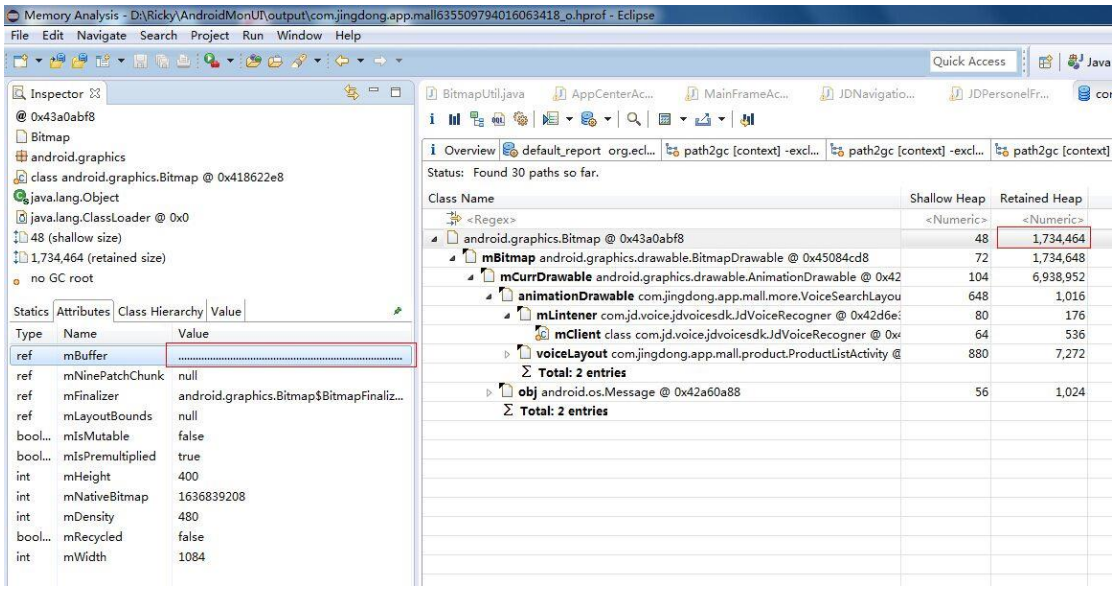
另外，还有一个根据设定的阀值自动抓取内存 dump 的功能。



接下来将 Dump 出来的内存文件导入 MAT 工具进行分析。找出内存消耗比较大的对象。



以下是对于其中一个消耗较大的对象做进一步的分析。电商的 App 的特点，很多内存的消耗是由于图片使用带来的。



将内存中的二进制数据，对应上图中的 mBuffer 字段，存入到 rgba 文件中。然后使用转换工具将对应的内存文件转化成可读的 PNG 格式，宽和高的参数来自前面图上的结果。

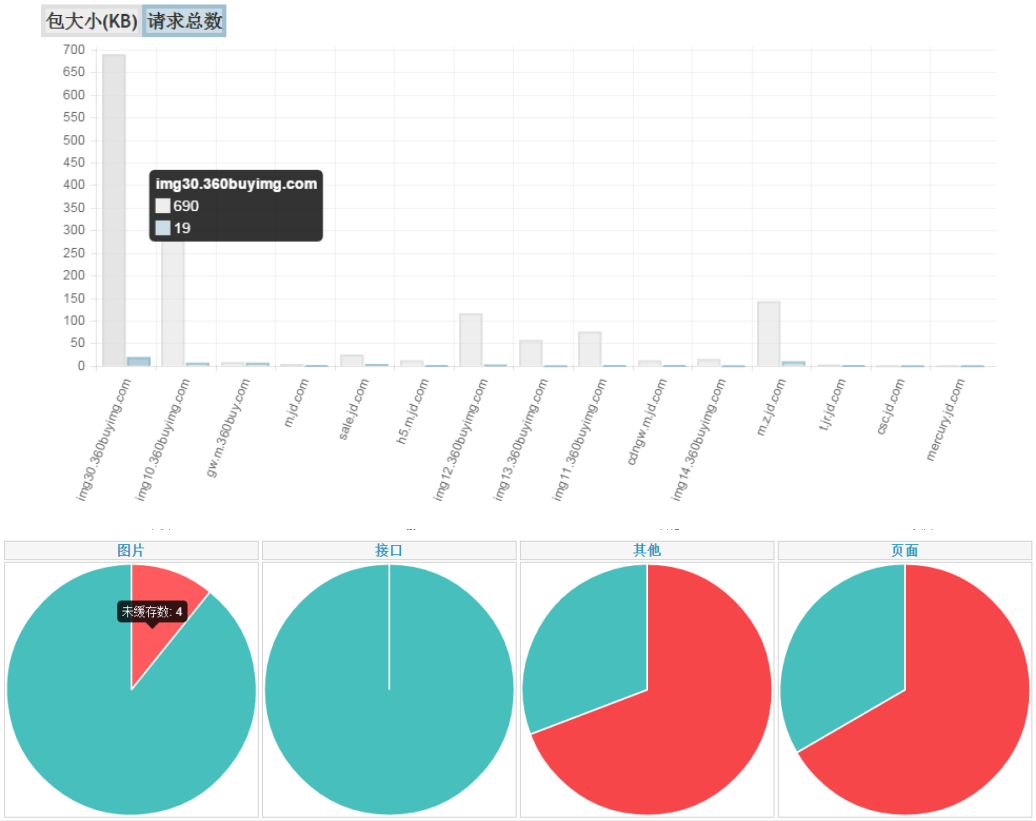
```
C:\Program Files\ImageMagick-6.8.8-Q16>convert -size 1084x400 -depth 8 D:\Ricky\
MAT\2.rgba D:\Ricky\MAT\2.png
```

这个图的例子，转换完之后是一个下面的类似波形的文件。

通过对比 App 中的素材以及和开发确认，了解到这个是语音搜索对应的波形图，用于显示当前的操作。但是有两点不合理，一是图片过大，而是在用户没有使用语音搜索的时候也会常驻内存。后面做了针对性的优化，减少了这部分内存的消耗。

2) 流量测试及自动化方法

流量是 App 的用户比较关注的部分，为了避免这方面给用户带来困扰，测试团队会进行一些流量相关的专项测试。技术方案上可以通过代码植入或者抓包分析的方法。不过无论哪种方法，都会有大量的手工操作和分析的过程。我们基于 Fiddler 工具的核心 FiddlerCore 做了一些二次开发，完成了一个流量的自动化分析工具。可以按照很多预先设定的维度来对 App 消耗的流量进行分析，比如按域名，按 content-type 等，也可以分析请求数和缓存的情况。以下是报告的样例。



使用上述工具后，在测试中，我们只要在测试开始和结束时通知我们的统计工具即可，最后的报告会在几秒内自动生成。让这个流量分析过程的效率大大提升。

2. 质量管理，而不只是测试

如果我们的目标是提高质量，那么除了上面提到的黑盒功能测试，自动化测试，以及针对 App 的各种专项测试等技术方法，还可以借助质量流程的方法来提升。除了常规的需求评审和测试用例评审之外，我们还采用了几个非常有效的质量提升的做法，分别是开发自测、设计走查和产品走查。

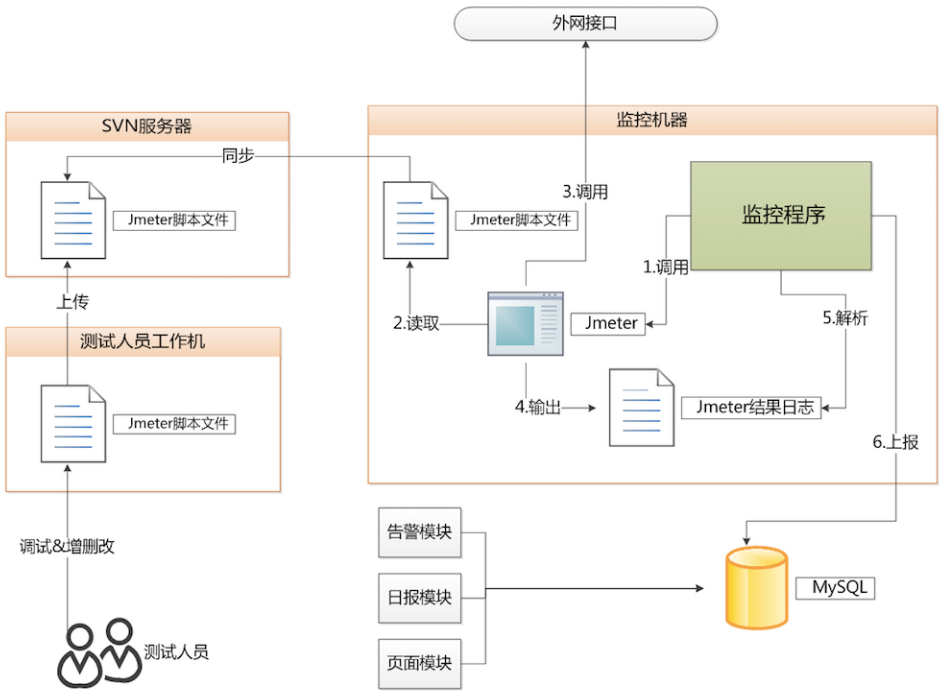
对于开发自测，测试人员会在开发提测前挑选部分测试用例给开发用于自测，开发在提测邮件中附上自测用例的结果，测试在执行用例的过程中也会关注是否有一些缺陷在开发自测用例中。

产品和设计的走查，是在开发提测之后立即进行的，这样产品和设计的同事可以快速的知道做出来的产品是否符合预期的要求。如果有问题，可以马上提出来，进行改进，而不是像传统流程等到快要上线的时候才发现问题。另外，也有一些产品体验和设计方面的问题是测试人员无法发现的。

3. 运营质量，而不只是研发质量

互联网产品的特点决定了测试完成并发布并不是研发流程的结束，而是刚刚开始接受用户的检验。发布时的功能可用并不代表功能的持续可用，所以需要一些手段来持续的检验。测试这边基于接口自动化的用例开发了一套线上的接口监控系统，可以第一时间发现一些后台系统的问题。

以下是一个监控系统的架构图。



当出现问题的时候，会第一时间发出邮件和微信的告警。

JD Mobile API 告警（5分钟间隔）

最近10分钟内概况

CGI平均响应时间（ms）	最大响应时间（ms）	正确率	网络延迟（ms）
112.88	10001.00	99.73%	5.36

异常详情

类型	CGI	状态码	响应速度（ms）	明细	状态
异常	母婴周年庆 - 强生婴儿	0	10001	详细数据	待处理
异常	sale.jd.com - /m/act/RxNeOcKfqs0o.html?resourceType=	0	10000	详细数据	待处理
异常	functionId=hasNewCoupons - 是否有待领取优惠券	200	2071	详细数据	待处理
异常	functionId=jdHomeShowItem - 查看京豆数量	200	2070	详细数据	待处理

超时详情

类型	CGI	状态码	响应速度（ms）	明细	状态
超时	m.jd.com - /download/downApp.html	200	1145	详细数据	待处理
超时	/client.action?functionId=catelogy - 手机	200	9074	详细数据	待处理
超时	/client.action?functionId=catelogy - 钟表	200	9071	详细数据	待处理
超时	/client.action?functionId=catelogy - 生鲜	200	9074	详细数据	待处理

并且可以按天汇总所有接口监控看到的可用性和性能的情况。

JD Mobile API 日报

CGI平均响应时间（ms）	总次数	最大响应时间(ms)	异常率	超时率	网络延迟
68.9831	292600	11199	0.03%	0.35%	11ms

当天概况

用例详情

用例名称	总次数	平均响应时间(ms)	最大响应时间(ms)	异常率	超时率
M版和原生机票主流程.jmx	7090	141	5199	0%	1.94%
M版电子书主流程.jmx	3534	122	1409	0.08%	0.11%
订单主流程.jmx	3540	70	3056	0%	0.08%
充值主流程.jmx	7090	52	4042	0.01%	0.13%
分类主流程.jmx	46085	52	3047	0%	0.07%
发现主流程.jmx	7799	61	3486	0.01%	0.1%
店铺主流程.jmx	3145	86	4031	0.03%	0.13%
彩票主流程.jmx	12762	49	1549	0%	0.04%
我的主流程.jmx	26217	57	3048	0.22%	0.04%
搜索筛选主流程.jmx	20816	66	4045	0.05%	0.31%
摇啊摇主流程.jmx	3545	84	705	0%	0%
检查M站首页活动页.jmx	24866	79	9991	0%	0.1%
检查商品详情页.jmx	28360	65	3044	0.02%	0.2%

除了以上接口的自动化监控之外，测试团队也会跟进一些外网问题的验证，并基于这些经验来优化测试用例，不断提升产品在外网的运营质量。