申请上海交通大学学士学位论文

Android 应用崩溃与用户行为分析

论文作者 熊 伟 伦

学　　号 5120379076

导　　师 戚正伟教授

专　　业 软件工程

答辩日期 2016 年 6 月 13 日

Submitted in total fulfillment of the requirements for the degree of Bachelor in Software Engineering

ANDROID APPLICATION CRASH AND USER BEHAVIOR ANALYSIS

Weilun Xiong

Advisor

Prof. Zhengwei Qi

School of Electronic Information and Electrical Engineering

Shanghai Jiao Tong University

Shanghai, P.R.China

Jun. 13th, 2016

Android 应用崩溃与用户行为分析

摘　要

随着移动互联网的兴起，运行 Android 系统的设备数量爆发式增长。通常每台 Android 系统的手机或者平板电脑都运行着数十到上百个应用程序，这些移动设备上的应用程序简称 App。其中部分 App 属于系统相关，由系统开发商开发，但更大一部分的 App 是由第三方开发者提供，用户通过应用商店下载安装。

开发者从用户设备上获取 App 使用的反馈，根据反馈信息对 App 的设计进行改进，调整运营方式，提高用户体验，是移动互联网开发中的重要环节。

由于 Android 系统的开放式策略，Android 平台的碎片化问题非常严重，包括系统版本碎片化和设备硬件型号碎片化，因此要开发出兼容多个系统版本和海量不同型号设备的 App 是一件具有挑战性的事情。获取 App 在不同设备、不同环境下的运行状况，是否崩溃，对提高用户体验也非常重要。

本篇论文介绍了 Appetizer，是一套收集 Android 平台用户使用 App 行为以及 App 卡顿、崩溃信息的系统，包括一个能够集成在 Android 应用中的轻量级软件开发工具包，和具有良好可扩展性的用于接收并处理数据展现给开发者的服务端程序。

本文描述了 Appetizer 的功能和设计，并与市面上现有的类似产品进行比较，结果显示 Appetizer 在 Android 设备信息收集的覆盖面较广，开发工具包占用空间明显小于同类产品，造成的性能开销不会影响用户体验。

关键词： Android　碎片化　移动设备　信息收集　用户行为

ANDROID APPLICATION CRASH AND USER BEHAVIOR ANALYSIS

ABSTRACT

With the development of mobile networks, the mobile Internet connectivity becomes universe, which also motivates a great surge in the availability of various mobile devices, especially Android devices initially developed by Google. Nowadays, every Android device, like smartphone and tablet runs tens or even hundreds of mobile applications, also terms "App" in short. Though some these apps are prebuilt in the mobile operating system or provided by the device vendors, most of the installed apps are 3rd-party apps downloaded from online app market websites.

It is quite common that app developers want to retrieve feedbacks about their apps from users' devices, to improve app design, adjust operation strategies and improve user experience. This feedback phase serves as an important stage in mobile app developments.

Due to the openness nature of the Android ecosystem, there exist a large number of different Android devices with slight difference from the official distribution, also termed as the "Android fragmentation problem". App developers face the challenges to develop an app that can adjust to hundreds of different device models. Whether the app would crash on certain device at certain environment is a vital information for the developers to improve app quality.

This thesis introduces Appetizer, a system that collects app runtime behavior, crash and lag information. Appetizer serves as a lightweight development kit, being integrated into developers' apps. It also has a server side unit that process incoming data and render statistics for developers.

The design features, implementation and comparison with various off-the-shelf commercial solutions are presented in this thesis. The result showed that Appetizer cover wide functions in Android device information collecting, Appetizer SDK size is significantly smaller than commercial solutions, overhead won't decrease the user experience.

关键词： Android fragmentation mobile device information collection user beavior

目 录

第一章　绪论

运行 Android 系统的设备数量随着移动互联网的兴起爆发式增长，2014 年著名的视频网站 YouTube 超过一半的流量来自于移动设备 [1]。

根据 Google 公司的统计，截止 2015 年 9 月 30 日，在全球范围内已激活的 Android 设备已经达到 14 亿台 [2]，超过微软公司的 Windows，苹果公司的 OS X 和 iOS，是全球运行设备数量最多的商用操作系统。

1.1　Android 生态系统

图 1–1 中国手机应用商店产业链 [3]

Fig 1–1 Mobile App store industry chain in China

通常每台智能手机运行着数十至上百个应用程序，简称 App。其中一部分是制造设备的厂商或定制 Android 操作系统厂商的 App，但用户每天接触时间最长的是第三方的应用厂商或者独立开发者开发的，并且需要连接互联网的 App。

在 Android 生态系统中，开发者完成 Android App 的开发完后，需要通过诸如 Google Play Store、腾讯应用宝、小米应用商店等 Android 应用市场发布 App，应用市场会对开发者提交的 App 进行安

全、内容和程序稳定性上的审核，通过审核后，用户再从应用市场搜索 App 进行下载安装和使用，流程如图1–1所示。

1.1.1 用户信息价值

从用户处获得信息反馈，根据反馈信息改进 App 的设计与体验、调整运营方式是移动互联网生态链中很重要的一个环节，因为用户体验直接影响到 App 的用户数量。

部分用户反馈可以通过应用市场的打分和评价中体现，但这些反馈包含的信息太少，只有简单的文字描述和分数评价，只能对其他用户起参考作用，难以进行更加深入的分析获取有价值的数据，对开发团队价值较小。除此之外，从应用市场得到的 App 反馈信息受制于应用商店，还会存在恶意竞争和刷单等问题，是 App 开发运营团队不可控的。

1.1.2 碎片化

Android 操作系统的一大问题是碎片化 [4]，全球各个手机厂商都有不计其数的不同型号的手机运行着不同版本的 Android 操作系统，这些手机的硬件参数不同，系统版本不一。但有“开放手机联盟（Open Handset Alliance）”的存在，规定了生产运行 Android 系统的设备都需要满足一定的兼容性要求，这使得 Android App 能够运行在绝大部分遵守规范的 Android 设备上，一定程度上增强了这个松散阵营的兼容性。

Android 碎片化主要包括设备碎片化和系统版本碎片化。设备碎片化是指不同厂商不同型号的设备，在硬件配置上的不同。例如 Android 手机使用各种各样的屏幕，这些屏幕的参数包括尺寸、分辨率、电容屏或电阻屏、是否支持压力、多点触控支持的上限等，不同的屏幕参数对于 App 的使用体验不一样，尤其是分辨率，直接涉及到 App 前端显示的内容多少。除了屏幕以外，CPU、GPU、内存、摄像头等硬件的参数同样多种多样，造成的影响就是在不同型号的 Android 设备上运行同一个 App，产生的效果截然不同。

系统版本碎片化是指不同的设备运行了不同版本的 Android 系统，不同版本系统号提供的 API 和库实现不一样，变动较大的版本号之间系统底层的机制也会不一样，每个 Android App 都指定了能够运行的系统大版本号范围。Android 系统版本非常复杂，当前情况下由 Google 公司主导“Android Open Source Project”，简称 AOSP，该分支开放给开源社区，属于最纯净的 Android 分支，通常意义上的 Android 系统大版本号跟随的是 AOSP 分支。Google 公司的 Nexus 系列设备，运行的是 Google 公司融合了 AOSP 和不开源的 Google Framework 的 ROM，通常被称为原生 Android ROM。但在中国，用户使用更为广泛的是各大厂商基于 AOSP 修改定制的 Android 操作系统，以小米公司的 MIUI 为代表。绝大部分能在 AOSP 上运行的 App 同样能够运行在这些系统上，但这些系统在一些地方对 AOSP 源码做了修改，行为会稍有不同，这些不同点容易产生意想不到的问题。

截止 2016 年 5 月 23 日，距离 Android 6.0 版本发布近一年，该版本的市场占有率还不足 10%，如图1–2所示是 Android 版本分布趋势走向。如图1–3所示是 Google 发布的 2016 年 5 月 2 日的 Android 版本分布。

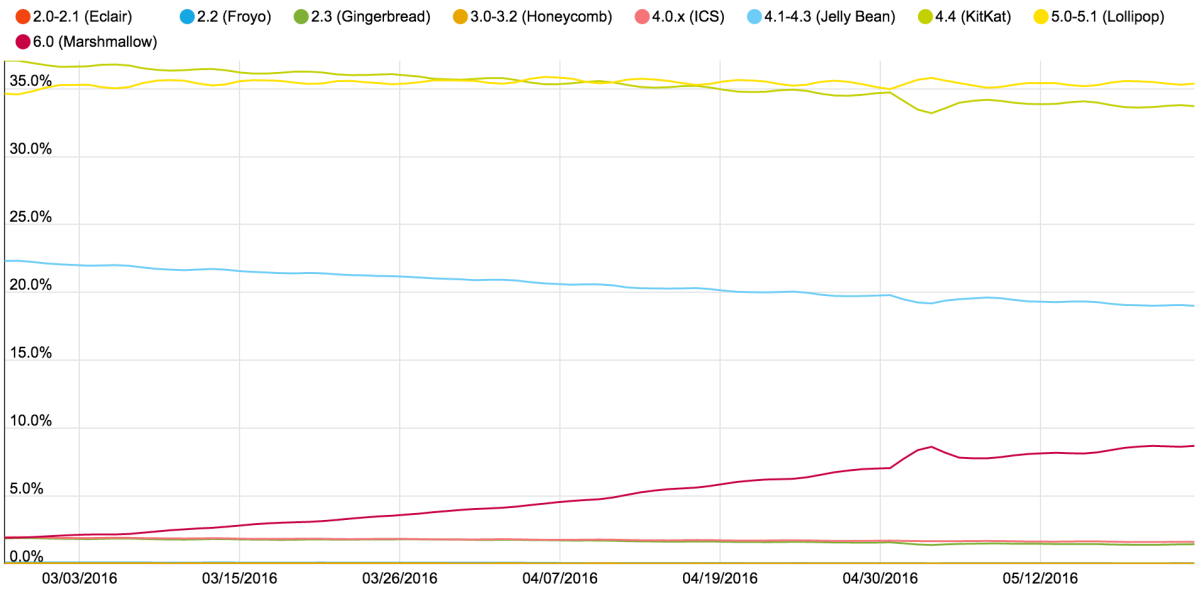

图 1–2 Android 系统版本趋势 [5]

Fig 1–2 Android system version trendency

Version	Codename	API	Distribution
2.2	Froyo	8	0.1%
2.3.3 - 2.3.7	Gingerbread	10	2.2%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	2.0%
4.1.x	Jelly Bean	16	7.2%
4.2.x		17	10.0%
4.3		18	2.9%
4.4	KitKat	19	32.5%
5.0	Lollipop	21	16.2%
5.1		22	19.4%
6.0	Marshmallow	23	7.5%

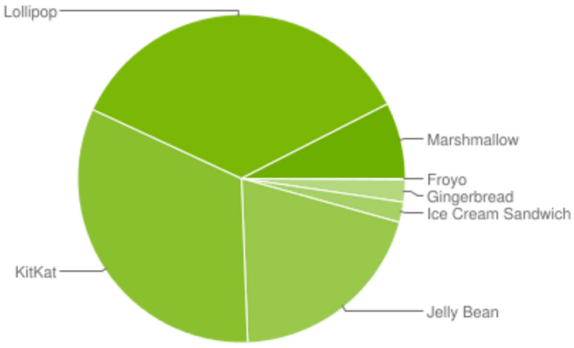

图 1–3 Android 系统版本分布 [6]

Fig 1–3 Android system version distribution

1.1.3 存在的问题

对于软件开发者，实现从所有用户设备中收集 App 的使用状况，需要考虑大规模并发、持久化存储、稳定性等各种问题，难度高而且工作量大。对于大团队或许有能力实现收集反馈信息的功能，但对于想要了解用户使用状况的独立开发者，开发这类功能的工作量过大，甚至会超过了 App 业务功能本身。

除此之外，开发者还要面对硬件参数各异、运行各种版本系统的用户设备，保证开发的 App 在碎片化的 Android 设备上不崩溃，并且都拥有较为良好的用户体验。开发团队在测试阶段不可能覆盖所有的 Android 设备，并且测试也难以覆盖所有情况。存在一种情况，在绝大部分设备上都能正确运行的 App，在某款用户量很少的 Android 设备上会崩溃，导致这部分用户无法正常使用。如果能够将这款设备的 App 崩溃信息发送给开发者，开发者就能够定位并解决问题，通过更新 App 提高兼容性和稳定性。

1.2 项目解决的问题

本文介绍的工具代号“Appetizer”，解决的是1.1.3小节中描述的问题。“Appetizer”包括一个能够集成在 Android 应用中的轻量级软件开发工具包，简称为客户端 SDK，和具有良好可扩展性的用于接收并处理数据的服务端程序。

Appetizer 客户端 SDK 能够收集用户使用 App 的操作路径和时间，在 App 崩溃时收集程序调用栈和设备相关状态信息，侦测 Android 应用程序未响应事件，记录 App 每次启动时的黑白屏时间，存储这些使用信息，并在网络条件允许的情况下将收集到的信息发送到服务端。

Appetizer 服务端能够持久化存储客户端 SDK 发送的消息，对 App 使用信息做处理和统计分析，展现有价值的内容给 App 的开发团队和运营团队，包括日活跃用户、崩溃信息分类、机型统计等处理过的数据。

Appetizer 客户端 SDK 的特点是轻量和稳定。不使用任何第三方库，尽可能减小 SDK 的空间占用，减少使用该 SDK 的 App 的空间占用增加。同时保证稳定性，不会因为 SDK 本身的问题造成 App 崩溃。Appetizer 服务端在架构上保证了良好的可扩展性，可以通过简单的增加服务器数量承受更大的流量压力。

1.3 相关工具产品

中国市场上，提供用户行为分析统计和崩溃信息收集的商业产品主要有腾讯 Bugly[7]，友盟 [8] 和 TalkingData[9]。他们的产品对于 Android App 用户行为分析方面的功能包括用户活跃度、用户构成、渠道分析、页面访问路径统计、事件转化率等，主要是对 session 统计得到的数据进行处理的结果，其中腾讯 Bugly 对应用崩溃信息分析处理功能较强。

国际市场上，提供相关功能的商业产品主要有 Google FireBase[10]，Yahoo Flurry Analytics[11] 和 Twitter Fabric[12]。其中 Google FireBase 同时包括用户行为统计和崩溃信息收集，其他功能也较为丰富，是 Google 公司主推的 Android 开发者服务工具。Flurry Analytics 可以对用户行为统计数据进行较为复杂的分析和报表输出。Fabric 可以将所有用户的 App 崩溃信息反馈给开发团队，并且还有一整套团队测试工具进行辅助。

开源软件 ACRA[13] 是做 Android App 崩溃信息统计的工具，因为是开源软件需要使用者自己搭建服务端，崩溃信息收集功能完善并且发送的信息有很好的可定制性。

国内外的学术界也有对 Android 应用的用户使用行为进行分析的研究 [14, 15]。

1.4 本章小结

本章是整篇文章的绪论，描述了整篇文章研究的主要问题。

1.1节介绍了 Android 生态系统的大致流程、碎片化的问题和整个生态系统面临的问题，1.2节介绍了“Appetizer”的主要功能、特点和能够解决的问题，1.3节列出了国内外解决 Android 生态问题的相关商业产品和开源产品以及它们的特点。

第二章　背景

2.1　Android 模型

每个操作系统都有独特的系统模型，Android 系统底层基于 Linux 内核，同时做了许多扩展。本节介绍 Android 系统架构和应用模型，为阐述本文解决的问题和方法做铺垫。

2.1.1　Android 系统架构

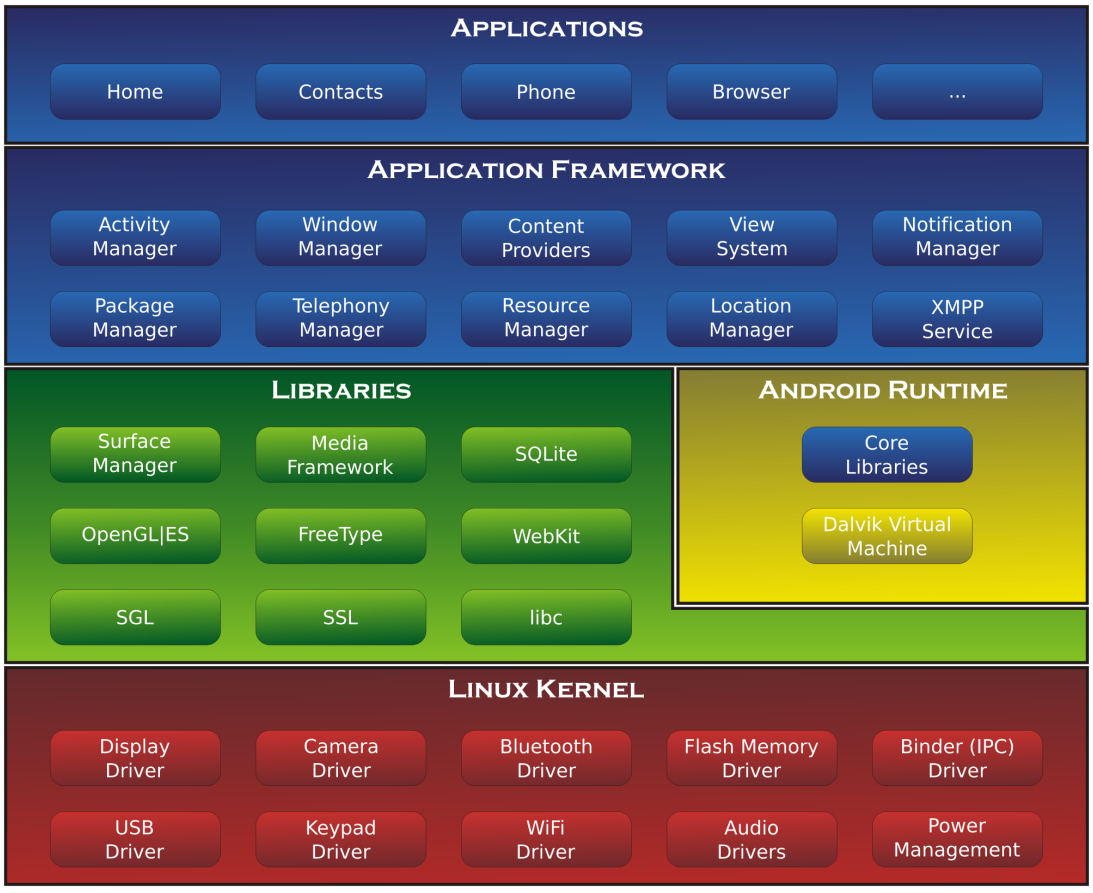

图 2–1 Android 系统架构[1]

Fig 2–1 Android system architecture

图2–1展现了整个 Android 系统的架构。Android 系统底层基于 Linux 内核，使用 Linux 内核文件形式的硬件驱动。Android 系统中间层提供了常用的工具类库，包括轻量级数据库 SQLite、浏览器引擎 WebKit、图形程序接口 OpenGL ES 等，Android 系统中间层还有 Android Runtime，使用 Dalvik

[1] https://en.wikipedia.org/wiki/Android_(operating_system)

虚拟机运行编译成字节码的程序。Android 系统上层提供了窗口管理器、包管理器、资源管理器等能够让 Android 应用程序直接调用的工具和类库。

Android 应用建立在 Android 系统三层架构之上，开发者通过调用 Android 系统提供的接口实现各种 Android 应用程序。

Android 系统提供了名为 SharedPreferences 的轻量级事务性 key-value 存储系统，每个 Android 应用程序有独立的存储空间，Appetizer 基于该轻量级 key-value 存储系统和文件系统，实现了满足原子性的轻量级二进制数据持久化消息队列。

2.1.2 Android 应用模型

一台 Andorid 设备里的每个 Android 应用拥有独立的 UID（User Identifier），每个 App 都相当于 Linux 中独立用户，通过复用 Linux 的用户级隔离机制来实现 Android 应用的权限控制和数据隔离。

Android 应用是组件化的，每个 App 包括运行在前台和后台的若干组件。一个组件一般由一些 Java 类定义，默认多个组件在同一个 Linux 进程中执行，也可以通过特别的设置将前台与后台的组件运行在不同的 Linux 进程中。例如同一个开发团队两个 App 分别有一个前台 Activity 组件和后台 Service 组件，可以设置将 Activity 和 Service 拆开在两个不同的进程中运行，还可以设置两个 App 公用同一个 Service 组件，这样当两个 App 同时运行的时候系统中只运行了三个组件。

Android App 可以注册侦听某些事件，例如网络断开、屏幕开启、电池电量不足等，当事件发生时执行一段程序。一个组件在运行时也可以产生自定义事件，并可以指定把事件告知某个或者某一类组件。

2.2 反馈信息

Android App 使用过程中会产生许多数据，其中有些数据对于开发运营团队有价值，可以帮助完善优化程序，调整运营策略，提高用户体验。

Appetizer 客户端 SDK 会收集用户会话信息、应用崩溃信息、ANR 信息和黑白屏时长信息，本节的每个小节分别介绍了各个反馈信息的具体内容，对用户体验的影响及其数据价值。

2.2.1 用户会话

Android 应用程序的用户会话，可以类比网站统计中的会话，简称 session，这里可以称为 App session，指的是用户使用一次 App 的整个过程，包括起止时间，打开的内容，内容跳转的路径，每个内容浏览的时间。

在 Appetizer 中，“页面”定义为 Android 中的 Activity 对象跟 Fragment 对象，是 App session 中用户浏览路径的基本单元。对于一次用户会话的具体定义如下：

- 从启动应用到关闭应用
- 从启动应用到应用退至后台，且在后台运行时间超过 30 秒
- 从启动应用到应用停留在某个页面未跳转时间超过 30 秒

例如用户在一次使用 App 的过程中跳转了多次页面，一个用户会话会包括多个浏览路径的信息。用户在将 App 切换至后台运行，超过 30 秒后再打开，Appetizer 认为这是两次不同的会话，相当于

用户打开了两次 App。

用户会话可以用于统计日活跃用户，周活跃用户等数据，结合设备信息可以分析出日新增用户，老用户回流等更深层次的数据，这些分析数据在网站统计中已经较为成熟。基于用户会话统计的用户活跃度分析，是应用运营、市场调研最重要的数据来源，可以帮助开发者设计追踪营销策略，了解市场，并且获得新功能的用户喜爱度反馈。

2.2.2 Android 应用崩溃

Android 应用程序最终以字节码的形式打包，所以通常使用能够运行在 Java 虚拟机上的程序语言进行开发，绝大部分 Android 应用使用 Java 语言开发，少部分使用 Scala 和 Kotlin 等 JVM 系语言开发，还有很少一部分使用 Go 语言和 Swift 语言开发。Android 系统同时提供 Native Development Kit，支持 C/C++ 语言开发。

Android 应用的崩溃可能发生在 Java 层，以 Java 抛出未被捕获的异常 (Exception) 或者错误 (Error) 的形式发生，崩溃也可能发生在 Native 层，本篇文章和 Appetizer 只考虑收集 App 发生在 Java 层的崩溃信息。

Android 应用崩溃会影响用户的正常使用，崩溃的原因可能是程序逻辑代码本身的问题，设备碎片化导致的极少数设备运行问题，或者 Android 系统本身的问题。App 崩溃会影响用户体验，在关键位置崩溃会导致 App 根本无法使用，崩溃会直接影响到用户量，对于用户量非常大的 Android 应用，低概率发生的崩溃就可能影响到成百上千万的用户。

Andorid 应用崩溃信息收集功能可以及时把用户使用产生的崩溃问题告知开发团队，开发团队根据函数调用栈、设备型号、系统状态等信息对崩溃原因进行分析，及时解决问题，减少程序崩溃带来的负面影响。

2.2.3 应用程序未响应

Android ANR 全称 Application not responding（应用程序未响应），是指 App 运行某一段代码逻辑的时间过长，而无法响应用户的操作，通俗的说法就是程序卡死。

在 Android 里，应用程序的响应性是由 Activity Manager 和 WindowManager 系统服务监视的，当它监测到以下情况中的一个时，系统就会针对特定的应用程序显示 ANR[16]：

- 在 5 秒内没有响应用户的输入事件
- BroadcastReceiver 在 10 秒内没有执行完毕

Android App 和用户交互的界面由主线程更新，因此主线程又称为 UI 线程。应用程序长时间不响应用户输入的原因，可能是 UI 线程在长时间执行其他任务，例如数据库操作、网络操作、复杂的计算操作，也可能是 UI 线程死循环或者死锁。

间歇性发生 ANR 会让用户感觉应用卡顿，使用体验下降，长时间发生 ANR 会导致用户无法与应用交互，也就无法使用 App，只能让系统强制关闭该应用。收集 ANR 发生时的函数调用栈信息和设备实时状态，发送给开发团队，有利于开发团队解决 ANR 问题，提升 App 的用户体验。如图2–2是 Android 应用程序发生 ANR 时系统的弹窗提示。

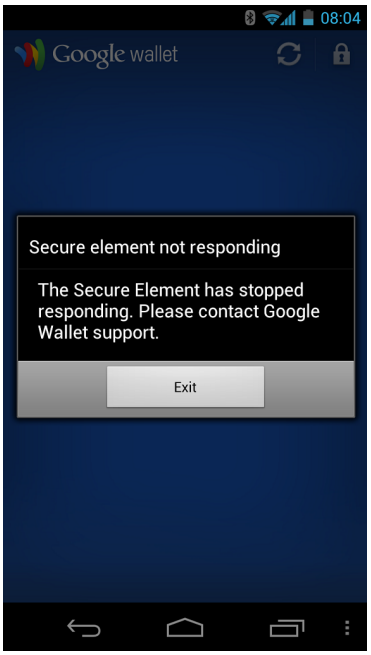

图 2–2 Android ANR 事件
Fig 2–2 Android ANR event

2.2.4 启动黑白屏

Android 应用在启动的时候需要加载作为入口的 Activity 页面。在应用启动成功之后，入口 Activity 的布局内容加载完成之前，显示在设备屏幕上的是 Android 应用的主题背景，默认的通常是黑色或者白色，在这段时间用户只能看到黑色或白色的屏幕，称为 Android 应用启动时的黑白屏问题。

黑白屏问题是程序启动加载需要时间导致的必然存在的问题，无法彻底解决。黑白屏持续的时间和应用程序本身、Android 系统版本、硬件性能、设备实时状态都有关，一些 Android 应用选择透明背景或者一张图片掩盖黑白屏问题，缓解用户在这段应用启动等待时间的急躁心理，应用启动后的等待时间过长会影响用户体验。

Appetizer 客户端 SDK 能够记录每次 Android 应用启动后，加载入口 Activity 布局文件的等待时间，也就是黑白屏时间。该数据发送给开发团队，可以让开发团队判断启动加载时间长度是否能够接受，决定是否需要对程序进行权衡和优化，以减少应用启动后的等待时间，提升 App 的用户体验。

2.3 本章小结

本章介绍了 Appetizer 项目和所解决的问题相关的技术背景，为后文的设计和实现做铺垫。

2.1节介绍了 Android 系统相关的背景知识，包括 Android 系统架构和 Android 应用组件化的模型。2.2节介绍了 Appetizer 客户端 SDK 所收集信息的相关背景知识，以及这些信息对开发团队的价值。

第三章　客户端 SDK 设计与实现

3.1　客户端 SDK 概要

Appetizer 客户端 SDK 是提供给开发者用于集成到 App 的部分，是达成用户使用 App 信息收集任务的必要程序。客户端 SDK 提供给开发者的形式是一个 jar 包，开发者通过在原来的应用程序代码中插入 jar 包提供的 API 达到实现信息收集的目的。

客户端 SDK 的主要功能包括

- 用户 Session 信息收集
- 用户浏览路径和时长信息收集
- 应用崩溃信息和发生崩溃时设备状态收集
- 应用 ANR 侦测和发生 ANR 时设备状态收集
- 应用启动黑白屏时间信息收集

Appetizer 客户端 SDK 的设计目标是在达到完成需求功能的前提下。为了使 jar 包占用空间尽可能小，客户端 SDK 的实现没有使用任何第三方库。为了减少 Android 应用程序因为集成 Appetizer 客户端 SDK 造成的额外性能开销和内存开销，客户端 SDK 不会启动任何新的进程，所有任务都在原有的 Android 应用程序进程中实现，包括收集数据、存储数据、发送数据。本章后续几节从设计到实现都体现了 Appetizer 客户端 SDK“轻量级”的目标。

Appetizer 客户端 SDK 需要持久化存储的数据都在每个 Android 应用程序的独立存储区域中，一台 Android 设备安装了多个集成 Appetizer 客户端 SDK 的应用程序，它们的数据不会发生冲突。

设备的本地时间是不可信的，但是 Appetizer 客户端 SDK 中记录的时间相关信息只能使用设备本地时间，因此在每个发送到服务端的信息中会加入发送时刻的设备本地时间，服务端可以根据这个时间和服务端本地时间对所有数据的时间进行校准。

3.2　客户端 SDK 架构

客户端 SDK 主体流程部分架构如图3–1所示，Appetizer 客户端 SDK 的入口类名是“Appetizer”，在 Android 应用程序的 Application 类初始化的同时进行初始化，不会创建任何新的进程。Appetizer 会读取配置文件，创建自定义的 Logger 对象，以及使用 SharedPreferences 访问该 Android 应用程序与其他应用程序隔离的独立存储空间。

所有需要收集的信息会通过特定的方式传递给 Collector，各个信息收集的具体实现方式参见3.3节。Collector 会把收集到的数据存入持久化消息队列，该队列实现的类名为 PersistentQueue，实现了 Java 队列类的全部接口，提供给上层的抽象是程序关闭后，重新运行程序后队列的数据能够恢复，不会丢失，该轻量级持久化消息队列的具体实现参见3.3.2小节。

Appetizer 会在 Android 应用程序的主进程中创建一个 HandlerThread，称为 Shepherd，这是一个长时间存在的线程，当有任务需要执行的时候才会被唤醒，避免了一般线程多次销毁创建的开销。Shepherd 的作用是判断网络状况，发送持久化消息队列中的数据到服务端。当 Collector 收到新的数

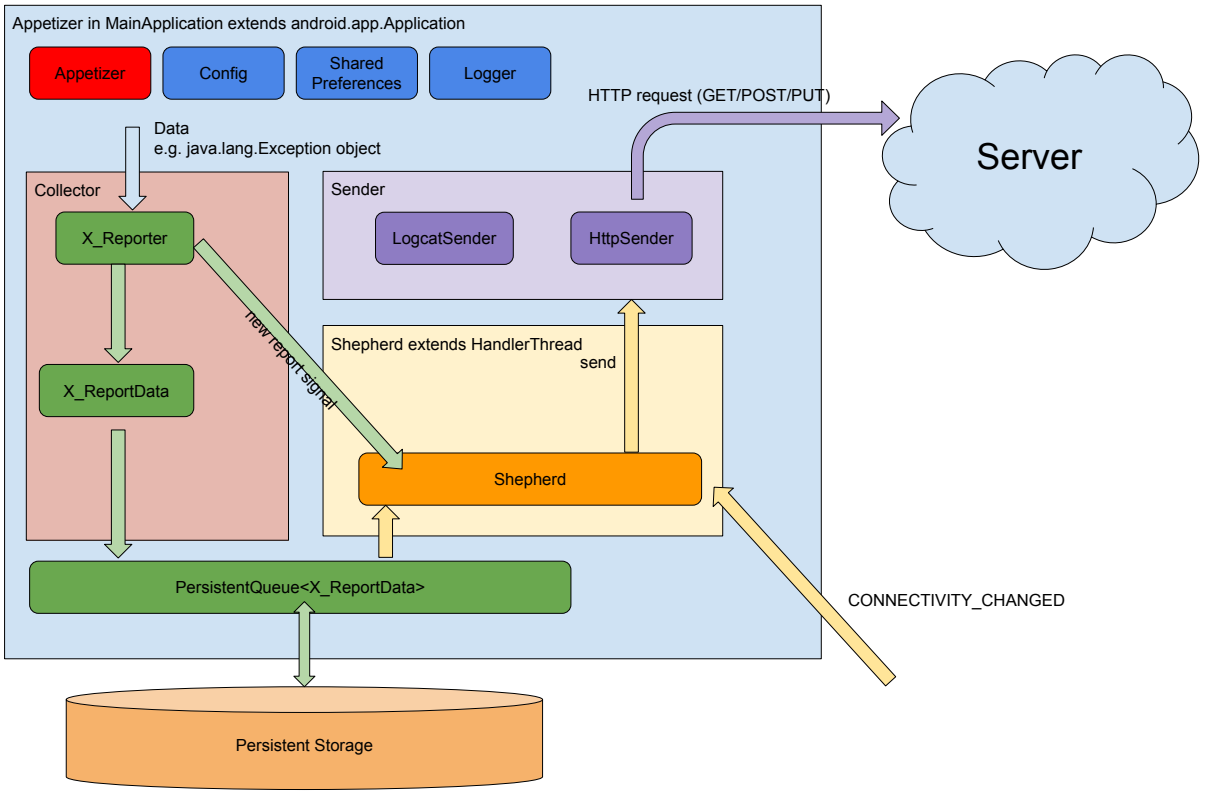

图 3–1 客户端 SDK 架构
Fig 3–1 Client SDK architecture

据，会向 Shepherd 发送消息，Shepherd 得到消息会首先判断网络状况，网络状况允许的情况下再将持久化消息队列中积存的消息通过 HTTP 请求发送给服务端。除此之外，Shepherd 注册侦听设备的网络变化状况，当 Android 设备从未连接到连接互联网成功时，系统会发送网络状况变化的事件消息给所有注册侦听该事件的进程，接收到该消息后 Shepherd 同样会发送持久化消息中积存的消息给服务端。

3.3 客户端 SDK 实现

Appetizer 客户端 SDK 最低支持 Android API 9，即 Android 2.3 Gingerbread。根据 Google 官方的数据，截止 2016 年 5 月 2 日，系统版本低于 Android 2.3 版本的活跃设备数量不足全部活跃设备数量的 0.1%，可以认为 Appetizer 客户端 SDK 支持 99.9% 以上的 Android 设备。

3.3.1 网络模块

图3–1中的 HttpSender 模块，即客户端 SDK 的网络发送部分，该模块基于 Apache HTTP Client 封装实现，用于将 Appetizer 客户端 SDK 收集到的数据通过 HTTP POST 发送到服务端。

在 Android 6.0 及以上的系统中，Apache HTTP Client 已被弃用，采用 Ok HTTP 作为默认的 HTTP 客户端。但是由于 Android 6.0 及以上的系统使用 ART RunTime 取代 Dalvik 虚拟机，ART RunTime 不支持在 Android 应用程序崩溃后创建新线程，Ok HTTP Client 发送 Http 请求后为了不阻塞原线程继续运行，采用创建新线程的方式异步接受服务端的返回值，导致的问题是在 ART RunTime 下 Andorid 应用崩溃后无法使用 Ok HTTP[17]。因此 Appetizer 客户端 SDK 的网络模块，基于老版本 Android 系统默认的 Apache HTTP Client 进行封装实现。

3.3.2 持久化队列

Appetizer 客户端 SDK 的持久化队列 PersistentQueue，能够存储任何二进制数据，实现了 Java Queue 的全部接口，任何 Java 使用抽象类 Queue 申明的对象都可以被赋值为 PersistentQueue，而且对象的操作方法保持不变。

每次程序对 PersistentQueue 对象的入队或者出队操作成功后，队列的持久化存储数据都会发生变化。Android 应用程序关闭、设备重启或者内存断电后，当 Android 应用程序重启启动时，Appetizer 初始化时能够恢复持久化消息队列到上一次操作成功后的状态。Appetizer 客户端 SDK 实现的持久化队列 PersistentQueue 基于 Android 的轻量级 key-value 存储系统 SharedPreferences 和 Android 的文件系统进行实现，并且保证了操作的原子性，在 PersistentQueue 方法执行的过程中发生了断电、存储硬件被取出等会导致方法无法正常执行或中断的情况时，Appetizer 重新初始化后 PersistentQueue 能够还原到该操作没有进行的状态。

Android 的 key-value 存储系统 SharedPreferences 底层是基于文件系统和 XML 格式实现的，并且保证了原子性和一致性。在 Appetizer 客户端 SDK 的功能需求上，需要持久化队列保存 Java 对象序列化后的二进制数据，SharedPreferences 通常用于实现配置文件存储，简单的用户信息存储等。但是由于 SharedPreferences 存储实现的编码问题，不同的编码对于文本换行符的定义不一致，单个字节 0x0A 通过 SharedPreferences 写入后再读出会变成字节 0x0D，因此无法使用 SharedPreferences 保存 Java 序列化对象。一种解决办法是存储 Base64 编码的对象序列化数据，但是比较复杂的对象序列化数据很大，每次操作进行 Base64 编码的计算开销过大不能接受。

Android 的文件系统可以写入和读取二进制数据，并且数据不会发生变化，但是 Android 系统提供的文件操作无法保证操作的原子性，如果在文件写入操作执行过程中发生了程序崩溃、设备断电或者存储硬件被取出等问题，会导致只有部分数据成功写入，而且程序无法判断写入操作是否成功执行，因此再进行读取操作时会发生问题。

Appetizer 客户端 SDK 实现的原则是不使用第三方库，对文件系统的操作进行封装，同时支持原子性和 Java 对象序列化持久存储，比较复杂而且难以保证实现的正确性。结合支持原子性操作的 SharedPreferences 和支持 Java 对象序列化持久存储的文件系统操作 API，可以通过简单的方式实现日志式解决方案的支持原子性的持久化消息队列。

每个 PersistentQueue 对象中包括四个成员对象，SharedPreferences 句柄、两个文件句柄和存储在内存的 LinkedList 对象。

创建 PersistentQueue 对象时会进行初始化，首先从 SharedPreferences 中读取持久化保存的当前 PersistentQueue 的状态，获取最后一次成功的原子操作后数据存储的文件，然后调用文件 IO API 读取文件中的数据流，数据流反序列化成对象赋值给 LinkedList 对象。如果不存在之前的 PersistentQueue

存储文件，会创建空的 LinkedList 对象，再调用原子性文件写入操作。

PersistentQueue 的关键技术是原子性文件写入操作，步骤如下：

1. 读取 SharedPreferences 的状态信息，获取当前数据保存的文件，选择写入另一个文件。
2. SharedPreferences 的状态改变为“文件写入操作开始”。
3. 序列化 LinkedList 对象并写入文件流。
4. SharedPreferences 的状态改变为“文件写入操作完成”。

如果操作在第 1 步崩溃，因为没有开始写入操作，重新载入不会出现问题。操作第 2、4 步由 SharedPreferences 保证原子性。如果在操作第 3 步崩溃，重新载入恢复的时候根据 SharedPreferences 的状态信息判断在文件写入时崩溃，读取另一个文件反序列化恢复 LinkedList 对象数据。图3–2展现了 Appetizer 操作持久化队列的程序执行流程，其中 File A 和 File B 互为备份，交替写入，状态都保存在 SharedPreferences 中。

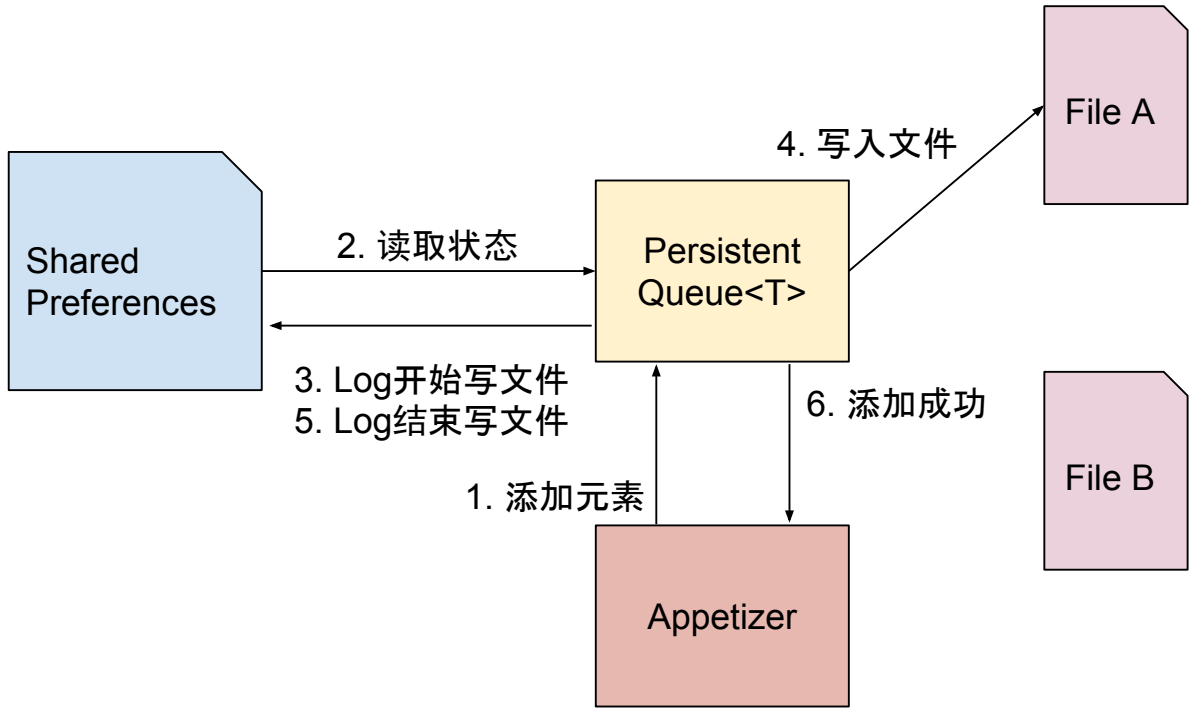

图 3–2 持久化队列

Fig 3–2 Persistent Queue

PersistentQueue 的单个元素修改、增加、删除操作，首先操作 LinkedList 的元素，然后调用原子性文件写入操作保证数据持久化存储，最后函数执行完成，返回函数返回值。PersistentQueue 的多个元素操作为了提高性能避免多次原子性文件写入操作，在对内存中的 LinkedList 对象修改全部执行完毕后调用一次原子性文件写入操作，而不是简单的复用单个元素的修改进行实现。

PersistentQueue 的元素读取操作直接读取内存中的 LinkedList 对象，不需要其他的修改。

在 Appetizer 客户端 SDK 的实现中，存在三个 PersistentQueue 对象，分别是用户会话信息的持

久化队列、崩溃信息和 ANR 信息的持久化队列、黑白屏时长信息的持久化队列，其中 ANR 信息和崩溃信息收集的数据类似，因此合并成一个数据类，共用同一个持久化队列。

3.3.3 用户会话收集

用户使用 Android 应用的会话信息 (App session) 是所有信息收集功能中被调用最频繁的一个，Android 应用程序的浏览“页面”被定义为每个 Activity 和 Fragment，因此当用户每次切换“界面”的时候都需要收集信息。Appetizer 客户端 SDK 收集的浏览路径时间，定义为从 Activity 和 Fragment 生命周期的 onResume 开始，到 onPause 结束。在每个开发者想做数据收集的 Activity 和 Fragment 的这两个生命周期函数开头，需要插入一行代码调用 Appetizer 客户端 SDK 提供的 API，并将 this (Context 对象) 作为参数传入。用户会话信息的时间全部使用从 epoch 开始的时间戳表示，单位毫秒。

因为在 Android 系统中，每个进程被杀死的时候进程本身不能再执行任何操作，无法在强制退出前保存状态，因此为了在 Android 应用程序进程被强制杀死的时候用户会话信息不丢失，在一个会话里每次浏览路径更新的时候需要持久化存储会话当前的快照。快照更新事件的频率较高，如果快照通过 PersistentQueue 进行持久化存储，开销过大。用户会话信息包括浏览路径可以通过字符串表示，因此不需要进行序列化存储在 PersistentQueue 中，可以使用 SharedPreferences 完成快照持久化存储的目的。

Android 应用程序正常使用并正常退出的情况下，用户会话信息变化如下:

1. Android 应用程序启动，内存中创建新的 session 对象，并写入会话开始时间和浏览路径节点开始时间和节点名，当前会话快照存入 SharedPreferences。
2. 用户切换“界面”，session 对象写入上个浏览路径节点的结束时间，判断上个节点浏览时间是否超过 30 秒，如果未超过则写入新的浏览路径节点的开始时间，如果超过 30 秒则将上个 session 对象存入 PersistentQueue，创建新的 session 对象，并写入会话开始时间和浏览路径节点开始时间和节点名，当前会话快照存入 SharedPreferences。该步骤循环零次或者若干次。
3. 用户将该 Android 应用程序退到后台，session 对象写入当前浏览路径节点的结束时间，当前会话快照存入 SharedPreferences。
4. 用户将该 Android 应用程序从后台切换到前台，判断距离上个节点浏览结束的时刻是否超过 30 秒，如果未超过则写入新的浏览路径节点的开始时间，如果超过 30 秒则将上个 session 对象存入 PersistentQueue，创建新的 session 对象，并写入会话开始时间和浏览路径节点开始时间和节点名，当前会话快照存入 SharedPreferences。

在很多情况下 Android 应用程序会被强制杀死，例如系统内存不足的时候，系统会决定杀死部分进程释放内存，或者内存清理工具、安全管家等软件会将进程杀死。在进程被杀死后，用户会话信息只存储在 SharedPreferences 里，并没有作为一个完整的会话数据加入持久化消息队列中。Appetizer 客户端 SDK 考虑到了这种情况，在该 Android 应用程序下次启动的时候，用户会话收集的 Collector 实例初始化阶段会从 SharedPreferences 里恢复应用程序被杀死前的用户会话快照，写入持久化消息队列并清除快照。

3.3.4 崩溃信息收集

崩溃信息收集是 Appetizer 客户端 SDK 的重要功能，开发者只需要在应用程序的 Application 调用 Appetizer 的初始化方法，就自动集成好了崩溃信息收集功能。

崩溃信息收集功能的核心部分是单例类 CrashReporter，在 Appetizer 初始化的时候创建。CrashReporter 继承了接口 Thread.UncaughtExceptionHandler，实现了 uncaughtException 方法，并调用 Android SDK 方法 Thread.setDefaultUncaughtExceptionHandler 传入自身作为参数，把 CrashReporter 设置为默认的异常处理类，可以捕获全局异常。所有未被捕获的异常都会导致 Android 应用崩溃，如果设置了全局异常捕获，在应用发生崩溃前异常会被 CrashReport 的 uncaughtException 方法捕获，可以在该方法内实现崩溃信息和设备实时状态信息的收集。

在 uncaughtException 方法中，首先保存用户会话信息到持久化消息队列，因为此时 Android 应用程序发生崩溃，本次会话结束。然后读取异常、线程和设备状态的数据，收集需要的信息加入持久化消息队列，并发送信号到 Shepherd 触发网络发送事件。崩溃时收集的数据如下：

- 设备国际移动设备标识（IMEI），15 位数字字符串，例如“358239058889108”。
- 应用版本名（App version name），字符串数据，例如“2.1.1 Alpha”。
- 应用版本号（App version code），数值数据，例如“10”。
- 应用包名（App version name），字符串数据，例如“io.appetizer.demo”。
- 文件路径（File path），例如“/data/user/0/io.appetizer.demo/files”。
- 设备型号（Phone model），例如“Nexus 5”。
- 设备品牌（Brand），例如“google”。
- 产品名（Product），例如“hammerhead”。
- 系统版本（Android system version），例如“6.0”。
- 系统构建数据（Android build information），包含了系统 ROM 的详细信息。
- 内存总大小和内存剩余大小（RAM size），内存剩余大小指崩溃时的设备内存剩余大小。
- 闪存总大小和闪存剩余大小（Flash size），通常是手机内不可取出的存储硬件。
- 外部总大小和外部剩余大小（External size），通常是手机内可取出的存储卡。
- 函数调用栈（Stack trace），Exception 对象有价值的数据转为 JSON 格式发送，格式化的数据方便服务端分析，没有信息丢失。
- 函数调用栈哈希值（Stack trace hash），用于快速查找合并相同的崩溃原因。
- 应用启动时设备设置信息（Initial configuration），包括字体大小、音量大小等。
- 应用崩溃时设备设置信息（Crash configuration），具体内容同上。
- 屏幕信息，包括屏幕尺寸、分辨率等。
- 设备硬件支持列表（Device features），显示设备支持的功能，包括蓝牙、相机、多点触控、GPS、NFC 等。
- 网络信息（IP），设备崩溃时的网络环境信息和 IP 地址。

上述收集的崩溃数据以 Java 序列化对象的形式存储在持久化消息队列中，当 Shepherd 判断网络可以发送时，反序列化消息队列的崩溃信息，再转为 JSON 格式放入 HTTP POST body 中发送到服务端。崩溃信息数据量庞大，转为 JSON 格式开销较大，但是相比于用户会话，崩溃信息收集事件发生的频率较低，因此格式转换的开销影响不大。

崩溃时收集的部分信息设计到读取设备敏感信息，需要额外的权限，Appetizer 客户端 SDK 需要 Android 应用程序提供的权限原由来自于此。包括：

- android.permission.READ_PHONE_STATE
- android.permission.ACCESS_NETWORK_STATE
- android.permission.ACCESS_WIFI_STATE

如果开发者的应用程序源代码中的 Manifest 没有申明这些权限，Appetizer 客户端 SDK 的崩溃信息收集功能会有部分数据无法收集。

3.3.5 ANR 侦测

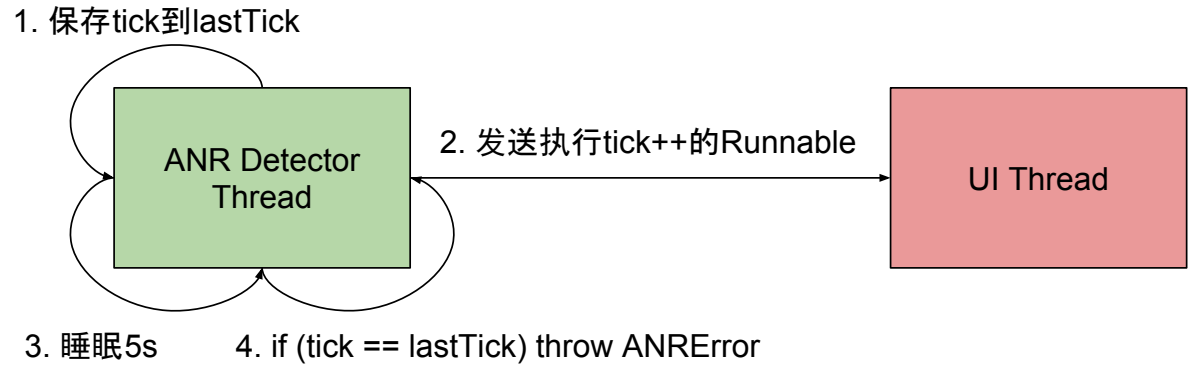

图 3–3 ANR 侦测流程

Fig 3–3 ANR detector workflow

2.2.3小节介绍了 Android 应用程序未响应（ANR）的相关背景知识，Appetizer 客户端 SDK 的 ANR 侦测参考了 ANR-WatchDog[18] 并复用崩溃信息收集功能，在实现上进行了一定程度的简化。

集成了 ANR 侦测功能的 Android 应用程序在前台运行时，ANR 侦测功能开启。ANR 侦测模块会创建 volatile 整形数值变量 tick，并在 Android 应用程序主进程中创建一个新的线程，volatile 关键字会保证一个线程更新了某个对象后会立刻同步到其他线程，不会因为线程变量缓存的问题造成数据同步延迟问题，如图3–3所示，ANR 侦测线程会保存当前 tick 的数值到 lastTick 变量，再发送一个会对 tick 进行加一操作的 Runnable 给主线程（UI 线程）执行，volatile 关键字的存在保证了主线程更新 tick 之后 ANR 侦测线程可以获取到最新的 tick 的值，然后睡眠 5 秒检查 tick 和 lastTick 变量是否相同。如果数值相同，则说明主线程在 5 秒内没有空余时间执行发送过去的 Runnable，认为此时主线程发生应用程序未响应，ANR 侦测程序会创建 ANRError 对象，ANRError 继承 Java Error 类并且能够收集所有线程的函数调用栈，ANR 侦测程序再抛出 ANRError 对象，3.3.4小节描述的应用崩溃信息收集模块会捕获该错误，并按照发生崩溃的流程收集此时的设备状态信息，ANRError 错误中包含了此时所有线程函数调用栈信息，方便开发者进行应用程序未响应错误原因的定位。通过抛出 ANRError 的操作复用了应用程序崩溃信息收集模块，避免单独开发，减少 Appetizer 客户端 SDK 的占用空间，并且可以和一般的应用程序崩溃信息共用一个持久化消息队列。如果数值不同，则说明

主线程在 5 秒内执行了发送过去的 Runnable，可以认为过去的 5 秒主线程没有发生应用程序未响应的问题，该 ANR 侦测线程继续保存 tick 到 lastTick，发送 Runnable 并睡眠 5 秒，如此循环。

如果 ANR 侦测线程不间断运行，会导致设备的处理器无法休眠，耗电量增大。为了减少 ANR 侦测的开销，Appetizer 客户端 SDK 注册侦听设备屏幕关闭事件，当 Android 应用程序切换到后台或者设备屏幕关闭，ANR 侦测线程会长时间休眠，当该 Android 应用程序切换到前台使用时 ANR 侦测模块才会重新开启。ANR 侦测程序的逻辑代码计算量不大，而且每 5 秒执行一次，对 Android 应用程序造成的性能影响不大。

3.3.6 黑白屏时长记录

黑白屏时长是指 Android 应用程序启动后到加载第一个页面完成之间的时间，在 Android Activity 生命周期中是从 onCreate 开始到 onResume 这一段时间。因此开发者在 Android 应用程序中启动后的第一个 Activity 的 onCreate 开头和 onResume 结束调用 Appetizer 客户端 SDK 提供的两个 API，就能够记录 Android 应用启动时黑白屏时长。Appetizer 客户端 SDK 的这两个 API 会记录被调用时设备的本地时间，加入到持久化消息队列等待发送到服务端。本地时间是不可信的，但是黑白屏记录的两个时间点间隔较小，而且只需要两个时间点的时间差，因此两个本地时间点的差值可以直接作为可信的黑白屏时长数据。

3.4 本章小结

本章内容篇幅较大，介绍了 Appetizer 客户端 SDK 的方方面面，是整个项目技术实现的核心部分。

3.1节介绍了 Appetizer 客户端 SDK 的主要功能、轻量级的特点、多个应用数据隔离和对不可信的本地时间校准的方法。

3.2节通过架构图从宏观上介绍了 Appetizer 客户端 SDK 的各个模块的作用和各个模块之间的关联。

3.3节详细介绍了 Appetizer 客户端 SDK 各个功能模块的设计方案、取舍原因和实现过程。3.3.1小节介绍了选择 Apache HTTP 客户端的原因。3.3.2小节介绍了实现持久化消息队列的重要性，选择基于 SharedPreferences 和 Android 文件系统实现持久化消息队列的原因，以及持久化消息队列原子性的实现细节和使用方法，比较深入详细。3.3.3小节介绍了用户会话的定义，在 Android 系统下用户会话信息收集的难点，以及通过结合 SharedPreferences、快照和持久化消息队列的解决方法，用户会话信息收集是 Appetizer 客户端 SDK 的核心功能之一。3.3.4小节介绍了崩溃信息收集模块，主要包括崩溃时收集数据的方法，数据的具体内容和需要的系统权限，崩溃信息收集也是 Appetizer 客户端 SDK 的核心功能之一。3.3.5小节介绍了 Appetizer 客户端 SDK 实现 Android 应用程序未响应侦测的方法，还分析了该功能对 Android 应用程序性能的影响。3.3.6小节介绍了黑白屏时长记录的实现方法和向开发者提供的 API 的用法。

第四章　服务端设计与实现

4.1　服务端概要

Appetizer 服务端的作用主要有三个：

- 接收并存储集成了 Appetizer 客户端 SDK 的 Android 应用程序发送的数据。
- 计算分析接收的数据，得到更有价值的结果。
- 提供数据给开发者查看。

其中提供数据给开发者查看，是服务端提供查询 API，由另外的客户端调用查询 API 完成展现数据给开发者的功能，该部分与核心功能关系不大，本篇文章不讨论该部分。本章主要介绍 Appetizer 服务端如何处理接收集成了 Appetizer 客户端 SDK 的 Android 应用程序发来的数据，简要介绍对数据的处理方法。

4.2　服务端架构

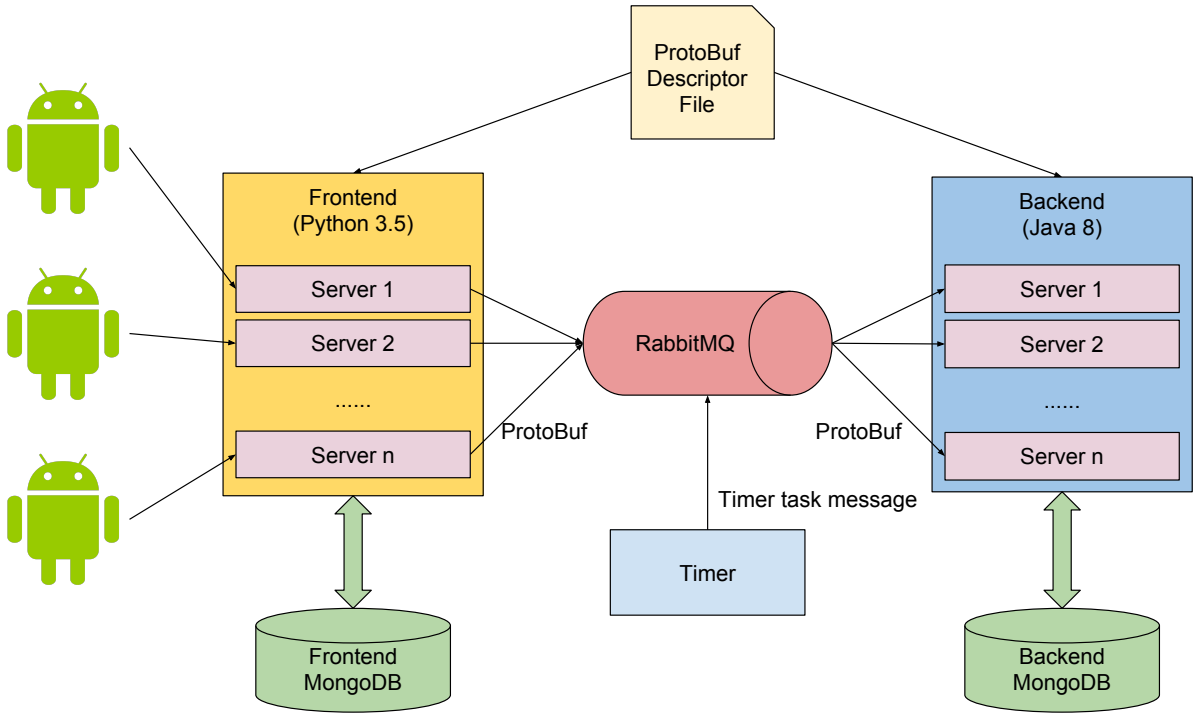

图 4–1 Appetizer 服务端架构

Fig 4–1 Appetizer server architecture

如图4–1所示是服务端架构，服务端的程序主要分为前台（Frontend）和后台（Backend）两部

分，其他重要的部分包括数据库（Database），前台后台的消息队列通信（MessageQueue）和定时器（Timer）。

前台、后台、数据库和消息队列四个部分采用的都是可扩展性良好的设计方案，并且结合每个部分处理的业务定制了专门的解决方案。

4.2.1 前台

前台的主要作用是完成输入输出（IO）密集型业务。包括接收集成了 Appetizer 客户端 SDK 的 Android 应用程序发送的数据，对数据做简单的处理转发给后台，提供查询 API 供面向开发者的客户端使用，前台系统大部分时间在处理存储、网络任务，只有少量计算型任务。

Appetize 服务端的前台（Frontend）部分使用 Python 3.5 开发，接收 Andorid 应用程序的数据部分基于 asyncio HTTP 实现。选择使用 Python 实现前台的首要原因 Python 是动态类型，提供原生 Map 类型便于操作 JSON 格式数据，可以有效提高前台业务的开发效率。Python 的劣势是性能，但是前台的业务偏向输入输出（IO）密集型而不是计算密集型，因此性能的劣势不会称为整个系统运行效率的瓶颈，除此之外 Python 3.5 新增的协程异步关键字能很方便的开发异步程序，适合输入输出密集型的程序。

前台的业务逻辑主要包括数据格式处理、时间校准、数据库存储和数据转发到后台，业务逻辑的任务流主要使用 Python 3.5 提供的 async IO 进行分配执行，数据库相关的操作使用第三方库 motor 实现，motor 是一个结合 async IO 和 MongoDB 的工具库。

整个前台程序，包括 HTTP 请求路由、业务逻辑执行、数据库操作都支持 Python 的异步协程（async）机制，便于写出支持多核的不阻塞异步业务逻辑代码。

4.2.2 后台

后台的主要作用是持久化存储数据和完成计算密集型业务。后台接收前台做过简单处理的数据，持久化存储数据，对数据做增量型数据统计处理，定时做计算量较大的数据处理业务，处理后的数据定时发送给前台共查询 API 调用。

Appetizer 服务端的后台（Backend）部分使用 Java 8 开发，主要考虑到 Java 较好的计算性能以及强大的工具类库，依赖的第三方库包括 ProtocolBuffer[19]、MongoDB Driver 和 RabbitMQ Client。后台框架将 RabbitMQ 通道传来的消息分配到不同的任务执行单元，框架自动将消息内的 ProtocolBuffer 数据根据描述文件转成 MongoDB 的 BSON（二进制 JSON）对象，然后进行增量处理或者定时的全量处理。

定时任务的触发由另一块单独的 Python 程序完成，定时器同样通过 RabbitMQ 发送消息触发 Appetizer 服务端 Java 后台程序的定时任务，复用 Java 后台的消息分配机制，并且 RabbitMQ 生产者消费者模型有断电容错恢复机制，定时器复用消息模型增强了定时任务的稳定性。

4.2.3 数据库

前台和后台的数据库使用的都是 MongoDB，因为 Appetizer 面向的数据都不是简单的用户信息，单个数据都涉及到较为复杂的多级结构，例如 Android 系统 ROM 构建信息本身还包含若干信息，直

接使用 JSON 格式存储该信息的内容比较合适，Android 系统 ROM 构建信息本身又是整个崩溃设备实时状态的一部分。因此存储单元为文档（Document）的 NoSQL 数据库 MongoDB 作为 Appetizer 服务端系统的数据库非常合适。

MongoDB 的单个文档（Document）最多存 4MB 的数据，建议低于 1MB，部分业务的数据可能大于 1MB，例如统计日活跃用户的用户列表信息对于百万级用户量以上的 Android 应用数据会大于 1MB，对于可能发生这种情况的任务，程序逻辑会对文档进行拆分。

4.2.4 通信

前台和后台之间的通信采用 RabbitMQ，支持多机与多机之间的生产者消费者模型消息传递。消息的内容使用 ProtocolBuffer 对象，前台和后台的数据模型可以使用同一份 ProtocolBuffer 描述文件，通过工具自动生成 Python 和 Java 的数据对象代码。为了避免写过多的后台 Java 模型类序列化和反序列化代码，后台部分实现了自动从 ProtocolBuffer 生成的 Java 模型类到 MongoDB 使用的 BSON 格式对象基础类型的转换。

采用生产者消费者消息模型的原因是让整个系统架构具有可伸缩性，即使某个时间段来自 Android 应用程序的消息数量过多，后台相比前台耗时更大，后台的任务不需要立刻反馈，无法处理的消息可以堆积在消息队列中，需要加大处理能力可以简单的增加机器，生产者消费者消息模型保证系统的稳定性和可伸缩性。

4.3 服务端实现

Appetizer 的服务端实现偏向具体业务，不是本篇文章介绍的重点，相比于 Appetizer 客户端 SDK 的实现内容篇幅更少，主要介绍核心部分。其中时间校准对应 Android 客户端 SDK 所有带有时间信息的数据，用户统计对应 Android 客户端 SDK 收集的用户会话信息，崩溃信息处理对应 Android 客户端 SDK 收集的应用崩溃信息和应用程序未响应信息。

4.3.1 时间校准

时间校准是前台业务非常重要的预处理部分。集成了 Appetizer 客户端 SDK 的 Android 应用程序收集的数据中的时间信息，如果从远端服务器获取精确的时间存在两个问题，首先是开销太大，所有包含时间的信息都要做至少一个网络输入输出（IO）操作，其次是不能保证所有设备都在网络通畅的环境，因此 Appetizer 客户端 SDK 收集的数据从远端服务器获取时间是不可取的，只能从本地设备获取。

虽然真实情况下大部分 Android 系统 ROM 都会自动从服务器校准时间，时间比较准确，但 Android 设备的时间可以由用户自己修改，所以从本地获取的时间对于服务端依然是不可信的，服务端所有的时间信息需要校准到服务器的本地时间，服务器的本地时间是可控的，认为是准确可信的。

Appetizer 时间校准的解决办法需要 Android 客户端 SDK 配合，对于所有涉及时间的信息，Appetizer 客户端 SDK 在发送到服务端之前会添加发送时的本地时间数据到发送的数据包中，Appetizer 服务端前台接收到数据时，服务端当前时间减去数据中发送时间得到差值，认为这个差值是服务端本地时间和 Android 设备本地时间的差值，再对数据中所有从 Android 设备获取的时间加上该差值，

这样就把所有在 Android 设备本地获取的时间校准到服务端的时间。时间校准是 Appetizer 服务端前台为数不多计算量稍大的任务。

该时间校准方法依然存在两个问题:

1. Android 应用程序发送数据中的发送时间和服务端接收数据的服务端本地时间不是物理世界中的同一时刻，存在网络延迟的时间间隔，该间隔的大小是不确定的。
2. Android 设备可能在一次信息收集过程中记录多个时间节点之间，修改设备本地时间。例如记录用户会话信息时在开始和结束之间修改设备本地时间。

问题 1 虽然会影响时间校准的准确性，但是在本篇文章介绍的系统中是可以容忍的。因为需要精确计时的业务都是计算时间间隔，时间校准的精确程度对于时间间隔没有影响。其他需要记录时刻的业务都不需要高精度的计时，可以容忍数秒的时刻偏移。

问题 2 如果要彻底解决，需要 Appetizer 客户端 SDK 监听设备本地时间修改事件，还需要额外的权限。考虑到问题 2 发生的概率较小，即使发生也不会造成很严重的影响，因此整个系统容忍问题 2 的存在。

4.3.2 用户统计

用户统计是能够依靠简单的数字直观展现 Android 应用程序用户活跃趋势的度量维度。从宏观上考虑，用户活跃度主要受应用体验和运营策略两个因素的影响，应用体验包括界面设计、应用卡顿、应用崩溃等因素，Appetizer 客户端 SDK 可以收集这些数据，开发和运营团队可以根据用户统计对应用界面和运营方式进行调整。

用户统计的业务实现主要包括两部分，实时用户使用量和以天为单位的日活跃用户（DAU)、日活跃新增用户（DNU)、日活跃老用户（DOU)。

实时使用量的统计难以做到精确，因为实时统计需要每台设备在使用的过程中能够顺利发送数据到 Appetizer 服务端，网络环境不允许的设备数据的实时性遭到了破坏，因此 Appetizer 服务端实现的实时使用量统计会比真正的实时使用量低。实时使用量的业务在后台实现，前台转发时间校准后的用户会话数据到后台，实时使用量不需要考虑同一个用户多次开启应用程序合并次数，后台在存储用户会话数据的同时进行增量统计，定时器间隔五分钟触发定时任务，后台将每个 Android 应用程序近 7 天的以小时为单位的应用使用量发送到前台，前台更新接收的数据到前台数据库，供查询 API 调用。

日活跃相关数据业务统计更为复杂，会话数据需要和设备信息进行关联。定时器每天触发日活跃相关数据任务一次，对于每个应用程序，后台程序对当日时间内的会话数据以设备 ID 为键进行聚合统计，得到该应用程序日活跃用户列表。Appetizer 对于“老用户”的定义为 7 天之内使用过应用程序的用户，因此日活跃新增用户（DNU）通过日活跃用户（DAU）列表减去之前 7 天的日活跃用户（DAU_last_7_days）列表计算得到。日活跃新增（DNU）用户通过日活跃用户（DAU）列表减去日活跃老用户（DOU）列表计算得到。三个日活跃相关数据计算得到后只发送用户数量给前台，因为前台只关注用户数量不需要具体的用户列表。

4.2.3小节介绍了 MongoDB 单个文档 (Document) 支持的最大空间为 4MB，用户统计的用户列表作为一个 List 对象存储在一个文档（Document）中，List 中的元素是设备 ID，因此当某个 Android 应用的日活跃用户超过一百万时，存储一百万个设备 ID 在一个 List 会造成整个文档的大小超过 4MB。

用户统计业务逻辑考虑到这个问题，对于用户量较多的单个 Android 应用程序进行日活跃度相关的计算时，单个文档过大会进行分文档存储。

4.3.3 崩溃信息处理

不同的 Android 设备发送的崩溃信息可能是由于同一个原因导致的，因此对崩溃信息做聚类可以更直观的展现给开发者查看，每个崩溃原因作为一个独立的事项，还包括该崩溃发生的设备系统分布、机型分布等有价值的信息，可以用来判断是否是设备或者系统的原因而不是程序本身造成的应用崩溃。

Android 应用崩溃根据函数调用栈的信息进行分类，从调用栈最底层开始到最顶层看做一个单项链表，链表节点内容为文件名、函数名、行号的元组。链表节点主要分为两类，一类是开发者写的 Android 应用程序部分的代码，特征为包名开头为项目的名字，可以直接通过包名进行区分，另一类是 Android 系统本身的代码、Android 支持库（support library）的代码和其他第三方库的代码，特征为包名开头是“android”、“java”或者项目目录 Manifest 里依赖的第三方库的包名。

两个崩溃信息根据满足以下规则 2 或规则 3 中的一个并且满足规则 1，则认为是同一个崩溃原因，否则认为是不同的崩溃原因：

1. 首先抛出的异常或者错误的类名相同。
2. 链表的第一个节点如果是开发者的代码，则从第一个节点依次到首个 Android 系统或者支持库的代码节点之前，节点内容全部相同。
3. 链表的第一个节点如果是 Android 系统或者支持库的代码，则从第一个节点依次到首个开发者的代码节点（包括首个开发者的代码节点），节点内容全部相同。

相同的崩溃原因造成的崩溃集合，称为一个崩溃项。Appetizer 服务端接收到 Android 设备发送的崩溃信息后，会做如下处理：

1. 找到该崩溃信息对应的 Android 应用程序。
2. 将崩溃信息的哈希值和该 Android 应用程序中已有崩溃信息的哈希值进行对比，如果存在相同的哈希值，将此次崩溃加入到该崩溃项的列表中，业务逻辑程序结束。
3. 根据上述的链表匹配规则和该 Android 应用程序中的每个崩溃信息链表进行对比，如果判断为同一个崩溃原因，将此次崩溃加入到该崩溃项的列表中，业务逻辑程序结束。
4. 创建新的崩溃项，将此次崩溃加入到新的崩溃项的列表中。

步骤 2 的哈希值相当于崩溃信息索引，通常情况下一个 Android 应用的大部分崩溃信息都是由少部分崩溃原因造成的，如果对于每个崩溃信息都进行链表匹配算法会造成巨大的开销，因此通过哈希索引崩溃信息可以大幅提高业务流程处理的的速度。

在崩溃信息加入到崩溃项中之后，会对该崩溃项的设备分布、版本分布统计信息做增量计算，可以给开发者提供每个崩溃项的设备分布统计和版本分布统计数据。

4.4 本章小结

本章内容主要介绍了 Appetizer 服务端的架构以及核心业务功能，服务端是整个系统不可或缺的一部分，用于存储、处理 Appetizer 客户端收集的数据，提炼出有价值的信息提供给开发者和运营团

队进行查看。

4.1节介绍了服务端收集 Android 设备发送的数据、对数据进行处理、提供给开发者和运营团队查看处理后数据的三个主要功能。

4.2节介绍了服务端整体架构，对各个组件的技术选型原因进行了分析。4.2.1小节介绍了前台系统的主要功能，解释了前台业务偏向输入输出（IO）密集型的特点，分析了选择 Python 实现前台系统的原因以及前台在实现上选择工具库做出的权衡。4.2.2小节介绍了后台系统的主要功能，后台业务偏向计算密集型的特点，说明了后台系统选择 Java 开发的原因，消息队列 RabbitMQ 在后台系统的作用和定时器的设计。4.2.3小节介绍了 MongoDB 的优势以及 Appetizer 服务端选择 MongoDB 作为前台和后台数据库的原因，还说明了可能需要拆分文档的应用场景。4.2.4小节介绍了 Appetizer 服务端前台和后台之间的通信机制，通信传递的内容采用 ProtocolBuffer 同时具备性能和便于开发的优势，以及相关通信架构提供的系统可伸缩性和可扩展性。

4.3节介绍了 Appetizer 服务端有展现价值的部分。4.3.1小节介绍了在 Appetizer 整个业务流程中发生的时间偏差的原因，Appetizer 客户端 SDK 只从本地设备获取不可信时间，设计并实现了依靠客户端 SDK 和服务端合作的时间校准方法，分析了该方法可能发生的问题，得到该时间校准方法有效可行的结论。4.3.2小节介绍了 Appetizer 核心功能用户统计的价值、作用和实现方法。4.3.3小节介绍了 Appetizer 另一个核心功能崩溃信息处理的设计实现，描述了一种对 Android 应用崩溃信息的原因进行分类的方法。

第五章　评估测试

评估测试部分主要涉及 Appetizer 客户端 SDK 在 Android 设备上的影响，因为对于使用 Appetizer 客户端 SDK 的开发者来说，他们关心的主要问题是 SDK 对于原来 Android 应用程序的性能影响、空间占用应用等，Appetizer 服务端是隐藏的黑盒。因此本篇文章介绍的 Appetizer 整套系统中，和同类产品相比核心竞争力之一是 Appetizer 客户端 SDK 的各项指标。

5.1　功能对比

Appetizer 在国内外的同类产品不少，这些产品在功能上互相借鉴又各有千秋。表5–1罗列了 Appetizer 和几个同类产品的客户端 SDK 功能，各 SDK 的具体版本为 Umeng v6.0.1，TalkingData v2.2.25，Flurry Analytics v6.3.1，Bugly v2.1.5。从对比中可以看出在客户端 SDK 功能上，Appetizer 收集的信息较为全面。

除了表中罗列的功能以外，其他产品还有各自的特色功能，例如 TalkingData 支持开发者在控制台上动态添加用户自定义事件，该功能不需要改变程序代码重新打包 Android apk 文件，即可让每台连接互联网的 Android 设备自动更新收集的信息。Flurry Analytics 在服务端上的功能非常出众，对展现给开发者和运营团队的报表有很好的数据处理和可自定义性。Bugly 收集了一些人工解决常见崩溃原因的方案库，在展现崩溃信息给开发者的同时还会提供相应的崩溃解决方案供开发者参考。

表 5–1 产品 SDK 功能列表

Table 5–1 Product SDK function list

	Appetizer	Umeng	TalkingData	Flurry Analytics	Bugly
Activity session	√	√	√	√	√
Fragment session	√	x	x	√	√
User defined event	√	√	√	√	√
Crash report	√	√	√	√	√
ANR	√	x	x	x	√
Launch time	√	x	x	x	x

5.2　SDK 空间占用对比

对于做用户数据收集的 Android SDK，集成到 Android 应用程序占用的额外空间同样是一个重要指标。因为作为第三方工具库，本身占用空间的增大会造成所有集成该工具库的占用空间变大，影响范围较广，因此第三方工具库的占用空间越小越好。

表5–2展示的是 Appetizer 和同类 Android SDK 占用空间的数据，表中列举的产品中 ACRA 的功能覆盖面最小，本文介绍的 Appetizer 客户端 SDK 在功能覆盖上是 ACRA 的超集，减少部分崩溃信

息可定制化的功能，增加了注入用户会话信息、Andorid 程序未响应侦测等功能，但是占用空间更小。其他产品在功能上和 Appetizer 相比各有千秋，占用空间都远超 Appetizer 客户端 SDK。

表 5–2 产品 SDK 占用空间表

Table 5–2 Product SDK size table

Product	Version	Size
Appetizer	v0.1	71 kB
Umeng	v6.0.1	301 kB
TalkingData	v2.2.25	386 kB
ACRA	v4.9.0	152 kB
Flurry Analytics	v6.3.1	260 kB

5.3 性能测试

性能测试部分主要计算 Appetizer 客户端 SDK 各个功能对原本的 Android 应用程序造成的额外开销，测试在不同设备不同系统上对比，测试在一个专门为测试编写的 Appetizer Test App 上进行。相关的测试设备信息如表5–3所示。

表 5–3 测试设备列表

Table 5–3 Test device list

	System version	ROM	Chipset
Nexus 5 (Device A)	4.4.2	CyanogenMod 11	Qualcomm Snapdragon 800
Nexus 5 (Device B)	6.0.0	Google ROM	Qualcomm Snapdragon 800
XiaoMi 4	6.0.1	MIUI 7.3.2.0	Qualcomm Snapdragon 801
Samsung Galaxy Note 4	5.1.1	Samsung ROM	Qualcomm Snapdragon 805
HUAWEI Ascend Mate7	4.4.2	EMUI 3.0	HiSilicon Kirin 925
OPPO R7s	4.4.4	ColorOS 2.1	Qualcomm Snapdragon 615

其中 Nexus 5 有两部，分别运行 Google 官方 ROM 的 Android 6.0.0 和 CyanogenMod 11，其他 4 部设备分别是小米 4，三星 Galaxy Note4，华为 Ascend Mate 7 和 OPPO R7s，都是几大 Android 手机厂商 2015 年到 2016 年之间在中国市场使用量较大的主流设备，这些设备对于 Appetizer 客户端 SDK 的测试具有代表性。

所有测试数据根据设备状况不同都会在一定区间内浮动，本节的数据取得是多次测试结果的平均值。

5.3.1 初始化开销

Appetizer 客户端 SDK 需要被集成的 Android 应用程序在启动时，调用 Application 类的 onCreate 方法的时候调用 Appetizer 的初始化方法，该初始化方法是 Appetizer 客户端 SDK 开启所有功能必须

执行的操作，该方法的开销直接影响到所有集成了 Appetizer 客户端 SDK 的 Android 应用程序的启动耗时。

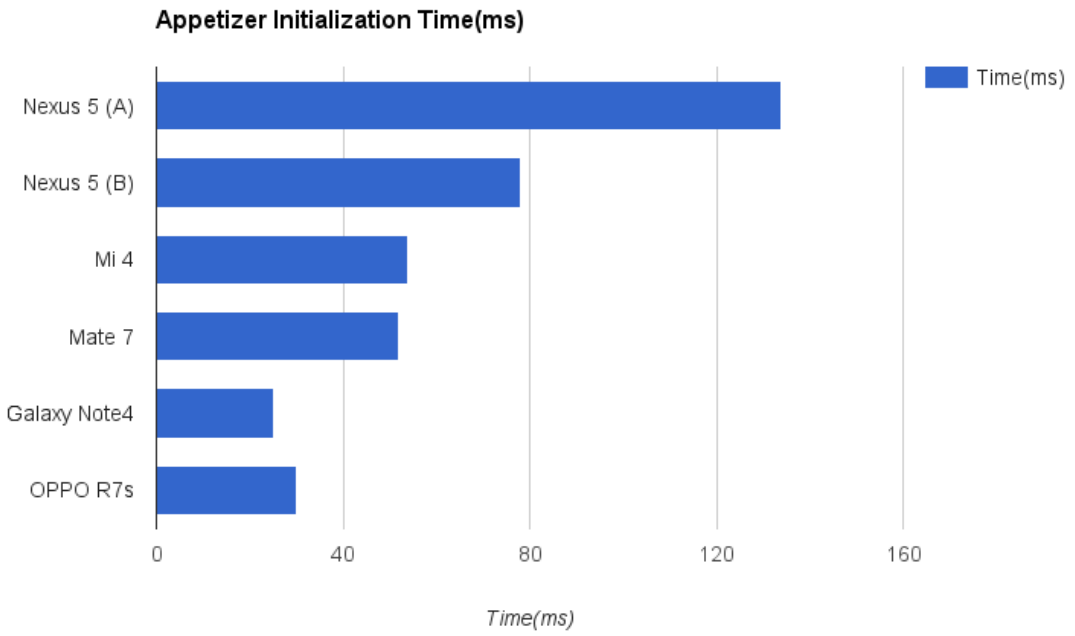

图 5–1 Appetizer 初始化时间

Fig 5–1 Appetizer intialization time

因为 Appetizer 初始化的速度也就是程序运行的速度，和各种软件、硬件相关，从表5–1中难以得到 Appetizer 初始化开销和某一类因素的相关性，但可以看出 Appetizer 初始化开销在主流设备和较新的 Android 系统上都不大于 100 毫秒，对程序启动时间的影响不大。

5.3.2 用户会话开销

用户会话主要在两种情况下存在开销，一种在用户切换页面的时候对会话信息的临时更新（保证崩溃可恢复），另一种是在用户结束会话的时候对完整会话的持久化存储。第二种情况通常在应用长时间运行在后台或者退出的时候发生，不会影响用户体验，因此只对第一种情况进行性能测试。

如表5–4所示是 Android 应用程序切换界面时，Appetizer 用户会话更新一次的时间开销，所有测试设备的时间均在 10 毫秒以内，可以认为 Appetizer 客户端 SDK 用户会话收集功能在 Android 应用程序中产生的额外开销，对用户体验不会造成影响。

5.3.3 ANR 侦测开销

Appetizer 的 Android 应用程序未响应（ANR）侦测功能，由一个 ANR 侦测线程每隔 5 秒发送 Runnable 到主线程（UI 线程）执行一段程序，因此在 Android 应用程序在前台运行时 ANR 侦测发送到主线程执行的程序运行频率较高，而且是在 UI 线程执行，如果占用时间过长会让用户感到卡顿，因此在各个设备上对 ANR 侦测的开销进行测试。

测试结果如表5–5所示，测试结果为数十次测试得到时间开销的平均值，不包含主线程对 Runnable 调度的开销。从结果可以看出 ANR 侦测 Runnable 的运行开销不超过 2 毫秒，因此 ANR 侦测对 Android 应用程序的额外开销可以忽略不计。

5.4 本章小节

本章内容主要介绍了 Appetizer 客户端 SDK 和同类产品的功能对比，以及自身性能和开销的测试。

5.1节对比了 Appetizer 同类产品友盟（Umeng）、TalkingData、雅虎公司的 Flurry Analytics 和腾讯公司的 Bugly 功能上的差异，展现了 Appetizer 客户端 SDK 在信息收集上覆盖了同类产品所没有的部分。

5.2节对比了 Appetizer 客户端 SDK 和同类产品 SDK 之间空间占用大小，Appetizer 客户端 SDK 在 Android 设备信息收集 SDK 中拥有占用空间最小的优势。

表 5–4 Appetizer 用户会话更新时间
Table 5–4 Appetizer app session update time

	Session update time (ms)
Nexus 5 (Device A)	3ms∼9ms
Nexus 5 (Device B)	1ms∼4ms
XiaoMi 4	1ms∼4ms
Samsung Galaxy Note 4	1ms∼6ms
HUAWEI Ascend Mate7	1ms∼4ms
OPPO R7s	1ms∼2ms

表 5–5 Appetizer ANR 侦测主线程开销
Table 5–5 Appetizer ANR detector main thread cost

	ANR Runnable time (ms)
Nexus 5 (Device A)	1ms∼2ms
Nexus 5 (Device B)	1ms∼2ms
XiaoMi 4	1ms
Samsung Galaxy Note 4	1ms∼2ms
HUAWEI Ascend Mate7	1ms
OPPO R7s	1ms∼2ms

5.3节对 Appetizer 客户端 SDK 的性能开销在不同系统和不同主流 Android 设备上进行了测试。5.3.1小节对 Appetizer 客户端初始化的开销进行了测试，主流 Android 设备在 Appetizer 客户端 SDK 初始化阶段耗时都低于 100 毫秒，对用户体验影响不大。5.3.2小节对 Appetizer 客户端 SDK 的用户会话更新功能进行了开销测试，该事件会在用户切换 Android 应用界面时发生，在主流 Android 设备上该功能的耗时都低于 10 毫秒，对用户体验没有影响。5.3.3小节对 Appetizer 客户端 SDK 的应用程序未响应（ANR）功能开销进行了测试，得到结果每隔 5 秒执行的功能额外开销对主线程的占用时间，在主流 Android 设备上不超过 2 毫秒，不会影响用户体验。

全文总结

本篇文章介绍了 Appetizer，是一个能够收集 Android 平台用户使用应用程序的行为、应用程序卡顿、应用程序溃信息的系统。Appetizer 包括一个能够集成在 Android 应用中的轻量级软件开发工具包，和一套具有良好可扩展性的用于接收并处理数据的服务端程序。

第一章绪论介绍了 Android 生态系统的构成，Android 生态系统中存在的系统碎片化和设备碎片化的问题，以及碎片化所带来的影响，分析了 Android 生态系统中开发者和用户之间存在使用体验反馈难的问题，解释用户使用信息对于 Android 应用程序开发和运营的价值。根据 Android 生态系统中存在的问题，提出了 Appetizer 的作用就是解决这些问题，同时罗列了相关同类产品。

第二章背景介绍了 Android 系统和应用的架构模型及平台特点，是理解 Appetizer 客户端 SDK 设计方案的背景知识。还介绍了 Appetizer 客户端 SDK 反馈信息的内容、原因和作用。

第三章是整篇文章的核心章节，该章描述了客户端 SDK 的架构、设计方案以及制定出最后设计的权衡过程，架构体现了 Appetizer 客户端 SDK 相比于同类产品更加轻量化的特点。该章着重介绍了 Appetizer 客户端 SDK 关键功能的实现方式，包括网络模块的选择，持久化队列的设计实现以及如何同时保证原子性、一致性和轻量化的，用户会话收集和崩溃信息收集的实现方式，应用程序未响应（ANR）的巧妙实现方式和黑白屏时长记录。

第四章介绍了接收 Appetizer 客户端 SDK 所收集信息的服务端，着重介绍了服务端的架构设计和方案选择过程中的权衡，分别对前台（Frontend）、后台（Backend）、数据库和前后台通信的技术方案选择设计与实现进行了详细介绍。业务部分，在时间校准上设计了一种适合 Appetizer 应用场景的解决方案，还介绍了用户统计的实现方法，以及对崩溃信息进行快速归类的算法。

第五章对 Appetizer 客户端 SDK 进行了测试，从功能上和 SDK 所占用的空间大小上和同类产品进行对比，结论是 Appetizer 客户端 SDK 收集的 Android 应用程序使用信息覆盖面较广，而且 SDK 占用空间明显小于同类产品，具有轻量化的优势。还从性能角度对 Appetizer 客户端 SDK 的初始化、用户会话收集、应用程序未响应（ANR）侦测功能进行了测试实验，在多个系统版本的主流 Android 设备上的实验结果表明，集成 Appetizer 客户端 SDK 对 Android 应用程序性能影响小到可以忽略，不会影响用户体验。

整篇文章从需求背景、技术背景、设计实现、功能对比和性能测试等多个角度介绍了 Appetizer，展现了集成 Appetizer 客户端 SDK 能够以较小的代价解决 Android 开发者的痛点，开发者可以从 Appetizer 服务端获取到提炼过的有价值的数据，并且 Appetizer 和同类产品相比在轻量化和信息收集覆盖面等方面具有一定优势。

在技术角度上，本文提出并介绍了基于 Android SharedPreferences 和文件系统，保证原子性和一致性的轻量级持久化队列，以及客户端 SDK 收集信息在服务端的时间校准方法，具有一定创新性和实用价值，可以供其他开发者进行参考。

参考文献

[1] *Half of YouTube's traffic is now from mobile* [EB/OL]. **2014-10-28**. http://www.cnbc.com/id/102128640.

[2] 谷歌宣布全球 *Android* 设备数量已达 *14* 亿 [EB/OL]. **2015-09-30**. http://news.yesky.com/248/97800248.shtml.

[3] 艾媒咨询：*2015-2016* 中国手机应用商店年度报告 [R/OL]. **2016-01-17**. http://www.iimedia.cn/40366.html.

[4] *Android Fragmentation Report August 2015* [R/OL]. **2015-08**. http://opensignal.com/reports/2015/08/android-fragmentation/.

[5] *Android SDK version market - AppBrain* [EB/OL]. **2016-05-23**. http://www.appbrain.com/stats/top-android-sdk-versions.

[6] *Dashboards - Android Developer* [EB/OL]. **2016-05-02**. https://developer.android.com/about/dashboards/index.html.

[7] *Bugly* [CP/OL]. http://bugly.qq.com/.

[8] *Umeng Analytics* [CP/OL]. http://www.umeng.com/analytics.

[9] *TalkingData* [CP/OL]. http://www.talkingdata.com/products.jsp.

[10] *FireBase* [CP/OL]. http://firebase.google.com.

[11] *Flurry Analytics* [CP/OL]. https://developer.yahoo.com/analytics/.

[12] *Fabric* [CP/OL]. https://get.fabric.io/.

[13] *ACRA* [CP/OL]. https://github.com/ACRA/acra.

[14] 黄文雄. "面向 *Android* 应用的用户行为分析方法" [J]. 软件, **2014**, *12*: 83–87.

[15] X. S. Nikravesh A Yao H. "*Mobilyzer: An open platform for controllable mobile network measurements*" [C]. *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services. ACM*, **2015**: 389–404.

[16] *Keeping Your App Responsive* [EB/OL]. https://developer.android.com/training/articles/perf-anr.html.

[17] *Thread starting during runtime shutdown* [CP/OL]. https://github.com/ACRA/acra/issues/136.

[18] *ANR-WatchDog* [CP/OL]. https://github.com/SalomonBrys/ANR-WatchDog.

[19] *ProtocolBuffers* [CP/OL]. https://developers.google.com/protocol-buffers.

致　谢

这篇论文的完成，为我的本科学习生活画上了句号。

首先要感谢我的母亲，对我从小到大的抚养和教育，以及在我本科前半段时间对我的经济支持。

感谢戚正伟老师给我在实验室进行学习深造的机会，以及在学术上对我的指导和帮助。

感谢夏鸣远博士在毕业设计上对我的指导和帮助，让我开阔了视野。感谢龚路学长在项目开发、系统设计和编码规范上对我的指导和帮助。

感谢室友和 Trusted Cloud 实验室的同学，在技术上的探讨和生活上的帮助。

感谢上海交通大学软件学院的王赪老师、肖凯老师、李垚学长、李强学长在我本科期间对我的项目指导，让我能在本科低年级的时候接触到复杂的项目开发。

感谢以上所有人，因为有你们我才能顺利完成本科学业。

ANDROID APPLICATION CRASH AND USER BEHAVIOR ANALYSIS

With the development of mobile networks, the mobile Internet connectivity becomes universe, which also motivates a great surge in the availability of various mobile devices, especially Android devices initially developed by Google. Nowadays, every Android device, like smartphone and tablet runs tens or even hundreds of mobile applications, also terms "App" in short. Though some these apps are prebuilt in the mobile operating system or provided by the device vendors, most of the installed apps are 3rd-party apps downloaded from online app market websites.

It is quite common that app developers want to retrieve feedbacks about their apps from users' devices, to improve app design, adjust operation strategies and improve user experience. This feedback phase serves as an important stage in mobile app developments.

Due to the openness nature of the Android ecosystem, there exist a large number of different Android devices with slight difference from the official distribution, also termed as the "Android fragmentation problem". App developers face the challenges to develop an app that can adjust to hundreds of different device models. Whether the app would crash on certain device at certain environment is a vital information for the developers to improve app quality.

This thesis introduces Appetizer, a system that collects app runtime behavior, crash and lag information. Appetizer serves as a lightweight development kit, being integrated into developers' apps. It also has a server side unit that process incoming data and render statistics for developers.

The design features, implementation and comparison with various off-the-shelf commercial solutions are presented in this thesis. The result showed that Appetizer cover wide functions in Android device information collecting, Appetizer SDK size is significantly smaller than commercial solutions, overhead won't decrease the user experience.

Typically, each smartphone running tens to hundreds of applications, shortly named App. Part of which is manufacturing equipment manufacturer or operating system manufacturer customize Android App, but the user spends more time playing with third-party application manufacturer or independent developers to develop and requires an Internet connection of App.

In the Android ecosystem, the developer after the completion of Android App development, such as through Google Play Store, Tencent application market, XiaoMi application market and other Android App Store application market release App, those application market will check developers' applications have submitted security, content and audit procedures on the stability, approved, users then download, install and use the application from App market.

Obtain feedback from the user, based on feedback information to improve App design and experience, adjust the way they operate is a very important aspect in the mobile Internet ecosystem, because the user experience a direct impact on the number of users of App.

Some user can grade and comment applications market reflected as feedback, but such feedback way only including little information, only a simple text description and evaluation scores,

can only play a reference role to other users, it is difficult to obtain more in-depth analysis of value data, the smaller the value of the development team. In addition, feedback from the Android application market has been subject to the application store, also exist malicious competition and other issues, what's more, the developing team can't fully control those feedbacks.

A major problem in Android operating system is the fragmentation of the various mobile phone manufacturers worldwide have countless different models of mobile phones running different versions of the Android operating system, these different phone hardware parameters, different system versions. Thanks for the "Open Mobile Alliance (Open Handset Alliance)", the existence of the provisions of the equipment running the Android system will need to meet certain compliance requirements, which makes the Android App can be run on most Android devices comply with norms, certain enhance the this loosely Android camp.

Android fragmentation problems including device fragmentation and system version fragmentation. Device fragmentation refers to the different types of equipment from different manufacturer, different hardware configurations. Such as Android phones use a variety of screens, these parameters include screen size, resolution, screen capacitive and resistive screen, support pressure, multi-touch support ceiling for different screen parameters for App experience is not same, in particular resolution directly related to the content displayed in App front end display how much contents.

In addition to the screen, CPU, GPU, memory, video cameras, and so many other hardware like flowers bloom together, the result is the same App runs on different types of Android devices, the effect is totally different, it's impossible to test an App is suitable for "all" devices.

System version fragmentation refers to different devices running different versions of the Android system, different versions of the system and the number provided by the API library implementation is not the same, larger changes in the underlying mechanisms of the system version number will be different for each Android App specify the range of the system can run major version number.

Android system version branch is very complex, the current situation is dominated by Google's "Android Open Source Project", shortly named AOSP, the branch is open to the open source community, and was sees as the purest Android branch, Android system is the major version number in the usual sense is followed by AOSP branch.

Google's Nexus series of devices running the Google Inc. combines AOSP and not open source Google Framework of ROM, commonly referred to as native Android ROM. But in China, the use of a wider range of users is the major manufacturers to modify AOSP based custom Android operating system, MIUI millet company is represented. App can run on most of the AOSP also able to run on these systems, but these systems in some places on the AOSP source code has been modified, the behavior is slightly different, these differences prone to unexpected problems.

For software developers, the App achieve collect usage from all user devices, you need to consider large-scale concurrent, persistent memory problems, stability, high degree of difficulty and workload. For large team, they may be able to achieve collect feedback function specially, but for personal developers who want to learn the status of users' behaviors, such a special function is too large, even heavier than the App major business function itself.

In addition, developers will have to face hardware fragmentation problem, running various versions of the system user equipment to ensure the development of the App does not crash on some special Android devices, and have a more positive user experience. During the testing phase

development team can't cover all of Android devices, what's more, the test is difficult to cover all situations.

There is a case, that an App can run on most devices can run correctly, only a section of users in a few Android devices will collapse, resulting in this part of the user can not use. If this device can send App crash information to the developer, the developer will be able to locate and solve the problem, and updating the App improve the compatibility and stability.

This article describes the tool, code-named "Appetizer", to solve the problems described above. "Appetizer" includes an Android application can be integrated in a lightweight software development kit, referred to as the client SDK, and has good scalability for receiving and processing data in the server program.

Appetizer client SDK can gather user using the operation time App paths and collected when App crash program call stack and device status information to detect Android application not respond, recording black and white screen every time App startup, storage those information, send the information to server when and network conditions allowed.

Appetizer server capable of storing persistent client message sent SDK for App use information, and make processing and statistical analysis, to calculate and show valuable content to the App development team and operations team, including daily active users, crash information classification, App device distribution and other processed data.

Appetizer client SDK is characterized as lightweight and stable, Appetizer client SDK don't use any third-party libraries, minimize SDK size, reduce the use of the SDK App size increase. For stability, will not cause a App crash problem because of the SDK itself crashes. Appetizer server architecture ensures good scalability, Appetizer server can withstand greater client pressure by simply increasing the number of servers.