



南京大學

研究生畢業論文

(申請工程碩士學位)

論 文 題 目 面向安卓应用的崩溃信息线上分析系统的设计与实现

作 者 姓 名 李秋婷

学 科、专 业 名 称 软件工程（软件工程领域）

研 究 方 向 软件工程

指 导 教 师 陈振宇 教授

2019 年 5 月 27 日

学 号 : MF1732072
论文答辩日期 : 2019 年 5 月 13 日
指 导 教 师 : (签字)



The Design and Implementation of Online Analysis System of Crash Information for Android Apps

By

Qiuting Li

Supervised by

Professor **Zhenyu Chen**

A Thesis

Submitted to the Software Institute

and the Graduate School

of Nanjing University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Engineering

Software Institute

May 2019

南京大学研究生毕业论文中文摘要首页用纸

毕业论文题目：面向安卓应用的崩溃信息线上分析系统的设计与实现

软件工程（软件工程领域） 专业 2017 级硕士生姓名：李秋婷

指导教师（姓名、职称）：陈振宇 教授

摘 要

近年来，随着移动互联网产业的持续高速增长，移动应用质量备受关注。移动应用更新频繁，使其难以全面测试因而引发质量问题。崩溃是移动应用最严重的一类质量问题。然而，移动端崩溃通常无法及时准确反馈给开发者，使得开发者错失最佳修复时间，从而导致用户流失和经济损失。

为了解决移动应用崩溃收集和分析中存在的问题，本文对移动应用中真实发生的崩溃进行了分析，并结合实际场景中的开发团队需求，设计并实现了一个面向安卓移动应用的崩溃线上分析系统。本系统包含三个模块：（1）崩溃收集模块部署于目标安卓应用，负责监控应用程序运行状态，采用面向切面编程技术AOP自动记录应用页面跟踪和崩溃截图，通过安卓异常处理机制UncaughtExceptionHandler捕获应用运行期间崩溃信息并上传服务器。（2）崩溃解析处理模块部署于线上服务器，对崩溃收集模块回传的崩溃数据进行解析、去重和分类。该模块通过模式匹配和不一致性分析的方法实现对崩溃数据的去重，并采用朴素贝叶斯和支持向量机分类算法实现对崩溃数据的分类。（3）崩溃可视化模块部署于线上服务器，该模块后台使用Flask框架，前端采用Antd框架，从堆栈信息、页面跟踪、机型设备、系统版本、应用版本等多个维度生成可视化崩溃报告。此外，该模块还提供实时数据分析功能来监测应用运行状态。

面向安卓应用的崩溃线上分析系统能够全面适配市场主流安卓应用类别。通过在20台安卓设备上测试10个不同开源应用的实验。实验结果表明，本系统能够自动捕获安卓应用崩溃，对崩溃信息有效进行分类、去重和可视化，其分类正确率为88.1%，去重率为60.7%。综上所述，本系统为安卓应用的开发和测试提供了一站式崩溃监控和分析，弥补了现有崩溃收集分析手段存在的不足，全方位提升移动应用质量。

关键词：崩溃捕获，页面跟踪，模式匹配，崩溃分类，数据可视化

南京大学研究生毕业论文英文摘要首页用纸

THESIS: The Design and Implementation of Online Analysis System of
Crash Information for Android Apps

SPECIALIZATION: Software Engineering

POSTGRADUATE: Qiuting Li

MENTOR: Professor Zhenyu Chen

Abstract

In recent years, as mobile Internet industry continues to grow rapidly, much attention has been paid to the quality of mobile applications. The frequent update of mobile apps makes it difficult to fully test the apps and in turn causes quality issues. Crash is one of the most serious quality issues in mobile apps. However, crashes on mobile endpoints often fail to provide accurate feedbacks to developers in a timely manner. As a result, developers may miss the best repair time, resulting in the loss of users and economic loss.

In order to address the above-mentioned issues in the process of mobile application crash collection and analysis, this thesis analyzes the real crash instances in mobile applications, and further designs and implements the crash online analysis system for Android mobile applications that integrates the requirements of Android developers. The system consists of three modules: (1) The crash collection module is deployed in the target Android application, for the purpose of monitoring the running state of the application. This module automatically records the application page tracking data and crash screenshots using AOP. In addition, the crash information of application runtime exception is captured by the Android exception handling mechanism (UncaughtExceptionHandler). (2) The crash processing module is deployed on the online server to parse, deduplicate, and classify the crash data collected by the crash collection module. This module implements crash deduplication functionality by employing pattern matching and inconsistency analysis method. This module uses naive Bayes and support vector machine, two classification algorithms that are suitable for text classification, to implement the classification of crash data. (3) The crash visualization

module is deployed on the online server. This module uses Flask framework to implement back-end functionalities. The front-end uses Antd framework to generate visual crash reports from multiple dimensions, such as stack information, page tracking, device model, system version, and application version. In addition, this module supports real-time data analysis for monitoring application health status.

The online crash analysis system fully adapts to the mainstream categories of Android applications in the market. Extensive experiments have been performed for 10 different open-source applications 20 Android devices. Experimental results demonstrate that, the system implemented in this thesis not only can automatically capture Android application crashes, but also can effectively classify, deduplicate and visualize crash information. The classification accuracy rate achieves 88.1%, and deduplication rate achieves 60.7%. To summarize, this system provides an one-stop crash monitoring and analysis solution for the development and testing of Android applications that overcomes the shortcomings of existing crash collection and analysis methods, and improves the quality of mobile applications from all perspectives.

Keywords: Crash Capturing, Page Tracking, Pattern Matching, Crash Classification, Data Visualization

目 录

表目录	ix
图目录	xii
第一章 引言	1
1.1 研究的背景与意义	1
1.2 国内外研究现状	2
1.3 本文主要研究工作	3
1.4 本文的组织结构	4
第二章 相关技术综述	7
2.1 崩溃跟踪相关技术	7
2.1.1 异常处理机制	7
2.1.2 切面编程技术	8
2.2 分类算法	8
2.2.1 朴素贝叶斯	9
2.2.2 支持向量机	9
2.3 系统核心技术栈	10
2.3.1 RESTful架构	10
2.3.2 Flask框架	11
2.3.3 SQLAlchemy工具	13
2.3.4 Ant Design React框架	14
2.4 本章小结	15
第三章 面向安卓应用的崩溃信息线上分析系统分析与设计	17
3.1 系统需求分析	17
3.1.1 功能性需求	17
3.1.2 用例描述	19

3.1.3	非功能性需求	22
3.2	系统架构设计	23
3.3	崩溃收集模块设计	27
3.3.1	监听Activity生命周期设计	28
3.3.2	安卓全局崩溃捕获设计	30
3.4	崩溃解析处理模块设计	32
3.4.1	崩溃去重模块设计	34
3.4.2	崩溃分类模块设计	36
3.5	崩溃可视化模块设计	38
3.5.1	实时监控分析设计	40
3.5.2	崩溃报告设计	40
3.5.3	产品管理设计	41
3.6	数据库设计	41
3.7	本章小结	44
第四章	面向安卓应用的崩溃信息线上分析系统实现与测试	45
4.1	崩溃收集模块的实现	45
4.1.1	监听Activity生命周期	45
4.1.2	安卓全局崩溃捕获	47
4.2	崩溃解析处理模块的实现	49
4.2.1	崩溃去重实现	49
4.2.2	崩溃分类实现	50
4.3	崩溃可视化模块的实现	52
4.3.1	实时监控分析	52
4.3.2	崩溃列表	54
4.3.3	崩溃详情	55
4.3.4	产品管理	58
4.4	系统测试与分析	59
4.4.1	测试环境	59
4.4.2	功能测试	60
4.4.3	性能测试	62

4.4.4 实验分析	63
4.5 本章小结	66
第五章 总结与展望	67
5.1 总结	67
5.2 展望	68
参考文献	69
简历与科研成果	73
致谢	75
版权及论文原创性说明	77

表 目 录

2.1	上下文对象	13
3.1	需求列表	18
3.2	产品管理用例描述	20
3.3	SDK集成配置用例描述	20
3.4	实时监控分析用例描述	21
3.5	崩溃列表用例描述	21
3.6	崩溃详情用例描述	22
3.7	产品信息表	43
3.8	崩溃信息表	43
3.9	崩溃去重表	44
4.1	监听Activity生命周期类表	45
4.2	安卓全局崩溃捕获函数	47
4.3	收集的崩溃信息	48
4.4	部分崩溃的类别与解决方案	52
4.5	崩溃详情接口表	56
4.6	产品管理接口表	59
4.7	崩溃收集与处理测试用例	60
4.8	实时监控分析测试用例	60
4.9	崩溃列表测试用例	61
4.10	崩溃详情测试用例	61
4.11	产品管理测试用例	62
4.12	性能压测结果表	63
4.13	崩溃收集类型表	65

图 目 录

2.1	Flask请求处理流程图	12
3.1	系统用例图	19
3.2	系统整体架构设计图	24
3.3	系统逻辑视图	25
3.4	系统进程视图	25
3.5	系统开发视图	26
3.6	系统物理视图	27
3.7	崩溃收集模块功能图	28
3.8	安卓Activity生命周期图	29
3.9	监听Activity生命周期类图	30
3.10	安卓全局崩溃捕获类图	31
3.11	SDK重传崩溃流程图	32
3.12	崩溃解析处理模块功能图	33
3.13	崩溃解析处理模块时序图	33
3.14	堆栈信息结构图	34
3.15	堆栈信息模式匹配流程图	35
3.16	崩溃去重设计流程图	36
3.17	崩溃分类设计流程图	37
3.18	崩溃可视化模块功能图	38
3.19	崩溃可视化模块时序图	39
3.20	E-R图	42
4.1	替换Instrumentation类实现关键代码	46
4.2	监听Activity生命周期Java风格伪代码	46
4.3	获取内存占用率关键代码	49
4.4	堆栈信息模式匹配代码	50
4.5	崩溃分类实现的关键代码	51

4.6	今日问题统计接口实现的Python风格伪代码	53
4.7	崩溃上报趋势接口实现伪代码	53
4.8	实时监控分析界面	54
4.9	崩溃列表接口实现伪代码	54
4.10	崩溃列表界面	55
4.11	页面分布接口实现伪代码	56
4.12	页面分布界面	57
4.13	简要详情界面	57
4.14	机型设备分布界面	58
4.15	错误日志详情界面	58
4.16	产品修改界面	59
4.17	JMeter压力测试汇总报告	62
4.18	不同应用在多个设备上的崩溃发生总次数	64
4.19	不同设备上多个应用的崩溃发生总次数	64

第一章 引言

1.1 研究的背景与意义

随着移动互联网产业的持续高速增长，移动终端普及率越来越高。数据统计公司Statista¹发布的全球移动终端操作系统市场份额数据表明，2018年第一季度，安卓系统市场份额高达88%。作为用户最多的操作系统，安卓平台上每天都有成千上万的新应用发布，其应用总数也早已到达百万级别。用户期望在移动端设备获得与PC端相同的使用体验，更快更顺畅的反应和更多样性的交互方式[1]。移动应用具有开发周期短、迭代频率高的特性，这使得如何保证安卓移动应用的软件质量成为测试和开发人员面临的新挑战[2]。其中，安卓碎片化是一个棘手的问题[3]。国内的安卓生态环境较为独特，各大手机厂商对安卓系统进行深度定制，缺少统一的设备软硬件标准，且没有统一的应用发布平台，各大厂商各自为战，使得安卓碎片化问题尤为严重。在这种情况下，兼容不同厂商的安卓系统版本和设备型号对移动应用质量无疑有着严苛的要求。

大多数商业应用在发布前都会经过严格的测试流程，测试阶段检测到的缺陷被修复后才会将应用发布上线，然而未检测的缺陷会被直接暴露给用户。其中，崩溃是应用缺陷中严重等级最高的一类，通常会直接导致用户流失。据统计，崩溃首次在使用过程中发生时，大约21%的用户会卸载该应用，70%的用户会选择给应用差评²。对于开发者来说，用户使用应用过程中发生的崩溃是非常隐蔽且不易复现的。大部分移动应用开发者对于崩溃的跟踪通常源于小部分核心用户的反馈。用户上报缺陷给客服，客服对用户反馈结果进行整合后反馈给测试和开发人员。同时由于用户专业水平不一，无法对缺陷准确描述，且不能记录缺陷发生时的log日志和相关调试信息，导致测试人员复现缺陷和开发人员定位及修复缺陷都需要较大的成本。小部分开发者为了获取更为详细的崩溃信息会选择在应用程序中嵌入一部分异常处理代码，用于将应用产生的崩溃上传到线上服务器，但这无疑增加了许多额外的开发成本，且对于不同的应用还需要重新编写异常处理代码，导致开发者在应用的非功能需求上投入了大量精力。因此，通过崩溃监控分析工具实时监测上线应用运行过程中的崩溃和闪退等异常，以协助开发者分析崩溃原因和定位崩溃位置已成为移动应用测试中的一种非常重要方式[4]。

¹<https://www.statista.com/>

²<http://crash.163.com/>

本文面向安卓应用，搭建崩溃信息线上分析系统。安卓应用通过指定SDK(Software Development Kit)接入系统，系统自动捕获应用运行时发生的崩溃，并收集堆栈信息、设备软硬件信息、用户页面路径跟踪和崩溃截图等崩溃相关信息上报至服务器。线上分析系统将利用模式匹配和不一致性分析实现崩溃去重，使用朴素贝叶斯[5]和支持向量机[6]算法实现崩溃分类，并对处理后的数据进行统计分析，从页面分布、机型设备分布、系统版本分布、崩溃上报趋势、详细堆栈、用户页面路径跟踪等多个维度展示崩溃报告，帮助安卓开发团队快速准确地定位和修复崩溃。此外，本系统还提供实时监控分析服务，以帮助安卓开发团队实时掌握安卓应用的崩溃分布情况和整体崩溃趋势，全方位提升安卓应用质量。

1.2 国内外研究现状

移动应用崩溃分析相关领域一直是学术界的研究热点。Kim等[7]发现，单一的崩溃信息展示导致了开发者修复过程中的大量开销，他们提出了一种聚合崩溃视图的概念，通过聚合多个崩溃，减少了误分类。经评估，聚合崩溃视图能够有效地帮助开发者识别可进行修复的崩溃。White等[8]提出的CrashDroid工具通过将崩溃异常堆栈信息转换为形象的操作步骤从而实现了崩溃复现，以帮助开发者收集和重现崩溃。然而该方法无法自动发现崩溃，而是依赖于预先存在的崩溃堆栈轨迹，且需要真实用户手动操作路径集合，使用自然语言描述进行标记来构建错误报告和可重放的方案。Shahriar等[9]提出内存泄漏是导致应用崩溃的原因之一，他们基于特定于安卓应用程序的常见内存泄漏模式生成测试用例，进而模拟内存泄漏。但是该方案只能发现因内存泄漏引发的应用崩溃，不能分析因其他原因导致的崩溃问题。AppDoctor[10]采用“近似执行”方法，通过对应用程序GUI组件的时间执行Hook操作来处理程序。虽然这种方法提供了崩溃重放功能、异常堆栈轨迹等功能，但它必须重放崩溃轨迹以排除误报，并且不能提供便于测试和开发人员理解的高度详细和富有表现力的崩溃报告。CrashScope[11]根据静态和动态分析来用系统的策略探索给定的安卓应用程序，其内在目标是触发崩溃。当检测到崩溃时，CrashScope会生成一个崩溃报告，其中包含屏幕截图、详细的崩溃再现步骤、捕获的崩溃堆栈轨迹以及可自动重现目标设备崩溃的可重放脚本。杨高翔[12]针对安卓应用实现了一个崩溃信息的收集与处理系统，通过嵌入式SDK对崩溃信息进行自动收集，利用聚类算法对崩溃信息进行自动分组，帮助移动开发团队清晰地定位移动应用中Bug的类型。但是该平台仅对应用的崩溃信息进行了收集和统计分析，未实现用户页面路径跟踪和进一步为开发者提供崩溃解决建议。

研究发现，传统的移动性能管理方案和软件性能检测技术存在其局限性[13]，如无法复现、无法覆盖所有分支，无法对用户体验的数据进行收集和分析等。对此，如何建立完善的应用性能管理方案(Application Performance Management, APM)已成为各大企业及应用开发者最为关注的问题。APM被定义为使用相关工具来管理应用程序的性能，实现监测用户体验、提高测试准确性和快速识别应用性能瓶颈的功能[14]。为完善移动应用的性能管理服务，工业界面向移动应用实际场景中的崩溃推出了一大批相应的平台和框架。除运营统计等服务外，最强大功能就是崩溃监控分析，用于实时监测已上线移动应用使用过程中的崩溃，并对崩溃进行统计、分析及处理，展示详细的崩溃报告，以协助测试和开发人员快速复现、定位及修复崩溃。

在国外，Bugsnag³是一款简单、易用的实时自动报告未处理的崩溃检测工具，Bugsnag最大的优势是可以对应用程序进行全方位的检测，包括后端和客户端的所有开发堆栈错误，并针对这些错误提供详细崩溃报告。Crashlytics⁴成立于2011年，是一款用于保存和分析移动应用崩溃信息的工具，该工具可帮助管理崩溃日志和提示各类有助于诊断的信息，但是Crashlytics服务器在国外，以致国内开发者使用时访问速度慢，且易丢失崩溃数据。

国内比较出名的平台有Mobile Insight⁵、友盟⁶及Bugly⁷。Mobile Insight是OneAPM公司推出的以多维度分析移动应用性能及用户体验的性能检测管理系统，具有崩溃检测和分析功能，可生成详细的崩溃报告。友盟为移动开发者提供专业的数据统计分析、开发和运营等服务，但是友盟并未完善质量跟踪，缺少用户页面路径跟踪功能。同时，友盟崩溃数据的上传在应用程序二次启动时进行，若用户直接卸载应用程序则会导致崩溃数据丢失且崩溃发生时间统计不准确等问题。Bugly是腾讯推出的质量跟踪平台，提供专业的崩溃监控分析、崩溃去重、崩溃分布、缺陷代码级定位以及详细的崩溃报告等。但Bugly未提供用户提交个人解决方案的接口，无法收集安卓开发团队的反馈。

1.3 本文主要研究工作

为了解决上线后的安卓应用发生崩溃时，安卓应用开发团队难以复现和定位Bug的问题。本文深入研究了安卓异常处理机制和切面编程技术的原理，设计并实现了动态崩溃捕获技术及相关信息收集，并结合安卓应用开发团队的需

³<https://www.bugsnag.com/>

⁴<http://try.crashlytics.com/>

⁵<https://www.oneapm.com/>

⁶<https://developer.umeng.com/>

⁷<https://bugly.qq.com/v2/>

求，开发了面向安卓应用的崩溃信息线上分析系统。本文主要研究工作体现在以下几个方面：

(1) 基于对崩溃线上分析系统用户的详细需求分析，采用结构化分析方法设计并实现了由崩溃收集模块、解析处理模块以及可视化模块构成的崩溃分析系统，满足了用户实时把握应用质量，及时针对性修复崩溃的要求。

(2) 设计并实现了安卓应用运行时崩溃捕获及相关信息的收集。通过安卓异常处理机制和监听Activity生命周期[15]实现了应用崩溃捕获、设备软硬件信息收集、用户页面路径跟踪和崩溃截图获取。

(3) 设计并实现了崩溃去重和分类方法。根据异常堆栈的结构化和多样性的特性，对大量的堆栈信息进行分析，提出了基于模式匹配提取和不一致性分析的崩溃去重方法。崩溃分类方法基于预处理数据，利用哈希表将数据向量化，结合朴素贝叶斯和支持向量机分类算法训练模型并预测崩溃类别。同时，将崩溃类别与解决方案一一对应，减少崩溃的定位和修复时间。

(4) 设计并实现了以产品管理、崩溃列表、崩溃详情及实时监控分析为一体的崩溃可视化模块。该模块从页面分布、系统版本分布、机型设备分布、详细堆栈、用户页面路径跟踪等多个维度展示崩溃报告，以全方位分析和定位崩溃。此外，该模块提供了解决方案的上报入口来收集开发者在各崩溃问题上的解决方案，以解决由系统提供的崩溃解决方案的不完善性问题。

综合上述系统设计与实现，本系统采用前后端分离架构进行开发，前端使用TypeScript[16]强类型语言并基于Ant Design React框架实现，系统后台使用Python开发语言并基于Flask[17]框架实现，前后端交互通过Nginx[18]代理转发分发至相应RESTful[19]接口完成。

1.4 本文的组织结构

本文的组织结构如下：

第一章 引言部分。本章首先对课题研究的背景和意义进行了综述，然后介绍了国内外在崩溃分析与处理方面的研究现状及相关平台，最后介绍了本文的主要研究工作和论文组织结构。

第二章 相关技术综述。本章介绍了系统实现所涉及的技术难点、主要框架及相关技术。主要包括异常处理机制、面向切面编程、分类算法、Flask框架、SQLAlchemy、Ant Design React等。

第三章 面向安卓应用的崩溃信息线上分析系统分析与设计。本章首先从安卓开发团队角度明确了系统的功能性需求和非功能性需求，进而详细设计了

系统架构并对各个功能模块进行了划分。然后对各个功能模块的具体设计进行了阐述，最后详细描述了系统的数据库设计。

第四章 面向安卓应用的崩溃信息线上分析系统实现与测试。在系统分析与设计的基础上，本章重点阐述了崩溃收集模块、崩溃解析处理模块及崩溃可视化模块的实现细节，并展示了相应的关键代码、接口实现和界面截图。最后，对系统进行了功能测试和性能测试，并设计了相关实验来验证系统的可用性和可靠性。

第五章 总结与展望。本章总结了论文的总体内容和主要工作，就面向安卓应用的崩溃信息线上分析系统的未来扩展作了进一步展望。

第二章 相关技术综述

本章主要对系统设计与实现过程中使用的关键技术进行介绍，将分别从崩溃跟踪相关技术、分类算法及系统核心技术栈三个方面进行阐述，主要包括异常处理机制、切面编程技术、朴素贝叶斯、支持向量机、RESTful架构、Flask框架、SQLAlchemy工具以及Ant Design React框架等内容。

2.1 崩溃跟踪相关技术

在安卓应用中，应用程序运行时崩溃通常由安卓异常处理机制来捕获。为获悉更多与用户行为相关的信息，可通过切面编程技术监听Activity生命周期，以跟踪用户行为路径和获取崩溃截图。下文将分别对两种技术进行阐述。

2.1.1 异常处理机制

异常是指程序运行期间出现的不正常或背离正常程序流程的错误。为了及时有效地处理程序中的运行错误，Java语言引入了一套完备的异常处理机制[20]。Exception和Error都属于Throwable类的子类，Exception属于可控制的异常，应被应用程序级处理；Error属于不可控制异常，是应用程序难以处理或无法处理的错误，通常与系统或JVM出现错误有关，标志着严重的系统错误。Java Thread类中定义了一个UncaughtExceptionHandler接口，用于处理导致线程终止的未捕获的崩溃。Thread类中存在两个异常处理器。一个是静态的defaultUncaughtExceptionHandler，另一个是非静态UncaughtExceptionHandler。defaultUncaughtExceptionHandler指的是在进程fork时为所有进程设置一个静态默认的UncaughtExceptionHandler，所有线程的未捕获异常都会由该异常处理器进行处理，管辖范围较大。UncaughtExceptionHandler指的是单独为某个线程设置一个异常处理器，该异常处理器只能处理当前线程的未捕获异常，管辖范围较小。因此，可以为不同的线程设置不同的UncaughtExceptionHandler，在线程终止时执行不同的操作。

安卓应用基于Java语言编写，因此安卓应用异常的处理也是基于Java异常处理机制来实现，即通过继承Thread.UncaughtExceptionHandler自定义异常处理器，使用Thread.setDefaultUncaughtExceptionHandler将自定义异常处理器设置为全局默认的崩溃处理器，这样系统中所有未设置崩溃处理器的线程将使用这个默认的崩溃处理器进行处理。UncaughtExceptionHandler存在于线程中，当

崩溃发生且未被捕获时，崩溃会通过UncaughtExceptionHandler抛出，并且该线程会终止。所以，在安卓中子线程的死亡是允许的，但是主线程死亡就会导致ANR(Application Not Responding)。

本文通过继承UncaughtExceptionHandler构造自定义异常处理器，实现了自动捕获用户使用过程中的应用崩溃，同时收集异常堆栈、设备硬件和设备性能状态等信息，将其保存到本机设备或上传到服务器等功能。

2.1.2 切面编程技术

面向切面的编程(Asspect-Oriented Programming, AOP)指的是通过在预编译或运行时，在不修改源代码的基础上，动态地将代码切入到类的指定方法或指定位置来为统一为程序添加功能的一种技术[21]。传统的面向对象编程(Object Oriented Programming, OOP)是根据需求功能的不同将其划分为相对独立且封装良好的类，定义对象的属性和行为，依靠继承和多态来定义彼此的关系，其特点是封装、继承与多态[22]。AOP是一种新的方法论，也是对传统OOP编程的一种补充，主要作用是将与业务不相关的功能抽离出来，以动态插入的方式嵌入多个类中，使得多个类共享同一行为，当功能需求发生改变，无需修改多个类，只需修改这个行为即可。同时，AOP支持静态织入，即在编译期间引入特定的语法来织入有关切面代码。AOP面向切面编程中的重要概念包括切面、关注点、横切关注点、连接点、切入点、目标对象、代理对象、通知、引入、编织等[23]。使用AOP能够降低系统的重复代码，降低模块间的耦合度。

在Java语言中，AOP建立在反射机制与动态代理机制之上，其基本原理为业务逻辑组件在运行时会动态生成一个代理对象供使用者调用，该代理对象已经将切面成功切入到目标方法的连接点上，从而达到切面功能与业务逻辑功能同时执行的目的。反射机制主要是指程序可以访问、检测和修改它本身状态或行为的一种能力。动态代理机制是在实现阶段不用手动实现和维护代理类，而是在程序运行阶段指定需要代理的特定对象，并利用反射机制为其动态生成代理类，在使用后会自动销毁，避免代理类角色的冗余。

本文使用面向切面编程技术，基于Java语言反射机制，Hook安卓框架中Instrumentation类，实现对Activity生命周期的监听，由此跟踪用户页面路径和获取页面崩溃截图。

2.2 分类算法

机器学习中常见的分类算法包括决策树[24]、随机森林[25]、KNN[26]、支持向量机、AdaBoost[27]、朴素贝叶斯等。其中，支持向量机和朴素贝叶斯都

适用于文本分类。由于本文待分类数据是以文本信息为主的安卓异常堆栈，故选用朴素贝叶斯和SVM支持向量机作为本文的分类算法实现。因此，本文基于sklearn包中这两个算法的实现，使用朴素贝叶斯MultinomialNB模型[28]和支持向量机算法中由一对一法组合的SVC分类模型来实现崩溃数据的分类。以下将针对这两个算法的分类原理进行阐述。

2.2.1 朴素贝叶斯

朴素贝叶斯(Naive Bayesian, NB)是以特征条件独立假设和贝叶斯定理为基础的简单概率分类器。其中，贝叶斯定理公式为：

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)} \quad (2.1)$$

基于上述公式，对于训练集中的数据，其中c是类别，x是特征向量。假设每个特征属性独立，可以求得：

$$P(c|x) = \frac{P(c) \prod_{r=1}^d P(x_r|c)}{P(x)} \quad (2.2)$$

由于分母对于所有类别为常数，因此可直接分子最大化，即计算所有类别的条件概率，选取条件概率最大的类别作为预测类别，最终推出朴素贝叶斯分类器的分类公式如下式所示：

$$f(x) = \max \left(P(c) \prod_{r=1}^d P(x_r|c) \right) \quad (2.3)$$

朴素贝叶斯分为多项式模型(Multinomial model)、伯努利模型(Bernoulli model)以及高斯模型(Gaussian model)。高斯模型适用于连续的特征，多项式模型和伯努利模型适用于离散的特征。多项式模型和伯努利模型都常用于文本分类，不同的是多项式模型的特征是单词，而伯努利模型的特征是文件且每个特征的取值是布尔型。

2.2.2 支持向量机

支持向量机(Support Vector Machine, SVM)是一种基本模型定义为特征空间上的间隔最大的线性分类器，其学习策略是间隔最大化，可形式化转化为一个凸二次规划问题的求解。其二分类基本原理为：

对已知样本 (x_i, y_i) , $y_i = \pm 1$, 利用支持向量机寻求最优分类平面, 应满足下列表达式:

$$w \cdot x + b = 0 \quad (2.4)$$

那么分类间隔为 $\frac{\|w\|}{2}$, 为了寻求分类间隔最大, 相当于求 $\frac{\|w\|^2}{2}$ 的最小值, 即满足 $\|w\|$ 最小的分类线就是最优分类线。同时, 训练集中的样本应满足:

$$y_i(w \cdot x_i + b) - 1 \geq 0 \quad (2.5)$$

在以上两个约束条件的限制下, 利用拉格朗日对偶性将其转化为二次回归问题, 其中 C 为惩罚因子:

$$\min\left(\frac{\|w\|^2}{2} + C\left(\sum_{r=1}^n \xi_i\right)\right) \quad (2.6)$$

$$s.t. \begin{cases} y_i(w \cdot x_i + b) \geq 1 - \zeta_i \\ \zeta_i \geq 0 \end{cases}, i = 1, 2, \dots, n \quad (2.7)$$

最后, 推导出支持向量的线性判别函数:

$$F(x) = \operatorname{sgn}\left(\sum_{r=1}^n a_r y_r x_i + b\right) \quad (2.8)$$

对于非线性分类问题, 引入核函数 $k(x_i, y_i)$, 可以得到:

$$F(x) = \operatorname{sgn}\left(\sum_{r=1}^n a_r y_r k(x_i, y_i) + b\right) \quad (2.9)$$

支持向量机最初为二分类而设计, 处理多分类问题时, 可通过组合多个二分类器来实现, 组合法主要包括一对多法和一对一法。其中, 一对一法即在任意两类样本之间设计一个SVM, 这样 k 个类别的样本就需设计 $k(k-1)/2$ 个SVM。当对一个未知样本进行分类时, 对各个子分类器进行测试, 其中得票最多的类别即为该未知样本的类别。

2.3 系统核心技术栈

2.3.1 RESTful架构

REST即Representational State Transfer的缩写, 可译为“表现层状态转化”。RESTful架构风格最初由Roy T. Fielding在其2000年的博士学位论文中提出,

HTTP就是该架构风格的一个典型应用。从其诞生之日开始就因其可扩展性和简单性受到越来越多的架构师和开发者们的青睐。REST定义了6个架构约束：

(1) 客户端-服务器(Client-server): 客户端和服务端可以单独替换和开发, 并且彼此间不相互依赖, 客户端只知道资源URI。通过分离用户界面与数据存储提高了跨平台用户界面的可移植性; 通过简化服务器组件提升了服务器的可伸缩性。

(2) 无状态(Stateless): 客户端与服务端交互的每个请求应包含理解请求所需的所有内容, 且服务端上不会存储客户端上下文。客户端仅负责管理应用程序的状态。

(3) 可缓存(Cacheable): 响应必须直接或间接定义自身是否可被缓存, 避免客户端利用过期响应数据来发送其他请求。通过良好的缓存策略可有效减少客户端-服务端之间的交互, 从而进一步提高系统的可伸缩性和性能。

(4) 统一接口(Uniform interface): 统一接口简化和解耦的架构, 使得每个部分可以独立被修改。统一接口包含了四个约束: 标识资源、通过表述来操作资源、自描述的消息、超媒体作为应用程序状态的引擎。标识资源是指使用URI作为资源标识符需在请求中标识各个资源, 资源通常以JSON[29]、HTML或XML[30]作为载体。通过表述来操作资源指通过数据的元操作来修改和删除资源。自描述的消息指每条消息包含足够多的信息来描述如何处理该消息。超媒体作为应用程序状态的引擎指客户端通过body内容查询字符串参数, 请求URI和请求header来提供状态; 服务端通过正文内容响应代码和响应标头向客户提供状态。

(5) 分层系统(Layered system): 分层系统风格允许通过约束组件行为来使体系结构由分层组成, 使得每个组件不能“看到”超出与它们交互的直接层。

(6) 按需代码(Code on demand): 此约束是可选的。REST允许通过以小程序或脚本的形式下载和执行代码来扩展客户端功能。

本文基于RESTful约束设计接口, 客户端通过资源URI发送HTTP请求, 服务端返回以JSON为载体的response对象。本文的接口设计有利于前后端独立开发, 彼此间不相互依赖, 使得后端专注于数据处理, 具有提升开发效率, 降低耦合度的优点。

2.3.2 Flask框架

Flask是一个使用Python编写的轻量级和模块化设计Web应用框架, 其依赖于两个外部库: Werkzeug WSGI工具箱和Jinja2模板引擎。WSGI(Web Server

Gateway Interface)是基于CGI(Common Gateway Interface)通用网管接口的标准,为Python语言专门定义的Web服务器和Web应用程序或框架之间专门设计的接口。WSIG封装了接受HTTP请求、解析HTTP请求、发送HTTP响应等底层操作。Flask是一个微框架(Micro Framework),其中的“微”指的是Flask旨在保持核心功能的简单而易于扩展。Flask提供了众多第三方扩展,包括对象关系映射(Object Relational Mapping, ORM)、数据库集成、表单验证、文件上传及各种开放式身份验证技术等功能。Flask响应一次HTTP请求的过程需主要经过两个过程:(1)创建上下文对象。(2)路由分发和响应。如图 2.1所示。

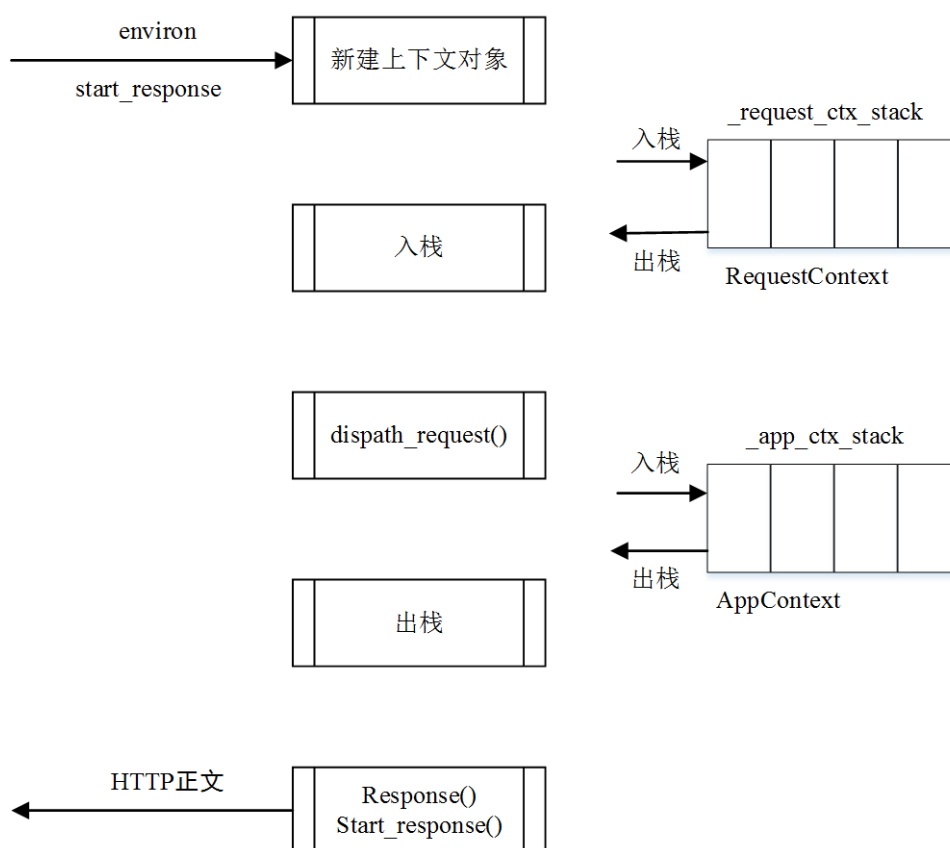


图 2.1: Flask请求处理流程图

创建上下文对象: Flask基于WSGI对HTTP请求解包后封装的environ对象创建RequestContext和AppContext两个上下文对象,将其分别push到对应栈中。在此次请求期间, request对象、 session对象和g对象都指向对应栈的栈顶。上下文指的是临时把某些对象变为全局可访问。Flask使用上下文让特定的变量在一个线程中全局可访问,与此同时不干扰其他线程。Flask包括请求上下文(RequestContext)和程序上下文(AppContext)。

由表 2.1可知，`current_app`和`g`属于程序上下文，`request`对象和`session`对象属于请求上下文。`current_app`指向当前激活程序实例；`g`作为全局对象适用于在应用程序上下文之间存储和传递临时值；`request`对象封装了客户端发送的HTTP请求的所有内容；`session`对象用于用户会话，根据`cookie`重载用户相关会话信息。

表 2.1: 上下文对象

变量名	类别	解释
<code>current_app</code>	程序上下文	当前激活程序的程序实例。
<code>g</code>	程序上下文	临时存储对象，每次请求都会重设该变量。
<code>request</code>	请求上下文	请求对象，封装HTTP请求中的内容。
<code>session</code>	请求上下文	根据请求 <code>cookie</code> 重载用户相关会话内容。

路由分发：Flask中使用Werkzeug WSGI来做路由分发。通过访问程序上下文`request`对象获得请求信息，调用`dispatch_request()`方法进行请求的分发，从URL映射中查找处理该请求的视图函数，根据匹配的路由结果调用对应视图函数进行处理，由WSIG调用`response`对象返回其结果作为HTTP正文。此时，将两个上下文对象出栈，为下一次的HTTP请求做准备。其中URL映射规则基于Werkzeug的路由模块。Werkzeug基于字典定义了一个规则列表，用于映射URL和视图函数的对应关系，其中`key`是URL，`value`是对应的视图函数。路由匹配通过正则匹配真实的路径信息并保存至字典中并返回。URL的风格没有限制，兼容RESTful的风格，但URL应保持唯一，避免索引相同的页面。

本文采用前后端分离架构，因此这里不对Flask的模版渲染进行讨论。本文使用Flask提供的`request`对象获取HTTP请求参数，利用路由分发将各URL请求映射至对应的视图函数处理，并返回`response`对象作为HTTP响应的正文。其中，URL设计过程遵循RESTful架构风格，有利于同SDK以及前端的对接。

2.3.3 SQLAlchemy工具

SQLAlchemy[31]是Python编程语言下的一款开源软件，提供了SQL工具包及对象关系映射工具，允许使用Python代码定义数据库模式来查询数据。业务实体可分为对象和关系数据两种表现形式，其在内存中以对象的形式存在，而在数据库中则表现为关系数据。SQLAlchemy使用Python语言，将程序中的对象映射为数据库中的数据，实现了高性能的数据库访问和完整的企业级持久模型。与面向对象语言和关系数据库之间的传统交换技术相比，ORM通常会减少需要编写的代码量[32]。使用SQLAlchemy具有以下几个优势。

(1) 使用Python构建查询语句。SQLAlchemy允许SQL语句和Python变量的结合，在内部重载各种比较运算符，再将其转换成SQL语句，降低了含有安全隐患的查询字符串，杜绝了SQL注入攻击。

(2) 简化数据库交互。SQLAlchemy隐藏了数据访问细节，将复杂的SQL语句的转换过程放置内部执行，使得数据库交互变得简单。每一个类对应数据库中的一张表，可以在类中添加自定义的类方法，将相关功能代码放在一起，方便维护。

(3) 支持自动同步数据库模式。可使用Alembic插件自动更新数据库模式，对于表结构的更改，无需写复杂SQL语句，如添加表或列等小改动都十分便捷。同时，可利用Alembic实现数据库迁移。

(4) 预先加载提高性能。SQLAlchemy使用预先加载策略，第一次加载某个对象时，预先通知SQLAlchemy需要使用的关系，在一次查询中加载需要的所有数据。

(5) 允许自定义库。SQLAlchemy允许创建自定义工厂函数，可以在会话中自动给所有查询应用过滤条件，防止代码编写中忘记添加过滤条件，使得某个用户看到其他用户信息，保证用户隐私。

本文使用SQLAlchemy封装数据结构，实现了关系数据库到对象的映射。利用Alembic实现对表结构的更新和迁移，利用session会话对象完成数据的查询、更新、插入等操作，实现了前端数据展示和交互的业务需求，同时减少了前端恶意查询带来的安全问题。

2.3.4 Ant Design React框架

React起源于Facebook的内部项目，于2013年5月开源，是一套性能出众单向数据流的用于构建网站页面的前端框架[33]。考虑频繁DOM操作引发的性能瓶颈，React为此引入了虚拟DOM(Virtual DOM)的机制，提出了高性能的diff算法，将算法时间复杂度从 $O(n^3)$ 简化为 $O(n)$ 。当数据更新时，通过diff算法计算出相应的更新策略，只对变化的部分进行实际的浏览器DOM更新，而无需重新渲染整个DOM树。React提供最大的灵活性、可扩展性和组件化开发。开发者使用React只需设计视图结构和关心组件间状态的传递，视图数据或结构会跟随状态的变化而自动更新，无需开发者手动修改视图。虚拟DOM同时也带来了组件化的开发的思想，通过将UI分成不同组件并独立封装，复用组件可大大加快软件项目的开发。同时，开发者可将不同组件组合或嵌套使用，构建更为复杂的组件。相比于采用模板编写HTML，使用JavaScript编写组件的业务逻辑，具有在组件间传递复杂的数据流动，实现状态与视图的隔离等优势。同时，React结

合第三方库拥有完善的生态系统，**React-Router**提供页面路由，通过管理URL，实现**React**组件的切换和状态的变化；**Redux**[34]集中式存储状态集合，可预测状态的更改，降低了数据管理的复杂度；**Mocha**¹测试框架可为应用添加测试，保证代码质量。

Ant Design React，简称**Antd**²，是基于**Ant Design**设计语言的**React**实现，具备**React**的良好特性。**Ant Design**设计语言不止是设计原则、控件规范和视觉尺寸，还配有前端代码实现方案，使得UI设计和前端界面研发可同步完成，效率大大提升，目前有阿里、美团、滴滴等企业采用。**Antd**除了具备**React**已有的优势和生态外，还拥有以下几个特性。

(1) **Antd**视觉风格遵循**Ant Design**，风格统一。开发者可基于**Antd**搭建风格一致的网页，让开发者专注于更好的用户体验。

(2) **Antd**拥有开箱即用的高质量组件库。**Antd**提供许多常用组件且组件样式和功能完善，可在组件的基础上进行一定范围内的自定义，使开发者无需重复造轮子，大大减少了代码的编写。

(3) **Antd**使用**TypeScript**编写，提供所有组件的类型定义文件。**TypeScript**静态类型检查能够在编译期间检测变量的类型，使得大多数错误在编译期间就被发现，减少修复成本。相比于动态类型语言，使用静态类型语言更有助于后期重构和代码维护。

本文基于**Antd**组件库，结合**TypeScript**、**HighCharts**、**Less**等技术编写前端页面，完成了崩溃可视化模块的页面展示。使用**Antd**大大减少了组件编写时间，避免了直接操作**DOM**而导致的性能问题，同时也降低了大量编写和修改**DOM**代码时因疏忽而出现错误的可能性。

2.4 本章小结

本章主要对安卓应用崩溃信息的收集、解析处理和可视化三个方面所用到的技术进行了阐述。首先介绍了安卓异常处理机制对崩溃信息的捕获原理，接着对比分析了面向切面编程与面向对象编程的区别，然后介绍了朴素贝叶斯和支持向量机算法在文本分类方面的实现原理，最后介绍了整个系统搭建所需要的技术。其中，详细阐述了**RESTful**架构特性，**Flask**框架的路由分发和匹配过程，使用**SQLAlchemy**与数据库交互的优势，及**Web**前端框架**Ant Design React**。

¹<https://mochajs.org/>

²<https://ant.design/docs/react/introduce-cn>

第三章 面向安卓应用的崩溃信息线上分析系统分析与设计

本章从用户角度出发，提炼并分析真实场景下用户对崩溃信息线上分析系统的需求，转换为系统产品需求。同时基于需求分析结果，遵循系统易拓展、高可用的设计原则，将系统总体架构分为三个关键子模块：崩溃收集模块、崩溃处理模块、崩溃可视化模块。满足模块内高内聚，模块间松耦合的设计要求，对各模块进行了详细的功能设计和数据库设计。

3.1 系统需求分析

全面且准确的需求分析是一个成功软件项目的前提，也是项目开发过程的根本依据。下面将根据系统业务分析的结果，从功能性需求和非功能性需求两方面明确定义系统整体需求。

3.1.1 功能性需求

作为完整的软件系统，在满足用户管理功能（用户登录、注册和用户信息增删查改等）的基础上，功能需求分析部分主要对崩溃收集模块、崩溃解析处理模块以及崩溃可视化模块三部分内容进行分析。

应用管理者可在产品管理界面创建产品，系统将自动为产品生成全局唯一的appKey，用于产品配置。同时，系统支持产品修改和产品删除等基本功能。

应用开发者基于崩溃收集模块SDK，通过在目标应用源码中配置对应产品appKey接入系统，再将应用打包发布至应用商店供移动用户下载。

崩溃收集模块SDK实时监控应用在用户端运行状态，自动捕获移动用户使用过程中应用发生的崩溃，对用户友好提示的同时收集崩溃相关信息，如异常堆栈、设备硬件信息、设备性能状态信息、用户页面路径跟踪和崩溃截图等，并将整体崩溃数据上报至系统服务器。

崩溃解析处理模块自动解析处理由崩溃收集模块SDK上报的崩溃数据，通过堆栈信息特征模式匹配和不一致性分析实现崩溃去重，通过解析堆栈信息实现代码定位和页面定位，基于朴素贝叶斯和支持向量机算法实现崩溃分类。

崩溃实时监控分析界面展示近24小时崩溃数据统计分析。崩溃异常问题统计Top5支持点击查看崩溃详情。

崩溃列表界面展示当前产品所有崩溃。用户可根据崩溃类型或崩溃状态对崩溃数据进行过滤，同时可按崩溃名称进行搜索。

崩溃详情界面详细展示了特定类别崩溃详情以及该类别崩溃实例信息。用户可在该页面编辑崩溃状态、查看第三方解决方案并点赞、支持提交自定义解决方案、查看设备软硬件信息、查看崩溃详细堆栈信息、查看页面路径跟踪、查看不同视图分布及查看崩溃截图等。通过查看崩溃详情页面的多维度崩溃信息，可以帮助开发者快速定位崩溃原因，进行针对性修复。

本系统主要用户需求和产品需求如表 3.1所示。其中，R1、R4、R5、R6、R7为用户需求，R2和R3为基于用户需求提炼出的产品需求。

表 3.1: 需求列表

需求ID	需求名称	需求描述
R1	产品管理	应用管理者可对产品进行管理，包括产品创建、产品修改和产品删除。
R2	SDK集成配置	用户能查看SDK集成说明并下载SDK，将SDK嵌入安卓应用。
R3	崩溃自动收集与上报	系统自动收集移动用户使用过程中发生的崩溃相关信息，如异常堆栈、设备硬件信息、设备性能状态信息、用户页面路径跟踪和崩溃截图等，再将其上传至服务器。
R4	崩溃自动解析处理	系统自动解析崩溃的异常堆栈信息，并对崩溃进行分类和去重。
R5	实时监控分析	用户点击实时监控菜单栏，系统将以图表形式展示近24小时的实时崩溃数据的统计分析。
R6	崩溃列表	用户点击某个产品可跳转至该产品的崩溃列表界面，该界面展示了收集的所有崩溃数据。用户可根据崩溃状态、崩溃类型进行数据过滤，还可输入异常名进行崩溃搜索。
R7	崩溃详情	用户点击某个崩溃名可查看崩溃详情，崩溃详情以图表形式丰富展示了详细的崩溃相关信息。用户可点击查看第三方解决方案并对其点赞；可修改崩溃状态，并提交自己的解决方案；可查看机型设备分布、应用版本分布及上报趋势；用户可通过查看不同页面分布得知当前崩溃发生的页面、代码行数和崩溃截图；通过查看崩溃实例查看设备硬件信息、设备性能状态信息、详细堆栈及用户页面路径跟踪等。

根据上述系统功能性需求的分析，本系统用例图如图 3.1所示。本系统核心用户分为应用管理员和应用开发者，应用管理员负责上线产品，具备应用管理相关权限；应用开发者负责移动应用产品开发，具备崩溃解析处理与崩溃报告查看等相关权限。

由图 3.1可知，应用管理者的用例主要包括产品管理，分为产品创建、产品修改和产品删除。开发者用例主包括崩溃收集模块SDK集成配置、实时监控

分析、崩溃列表查看和崩溃详情查看。其中，崩溃实时监控分析功能包括查看崩溃统计、上报趋势、新发现问题数、影响用户占比及今日问题Top5。崩溃列表功能包括过滤和搜索等。崩溃详情功能包括不同机型分布、不同应用版本分布、不同系统分布、查看解决方案、修改崩溃状态、查看崩溃统计、上报趋势、页面路径跟踪以及详细堆栈等。本系统为安卓应用开发团队提供全方位崩溃实时监控，可实现及时准确地收集移动用户使用应用程序时发生的崩溃数据，并统计、分析和展示崩溃影响用户数、崩溃覆盖机型设备等详细数据，协助开发者从多个不同维度快速、精确定位并修复Bug，全面保障应用程序质量。

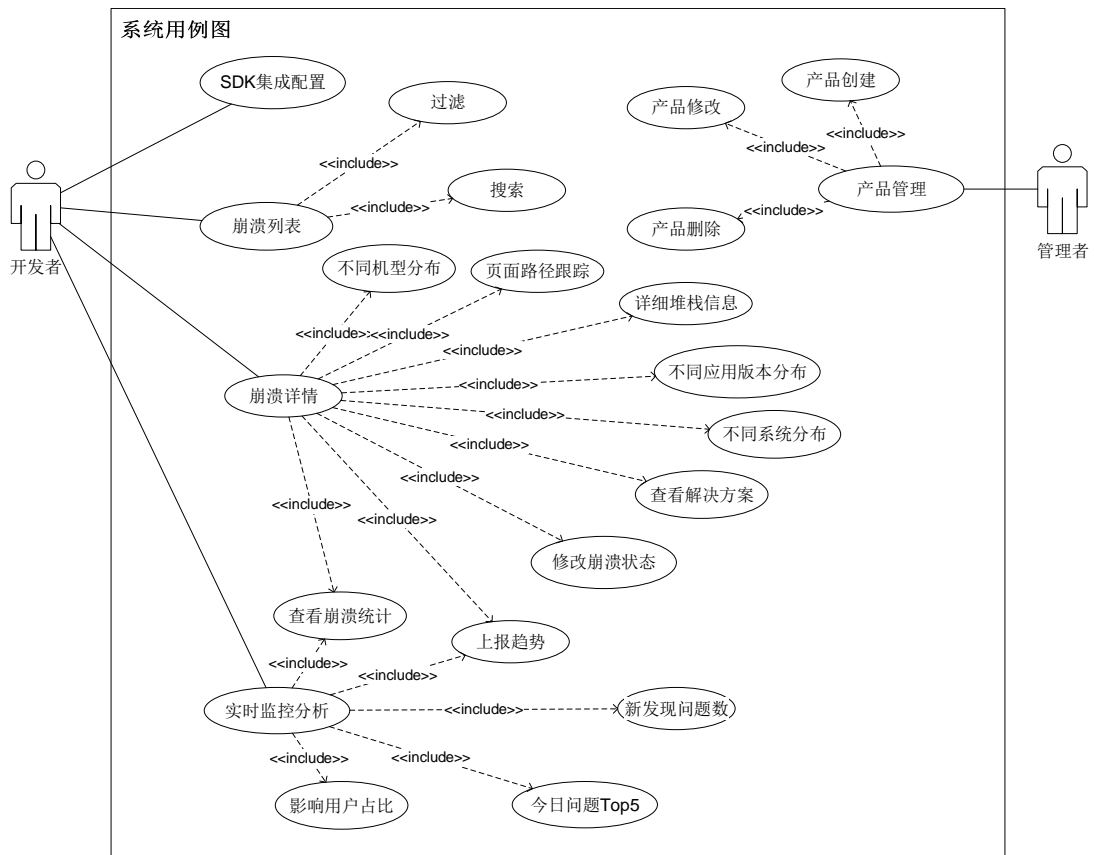


图 3.1: 系统用例图

3.1.2 用例描述

产品管理是应用管理员特有的功能，支持管理员进行产品创建、产品修改和产品删除等操作。用户通过点击产品创建按钮并填写产品名称、产品类型等描述信息，当用户填写内容不符合系统要求时，系统应给出相应错误提示。其用例描述如表 3.2 所示。

表 3.2: 产品管理用例描述

用例名称	产品管理
参与者/角色	应用管理员
优先级	高
前置条件	用户登录并进入系统
后置条件	点击进入产品列表界面
主事件流	<ol style="list-style-type: none"> 1.用户输入应用名称 2.选择创建类型 3.输入应用描述 4.用户上传应用图标 5.用户点击保存。 6.系统提示修改成功并跳转至产品列表界面 7.用户点击产品修改按钮， 8.系统跳转至产品修改界面 8.用户填写产品信息并点击保存 9.系统显示产品修改成功 10.用户点击产品删除按钮并点击确定按钮 11.系统提示产品删除成功
备选事件流	<ol style="list-style-type: none"> 1a.应用名称不是字母、数字或下划线 4a.图片大小超过2MB 5a.应用类型未选择 5b.应用描述未填写 9a.应用类型未选择、应用描述未填写 10a.用户点击取消按钮 11a.系统取消删除

SDK集成配置是整个系统的基础，主要用于将安卓应用接入本系统。用户可在集成说明页面点击SDK下载按钮获取SDK，在产品修改页面一键复制appKey，根据集成说明页面说明完成产品接入。其用例描述如表 3.3所示。

表 3.3: SDK集成配置用例描述

用例名称	SDK集成配置
参与者/角色	应用开发者
优先级	高
前置条件	用户登录并进入系统
后置条件	点击进入SDK集成说明页面
主事件流	<ol style="list-style-type: none"> 1.用户点击下载SDK 2.页面自动下载SDK 3.用户点击设置按钮进入产品修改页面并点击复制appKey 4.系统显示复制成功

实时监控分析功能主要用于展示当前产品近24小时内所有崩溃，用户可在该页面查看崩溃发生次数、影响用户数、上报趋势和问题统计等内容，用户可通过点击问题统计Top5的崩溃名称进入崩溃详情页面，用户可切换不同时间段查询相应时间段内的崩溃上报趋势。通过实时监控可以帮助开发者及时掌握应用运行状况及崩溃趋势。用例描述如表 3.4 所示。

表 3.4: 实时监控分析用例描述

用例名称	实时监控分析
参与者/角色	应用开发者
优先级	高
前置条件	用户登录并进入系统
后置条件	无
主事件流	<ol style="list-style-type: none"> 1.用户点击实时监控分析菜单栏 2.页面跳转至实时监控分析界面 3.用户切换不同时间段查询上报趋势 4.页面展示对应时间段上报趋势 5.用户点击Top5表格中的一行 6.页面跳转至对应的崩溃详情界面

崩溃列表的目的是帮助开发者准确掌握移动产品所有异常，全面把握崩溃类型、数目及状态，以便快速处理高发未修复异常。崩溃列表展示了当前产品所有崩溃类型，用户可在该页面按照崩溃类别或崩溃状态进行过滤查询，也可输入崩溃名称直接搜索。用例描述如表 3.5所示。

表 3.5: 崩溃列表用例描述

用例名称	崩溃列表
参与者/角色	应用开发者
优先级	高
前置条件	用户登录并进入系统
后置条件	无
主事件流	<ol style="list-style-type: none"> 1.用户点击产品名 2.页面跳转至产品对应崩溃列表界面 3.用户选择崩溃状态 4.页面重新加载对应状态的崩溃 5.页面选择崩溃类型 6.页面重新加载对应类型的崩溃 7.用户输入崩溃名称并点击搜索 8.页面重新加载符合条件的崩溃

崩溃详情主要用于展示特定崩溃类别详细信息及该类别崩溃实例，崩溃实例展示该崩溃发生时详细的设备软硬件信息、堆栈信息和用户页面路径跟踪。用户可根据崩溃不同分布视图，选择特定机型、系统或页面来进行针对性修复。同时，通过系统提供的官方解决建议，可在一定程度上减少开发者定位和修复异常的开销。通过崩溃截图帮助开发者复现崩溃场景。用例描述如表 3.6所示。

表 3.6: 崩溃详情用例描述

用例名称	崩溃详情
参与者/角色	应用开发者
优先级	高
前置条件	用户登录并进入系统
后置条件	无
主事件流	<ol style="list-style-type: none"> 1.用户点击崩溃名 2.页面跳转至崩溃对应崩溃详情界面 3.用户点击不同页面分布 4.页面展示崩溃的不同页面分布视图 5.用户点击不同机型设备分布 6.页面展示崩溃的不同机型设备分布视图 7.用户点击查看截图 8.系统显示崩溃截图 9.用户点击上报趋势 10.页面加载崩溃在相应时间段的上报趋势图 11.用户点击不同崩溃实例 12.页面重新加载对应实例的堆栈信息和页面路径跟踪信息 13.用户点击更改崩溃状态按钮 14.用户选择已处理，并填写解决方案 15.页面提示更改成功，且崩溃名被横线划去
备选事件流	<ol style="list-style-type: none"> 14a.用户点击取消 15a.系统取消更改

3.1.3 非功能性需求

面向安卓应用的崩溃信息线上分析系统的主要目标是协助安卓应用开发人员定位和修复已上线应用在测试阶段未被检测到的Bug，保证应用稳定、快速地故障处理，以提升应用的服务质量。系统在满足保障基本功能性需求前提下，对于崩溃收集、分析处理以及展示的及时性、可靠性是至关重要的，这也对本系统提出了一系列非功能性需求。本小节将基于安卓应用开发人员视角分析本系统的非功能性需求。

及时性：本系统应保证崩溃收集的及时性，即崩溃发生到系统成功捕获上

传、解析及显示监控数据应控制在1min内，以帮助开发人员快速定位Bug，减少用户损失。

可靠性：系统应有完善的错误处理机制以及充足的系统数据备份，能够保证当系统出现死机或重启后，相关数据的安全性和可恢复性。同时随着数据量的增加，保证系统也能相对稳定的运行。应考虑用户使用过程中由于网络问题等原因无法将崩溃信息上传到服务器的情况，应保存副本至设备本地并设计崩溃重传机制以防止崩溃数据的丢失。

容错性：当用户操作出现不正确时，系统应进行相应检测并给出温馨提示并给出正确的操作方式，不能出现系统无法正常使用或崩溃的情况。

扩展性：保证系统的可扩展性应设计良好的接口模块，以保证随着业务的扩张能够满足系统的需求，无需大的改动即可添加新的功能模块。

3.2 系统架构设计

本文搭建的面向安卓应用的崩溃线上分析系统重点关注安卓应用在使用过程中发生的崩溃信息的收集、解析处理与可视化。系统整体架构设计如图 3.2所示。该系统全面适配市场主流安卓应用类别，如工具类App、信息类App、购物类App等。本系统自顶向下可以分为崩溃收集模块、崩溃解析处理模块以及崩溃可视化模块。

崩溃收集模块：应用开发者通过崩溃收集SDK和唯一appKey接入系统。崩溃收集SDK自动检测应用程序运行状态，自动捕获用户使用过程中发生的崩溃，收集崩溃相关数据，包括堆栈信息、设备硬件信息、设备性能状态信息、用户页面路径跟踪、崩溃截图等。最后将结构化崩溃数据上传至服务器。

崩溃解析处理模块：本模块是整个系统的核心，具体业务主要分为两个方面：（1）崩溃分类。系统对崩溃收集模块上传的崩溃数据进行预处理，训练分类模型，并利用训练好的分类模型实现对预处理后数据的分类。（2）崩溃去重。解析崩溃收集模块上传的崩溃数据，利用特征模式匹配提取崩溃堆栈中关键信息并生成结构化数据，对结构化崩溃数据进行不一致性分析，基于不一致性进行数据去重。

崩溃可视化模块：该模块主要实现崩溃信息的多维度展示。通过对崩溃信息的统计分析服务，提供崩溃解析、崩溃定位、崩溃分布及崩溃统计等功能。基于这些服务，用户可通过可视化模块进行产品管理、查看崩溃列表、查看崩溃详情、实时崩溃数据监控分析等操作。

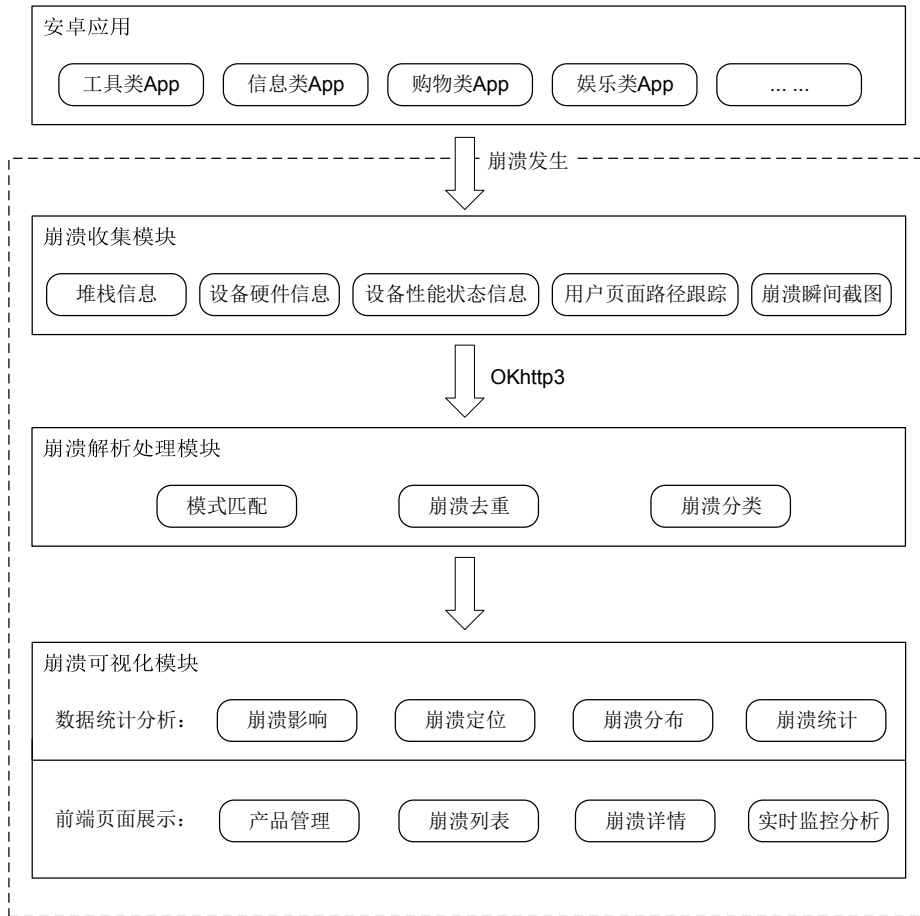


图 3.2: 系统整体架构设计图

以下将从场景视图、逻辑视图、进程视图、开发视图和物理视图五个方面详细介绍本系统的总体架构。

场景视图是从用户需求出发来识别业务及产品需求，以用例图的方式描述业务场景，是系统设计的起点。本系统的用例图如图 3.1 所示。

图3.3是崩溃线上分析系统的逻辑视图，该视图描述系统为用户提供的功能和服务。其中，Screenshot用于截取应用崩溃瞬间截图；HookActivity负责监听应用Activity生命周期功能；CrashHandler是崩溃收集模块的核心，可自动捕获应用使用过程中的崩溃并收集相关信息；UploadFile提供崩溃上传功能；CrashAccept是崩溃解析处理模块的调度中心，负责处理崩溃上报请求；Classify提供上报崩溃的分类功能；Deduplicate对分类后数据进行去重功能；MysqlServer负责整个数据库的读写操作；AnalysisCrash提供崩溃数据的统计分析相关接口；CrashList提供崩溃列表查询功能；GetToday提供实时监控分析功能；CrashDetail负责生成详细的崩溃详情报告；Product提供产品管理功能。

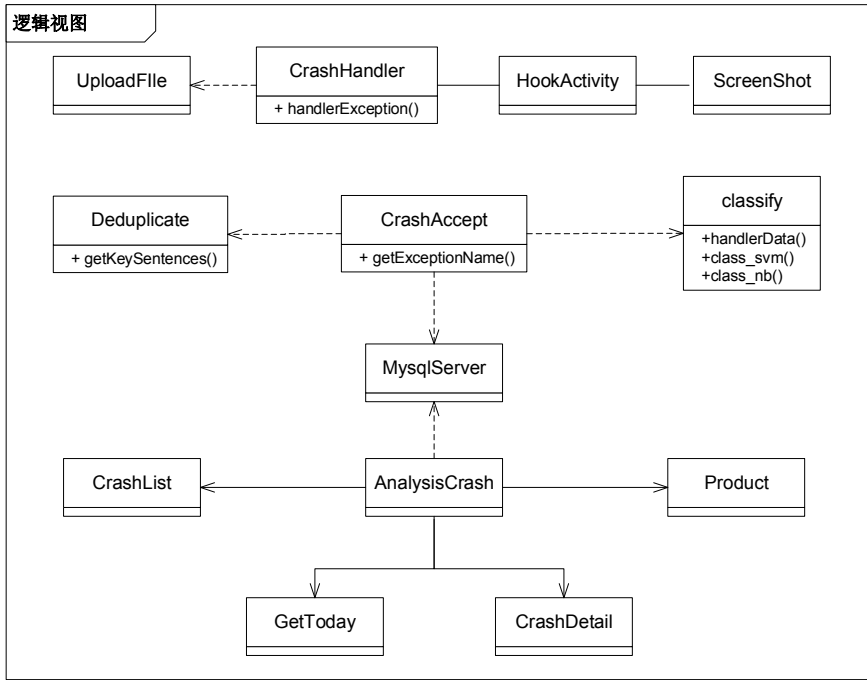


图 3.3: 系统逻辑视图

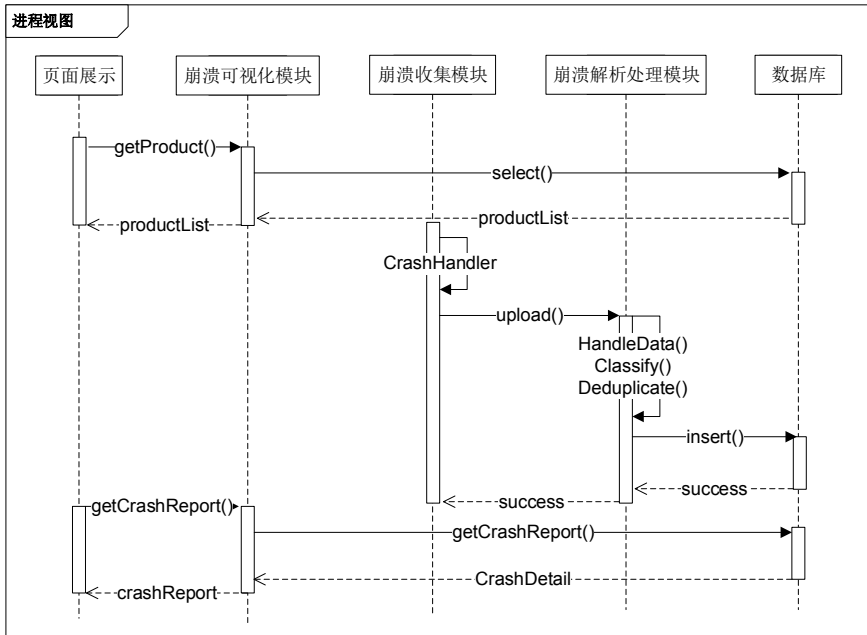


图 3.4: 系统进程视图

图3.4是崩溃线上分析系统的进程视图，该视图面向系统集成者视角，描述整个系统的进程与通信。首先由用户与可视化模块进行交互，进行产品创建与绑定等操作。系统将通过崩溃收集模块调用自身CrashHandler()方法自动捕捉移

动用户使用过程中的应用崩溃，收集相关信息，并同步上传至服务器。崩溃解析处理模块在接收到崩溃信息后，将同步调用数据预处理HandleData()、崩溃分类Classify() 和崩溃去重Deduplicate()等方法，并存储至数据库。此后，用户与可视化模块交互时，可调用崩溃相关的服务，如崩溃列表、崩溃详情和实时监控分析等服务掌握应用运行状况、定位和修复崩溃。

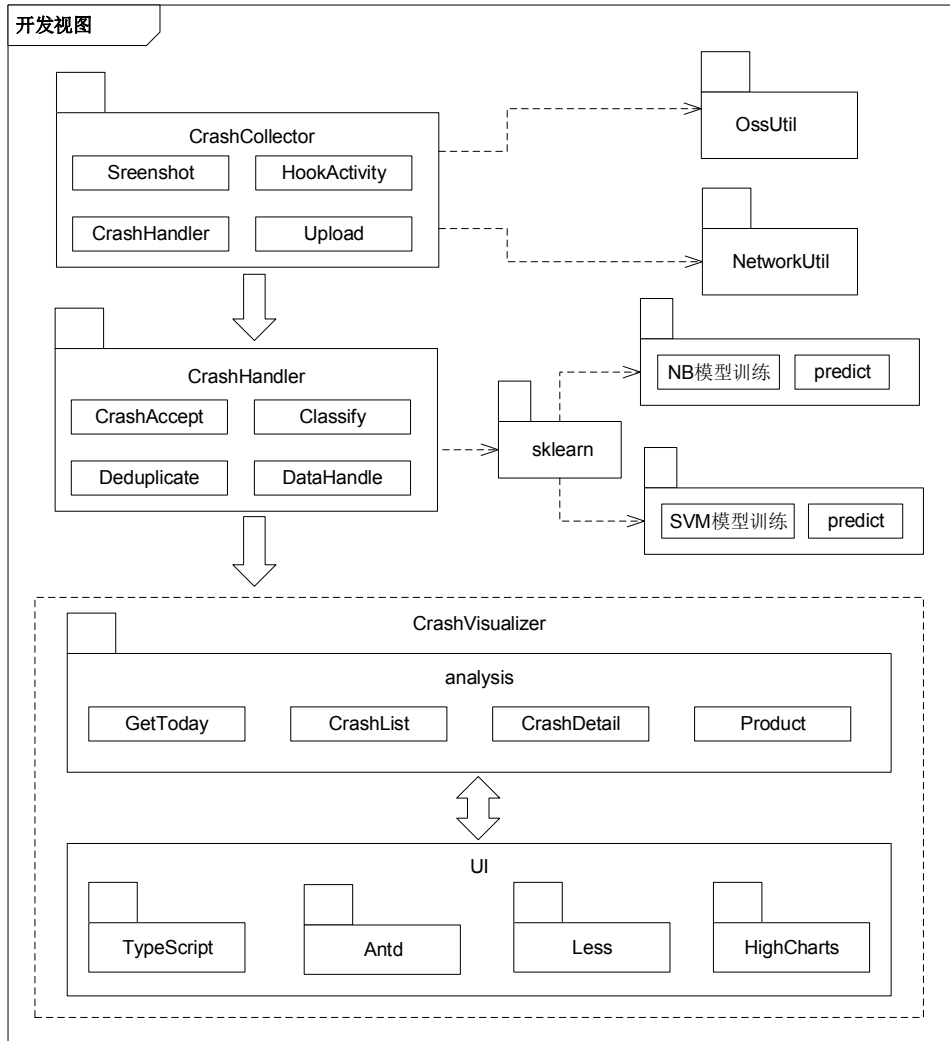


图 3.5: 系统开发视图

图3.5是崩溃线上分析系统的开发视图，该视图从开发人员的角度，描述系统的静态组织结构。本系统由CrashCollector、CrashHandler、CrashVisualizer三个模块组成。其中，CrashCollector负责崩溃数据的收集，OssUtil和NetworkUtil分别负责上传截图和当前设备网络状况。CrashHandler负责崩溃数据的解析和处理，通过sklearn包中的朴素贝叶斯和支持向量机算法来训练分类模型和预测崩溃类别。CrashVisualizer负责崩溃数据的可视化，主要分为analysis和UI两部

分。analysis提供数据统计分析接口服务，包括实时监控分析相关接口、崩溃列表相关接口、崩溃详情相关接口及产品管理相关接口。UI展示用户交互界面，通过Antd、TypeScript、Less和HighCharts[35]等技术实现。

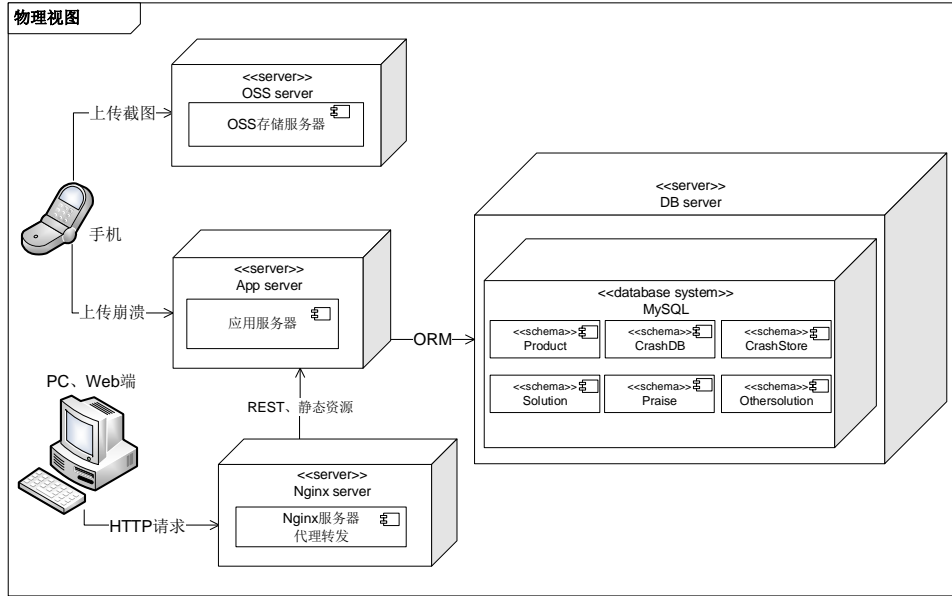


图 3.6: 系统物理视图

图3.6是崩溃线上分析系统的物理视图,该视图面向运维人员视角，描述系统部署结构。安卓应用通过嵌入崩溃收集SDK接入系统，由SDK自动捕获崩溃并将崩溃信息上传至后台服务器，将截图上传至阿里云对象存储(Object Storage Service, OSS)服务器。本系统采用前后端分离架构，前端通过Webpack打包前端资源并部署于Nginx服务器。在本系统的前端展示页面中，需大量加载静态Web资源[36]，如JS、CSS和Image等。对于静态资源请求访问，由Nginx直接处理并返回；对于动态数据请求访问，则需通过Nginx实现URL的代理转发，并解决跨域问题。本系统后台服务部署于应用服务器，后台基于Flask框架实现，HTTP请求的处理由Flask自带的Werkzeug WSGI服务器网关接口进行HTTP请求的解析，由URL映射到对应的视图函数进行处理并同步返回HTTP响应。其中，后台服务访问关系型数据库MySQL则是利用ORM框架实现。

3.3 崩溃收集模块设计

崩溃收集模块用于自动捕获安卓应用使用过程中发生的崩溃信息。对于开发者来说，仅基于单一维度数据定位崩溃产生原因非常困难且耗时，加之安卓平台天生的碎片化特性，利用多维度数据整体刻画崩溃是十分必要的。本模块

支持自动捕获应用崩溃时抛出的异常堆栈信息、收集设备硬件信息和设备性能状态信息，并上传结构化崩溃数据至服务器。除此之外，为实现崩溃和用户行为数据的关联分析，本模块应具备监听Activity生命周期的功能，实现用户页面路径跟踪和崩溃截图获取。如图 3.7所示，崩溃收集模块主要包括监听Activity生命周期和安卓崩溃全局捕获等2个功能子模块。以下将分别针对各个功能模块的具体设计进行阐述。

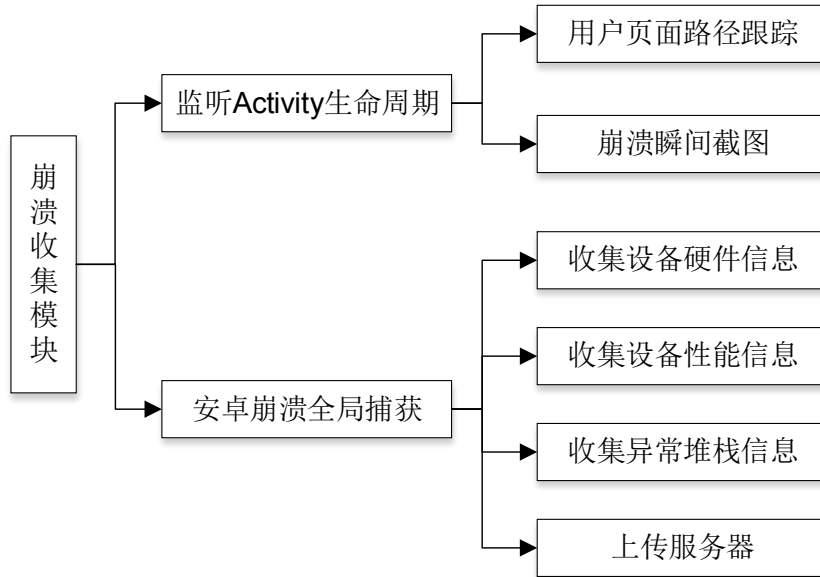


图 3.7: 崩溃收集模块功能图

3.3.1 监听Activity生命周期设计

Activity是安卓四大核心组件之一，通常一个Activity对应一个用户可见屏幕，要实现准确监控应用实时运行情况，Activity生命周期跟踪技术必不可少。要实现安卓Activity生命周期的监听，首先应分析安卓Activity生命周期中的不同状态，不同的Activity状态对于外部事件有着不同的响应事件。如图 3.8 所示，每个Activity的启动，都会先依次经历onCreate()、onStart()、onResume()这3个过程，此时Activity对应的可视化页面才能为用户所见。随着用户点击或执行其他操作，Activity会调用相应的事件监听器响应用户操作，并调用对应Listener方法触发状态的转化，Activity状态的变化伴随着生命周期关键函数的自动调用，包括onStart()、onResume()、onPause()、onStop()、onRestart()等。其中onCreate()和onDestory()在页面第一次被初始化和页面最终被销毁时调用，整个生命周期内只会执行一次。

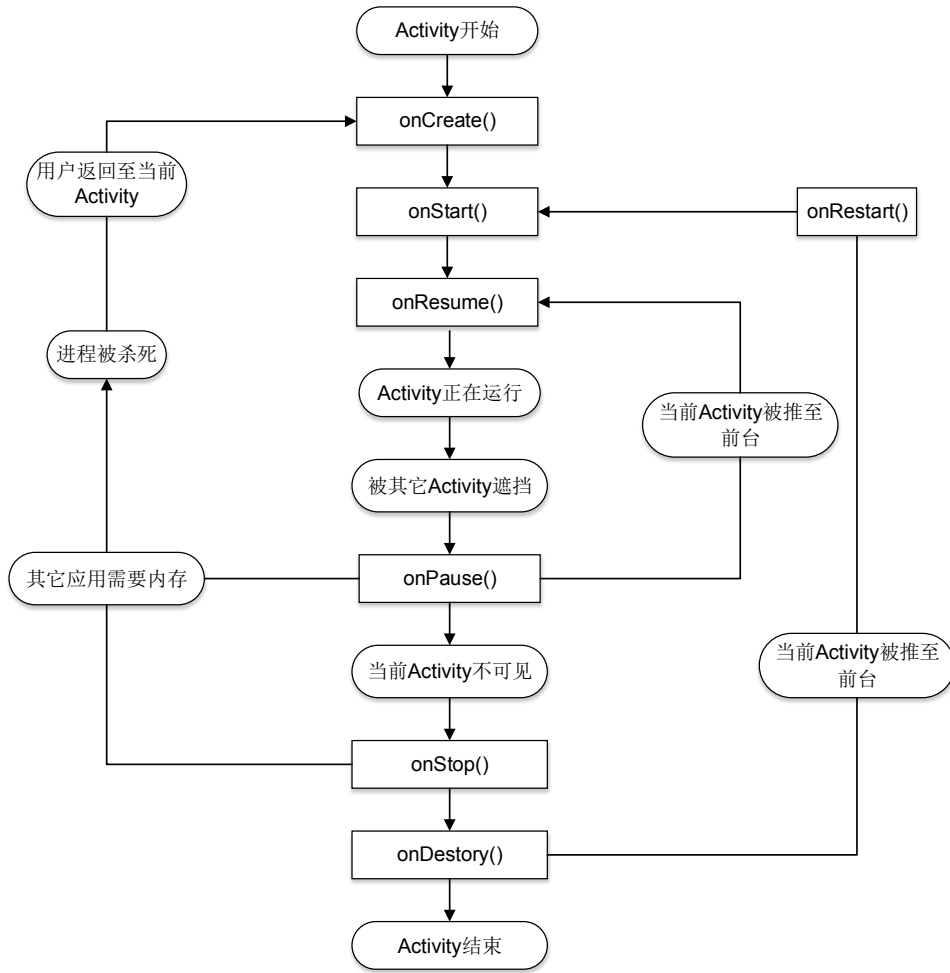


图 3.8: 安卓Activity生命周期图

基于上述分析，监听Activity的生命周期可利用切面编程技术来实现。如类图 3.9所示，关键步骤是在Activity状态变化且自动调用对应方法时，通过切面编程技术，动态织入自定义业务逻辑（如记录该方法调用时间和参数），进而实现对Activity完整生命周期的监听。具体实现类和接口包括HookActivity、ActivityLiftManager、MyActivityLifyCycleCallbacks、MyInstrumentation、ScreenShot。其中，HookActivity是监听Activity的生命周期的核心，自定义业务逻辑在该类实现，实现获取当前Activity名称、当前时间、以时间戳命名的当前页面截图等数据并写入文件等功能。MyActivityLifyCycleCallbacks作为切入点，实现对Activity原生生命周期函数的重写。ScreenShot工具类实现截图功能并存储至设备。通过跟踪用户页面路径，记录当前用户从启动应用到崩溃发生时所经过的所有页面，进一步分析当前崩溃的发生是否跟页面跳转有关。通过获取用户每个页面的截图，不仅能够获取崩溃瞬间截图，还能够将用户页面路径跟踪

与截图一一对应，通过崩溃瞬间截图可以更直观地重现崩溃发生场景，并交代崩溃上下文信息。

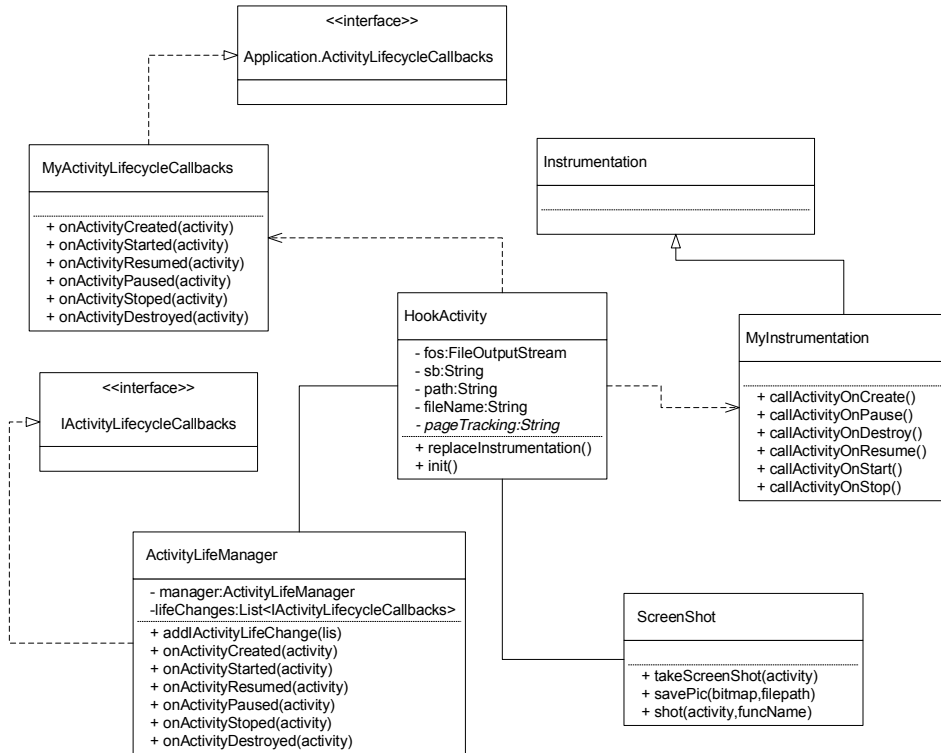


图 3.9: 监听Activity生命周期类图

3.3.2 安卓全局崩溃捕获设计

安卓全局崩溃捕获即自动捕获应用程序抛出的任意崩溃，获取异常堆栈信息，截取当前页面，收集设备硬件信息和设备性能状态信息，最后上传至服务器。具体功能模块类图 3.10所示，安卓全局崩溃捕获主要包括CrashHandler、NetWorkUtil、UploadFile、OssUtil。NetWorkUtil工具类提供网络连接及运营商获取的功能；OssUtil工具类中封装了OSS的文件上传、文件获取等功能；UploadFile实现上传崩溃相关信息至服务器和上传崩溃截图至OSS的功能；CrashHandler作为崩溃捕获调度中心，当应用发生未捕获异常，CrashHandler调用自身函数用于获取堆栈信息和设备信息，同时调用HookActivity类获取用户页面跟踪数据，最后调用UploadFile将所有信息上传。以下将针对异常堆栈、设备硬件信息和设备性能状态信息的获取以及上传服务器的设计进行详细说明。

异常堆栈信息收集：安卓应用主流开发语言是Java，因此安卓应用的绝大部分崩溃和闪退都可以归为Java异常体系结构中的Exception或者Error。通过捕

获程序运行时抛出的任意Throwable对象以实现全面监听程序发生的错误或异常。异常堆栈信息的收集通过继承 Thread.UncaughtExceptionHanler 接口，构造自定义异常处理器来收集定制化异常堆栈信息，同时调用 Thread.setDefaultUncaughtExceptionHandler 将自定义异常处理器设置为应用程序全局异常处理器来实现全局崩溃监听。

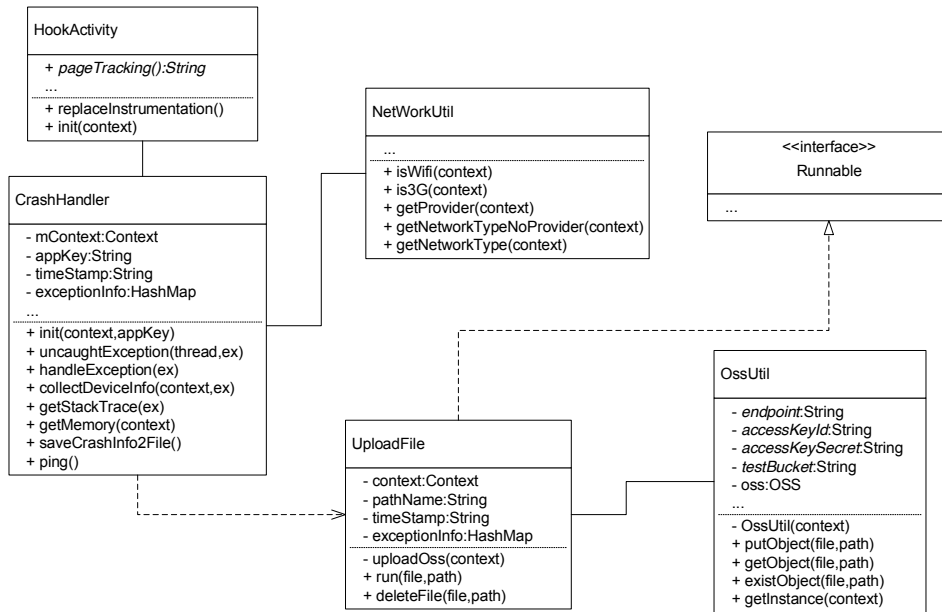


图 3.10: 安卓全局崩溃捕获类图

设备硬件信息的收集：安卓系统一直饱受碎片化特性困扰，不同品牌、不同型号移动设备所搭载的不同软硬件配置是导致移动应用崩溃发生的常见原因之一。由于安卓移动设备的多硬件配置，导致硬件设备差异导致的崩溃甚至比设备间的软件差异更严重。对于大多数应用开发者来说，无法在市场上如此繁多的设备上详尽的测试，而设备软硬件信息导致的崩溃往往很难被发现和定位。因此，除了对移动应用崩溃堆栈的收集外，对崩溃设备本身的软硬件信息的收集也显得尤为重要。本文通过安卓系统自带的android.os.Build类去收集设备硬件信息，如品牌、CPU、序列号等，该类的使用不需要任何权限。

设备性能状态信息的收集：运行时内存占用是用于评价移动应用性能的重要指标，占用过多内存通常会导致应用卡顿甚至引发崩溃。此外，内存占用不仅影响应用自身性能，还影响整台设备对用户的响应，过高的设备内存占用导致用户卸载应用显然是应用开发者需极力避免的。因此，要准确反映崩溃的发生是否与设备性能有关，必须提供应用程序运行时的设备内存状态信息，如内存占用等。内存状态信息通过ActivityManager.getMemoryInfo()方法去获取。

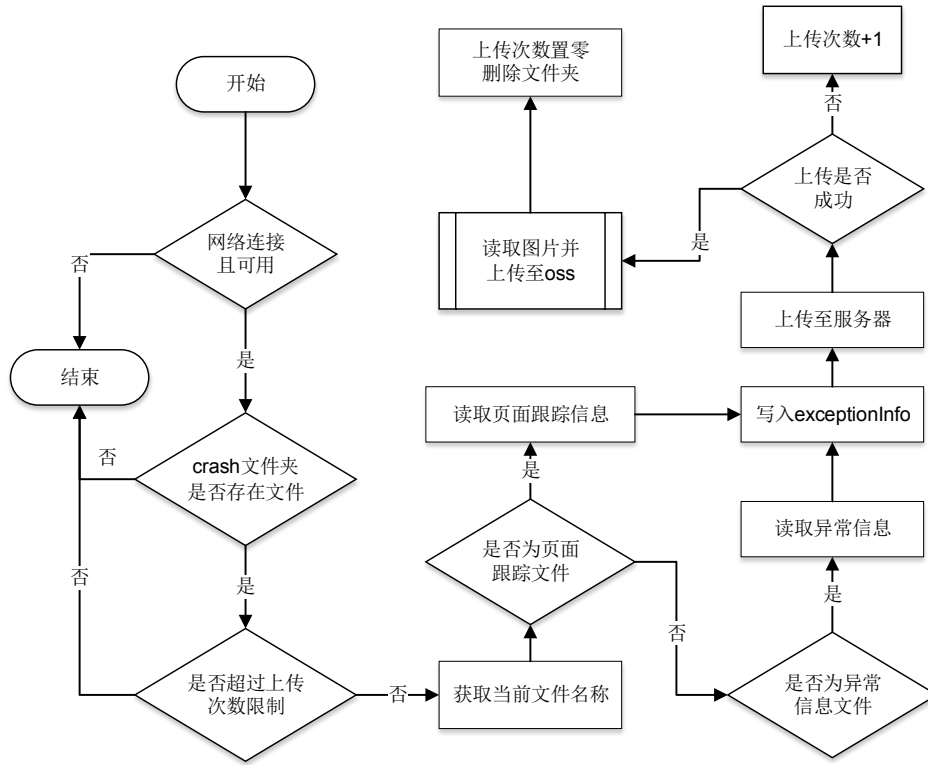


图 3.11: SDK重传崩溃流程图

上传服务器：将所有收集到的堆栈信息、设备硬件信息和设备性能状态信息都上传至服务器，从而帮助开发者及时了解应用运行状态并针对性地快速修复崩溃代码。本系统设计了崩溃实时上传功能，同时也考虑用户因网络问题无法上报崩溃的情况，设计了如图 3.11所示的崩溃重传流程。当应用启动时，首先判读网络是否可用，如果无法连接则结束；否则继续判断是否存在崩溃文件夹，如果不存在则结束流程；否则继续判断上传次数是否超过限制，如果是则结束；否则将逐个文件进行判断，若为页面跟踪文件则读取页面跟踪信息并写入exceptionInfo，若为异常信息文件则读取异常信息并写入exceptionInfo，将exceptionInfo作为HTTP请求的body内容上传至服务器；如果上传失败，将上传次数加一；否则将读取文件夹的截图并上传至OSS，最后将上传次数置零并删除崩溃文件夹。

3.4 崩溃解析处理模块设计

如图 3.12所示，崩溃解析处理模块主要包括去重和分类两个功能模块。崩溃收集模块上传的数据，首先应由崩溃解析处理模块对崩溃数据进行数据预处理，并使用朴素贝叶斯和支持向量机算法对处理后的数据进行分类，将预测类

别与解决方案一一对应，帮助开发者根据解决方案快速修复异常。然后利用特征模式匹配和不一致性分析实现崩溃数据的去重。

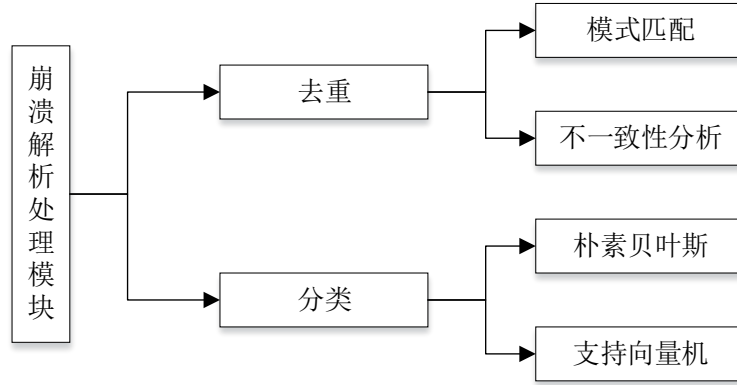


图 3.12: 崩溃解析处理模块功能图

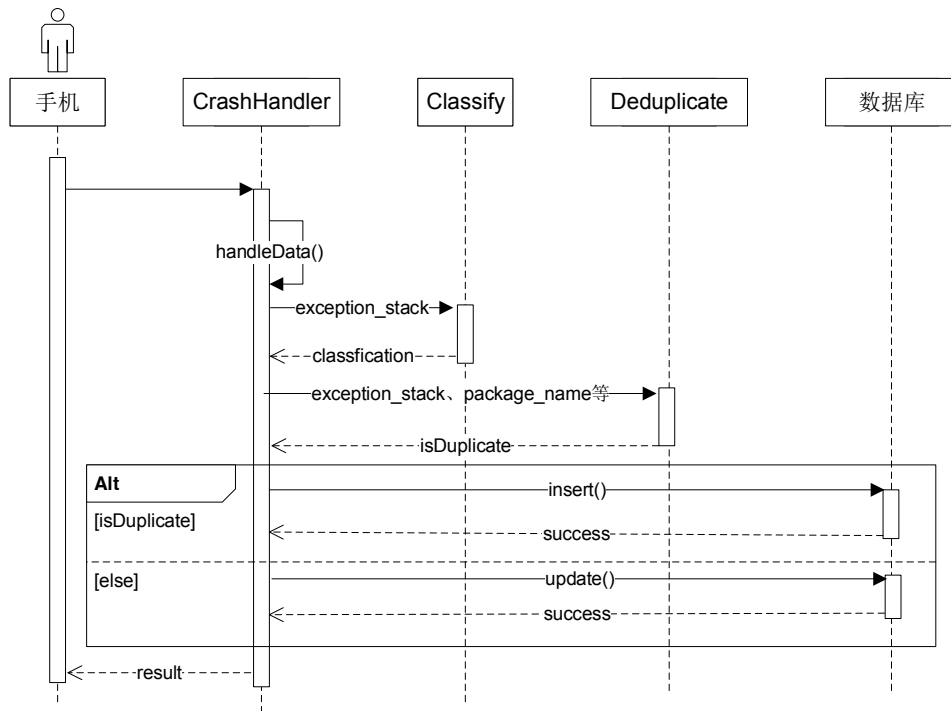


图 3.13: 崩溃解析处理模块时序图

崩溃解析模块的时序图如图 3.13 所示。当移动用户使用安卓应用过程中发生崩溃时，崩溃收集模块将自动捕获异常并上报。此时由 CrashHandler 负责对数据进行预处理，并将处理后的 exception_stack 字段传入崩溃分类模块 Classify 进行崩溃分类，崩溃分类模块将回传 classification 字段作为崩溃类别。

然后将exception_stack、package_name等字段传入崩溃去重模块Deduplicate进行数据去重，崩溃去重模块回传isDuplicate字段。最后由CrashHandler根据该字段判断当前崩溃是否需要忽略当前崩溃，并负责完成数据库写入操作。以下将针对崩溃去重模块和崩溃分类模块的设计进行详细说明。

3.4.1 崩溃去重模块设计

在移动应用的实际场景中，崩溃的发生在时间段上来说通常是呈聚集型表现。如发布新功能中存在的潜在Bug导致使用此功能的所有用户会触发导致崩溃，且绝大部分用户使用习惯是相似的。因此，从不同用户端收集到的重复崩溃必须进行整合，即实现崩溃数据的去重工作。

此外，即使是同一个Bug触发的崩溃堆栈信息也会存在差异，因此崩溃去重主要针对的重复的数据类型指的是数据类别相同，而堆栈信息有略微差别。重复数据的略微差别主要包括两个方面，一是报错的代码行数不同，二是崩溃所在的Activity不同，Activity不同又分为不同Activity发生了相同的错误或者是不同Activity调用了相同的代码，如使用同一控件、同一方法、同一接口。因此，崩溃去重应该忽略与Activity、代码行数相关的描述，重点关注堆栈信息中的异常名和异常报错原因的描述。

```

异常名+细节信息+路径组成
Caused by:异常名+细节信息+路径组成
... \d more
Caused by:异常名+细节信息+路径组成
... \d more
...
最后一个异常的全部细节信息和路径组成
    
```

图 3.14: 堆栈信息结构图

如图 3.14所示，Java堆栈信息的报错结构通常由多个异常构成，每个异常都表现为异常名+细节信息+路径组成的结构，异常名从行首开始或紧随“Caused by”，之后的每一行都是路径中的一个位置。基于Java异常体系结构的先验知识，结合对堆栈信息报错结构分析，发现堆栈信息特点主要包括两个方面：（1）异常栈以FILO顺序打印，即最下方的堆栈异常最早被抛出，逐渐导致上方异常被抛出，因此最上方的堆栈异常是最晚被抛出且未被捕获。从上到下，第i+1个异常是第i个异常被抛出的原因，以“Caused by”开头。（2）对于一个

异常块，其中每一行异常代码是以FIFO的顺序打印，位于打印内容最上方的代码段是异常发生时调用的最内层代码，异常逐层向外抛出，因此最早经过的位置即是异常产生的根本原因，逆向Debug通常从此处开始。

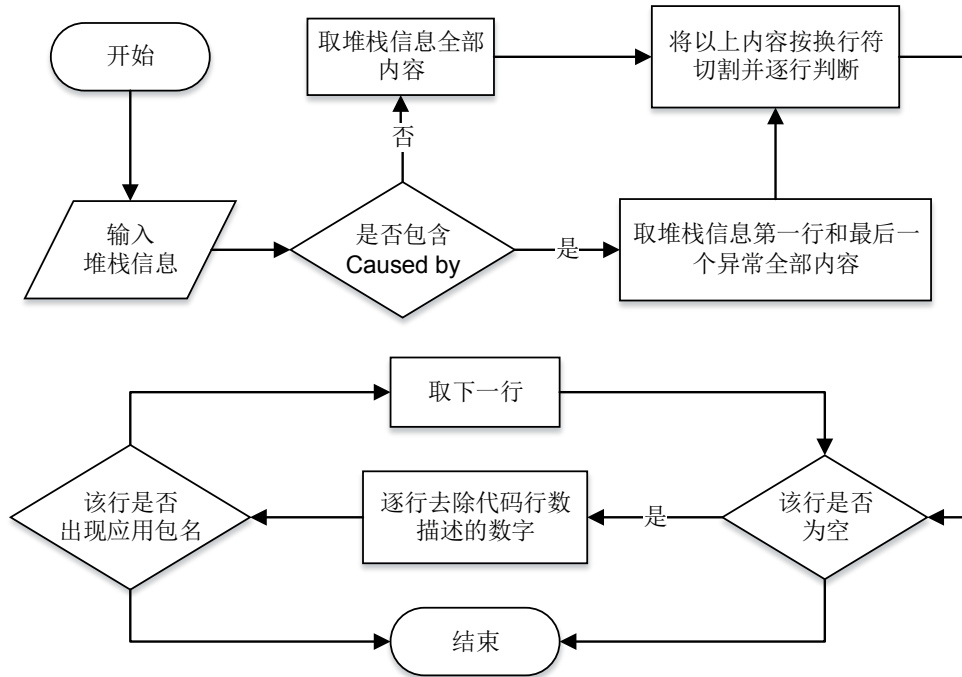


图 3.15: 堆栈信息模式匹配流程图

根据异常栈先进后出的特性，可以得知当堆栈信息中存在多个以“Caused by”开头的异常时，堆栈的最后一个异常才是造成崩溃的根本原因。根据一个异常块内代码段先进先出的特性，可以得知异常细节描述中首先出现与开发者编写代码有关的信息，如应用包名等，这些信息所在位置通常为真实错误所在位置。在此基础上，本文提出了对异常堆栈信息使用模式匹配来提取关键信息的方法：首先判断exception_stack中是否存在“Caused by”子句，若存在，则过滤重复堆栈信息内容，并将过滤后的内容作为当前异常堆栈信息的主要内容；否则将堆栈信息的全部内容作为当前异常堆栈信息的主要内容；最后对主要内容逐行去除对代码行数描述的数字，判断该行是否包括应用包名，若包括则直接结束循环；否则继续取下一行，直到当前行为空。其流程图如图 3.15所示。

如图 3.16所示，去重的过程需首先从request对象中获取崩溃数据，进而判断exception_stack是否为空，若不为空则从exception_stack中获取package_name，基于exception_stack进行模式匹配，将提取出的关键堆栈信息作为当前崩溃的sub_crash_desc字段。在本文中，利用package_name(应用包名)、sub_crash_desc

(堆栈关键信息)、`app_key`(应用唯一标识)、`app_version`(应用版本)这四个字段唯一标识一个崩溃，即从数据库中已有数据与当前崩溃进行不一致性对比。若`package_name`、`key_stack`、`app_key`、`app_version`四个字段完全相同，则忽略当前崩溃，仅更新当前崩溃的发生次数和最近发生时间；否则将其作为一条新的崩溃插入数据库。

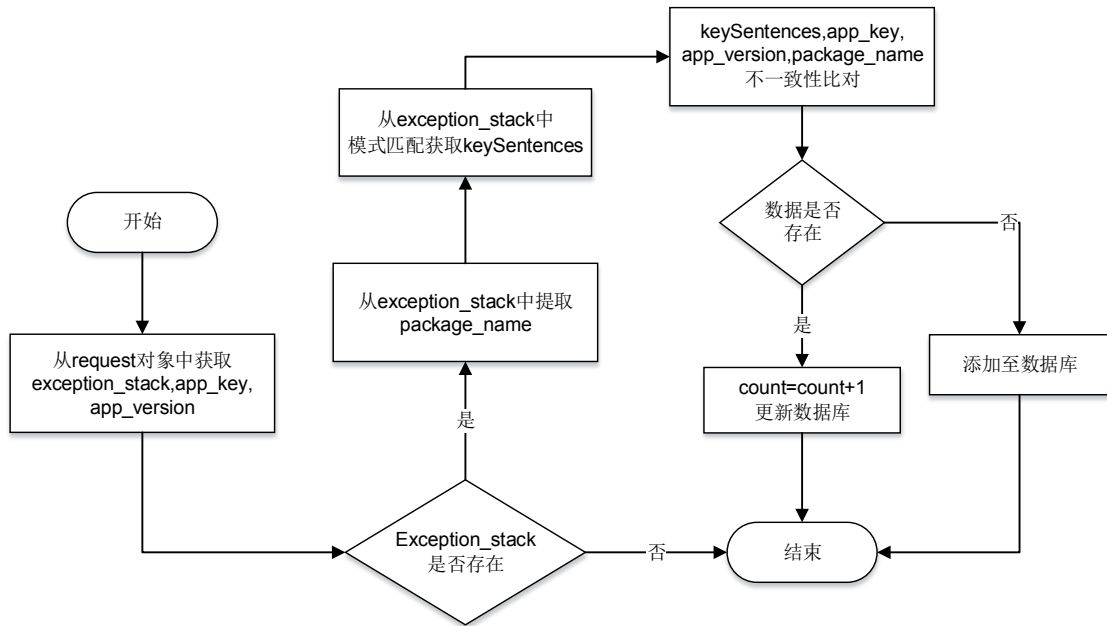


图 3.16: 崩溃去重设计流程图

3.4.2 崩溃分类模块设计

安卓应用主要基于Java语言编写，安卓异常堆栈结构跟Java异常堆栈十分类似。因此，正如Java异常一般，大部分崩溃堆栈信息中异常名都可直接作为其类别标签。但若使用形如`java.lang.RuntimeException`这种高层级异常类别名直接作为某个崩溃实例的类别标签，分类的粒度显然过于粗糙。如`java.lang.RuntimeException`类别下还包含了 `NullPointerException`、`IndexOutOfBoundsException` 等具体异常子类。对于这类仅根据异常名无法准确分类的崩溃数据，可根据除异常名外的堆栈信息对其进一步细分。此外，绝大多数异常描述中还会出现具体的类名、代码行数等影响因子，因此仅通过字符串匹配的方式无法适应多样化的异常堆栈结构。为解决这些问题，本文设计了如图 3.17所示的崩溃分类方案，主要包括数据收集、数据预处理、文本向量化、模型训练、结果预测和模型评估几个步骤。

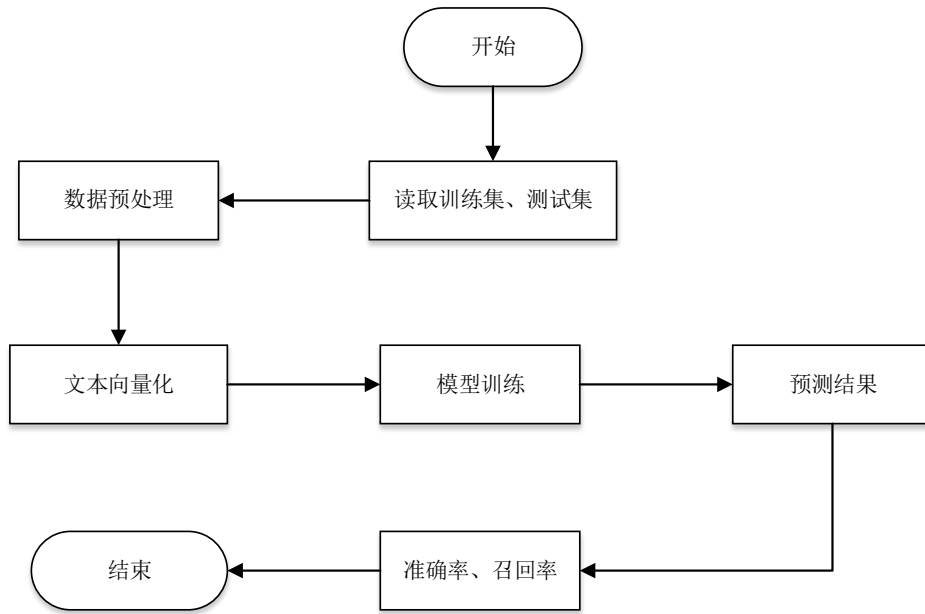


图 3.17: 崩溃分类设计流程图

崩溃数据分类是崩溃解析处理模块中的一个重要功能，分类的准确性对系统解决方案提示的性能指标有着决定性的影响。因此，如何选择合适的分类模型，对崩溃数据进行规则提取和分类预测，是崩溃解析处理中的一个关键任务。有别于本系统的其他功能性模块，崩溃数据分类模块主要基于机器学习朴素贝叶斯模型和支持向量机分类算法进行设计。

对本系统中的崩溃堆栈数据进行分类，首先确定分类模型中的特征变量和预测类别。本文利用自动化工具对开源应用进行自动化测试，收集logcat日志中的异常错误的堆栈信息来构建分类预测的训练集。根据异常栈FILO的特性可得，堆栈的最后一个异常才是崩溃发生的根本原因。同时，堆栈行首通常以异常名开头，且该行通常会详细描述异常错误原因。因此，收集的崩溃数据预处理过程主要为：（1）特征提取：堆栈中如果包含“Caused by”，则取堆栈中的最后一个异常的第一行作为崩溃特征；否则直接取堆栈中的第一行作为分类特征。（2）类型标注：对于大部分崩溃数据来说，如java.lang.ClassCastException等的类别标注可直接取崩溃特征中的异常名作为崩溃类别，但是对于小部分崩溃数据而言，其异常名通常描述范围太大，如java.lang.NullPointerException等，这类数据类别需进行人工审查，根据崩溃特征中详细的错误描述更改其类别。

实现特征提取和类别标注后，还需将特征转化为向量。在机器学习中，特征向量化主要包括词袋(Bag-of-Words, BoW)模型、TF-IDF模型以及哈希向量模

型。其中词袋模型是统计词汇的出现次数，TF-IDF不仅要统计词汇出现次数，还根据单词的权重来进行向量化。但是二者都存在生成的词汇表可能会非常庞大的限制。为解决该问题，本文利用由sklearn提供的HashingVectorizer方法将文本转换为数字，哈希向量化不需要词汇表，可以用任意长度的向量来表示。

建立训练集之后，分类模型需要对训练数据进行学习，即提取崩溃类别和异常堆栈之间的关联规则。常用的文本分类方法包括朴素贝叶斯和支持向量机。朴素贝叶斯分类器包含了朴素贝叶斯概率模型和最大后验决策准则，其基本原理是：对于给定的训练数据集，首先基于特征条件独立假设学习输入/输出的联合概率分布；然后基于此模型，对给定的输入x，利用贝叶斯定理求出后验概率最大即为最终分类结果。支持向量机二分类的基本原理是通过在特征空间寻求一个最优超平面将不同类别分割开，使得分类间隔最大化。在此基础上，通过不断组合二分类训练器来达到多分类的效果。

对于本系统中崩溃数据的分类问题，当应用朴素贝叶斯进行分类时，本文选择适合文本分类的多项式模型来进行分类，设置其alpha参数来训练分类模型并预测类别。当应用支持向量机进行分类时，本文选择由一对一法组合的SVC模型来进行分类，同时设置相应的核函数、惩罚系数及gamma值来训练分类模型并预测类别。最后通过计算预测结果的正确率和召回率来评估分类模型的可靠性。

3.5 崩溃可视化模块设计

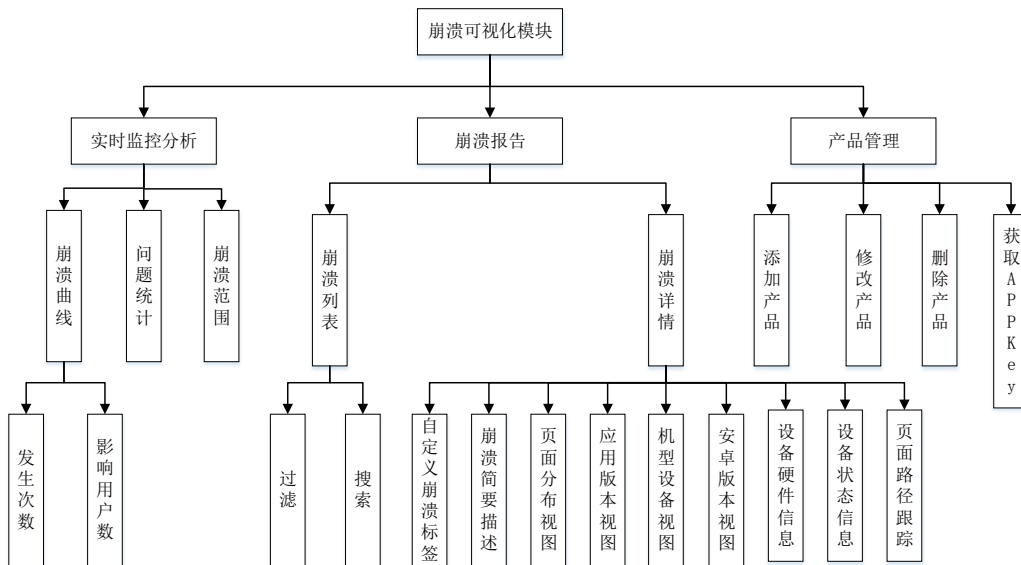


图 3.18: 崩溃可视化模块功能图

基于前文的需求分析，崩溃可视化模块不仅要实现应用管理的功能，还应从各方面丰富详细地展示崩溃解析报告和实时数据统计。如图 3.18 所示，主要包括实时监控分析模块、崩溃报告模块和应用管理模块。其中，崩溃报告模块可进一步分为崩溃列表和崩溃详情。

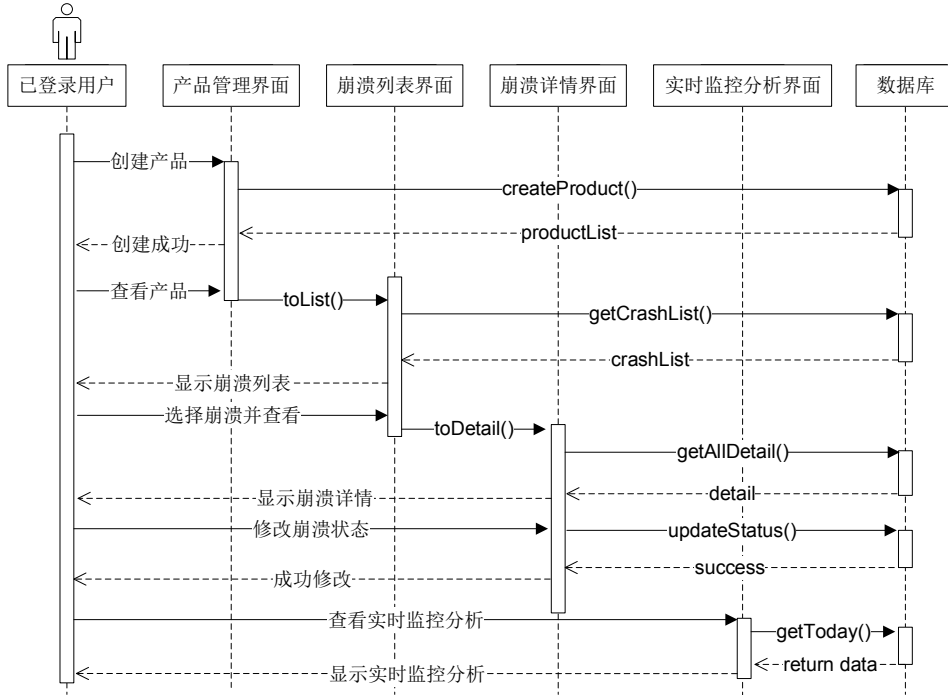


图 3.19: 崩溃可视化模块时序图

崩溃可视化模块的前后端交互将采用前后端分离的架构来实现。该模块的交互时序图如图 3.19所示，其主要交互接口包括createProduct、getCrashList、getAllDetail、getToday等，分别对应于产品创建、查看崩溃列表、查看崩溃详情、实时监控分析这四个功能。

已登录用户通过产品管理界面创建产品，填写应用名称、应用类型、应用描述等产品信息，然后发送RESTful请求，由Nginx和WSIG分发到对应视图函数进行处理。当视图函数处理完成后，会给前端返回HTTP响应，前端会根据HTTP响应结果自动请求产品列表并展示。当用户成功创建产品并按照配置将崩溃收集SDK嵌入至安卓应用后，系统会自动对去重和分类后的崩溃进行查询，进而展示详细的崩溃报告。同时，本系统为用户提供实时监控分析、查看崩溃列表、查看崩溃详情及产品管理等服务，这些服务都将通过发送RESTful请求来完成。下面将分别对实时监控分析模块、崩溃报告模块和产品管理模块的设计进行详细说明。

3.5.1 实时监控分析设计

实时监控分析模块用于展示被测应用在近24小时崩溃数据的统计。已登录用户在进入到实时监控分析页面后，系统会自动发送请求至后台，后台将统计后的数据以json为格式返回给前端，前端组件会自动根据状态和属性的变化完成页面渲染。该模块的展示方式包括实时崩溃数据、崩溃曲线、今日问题统计和今日排名Top5问题。

实时崩溃数据：统计被测应用在近24小时内的崩溃次数、影响用户数以及影响用户占比，可直观地看出当前应用出现故障的次数和被影响的用户数，以提示开发人员应及时对应用进行检测和修复。

崩溃曲线：统计用户使用被测应用的过程中在每个时间段（每小时或每天）内崩溃发生次数和被影响用户数，以折线图的方式展现。右上方用户可选择只查看某个时间段的数据统计，以帮助开发人员实时监控应用的运行状态。

今日问题统计：总问题数，已修复问题数和新发现问题数，以饼图的形式进行展示。总问题数是指当前应用已经发现的崩溃的数量，已修复问题数是指当前应用已经被修复的问题数，新发现问题数指的是当前崩溃的第一次发现时间小于24小时。今日问题统计能够帮助开发人员明确应用的修复情况，对新发现的崩溃及时地针对性修复。

今日排名Top5问题：指的是被测应用在近24小时内，发生次数最多的崩溃，以数据表格的形式展现。用户可点击每行进入到崩溃的详情页面，查看崩溃发生的代码行数、原因及解决方案。

3.5.2 崩溃报告设计

崩溃报告的前后端执行过程与实时监控分析相同，仅仅是数据和页面展示不同。崩溃报告主要包括崩溃列表和崩溃详情两部分内容，其中崩溃列表展示了当前App发生的所有崩溃的名称、最近上报时间、发生次数及影响的用户数，可根据条件对崩溃数据进行筛选。用户点击查看最新崩溃，可以进入崩溃详情页面。崩溃详情主要由崩溃概要、上报趋势、页面分布、应用版本、机型设备分布、所有崩溃实例几部分组成。

崩溃概要主要包括崩溃名称、崩溃报错原因及崩溃解决方案，用于提示开发者当前崩溃的类型、原因及相应的解决方案。本系统提供崩溃解决建议匹配与推荐功能，主要针对于数据库中已有的崩溃数据及解决方案，根据崩溃分类结果匹配对应系统提供的解决方案。除此之外，本系统提供用户提交解决方案的接口，用户可通过该接口提交个人解决方案，同时并开放推荐给其他用户。

同时，用户可点击右下方查看更多去查看其他用户提供的解决方案，每个用户可对解决方案进行点赞，系统将根据解决方案的点赞数从高到低排序展示。

上报趋势是当前崩溃在近48小时或近7天内分别以小时或天为时间间隔去统计每个时间段崩溃发生次数和影响用户数而绘制的折线图，可以直观地看出崩溃在各个时间段内的分布情况，以帮助开发者去分析是否某个时间段内出现了大规模的崩溃，从而针对性的查找问题和进行修复。

机型设备包括崩溃在不同机型设备或安卓版本上的分布情况，机型视图能够判断该崩溃是否跟机型或安卓版本有关，进一步针对特定机型或特定的安卓版本进行修复。

页面分布指的是同一崩溃在不同页面上的分布，页面视图能够根据一个崩溃去追溯类似的崩溃，进而针对性查找相似或相同的崩溃去一次性修复，避免遗漏。

所有的崩溃实例由一个表格构成，表格的每行可点击，实例展示崩溃详情日志，主要内容包括精简后的堆栈信息、用户页面路径跟踪、设备硬件信息和设备状态信息。

自定义崩溃标签是指用户通过更改崩溃状态为未处理、处理中和已处理，用户在将崩溃状态更改为已处理时，可选择官方解决方案，也可提交自己的解决方案。

3.5.3 产品管理设计

产品管理的前后端执行过程与实时监控分析相同，主要包括产品创建、产品修改及产品删除等功能。用户点击产品创建按钮，并填写产品名称、产品类型、产品描述、上传产品图标完成产品的创建。当用户创建产品后，用户可点击设置进入到产品修改界面，用户可在该页面修改产品名称、类型及描述，同时用户可以一键复制appKey，待下载SDK后，将SDK和appKey按照系统集成说明配置一并嵌入到被测应用中。

3.6 数据库设计

系统的数据库概念结构模型图用于表示实体、属性以及不同实体间的关系。根据概念模型生成本系统的实体关系(Entity-Relation, E-R)图[Chen, 1976]，如图 3.20所示。该模型的实体主要包括用户信息、产品信息、崩溃信息，崩溃去重、解决建议、用户信息等，属性即实体所具有的特性，所有的实体可通过不同的关系联系在一起。

数据库表设计将根据实体关系图分解并重新组合构成数据库结构，主要包括用户信息表、产品信息表、崩溃信息表、崩溃去重表、官方解决建议表、用户解决建议表以及用户点赞表。

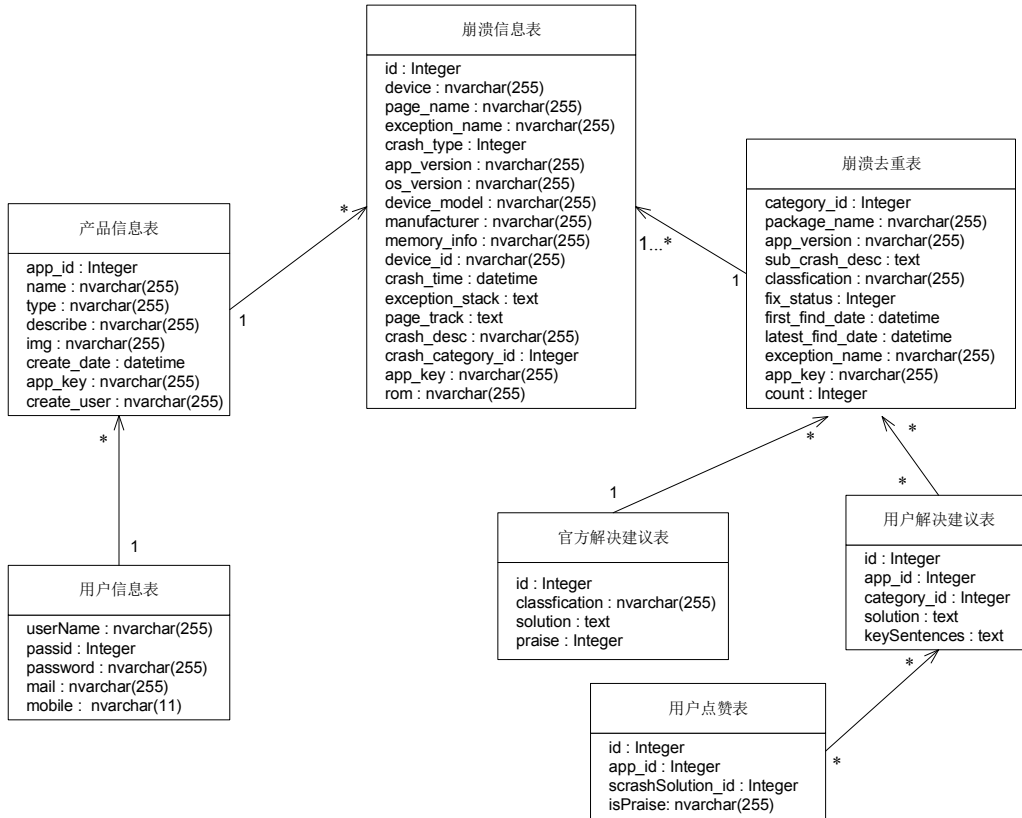


图 3.20: E-R图

产品信息表用于记录用户创建应用时去记录一些App的信息，主要包括app_key、name、create_time、create_user等。其中，app_key是服务器根据应用名称和创建时间自动生成的，用于唯一标识应用，name是应用的名称，create_time是应用创建时间，create_user是创建应用的用户。产品信息表详情参见表 3.7。

崩溃信息表主要用于记录应用在使用过程中发生崩溃相关信息，如当前的堆栈信息、设备硬件信息及设备性能状态信息，具体包括安卓版本、手机型号、渠道号、厂商、当前App版本、应用包名、崩溃名称、系统设备等。其中，device_id是设备型号，用于唯一标识一个用户；crash_category_id是崩溃所属类别；page_track是崩溃的用户页面路径跟踪；os_version是设备系统版本；manufacturer是设备厂商；exception_stack是崩溃堆栈信息。该表中的崩溃数据可以重复，详细的崩溃信息参见表 3.8。

表 3.7: 产品信息表

字段	类型	默认值	说明
app_id	Integer	0	应用id
name	nvarchar	NULL	应用名称
type	nvarchar	NULL	应用类型
describe	nvarchar	NULL	应用描述
app_key	nvarchar	NULL	应用唯一标识
img	nvarchar	NULL	应用图标
create_time	datetime	NULL	创建时间
create_user	nvarchar	NULL	创建用户

表 3.8: 崩溃信息表

字段	类型	默认值	说明
id	integer	0	崩溃id
device	nvarchar	NULL	设备
package_name	nvarchar	NULL	应用包名
crash_type	integer	NULL	崩溃是否被捕获
app_version	nvarchar	NULL	应用版本
os_version	nvarchar	NULL	系统版本
device_model	nvarchar	NULL	手机型号
manufacturer	nvarchar	NULL	厂商
device_id	nvarchar	NULL	手机唯一标识
memory_info	nvarchar	NULL	内存占用
crash_time	datetime	NULL	发生时间
exception_stack	text	NULL	堆栈信息
page_track	text	NULL	路径跟踪
classification	nvarchar	NULL	崩溃类型
crash_category_id	integer	NULL	崩溃类型id
app_key	nvarchar	NULL	应用唯一标识
rom	nvarchar	NULL	ROM

崩溃去重表用于记录应用的所有不同的崩溃，主要包括sub_crash_desc、fix_status、classification、first_find_date、latest_find_date等字段。其中，sub_crash_desc指的是从堆栈中提取的关键信息，用于标识当前崩溃的唯一性；fix_status指崩溃的修复状态，包括未处理、已处理和正在处理三个状态；classification为崩溃类别；first_find_date为崩溃首次发现时间；latest_find_date为崩溃最近发生时间。崩溃去重表详情参见表 3.9。该表中不存在sub_crash_desc、package_name、category_id、app_version四个字段完全一致的数据，即所有数据都不重复。

表 3.9: 崩溃去重表

字段	类型	默认值	说明
category_id	integer	0	类别id
package_name	nvarchar	NULL	应用包名
app_version	nvarchar	NULL	应用版本
sub_crash_desc	text	NULL	堆栈关键信息
classification	nvarchar	NULL	类别
fix_status	integer	NULL	修复状态
first_find_date	datetime	NULL	第一次发现时间
latest_find_date	datetime	NULL	最后一次发现时间
exception_name	nvarchar	NULL	崩溃名称
app_key	nvarchar	NULL	应用唯一标识
count	integer	NULL	发生次数

3.7 本章小结

本章基于对国内外现有崩溃分析平台的调研，结合真实场景下安卓开发团队的实际需求，通过对整个系统的功能性需求和非功能性需求的分析，设计了以崩溃收集模块、崩溃解析处理模块和崩溃可视化模块组成的总体架构，并从这三个模块的功能上进行了详细设计。数据库设计中，对于系统中所需要的数据库表进行了详细的说明，将持久化的数据存储于关系型数据库中，保证了数据的一致性。

第四章 面向安卓应用的崩溃信息线上分析系统实现与测试

本章基于系统架构设计的需求分析和功能设计要求，重点针对崩溃收集模块、崩溃解析处理模块及崩溃可视化模块的内部实现进行详细说明，同时对系统的功能和性能方面进行了测试，并设计了相关实验来分析和验证系统的可靠性和可用性。

4.1 崩溃收集模块的实现

基于崩溃收集模块的需求分析和功能设计，本节使用安卓异常处理机制和面向切面编程技术收集崩溃相关数据，最后利用OKHttp3包将崩溃数据进行上传。下文将针对监听Activity生命周期和安卓全局崩溃的捕获两个模块的实现进行详细说明。

4.1.1 监听Activity生命周期

监听Activity生命周期，即监听Activity生命周期的关键方法，包括 onCreate()、onStart()、onResume()、onPause()、onStop()及onDestory()。安卓4.0以上版本支持调用Application的registerActivityLifecycleCallbacks接口来监听所有的Activity的生命周期，可以在该接口方法里根据需要添加相应功能。因此安卓版本4.0以上的手机设备，可通过调用Application的 registerActivityLifecycleCallbacks 接口来实现页面路径跟踪。

表 4.1: 监听Activity生命周期类表

类名	功能说明
IActivityLifecycleCallbacks	声明Activity生命周期回调接口
ActivityLifeManager	继承IActivityLifecycleCallbacks，用于管理Activity生命周期
MyInstrumentation	重写Activity生命周期方法
MyActivityLifecycleCallbacks	继承ActivityLifecycleCallbacks，重写Activity生命周期回调接口
HookActivity	实现监听Activity生命周期
ScreenShot	实现截图功能

Instrumentation类拥有强大的跟踪Application及Activity生命周期的功能。针对安卓版本4.0以下的手机设备，可通过自定义MyInstrumentation类继承 Instrumentation 类，并重写Activity生命周期的方法，最后通过Java反射机制将系统原

本的Instrumentation替换来实现监听Activity生命周期的功能。Activity生命周期监听实现的相关类如表 4.1所示。

```
public class replaceInstrumentation (){
    //获取当前进程名activityThreadClass
    Class activityThreadClass = Class.forName("android.app.ActivityThread");
    Method method = activityThreadClass
    getDeclaredMethod("currentActivityThread");
    Object currentActivityThread = method.invoke(null);
    Field field = activityThreadClass.getDeclaredField("mInstrumentation");
    field.setAccessible(true);
    MyInstrumentation mInstrumentation = new MyInstrumentation();
    //覆盖Instrumentation类
    field.set(currentActivityThread, mInstrumentation);
}
```

图 4.1: 替换Instrumentation类实现关键代码

利用反射机制替换Instrumentation类的具体流程是，首先显示加载ActivityThread类，并获得其Class对象作为反射的入口，通过Class.getDeclaredMethod()方法获取静态方法currentActivityThread的Method对象，再调用Method.invoke()方法执行反射并获取currentActivityThread方法的返回值，最后调用 Filed.set ()方法覆盖currentActivityThread对象中的Instrumentation类。关键代码如图 4.1所示。

```
public class HookActivity (){
    1. IActivityLifecycleCallbacks声明Activity生命周期回调接口
    2. ActivityLifeManager继承Activity生命周期回调接口并重写
    3. 调用ActivityLifeManager实现MyInstrumentation类
    4. 继承Application.ActivityLifecycleCallbacks实现MyActivityLifecycleCallbacks
    5. if 安卓版本 > 4.0
    6.     app. RegisterActivityLifecycleCallbacks(new MyActivityLifecycleCallbacks())
    7. else
    8.     MyInstrumentation替换Instrumentation
    9. 统一调用重写后的生命周期方法
}
```

图 4.2: 监听Activity生命周期Java风格伪代码

其中，需要重写Activity生命周期所有方法，包括onActivityCreated、onActivityResumed、onActivityPaused、onActivityDestroyed、onActivityCreated、onAc-

tivityStoped，以实现对Activity的启动、暂停、恢复、销毁等状态变化的全过程监听。在各生命周期状态函数中，需要获取当前设备时间、Activity名称，Activity当前状态以及调用ScreenShot.shoot()函数截取当前页面，并将这些信息写入设备本地文件。监听Activity生命周期的伪代码如图 4.2 所示。

4.1.2 安卓全局崩溃捕获

安卓全局崩溃捕获的实现原理是修改应用的默认异常处理程序 Default-UncaughtExceptionHandler 为自定义异常处理器。自定义异常处理器继承自 Thread.UncaughtExceptionHandler，并重写其uncaughtException方法，最后将自定义异常处理器设置为应用程序级的全局方法，在应用程序全局范围内调用该方法，在安卓应用Application类中传入App上下文即可完成配置。自定义异常处理器的异常处理方法uncaughtException中需添加以下的自定义功能：（1）崩溃发生后，不立即退出应用，等待3s，并弹出友好的异常提示。（2）收集应用程序的崩溃相关信息，如异常堆栈等，并将其都写入变量和设备本地文件。（3）上传至服务器，若上传服务器成功则删除本地文件，否则在下次启动应用的时候会进行本地崩溃文件扫描，重新上传未成功上传的崩溃文件。

每个应用程序在启动时都会首先创建一个Application类并执行该类中的onCreate()方法，进而再启动相应的Activity和Service。其中的onCreate()方法在应用程序的整个生命周期内只会执行一次。因此，将自定义异常处理程序设置为应用程序级的全局方法可通过在onCreate()方法中创建并初始化来实现。

表 4.2: 安卓全局崩溃捕获函数

类或函数	功能说明
collectDeviceInfo()	收集与崩溃相关的设备信息
getStackTrace()	收集堆栈信息
saveCrashInfo2File()	保存错误信息至文件
handleException()	收集错误信息并上传至服务器
uncaughtException()	回调函数，提示用户应用异常
ossUtil	OSS工具类，封装OSS上传函数
UploadFile	用于上传崩溃信息

安卓全局崩溃捕获相关类和函数如表 4.2所示。崩溃发生时，首先由自定义异常处理器捕获异常，并调用uncaughtException方法来进行处理。在uncaughtException中调用了handleException方法来进一步对崩溃信息进行收集、保存和上传。handleException通过collectDeviceInfo方法收集发生崩溃的设

备硬件和设备状态性能信息；使用`getStackTrace()`获取`Throwable`对象的抛出的堆栈信息；利用`saveCrashInfo2File()`将崩溃信息保存至设备本地；最后调用`UploadFile`类新起一个线程上传崩溃信息和崩溃截图。

表 4.3: 收集的崩溃信息

变量名	解释说明
<code>device</code>	设备型号
<code>package_name</code>	应用包名
<code>exception_name</code>	崩溃名称
<code>crash_type</code>	崩溃类型
<code>app_version</code>	应用版本
<code>os_version</code>	Android系统版本
<code>device_model</code>	手机型号
<code>manufacturer</code>	厂商
<code>memory_info</code>	内存占用
<code>exception_stack</code>	Stack堆栈信息
<code>rom</code>	ROM
<code>device_id</code>	设备id
<code>channelId</code>	渠道号

通常来讲，应用源代码错误、设备硬件不兼容和应用运行时设备性能限制都可能导致安卓应用崩溃的发生。因此，为了给开发者提供更多与崩溃相关的有效信息，本文收集的崩溃信息包括异常堆栈信息、设备硬件信息及设备性能状态信息等，具体内容参见表 4.3。下面就设备硬件信息和设备性能状态信息收集的具体实现进行详细说明。

设备硬件信息获取：安卓系统自带一个`android.os.Build`类，通过引入该类即可调用相应的信息参数。该类获取本机设备硬件参数的基本原理是在运行时通过Java的反射机制实现。`Build`类包括设备的所有信息参数，本文只收集可能造成硬件不兼容的相关参数，如系统版本、ROM等，具体内容可参见表 4.3。

设备性能状态信息获取：获取系统内存信息通过将`Context.ACTIVITY_SERVICE`作为参数传入`context.getSystemService()`方法创建`ActivityManager`对象实现。`ActivityManager`能够获取当前运行程序的内容，其主要负责`ActivityThread`的创建，维护`Activity`的生命周期。同时，它提供了`getMemoryInfo()`方法获取`MemoryInfo`实例，其中`totalMem`、`availMem`和`lowMemory`分别对应设备总内存、剩余内存和设备是否处于低内存运行状态。通过计算`totalMem`与`availMem`的差值即可获得设备的内存占用，内存占用率指的是内存占用与总内存的比率。获取内存占用率的关键代码如图 4.3 所示。


```
Private float getMemory(Context ctx){
    //初始化ActivityManager
    ActivityManager am = (ActivityManager) ctx
        .getSystemService(Context.ACTIVITY_SERVICE);
    String runningActivity = ctx.getClass().getName();
    //获取系统内存参数
    MemoryInfo mi = new MemoryInfo();
    am.getMemoryInfo(mi);
    //计算内存占用率
    float ratio = (float) ((mi.totalMem-mi.availMem)*100.0/(float)mi.totalMem);
    return ratio;
}
```

图 4.3: 获取内存占用率关键代码

4.2 崩溃解析处理模块的实现

基于崩溃解析处理模块的需求分析和模块设计，该模块基于Flask框架，通过模式匹配提取关键堆栈信息来分析数据不一致性，实现崩溃数据去重；使用sklearn中朴素贝叶斯MultinomialNB模型和支持向量机SVC模型实现崩溃数据分类。以下将针对崩溃去重和崩溃分类的实现进行详细说明。

4.2.1 崩溃去重实现

基于崩溃去重模块的功能设计，本节将对崩溃去重的实现进行详细说明。首先应对崩溃的堆栈信息进行模式匹配提取关键信息，再对崩溃进行去重。去重过程包括以下四个步骤。其中特征匹配方法的代码如图 4.4所示。

(1) 判断堆栈信息中是否出现“Cause by”子句，若包含“Cause by”则只截取堆栈信息的最后一个“Cause by”之后的所有内容赋值给causeBy变量，再对causeBy变量正则匹配“…\d more”，截取“…\d more”以后的所有内容并赋值给mainReason变量；否则取堆栈信息的全部内容赋值给mainReason变量。

(2) 对步骤(1)中的mainReason变量按换行符进行切割，逐行判断该行是否包含应用包名，若包含应用包名，则跳出循环到步骤(3)；否则将该行有关提示代码行数的数字去除，并将其添加至keySentences变量末尾。

(3) 返回keySentences变量，到步骤(4)。

(4) 对步骤(3)返回的keySentences(堆栈关键信息)结合当前崩溃的package_name(应用包名)、appKey(应用唯一标识)、app_version(应用版本)等属性与

崩溃类型表中的数据进行比对，若表中存在与当前崩溃的四项属性完全一致的数据，则只更新当前崩溃的最近发生时间及发生次数；否则应向崩溃去重表中执行insert操作来添加新的崩溃数据。

```
def getKeySentences(exception_stack,package_name):
    #判断有无Caused by取最后一个异常
    if "Caused by: " in exception_stack:
        causeBy = exception_stack.split("Caused by: ")[-1]
    #过滤重复字段
    temp = re.split(r'... \d+ more', causeBy)[-1]
    mainReason = exception_stack.split("\n")[0]+\nCaused by: '+temp
    else:
        mainReason = exception_stack
    #变量初始化，将主要内容按换行符进行分割
    list = mainReason.split("\n")
    keySentences = list[0]+\n"
    #按行遍历，去除数字等代码描述
    for i in range(1,len(list)):
        if "at "+package_name in list[i] and ".java:" in list[i]:
            break
        else:
            newLine = re.sub(r'java:(\d){1,}','java:', list[i])
            keySentences = keySentences + newLine +'\n'
    return keySentences
```

图 4.4: 堆栈信息模式匹配代码

4.2.2 崩溃分类实现

崩溃类别是向开发者提供崩溃展示时的关键信息，对于不同类别有针对性解决方案，准确的崩溃分类可以帮助开发者迅速定位并解决应用程序问题。因此应首先确保分类模型的准确性，本文设计了如下实验过程对分类模型的可靠性进行评估。

本文利用自动化工具对20个开源应用进行自动化测试，收集测试设备logcat日志中异常堆栈信息，对收集数据进行特征提取及类别标注，并将标注后数据哈希向量化。实验数据集包括45类错误，共539条数据，按3:1的比例将其分为训练集和测试集。最终训练集共360条数据，测试集共179条数据。朴素贝叶斯和支持向量机分类算法均采用sklearn包中的算法实现。

朴素贝叶斯分类实验过程及结果：使用多项式模型MultinomialNB实现崩溃分类，将alpha设置为0.01，再将训练集和测试集数据分别用于训练和预测，最终预测的正确率为0.821，召回率为0.853。

支持向量机分类实验过程及结果：使用一对一法组合的SVC模型实现崩溃分类，将kernel设置为linear，C设置为3，gamma设置为1，再将训练集和测试集数据分别用于训练和预测，最终预测的正确率为0.931，召回率为0.941。

```
def classfiy(exception_stack,train_words,train_tags):
    #数据预处理
    data = handleData(exception_stack)
    #哈希向量化
    v = HashingVectorizer(n_features=200, non_negative=True)
    train_data = v.fit_transform(train_words)
    test_data = v.fit_transform(data)
    #支持向量机SVC模型训练
    model_svm = svm.SVC(kernel='linear', C=3, gamma=1)
    model_SVM = model_svm.fit(train_data, numpy.asarray(train_tags))
    #类别预测
    result_SVM = model_SVM.predict(test_data)
    return result_SVM
```

图 4.5: 崩溃分类实现的关键代码

根据上述的正确率结果，本文最终选取支持向量机算法作为崩溃数据的分类算法。在本文中，崩溃分类使用sklearn包提供的SVM.SVC模型实现。崩溃分类关键实现代码如图 4.5所示。首先调用handleData()实现数据预处理过程；将处理后的数据作为HashingVectorizer.fit_transform()的输入，将其转化为向量并返回；再调用SVM.SVC().fit()实现分类模型的训练并返回训练好的模型；然后调用训练好的模型的predict()函数预测当前崩溃的类别，最后将预测结果作为最终类别并返回。

此外，本文还针对这45类错误分别收集了对应的崩溃解决方案，如表 4.4所示，崩溃类别与解决方案一一对应。比如崩溃类型Attempt to invoke virtual method on a null object reference，该类型错误的发生通常因使用某个对象的方法时，未考虑对象为空时如何处理。针对这些常见的异常错误类型，收集其对应的解决方案，以便开发者查看崩溃报告时快速根据解决方案进行修复。

表 4.4: 部分崩溃的类别与解决方案

类别	解决方案
OutOfBoundsException	在遍历一个数组/集合时, 要预判数组/集合是否为空; 使用数组/集合中的元素时, 预判数组/集合长度。
Attempt to invoke virtual method on a null object reference	使用一个对象的某个方法时, 对象为空, 没有实例化。凡是调用一个对象的方法之前, 一定要进行判空或者进行try-catch, 这样基本可以规避大部分空指针异常。
ClassCastException	这类问题都是由于强制类型转换导致的, 可通过使用安全类型转换函数, 即为其指定转换失败时的默认值。
Unable to add window – permission denied for this window type	此异常多发于WindowManager自定义弹出框时, 没有设置权限。可通过在AndroidManifest.xml配置文件中添加uses-permission。
doInBackground	doInBackground()这个方法运行在工作线程中, 不允许做UI显示。常出现于使用AsyncTask做网络数据请求。
.....

4.3 崩溃可视化模块的实现

崩溃可视化模块以展示崩溃报告和用户交互界面为主, 基于RESTful架构, 通过接口访问数据。前端使用React的this.setState()函数修改变量, React会自动调用shouldComponentUpdate方法去根据State和Props的变化来判断页面是否需要重新渲染, 由此实现页面的自动刷新。该模块主要分为实时监控分析、崩溃列表、崩溃详情以及产品管理。以下将分别对这四个功能模块进行详细说明。

4.3.1 实时监控分析

实时监控分析的API接口为getToday, 该接口用于展示当前应用在近24小时内发生的崩溃, 主要包括今日实时崩溃数据、今日问题统计及崩溃趋势上报, 分别对应于getTodayTab()、getTodayTop5()、getTodayPie()函数。其中, getTodayProblem()用于统计今日崩溃发生总问题数、已修复问题数、新发现问题数; getTodayTop5()统计今日崩溃次数发生次数最多且排名Top5的崩溃; getTodayTab()用于实时展示崩溃发生次数、崩溃用户数和影响用户占比。因篇幅限制, 仅展示getTodayProblem()函数实现的伪代码, 如图 4.6所示。

除此之外, 实时监控分析还需展示今日崩溃上报趋势。崩溃上报趋势的视图函数为getTrend(), 主要用于统计崩溃在不同时间段的上报次数及影响用户数。根据需求设计, 用户可自定义时间段查看该段时间崩溃上报趋势, 因此应将用户自定义时间time作为参数传入到getTrend()函数中。首先在数据库查询大于time的所有数据, 并将crash_time格式化为以小时/天数为单位; 再

```
def getTodayProblem(lastDay)
    1. 从崩溃类型表中查询latest_find_date大于lastDay的数据并统计次数，将结果存储在total变量中
    2. 从崩溃类型表中查询first_find_date大于lastDay的数据并统计次数，将结果存储在new变量中
    3. 从崩溃类型表中查询first_find_date大于lastDay且fix_status=2的数据并统计次数，将结果存储在repair变量中
    4. 初始化统计变量
    5. pie.total = total    if total    else:pie.total = 0
    6. pie.new = new      if new      else:pie.new = 0
    7. pie.repair = repair if repair   else:pie.repair = 0
```

图 4.6: 今日问题统计接口实现的Python风格伪代码

按device_id(用户唯一标识)和crash_time分组并统计该时间段内当前用户的崩溃发生次数；最后根据时间段统计崩溃的发生次数以及影响的用户数。其伪代码如图 4.7所示。

```
def getTrend(time):
    1. 从数据库查询大于time的所有数据，查询结果存储在数组data数组中；
    2. result数组存放统计的结果数据
    3. 初始化统计变量
    4. if len(data)>1:
    5.     for i in range(1,len(data)):
    6.         if 本条记录 in 当前统计的时间段
    7.             增加当前时间段发生次数、影响用户数
    8.         else:
    9.             result.append(当前时间段统计数据)
    10.        初始化统计变量
    11.        if i == len(data)-1:
    12.            result.append(当前时间段统计数据)
    13.    else:
    14.        result.append(统计数据初始值)
    15.    return result
```

图 4.7: 崩溃上报趋势接口实现伪代码

实时监控分析的实现界面如图 4.8 所示，用户可根据崩溃上报趋势图查看应用在各个时间段的整体运行状况，可依据今日问题统计的掌握当前应用的修复情况和新发现问题数，还可点击查看今日问题排名前五的崩溃详情。

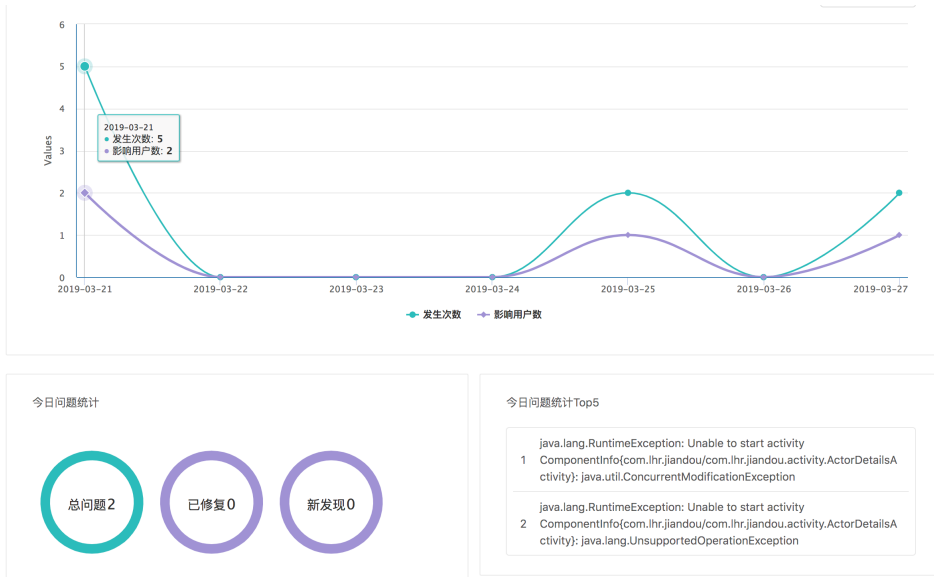


图 4.8: 实时监控分析界面

4.3.2 崩溃列表

```
def geCrashList(fix_status,classification,app_name):
    1. if fix_status!=None and fix_status!= '4'
    2.     data = 查询对应状态的crash
    3. if fix_status=='4'
    4.     data = 查询所有状态的crash
    5. for crash in data:
    6.     if roughCategory(crash)==classification
    7.         result.append(处理后的crash)
    8. return result
```

图 4.9: 崩溃列表接口实现伪代码

崩溃列表的API接口为getCrashList，用于查询当前应用中符合当前查询条件的发生的所有崩溃。该接口参数为fix_status，classification，app_name。其中，fix_status表示崩溃修复状态，包括已处理、未处理和处理中三个状态，

classification表示用户选择的崩溃类别，app_name表示当前应用名称。该接口将根据HTTP请求这三个参数实现对崩溃数据的过滤查询，崩溃列表数据获取实现的伪代码如图 4.9所示，其中，roughCategory()用于判断当前classification所属大类，主要包括Java语法相关异常、Activity相关异常、序列化相关异常、列表相关异常、窗体相关异常、资源相关异常、系统碎片化相关异常、SQLite相关异常、其他情况等9个大类[37]，用户可选择这些类型实现对异常的过滤。

异常名称	最近上报	发生次数	影响用户数
<input type="checkbox"/> 1 - java.lang.NumberFormatException 1.1 java.lang.RuntimeException: Unable to start activity ComponentInfo{com.lhr.jiandou/com.lhr.jiandou.activity.ActorDetailsActivity}: java.lang.NumberFormatException: Invalid int: "123xxx45"	2019-03-21 12:50:09	3	2
<input type="checkbox"/> 2 - java.util.ConcurrentModificationException 1.1 java.lang.RuntimeException: Unable to start activity ComponentInfo{com.lhr.jiandou/com.lhr.jiandou.activity.ActorDetailsActivity}: java.util.ConcurrentModificationException	2019-03-21 12:35:18	1	1
<input type="checkbox"/> 3 - java.lang.UnsupportedOperationException 1.1 java.lang.RuntimeException: Unable to start activity ComponentInfo{com.lhr.jiandou/com.lhr.jiandou.activity.ActorDetailsActivity}: java.lang.UnsupportedOperationException	2019-03-21 12:52:22	1	1

图 4.10: 崩溃列表界面

崩溃列表实现界面如图 4.10所示，以表格的形式进行展现，包括异常名称、最近上报时间、发生次数及影响用户数。异常名称第一行是崩溃名称及当前应用版本，第二行是崩溃发生原因。用户可点击最上方的选择框切换应用；可选择崩溃状态进行条件过滤，包括未处理、处理中和已处理；可选择崩溃类型进行条件过滤；也可输入异常名称直接搜索。

4.3.3 崩溃详情

崩溃详情API接口主要方法如表 4.5所示。崩溃详情用于展示当前崩溃的详细报告：简要详情、页面分布、机型设备分布、应用版本、上报趋势和错误日志详情。基于HighCharts插件，绘制了折线图、散状图、饼图来描述崩溃分布状况。getCurrent崩溃用于获取当前崩溃的详情，包括异常名、异常概要、发生次数、影响用户数、崩溃解决方案；insertPraise实现用户对第三方提交解决方案进行点赞的功能；getPageDistribution用于获取当前崩溃在不同页面的分布，包括出错页面、报错代码行数和崩溃瞬间截图；getTrend用于获取当前崩溃上报趋势，包括不同时间段内崩溃的发生次数及影响用户数；getDeviceVersion用于获取当前崩溃在不同机型及安卓版本的分布；updateStatus用于自定义崩溃标

签，可更改崩溃状态，若选择自定义解决方案，用户需提交自己的解决方案；`getSolution`用于获取当前崩溃类别的解决方案；`getOtherSolution`用户获取第三方提交的解决方案；`getTartgetInstance`用于获取当前崩溃的所有实例，展示详细的设备信息、堆栈信息和用户页面跟踪等。

表 4.5: 崩溃详情接口表

接口	功能描述
<code>getCurrentCrash</code>	获取当前崩溃详情
<code>insertPraise</code>	实现点赞功能
<code>getOtherSolution</code>	获取第三方提交的解决方案
<code>getTrend</code>	获取当前崩溃上报趋势
<code>getDeviceVersion</code>	获取当前崩溃机型设备视图
<code>updateStatus</code>	自定义崩溃标签，更改崩溃状态
<code>getSolution</code>	获取当前崩溃类别解决方案
<code>getOtherSolution</code>	获取第三方提交的解决方案
<code>getTargetInstance</code>	获取当前崩溃的有实例

崩溃上报趋势的实现与实时监控分析中的崩溃上报趋势的实现相同，其不同点是实时监控分析统计的是整个应用的实时上报数据，而崩溃上报趋势统计的是当前崩溃的上报数据。下面就页面分布这个接口进行详细说明，主要展示简要详情、机型设备分布、错误日志详情的界面。

```
def get_xxx_page_distribution():
```

1. 从崩溃类型表中查询相似/相同的数据，并存入`data`变量中
2. `result`数组存放统计的结果数据
3. 初始化统计变量;
4. `for crash in data:`
5. 定位代码行数和提取报错页面，将类型标志为相似/相同
6. `result.append(处理后的crash)`
7. `return result`

图 4.11: 页面分布接口实现伪代码

页面分布接口实现伪代码如图 4.11所示。页面分布视图中包括相同和相似的崩溃。相同的崩溃指的是崩溃的`sub_crash_desc`(堆栈关键信息)字段完全一致，即`category_id`相同；相似的崩溃指的是`category_id`不同，但`classification`(类别)字段完全一致。因此，该接口包括`get_different_page_distribution()`和`get_sameClass_`

page_distribution() 两个函数。两个函数步骤基本一致，都是先从崩溃信息表中查询到相同或相似的数据，再将数据的exception_stack字段作为参数调用targetingLines()函数定位代码行数，调用getPage()函数提取报错页面，最后标记崩溃的类型为相同或相似。

页面分布实现界面如图 4.12所示。该界面以表格形式展示了与当前崩溃相同或相似的数据在不同页面的分布。表格每行展示了详细报错原因及出错代码行数，用户可以点击查看崩溃截图，以帮助开发者重现崩溃场景。针对类型为similar的崩溃，用户可点击当前行进入到相似崩溃的详情界面，实现追溯修复相似崩溃的功能，减少重复工作时间。

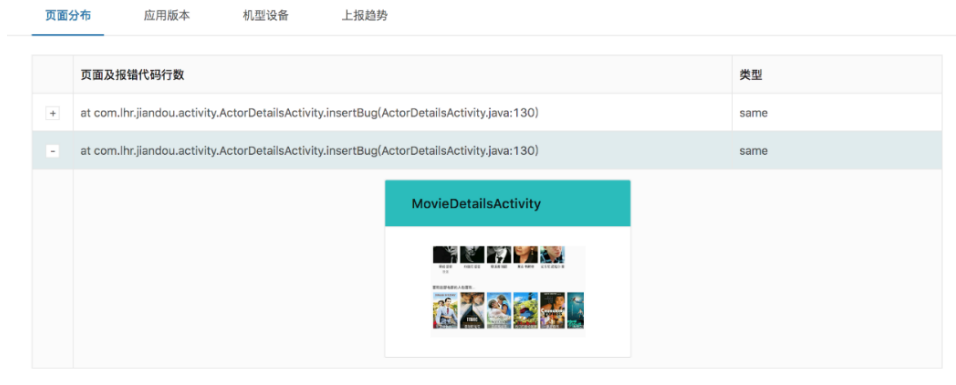


图 4.12: 页面分布界面

简要详情如图 4.13所示。该界面主要展示了崩溃名称、报错原因、发生次数、影响用户数及解决方案，用户可点击最下方查看更多去查看第三方解决方案，并根据有用程度进行点赞。用户可点击状态更改按钮来更改崩溃状态，若将状态更改为已处理，用户应提交自己的解决方案。



图 4.13: 简要详情界面

机型设备分布界面如图 4.14所示。该界面以饼图的形式展现了当前崩溃在机型设备及安卓版本上的分布，包括详细的占比和发生次数等，以帮助开发者针对特定机型或安卓版本进行修复。

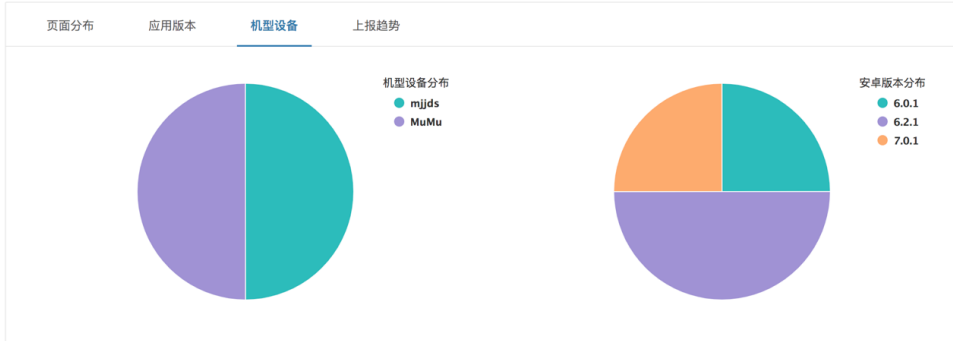


图 4.14: 机型设备分布界面

错误日志详情如图 4.15所示。错误日志详情左边以表格展示了当前崩溃发生的所有实例，右边则详细展示当前实例的所有设备和应用相关的信息，如应用包名、系统版本、内存占用等。可点击tab按钮切换查看堆栈详细信息和用户页面路径跟踪信息。



图 4.15: 错误日志详情界面

4.3.4 产品管理

产品管理API接口主要方法如表 4.6所示。产品管理主要包括产品列表、产品创建和产品修改等界面。其中creatProduct用于创建产品；updateProduct用于

修改产品信息；`deleteProduct`用于产品删除；`getProduct`用于获取所属用户的所有产品；`getCurrentProduct`用于获取当前产品的所有信息。

表 4.6: 产品管理接口表

接口	功能描述
<code>createProduct</code>	产品创建
<code>updateProduct</code>	产品修改
<code>deleteProduct</code>	删除产品
<code>getProduct</code>	获取所有产品
<code>getCurrentProduct</code>	获取当前产品所有信息

其中产品修改的界面如图 4.16所示。用户可通过点击产品列表的设置按钮可进入到当前产品的修改界面。该界面可以修改产品名称、产品图标、产品类型、产品描述等，此外用户还可复制`appKey`，用于绑定产品。

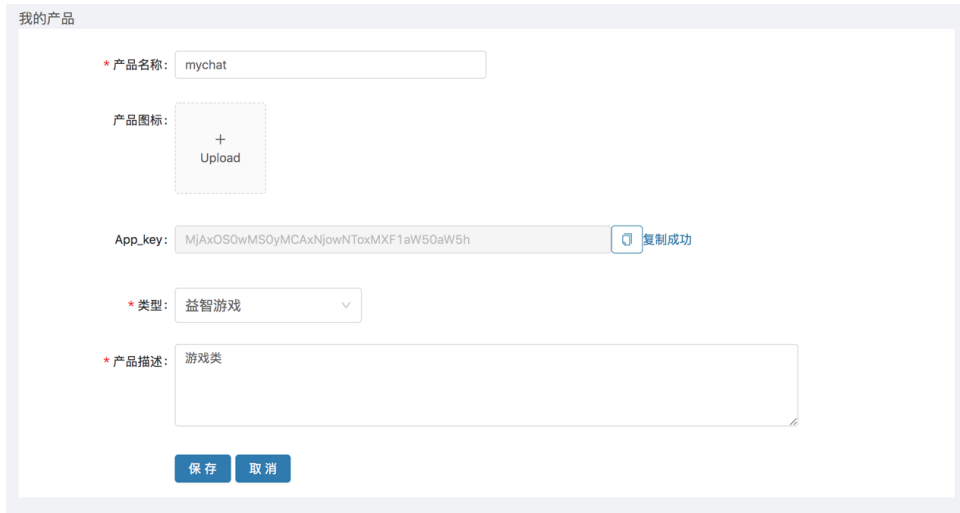


图 4.16: 产品修改界面

4.4 系统测试与分析

4.4.1 测试环境

本系统中的Web服务和后台服务部署于同一台阿里云服务器，基本配置为：Intel(R) Xeon(R) CPU E5-2630 2.40GHz，32G RAM。操作系统为Linux，系统版本为Red Hat Enterprise Linux Server release 7.3 (Maipo)。Web服务器采用Nginx作为前端运行容器，并使用Webpack打包部署。数据库服务器使用MySQL。

4.4.2 功能测试

本系统的功能测试主要从崩溃收集与处理、崩溃可视化两个方面展开。其中崩溃收集与处理是一个连续的过程，包括崩溃捕获与上报、崩溃处理两个过程。崩溃可视化则主要涉及用户界面交互。由于开发时间的限制，本文中崩溃收集与处理、崩溃可视化两部分的测试主要通过手工测试完成，以下将具体描述相关测试用例。

(1) 崩溃收集与处理

表 4.7是崩溃收集与处理流程的测试用例，该测试用例主要用于测试一个崩溃的收集、上报和处理的过程，主要包括崩溃捕获与上报、崩溃重传、崩溃处理等测试项。实际测试结果符合预期，测试用例全部通过。

表 4.7: 崩溃收集与处理测试用例

测试项	操作/输入	预期结果	测试结果
崩溃捕获与上报	手动在安卓应用中触发一个崩溃	安卓应用提示“当前应用开小差了”，崩溃上传成功，OSS服务器中新增一个文件夹且存放着崩溃截图	通过
崩溃重传	断开网络，并触发一个崩溃，二次启动应用	崩溃正常上传至服务器，且Crash文件夹被清空	通过
崩溃处理	利用postman发送崩溃解析处理请求	正确分类和去重，并将处理后的数据存储至数据库	通过

(2) 崩溃可视化

表 4.8是实时监控分析的测试用例，该测试用例主要用于应用开发者查看应用的整体运行状况，主要包括实时监控查看和崩溃详情查看等测试项。实际测试结果符合预期，测试用例全部通过。

表 4.8: 实时监控分析测试用例

测试项	操作/输入	预期结果	测试结果
实时监控查看	点击实时分析菜单栏	展示实时监控分析结果，包括崩溃发生次数、影响用户数、应用用户占比、上报趋势和今日问题统计	通过
崩溃详情查看	点击Top5表格中的每行	进入对应崩溃详情页面	通过

表 4.9是崩溃列表的测试用例，该测试用例主要用于应用开发者查看应用中发生的所有崩溃，主要包括状态查询、类别查询和崩溃搜索等测试项。实际测试结果符合预期，测试用例全部通过。

表 4.9: 崩溃列表测试用例

测试项	操作/输入	预期结果	测试结果
崩溃列表	选择相应产品查看	展示产品的崩溃列表，表中信息包括异常名、所示应用版本、异常最近上报时间、异常发生次数及影响用户数。	通过
状态查询	选择查看不同状态的崩溃	展示相应状态的崩溃	通过
类别查询	选择查看不同类别的崩溃	展示相应类别的崩溃	通过
崩溃搜索	输入异常名并点击搜索按钮	展示异常名匹配的崩溃	通过
崩溃详情查看	点击异常名	跳转至崩溃详情界面	通过

表 4.10是崩溃详情的测试用例，该测试用例主要从崩溃详情展示的页面及用户相关操作进行，主要包括第三方解决方案查看、更改崩溃状态、点赞功能、页面分布查看、应用分布查看、不同机型设备分布查看、上报趋势查看、上报趋势不同时间段查询以及不同实例查看等测试项。实际测试结果符合预期，测试用例全部通过。

表 4.10: 崩溃详情测试用例

测试项	操作/输入	预期结果	测试结果
第三方解决方案查看	点击查看更多解决方案	显示该崩溃的所有解决方案	通过
更改崩溃状态	1.点击更改状态按钮 2.选择状态，填写解决方案 3.点击保存按钮。	显示保存成功，且崩溃报告中的异常名以横线划去，表明该异常已被处理。	通过
点赞功能	对有用的解决方案点赞	点赞成功，点赞数加1，且点赞按钮不可再被点击	通过
页面分布查看	点击页面分布切换页	显示该崩溃的具体报错页面、代码行数及崩溃截图	通过
应用分布查看	点击应用版本切换页	展示该崩溃在不同应用版本的分析，包括当前应用版本等信息	通过
机型设备分布查看	点击机型设备切换页	以饼图形式展示该崩溃在不同机型、不同系统版本上的分布	通过
上报趋势查看	点击上报趋势切换页	以折线图形式展示该崩溃的上报趋势，图中的每个点都包括该崩溃的发生次数和影响用户数	通过
上报趋势不同时间段查询	选择不同时间	页面重新加载该时间段内的崩溃统计	通过
不同实例查看	点击每行	页面加载对应的崩溃实例，可查看到详细的设备信息、堆栈信息和页面路径跟踪信息	通过

表 4.11是产品管理的测试用例，该测试用例主要用于应用管理者管理产品，主要包括应用创建、应用修改、应用删除及appKey复制等测试项。实际测试结果符合预期，测试用例全部通过。

表 4.11: 产品管理测试用例

测试项	操作/输入	预期结果	测试结果
应用创建	1.点击创建产品按钮 2.输入应用名称、应用类型、应用描述等信息 3.点击创建按钮	页面提示产品创建成功并跳转至产品列表界面	通过
应用修改	1.点击设置按钮 2.修改产品信息 3.点击保存	页面提示产品修改成功并跳转至产品列表界面	通过
应用删除	1.点击删除按钮 2.点击确认删除按钮 3.点击保存	页面提示是否确认删除，页面提示删除成功并转至产品列表界面	通过
appKey复制	1.点击设置按钮 2.点击复制按钮	页面提示appKey复制成功	通过

4.4.3 性能测试

本文选用性能压测工具Apache JMeter进行性能测试，该工具可以同时模拟多个并发请求，用于测试网站性能，使用简单且方便。

本文对崩溃信息线上分析系统的可视化模块主要接口进行性能压测，按照并发用户的值分别取100、200、300、400、500进行连续测试。图4.17为当前并发数等于100时的反馈测试图。

Label	# 样本	平均值	最小值	最大值	异常%	吞吐量
不同页面分布	100	505	91	2507	0.00%	33.7/sec
更改崩溃状态	100	293	34	1843	0.00%	35.5/sec
获取机型设备分布	100	306	57	2037	0.00%	33.3/sec
获取当前崩溃	100	411	79	1638	0.00%	33.4/sec
获取上报趋势	100	243	33	2379	0.00%	34.5/sec
获取第三方解决方案	100	254	27	1545	0.00%	37.4/sec
获取当前产品	100	169	8	1067	0.00%	44.4/sec
TOTAL	700	312	8	2507	0.00%	205.4/sec

图 4.17: JMeter压力测试汇总报告

根据测试报告可以看出，当测试数据请求为100条时，不同页面分布接口的平均响应时间为505ms；更改崩溃状态接口的平均响应时间为293ms；获取机型设备分布接口的平均响应时间为306ms；获取当前崩溃详情接口的平

均响应时间为411ms；获取上报趋势接口的平均响应时间为243ms；获取第三方解决建议接口的平均响应时间为254ms；获取当前产品信息接口的平均响应时间为169ms；所有测试请求的平均响应时间为312ms；接口总吞吐量为每秒205.4条，且各接口的异常率为0.00%。

表 4.12: 性能压测结果表

并发数	100	200	300	400	500
异常率	0	0	0	0	0
平均响应时间(毫秒)	312	613	880	1216	1750
吞吐量(请求数/秒)	205.4	231.1	241.1	235.7	232.3
最大响应时间(毫秒)	2507	4765	8096	9945	11284
最小响应时间(毫秒)	8	3	3	3	1

通过不断更改接口并发数量，经过一组连续的测试，整理分析得出如表 4.12所示的性能压测结果表。由表可知，当并发数从100增长到500时，各接口异常率为0%，用户平均响应时间随之增加，系统平均每秒吞吐量约为229.3条。

综上所述，本系统能够满足真实场景下的高并发请求，系统在高峰值请求场景下，依然能够正常运行，维持正常的业务流程，保障了数据一致性和请求有序性。

4.4.4 实验分析

本文实现的崩溃信息线上分析系统主要面向安卓应用真实使用场景。因此，为确保本系统在实际场景下的可用性和可靠性，本节通过实验评估本系统的可靠性和有效性。本实验选取了10个属于不同类别的开源应用接入系统以全面评估系统功能，同时考虑到相当一部分崩溃产生原因与特定设备软硬件信息相关，本实验使用了具有20台不同安卓设备的测试设备集群，将应用同时运行于集群各台设备上，以模拟用户真实使用应用时各个用户不同的设备特性。此外，本实验基于GUI自动化测试工具来自动生成模拟的用户事件序列对应用进行操作，以触发应用潜在错误。

本实验选取了10个开源应用，并在各应用中嵌入了崩溃收集模块SDK。使用自动化测试工具在20台设备上安装并运行应用，为其生成模拟用户操作的事件序列。崩溃收集模块监控并捕获各应用崩溃，并上报至远程服务器。服务器端崩溃解析处理模块对服务器数据库中崩溃数据进行统计和分析，最后由崩溃可视化模块绘制并多维度展示崩溃数据。以下将分别从不同应用和不同设备的角度对本次实验的结果进行分析。

由图4.18可知，本实验10个应用分别为2048、A Photo Manager、BirthDay-Droid、Calculator、Etar、music、Quick、RGB Tool、Solar Compass、redMoon，这些应用的源代码可在开源安卓应用发布平台F-droid¹中下载。其中，在20台安卓设备上，崩溃发生次数最多的应用为Calculator，共22次，其次是Solar Compass，共17次，崩溃发生次数最少的是2048和BirthDayDroid，两个应用都仅发生了3次错误。

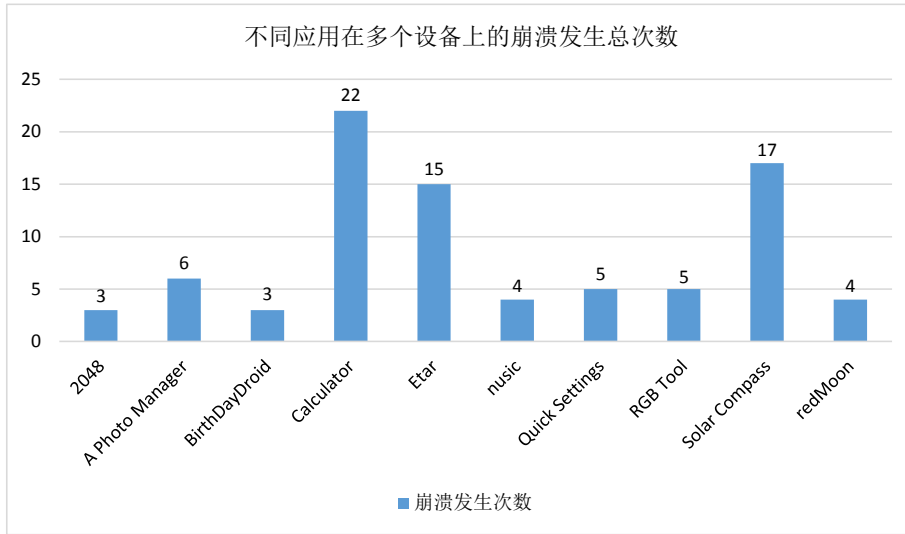


图 4.18: 不同应用在不同设备上的崩溃发生总次数

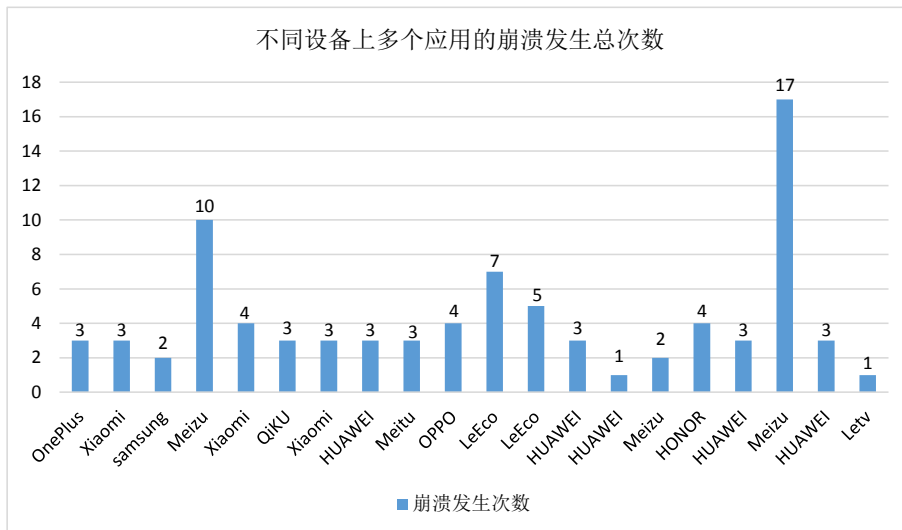


图 4.19: 不同设备上多个应用的崩溃发生总次数

¹<https://f-droid.org/>

由图4.19可知,本实验使用的20台安卓设备品牌主要包括Xiaomi、Samsung、Meizu、Qiku、HUAWEI、Meitu、OPPO、LeEco、HONOR、Letv等。其中, HUAWEI共5台不同设备, Meizu共4台不同设备, Xiaomi共3台不同设备, 其它各一台设备。从图中可以得知,各个安卓设备在运行这10个应用时的崩溃发生总次数。其中, Meizu的崩溃次数发生最多, Letv的崩溃次数发生最少, HUAWEI的平均崩溃发生次数为2.6, Xiaomi的平均崩溃发生次数为3.3, Meizu的平均崩溃发生次数为8。

经统计,本实验中系统一共捕获到84个崩溃,共10个类型的异常,如表4.13所示。其中, java.io.FileNotFoundException异常的崩溃发生次数最多,共21次,其次是android.system.ErrnoException,共17次,崩溃发生次数最低的为java.lang.Exception,仅1次。

本实验收集到的十种异常类别中,崩溃分类正确率为88.1%,崩溃去重率为60.7%。其中系统未准确划分android.os.DeadObjectException和java.net.MalformedURLException的类别。经查证,预测类别出现误差的原因是训练集中的数据不包括这两种类型的异常。因此当预测错误的崩溃数量达到一个阈值时,应考虑利用这批未分类数据再次训练分类模型,不断完善本系统分类法。除此之外, java.lang.RuntimeException: Performing stop of activity that is not resumed被分类为java.lang.RuntimeException类别。由于训练集中未出现这样类别的异常,导致系统默认将其归类为java.lang.RuntimeException大类。可见,对于部分高层父类的具体子类,若其不存在于本系统训练集中,则将会被直接归类到其高层父类。

表 4.13: 崩溃收集类型表

真实类型	预测类别	次数
android.os.DeadObjectException	android.os.RemoteException	2
android.system.ErrnoException	android.system.ErrnoException	17
NumberFormatException	NumberFormatException	15
java.lang.RuntimeException: Failure delivering result ResultInfo	java.lang.RuntimeException: Failure delivering result ResultInfo	3
java.lang.RuntimeException: Performing stop of activity that is not resumed	java.lang.RuntimeException	6
java.net.MalformedURLException	java.net.SocketTimeoutException	2
java.lang.Exception	java.lang.Exception	1
java.lang.SecurityException	java.lang.SecurityException	13
java.io.FileNotFoundException	java.io.FileNotFoundException	21
java.io.IOException	java.io.IOException	4

综上所述，本系统在真实场景下的分类和去重效果良好，其正确率为88.1%，去重率为60.7%。同时，在此次实验中，系统能够处理一定规模的并发崩溃数据上传请求，并提供详细丰富地崩溃可视化报告，系统各功能模块表现正常。本次实验结果表明，本系统具有较好的性能，且各功能模块具有良好的可用性和可靠性。

4.5 本章小结

本章对崩溃收集模块、崩溃解析处理模块、崩溃可视化模块的具体实现进行了详细的阐述，包括SDK实现、崩溃去重和分类、接口设计和前端展示等，重点对崩溃去重和分类进行了详细说明和解释。在对各个模块介绍过程中，给出了部分接口实现伪代码以及系统界面截图。最后，对系统的功能和性能方面进行了测试，同时设计了相关实验来模拟真实用户使用场景，验证了系统的可用性和可靠性，

第五章 总结与展望

5.1 总结

本文基于安卓开发团队对于安卓应用崩溃分析的真实需求，设计并实现了集崩溃收集、崩溃解析处理、崩溃可视化于一体的崩溃信息线上分析系统。本系统帮助安卓开发团队自动捕获和收集移动用户使用过程中应用发生的崩溃相关信息，实时监控安卓应用运行状态。本系统从多个不同维度统计和分析崩溃信息并展示详细的崩溃报告，帮助开发者快速定位和修复应用程序错误，全方位提升安卓应用质量。本文的主要工作如下：

(1) 实现了安卓应用实时崩溃捕获和安卓Activity生命周期监听。收集了崩溃发生设备的硬件信息和性能状态信息，实现了用户页面路径跟踪和崩溃截图功能。同时，为了防止崩溃数据的丢失，实现了实时上传和崩溃重传功能，保证了崩溃数据的及时性和完整性。此外，本系统还为开发者提供了系统集成文档及SDK下载服务。

(2) 实现了崩溃去重和分类。本文提出了基于堆栈信息模式匹配和不一致性分析的崩溃去重方法，经评估，该方法有效减少了重复的崩溃数据。同时，在特征提取和类别标注的基础上，使用了朴素贝叶斯和支持向量机算法分类崩溃。通过对比实验发现，支持向量机在崩溃分类问题表现更优，其准确率达到93%。此外，针对已收集的崩溃类型匹配了对应解决方案，减少开发者定位和修复问题的开销。

(3) 崩溃报告展示和实时监控分析。崩溃报告展示为用户提供了崩溃过滤、崩溃查询及查看崩溃详情等服务。其中，崩溃详情从不同页面视图、机型设备视图及安卓版本视图等多个维度展示崩溃分布，通过崩溃实例展示详细堆栈信息、设备信息和用户页面路径跟踪信息，帮助开发者针对性修复异常。同时为开发者开放解决方案提交入口，以及时收集安卓团队的反馈并完善崩溃解决方案。实时监控分析为用户提供了近24小时内崩溃数据分析服务，以实时掌握应用运行状态。

(4) 系统测试与分析。在部署安卓应用崩溃线上分析系统的基础上，对系统进行了功能测试和性能分析。同时，通过相关实验分析和评估了真实使用场景下系统的运行状况。系统整体测试结果表明，本系统具有良好的可用性、可靠性，能够应对一定规模的并发请求。

5.2 展望

本文实现了一个面向安卓崩溃的线上分析系统并已部署到线上服务器，结合相关技术的发展现状和安卓开发团队对于崩溃分析处理的真实需求，本系统仍存在可以进一步改进和优化之处，未来将针对以下几个方面持续完善系统：

(1) 本文基于安卓平台实现了崩溃信息线上分析系统，因开发时间的限制，本系统暂不支持iOS平台。但本系统提供了统一的数据上报及前后端交互接口，未来只需针对iOS平台的特性实现对应的SDK中的数据收集的功能，就可实现iOS平台的接入。

(2) 本文使用自动化测试工具运行安卓应用，收集了54类目前比较典型的错误异常，并针对这些错误异常收集对应的解决方案。由于安卓第三方库众多，导致本系统难以覆盖所有异常类型。针对这部分异常，系统未对其提供错误解决方案。但是系统提供了用户提交解决方案的接口，能够在一定程度上弥补解决方案的不完善性。未来也将进一步不断补充分类及对应的解决方案，以保证分类的准确性和唯一性。

(3) 本文实现了对用户行为路径的跟踪，因开发时间的限制，暂未对用户行为跟踪进下一步的分析。但是，本系统用户页面路径跟踪中收集了用户进入每个页面的时间、页面名称等上下文信息，为未来用户行为分析提供了数据基础。未来将针对用户行为的点击流数据进行分析并提供用户聚类等功能，以便安卓应用团队为移动用户提供个性化服务。

参考文献

- [1] D. Gallego, W. Woerndl, G. Huecas, Evaluating the Impact of Proactivity in the User Experience of a Context-Aware Restaurant Recommender for Android Smartphones, *Journal of Systems Architecture* 59 (9) (2013) 748–758.
- [2] B. Kitchenham, S. L. Pfleeger, Software Quality: The Elusive Target [Special Issues Section], *IEEE Software* 13 (1) (1996) 12–21.
- [3] J. Park, Y. B. Park, H. K. Ham, Fragmentation Problem in Android, in: *Proceedings of the International Conference on Information Science and Applications*, IEEE, 2013, pp. 1–2.
- [4] L. Ravindranath, S. Nath, J. Padhye, H. Balakrishnan, Automatic and Scalable Fault Detection for Mobile Applications, in: *Proceedings of the Annual International Conference on Mobile Systems, Applications, and Services*, ACM, 2014, pp. 190–203.
- [5] H. Zhang, The Optimality of Naive Bayes, in: *Proceedings of the International Florida Artificial Intelligence Research Society Conference*, AAAI, 2004, pp. 562–567.
- [6] J. A. Suykens, J. Vandewalle, Least Squares Support Vector Machine Classifiers, *Neural Processing Letters* 9 (3) (1999) 293–300.
- [7] S. Kim, T. Zimmermann, N. Nagappan, Crash Graphs: An Aggregated View of Multiple Crashes to Improve Crash Triage, in: *Proceedings of the International Conference on Dependable Systems & Networks*, IEEE, 2011, pp. 486–493.
- [8] M. White, M. Linares-Vásquez, P. Johnson, C. Bernal-Cárdenas, D. Poshyanyk, Generating Reproducible and Replayable Bug Reports from Android Application Crashes, in: *Proceedings of the International Conference on Program Comprehension*, IEEE, 2015, pp. 48–59.
- [9] H. Shahriar, S. North, E. Mawangi, Testing of memory leak in android applications, in: *Proceedings of the International Symposium on High-Assurance Systems Engineering*, IEEE, 2014, pp. 176–183.

- [10] G. Hu, X. Yuan, Y. Tang, J. Yang, Efficiently, Effectively Detecting Mobile App Bugs with Appdoctor, in: Proceedings of the European Conference on Computer Systems, ACM, 2014, pp. 18–18.
- [11] K. Moran, M. Linares-Vásquez, C. Bernal-Cárdenas, C. Vendome, D. Poshy-vanyk, Automatically Discovering, Reporting and Reproducing Android Application Crashes, in: Proceedings of the International Conference on Software Testing, Verification and Validation, IEEE, 2016, pp. 33–44.
- [12] 杨高翔, Android应用程序崩溃信息收集与展示系统的设计与实现, Master’s thesis, 北京大学(2014).
- [13] J. Gao, X. Bai, W.-T. Tsai, T. Uehara, Mobile Application Testing: A Tutorial, *Computer* 47 (2) (2014) 46–55.
- [14] J. M. Miller F. P., Vandome A. F., Application Performance Management, Alphascript Publishing, 2010.
- [15] B. Hardy, B. Phillips, Android Programming: The Big Nerd Ranch Guide, Addison-Wesley Professional, 2013.
- [16] G. Bierman, M. Abadi, M. Torgersen, Understanding Typescript, in: Proceedings of the European Conference on Object-Oriented Programming, Springer, 2014, pp. 257–281.
- [17] M. Grinberg, Flask Web Development: Developing Web Applications with Python, O’Reilly Media, Inc., 2018.
- [18] X. Chi, B. Liu, Q. Niu, Q. Wu, Web Load Balance and Cache Optimization Design Based Nginx under High-Concurrency Environment, in: Proceedings of the International Conference on Digital Manufacturing & Automation, IEEE, 2012, pp. 1029–1032.
- [19] R. T. Fielding, R. N. Taylor, Principled Design of the Modern Web Architecture, *ACM Transactions on Internet Technology* 2 (2) (2002) 115–150.
- [20] B. Cabral, P. Marques, Exception Handling: A Field Study in Java and .Net, in: Proceedings of the European Conference on Object-Oriented Programming, Springer, 2007, pp. 151–175.

- [21] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, J. Irwin, Aspect-Oriented Programming, in: Proceedings of the European Conference on Object-Oriented Programming, Springer, 1997, pp. 220–242.
- [22] D. W. Fellner, Object-Oriented Programming - Does it Help in Computer Graphics?, in: Proceedings [on Occasion of H. Maurer’s 50th Birthday], IEEE, 1991, pp. 132–151.
- [23] T. Elrad, R. E. Filman, A. Bader, Aspect-Oriented Programming: Introduction, Communications of the ACM 44 (10) (2001) 29–32.
- [24] S. R. Safavian, D. Landgrebe, A Survey of Decision Tree Classifier Methodology, IEEE Transactions on Systems, Man, and Cybernetics 21 (1991) 660–674.
- [25] T. K. Ho, Random Decision Forests, in: Proceedings of the International Conference on Document Analysis and Recognition, IEEE, 1995, pp. 278–282.
- [26] J. M. Keller, M. R. Gray, J. A. Givens, A Fuzzy K-Nearest Neighbor Algorithm, IEEE Transactions on Systems, Man, and Cybernetics 15 (1985) 580–585.
- [27] Y. Freund, R. E. Schapire, A Decision-Theoretic Generalization of on-line Learning and an Application to Boosting, Journal of Computer and System Sciences 55 (1) (1997) 119–139.
- [28] S. Xu, Y. Li, Z. Wang, Bayesian Multinomial Naïve Bayes Classifier to Text Classification, in: Proceedings of the Advanced Multimedia and Ubiquitous Engineering, Springer, 2017, pp. 347–352.
- [29] D. Crockford, The Application/Json Media Type for Javascript Object Notation (Json), Tech. rep. (2006).
- [30] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau, Extensible Markup Language (XML) 1.0, W3C Recommendation October, 2000.
- [31] R. Copeland, Essential SQLAlchemy, O’Reilly Media, Inc., 2008.
- [32] D. Barry, T. Stanienda, Solving the Java Object Storage Problem, Computer 31 (1998) 33–40.
- [33] C. Gackenheim, Introduction to React, Apress, 2015.

- [34] A. Banks, E. Porcello, Learning React: Functional Web Development with React and Redux, O'Reilly Media, Inc., 2017.
- [35] J. Kuan, Learning Highcharts, Packt Publishing Ltd, 2012.
- [36] L. Richardson, S. Ruby, RESTful Web Services, O'Reilly Media, Inc., 2008.
- [37] 包建强, App研发录:架构设计、崩溃分析和竞品技术分析, 机械工业出版社, 2015.

简历与科研成果

基本情况 李秋婷，女，汉族，1994年10月出生，四川省泸州市人。

教育背景

2017.9~2019.6 南京大学软件学院 硕士

2013.9~2017.6 南京理工大学计算机科学与工程学院 本科

参与项目

1. 南京南瑞集团项目：移动众测平台软件（20170413），2017-2018
2. 国家重点研发计划课题：基于协同编程现场的智能实时质量提升方法与技术（2018YFB1003901），2018-2021

致 谢

转眼间，美好的两年研究生生活即将画上句号，这两年时光里，我努力学习专业知识，参加相关项目实践工作，不断地从导师、同学、挚友身上学习到优秀的品格和乐观的生活态度。在毕业论文期间，有太多的人给予了帮助，在此向所有关心我的人致以诚挚的感谢。

首先感谢自己的指导老师陈振宇老师，我的毕业成果离不开老师的辛勤指导。陈老师学识渊博、经验丰富，在论文选题期间就提供了许多宝贵的意见。在论文写作过程中更是悉心指导，多次对论文提出修改建议。再次向陈老师表示衷心的感谢。

其次，还要感谢我的室友、师兄师姐们。这两年与室友携手同行跨过了许多艰辛的时光，从师兄师姐们身上学习到了项目开发技术和经验，师兄师姐们在我找工作期间，更是提供了许多宝贵建议。

最后，我还要感谢我的父母，他们总是在精神上不断支持我，尊重我做的每一个决定，父母永远是最坚强的后盾、最温馨的港湾。

版权及论文原创性说明

任何收存和保管本论文的单位和个人，未经作者本人授权，不得将本论文转借他人并复印、抄录、拍照或以任何方式传播，否则，引起有碍作者著作权益的问题，将可能承担法律责任。

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含其他个人或集体已经发表或撰写的作品成果。本文所引用的重要文献，均已在文中以明确方式标明。本声明的法律结果由本人承担。

作者签名：

日期： 年 月 日