# Android Apps 数据收集工作报告

> 科目：移动互联网安全技术
> 指导老师：刘银龙　时间：2021.06.01
> 姓名：梁飞　学号：2020E8018682045　单位：信工所四室

## 摘要

为自动化批量获取Android应用的用户数据，本文采用基于Appium的应用数据收集方法，搭建好所需环境后，建立了一个用于批量获取数据的初步框架，使得后续只需更改不同的配置文件即可自动收集不同App的信息，最后应用以上方法对Android终端的4个不同类型的应用进行收据收集，实验结果表明，该方法能够有效绕过安全机制，能够获取到包括用户隐私数据在内的多种信息。本文首先第一章介绍前期做所的一些工作，其次第二章介绍本文所使用的的主要方法及实验结果，最后做了简单总结。

## 目录



# 1 前期工作

本章主要内容为介绍前期工作内容及结果，主要包括对课题的理解、对当前Android Apps数据收集方法的调研和前期的一些工作成果介绍。

## 1.1 课题介绍与理解

Android端应用数据收集是一项比较宽泛的课题，原因在于现在主流的应用都朝着“超级App”的方向发展，尤其是微信、QQ、支付宝等头部应用，这些应用已经具备了实质性的垄断地位，因此对于这些应用来说，对用户数据乃至敏感隐私数据的收集工作相对来说更加方式多样、更加容易，因此，对于不同的主体来说，想要达成数据收集的目标其难度是不同的，有关这一话题也做了一部分调研，将在下一节“1.2”中简要阐述。

**课题介绍**

本文所做工作是以第三方的角度，在获取用户登录权限的前提下尝试获取一般Android终端的应用数据，经过一番曲折探索，最终在老师的指导下，确立了课题的以下两个前提假设：

- 已获取用户应用的登录权限；

- 能在完全可控的Android终端正常登录用户应用；

任务目标便是在以上两个前提的基础上获取到Android应用的用户数据，例如微信的联系人数据、QQ联系人及聊天记录、支付宝账单等，最终结果可在“2.4”中详细看到。

**课题意义**

诚然，在已获取用户登录权限的情况下获取用户数据已经减轻了大量工作难度，似乎使得本工作的意义变得十分有限。但两者本身就可以看做两个独立的复杂问题，前者本身是个涉及多个维度的安全问题，后者的主要意义在于批量、自动化获取应用数据。

## 1.2 Android安全机制

> 即使是Android，也远比自己想象中安全的多。

当前主流的智能手机操作系统主要有Android与iOS，但即使是Android，也远比自己想象中安全的多。Android至今已有11个大版本，最新的Android 12也开放测试版更新，无论是Android还是iOS，其安全机制愈加完善。本节介绍Android数据管理、安全机制和前期的一些工作成果[1]。

**数据管理机制**

| 名称 | 数据存储方式 | 存储位置 |
| --- | --- | --- |
| 文件存储 | 文字、图片等资源文件 | ./data/data/ |
| SharedPreferences | 键值对 | ./data/data/ |
| SQLite数据库存储 | 轻量数据库 | ./data/data/ |
| ContentProvider | 应用可公开数据对外接口 | ./data/data/ |
| 网络服务器存储 | 各种类型 | WebServer |

上述表格的前4种方式是存储于用户终端之上，但由于安全机制的存在，"./data/data/"目录未经Root访问不到，而存储于外部开放文件夹内的文件通常是无关痛痒的公共资源文件、下载类型的文件等[2]。

**Android安全机制**[3]

| 名称 | 介绍 |
| --- | --- |
| 进程沙箱隔离机制 | 应用在安装时赋予唯一用户标识（UID），应用及其虚拟机运行在独立的进程空间，与其它应用完全隔离 |
| 应用程序签名机制 | 应用必须拥有开发者的数字签名才能安装 |
| 权限声明机制 | 对应用的行为进行权限管理，敏感操作需要高级别授权才可以执行，某些特殊权限（如Signature和Signatureorsystem级别）只有系统能够使用 |
| 访问控制机制 | 确保系统文件和用户数据不受非法访问 |
| 进程通信机制 | 通过接口描述语言（AIDL）定义接口与交换数据的类型，确保进程间通信的数据不会溢出越界 |
| 内存管理机制 | 将进程重要性分级、分组，当内存不足时，自动清理级别进程所占用的内存空间 |

**其他安全机制**

以上表格所列举的安全机制是Android系统内原生的安全措施，实际上，还有许多安全机制阻止获取用户数据，以下列举两个与本工作关联密切的机制：

1. 数据存储加密。在上述Android原生安全机制的基础上采用加密的方式保护应用程序敏感数据，如利用SQLCipher加密SQLite数据库等，这是主流较为完善的App都会采用的方式（例如微信、QQ、支付宝等），该过程会采用多种多样的加密方式，而且还有可能包括厂商私有的加密方法，总之较为难以破解，而且主动权在于厂商，在破解后也很容易地更换加密方式；
2. Root权限越来越难以获取。有的系统甚至不开放用户Root（BL锁），Root权限对于从底层获取App数据来说是十分必要的，Root权限可以解除上述Android安全机制影响。

## 1.3 现有方法调研

**ContentProvider区域内的共享数据**

应用之间共享数据的一般方法（公共接口），例如：在一个应用内拉起另一个应用时，通过此方法可直接通过ContentProvider接口共享的数据跳转到拉起应用的目标界面，因此该方法属于比较友好的方法，其缺陷也十分明显——只有原应用主动共享的数据才可以获得[1]。

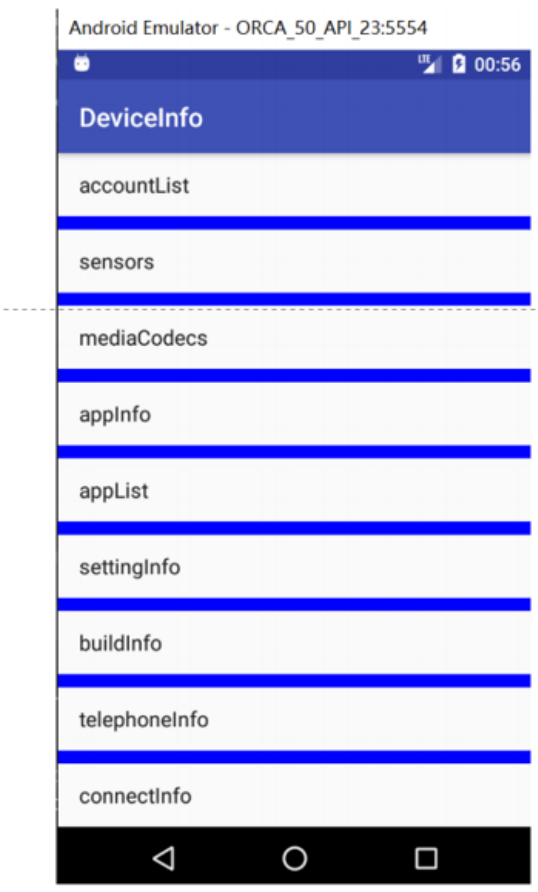
**App自身用户行为数据与共享数据联盟**

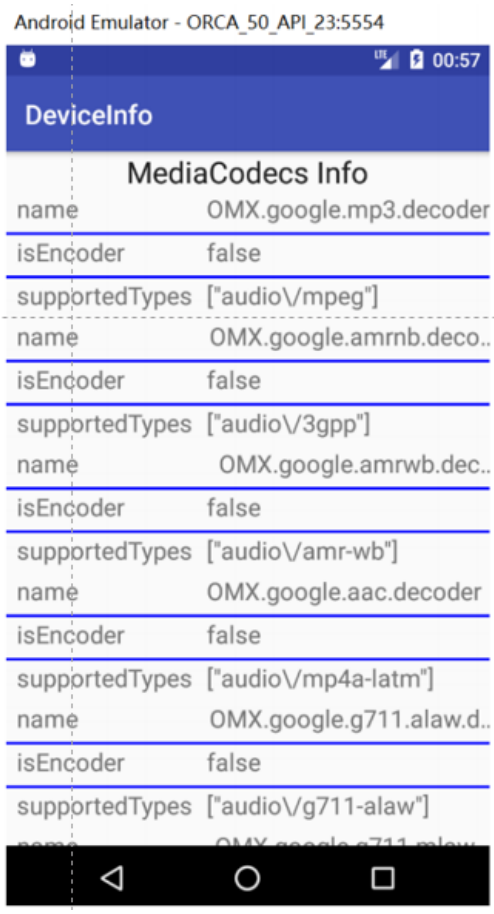
用户使用App时该App本身记录的用户数据，例如诸多推荐算法等依托的正是此类数据。因此，该方法属于App自身数据管理应用范畴，不符合本任务获取用户数据的任务。

一些厂商以公开自己平台用户数据为代价（无论是否经得用户同意）来换取其他平台的用户数据，渐渐形成了“共享数据联盟”，例如友盟，TalkingData，神策，诸葛IO，Growingl等。

**前期方案一：设计一个私有App，通过安装到用户终端获取其他App的用户数据**

该方法诣在通过安装“非法应用”来获取用户终端上的信息，但就调研结果而言，由于Android的进程沙箱隔离机制及其他安全措施，目前没有一种应用能够获取到终端上的其他应用的数据。实验结果为设计了一个Android App，但是只能收集终端上的公开信息，不能获取到其他应用的用户数据。



**前期方案二：基于Python的信息爬取**

以Python为工具，在未Root的情况下，爬取公开目录的所有疑似包含用户数据的文件夹，实验结果：大部分为资源文件，其中以缓存图片居多，例如资源加载等待gif，icon等。

| | key | value |
|---|---|---|
| 1 | gc_http://q.qlogo.cn/qqapp/1104466820/038C53B750 | { "time": 1582787381, "value": "cache |
| 2 | gc_http://q.qlogo.cn/qqapp/1104466820/858AC7599A | { "time": 1582787381, "value": "cache |
| 3 | gc_http://q.qlogo.cn/qqapp/1104466820/D377A6AB35 | { "time": 1582787397, "value": "cache |
| 4 | gc_http://q.qlogo.cn/qqapp/1104466820/E6362D61B2 | { "time": 1583061913, "value": "cache |
| 5 | gc_http://q.qlogo.cn/qqapp/1104466820/0F7C75A402 | { "time": 1583061914, "value": "cache |

该方法在未获取Root权限的情况下获取到的资源有限，即使像上图一样获取到SQLite类型文件，也是一些无足轻重的内容，即使Root后获取到的内容也可能是经过加密的用户数据。

# 2 基于Appium的Android应用数据收集方法

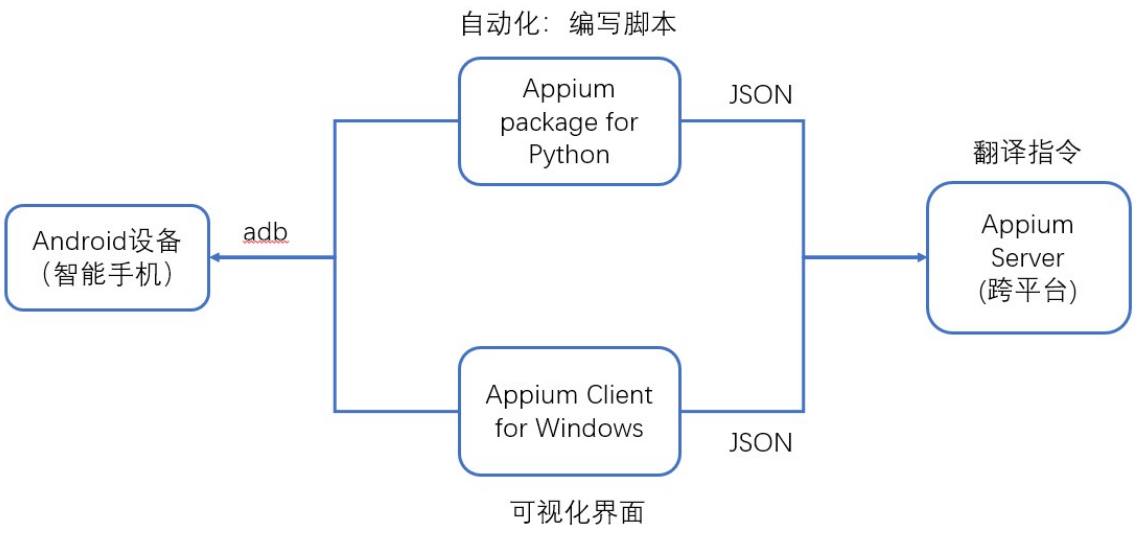
本章介绍一种可行的方法——基于Appium的Android应用数据收集方法，也是本文成功获取到App信息的方法，首先简要介绍Appium，其次简要介绍环境配置，最后概述方法过程[4]。

## 2.1 Appium简介

Appium是一个自动化测试开源工具，支持IOS、Android平台上的原生应用、web应用和混合应用[5]。

- Appium采用C/S架构，Client端是可供用户下载的客户端，Server端是NodeJS的web服务器，Client与Server之间采用HTTP通信，在上层发送JSON格式数据[6]；
- Appium在webDriver（Selenium WebDrier）的基础上丰富了更多接口，形成了自己的一套API；
- Appium的核心即Server端的指令翻译库，其作用是将来自于客户端的不同语言、不同平台的指令翻译成可供目标设备（例如本工作中的Android 10设备）执行的指令，这也是Appium最核心的跨平台特性（iOS、Android、Windows）原因所在；
- Appium是开源的，他属于JS Foundation（Open JS Foundation，是关键开源 JavaScript 项目的主要家园，包括 Appium、Dojo、jQuery、Node.js 和 webpack 等 27 个项目），JS Foundation在层级上又属于The LINUX Foundation。

**工作原理**



## 2.2 环境配置

- Java + Python（Packages）：Appium-Python-Client v1.1.0
- Android SDK（Android 11 R API Level 30）
- Appium（for Windows v1.21）
- Android设备（HMA-AL00 HUAWEI Mate20 Android 10 条件：USB调试+亮屏）
- adb: Windows环境变量[7]

最后，使用以下命令验证连接设备是否成功[8]：

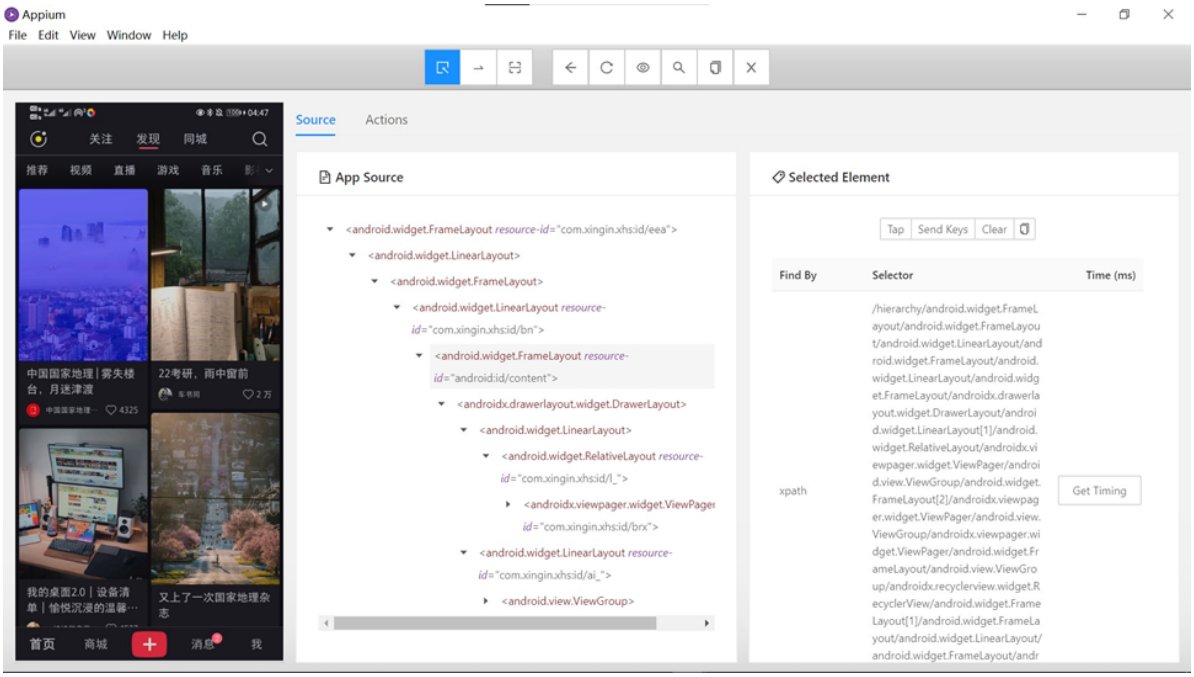
```
adb devices
```

shell 返回值：

```
List of devices attached
HJS5T19829001972        device
```

代表已接入Android设备。

在Appium Client中配置变量验证是否工作正常：




点击左侧元素，右侧会出现Android界面xml解析。

## 2.3 方法概述

### 流程总览

整个流程可分为5个部分，注意，以下步骤均建立在上述环境配置正常的条件下。

1. 编写Python程序，用于自动化测试；
2. 获取App包的启动信息；
3. 在Appium客户端中查看元素xpath属性值，设计点击顺序[9]；
4. 在Python程序中编写配置文件；
5. 运行抓取App数据；

### Python程序

程序共分为4个文件，架构如下：

- main.py：主程序文件，所有执行均在此调用；
- tools.py：工具函数文件，main中所有执行的函数均在此定义；
- caps.py：App链接文件，包含关键的App包信息和启动Activity；
- xpath.py：App的xpath点击顺序，不同App只需定义不同的xpath变量即可，更换不同App时需在此修改；

核心代码如下：

（tools.py：上滑函数）

```
# 上滑 手势从下至上 0.5为一整页
def swipe_up(driver, n=0.3):
    # tuple: 获取屏幕大小
    size = (driver.get_window_size()['width'],
            driver.get_window_size()['height'])
    # 定义滑动参数
    x = int(size[0] * 0.5)  # x坐标
    y1 = int(size[1] * 0.9)  # 起始y坐标
    y2 = int(size[1] * (0.9 - 0.9 * n))    # 终点y坐标
    # 执行动作
    driver.swipe(x, y1, x, y2)
```

（tools.py：返回函数，利用了某些Android系统边缘滑动返回上一级的手势操作）

```
# 返回 实际为左右边缘滑动 只适用于部分边缘滑动返回的手机
def back(driver):
    # tuple: 获取屏幕大小
    size = (driver.get_window_size()['width'],
            driver.get_window_size()['height'])
    # 定义滑动参数
    x = int(size[0] * 0.3)  # x坐标
    y = int(size[1] * 0.5)  # 起始y坐标
    # 执行动作 屏幕边缘滑动返回
    driver.swipe(0, y, x, y)
```

（tools.py：主要函数，通用的App抓取逻辑）

```
# 执行函数
# @param: search_num -Int 每页上搜索数目
#         view_num -Int 翻页次数
#         swipe_up_dis -Float 上滑幅度
def do_xpath_unit(driver, xpath_set, search_num=20, view_num=3,
swipe_up_dis=0.5):

    result = []

    # 若开启遍历
    if xpath_set[1]:
        # 上滑几次
        for _ in range(view_num):
            # 每个页面默认搜索15个元素
            for i in range(search_num):
                print(i)

                xpath_value_mod = xpath_set[0][0][1]
                xpath_target = get_xpath_serial(xpath_value_mod, i)
```

```
                # 验证序列目标是否存在
                if has_element(driver, 'xpath', xpath_target):

                    # 开启循环
                    for xpath_unit in xpath_set[0]:

                        result_dict = {}

                        try:
                            if xpath_unit[0] == 'get_text':
                                text = driver.find_element_by_xpath(
                                    get_xpath_serial(xpath_unit[1], i)
                                ).text

                                result_dict[xpath_unit[2]] = text

                            if xpath_unit[0] == 'click':
                                driver.find_element_by_xpath(
                                    get_xpath_serial(xpath_unit[1], i)
                                ).click()

                            if xpath_unit[0] == 'back':
                                back(driver)
                        except Exception:
                            continue
                        finally:
                            # 防止重复
                            if result_dict not in result:
                                result.append(result_dict)

                # 目标序列不存在
                else:
                    continue
            # 上滑翻页
            swipe_up(driver, swipe_up_dis)

    else:
        result_dict = {}
        # 开启循环
        for xpath_unit in xpath_set[0]:
            try:
                if xpath_unit[0] == 'get_text':
                    text = driver.find_element_by_xpath(xpath_unit[1]).text
                    result_dict[xpath_unit[2]] = text

                if xpath_unit[0] == 'click':
                    driver.find_element_by_xpath(xpath_unit[1]).click()

                if xpath_unit[0] == 'back':
                    back(driver)
            except Exception:
                continue
            finally:
                # 防止重复
                if result_dict not in result:
                    result.append(result_dict)

```

```
    if result:
        return result
    else:
        return

```

（caps.py：微信启动参数）

```
# 微信
    'weixin': {
        'platformName': GLOBAL_PLATFORMNAME,
        'deviceName': GLOBAL_DEVICENAME,
        'platformVersion': GLOBAL_PLATFORMVERSION,
        'noReset': GLOBAL_NORESET,
        'unicodeKeyboard': GLOBAL_UNICODEKEYBOARD,
        'appPackage': 'com.tencent.mm',
        'appActivity': '.ui.LauncherUI',
    },
```

在上述结构组织下，更换抓取目标App只需在xpath.py文件内配置好参数即可运行抓取数据，更多详细代码请参照”4 备注“部分关于源代码的注释。

**获取APP包信息：查看包名称及启动Activity**

获取App的启动信息是第一步[10]，所有测试App的启动参数配置均在caps.py文件中，例如上述微信的启动参数，Appium所需的核心启动参数如下（以小红书App为例）：

```
{
    'platformName': GLOBAL_PLATFORMNAME,          # 平台名称
    'deviceName': GLOBAL_DEVICENAME,              # 设备名称
    'platformVersion': GLOBAL_PLATFORMVERSION,    # 系统版本号
    'noReset': GLOBAL_NORESET,                    # 免登陆TRUE
    'unicodeKeyboard': GLOBAL_UNICODEKEYBOARD,    # 使用Unicode以输入中文
    'appPackage': 'com.tencent.mm',               # apk的包名
    'appActivity': '.ui.LauncherUI',              # App启动关键Activity
}
```

对于测试过程中的同一台设备而言，'platformName'（平台名称）、'deviceName'（设备名称）、'platformVersion'（系统版本号）、'noReset'（免登陆TRUE）、'unicodeKeyboard'（开启Unicode）参数不需要改变，而关键的'appPackage'（apk的包名）、'appActivity'（Activity名）等不容易获取，前者为App包的id，后者为启动主页面的进程id。

网上有大量的关于常见APP的包信息和Activity信息资料，但一般已经失效（App版本更新等原因）或不正确，显然，默认的方法字段不适用于大部分App，本文采用以下方法获取上述两项关键参数。使用以下shell命令查看当前活动App相关信息：

```
adb shell dumpsys window w |findstr \/ |findstr name=
```

shell 返回值如下：

```
mSurface=Surface(name=GestureNavBottom)/@0x3fb13a1
mSurface=Surface(name=GestureNavRight)/@0x3fb1215
mSurface=Surface(name=GestureNavLeft)/@0x3fb1323
mAnimationIsEntrance=true
mSurface=Surface(name=StatusBar)/@0x392b383
mSurface=Surface(name=com.tencent.mm/com.tencent.mm.ui.LauncherUI)/@0x411d3bf
mAnimationIsEntrance=true
mSurface=Surface(name=com.android.systemui.HwImageWallpaper)/@0x3b33711
```

该命令以微信为例，以上返回值第6行：'com.tencent.mm/com.tencent.mm.ui.LauncherUI'即为所需参数信息，'com.tencent.mm'为所需的微信包名称，'.ui.LauncherUI'为微信启动Activity。其他APP可采用同样方法获得，该方法能比较准确的获取上述两项关键参数。

**运行抓取程序**

在main.py中运行主程序抓取数据，主要代码如下（以微信App为例）：

```
# 全局变量 HTTP链接地址
GLOBAL_WEBDRIVERHTTPLOC = 'http://127.0.0.1:4723/wd/hub'


# main:
if __name__ == '__main__':

    # ==== 微信 == start ================================================
    # 打开app
    driver = webdriver.Remote(GLOBAL_WEBDRIVERHTTPLOC, caps['weixin'])

    # 打开app需要时间较长 最短3秒
    time.sleep(3)

    # ----------------------------------------------------
    # 微信 任务一 抓取信息 最近聊天
    result = do_xpath_unit(driver, xpath_weixin[0], 20, 10)
    # 输出抓取结果 聊天信息
    print_result(result, 3)

    # ----------------------------------------------------
    # 微信 任务二 抓取朋友圈信息
    # 点击发现按钮
    do_xpath_unit(driver, xpath_weixin[1])
    time.sleep(0.5)

    # 点击朋友圈按钮
    do_xpath_unit(driver, xpath_weixin[2])
    time.sleep(0.5)

    # 获取朋友圈数据
    result = do_xpath_unit(driver, xpath_weixin[3], 5, 20)
    # 输出抓取结果 朋友圈信息
    print_result(result, 2)

    # ==== 微信 == end ==================================================

```

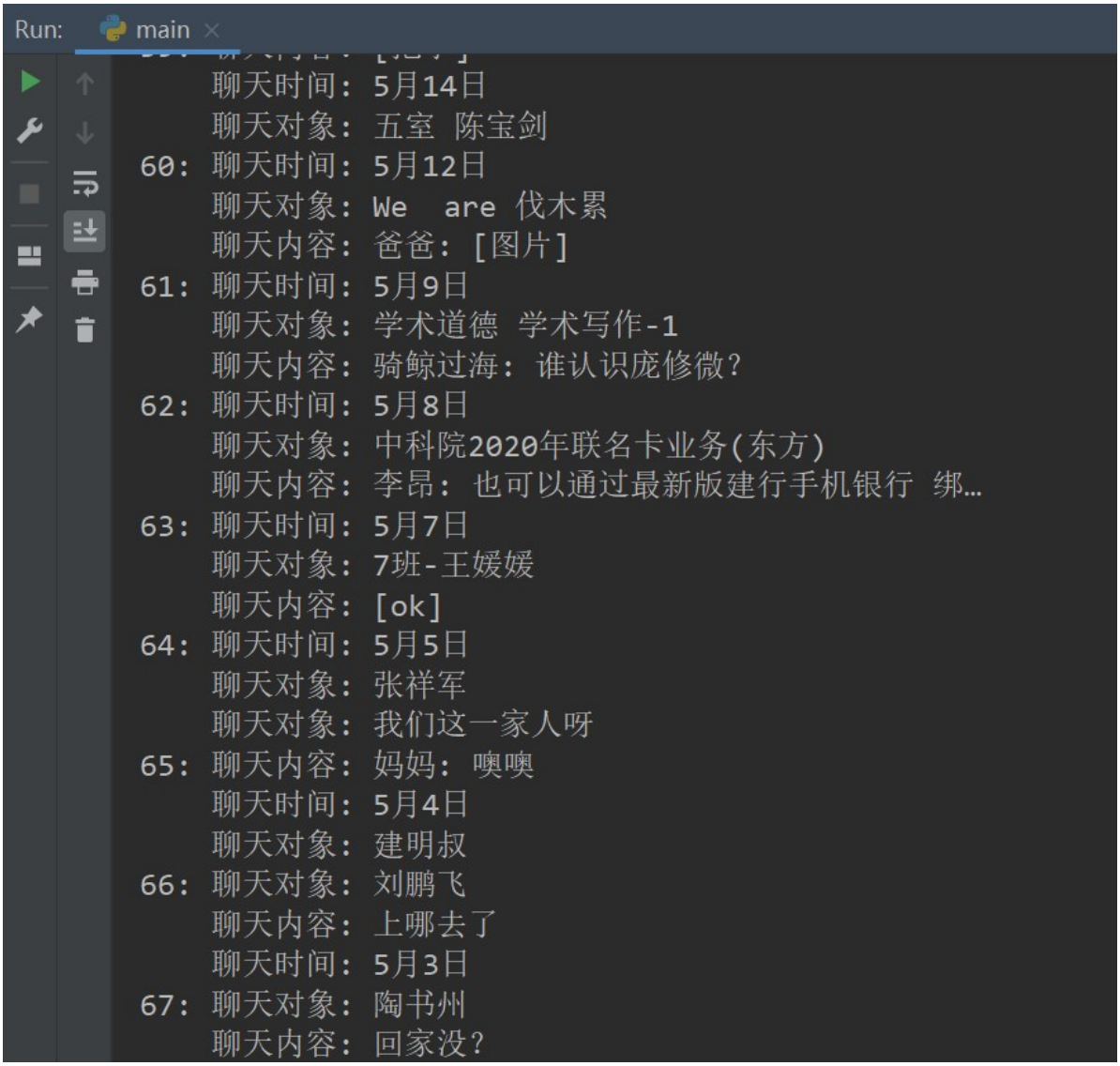
## 2.4 实验结果

根据上述方法步骤，分别对微信、QQ、便签、知乎4个App抓取用户数据，详细如下：

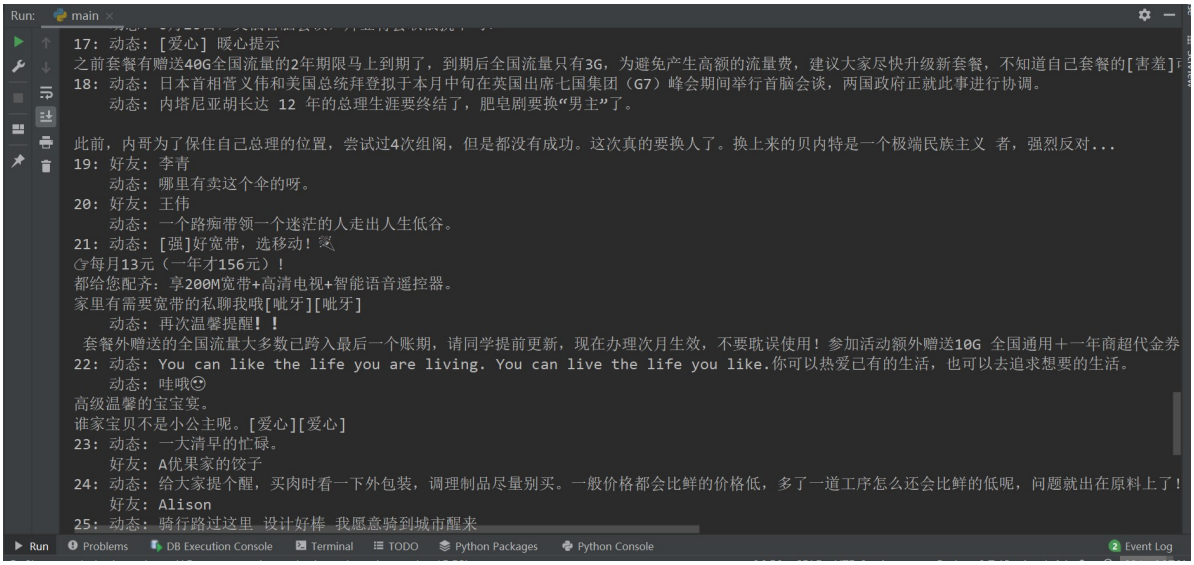
| 序号 | App名 | 类别 | 操作 |
| --- | --- | --- | --- |
| 1 | 微信 | 即时聊天 | 抓取最近聊天信息、抓取朋友圈内容 |
| 2 | QQ | 即时聊天 | 抓取群聊聊天记录 |
| 3 | 便签 | 笔记应用 | 抓取便签笔记内容 |
| 4 | 知乎 | 信息流应用 | 抓取信息流内容 |

**1 微信：抓取最近聊天信息、抓取朋友圈内容**

获取到70条最近聊天信息（设置70，实际还可以更多，部分截图如下，下同）：

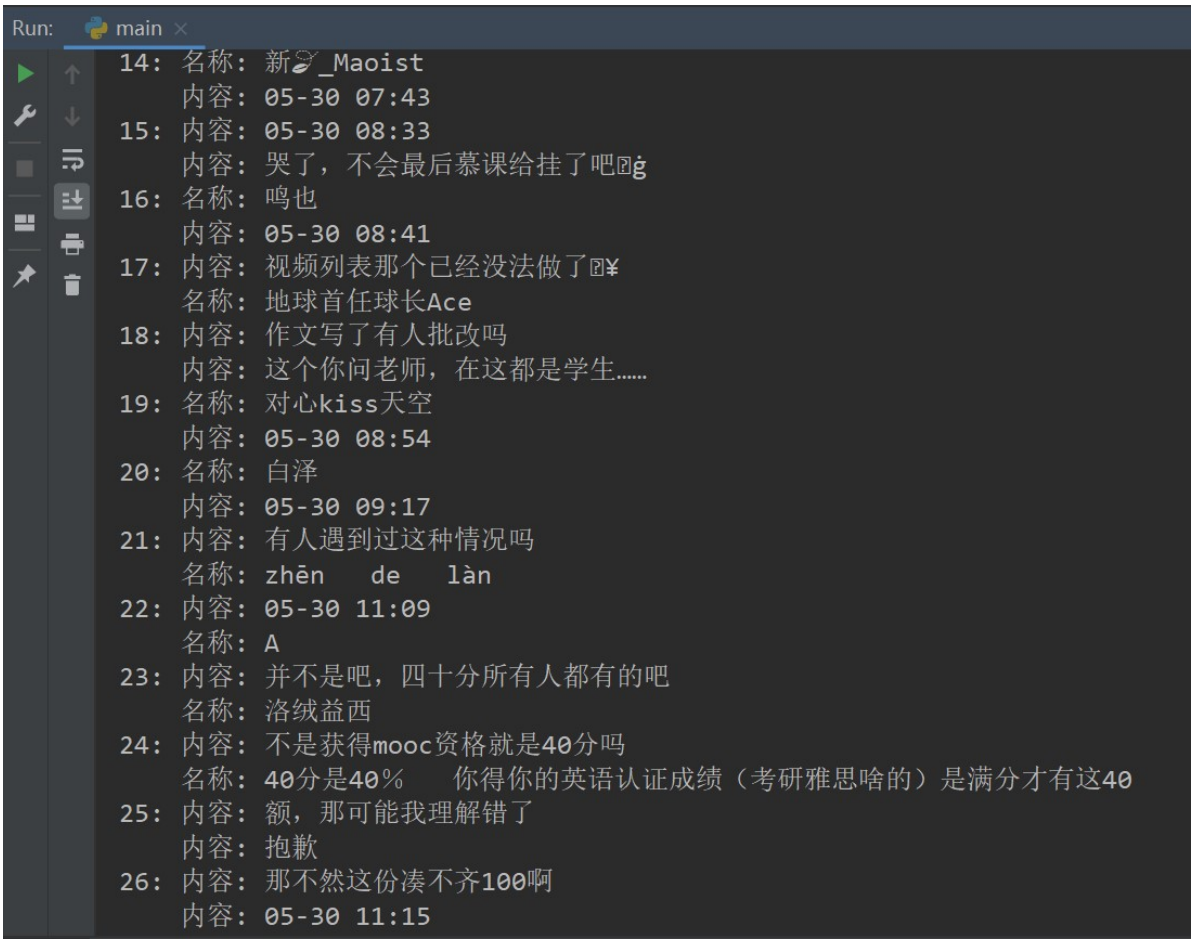



获取到40条朋友圈动态：



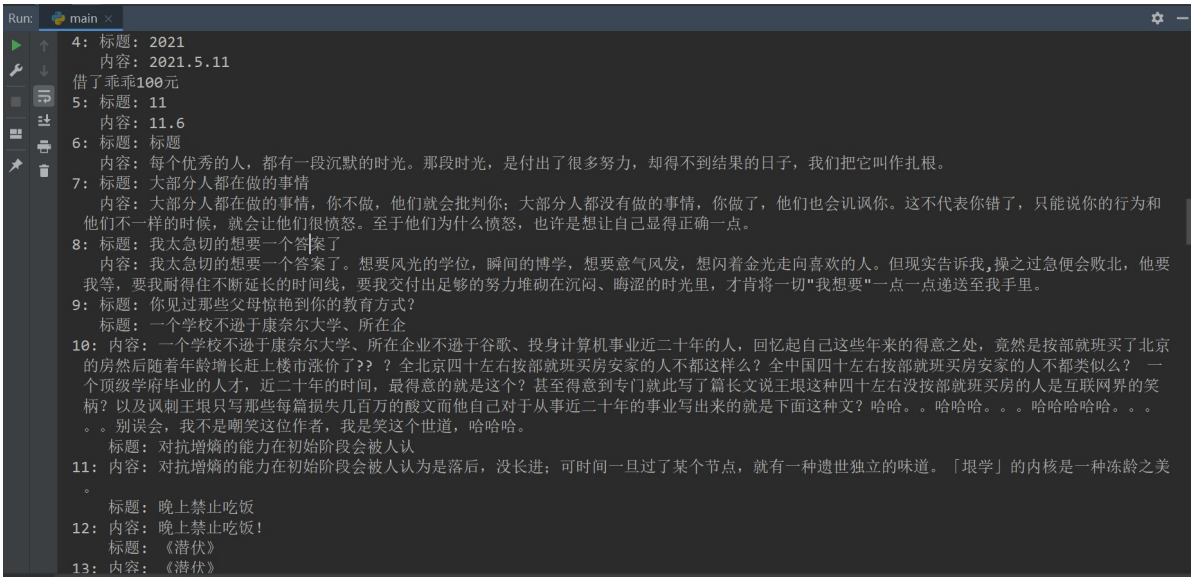

### 2 QQ：获取群聊聊天记录

获取到70条聊天记录，包括发送人、发送内容、时间等信息：



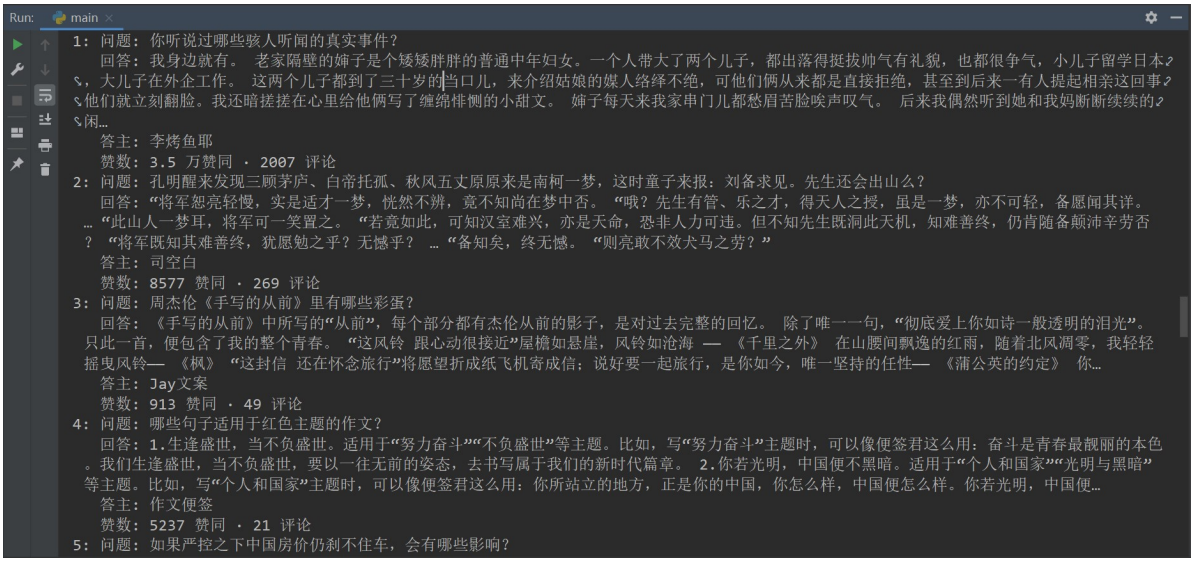

### 3 便签：获取便签笔记内容

获取到100+条便签记录（便签笔记均来自本人记录）：




**4 知乎：获取首页推荐信息流**

获取到100+信息，包括问题名称、答主昵称、回答内容、获赞数目等：




# 3 总结

本文的主要工作在于：

- 利用Appium实现了信息抓取的简单框架，只需少量的配置信息就可以运行通用的抓取函数自动获取APP数据；
- 基于Appium的信息抓取方案本质上是基于界面层的信息获取，因此可以绕过Android底层复杂的安全机制，对于设备本身免Root，减少对设备的不利影响，也可以绕过APP厂商自定义的种种规则限制，比较通用灵活；
- 成功获取到了微信、QQ、便签、知乎等App数据，其中包括用户隐私数据，总体来说，对于信息流等数据的获取比较简单，对于微信等超级App的数据获取比较复杂，原因在于此类超级应用采用的框架更加复杂，Appium平台动作部分受限。

不足之处：

- 有线连接设备，开启USB调试，设备需处于一直亮屏可操作状态；
- 需要提前获取用户应用登录权限；

# 4 备注

1. 本文涉及到的App个人隐私信息均为本人隐私数据，不涉及侵犯他人隐私的情况，也请勿散播；

2. 更多源代码、本文档的Markdown版本等均可在此地址获得：https://github.com/liangfree/get_AndroidApps_info

# 参考文献


1. Jianshu.ContentProvider详解[OL].https://www.jianshu.com/p/5e13d1fec9c9 .2018-7-16. ↩ ↩

2. 张玉清, 王凯, 杨欢,等. Android安全综述[J]. 计算机研究与发展, 2014, 051(007):1385-1396. ↩

3. 符易阳, 周丹平. Android安全机制分析[J]. 信息网络安全, 2011(9):23-25. ↩

4. Alan.Python爬虫App数据抓取-Appium[OL].https://alanhou.org/appium/ .2019-12-6. ↩

5. cnblogs.Appium介绍[OL].https://www.cnblogs.com/xiaonian8/p/13825952.html .2021-6-7. ↩

6. Pandorym.Appium介绍[OL].http://appium.io/docs/cn/about-appium/intro/ .2018-8. ↩

7. cnblogs.使用Appium+python爬取手机App[OL].https://www.cnblogs.com/songzhixue/p/12450354.html .2021-6-7. ↩

8. Jianshu-websky.appium+Python脚本编写[OL].https://www.jianshu.com/p/4adc324fb48b .2018-12-4. ↩

9. CSDN.Python+Appium实现控制app[OL].https://blog.csdn.net/qq_40279964/article/details/88354715 .2019-3-26. ↩

10. cnblogs.Python3+Appium安装使用教程[OL].https://www.cnblogs.com/graybird/p/10793423.html .2019-4-29. ↩