一次存款的背后

• 前言

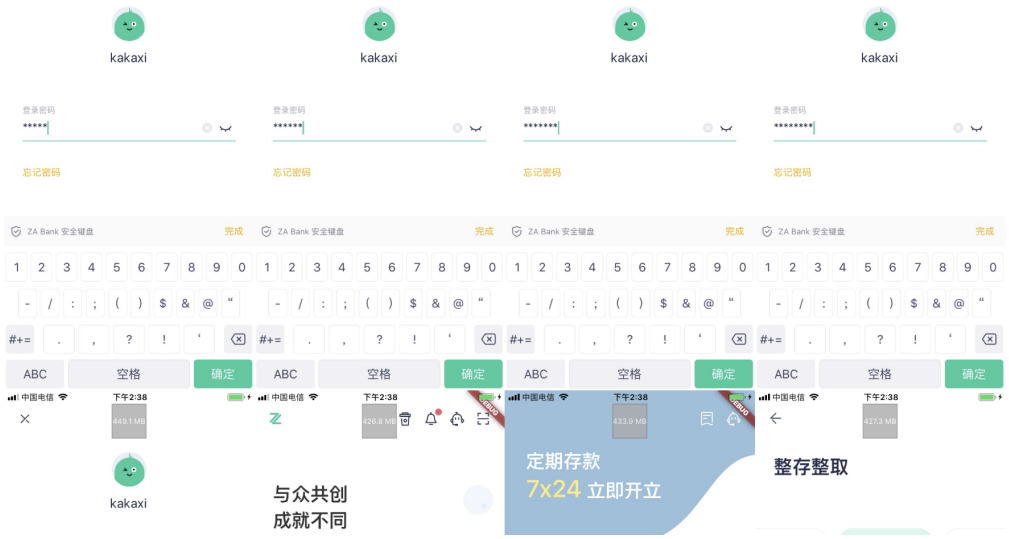
> *UI 自动化测试是模拟用户在 APP或者浏览器内的操作和输入事件（点击、滑动等）来完成功能测试，但从测试角度而言，仅仅完成了功能测试还远远不够。对于用户而言，点击某个按钮后的crash，滑动一次列表的后的卡顿，app 后台偷跑流量等等伤害用户体验的问题如何通过 UI 自动化去定位呢？ App操作过程中会触发哪些接口请求，客户端当时的性能如何，这些都是需要深究的问题。*

本文通过传统的 UI 自动化（appium/wda）+抓包（whistle）+性能测试工具（perfdog），将时间拉长，分析一下在 app 完成一笔200元存款的过程中发生了哪些事情，让我们先分享下结果：

• 结果分享

- **UI 自动化执行结果，可以直观看到测试用例执行情况和结果；**

19	1	✓ 成功
20	0	✓ 成功
21	0	✓ 成功
22	收起键盘	✓ 成功
23	马上存入	✓ 成功
24	继续	✓ 成功
25	确定	✓ 成功
26	存入成功	✓ 成功



- **App 启动一共请求了9次接口，其中https://bankpreidc.zatech.com/ci/punicy/dict/find耗时最久达442ms；**
- **点击登录后共有25次请求接口，其中https://bankpreidc.zatech.com/core/normal/account/info共有5次重复请求，请开发同学 review 是否存在冗余请求；**
- **点击存款确定后，接口https://bankpreidc.zatech.com/ci/nbcvfs/deposit/new 耗时621ms；**
- **仅就登录后接口做分析，共请求接口25次，但如果接口做uri去重分析，涉及到接口仅有13个；**

点击登录：

接口 uri	请求次数
https://bankpreidc.zatech.com/ci/nyctro/acc/get	3
https://bankpreidc.zatech.com/acc/pav2bk/deposit/ccy-list	2
https://bankpreidc.zatech.com/core/normal/account/info	5
https://bankpreidc.zatech.com/acc/ncsate/asset/find	2
https://bankpreidc.zatech.com/acc/nishid/tran/query	2
https://bankpreidc.zatech.com/acc/tutysr/current/find	2
https://bankpreidc.zatech.com/letter/nckfj3/reddot/num	2
https://bankpreidc.zatech.com/letter/pkaogf/announcement/query	2

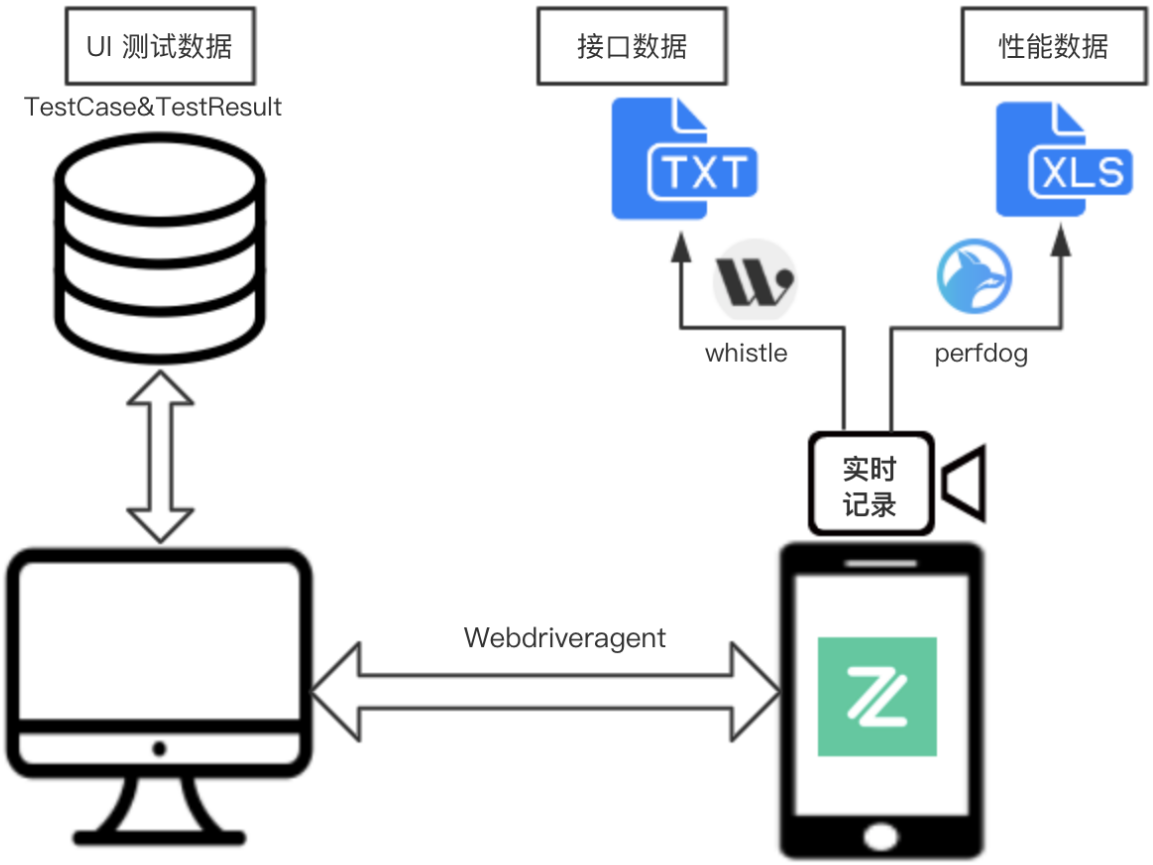
UI事件（接口请求数量）	时间戳(s)	接口	耗时(ms)	接口详情
launch（9）	2019-12-10 14:35:36			查看详情
	2019-12-10 14:35:36	https://bankpreidc.zatech.com/ci/punicy/dict/find	442	查看详情
	2019-12-10 14:35:38	https://bankpreidc.zatech.com/mb/pwd23q/queryDict	32	查看详情
	2019-12-10 14:35:38	https://bankpreidc.zatech.com/loan/pyazye/product/list	104	查看详情
	2019-12-10 14:35:38	https://bankpreidc.zatech.com/appConfig/pi1db3/version/app-version?sign=11F7F4E81…	69	查看详情
	2019-12-10 14:35:38	https://bankpreidc.zatech.com/acc/pav2bk/deposit/ccy-list	74	查看详情
	2019-12-10 14:35:38	https://bankpreidc.zatech.com/letter/pkaogf/announcement/query?sign=ED2F37A4BAC9…	27	查看详情
	2019-12-10 14:35:38	https://bankpreidc.zatech.com/loan/pxmifj/application/get-pending	53	查看详情
	2019-12-10 14:35:38	https://bankpreidc.zatech.com/acc/pid8dc/rate/find	189	查看详情
	2019-12-10 14:35:38	https://bankpreidc.zatech.com/mb/pwd23q/queryDict	33	查看详情

- 执行过程中平均内存使用71 MB，平均 FPS 29.54，平均 CPU使用率3.1%（机型 iPhone 8）；
- 在启动和登录操作时 FPS 变化较大，出现 BigJank（FPS 瞬间波动），可能是 RN和 flutter 页面相互跳转导致；
- 注：性能数据分析，这一块性能数据是通过 perfdog 获取，暂时未进行二次分析处理，有一定参考意义。由于 app 是采用混合开发模式，FPS并不一定能反应当前的流畅度；

> *那么这些数据是如何统计出来的呢，虽然通过手工测试的方式也可以记录和收集，但显得过于笨拙，下面来介绍下测试方案的设计。*

方案设计

> ***整体方案如下图，由于本方案不用通过开发同学写log 记录点击事件和接口请求的方式，理论上只要 App 没有做 https 双向校验都可以适用本方案，通用性极强。具体测试步骤和关键点：***



- UI 测试数据。先一句话简要介绍UI 自动化的实现原理：在手机安装testdriver（appium/webdriveragent） 后，可以通过 json protocol形式的 API 实现对手机 app 的操作。具体详情可见km。其中关键的点是在**点击 UI 元素后记录点击的时间戳**，用作与后面的接口，性能数据做关联分析，用例执行结果详情结构如下：

```
⊟{
    "timestamp":1574754767,
    "stepresult":"success",
    "stepname":"点击继续",
    "uielement":"继续"
},
⊟{
    "timestamp":1574754772,
    "stepresult":"success",
    "stepname":"点击确定",
    "uielement":"确定"
},
⊟{
    "timestamp":1574754776,
    "stepresult":"success",
    "stepname":"判断昵称是否存入成功",
    "uielement":"存入成功"
}
```


- 接口数据。通过 whistle 的 autosave 插件保存自动化用例执行过程中的请求，**设置host过滤规则通过写文件方式记录 https接口请求**，保存文件结构如下：


```
"startTime":1575959901785,
"id":"1575959901785-119",
"url":"https://bankpreidc.zatech.com/acc/ncsate/asset/find",
"req":⊟{
    "method":"POST",
    "httpVersion":"1.1",
    "ip":"172.16.238.139",
    "port":62660,
    "rawHeaderNames":⊟{
        "host":"Host",
        "language":"language",
        "did":"did",
        "reqseq":"reqSeq",
```


- 性能数据。执行用例过程中通过Perfdog记录 app 实时性能数据，导出到excel 文件，文件如下：

3	DeviceInfo								
4	Device Nar	Device Typ	OS	CPU Type	CPU Arch	CPU CoreN	CPU Freq	GPU Type	
5	iPhone 8	iPhone 8	12.4.1(16G102)	Apple A11	Apple A11	6	[0_ 2390]	Apple A10X Fusion	
6									
7	Stat								
8	AvgFPS	FPS>=18(%	FPS>=25(%s)	VarFPS	DropFPS(/h	Jank(/10mi	BigJank(/10	FTime>=10	DeltaFT
9	29.54	47.7	47.1	739.03	162	223.2	219.6	1.6	146
10									
11	Data_v2								
12	Num	time	absTime	label	Notes	FPS	Jank	BigJank	Total(%)
13	1	0	1.57596E+12	label1		40	0	0	
14	2	1010	1.57596E+12	label1		57	0	0	
15	3	2020	1.57596E+12	label1		58	0	0	
16	4	3027	1.57596E+12	label1		59	0	0	
17	5	4036	1.57596E+12	label1		59	0	0	

- 优化与展望
  - 三份不同纬度数据的展现形式问题。目前只做到接口和 UI 点击事件的联动， app 实时性能（主要指CPU、内存、FPS 三项）的数据需要处理分析后进行联动，待完成。
  - 加强各业务场景具体 case 的分析，比如在某一特定机型，某一特定业务场景下出现的用户卡顿问题，需要完善测试用例来覆盖。
  - 要完全做到E2E测试，客户端性能和接口请求可以通过 UI 自动化的方式来测试，服务端的耗时分布和负载情况改如何通过用户操作前端传入的动作或事件来测试呢？