

Table of Contents

Introduction	0
启动参数介绍	1
Android遍历	2
iOS遍历	3
遍历控制	4
动作触发器	5
自动化测试结合	6
兼容性测试	7
XPath表达式学习	8
插件	9
插件开发	9.1
代理插件	9.2
Log插件	9.3
JDIAutoDebug插件	9.4

AppCrawler

为什么做这个工具

- 各大云市场上自动遍历功能都多有限制企业无法自由定制.
- 解决monkey等工具可控性差的缺点
- 发现深层次的UI兼容性问题
- 通过新老版本的diff可以发现每个版本的UI变动范围

设计目标

- 自动爬取加上规则引导(完成)
- 支持定制化, 可以自己设定遍历深度(完成)
- 支持插件化, 允许别人改造和增强(完成)
- 支持滑动等更多动作(完成)
- 支持自动截获接口请求(完成)
- 支持新老版本的界面对比(Doing)
- 云端兼容性测试服务利用, 支持Testin MQC MTC(Doing)

安装依赖

mac下安装appium

```
brew install node  
npm install -g appium
```

真机或者模拟器均可. 确保adb devices可以看到就行

启动appium

已经在最新版本的appium 1.5.1下做过测试.推荐使用最新的版本启动appium

```
#ios测试默认连接4724默认端口  
appium --session-override  
#android测试默认连接4730默认端口  
appium --session-override -p 4730
```

下载appcrawler.

下载appcrawler工具, 解压. 只要有java即可, 目前是打包为一个jar文件. 开源日期还未确定, 等不及的同学可自行反编译.

快速遍历

```
#运行测试  
java -jar appcrawler-1.3.0.jar -a xueqiu.apk -o ~/temp/android_7  
appcrawler -a xueqiu.apk  
#查看帮助文档  
java -jar appcrawler-1.2.1.jar --help  
#如果是zip包执行运行bin目录下appcrawler即可  
appcrawler --help
```

配置文件运行方式

```
#配置文件的方式运行  
#Android测试  
appcrawler -c conf/xueqiu.json -a xueqiu.apk  
#iOS测试  
appcrawler -c conf/xueqiu.json -a xueqiu.app
```

输出结果

在当前目录下会生成一个包含输出结果的目录, 以时间命名. 包含了如下的测试结果

- 每个步骤的截图
- 所有遍历过的控件组成的思维导图
- 一些点击和元素log

设计理念

界面唯一性

每个screen都有一个唯一的id, 这样可以类比为普通的接口测试中的url.
android的url默认为当前的activity名字.

iOS没有activity概念, 默认使用当前页面dom的md5值的后五位作为标记. 如果页面不变. 那么这个md5值也不会变.

也可以自己指定某些特征作为url, 比如title或者某些关键控件的文本

控件的唯一性

取决于这个url和控件自身的id name tag text loc等属性.

唯一原则

界面的编码和控件的编码基本决定了他们的唯一性. 可以认为app里面的每个控件都具备一个url.

```
KwlOpenActivity-action_bar_title_decor_content_parent-content-fr  
雪球-SNBPublicTimelineView_雪球-选股策略-选股策略
```

每个唯一的控件只会被遍历一次.

举个例子.

一个输入框id=input, 在多个页面中都出现了.

如果url为空, 那么它只会被点击一次. 如果url设置为当前activity的名字, 那么有多少页面包含它他就会被点击多少次.

url的定义是一门艺术, 可以决定如何优雅的遍历

启动参数介绍

```
java -jar target/scala-2.11/appcrawler-1.3.0.jar
```

AppCrawler 1.3.0

app爬虫，用于自动遍历测试。支持Android和iOS，支持真机和模拟器

灵感来源：晓光 泉龙 杨榕 雪球测试团队出品

移动测试技术交流 <https://testerhome.com>

Usage: appcrawler [sbt] [options] <args>...

-a <value> | --app <value>

Android或者iOS的文件地址，可以是网络地址，赋值给appium的app选项

-c <value> | --conf <value>

配置文件地址

-p <value> | --platform <value>

平台类型android或者ios，默认会根据app后缀名自动判断

-t <value> | --maxTime <value>

最大运行时间。单位为秒。超过此值会退出。默认最长运行3个小时

-u <value> | --appium <value>

appium的url地址

-o <value> | --output <value>

遍历结果的保存目录。里面会存放遍历生成的截图，思维导图和日志

--capability k1=v1,k2=v2...

appium capability选项，这个参数会覆盖-c指定的配置模板参数，用于覆盖

-v | --verbose

是否展示更多debug信息

--help

示例

```
appcrawler -a xueqiu.apk
```

```
appcrawler -a xueqiu.apk --capability noReset=true
```

```
appcrawler -c conf/xueqiu.json
```

```
appcrawler -c xueqiu.json --capability udid=[你的udid] -a Snowball
```

```
appcrawler -c xueqiu.json -a Snowball.app -u 4730
```

```
appcrawler -c xueqiu.json -a Snowball.app -u http://127.0.0.1:47
```

Command: sbt [<sbt params>...]

sbt是一个调用sbt命令运行测试的开关. 可以传递sbt的参数

<sbt params>...

sbt的参数列表

Android遍历

在android上运行

启动appium

```
appium
```

启动遍历

```
appcrawler -c src/universal/conf/xueqiu.json -p android -a ~/D
```

iOS遍历

模拟器运行

启动appium

```
appium
```

开始遍历

```
appcrawler -c src/universal/conf/xueqiu.json -p ios -a <你的api
```

\$USER表示当前的用户. xcode编译出来的app地址默认是

```
/Users/$USER/Library/Developer/Xcode/DerivedData/Snowball-ckpjeg
```

真机运行

使用xcode编译源代码. 使用开发证书才能做自动化. 编译出真机可自动化的.app或者.ipa包

```
appcrawler -c src/universal/conf/xueqiu.json -p ios -a /Users/
```

遍历控制

后退标记back

android默认是back键,不需要设定.

iOS上没有back键, 需要自己指定, 通过xpath定位方式指定遍历完所有控件应该点击什么控件返回.

```
"backButton": [  
    "//*[@name='nav_icon_back']",  
    "//UIAButton[@name='取消']",  
    "//UIAButton[@name='Cancel']",  
    "//UIAButton[@name='关闭']",  
    "//*[@value='首页']",  
    "//UIAButton[@name='首页']"  
,
```

黑名单black 控件黑名单为black方法. 他会绕过id name或者text中包含特定关键词的控件. url黑名单可以绕过特定的activity或者window

```
"blackList" : [ "消息", "聊天室" ]
```

遍历顺序控制 适用于在一些列表页或者tab页中精确的控制点击顺序
selectedList表示默认要遍历的元素特征 first表示优先遍历元素特征 last表示最后应该遍历的元素特征 统一使用XPath来表示

```
"firstList": [
    "//android.widget.ListView//android.widget.TextView",
    "//android.widget.ListView//android.widget.Button"
],
"selectedList": [
    "//*[@enabled='true' and @resource-id!='' and not(@clickable='true')]",
    "//*[@enabled='true' and @content-desc!='' and not(@clickable='true')]",
    "//android.widget.TextView[@enabled='true' and @clickable='true']",
    "//android.widget.ImageView[@clickable='true']",
    "//android.widget.ImageView[@enabled='true' and @clickable='true']"
],
"lastList": [
    "//*[contains(@resource-id, 'group_header_view')]//a"
],
```

url控制

支持黑名单, 最大遍历深度, 和重命名url, 重新设定初始url

```
"defineUrl": "//*[contains(@resource-id, '_title')]",
"baseUrl": ". *Main.*",
"maxDepth": 3,
"blackUrlList": [
    "StockMoreInfoActivity",
    "StockDetailActivity",
    "UserProfileActivity"
],
```


动作触发器

启动脚本

用于划过开屏的各种操作. 遍历开始前会先运行这个命令序列. 目前默认就会尝试滑动 你可以用这个配置作微调.

```
"startupActions" : [  
    "scroll left",  
    "scroll left",  
    "scroll left",  
    "scroll left",  
    "scroll down"  
]
```

触发配置

idOrName 表示只要是id或者text属性匹配到了给定的值, 就会触发action的动作.

idOrName支持严格正则表达式. 比如某个按钮的文本是 功能搬到这里啦 的提示控件可以通过 .*这里.* 来匹配到.

action支持如下动作

```
click  
scroll  
scroll left  
scroll up  
scroll down  
任意文本非上述文本会被当做输入
```

times表示规则被应用几次后删除, 如果是0表示永久生效.

控件触发器示例

action如果是click就是点击. 如果是非click 就认为是输入内容. idOrName可以是xpath表达式

```
"elementActions" : [ {
    "action" : "click",
    "idOrName" : "登 录",
    "times" : 2
}, {
    "action" : "click",
    "idOrName" : "登录",
    "times" : 2
}, {
    "action" : "15600534760",
    "idOrName" : "account",
    "times" : 1
}, {
    "action" : "scroll left",
    "idOrName" : "专题",
    "times" : 1
},
{
    "action" : "scroll down",
    "idOrName" : "专题",
    "times" : 1
},
{
    "action" : "xxxxxxx",
    "idOrName" : "password",
    "times" : 1
}, {
    "action" : "click",
    "idOrName" : "button_next",
    "times" : 1
}, {
    "action" : "click",
```

```
"idOrName" : "稍后再说",
"times" : 0
}, {
"action" : "click",
"idOrName" : "这里可以.*",
"times" : 0
}
]
```

自动化测试结合

测试框架选择

使用了scalatest这个强大的测试框架作为依托.

他本身是支持BDD和TDD风格的.

http://www.scalatest.org/user_guide/selecting_a_style

scalatest自带的selenium支持

详情可参考 http://www.scalatest.org/user_guide/using_selenium

这个selenium框架易用性很好. 支持selenium意味着可以支持appium, 先看示例

```
class BlogSpec extends FlatSpec with ShouldMatchers with WebBrowser
  implicit val webDriver: WebDriver = new HtmlUnitDriver
  val host = "http://localhost:9000/"

  "The blog app home page" should "have the correct title" in {
    go to (host + "index.html")
    pageTitle should be ("Awesome Blog")
  }
}
```

Appium支持

在selenium支持的基础上做了一个针对appium的封装 为了让自动化能过做到极简. 我做了一些极端的设计. 在selenium的基础之上封装了几个新的功能. 效果如下

```
test("股票"){
    click on see("自选")
    tree()
    click on see("股票")
    tree("//android.widget.ImageView")
    click on see("大港股份")
    tree()
    tree("action_bar_title")("text") should be equals("大港股份")
    crawl("/Users/seveniruby/projects/LBSRefresh/src/universal/c
}
}
```

appium关键字

只增加了少数几个方法. 加上原来的click on方法. 只有4个关键词可用 see 定位方法 tree 访问方法 send 发送文字 click on 点击

see

唯一的元素定位方式.

see是引用了<阿凡达>电影里面一句台词"I See You". 它的作用是当你看到一个控件, 你应该可以根据看见的东西就可以定位它, 无须借助其他工具或者使用findElementByXXX之类的函数.

比如有个Button, 名字是"登录", 它的id是account, 定位它可以通过如下多种方式的任何一种

- see("登录")
- see("登")
- see("录")
- see("account")
- see("acc")
- see("//UIAButton[@id='account']")

如果当前界面中存在了有歧义的空间, 比如其他一个名字为"登录"的输入框. 那么上述定位方法中定位中两个控件的定位方法会失败, 你需要自己调整即可.

这就是关于元素定位你只需要用**see**这个方法即可.

tree

打印当前界面布局结构. 是个格式化的xml内容.

如果传入一个参数 则会打印符合你给定关键词或者xpath定位的元素列表
这个关键词是为了让你摆脱各类的元素定位工具, 比如appium inspector, uiautomator之类的工具.

- tree() 输出当前的Android或者iOS的dom树结构
- tree("登录") 输出满足see("登录")的所有控件的所有属性

用于获取当前dom树里面满足条件的第一个控件的属性. 比如我想获取某个 TextView的相关属性. 可用

- tree("action_bar_title")("text") 文本
- tree("action_bar_title")("tag") 类型
- tree("action_bar_title")("selected") 是否选中

断言

支持标准的scalatest的should风格的断言. 支持两种风格的断言

assert风格

```
assert(2>1)
assert(attempted == 1, "Execution was attempted " + left + " tim
```

用法参考 http://www.scalatest.org/user_guide/using_assertions

should风格

这也是我喜欢的风格

```
//数字比较
one should be < 7
one should be > 0
one should be <= 7
one should be >= 0
sevenDotOh should be (6.9 +- 0.2)

//字符串断言
traversable should contain ("five")
string should startWith regex "Hel*o"
string should fullyMatch regex """(-)?(\d+)(\.\d*)?"""
"howdy" should contain oneOf ('a', 'b', 'c', 'd')
```

完整用法参考 http://www.scalatest.org/user_guide/using_matchers

TODO

目前的功能已经可以支持基础的自动化测试了. 仍然有一些内容需要完善. 我可能没有精力去维护了. 留给大家做参考吧.

- 支持滚动
- 支持单选 多选 滑块等封装动作
- 交互式调试

兼容性测试

ToDo

结合MQC和Testin在线测试服务

本地兼容性测试

本地通过截图来判断不同设备上的部分功能是否满足需要. 只是一个初级的demo供参考. 真实的应用需要做到设备参数和用例分离.

```
class TestAppiumDSL extends AppiumDSL {
    import org.scalatest.prop.TableDrivenPropertyChecks._
    val table = Table(
        ("iPhone 4s", "9.1"),
        ("iPhone 5", "8.1"),
        ("iPhone 5", "9.2"),
        ("iPhone 5s", "9.1"),
        ("iPhone 6", "8.1"),
        ("iPhone 6", "9.2"),
        ("iPhone 6 Plus", "9.1"),
        ("iPhone 6s", "9.1"),
        ("iPhone 6s", "9.2"),
        ("iPad Air", "9.1"),
        ("iPad Air 2", "9.1"),
        ("iPad Pro", "9.1"),
        ("iPad Retina", "8.1"),
        ("iPad Retina", "8.2")
    )
    forAll(table) { (device: String, version: String) => {
        test(s"兼容性测试-${device}-${version}_登录验证iphone", Tag("7.
            ios(true)
            config("deviceName", device)
```

```
config("platformVersion", version)
setCaptureDir("/Users/seveniruby/temp/crawl4")
appium()
captureTo(s"${device}-${version}_init.png")
click on see("手机号")
send("1560053xxxx")
click on see("//UIASecureTextField")
send("password")
captureTo(s"${device}-${version}_login.png")
click on see("登录")
captureTo(s"${device}-${version}_main.png")
if(device.matches(".*iPad.*")){
    click on see("//UIAButton[@path=\"/0/0/0/5\"]")
}else {
    click on see("//UIAButton[@path=\"/0/0/3/5\"]")
}
tree("seveniruby")("name") should be equals "seveniruby"
captureTo(s"${device}-${version}_profile.png")
}

}

}

override def afterEach(): Unit ={
  log.info("quit")
  quit()
}
}
```

XPath表达式学习

学习渠道

w3school肯定是最好的教程

常见用法

Android和iOS控件差异

tag名字是不一样的.

```
UIAXXX  
android.view.View  
android.widget.XXXXX
```

关键的定位属性也不一样

iOS

```
name  
label  
value
```

Android

```
resource-id  
content-desc  
text
```

常见XPath表达式用法

```
//*[not(ancestor-or-self::UIITableView)]  
//*[not(ancestor-or-self::UIStatusBar)]  
//*[@resource-id='com.xueqiu.android:id/action_search']/parent::  
//*[@resource-id='com.xueqiu.android:id/action_search']  
//*[contains(name(), 'Text')]  
//*[@resource-id!='' and not(contains(name(), 'Layout'))]
```

插件

插件开发

期望是和burp suite一样, 允许用户可以使用python ruby java scala等语言来设计插件.

代理插件

自动获取app上每次点击对应的网络请求. 支持http和https

安装

目前是默认自带.

启用

在配置文件中加入插件

```
"pluginList" : [  
    "com.xueqiu.qa.appcrawler.plugin.TagLimitPlugin",  
    "com.xueqiu.qa.appcrawler.plugin.IDeviceScreenshot",  
    "com.xueqiu.qa.appcrawler.plugin.ProxyPlugin"  
,
```

代理插件默认开启7771端口.

配置你的Android或者iOS的设备的代理. 指向你当前运行appcrawler的机器和7771端口

结果

在做每个点击的时候都会保存这期间发送的请求. 也就是记录前后两次点击中间的所有通过代理的请求.

最后会在结果目录里面生成后缀名为har的文件.

Log插件

作用

自动记录Android的LogCat或者iOS的syslog.

安装

目前是默认自带.

启用

在配置文件中加入插件

```
"pluginList" : [  
    "com.xueqiu.qa.appcrawler.plugin.TagLimitPlugin",  
    "com.xueqiu.qa.appcrawler.plugin.LogPlugin"  
,
```

结果

记录一次点击事件后所发生的log记录. 并保存为后缀名为.log的文件中.

JDIAndroid自动调试插件

基于JDI技术自动调试Android应用. 捕获应用内的每次事件触发的代码. 建立起一个事件和代码的关联关系. 可以获得如下的好处.

- 用于将每次的git diff和业务建立关联. 评估变更的影响范围.
- 捕获代码flow找出空指针等有规则的调用序列