

Table of Contents

介绍	0
启动参数介绍	1
Android遍历	2
遍历控制	3
iOS遍历	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
TagLimit插件	9.4
常见问题	10

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appium 1.5.3下做过测试 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.jar -a xueqiu.apk -o ~/temp/android_7.7.1_2
appcrawler -a xueqiu.apk
#查看帮助文档
java -jar appcrawler.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.

```
Kw1openActivity-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 appcrawler-1.4.0.jar
```

AppCrawler 1.4.0

app爬虫，用于自动遍历测试。支持Android和iOS，支持真机和模拟器

移动测试技术交流 <https://testerhome.com>

感谢：晓光 泉龙 杨榕 恒温 mikezhou

Usage: appcrawler [options]

-a, --app <value>	Android或者iOS的文件地址，可以是网络地址，
-c, --conf <value>	配置文件地址
-p, --platform <value>	平台类型android或者ios，默认会根据app后缀
-t, --maxTime <value>	最大运行时间。单位为秒。超过此值会退出。默认
-u, --appium <value>	appium的url地址
-o, --output <value>	遍历结果的保存目录。里面会存放遍历生成的截图
--capability k1=v1,k2=v2...	appium capability选项，这个参数会覆盖-c指定的配置文件中的设置
-vv, --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.app
```

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

```
appcrawler -c xueqiu.json -a Snowball.app -u http://127.0.0.1:4730
```

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 ~/Downloads/xueqiu.apk
```

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表示要遍历的元素特征

firstList表示优先遍历元素特征

lastList表示最后应该遍历的元素特征

统一使用XPath来表示

需要注意的是firstList和lastList指定的元素必须包含在selectedList

```
"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"
],
```

#

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app地址比如xueqiu.app>
```

xcode编译出来的app地址可通过编译过程自己查看

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 \
-a Snowball.app
```

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" : "xxxxxxxx",  
    "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

appcrawler在selenium支持的基础上做了一个针对appium的封装,类名叫 MiniAppium.他具有如下的特色

- 设计极简, 除了selenium的自身支持外,增加了几个api用于app的测试
- 封装了appium命令的启动停止

- 强大的断言

```
test("验证雪球登陆功能"){
    //启动appium
    start()
    //配置appium client
    config("app", "/Users/seveniruby/Downloads/xueqiu.apk")
    config("appPackage", "com.xueqiu.android")
    config("appActivity", ".view.WelcomeActivityAlias")
    appium()
    //自动化
    see("输入手机号").send("13067754297")
    see("password").send("xueqiu4297")
    see("button_next").tap()
    see("tip").tap().tap().tap()
    swipe("down")
    see("user_profile_icon").tap()
    //断言
    see("screen_name").nodes.head("text") should equal("huangyar")
    see("screen_name")("text") shouldEqual "huangyansheng"
}
```

appium关键字

在selenium支持的基础上只增加了少数几个方法.

命令	用途
see	元素定位与属性提取
tap	点击
send	输入文本
swipe	滑动

原来scalatest的selenium的支持仍然全部可用. 比如

```
#
```

```
click on id("login")
```

see

唯一的元素定位api.

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, 并获取到这个控件的属性, 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']")
- see("screen_name")("text")
- see("screen_name").nodes.head("text")
- see("action_bar_title")("text") 文本
- see("action_bar_title")("tag") 类型
- see("action_bar_title")("selected") 是否选中

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.

动作

目前只封装了3个动作. tap send swipe.

#

```
see("输入手机号").send("13067754297")
see("password").send("xueqiu4297")
see("button_next").tap()
```

支持链式调用. 当然不推荐日常使用

```
//对三次连续出现的tip控件点击三次.
see("tip").tap().tap().tap()
see("输入手机号").send("13067754297").see("password").send("xueqiu4297")
```

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 + " times")
```

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
sevenDot0h 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

第一个自动化测试用例

安装scala与sbt, windows同学请自行找办法解决

```
brew install scala sbt
```

进入appcrawler的项目目录, 或者自己创建一个目录也是可以的. 目录结构如下 在lib下存放appcrawler.jar文件.

在src/test/scala下存放测试用例.

build.sbt是个固定模板. 使用appcrawler自己的即可.

具体的配置可参考scala的sbt构建管理帮助文档, 可以使用idea等IDE管理

```
build.sbt
lib/
src/
  main/
  test/
    scala/
```

在src/test/scala下编写自己的测试用例.

```
package com.xueqiu.qa.appcrawler.it

import com.xueqiu.qa.appcrawler.MiniAppium

/**
 * Created by seveniruby on 16/5/21.
 */

class TestAndroidSimulator extends MiniAppium {
    override def beforeAll(): Unit = {
        start()
        config("app", "/Users/seveniruby/Downloads/xueqiu.apk")
        config("appPackage", "com.xueqiu.android")
        config("appActivity", ".view.WelcomeActivityAlias")
        config("fullReset", "false")
        config("noReset", "true")
        appium()
        login()
        quit()
    }

    def login(): Unit = {
        swipe("left")
        swipe("down")
        see("输入手机号").send("13067754297")
        see("password").send("xueqiu4297")
        see("button_next").tap()
        see("tip").tap().tap().tap()
        swipe("down")
    }

    override def beforeEach(): Unit = {
        config("appPackage", "com.xueqiu.android")
        config("appActivity", ".view.WelcomeActivityAlias")
        appium()
    }

    test("test android simulator") {
```

```
#
```

```
    see("user_profile_icon").tap()
    see("screen_name").nodes.head("text") should equal("huangyar")
    see("screen_name").nodes.last("text") should be("huangyanshe")
    see("screen_name").attribute("text") shouldBe "huangyansheng"
    see("screen_name")("text") shouldEqual "huangyansheng"
}

test("自选") {
    swipe("down")
    see("自选").tap
    see("雪球100").tap
    swipe("down")
    see("stock_current_price")("text").toDouble should be > 1000
}

override def afterEach: Unit = {
    quit()
}
override def afterAll(): Unit = {
    stop()
}
}
```

执行测试

```
sbt test
```

xml和html格式的测试报告会生成在项目的target/report目录下

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.

- 支持单选 多选 滑块等封装动作

-
- 交互式调试
 - 生成robotframework style的测试报告

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.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.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.

TagLimit插件

作用

智能判断列表和其他的相似布局元素. 只遍历前3个相似空间. 适用于微博这种无限刷新的列表. 用于节省时间. 原理是利用特定元素的tag布局层级是否完全一样.

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"  
,
```

结果

无

Path must be a string

错误堆栈为

```
packageAndLaunchActivityFromManifest failed. Original error: Pat
```

这是appium的bug导致的. 一般表示没有成功读取app的appPackage和appActivity, 自己在配置文件或者命令行中显式指定即可. 比如

```
appcrawler -a xueqiu.apk -u 4730 \
--capability appPackage=com.xueqiu.android,appActivity=.view.Wel
```

找不到appcrawler工具

appcrawler的分发形式有两种. jar包方式和正常的zip格式. appcrawler在zip解压后的bin目录下. 如果下载的是jar包, 可执行

```
java -jar appcrawler.jar --help
```