

目录

1. 前期准备和调研	1
1.1 代码环境	1
1.2 Appium 所需环境	2
1.3 Linux 环境	3
1.4 Evosuite 的使用	3
2. Appium 的使用	3
2.1 Appium-desktop 的使用	3
2.2 Appium 错误解决	5
2.3 Appium 启动 iOS 的 app	5
2.4 Appium 日志	6
2.5 Appcrawler 的使用	7
3. Bash 知识了解	9
4. Appium 进阶	10
4.1 常用命令	10
4.2 自动化测试用例的编写	10
4.3 webdriver 协议	11
4.4 调试分析方法	12
4.5 toast 识别条件	12
4.6 等待	13
4.7 API	13
4.8 xpath 定位	14
4.9 webview 切换	14
4.10 capability 高级讲解	15
4.11 Grid 模式	15
4.12 pageObject	16
4.13 兼容性测试	16
4.14 常用测试类	18
5. Jenkins	19
5.1 Jenkins 基础	19

5.2 Jenkins 部署和安装方法	20
5.3 Jenkins 邮件及节点配置	20
5.4 分布式架构	21
5.5 分布式架构	21
5.6 集成 Maven TestNG	22
5.7 DevOps 持续交付和 pipeline 机制	23
5.8 Blue Ocean	26
5.9 Jenkins 总结	26
6 iOS 测试	27
6.1 启动方法	27
6.2 使用 Inspector 定位	27
6.3 iOS 用例演练	28
7.接口测试	28
7.1 Charles	28
7.2 Postman	29
7.3 JSON 字符串的处理	29
8.Spring	29
8.1 SpringIoC 概述	29
8.2 装配 Spring Bean	30
8.3 Spring AOP 简介	30
8.4 Spring 和数据库编程	30
8.5 MVC 设计概述	30
9. SpringBoot	31
9.1 简单配置	31
9.2 封装错误解决	32
9.3 热部署	33
9.4 Spring Boot 使用	33

1. 前期准备和调研

1.1 代码环境

1.1.1 Maven 配置

下载 Maven 压缩文件并解压后配置本地仓库路径，指向新建的本地仓库文件夹。在系统环境变量中完成配置。在 IntelliJ IDEA 中配置 maven 的 home directory、User setting files 和 Local repository，如图 1 所示。如遇到 maven 仓库移动到了其他位置也可参考此图配置。需要点击 Import Changes 时可点击并等待。

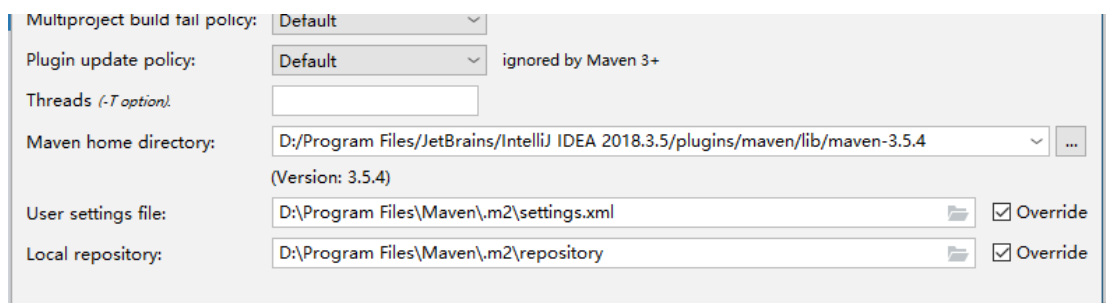


图 1-1 IntelliJ IDEA 中配置 maven 结果图

在命令行中简单测试配置成功，如图 2。

```
C:\Users\54251>javac -version
javac 1.8.0_171

C:\Users\54251>mvn -version
Apache Maven 3.6.0 (97c98ec64a1fdfee7767ce5fffb20918da4f719f3; 2018-10-25T02:41:47+08:00)
Maven home: D:\Program Files\Maven\apache-maven-3.6.0\bin\..
Java version: 1.8.0_171, vendor: Oracle Corporation, runtime: D:\Program Files\Java1.8.0_171\jdk1.8.0_171\jre
Default locale: zh_CN, platform encoding: GBK
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"

C:\Users\54251>
```

图 1-2 运行简单命令图

1.1.2 遇到的问题与解决

错误 1：使用 mvn 命令时提示错误。见图 3。通过只保留一个版本的 JAVA_HOME 解决。

```
The JAVA_HOME environment variable is not defined correctly
This environment variable is needed to run this program
NB: JAVA HOME should point to a JDK not a JRE
```

图 1-3 mvn 命令错误提示图

错误 2：可以使用 java 命令，不能使用 javac 命令。通过下移图 4 的第一个环境变量解决。

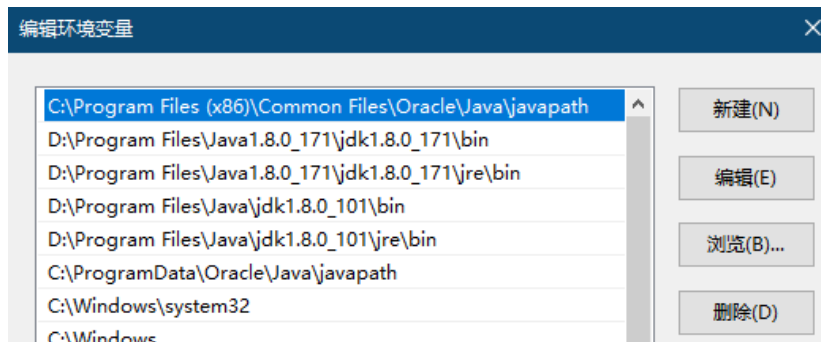


图 1-4 环境变量图

错误 3：运行项目时 jar 包解析失败，见图 5。通过更换为最新版本 jar 包解决。

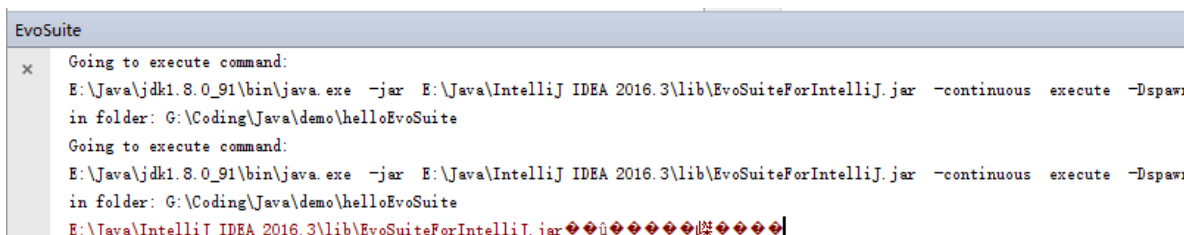


图 1-5 解析失败图

1.2 Appium 所需环境

包括 Android Studio 的安装、Android Emulator 的运行（可通过 `D:\Android\sdk\tools\emulator.exe -netdelay none -netspeed full -avd Nexus_5X_API_26_x86` 命令直接启动模拟器），（包括 adb）。IntelliJ IDEA 的集成开发环境。Node.js 的安装。

关于 Appium 的安装，可在命令行处通过 `npm install -g appium` 进行，并配置 `ANDROID_HOME` 的环境变量。通过输入 `npm install -g appium-doctor` 命令并运行 `appium-doctor` 可以检测 appium 相关的依赖是否已经安装完成，见图 1-6。

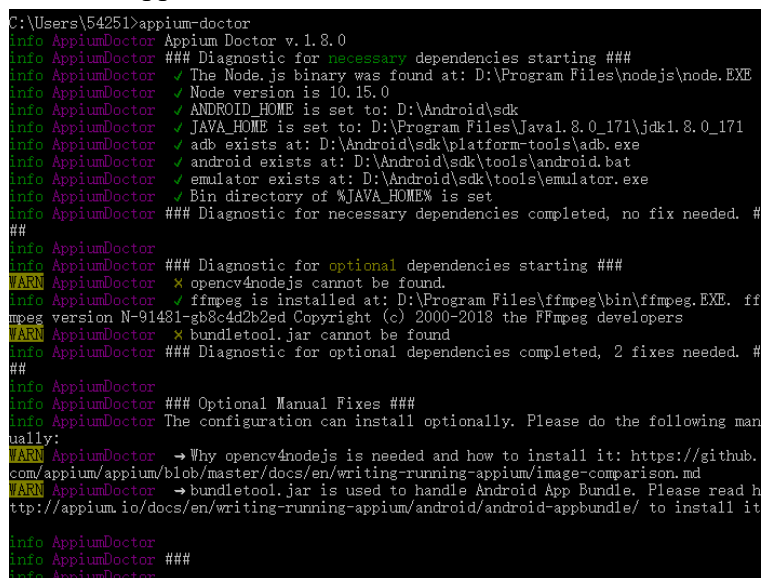


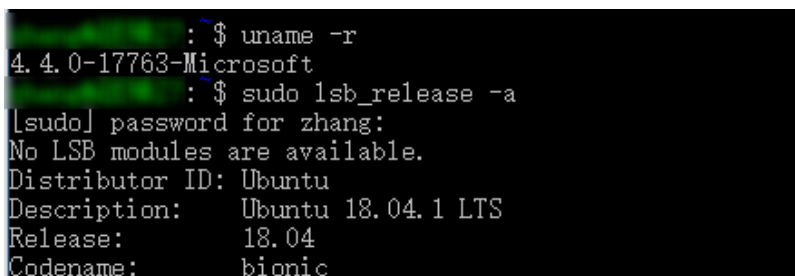
图 1-6 appium-doctor 图

同时可以通过 `pip install Appium-Python-Client` 安装 Appium-Python-Client，以便在 Python 环境中运行。在 Java 环境中则为 `appium.java_client`。

1.3 Linux 环境

1.3.1 WSL 的安装与使用

通过安装适用于 Windows 的 Linux 子系统（WSL）。可在 Windows 应用商店下载 Ubuntu 并安装。安装完成后见图 1-7。



```
zhang ~ : $ uname -r
4.4.0-17763-Microsoft
zhang ~ : $ sudo lsb_release -a
[sudo] password for zhang:
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 18.04.1 LTS
Release:        18.04
Codename:       bionic
```

图 1-7 Linux 子系统图

可以通过输入 `sudo apt-get install mate-desktop-environment` 等来开启图形化界面。也可以启用 SSH 并使用 SSH 客户端登录。

1.3.2 遇到的问题与解决

在安装和使用过程中遇到了如下问题：

问题 1：WslRegisterDistribution failed with error: 0x8007019e ：需再通过 Shell 打开子功能。

问题 2：unable to locate package mate-desktop-environment ：需 `sudo apt-get update`

1.4 Evosuite 的使用

略。

2. Appium 的使用

2.1 Appium-desktop 的使用

可安装 appium-desktop 版本。针对 Android 的 app。如图 2-1 的界面叫作 Inspector，通过输入 `deviceName`、`appPackage`、`platformName`、`appActivity` 等参数对所选择的 app 进行测试。

其中 `appPackage` 可以通过输入命令 `aapt dump badging 1.apk` 进行查找，或者使用 `adb logcat |grep ActivityManager` 并打开 app 后通过 `cmp` 查找；`appActivity` 可以使用 `adb logcat |grep ActivityManager` 并打开 app 后通过 `cmp` 查找。

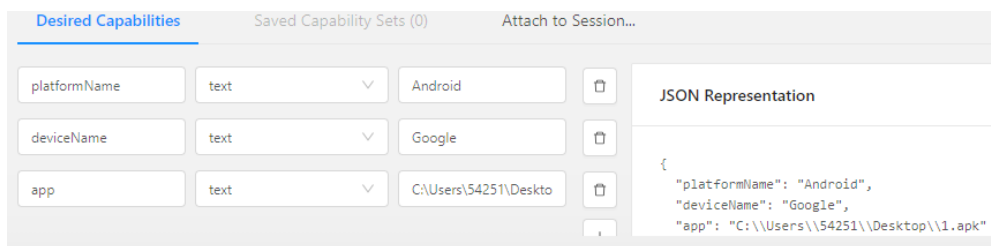


图 2-1 appium 参数设置图

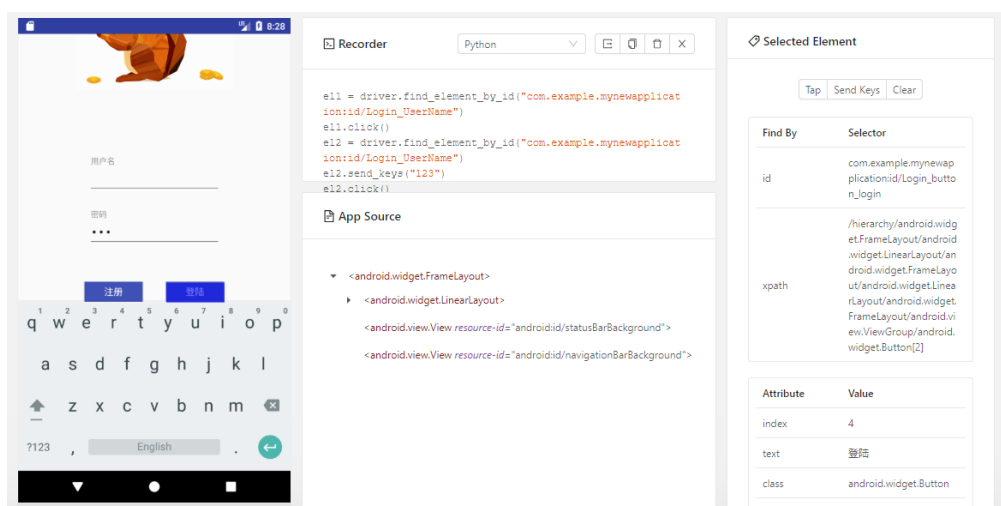


图 2-2 appium 截图

可以在 Appium 中通过 Recorder 录制动作生成代码，如图 2-2，并以 Python 或 Java 语言导出。在该界面还可以使用 Search for element 等功能。

在 Python 中运行时，可以通过直接运行代码来启动模拟器并进行自动化测试。

在 Java 中运行时，可在在搭配 maven 的环境中运行，同时支持 Junit。IntelliJ IDEA 中配置 Run 的 Edit Configuration 可参考图 2-3。

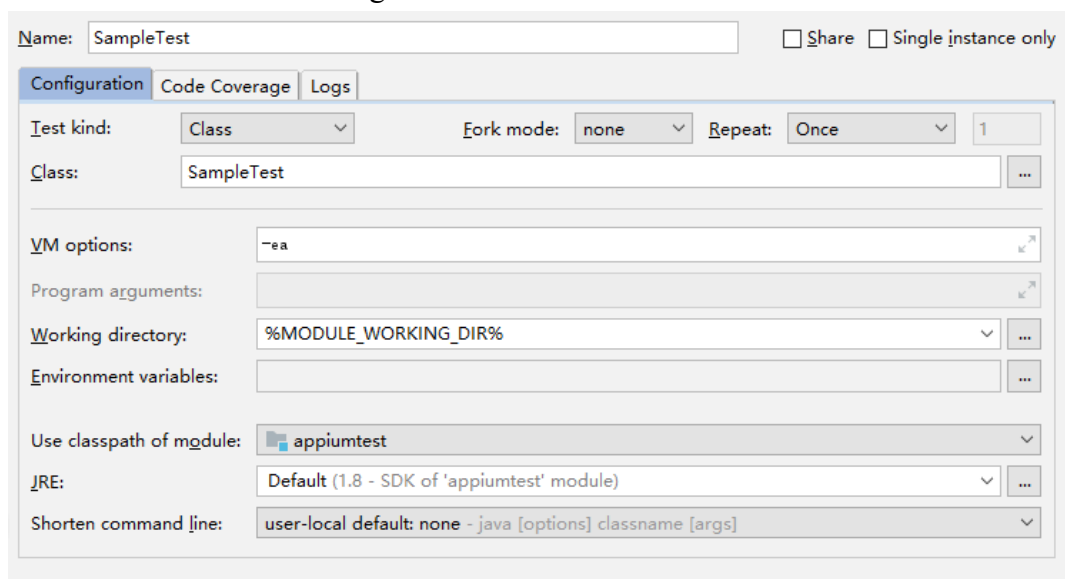


图 2-3 IntelliJ IDEA 配置 Run 图

通过不同语言的代码运行自动化测试时，Appium-desktop 中也会有对应的 LOG 输出，见图 2-4。

```
{ "cmd": "action", "action": "find", "params":
{"strategy": "id", "selector": "com.example.mynewapplication:id/Register_UserPassword",
[AndroidBootstrap] [BOOTSTRAP LOG] [debug] Got data from client:
{"cmd": "action", "action": "find", "params":
{"strategy": "id", "selector": "com.example.mynewapplication:id/Register_UserPassword",
[AndroidBootstrap] [BOOTSTRAP LOG] [debug] Got command of type ACTION
[AndroidBootstrap] [BOOTSTRAP LOG] [debug] Got command action: find
[AndroidBootstrap] [BOOTSTRAP LOG] [debug] Finding
'com.example.mynewapplication:id/Register_UserPassword' using 'ID' with the
contextId: '' multiple: false
[AndroidBootstrap] [BOOTSTRAP LOG] [debug] Using UiSelector[INSTANCE=0
```

图 2-4 Appium 的 Log 图

2.2 Appium 错误解决

模拟器安装某应用时如提示 `INSTALL_FAILED_NO_MATCHING_ABIS`，是由于模拟器 ABI 为 x86。

在 Python 中运行时，遇到的问题：

问题 1: cannot import name 'webdriver': 文件名命名成了 appium , 和 from ap
pium 重复

问题 2: name 'TouchAction' is not defined : 添加 from appium.webdriver.common.touch_action import TouchAction

在 Java 中运行时，遇到的问题：

问题 1. `import junit.framework.TestCase` 失败：将类放在标记为 `Test Resources Root` 的 `test` 目录下

问题 2. cannot resolve symbol TouchAction : import io.appium.java_client.TouchAction;

问题 3. cannot resolve method tap(int,int) : import io.appium.java_client.touch.offset.PointOption; 并将 (new TouchAction(driver)).tap((991, 1682)).perform() 改为 (new TouchAction(driver)).tap(PointOption.point(991, 1682)).perform();

2.3 Appium 启动 iOS 的 app

Register iOS App IDs、Bundle ID、Enable Services 等注册信息

iOS Provisioning Profiles

发布方式: Ad-hoc、In House、App Store。

元素定位方式: name、label、value。

2.3.1 常见命令

依赖工具包安装 `brew install libimobiledevice`

查看模拟器列表 `instruments -s devices`

查看真机列表 `idevice_id -l`

安装 app `ideviceinstaller -i demo.app`

命令行编译 `xcodebuild -scheme UICatalog -target iOS clean build`

`xcodebuild -scheme UICatalog -target iOS archive`

2.4 Appium 日志

Appium 日志中有许多输出。也对应着下方的命令。

如 Appium 日志中 info: [debug] [ADB] Running ‘/usr/local/opt/android-sdk/platform-tools/adb’ with args: [“-p”,5037,”-s”,”192.168.58.101:5555”,”shell”,”pm”,”list”,”packages”,”com.xueqiu.android”]

对应 `/usr/local/opt/android-sdk/platform-tools/adb -p 5037, -s "192.168.58.101:5555" shell pm list packages com.xueqiu.android`

如 Appium 日志中 info: [debug] [ADB] Running ‘/usr/local/opt/android-sdk/platform-tools/adb’ with args: [“-p”,5037,”-s”,”192.168.58.101:5555”,”wait-for-device”]

对应 `/usr/local/opt/android-sdk/platform-tools/adb -p 5037, -s “192.168.58.101:5555” wait-for-device`

Appium 日志中还有 info: [debug] [ADB] Running ‘/usr/local/opt/android-sdk/platform-tools/adb’ with args: [“-p”,5037,”-s”,”192.168.58.101:5555”,”shell”,”echo”,”ping”] 等，来测试设备是不是真的可以执行简单命令。

`uninstall io.appium.setting`、`install /usr/.../1.apk`、`shell dumpsys package io.appium.setting`、`shell ps` (查看当前有多少进程)

`shell getprop ro.build.version.sdk` 可以查看 api 对应的版本。还可以通过 `shell` 查看 `ro.build.version.release`、`ro.product.model` 和 `ro.product.manufacturer` 等内容。

`shell wm size` 可以获取屏幕分辨率、`forward tcp:4724 tcp:4724` 等

`shell am force-stop io.appium.setting/com.xueqiu.android`、`pm clear com.xueqiu.android`。其中 `am` 代表 activity manager，`pm` 代表 package manager。

重点命令 `push /usr/local/lib/.../bin/AppiumBootstrap.jar /data/local/tmp/`

重点命令 `shell uiautomator runtest AppiumBootstrap.jar -c io.appium.android.bootstrap -e pkg com.xueqiu.android -e disableAndroidWatchers false -e acceptSslCerts false` 从而支持自动化。

核心点命令 `shell am start -W -n com.xueqiu.android/.view.WelcomeActivityAlias -S -a android.intent.action.MAIN -c android.intent.category.LAUNCHER -f 0x10200000` 开始启动包。

以上是自动化的前置条件。

可以看到[HTTP] <-- POST /wd/hub/session/d1601a25-.../elements 这一请求。另外。在浏览器访问 127.0.0.1:4723/wd/hub/session/d1601a25-.../source 和在 Appium Desktop 中 App Source 的内容是一样的。

还可以使用 `grep run appium.log`、`grep Running appium.log` 等命令。

2.5 Appcrawler 的使用

2.5.1 运行方法

参照 Appcrawler 视频教程，在 cmd 中输入 `appium --session-override` 来运行 Appium，如图 2-5。同时启动 Android Emulator 模拟器，例如在 cmd 中输入 `D:\Android\SDK\tools\emulator.exe -netdelay none -netspeed full -avd Nexus_6_API_26` 来启动。

```
C:\Users\54251>appium --session-override
[Appium] Welcome to Appium v1.10.1
[Appium] Non-default server args:
[Appium]   sessionOverride: true
[Appium] Appium REST http interface listener started on 0.0.0.0:4723
```

图 2-5 在 cmd 中启动 Appium 图

下载好 Appcrawler 的 jar 包，并准备好测试用的 apk 文件。在新打开的 cmd 中输入 `java -jar appcrawler-2.4.0.jar`，如图 2-6，即可运行 Appcrawler。

```
e:\test>java -jar appcrawler-2.4.0.jar

-----
AppCrawler 2.4.0 [霍格沃兹测试学院特别纪念版]
Appium 1.8.1 Java8 tested
app爬虫，用于自动遍历测试。支持Android和iOS，支持真机和模拟器
项目地址: https://github.com/seveniruby/AppCrawler
移动测试技术交流: https://testerhome.com
联络作者: seveniruby@testerhome.com (思寒)
致谢: 晓光 泉龙 杨榕 恒温 mikezhou yaming116 沐木

-----

Usage: appcrawler [options]

-a, --app <value>      Android或者iOS的文件地址，可以是网络地址
-e, --encoding <value> set encoding, such as UTF-8 GBK
-c, --conf <value>     配置文件地址
```

图 2-6 启动 Appcrawler 图

Appcrawler.jar 包具体使用方法 `java -jar appcrawler.jar [options]`，如 `java -jar appcrawler.jar --capability "appPackage=com.android.xx,appActivity=.view.xxActivity"`

对 apk 文件进行测试后，会生成相应的测试文件与测试报告，见图 2-7 和图 2-8。

文档 (E:) > test > 20190225143011

名称	修改日期	类型	大小
css	2019/2/25 14:31	文件夹	
images	2019/2/25 14:31	文件夹	
js	2019/2/25 14:31	文件夹	
tmp	2019/2/25 14:31	文件夹	
0_com.example.mynewapplication-UserLoginActivity.tag=start.id=start.clicked.png	2019/2/25 14:30	PNG 文件	205 KB
0_com.example.mynewapplication-UserLoginActivity.tag=start.id=start.dom	2019/2/25 14:30	DOM 文件	9 KB
1_Steps.tag=.name=NOT_FOUND.click.png	2019/2/25 14:30	PNG 文件	205 KB
1_Steps.tag=.name=NOT_FOUND.clicked.png	2019/2/25 14:30	PNG 文件	205 KB
1_Steps.tag=.name=NOT_FOUND.dom	2019/2/25 14:30	DOM 文件	9 KB
2_com.example.mynewapplication-UserLoginActivity.tag=EditText.0.depth=10.id=Login_UserName.click.png	2019/2/25 14:30	PNG 文件	236 KB

图 2-7 Appcrawler 测试文件图

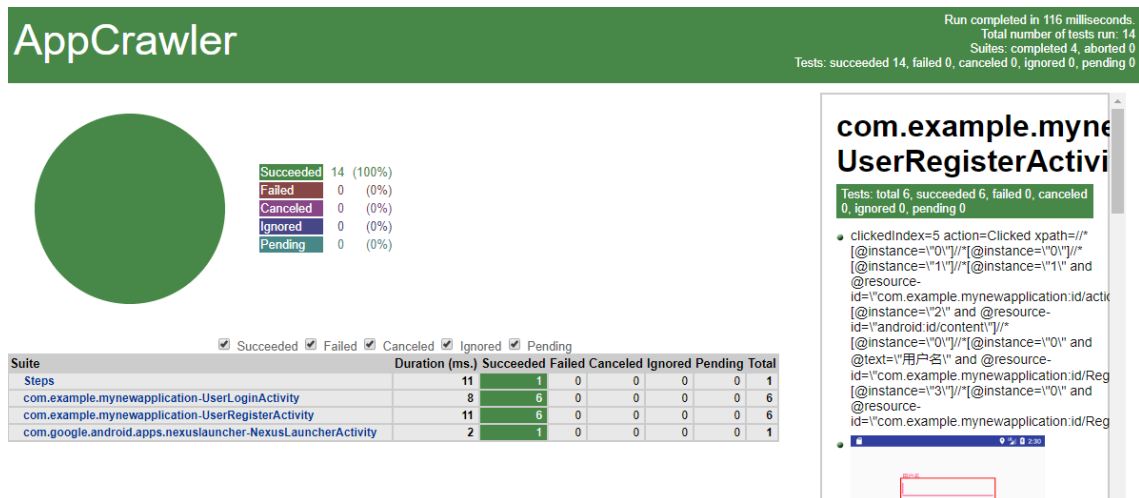


图 2-8 Appcrawler 测试报告图

在 IntelliJ IDEA 中，在有 Appcrawler 源码的情况下，可依次选中如图 2-9 中的内容后点击 Run Maven Build。成功后左侧会生成如图 2-10 所示的文件列表。

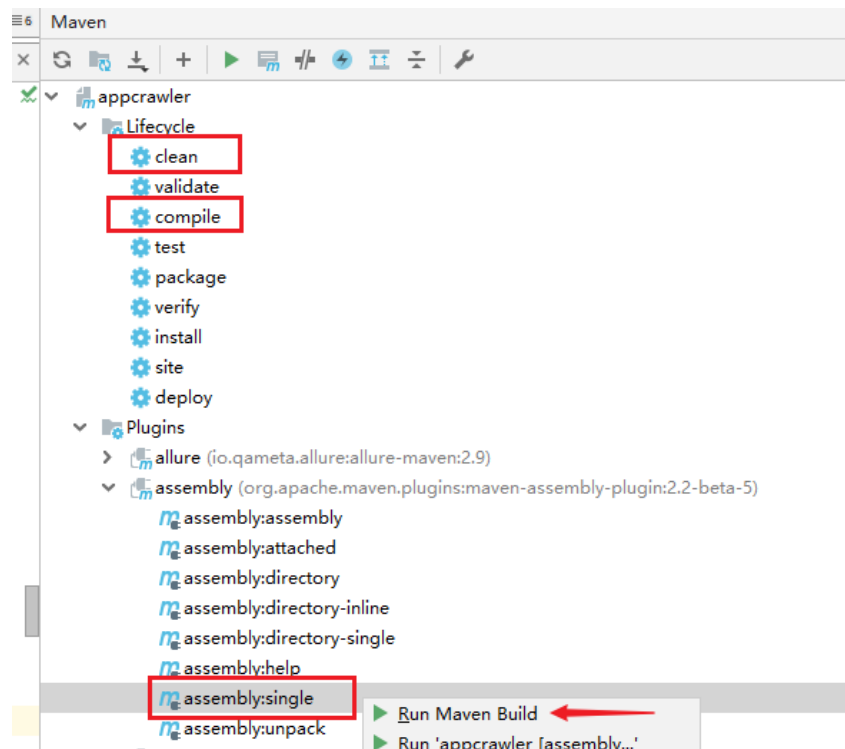


图 2-9 Run Maven Build 图

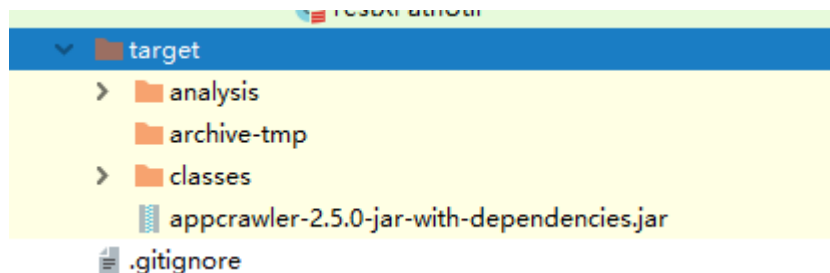


图 2-10 文件图

2.5.2 appcrawler 错误解决

运行 Appcrawler 时出现 AppiumClient.asyncTask 等错误，见图 2-11：使用新版 appcrawler 的 jar 包，如将 appcrawler-2.1.3.jar 的改为 appcrawler-2.4.0.jar。

```
2019-02-25 14:24:00 ERROR [AppiumClient.asyncTask.155] java.util.concurrent.FutureTask.report(FutureTask.java:122)
java.util.concurrent.FutureTask.get(FutureTask.java:206)
com.testertome.appcrawler.driver.WebDriver.$anonfun$asyncTask$1(WebDriver.scala:138)
scala.util.Try$.apply(Try.scala:209)
com.testertome.appcrawler.driver.WebDriver.asyncTask(WebDriver.scala:129)
com.testertome.appcrawler.driver.WebDriver.asyncTask(WebDriver.scala:128)
com.testertome.appcrawler.driver.AppiumClient.asyncTask(AppiumClient.scala:25)
com.testertome.appcrawler.driver.AppiumClient.$anonfun$getPageSource$1(AppiumClient.scala:311)
scala.collection.immutable.Range.foreach$mVc$sp(Range.scala:156)
com.testertome.appcrawler.driver.AppiumClient.getPageSource(AppiumClient.scala:310)
com.testertome.appcrawler.Crawler.refreshPage(Crawler.scala:555)
com.testertome.appcrawler.Crawler.start(Crawler.scala:146)
com.testertome.appcrawler.AppCrawler$.startCrawl(AppCrawler.scala:344)
com.testertome.appcrawler.AppCrawler$.parseParams(AppCrawler.scala:312)
com.testertome.appcrawler.AppCrawler$.main(AppCrawler.scala:92)
com.testertome.appcrawler.AppCrawler.main(AppCrawler.scala)
2019-02-25 14:24:00 WARN [AppiumClient.$anonfun$getPageSource$1.340] get page source error
2019-02-25 14:24:00 WARN [Crawler.refreshPage.562] page source get fail, go back
2019-02-25 14:24:00 INFO [Crawler.setElementAction.660] set action to back
2019-02-25 14:24:00 INFO [Crawler.runStartupScript.236] first refresh
2019-02-25 14:24:00 INFO [Crawler.doElementAction.976] current element = _startupActions-Start-0
2019-02-25 14:24:00 INFO [Crawler.doElementAction.977] current index = 0
2019-02-25 14:24:00 INFO [Crawler.doElementAction.978] current action =
2019-02-25 14:24:00 INFO [Crawler.doElementAction.979] current url =
2019-02-25 14:24:00 INFO [Crawler.doElementAction.980] current xpath = startupActions-Start-0
2019-02-25 14:24:00 INFO [Crawler.doElementAction.981] current tag path = _startupActions-Start-0
2019-02-25 14:24:00 INFO [Crawler.doElementAction.982] current file name =
2019-02-25 14:24:00 INFO [Crawler.doElementAction.983] current uri = startupActions-Start-0
startupActions
Exception in thread "main" java.util.NoSuchElementException: last of empty ListBuffer
    at scala.collection.mutable.ListBuffer.last(ListBuffer.scala:401)
    at com.testertome.appcrawler.DataRecord.last(DataRecord.scala:40)
    at com.testertome.appcrawler.Crawler.doElementAction(Crawler.scala:985)
    at com.testertome.appcrawler.Crawler.runStartupScript(Crawler.scala:238)
    at com.testertome.appcrawler.Crawler.start(Crawler.scala:152)
    at com.testertome.appcrawler.AppCrawler$.startCrawl(AppCrawler.scala:344)
    at com.testertome.appcrawler.AppCrawler$.parseParams(AppCrawler.scala:312)
    at com.testertome.appcrawler.AppCrawler$.main(AppCrawler.scala:92)
    at com.testertome.appcrawler.AppCrawler.main(AppCrawler.scala)
```

图 2-11 Appcrawler 错误提示图

3. Bash 知识了解

打开 Ubuntu 应用或者 cmd 内 bash。主要知识包括变量定义：简单、数组变量、转移字符、运算、字符串操作和布尔类型，控制结构：for 循环、while 循环、break、continue，shell 环境运行、管道、文件描述符，curl，grep、awk、sed 等命令。一些命令的运行见图 3-1。

```
~$ a=2019
~$ echo $a
2019
~$ b=January
~$ echo $b
January
~$ c="23日 12:34:56"
~$ echo $c
23日 12:34:56
```

图 3-1 简单的命令图

4. Appium 进阶

4.1 常用命令

常用命令有 adb devices、adb shell，aapt dump badging 1.apk、aapt list 1.apk。图 4-1 是其中某条命令在 D:\Android\sdk\build-tools\26.0.0 路径下运行后的输出结果。

```
D:\Android\sdk\build-tools\26.0.0>aapt dump badging e:\test\1.apk
package: name='com.example.mynewapplication', versionCode='1', versionName='2.33', platformBuildVersionName='8.0.0'
sdkVersion:'15'
targetSdkVersion:'26'
uses-permission: name='android.permission.INTERNET'
uses-permission: name='android.permission.WRITE_EXTERNAL_STORAGE'
application-label:'?????????????'
application-label-ar:'?????????????'
```

图 4-1 某一 adb 命令图

还包括 adb logcat |grep ActivityManager 等命令。

Adb 输出中的 D 代表 debug，I 代表 Info。

4.2 自动化测试用例的编写

Appium 中 Inspector 界面输入{"platformName": "android","deviceName": "Google", "appPackage": "io.appium.android.apis","appActivity": ".ApiDemos"}，可增加一项"app","e:\test\apiDemos.apk"或者在 cmd 使用 adb install e:\test\apiDemos.apk

4.2.1 框架改造和编写

1. 编写脚本包括：导入依赖、capabilities 设置（单独函数）、元素定位与操作 find+action、断言 assert。测试步骤包括：定位、交互、断言。

2. 改造参照 https://github.com/appium-boneyard/sample-code/blob/master/sample-code/examples/python/android_simple.py，包括 def setUp(self)、def tearDown(self)、def test_simple_actions(self)等

3. 定位：dom、dom 应用（节点和属性）和 xpath。

id(resource-id,accessibility id→content-desc,atring.xml id)、xpath、accessibilityId:content-desc(一般用不到，但还是用了)等

```
curl -X POST http://127.0.0.1:4723/wd/hub/session/$session_id/elements --data-binary '{"using":"id","value":"action_message"}' -H "Content-Type: application/json; charset=UTF-8" {"status":0, "value":[{"ELEMENT": "12"}],"sessionId":"d1601a25-..."}
curl -X POST http://127.0.0.1:4723/wd/hub/session/$session_id/elements --data-binary '{"using":"xpath","value":"///*[@resource-id=\'com.xueqiu.android.id/action_message\']"}' -H "Content-Type: application/json; charset=UTF-8" {"status":0, "value":[{"ELEMENT":"12"}],"sessionId":"d1601a25-..."}
```

优先使用 find_element_by_id，id 不存在或重复时使用 xpath。find_element_by_xpath...需要把绝对路径改成相对("//*[@text='推荐']")，如果重复可继续添加限制("//*[@text='推荐' and @resource-id='...']").两种方法时间差别不大。如遇到延迟问题可使

用 `sleep(3)`或 `wait`(在 python 中)。

4. java 中的依赖，通过 github 中 sample-code 中的 pom.xml 或者使用如下方法。

```
<dependencies>
<!-- https://mvnrepository.com/artifact/io.appium/java-client -->
<dependency>
  <groupId>io.appium</groupId>
  <artifactId>java-client</artifactId>
  <version>6.1.0</version>
</dependency>
<!-- https://mvnrepository.com/artifact/junit/junit -->
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
  <scope>test</scope>
</dependency>
</dependencies>
```

5. 封装多种定位方法，参考下面代码。

```
public WebElement locate(String locator) {
    if (locator.matches("\\\\ \\/.*"))
        { return driver.findElementByXPath(locator);}
    else{return driver.findElementById(locator); }
}
driver.find_element_by_xpath("//*[@text='推荐']").click();→locate("//*[@text='推荐']").click();
```

6.几种具体使用方法。

```
driver.findElementById("com.example.mynewapplication:id/Register_UserName");
driver.findElementByXPath("//*[@contains(@text,'Search')]");
driver.findElementByXPath("//*[@@text='Search']");
driver.findElementByAccessibilityId("Popup Menu").click();
```

4.3 webdriver 协议

具体组成:

<http://127.0.0.1:4723/wd/hub/sessions>

<http://127.0.0.1:4723/wd/hub/session/d1601a25-.../source>

http://127.0.0.1:4723/wd/hub/session/{session id}/element/{element id}/element

```
curl -X POST http://127.0.0.1:4723/wd/hub/session/{session id}/elements --data-binary '{"using":"id","value":"com.xueqiu.android:id/tv_login_by_wx"}' -H "Content-Type: application/json; charset=UTF-8" -vv
```

```
→ {"statues":0, "value":[{"ELEMENT":"7"}],.....}
```

```
curl -X GET http://127.0.0.1:4723/wd/hub/session/{session id}/element/7(为{element id})/text -H "Content-Type: application/json; charset=UTF-8" -vv
```

→{"statues":0, "value":"微信登陆",.....}

点击请求: curl -X POST http://127.0.0.1:4723/wd/hub/session/{session id}/element/{emelent id}/click -H "Content-Type: application/json; charet=UTF-8" -vv

改造成函数: click(){curl -X POST http://127.0.0.1:4723/wd/hub/session/{session id}/element/\$1(第一个参数)/click -H "Content-Type: application/json; charet=UTF-8" -vv; } →click 7

4.4 调试分析方法

Appium 的 log, 所有请求和结果, 都在 desktop 或者 cmd 里。

getPageSource, 可获得当前布局 dom 结构所有控件.xml 文件, 如下方代码。

```
try{
...
}finally{
System.out.println(driver.getPageSource());
}
```

也可以在脚本内调试, 利用 xpath 获取空间所有元素, driver.findElementsByPath("//*[@*")], 如下方代码。

```
try{
...
}finally{
for(AndroidElement e:driver.findElementsByXPath("//*[@*"))
{
System.out.println(e.getTagName());
System.out.println(e.getTagText());
///System.out.println(e.getAttribute("resource-id"));
//System.out.println(e.getAttribute("content-desc"));
}
}
```

4.5 toast 识别条件

toast 在 ApiDemos 中 Views 的 Popup Menu 中有一个。

新的框架 aotumationName: uiautomator2。使用 xpath 查找, //*[@class='android.widget.Toast']和//*[contains(@text, "xxx")]

Appium 中 Inspector 界面输入{"platformName": "android","deviceName": "Google", "appPackage": "io.appium.android.apis","appActivity": ".ApiDemos","automationName": "uiautomator2"}

例如, while true;do sleep 0.5; curl http://localhost:8200/wd/hub/session/3e943631-f896-452c-9cd5-3063f3452792/elements --data-binary '{"strategy":"xpath","selector":"//*[@class=\"android.widget.Toast\"]","context":"","multiple":true}';echo ;done 后可得到如图 4-2 所示的结果。

```

{"sessionId":"3e943631-f896-452c-9cd5-3063f3452792","status":0,"value":[]}
{"sessionId":"3e943631-f896-452c-9cd5-3063f3452792","status":0,"value":[]}
{"sessionId":"3e943631-f896-452c-9cd5-3063f3452792","status":0,"value":[]}
{"sessionId":"3e943631-f896-452c-9cd5-3063f3452792","status":0,"value":[]}
{"sessionId":"3e943631-f896-452c-9cd5-3063f3452792","status":0,"value":[]}
{"sessionId":"3e943631-f896-452c-9cd5-3063f3452792","status":0,"value":[]}
{"sessionId":"3e943631-f896-452c-9cd5-3063f3452792","status":0,"value":[{"ELEMENT":"b28be9b5-19cb-4bbb-a05a-10777440af1b"}]}
{"sessionId":"3e943631-f896-452c-9cd5-3063f3452792","status":0,"value":[{"ELEMENT":"ba280ed2-3785-40ed-962b-e3a8112683ec"}]}
{"sessionId":"3e943631-f896-452c-9cd5-3063f3452792","status":0,"value":[{"ELEMENT":"29d2e8ff-bbbd-403a-8379-ea239a71fb63"}]}
{"sessionId":"3e943631-f896-452c-9cd5-3063f3452792","status":0,"value":[{"ELEMENT":"85e02351-9b72-4c75-88a3-4729630cba13"}]}
{"sessionId":"3e943631-f896-452c-9cd5-3063f3452792","status":0,"value":[{"ELEMENT":"74afd84e-62af-41b1-b0ab-c242c0cb7dd3"}]}
{"sessionId":"3e943631-f896-452c-9cd5-3063f3452792","status":0,"value":[]}
{"sessionId":"3e943631-f896-452c-9cd5-3063f3452792","status":0,"value":[]}
{"sessionId":"3e943631-f896-452c-9cd5-3063f3452792","status":0,"value":[]}

```

图 4-2 toast 识别图

遇到的问题：Junit 运行时提示 Error: Could not access the Package Manager. Is the system running?：模拟器死机了，重启。

直接在 Appium-desktop 中录制比写代码快，一般情况下可以使用录制功能。

代码在参照 \appium 自动化测试\appiumtest - findToast 中的 testToast1()，可运行并得到 toast 的 text。具体改进，参照 testToast2()和 testToast3()。

4.6 等待

需等待控件加载，同时控件加载时位置会变动，是由其他控件还未加载出，需要等到位置处于稳定状态。

精确等待 Thread.sleep(1000);

全局隐式等待 driver.manage().timeouts().implicitlyWait(1, TimeUnit.SECONDS);

显式等待 WebDriverWait wait = new WebDriverWait(driver, 1);

WebElement element = wait.until(ExpectedConditions.elementToBeClickable(By.id("xx")));

4.7 API

可封装为单独的函数。

driver.rotate(ScreenOrientation.LANDSCAPE);

driver.rotate(ScreenOrientation.PORTRAIT);

driver.navigate().back();

driver.openNotifications();

System.out.println(driver.manage().logs().getAvailableLogTypes());

for(Object l: driver.manage().logs().get("logcat").getAll().toArray()){System.out.println(l);}

System.out.println(driver.getSupportedPerformanceDataTypes());→[memoryinfo,cpuinfo,batteryinfo,networkinfo]

```
System.out.println(driver.getPerformanceData("io.appium.android.apis(是 packagename)", "memoryinfo/cpuinfo/batteryinfo/networkinfo", 10))
```

```
@Test
public void testPerformance() throws Exception {
    System.out.println(driver.getPerformanceData("com.example.mynewapplication", "memoryinfo", 10));
    System.out.println(driver.getPerformanceData("com.example.mynewapplication", "cpuinfo", 10));
}
```

4.8 xpath 定位

可以使用 <https://www.freeformatter.com/xpath-tester.html> 提供的工具或者可以在 appium desktop 内通过放大镜测试

几种具体使用方法

```
driver.findElementByXPath("//*[@*");
```

```
driver.findElementByXPath("//*[@@test='Search']");
```

```
driver.findElementByXPath("//*[@contains(@text,'Search')]");
```

```
driver.findElementByXPath("//*[@contains(@class,'Image') and @instance=3]"); instance 为同类型的序号
```

```
//*[@contains(@text,'Search')] /ancestor::*[contains(name(),'EditText')] 父类
```

```
//*[@clickable="true"] //android.widget.TextView[string-length(@text)>0 and string-length(@text)<20] 遍历可点击的指定长度范围内的文本菜单
```

4.9 webview 切换

自动向下滚动, 代码在 \appium 自动化测试\appiumtest - findToast 中的 rolldown()

```
public void testWebView() throws InterruptedException { //ApiDemos
    driver.findElementByAccessibilityId("Views").click();
    WebElement list = driver.findElement(By.id("android:id/list")); //not AndroidElement
    MobileElement radioGroup = list.findElement(MobileBy
        .AndroidUIAutomator("new UiScrollable(new UiSelector()).scrollIntoView("
            + "new UiSelector().text(\"WebView\");"));
}
```

完善后的自动向下滚动函数如下, 代码在 \appium 自动化测试\appiumtest - findToast 中的 testWebView()

```
public void testWebView() throws InterruptedException { //对应 ApiDemos 的 app 对应 setUpForWebView()
    driver2.findElementByAccessibilityId("Views").click();
    Thread.sleep(1000);
    AndroidElement list = driver2.findElement(By.id("android:id/list"));
    MobileElement webview = list
        .findElement(MobileBy
            .AndroidUIAutomator("new UiScrollable(new UiSelector()).scrollIntoView("
                + "new UiSelector().text(\"WebView\");"));
    webview.click();
    System.out.println(driver2.getContextHandles());
    for(AndroidElement e: driver2.findElementsByXPath("//*[@*")){
        System.out.println(e.getTagName());
        System.out.println(e.getText());
    }
}
```



```
adb shell pm list packages |grep webview
adb shell dumpsys package com.adnroid.webview
less /tmp/appium.log
```

问题 1：见图 4-3。首先 setUp() 中无 setCapability("automationName", "uiautomator2"); 接着在类中添加 private static AppiumDriver<AndroidElement> driver2; 最后在 setUp() 中改成 driver2 = new AppiumDriver<>(remoteUrl, desiredCapabilities);

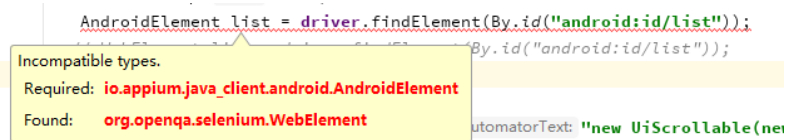


图 4-3 webview 遇到的问题图

问题 2：提示[Chromedriver][STDERR][1.123][DEBUG]: DevTools request failed 等错误。需要注意对应版本 <https://github.com/appium/appium/blob/master/docs/en/writing-running-appium/web/chromedriver.md>。

在打开 webview 时 adb shell cat /proc/net/unix | grep webview 可查看到进程
或 adb shell cat /proc/net/unix | grep --text _devtools_remote

通过 adb forward tcp:12184 localabstract:webview_devtools_remote_4366(上一步进程中) 映射到端口。

4.10 capability 高级讲解

主要参照 <https://github.com/appium/appium/blob/master/docs/en/writing-running-appium/caps.md> 和视频进阶第 11 阶。

包括 browserName、printPageSourceOnFindFailure、chromedriverExecutable、dontStopAppOnReset

在代码中设置 Appium 时：

```
capabilities.setCapability("noReset",true)
capabilities.setCapability("fullReset",false)
capabilities.setCapability("dontStopAppOnReset",true)
```

4.11 Grid 模式

参照：<https://github.com/appium/appium/blob/master/docs/en/advanced-concepts/grid.md>

其中 configuration 中的 ip 地址需要外部可访问

安装 wget [http:// selenium-release.storage.googleapis.com/3.9/selenium-server-standalone-3.9.1.jar](http://selenium-release.storage.googleapis.com/3.9/selenium-server-standalone-3.9.1.jar)

启动 java -jar selenium-server-standalone-3.9.1.jar -role hub
浏览器可访问 127.0.0.1:4444/grid/console

4.12 pageObject

TestPageObject 文件和 MainActivity.fxml 文件中添加

```
public void testPageObject() throws InterruptedException
{
    MainActivity main = new MainActivity();
    PageFactory.initElements(driver,main);
    Welcome welcome = new Welcome();
    PageFactory.initElements(driver,welcome);
    welcome.跳过欢迎页();
    main.enter 自选();
}
```

```
<element name="Views">
    <find>findElementById</find>
    <locator>Views</locator>
    <action>click</action>
</element>
```

MainAcitivy 类和 Welcome 类。

```
public class MainActivity {
    @FindBy(xpath = "//*[@text='蛋卷基金']")
    public WebElement 蛋卷基金;
    @FindBy(xpath = "//*[@text='首页']")
    public WebElement 首页;
    @FindBy(xpath = "//*[@text='动态']")
    public WebElement 动态;
    @FindBy(xpath = "//*[@text='自选']")
    WebElement 自选;
    public void enter 自选(){
        自选.click();
    }
}
```

```
public class Welcome {
    @FindBy(xpath = "//*[@text='登陆']")
    public WebElement 登陆;
    @FindBy(xpath = "//*[@text='跳过']")
    public WebElement 跳过;
    @FindBy(xpath = "//*[@text='下一步']")
    public WebElement 下一步;

    public void 跳过欢迎页() throws InterruptedException {
        Thread.sleep(1000);
        跳过.click();
    }
}
```

4.13 兼容性测试

包括尺寸、版本、定制 ROM、硬件。

4.13.1 测试网站与命令

阿里 <https://mqc.aliyun.com>、腾讯 <https://wetest.qq.com>，具体界面截图与测试结果见图 4-4。



图 4-4 腾讯 Wetest 截图

命令：

adb connect adb.wetest.qq.com:40470 → connected to adb.wetest.qq.com:40470

adb devices → List of devices attached adb.wetest.qq.com:40470 device

启动 appium 并运行测试代码后即可在远程自动测试。

4.13.2 真机管理平台

<https://openstf.io> STF Android (仅支持 Mac OS)

命令：

brew install rethinkdb graphicsmagick zeromq protobuf yasm pkg-config 安装依赖

npm install -g stf 安装

rethinkdb & 启动

stf local --public -ip 0.0.0.0

ps -ef |grep rethinkdb (查看已经启动的) pkill rethinkdb 和 rm -r rethinkdb_data/ 删掉

浏览器可访问 <http://localhost:7100> 并登陆

通过 ifconfig |grep 192.168 查询到 ip 地址可以外部连接真机。

4.13.3 远程调试平台演练

```
capabilities.setCapability("deviceName", System.getenv("deviceName"));
shell 中 mvn test -Dtest="TestPageObject(所在的类名)"
deviceName="xxx" mvn test -Dtest="TestPageObject" 连接真实设备
从而完成多个设备的测试，自动化调度。
```

4.13.4 常见问题

账号互踢、图形短信验证码、定位和复现等。

4.14 常用测试类

xUnit 测试用例，体系：JUnit、UnitTest、TestNG

测试用例 testcase（名字、过程：单元测试，UI 自动化测试 Appium Selenium，接口自动化测试 RestAssured 等、断言：JUnit Assert，hamcrest）、测试类 class、测试套件 suite

4.14.1 执行顺序

Default 取决于反射方法获得的列表，顺序固定

@FixMethodOrder(MethodSorters.JVM)顺序可能变化

@FixMethodOrder(MethodSorters.NAME_ASCENDING)按名字排序

4.14.2 具体类

带标签的测试类（基本的测试类、方法级别的标签、类级别的标签@Category({类名 class,类名.class})), 注释忽略等。

数据驱动：改造类，如下图。可在本地批量同时运行(模拟)设备。

```
@RunWith(Parameterized.class)
public class SampleTest {
    Parameterized.Parameters(name="{index}")
    public static List<String[]> data(){
        return Arrays.asList(new String[][]{ {"a","11.0"}, {"b","11.1"}, {"c","11.3"} });
    } //数据也可以来源于外部文件

    @Parameterized.Parameter
    public String deviceName;
    @Parameterized.Parameter(1)
    public String version;

    @Before
    public void setUp() throws MalformedURLException {
        .....
        capabilities.setCapability("deviceName", deviceName);
        capabilities.setCapability("platformVersion", version);
        .....
    }
}
```

测试套件：多个测试类时，可定义所有类的执行顺序等。添加 TestSuite 类，如下方左侧代码。

基于标签运行：代码中添加@Categories.IncludeCategory(类名.class)，则只运行带有@Category(类名.class)的类，如下方右侧代码。exclude 则为排除。

```
@RunWith(Suite.class)
@Suite.SuiteClasses( {
    SampleTest.class,
    SampleTest_PageObject.class
} )
public class TestSuite { }
```

```
@RunWith(Categories.class)
@Categories.IncludeCategory(类名.class)
@Suite.SuiteClasses( {
    SampleTest.class,
    SampleTest_PageObject.class
} )
public class TestSuite { }
```

4.14.3 执行顺序控制

class 与 method 的执行顺序：@BeforeClass setUpClass（配置 Capability，初始化 driver，安装 app）、@Before setUp（启动并进入特定界面）、@Test test1()、@After tearDown（关闭 app）、@Before setUp、@Test test2()、@After tearDown、@AfterClass tearDownClass（driver.quit）

按照功能模块分组，不同的产品和模块进行分包，不同包下要有一个运行套件，考虑尽可能并行

4.14.4 mvn 执行测试用例

```
mvn -Dtest=TestCircle test
mvn -Dtest=TestSquare,TestCircle test
mvn -Dsurefire.returnFailingTestsCount=2 test
mvn 调试测试用例
mvn -Dmaven.surefire.debug tset 等
mvn 测试结果（mvn test 默认所有测试用例的 class 以 Test 开头）
mvn site
mvn surefire-report:report
```

5. Jenkins

5.1 Jenkins 基础

持续集成：快速发现错误、防止分支大幅偏离主干

持续集成(部署)流程-CI，提交到代码仓库。持续集成服务器：检测代码变动、自动构建编译测试打包。

持续集成 CI、持续交付 CD、持续部署 CD，包括 Develop、Build、Test、Release、Deploy to production。CI 1~3，CD 1~4，CD 1~5。

Jenkins 持续集成管理工具。跨平台。https://en.wikipedia.org/wiki/Comparison_of_continuous_integration_software

5.2 Jenkins 部署和安装方法

Jenkins 界面首页：Configure System、Configure Global Security、Manage Plugins、Manage Nodes 等。

例子 1：新建 Jenkins Job。 New Item - Freestyle project - 输入 Project name (example_job1) 后可进入 configure 界面。General 设置、Build Triggers 设置（可 remotely 通过 token 在浏览器远程触发，stable、unstable、fails，Build periodically 和 Poll SCM(有变动后才运行)）、Build Environment 设置等。进入 Project 后，手工触发编译：Build with Parameters 并点 Build 按钮。回到 Jenkins 首页可以看到 Project 列表。（可参考第 1 讲，具体的参数配置）

例子 2：example_job2。Freestyle project、勾选 Build After other projects are built、Projects to watch 填 example_job1 并选 Trigger only if build is stable，Build 中 Execute shell 的 Command 填 echo "this is job2"。并 Apply。现象 1：可在 example_job1 中看到 Downstream Projects 中有 example_job2；example_job2 中看到 Upstream Projects 中有 example_job1。

example_job2 的 Build 中 Execute shell 的 Command 中添加 echo "parameter is", \$tester_home，并在 example_job1 中选择 Build with Parameters，参数仍填 hello 并点 Build 按钮。在 example_job2 中的 Console Output 内看不到 example_job1 传递的参数（Console Output 在 Build History 中点击#序号后打开）。

取消勾选 example_job2 的勾选 Build After other projects are built。在 example_job1 的 Post-build Actions 中的 Projects to build 填写 example_job2 并选 Trigger only if build is stable。并 Apply。同样会有现象 1。

5.3 Jenkins 邮件及节点配置

邮件服务器的配置：Jenkins 首页选择 configure，在 Extended E-mail Notification 中填写 SMTP server 并勾选 Use SMTP Authentication，填写用户名和密码后勾选 Use SSL，填写 SMTP port 和默认收件人（们）Default Recipients (\$DEFAULT_RECIPIENTS) 等信息。Default Triggers 中为发邮件的情况，如 Failure - Any。

在 E-mail Notification 中再填写一遍，Default Recipients 此处为 Reply-To Address，勾选 Test configuration by sending test e-mail 并填写 Test e-mail recipient。上方 Jenkins Location 的 System Admin e-mail address 内同时填写用户名。点击 Test configuration 按钮检测，“默认收件人邮箱”即可以收到一封来自“用户名邮箱”的测试邮件。

在 example_job1 中的 Add post-build action 选择 Editable E-mail Notification，Project Recipient List 默认即为 \$DEFAULT_RECIPIENTS，其他类似。Advance Settings 可以看到选择的 Failure - Any，也可以添加在当前作用域范围内的 Always 选项（在 e

example_job2 中的 Add post-build action 选择 Editable E-mail Notification 后看不到新增的选项)等。并 Apply。选择 Build with Parameters 填写参数并点 Build 按钮,可在 Console Output 可以看到不同步骤是否有 Checking if email needs to be generated。“默认收件人邮箱”即可以收到一封邮件。

可以观察到邮件标题和内容是按照 Jenkins 首页的 configure 内 Default Subject 和 Default Content 的格式发送的。

Extended E-mail Notification 可能需要在 Plugin Manager 里面添加。还可以添加 Locale 插件选择其他语言。

5.4 分布式架构

Master 和 Slave 架构, Console Output 中可以看到 Building on master in workspace...等语句。

Jenkins 首页选择 Manage Nodes 后点击 New Node, 输入名字如 mac, #of executors 代表最多同时运行的 build 的数目, 可选 1。Remote root directory 任务下放目录, 可先创建好后并填写, 如/A/B。Labels 选定是在哪里运行, 如 mac。Usage 选择 Only build..., 从而确保环境匹配。Launch method 可选 via Java Web Start, Node Properties 可添加环境变量等。点击名字后下载 slave.jar 并复制该行命令后在 shell 运行, 即可以看到 Agent discovery successful。

在首页 configure Security 内 Agent protocols 一般会选上 Java Web Start Agent Protocol/1/2/4。

可在 master 机器 shell 中查看 slave.log 文件查看错误信息, 如果查询不到说明网络连接或者端口有问题。

创建成功后可点进 example_job2 的 configure 可以看到 General 下多了一个选项 Restrict where this project can be run, 勾选后在 Label Expression 可以填写 Labels 的内容如 mac, 即会在其上运行(可删掉 Command 中的 echo "parameter is", \$tester_home)。此时在 example_job1 中点击 Build with Parameters 并 Build, 可以观察到 Build History 运行, 结束后点进 Downstream Projects 的 example_job2 其内的 Build History 也开始运行(结束前会显示 Pending), 在 example_job2 的 Console Output 中可以看到 Building remotely on mac(mac) in workspace /A/B/workspace/example_job2 等语句。观察到 example_job1 的 console 显示其 Building on master in workspace 等语句, 目录不包含/A/B 路径。

5.5 分布式架构

例子 3: 把 svn 的代码能够在 master 运行并收到结果通知。新建 newJob 名为 example_svn, Restrict where this project can be run 的 Label Expression 填写 mac, So

Source Code Management 选择 Subversion, Repository URL 填写 SVN(?)地址, Credentials 点击 Add 并添加内容 (SVN 的用户名和密码) (第三讲 05:50), Repository depth 一般选 infinity, Build 中 Execute shell 的 Command 输入 `pwd` 回车 `python ./C/D/test.py` (什么目录?), 添加 post-build action 为 E-mail Notification。(已在本地的目录下有了代码) 在 Console Output 中等待, 并看到 `checking out...` 等内容和文件列表, 同时运行了其中的 .py 文件, 并发送了邮件。

修改目录下的代码为错的 (不是编译错误, 是 `self.assertEqual('a'.upper(), 'B')`), 并在 shell 中输入 `svn commit -m "incorrect test.py" test.py` 提交代码。example_svn 开始 build, 查看 Console Output, 会看到提示 FAILED (failures=1), 同时有邮件发送 (Jenkins 邮件及节点配置勾选了 Always)。

example_svn 的 configure 中 Build Triggers 勾选 Build periodically 并在 Schedule 内填写 `H/2 * * * *` (每隔两分钟), 可以看到 Build History 中每两分钟会出现一次 Build。

例子 4: github 集成。新建 newJob 名为 example_github, 勾选 GitHub project, url 填写项目地址。勾选 Restrict where this project can be run 并在 Label Expression 填写 mac, Source Code Management 下勾选 Git, Repositories URL 填写以 .git 结尾的仓库地址, Branches to build 为 `*/master`, Build 选择 Execute shell 并在 Command 中填写 `pwd` 回车 `echo "git remote url is" + $GIT_URL` 回车 `echo "build num"+ $BUILD_NUMBER` 回车 `python test/test.py`, 可以通过点击 the list of... 查看详细说明。并 Apply。点击 Build History 并进入 Console Output, 可以看到相应的结果。

附: 使用 github, 复制 .git 网址, `git clone https://.../xx.git` 下载到本地。可查看 git log, 可通过 `vim test.py` 修改下载到的代码。输入 `git add test.py`, `git commit test.py`, 输入描述信息。最终输入 `git push origin master` 即可上传提交代码。

example_github 的 configure 中 Build Triggers 勾选 Poll SCM 并在 Schedule 内填写 `TZ=Asia/Beijing` 回车 `H/Z * * * *` 从而修改时区并每两分钟 Build 一次, 然而并没有每两分钟 Build 一次。

5.6 集成 Maven TestNG

例子: Maven 集成。略。

pluginManager 中安装 HTML Publisher, 修改 example_maven 的 Add post-build action 为 Publish HTML reports, 在 HTML directory to archive 中填写路径, Index page[s] 中填写路径下的 emailable-report.html。可以直接在 HTML Report 中查看相应的结果。

例子 5: appium 集成。新建 newJob 名为 example_appium, 勾选 Restrict where t

his project can be run 并在 Label Expression 填写 mac, Build 选择 Execute shell 并在 Command 中填写 `cd /Users/.../appium_sample/sampe-code/examples/python;python and roid_contacts.py`。并 Apply。点击 Build Now。

集成成功和脚本成功。

Command 中填写 `cd /Users/.../appium_sample/sampe-code/examples/python;python android_contacts.py >result.txt 2>&1; ./result_analysis.sh`。真正检测成功或失败。

```
python android_contacts.py > result.txt 2>&1
gerp -i failure result.txt
if [ $? == 0 ]; then
    echo "failure is found"
    exit 1
else
    exit 0
fi
```

5.7 DevOps 持续交付和 pipeline 机制

一系列 Jenkins 插件将整个持续交付流程用解释性代码 Jenkinsfile 来描述，如图 5-1 所示。

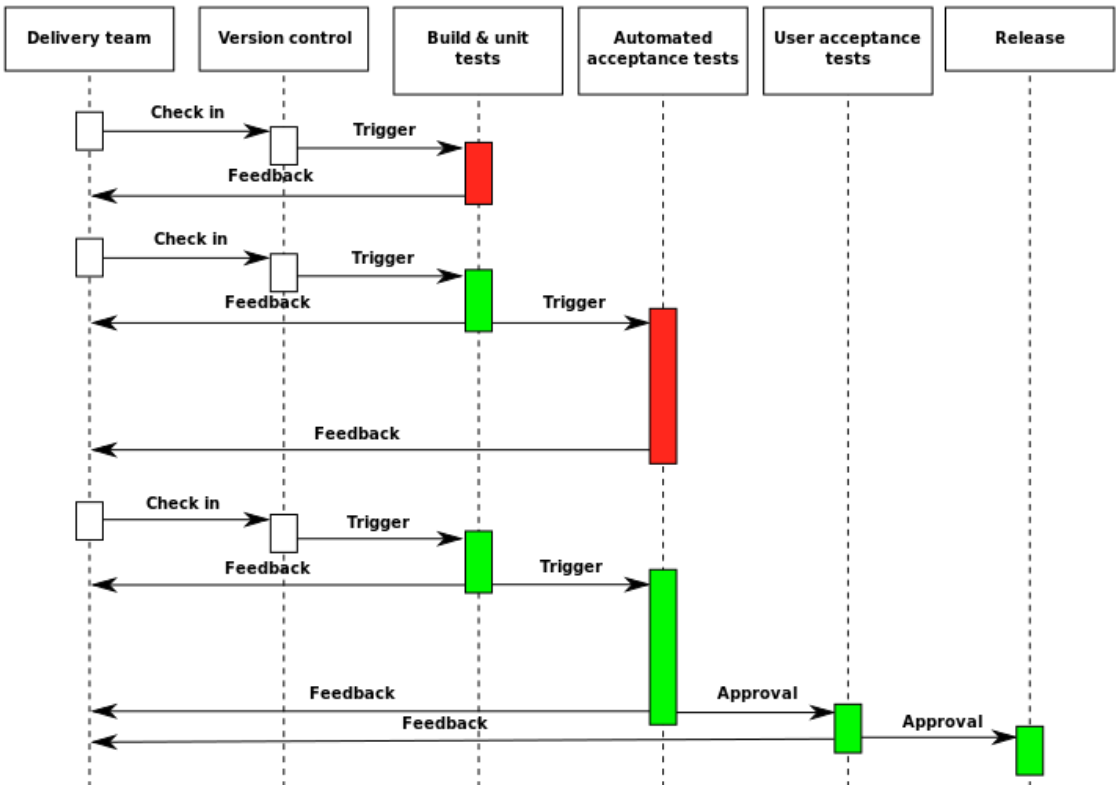


图 5-1 持续交付图

例子 6: Pipeline。新建 newJob 名为 example_pepipeline，下方勾选 Pipeline。Pipeline 的 Definition 选择 Pipeline script, Script 选择 try sample Pipeline，需提前在 configureTools 中添加 Name 为 M3 的 Maven，版本可为 3.5.2。并 Apply。点击 Build No

w。可以看到 Stage View，其中可以查看每个阶段（Preparation、Build、Results）的 Logs 信息，在 Console Output 中可以看到三个阶段合起来的信息。

Declarative Pipeline Syntax（推荐）、Scripted Pipeline Syntax

```
pipeline{
  agent any
  environment {
    paral = "abc"
  }
  stages{
    stage('Preparation'){
      //get code from GitHub
      steps{
        sh "echo git 'https://xx.git'"
      }
    }
    stage('Build'){
      steps{
        sh "echo build"
        sh "echo '${paral}'"
      }
    }
    stage('test'){
      steps{
        sh "echo test"
      }
    }
  }
}
```

```
pipeline{
  agent any
  environment {
    mvnHome = tool 'M3'
  }
  stages{
    stage('Preparation'){
      steps{
        git 'https://xx.git'
      }
    }
    stage('Build'){
      steps{
        script{
          if (isUnix()) {
            sh "'${mvnHome}/bin/mvn' -Dmaven.test.failure.ignore clean package"
          }
          else {
            bat("/"${mvnHome}\bin\mbn" -Dmaven.test.failure.ignore clean package/)
          }
        }
      }
    }
    stage('Results'){
      steps{
        junit '**/target/surefire-reports/TEST-*.xml'
        archive 'target/*.jar'
      }
    }
  }
}
```

将上方右代码拷贝到 example_pipeline 的 Pipeline 下的 Script 栏中，其中 Definition 为 Pipeline script。并 Apply。点击 Build Now。

例子 7: Selenium sample use declarative pipeline。新建 newJob 名为 example_selenium_pipeline，下方勾选 Pipeline。将上图右修改后的代码拷贝到 example_pipeline 的 Pipeline 下的 Script 栏中，其中 Definition 为 Pipeline script。其中代码修改 agent { label 'mac' } 和 mvnHome = tool 'maven' 两处，并修改 git 后地址。build 中的 script 内修改为 sh "cd ./SeleniumTest; '\${mvnHome}/bin/mvn' -Dmaven.test.failure.ignore clean package"。并 Apply。点击 Build Now。可以在 Stage View 中看到 Preparation 的 Logs 中看到 git 的进度；Results 的 Logs 中看到 Archive Junit-formatted test results --**/target/...../TEST-*.xml --(self time 1s) 和 Archive artifacts --target/*.jar --(self time 88ms) 等内容。点进 Build History 中的序号，可以看到 Test Result，依次点击 All Tests 下的 Package、Class、Test name 可看到 Passed 等信息。

例子 8: Pipeline 自定义模块之自定义测试报告。 点击 example_selenium_pipeline 的 Configure，在上图右修改后的代码中 stage('Results')内 steps 段落下方添加 post { always { } }，点击 Pipeline Syntax，在 Sample Step 选择 publishHTML: Publish HTML reports，在 HTML directory to archive 中填写 Selenium Test/target/surefire-reports，Index page[s]中填写 emailable-report.html，Report title 可填写 HTML_Selenium Report，点击 Generate Pipeline Script，将生成的代码（第 7 讲 16:47）添加到 always { }中。并 Apply。点击 Build Now。左侧会出现 HTML-Selenium Report，点击可以看到报告。点进 Build History 中的序号，可以看到本次相对于例子 7 内多了 Test Result。

例子 9: Pipeline 自定义模块之发送邮件。 点击 example_selenium_pipeline 的 Configure，点击 Pipeline Syntax，在 Sample Step 选择 emailxext: Extended Email，填写 To、Subject 和 Body 等信息，如 Body 填写\$DEFAULT_CONTENT，点击 Generate Pipeline Script，将生成的代码（第 7 讲 22:32）添加到例子 8 中生成代码的下一行，（仍在 always { }内）。并 Apply。点击 Build Now。可以收到邮件。

例子 10: paraller samples。 新建 newJob 名为 example_parallel_pipeline，下方勾选 Pipeline。将下图内代码拷贝到 Pipeline 下的 Script 栏中，其中 Definition 为 Pipeline script。并 Apply。点击 Build Now。可以在 Stage View 看到各个 Stage 运行所需的时间。

<pre> pipeline { agent none stages { stage('Preparation') { parallel { stage('First Preparation') { agent { node { label 'mac' } } steps { git 'https://github.com/xxx/Code.git' } } stage('Second Preparation') { agent any steps { sh 'pwd' sh 'echo preparation2' } } } } } } </pre>	<pre> stage('build') { agent any steps { sh 'pwd' echo 'abc' } } stage('Test') { parallel { stage('node 1') { agent any steps { sh 'pwd' sh 'sleep 20s' sh 'echo hstream1' } } stage('node 2') { agent { label 'mac' } steps { sh 'pwd' sh 'sleep 20s' sh 'echo hello2' sh 'python ./test/test.py' } } } } </pre>
---	---

代码：整个的 agent 是 none，stage Preparation 内 parallel 中的 First Preparation 和 Second Preparation 是并行。stage Preparation 和 stage build 是串行，stage Test 内 par

allel 中的 node 1 和 node 2 是并行。

点击左侧 Open Blue Ocean，并点进 MESSAGE 为 Stared by user XXX，看到具体的并行串行的图示，如图 5-2。点击绿色的勾可以看到具体的指令和 agent 等内容。

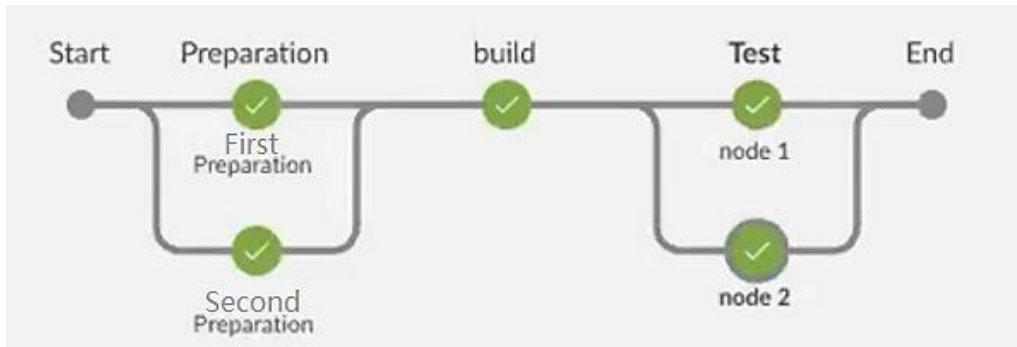


图 5-2 Blue Ocean 图示

例子 11: SCM Jenkinsfile example。选择 example_parallel_pipeline，其中 Pipeline 的 Definition 为 Pipeline script form SCM，代表着 Jenkinsfile 可以放在源代码管理仓库，跟开发者代码放在一起。SCM 可选择 Git，Repositor URL 填写.git 结尾的路径地址，Branches to build 一般为*/master。并 Apply。点击 Build Now。可以看到在 Open Blue Ocean 界面中每个 Step 下多了一条 General SCM。

5.8 Blue Ocean

Plugin Manager 中安装 Blue Ocean 和 Blue Ocean Pipeline Editor。可以控制已有 Pipeline，也可以创建新的 Pipeline。

在 Jenkins 中点击左侧 Open Blue Ocean，进入 Pipelines 界面点击右侧 New Pipeline，Where do you store your code 可选的 GitHub，并选择账户后选择 repositories 并点击 Creat Pipeline，输入 Naming conflict 后创建 Pipeline。

在某一 Pipeline 界面中，点击右上侧铅笔图标，即可以在图形化界面中添加节点，在添加时点击已有节点下方的+，右侧输入节点名字。并点击 Add step 按钮，可选择 Shell Script 选项，可输入 pwd 并点击 save。输入 Description 后并选择 Commit to，点击 Save & run。但是添加时可能没有错误检测，如没添加 agent 会导致节点运行错误。

5.9 Jenkins 总结

Jenkins 运行失败的原因：Jenkins 的 master 环境不能满足所有测试环境，集成到 Jenkins 等 SSH 的环境要一致。

排查：确保脱离编译器时在 Shell 能运行、Master 连接 Node 后的系统环境和本地 Node 运行环境是否一样，通过 export 查看。

本章主要学习了持续集成测试流程和 Jenkins 介绍、Jenkins 在 Ubuntu/Centos/Doc

ker 系统上的搭建部署、使用 Jenkins 来创建任务和配置、创建和部署 slaves 和 nodes、SVN 的集成和任务管理配置、Git/Github 的集成和任务管理配置、Jenkins 实例：Maven & Jenkins with Selenium Jenkins 2.0、DevOps 和持续交付、Jenkins 2.0 Pipeline 机制、Pipeline 定义和使用和 Blue Ocean 的使用。

6 iOS 测试

iOS 平台的封闭性。

主流移动测试框架。

Appium、Calabash-iOS、KIF、XCTest（苹果）、WebDriverAgent、Uiautomation。证书体系。

6.1 启动方法

点击左侧气泡图标，UICatalog（App 名字）中 Build 成功后，找到绿色箭头 Sign UICatalog.app 下 Signing Identity 内.app 的路径地址。

打开 Appium Desktop，输入 platformName 为 ios、deviceName 为 iPhone 8、platformVersion 为 11.2、app 为上一步中的路径地址、automationName 为 xcuitest。启动模拟器。尽量新建一个端口。

6.2 使用 Inspector 定位

安装依赖，运行 WebDriverAgent 通过输入 ./Scripts/bootstrap.sh。参照 <https://testerrhome.com/topics/9666>

使用 Inspector 主要参照 <https://github.com/facebook/WebDriverAgent/wiki/Using-the-Inspector>

设置 dev tools 和 deps:

DevToolsSecurity --enable

carthage bootstrap --platform ios

Inspector GUI 图形化: <http://localhost:8100/inspector>

JSON elements tree: <http://localhost:8100/source>

构建方法: `xcodebuild -project WebDriverAgent.xcodeproj \`
`-scheme WebDriverAgentRunner \`
`-destination 'platform=iOS Simulator,name=iPhone 8' \`
`test`

在 Inspector 中元素一般为<XCUIElementTypeCell>、<XCUIElementTypeStaticText>等。

如某个<XCUIElementTypeButton>的 value、name、label 都为 Buttons，而某个<X

CUIElementTypeTextField>的 value 是 Placeholder text，即其输入框内部默认值，label 为 1 为空，没 name 属性。

6.3 iOS 用例演练

以 Java 代码形式完成 appium 的初始化操作，并完成简单的用例，如图 6-1。

```
import org.junit.Test;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.openqa.selenium.remote.RemoteWebDriver;

import java.net.MalformedURLException;
import java.net.URL;

public class TestIOS {
    @Test
    public void UICatalogTest() throws MalformedURLException {
        DesiredCapabilities caps=new DesiredCapabilities();
        caps.setCapability("platformName", "ios");
        caps.setCapability("platformVersion", "11.2");
        caps.setCapability("deviceName", "iPhone 8");
        caps.setCapability("app",
            "/Users/seveniruby/Library/Developer/Xcode/DerivedData/UICatalog-ftyzdbgapijmxobezrnrxssh
            "/Products/Debug-iphonesimulator/UICatalog.app");

        RemoteWebDriver driver=new RemoteWebDriver(new URL( spec: "http://127.0.0.1:4723/wd/hub"), caps);
        System.out.println(driver.getPageSource());
        for(WebElement e: driver.findElementsByXPath( using: "//*")){
            System.out.println(e.getText());
            System.out.println(e.getTagName());
        }
    }

    @Test
    public void UICatalogTestClick() throws MalformedURLException {
        DesiredCapabilities caps=new DesiredCapabilities();
        caps.setCapability( capabilityName: "platformName", value: "ios");
        caps.setCapability( capabilityName: "platformVersion", value: "11.2");
        caps.setCapability( capabilityName: "deviceName", value: "iPhone X");
        caps.setCapability( capabilityName: "app",
            value: "/Users/seveniruby/Library/Developer/Xcode/DerivedData/UICatalog-ftyzdbgapijmxobezrn
            "/Products/Debug-iphonesimulator/UICatalog.app");

        AppiumDriver driver=new AppiumDriver(new URL( spec: "http://127.0.0.1:4723/wd/hub"), caps);
        driver.manage().timeouts().implicitlyWait( time: 10, TimeUnit.SECONDS);
        System.out.println(driver.getPageSource());
        driver.findElementByAccessibilityId( using: "Buttons").click();
        System.out.println(driver.findElementsByXPath( using: "//*[contains(@type, 'Button')]"));
    }
}
```

图 6-1 iOS 测试用例图

7.接口测试

7.1 Charles

模拟器运行 app 并使用 Charles 抓包。Sequence 中可以看到 GET 和 POST 请求，下方 Contents 中有各项具体的 json 内容以及有关参数。

可设置 Breakpoints 抓取接口，并进一步设置断言。点击 Execute 后可继续。

可使用 Rewrite 重写接口，可改写 host、参数等数据内容。可以以此查看字符串的展示与长度的测试等。

需提前配置 SSL Proxying Settings 的*: 443 的 Location 端口。

弱网测试：通过 Throttling Settings 来模拟，如设置修改 Bandwidth 的 Upload 和

Download。

7.2 Postman

可以发送 POST 请求，填写接口链接、Headers 和 Body 后点击 Send 即可以发送。

7.3 JSON 字符串的处理

net.sf.json-lib、com.jayway.jsonpath 包等。

json object 和 JsonPath

```
Object object = JsonPath.read(json, "$.topics[*].user.login");
```

rest 相关包：io.rest-assured 的 rest-assured、json-path、jason-schema-validator

```
post("http://localhost:8080/test/function");
```

```
Response response = get("http://localhost:8080/test/function?name=1&id=2");
```

```
System.out.println(response.asString());
```

```
System.out.println(response.getStatusCode());
```

还可以 givet().param("name",1).get("http://localhost:8080/test/function").prettyPeek();

或 post

host 虚拟包：io.leopard 的 javahost

8.Spring

本章参考 <https://www.cnblogs.com/wmyskxz/p/8820371.html>

Spring 是一个轻量级的控制反转（IoC）和面向切面（AOP）的容器框架。

Spring 的框架结构。

8.1 SpringIoC 概述

IoC: Inverse of Control（控制反转）。

设计思想，就是将原本在程序中手动创建对象的控制权，交由 Spring 框架来管理。

反控：若要使用某个对象，只需要从 Spring 容器中获取需要使用的对象，不关心对象的创建过程，也就是把创建对象的控制权反转给了 Spring 框架。

8.1.1 Spring IoC 容器

BeanFactory。ApplicationContext 及其常见实现类、和 BeanFactory 区别。

8.1.2 Spring IoC 的容器的初始化和依赖注入、IoC 是如何实现的

Spring 创建对象的过程中，将对象依赖属性（简单值，集合，对象）通过配置设值给该对象。

8.2 装配 Spring Bean

方式选择的原则。

通过 XML 配置装配 Bean（装配简易值、装配集合、命名空间装配、引入其他配置文件）、通过注解装配 Bean、使用 `@Component` 装配 Bean、自动装配——`@Autowired`（自动装配的歧义性（`@Primary` 和 `@Qualifier`）、使用 `@Bean` 装配 Bean、Bean 的作用域、Spring 表达式语言简要说明）

8.3 Spring AOP 简介

AOP 即 Aspect Oriented Program 面向切面编程。

AOP 的目的、概念。

例子与代码。

使用注解来开发 Spring AOP（选择连接点、第二步：创建切面、第三步：定义切点、第四步：测试 AOP。环绕通知）、使用 XML 配置开发 Spring AOP。

8.4 Spring 和数据库编程

传统 JDBC 回顾与优化（创建 `DBUtil` 类、使用 `try-catch` 语句自动关闭资源、进一步改进 `DBUtil` 类）

Spring 中的 JDBC（配置数据库资源、使用简单数据库配置、使用第三方数据库连接池、`Jdbc Template`）

8.5 MVC 设计概述

Spring MVC 架构。

在 IDEA 中新建 Spring MVC 项目、修改 `web.xml`、编辑 `dispatcher-servlet.xml`、编写 `HelloController`、准备 `index.jsp`、部署 Tomcat 及相关环境和重启服务器。

8.5.1 跟踪 Spring MVC 的请求

`DispatcherServlet`、处理器映射（`HandlerMapping`）、控制器、返回 `DispatcherServlet`、视图解析器和视图。

8.5.2 使用注解配置 Spring MVC

为 `HelloController` 添加注解、取消之前的 XML 注释、重启服务器。`@RequestMapping` 注解细节

8.5.3 配置视图解析器

解决方案。修改 `HelloController`、配置视图解析器、剪贴 `index.jsp` 文件、和更新资源重启服务器。

8.5.4 控制器接收请求数据

使用 Servlet 原生 API 实现。使用同名匹配规则。使用模型传参。中文乱码问题。

8.5.5 控制器回显数据

使用 Servlet 原生 API 来实现、使用 Spring MVC 所提供的 ModelAndView 对象、使用 Model 对象。

8.5.6 客户端跳转和文件上传

配置上传解析器、编写 JSP、编写控制器和测试。

9. SpringBoot

SpringboottestApplication 成功运行，见图 9-1。

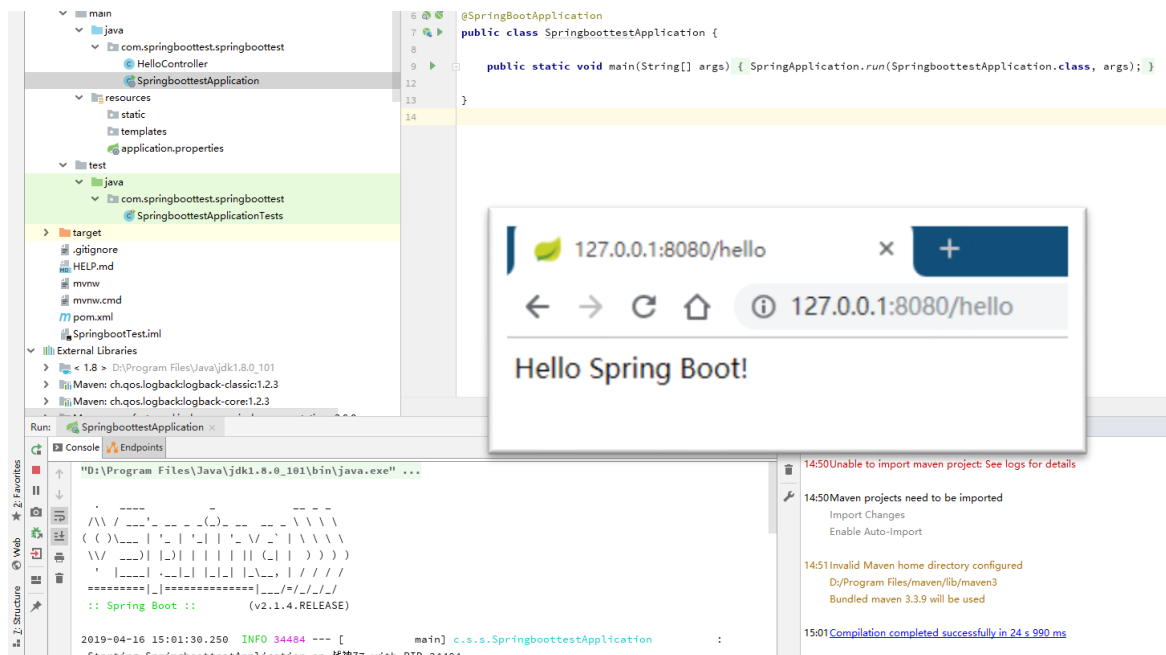


图 9-1 SpringBoot 运行图

9.1 简单配置

用 @Value 获取配置属性，见图 9-2。

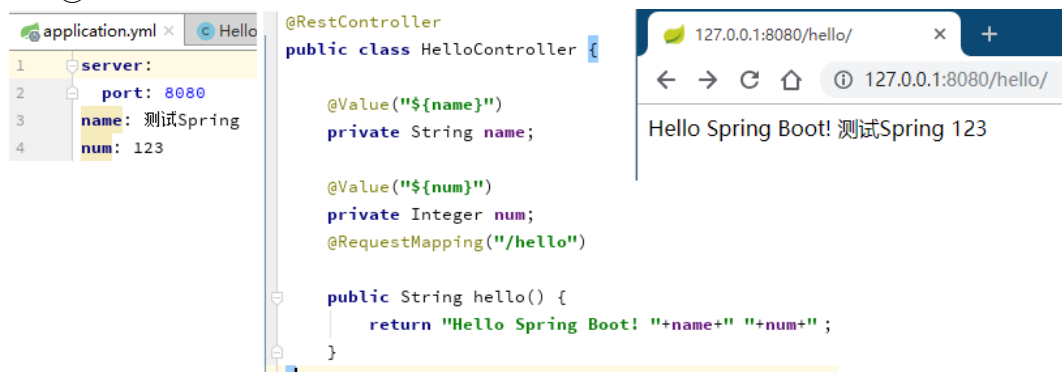


图 9-2 SpringBoot 配置图

简单修改内容。在配置文件中使用当前配置，见图 9-3。

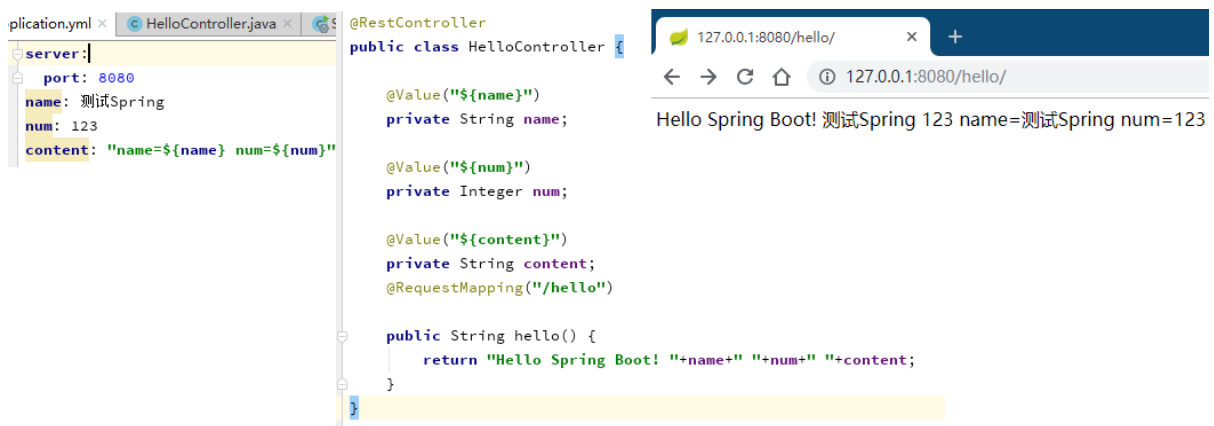


图 9-3 SpringBoot 进一步配置图

9.2 封装错误解决

错误 1: yml 文件中删掉了 content 而 HelloController 中忘记删掉相关的@Value。已解决。

错误 2: 使用@ConfigurationProperties 注解时 idea 提示 Spring Boot Annotation processor not found in classpath。

错误 2 尝试 1: 加入@PropertySource("classpath:application.yml")后仍然提示

错误 2 尝试 2: 加入<artifactId>spring-boot-configuration-processor</artifactId>依赖，提示会变成 Re-run Spring Boot Configuration Annotation Processor to update generated metadata。两个尝试参照 <https://www.imooc.com/qadetail/208912>

错误 3: Spring 无法运行

错误 3 原因 1: 发现 HelloController 类中的@Value("\${name}")要改成\${student.name}

错误 3 原因 2: Cannot resolve configuration property, 名字起成了 Info, 有多个 InfoProperties。改成了 Student。

错误 3 原因 3: Could not autowire. No beans of 'StudentProperties' type found。暂时不用管。参照 https://blog.csdn.net/qq_40147863/article/details/86103857

错误 3 原因 4: Failed to bind properties under 'student' to com.springboottest.springboottest.StudentProperties。....。Action: Update your application's configuration。

错误 3 解决: 在 yml 文件 name 前添加包名前缀。参照 <https://bbs.csdn.net/topics/392346212>。后来再次尝试发现不添加也可运行了。

错误 4: studentp.getName()等在浏览器中显示为 null。

错误 4 尝试 1: yml 文件中 name 改成 springboottest.name 或者 StudentProperties.name 或 com.name 或 student.name 不行。

错误 4 尝试 2: StudentProperties 类改成@ConfigurationProperties(prefix ="springboottest.student"), 不行。

错误 4 尝试 3: SpringboottestApplication 类上添加@PropertySource("classpath:application.yml"), 不行。

配置情况如图 9-4 所示。

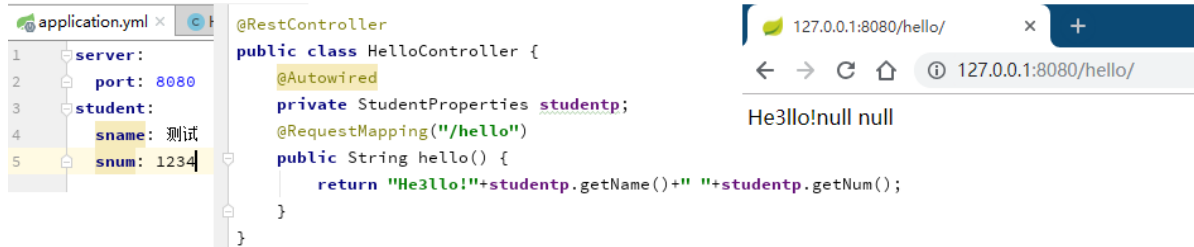


图 9-4 SpringBoot 进一步配置图

9.3 热部署

引入 spring-boot-devtools 依赖。

IDEA 的 Setting 中选中 Build project automatically

Shift+Ctrl+Alt+/后选择"Registry"勾选"compiler.automake.allow.when.app.running"

9.4 Spring Boot 使用

Spring Boot 支持 JSP (添加依赖)

错误: 把 addAttribute 写成了 addAllAttributes

集成 MyBatis。