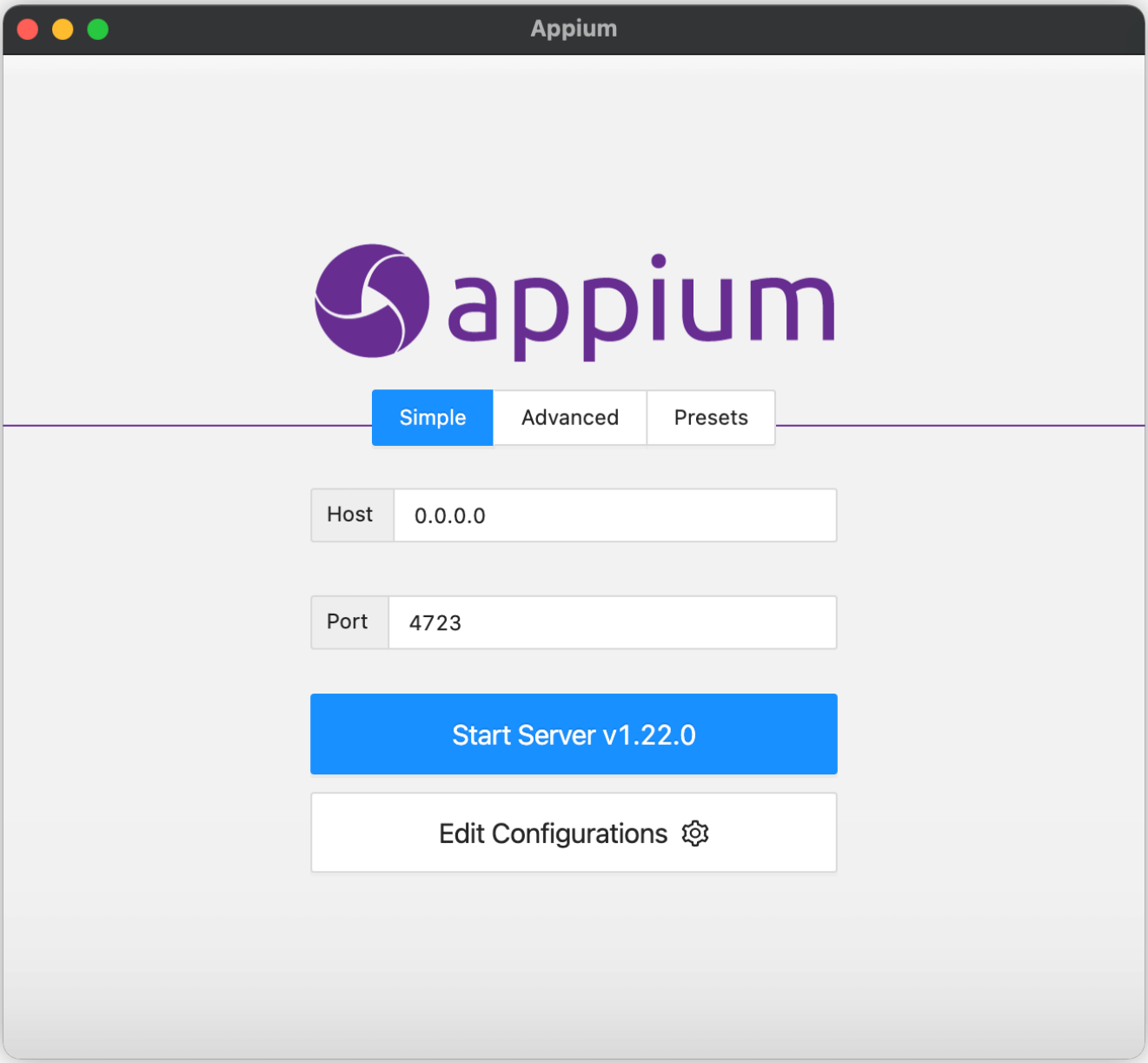
导读：
该篇文章主要记录基于 `Appium` 的 `自动化测试` 开发实践。



Appium 简介

`Appium`（官网 - http://appium.io/） 是一个 `移动端的自动化框架`，可用于测试 `原生应用`、`移动网页应用` 和 `混合型应用`，且是 `跨平台` 的。可用于 `iOS` 和 `Android` 操作系统。

原生的应用是指用 Android 或 iOS 的 SDK 编写的应用，移动网页应用是指网页应用，类似于 iOS 中 Safari 应用或者 Chrome 应用或者类浏览器的应用。混合应用是指一种包裹 Webview 的应用，原生应用于网页内容交互性的应用。

重要的是 Appium 是跨平台的，何为跨平台，意思就是可以针对不同的平台用一套 API 来编写测试脚本。

> Appium 官方简介

Appium 环境配置

使用 `Appium` 和 `python` 来进行自动化测试，需要安装两个东西：一个是 `Appium 的客户端`，一个是 `Appium-python库`。

这两个需要安装的东西在加上手机就可以进行自动化测试，它们之间的关系是：**`python代码 -> Appium-python库 -> Appium -> 手机`**。

0、所需环境

- `MAC OS 系统`（MacOS Big Sur 11.5.2）或 `Windows 系统`
- `Xcode`（Xcode_v13.0(13A233)）和 `Android Studio`（v4.0）
- `待测试的 iOS 项目或安卓项目`
- `Appium Desktop 客户端`（Appium_v1.22.0）
  - Github - Appium 下载地址（https://github.com/appium/appium-desktop/releases）
- `python 环境`（Python 2.7.16、python 3.8.9）
  - python 环境下载（https://www.python.org/ftp/python/）
- `Pycharm（Python 开发软件）`（Pycharm_v2021.2.2）
  - Pycharm 官方软件下载（https://www.jetbrains.com/pycharm/）
  - Pycharm 激活破解方法（https://www.macwk.com/article/jetbrains-crack）
- `node.js 环境`（node_v14.17.3）
  - Node 环境下载（https://npm.taobao.org/mirrors/node）
  - Node 环境下载（https://nodejs.org/dist/）
- `npm 环境`（npm_v6.14.13）
- `cnpm 环境`
  - ```
    sudo npm install -g cnpm --registry=https://registry.npm.taobao.org
    ```
- `ios-deploy 依赖库`
  - ```
    sudo cnpm install -g ios-deploy
    ```
- `homebrew 环境`
  - ```
    /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
    ```
- `libimobiledevice 依赖库`
  - ```
    brew install --HEAD libimobiledevice
    ```
- `Carthage 依赖库`
  - ```
    brew install carthage
    ```
- `WebDriverAgent`
  - Github - WebDriverAgent（https://github.com/facebook/WebDriverAgent）
    ```

1、安装 Xcode、Appium Desktop、Pycharm 客户端（以 MacOS 环境为例）

Xcode

- Xcode - App Store

Appium Desktop 客户端

- Appium 官网 - http://appium.io/
- Appium 官方简介
- Appium Github 下载地址（https://github.com/appium/appium-desktop/releases）

Pycharm - python 开发工具

- Pycharm 官方软件下载（https://www.jetbrains.com/pycharm/）
- Pycharm 激活破解方法（https://www.macwk.com/article/jetbrains-crack）

2、配置 Python、Node.js、npm、cnpm、ios-deploy 环境

Python 环境

- python 环境下载（https://www.python.org/ftp/python/）

```
# 获取 Python 版本号
python -v     # 获取 python2 环境版本号 Python 2.7.16 (default, Jun 18 2021, 03:23:53)
python3 -v    # 获取 python3 环境版本号 Python 3.8.9 (default, Aug 21 2021, 15:53:23)
```

Node.js 环境

- Node 环境下载（https://npm.taobao.org/mirrors/node）
- Node 环境下载（https://nodejs.org/dist/）

```
# 获取 node 版本号
node -v     # v14.17.3
```

npm

```
# 当前 npm 版本号 6.14.13
{21-10-25 15:10}[ruby-2.7.0]mxgx:~ mxgx% npm version
{
  npm: '6.14.13',
  ares: '1.17.1',
  brotli: '1.0.9',
  cldr: '39.0',
  icu: '69.1',
  llhttp: '2.1.3',
  modules: '83',
  napi: '8',
  nghttp2: '1.42.0',
```

```
  node: '14.17.3',
  openssl: '1.1.1k',
  tz: '2021a',
  unicode: '13.0',
  uv: '1.41.0',
  v8: '8.4.371.23-node.67',
  zlib: '1.2.11'
}
```

cnpm

> 说明：
>
> 1、之所以要安装 `cnpm` 是因为需要安装 `ios-deploy` 依赖库，其所依赖的安装环境就是 `npm` 或 `cnpm` 环境；
> 2、而 `cnpm` 和 `npm` 的其实并没有什么差别，只不过 `npm` 是 `外网环境`，对于国内的开发人员来讲并不是很友好，所以就有了 `cnpm` 的 `国内镜像环境`。

```
sudo npm install -g cnpm --registry=https://registry.npm.taobao.org
```

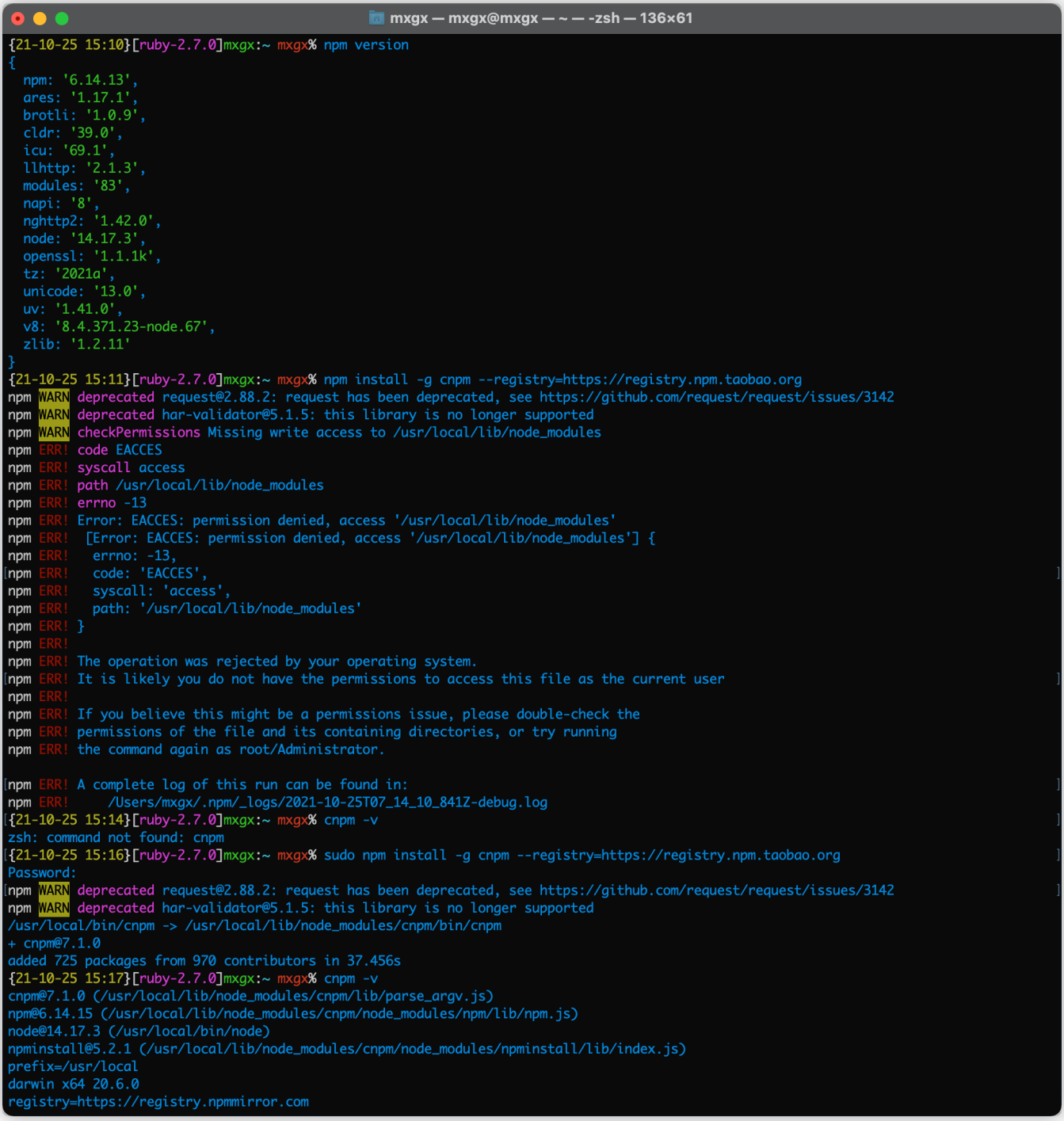
如果使用 `npm install -g cnpm --registry=https://registry.npm.taobao.org` 去执行指令安装的话，会报错没有权限：

```
{21-10-25 15:11}[ruby-2.7.0]mxgx:~ mxgx% npm install -g cnpm --
registry=https://registry.npm.taobao.org
npm WARN deprecated request@2.88.2: request has been deprecated, see
https://github.com/request/request/issues/3142
npm WARN deprecated har-validator@5.1.5: this library is no longer supported
npm WARN checkPermissions Missing write access to /usr/local/lib/node_modules
npm ERR! code EACCES
npm ERR! syscall access
npm ERR! path /usr/local/lib/node_modules
npm ERR! errno -13
npm ERR! Error: EACCES: permission denied, access '/usr/local/lib/node_modules'
npm ERR!  [Error: EACCES: permission denied, access '/usr/local/lib/node_modules'] {
npm ERR!   errno: -13,
npm ERR!   code: 'EACCES',
npm ERR!   syscall: 'access',
npm ERR!   path: '/usr/local/lib/node_modules'
npm ERR! }
npm ERR!
npm ERR! The operation was rejected by your operating system.
npm ERR! It is likely you do not have the permissions to access this file as the
current user
npm ERR!
npm ERR! If you believe this might be a permissions issue, please double-check the
npm ERR! permissions of the file and its containing directories, or try running
npm ERR! the command again as root/Administrator.
```

```
npm ERR! A complete log of this run can be found in:
npm ERR!     /Users/mxgx/.npm/_logs/2021-10-25T07_14_10_841Z-debug.log
```

正确的结果是：

```
{21-10-25 15:16}[ruby-2.7.0]mxgx:~ mxgx% sudo npm install -g cnpm --
registry=https://registry.npm.taobao.org
Password:
npm WARN deprecated request@2.88.2: request has been deprecated, see
https://github.com/request/request/issues/3142
npm WARN deprecated har-validator@5.1.5: this library is no longer supported
/usr/local/bin/cnpm -> /usr/local/lib/node_modules/cnpm/bin/cnpm
+ cnpm@7.1.0
added 725 packages from 970 contributors in 37.456s
{21-10-25 15:17}[ruby-2.7.0]mxgx:~ mxgx% cnpm -v
cnpm@7.1.0 (/usr/local/lib/node_modules/cnpm/lib/parse_argv.js)
npm@6.14.15 (/usr/local/lib/node_modules/cnpm/node_modules/npm/lib/npm.js)
node@14.17.3 (/usr/local/bin/node)
npminstall@5.2.1
(/usr/local/lib/node_modules/cnpm/node_modules/npminstall/lib/index.js)
prefix=/usr/local
darwin x64 20.6.0
registry=https://registry.npmmirror.com
```



ios-deploy 环境

```
sudo cnpm install -g ios-deploy
```

同安装 `cnpm` 类似，如果未使用管理员命令的话，执行 `cnpm install -g ios-deploy`，一样会报错：

```
{21-10-25 15:17}[ruby-2.7.0]mxgx:~ mxgx% cnpm install -g ios-deploy
Downloading ios-deploy to /usr/local/lib/node_modules/ios-deploy_tmp
Error: EACCES: permission denied, mkdir '/usr/local/lib/node_modules/ios-deploy_tmp'
npminstall version: 5.2.1
npminstall args: /usr/local/bin/node
/usr/local/lib/node_modules/cnpm/node_modules/npminstall/bin/install.js --fix-bug-
versions --china --userconfig=/Users/mxgx/.cnpmrc --
disturl=https://npmmirror.com/mirrors/node --registry=https://registry.npmmirror.com -g
ios-deploy
```

正确的执行结果：

```
{21-10-25 16:06}[ruby-2.7.0]mxgx:~ mxgx% sudo cnpm install -g ios-deploy
Password:
Downloading ios-deploy to /usr/local/lib/node_modules/ios-deploy_tmp
...
...
...
** BUILD SUCCEEDED **

All packages installed (used 17s(network 17s), speed 0B/s, json 0(0B), tarball 0B)
[ios-deploy@1.11.4] link /usr/local/bin/ios-deploy@ -> /usr/local/lib/node_modules/ios-
deploy/build/Release/ios-deploy
```

3、配置 homebrew、libimobiledevice、carthage 环境

homebrew

> Homebrew 官网（https://brew.sh/）

```
/usr/bin/ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

02 - 安装 libimobiledevice

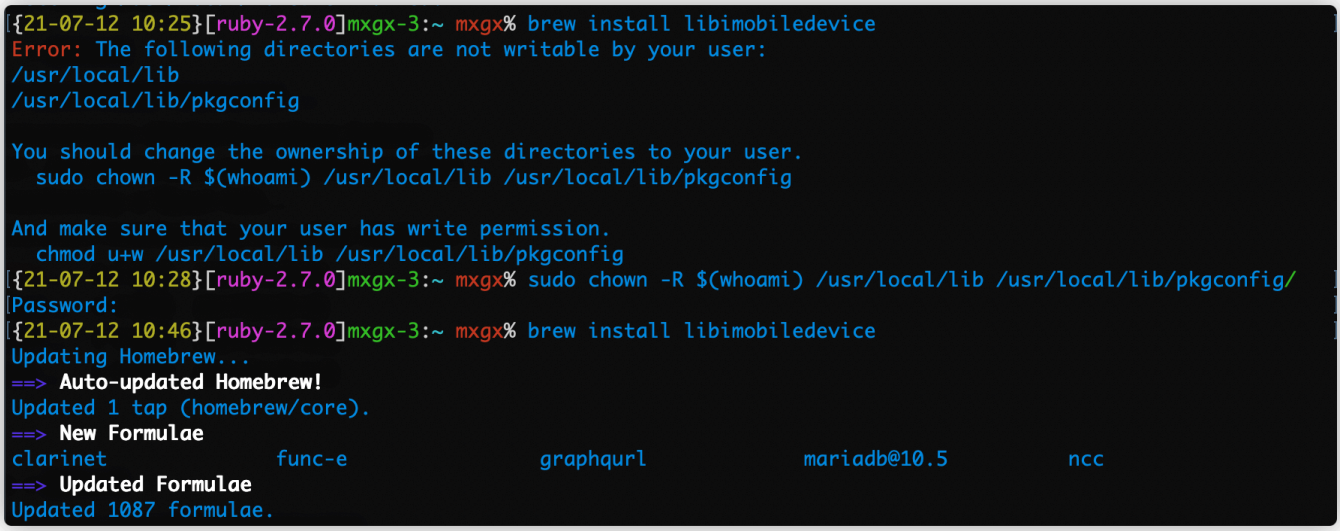
```
brew install libimobiledevice
```

执行过程中可能会出现如下问题：

```
{21-07-12 10:25}[ruby-2.7.0]mxgx-3:~ mxgx% brew install libimobiledevice
Error: The following directories are not writable by your user:
/usr/local/lib
/usr/local/lib/pkgconfig

You should change the ownership of these directories to your user.
  sudo chown -R $(whoami) /usr/local/lib /usr/local/lib/pkgconfig
```

```
And make sure that your user has write permission.
  chmod u+w /usr/local/lib /usr/local/lib/pkgconfig
{21-07-12 10:28}[ruby-2.7.0]mxgx-3:~ mxgx% sudo chown -R $(whoami) /usr/local/lib
/usr/local/lib/pkgconfig/
Password:
{21-07-12 10:46}[ruby-2.7.0]mxgx-3:~ mxgx% brew install libimobiledevice
Updating Homebrew...
==> Auto-updated Homebrew!
Updated 1 tap (homebrew/core).
==> New Formulae
clarinet               func-e                 graphqurl              mariadb@10.5
 ncc
==> Updated Formulae
Updated 1087 formulae.
......
......
......
```



只需要依照终端返回的信息，执行对应的命令即可，中途中可能会需要输入 `管理员密码`，输入回车即可。

```
# 本人此处执行该指令：
sudo chown -R $(whoami) /usr/local/lib /usr/local/lib/pkgconfig/
```

最后执行结果：

```
{21-07-12 10:46}[ruby-2.7.0]mxgx-3:~ mxgx% brew install libimobiledevice
Updating Homebrew...
==> Auto-updated Homebrew!
Updated 1 tap (homebrew/core).
==> New Formulae
clarinet               func-e                 graphqurl              mariadb@10.5
 ncc
==> Updated Formulae
```

```
Updated 1087 formulae.

==> Downloading https://ghcr.io/v2/homebrew/core/libplist/manifests/2.2.0
######################################################################## 100.0%
==> Downloading
https://ghcr.io/v2/homebrew/core/libplist/blobs/sha256:1ac05ef69cc02f4663fbb1c3d6d6e964
c70a5ba
......
......      ----> 该处省略
......
==> Downloading from https://pkg-
containers.githubusercontent.com/ghcr1/blobs/sha256:277577a3a30ff9bf60d0e4b81
######################################################################## 100.0%
==> Installing dependencies for wget: libidn2
==> Installing wget dependency: libidn2
==> Pouring libidn2--2.3.1.big_sur.bottle.1.tar.gz
🍺  /usr/local/Cellar/libidn2/2.3.1: 73 files, 812.3KB
==> Installing wget
==> Pouring wget--1.21.1.big_sur.bottle.1.tar.gz
🍺  /usr/local/Cellar/wget/1.21.1: 88 files, 4MB
Removing: /usr/local/Cellar/wget/1.20.3_2... (50 files, 4.0MB)
==> Checking for dependents of upgraded formulae...
==> No broken dependents found!
{21-07-12 11:24}[ruby-2.7.0]mxgx-3:~ mxgx%
```

有时候安装 `libimobiledevice` 可能会出现如下类似报错：

```
Requested 'libusbmuxd >= 1.1.0' but version of libusbmuxd is 1.0.10
```

解决方法如下：

```
brew update
brew uninstall --ignore-dependencies libimobiledevice
brew uninstall --ignore-dependencies usbmuxd
brew install --HEAD usbmuxd
brew unlink usbmuxd
brew link usbmuxd
brew install --HEAD libimobiledevice
```

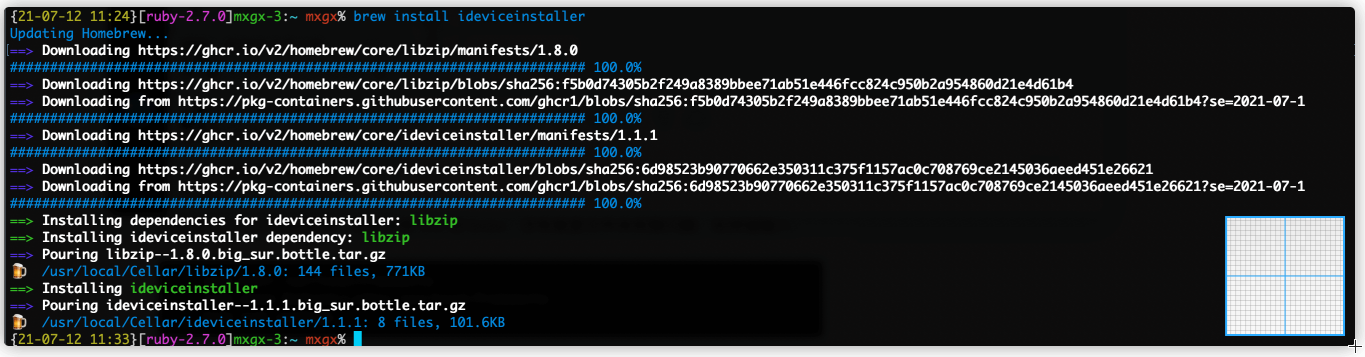
03 - 安装 ideviceinstaller（真机安装相关）

```
brew install ideviceinstaller
```

执行结果如下：

```
{21-07-12 11:24}[ruby-2.7.0]mxgx-3:~ mxgx% brew install ideviceinstaller
Updating Homebrew...
```

```
==> Downloading https://ghcr.io/v2/homebrew/core/libzip/manifests/1.8.0
######################################################################## 100.0%
==> Downloading
https://ghcr.io/v2/homebrew/core/libzip/blobs/sha256:f5b0d74305b2f249a8389bbee71ab51e44
6fcc824c950b2a954860d21e4d61b4
==> Downloading from https://pkg-
containers.githubusercontent.com/ghcr1/blobs/sha256:f5b0d74305b2f249a8389bbee71ab51e446
fcc824c950b2a954860d21e4d61b4?se=2021-07-1
######################################################################## 100.0%
==> Downloading https://ghcr.io/v2/homebrew/core/ideviceinstaller/manifests/1.1.1
######################################################################## 100.0%
==> Downloading
https://ghcr.io/v2/homebrew/core/ideviceinstaller/blobs/sha256:6d98523b90770662e350311c
375f1157ac0c708769ce2145036aeed451e26621
==> Downloading from https://pkg-
containers.githubusercontent.com/ghcr1/blobs/sha256:6d98523b90770662e350311c375f1157ac0
c708769ce2145036aeed451e26621?se=2021-07-1
######################################################################## 100.0%
==> Installing dependencies for ideviceinstaller: libzip
==> Installing ideviceinstaller dependency: libzip
==> Pouring libzip--1.8.0.big_sur.bottle.tar.gz
🍺  /usr/local/Cellar/libzip/1.8.0: 144 files, 771KB
==> Installing ideviceinstaller
==> Pouring ideviceinstaller--1.1.1.big_sur.bottle.tar.gz
🍺  /usr/local/Cellar/ideviceinstaller/1.1.1: 8 files, 101.6KB
{21-07-12 11:30}[ruby-2.7.0]mxgx-3:~ mxgx%
```



4、配置 Java、Android 相关环境

Java SDK（JDK）

- Java 官方 SDK 下载地址

```
# 获取 jdk 环境
java -version     # java version "1.8.0_291"
javac -version    # javac 1.8.0_251
```

Android

- Android Studio 下载地址

需要在如下指定路径中下载对应的 `Android SDK`：

> Android Studio -> Preferences -> Appearance & Behavior -> System Settings -> Android SDK

```
# 获取 Android 环境 -> adb
adb --version
adb version

# 执行结果
Android Debug Bridge version 1.0.41
Version 30.0.3-6597393
Installed as /Users/mxgx/Library/Android/sdk/platform-tools/adb
```

在使用 `adb` 指令时，由于环境变量配置问题，经常会出现如下报错：

```
{21-10-14 11:43}[ruby-2.7.0]mxgx:~ mxgx% adb version
zsh: command not found: adb
```

参考链接 - Mac下adb环境变量的配置（解决zsh: command not found: adb问题）

```
# 说明：
# 此处是 zsh 配置，如果 Mac 是 bash 配置的话，则需要在 .bash_profile 文件中配置，配置和 .zshrc 配置相同

# 在终端（Terminal）中输入命令会自动进入编辑模式（如果没有对应文件会自动创建）
vim ~/.zshrc

# 将 adb 命令添加到 PATH 中
# androidsdk 环境变量
# export PATH=${PATH}:"/Users/apple/Library/Android/sdk/platform-tools"
export ANDROID_HOME=/Users/mxgx/Library/Android/sdk
export PATH=$PATH:$ANDROID_HOME/platform-tools

# 执行生效
source ~/.zshrc

# 验证 adb 环境
adb version

# 执行结果
Android Debug Bridge version 1.0.41
Version 30.0.3-6597393
Installed as /Users/mxgx/Library/Android/sdk/platform-tools/adb
```

Android 模拟器

Genymotion - 官方网站
Virtualbox - 官方下载地址

adb 相关知识点

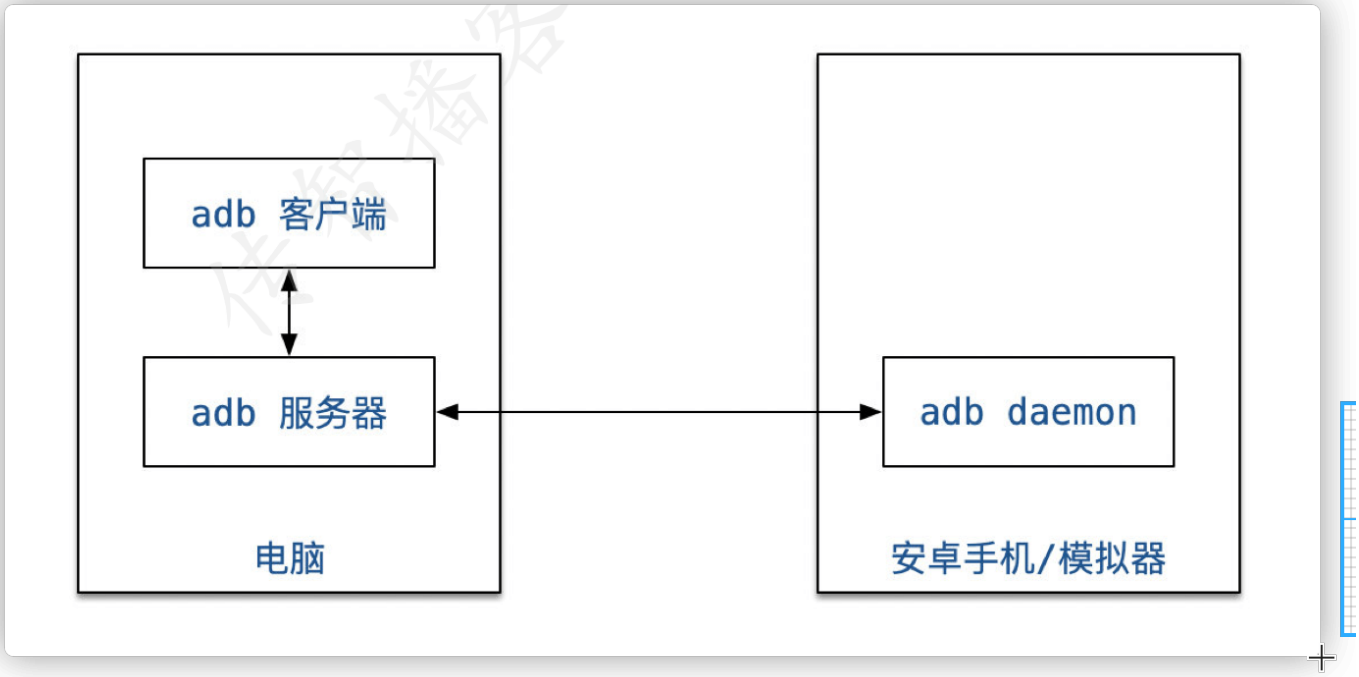
adb 简介

> adb - 官方介绍

`adb` 全称 `Android Debug Bridge`，是一个命令行调试工具，可以让其与设备进行通信。`adb` 命令可用于执行各种设备操作（例如安装和调试应用），并提供对 `Unix shell`（可用来在设备上运行各种命令）的访问权限。

`adb` 是一种客户端-服务器程序，包括以下三个组件：

- `Client 端`：
  - `用于发送命令，客户端在开发机器上运行`。您可以通过发出 `adb` 命令从命令行终端调用客户端。
- `Daemon 守护进程（守护程序）- adbd`：
  - `用于在设备上运行命令`。守护程序在每个设备上作为后台进程运行；
  - 运行在调试设备中，手机或模拟器，用来接收并执行 `adb` 命令。
- `Server 端`：
  - `用于管理客户端与守护程序之间的通信`。服务器在开发机器上作为后台进程运行。



adb 工作原理

- 当启动某个 `adb` 客户端时，该客户端会先检查是否有 `adb` 服务器进程正在运行；
- 如果没有，它会启动服务器进程。服务器在启动后会与 `本地 TCP 端口 5037` 绑定，并监听 `adb` 客户端发出的命令：`所有 adb 客户端均通过端口 5037 与 adb 服务器通信`；
- 然后，`服务器会与所有正在运行的设备建立连接`。它通过扫描 5555 到 5585 之间（该范围供前 16 个模拟器使用）的奇数号端口查找模拟器。服务器一旦发现 `adb 守护程序 (adbd)`，便会与相应的端口建立连接。

    - 请注意，每个模拟器都使用一对按顺序排列的端口 - 用于控制台连接的偶数号端口和用于 adb 连接的奇数号端口。例如：
        - 模拟器 1，控制台：5554
        - 模拟器 1，adb：5555
        - 模拟器 2，控制台：5556
        - 模拟器 2，adb：5557
        - 依此类推
    - 如上所示，在端口 5555 处与 adb 连接的模拟器与控制台监听端口为 5554 的模拟器是同一个。
- 服务器与所有设备均建立连接后，便可以使用 `adb` 命令访问这些设备。由于服务器管理与设备的连接，并处理来自多个 adb 客户端的命令，因此您可以从任意客户端（或从某个脚本）控制任意设备。

adb 常用命令

获取包名（package）和界面名（activity）

```
# 获取包名（package）和界面名（activity）
# Mac / Linux
adb shell dumpsys window windows | grep mFocusedApp
# Windows
adb shell dumpsys window windows | findstr mFocusedApp

# 执行结果
# 包名：com.android.settings
# 界面名：.Settings
mFocusedApp=AppWindowToken{53309da token=Token{2e2fa785 ActivityRecord{2928d4fc u0
com.android.settings/.Settings t1127}}}
```

文件传输 - 发送文件到手机 和 从手机中拉取文件

```
# 发送文件到手机
adb push 电脑的文件路径 手机的文件夹路径
# 将桌面的 a.txt 发送到手机的 sd 卡
adb push C:\Users\hm\Desktop\a.txt /sdcard

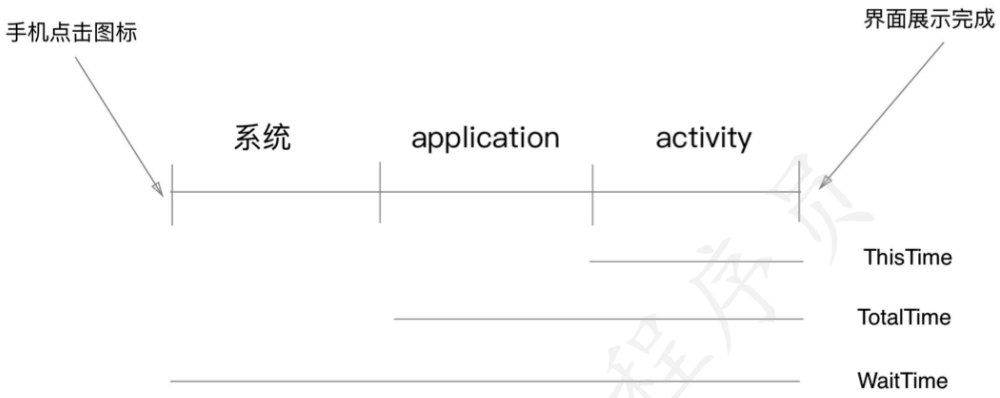
# 从手机中拉取文件
adb pull 手机的文件路径 电脑的文件夹路径
# 将手机的 sd 卡的 a.txt 拉取到桌面
adb pull /sdcard/a.txt C:\Users\hm\Desktop
```

获取 App 的启动时间

```
# 获取 App 的启动时间
adb shell am start -W 包名/启动名
# 启动 com.android.settings 程序并且进入主界面 (.Settings)
adb shell am start -W com.android.settings/.Settings
```

解释

1. `ThisTime`：该界面（`activity`）启动耗时（毫秒）
2. `TotalTime`：应用自身启动耗时 = `ThisTime` + 应用 application 等资源启动时间（毫秒）
3. `WaitTime`：系统启动应用耗时 = `TotalTime` + 系统资源启动时间（毫秒）



adb 其它指令

```
# 获取 log 日志
adb logcat
# 安装 app 到手机
adb install 路径/xx.apk
# 卸载手机上的 app，需要指定包名
adb uninstall apk
# 获取当前电脑已经连接设备和对应的设备号
adb devices
# 进入到安卓手机内部的linux系统命令行中
adb shell
# 启动 adb 服务端，出 bug 时使用可以重启服务器，先关 闭再启动
adb start-server
# 停止 adb 服务端，出 bug 时使用可以重启服务器，先关 闭再启动
adb kill-server
# 获取 adb 指令帮助
adb --help
```

UIAutomatorViewer

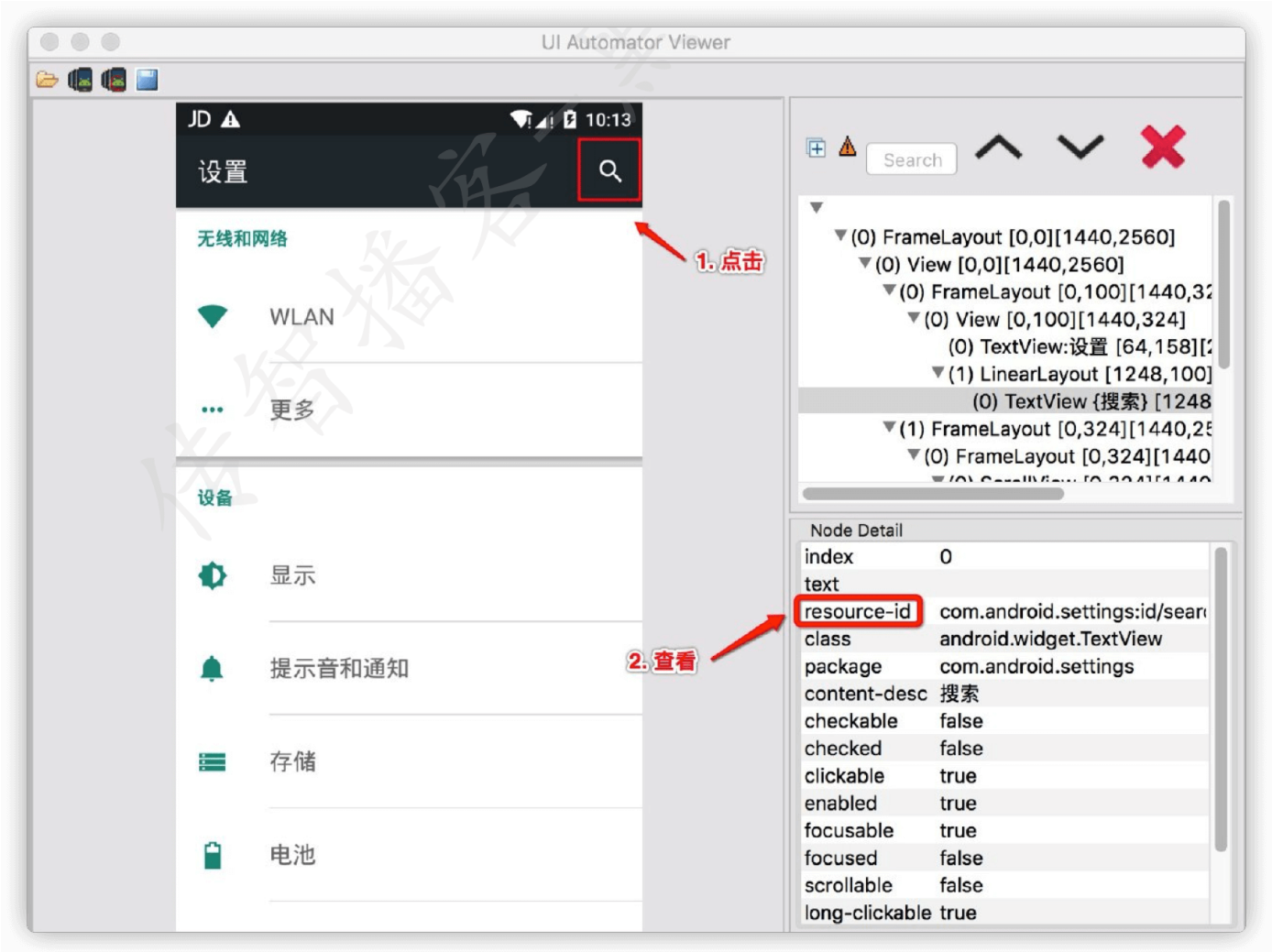
UIAutomatorViewer 简介

`UIAutomatorViewer` 用来扫描和分析 `Android` 应用程序的 `UI` 控件的工具。

定位元素的时候必须根据元素的相关特征来进行定位，而 `UIAutomatorViewer` 就是用来获取元素特征的。

UIAutomatorViewer 使用步骤

- 进入SDK目录下的目录
  - `Mac` 在 `tools/bin` 目录下，打开 `uiautomatorviewer`；
  - `Windows` 在 `tools` 目录下，打开 `uiautomatorviewer.bat`。
- 电脑连接真机或打开 `Android` 模拟器
- 启动待测试 App
- 点击 `uiautomatorviewer` 的左上角 `Device Screenshot`（从左数第二个按钮）
- 点击希望查看的控件
- 查看右下角 `Node Detail` 相关信息



Appium 使用

Appium 使用步骤

- 打开 `模拟器`
- 打开 `Appium` 调试工具
- 打开 `PyCharm` 开发工具，创建一个 python 项目
- 创建 `demo.py` 文件
- 执行相应 python 代码，并运行程序即可

Appium 启动流程

- 写的 `python` 代码会访问本机的 `appium 服务器`，并获取 `driver 对象`；
- `appium` 会将我们的 `driver 对象` 调用的方法转化成 `post 请求`，提交给 `appium 服务器`；
- `appium` 通过接收到的 `post 请求` 发送给 `手机`，再由手机进行执行。

Appium 在 Python 文件中的前置代码

```
# iOS 自动化测试 在 python 文件中的前置代码
from appium import webdriver

desired_caps = dict()
desired_caps['platformName'] = 'iOS'
desired_caps['platformVersion'] = '12.1'
desired_caps['deviceName'] = 'iPhone 8'
desired_caps['app'] = 'com.itcast.HMiOSTest'

driver = webdriver.Remote('http://localhost:4723/wd/hub', desired_caps)

# 此处写自动化测试代码

driver.quit()
```

```
# Android 自动化测试 在 python 文件中的前置代码
from appium import webdriver

desired_caps = dict()
desired_caps['platformName'] = 'Android'
desired_caps['platformVersion'] = '5.1'
desired_caps['deviceName'] = '192.168.56.101:5555'
desired_caps['appPackage'] = 'com.android.settings'
desired_caps['appActivity'] = '.Settings'

driver = webdriver.Remote('http://localhost:4723/wd/hub', desired_caps)

# 此处写自动化测试代码

driver.quit()
```

Appium 在 Python 文件中的前置代码 参数解释

```
# iOS 自动化测试 在 python 文件中的前置代码
# 导模块
from appium import webdriver

# 创建一个字典，包装相应的启动参数
desired_caps = dict()
# 需要连接的手机的平台(不限制大小写)
desired_caps['platformName'] = 'iOS'
# 需要连接的手机的版本号
desired_caps['platformVersion'] = '12.1'
# 需要连接的手机的设备号
desired_caps['deviceName'] = 'iPhone 8'
# 需要启动的程序的包名
desired_caps['app'] = 'com.itcast.HMiOSTest'

# 连接 appium 服务器
driver = webdriver.Remote('http://localhost:4723/wd/hub', desired_caps)

# 此处写自动化测试代码

# 退出
driver.quit()
```

```
# Android 自动化测试 在 python 文件中的前置代码
# 导模块
from appium import webdriver

# 创建一个字典，包装相应的启动参数
desired_caps = dict()
# 需要连接的手机的平台(不限制大小写)
desired_caps['platformName'] = 'Android'
# 需要连接的手机的版本号(比如 5.2.1 的版本可以填写 5.2.1 或 5.2 或 5 ，以此类推)
desired_caps['platformVersion'] = '5.1'
# 需要连接的手机的设备号(andoird平台下，可以随便写，但是不能不写)
desired_caps['deviceName'] = '192.168.56.101:5555'
# 需要启动的程序的包名
desired_caps['appPackage'] = 'com.android.settings'
# 需要启动的程序的界面名
desired_caps['appActivity'] = '.Settings'

# 连接 appium 服务器
driver = webdriver.Remote('http://localhost:4723/wd/hub', desired_caps)

# 此处写自动化测试代码

# 退出
```

```
driver.quit()
```

Appium 在 Python 中常用方法

```
# 脚本内启动其他 App
# appPackage：要打开的程序的包名
# appActivity：要打开的程序的界面名
driver.start_activity(appPackage, appActivity)

# 获取包名
driver.current_package
# 获取界面名
driver.current_activity

# 关闭当前操作的 App，不会关闭驱动对象
driver.close_app()
# 关闭驱动对象，同时关闭所有关联的 App
driver.quit()

# 安装 App
# app_path：apk路径
driver.install_app(app_path)
# 卸载 App
# app_id：应用程序包名
driver.remove_app(app_id)
# 判断 App 是否已经安装
# app_id：应用程序包名
# 返回值：布尔类型，True 为安装，False 为没有安装
driver.is_app_installed(app_id)

# App 放置到后台一定时间后再回到前台，模拟热启动
# seconds：后台停留多少秒
driver.background_app(seconds)
```

Appium 简单自动化测试

需求：

- 1、点击按钮 进入列表
- 2、获取所有列表的文字内容并输出
- 3、滑动一次
- 4、点击 20
- 5、清空文本框内容
- 6、在文本框中输入 “hello”
- 7、单击返回按钮

```
# iOS 自动化测试 在 python 文件中的前置代码
from appium import webdriver
```

```
desired_caps = dict()
desired_caps['platformName'] = 'iOS'
desired_caps['platformVersion'] = '12.1'
desired_caps['deviceName'] = 'iPhone 8'
desired_caps['app'] = 'com.itcast.HMiOSTest'

driver = webdriver.Remote('http://localhost:4723/wd/hub', desired_caps)

# 此处写自动化测试代码
# 1、点击按钮 进入列表
time.sleep(1)
driver.find_element_by_id("进入列表").click()

# 2、获取所有列表的文字内容并输出
time.sleep(1)
eles = driver.find_elements_by_class_name("XCUIElementTypeStaticText")
for i in eles:
  print(i.text)

# 3、滑动一次
time.sleep(1)
# 无法使用 # driver.swipe(100, 2000, 100, 1000)
driver.execute_script("mobile: swipe", {"direction": "up"})

# 4、点击 20
time.sleep(1)
driver.find_element_by_xpath("//*[@name='20']").click()

# 5、清空文本框内容
time.sleep(1)
text_view = driver.find_element_by_class_name("XCUIElementTypeTextField")
text_view.clear()

# 6、在文本框中输入 “hello”
time.sleep(1)
text_view.send_keys("hello!!!")

# 7、单击返回按钮
time.sleep(1)
driver.find_element_by_xpath("//XCUIElementTypeButton[@name='Back']").click()

driver.quit()
```

Appium 元素操作

Appium 元素定位

```
# Appium 定位一个元素

# 通过 id 定位一个元素
# id_value：元素的 resource-id 属性值
# 返回值：定位到的单个元素
driver.find_element_by_id(id_value)

# 通过 class_name 定位一个元素
# class_value：元素的 class 属性值
# 返回值：定位到的单个元素
driver.find_element_by_class_name(class_value)

# 通过 xpath 定位一个元素
# xpath_value：定位元素的 xpath 表达式
# 返回值：定位到的单个元素
driver.find_element_by_xpath(xpath_value)


# Appium 定位一组元素

# 通过 id 定位一组元素
# id_value：元素的 resource-id 属性值
# 返回值：列表，定位到的所有符合调价你的元素
driver.find_elements_by_id(id_value)

# 通过 class_name 定位一组元素
# class_value：元素的 class 属性值
# 返回值：列表，定位到的所有符合调价你的元素
driver.find_elements_by_class_name(class_value)

# 通过 xpath 定位一组元素
# xpath_value：定位元素的 xpath 表达式
# 返回值：列表，定位到的所有符合调价你的元素
driver.find_elements_by_xpath(xpath_value)
```

注意点：

- 如果使用 `find_element_by_xx` 方法，如果传入一个没有的特征，会报 `NoSuchElementException` 的错误；
- 如果使用 `find_elements_by_xx` 方法，如果传入一个没有的特征，不会报错，会返回一个空列表。

Appium 元素等待

应用场景：

- 可能由于一些原因，我们想找的元素并没有立刻出来，此时如果直接定位可能会报错，比如以下原因：
  - 由于网络速度原因；
  - 服务器处理请求原因；
  - 电脑配置原因。

元素等待 概念：

`WebDriver` 定位页面元素时如果未找到，会在指定时间内一直等待的过程，元素等待一共分为两种类型：

- **显式等待**
  - 核心代码：`wait.until(method)`
  - 等待元素加载指定的时长，超出时长抛出 `TimeoutException` 异常。
- **隐式等待**
  - 核心代码：`implicitly_wait(timeout)`
  - 等待元素加载指定的时长，超出时长抛出 `NoSuchElementException` 异常。

元素等待显式与隐式的选择：

- 作用域：显式等待为单个元素有效，隐式为全局元素；
- 方法：显式等待方法封装在 WebDriverWait 类中，而隐式等待则直接通过 driver 实例化对象调用。

关于 sleep：

- sleep 是固定死一个时间，不是不行，是不推荐；
- 元素等待可以让元素出来的第一时间进行操作，sleep 可能造成不必要的浪费。

```
# 场景：
# 在 5 秒钟内，在《设置》程序中的 “返回“ 按钮，如果找到则点击。如果找不到则观察对应错误信 息。


# 显式等待
# 创建WebDriverWait对象
# driver：驱动对象
# timeout：超时的时长，单位：秒
# poll_frequency：检测间隔时间，默认为 0.5 秒
# 返回值：WebDriverWait 对象
wait = WebDriverWait(driver, 5, poll_frequency=1)
# 获取元素并设置超时时间和频率
# method：lambda 查找元素表达式
# 返回值：定位到的元素，如果没有定位到会抛出 TimeoutException 异常
search_button = wait.until(lambda x: x.find_element_by_xpath("//* [contains(@content-
desc,'收起')]"))
# 点击搜索按钮
search_button.click()
```

```
# 隐式等待
driver.implicitly_wait(5)
search_button = driver.find_element_by_xpath("//*[contains(@content-desc,'收 起')]")
search_button.click()
```

元素操作

```
# 元素操作

# 对 element 按钮进行点击操作
element.click()

# 对 element 输入框进行输入操作
# value：输入的内容
# 注：默认输入中文无效，但不会报错，需要在 “前置代码” 中增加两个参数
#   - desired_caps['unicodeKeyboard'] = True
#   - desired_caps['resetKeyboard'] = True
element.send_keys(value)
# 对 element 输入框进行清空操作
element.clear()

# 获取 element 控件（按钮、输入框、文本框等）的文本内容
element.text

# 获取 element 的位置
# 返回值：字典，x 为元素的 x 坐标，y 为元素的 y 坐标
element.location
# 获取 element 的大小
# 返回值：字典，width 为宽度，height 为高度
element.size

# 对 element 进行点击操作
# value：要获取的属性名
# 返回值：根据属性名得到的属性值
# 注意点：
#   - value = 'text'         返回 text 的属性值
#   - value = 'name'         返回 content-desc / text 属性值
#   - value = 'className'    返回 class 属性值，只有 API => 18 才能支持
#   - value = 'resourceId'   返回 resource-id 属性值，只有 API => 18 才能支持
element.get_attribute(value)
# 示例
title.get_attribute("text")
title.get_attribute("ClassName")
```

滑动和拖拽事件

swipe 滑动事件

- 概念：
  - 从一个坐标位置滑动到另一个坐标位置，只能是两个点之间的滑动。
- 核心代码：
  - `driver.swipe(start_x, start_y, end_x, end_y, duration=None)`
- 结论：
  - `距离相同时，持续时间越长，惯性越小；`
  - `持续时间相同时，手指滑动的距离越大，实际滑动的距离也就越大。`

```
# swipe 滑动事件
#
# 从一个坐标位置滑动到另一个坐标位置，只能是两个点之间的滑动
# start_x:  起点 X 轴坐标
# start_y:  起点 Y 轴坐标
# end_x:    终点 X 轴坐标
# end_y:    终点 Y 轴坐标
# duration: 滑动这个操作一共持续的时间长度，单位：ms
driver.swipe(start_x, start_y, end_x, end_y, duration=None)

# 示例1：模拟手指从（100, 2000），滑动到（100, 1000）的位置
driver.swipe(100, 2000, 100, 1000)
# 示例2：模拟手指从（100, 2000），滑动到（100, 100）的位置
driver.swipe(100, 2000, 100, 100)
# 示例3：模拟手指从（100, 2000），滑动到（100, 100）的位置，持续5秒
driver.swipe(100, 2000, 100, 100, 5000)

# 结论：
# - 距离相同时，持续时间越长，惯性越小；
# - 持续时间相同时，手指滑动的距离越大，实际滑动的距离也就越大。
```

scroll 滑动事件

- 概念：
  - 从一个元素滑动到另一个元素，直到页面自动停止。
- 核心代码：
  - `driver.scroll(origin_el, destination_el)`
- 结论：
  - `不能设置持续时间，惯性很大。`

```
# scroll 滑动事件
#
# 从一个元素滑动到另一个元素，直到页面自动停止
# origin_el：滑动开始的元素
```

```
# destination_el：滑动结束的元素
driver.scroll(origin_el, destination_el)

# 示例1：从 “存储” 滑动到 “更多”
save_button = driver.find_element_by_xpath("//*[@text='存储']")
more_button = driver.find_element_by_xpath("//*[@text='更多']")
driver.scroll(save_button, more_button)

# 结论：
# - 不能设置持续时间，惯性很大。
```

drag_and_drop 拖拽事件

- 概念：
  - 从一个元素滑动到另一个元素，第二个元素替代第一个元素原本屏幕上的位置。
- 核心代码：
  - `driver.drag_and_drop(origin_el, destination_el)`
- 结论：
  - `不能设置持续时间，没有惯性。`

```
# drag_and_drop 滑动事件
#
# 从一个元素滑动到另一个元素，第二个元素替代第一个元素原本屏幕上的位置
# origin_el：滑动开始的元素
# destination_el：滑动结束的元素
driver.drag_and_drop(origin_el, destination_el)

# 示例1：从 “存储” 滑动到 “更多”
save_button = driver.find_element_by_xpath("//*[@text='存储']")
more_button = driver.find_element_by_xpath("//*[@text='更多']")
driver.drag_and_drop(save_button, more_button)

# 结论：
# - 不能设置持续时间，没有惯性。
```

滑动和拖拽事件选择

- 滑动和拖拽无非就是考虑是否有 `“惯性”`，以及传递的参数是 `“元素”` 还是 `“坐标”`：
  - `有 “惯性”，传入 “元素”`
    - scroll
  - `无 “惯性”，传入 “元素”`
    - drag_and_drop
  - `有 “惯性”，传入 “坐标”`
    - swipe，并且设置较短的 duration 时间
  - `无 “惯性”，传入 “坐标”`

- swipe，并且设置较长的 duration 时间

高级手势 - TouchAction

`TouchAction` 可以实现一些针对手势的操作，比如 `滑动、长按、拖动` 等。我们可以将这些基本手势组合成一个相对复杂的手势。比如，我们解锁手机或者一些应用软件都有手势解锁的这种方式。

执行步骤：

- 1、创建 `TouchAction` 对象
- 2、通过对象调用想执行的手势
- 3、通过 `perform()` 执行动作
  - 所有手势都要通过执行 `perform()` 函数才会运行。

轻敲操作

```
# 轻敲操作

# 模拟手指对元素或坐标的轻敲操作
# element：元素
# x：x 坐标
# y：y 坐标
TouchAction(driver).tap(element=None, x=None, y=None).perform()


# 示例 - 1. 打开《设置》；2. 轻敲 “WLAN”
el = driver.find_element_by_xpath("//*[contains(@text,'WLAN')]")
TouchAction(driver).tap(el).perform()
```

按下和抬起操作

```
# 按下和抬起操作
# 模拟手指一直按下，模拟手指抬起。可以用来组合成轻敲或长按的操作

# 模拟手指对元素或坐标的按下操作
# el：元素
# x：x 坐标
# y：y 坐标
TouchAction(driver).press(el=None, x=None, y=None).perform()

# 模拟手指对元素或坐标的抬起操作
TouchAction(driver).release().perform()


# 示例1 - 使用坐标的形式按下 WLAN （650，650），2 秒后，按下（650，650）的位置
TouchAction(driver).press(x=650, y=650).perform()
time.sleep(2)
TouchAction(driver).press(x=650, y=650).perform()
```

```
# 示例2 - 使用坐标的形式按下 WLAN （650, 650），2 秒后，按下（650, 650）的位置，并抬起
TouchAction(driver).press(x=650, y=650).perform()
time.sleep(2)
TouchAction(driver).press(x=650, y=650).release().perform()
```

等待操作

```
# 等待操作

# 模拟手指暂定操作
# ms：暂停的毫秒数
TouchAction(driver).wait(ms=0).perform()


# 示例 - 使用坐标的形式点击 WLAN （650, 650），2 秒后，按下（650, 650）的位置，暂停 2 秒，并抬 起
TouchAction(driver).tap(x=650, y=650).perform()
time.sleep(2)
TouchAction(driver).press(x=650, y=650).wait(2000).release().perform()
```

长按操作

```
# 长按操作

# 模拟手指对元素或坐标的长按操作
# el：元素
# x：x 坐标
# y：y 坐标
# duration：长按时间，毫秒
TouchAction(driver).long_press(el=None, x=None, y=None, duration=1000).perform()


# 示例 - 使用坐标的形式点击 WLAN （650, 650），2 秒后，长按（650, 650）的位置持续 2 秒
TouchAction(driver).tap(x=400, y=400).perform()
time.sleep(2)
TouchAction(driver).long_press(x=400, y=400, duration=2000).release().perform()
```

移动操作

```
# 移动操作

# 模拟手指对元素或坐标的移动操作
# el：元素
# x：x 坐标
# y：y 坐标
TouchAction(driver).move_to(el=None, x=None, y=None).perform()


# 示例 - 在手势解锁中，画一个手势解锁路径
TouchAction(driver).press(x=246, y=857).move_to(x=721, y=867).move_to(x=1200,
y=851).move_to(x=1200, y=1329).move_to(x=724, y=1329).move_to(x=246,
y=1329).move_to(x=718, y=1815).release().perform()
```

手机操作 API

```
# 获取手机分辨率
driver.get_window_size()
# 结果：{'height': 800, 'width': 480}

# 手机截图
# filename：指定路径下，指定格式的图片
get_screenshot_as_file(filename)
# 示例 - 1、打开设置页面；2、截图当前页面保存到当前目录，命名为 screen.png
driver.get_screenshot_as_file(os.getcwd() + os.sep + './screen.png')

# 获取手机网络
driver.network_connection

# 设置手机网络
# connectionType：网络类型
driver.set_network_connection(connectionType)

# 发送键到设备
# keycode：发送给设备的关键代码
# metastate：关于被发送的关键代码的元信息，一般为默认值
# 按键对应的编码，可以在百度搜索关键字 “android keycode”
# 例如：https://blog.csdn.net/feizhixuan46789/article/details/16801429
driver.press_keycode(keycode, metastate=None)

# 打开手机通知栏
# Appium 官方并没有为我们提供关闭通知的 api，那么现实生活中怎么关闭，就怎样操作就行，比 如，手指从下往上滑动，或者，按返回键
driver.open_notifications()
```

```
Possible values:
    Value (Alias)      | Data | Wifi | Airplane Mode
    ---------------------------------------------------
    0 (None)           | 0    | 0    | 0
    1 (Airplane Mode)  | 0    | 0    | 1
    2 (Wifi only)      | 0    | 1    | 0
    4 (Data only)      | 1    | 0    | 0
    6 (All network on) | 1    | 1    | 0
```

参考链接

- Appium - 官网
- Appium - 简介

  -
- MAC下搭建 appium+ios+python 自动化测试环境（一）
- Mac 下 appium 自动化测试iOS 测试配置和脚本编写(二)
- Mac 下 搭建appium +android+python 自动化测试环境(三)
- Mac 下 appium 自动化测试 Android 测试配置和脚本编写(四)

版权声明