干货|app自动化测试之Appium 源码修改定制分析

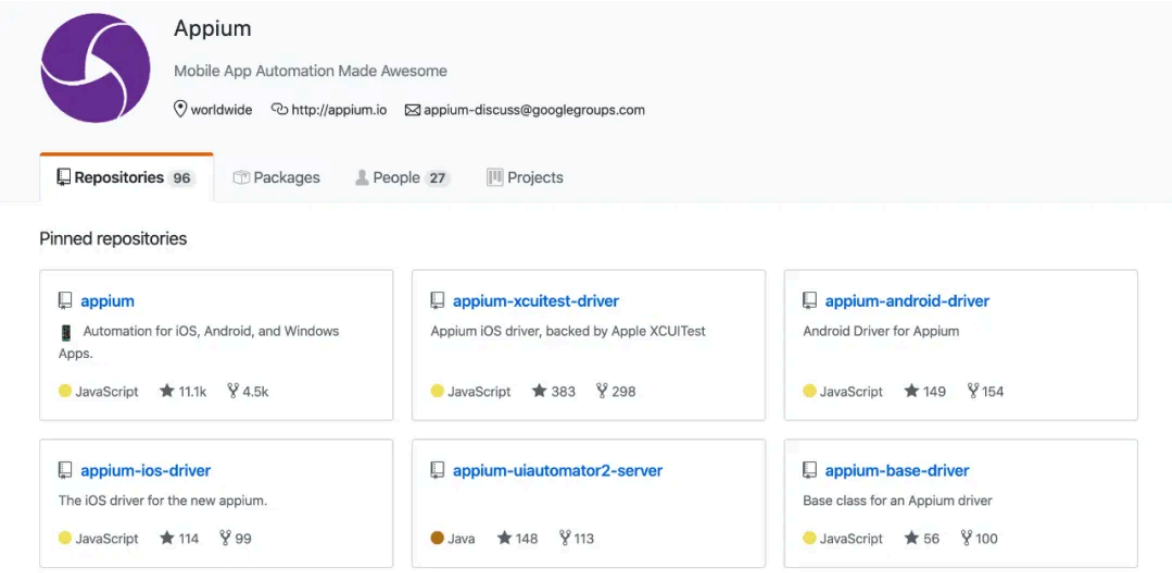
原创 子奕 霍格沃兹测试学院 2021年11月01日 08:00

> 本文节选自霍格沃兹测试学院内部教材

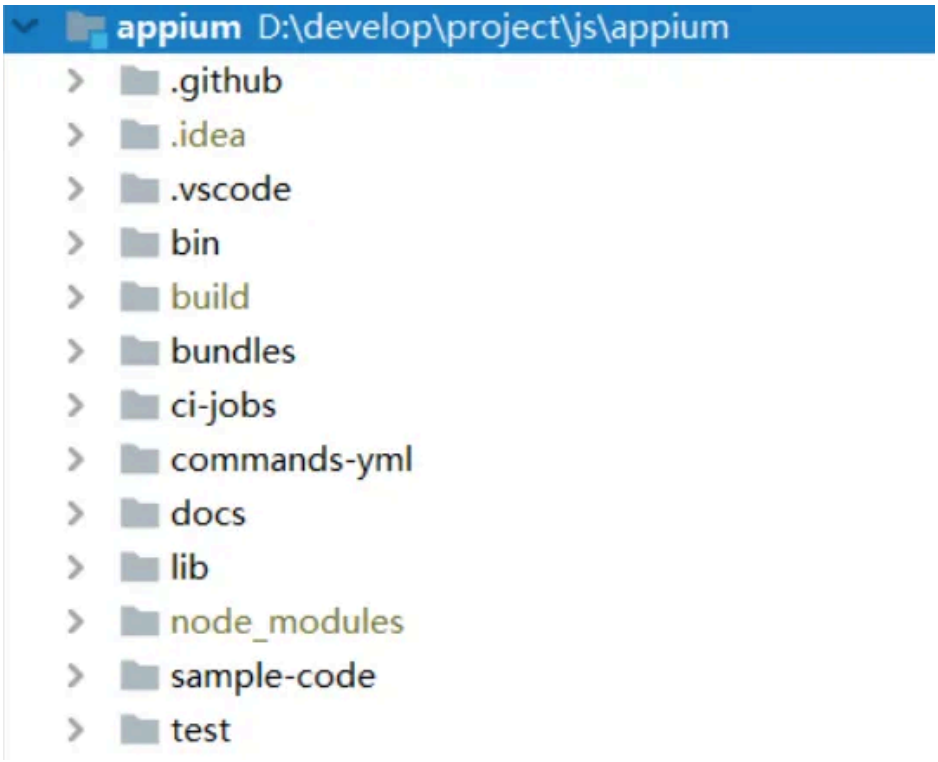
Appium 是由 Node.js 来实现的 HTTP 服务，它并不是一套全新的框架，而是将现有的优秀的框架进行了集成，在 Selenium WebDriver 协议（JsonWireProtocol/Restful web service）的基础上增加了移动端的支持，使 Appium 满足多方面的需求。
官方提供更详细的 Appium 结构说明：https://appium.io/docs/en/contributing-to-appium/appium-packages/

Appium 框架结构

Appium 是由多个子项目构成的，github 访问如下图：



Appium 由 Appium 以及其它的工作引擎包括：appium-xcuitest-driver、appium-android-driver、appium-ios-driver、appium-uiautomator2-server、appium-base-driver 等组成。下载 Appium 这个项目进行分析，发现 Appium 有着非常复杂的目录结构，如下图：



其中重要的目录如下：

- bin 文件夹 node.js 项目的可执行文件配置项
- docs 文件夹 说明文档（有中文）
- lib 文件夹 node.js 的源码文件夹
- node_modules 文件夹 node.js 项目默认存放插件的目录
- test 文件夹 测试代码所在文件夹

项目中有个文件 package.json ，这个文件是项目的描述文件。对项目或者模块包的描述，比如项目名称，项目版本，项目执行入口文件，项目贡献者等等。`npm install`命令会根据这个文件下载所有依赖模块，查看这个文件可以看到如下的信息：

```
"dependencies": {
    "@babel/runtime": "^7.6.0",
    "appium-android-driver": "^4.20.0",
    "appium-base-driver": "^5.0.0",
    "appium-espresso-driver": "^1.0.0",
    "appium-fake-driver": "^0.x",
    "appium-flutter-driver": "^0",
    "appium-ios-driver": "4.x",
    "appium-mac-driver": "1.x",
    "appium-support": "2.x",
    "appium-tizen-driver": "^1.1.1-beta.4",
    "appium-uiautomator2-driver": "^1.37.1",
    "appium-windows-driver": "1.x",
    "appium-xcuitest-driver": "^3.0.0",
    ...
```

```
16    },
```

dependencies 表示此模块依赖的模块和版本信息。从这里面可以看到它依赖很多 driver ，比如 appium-android-driver、appium-base-driver、appium-espresso-driver、appium-ios-driver、appium-uiautomator2-driver 等等。下面我们会根据 appium-uiautomator2-driver 重点对 Android 测试驱动的源码进行分析。

appium-uiautomator2-server

appium-uiautomator2-server 是针对 UiAutomator V2 提供的服务，是一个运行在设备上的 Netty 服务器，用来监听指令并执行 UiAutomator V2 命令。

早期版本 Appium 通过 appium-android-bootstrap 实现与 UiAutomator V1 的交互，UiAutomator2 修复了 UiAutomator V1 中遇到的大多数问题，最重要的是实现了与 Android 系统更新的分离。

Appium 底层执行 Android 测试真正的工作引擎是一个 JAVA 项目 appium-uiautomator2-server。可以将这个项目克隆到本地，使用 Android Studio 工具或者其它的 JAVA 项目 IDE 工具打开这个项目。

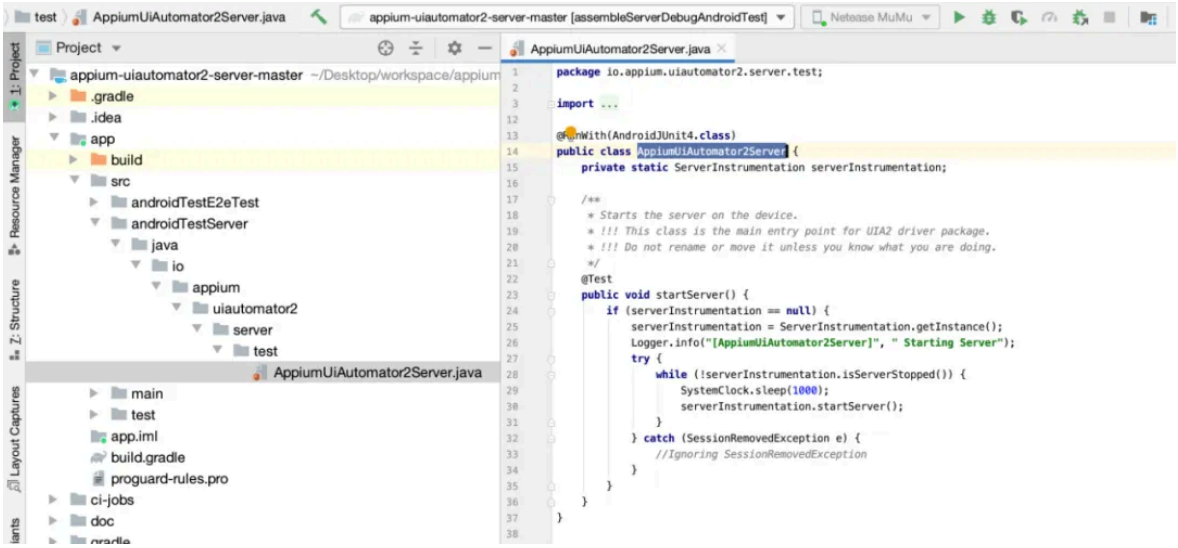
appium-uiautomator2-server启动

从 README 文件可以看到启动服务的方式：

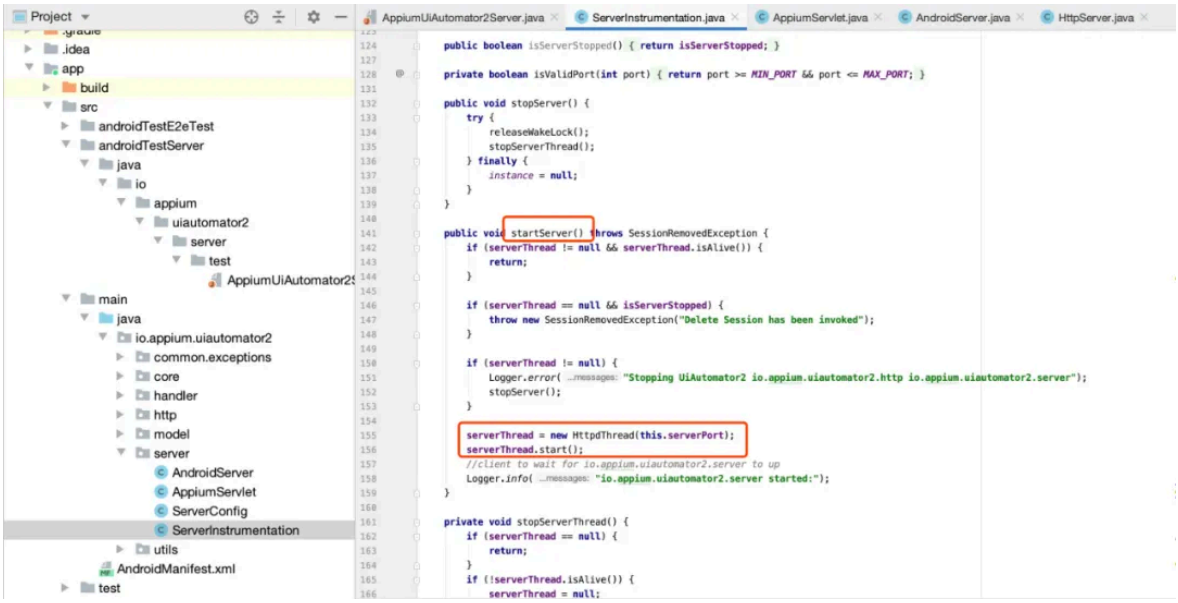
```
Starting server
push both src and test apks to the device \
and execute the instrumentation tests.

adb shell am instrument -w \
io.appium.uiautomator2.server.test/\
androidx.test.runner.AndroidJUnitRunner
```

找到 AppiumUiAutomator2Server.java 这个文件，如下图：



startServer( ) 方法就是它的启动入口函数。这个函数里面调用了 ServerInstrumentation 类里面的 startServer( ) 方法。如下图：



startServer( ) 方法会创建一个新的线程来处理一系列的逻辑。

- 首先会开启 Netty 服务，创建一个 AndroidServer 对象
- 然后设置好端口并调用 AppiumServlet（AppiumServlet 是用于管理请求的路由，将 Driver 发过来的请求转发给对应 Handler）
- Handler 会调用 UiAutomatorV2 去执行指定操作
- 操作的结果由 AppiumResponse 统一封装（AppiumResponse 会将操作结果返回给 appium-uiautomator2-driver，再将结果返给客户端）

AppiumServlet解析

AppiumServlet 是一个典型 HTTP 请求的处理协议。使用 AppiumServlet 来管理请求，并将 Driver 发过来的请求转发给对应 RequestHandler，它会监听下面的 URL

```
...
register(postHandler, new FindElement("/wd/hub/session/:sessionId/element"));
```

```
register(postHandler, new FindElements("/wd/hub/session/:sessionId/elements"))
...
```

当这些 URL 有请求过来，AppiumServlet 会对它执行相关的处理。比如查找元素、输入、点击等操作。以查找元素为例，实现类里可以看到下面一段代码：

```
...
final String method = payload.getString("strategy");
final String selector = payload.getString("selector");
final String contextId = payload.getString("context");
...
```

通过这三个属性"strategy"、"selector"、"context" 来定位元素。在 Appium 对应的日志中可以看到这个操作。

```
2020-04-08 10:42:37:928 [HTTP] --> POST /wd/hub/session/f99fe38b-445b-45d2-bc
2020-04-08 10:42:37:929 [HTTP] {"using":"xpath",\
"value":"//*[@text=\"交易\"]"}
2020-04-08 10:42:37:930 [W3C (f99fe38b)] Calling \
AppiumDriver.findElement() with args: ["xpath","//*[@text=\"交易\"]","f99fe38
...
2020-04-08 10:42:37:931 [WD Proxy] Matched '/element' to \
command name 'findElement'
2020-04-08 10:42:37:932 [WD Proxy] Proxying [POST /element] to \
[POST http://127.0.0.1:8200/wd/hub/session/\
0314d14d-b580-4098-a559-602559cd7277/element] \
with body: {"strategy":"xpath","selector":\
"//*[@text=\"交易\"]","context":"","multiple":false}
...
2020-04-08 10:42:39:518 [W3C (f99fe38b)] Responding \
to client with driver.findElement() \
result: {"element-6066-11e4-a52e-4f735466cecf":\
"c57c34b7-7665-4234-ac08-de11641c8f56",\
"ELEMENT":"c57c34b7-7665-4234-ac08-de11641c8f56"}
2020-04-08 10:42:39:519 [HTTP] <-- POST /wd/hub/session/f99fe38b-445b-45d2-bc
```

上面代码，定位元素的时候会发送一个 POST 请求，Appium 会把请求转为 UiAutomatorV2 的定位，然后转发给 UiAutomatorV2。

扩展功能

在 FindElement.java 中实现了 findElement( ) 方法，如下图：

```
private Object findElement(By by) throws UiAutomator2Exception, UiObjectNotF
        refreshAccessibilityCache();
        if (by instanceof ById) {
            String locator = rewriteIdLocator((ById) by);
            return CustomUiDevice.getInstance().findObject(androidx.test.uiau
        } else if (by instanceof By.ByAccessibilityId) {
            return CustomUiDevice.getInstance().findObject(androidx.test.uiau
        } else if (by instanceof ByClass) {
            return CustomUiDevice.getInstance().findObject(androidx.test.uiau
        } else if (by instanceof By.ByXPath) {
            final NodeInfoList matchedNodes = getXPathNodeMatch(by.getElement
            if (matchedNodes.isEmpty()) {
                throw new ElementNotFoundException();
            }
            return CustomUiDevice.getInstance().findObject(matchedNodes);
        }
        ...
    }
```

findElement( ) 方法具体的提供了 ById、ByAccessibilityId、ByClass、ByXpath 等方法，可以扩展这部分功能，如果将来引申出来一些功能，比如想要通过图片、AI 定位元素，可以在上面的 findElement( ) 方法里面添加 `else if (by instanceof ByAI)` 方法，来创建新类型ByAI并且增加功能的实现。比如未来新增了 AI 来定位元素的功能，可以使用 AI 的插件（基于 nodejs 封装的一个插件）test.ai 插件（https://github.com/testdotai/appium-classifier-plugin）
用法：

```
driver.find_element('-custom', 'ai:cart');
```

项目构建与apk安装

完成代码的修改之后需要重新编译生成相应的 apk 文件，并放到 Appium 对应的目录下。

项目构建

Android Studio -> 项目 Gradle -> appium-uiautomator2-server-master -> Task-other 下。
分别双击 assembleServerDebug 与 assembleServerDebugAndroidTest 即可完成编译，编译完成会在目录下生成对应的两个 apk 文件。

- assembleServerDebugAndroidTest.apk

构建后 apk 所在目录：app/build/outputs/apk/androidTest/server/debug/appium-uiautomator2-server-debug-androidTest.apk 这个 apk 是个驱动模块，负责创建会话，安装

UiAutomator2-server.apk 到设备上，开启 Netty 服务。

- assembleServerDebug

构建后 apk 所在目录：app/build/outputs/apk/server/debug/appium-uiautomator2-server-v4.5.5.apk，这是服务器模块，当驱动模块初始化完毕，服务器就会监听 PC 端 Appium 发送过来的请求，将请求发送给真正底层的 UiAutomator2。
另外，也可以使用命令来进行构建：

```
gradle clean assembleE2ETestDebug assembleE2ETestDebugAndroidTest
```

将编译完成的 APK，覆盖 Appium 目录下对应的 APK 文件。需要先使用命令查找 Appium 安装目录下的 Uiautomator server 对应的 APK，MacOS 操作命令如下：

```
find /usr/local/lib/node_modules/appium -name "*uiautomator*.apk"
```

使用上面的命令会发现关于 Uiautomator 的两个 apk 文件，如下：

```
$ find /usr/local/lib/node_modules/appium -name \
"*uiautomator*.apk"
/usr/local/lib/node_modules/appium/node_modules\
/appium-uiautomator2-server/apks/\
appium-uiautomator2-server-v4.5.5.apk
/usr/local/lib/node_modules/appium/\
node_modules/appium-uiautomator2-server\
/apks/appium-uiautomator2-server-debug-androidTest.apk
```

将编译好的 APK 替换这个目录下的 APK 即可。
客户端会传递 Desired Capabilities 给 Appium Server 创建一个会话，Appium Server 会调用 appium-uiautomator2-driver 同时将 UiAutomator2 Server 的两个 apk 安装到测试设备上（也就是上面生成的两个 apk 文件）。