Appium 在移动自动化测试中的应用

Appium简介

Appium（http://appium.io/、https://github.com/appium/appium）是一个流行的开源App自动化测试工具，目前所支持的App类型包括iOS、Android、Windows以及Mac，在GitHub上收获的star数量已经超过5700个。

为了兼容众多的App类型，它实现了WebDriver规范，以JSON Wire Protocol的方式实现各种测试环境的初始化。

Appium基于Node.js技术实现了WebDriver规范，抽象出了测试各种平台App的统一接口，支持使用各种语言编写测试用例，包括Java、Python、JavaScript、C#等。另外，旧版本的Appium提供了针对各种操作系统下的GUI工具，简化它的使用；在较新版本的Appium中，我们可以使用图形化的Appium Desktop。

Appium架构与使用

关于WebDriver

WebDriver规范最初用于Web应用的自动化测试。该规范定义了一组与平台、语言无关的接口，包括发现和操作页面上的元素以及控制浏览器行为。

WebDriver的核心是通过findElement方法返回DOM对象（WebElement），通过WebElement可以对DOM对象进行操作（获取属性、触发事件等）。其中findElement方法进行元素查找所需要的元素定位器（Locator）支持ID、XPath、CSS、超链接文本等多种方式。倡导并实现该规范的Selenium已经是Web UI测试领域中当仁不让的标准配置了。

简而言之，WebDriver的主要作用就是初始化测试环境（打开浏览器）、定位目标元素（找到页面上需要操作的内容）和执行测试步骤（模拟点击、输入等UI操作）。

Appium的环境搭建

关于Appium的环境搭建，网上已经有不少的文章介绍了，这里不再赘述。

简而言之，如果使用图像化的界面的话，下载Appium Desktop（目前已支持Appium V1.6.5）；如果喜欢使用命令行的话，直接使用npm安装appium即可（可能需要翻墙或使用cnpm）。

Appium的理念与设计

根据官方文档的阐述，Appium遵循的理念如下：

1. 无需为了自动化，而重新编译或者修改你的应用。
2. 不必局限于某种语言或者框架来编写和运行测试脚本。
3. 一个移动自动化的框架不应该在接口上重复造轮子。
4. 无论是精神上，还是名义上，都必须开源。

为了实现上述的理念，Appium采用了一种扩展性非常强的设计，从而实现了多应用类型多客户端语言的支持。

首先，Appium是一个基于Node.js的HTTP服务器。

当使用“appium”命令启动之后，会看到如下的界面：

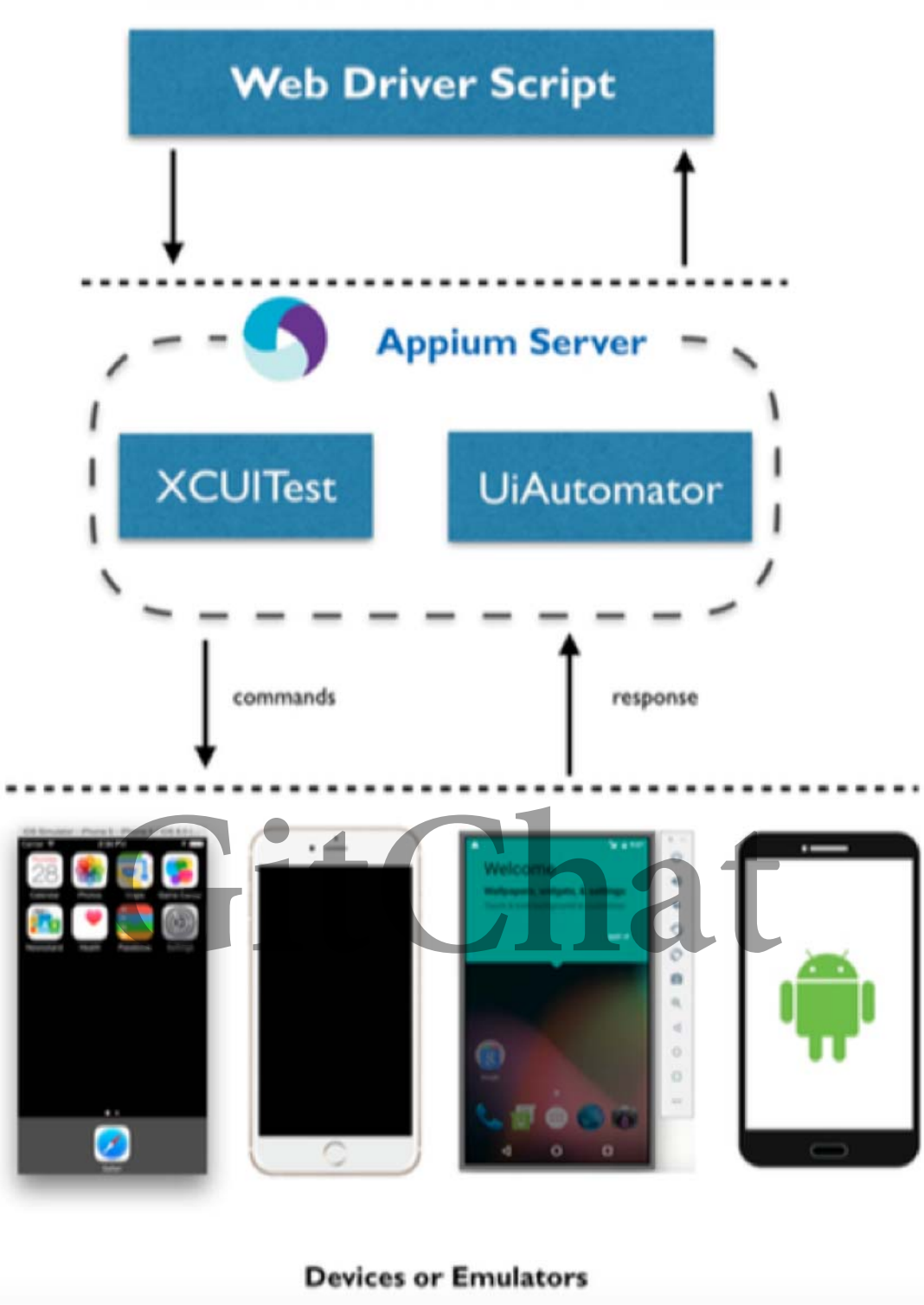
此时，已经在本机的4723端口上启动了appium服务，但是如果我们访问http://127.0.0.1:4723/wd/hub/sessions地址的话，显示结果如下：

这说明，appium已经启动，但是还没有创建session，而session又是什么呢？

Session是appium客户端与服务端进行交互的会话，有了这个会话，appium才能驱动设备执行各种UI操作。在创建session的时候，我们要提供很多初始化的信息，这些初始化信息包括：要基于什么平台进行测试、要测试哪个应用、要在哪台设备上执行测试等等。这些基本信息都会以HTTP POST请求发送给服务器，数据按照JSON格式进行组织，这些数据被称之为“**Desired Capabilities**”。服务器在接收到这种类型的请求之后，就会开启一个会话，对应会有一个会话的id，在调用session初始化的客户端代码中，我们可以得到一个Driver实例，后续的UI操作或页面结构的获取都可以通过这个Driver实例来执行。Driver继承自Selenium的RemoteWebDriver，以统一的接口实现元素的获取以及后续的UI操作。

Desired Capabilities是一组键值对的组合，针对不同的测试环境和测试需求，我们组装好这些键值对，发送给服务器端，据此生成一个测试会话。Desired Capabilities有一些通用的配置项，还有针对Android、iOS等平台的特殊配置，具体参见：http://appium.io/slate/en/master/?ruby#appium-server-capabilities。在后文中，会对其中的一些配置项进行讲解。需要注意的是，一个appium服务只能创建一个会话。

Appium的整体架构，可以如下图所示：



（图片来源“Mobile Test Automation with Appium”一书）

下面我们以Java语言的Appium客户端为例，介绍它的基本用法。

测试Android应用

要测试Android应用，我们首先要安装和配置Android SDK和Java运行环境。

使用Eclipse或IntelliJ IDEA创建测试项目，下载appium的Java客户端包并添加到classpath中，或者创建Maven项目并在pom.xml中添加如下依赖：

```
<dependency>
  <groupId>io.appium</groupId>
  <artifactId>java-client</artifactId>
  <version>4.1.0</version>
</dependency>
```

创建JUnit测试用例，基本代码如下所示：

```
public class AndroidContactsTest {

    private AppiumDriver driver;

    @Before
    public void setUp() throws Exception {
        File classpathRoot = new
File(System.getProperty("user.dir"));
        File appDir = new File(classpathRoot,
"apps/ContactManager");
        File app = new File(appDir, "ContactManager.apk");
        DesiredCapabilities capabilities = new
DesiredCapabilities();
        capabilities.setCapability("deviceName","Android");
        capabilities.setCapability("platformName","Android");
        capabilities.setCapability("platformVersion", "6.0");
        capabilities.setCapability("app", app.getAbsolutePath());
        driver = new AndroidDriver(new
URL("http://127.0.0.1:4723/wd/hub"), capabilities);
    }

    @Test
    public void addContact(){
        WebElement el =
                driver.findElement(By.xpath("//*[(@content-
desc='Add Contact' or @text='Add Contact')]"));
        el.click();
        List<WebElement> textFieldsList =

driver.findElementsByClassName("android.widget.EditText");
        textFieldsList.get(0).sendKeys("Some Name");
        textFieldsList.get(2).sendKeys("Some@example.com");
        driver.findElement(By.xpath("//*[(@content-desc='Save' or
@text='Save')]")).click();
        System.out.println(driver.getPageSource());
    }
    @After
    public void tearDown() throws Exception {
        driver.quit();
    }
```

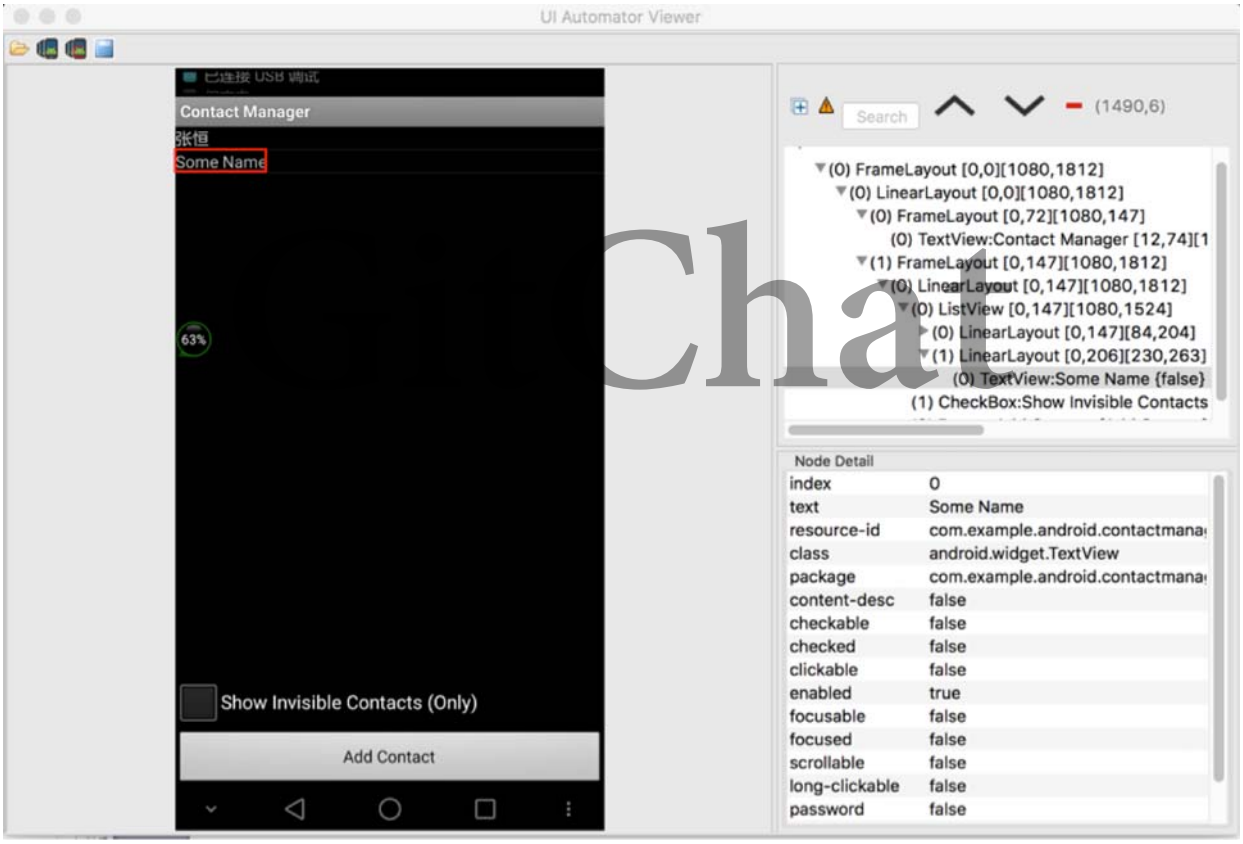
注：以上的代码，根据Appium的示例代码修改而来。

1. 在setUp方法中，我们需要通过DesiredCapabilities类指定初始化appium session的配置，包括platformName、app等信息，其中app指向了要测试应用的绝对路径，然后调用AndroidDriver的构造方法，并将服务器的地址http://127.0.0.1:4723/wd/hub传递过来，此时就能初始化session，并得到一个AppiumDriver的实例；

2. 在addContact方法中，我们就能通过这个Driver实例，获取页面元素，进而执行UI相关的操作。在这个过程中，我们可以通过xpath、classname等方式来获取页面元素。

3. 在测试执行完成的tearDown方法中，我们调用了driver的quit方法将session关掉。

在进行测试时，我们需要获取页面上各元素的xpath、文本等基本信息，可以借助Android SDK中自带的uiautomatorviewer工具进行查看：



测试iOS应用

测试iOS应用，必须要在Mac操作系统上执行，并且安装与测试设备相对应的Xcode版本和一些辅助的包，比如libimobiledevice、ideviceinstaller等。

在iOS 10以上的环境中，我们需要使用Xcode 8版本。因为在Xcode 8之后，Apple将原有的自动化测试工具UIAutomation移除掉了，只能使用XCode UITest作为测试工具，虽然Appium将这些差异进行了抽象，但是环境搭建方面估计需要大家多花费一些时间。具体的搭建过程，可以参考http://blog.csdn.net/wuxuehong0306/article/details/54377957这篇文章，在这里对原作者的贡献表示感谢。

在环境搭建完成之后，我们可以编写类似于Android的样例，只不过需要注意的是，在初始化driver时候，要这样声明DesiredCapabilities：

```
        DesiredCapabilities capabilities = new
DesiredCapabilities();
        capabilities.setCapability(CapabilityType.BROWSER_NAME,
"");
        capabilities.setCapability("deviceName", "iPhone 7");
        capabilities.setCapability("platformVersion", "10.2");
        capabilities.setCapability("platformName", "ios");
        capabilities.setCapability("udid", "XXXXX");
        capabilities.setCapability("app",
"/Users/levinzhang/Desktop/demo.ipa");
        capabilities.setCapability("automationName", "XCUITest");
        capabilities.setCapability("newCommandTimeout", 360000);
        driver = new IOSDriver(new
URL("http://127.0.0.1:4723/wd/hub"),
        capabilities);
```

在这里需要将platformName设置为iOS，automationName属性设置为“XCUITest”，同时还需要提供设备的唯一标识。

UI元素的获取以及测试步骤的执行与Android类似，在测试执行完成之后也需要调用Driver的quit方法，关闭当前的session。

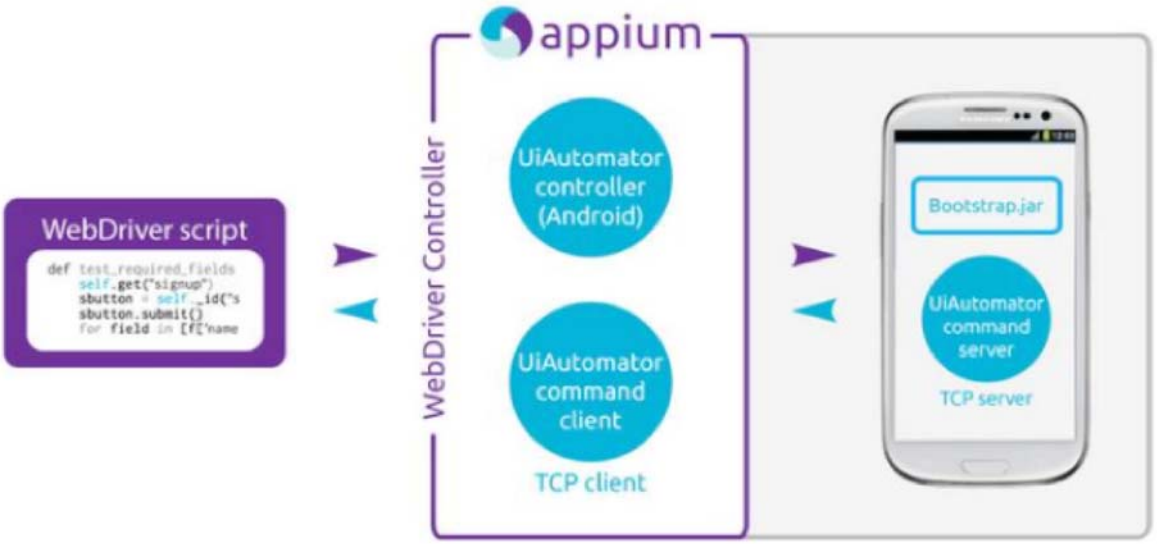
在UI元素的探查方面，Xcode并没有提供原生的工具来探查页面元素，我们可以使用Appium Desktop以及该文章（https://medium.com/@chenchaoyi/the-options-of-inspecting-ios-10-app-with-appium-1-6-534ba166b958）推荐的方式查看页面元素，当前，我们还可以通过driver.getPageSource()方法获取到的结果，自行解析页面结构。

Appium的实现原理

在了解完Appium的基本用法之后，我们看一下它在Android和iOS平台的实现原理。

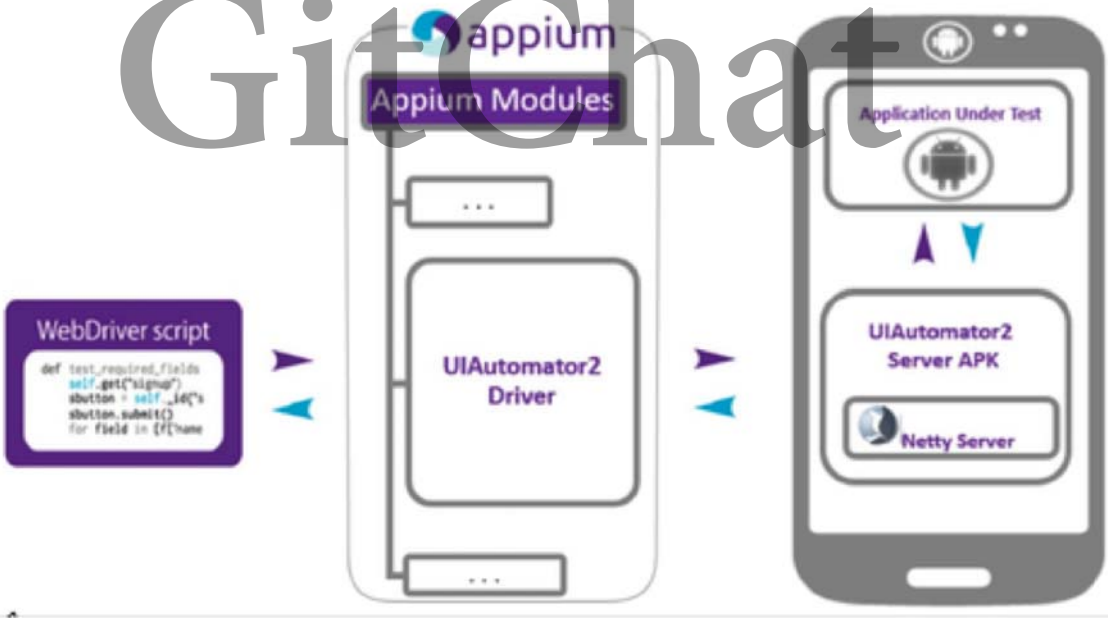
Android平台

在Android平台下，在4.1版本下底层使用的是seledroid技术，4.1以上版本使用uiautomator。使用uiautomator的实现原理如下图所示：



也就是将符合uiautomator测试规范的jar包推送至手机中，并开启一个TCP Server，通过端口转发接受来自WebDriver脚本的命令。

目前，Appium已经支持uiautomator 2，在采用uiautomator2底层方案的时候，实现原理如下图所示：

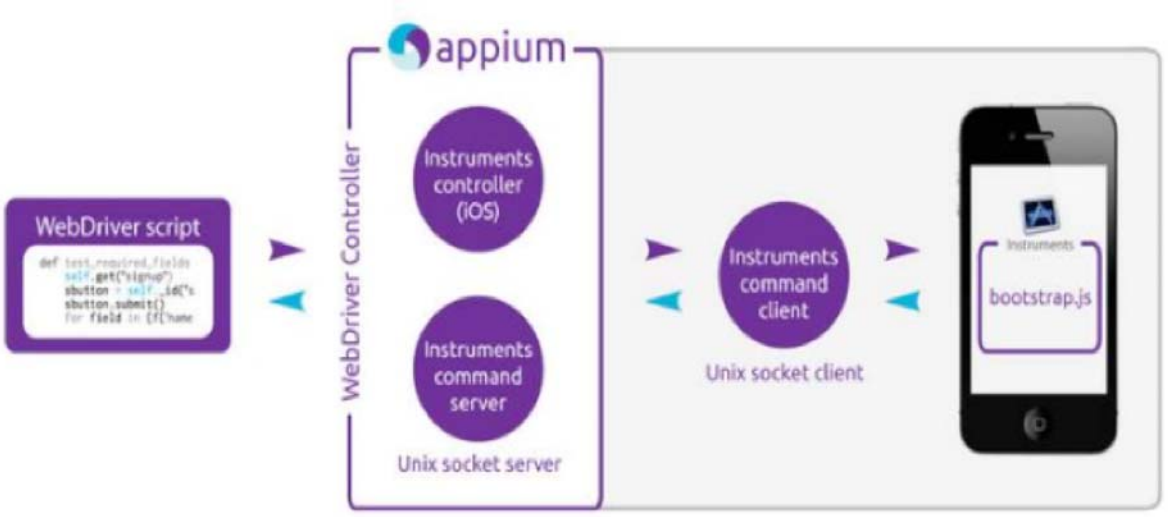


uiautomator 2方案与uiautomator的方案在实现原理上非常类似，差异有两点：第一，推送到设备中的不是jar包，而是apk文件，这是因为uiautomator 2本身集成的是instrumentation底层技术；第二，在这种方案中，appium采用了Netty构建运行于手机中的Server。

iOS

在iOS 10之前的版本中，Appium底层采用的是UIAutomation方案，在iOS 10之后，采用的是XCode UITest。

UIAutomation方案的实现原理，可以如下图所示:



核心原理是将一个符合UIAutomation语法的脚本推送到设备上，这个脚本使用一个核心的API performTaskWithPathArgumentsTimeout读取来自测试脚本中的测试命令。这个核心API实现了UIAutomation与外部程序的通信，从而能够读取用户发出的执行命令，关于该API，可以参考芈峮在一篇访谈中的介绍：http://www.infoq.com/cn/news/2014/04/interview-ios-test-guide-author。

在iOS 10以上的版本中，appium底层使用了XCode UITest测试方案，关于该技术的直接用法，这个项目介绍的比较详尽：https://github.com/joemasilotti/UI-Testing-Cheat-Sheet，读者可以学习参考。

Appium对XCode UITest的抽象直接复用了Facebook开源的WebDriverAgent（https://github.com/facebook/WebDriverAgent）项目。它的底层做法是链接XCTest.framework并调用Apple的API直接在设备上执行命令。

使用Appium所遇到的问题

我们在直接使用Appium时，遇到过一些问题，简单总结如下：

中文输入慢

在进行赋值操作时，Appium会使用一个自带的输入法模拟字符的输入。英文字母和数字的输入速度还是不错的，但是在遇到中文时，输入速度会非常慢，影响到用户的体验。针对这种情况，我们采用了复制粘贴的方式，控制设备上的粘贴板，实现了快速地输入。

点击位置的有效性问题

在获取到目标元素后，执行click操作时，在有些场景下，会出现点击无效的情况，这可能是因为Appium在执行点击操作时，点击的是元素的中心点。被测应用本身可能会因为

一些布局类的容器导致该区域的点击不一定有效。

遇到这种情况，比较好的做法是计算出目标元素的宽度、高度以及坐标值。基于一定的垂直和水平偏移量，计算出一个点击点，然后使用坐标执行真正的点击操作。

偶尔的弹出框影响测试进度

在测试步骤的执行过程中，有些应用偶尔会弹出一些提示框，比如抽奖、更新提示等。这样的弹出框会改变页面结构，造成目标元素无法按照原有的方式进行获取，从而影响到整个测试用例的稳定执行。
针对这种情况，在测试用例的编写时，我们可以加一些可选的校验类的步骤，首先判断弹出框是否存在，如果存在的话，自动将其点掉，然后再执行后续的测试步骤。

大规模测试时，还需为Appium增加哪些辅助功能

Appium作为底层的测试技术，具有很高的技术领先性，但是总体而言，它的学习成本较高（需要测试人员进行代码的编写）、环境搭建也稍显复杂（对于iOS尤为如此）。另外，在组织中实际使用时，并不像我们编写一个demo实例那样简单，每个组织都有自己的测试流程和规模化管理手段。如果想成为工业级的测试工具，页面对象的管理和测试用例的组织都是需要解决的重要问题。

页面对象的维护

Appium在执行时，我们首先需要获取页面结构，得到各个页面元素的xpath等信息，然后才能执行后续的UI操作。如前文所述，我们可以通过一些工具，探查页面元素。借助Appium Driver的getPageSource方法，也能够得到页面结构的XML格式描述，对其进行解析，可以形成整个页面树状的结构。

在编写单个测试用例时，我们可以这样做，但是如果测试用例数量巨大，这样的过程就会非常繁琐。另外，移动应用的迭代和变更非常频繁，如果只是根据xpath路径进行元素定位，很容易出现测试脚本维护的噩梦，开发稍微改变页面结构，就会造成测试脚本的大量修改。

针对这种状况，可以采用PageObject的模式，实现页面对象的复用，另外，在获取页面对象时，通过getPageSource方法记录尽可能多的信息，比如xpath、文本、内容描述、id等信息，在测试实际执行时，组合使用这些信息，进行元素查找，从而尽可能地提升测试脚本的稳定性。

测试用例的管理

作为测试执行的工具，Appium不具备测试用例管理、测试场景组合等功能，但是这都是实现测试规模化管理所必备的特性。

因此，如果想要实现复杂移动应用的测试，针对测试用例的组织，还需要测试团队进行设计和统一化的管理。

输入值的参数化

在复杂的应用中，我们经常需要针对某一套测试脚本，使用多套数据测试各种边界情景，比如，我们可能想要测试在不同金额范围内的银行转账功能。
这样的话，就需要将输入值提取为参数。每次执行时，根据不同的参数进行测试用例的执行，Appium本身不会提供这样的功能，这需要测试人员设法去解决的问题。

测试过程的管理

对于一些复杂的流程，我们有可能希望长时间循环执行，比如员工下班之前，将测试任务分发到多台设备上依次或并行执行，等到第二天早上查看结果，确定应用的稳定性。

对于这种常见的调度执行功能，作为一个测试执行的工具，Appium本身难以直接实现，这也需要测试人员结合组织的实际情况，通过与缺陷管理和持续集成等工作的协同使用，做出符合需求的实现。

在上面的内容中，我们简单介绍了Appium的功能和实现原理，以及在真正使用时，我们还需要解决哪些关键的问题。

综上所述，Appium是一项强大的移动测试管理工具，但是在将其真正用到较大规模的团队中时，还需要我们开展一定的外围工作，使其更适合团队和组织的需求。