中文Appium API 文档

第一章：关于appium

1.1 appium客户端

客户端类库列表及Appium服务端支持

这些类库封装了标准Selenium客户端类库，为用户提供所有常见的JSON 格式selenium命令以及额外的移动设备控制相关的命令，如多点触控手势和屏幕朝向。

Appium客户端类库实现了Mobile JSON Wire Protocol（一个标准协议的官方扩展草稿）和W3C Webdriver spec（一个传输不可预知的自动化协议，该协议定义了MultiAction 接口）的元素。

Appium 服务端定义了官方协议的扩展，为Appium 用户提供了方便的接口来执行各种设备动作，例如在测试过程中安装/卸载app。这就是为什么我们需要Appium 特定的客户端，而不是通用的Selenium 客户端。当然，Appium 客户端类库只是增加了一些功能，而实际上这些功能就是简单的扩展了Selenium 客户端，所以他们仍然可以用来运行通用的selenium会话。

语言/框架 Github版本库以及安装指南

Ruby https://github.com/appium/ruby\_lib

Python https://github.com/appium/python-client

Java https://github.com/appium/java-client

JavaScript (Node.js) https://github.com/admc/wd

Objective C https://github.com/appium/selenium-objective-c

PHP https://github.com/appium/php-client

C# (.NET) https://github.com/appium/appium-dotnet-driver

RobotFramework https://github.com/jollychang/robotframework-appiumlibrary

1.2 appium介绍

Appium 介绍

Appium 是一个自动化测试开源工具，支持 iOS 平台和 Android 平台上的原生应用，web 应用和混合应用。

所谓的“移动原生应用”是指那些用 iOS 或者 Android SDK 写的应用。所谓的“移动 web 应用”是指使用移动浏览器访问的应用（Appium 支持 iOS 上的 Safari 和 Android 上的 Chrome）。所谓的“混合应用”是指原生代码封装网页视图——原生代码和 web 内容交互。比如，像 Phonegap，可以帮助开发者使用网页技术开发应用，然后用原生代码封装，这些就是混合应用。

重要的是，Appium 是一个跨平台的工具：它允许测试人员在不同的平台（iOS，Android）使用同一套API来写自动化测试脚本，这样大大增加了 iOS 和 Android 测试套件间代码的复用性。

想知道 Appium 如何支持平台，版本和自动化形态的详细信息，请参见platform support doc。

Appium 的理念

为了满足移动自动化需求，Appium 遵循着一种哲学，重点体现于以下4个需求：

你无需为了自动化，而重新编译或者修改你的应用。

你不必局限于某种语言或者框架来写和运行测试脚本。

一个移动自动化的框架不应该在接口上重复造轮子。（移动自动化的接口应该统一）

无论是精神上，还是名义上，都必须开源。

Appium 设计

那么 Appium 架构是如何实现这个哲学呢？为了满足第一条，Appium 真正的工作引擎其实是第三方自动化框架。这样，我们就不需在你的应用里植入 Appium 相关或者第三方的代码。这意味着你测试使用的应用与最终发布的应用并无二致。我们使用以下的第三方框架：

iOS: 苹果的 UIAutomation

Android 4.2+: Google's UiAutomator

Android 2.3+: Google's Instrumentation. (Instrumentation由单独的项目Selendroid提供支持 )

为了满足第二点，我们把这些第三方框架封装成一套 API，WebDriver API.WebDriver（也就是 "Selenium WebDriver"） 指定了客户端到服务端的协议。

(参见 JSON Wire Protocol)。使用这种客户端-服务端的架构，我们可以使用任何语言来编写客户端，向服务端发送恰当的 HTTP 请求。

目前已经实现了大多数流行语言版本的客户端，这意味着你可以使用任何测试套件或者测试框架。客户端库就是简单的HTTP 客户，可以以任何你喜欢的方式潜入你的代码。换句话说，Appium 和 WebDriver 客户端不是技术意义上的“测试框架”，而是“自动化库”。你可以在你的测试环境中随意使用这些自动化库！

事实上 WebDriver 已经成为 web 浏览器自动化的标准，也成了 W3C 的标准 —— W3C Working Draft。我们又何必为移动做一个完全不同的呢？所以我们扩充了WebDriver 的协议，在原有的基础上添加移动自动化相关的 API 方法，这也满足了第三条理念。

第四条就不用说了，Appium 是开源的。

Appium 概念

C/S 架构<br/>

Appium 的核心是一个 web 服务器，它提供了一套 REST 的接口。它收到客户端的连接，监听到命令，接着在移动设备上执行这些命令，然后将执行结果放在 HTTP响应中返还给客户端。事实上，这种客户端/服务端的架构给予了许多的可能性：比如我们可以使用任何实现了该客户端的语言来写我们的测试代码。比如我们可以把服务端放在不同

的机器上。比如我们可以只写测试代码，然后使用像 Sauce Labs 这样的云服务来解释命令。

自动化始终围绕一个session进行，客户端初始化一个seesion（会话）来与服务端交互，不同的语言有不同的实现方式，但是他们最终都是发送为一个POST请求给服务端，请求中包含一个JSON对象，被称作“desired capabilities”。此时，服务端就会开启一个自动化的 session，然后返回一个 session ID，session ID将会被用户发送后续的命令。

Desired capabilities 是一些键值对的集合 (比如，一个 map 或者 hash），客户端将这些键值对发给服务端，告诉服务端我们想要怎么测试。比如，我们可以把platformName capability 设置为 iOS，告诉 Appium 服务端，我们想要一个iOS 的 session，而不是一个 Android 的。我们也可以设置 safariAllowPopups capability 为 true，确保在 Safari 自动化 session 中，我们可以使用 javascript 来打开新窗口。参见 capabilities 文档，查看完整的 capabilities 列表。

Appium Server

Appium server 是用 Node.js 写的。我们可以用源码编译或者从 NPM 直接安装。

Appium 服务端有很多语言库 Java, Ruby, Python, PHP, JavaScript 和 C#，这些库都实现了

Appium 对 WebDriver 协议的扩展。当使用 Appium 的时候，你只需使用这些库代替常规的 WebDriver 库就可以了。

你可以从这里看到所有的库的列表。

我们提供了 GUI 封装的 Appium 服务端下载，它封装了运行 Appium服务端的所有依赖，而不需要担心怎样安装Node.js。其中还包括一个Inspector工具，可以帮助你检查应用的界面层级，这样写测试用例时更方便。

Getting Started

恭喜！你现在有足够的知识来使用 Appium 了。 来我们回到 getting started doc 继续了解更加

细节的需求和指南。

第二章：进阶指南

2.1 selenium配置

Selenium Grid

使用 <b>"--nodeconfig"</b> 服务器参数，你可以在本地 selenium grid 里注册你的 appium 服务器。

> node . -V --nodeconfig /path/to/nodeconfig.json

在 node 的配置文件里，你需要定义 <b>"browserName"</b>，<b>"version"</b> 和 <b>"platform"</b>。

基于这些参数，selenium grid 会将你的测试定向到正确的设备上去。你还需要配置你的 <b>host</b> 详细信息和

<b>selenium grid</b> 的详细信息。你可以在 <a href="http://code.google.com/p/selenium/source/browse/java/server/src/org/openqa/grid/common/defaults/GridParameters.properties">这里</a> 找到详细的参数列表和描述信息。

一旦你启动了 appium 服务器并且在 grid 里注册了信息，你会在 grid 控制台发现你的设备：

"http://<b><grid-ip-adress></b>:<b><grid-port></b>/grid/console"

Grid 配置文件例子

{

"capabilities":

[

{

"browserName": "<e.g.\_iPhone5\_or\_iPad4>",

"version":"<version\_of\_iOS\_e.g.\_6.1>",

"maxInstances": 1,

"platform":"<platform\_e.g.\_MAC\_or\_ANDROID>"

}

],

"configuration":

{

"cleanUpCycle":2000,

"timeout":30000,

"proxy": "org.openqa.grid.selenium.proxy.DefaultRemoteProxy",

"url":"http://<host\_name\_appium\_server\_or\_ip-address\_appium\_server>:<appium\_port>/wd/hub",

"maxSession": 1,

"register": true,

"registerCycle": 5000,

"hubPort": <grid\_port>,

"hubHost": "<Grid\_host\_name\_or\_grid\_ip-address>"

}

}

可以在 <a href="http://selenium.googlecode.com/git/docs/api/java/org/openqa/selenium/Platform.html">这里</a>查看有效的 platform 参数。

如果没有给出url、host和 port，配置会自动指向本地：whatever-port-Appium-started-on。

如果你的Appium服务和Selenium Grid服务没有运行在同一台机器上，为确保Selenium Grid连接正常，请在你的host & url docs上使用外部其它名称或IP地址，而非localhost 和 127.0.0.1

2.2 自动化混合应用

自动化混合应用

Appium 其中一个理念就是你不能为了测试应用而修改应用。为了符合这个方法学，我们可以使用 Selenium 测试传统 web 应用的方法来测试混合 web 应用 (比如，iOS 应用里的元素 "UIWebView" )，这是有可能的。这里会有一些技术性的复杂，Appium 需要知道你是想测试原生部分呢还是web部分。幸运的是，我们还能遵守 WebDriver 的协议。

在你的 Appium 测试里，你需要以下几步来和 web 页面交涉：

前往到应用里 web 视图激活的部分。

调用 GET session/:sessionId/window\_handles

这会返回一个我们能访问的 web 视图的 id 的列表。

使用你想访问的这个 web 视图的 id 作为参数，调用 POST session/:sessionId/window

(这会将你的 Appium session 放入一个模式， 在这个模式下，所有的命令都会被解释成自动化web视图而不是原生的部分。比如，当你运行 getElementByTagName，它会在 web 视图的 DOM 上操作，而不是返回 UIAElements。当然，一个 Webdriver 的方法只能在一个上下文中有意义，所以如果在错误的上下文，你会收到错误信息。)

如果你想停止 web 视图的自动化，回到原生部分，你可以简单地使用 execute\_script 调用 "mobile: leaveWebView" 方法来离开 web 层。

在 iOS 真机上运行

appium 使用一个远程调试器建立连接来实现和 web 视图的交互。当在模拟器上执行下面例子的时候，我们可以直接建立连接，因为模拟器和 appium 服务器在同一台机器上。

当在真机上运行用例时，appium 无法直接访问 web 视图，所以我们需要通过 USB 线缆来建立连接。我们使用 ios-webkit-debugger-proxy建立连接。

使用 brew 安装最新的 ios-webkit-debug-proxy。在终端运行一下命令:

# 如果你没有安装 brew 的话，先安装 brew。

> ruby -e "$(curl -fsSL https://raw.github.com/mxcl/homebrew/go/install)"

> brew update

> brew install ios-webkit-debug-proxy

你也可以通过 git 克隆项目来自己安装最新版本：

# Please be aware that this will install the proxy with the latest code (and not a tagged version).

> git clone https://github.com/google/ios-webkit-debug-proxy.git

> cd ios-webkit-debug-proxy

> ./autogen.sh

> ./configure

> make

> sudo make install

一旦安装好了， 你就可以启动代理：

# 将udid替换成你的设备的udid。确保端口 27753 没有被占用

# remote-debugger 将会使用这个端口。

> ios\_webkit\_debug\_proxy -c 0e4b2f612b65e98c1d07d22ee08678130d345429:27753 -d

<b>注意：</b> 这个 ios-webkit-debug-proxy 需要 <b>"web inspector"</b> 打开着以便建立连接。在 <b> settings > safari > advanced </b> 里打开它。请注意 web inspector <b>在 iOS6 时候加入的</b> 以前的版本没有。

Wd.js Code example

// 假设我们已经有一个初始化好了的 `driver` 对象。

driver.elementByName('Web, Use of UIWebView', function(err, el) { // 找到按钮，打开 web 视图

el.click(function(err) { // 引导到 UIWebView

driver.windowHandles(function(err, handles) { // 得到能访问的视图列表。

driver.window(handles[0], function(err) { // 因为只有一个，所以选择第一个。

driver.elementsByCss('.some-class', function(err, els) { // 通过 css 拿到元素。

els.length.should.be.above(0); // 肯定有元素。

els[0].text(function(elText) { // 得到第一个元素的文本。

elText.should.eql("My very own text"); // 比较匹配文本。

driver.execute("mobile: leaveWebView", function(err) { // 离开web视图上下文。

// 如果你想的话，做一些原生应用的操作。

driver.quit(); // 退出。

});

});

});

});

});

});

});

想看到具体的上下文，请看该node 的例子

\*我们正在完善 web 视图下面的方法。加入我们！

Wd.java 代码例子

//配置 webdriver 并启动 webview 应用。

DesiredCapabilities desiredCapabilities = new DesiredCapabilities();

desiredCapabilities.setCapability("device", "iPhone Simulator");

desiredCapabilities.setCapability("app", "http://appium.s3.amazonaws.com/WebViewApp6.0.app.zip");

URL url = new URL("http://127.0.0.1:4723/wd/hub");

RemoteWebDriver remoteWebDriver = new RemoteWebDriver(url, desiredCapabilities);

// 切换到最新的web视图

for(String winHandle : remoteWebDriver.getWindowHandles()){

remoteWebDriver.switchTo().window(winHandle);

}

//在 guinea-pig 页面用 id 和 元素交互。

WebElement div = remoteWebDriver.findElement(By.id("i\_am\_an\_id"));

Assert.assertEquals("I am a div", div.getText()); //验证得到的文本是否正确。

remoteWebDriver.findElement(By.id("comments")).sendKeys("My comment"); //填写评论。

//离开 webview，回到原生应用。

remoteWebDriver.executeScript("mobile: leaveWebView");

//关闭应用。

remoteWebDriver.quit();

Wd.rb cucumber 的例子

TEST\_NAME = "Example Ruby Test"

SERVER\_URL = "http://127.0.0.1:4723/wd/hub"

APP\_PATH = "https://dl.dropboxusercontent.com/s/123456789101112/ts\_ios.zip"

capabilities =

{

'browserName' => 'iOS 6.0',

'platform' => 'Mac 10.8',

'device' => 'iPhone Simulator',

'app' => APP\_PATH,

'name' => TEST\_NAME

}

@driver = Selenium::WebDriver.for(:remote, :desired\_capabilities => capabilities, :url => SERVER\_URL)

## 这里切换到最近一个窗口是因为在我们的例子里这个窗口一直是 webview。其他的用例里，你需要自己指定。

## 运行 @driver.window\_handles，查看 appium 的日志，找出到底哪个窗口是你要的，然后找出相关的数字。

## 然后用 @driver.switch\_to\_window(number)，切换过去。

Given(/^I switch to webview$/) do

webview = @driver.window\_handles.last

@driver.switch\_to.window(webview)

end

Given(/^I switch out of webview$/) do

@driver.execute\_script("mobile: leaveWebView")

end

# 你可以使用 CSS 选择器在你的 webview 里来选择元素

And(/^I click a webview button $/) do

@driver.find\_element(:css, ".green\_button").click

end

用 ruby 调试 web 视图：

我在我的帮助类里创建了一个快速方法来定位web元素，无论它在哪一个窗口视图。

（这非常有帮助，特别是你的 webview 的 id 变化或者你用同一份代码来测试 Android 和 iOS。）

https://gist.github.com/feelobot/7309729

自动化混合 Android 应用

Appium 通过 Chromedriver 内建混合应用支持。Appium 也可以使用 Selendroid 做为 4.4 之前的设备对 webview 支持的背部引擎。（你需要在 desired capability 里指定 "device": "selendroid"）。然后：

前往你应用里 web 视图激活的部分。

用 "WEBVIEW" 做窗口句柄调用 POST session/:sessionId/window ， 比如 driver.window("WEBVIEW")。

(这会将你的 Appium session 放入一个模式， 在这个模式下，所有的命令都会被解释成自动化web视图而不是原生的部分。比如，当你运行 getElementByTagName，它会在 web 视图的 DOM 上操作，而不是返回 UIAElements。当然，一个 Webdriver 的方法只能在一个上下文中有意义，所以如果在错误的上下文，你会收到错误信息。)

如果要停止web上下文里的自动化，回到原生部分的自动化，简单地使用 "NATIVE\_APP" 调用 window 方法。比如 driver.window("NATIVE\_APP")。

注意：我们可以像上面说的，使用同样的策略。然而，Selendroid 使用 WEBVIEW/NATIVE\_APP 窗口设置策略。 Appium 常规的混合支持也使用这种策略。

Wd.js 代码例子

// 假设我们已经初始化了一个 `driver` 实例。

driver.window("WEBVIEW", function(err) { // 选择唯一的 WebView

driver.elementsByCss('.some-class', function(err, els) { // 通过 CSS 取得元素

els.length.should.be.above(0); // 验证元素存在

els[0].text(function(elText) { // 得到第一个元素的文本

elText.should.eql("My very own text"); // 验证文本内容

driver.window("NATIVE\_APP", function(err) { // 离开 webview 上下文

// 可以做些原生应用的测试

driver.quit(); // 关闭 webdriver

});

});

});

});

Wd.java 代码例子

//配置 webdriver 并启动 webview 应用。

DesiredCapabilities desiredCapabilities = new DesiredCapabilities();

desiredCapabilities.setCapability("device", "Selendroid");

desiredCapabilities.setCapability("app", "/path/to/some.apk");

URL url = new URL("http://127.0.0.1:4723/wd/hub");

RemoteWebDriver remoteWebDriver = new RemoteWebDriver(url, desiredCapabilities);

// 切换到最新的web视图

remoteWebDriver.switchTo().window("WEBVIEW");

//在 guinea-pig 页面用 id 和 元素交互。

WebElement div = remoteWebDriver.findElement(By.id("i\_am\_an\_id"));

Assert.assertEquals("I am a div", div.getText()); //验证得到的文本是否正确。

remoteWebDriver.findElement(By.id("comments")).sendKeys("My comment"); //填写评论。

//离开 webview，回到原生应用。

remoteWebDriver.switchTo().window("NATIVE\_APP");

//关闭应用。

remoteWebDriver.quit();

2.3 用例迁移

把appium 0.18.x上的测试用例集迁移到appium1.x上

Appium 1.0 已经从先前的版本中移除了一部分过时的特性, 这个指导文档会帮助你了解使用Appium 1.0需要做的具体改变.

新的客户端库

你需要关注的最大的改变是利用新的appium的client libraries来替换原生的WebDriver ciients. 访问Appium client list 去寻找符合你自己编程语言的客户端库吧. 在每个客户端的网站上都可以找到用于集成到你代码中的依赖库相关介绍和下载

基本上, 你需要做如下的改变(以Python作为例子)

用

from appium import webdriver

替换原来的:

from selenium import webdriver

新的适配参数

下面的适配参数将不再使用

\* device

\* version

取而代之的是利用下面的配置参数

platformName ("iOS" 或者 "Android")

platformVersion (你希望测试的os版本)

deviceName (你想用的设备, 比如 "iPhone Simulator")

automationName ("Selendroid" 如果你想使用Selendroid的话, 否则可以省略)

app 配置参数保持不变, 但是特指非浏览器的app, 如果你想使用类似Safari或者Chrome这样的浏览器, 你需要设置browserName. 这代表app和browserName是互斥的.

我们把appium的配置参数都规范为驼峰拼写法(camelCase), 这代表着原来的app-package或者app-wait-activity现在会变成appPackage和appWaitActivity. 当然目前android的app package和activity都已经是自动探测了, 大部分情况下你可以省略这两个配置项.

新的定位方式

我们已经移除了下面的定位方式

name

tag name

我们增加了accessibility\_id定位方法去做过去name做的事情. 具体的细节还得跟你使用的Appium客户端库有关.

tag name已经被替换为class name. 所以想通过UI的类型来定位某个元素, 你需要使用class name定位方式

关于class name和xpath的定位方式: 现在需要使用完整的全类名, 这意味着如果你有一个如下的定位用的xpath

//table/cell/button

现在需要改成

//UIATableView/UIATableCell/UIAButton

如果是android的话, button需要改变成android.widget.Button

我们也增加了如下的定位方式

-ios uiautomation

-android uiautomator

根据你使用的客户端去相应的使用新的定位方式

使用xml, 不再是json了

App source方法先前返回JSON, 现在修改成返回XML. 所以如果你有代码是依赖解析app source的, 那么就需要更新

通过context支持混合应用, 不再是window了

以前混合app的切换支持是通过"windows"

window\_handles

window

switch\_to.window

现在Appium支持"context" 概念了, 要获得当前环境下所有的上下文(contexts), 或者特定的context, 你可以用

driver.contexts

current = driver.context

在这些context之间切换, 可以使用

driver.switch\_to.context("WEBVIEW")

没有了execute\_script("mobile: xxx")

所有的mobile:方法都已经被移除, 并且被替换为appium client libraries的原生方法. 这意味着如果一个方法调用原来的方式是

driver.execute("mobile: lock", [5])

现在需要更新为

driver.lock(5)

在这个地方lock已经变成了原生的客户端方法. 当然具体的调用细节在不同的客户端库中的实现可能会有所差别.

特别需要注意的是, 手势(gesture)方法已经被替换为TouchAction / MultiAction API, 它允许更强大通用的组合手势的自动化. 可以参考你的客户端库的具体用法.

这就是全部啦, 祝迁移愉快

2.4 appium设置

Settings

Settings 是 Appium 引入的一个新的概念。 它目前还没有被纳入 Mobile JSON Wire Protocol 及 Webdriver 标准之中。

Settings 是一种用来配置 Appium 服务器的方式。

Settings 有以下特点：

- 可变的，它在同一会话中是可以被修改的。

- 唯一的，它在被测应用会话中是唯一的。 它在每创建一个新会话时会被重置。

- 可控的，它在自动化测试过程中控制着 Appium 服务器的运行。 它们不会被用来控制被测应用或被测终端。

在 Android 环境中 以 ignoreUnimportantViews 设置举例，该参数在 Android 环境中可以被设置成忽略所有与当前视图无关的元素，它将使测试过程更加有效率。 然而当我们希望能够访问被忽略的元素时，我们必须在将 ignoreUnimportantViews 设置成 true 后，重新修改成 false 。

另外也可以通过 Settings 配置让 Appium 忽略所有当前不可见的元素。

Settings 可以通过以下 API 方法进行配置：

POST /session/:sessionId/appium/settings

以 JSON 格式提交 key:value 键值对形式的Settings配置。

{

settings: {

ignoreUnimportantViews : true

}

}

GET /session/:sessionId/appium/settings

以 JSON 格式返回当前所有 Settings 配置。

{

ignoreUnimportantViews : true

}

其它 Settings 配置参考

"ignoreUnimportantViews" -该参数值为 true 或 false。 如果你希望能够尽量减少测试过程的交互确认过程，或希望测试脚本能更快的执行，可以在 Android 终端环境下使用 setCompressedLayoutHeirarchy() 参数。它将忽略所有被标记为 IMPORTANT\_FOR\_ACCESSIBILITY\_NO 或 IMPORTANT\_FOR\_ACCESSIBILITY\_AUTO（以及那些被认为不是很重要的系统元素）的元素。

第三章：appium 设置

3.1 加速器管理

Intel® 硬件加速器管理

如果你发现android模拟器太慢, 并且你的系统运行在Intel® 的cpu上. 那么你可以尝试下HAXM, HAXM能够让你充分利用硬件虚拟化技术来加速android模拟器。

要安装HAXM, 你可以打开Android SDK Manager, 你可以在Extras中发现这个安装选项；

你可以在Intel官方网站找到所有相关的文档；

这将需要x86的模拟镜像；

利用Intel的包来安装HAXM; Android SDK Manager有时候会安装不成功，这主要取决于你安装的版本是否兼容。

3.2 android 设置

Android Setup

使用前，你需要安装node.js(版本大于等于0.10)。 请参照 instructions for your flavor of linux。

当node.js安装成功后，请安装 Android SDK。

运行'android' tool(位于SDK，tool文件目录下)。

运行'android' tool 来安装大于等于Level 17的API。

(如果你想从Appium的源码来运行，可在真机或者模拟器上用 Apache Ant 来编译bootstrap jar包)。

最后，将环境变量$ANDROID\_HOME设置为 Android SDK 的路径。例如,如果你将Android SDK 解压到 /usr/local/adt/，你需要把这个路径加到你的shell环境变量中去：

export ANDROID\_HOME="/usr/local/adt/sdk"

现在就可以启动Appium了！如果你在源码中运行Appium请运行

./reset.sh --android 版本从Appium checkout会安装所有的依赖。

老版本的额外安装

当android的版本是2.3到4.1的时候，appium用的是selendroid。 当它检测到时低版本时，它会自动应用Selendroid。但是需要配置一些额外的设置如果从source运行。

已经安装 Maven 3.1.1 或更新 (mvn)

运行 ./reset.sh --selendroid 从checkout的Appium源码

（运行Appium Android 测试）

在Linux上运行，启动一个API大于等于level17的AVD。 在源文件目录下运行 (appium) 在安装好 NPM, 或者 node。如果你选择的是从源代码方式运行。

参照 server documentation 来了解所有命令和参数。

注意

Android 加速模拟器需要存在，它有自己的局限性，如果想了解更多，请看这里 page。

如果你想运行任何Appium的测试，或者任何强大的命令，确保你的 hw.battery=yes 在 AVD's config.ini文件中。

Selendroid 需要你APP中的如下权限： <uses-permission android:name="android.\*\*permission.INTERNET"/>, 如果你在使用selendroid或者低版本的android(如版本2.3到4.1)，请确保你的App已设置internet权限。

3.3 部署iOS app 到手机上

部署iOS app 到手机上

准备在真机上执行appium测试, 需要如下准备：

用特殊的设备参数来构建app

使用 fruitstrap， 这是一个第三方程序，可以用来部署你构建的app到手机上

Xcodebuild 命令的参数：

新的参数运行指定设置. 参考 developer.apple.com：

xcodebuild [-project projectname] [-target targetname ...]

[-configuration configurationname] [-sdk [sdkfullpath | sdkname]]

[buildaction ...] [setting=value ...] [-userdefault=value ...]

这有一个资料来参考可用的设置

CODE\_SIGN\_IDENTITY (Code Signing Identity)

介绍: 标识符，指定一个签名。

例如: iPhone Developer

PROVISIONING\_PROFILE 已经从可用的的命令中消失了，但还是有必要设置的。

在xcodebuild命令中设置 "CODE\_SIGN\_IDENTITY" & "PROVISIONING\_PROFILE"：

xcodebuild -sdk <iphoneos> -target <target\_name> -configuration <Debug> CODE\_SIGN\_IDENTITY="iPhone Developer: Mister Smith" PROVISIONING\_PROFILE="XXXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXX"

成功的话， app会构建到如下目录 <app\_dir>/build/<configuration>-iphoneos/<app\_name>.app

用Fruitstrap进行部署

clone一个fruitstrap的fork版本在ghughes version ，这个已经不再维护. 已确认该fork可用unprompted fork, 但是其它的据说也可用。

clone成功的话， 执行 make fruitstrap

然后, 然后复制生成的 fruitstrap到app的所在的目录或上级目录下。

运行fruitstrap 通过输入以下命令 (命令是否可用依赖于你fork的 fruitstrap)：

./fruitstrap -d -b <PATH\_TO\_APP> -i <Device\_UDID>

如果是为了持续集成，你可以发现很有用的方法来记录fruitstrap命令行和日志文件中的记录， 像这样：

./fruitstrap -d -b <PATH\_TO\_APP> -i <Device\_UDID> 2>&1 | tee fruit.out

在node服务启动前fruitstrap进行需要被结束， 一个方法是扫描fruitstrap的输出来得知app完成启动。 有一个有效的方法是通过一个Rakefile 和一个 go\_device.sh 脚本：

bundle exec rake ci:fruit\_deploy\_app | while read line ; do

echo "$line" | grep "text to identify successful launch"

if [ $? = 0 ]

then

# Actions

echo "App finished launching: $line"

sleep 5

kill -9 `ps -aef | grep fruitstrap | grep -v grep | awk '{print $2}'`

fi

done

一旦fruitstrap的进程被结束, node 服务就可以启动并且appium测试可以被执行!

下一步:

在真机上运行appium

3.4 Android并发测试

Android并发测试

Appium提供了在一台设备上启动多个Android会话的方案，而这个方案需要你输入不同的指令来启动多个Appium服务来实现。

启动多个Android会话的重要指令包括：

-p Appium的主要端口

-U 设备id

-bp Appium bootstrap端口

--chromedriver-port chromedriver端口（当使用了webviews或者chrome）

--selendroid-port selendroid端口（当使用了selendroid）

更多参数的解释详见 here。

如果我们有两台设备，设备ID分别为43364和32456，我们应该用下面的命令启动来两个不同的Appium服务：

node . -p 4492 -bp 2251 -U 32456

node . -p 4491 -bp 2252 -U 43364

只要你的Appium和Appium bootstrap端口介于0和65536即可，并且保证是两个不同的端口以便两个Appium服务不会监听相同的端口。确认你的-u参数绑定正确的设备ID。这可以让Appium知道连接哪台设备，所以参数一定要准确。

如果你用了chromedriver或selendroid，不同的服务要设置不同的端口。

iOS并发测试

不幸的是，IOS不能进行本地并发测试。跟Android不一样，IOS在同一时间只能启动一个版本的模拟器来运行多个测试。

如果你想在IOS上进行并发测试，你需要用到Sauce。只需上传你的Appium测试脚本到Sauce，它就可以按照你的设置执行多个IOS或Android的并发测试。

3.5 Appium支持的平台

Appium支持很多的运行平台和测试方式(包括原生、混合应用、内嵌浏览器、真机、模拟器等)。这篇文档主要用来让大家明确在使用

Appimu的时候支持的平台版本和上述测试方式的必备条件。

iOS平台支持

版本号：6.1，7.0，以及7.1。

支持设备：iPhone模拟器，iPad模拟器以及iPhones和iPads真机。

是否支持原生应用：支持。同时支持模拟器中调试应用版本和正确签名的真机ipa。其他相关支持由苹果的UIAutomation框架提供。

是否支持内置移动浏览器：支持。Safari浏览器已经通过测试。对于真机，则需要安装调试工具ios-webkit-remote-debugger。很遗憾，对于Safari的原生界面的自动化是不支持的。更多信息请移步至mobile web doc 。

是否支持混合应用：支持。同样对于真机需要安装调试工具ios-webkit-remote-debugger，更多详情请移步至hybrid doc 查看详情。

是否支持在同一个session中执行多个应用的自动化：不支持。

是否支持同时再多个设备上执行自动化：不支持。

是否支持第三方提供应用：只支持在模拟器上有限的第三方应用（例如：喜好设置、地图等）。

是否支持自定义的、非标准UI控件的自动化：仅支持很少一部分。最好对控件添加可识别信息，以方便对元素进行一些基础的自动化操作。

Android平台支持

请移步至 Running on OS X: Android，Running on Windows，或者Running on Linux 获得在不同操作系统下android平台对appium的支持和安装配置文档。

支持版本：android 2.3平台及以上。

android 4.2平台及以上通过Appium自有的UiAutomator类库支持。默认在自动化后台。

从android 2.3到4.3平台，Appium是通过绑定Selendroid，实现自动化测试的，你可以到android开发社区的Instrumentation。(仪表盘)中查看相关介绍。Selendroid拥有一套不同的命令行和不同的profile文件(这部分差距正在逐步缩小)。要获得在后台运行自动化的权限，需要配置automationName 组件的值为 Selendroid。

支持的设备：Android模拟器和Android真机。

是否支持原生应用：支持。

是否支持内置移动浏览器：支持(除了使用Selendroid后台运行的情况)。通过代理方式绑定到Chromedriver来运行自动化测试。在android4.2和4.3版本中，只有在官方版本的谷歌浏览器或者Chromium下才能运行自动化测试。伴随着android 4.4+版本的出现。自动化测试则可以运行在内置浏览器的应用程序。但是需要在测试设备环境下安装Chrome/Chromium/浏览器。请移步至mobile web doc 获取更多详情。

是否支持混合应用: 支持。请移步至hybrid doc参考相关文档。

通过默认的Appium的后台支持android 4.4以上的版本。

通过Selendroid的后台支持android 2.3以上的版本。

是否支持在同一个session中执行多个应用的自动化：支持（但是不支持使用Selendroid后台的场景）。

是否支持同时再多个设备上执行自动化：支持,。尽管Appium必须要启动另一个端口即通过添加参数的方式运行命令行，例如--port，--bootstrap-port（或者--selendroid-port）或者--chromedriver-port。更多详情请移步至server args doc。

是否支持第三方应用自动化：支持（但是不支持Selendroid后台运行的场景）。

是否支持自定义的、非标准UI控件的自动化：不支持。

3.6 Appium在真机上

Appium在真机上

Appium已经初步支持真机测试。

如果要在真机上执行测试，你将要做如下准备：

1.一个苹果的开发者ID和有效的开发者对应的配置文件和签名文件

2.一台iPad或者iPhone

你要测试的应用的源码

一台安装了XCode和XCode Command Line Developer Tools的Mac机器

Provisioning Profile

要在真机上测试就需要一个有效的iOS开发者的Distribution Certificate and Provisioning Profile。你可以在这个上面找到配置这些的相关信息Apple documentation

同样的，你还需要对你的应用签名，更多的信息可以查看sign your app.

你必须使用Xcode的执行按钮来安装你的应用

使用Appium运行你的测试

一旦你的设备和应用设置好了之后，你就能够用如下的命令在你的机器上执行测试：

node . -U <UDID> --app <bundle\_id>

这将会启动Appium并且开始在真机上测试应用。

疑问解答思路

确认UDID已经正确的在xcode organizar或itunes中设置了。很长的字符串（20多个字符串） 0.确认你测试代码中的测试对象设备的设置

再次确认你从instruments启动你的自动化测试

确认instruments已经关闭

3.7 在 Linux 上运行 Appium

在 Linux 上运行 Appium

限制

如果你在 Linux 上使用 Appium， 那么你没法使用已经构建好的 '.app'，那是为 OS X 准备的。 另外由于 Appium 在测试 iOS 应用时 依赖 OS X 特有的库， 所以你也没有办法测试在 Linux 上测试 iOS 应用。

配置

首先，安装版本高于或等于 0.8 的 nodejs。可以根据 instructions for your flavor of linux 进行安装。

安装好了 node.js 之后，安装 Android SDK。 你会需要运行 android adb 等工具，这些工具都在 SDK 里包含了， 你要做的是配置环境变量。当然你要确保你的 API level 大于等于 17。 你也需要使用 Ant 来构建 bootstrap jar 以便 Appium 使用它来测试 Android 应用。

最后， 设置 $ANDROID\_HOME 为你的 Android SDK 的路径。比如， 你将 Android SDK 解压在 /usr/local/adt/， 那你就要将如下添加到你的 .bashrc 或 .zshrc 或 .bash\_profile 等 shell 配置文件中去:

export ANDROID\_HOME="/usr/local/adt/sdk

现在你可以运行 Appium 了， 在你 checkout 出来的 Appium 目录里， 运行 .reset.sh --android， 它会帮助你安装好所有的依赖。

运行 Appium

运行测试前， 你需要启动一个 API Level 大于等于 17 的 Android 模拟器或者连接一个系统是 4.1 以上的 Android 真机。然后在 Appium 目录运行

node .

你可以在 server documentation 找到所有的命令行参数。

备注

There exists a hardware accelerated emulator for android, it has it's own limitations. For more information you can check out this Android 有一些硬件加速的模拟器，这些模拟器有自己的限制。你可以在 page 找到更多的信息。

确保你使用的 AVD 里面的 config.ini 有这条指令 hw.battery=yes。

3.8 在 Mac OS X 上使用 Appium

在 Mac OS X 上使用 Appium

在 OS X 上， Appium 支持 iOS 和 Android 测试

系统配置 (iOS)

Appium 需要 Mac OS X 10.7， 推荐 10.8。 （经过测试， 10.9 也能工作。）

确保 Xcode 和 iOS SDK 都已经安装好了。 （当前 Appium 支持 Xcode 4.6.3/iOS 6.1 和 Xcode 5/iOS 7.0。 注意不推荐在基于 Xcode 5 下且低于 7.0 的 iOS 版本进行测试。 参照下篇可以获取更多信息）

你需要授权 iOS 模拟器的使用。如果你是通过 NPM 安装的 Appium，那么你可以运行 sudo authorize\_ios （authorize\_ios）是来自 Appium npm 包里的一个二进制执行文件。如果你是从源代码运行 Appium，那么你可以简单的使用 sudo grunt authorize。如果你使用Appium.app, 那你只要用界面来操作。

如果你使用的是Xcode 6，在启动Appium之前，你需要打开模拟器，并且在你需要进行输入文字的操作之前，必须先将输入法提前调出。你可以通过点击输入区域或通过快捷键command-K来将软键盘唤出。

Xcode 6中，有一个Devices的模块（command-shift-2可唤出）。你必须确保Appium 的capabilities参数中，所使用到的deviceName要存在于Devices里。换句话说，如果capabilities中的deviceName为"iPhone 5s"，platformVersion为"8.0"，那么你必须确保Devices中要存在那么一个设备是"iOS8系统的iPhone5s"，否则Appium将不知道使用哪一个设备进行测试。

在iOS8设置中的开发者选项里面，你可以打开或关闭UIAutomation。如果你的是iOS8设备，请在运行Appium之前，确保UIAutomation是打开状态的。

使用多种 iOS SDK 进行测试

Appium 使用苹果提供的 instruments 来启动 iOS 模拟器，默认它会使用当前安装的 Xcode 和该 Xcode 下安装好的最高版本的 iOS SDK。这就意味着如果你想测试 iOS 6.1， 但是你安装了 iOS 7.0， 那么 Appium 会强制使用 7.0 的模拟器。 唯一的方法就是安装多个Xcode，然后在安装不同的 SDK。然后在启动 Appium 前，切换到你要测试的特定的版本。

另外，我们发现 Xcode 5 上的 iOS 6.1 测试，会很慢而且不稳定。所以我们推荐，如果要在 6.1 及 6.1 以下版本的 iOS 上进行测试，请使用 Xcode 4.6.3。如果要在 iOS 7.0 上测试，请使用 Xcode 5。假设我们的 Xcode 5 在 /Applications/Xcode.app， Xcode 4.6 在 /Applications/Xcode-4.6.app，我们就可以用下面的命令来切换到 Xcode 4.6 来为 iOS 6.1 测试做准备。

sudo xcode-select -switch /Applications/Xcode-4.6.app/Contents/Developer/

如果要回到 Xcode 5 的话，我们再运行一次:

sudo xcode-select -switch /Applications/Xcode.app/Contents/Developer/

系统配置 (Android)

在Mac OSX 上运行Android项目所需要的配置，与Linux的配置方法是一致的，请参考 Android setup docs。

3.9 在Windows上运行Appium

在Windows上运行Appium

局限性

如果你在windows上安装appium，你没法使用预编译专用于OS X的.app文件，你也将不能测试IOS apps，因为appium依赖OS X专用的库来支持IOS测试。这意味着你只能通过在mac上来运行IOS的app测试。这点限制挺大。

开始安装

安装nodejs 0.8版本及以上, 通过官方的安装程序来安装。

安装android的sdk包，(http://developer.android.com/sdk/index.html), 运行依赖sdk中的'android'工具。并确保你安装了Level17或以上的版本api。设置ANDROID\_HOME系统变量为你的Android SDK路径，并把tools platform-tools两个目录加入到系统的Path路径里。因为这里面包含有一些执行命令

安装java的JDK，并设置JAVA\_HOME 变量为你的JDK目录。

安装Apache Ant

或者直接使用Android Windows SDK自带的ant，地址在eclipse\plugins目录，你需要把这个目录加到你的系统PATH变量中

安装Apache Maven. 并且设置M2HOME和M2环境变量，把M2环境变量添加到你的系统PATH变量中。

安装Git. 确保你安装了windows下的Git，以便可以运行常用的command命令

现在，你已经下载安装了所有的依赖，开始运行

reset.bat

运行Appium

要在windows上运行测试用例，你需要先启动Android模拟器或者连接上一个API Level17以上的android真机。然后在命令行运行appium。

如果你是使用源码运行Appium的，请在你所安装的appium目录下执行node.js命令：

node .

在windows系统下运行appium.app时，需要使用管理员权限；当你通过源码的形式运行Appium时，也需要使用管理员权限启动CMD。

在windows系统下运行Android项目时，启动Appium时请带上--no-reset或--full-reset命令。

有一个硬件加速模拟器用于android，但是它有自己的一些限制

确保在你的AVD的config.ini中有一个配置项为hw.battery=yes

最简略的安装方式

出于对官方文档的尊重，我按照原文翻译，如下介绍我的安装心得。官方提到的一些工具，其实并不需要安装。

下面介绍我已经测试过的安装和使用过程

安装appium

安装nodejs

使用npm安装appium，npm install -g appium

运行appium

启动appium，直接运行appium 即可。

更新appium

通过npm install -g appium 来更新appium即可

如果有任何疑问，欢迎到testerhome.com来交流

3.10 Appium 故障排除

Appium 故障排除

当你遇到问题时，请不要急着将问题提交到Github，也不用急着发到appium-discuss discussion group，也许你可以在本文中找到答案。

常见问题解决办法

确保你的每一个步骤都是遵循 入门指南 来做的。

确保你的系统配置正确。（例如：Xcode是否升级到了最新版本，Android SDK是否有设置到环境变量ANDROID\_HOME中去。）

确保你的应用存放路径没有错误。

Appium.app运行出现问题的解决办法

升级Appium.app后重新打开即可解决。如果提示你不能升级，则需要重新下载Appium.app，下载地址：appium.io

通过源码启用Appium出现问题的解决办法

使用git pull拉取最新源码，确保运行的代码是当前最新版本。

针对你所自动化的平台，运行reset.sh命令：

命令 说明

./reset.sh # all

./reset.sh --ios # ios-only

./reset.sh --android # android-only

./reset.sh --selendroid # selendroid-only

当你需要下载以及构建测试应用时，运行reset.sh时你需要用到--dev指令。

你也可以使用appium-doctor来自动检测你的环境依赖都是否正常。如果你是使用源码运行，则需要使用到bin/appium-doctor.js或node bin/appium-doctor.js。

当你将Android SDK升级到22后，可能出现如下错误： {ANDROID\_HOME}/tools/ant/uibuild.xml:155: SDK does not have any Build Tools installed. 这是因为在Android SDK 22中，platform 和 build 工具被分拆到他们各自的SDK管理包中去了。你需要确保你的机器上正确安装了build-tools 和 platform-tools。

Android常见问题解决办法

确保 Android 模拟器启动并运行着。

出现设备连接问题时，运行adb kill-server && adb devices是非常有效的。它能够帮助重置和连接Android设备。

请确保环境变量 ANDROID\_HOME 指向的是正确的Android SDK的路径。

IOS常见问题解决方案

确保Instruments.app是关闭的。

如果你是使用模拟器运行的，请不要将真机设备连接电脑。

确保模拟器或真机中，设置里面的accessibility辅助功能是关闭状态的。

确保App是编译在当前运行的模拟器上。

确保App是编译在合适的模拟器（或真机）上，不然会出现posix spawn的报错。（比如：运行在debug模式下的模拟器）

如果你曾经用 sudo 运行过 Appium， 你需要先删除/tmp/instruments\_sock， 执行sudo rm /tmp/instruments\_sock。然后在不适用SUDO的情况下再次启动Appium即可。

第一次运行Appium时，需要对Instruments进行授权。不然的话会经常弹出对话框要求你输入密码。如果你从源代码运行 Appium，你只需在主分支上运行sudo grunt authorize来回避该弹窗。如果用 npm 安装的话，运行 sudo authorize\_ios 即可。注意，当你每次安装了新版本的xcode，你都需要重复以上操作。

如果检查路径正确，但仍然报 iOS Simulator failed to install the application.的错误的时候，请尝试重启你的电脑。

Webview/Hybrid/Safari 应用支持

确保真机上的'Web Inspector'为打开状态。

确保打开了Safari的开发模式。（Safari - Advance Preferences- Developer menu for simulators）

确保由client library提供的Appium命令-context能够正常得对contexts进行切换。

当你尝试打开代理的时候，出现如下错误：select\_port() failed，请参考discussion

FirefoxOS常见问题解决办法

确保 Boot-to-Gecko 模拟器启动并运行着。

确保模拟器的屏幕是亮着并无锁屏的(可能需要重启 B2G).

到社区寻求帮助

若通过上述方法你的问题依然没有得到解决，你可以：

如果你的 Appium 无法正常工作，然后错误信息不够清晰，欢迎加入 discussion group中发表你的问题，你的问题需要包括以下内容：

你是如何运行Appium的？(Appium.app, npm, source)

你使用的是什么操作系统？

你使用的是什么设备？版本是什么？ (i.e. Android 4.4, or iOS 7.1)

你使用的是真机还是模拟器？

给出你得到的客户端和服务端的出错日志 (比如,"我的Python代码中报了如下错误：balabala，在Appium server中的输出内容如链接中所示"）

除了上述， 贴出 Appium 服务器端的输出也非常重要，特别是运行在 verbose 模式。这样我们可以分析诊断问题在哪里。

如果你确信你发现的是一个BUG，请到issue tracker中提交一个issue，并将BUG的内容描述清楚。

已知问题

如果你从 Node 官网安装的 Node，那需要你使用 sudo 运行 npm。 但这么做并不是非常理想。请尝试从 n 获取node 或运行brew install node来安装 。

Webview通过代理可以支持iOS真机设备，请参考discussion

有时候， iOS 的 UI 元素在定位到之后几毫秒会突然变得无效。这会导致一个类似(null) cannot be tapped的错误。唯一的解决方法就是把finding-and-acting的代码放到 retry 块里。

如果你是通过MacPorts安装了Node和Npm，你必须确保MacPorts的bin文件夹已经被添加到环境变量PATH中去，不然Appium会出现难以找到可执行node的情况。

特定的错误

Action Error Resolution

Running reset.sh xcodebuild: error: SDK "iphonesimulator6.1" cannot be located 安装 iPhone 6.1 SDK 或者 使用单独的 SDK 构建 待测应用 比如： grunt buildApp:UICatalog:iphonesimulator5.1

Running reset.sh Warning: Task "setGitRev" not found. Use --force to continue. 使用git submodule update --init更新模块并再次运行reset.sh

Running reset.sh [ERROR] Failed to execute goal org.apache.maven.plugins:maven-compiler-plugin:2.3.2:compile (default-compile) on project selendroid-server: Compilation failure [ERROR] Failure executing javac, but could not parse the error: [ERROR] [ERROR] [ERROR] The system is out of resources. [ERROR] Consult the following stack trace for details. [ERROR] java.lang.StackOverflowError export MAVEN\_OPTS="-Xms1024m -Xmx2048m -Xss2048k"

Running ios test [INST STDERR] posix spawn failure; aborting launch 你的应用没有正确编译在模拟器或真机上。

Running mobile safari test error: Could not prepare mobile safari with version '7.1' 你可能需要在此运行授权脚本以保证使iOS SDK文件为可写状态。 E.g., sudo authorize\_ios

第四章：appium运行

4.1 从源码运行Appium

你想要从源码运行 Appium 并帮助修复 bug 和添加新的特性么？很好！ fork 这个项目，做一点更改，并且发送一个请求吧！

另外，在工作之前请先看下我们的代码风格指南。请在发送请求前，确保单元测试与功能测试都测试通过；

关于如何运行测试的更多信息，请继续阅读!

首先确保你阅读并遵循 README 中的安装说明。

###从源码配置Appium

Appium 的安装，包含在你的测试代码与设备/模拟器之间来回发送消息的 Appium 服务端，和一个用任何存在且兼容Appium的语言编写的测试脚本。

运行一个 Appium 服务器实例，然后进行你的测试。

快速开始的方式：

$ git clone https://github.com/appium/appium.git

$ cd appium

$ ./reset.sh

$ sudo ./bin/authorize-ios.js # for ios only

$ node .

Appium 开发环境搭建

确保你安装了 ant，maven，adb 并且将他们加入到了系统环境变量 PATH 中,与此同时你还需要安装 android-16 sdk(Selendroid) 和android-19 sdk。

从你本地仓库的命令行提示，使用下边的命令安装如下包（如果你没有使用homebrew包管理器安装 node，则你可能不得不使用 sudo 权限运行npm）:

npm install -g mocha

npm install -g grunt-cli

node bin/appium-doctor.js --dev

./reset.sh --dev

前两个命令安装测试和构建工具（如果你已经通过 Homebrew 包管理器安装了 node.js 就不需要 sudo 了）。

第三个命令验证所有的依赖关系是否设置正确（由于依赖关系构建 Appium 不同于简单的运行 Appium ），

第四个命令安装所有程序依赖关系和构建支持二进制文件和测试应用程序。

reset.sh 也是建议先从 master 上 pull 下改变后的内容再执行命令。

运行 reset.sh 加上 --dev 标志同时安装 git hooks 以确保代码质量在提交时是被保存过的。

此时，你可以启动 Appium 服务:

node .

查看完整的服务文档参数列表the server documentation

想要实现任务自动化，请检出Appium Grunt tasks来构建应用程序，安装程序，生成文档，等等。

搭建iOS运行环境

为了避免启动 iOS apps 时弹出安全警告，你可以通过以下两种方法修改 /etc/authorization 文件：

手动将 /etc/authorization 文件中 <key>system.privilege.taskport<key/> 下紧跟 <allow-root> 的元素改成 <true/>。

运行以下grunt命令来自动修改 /etc/authorization 文件:

sudo ./bin/authorize-ios.js

然后再运行以下命令：

./reset.sh --ios --dev

现在你的 appium 实例已经准备就绪，运行 node . 来启动 appium server.

搭建android运行环境

Bootstrap 通过运行以下命令来启动 android：

./reset.sh --android --dev

如果你想运行Selendroid 来支持2.3这样的旧的android平台，运行以下命令：

./reset.sh --selendroid --dev

确保你有且只有一个 Android 模拟器或者真机在运行，举个例子，在其它的设备上运行此命令（假设 emulator 命令已经在你的 path 中了）需执行：

emulator -avd <MyAvdName>

现在你可以通过 node . 启动 Appium server 了。

确保更新到最新版本

由于 Appium 使用一些包的开发版本，所以经常安装新的 npm 包和升级不同的包是很有必要的。以下命令可以将所有平台上的包进行更新（ --dev 标志会获取 npm dev 依赖和 Appium 测试套件中用到的应用程序）。当Appium提示版本更新时，你也可以用以下命令来更新：

./reset.sh --dev

或者你可以只更新指定的平台：

./reset.sh --ios --dev

./reset.sh --android --dev

./reset.sh --selendroid --dev

运行测试集

首先，看看我们的文档普通情况下执行测试 ，

然后确保你的环境在对应的平台上已经搭建好了且与你所期望的那样。

当你的环境搭建好了之后并且你的代码是最新的，你可以通过以下的方式来运行单元测试:

grunt unit

你可以在所支持的平台上运行一些功能测试（确保后 Appium 用 node . 在另外一个窗口中运行）：

bin/test.sh

或者你可以通过运行 test.sh 来对指定的平台环境进行测试:

bin/test.sh --android

bin/test.sh --ios

bin/test.sh --ios7

bin/test.sh --ios71

在提交代码时，请运行 grunt 执行一些基本的测试和核对代码质量标准的更改，请注意，这可能会自动发生的，

如果你已经运行 reset.sh --dev ，这于你预先提交代码的操作所关联起来的。

grunt lint

> Running "newer:jshint" (newer) task

>

> Running "newer:jshint:files" (newer) task

> No newer files to process.

>

> Running "newer:jshint:test" (newer) task

> No newer files to process.

>

> Running "newer:jshint:examples" (newer) task

> No newer files to process.

>

> Running "jscs:files" (jscs) task

> >> 303 files without code style errors.

运行单独的测试

如果你有一个 Appium 服务监听，你可以通过 Mocha 来运行单独的测试文件，例如：

DEVICE=ios71 mocha -t 60000 -R spec test/functional/ios/testapp/simple.js

或者单独的测试集（例如，测试名称中的单词 "alert" ）

DEVICE=ios6 mocha -t 60000 -R spec --grep "alert" test/functional/ios/uicatalog

对于 windows 操作系统，你可以用 set DEVICE=android 在 cmd 命令行的方式中运行以上所有测试集，例如：

set DEVICE=android

mocha -t 60000 -R spec test/functional/android/apidemos/alerts-specs.js

注意：对于安卓系统，你将需要一个屏幕大小为4.0（400x800）的模拟器/设备（emulator/device），有些测试集在不同的屏幕大小下可能会失败。

DEVICE 必须设置为一个有效的值:ios71, ios6, android, selendroid

4.2 appium 开发环境搭建

##从源码运行Appium

你想要从源码运行 Appium 并帮助修复 bug 和添加新的特性么？很好！ fork 这个项目，做一点更改，并且发送一个请求吧！

另外，在工作之前请先看下我们的代码风格指南。请在发送请求前，确保单元测试与功能测试都测试通过；

关于如何运行测试的更多信息，请继续阅读!

首先确保你阅读并遵循 README 中的安装说明。

###从源码配置Appium

Appium 的安装，包含在你的测试代码与设备/模拟器之间来回发送消息的 Appium 服务端，和一个用任何存在且兼容Appium的语言编写的测试脚本。

运行一个 Appium 服务器实例，然后进行你的测试。

快速开始的方式：

$ git clone https://github.com/appium/appium.git

$ cd appium

$ ./reset.sh

$ sudo ./bin/authorize-ios.js # for ios only

$ node .

Appium 开发环境搭建

确保你安装了 ant，maven，adb 并且将他们加入到了系统环境变量 PATH 中,与此同时你还需要安装 android-16 sdk(Selendroid) 和android-19 sdk。

从你本地仓库的命令行提示，使用下边的命令安装如下包（如果你没有使用homebrew包管理器安装 node，则你可能不得不使用 sudo 权限运行npm）:

npm install -g mocha

npm install -g grunt-cli

node bin/appium-doctor.js --dev

./reset.sh --dev

前两个命令安装测试和构建工具（如果你已经通过 Homebrew 包管理器安装了 node.js 就不需要 sudo 了）。

第三个命令验证所有的依赖关系是否设置正确（由于依赖关系构建 Appium 不同于简单的运行 Appium ），

第四个命令安装所有程序依赖关系和构建支持二进制文件和测试应用程序。

reset.sh 也是建议先从 master 上 pull 下改变后的内容再执行命令。

运行 reset.sh 加上 --dev 标志同时安装 git hooks 以确保代码质量在提交时是被保存过的。

此时，你可以启动 Appium 服务:

node .

查看完整的服务文档参数列表the server documentation

想要实现任务自动化，请检出Appium Grunt tasks来构建应用程序，安装程序，生成文档，等等。

搭建iOS运行环境

为了避免启动 iOS apps 时弹出安全警告，你可以通过以下两种方法修改 /etc/authorization 文件：

手动将 /etc/authorization 文件中 <key>system.privilege.taskport<key/> 下紧跟 <allow-root> 的元素改成 <true/>。

运行以下grunt命令来自动修改 /etc/authorization 文件:

sudo ./bin/authorize-ios.js

然后再运行以下命令：

./reset.sh --ios --dev

现在你的 appium 实例已经准备就绪，运行 node . 来启动 appium server.

搭建android运行环境

Bootstrap 通过运行以下命令来启动 android：

./reset.sh --android --dev

如果你想运行Selendroid 来支持2.3这样的旧的android平台，运行以下命令：

./reset.sh --selendroid --dev

确保你有且只有一个 Android 模拟器或者真机在运行，举个例子，在其它的设备上运行此命令（假设 emulator 命令已经在你的 path 中了）需执行：

emulator -avd <MyAvdName>

现在你可以通过 node . 启动 Appium server 了。

确保更新到最新版本

由于 Appium 使用一些包的开发版本，所以经常安装新的 npm 包和升级不同的包是很有必要的。以下命令可以将所有平台上的包进行更新（ --dev 标志会获取 npm dev 依赖和 Appium 测试套件中用到的应用程序）。当Appium提示版本更新时，你也可以用以下命令来更新：

./reset.sh --dev

或者你可以只更新指定的平台：

./reset.sh --ios --dev

./reset.sh --android --dev

./reset.sh --selendroid --dev

运行测试集

首先，看看我们的文档普通情况下执行测试 ，

然后确保你的环境在对应的平台上已经搭建好了且与你所期望的那样。

当你的环境搭建好了之后并且你的代码是最新的，你可以通过以下的方式来运行单元测试:

grunt unit

你可以在所支持的平台上运行一些功能测试（确保后 Appium 用 node . 在另外一个窗口中运行）：

bin/test.sh

或者你可以通过运行 test.sh 来对指定的平台环境进行测试:

bin/test.sh --android

bin/test.sh --ios

bin/test.sh --ios7

bin/test.sh --ios71

在提交代码时，请运行 grunt 执行一些基本的测试和核对代码质量标准的更改，请注意，这可能会自动发生的，

如果你已经运行 reset.sh --dev ，这于你预先提交代码的操作所关联起来的。

grunt lint

> Running "newer:jshint" (newer) task

>

> Running "newer:jshint:files" (newer) task

> No newer files to process.

>

> Running "newer:jshint:test" (newer) task

> No newer files to process.

>

> Running "newer:jshint:examples" (newer) task

> No newer files to process.

>

> Running "jscs:files" (jscs) task

> >> 303 files without code style errors.

运行单独的测试

如果你有一个 Appium 服务监听，你可以通过 Mocha 来运行单独的测试文件，例如：

DEVICE=ios71 mocha -t 60000 -R spec test/functional/ios/testapp/simple.js

或者单独的测试集（例如，测试名称中的单词 "alert" ）

DEVICE=ios6 mocha -t 60000 -R spec --grep "alert" test/functional/ios/uicatalog

对于 windows 操作系统，你可以用 set DEVICE=android 在 cmd 命令行的方式中运行以上所有测试集，例如：

set DEVICE=android

mocha -t 60000 -R spec test/functional/android/apidemos/alerts-specs.js

注意：对于安卓系统，你将需要一个屏幕大小为4.0（400x800）的模拟器/设备（emulator/device），有些测试集在不同的屏幕大小下可能会失败。

DEVICE 必须设置为一个有效的值:ios71, ios6, android, selendroid

4.3 Appium grunt 命令

Appium grunt 命令

Grunt 是 Node.js 的 Make! 我们用它来自动化所有的 appium 开发任务。 下面就是你能做的：

任务 描述

grunt lint 运行 JSLint

grunt test 运行所有的测试

grunt functional 运行整个功能测试集

grunt ios 运行 iOS 功能测试集

grunt android 运行 Android 功能测试集

grunt selendroid 运行 selendroid 功能测试集

grunt firefoxos 运行 firefoxos 功能测试集

grunt unit 运行所有的单元测试

grunt getSampleCode 下载示例代码和示例app. 接受:hardcore 参数

grunt buildApp:<AppName>:<SDK> 构建一个用于 iPhone 模拟器的 iOS 应用。 我们预计这个应用的路径是 sample-code/apps/<AppName>/build/Release-iphonesimulator/<AppName>.app. 默认的 SDK 是 'iphonesimulator7.1'

grunt signApp:<certName> 使用开发证书的绝对路径，签名测试应用。

grunt authorize 授权模拟器，使它不需要弹框请求权限。

grunt log 打印 appium.log (运行测试的时候很有用)

grunt configAndroidBootstrap 配置使用 ant 构建 Android 的 bootstrap.jar

grunt buildAndroidBootstrap 使用 ant 构建 bootstrap.jar

grunt buildSelendroidServer 构建 selendroid 服务器

grunt configAndroidApp:<AppName> 配置使用 ant 构建 android 测试应用。 我们期待有一个 sample-code/apps/<AppName> 的 Android 项目

grunt buildAndroidApp:<AppName> 使用 ant 构建项目. 会在 sample-code/apps/<AppName> 下生成应用。

grunt installAndroidApp:<AppName> 将安卓应用安装到模拟器和设备中去

grunt docs 生成文档

grunt generateAppiumIo 将 README.md 转成 appium.io 的 getting-started.html

grunt setConfigVer:<device> 将 package.json 里面 appium 的版本号和对应设备写入 .appiumconfig.json 文件

其他

grunt buildApp 默认使用 iPhone 6.1 模拟器的 SDK 来构建应用。你可以传其他的 SDK 给 grunt 命令。

(用 xcodebuild -showsdks 找出你所有的 sdk）:

grunt buildApp:UICatalog:iphonesimulator6.0

4.4 如何去写文档

如何去写文档

## 被用于写第二级标题。每个文档必须以第二级标题开头。

这是为了支持appium.io文档的生成，不要使用下划线---的方式来创建标题。

不要使用第一级标题或者 === 底线方式来创建标题（其中文件夹名字被用于第一级标题）

小标题

### 用于小标题。

普通标题

#### 用于不会出现在目录中的标题。

不要使用第五级标题 #####, 或者第六级标题 ######。

分隔线

不要使用分隔线例如 -- 或者 ---。 这会使 Slate 混乱。

链接

链接到 readme：

[readme](../../README.md)

链接到 contributing:

[contributing](../../CONTRIBUTING.md)

链接到其他文档：

[link text](filename.md)

链接到文档的内部, 使用 # 来标记 Slate 链接。

[go direct to json](filename.md#json-wire-protocol-server-extensions)

需要注意的是当标题改变时，哈希链接会损坏。所以链接到文档的开头是最好的( other.md 替换 other.md#something )。

appium.io兼容性

在appium.io中心对齐代码

Appium.io中文档使用 slate 来作为文档标准

如果在文件中的代码段不是特定语言或如果你想要代码片段保持与文本中心对齐在 appium.io 文档中，请把代码块放在中心位置

例子：

中心

代码片段放在这里

发布

发布文档请在appium.io中查看 api-docs 和

在 appium.io 中查看。

4.5 代码风格指南

贡献者的代码风格指南

感谢你们对 Appium 的贡献! 这些是我们书写 javascript 代码时使用的基本原则。

请遵守这些，避免风格的来回修改，以便我们可以合并你的 pull 请求。

基本原则就是：让你的代码看起来和周围的代码一致。

衍合（Rebasing）

每个 pull 请求中的提交（commits）必须包括 logical changes。

如果有多个作者，确认每个作者有自己的提交。最好不要修改作者信息。

合并（merge）提交必须从 pull 请求中 rebase 。

检错（Linting）

所有的代码 （除了 bootstrap.js 的代码，它使用了 Apple 的私有方法） 必须通过 JSLint。

为了检查你的代码，你可以在 Appium 存储目录下，简单地运行 grunt lint。

如果你已创建一个新的 .js 文件，请确认它在 grunt.js 中被通配符覆盖，或者被专门添加。

边输入边检错你的代码是容易实现的，使得整个进程更加顺利。

我们喜欢 jshint, 因为它有与许多源代码编辑器的集成。

文件 .jshintrc 加入到仓库中，它的内容是：

{

"laxcomma": true,

"strict": true,

"undef": true,

"unused": true,

"trailing": true,

"node": true,

"es5": true,

"white": true,

"indent": 2

}

因为jshint不再执行代码风格，我们也使用 jscs，它其中也存在许多源代码编辑器的集成。配置文件是：

{

"excludeFiles": ["submodules/\*\*", "node\_modules/\*\*",

"./lib/server/static/\*\*", "./lib/devices/firefoxos/atoms/\*.js",

"./test/harmony/\*\*/\*.js", "./sample-code/examples/node/\*\*/\*-yiewd.js",

"./sample-code/apps/\*\*", "./sample-code/examples/php/vendor/\*\*"],

"requireCurlyBraces": ["for", "while", "do", "try", "catch"],

"requireSpaceAfterKeywords": ["if", "else", "for", "while", "do", "switch",

"return", "try", "catch", "function"],

"disallowMixedSpacesAndTabs": true,

"disallowTrailingWhitespace": true,

"requireSpacesInFunctionExpression": {

"beforeOpeningCurlyBrace": true

}

}

你将在你喜欢的编辑器中看到这些在配置文件中定义的警告类型，查看this page for jshint 和

this page for jscs，编辑器和平台列表，找到使你的编辑器自动化检错的设置方法。

风格注意点

使用两个空格来缩进, 不要使用 tabs

在运算符两边，分别添加一个空格：

var x = 1;

而不是

js

var x=1;

在 lists, objects, function calls 等中，逗号和冒号后面需要添加一个空格：

var x = myFunc("lol", {foo: bar, baz: boo});

而不是

js

var x = myFunc("lol",{foo:bar,baz:boo});

代码语句一般以分号结尾

以逗号开头：

var x = {

foo: 'bar'

, baz: 'boo'

, wuz: 'foz'

};

左花括号应该和 function, if 等 写在同一行， else 被夹在两个花括号中间：

if (foo === bar) {

// do something

} else {

// do something else

}

if, for, 和 function 之后需要添加空格：

if (foo === bar) {

for (var i = 0; i < 10; i ++) {

var lol = function (foo) {

而不是

js

if(foo === bar) {

js

for(var i = 0; i < 10; i ++) {

js

var lol = function(foo) {

只有一行代码时，花括号也应该添加上：

if (foo === bar) {

foo++;

}

而不是

js

if (foo === bar)

foo++;

一般情况下，使用 ===, 而不是 ==； 使用 !==, 而不是 !=

单行长度不应超过79个字符

截断长字符串，方法如下：

myFunc("This is a really long string that's longer " +

"than 79 characters so I broke it up, woo");

注释需要和上一行代码左对齐：

if (foo === 5) {

myFunc(foo);

// foo++;

}

而不是

js

if (foo === 5) {

myFunc(foo);

//foo++;

}

除了出错后直接调用回调函数（callback）处理错误（error）的语句

if (err) return cb(err);

通过拓展原型，来创建子类：

var \_ = require('underscore');

var SuperClass = function () {

this.init();

};

SuperClass.prototype.init = function () {

// initialize

};

// Create a subclass

var SubClass = function () {

this.init();

};

\_.extend(SubClass.prototype, SuperClass.prototype);

函数定义中，最后使用回调函数：

var foo = function (arg1, arg2, cb) {

...

};

使用变量来定义函数：

var myFunc = function (a, b, c) {};

而不是

js

function myFunc (a, b, c) {}

变量名应该是驼峰式大小写风格：

var myVariable = 42;

而不是

js

var my\_variable = 42;

检查是否有未定义的变量：

typeof myVariable === "undefined"

而不是

js

myVariable === undefined

试验风格：

在代码语义通顺和长度许可下，可以保持在同一行：

样例：

driver.elementByTagName('el1').should.become("123")

.nodeify(done);

driver

.elementsByTagName('el1').should.eventually.have.length(0)

.nodeify(done);

或者使用缩进来提高代码的可读性：

h.driver

.elementById('comments')

.clear()

.click()

.keys("hello world")

.getValue()

.should.become("hello world")

.elementById('comments')

.getValue().should.become("hello world")

.nodeify(done);

h.driver

.execute("'nan'--")

.should.be.rejectedWith("status: 13")

.nodeify(done);

第五章：appium使用

5.1 Appium 客户端库

Appium 客户端库

Appium 有对应以下语言的客户端库:

语言 代码

Ruby GitHub

Python GitHub

Java GitHub

JavaScript GitHub

PHP GitHub

C# GitHub

Objective-C GitHub

请注意：有些方法，比如 endTestCoverage() 目前不能提供完整支持。

只有这个问题修复， 完整的覆盖率支持才会被添加。

如果你一定要用这些方法，请先查看 Github 上关于 bindings 的文档。

锁定

锁定屏幕

# ruby

lock 5

# python

driver.lock(5)

// java

driver.lockScreen(3);

// javascript

driver.lock(3)

// php

$this->lock(3);

// c#

driver.LockDevice(3);

// objective c

[driver lockDeviceScreen:3];

将 app 置于后台

把当前应用放到后台去

# ruby

background\_app 5

# python

driver.background\_app(5)

// java

driver.runAppInBackground(5);

// javascript

driver.backgroundApp(5)

// php

$this->backgroundApp(5);

// c#

driver.BackgroundApp(5);

// objective c

[driver runAppInBackground:3];

收起键盘

# ruby

hide\_keyboard

# python

driver.hide\_keyboard()

// java

driver.hideKeyboard();

// javascript

driver.hideKeyboard()

// php

$this->hideKeyboard();

$this->hideKeyboard(array('strategy' => 'pressKey', 'key' => 'Done'));

// c#

driver.HideKeyboard("Done");

// objective c

[driver hideKeyboard];

启动 Activity

在当前应用中打开一个 activity 或者启动一个新应用并打开一个 activity 。 只能在 Android 上使用

// java

driver.startActivity("appPackage","com.example.android.apis", null, null);

// javascript

driver.startActivity({appPackage: 'com.example.android.apis', appActivity: '.Foo'}, cb);

# python

driver.start\_activity('com.example.android.apis', '.Foo')

# ruby

start\_activity app\_package: 'io.appium.android.apis', app\_activity: '.accessibility.AccessibilityNodeProviderActivity'

// c#

driver.StartActivity("com.example.android.apis", ".Foo");

// php

$this->startActivity(array("appPackage" => "com.example.android.apis",

"appActivity" => ".Foo"));

// objective c

[driver startActivity:@"com.example.android.apis" package:@".Foo"];

打开通知栏 (Notifications)

打开下拉通知栏 只能在 Android 上使用

// java

driver.openNotifications();

// javascript

driver.openNotifications(cb);

# python

driver.open\_notifications()

# ruby

openNotifications

// c#

driver.OpenNotifications();

// php

$this->openNotifications();

// objective c

[driver openNotifications];

是否已经安装

检查应用是否已经安装

# ruby

is\_installed? "com.example.android.apis"

# python

driver.is\_app\_installed('com.example.android.apis')

// java

driver.isAppInstalled("com.example.android.apis")

// javascript

driver.isAppInstalled("com.example.android.apis")

.then(function (isAppInstalled) { /\*...\*/ })

// php

$this->isAppInstalled('com.example.android.apis');

// c#

driver.IsAppInstalled("com.example.android.apis-");

// objective c

[driver isAppInstalled:@"com.example.android.apis-"];

安装应用

安装应用到设备中去

# ruby

install 'path/to/my.apk'

# python

driver.install\_app('path/to/my.apk')

// java

driver.installApp("path/to/my.apk")

// javascript

driver.installApp("path/to/my.apk")

// php

$this->installApp('path/to/my.apk');

// c#

driver.InstallApp("path/to/my.apk");

// objective c

[driver installAppAtPath:@"path/to/my.apk"];

删除应用

从设备中删除一个应用

# ruby

remove 'com.example.android.apis'

# python

driver.remove\_app('com.example.android.apis')

// java

driver.removeApp("com.example.android.apis")

// javascript

driver.removeApp("com.example.android.apis")

// php

$this->removeApp('com.example.android.apis');

// c#

driver.RemoveApp("com.example.android.apis");

// objective c

[driver removeApp:@"com.example.android.apis"];

摇晃 (Shake)

模拟设备摇晃

# ruby

shake

# python

driver.shake()

// java

driver.shake()

// javascript

driver.shake()

// php

$this->shake();

// c#

driver.ShakeDevice();

// objective c

[driver shakeDevice];

关闭应用

# ruby

close\_app

# python

driver.close\_app();

// java

driver.closeApp()

// javascript

driver.closeApp()

// php

$this->closeApp();

// c#

driver.CloseApp();

// objective c

[driver closeApp];

启动 (Launch)

根据服务关键字 (desired capabilities) 启动会话 (session) 。请注意这必须在设定 autoLaunch=false 关键字时才能生效。这不是用于启动指定的 app/activities ————你可以使用 start\_activity 做到这个效果————这是用来继续进行使用了 autoLaunch=false 关键字时的初始化 (Launch) 流程的。

# ruby

launch

# python

driver.launch\_app()

// java

driver.launchApp()

// javascript

driver.launchApp()

// php

$this->launchApp();

// c#

driver.LaunchApp();

// objective c

[driver launchApp];

重置 (Reset)

应用重置

(翻译者注：相当于卸载重装应用)

# ruby

reset

# python

driver.reset()

// java

driver.resetApp()

// javascript

driver.resetApp()

// php

$this->reset();

// c#

driver.ResetApp();

// objective c

[driver resetApp];

可用上下文 (context)

列出所有的可用上下文

翻译备注：context可以理解为 可进入的窗口 。例如，对于原生应用，可用的context和默认context均为NATIVE\_APP。详情可查看对混合应用进行自动化测试

# ruby

context\_array = available\_contexts

# python

driver.contexts

// java

driver.getContextHandles()

// javascript

driver.contexts().then(function (contexts) { /\*...\*/ })

// php

$this->contexts();

// c#

driver.GetContexts()

// objective c

NSArray \*contexts = driver.allContexts;

当前上下文 (context)

列出当前上下文

# ruby

context = current\_context

# python

driver.current\_context

// java

driver.getContext()

// javascript

driver.currentContext().then(function (context) { /\*...\*/ })

// php

$this->context();

// c#

driver.GetContext()

// objective c

NSString \*context = driver.context;

切换到默认的上下文 (context)

将上下文切换到默认上下文

# ruby

switch\_to\_default\_context

# python

driver.switch\_to.context(None)

// java

driver.context();

// javascript

driver.context()

// php

$this->context(NULL);

// c#

driver.SetContext();

// objective c

[driver setContext:nil];

应用的字符串 (App Strings)

获取应用的字符串

# ruby

strings = app\_strings

# python

driver.app\_strings

// java

driver.getAppStrings();

// javascript

driver.getAppStrings().then(function (appStrings) { /\*...\*/ })

// php

$this->appStrings();

$this->appStrings('ru');

// c#

driver.GetAppStrings();

// objective c

[driver appStrings];

[driver appStringsForLanguage:"@ru"];

按键事件 (Key Event)

给设备发送一个按键事件

# ruby

key\_event 176

# python

driver.keyevent(176)

// java

driver.sendKeyEvent(AndroidKeyCode.HOME);

// javascript

driver.deviceKeyEvent(wd.SPECIAL\_KEYS.Home)

// php

$this->keyEvent('176');

// c#

driver.KeyEvent("176");

// objective c

NSError \*err;

[driver triggerKeyEvent:176 metastate:0 error:&err];

当前 Activity

获取当前 activity。只能在 Android 上使用

# ruby

current\_activity

# python

driver.current\_activity

// java

driver.currentActivity();

// javascript

driver.getCurrentActivity().then(function (activity) { /\*...\*/ })

// php

$this->currentActivity();

// c#

driver.GetCurrentActivity();

// objective c

NSError \*err;

[driver currentActivity];

触摸动作(TouchAction) / 多点触摸动作(MultiTouchAction)

生成触摸动作的接口。这部分文档很快将会补充更多的内容进来。

# ruby

touch\_action = Appium::TouchAction.new

element = find\_element :name, 'Buttons, Various uses of UIButton'

touch\_action.press(element: element, x: 10, y: 10).perform

# python

action = TouchAction(driver)

action.press(element=el, x=10, y=10).release().perform()

// java

TouchAction action = new TouchAction(driver)

.press(mapview, 10, 10)

.release().

perform();

// javascript

var action = new wd.TouchAction(driver);

action

.tap({el: el, x: 10, y: 10})

.release();

return action.perform(); // returns a promise

// php

$action = $this->initiateTouchAction();

->press(array('element' => $el))

->release()

->perform();

$action1 = $this->initiateTouchAction();

$action1->press(array('element' => $els[0]))

->moveTo(array('x' => 10, 'y' => 0))

->moveTo(array('x' => 10, 'y' => -75))

->moveTo(array('x' => 10, 'y' => -600))

->release();

$action2 = $this->initiateTouchAction();

$action2->press(array('element' => $els[1]))

->moveTo(array('x' => 10, 'y' => 10))

->moveTo(array('x' => 10, 'y' => -300))

->moveTo(array('x' => 10, 'y' => -600))

->release();

$multiAction = $this->initiateMultiAction();

$multiAction->add($action1);

$multiAction->add($action2);

$multiAction->perform();

// c#

ITouchAction action = new TouchAction(driver);

action.Press(el, 10, 10).Release();

action.Perform ();

滑动(Swipe)

模拟用户滑动

# ruby

swipe start\_x: 75, start\_y: 500, end\_x: 75, end\_y: 0, duration: 0.8

# python

driver.swipe(start=75, starty=500, endx=75, endy=0, duration=800)

// java

driver.swipe(75, 500, 75, 0, 0.8)

// javascript

function swipe(opts) {

var action = new wd.TouchAction(this);

action

.press({x: opts.startX, y: opts.startY})

.wait(opts.duration)

.moveTo({x: opts.endX, y: opts.endY})

.release();

return action.perform();

}

wd.addPromiseChainMethod('swipe', swipe);

// ...

return driver.swipe({ startX: 75, startY: 500,

endX: 75, endY: 0, duration: 800 });

// php

$this->swipe(75, 500, 75, 0, 800);

// c#

todo: c#

捏 (Pinch)

捏屏幕 (双指往内移动来缩小屏幕)

# ruby

pinch 75

# python

driver.pinch(element=el)

// java

driver.pinch(element);

// javascript

function pinch(el) {

return Q.all([

el.getSize(),

el.getLocation(),

]).then(function(res) {

var size = res[0];

var loc = res[1];

var center = {

x: loc.x + size.width / 2,

y: loc.y + size.height / 2

};

var a1 = new wd.TouchAction(this);

a1.press({el: el, x: center.x, y:center.y - 100}).moveTo({el: el}).release();

var a2 = new wd.TouchAction(this);

a2.press({el: el, x: center.x, y: center.y + 100}).moveTo({el: el}).release();

var m = new wd.MultiAction(this);

m.add(a1, a2);

return m.perform();

}.bind(this));

};

wd.addPromiseChainMethod('pinch', pinch);

wd.addElementPromiseChainMethod('pinch', function() {

return this.browser.pinch(this);

});

// ...

return driver.pinch(el);

// ...

return el.pinch();

$this->pinch($el);

// c#

driver.Pinch(25, 25)

放大 (Zoom)

放大屏幕 (双指往外移动来放大屏幕)

# ruby

zoom 200

# python

driver.zoom(element=el)

// java

driver.zoom(element);

// javascript

function zoom(el) {

return Q.all([

this.getWindowSize(),

this.getLocation(el),

]).then(function(res) {

var size = res[0];

var loc = res[1];

var center = {

x: loc.x + size.width / 2,

y: loc.y + size.height / 2

};

var a1 = new wd.TouchAction(this);

a1.press({el: el}).moveTo({el: el, x: center.x, y: center.y - 100}).release();

var a2 = new wd.TouchAction(this);

a2.press({el: el}).moveTo({el: el, x: center.x, y: center.y + 100}).release();

var m = new wd.MultiAction(this);

m.add(a1, a2);

return m.perform();

}.bind(this));

};

wd.addPromiseChainMethod('zoom', zoom);

wd.addElementPromiseChainMethod('zoom', function() {

return this.browser.zoom(this);

});

// ...

return driver.zoom(el);

// ...

return el.zoom();

// php

$this->zoom($el);

// c#

driver.Zoom(100, 200);

滑动到 (Scroll To)

滑动到某个元素。

# ruby

element = find\_element :name, 'Element Name'

execute\_script "mobile: scrollTo", :element => element.ref

# python

todo: python

// java

WebElement element = driver.findElement(By.name("Element Name"));

HashMap<String, String> arguments = new HashMap<String, String>();

arguments.put("element", element.getId());

(JavascriptExecutor)driver.executeScript("mobile: scrollTo", arguments);

// javascript

return driver.elementByName().then(function (el) {

return driver.execute('mobile: scrollTo', {element: el.value});

});

// php

$els = $this->elements($this->using('class name')->value('android.widget.TextView'));

$this->scroll($els[count($els) - 1], $els[0]);

// c#

todo: csharp

拉出文件 (Pull File)

从设备中拉出文件

# ruby

pull\_file 'Library/AddressBook/AddressBook.sqlitedb'

# python

driver.pull\_file('Library/AddressBook/AddressBook.sqlitedb')

// java

driver.pullFile("Library/AddressBook/AddressBook.sqlitedb");

// javascript

driver.pullFile("Library/AddressBook/AddressBook.sqlitedb")

.then(function (base64File) { /\*...\*/ })

// php

$this->pullFile('Library/AddressBook/AddressBook.sqlitedb');

// c#

driver.PullFile("Library/AddressBook/AddressBook.sqlitedb");

推送文件(Push file)

推送文件到设备中去

# ruby

data = "some data for the file"

path = "/data/local/tmp/file.txt"

push\_file path, data

# python

data = "some data for the file"

path = "/data/local/tmp/file.txt"

driver.push\_file(path, data.encode('base64'))

// java

byte[] data = Base64.encodeBase64("some data for the file".getBytes());

String path = "/data/local/tmp/file.txt";

driver.pushFile(path, data)

// javascript

driver.pushFile(path, data)

// php

$path = 'data/local/tmp/test\_push\_file.txt';

$data = 'This is the contents of the file to push to the device.';

$this->pushFile($path, base64\_encode($data));

// c#

driver.PushFile("/data/local/tmp/file.txt", "some data for the file");

设置

从这里你可以获取/设置 appium 的服务器设置。

想知道它如何工作，以及它支持哪些设置，请查看关于设置的文档

current\_settings = get\_settings

update\_settings someSetting: true

current\_settings = driver.get\_settings()

driver.update\_settings({"someSetting": true})

JsonObject settings = driver.getSettings()

// java-client doesn't support setting arbitrary settings, just settings which are already provided by appium.

// So for the 'ignoreUnimportantViews' setting, the following method exists:

driver.ignoreUnimportantViews(true);

var settings = driver.settings();

browser.updateSettings({'someSetting': true});

$settings = $this->getSettings();

$this->updateSettings(array('cyberdelia' => "open"));

Dictionary<String, Object>settings = driver.GetSettings();

// dotnet-driver doesn't support setting arbitrary settings, just settings which are already provided by appium.

// So for the 'ignoreUnimportantViews' setting, the following method exists:

driver.IgnoreUnimportantViews(true);

Appium 桌面应用

Appium 的桌面应用支持 OS X 和 Windows.

Appium.app for OS X

Appium.exe for Windows

5.2 Appium 服务关键字

Appium 服务关键字

<expand\_table>

关键字 描述 实例

automationName 你想使用的自动化测试引擎 Appium (默认) 或 Selendroid

platformName 你要测试的手机操作系统 iOS, Android, 或 FirefoxOS

platformVersion 手机操作系统版本 例如： 7.1, 4.4

deviceName 使用的手机类型或模拟器类型 iPhone Simulator, iPad Simulator, iPhone Retina 4-inch, Android Emulator, Galaxy S4, 等。在 iOS 上，这个关键字的值必须是使用 instruments -s devices 得到的可使用的设备名称之一。在 Android 上，这个关键字目前不起作用。

app .ipa or .apk文件所在的本地绝对路径或者远程路径,也可以是一个包括两者之一的.zip。 Appium会先尝试安装路径对应的应用在适当的真机或模拟器上。针对Android系统，如果你指定app-package和app-activity(具体见下面)的话，那么就可以不指定app。 会与 browserName 冲突 比如/abs/path/to/my.apk或http://myapp.com/app.ipa

browserName 需要进行自动化测试的手机 web 浏览器名称。如果是对应用进行自动化测试，这个关键字的值应为空。 iOS 系统上可以用 'Safari' ，Android 系统上可以用 'Chrome', 'Chromium', 或 'Browser'。

newCommandTimeout 设置命令超时时间，单位：秒。达到超时时间仍未接收到新的命令时 Appium 会假设客户端退出然后自动结束会话。 比如 60

autoLaunch Appium是否需要自动安装和启动应用。默认值true true, false

language (Sim/Emu-only) 设定模拟器 ( simulator / emulator ) 的语言。 如： fr

locale (Sim/Emu-only) 设定模拟器 ( simulator / emulator ) 的区域设置。 如： fr\_CA

udid 连接的物理设备的唯一设备标识 如： 1ae203187fc012g

orientation (Sim/Emu-only) 在一个设定的方向模式中开始测试 LANDSCAPE (横向) 或 PORTRAIT (纵向)

autoWebview 直接转换到 WebView 上下文。 默认值 false、 true, false

noReset 不要在会话前重置应用状态。默认值false。 true, false

fullReset (iOS) 删除整个模拟器目录。(Android) 通过卸载——而不是清空数据——来重置应用状态。在 Android 上，这也会在会话结束后自动清除被测应用。默认值 false true, false

Android特有

关键字 描述 实例

appActivity 你要从你的应用包中启动的 Android Activity 名称。它通常需要在前面添加 . (如：使用.MainActivity 而不是 MainActivity) MainActivity, .Settings

appPackage 你想运行的Android应用的包名 比如com.example.android.myApp, com.android.settings

appWaitActivity 你想要等待启动的 Android Activity 名称 SplashActivity

deviceReadyTimeout 设置等待一个模拟器或真机准备就绪的超时时间 5

androidCoverage 用于执行测试的 instrumentation 类。作为命令 adb shell am instrument -e coverage true -w 的 -w 参数。 com.my.Pkg/com.my.Pkg.instrumentation.MyInstrumentation

enablePerformanceLogging (仅适用于 Chrome 和 webview) 开启 Chromedriver 的性能日志。 (默认 false) true, false

androidDeviceReadyTimeout 等待设备在启动应用后准备就绪的超时时间。以秒为单位。 如 30

androidDeviceSocket 开发工具的 socket 名称。只有在被测应用是一个使用 Chromium 内核的浏览器时需要。 socket 会被浏览器打开，然后 Chromedriver 把它作为开发者工具来进行连接。 如 chrome\_devtools\_remote

avd 需要启动的 AVD (安卓虚拟设备) 名称。 如 api19

avdLaunchTimeout 以毫秒为单位，等待 AVD 启动并连接到 ADB 的超时时间。(默认值 120000) 300000

avdReadyTimeout 以毫秒为单位，等待 AVD 完成启动动画的超时时间。(默认值 120000) 300000

avdArgs 启动 AVD 时需要加入的额外的参数。 如 -netfast

useKeystore 使用一个自定义的 keystore 来对 apk 进行重签名。默认值 false true or false

keystorePath 自定义 keystore 的路径。默认： ~/.android/debug.keystore 如 /path/to.keystore

keystorePassword 自定义 keystore 的密码。 如 foo

keyAlias key 的别名 如 androiddebugkey

keyPassword key 的密码 如 foo

chromedriverExecutable webdriver 可执行文件的绝对路径 (如果 Chromium 核心提供了对应的 webdriver， 应该用它代替 Appium 自带的 webdriver) /abs/path/to/webdriver

autoWebviewTimeout 以毫秒为单位，等待 Webview 上下文激活的时间。默认值 2000 如 4

intentAction 用于启动 activity 的 intent action。 (默认值 android.intent.action.MAIN) 如 android.intent.action.MAIN, android.intent.action.VIEW

intentCategory 用于启动 activity 的 intent category。 (默认值 android.intent.category.LAUNCHER) 如 android.intent.category.LAUNCHER, android.intent.category.APP\_CONTACTS

intentFlags 用于启动 activity 的标识 ( flags ) (默认值 0x10200000) 如 0x10200000

optionalIntentArguments 用于启动 activity 的额外 intent 参数。请查看 Intent 参数 如 --esn <EXTRA\_KEY>, --ez <EXTRA\_KEY> <EXTRA\_BOOLEAN\_VALUE>

stopAppOnReset 在使用 adb 启动应用前停止被测应用的进程 ( process ) 。如果被测应用是被另一个应用创建的，当这个参数被设定为 false 时，允许另一个应用的进程在使用 adb 启动被测应用时继续存活。默认值 true true 或 false

unicodeKeyboard 使用 Unicode 输入法。默认值 false true 或 false

resetKeyboard 在设定了 unicodeKeyboard 关键字的 Unicode 测试结束后，重置输入法到原有状态。如果单独使用，将会被忽略。默认值 false true 或 false

noSign 跳过检查和对应用进行 debug 签名的步骤。只能在使用 UiAutomator 时使用，使用 selendroid 是不行。默认值 false true 或 false

ignoreUnimportantViews 调用 uiautomator 的函数 setCompressedLayoutHierarchy()。由于 Accessibility 命令在忽略部分元素的情况下执行速度会加快，这个关键字能加快测试执行的速度。被忽略的元素将不能够被找到，因此这个关键字同时也被实现成可以随时改变的 \*设置 ( settings ) \* 。默认值 false true 或 false

iOS特有

calendarFormat (Sim-only) 为iOS的模拟器设置日历格式 如 gregorian (公历)

bundleId 被测应用的 bundle ID 。用于在真实设备中启动测试，也用于使用其他需要 bundle ID 的关键字启动测试。在使用 bundle ID 在真实设备上执行测试时，你可以不提供 app 关键字，但你必须提供 udid 。 如 io.appium.TestApp

udid 连接的真实设备的唯一设备编号 ( Unique device identifier ) 如 1ae203187fc012g

launchTimeout 以毫秒为单位，在 Appium 运行失败之前设置一个等待 instruments 的时间 比如： 20000

locationServicesEnabled (Sim-only) 强制打开或关闭定位服务。默认值是保持当前模拟器的设定 true 或 false

locationServicesAuthorized (Sim-only) 通过修改 plist 文件设定是否允许应用使用定位服务，从而避免定位服务的警告出现。默认值是保持当前模拟器的设定。请注意在使用这个关键字时，你同时需要使用 bundleId 关键字来发送你的应用的 bundle ID。 true 或者 false

autoAcceptAlerts 当 iOS 的个人信息访问警告 (如 位置、联系人、图片) 出现时，自动选择接受( Accept )。默认值 false。 true 或者 false

autoDismissAlerts 当 iOS 的个人信息访问警告 (如 位置、联系人、图片) 出现时，自动选择不接受( Dismiss )。默认值 false。 true 或者 false

nativeInstrumentsLib 使用原生 intruments 库 (即关闭 instruments-without-delay ) true 或者 false

nativeWebTap (Sim-only) 在Safari中允许"真实的"，非基于 javascript 的 web 点击 (tap) 。 默认值： false。注意：取决于 viewport 大小/比例， 点击操作不一定能精确地点中对应的元素。 true 或者 false

safariInitialUrl (Sim-only) (>= 8.1) 初始化 safari 的时使用的地址。默认是一个本地的欢迎页面 如 https://www.github.com

safariAllowPopups (Sim-only) 允许 javascript 在 Safari 中创建新窗口。默认保持模拟器当前设置。 true 或者 false

safariIgnoreFraudWarning (Sim-only) 阻止 Safari 显示此网站可能存在风险的警告。默认保持浏览器当前设置。 true 或者 false

safariOpenLinksInBackground (Sim-only) Safari 是否允许链接在新窗口打开。默认保持浏览器当前设置。 true 或者 false

keepKeyChains (Sim-only) 当 Appium 会话开始/结束时是否保留存放密码存放记录 (keychains) (库(Library)/钥匙串(Keychains)) true 或者 false

localizableStringsDir 从哪里查找本地化字符串。默认值 en.lproj en.lproj

processArguments 通过 instruments 传递到 AUT 的参数 如 -myflag

interKeyDelay 以毫秒为单位，按下每一个按键之间的延迟时间。 如 100

showIOSLog 是否在 Appium 的日志中显示设备的日志。默认值 false true 或者 false

sendKeyStrategy 输入文字到文字框的策略。模拟器默认值：oneByOne (一个接着一个) 。真实设备默认值：grouped (分组输入) oneByOne, grouped 或 setValue

screenshotWaitTimeout 以秒为单位，生成屏幕截图的最长等待时间。默认值： 10。 如 5

waitForAppScript 用于判断 "应用是否被启动” 的 iOS 自动化脚本代码。默认情况下系统等待直到页面内容非空。结果必须是布尔类型。 例如 true;, target.elements().length > 0;, $.delay(5000); true;

5.3 元素定位

元素定位与交互

Appium支持webdriver定位策略的子集

根据"class"定位(例如， UI组件类型)

根据"xpath"定位 (例如，具有一定约束的路径抽象标示, 基于XPath方式)

另外， Appium 还支持部分 Mobile JSON 连接协议 的定位策略

ios uiautomation: 一个递归地、对应使用 UIAutomation library 搜索元素的字符串(iOS-only)

android uiautomator: 一个递归地、对应使用 UiAutomator Api搜索元素的字符串 (Android-only)

accessibility id: 一个递归地、使用本地Accessibility选项实现的Id/Name进行元素搜索的字符串。

存在的问题

如果遇到定位元素变得无效请联系并告知我们。我们将会努力修复

使用Appium Inspector来定位元素

(翻译备注: 这个工具目前只有Mac版本, 如果你使用的是windows, 可以使用android sdk自带的 uiautomatorviewer 工具来获得元素的位置)

Appium提供了一个灵活的工具Appium Inspector, 允许你在app运行的时候, 直接定位你正在关注的元素. 通过Appium Inspector(靠近start test按钮的小"i"按钮), 你可以通过点击预览窗口上的控件来获得它的name属性, 或者直接在UI导航窗口中定位

概述

Appium Inspector有一个简单的布局, 全部由如下窗口组成.

UI导航器, 预览, 录制与刷新按钮, 和交互工具

Step 1

例子

启动Appium Inspector后(通过点击app右上的小"i"按钮), 你可以定位任何预览窗口中的元素. 作为测试, 我正在查找id为"show alert"的按钮

Step 1

要找到这个按钮的id, 在定位预览窗口中我点击了"show alert"按钮, Appium Inspector在UI导航窗口中高亮显示了这个元素, 然后展示了刚被点击按钮的id和元素类型

Step 1

5.4 IOS谓词

iOS 谓词(Predicate)

在查看 '-ios uiautomation' 搜索策略时了解 谓词(Predicate) 十分必要。 UIAutomation JavaScript API有下列几种非常有用的方法:

(UIAElement) UIAElementArray.firstWithPredicate(PredicateString predicateString)

(UIAElementArray) UIAElementArray.withPredicate(PredicateString predicateString)

原生的JS搜索策略（由Apple提供）提供了更大的灵活性，并且和XPath很像。

谓词(Predicate) 可以通过使用多个匹配条件来准确定位某一个或某一组元素（相当于只有搜索条件与元素的计算结果为 true 时这些元素才会被认为是匹配的）。

（翻译备注：XPath 是一门用来定位 xml 文档中的元素的语言，能提供基于路径、元素属性等条件的定位策略）

例如:

// java

appiumDriver.findElementsByIosUIAutomation("collectionViews()[0].cells().withPredicate(\"ANY staticTexts.isVisible == TRUE\")")

- 将只选择那些在主视图第一个 UIACollectionView 元素下的、拥有可见子元素 UIAStaticText 的 UIACollectionCell 元素。在这里， staticTexts() 和 isVisible() 分别是UIAElementArray 和 UIAElement 的子方法。 注意： UIAElementArray 序列编号从 0 开始，而不是像 Xpath 那样从 1开始

以下是所有可用的谓词(Predicate)的列表（主要取自 谓词(Predicate) 编程指南)

基本比较

= , ==

- 左边表达式等于右边表达式:

tableViews()[1].cells().firstWithPredicate("label == 'Olivia' ")

same in Xpath: /UIATableView[2]/UIATableCell[@label = 'Olivia'][1]

>= , =>

- 左边表达式大于或等于右边表达式。

<= , =<

- 左边表达式小于或等于右边表达式。

>

- 左边表达式大于右边表达式。

<

- 左边表达式小于右边表达式。

!= , <>

- 左边表达式不等于右边表达式。

BETWEEN

- 左边表达式的值在右边表达式的两个边界值之间或等于其中一个边界值。右边表达式为一个有两个值的数组，数组的第一个值是上限，第二个值是下限(这个顺序是固定的) ，例如 1 BETWEEN { 0 , 33 }， 或者 $INPUT BETWEEN { $LOWER, $UPPER }。

在 Objective-C， 你可以创建一个自定义的 BETWEEN 谓词(Predicate)，如下面的示例所示:

NSPredicate \*betweenPredicate =

[NSPredicate predicateWithFormat: @"attributeName BETWEEN %@", @[@1, @10]];

这创建了一个等价于 ( ( 1 <= attributeValue ) && ( attributeValue <= 10 ) ) 的谓词

布尔值谓词

TRUEPREDICATE

- 计算结果恒等于 TRUE 。

FALSEPREDICATE

- 计算结果恒等于 FALSE。

基本的复合谓词

AND , &&

- 逻辑与。

OR , ||

- 逻辑或。

NOT , !

- 逻辑非。

字符串比较

在默认情况下，字符串比较是区分大小写和音调( diacritic )的，你可以在方括号中用关键字符 c 和 d 来修改操作符以相应的指定不区分大小写和变音符号。例如 名字的开头 firstName BEGINSWITH[cd] $FIRST\_NAME

（翻译备注：这里的音调是指英文字母的音调，如 "náive" 和 "naive"。如果不加关键字 d，这两个字符串会认为是不等价的。）

BEGINSWITH

- 左边的表达式以右边的表达式作为开始。

scrollViews()[3].buttons().firstWithPredicate("name BEGINSWITH 'results toggle' ")

same in Xpath: /UIAScrollView[4]/UIAButton[starts-with(@name, 'results toggle')][1]

CONTAINS

- 左边的表达式包含右边的表达式。

tableViews()[1].cells().withPredicate("ANY collectionViews[0].buttons.name CONTAINS 'opera'")

same in Xpath: /UIATableView[2]/UIATableCell[UIACollectionView[1]/UIAButton[contains(@name, 'opera')]]

ENDSWITH

- 左边的表达式以右边的表达式作为结束。

LIKE

- 左边表达式等于右边表达式: ? 和 \*可作为通配符， 其中 ? 匹配 1 个字符， \* 匹配 0 个或者多个字符。 在 Mac OS X v10.4， 通配符不能匹配换行符。

tableViews()[0].cells().firstWithPredicate("name LIKE '\*Total: $\*' ")

same in Xpath: /UIATableView[1]/UIATableCell[matches(@name, '.\*Total: \$.\*')][1]

MATCHES

- 左边表达式根据ICU v3的正则表达式风格比较，等于右边表达式 (详情请看ICU用户指南中的 正则表达式)。

tableViews().firstWithPredicate("value MATCHES '.\*of 7' ")

same in Xpath: /UIATableView[matches(@value, '.\*of 7')][1]

聚合操作

ANY , SOME

- 指定匹配后续表达式的任意元素。例如 ANY children.age < 18 。

tableViews()[0].cells().firstWithPredicate("SOME staticTexts.name = 'red'").staticTexts().withName('red')

same in Xpath: /UIATableView[1]/UIATableCell[UIAStaticText/@name = 'red'][1]/UIAStaticText[@name = 'red']

ALL

- 指定匹配后续表达式的所有元素。例如 ALL children.age < 18 。

NONE

- 指定不匹配后续表达式的元素。例如 NONE children.age < 18 。 逻辑上等价于 NOT (ANY ...) 。

IN

- 等价于 SQL 的 IN 操作，左边的表达必须出现在右边指定的集合中。例如 name IN { 'Ben', 'Melissa', 'Matthew' } 。 这个集合可以是一个数组( array )，一个列表( set )， 或者一个字典( dictionary )。当这个集合是字典时，这里使用的是它的值( value )。

array[index]

- 指定数组中特定索引处的元素。

array[FIRST]

- 指定数组中的第一个元素。

array[LAST]

- 指定数组中的最后一个元素。

array[SIZE]

- 指定数组的大小

elements()[0].tableViews()[0].cells().withPredicate("staticTexts[SIZE] > 2")

same in Xpath: /\*[1]/UIATableView[1]/UIATableCell[count(UIAStaticText) > 2]

标识符

C语言标识符

- 任何C语言的标识符都不是保留字。

#symbol

- 用来把一个保留字转义为用户标识符。

[\]{octaldigit}{3}

- 用来表示一个八进制数 ( \后面加上3位八进制数字)。

[\][xX]{hexdigit}{2}

- 用于表示十六进制数 ( \x 或 \X 后面加上2个十六进制数字)。

[\][uU]{hexdigit}{4}

- 用于表示 Unicode 编码 ( \u 或 \U 后面加上4个十六进制数字)。

文字 (Literals)

（翻译备注：Literals 在编程语言领域的意思是在代码中可以看得到的(或说可视的)那些值。例如字符串 "a"，数组 [1, 2]，你可以在代码中一眼看出这是一个字符串，数组还是别的数据类型并知道它的值。这一节说的就是这些值的写法）

单引号和双引号都能产生相同的结果，但他们不会匹配对方（单引号不会匹配双引号）。例如："abc" and 'abc' 都是可识别的 ，但是 "a'b'c" 等价于a, 'b', c。

FALSE , NO

- 表示逻辑上的 false。

TRUE , YES

- 表示逻辑上的 true。

NULL , NIL

- 空值。

SELF

- 代表被使用的对象本身。

"text"

- 一个字符串。

'text'

- 同上，也是一个字符串。

以逗号分隔的文本数组

- 举个例子 { 'comma', 'separated', 'literal', 'array' } 。

标准的整数和小数

- 举个例子 1 , 27 , 2.71828 , 19.75 。

带有幂指数的小数

- 举个例子 9.2e-5 。

0x

- 十六进制数的前缀， 如0x11表示十六进制数11，等同于十进制的17。

0o

- 八进制数的前缀。

0b

- 二进制数的前缀。

保留字

下面的都是保留字：

AND, OR, IN, NOT, ALL, ANY, SOME, NONE, LIKE, CASEINSENSITIVE, CI, MATCHES, CONTAINS, BEGINSWITH, ENDSWITH, BETWEEN, NULL, NIL, SELF, TRUE, YES, FALSE, NO, FIRST, LAST, SIZE, ANYKEY, SUBQUERY, CAST, TRUEPREDICATE, FALSEPREDICATE

Appium 谓词(predicate)帮助文档

Appium 在app.js中有 专门的谓词(predicate)使用帮助文档 :

getFirstWithPredicate

getFirstWithPredicateWeighted

getAllWithPredicate

getNameContains

如下是个Ruby的例子

# Ruby example

text = 'Various uses'

predicate = "name contains[c] '#{text}' || label contains[c] '#{text}' || value contains[c] '#{text}'"

element = execute\_script(%Q(au.mainApp().getFirstWithPredicate("#{predicate}");))

puts element.name # Buttons, Various uses of UIButton

5.5自动化手机网页应用

自动化手机网页应用

如果你正对于如何在iOS的Safari或Android上的Chrome做网页应用的自动化感兴趣，

那么Appium能够帮助你。基本上，你可以正常的写webdriver测试，只需要把Appium当

成一个有特殊设置的selenium Server。

iOS模拟器上的Safari浏览器

首先，要确保你的Safari浏览器参数中开启了开发者模式，这样Safari的远程调试端口也会被同时打开。

不管你使用模拟器还是真机，你必须使用Appium开始之前先开启Safari。

然后设置如下显示的这些信息以便于在设备中的Safari执行测试：

``

javascript

// javascript

{

platformName: 'iOS'

, platformVersion: '7.1'

, browserName: 'Safari'

, deviceName: 'iPhone Simulator'

}

```python

# python

{

'platformName': 'iOS',

'platformVersion': '7.1',

'browserName': 'Safari',

'deviceName': 'iPhone Simulator'

}

// php

public static $browsers = array(

array(

'desiredCapabilities' => array(

'platformName' => 'iOS',

'platformVersion' => '7.1',

'browserName' => 'Safari',

'deviceName' => 'iPhone Simulator'

)

)

);

// java

DesiredCapabilities capabilities = new DesiredCapabilities();

capabilities.setCapability(MobileCapabilityType.PLATFORM\_NAME, "iOS");

capabilities.setCapability(MobileCapabilityType.PLATFORM\_VERSION, "7.1");

capabilities.setCapability(MobileCapabilityType.BROWSER\_NAME, "Safari");

capabilities.setCapability(MobileCapabilityType.DEVICE\_NAME, "iPhone Simulator");

iOS真机上的Safari浏览器

为了能够在真机上的Safari执行测试，我们使用了SafariLauncher App来启动Safari。

一旦Safari被启动，则使用ios-webkit-webkit-proxy来自动启动Safari的远程调试功能。

提示: 目前在ios-webkit-debug-proxy中有一个问题。

你必须添加信任才能开始运行ios-webkit-debug-proxy。

前期设置

当你要在真机上的Safari中执行你的测试脚本之前你需要先注意以下几点：

安装并运行 ios-webkit-debug-proxy，并监听27753端口 (具体可以参考(hybrid docs)

打开iOS真机中的 web inspector，可以在iOS6.0或更高版本中的 设置 > safari > 高级找到。

创建一个 provisioning profile 能够帮助你配置safariLauncher。

你可以前往 Apple Developers Member Center 创建一个launcher profile:

\* 第一步: 创建一个 新的App Id 同时设置WildCard App ID这个选项置为""

\* \*第二步:\*\* 为步骤1的App Id创建一个 new Development Profile 。

\* 第三步: 选择你的 certificate(s) and device(s) 并选择下一步。

\* 第四步: 设置profile的名称以及 generate the profile。

\* 第五步: 下载profile并使用文本编辑器打开。

\* 第六步: 寻找并牢记你的 UUID

现在你有了自己的profile文件，可以在终端中输入如下的命令:

$ git clone https://github.com/appium/appium.git

$ cd appium

# 选项1:你可以不设置任何的参数。appium会把签名 (code signing identity) 设为'iPhone Developer'

$ ./reset.sh --ios --real-safari

# 选项2:你需要定义code signing identity并且允许xcode选择profile identity code

$ ./reset.sh --ios --real-safari --code-sign '<code signing idendity>'

# 选项3:你需要设置<code signing idendity>和<profile identity code>

$ ./reset.sh --ios --real-safari --code-sign '<code signing idendity>' --profile '<retrieved profile identity code>'

# 设置成功之后，就可以像往常一样启动服务

$ node /lib/server/main.js -U <UDID>

执行测试

如果要在safari下的运行你的测试, 只需要简单的配置"browserName"为safari即可

Java 范例

// java

// 配置web driver并启动webview应用

DesiredCapabilities desiredCapabilities = new DesiredCapabilities();

desiredCapabilities.setCapability(MobileCapabilityType.BROWSER\_NAME, "Safari");

URL url = new URL("http://127.0.0.1:4723/wd/hub");

AppiumDriver driver = new AppiumDriver(url, desiredCapabilities);

// 跳转到指定页面并在该页面所以用元素id进行交互

driver.get("http://saucelabs.com/test/guinea-pig");

WebElement div = driver.findElement(By.id("i\_am\_an\_id"));

Assert.assertEquals("I am a div", div.getText()); //跳转到指定页面并在该页面所以用元素id进行交互

driver.findElement(By.id("comments")).sendKeys("My comment"); //通过id查找评论框并输入

// 关闭应用

driver.quit();

Python 范例

# python

# 配置web driver并启动webview应用

capabilities = { 'browserName': 'Safari' }

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

# 跳转到指定页面并在该页面所以用元素id进行交互

driver.get('http://saucelabs.com/test/guinea-pig');

div = driver.find\_element\_by\_id('i\_am\_an\_id')

# 检查文本是否符合预期

assertEqual('I am a div', div.text)

# 通过id查找评论框并输入

driver.find\_element\_by\_id('comments').send\_keys('My comment')

# 关闭应用

driver.quit()

// php

class ContextTests extends PHPUnit\_Extensions\_AppiumTestCase

{

public static $browsers = array(

array(

'desiredCapabilities' => array(

'platformName' => 'iOS',

'platformVersion' => '7.1',

'browserName' => 'Safari',

'deviceName' => 'iPhone Simulator'

)

)

);

public function testThings()

{

$this->get('http://saucelabs.com/test/guinea-pig');

$div = $this->byId('i\_am\_an\_id');

$this->assertEquals('I am a div', $div->text());

$this->byId('comments')->sendKeys('My comment');

}

}

在真机或模拟器上的Chrome执行测试

需要做的准备:

确认Chrome已经安装在了你的真机或模拟器上 (应用的包名是com.android.chrome) 。在不编译Chromium的情况下, 不可能得到模拟器上的x86版本的chrome，你可以运行一个ARM的模拟器然后从真机上获取一个Chrome的APK安装在模拟器上。

如果你是使用NPM下载的， 或者是在.app运行的话，那你不需要其他额外的工作。如果你是使用源码运行，reset会下载ChromeDriver并放在build。 使用 --chromedriver-version 选项可以指定chromedriver的版本 (例如 ./reset.sh --android --chromedriver-version 2.8)， 否则使用最新版。

接着,像这样设置就可以在Chrome上执行测试了:

// javascript

{

platformName: 'Android'

, platformVersion: '4.4'

, deviceName: 'Android Emulator'

, browserName: 'Chrome'

};

# python

{

'platformName': 'Android',

'platformVersion': '4.4',

'deviceName': 'Android Emulator',

'browserName': 'Chrome'

}

// php

public static $browsers = array(

array(

'desiredCapabilities' => array(

'platformName' => 'Android',

'platformVersion' => '4.4',

'browserName' => 'Chrome',

'deviceName' => 'Android Emulator'

)

)

);

// java

DesiredCapabilities capabilities = new DesiredCapabilities();

capabilities.setCapability(MobileCapabilityType.PLATFORM\_NAME, "Android");

capabilities.setCapability(MobileCapabilityType.PLATFORM\_VERSION, "4.4");

capabilities.setCapability(MobileCapabilityType.DEVICE\_NAME, "Android Emulator");

capabilities.setCapability(MobileCapabilityType.BROWSER\_NAME, "Chrome");

在4.4以上的版本，你也可以用'Browser' browserName 来对内置浏览器进行自动化。

在所有版本你都可以用'Chromium' browserName来对Chromium进行自动化。

chromedriver故障排查

从Chrome 33开始，不再必须将设备root。在之前的版本，设备必须按要求进行root (ChromeDriver需要写 /data/local 目录来设定Chrome的命令行参数) 。

如果在版本33之前在Chrome上测试app，确保adb shell拥有设备中/data/local目录的读写权限：

$ adb shell su -c chmod 777 /data/local

更多关于chromedriver的文档详见ChromeDriver documentation。

5.6 调整网络设置

调整网络设置

Selenium 的 Mobile JSON Wire Protocol Specification 支持一个获取和设置设备网络连接的 API 。这个 API 通过位掩码（bitmask）工作，把所有可能的状态用一个整型数据表示：

值 (别名) 数据连接 Wifi 连接 飞行模式

0 (什么都没有) 0 0 0

1 (飞行模式) 0 0 1

2 (只有Wifi) 0 1 0

4 (只有数据连接) 1 0 0

6 (开启所有网络) 1 1 0

翻译备注：数据链接即2g, 3g, 4g的网络连接。

iOS

很不幸，目前 Appium 在 iOS 下不支持 Selenium 的网络连接 API。

Android

选择你想使用的设置，然后根据上面的表格发送正确的位掩码(bitmask)。

// javascript

// 设置网络连接为飞行模式

driver.setNetworkConnection(1)

// 设置网络连接为仅启用Wifi

driver.setNetworkConnection(2)

// 设置网络连接为仅启用数据连接

driver.setNetworkConnection(4)

// 设置网络连接为启用数据连接和Wifi

driver.setNetworkConnection(6)

获取网络连接设置会返回基于同样规则的位掩码，你可以将其解码来获得网络设置。

// javascript

driver.getNetworkConnection().then(function (connectionType) {

switch (connectionType) {

case 0:

// 无网络

break;

case 1:

// 飞行模式

break;

case 2:

// wifi

break;

case 4:

// 数据连接

break;

case 6:

// wifi和数据连接

break;

}

});

5.7 执行测试

执行测试

准备被测应用 (iOS)

在模拟器上测试apps必须要用模拟器专用的编译器，例如使用下列的命令来编译Xcode项目：

> xcodebuild -sdk iphonesimulator6.0

这行指令在Xcode项目底下创建了一个build/Release-iphonesimulator目录，并且生成一个可以透过Appium服务器来通讯的的.app封包。

如果需要，你可以把.app 目录压缩成一个zip压缩档！ Appium 会自行解压缩。让你能方便在非本地运行Appium。

准备被测应用 (Android)

用Appium去执行你的.apk档其实没什么特别需要注意的事项。如果需要，你可以把它压缩成zip压缩档。

用Appium测试你的app (iOS)

想知道如何编写测试脚本，请参照测试范例：

Node.js | Python | PHP | Ruby | Java

基本上来说，首先先确定你启动了Appium:

node .

然后执行你的WebDriver测试脚本，脚本必须包含下列的环境参数：

// javascript

{

platformName: 'iOS',

platformVersion: '7.1',

deviceName: 'iPhone Simulator',

app: myApp

}

# python

{

'platformName': 'iOS',

'platformVersion': '7.1',

'deviceName': 'iPhone Simulator',

'app': myApp

}

// php

public static $browsers = array(

array(

'desiredCapabilities' => array(

'platformName' => 'iOS',

'platformVersion' => '7.1',

'deviceName' => 'iPhone Simulator',

'app' => $myApp

)

)

);

// java

DesiredCapabilities capabilities = new DesiredCapabilities();

capabilities.setCapability(MobileCapabilityType.PLATFORM\_NAME, "iOS");

capabilities.setCapability(MobileCapabilityType.PLATFORM\_VERSION, "7.1");

capabilities.setCapability(MobileCapabilityType.DEVICE\_NAME, "iPhone Simulator");

capabilities.setCapability(MobileCapabilityType.APP, myApp);

在这个脚本集里，myApp必须是下列其中之一：

一个模拟器编译过的.app 目录或者.zip 文件的本地绝对路径

一个包含着你的.app封包的zip档的url

appium安装根目录下的一个示例app的相对路径

在你选择的WebDriver库里，设定remote session使用上述的环境参数然后使用端口4723来连接本地服务器(或者使用你在Appium启动时所设定的端口)。现在你已经完成设置了！

用Appium测试你的app (Android)

首先，先确定你有一个而且必须是只能一个Android模拟器或者设备连接着。如果你输入adb devices，你应该只看到一个设备连接着。这将是Appium所用来测试的设备。当然，要连接一个设备，你需要准备好一个Android AVD (参考(Windows，Mac，或者Linux)以了解更多)。 如果Android SDK工具在你的环境变量path下，你可以简单的执行:

emulator -avd <我的Avd名称>

然后等android模拟器启动。有时候，因为某些原因，adb会卡住。如果它没有显示任何的设备或其他故障，你可以使用下列指令来重启:

adb kill-server && adb devices

现在，确认Appium已经启动:

node .

有几种方法来启动一个Appium程序（效果和通过adb启动一模一样）：

只有apk或者zip，默认activity将会被启动。 (只设置了'app'环境参数)

apk + activity ('app' + 'appActivity' 环境参数)

apk + activity + intent ('app' + 'appActivity' + 'appIntent' 环境参数)

...

Activities 可以通过以下方式来指定:

名称 (如 appActivity: 'com.helloworld.SayHello')。

相对于 appPackage的路径 (如 appPackage: 'com.helloworld', appActivity='.SayHello')

如果“appWaitPackage'和'appWaitActivity”被指定，Appium

将自动等待，直到这些活动的被启动。你可以为实例指定多个等待的activity：

appActivity: 'com.splash.SplashScreen'

appPackage: 'com.splash' appActivity: '.SplashScreen'

appPackage: 'com.splash' appActivity: '.SplashScreen,.LandingPage,com.why.GoThere'

如果你不是很清楚在apk中有哪些activity，你可以通过以下方式来查看：

Mac/Linux: 'adb shell dumpsys window windows | grep mFocusedApp'

在 Ruby 控制台运行: 'adb shell dumpsys window windows`.each\_line.grep(/mFocusedApp/).first.strip'

在 Windows 终端运行 'adb shell dumpsys window windows' 然后去看mFocusedApp这一行的内容。

然后执行你的WebDriver测试脚本，脚本必须包含下列的环境参数：

// javascript

{

platformName: 'Android',

platformVersion: '4.4',

deviceName: 'Android Emulator',

app: myApp

}

# python

{

'platformName': 'Android',

'platformVersion': '4.4',

'deviceName': 'Android Emulator',

'app': myApp

}

// php

public static $browsers = array(

array(

'desiredCapabilities' => array(

'platformName' => 'Android',

'platformVersion' => '4.4',

'deviceName' => 'Android Emulator',

'app' => $myApp

)

)

);

// java

DesiredCapabilities capabilities = new DesiredCapabilities();

capabilities.setCapability(MobileCapabilityType.PLATFORM\_NAME, "Android");

capabilities.setCapability(MobileCapabilityType.PLATFORM\_VERSION, "4.4");

capabilities.setCapability(MobileCapabilityType.DEVICE\_NAME, "Android Emulator");

capabilities.setCapability(MobileCapabilityType.APP, myApp);

在这个脚本集里，myApp必须是下列其中之一：

一个.apk 或者.zip 档的本地绝对路径

一个包含着你的.apk档的zip压缩档的url

appium安装根目录下的一个示例app的路径

myAppPackage 必须是你的应用的java package，例如, com.example.android.myApp。

myAppActivity 必须是你的希望测试的Android activity, 例如, MainActivity。

在你选择的WebDriver库里，设定remote session使用上述的环境参数然后使用端口4723来连接本地服务器(或者是使用你在Appium启动时所设定的任意端口)。现在你已经设置完成了！

用Appium测试你的app (Android 设备 < 4.2, 以及混合app测试)

低于4.2版本的Android设备(API Level 17) 没有安装 Google 的UiAutomator framework.下面的范例是早期Appium在这些设备上的测试方法。对于早期的设备以及使用混合模式(webview-based)制作的apps, Appium 包含了另一种自动化测试工具Selendroid。

要使用Selendroid, 只需要在之前提到的环境参数上稍作修改即可，添加 automationName 参数并指定Seledroid作为测试工具。通常你还需要在你的activity名称前加上. (如：在appActivity参数中使用.MainActivity 而不是 MainActivity) :

// javascript

{

automationName: 'Selendroid',

platformName: 'Android',

platformVersion: '2.3',

deviceName: 'Android Emulator',

app: myApp,

appPackage: 'com.mycompany.package',

appActivity: '.MainActivity'

}

# python

{

'automationName': 'Selendroid',

'platformName': 'Android',

'platformVersion': '2.3',

'deviceName': 'Android Emulator',

'app': myApp,

'appPackage': 'com.mycompany.package',

'appActivity': '.MainActivity'

}

// php

public static $browsers = array(

array(

'desiredCapabilities' => array(

'automationName' => 'Selendroid',

'platformName' => 'Android',

'platformVersion' => '2.3',

'deviceName' => 'Android Emulator',

'app' => $myApp,

'appPackage' => 'com.mycompany.package',

'appActivity'=> '.MainActivity'

)

)

);

// java

DesiredCapabilities capabilities = new DesiredCapabilities();

capabilities.setCapability(MobileCapabilityType.AUTOMATION\_NAME, "Selendroid");

capabilities.setCapability(MobileCapabilityType.PLATFORM\_NAME, "Android");

capabilities.setCapability(MobileCapabilityType.PLATFORM\_VERSION, "2.3");

capabilities.setCapability(MobileCapabilityType.DEVICE\_NAME, "Android Emulator");

capabilities.setCapability(MobileCapabilityType.APP, myApp);

capabilities.setCapability(MobileCapabilityType.APP\_PACKAGE: "com.mycompany.package");

capabilities.setCapability(MobileCapabilityType.APP\_ACTIVITY: ".MainActivity");

这样Appium就会启动 Selendroid 测试会话取代默认的测试会话。使用Selendroid的缺点是有时候它的API跟Appium非常不同。所以我们建议你在为你的旧设备或者混合app写测试脚本之前先仔细的阅读Selendroid的说明文档。

5.8 服务器参数

Appium 服务器参数

使用方法: node . [标志]

服务器标志

所有的标志都是可选的，但是有一些标志需要组合在一起才能生效。

<expand\_table>

标志 默认值 描述 例子

--shell null 进入 REPL 模式

--localizable-strings-dir en.lproj IOS only: 定位 .strings所在目录的相对路径 --localizable-strings-dir en.lproj

--app null iOS: 基于模拟器编译的 app 的绝对路径或者设备目标的 bundle\_id； Android: apk 文件的绝对路径--app /abs/path/to/my.app

--ipa null (IOS-only) .ipa 文件的绝对路径 --ipa /abs/path/to/my.ipa

-U, --udid null 连接物理设备的唯一设备标识符 --udid 1adsf-sdfas-asdf-123sdf

-a, --address 0.0.0.0 监听的 ip 地址 --address 0.0.0.0

-p, --port 4723 监听的端口 --port 4723

-ca, --callback-address null 回调IP地址 (默认: 相同的IP地址) --callback-address 127.0.0.1

-cp, --callback-port null 回调端口号 (默认: 相同的端口号) --callback-port 4723

-bp, --bootstrap-port 4724 (Android-only) 连接设备的端口号 --bootstrap-port 4724

-k, --keep-artifacts false 弃用，无效。trace信息现在保留tmp目录下，每次运行前会清除该目录中的信息。 也可以参考 --trace-dir 。

-r, --backend-retries 3 (iOS-only) 遇到 crash 或者 超时，Instrument 重新启动的次数。 --backend-retries 3

--session-override false 允许 session 被覆盖 (冲突的话)

--full-reset false (iOS) 删除整个模拟器目录。 (Android) 通过卸载应用（而不是清除数据）重置应用状态。在 Android 上，session 完成后也会删除应用。

--no-reset false session 之间不重置应用状态 (iOS: 不删除应用的 plist 文件； Android: 在创建一个新的 session 前不删除应用。)

-l, --pre-launch false 在第一个 session 前，预启动应用 (iOS 需要 --app 参数，Android 需要 --app-pkg 和 --app-activity)

-lt, --launch-timeout 90000 (iOS-only) 等待 Instruments 启动的时间

-g, --log null 将日志输出到指定文件 --log /path/to/appium.log

--log-level debug 日志级别; 默认 (console[:file]): debug[:debug] --log-level debug

--log-timestamp false 在终端输出里显示时间戳

--local-timezone false 使用本地时间戳

--log-no-colors false 不在终端输出中显示颜色

-G, --webhook null 同时发送日志到 HTTP 监听器 --webhook localhost:9876

--native-instruments-lib false (IOS-only) iOS 内建了一个怪异的不可能避免的延迟。我们在 Appium 里修复了它。如果你想用原来的，你可以使用这个参数。

--app-pkg null (Android-only) 你要运行的apk的java包。 (例如， com.example.android.myApp) --app-pkg com.example.android.myApp

--app-activity null (Android-only) 打开应用时，启动的 Activity 的名字(比如， MainActivity) --app-activity MainActivity

--app-wait-package false (Android-only) 你想等待的 Activity 的包名。(比如， com.example.android.myApp) --app-wait-package com.example.android.myApp

--app-wait-activity false (Android-only) 你想等待的 Activity 名字(比如， SplashActivity) --app-wait-activity SplashActivity

--android-coverage false (Android-only) 完全符合条件的 instrumentation 类。 作为命令 adb shell am instrument -e coverage true -w 的 -w 的参数 --android-coverage com.my.Pkg/com.my.Pkg.instrumentation.MyInstrumentation

--avd null (Android-only) 要启动的 avd 的名字

--avd-args null (Android-only) 添加额外的参数给要启动avd --avd-args -no-snapshot-load

--device-ready-timeout 5 (Android-only) 等待设备准备好的时间，以秒为单位 --device-ready-timeout 5

--safari false (IOS-Only) 使用 Safari 应用

--device-name null 待使用的移动设备名字 --device-name iPhone Retina (4-inch), Android Emulator

--platform-name null 移动平台的名称: iOS, Android, or FirefoxOS --platform-name iOS

--platform-version null 移动平台的版本 --platform-version 7.1

--automation-name null 自动化工具的名称: Appium or Selendroid --automation-name Appium

--browser-name null 移动浏览器的名称: Safari or Chrome --browser-name Safari

--default-device, -dd false (IOS-Simulator-only) 使用instruments自己启动的默认模拟器

--force-iphone false (IOS-only) 无论应用要用什么模拟器，强制使用 iPhone 模拟器

--force-ipad false (IOS-only) 无论应用要用什么模拟器，强制使用 iPad 模拟器

--language null iOS / Android 模拟器的语言 --language en

--locale null Locale for the iOS simulator / Android Emulator --locale en\_US

--calendar-format null (IOS-only) iOS 模拟器的日历格式 --calendar-format gregorian

--orientation null (IOS-only) 初始化请求时，使用 LANDSCAPE (横屏) 或者 PORTRAIT (竖屏) --orientation LANDSCAPE

--tracetemplate null (IOS-only) 指定 Instruments 使用的 tracetemplate 文件 --tracetemplate /Users/me/Automation.tracetemplate

--show-sim-log false (IOS-only) 如果设置了， iOS 模拟器的日志会写到终端上来

--show-ios-log false (IOS-only) 如果设置了， iOS 系统的日志会写到终端上来

--nodeconfig null 指定 JSON 格式的配置文件 ，用来在 selenium grid 里注册 appiumd --nodeconfig /abs/path/to/nodeconfig.json

-ra, --robot-address 0.0.0.0 robot 的 ip 地址 --robot-address 0.0.0.0

-rp, --robot-port -1 robot 的端口地址 --robot-port 4242

--selendroid-port 8080 用来和 Selendroid 交互的本地端口 --selendroid-port 8080

--chromedriver-port 9515 ChromeDriver运行的端口 --chromedriver-port 9515

--chromedriver-executable null ChromeDriver 可执行文件的完整路径

--use-keystore false (Android-only) 设置签名 apk 的 keystore

--keystore-path (Android-only) keystore 的路径

--keystore-password android (Android-only) keystore 的密码

--key-alias androiddebugkey (Android-only) Key 的别名

--key-password android (Android-only) Key 的密码

--show-config false 打印 Appium 服务器的配置信息，然后退出

--no-perms-check false 跳过Appium对是否可以读/写必要文件的检查

--command-timeout 60 默认所有会话的接收命令超时时间 (在超时时间内没有接收到新命令，自动关闭会话)。 会被新的超时时间覆盖

--keep-keychains false (iOS) 当 Appium 启动或者关闭的时候，是否保留 keychains (Library/Keychains)

--strict-caps false 如果所选设备是appium不承认的有效设备，会导致会话失败

--isolate-sim-device false Xcode 6存在一个bug，那就是一些平台上如果其他模拟器设备先被删除时某个特定的模拟器只能在没有任何错误的情况下被建立。这个选项导致了Appium不得不删除除了正在使用设备以外其他所有的设备。请注意这是永久性删除，你可以使用simctl或xcode管理被Appium使用的设备类别。

--tmp null 可以被Appium用来管理临时文件的目录（绝对路径），比如存放需要移动的内置iOS应用程序。 默认的变量为 APPIUM\_TMP\_DIR ，在 \*nix/Mac 为 /tmp 在windows上使用环境便令 TEMP 设定的目录。

--trace-dir null 用于保存iOS instruments trace的 appium 目录，是绝对路径， 默认为 <tmp dir>/appium-instruments

--intent-action android.intent.action.MAIN (Android-only) 用于启动 activity 的intent action --intent-action android.intent.action.MAIN

--intent-category android.intent.category.LAUNCHER (Android-only) 用于启动 activity 的intent category --intent-category android.intent.category.APP\_CONTACTS

--intent-flags 0x10200000 (Android-only) 启动 activity 的标志 --intent-flags 0x10200000

--intent-args null (Android-only) 启动 activity 时附带额外的 intent 参数 --intent-args 0x10200000

--suppress-adb-kill-server false (Android-only) 如果被设定，阻止Appium杀掉adb实例。

5.9 移动手势的自动化

移动手势的自动化

虽然Selenium WebDriver的规范支持数种手机交互的方式，但它的参数并不能简单地映射到底层设备使用的自动化函数 (像在iOS上的UIAutomation) 。为此，Appium在规范的最新版本中定义了新的触摸操作/多点触控 API

(https://dvcs.w3.org/hg/webdriver/raw-file/tip/webdriver-spec.html#multiactions-1)。

注意，这跟在早期版本中使用原始JSON Wire Protocol 的触摸操作 API不同。

这些API可以让你使用多个驱动来建立任意手势。请参阅对应语言的Appium客户端文档，就可以找到使用这些API的例子。

触摸操作/多点触控 API的概述

触摸操作 (TouchAction)

TouchAction 对象包含一连串的事件。

在所有的appium客户端库中，触摸对象创建并给出一连串的事件。

规范中的可用事件有：

\* 短按 (press)

\* 释放 (release)

\* 移动到 (moveTo)

\* 点击 (tap)

\* 等待 (wait)

\* 长按 (longPress)

\* 取消 (cancel)

\* 执行 (perform)

这里有一个通过伪代码创建动作的例子：

TouchAction().press(el0).moveTo(el1).release()

上述模拟用户按下一个元素，滑动他的手指到另一个位置，然后从屏幕上释放其手指。

Appium按顺序执行这些事件。你可以添加一个 wait 事件来控制相应手势的时间。

appium客户端库有不同的方式来实现上述例子，比如：你可以传递一个坐标值或一个元素给 moveTo 事件。同时传递坐标和元素，会将坐标和元素对应起来，但这不是绝对的。

调用 perform 事件发送整个事件序列给appium，从而使触摸手势在设备上运行。

Appium客户端还允许人们直接通过驱动程序对象执行触摸操作, 而不是调用触摸操作对象的perform事件。

在伪代码中，以下两个是等价的：

TouchAction().tap(el).perform()

driver.perform(TouchAction().tap(el))

多点触控 (MultiTouch)

MultiTouch 对象是触摸操作的集合。

多点触控手势只有两个方法，添加 (add) 和执行 (perform) 。

add 用于将不同的触摸操作添加到一个多点触控中。

当 perform 被调用的时候，所有被添加到多点触摸中的触摸事件会被发送到appium并且被执行，就像它们同时发生一样。Appium会执行“触摸事件”中的第一个事件，然后第二个，以此类推。

用两只手指点击的代码示例：

action0 = TouchAction().tap(el)

action1 = TouchAction().tap(el)

MultiAction().add(action0).add(action1).perform()

缺陷和解决方法

不幸的是有一个缺陷存在于iOS的7.x的模拟器上，ScrollViews无法识别由UIAutomation创建的手势 (在iOS上Appium使用的是UIAutomation) 。 为了实现此功能，我们已经提供了新的函数， scroll, 在大部分情况下可以让你实现跟ScrollView一样的功能！

滚动

要使用这特殊的功能，我们重写了driver中的 execute 和

executeScript 方法。 可以通过在命令前加 mobile: 的前缀来使用滚动。

请参见下面的例子：

WD.js:

// javascript

// 把视图往下滑动

driver.execute("mobile: scroll", [{direction: 'down'}])

// 继续测试

Java:

// java

JavascriptExecutor js = (JavascriptExecutor) driver;

HashMap<String, String> scrollObject = new HashMap<String, String>();

scrollObject.put("direction", "down");

scrollObject.put("element", ((RemoteWebElement) element).getId());

js.executeScript("mobile: scroll", scrollObject);

滑块的自动化

iOS

Java

// java

// 滑动值使用0到1之间的数字以字符串的形式表示

// 例如，“0.1”代表10％，“1.0”代表100％

WebElement slider = driver.findElement(By.xpath("//window[1]/slider[1]"));

slider.sendKeys("0.1");

Android

与Android上的滑块进行交互的最佳方式是用触摸操作 (TouchActions) 。

5.10 uiautomator UiSelector

uiautomator UiSelector

Appium可以使用 UiSelectors

进行元素查找，同时也支持UiScrollable

.

注意：根据索引 (index) 进行查找并不可靠，请使用实例 (instance) 代替. 下面的示范是用Ruby语言编写的、针对 api demo (这是一个 appium 测试用的应用) 的实例。

翻译者备注：UiSelectors 和 UiScrollable 均是 Android UiAutomator 中的对象，因此以下用法仅适用于 Android 。

找到第一个文本控件 (TextView) 。

# ruby

first\_textview = find\_element(:uiautomator, 'new UiSelector().className("android.widget.TextView").instance(0)');

根据文本 (text) 找到第一个元素。

# ruby

first\_text = find\_element(:uiautomator, 'new UiSelector().text("Animation")')

first\_text.text # "Animation"

找到第一个可滚动的元素， 然后找到文本是 "Tabs" 的文本控件。

"Tabs" 元素就是将要滚动到的控件。

# ruby

element = find\_element(:uiautomator, 'new UiScrollable(new UiSelector().scrollable(true).instance(0)).getChildByText(new UiSelector().className("android.widget.TextView"), "Tabs")')

scrollIntoView 是一个特例，会返回滚动到指定控件的元素。

scrollIntoView 对任何的 UiSelector 都可以执行滚动操作。

# ruby

element = find\_element(:uiautomator, 'new UiScrollable(new UiSelector().scrollable(true).instance(0)).scrollIntoView(new UiSelector().text("WebView").instance(0));')

element.text # "WebView"

5.11 多语言支持

多语言支持

程序处理非拉丁字符时存在一个的问题：对于带音标的字符，存在多种编码形式。例如，对于é这样的字符，有两种编码方式：一种是单独的字符é（Unicode中的LATIN SMALL LETTER E WITH ACUTE(带有音标的小写的拉丁字母'E'))，另一种是音标在字符后边(COMBINING ACUTE ACCENT(字符和音标的组合))。为了解决这个问题，存在一个normalization (标准化) 方法，让"每个字符都有一个唯一的二进制表示"。

幸运的是，对ASCII字符（例如 不需要进行标准化的字符）进行标准化将不会产生任何变化，并且多次进行标准化

操作也不会发生额外的变化。因此，可以对所有字符使用标准化函数而不用担心产生不良影响。

// javascript

var unorm = require('unorm');

'some ASCII text' === unorm.nfd('some ASCII text');

unorm.nfd('Adélaïde Hervé') === unorm.nfd(unorm.nfd('Adélaïde Hervé'));

在测试的时候遇到Unicode字符，你需要对字符进行标准化，确保期望的值和接收到的值一致。

有很多方法可以用来进行标准化，所以你要确保执行的是同样的方法！

// javascript

var unorm = require('unorm');

driver

.elementByAccessibilityId('find')

.text()

.then(function (txt) {

unorm.nfd(txt).should.be(unorm.nfd("é Œ ù ḍ"));

});

一个由不同unicode文本编码导致的问题的标志是断言失败但报告却显示两个看起来一模一样的字符串：

AssertionError: expected 'François Gérard' to deeply equal 'François Gérard'

+ expected - actual

+"François Gérard"

-"François Gérard"

当发生只因编码导致的问题时，输出看上去一样。从标准的角度，它们的编码应该也和它们看上去那样相同。

查找器 (Finder)

需要被查找的字符也应该需要标准化。比如，你在一个iOS的app上有一个叫做Найти的按钮，你也应该在find命令中标准化它。

// javascript

var unorm = require('unorm');

driver

.findElementByXPath(unorm.nfd("//UIAButton[@name='Найти']"))

.should.eventually.exist;

否则这个按钮可能无法被找到。

文本框 (Text Field)

默认情况下，iOS和Android的自动化工具都不支持向输入框输入非ASCII字符。

iOS

Appium 完全绕过键盘直接向iOS设备的输入框发送非ASCII字符。虽然这让这些文本在测试中被成功输入，但必须记住由键盘输入触发的业务逻辑将不会被测试到。

像上边说的一样，断言收到的文本前应该先标准化它。

// javascript

var unorm = require('unorm');

var testText = unorm.nfd("é Œ ù ḍ");

driver

.elementsByClassName('UIATextField').at(1)

.sendKeys(testText)

.text()

.should.become(testText)

.nodeify(done);

Android

通过下载并安装一个特殊键盘 ， Android 可以支持输入 Unicode 字符，这个输入法允许文本通过ASCII在Appium和被测应用之间进行通讯。

为了使用这个功能，将unicodeKeyboard设置为true。如果想要键盘设置在测试完成后自动回到原始状态，

将resetKeyboard设置为true。否则Appium测试结束后，Appium的Unicode键盘仍然会被激活。

翻译备注：这个Unicode键盘并非虚拟键盘，在界面上不会显示出来，所以要进行其他类型的测试必须切换回其他输入法。

测试时可以通过send\_keys向输入框输入Unicode字符。

// javascript

var desired = {

app: '/path/to/app',

deviceName: 'Android Emulator',

deviceVersion: '4.4',

platformName: 'Android',

unicodeKeyboard: true,

resetKeyboard: true

};

var testText = 'é Œ ù ḍ';

driver

.elementByClassName('android.widget.EditText')

.sendKeys(testText)

.text()

.should.eventually.become(testText)

.nodeify(done);