3. appium命令与xpath

appium要想到**灵活与自然**，不能光靠appium desktop生成的代码，这些代码有很多无用的地方。例如你使用xpath时，会自动生成长达30字符的路径，但实际上，我们只用5字符就能轻松定位。本小节会介绍appium常用命令(java版本)，以及xpath的使用。

环境

PYTHON开发用的工具：

1. Android设备
2. Appium
3. `PyCharm`

> pycharm比较容易安装，大家看视频即可。

JAVA开发用的工具有：

1. Android设备
2. Appium
3. `IDEA`

前两个工具我们前面已经安装好，下面主要介绍IDEA和Maven。
IDEA：之所以要选择IntelliJ IDEA，是因为Maven+Eclipse实在太难用了。Eclipse有两个Maven插件 m2eclipse 和 Eclipse IAM。但是由于Eclipse和Maven设计上的矛盾，这两个插件可能都达不到你想要的效果。m2eclipse 会把你所有类型的项目都视为Maven类型项目，这对于一个单纯的Java项目或许可以，对于一个JEE或者WEB项目都是一场灾难。相比之下，IntelliJ IDEA做的要好的多，以至于你很难说出对它的不满。
Maven：我们需要它完美的依赖性管理，Maven会把所有依赖的包放在本机的一个目录下，所以实际上是脱离Project本身存在的。IntelliJ IDEA引入了一个External Library的概念，所有的Maven依赖性都会放在这里，和项目自带的库区分开。并且Module之间会智能的判断，你不需要Maven Install来进行引用代码的更新。
具体安装过程，大家可以观看霍格沃兹视频。

命令

这里的介绍和举例采用Python命令，如果有其它需要，可以访问官网查寻自己语言对应的命令。
官网：http://appium.io/

一，控件

1.1 查找元素By.id

```
driver.find_element_by_id("")
```

在driver下通过id查找一个元素，此用法通常适用于当前界面的driver**有且仅有一个唯一**的id元素标示。它有固定查找顺序，先从resourceId中寻找，如果没有，再去accessibilityId，最后到Strings.xml中查找，优先级递减。

由于find_element_by_id的要求很严苛，推荐使用xpath寻找。

1.2 查找元素ByName

```
find_element_by_class_name()
```

通过元素name查找当前页面的一个元素，使用方式与find_element_by_id相同，只是把匹配条件由id变为name。请参考find_element_by_id的调用方式

1.3 查找元素ByAccessibilityId

```
find_element_by_accessibility_id()
```

在uiautomatorviewer中，content-desc内容即为accessibility_id，在selenium库里可以用find_element_by_name()来匹配content-desc的内容；在Appium库里则用find_element_by_accessibility_id()来匹配content-desc的内容。因为Appium继承了Selenium类，所以如果find_element_by_name无法准确定位时，请试试看find_element_by_accessibility_id。

1.4 查找元素ByXpath

```
find_element_by_xpath("")
```

xpath是一个获取XML文档中你所需要的节点元素的组件。它允许你用**很少的代码**就能获取指定的路径下你所选取的节点的值，xpath一直是我们提倡的东西，后面我们介绍xpath语法。
同时，与本条命令相似：

```
find_elements_by_xpath()
```

通过元素xpath查找当前页面的多个目标元素，返回的是一个List。

1.5 其它

常用的获取控件类API就是以上这些。其他的查找和匹配的api还有find_element_by_link_text、find_elements_by_link_text、find_element_by_tag_name、find_elements_by_tag_name、find_element_by_css_selector、find_elements_by_css_selector等，用法都与上述类似。

二，元素操作

2.1 点击

```
click()
tap()
```

click和tap都能实现单击的效果。其区别在于click是作用于**driverelement**的实例化对象，而tap是对屏幕上的**坐标位置**进行点击。
click常用在find_element之后，比如：

```
el = self.driver.find_element_by_accessibility_id('SomeId')
el.click();
```

2.2 输入

```
send_keys()
set_text()
```

send_keys和set_text也都能满足输入文本内容的操作。其区别在于send_keys会调用设备当前系统输入法**键盘**，而set_text直接对目标元素设置**文本**。由此可推，send_keys的输入内容往往和预期内容不一致，而set_text的输入则是直接赋值，并不是键盘事件。
send_keys举例：

```
self.driver.find_element_by_accessibility_id('SomeAccessibilityID').send_keys(
```

2.3 滑动

```
swipe(int start x,int start y,int end x,int y,duration)
flick(int start x,int start y,int end x)
```

- int start x - 开始滑动的x坐标;
- int start y - 开始滑动的y坐标；
- int end x - 结束点x坐标;
- int end y - 结束点y坐标;
- duration 滑动时间（默认5毫秒）；

swipe和flick都是滑动操作，它们都是从[start_x, start_y]划到[end_x, end_y]的过程，唯一不同的是swipe比flick多了一个**duration**参数，有了这个参数就可以自定义从start到end动作的作用时间，以达到快速滑动或者慢速滑动的效果。

屏幕左上角为起点，坐标为（0，0）

2.4 长按

```
long_press()
```

长按方法是在**TouchAction类**中，所以在使用时需要先`import TouchAction`。
长按的的使用稍微有点复杂，例子如下：

```
action1 = TouchAction(self.driver)
driver_element = driver.find_element_by_xpath("sport_history_item_xpath")
action1.long_press(driver_element).wait(i * 1000).perform()
```

上面的代码中， i为长按控件的时间，**单位秒**。

三，其他

3.1 截屏

```
get_screenshot_as_file()
```

截屏还是很有用地，它的使用也很简单，接受一个保存的图片路径和名称的参数：

```
driver.get_screenshot_as_file('../screenshot/1.png')
```

通常来说，截屏要配合滑动一起食用更佳~~

3.2 获取控件各种属性

```
get_attribute()
```

获取属性也是常用api，用法：

```
driver.find_element_by_id().get_attribute(name)
```

name有很多我们熟悉的标志，比如`class，package，checkable，checked….`。返回值为**str**类型，即便是true or false，但是其实是"true" or "false"

3.3 等待

等待是非常有用的命令，试想这样的情况，你的手机卡了一下，页面加载慢了几秒，造成的结果就是查找元素失败，因此非常有必要加入等待方法。

- 第一种 sleep()：
  设置固定休眠时间。python 的 time 包提供了休眠方法 sleep()，导入 time包后就可以使用 sleep() 进行脚本的执行过程进行休眠。

```
#导入 time 包
import time
time.sleep()
```

- 第二种 implicitly_wait()：隐式等待
  它是 webdirver 提供的一个超时等待。隐式等待一个元素被发现，或一个命令完成。如果超出了设置时间仍未定位到元素则抛出异常。
  当使用了隐士等待执行测试的时候，如果 WebDriver没有在 DOM中找到元素，将继续等待，超出设定时间后则抛出找不到元素的异常。换句话说，当查找元素或元素并没有立即出现的时候，隐式等待将等待一段时间再查找 DOM，默认的时间是0。一旦设置了隐式等待，则它存在整个 WebDriver 对象实例的声明周期中，隐式的等到会让一个正常响应的应用的测试变慢，它将会在寻找每个元素的时候都进行等待，这样会增加整个测试执行的时间。

> implicitly_wait()方法比 sleep() 更加智能，后者只能选择一个固定的时间的等待，前者可以在一个时间范围内智能的等待。

```
driver.implicitly_wait(10)  #全局等待10s
```

- 第三种 WebDriverWait()：显式等待
  同样也是 webdirver 提供的方法。在设置时间内，默认每隔一段时间检测一次当前。页面元素是否存在，如果超过设置时间检测不到则抛出异常。
  详细格式如下：

```
WebDriverWait(driver, timeout, poll_frequency=0.5, ignored_exceptions=None)
```

driver： WebDriver 的驱动程序(Ie, Firefox, Chrome 或远程)
timeout：最长超时时间，默认以秒为单位
poll_frequency：休眠时间的间隔（步长）时间，默认为 0.5 秒
ignored_exceptions：超时后的异常信息，默认情况下抛 – NoSuchElementException 异常。

WebDriverWai()一般和 `until()`或 `until_not()`方法配合使用，下面是 until()和 until_not()方法的说明。

- until(method, message=''): 调用该方法提供的驱动程序作为一个参数，直到返回值不为 False。
- until_not(method, message=''): 调用该方法提供的驱动程序作为一个参数，直到返回值为 False。

```
from selenium.webdriver.support.ui import WebDriverWait
element = WebDriverWait(driver, 10).until(lambda x: x.find_element_by_id("some
is_disappeared = WebDriverWait(driver, 30, 1, (ElementNotVisibleException)).un
```

如果还不懂的同学，可以看看这篇文章：显式等待与隐式等待

四，按键对应码

在**Android keyevent**事件中，不同的值代表了不同的含义和功能，比如手机的返回键：keyevent(4);手机的HOME键: keyevent(3)等等，下面的表格详细的记录了这些事件。

行为	数字
KEYCODE_UNKNOWN	0
KEYCODE_SOFT_LEFT	1
KEYCODE_SOFT_RIGHT	2
KEYCODE_HOME	3
KEYCODE_BACK	4
KEYCODE_CALL	5
KEYCODE_ENDCALL	6
KEYCODE_0	7
KEYCODE_1	8
KEYCODE_2	9
KEYCODE_3	10
KEYCODE_4	11
KEYCODE_5	12
KEYCODE_6	13
KEYCODE_7	14
KEYCODE_8	15
KEYCODE_9	16
KEYCODE_STAR	17
KEYCODE_POUND	18
KEYCODE_DPAD_UP	19
KEYCODE_DPAD_DOWN	20
KEYCODE_DPAD_LEFT	21
KEYCODE_DPAD_RIGHT	22

行为	数字
KEYCODE_DPAD_CENTER	23
KEYCODE_VOLUME_UP	24
KEYCODE_VOLUME_DOWN	25
KEYCODE_POWER	26
KEYCODE_CAMERA	27
KEYCODE_CLEAR	28
KEYCODE_A	29
KEYCODE_B	30
KEYCODE_C	31
KEYCODE_D	32
KEYCODE_E	33
KEYCODE_F	34
KEYCODE_G	35
KEYCODE_H	36
KEYCODE_I	37
KEYCODE_J	38
KEYCODE_	K
KEYCODE_L	40
KEYCODE_M	41
KEYCODE_N	42
KEYCODE_O	43
KEYCODE_P	44
KEYCODE_Q	45
KEYCODE_R	46
KEYCODE_S	47
KEYCODE_T	48
KEYCODE_U	49
KEYCODE_V	50

行为	数字
KEYCODE_W	51
KEYCODE_X	52
KEYCODE_Y	53
KEYCODE_Z	54
KEYCODE_COMMA	55
KEYCODE_PERIOD	56
KEYCODE_ALT_LEFT	57
KEYCODE_ALT_RIGHT	58
KEYCODE_SHIFT_LEFT	59
KEYCODE_SHIFT_RIGHT	60
KEYCODE_TAB	61
KEYCODE_SPACE	62
KEYCODE_SYM	63
KEYCODE_EXPLORER	64
KEYCODE_ENVELOPE	65
KEYCODE_ENTER	66
KEYCODE_DEL	67
KEYCODE_GRAVE	68
KEYCODE_MINUS	69
KEYCODE_EQUALS	70
KEYCODE_LEFT_BRACKET	71
KEYCODE_RIGHT_BRACKET	72
KEYCODE_BACKSLASH	73
KEYCODE_SEMICOLON	74
KEYCODE_APOSTROPHE	75
KEYCODE_SLASH	76
KEYCODE_AT	77
KEYCODE_NUM	78

行为	数字
KEYCODE_HEADSETHOOK	79
KEYCODE_FOCUS	80
KEYCODE_PLUS	81
KEYCODE_MENU	82
KEYCODE_NOTIFICATION	83
KEYCODE_SEARCH	84
KEYCODE_MEDIA_PLAY_PAUSE	85
KEYCODE_MEDIA_STOP	86
KEYCODE_MEDIA_NEXT	87
KEYCODE_MEDIA_PREVIOUS	88
KEYCODE_MEDIA_REWIND	89
KEYCODE_MEDIA_FAST_FORWARD	90
KEYCODE_MUTE	91

xpath

XPath 是一门在 XML 文档中查找信息的语言，它使用路径表达式来选取 XML 文档中的节点或者节点集。这些路径表达式和我们在常规的电脑文件系统中看到的表达式非常相似。

表达式	描述
/	从根节点选取。
//	从匹配选择的当前节点选择文档中的节点，而不考虑它们的位置。
.	选取当前节点。
..	选取当前节点的父节点。
@	选取属性。
*	匹配任何元素节点。
@*	匹配任何属性节点。
conatins()	对应字段包含值

在下面的表格中，我们已列出了一些路径表达式以及表达式的结果：

路径表达式	结果
霍格沃兹	选取 霍格沃兹 元素的所有子节点。
/霍格沃兹	选取根元素 霍格沃兹。
霍格沃兹/思寒	选取属于 霍格沃兹 的子元素的所有 思寒 元素。
//思寒	选取所有 思寒 子元素，而不管它们在文档中的位置。
霍格沃兹//思寒	选择属于 霍格沃兹 元素的后代的所有 思寒 元素，而不管它们位于 霍格沃兹 之下的什么位置。
//@lang	选取名为 lang 的所有属性。
/霍格沃兹/*	选取 霍格沃兹 元素的所有子元素。
//*	选取文档中的所有元素。
//title[@*]	选取所有带有属性的 title 元素。
/霍格沃兹/思寒[1]	选取属于 霍格沃兹 子元素的第一个 思寒 元素。
/霍格沃兹/思寒[last()]	选取属于 霍格沃兹 子元素的最后一个 思寒 元素。
/霍格沃兹/思寒[last()-1]	选取属于 霍格沃兹 子元素的倒数第二个 思寒 元素。
/霍格沃兹/思寒[position()< 3]	选取最前面的两个属于 霍格沃兹 元素的子元素的 思寒 元素。
//title[@lang]	选取所有拥有名为 lang 的属性的 title 元素。
//title[@lang='eng']	选取所有 title 元素，且这些元素拥有值为 eng 的 lang 属性。
/霍格沃兹/思寒[price>35.00]	选取 霍格沃兹 元素的所有 思寒 元素，且其中的 price 元素的值须大于 35.00。
/霍格沃兹/思寒[price>35.00]/title	选取 霍格沃兹 元素中的 思寒 元素的所有 title 元素，且其中的 price 元素的值须大于 35.00。

当然，对于appium，我们还要知道一些补充：

表达式	描述
//*[@text='登录']	选取只有"登录"字段的元素，比如"登录"
//*[@conatins(@text='登陆')]	只要包含"登录"字段，就会被选取，比如"登陆霍格沃兹"
//*[@conatins(@text='登陆') and contains(@text,'霍格沃兹')]	利用了and操作符，表示既包含"登陆"，又得包含"霍格沃兹"的字段，相同使用方法的还有or

上面的内容足够我们定位元素了，xpath还有更多的用法，比如轴和运算符。

相关链接

霍格沃兹测试学院官网首页:https://testing-studio.com