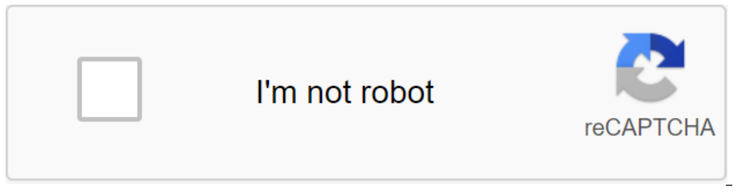
How to find element in appium android



Continue

MobileElementOne (MobileElement) driver.findElementByAccessibilityId (SomeAccessibilityID); MobileElement elementTwo (MobileElement) driver.findElementByClassName (SomeClassName); el - self.driver.find_element_by_accessibility_id ('SomeAccessibilityID') / webdriver.io example $(SomeAccessibilityId); wd example let elementOne and wait for the driver.elementByAccessibilityId (SomeAccessibilityID); Let elementTwo and wait for the driver.element (id, SomeID); for example ruby_lib find_element (:accessibility_id, 'SomeAccessibilityID') - ruby_lib_core for example, @driver.find-element (:accessibility_id, 'SomeAccessibilityID') AndroidElement elementOne. FindElementByAccesibilityID (Some ID); AndroidElement elementDWo and driver. FindElementByClassName (Some class); $el - $this-gt;accessibilityId (SomeAccessibilityID); Elements of MobileElement(ListOne) driver.findElementsByAccessibilityId (SomeAccessibilityID); The driver.findElementsByClassName (SomeClassName); el no self.driver.find_elements_by_accessibility_id (SomeAccessibilityID) / webdriver.io for example $$(SomeAccessibilityId); wd example let the elementsOne and wait for the driver.elementsByAccessibilityId (SomeAccessibilityID); Let the elementsTwo - wait for driver.elements (id, SomeID); - ruby_lib example of find_elements (:accessibility_id, SomeAccessibilityID) - ruby_lib_core for example @driver.find-elements (:accessibility_id, SomeAccessibilityID) IEnumerable'lt;AppiumWebElement'gt; elements. FindElementsByAccessibilityId (SomeAccessibilityID); IEnumerable'lt;AppiumWebElement'gt; elementsDwo and driver. FindElementsByClassName (SomeClassName); $els - $this--'gt; ($this-'gt;accessibility id'))) This article is the second in a series of several articles about test speed and reliability, inspired by a webinar I gave recently on the same topic (you can see the webinar here). You can also check out Part 1: Test Flakiness. One of the biggest superficial indicators of instability or flakiness is the element not found. Finding elements is a natural place of problem, because that's when we're trying to find an element, our assumption about the state of the application and its actual state are combined (whether in harmony or in conflict). We certainly want to avoid potential problems in finding items that we can do something about, such as using selectors that are not unique, or trying to find items on some dynamic attribute that can't be relied upon. This means that knowing your app and design is important. What can change? What's not? What items have ACCESSibility ID elements? Locator Strategies Before you go any further, it's worth saying what we mean by find and THE availability of ID, for example. In Appium (as in selenium), actions can be taken on the appium/AppiumWebElement/appium2Element. objects in the app's user interface. These objects (relevant to the elements of the web page, hence the name of the findElement API command) must be found before they can be interacted with. There are different ways to find items. Take a look at the example of the call below: In this example, By.className is a so-called locator strategy called class name, and Button is a selector that the strategy uses to find one or more items. The result of this call is (if all goes well) an object like WebElement that comes with a rich set of interaction APIs that you rely on for testing. The class name is just one of a number of locator strategies available in Appium, and refers to a platform specific UI of name class objects, such as XCUIElementTypeButton or android.widget.Button. Even now you can see that perhaps this locator strategy is not always perfect; What if you're testing a cross-platform app? Do you need to have a different set of code to find the iOS button or the Android button? If you rely on a class name locator strategy, the answer is yes. There is another problem with this strategy: there are often several elements of one type or another in the hierarchy. So you may well find a button with this locator strategy, but will it be the button you want? So we could say that a class name locator strategy is not a good choice because it is a platform specific (leading to branched iOS and Android code) and too generic (it is difficult to uniquely identify an item with). What other options are there? Take a look at this full set table: As you can see, many of the locator strategies have been moved from Selena, although not all are supported or even make sense in Appium (at least when automating your native app). Appium has also introduced a number of its own strategies, such as ID availability, to reflect the fact (and take advantage of the fact that we are dealing with a mobile UIs app and a completely different automation stack. XPath In another Appium Pro article, I go into details about the XPath locator strategy and why it should be avoided. To sum up here, many people find the XPath strategy attractive because it ensures that any user interface element can be found. The problem is that some items can only be found with the help of selectors that are fragile, meaning they can't find an item or other element if something changes in the design of the application. XPath can also be slow with Appium because it sometimes entails several recursive visualizations of the user interface hierarchy. Accessibility ID What should we use instead? When possible, I recommend using locator is an accessibility ID because it is (a) cross-platform, b) is unique, and (c) fast. Both iOS and Android have the concept of an accessibility tag, although on iOS it is called an accessibility ID, and on Android it is called content content (or content-desc). Because the availability label is a line set by developers, it can be a unique identifier (if developers use it in a way that I recommend them do, as long as it doesn't interfere with actual availability considerations). In appium Java, finding accessibility identifier items includes using MobileBy strategy: WebElement el s driver.findElement (MobileBy.AccessibilityID (foo); Because testers don't always have the ability to influence app development, sometimes availability labels are unavailable or not unique. iOS-specific locator strategies In the same Appium Pro article I referenced earlier, I've talked in detail about some iOS locator strategies that can be used as an XPath replacement because they are hierarchical query-based strategies. The most reliable strategy is the ios class chain strategy, which allows you to use a lite version of something like XPath mixed with iOS predicate lines. The advantage of this locator strategy is that it allows complex queries, staying in most cases much faster than the XPath. The downside, of course, is that it's specific to the platform, so it requires a branch of code (or adding additional differences to the object model). As an example of what you can do, check out this command: C/XCUIElementCellType driver.findElement (MobileBy.iOSClassChain); Here we find the 10th button, which is a child of the cell table anywhere in the user interface hierarchy, which has a name starting with the C symbol. Because of the tighter query form, the chain of class demands is better than XPath's. Android-specific strategy Locator Similar trick is available for Android, under the guise of a special parser team Appium implemented, which supports most of the UiSelector API. We make this parser available with the android locator strategy, and selectors must be strings that are valid bits of Java code, starting with the new UiSelector(). Let's take an example: String Selector is the new UiSelector.com getChildByText (new UiSelector ().className (android.widget.TextView), Tabs) ; driver.findElement (MobileBy.AndroidUIAutomator); Once again, we use MobileBy's strategy since this strategy is only available for Appium. What's happening here is that we've built a string that can be used as a valid UiAutomator test code, but will actually be disassembled and interpreted by Appium when you send the command. According to the semantics of the UiSelector API, we say that we want the first element we find with the text of the tab, which is also the child of the first ScrollView in the hierarchy. It's a little clunkier than XPath, but it's this used in a similar way, and again with a better performance profile in most cases. As with the iOS class chain strategy, the main drawback here is that selectors will be specific to the platform. (In addition, we cannot support arbitrary Java and there are limits to what we can provide from the UiSelector API). Determining which selectors to use so far we've seen some good recommendations on which strategies to use to find items reliably. But how do you know which selectors to use in conjunction with these strategies? I said before that knowledge of your app is essential to do this properly. How do you get that knowledge about your app? If you're one of the app developers, you might just take a look at the code, or maybe you'll remember that you've given a certain element of a certain accessibility tag. If you don't have access to the code, or if you want a method that will show you exactly what Appium sees in your app, it's best to use Appium Desktop. Appium Desktop is a GUI tool for launching Appium and checking apps. You can use it to run inspector sessions with arbitrary desired features. Inspector sessions show you a screenshot of your app, its user interface hierarchy (like XML) and a lot of metadata about any item you choose. It looks like this: One of the great things about the inspector is that when you click on an item in the hierarchy, it will be wise to offer a locator strategy and selectors to you. In the picture above, you can see that the top sentence for the selected item is the availability of the ID locator strategy used in conjunction with the selector's entry screen. Things can get a little more complicated, of course, but Appium Desktop Inspector is always a great place to start when figuring out what's going on with your application hierarchy. This is especially useful if you're working on issues where you think an item should exist on a certain view: just run the inspector and manually view the XML tree to see if the item actually exists. If it's not, it means that Appium (read: basic automation framework) can't see it, and you need to ask the app developer why. And this ends our discussion about the reliable search for items in Appium--- or at least one of its aspects. Just because you can find an item with the right locator strategy doesn't mean it will always be there when you look. Make sure to also check the next part, waiting for the status of the application (including the presence of items). elements). how to scroll and find element in appium android