UiAutomation自动化测试

专 业 的 I T 培 训 专 家

顾 翔

目录

- **Andriod 各种UI测试框架介绍**
- UiAutomator UI自动化测试框架
  - 环境准备
  - 建立测试工程
  - Uiautomator API详解
  - 建立测试集
  - 实践

Android 各种UI测试框架介绍

Andriod应用测试现状：

1. 移动应用改变着我们的生活；
2. Andriod手机数量全面超过IOS系统手机；
3. Andriod在各个智能领域逐步扩展；
4. Andriod应用的增加，带来测试行业的发展；
5. 手工黑盒测试占据绝大部分；
6. Andriod应用测试从业者逐年增加。

学习人群

1. 有志从事软件测试行业的人员
2. 当前正在从事APP软件测试开发的人员
3. 想充电，不断学习的人群

Android 各种UI测试框架介绍

MonkeyRunner

1. 编写语言：Python
2. 运行环境：Python 环境，adb链接PC运行
3. 测试对象：UI测试
4. 测试限制：主要使用坐标，逻辑判断能力差

- MonkeyRunner代码风格示例

```
device=MonkeyRunner.waitForConnection()#链接手机设备
device.press('KEYCODE_HOME',MonkeyDevice.DOWN_AND_UP)#点击Home键
```

Android 各种UI测试框架介绍

Instrumentation

编写语言：Python

运行环境：adb命令启动或者手机直接启动测试

测试对象：单个Activity测试，需与测试应用相同的签名

测试限制：主要用于白盒测试和UI测试

```
#Instrumentation代码风格示例
mActive.runOnUiThread(new Runnable(){
    public void run(){
        mSpinner.requestFocus();
    }
});
```

Android 各种UI测试框架介绍

Robotium

Robotium结合Android官方提供的测试框架达到对应用程序进行自动化的测试。另外，Robotium 4.0版本已经支持对WebView的操作。Robotium 对Activity，Dialog，Toast，Menu 都是支持的。

Android 各种UI测试框架介绍

优点

- Easy to use
- Tests are easy to read
- Doesn't require access to source code. Can test a APK
- Can identify elements easily (with caveat...more on this later)
- Can fall back on default Android framework
- Great support

缺点

- Not all views and objects are currently supported e.g. SlidingDrawer
- Slower compared to unit testing
- Single class containing all methods, Selenium 1 style. This is going to get messy

Android 各种UI测试框架介绍

UiAutomator

谷歌在Andtopd 4.1推出UiAutomator测试框架

作用

显而易见主要用于UI自动化测试

功能

模拟人对手机的操作；比如点击，长按，滑动，下拉等操作

优点

1. 编写快速
2. API简单易学
3. 无Active限值
4. 无需签名
5. 几乎可以模拟人的所有操作

缺点

1. 是只支持SDK 16（Android 4.1）及以上
2. 不支持Hybird App、WebApp。

Android 各种UI测试框架介绍

为啥要使用UiAutomator

1. 编写灵活，使用方便
2. 可快速学习
3. 限制少
4. 模拟目前90%以上的手工操作
5. 扩展性好

Android 各种UI测试框架介绍

UiAutomator

UIAutomator是android的自动化测试框架，可跨APP。与instrumentation框架不同，UIAutomator不需要测试对象源码，因此，为黑盒测试框架。同时，与Monkey不同，UIAutomator不以坐标为主线，而是通过控件属性过滤（比如搜索文本为“提交”的按钮），获取控件本身。

Android 各种UI测试框架介绍

UIAutomator2.0与1.0的区别

1. 2.0基于 Instrumentation， 可以获取应用Context，可以使用Android服务及接口。
2. 2.0基于 Junit4，测试用例无需继承于任何父类，方法名不限，使用Annotation进行，1.0需要继承UiAutomatorTestCase，测试方法需要以test开头。
3. 2.0采用Gradle进行构建，1.0使用Maven或Ant。
4. 2.0新增UiObject2、Until、By、BySelector等接口。
5. 2.0输出到Logcat，1.0可以使用System.out.print输出流回显至执行端。
6. 2.0输出为APK，1.0输出为JAR。

Android 各种UI测试框架介绍

UiAutomator与Appium关系

- Appium同时支持Native APP和HTML5
- Native APP利用UiAutomator
- HTML5利用Selenium

目录

- Andriod 各种UI测试框架介绍
- **UiAutomator UI自动化测试框架**
  - **环境准备**
  - 建立测试工程
  - Uiautomator API详解
  - 建立测试集
  - 实践

环境准备

安装JDK 1.8

Path:%JAVA_HOME%\jre\bin;%JAVA_HOME%\bin\;
CLASSPATH:%JAVA_HOME%\lib\dt.jar;%JAVA_HOME%\lib\tools.jar;

```
C:\Users\Jerry>javac
用法: javac <options> <source files>
其中, 可能的选项包括:
  -g                         生成所有调试信息
  -g:none                    不生成任何调试信息
  -g:{lines,vars,source}     只生成某些调试信息
  -nowarn                    不生成任何警告
  -verbose                   输出有关编译器正在执行的操作的消息
  -deprecation               输出使用已过时的 API 的源位置
  -classpath <路径>            指定查找用户类文件和注释处理程序的位置
  -cp <路径>                   指定查找用户类文件和注释处理程序的位置
  -sourcepath <路径>           指定查找输入源文件的位置
...
```

环境准备

安装SDK

```
C:\Users\Jerry> adb ssh
Android Debug Bridge version 1.0.36
Revision 0e9850346394-android
 -a                      - directs adb to listen on all interfaces for a connection
 -d                      - directs command to the only connected USB device
                           returns an error if more than one USB device is present.
 -e                      - directs command to the only running emulator.
                           returns an error if more than one emulator is running.
 -s <specific device>        - directs command to the device or emulator with the given
                           serial number or qualifier. Overrides ANDROID_SERIAL
...
```

环境准备

升级SDK License

切记：如果是Eclipce用的SDK不允许update

cd %ANDROID_HOME%

把目录名tools改为tool

cd tool/bin

./sdkmanager --update

在update工程中，在%ANDROID_HOME%目录下产生新的文件夹tools

update完毕把新产生的tools目录下所有文件拷贝到tool下

删除tools目录，把tool目录改名为tools

cd %ANDROID_HOME%/tools/bin

./sdkmanager ----licenses

环境准备

安装Android Stdio

选择SDK

"File"->"Other Settings"->"Default Project Structure"

SDK Location

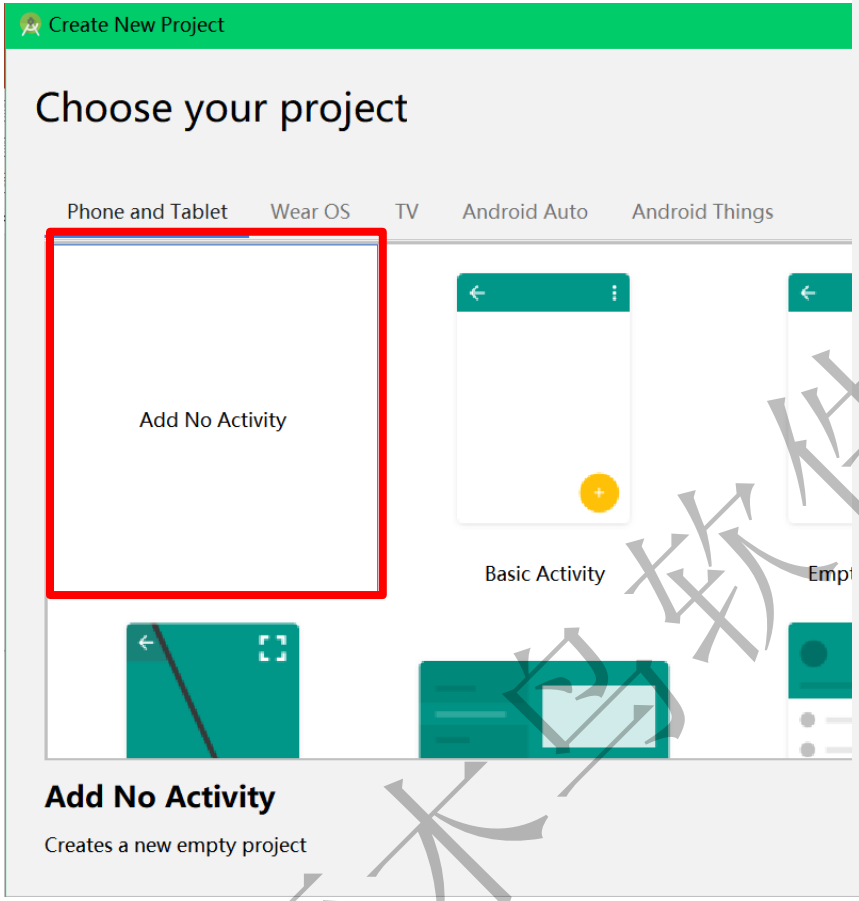
Android SDK location:
The directory where the Android SDK is located. This location will be used for new projects, and for existing projects that do not have a local.properties file with a sdk.dir property.

C:\ADT\sdk ...

目录

- Andriod 各种UI测试框架介绍
- **UiAutomator UI自动化测试框架**
  - 环境准备
  - **建立测试工程**
  - Uiautomator API详解
  - 建立测试集
  - 实践

建立测试工程



Name

MyAPPTest　Project名字

Package name

com.example.myapptest　包的名字，建议不要修改

Save location

C:\Users\Jerry\AndroidStudioProjects\MyAPPTest

Project的工作目录

Language

Java　使用语言，不要修改

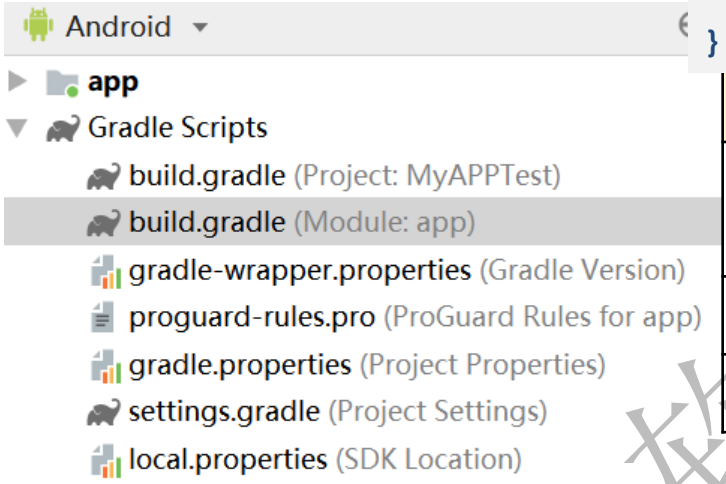
Minimum API level

API 18: Android 4.3 (Jelly Bean)　最小API版本，请大于API 18

Your app will run on approximately **95.9%** of devices.

建立测试工程

配置build.gradle(Moudual:a




```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:27.1.1'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1.0.2'
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'
}
```

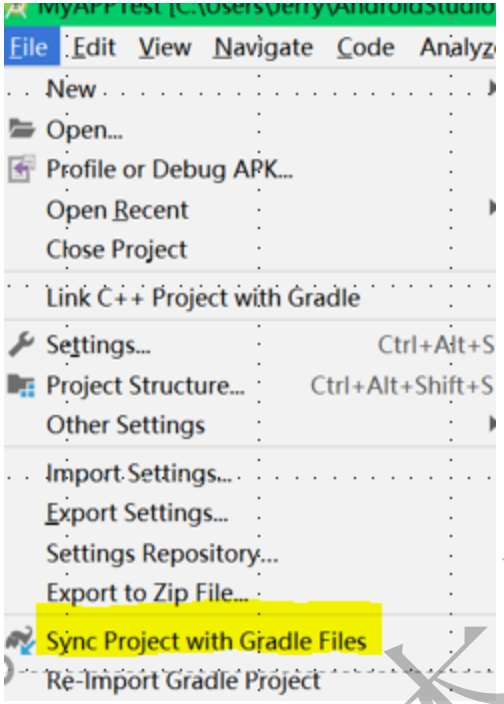
模块	命令
Android Test代码	androidTestImplementation '包名' testImplementation '包名'
Android 产品代码	testImplementation '包名'
普通Test代码	implementation '包名'

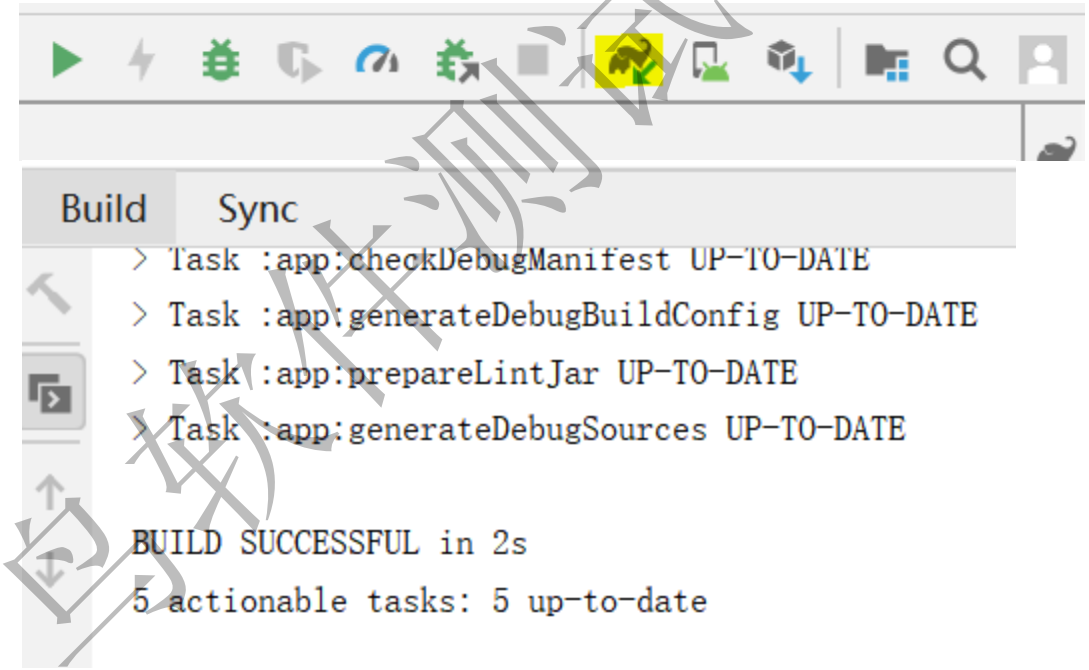
在AS 2.0中所有Implementation改为compile

```
dependencies {
    androidTestImplementation 'com.android.support.test.uiautomator:uiautomator-v18:2.1.1'
}
```

建立测试工程

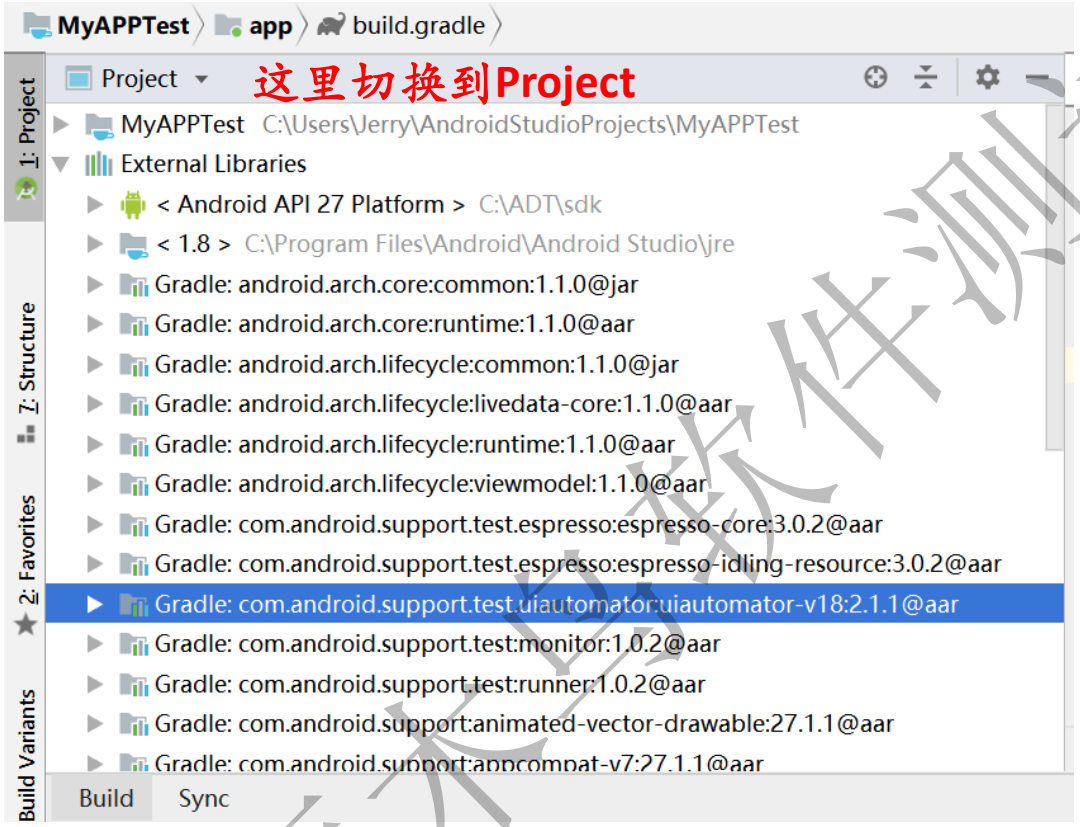
同步build.gradle(Moudual:app)





建立测试工程

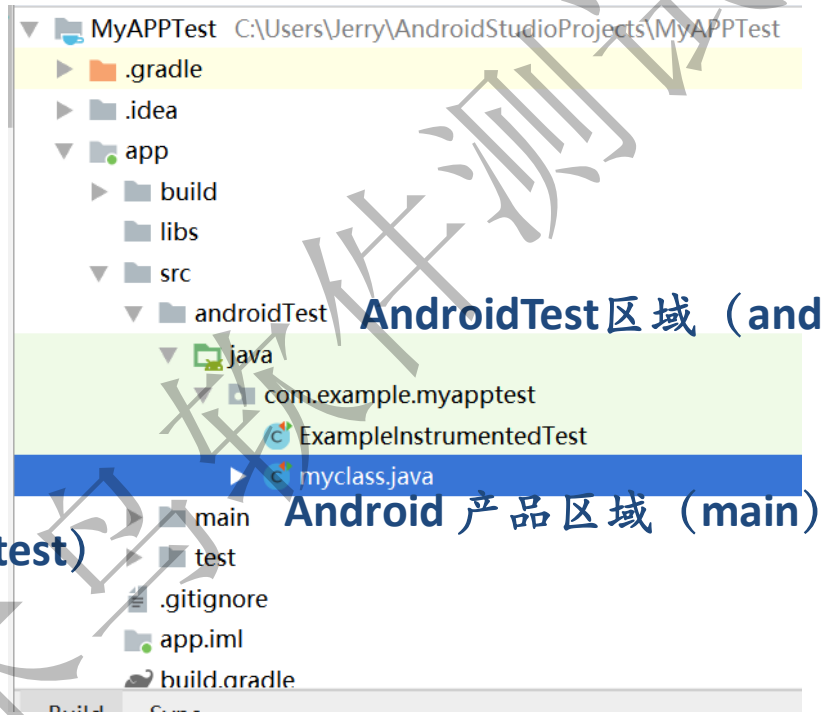
查看com.android.support.test.uiautomator:uiautomator-v18:2.1.1是否产生



建立测试工程

建立测试文件

由于我们是用androidTestImplementation同步的，所以文件要建立在AndroidTest区域内



AndroidTest区域（androidTest）

Android 产品区域（main）

普通Test区域（test）

建立测试工程

建立测试文件

```
import android.support.test.uiautomator.UiScrollable;
import android.test.InstrumentationTestCase;
import android.support.test.uiautomator.UiObject;
import android.support.test.uiautomator.UiObjectNotFoundException;
import android.support.test.uiautomator.UiDevice;
import android.support.test.uiautomator.UiSelector;
@RunWith(AndroidJUnit4.class)
public class myclass extends InstrumentationTestCase {
  public void testHome(){
     instrumentation = InstrumentationRegistry.getInstrumentation();
     UiDevice.getInstance(getInstrumentation()).pressHome();}

  public void testMenu() {
     UiDevice.getInstance(getInstrumentation()).pressMenu();}
}
```

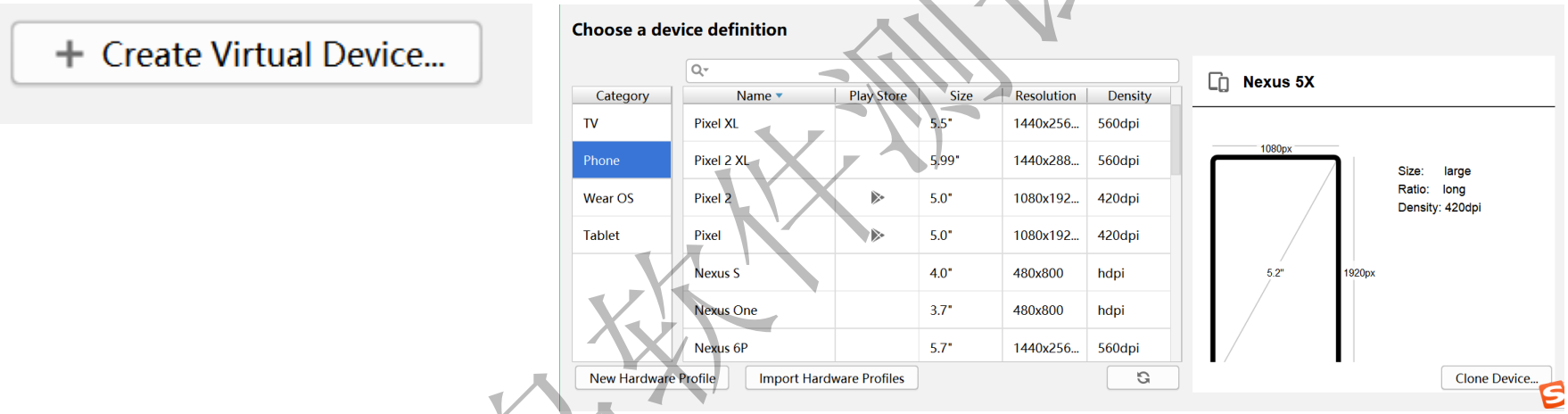
建立测试工程

建立测试文件

Eclipse中的方法	Android Studio中的方法
import android.support.test.uiautomator	import com.android.uiautomator.core
extends UiAutomatorTestCase	extends InstrumentationTestCase
UiDevice.getInstance()	UiDevice.getInstance(getInstrumentation())

建立测试工程

建立虚拟设备



选择Phone,尽量选择分辨率不要超过1000的，否则启动很慢或者启动不起来

建立测试工程

建立虚拟设备

Select a system image

Recommended x86 Images Other Images

Release Name	API Level	ABI	Target
Jelly Bean Download	18	armeabi-v7a	Android 4.3 (Google APIs)
Jelly Bean Download	18	armeabi-v7a	Android 4.3
Jelly Bean Download	17	armeabi	Android 4.2 (Google APIs)
Jelly Bean Download	17	armeabi-v7a	Android 4.2 (Google APIs)
Jelly Bean Download	17	mips	Android 4.2
Jelly Bean Download	17	armeabi-v7a	Android 4.2
Jelly Bean Download	16	armeabi	Android 4.1 (Google APIs)
Jelly Bean Download	16	armeabi-v7a	Android 4.1 (Google APIs)
Jelly Bean Download	16	mips	Android 4.1

系统会有提示，建议选择x86 Image,但是由于与计算机显卡原因不一定兼容，我不建议选择。

建立测试工程

建立虚拟设备

Verify Configuration

AVD Name: Nexus 5X API 19

Nexus 5X — 5.2 1080x1920 420dpi — Change...

KitKat — Android 4.4 armeabi-v7a — Change...

Startup orientation: Portrait / Landscape

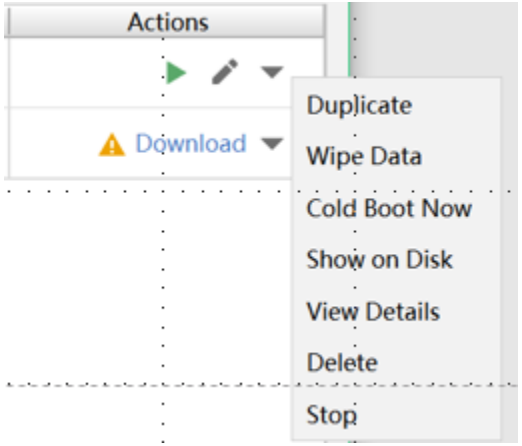
Emulated Performance — Graphics: Software - GLES 1.1

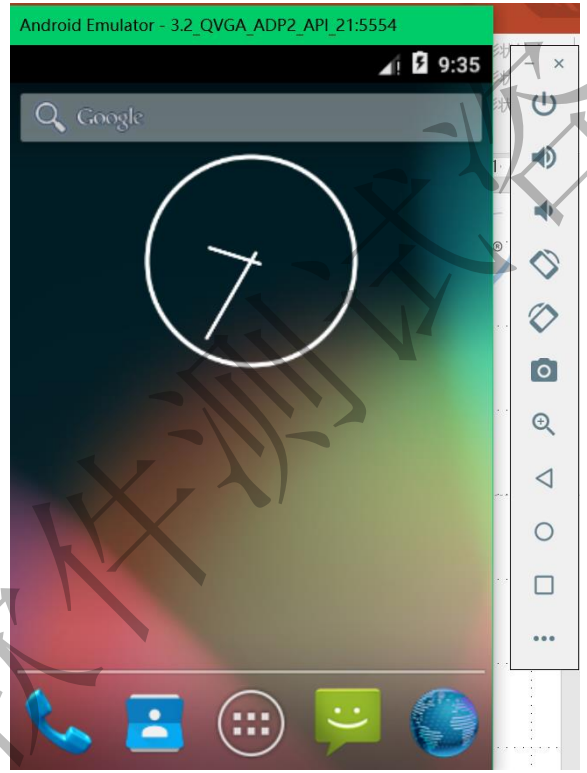
Device Frame: Enable Device Frame

Show Advanced Settings

建立测试工程

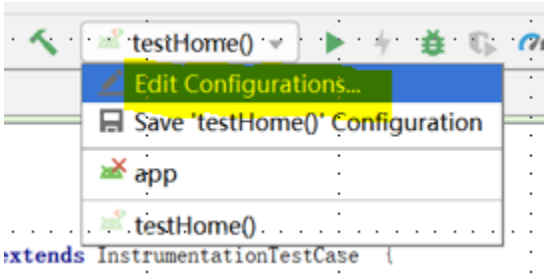
建立虚拟设备

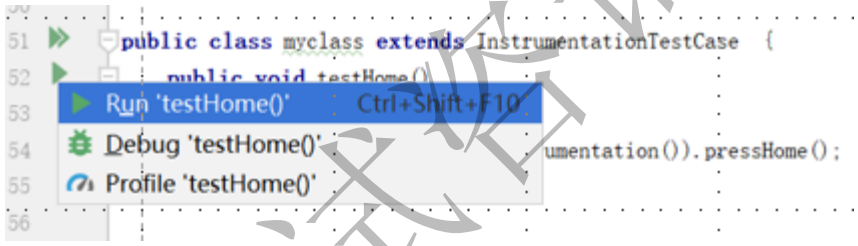


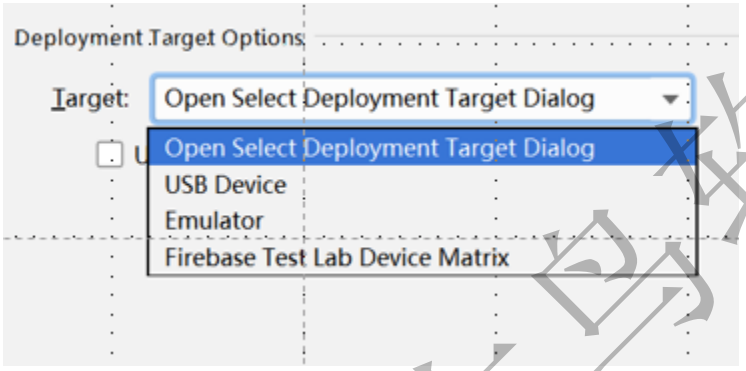


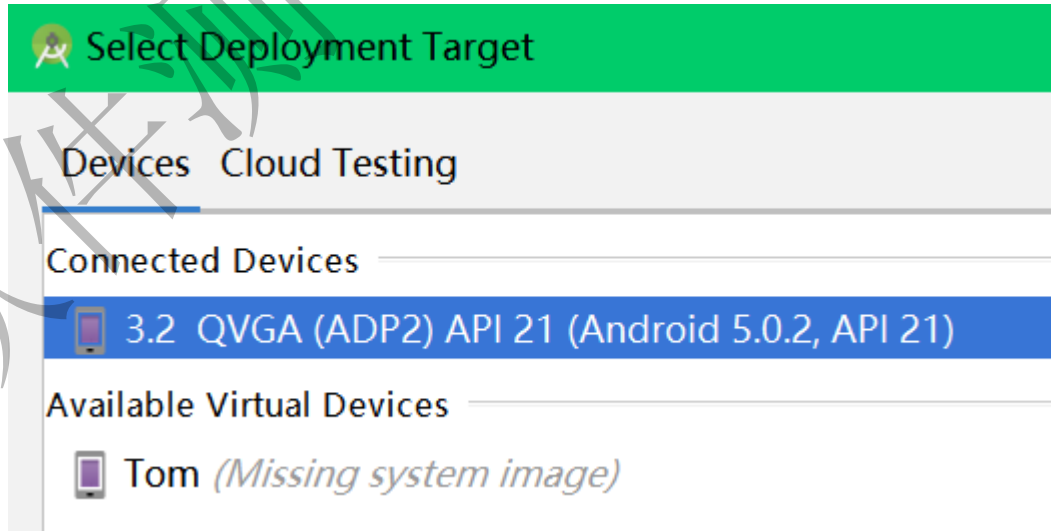
建立测试工程

建立虚拟设备







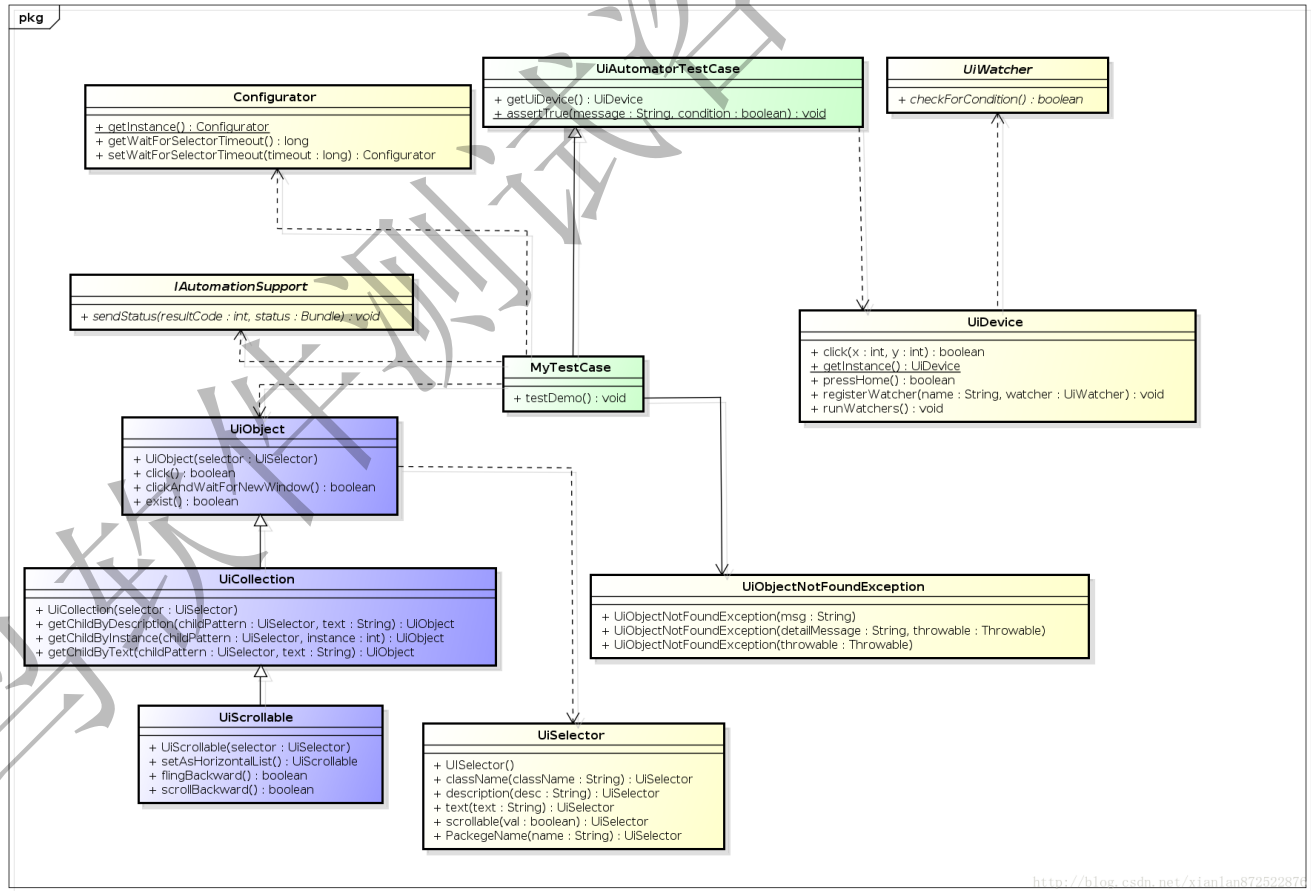


目录

- Andriod 各种UI测试框架介绍
- **UiAutomator UI自动化测试框架**
  - 环境准备
  - 建立测试工程
  - **Uiautomator API详解**
  - 实践

Uiautomator API详解

Uiautomator 1.0类图



Uiautomator API详解

Uiautomator 2.0所有类

类名	描述
InstrumentationRegistry类	持有instrumentation运行的进程和参数
UiDevice	模拟用户在设备上的操作
BySelector类	指定搜索条件进行匹配UI元素,通UiDevice.findObject(BySelector)方式进行使用
By类	以简洁的方式创建BySelectors对象
UiSelector类	通过控件的各种属性与节点关系定位组件
UiObject	代表一个组件对象，它提供一系列方法和属性来模拟在手机上的实际操作
UiObject2类	是UiObject的升级类
UiCollection类	UiObject的子类，用来表示一个父控件，该控件下包含了子元素的集合
UiScrollable类	UiCollection的子类，专门处理滚动时间，提供各种滚动方法
UiWatcher类	用于处理脚本执行过程中遇到非预想的步骤
Configurator类	用于设置脚本动作的默认延时

Uiautomator API详解

- **InstrumentationRegistry类**
- UiDevice 类
- By Selector和By类
- UiSelector类
- UiObject类
- UiObject2类
- UiCollection类
- UiScrollable类
- UiWatcher类
- Configurator类
- 断言

InstrumentationRegistry类

一个暴露的注册实例，持有instrumentation运行的进程和参数，还提供了一种简便的方法

返回值	方法名	描述
static Bundle	getArguments()	返回一个instrumentation参数副本
static Context	getContext()	返回instrumentation对应包的Context InstrumentationRegistry.getContext() == instrumentation.getContext()
static Instrumentation	getInstrumentation()	返回当前运行的instrumentation
static Context	getTargetContext()	返回一个目标应用程序的Context
static void	registerInstance(Instrumentation instrumentation, Bundle arguments)	记录或暴露当前instrumentation运行和instrumentation参数包的副本，存储在注册中心

InstrumentationRegistry类

```
@Test
public void InstrumentationRegistryTest() {
   Instrumentation instrumentation = InstrumentationRegistry.getInstrumentation();
   Context context1 = InstrumentationRegistry.getContext();
   Context context2 = InstrumentationRegistry.getTargetContext();
   Context context3= instrumentation.getContext();
   if(context1 == context2) {
      Log.i("Chris",  "InstrumentationRegistry getContext == getTargetContext");
   }else {
      Log.i("Chris",  "InstrumentationRegistry getContext != getTargetContext");
   }
   if(context1 == context3) {
     Log.i("Chris",  "InstrumentationRegistry getContext == Instrumentation getContext");
   }else {
      Log.i("Chris",  "InstrumentationRegistry getContext != Instrumentation getContext");
   }
   Intent intent = context2.getPackageManager().getLaunchIntentForPackage("xxx.xxx.xxx");
  intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK);
   context2.startActivity(intent);}
```

Uiautomator API详解

- InstrumentationRegistry 类
- **UiDevice 类**
- By Selector和By类
- UiSelector 类
- UiObject 类
- UiObject2 类
- UiCollection 类
- UiScrollable 类
- UiWatcher 类
- Configurator 类
- 断言

UiDevice类

对象定义

UiDevice用与访问关设备状态的信息，也可以使用这个类来模拟用户在设备上的操作。

在UiAutomation 2.0如下定义

```
instrumentation = InstrumentationRegistry.getInstrumentation();
UiDevice.getInstance(instrumentation)
```

在UiAutomation 1.0如下定义

```
UiDevice.getInstance()
```

曾经有的定义方式

```
getUiDevice()
```

由于会出现空指针异常，已经被淘汰

UiDevice 类

对象定义

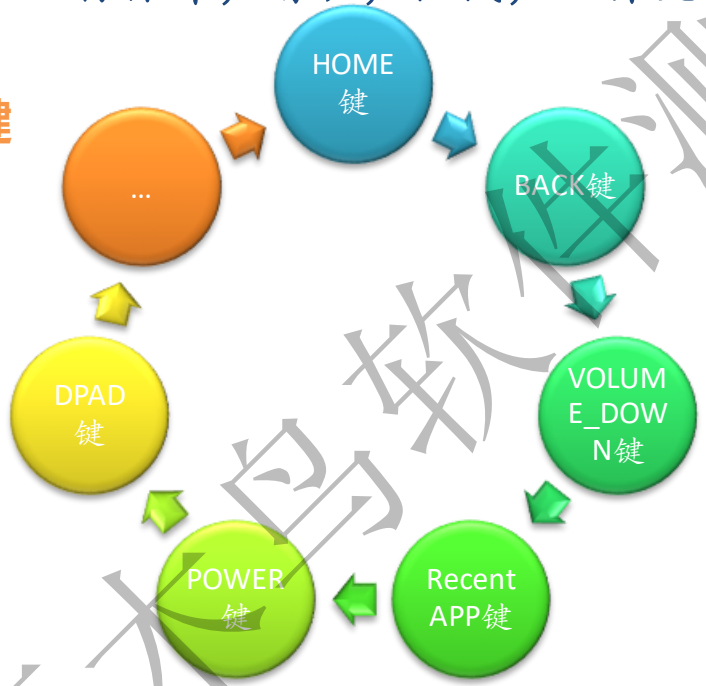
```
import android.support.test.uiautomator.UiDevice;

public class myclass extends InstrumentationTestCase  {
        public void testHome()
        {
                instrumentation = InstrumentationRegistry.getInstrumentation();
                UiDevice.getInstance(instrumentation).pressHome();
        }

        public void testMenu()
        {
                 instrumentation = InstrumentationRegistry.getInstrumentation();
                UiDevice.getInstance(instrumentation).pressMenu();
        }
```

UiDevice 类

UiDevice 操作功能

获取设备信息：屏幕分辨率，旋转状态，亮灭屏状态等
操作：按键，坐标操作，滑动，拖拽，灭屏唤醒屏幕，截图等
监听功能

手机常见按键



UiDevice 类

按键API

返回值	方法名	描述
boolean	pressBace()	模拟短按返回back键
boolean	pressDPadCenter()	模拟轨迹球中点按键
boolean	pressDPadDown()	模拟轨迹球向下按键
boolean	pressDPadLeft()	模拟轨迹球向左按键
boolean	pressDPadRight()	模拟轨迹球向右按键
boolean	pressDPadUp()	模拟轨迹球向上按键
boolean	pressDelete()	模拟短按删除delete按键
boolean	pressEnter()	模拟短按回车键
boolean	pressHome()	模拟短按home键
boolean	pressKeyCode(int keyCode, int metaState)	模拟短按键盘代码keyCode
boolean	pressKeyCode(int keyCode)	模拟短按键盘代码keyCode
boolean	pressMenu()	模拟短按menu键
boolean	pressRecentApps()	模拟短按最近使用程序
boolean	pressSearch()	模拟短按搜索键

UiDevice 类

案例

```
public class myclass extends InstrumentationTestCase {
        public void testHome(){
                instrumentation = InstrumentationRegistry.getInstrumentation();
                UiDevice.getInstance(instrumentation).pressHome();
        }
        public void testMenu(){
                instrumentation = InstrumentationRegistry.getInstrumentation();
                UiDevice.getInstance(instrumentation).pressMenu();
        }
        public void testRecent() throws InterruptedException, RemoteException {
                instrumentation = InstrumentationRegistry.getInstrumentation();
                UiDevice.getInstance(instrumentation).pressRecentApps();
                Thread.sleep(1000);;
        }
}
```

UiDevice 类

KEYCODE 键盘映射码：

1. KEYCODE 按键事件
2. META KEY 辅助功能键：ALT、SHIFT、CAPS LOCK

列	激活状态	Meta State
base	META_KEY未被激活	0
caps	SHIFT或CAPS LOCK被激活	1
fn	ALT被激活	2
caps_fn	ALT、SHIFT或CAPS_LOCK同时被激活	3

UiDevice 类

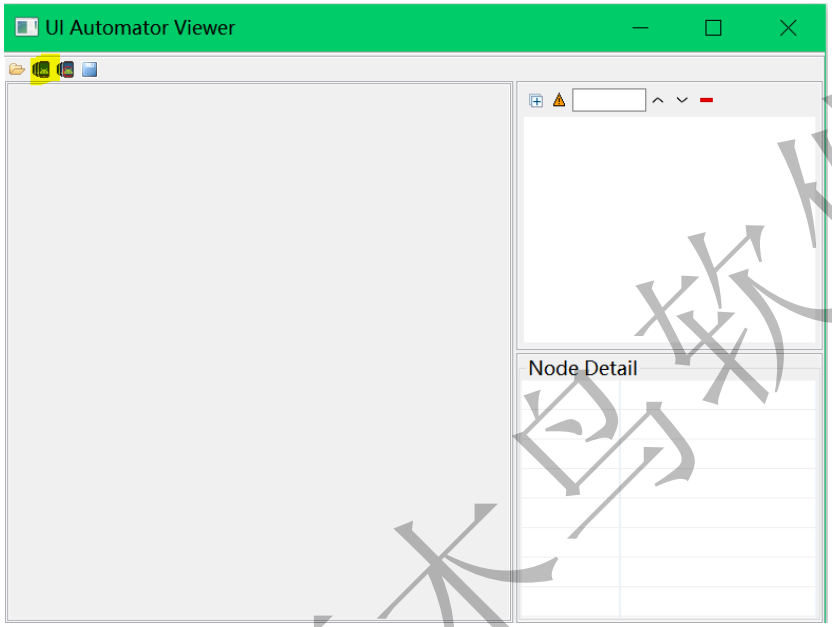
案例

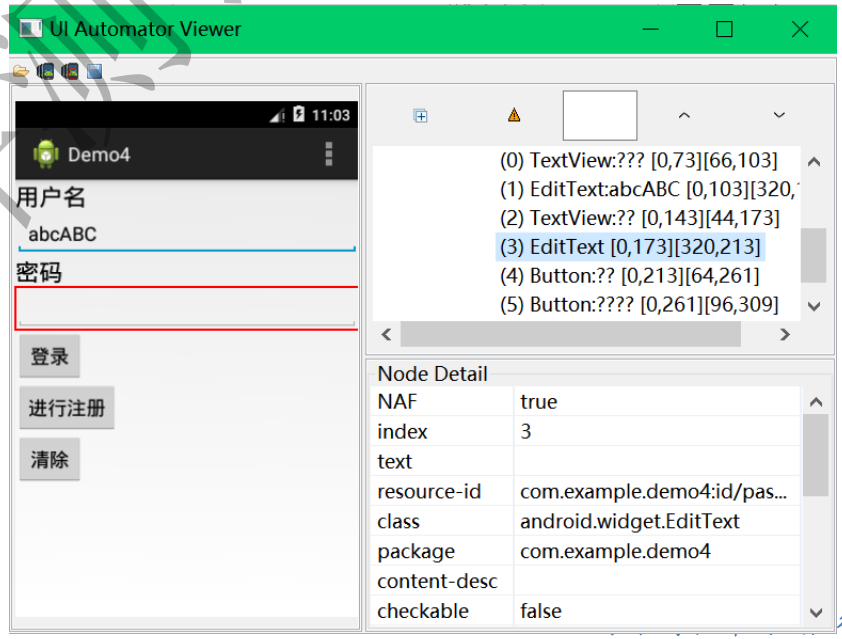
```
import android.view.KeyEvent;
...
public void testAndroidKey() throws UiObjectNotFoundException {
      instrumentation = InstrumentationRegistry.getInstrumentation();
      UiDevice.getInstance(instrumentation).pressHome();
      Util util=new Util("Demo4");
      util.comming();
      UiDevice.getInstance(getInstrumentation()).pressKeyCode(KeyEvent.KEYCODE_A); //小写a
      UiDevice.getInstance(getInstrumentation()).pressKeyCode(KeyEvent.KEYCODE_B); //小写b
      UiDevice.getInstance(getInstrumentation()).pressKeyCode(KeyEvent.KEYCODE_C); //小写c
      UiDevice.getInstance(getInstrumentation()).pressKeyCode(KeyEvent.KEYCODE_A,1); //大写A
      UiDevice.getInstance(getInstrumentation()).pressKeyCode(KeyEvent.KEYCODE_B,1); //大写B
      UiDevice.getInstance(getInstrumentation()).pressKeyCode(KeyEvent.KEYCODE_C,1); //大写C
   }
```

UiDevice 类

UiAutomation View

%sdk_home%\tools\uiautomatorviewer.bat





UiDevice 类

坐标相关的API

返回值	方法名	描述
boolean	click(int x, int y)	使用坐标点击屏幕
int	getDisplayHeight()	获取屏幕高度
Point	getDisplaySizeDP()	获取显示尺寸返回显示大小（设备独立像素） 屏幕旋转返回的显示大小调整
int	getDisplayWidth()	获取屏幕宽度

UiDevice 类

```
...System.out: The display width is: 320
...System.out: The display height is: 480
...System.out: Point(320, 480)
```

案例

```
import android.graphics.Point;
...
public void testClick() throws UiObjectNotFoundException {
    instrumentation = InstrumentationRegistry.getInstrumentation();
     UiDevice.getInstance(instrumentation)pressHome();
    Util util=new Util("Browser");
    util.enterAPP();
    int h=UiDevice.getInstance(getInstrumentation()).getDisplayHeight();
    int w=UiDevice.getInstance(getInstrumentation()).getDisplayWidth();
    Point p=UiDevice.getInstance(getInstrumentation()).getDisplaySizeDp();
    System.out.println("The display width is: " + w);
    System.out.println("The display height is: "+h);
    System.out.println(p);
    UiDevice.getInstance(getInstrumentation()).click(100,300);
    UiDevice.getInstance(getInstrumentation()).pressHome(); }
```

UiDevice 类

案例

```
UiObject object = new UiObject(new UiSelector.resoutceId("com.andr....控件Id"));//根据ID获取控件
Rect r = object.getBounds();//获取控件对应的矩形区域相关属性
r.left; //矩形左上角顶点X坐标
r.top; //矩形左上角顶点Y坐标
r.right; //矩形右下角顶点X坐标
r.bottom; //矩形右下角顶点Y坐标
r.centerX(); //矩形的中心点X坐标
r.centerY(); //矩形的中心点Y坐标
```

UiDevice 类

拖拽与滑动

- 拖拽：将对象从一个坐标移动到另一个坐标
- 移动：从一个坐标点移动到另一个坐标点
- 步长：从一点滑动到另一点使用的时间

拖拽与滑动的相关API

返回值	方法名	描述
boolean	drag(int startX, int startY, int endX, int endY, int steps)	拖动对象从[startX, startY]到[endX, endY], 步长为steps
boolean	swipe(Point[] segments, int segmentSteps)	在点阵列中滑动，5ms一步
boolean	swipe(int startX, int startY, int endX, int endY, int steps)	通过坐标滑动屏幕

UiDevice 类

案例

```
public void testDragAndSwipe() throws UiObjectNotFoundException {
    instrumentation = InstrumentationRegistry.getInstrumentation();
    UiDevice.getInstance(instrumentation).pressHome();
    Util util=new Util("Browser");
    util.enterAPP();
    int startX, startY, endX, endY, steps;
    startX=128;
    startY=200;
    endX=200;
    endY=300;
    steps=100;
    UiDevice.getInstance(getInstrumentation()).drag(startX, startY, endX, endY, steps);

    Point p1=new Point();
    Point p2=new Point();
    Point p3=new Point();
    Point p4=new Point();
```

UiDevice 类

案例

```
    p1.x=100;p1.y=100;
    p2.x=100;p2.y=300;
    p3.x=300;p3.y=300;
    p4.x=300;p4.y=100;
    Point[] ps={p1,p2,p3,p4,p1};
    UiDevice.getInstance(getInstrumentation()).swipe(ps, 50);

    startX=278;
    startY=374;
    endX=69;
    endY=373;
    steps=100;
    UiDevice.getInstance(getInstrumentation()).swipe(startX, startY, endX, endY, steps);
    UiDevice.getInstance(getInstrumentation()).pressHome();
}
```

UiDevice 类

屏幕旋转

旋转方向：0度，90度（向左转），180度，270度（向右转）
重力感应器：重力感应器是旋转所依靠的
固定位置：指将屏幕方向固定在0度，90度或者180度等
物理旋转：物理旋转与重力感应器关联在一块，关闭物理旋转就是关闭了重力感应器，反之亦然）

旋转屏幕相关API

返回值	方法名	描述
void	setOrientationLeft()	通过禁用传感器，然后模拟设备向左转，并且固定位置
void	setOrientationNatural()	通过禁用传感器，然后模拟设备转到其自然默认的方向，并且固定位置
void	setOrientationRight()	通过禁用传感器，然后模拟设备向右转，并且固定位置
void	unfreezeRotation()	重新启动传感器和允许物理旋转
boolean	isNaturalOrientation()	检测设备是否处于默认旋转状态
int	getDisplayRotation()	返回当前的显示旋转，0度，90度，180度，270度值分别为：0（Surface.ROTATION_0）、1（Surface.ROTATION_90）、2（Surface.ROTATION_180）、3（Surface.ROTATION_270）
void	freezeRotation()	禁用传感器和冻结装置物理旋转在其当前旋转状态

UiDevice 类

```
public void testDemo5() throws UiObjectNotFoundException, RemoteException, InterruptedException {
    instrumentation = InstrumentationRegistry.getInstrumentation();
    UiDevice.getInstance(instrumentation).pressHome();
    Util util=new Util("Demo5");
     util.comming();
    UiObject myroot=new UiObject(new UiSelector().className("android.view.View").instance(0));
    myroot.click();
     myroot.click();
    int r = UiDevice.getInstance(getInstrumentation()).getDisplayRotation();
    if (r == 0) {
       UiDevice.getInstance(getInstrumentation()).setOrientationLeft();}
    if (r == 1) {
       UiDevice.getInstance(getInstrumentation()).setOrientationNatural();
       UiDevice.getInstance(getInstrumentation()).setOrientationLeft();}
    if (r == 2) {
       UiDevice.getInstance(getInstrumentation()).setOrientationNatural();
       UiDevice.getInstance(getInstrumentation()).setOrientationLeft();}
    if (r == 3) {
       UiDevice.getInstance(getInstrumentation()).setOrientationNatural();}
     UiDevice.getInstance(getInstrumentation()).pressHome();}
```

UiDevice 类

灭屏与唤醒

返回值	方法名	描述
void	wakeUp()	模拟按电源键，如果屏幕是唤醒的没有任何作用
void	sleep()	模拟按电源键，如果屏幕已经是关闭的则没有任何作用
boolean	isScreenOn()	检查屏幕是否亮屏

等待空闲

返回值	方法名	描述
void	waitForIdle(long timeout)	自定义超时等待当前应用处于空闲状态
void	waitForIdle()	等待当前应用处于空闲状态，默认等待10s；即10s后还不处于空闲状态则报错，程序在该句代码处中断；10s内程序处于空闲状态，则该句代码执行完毕。
boolean	waitForWindowUpdate(String packageName, long timeout)	等待窗口内容更新事件的发生

UiDevice 类

截图

1. 图片缩放比例：如缩小1/2，即将100*100px的图片长宽都缩小为原来的1/2，50*50px
2. 图片质量：一般是指图片的大小，质量越高图片越大
3. File 类：文件或者文件夹
4. 图片格式：截图的格式都是PNG
5. 空闲状态：窗口没有更新或界面无动作
6. 窗口更新事件

返回值	方法名	描述
boolean	takeScreenshot(File storePath)	把当前窗口截图并将其存储为png默认1.0f的规模(原尺寸)和90%质量，参数为file类的文件路径。
boolean	takeScreenshot(File storePath, float scale, int quality)	把当前窗口截图为png格式图片，可以自定义缩放比例与图片质量。

- storePath：存储路径，必须为png格式
- scale：缩放比例，1.0为原图
- quality：图片压缩质量，范围为0-100

UiDevice 类

获取包名&开启通知栏&快速设置&获取布局文件

1. 包名：应用的唯一标识。
2. 通知栏：从手机顶部下滑，出现的下拉界面即通知栏。
3. 快速设置：即通知栏中的快速设置控件，快速设置界面可设置网络、屏幕亮度、飞行模式等。

返回值	方法名	描述
void	getCurrentPackageName()	获取当前界面的包名，即目前处于手机前台的应用的包名
void	dumpWindowHierarchy(String fileName)	取当前界面的布局文件，fileName给布局文件命名如"layout.xml"，保存在/data/local/tmp/目录下
boolean	openNotification()	打开通知栏
boolean	openQuickSettings()	打开快速设置

UiDevice 类

UiAutomator2在UiDevice新增的API

返回值	方法名	描述
void	dumpWindowHierarchy(OutPutStream out)	获取当前页面层级到输出流
String	executeShellCommand(String cmd)	执行一个shell命令。备注：此方法只支持api21以上，手机需要5.0系统以上
UiObject2	findObject(BySelector selector)	返回第一个匹配条件的对象
UiObject	findObject(UiSelector selector)	返回一个匹配条件的代表视图的UiObject对象
List<UiObject2>	findObjects(BySelector selector)	返回所有匹配条件的对象
<R> R	wait(SearchCondition<R> condition, long timeout)	等待的条件得到满足

```
UiDevice.getInstance(getInstrumentation()).executeShellCommand("am force-stop packageName");
```

Uiautomator API详解

- InstrumentationRegistry 类
- UiDevice 类
- **By Selector和By类**
- UiSelector 类
- UiObject 类
- UiObject2 类
- UiCollection 类
- UiScrollable 类
- UiWatcher 类
- Configurator 类
- 断言

By Selector和By类

BySelector和By是UiAutomator2.0的类。BySelector类为指定搜索条件进行匹配UI元素,通UiDevice.findObject(BySelector)方式进行使用。By类是一个实用程序类，可以以简洁的方式创建BySelectors对象。主要功能是使用缩短语法，提供静态工厂方法来构造BySelectors对象。例如：你将使用findObject(By.text("foo")),而不是findObject(newSelector().text("foo"))的方式来查找文本值为"foo"的UI元素。

By Selector和By类

返回值	方法名	描述
BySelector	clazz(String calssName), clazz(String packageName, String className), clazz(Class clazz), clazz(Pattern className)	通过class来匹配UI
BySelector	desc(String contentDescription) descContains(String substring) descStartsWith(String substring) descEndsWith(String substring) desc(Pattern contentDescription)	通过contentDescription来匹配UI
BySelector	ext(String contentDescription) textContains(String substring) textStartsWith(String substring) textEndsWith(String substring) text(Pattern contentDescription)	通过text来匹配UI

By Selector和By类

返回值	方法名	描述
BySelector	res(String resourceName) res(String resourcePackage, String resourceId) res(Pattern resourceName)	通过id来匹配UI
BySelector	checkable(boolean isCheckable)	
BySelector	clickable(boolean isClickable)	
BySelector	enabled(boolean isEnabled)	
BySelector	focusable(boolean isFocusable)	
BySelector	focused(boolean isFocused)	
BySelector	longClickable(boolean isLongClickable)	
BySelector	scrollable(boolean isScrollable)	

Uiautomator API详解

- InstrumentationRegistry类
- UiDevice 类
- By Selector和By类
- **UiSelector类**
- UiObject类
- UiObject2类
- UiCollection类
- UiScrollable类
- UiWatcher类
- Configurator类
- 断言

UiSelector类

new UiSelector().

UiSelector功能

UiSelector可通过控件的各种属性与节点关系定位组件。

Android常用组件

- TextView 文本框
- EditView 编辑框
- Button 按钮
- RadioButton 单选按钮
- CheckBox 复选框
- ToggleButton 状态开关按钮
- Switch 开关
- SeekBar 拖动条
- AnalogClock DigitalClock 时钟
- Chronometer 计时器
- ListView 列表视图
- GridView 网格视图
- ProgressBar 进度条
- RatingBra 星际评分条
- Toast 提示信息框
- ScrollView 滚动视图

UiSelector类

Android组件的属性

属性值	值类型	说明
index	int	索引：同级组件的下标；从0开始计
instance	int	界面中同一类View的所有实例的下标；从0开始计
class	String	组件的类名，如android.widget.TextView
package	String	包名
Content-desc	String	描述
checkable	boolean	是否可选，一般只对单选或复选框有用。
checked	boolean	单选或复选框是否被选中
clickable	boolean	是否可点击
enabled	boolean	是否可操作，如按钮置灰不可操作状态
focusable	boolean	是否可获取焦点
focused	boolean	是否获取到焦点
Scrollable	boolean	是否可滚动，一般是list
Long-clickable	boolean	是否可长按
password	boolean	是否密码
selected	boolean	是否具有背景选择属性，如按钮点击后背景色变化
bounds	Rect	坐标，如[10,10][50,50]矩形左上和右下坐标点

UiSelector类

查找组件的匹配方式

1. 默认： 完全匹配
2. Contains:包含匹配
3. StartsWith:起始匹配
4. Matches:正则匹配（完全匹配、包含匹配、起始匹配）

XML文档节点关系

1. 父 Patent:父控件
2. 子 Children:子控件
3. 同胞 Sibling： 同级控件
4. 先辈 Ancestor:父控件的上一级、上上级...
5. 后代 Descendant:子控件的下一级、下下级...

UiSelector类

属性-文本

返回值	API	说明
UiSelector	text(String text)	文本
UiSelector	textContains(String text)	文本包含
UiSelector	textMatches(String regex)	文本正则
UiSelector	textStartsWith(String text)	文本起始匹配

```
UiSelector uiSelector = new UiSelector().text("天气");//查找文本为“天气”的所有组件
```

UiSelector类

属性-描述

返回值	API	说明
UiSelector	description(String desc)	描述
UiSelector	descriptionContains(String desc)	描述包含
UiSelector	descriptionMatches(String regex)	描述正则
UiSelector	descriptionStartsWith(String desc)	描述开始字符匹配

```
UiSelector uiSelector = new UiSelector(). descriptionContains("天气");//查找属性包含“天气”的所有组件
```

UiSelector类

属性-类名

返回值	API	说明
UiSelector	className(String className)	类名
UiSelector	classNameMatches(String regex)	正则类名

输入参数className的快速书写方式：

1. class.getName方式
2. 完整类名方式：android.widget.LinearLayout
3. 正则方式
4. 常量方式

```
UiSelector x=new UiSelector().className("android.widget.CheckBox");
```

```
UiSelector wx=new UiSelector().className("android.widget.TextView").text"微信");
```

UiSelector类

属性-包名

返回值	API	说明
UiSelector	packageName(String name)	包名
UiSelector	packageNameMatches(String regex)	包名正则

属性-索引与实例

返回值	API	说明
UiSelector	index(int index)	索引
UiSelector	instance(int instance)	实例

```
UiSelector uiSelector = new UiSelector().className("android.widget.ImagView").index(1);//查找ImageView在同级中下标为1的组件
```

```
UiSelector uiSelector = new UiSelector().text("天气").instance(2);//查找界面中文本为“天气”的第3个组件
```

UiSelector类

特殊属性

返回值	API	说明
UiSelector	checked(boolean val)	选择属性
UiSelector	clickable(boolean val)	可点击属性
UiSelector	enabled(boolean val)	可操作属性
UiSelector	focusable(boolean val)	焦点属性
UiSelector	focused(boolean val)	当前焦点属性
UiSelector	longClickable(boolean val)	长按属性
UiSelector	scrollable(boolean val)	滚动属性
UiSelector	selected(boolean val)	背景选择属性

```
UiSelector x=new UiSelector().fousable(true).className("android.widget.CheckBox");
```

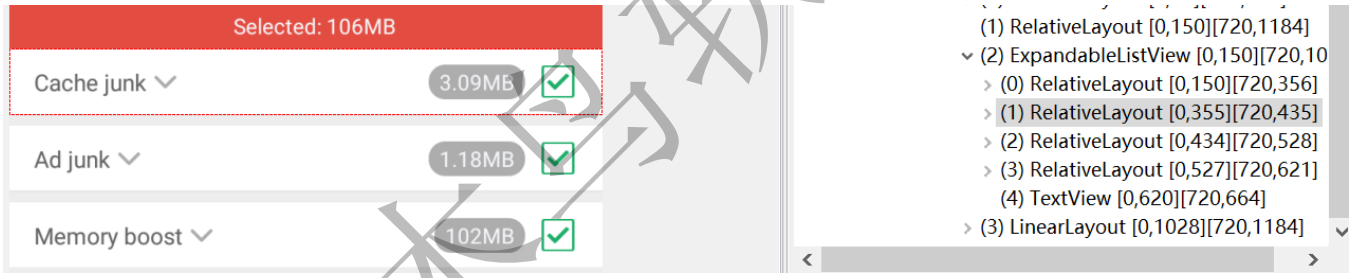
UiSelector类

节点

```
UiObject parentView = new UiObject(new UiSelector().className("android.view.View"));
save = parentView.getChild(new UiSelector().text("Save"));
```

返回值	API	说明
UiSelector	childSelector(UiSelector selector)	从当前组件往下查找其子孙中符合条件的所有组件
UiSelector	fromParent(UiSelector selector)	从当前组件往上查找其长辈中符合条件的所有组件

```
UiObject uio=new UiObject(new UiSelector().text("Cache junk").fromParent(new
UiSelector().className("android.widget.CheckBox")));
```



UiSelector类

属性-资源ID

返回值	API	说明
UiSelector	resourceId(String id)	资源ID
UiSelector	resourceIdMatches(String regex)	资源ID正则

android4.3及以上系统才可通过资源ID的方式定位控件

Uiautomator API详解

- InstrumentationRegistry类
- UiDevice 类
- By Selector和By类
- UiSelector类
- **UiObject类**
- UiObject2类
- UiCollection类
- UiScrollable类
- UiWatcher类
- Configurator类
- 断言

UiObject类

UiObject用来代表一个组件对象，它提供一系列方法和属性来模拟在手机上的实际操作。如：文本的输入和清除、点击、长按、拖动、滑动，也可获取组件的属性、判断组件对象是否存在等。

点击与长按

```
new UiObject(UiSelector Object).actionFunction;
```

返回值	API	说明
boolean	click()	点击对象
boolean	clickAndWaitForNewWindow(long timeout)	点击对象，等待新窗口出现，参数为等待超时时长
boolean	clickAndWaitForNewWindow()	点击对象，等待新窗口出现
boolean	clickBottomRight()	点击对象的右下角
boolean	clickTopLeft()	点击对象的左上角
boolean	longClick()	长按对象，对对象执行长按操作
boolean	longClickBottomRight()	长按对象的右下角
boolean	longClickTopLeft()	长按对象的左上角

```
//点击
new UiObject(new UiSelector().text("QQ") ).click();
```

```
//长按
new UiObject(new UiSelector().text("QQ")).longClick();
```

UiObject类

拖拽与滑动

返回值	API	说明
boolean	dragTo(UiObject destObj, int steps)	拖拽对象到另一个对象位置上，步长可设置拖动的速度
boolean	dragTo(int destX, int destY, int steps)	拖拽对象到屏幕某个坐标位置上，步长可设置拖动速度
boolean	swipeDown(int steps)	拖动对象往下滑动
boolean	swipeLeft(int steps)	拖动对象往左滑动
boolean	swipeRight(int steps	拖动对象往右滑动
boolean	swipeUp(int steps)	拖动对象往上滑动

```
//把QQ移动到【560,600】坐标处，40为step
new UiObject(new UiSelector().text("QQ")).dragTo(560,600,40);
```

UiObject类

案例

```
//交换QQ与计算器的位置
new UiObject(new UiSelector().text("QQ")).drapTo(new UiObject(new UiSelector().text("计算器")),40);
```

```
//向左划屏，按着第3个（0开始，所以为2）APP左划
UiObject obj = new UiObject(new
UiSelector().className("android.view.View").instance(2)).SwipeLeft(10);
```

UiObject类

输入文本与清除文本

返回值	API	说明
boolean	setText(String text)	在对象中输入文本（实现方式：先清除文本再输入）
void	clearTextField()	清除编辑框中的文本（实现方式：长按再清除）

```
//输入登录信息
new UiObject(new UiSelector().resourceId("com.example.demo4:id/username")).setText(username);
new UiObject(new UiSelector().resourceId("com.example.demo4:id/password")).setText(password);
```

UiObject类

案例

```
public void testClear() throws UiObjectNotFoundException, InterruptedException {
    instrumentation = InstrumentationRegistry.getInstrumentation();
    UiDevice.getInstance(instrumentation).pressHome();
    Util util = new Util("Demo4");
    util.comming();
    UiSelector enusername=new UiSelector().resourceId("com.example.demo4:id/username");
    UiObject obj1 = new UiObject(enusername);
    obj1.setText("hi");
    Thread.sleep(1000);
    obj1.clearTextField();
    Thread.sleep(1000);
    obj1.setText("hello");
    Thread.sleep(1000);
    UiDevice.getInstance(getInstrumentation()).pressHome();
}
```

UiObject类

获取对象的属性与属性的判断

返回值	API	说明
Rect	getBounds()	获得对象矩形坐标，矩形左上角坐标与右下角坐标
int	getChildCount()	获得下一级子类数量
String	getClassName()	获得对象类名属性的类名文本
String	getContentDescription()	获得对象的描述属性的描述文本
String	getPackageName()	获得对象包名属性的包名文本
String	getText()	获得对象的文本属性中的文本
Rect	getVisibleBounds()	返回可见视图的范围，如果视图的部分是可见的，只有可见部分报告的范围

UiObject类

获取对象的属性与属性的判断

```
int count = new UiObject(new
UiSelector().className("android.view.View").instance(4)).getChildCount();
for(int j=0;j<count;j++){
                UiObject child = obj.getChild(new UiSelector().index(j));
               System.out.println(child.getText());
}
String rect = new UiObject(new UiSelector().text("QQ")).getBounds().toString();
assertEquals("Rect(500,534-710,720)",rect);
```

获取父类与子类节点

返回值	API	说明
UiObject	getChild(UiSelector selector)	获得对象的子类对象，可以递归获取子孙当中某个对象
UiObject	getFromParent(UiSelector selector)	从父类获取子类，按照UiSeletor获取兄弟类（递归）

UiObject类

属性的判断

返回值	API	说明
boolean	isCheckable()	检查对象的checkable属性是否为true
boolean	isChecked()	检查对象的checked属性是否为true
boolean	isClickable()	检查对象的clickable属性是否为true
boolean	isEnabled()	检查对象的enabled属性是否为true
boolean	isFocusable()	检查对象的focusable属性是否为true
boolean	isFocused()	检查对象的focused属性是否为true
boolean	isLongClickable()	检查对象的longclickable属性是否为true
boolean	isScrollable()	检查对象的scrollable属性是否为true
boolean	isSelected()	检查对象的selected属性是否为true

UiObject类

手势的操作

- 两指平移
- 多指平移
- 两指合拢o---> <---o
- 两指扩张<---oo--->

返回值	API	说明
boolean	performMultiPointerGesture(PointerCoords[]... touches)	执行单手指触控手势，可定义任意手势，与形状
boolean	performTwoPointerGesture(Point startPoint1, Point startPoint2, Point endPoint1, Point endPoint2, int steps)	执行任意两个手指触控手势，模拟两个手指手势
boolean	pinchIn(int percent, int steps)	手势操作，两点向内收缩
boolean	pinchOut(int percent, int steps)	手势操作，两点向外张开

UiObject类

判断对象是否存在

返回值	API	说明
boolean	waitForExists(long timeout)	等待对象出现
boolean	waitUntilGone(long timeout)	等待对象消失
boolean	exists()	检查对象是否存在

```
UiObject obj = new UiObject(new
UiSelector().resourceId("com.example.demo4:id/welcomeinfo"));
assertNotNull(obj);
```

Uiautomator API详解

- InstrumentationRegistry类
- UiDevice 类
- By Selector和By类
- UiSelector类
- UiObject类
- **UiObject2类**
- UiCollection类
- UiScrollable类
- UiWatcher类
- Configurator类
- 断言

UiObject2类

UiObject2类是UiObject的升级类，由于目前不太完善，仅做简单介绍

```
instrumentation = InstrumentationRegistry.getInstrumentation();
mDevice = UiDevice.getInstance(instrumentation);
mDevice.pressHome();
UiObject2 UiObject2=mDevice.findObject(By.desc("Apps"))
```

UiObject2类

动作函数

API	说明
clear()	清楚编辑框内的内容
click()	点击一个对象
clickAndWait(EventCondition<R> condition, long timeout)	点击一个对象然后等待在超时的时间内条件满足则通过，否则抛出异常
drag(Point dest, int speed)	自定义速度拖拽这个对象到指定位置
drag(Point dest)	拖拽这个对象到指定位置
longClick()	长按某个对象
scroll(Direction direction, float percent)	对该对象执行一个滚动操作
scroll(Direction direction, float percent, int speed)	自定义速度，对该对象执行一个滚动操作
setText(String text)	设置文本内容
legacySetText(String text)	通过发送keycode，设置文本内容

UiObject2类

手势动作模拟

API	说明
pinchClose(float percent, int speed)	自定义速度执行收缩手势
pinchClose(float percent)	执行收缩手势
pinchOpen(float percent, int speed)	自定义速度执行展开手势
pinchOpen(float percent)	执行展开手势
fling(Direction direction)	执行一个扫动手势，Direction代表为起点方向
fling(Direction direction, int speed)	自定义速度，执行一个扫动手势
swipe(Direction direction, float percent, int speed)	执行一个滑动操作，可自定义滑动距离和速度
swipe(Direction direction, float percent)	执行一个滑动操作
setGestureMargin(int margin)	以像素为单位，设置手势边缘
setGestureMargins(int left, int top, int right, int bottom)	以像素为单位，设置手势边缘
pinchClose(float percent, int speed)	自定义速度执行收缩手势

UiObject2类

获取层级与条件判断

API	说明
findObject(BySelector selector)	搜索在这个对象之下的所有元素，并返回第一个与搜索条件匹配的
findObjects(BySelector selector)	搜索在这个对象之下的所有元素，并返回所有与搜索条件匹配的
getChildCount()	返回这个对象直属子元素的数量
getChildren()	返回这个对象下的直接子元素的集合
getParent()	返回该对象的父类
equals(Object object)	比较两个对象是否相等
hashCode()	获取对象的哈希码
hasObject(BySelector selector)	返回该对象是否存在
recycle()	回收该对象
wait(UiObject2Condition<R> condition, long timeout)	等待条件被满足
wait(SearchCondition<R> condition, long timeout)	等待条件被满足

UiObject2类

案例

```
public void testUiObject2() throws UiObjectNotFoundException, InterruptedException {
    instrumentation = InstrumentationRegistry.getInstrumentation();
    mDevice = UiDevice.getInstance(instrumentation);
    mDevice.pressHome();
    mDevice.findObject(By.desc("Apps")).click();
    UiScrollable appViews = new UiScrollable(new UiSelector().scrollable(true));
    appViews.setAsHorizontalList();
     UiObject settingsApp = appViews.getChildByText(
          new UiSelector().className(android.widget.TextView.class.getName()), "Demo5");
    settingsApp.clickAndWaitForNewWindow();
    mDevice.findObject(By.res("android:id/decor_content_parent")).click();
    Thread.sleep(4000);
    mDevice.findObject(By.res("android:id/decor_content_parent")).click();
    Thread.sleep(4000);
    mDevice.pressHome();
  }
```

Uiautomator API详解

- InstrumentationRegistry类
- UiDevice 类
- By Selector和By类
- UiSelector类
- UiObject类
- UiObject2类
- **UiCollection类**
- UiScrollable类
- UiWatcher类
- Configurator类
- 断言

UiCollection类

UiCollection是UiObject的子类，用来表示一个父控件，该控件下包含了子元素的集合。
一般使用一个容器类控件作为UiCollection对象，再通过两个条件来准确定位到UiCollection对象下的子元素(包含子孙元素)。
通过UiCollection，可以获取到某一控件下的某一个子控件 或者 获取其子控件的数目(包含子孙元素)。

```
UiCollection mUiCollection = new UiCollection(UiSelector Object);
```

UiCollection类

从集合中查找对象

返回值	API	说明
UiObject	getChildByDescription (UiSelector childPattern, String text)	通过两个条件：selector和description属性，定位到其下符合条件的子控件(包含子孙集合)
UiObject	getChildByText (UiSelector childPattern, String text)	通过两个条件：selector和text属性，定位到其下符合条件的子控件(包含子孙集合)
UiObject	getChildByInstance (UiSelector childPattern, int instance)	通过两个条件：selector和instance属性，定位到其下符合条件的子控件(包含子孙集合)

```
UiCollection collection = new UiCollection(new UiSelector().resourceId(resid_radioGroup));
UiSelector childPattern = new UiSelector().className("Android.widget.RadioButton");
UiObject radioBtn1 = collection.getChildByInstance(childPattern,1);
```

获取某种搜索条件组件的数量

返回值	API	说明
int	getChildCount (UiSelector childPattern)	按照UiSelector查找条件递归查找所有符合条件的子孙集合的数量

Uiautomator API详解

- InstrumentationRegistry类
- UiDevice 类
- By Selector和By类
- UiSelector类
- UiObject类
- UiObject2类
- UiCollection类
- **UiScrollable类**
- UiWatcher类
- Configurator类
- 断言

UiScrollable类

1. UiScrollable是UiCollection的子类
2. UiScrollable专门处理滚动时间，提供各种滚动方法

常用功能有：向前滚动、向后滚动、快速滚动、滚动到某个对象、设置滚动方向、设置滚动次数等。

```
UiScrollable scroll=new UiScrollable(UiSelector Object)
```

快速滚动

- 步长：从一点到另一点使用的时间。步长越短滚动越快，反之步长越长滚动越慢。
- 扫动次数：触发滚动的次数。

返回值	API	说明
boolean	flingBackward()	以步长为5快速向后滑动
boolean	flingForward()	以步长为5快速向前滑动
boolean	flingToBeginning(int maxSwipes)	自定义扫动次数以步长为5快速滑动到开启
boolean	flingToEnd(int maxSwipes)	自定义扫动次数以步长为5快速滑动到结束

UiScrollable类

- lingBackward(),模拟的动作是：固定住一点，然后往后拉。
- flingForward(),模拟的动作是：固定住一点，然后往前拉。
- flingToBeginning(int maxSwipes),模拟的动作是：往列表顶部方向拖动（效果等同于flingBackward()）,经过实测1次步长并没有一直到列表顶部。
- flingToEnd(int maxSwipes),模拟的动作是：往列表底部方向拖动（效果等同于flingForward()),经过实测1次步长并没有一直到列表底部。

UiScrollable类

案例

```
public void testFling() throws UiObjectNotFoundException
  {
     instrumentation = InstrumentationRegistry.getInstrumentation();
     UiDevice.getInstance(instrumentation).pressHome();
     Util util=new Util("Browser");
     util.enterAPP();
     UiObject appsTab = new UiObject(new UiSelector().text("Apps"));
     appsTab.click();
     UiScrollable scroll = new UiScrollable(new UiSelector().scrollable(true));
     scroll.setAsHorizontalList();
     scroll.flingToEnd(5);
     scroll.flingToBeginning(5);
     UiDevice.getInstance(getInstrumentation()).pressHome();
  }
```

UiScrollable类

获取列表子元素

返回值	API
UiObject	getChildByDescription(UiSelector childPattern, String text, boolean allowScrollSearch)
	是否允许滚动查找获取具备UiSelector条件元素集合后再以文本描述条件查找对象
UiObject	getChildByDescription(UiSelector childPattern, String text)
	默认滚动获取具备UiSelector条件的元素集合后再以文本描述条件查找对象
UiObject	getChildByInstance(UiSelector childPattern, int instance)
	获取具备UiSelector条件的子集，再从子集中按照实例筛选想要的元素（不滚动）
UiObject	getChildByText(UiSelector childPattern, String text, boolean allowScrollSearch)
	是否允许滚动获取具备UiSelector条件的元素集合后再以文本条件查找对象
UiObject	getChildByText(UiSelector childPattern, String text)
	默认滚动获取具备UiSelector条件元素集合后再以文本条件的查找对象

UiScrollable类

案例

```
UiObject settingsApp = appViews.getChildByText(
        new UiSelector().className(android.widget.TextView.class.getName()), "Demo4");
//通过滚动来查找Demo4
```

注：默认滚动方向是向列表底部，到列表底部或者顶部会换个方向继续查找。

获取与设置最大滚动次数常量值

返回值	API	说明
int	getMaxSearchSwipes()	获取执行搜索滑动过程中的最大滑动次数，默认常量为30
UiScrollable	setMaxSearchSwipes(int swipes)	设置最大可扫动次数

UiScrollable类

滑动区域校准常量设置与获取

校准常量：指的是滑动操作坐标时（起点和终点）的偏移量，即控件上不可滑动的区域占比。

返回值	API	说明
double	getSwipeDeadZonePercentage()	校准常量默认值为0.1(10%)
UiScrollable	setSwipeDeadZonePercentage(double swipeDeadZonePercentage)	设置控件上不可滑动的区域占比

```
UiScrollable scroll = new UiScrollable(new  UiSelector.className("android.widget.ListView"));
scroll.setSwipeDeadZonePercentage(0.15); //设置listview校准常量为0.15，即距离listview顶部15%和底部15%的区域不可滑动，只有控件中部70%的区域可滑动。当校准常量设置为0.5时，控件的可滑动区域为0，滑动的动作效果将和单击的效果一样。
```

UiScrollable类

向前与向后滚动

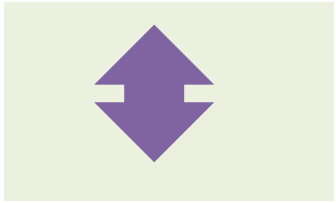
返回值	API	说明
boolean	scrollBackward(int steps)	自定义步长向后滑动
boolean	scrollBackward()	以默认步长55向后滑动
boolean	scrollDescriptionIntoView(String text)	滚动到描述所在位置，并且尽量让它居于屏幕中央
boolean	scrollForward()	以默认步长55向前滚动
boolean	scrollForward(int steps)	自定义步长向前滚动。 当步长很短（如1）时，滑动效果同单机；当步长很长时，滑动效果同双击。

UiScrollable类

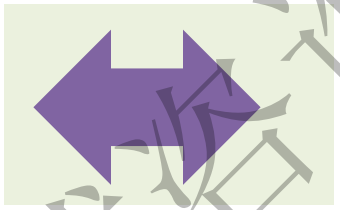
滚动到某个对象

返回值	API	说明
boolean	scrollIntoView(UiSelector selector)	滚动到条件元素所在位置，并且尽量让其居于屏幕中央
boolean	scrollIntoView(UiObject obj)	滚动到对象所在位置，并且尽量让其居于屏幕中央
boolean	scrollTextIntoView(String text)	滚动到文本对象所在位置，并且尽量让其居于屏幕中央
boolean	scrollToBeginning(int maxSwipes)	自定义最大滚动次数，滚动到开始位置
boolean	scrollToBeginning(int maxSwipes, int steps)	自定义最大滚动次数与步长，滚动到开始位置
boolean	scrollToEnd(int maxSwipes, int steps)	自定义最大滚动次数与步长，滚动到结束位置
boolean	scrollToEnd(int maxSwipes)	自定义最大滚动次数，滚动到结束位置

UiScrollable类



垂直滚动



水平滚动

设置滚动方向

- 纵向滚动：上方向为Forward,下方向为Backward()

scrollBackward()效果等同于flingBackward()

scrollForward()效果等同于flingForward()

- 水平滚动：左为Forward,右方向为Backward()

滑动属于平动，在物体上找两点连线始终平行。

滚动属于平面运动，可以分解为平动和转动

举个很简单的例子吧，汽车急刹车的时候轮胎就是在滑动，正常前进的时候轮胎就是在滚动.

返回值	API	说明
UiScrollable	setAsHorizontalList()	设置滚动方向设置为水平滚动
UiScrollable	setAsVerticalList()	设置滚动方向设置为纵向滚动

UiScrollable类

案例

```
public void testScroll() throws UiObjectNotFoundException
   {
       instrumentation = InstrumentationRegistry.getInstrumentation();
      UiDevice.getInstance(instrumentation).pressHome();
      Util util=new Util("Browser");
      util.enterAPP();
      UiObject appsTab = new UiObject(new UiSelector().text("Apps"));
      appsTab.click();
      UiScrollable scroll = new UiScrollable(new UiSelector().scrollable(true));
      scroll.setAsHorizontalList();
      scroll.scrollForward();
      scroll.scrollBackward();
      UiDevice.getInstance(getInstrumentation()).pressHome();
   }
```

Uiautomator API详解

- InstrumentationRegistry类
- UiDevice 类
- By Selector和By类
- UiSelector类
- UiObject类
- UiObject2类
- UiCollection类
- UiScrollable类
- **UiWatcher类**
- Configurator类
- 断言

UiWatcher类

1. Uiwatcher用于处理脚本执行过程中遇到非预想的步骤
2. UiWatcher使用场景
   ① 测试过程中来了一个电话
   ② 测试过程中来了一条短信
   ③ 测试过程中闹钟响了
   ④ 出现各种非预想的步骤
3. 中断监听检查条件
4. public boolean checkForCondition();

```
UiDevice.getInstance(getInstrumentation()).registerWatcher("phone", new UiWatcher(){...});
```

返回值	API	说明
void	registerWatcher(String name,Uiwatcher watcher)	注册一个监听器，当Uiselector无法匹配到对象的时候，触发监听器
void	removeWatcger(String name)	取消之前注册的指定监听器
void	resetWatcherTriggers()	充值已触发过的UiWatcher,重置后相当于没运行过
void	runWatchers()	强制运行所有的监听器

UiWatcher类

1. UiWatcher写在所有的用例之前，要在用例之前启动，启动之后后面的正常用例才能执行，执行过程中出现异常的时候调监听器
2. 问题：若测试用例运行较快，有时候不是完美的被打断的时候，监听器有时候也会失败，不会刚好监听到异常情况，这个时候可以把用例之间执行步骤停顿的时间加长，sleep（2000）之类
3. 如果循环体（方法体也是）被打断以后，就算监听器处理完异常也不可能再回到循环体里面，可以简单的复制代码循环，如果代码太长就算了...
4. UiDevice 是不会触发监听器的，比如我们按home键、菜单键调用到UiDevice的功能，他的顺序执行是不会调用到UiWatcher的

UiWatcher类

检查监听器

返回值	API	说明
boolean	hasAnyWatcherTriggered()	检查是否有监听器触发过
boolean	hasWatcherTriggered(String watcgerName)	检查某个特定的监听器是否触发过

检查监听器代码卸载所有的正常测试用例代码之后，在测试完成之后，打印是否触发过监听器。

UiWatcher类

案例

先启动一个通话，运行下面程序，在测试程序运行过程中，激活开始的通话

```
public void testWatcher() throws UiObjectNotFoundException {
    instrumentation = InstrumentationRegistry.getInstrumentation();
    UiDevice.getInstance(instrumentation).registerWatcher("phone", new UiWatcher() {
    @Override
    public boolean checkForCondition() {
      //电话监听
      UiObject call = new UiObject(new UiSelector().resourceId("com.android.dialer:id/primary_call_banner"));//由接听按钮判断为来电
      UiObject call_reject = new UiObject(new UiSelector().resourceId("com.android.dialer:id/floating_end_call_ac
      if (call.exists()) {
        System.out.println("电话监听器被触发啦!!!");
        try {
          call_reject.click();
          return true;
        } catch (UiObjectNotFoundException e) {
          e.printStackTrace();
        }}
```

Node Detail	
index	0
text	
resource-id	com.android.dialer:id/floating_end_call_action_button
class	android.widget.ImageButton
package	com.android.dialer
content-desc	End

Node Detail	
index	0
text	
resource-id	com.android.dialer:id/primary_call_banner
class	android.widget.LinearLayout
package	com.android.dialer
content-desc	

UiWatcher类

案例

```
            return false;
        }
    });
    testScroll();
    System.out.println("是否有监听器触发过："+
UiDevice.getInstance(getInstrumentation()).hasAnyWatcherTriggered());
    System.out.println("电话监听器是否被触发过："+
UiDevice.getInstance(getInstrumentation()).hasWatcherTriggered("phone"));
    UiDevice.getInstance(getInstrumentation()).removeWatcher("phone");///移除之后，后面的测试过程中有电话拨进来，不会调用监听器
  }
```

Uiautomator API详解

- InstrumentationRegistry类
- UiDevice 类
- By Selector和By类
- UiSelector类
- UiObject类
- UiObject2类
- UiCollection类
- UiScrollable类
- UiWatcher类
- **Configurator类**
- 断言

Configurator类

用于设置脚本动作的默认延时

1. 可调节两个模拟动作之间的默认间隔
2. 可调节输入文本的输入时间间隔
3. 可调节每次滚动的时间间隔
4. 可调节等待系统空闲的默认时间

Configurator.getInstance().

延时项	默认延时	功能描述	API
动作	3s	设置延时	setActionAckonwledgmentTimeout(long timeout)
		获取默认延时	getActionAckonwledgmentTimeout()
键盘输入	0s	设置延时	setKeyInjectionDelay(long delay)
		获取默认延时	getKeyInjectionDelay()
滚动	200ms	设置延时	setScrollAcknowledgmentTimeout(long timeout)
		获取默认延时	getScrollAcknowledgmentTimeout()
空闲	10s	设置延时	setWaitForIdleTimeout(long timeout)
		获取默认延时	getWaitForIdleTimeout()
查找组件	10s	设置延时	setWaitForSelectorTimeout(long timeout)
		获取默认延时	getWaitForSelectorTimeout()

Configurator类

案例

```
public void testConfigurator() throws UiObjectNotFoundException, InterruptedException {
    instrumentation = InstrumentationRegistry.getInstrumentation();
    UiDevice.getInstance(instrumentation).pressHome();
    Util util=new Util("Browser");
    util.enterAPP();
    UiObject appsTab = new UiObject(new UiSelector().text("Apps"));
    appsTab.click();
    //1.动作延时
    long action= Configurator.getInstance().getActionAcknowledgmentTimeout();
    //输出默认延时
    System.out.println("动作默认延时为： "+action);
    //获取屏幕高度和宽度
    int y=UiDevice.getInstance(getInstrumentation()).getDisplayHeight();
    int x=UiDevice.getInstance(getInstrumentation()).getDisplayWidth();
    //默认延时下动作间隔时间感受
    UiDevice.getInstance(getInstrumentation()).swipe(x-10, y/2, x-200, y/2, 10);
    UiDevice.getInstance(getInstrumentation()).swipe(x-10, y/2, x-200, y/2, 10);
    Thread.sleep(2000);
    //设置延时
    Configurator.getInstance().setActionAcknowledgmentTimeout(100000);
```

动作默认延时为： 3000

Configurator类

```
//使用设置延时下动作间隔时间感受
UiDevice.getInstance(getInstrumentation()).swipe(x-10, y/2, x-200, y/2, 10);
UiDevice.getInstance(getInstrumentation()).swipe(x-10, y/2, x-200, y/2, 10);
//使用完毕后调回默认延时，切记！！！
Configurator.getInstance().setActionAcknowledgmentTimeout(action);

//2.键盘延时
long print=Configurator.getInstance().getKeyInjectionDelay();
//输出键盘输入默认延时
System.out.println("键盘输入默认延时为： "+print);
new UiObject(new UiSelector().className("android.widget.TextView").index(0)).click();
UiScrollable appViews = new UiScrollable(new UiSelector().scrollable(true));
UiObject settingsApp = appViews.getChildByText(
      new UiSelector().className(android.widget.TextView.class.getName()), "Demo4");
settingsApp.clickAndWaitForNewWindow();
//获取输入框
UiObject set=new UiObject(new UiSelector().resourceId("com.example.demo4:id/username"));
//感受默认延时下输入间隔
set.setText("aabbccd");
Thread.sleep(2000);
```

键盘输入默认延时为： 0

Configurator类

```
set.clearTextField();
//修改默认延时为1秒
Configurator.getInstance().setKeyInjectionDelay(1000);
//感受修改延时后输入间隔
set.setText("aabbccd");
Thread.sleep(2000);
set.clearTextField();
//切记改回原始时间
Configurator.getInstance().setKeyInjectionDelay(print);

//3.滚动延时
long scroll=Configurator.getInstance().getScrollAcknowledgmentTimeout();
//输出默认延时
System.out.println("滚动默认延时： "+scroll);
//获取列表集合
UiScrollable scroll1=new UiScrollable(new UiSelector().className("android.widget.EditText"));
//感受默认滚动延时间隔
scroll1.flingToEnd(5);
//设置默认滚动时间间隔为2S
Configurator.getInstance().setScrollAcknowledgmentTimeout(2000);
```

滚动默认延时： 200

Configurator类

```
//感受设置后的滚动时间间隔
scroll1.flingToEnd(5);
//切记使用完毕改回为默认时间间隔
Configurator.getInstance().setScrollAcknowledgmentTimeout(scroll);
//4.空闲延时
long wait=Configurator.getInstance().getWaitForIdleTimeout();
System.out.println("空闲默认延时为： "+wait);
//5.查找组件延时
long selector=Configurator.getInstance().getWaitForSelectorTimeout();
//输出默认时间间隔
System.out.println("查找组件默认延时为： "+selector);
//随意查找一个对象
UiScrollable scroll2=new UiScrollable(new UiSelector().resourceId("com.example.demo4:id/username"));
scroll2.click();
//修改默认时间间隔
Configurator.getInstance().setWaitForSelectorTimeout(3000);
//感受修改后的时间间隔
UiScrollable scroll3=new UiScrollable(new UiSelector().resourceId("com.example.demo4:id/password"));
scroll3.click();
UiDevice.getInstance(getInstrumentation()).pressHome();}
```

空闲默认延时为： 10000

查找组件默认延时为 ： 10000

Configurator类

Configurator实际使用

模拟双击与快速多单击动作

在Util.java中加入

```
public void quicklyClick(int num,int x,int y){
    //获取动作间隔时间
    long actionTime=Configurator.getInstance().getActionAcknowledgmentTimeout();
    //设置动作间隔时间
    Configurator.getInstance().setActionAcknowledgmentTimeout(0);
    //设置操作步骤
    for(int i=0;i<=num;i++){
      UiDevice.getInstance(getInstrumentation()).click(x, y);
    }
    //最后别忘了恢复默认时间间隔，避免影响后面其他的用例
    Configurator.getInstance().setActionAcknowledgmentTimeout(actionTime);
  }
```

```
public void testQuicklyClick() throws UiObjectNotFoundException {
    instrumentation = InstrumentationRegistry.getInstrumentation();
    UiDevice.getInstance(instrumentation).pressHome();
    Util util=new Util("Browser");
    util.enterAPP();
    //获取屏幕高和宽
    int y=UiDevice.getInstance(getInstrumentation()).getDisplayHeight();
    int x=UiDevice.getInstance(getInstrumentation()).getDisplayWidth();
    util.quicklyClick(4, x/2,y/2);
  }
```

Configurator类

Configurator实际使用

使用keyCode快速输入

在Util.java中加入

```
public void quicklyKeyCode(String input){
    //获取动作间隔时间
    long action=Configurator.getInstance().getActionAcknowledgmentTimeout();
    //设置动作间隔时间
    Configurator.getInstance().setActionAcknowledgmentTimeout(1);
    //方法中用到的数字来自androidKeyCode表
    for (int i=0;i<input.length();i++){
       char c=input.charAt(i);
       //判断输出
       if (c>48&&c<=57){
          UiDevice.getInstance(getInstrumentation()).pressKeyCode(c-41);
       }else if(c>=97&&c<=122){
          UiDevice.getInstance(getInstrumentation()).pressKeyCode(c-68);
       }else if (c>=65&&c<=90){
          UiDevice.getInstance(getInstrumentation()).pressKeyCode(c-36,1);
       }
    }
    //千万别忘记恢复默认
    Configurator.getInstance().setActionAcknowledgmentTimeout(action);
 }
```

Configurator类

Configurator实际使用

使用keyCode快速输入

```
public void testQuicklyKeyCode() throws UiObjectNotFoundException {
      instrumentation = InstrumentationRegistry.getInstrumentation();
      UiDevice.getInstance(instrumentation).pressHome();
      Util util=new Util("Demo4");
      util.comming();
      util.quicklyKeyCode("jerry");
      UiDevice.getInstance(getInstrumentation()).pressHome();
   }
```

Uiautomator API详解

- InstrumentationRegistry类
- UiDevice 类
- By Selector和By类
- UiSelector类
- UiObject类
- UiObject2类
- UiCollection类
- UiScrollable类
- UiWatcher类
- Configurator类
- **断言**

断言

方法名	描述
assertEquals(boolean,boolean)	如果期望(expected)和实际(actual)相等则通过，否则失败
assertEquals(String,boolean,boolean)	如果期望(expected)和实际(actual)相等则通过，否则失败
assertEquals(byte,byte)	如果期望(expected)和实际(actual)相等则通过，否则失败
assertEquals(String,byte,byte)	如果期望(expected)和实际(actual)相等则通过，否则失败
assertEquals(char,char)	如果期望(expected)和实际(actual)相等则通过，否则失败
assertEquals(String,char,char)	如果期望(expected)和实际(actual)相等则通过，否则失败
assertEquals(int,int)	如果期望(expected)和实际(actual)相等则通过，否则失败
assertEquals(String,int,int)	如果期望(expected)和实际(actual)相等则通过，否则失败
assertEquals(long,long)	如果期望(expected)和实际(actual)相等则通过，否则失败
assertEquals(String,long,long)	如果期望(expected)和实际(actual)相等则通过，否则失败
assertEquals(Object,Object)	如果期望(expected)和实际(actual)相等则通过，否则失败

断言

方法名	描述
assertEquals(String,Object,Object)	如果期望(expected)和实际(actual)相等则通过，否则失败
assertEquals(short,short)	如果期望(expected)和实际(actual)相等则通过，否则失败
assertEquals(String,short,short)	如果期望(expected)和实际(actual)相等则通过，否则失败
assertEquals(String,String)	如果期望(expected)和实际(actual)相等则通过，否则失败
assertEquals(String,String,String)	如果期望(expected)和实际(actual)相等则通过，否则失败
assertEquals(double,double,double)	如果期望(expected)和实际(actual)相等则通过，否则失败
assertEquals(String,double,double,double)	如果期望(expected)和实际(actual)相差不超过精度值(delta)则通过，否则失败
assertEquals(float,float,float)	如果期望(expected)和实际(actual)相差不超过精度值(delta)则通过，否则失败
assertEquals(String,float,float,float)	如果期望(expected)和实际(actual)相差不超过精度值(delta)则通过，否则失败

断言

方法名	描述
assertFalse(boolean)	如果条件(condition)为False则通过，否则失败
assertFalse(String,boolean)	如果条件(condition)为False则通过，否则失败
assertTrue(boolran)	如果条件(condition)为True则通过，否则失败
assertTrue(String,boolran)	如果条件(condition)为True则通过，否则失败
assertNotNull(Object)	如果条件(condition)为非空则通过，否则失败
assertNotNull(String,Object)	如果条件(condition)为非空则通过，否则失败
assertNull(Object)	如果条件(condition)为空则通过，否则失败assertNull
assertNotSame(Object,object)	如果期望(expected)和实际(actual)引用不同的内存对象对象则通过，否则失败
assertNoteSame(String,Object,Object)	如果期望(expected)和实际(actual)引用不同的内存对象对象则通过，否则失败
assertSame(Object,Object)	如果期望(expected)和实际(actual)引用相同的内存对象对象则通过，否则失败

断言

方法名	描述
assertSame(String,Object,Object)	如果期望(expected)和实际(actual)引用相同的内存对象对象则通过，否则失败
fail()	用例立即失败
fail(String)	用例立即失败，且抛出指定消息
failNotEquals(String,Object,Object)	用例立即失败，且抛出指定消息与期望、实际值不相等的消息
failNotSame(String,String,String)	用例立即失败，且抛出指定消息与期望、实际值不相等的消息
failSame(String)	用例立即失败，且抛出指定消息

目录

- Andriod 各种UI测试框架介绍
- **UiAutomator UI自动化测试框架**
  - 环境准备
  - 建立测试工程
  - Uiautomator API详解
  - **建立测试集**
  - 实践

建立测试集

建立JUnit 4测试体系

在Util.java中建立

```
//删除最近使用的APP（删除测试的APP）
public void deleteAPP(Instrumentation instrumentation) throws UiObjectNotFoundException,
RemoteException {
   UiDevice.getInstance(instrumentation).pressRecentApps();
   UiObject recentapp = new UiObject(new
UiSelector().resourceId("com.android.systemui:id/dismiss_task"));
   do{
      recentapp.waitForExists(2000);
      if(recentapp.exists()){
         recentapp.swipeLeft(5);
      }
   }while(recentapp.exists());}
```

建立测试集

建立JUnit 4测试体系

在测试程序中建立

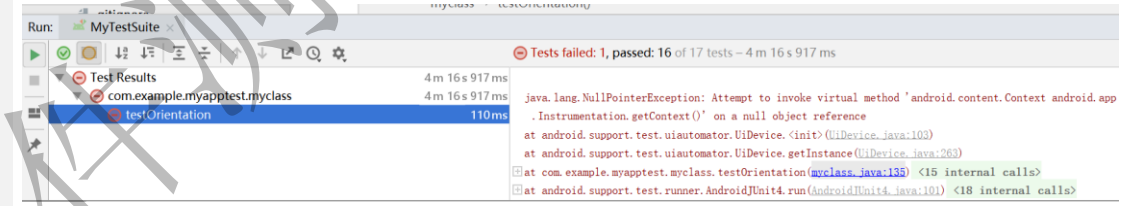
```
import org.junit.Test;
import org.junit.Before;
import org.junit.After;
...
   @Before
   public void setup(){
      instrumentation = InstrumentationRegistry.getInstrumentation();
      UiDevice.getInstance(instrumentation).pressHome();}
   @After
   public void Tear() throws RemoteException,UiObjectNotFoundException {
      UiDevice.getInstance(instrumentation).pressHome();
      Util util=new Util("Demo4");
      util.deleteAPP(instrumentation);}
```

然后把测试程序开始以及结尾处的响应代码删除。

建立测试集

MyTestSuite.java

```
package com.example.myapptest;

import  org.junit.runner.RunWith;
import  org.junit.runners.Suite;

@RunWith(Suite.class )
@Suite.SuiteClasses( {
        myclass.class,
        } )

public class MyTestSuite {
}
```



目录

- Andriod 各种UI测试框架介绍
- **UiAutomator UI自动化测试框架**
  - 环境准备
  - 建立测试工程
  - Uiautomator API详解
  - **实践**

实践

1. 进入Demo4
2. 注册“用户名”和“密码”
3. 用注册的“用户名”和“密码”登录系统
4. 检查是否登录成功
5. 清尾操作:
6. 退出登录页面
7. 点击【清除】按键

结束

Thanks !