

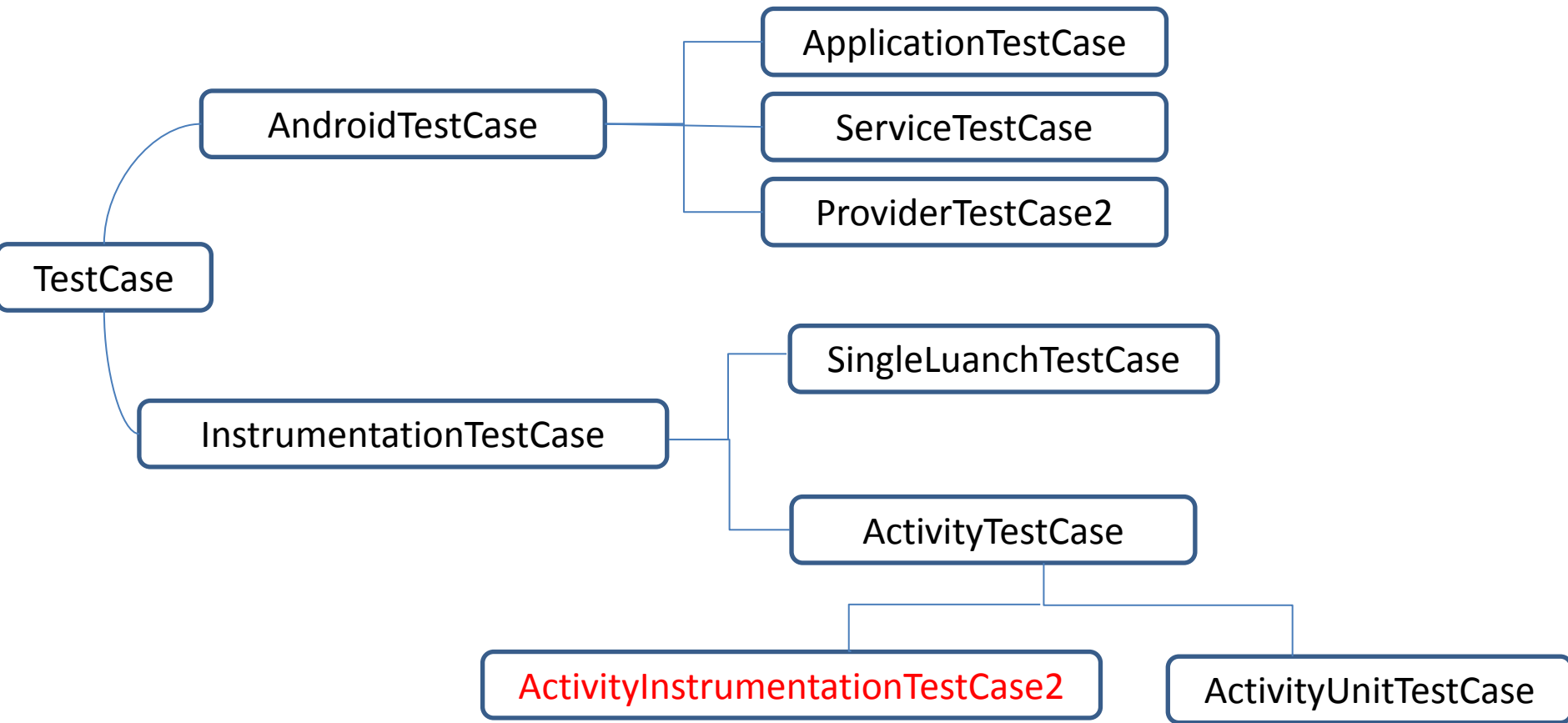
# Robotium 自动化测试框架

<https://github.com/RobotiumTech/robotium/wiki/Downloads>

# 本章大纲

- **Android单元测试类**
- Robotium是什么
- Robotium白盒自动化测试
- Robotium录制回放工具
- Robotium黑盒自动化测试
- Robotium常用API
- 测试脚本的批量运行

# Android单元测试类

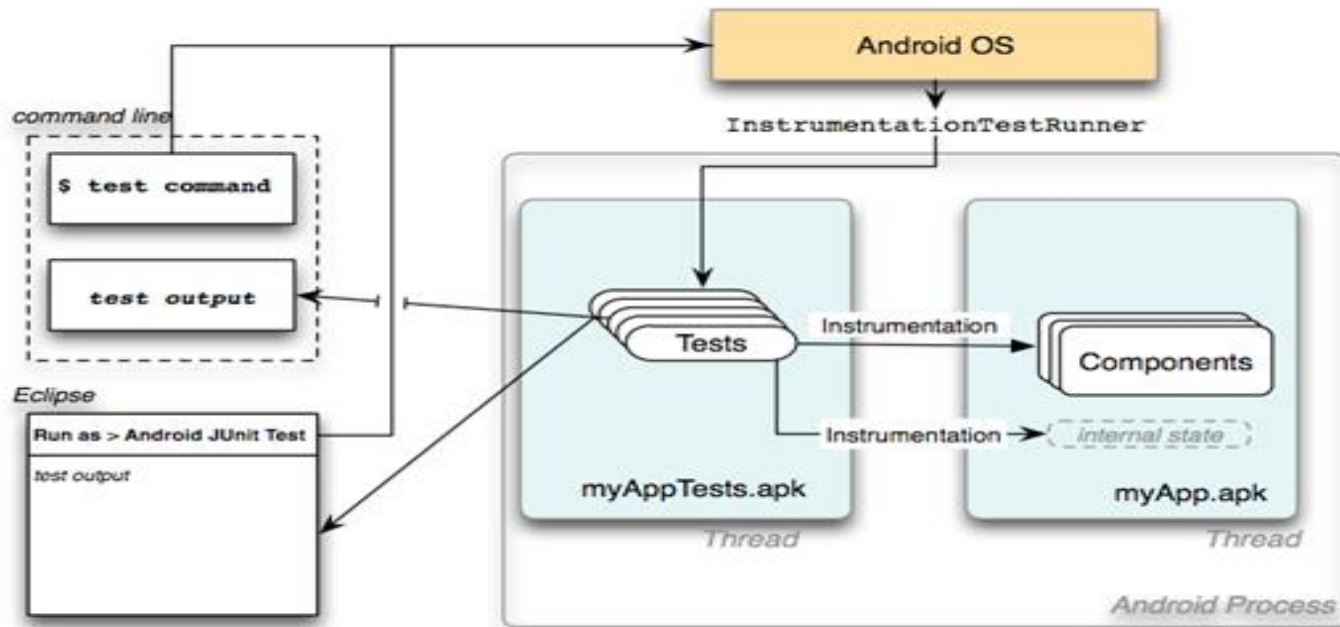


# Instrumentation框架

Instrumentation是Android测试的核心框架，可使用它进行Android应用的单元测试和自动化测试。

Instrumentation可以在主程序启动之前，**创建模拟的Context**；发送UI事件给应用程序；检查程序当前运行的状态；控制Android如何加载应用程序，控制应用程序和控件的生命周期；可直接调用控件的方法，对控件的属性进行检查和修改。Instrumentation框架通过将主程序和测试程序运行在同一个进程来实现这些功能。通过在测试工程的manifest文件中添加元素来指定要测试的应用程序。

# Instrumentation 框架



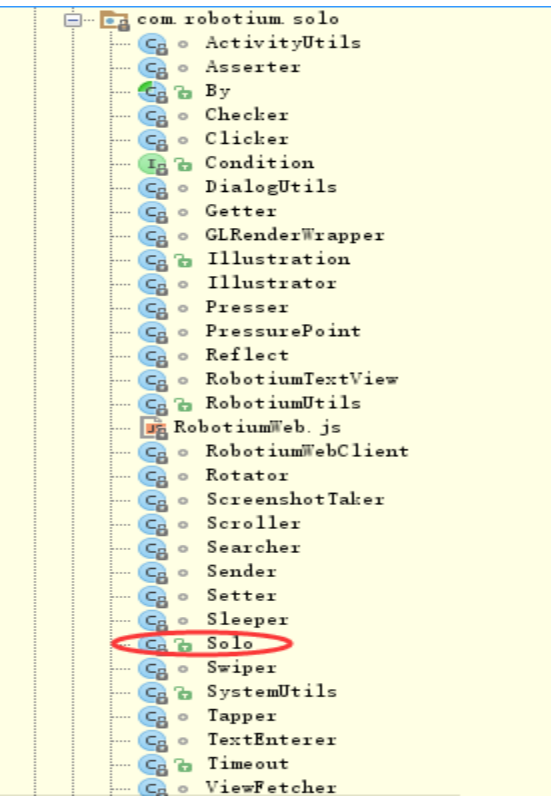
# 本章大纲

- Android单元测试类
- Robotium介绍
- Robotium白盒自动化测试
- Robotium录制回放工具
- Robotium黑盒自动化测试
- Robotium常用API
- 测试脚本的批量运行

# Robotium介绍

- Robotium是一款开源的Android自动化测试框架，主要针对Android平台的应用进行黑盒自动化测试，它提供了模拟各种手势操作（点击、长按、滑动等）、查找和断言机制的API，能够对各种控件进行操作。
- 优势：
  - ✓ 同时支持Native应用和Hybrid应用。
  - ✓ 由于是基于Instrumentation的测试，测试代码运行于被测应用所在的进程，控件识别与模拟UI事件都可以快速执行，因此测试用例执行速度更快。
  - ✓ 由于是通过在运行时识别控件而非通过固定坐标方式，因此测试用例可以更健壮。
  - ✓ 由于支持黑盒方式，不需要深入了解被测应用即可开展测试，因此编写用例花费的时间可以更少。
  - ✓ 由于可以通过Maven、Gradle或者Ant运行测试用例，因此可以很好地作为持续集成的一部分
- 局限性：
  - ✓ 由于是基于Instrumentation的事件发送，因此无法跨应用。
  - ✓ 代码运行在被测进程，可能影响被测进程的内存、CPU占用，若用于性能监控数据会有误差

# Robotium提供的类



- By: Web元素的选择器
- Condition: 接口类, 用于等待
- RobotiumUtils: 工具类
- Solo.Config: Solo配置类
- SystemUtils: 系统级工具类
- Timeout: Solo配置类。
- WebElement: Web元素的抽象类
- Getter: 提供控件获取相关API
- ActivityUtils: 提供Activity相关
- Asserter: 提供断言相关的API
- Clicker: 提供模拟点击相关的API
- ScreenshotTaker: 提供截图相关的API



# Robotium提供的类

- ✓ Scroller: 提供滚动相关的API
- ✓ Searcher: 提供控件搜索相关的API
- ✓ ViewFetcher: 提供控件过滤相关的API
- ✓ Waiter: 提供控件等待相关的API
- ✓ WebUtils: 提供Web支持相关的API
- ✓ Solo: 对外提供各种API, Solo类采用中介者模式, 持有com.robotium.solo包下的其他类的实例对象, 当我们调用Solo类中的API时, 大多数是转而调用com.robotium.solo包下其他类的方法

Robotium为了简化测试用例的编写, 将以上的这些类都置为protected, 对外只提供Solo类, 因此, 在编写测试用例时, 主要实例化Solo类即可。

# 本章大纲

- Android单元测试类
- Robotium是什么
- **Robotium白盒自动化测试**
- Robotium录制回放工具
- Robotium黑盒自动化测试
- Robotium常用API
- 测试脚本的批量运行



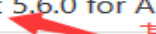

# 第一个Robotium实例

- <https://github.com/RobotiumTech/robotium/wiki/Downloads>

## Downloads

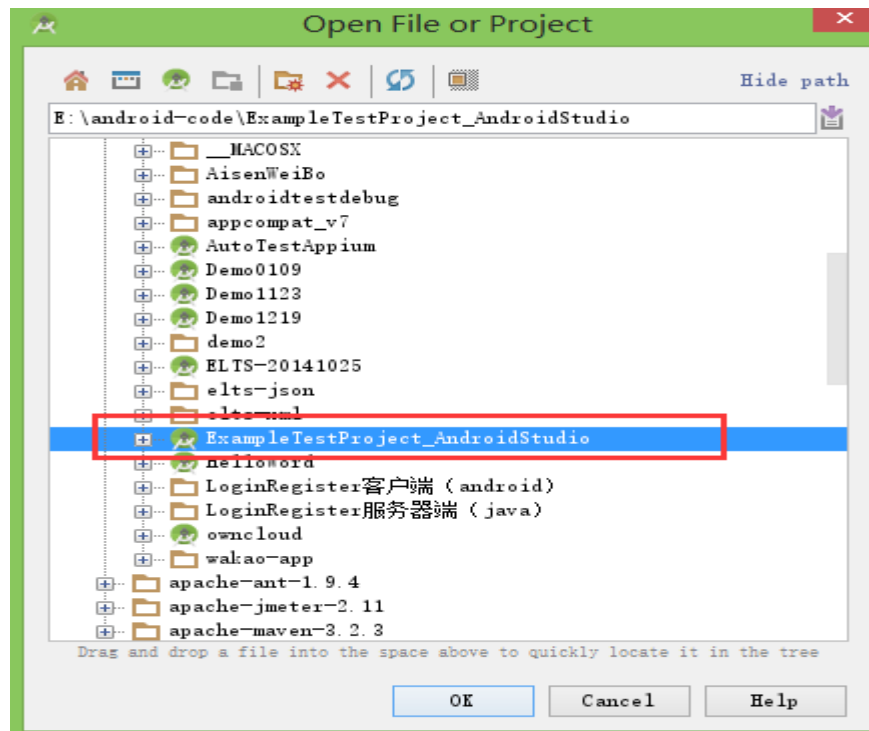
The files are hosted at [Bintray](#).

### Current Direct Downloads

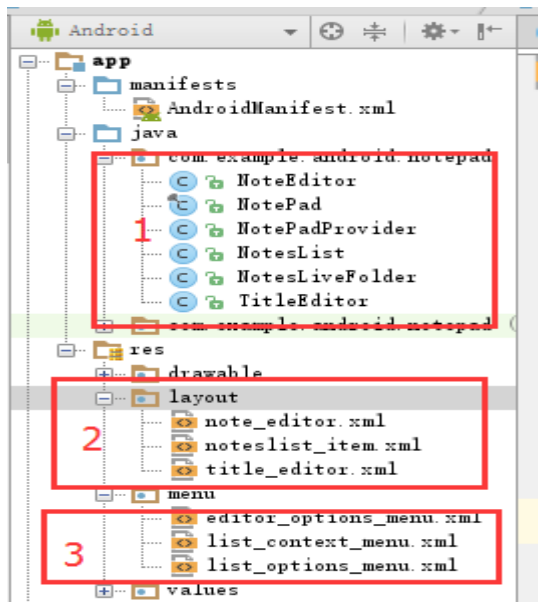
File	Description
<a href="#">robotium-solo-5.6.3.jar</a>	Robotium Solo 5.6.3  <b>jar包文件</b>
<a href="#">robotium-solo-5.6.3-javadoc.jar</a>	Robotium Solo 5.6.3 Javadoc  <b>帮助文件</b>
<a href="#">ExampleTestProject_AndroidStudio.zip</a>	Example Test Project 5.6.0 for Android Studio  <b>基于android</b>
<a href="#">ExampleTestProject_Eclipse_v5.5.1.zip</a>	Example Test Project 5.5.1 for Eclipse  <b>studio的样例源</b>

# 导入样例项目

- File->open



# 样例结构分析



1. 源代码的实现
2. 布局文件
3. 菜单文件

# 样例代码分析

- `NotesList Activity`: 列出所有的日记的界面。同时这个也是NotePad的主界面，点击NotePad应用图标之后就会进入到这个界面
- `NoteEditor Activity`: 日记增加或者编辑界面。用户可以进入到这个界面增加一个日记，同时也可以对已经有的日记进行编辑
- `TitleEditor Activity`: 日记标题修改界面。用户可以进入该界面进行日记标题的修改
- 上下文菜单`Context Menu`: 在`NotesList`界面长按一个日记项会弹出一个上下文菜单并提供多个选项给用户选择，用户可以通过上下文菜单来对该日记进行多个操作
- 系统菜单`Option Menu`: 用户按下你的安卓手机的系统菜单物理键会弹出系统菜单选项，在不同的界面会提供不同的选项功能帮组大家对日记进行操作

# robotium测试代码解析

- 测试类在设备或仿真器上运行，必须在前面加上`@RunWith(AndroidJUnit4.class)`注释
- `AndroidJUnitRunner`是一个可以用来运行JUnit 3和JUnit 4样式的测试类的Test Runner
- `setUp()`函数是在运行测试用例之前做一些准备的工作，通常调用`getInstrumentation()`和`getActivity()`来获取当前测试的设备和待测应用启动的活动对象，并创建solo实例。
- `tearDown()`函数是在测试用例运行完成之后做的一些收尾工作，关闭所有打开的Activity。

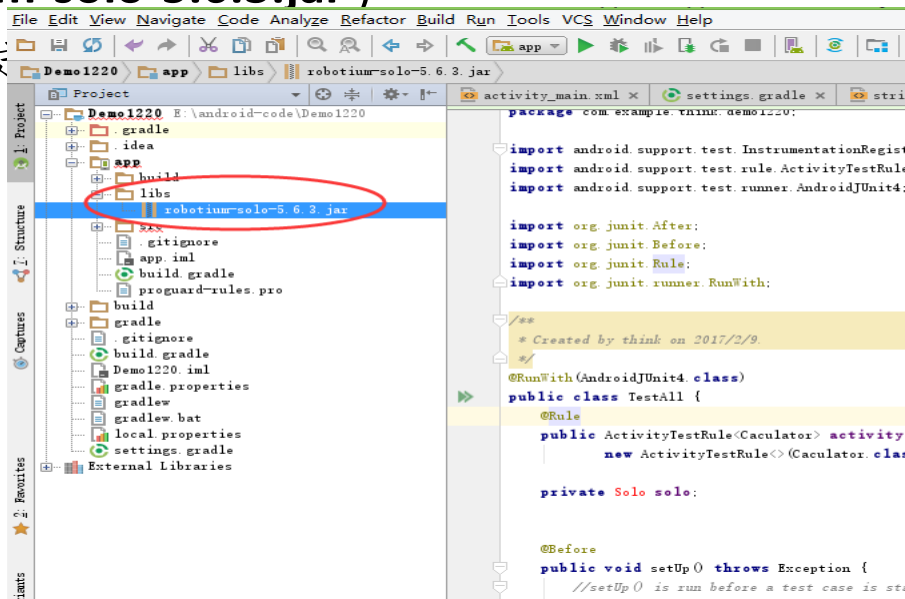
# robotium测试代码解析

```
public void testAddNote() throws Exception {  
    //解锁屏幕（非安全的锁，即滑动解锁）  
    solo.unlockScreen();  
    //点击id为menu_add的菜单  
    solo.clickOnView(solo.getView(R.id.menu_add));  
    //判断当前的Activity是NoteEditor  
    solo.assertCurrentActivity("Expected NoteEditor Activity", NoteEditor.class);  
    //在第一个输入框输入 Note 1  
    solo.enterText(0, NOTE_1);  
    //点击id为menu_save的菜单  
    solo.clickOnView(solo.getView(R.id.menu_save));  
    //点击id为menu_add的菜单  
    solo.clickOnView(solo.getView(R.id.menu_add));  
    //在第一个输入框输入 Note 2  
    solo.typeText(0, NOTE_2);  
    //点击id为menu_save的菜单  
    solo.clickOnView(solo.getView(R.id.menu_save));  
}
```



# robotium白盒自动化测试

- 以robotiumdemo为例
- 导入robotium-solo-5.6.3.jar，在 build.gradle 下 dependencies 添加 androidTestCompile 'com.jayway.android.robotium:robotium-solo:5.6.3' 或者 compile files('libs/robotium-solo-5.6.3.jar')
- 建议采用源码布局文件来定义 project 中



# robotium白盒自动化测试

- 使用ActivityTestRule  
获取入口的Activity

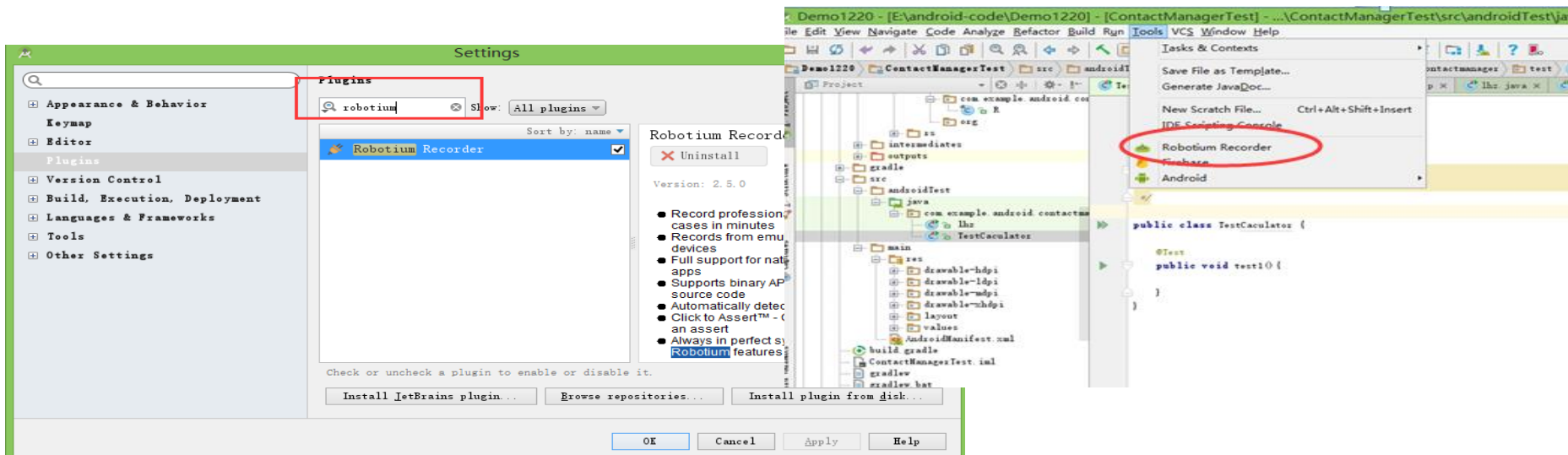
```
public class FindPrice {  
  
    @Rule  
    public ActivityTestRule<CheckoutActivity> activityTestRule =  
        new ActivityTestRule<>(CheckoutActivity.class);  
  
    private Solo solo;  
  
    @Before  
    public void setUp() throws Exception {  
        solo = new Solo(InstrumentationRegistry.getInstrumentation(),  
            activityTestRule.getActivity());  
    }  
  
    @After  
    public void tearDown() throws Exception {  
        solo.finishOpenedActivities();  
    }  
  
    @Test  
    public void testAddShop() {  
        solo.typeText((EditText) solo.getView(R.id.goods_code), "123456");  
        solo.clearEditText((EditText) solo.getView(R.id.goods_count));  
        solo.typeText(1, "2");  
        solo.clickOnButton("添加");  
        solo.sleep(15);  
    }  
}
```

# 本章大纲

- Android单元测试类
- Robotium是什么
- Robotium白盒自动化测试
- **Robotium录制回放工具**
- Robotium黑盒自动化测试
- Robotium常用API
- 测试脚本的批量运行

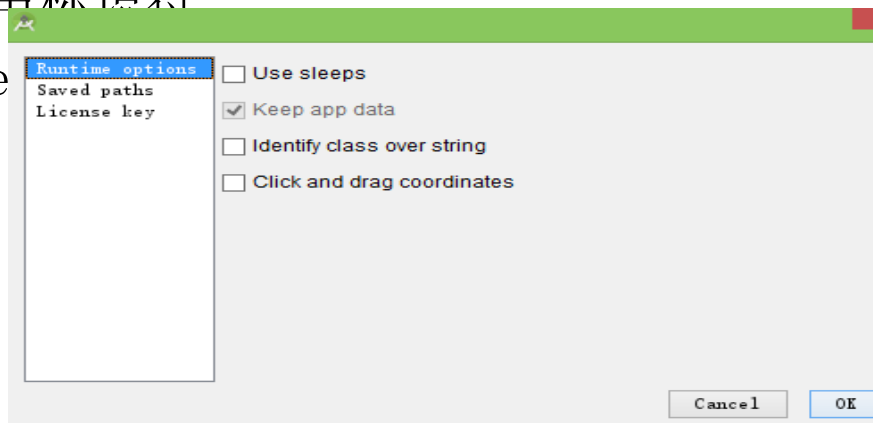
# robotium录制回放工具

- 下载插件File->Settings->Plugins, 搜索Robotium Recorder插件, 进行安装
- 更多: <http://robotium.com/pages/installation-android-studio>

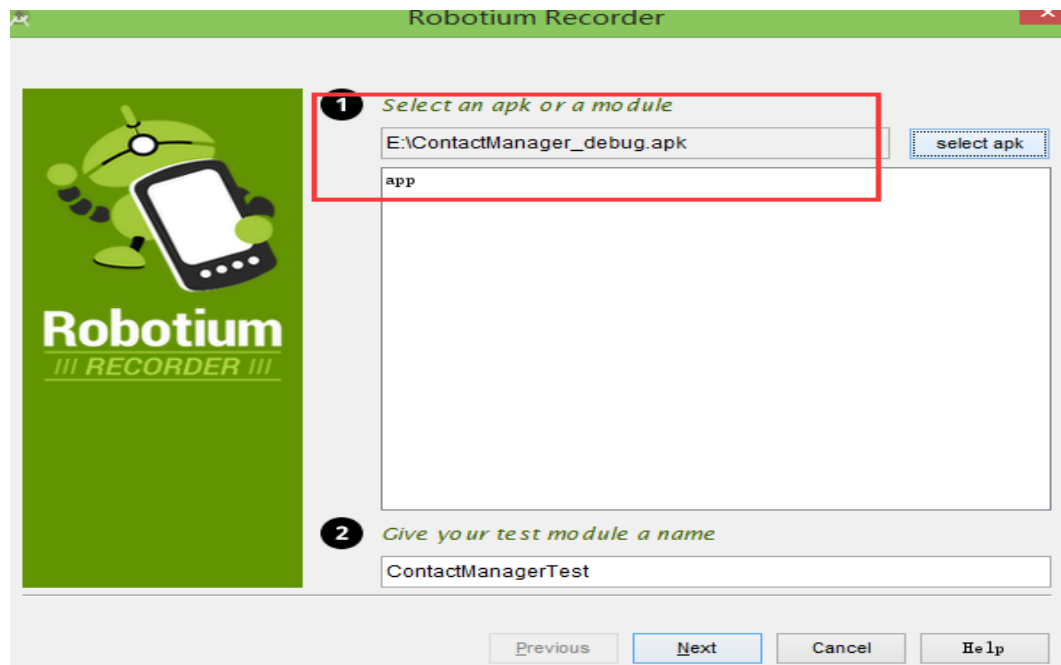


# robotium recorder-Settings

- 用户文档: <http://robotium.com/pages/user-guide-android-studio>
- Use sleeps : 如果想要测试用例在回放时也同样使用录制时的相同速度, 需要选择此项
- Keep app data: 是否保留应用程序的数据相关信息
- Identify class over string : 默认的视图标识符是资源ID, 如果资源ID丢失, 可以使用字符串标识符
- Click and drag coordinate 坐标的操作。

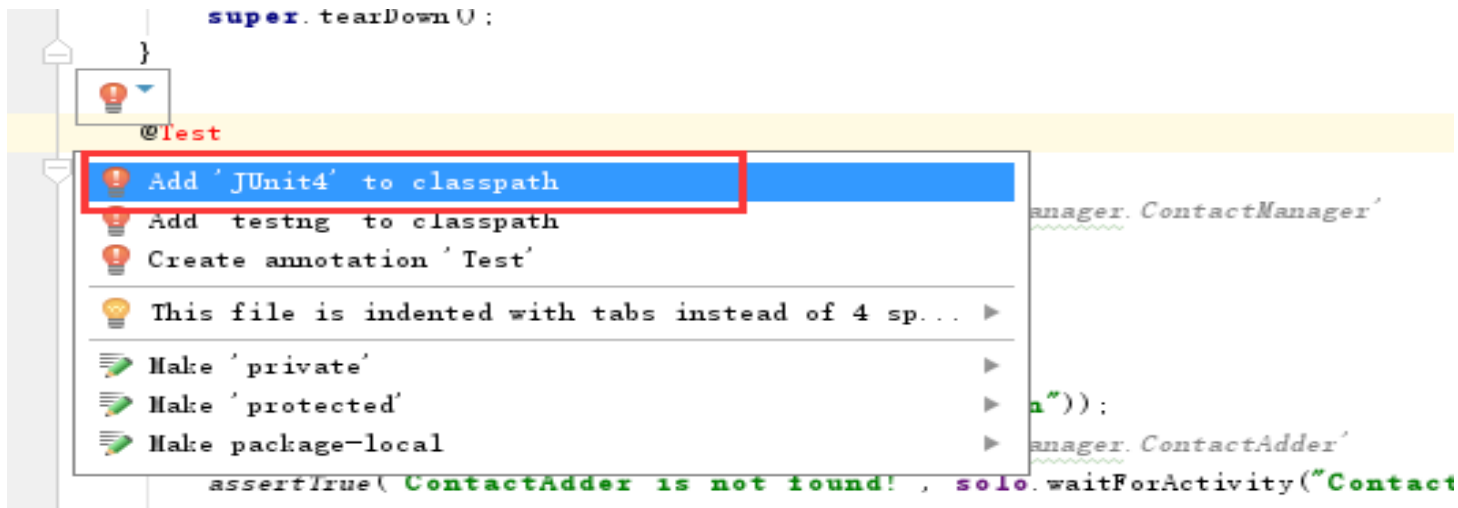


# robotium recorder 录制回放脚本

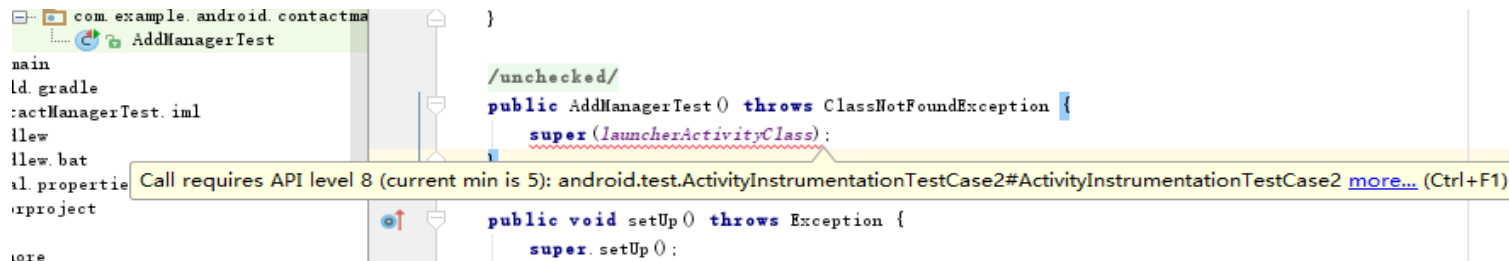


# 转换为JUnit4

- 在方法输入@Test，导入JUnit4的依赖



# 常见错误



解决办法:

```
android {
    compileSdkVersion 'android-19'
    buildToolsVersion '23.0.3'
```

```
    android {
        defaultConfig {
            minSdkVersion 11
        }

        lintOptions {
            abortOnError false
        }

        sourceSets
    }
```

修改为11即可



# 本章大纲

- Android单元测试类
- Robotium是什么
- Robotium白盒自动化测试
- Robotium录制回放工具
- **Robotium黑盒自动化测试**
- Robotium常用API
- 测试脚本的批量运行

# robotium录制回放工具

- APK包重新签名
  - re-sign.jar重签名工具: <http://recorder.robotium.com/downloads/re-sign.jar>;
  - 环境配置

配置ANDROID\_HOME为android sdk的目录, 例如: D:\android-sdk

在path下添加这两个: %ANDROID\_HOME%\tools;%ANDROID\_HOME%\platform-tools;

- 如何使用

双击re-sign.jar, 将要重签名的应用拖入



# robotium录制回放工具



- 解决办法：下载zipalign.exe置于SDK\tools\目录下

# robotium黑盒自动化测试

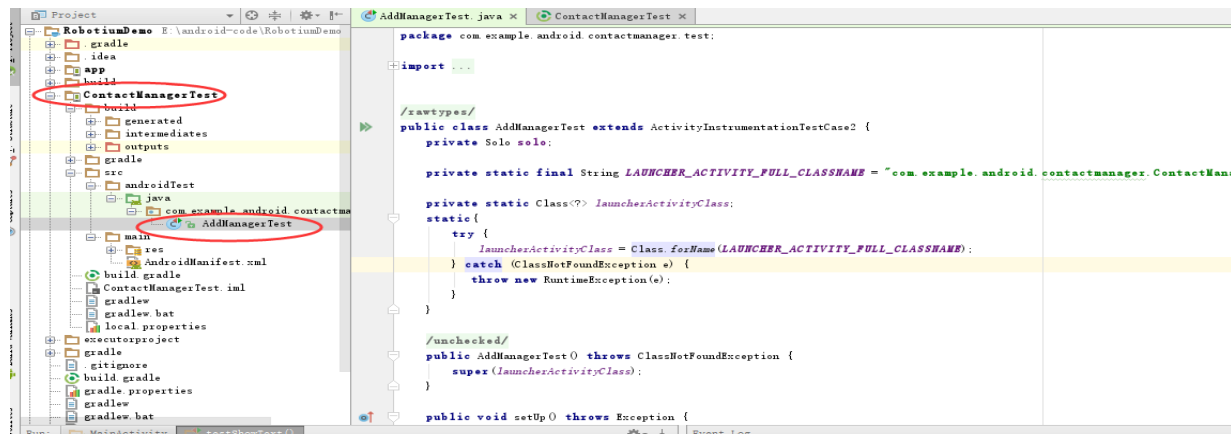
1.测试类需要继承ActivityInstrumentationTestCase2

2.使用反射机制获取入口Activity

```
public class AddManagerTest extends ActivityInstrumentationTestCase2 {  
    private Solo solo;  
  
    private static final String LAUNCHER_ACTIVITY_FULL_CLASSNAME = "com.example.android.contactmanager.ContactManager";  
  
    private static Class<?> launcherActivityClass;  
  
    static {  
        try {  
            launcherActivityClass = Class.forName(LAUNCHER_ACTIVITY_FULL_CLASSNAME);  
        } catch (ClassNotFoundException e) {  
            throw new RuntimeException(e);  
        }  
    }  
  
    public AddManagerTest() throws ClassNotFoundException {  
        super(launcherActivityClass);  
    }  
}
```

# robotium黑盒自动化测试

- 定位方式通过solo的getView来定位控件
- 如: `solo.enterText((android.widget.EditText)solo.getView("contactNameEditText"), "lhz");`
- **注意:** 查看getView源代码, 发现如果找不到这个控件就会断言失败, 终止case运行。避免这个问题的产生, 需要封装源代码
- 可以借助工具uiautomatorviewer, 目录sdk\tools



# 本章大纲

- Android单元测试类
- Robotium是什么
- Robotium白盒自动化测试
- Robotium黑盒自动化测试
- Robotium常用API
- 测试脚本的批量运行

# Robotium的控件获取、操作及断言

- <http://recorder.robotium.com/javadoc/>
- 完成对手机的模拟操作，应该包含以下几个基本操作：
  - (1) 需要知道所要操作控件的坐标。
  - (2) 对要操作的控件进行模拟操作。
  - (3) 判断操作完成后的结果是否符合预期。

根据被测应用的控件ID来获取

# 控件获取

返回值	方法及说明
View	getView(int ID)
View	getView(String ID)
Activity	getCurrentActivity()
TextView	getText(int index)
TextView	getText(String text)
Button	getButton(int index)
Button	getButton(String text)

```
solo.enterText((EditText) solo.getView(R.id.goods_code), "123456");  
solo.typeText((EditText) solo.getView("goods_count"), "4");
```



# 控件获取

- 处理id相同的控件

需要先获取节点控件的父视图，通过父视图再查找相应的子视图

```
RelativeLayout rel = (RelativeLayout) solo.getView("e1");  
ArrayList<TextView> textViews =  
solo.getCurrentViews(TextView.class, rel);  
solo.enterText(textViews.get(1),"hello");
```

# 控件获取

```
public void testAddNote() {  
    solo.clickOnMenuItem("Add Note");  
    solo.sleep(1000);  
    EditText additem = (EditText) solo.getCurrentActivity().getCurrentFocus();  
    additem.setText("hello");  
    solo.goBack();  
}
```

# 控件操作-点击、长按操作

返回值	方法及说明
void	clickOnText(String text)
void	clickOnButton(String text)
void	clickOnView(View view) clickLongOnView(View view)
void	clickOnScreen(float x, float y) clickLongOnScreen(float x, float y)

```
Button loginBtn = (Button) solo.getButton("btn_ok");  
solo.clickOnView(loginBtn);
```

# 控件操作-操作输入框

返回值	方法及说明
void	enterText(EditText editText, String text) enterText(int index, String text)
void	typeText(EditText editText, String text) typeText(int index, String text)
void	clearEditText(int index) clearEditText(EditText editText)

```
EditText et1= (EditText)  
solo.getView(R.id. goods_code) ;  
solo.enterText(et1, "123456");  
solo.typeText(et1, "123456");
```

 会展示输入的过程

# 控件操作-操作输入框

- typeText方法是robotium框架调用系统Instrumentation类里面的sendStringSync方法来实现的
- enterText是调用TextView里面setText方法来实现的
- 由于调用方法的不同，两个方法在测试过程中显示也是不一样的typeText有输入的痕迹，模拟按键输入，然而enterText直接显示文字

# 控件操作-滑动、滚动

在滑动方面，测试框架主要提供了两类支持

1. 根据坐标进行滑动从而可以模拟各类手势操作，
2. 则是根据控件来直接进行滚动操作

返回值	方法及说明
void	<b>drag(float fromX, float toX, float fromY, float toY, int stepCount)</b> 从起始点x,y滑至终点x,y坐标；通过stepCount参数指定滑动时的步长
void	<b>scrollToTop()</b> 滑动至顶部 <b>scrollToBottom()</b> 滑动至底部
void	<b>scrollUp()</b> 向上滑动屏幕 <b>scrollDown()</b> 向下滑动屏幕
void	<b>scrollListToLine(AbsListView absListView, int line)</b> 滑动列表至第line行

其中步长stepCount的意思是，假如要从A点滑到B点，如果步长为1，那么将直接产生从A点到B点的手势操作，滑动速度很快；如果步长为100，则将从A到B分成100等份，例如A、A1、A2...B，然后依次从A滑到A1，再从A1滑到A2、A2滑到A3.....这样滑动更慢但结果也更精确

# 控件操作-搜索与等待

返回值	方法及说明
void	<code>sleep(int time)</code> 休眠指定时间，单位毫秒
boolean	<code>searchText(String text)</code>
boolean	<code>searchButton(String text)</code>
boolean	<code>searchEditText(String text)</code>

# 控件操作-搜索与等待

返回值	方法及说明
boolean	<code>waitForView(int id)</code> <code>waitForText(String text)</code>
boolean	<code>waitForActivity(String name)</code>
boolean	<code>waitForLogMessage(String logMessage)</code>
boolean	<code>waitForDialogToOpen()</code> <code>waitForDialogToClose()</code>



# 控件操作-截图及其他

返回值	方法及说明
void	<code>takeScreenshot(String name)</code> 截图，图片名称为指定的name参数，图片默认路径为 /storage/ emulated/0/Robotium-Screenshots
void	<code>finishOpenedActivities()</code> 关闭当前已经打开的所有的Activity
void	<code>goBackToActivity(String name)</code> 不断地点击返回键直至返回到指定的Activity
void	<code>goBack()</code> 点击返回键
void	<code>hideSoftKeyboard()</code> 收起键盘
void	<code>setActivityOrientation(int orientation)</code> 等待设置Activity的转屏方向 Solo.LANDSCAPE 手机横向显示 Solo.PORTRAIT 手机纵向显示

# 控件操作-WebView支持

返回值	方法及说明
ArrayList<WebElement>	getCurrentWebElements() 获取当前Webview的所有WebElement元素
ArrayList<WebElement>	getWebElements(By by) 根据指定的元素属性获取当前的Webview的所有WebElement元素
void	clickOnWebElement(WebElement webElement) 点击WebElement
void	clickOnWebElement(By by) 通过By 根据指定的元素属性点击WebElement
void	enterTextInWebElement(By by, String text) 根据By找到指定的WebElement，并输入指定的文本text
boolean	waitForWebElement(By by) 等待根据by获得的WebElement出现

# 控件操作-WebView支持

- 在Robotium中对WebElement进行操作有两种方式，一种是先获取相应的WebElement，然后发送点击事件，另一种则是直接调用clickOnWebElement (By by) 进行点击。
- 例如：
- solo.clearTextInWebElement (By.id("login-username"))
- solo.clearTextInWebElement (By.id("login-password"))
- solo.enterTextInWebElement (By.id("login-username"),  
"1399811201@qq.com")

# Robotium中的断言

返回值	方法及说明
void	<code>assertCurrentActivity(String message, String name)</code> 断言当前的界面是否为指定的activity，若不是则抛出一个带有message提示的异常
void	<code>assertMemoryNotLow()</code> 断言当前是否处于低内存状态

# 本章大纲

- Android单元测试类
- Robotium是什么
- Robotium白盒自动化测试
- Robotium黑盒自动化测试
- Robotium常用API
- 测试脚本的批量运行

# 测试用例脚本的批量运行

- 需要根据不同测试的情况选择不同的测试策略，  
选择不同优先级别的测试用例来执行。

```
@RunWith(Suite.class)
```

```
// 指定运行器
```

```
@Suite.SuiteClasses({ FindPrice.class, FindPrice1.class })
```

```
public class MyTestSuite {
```

```
}
```