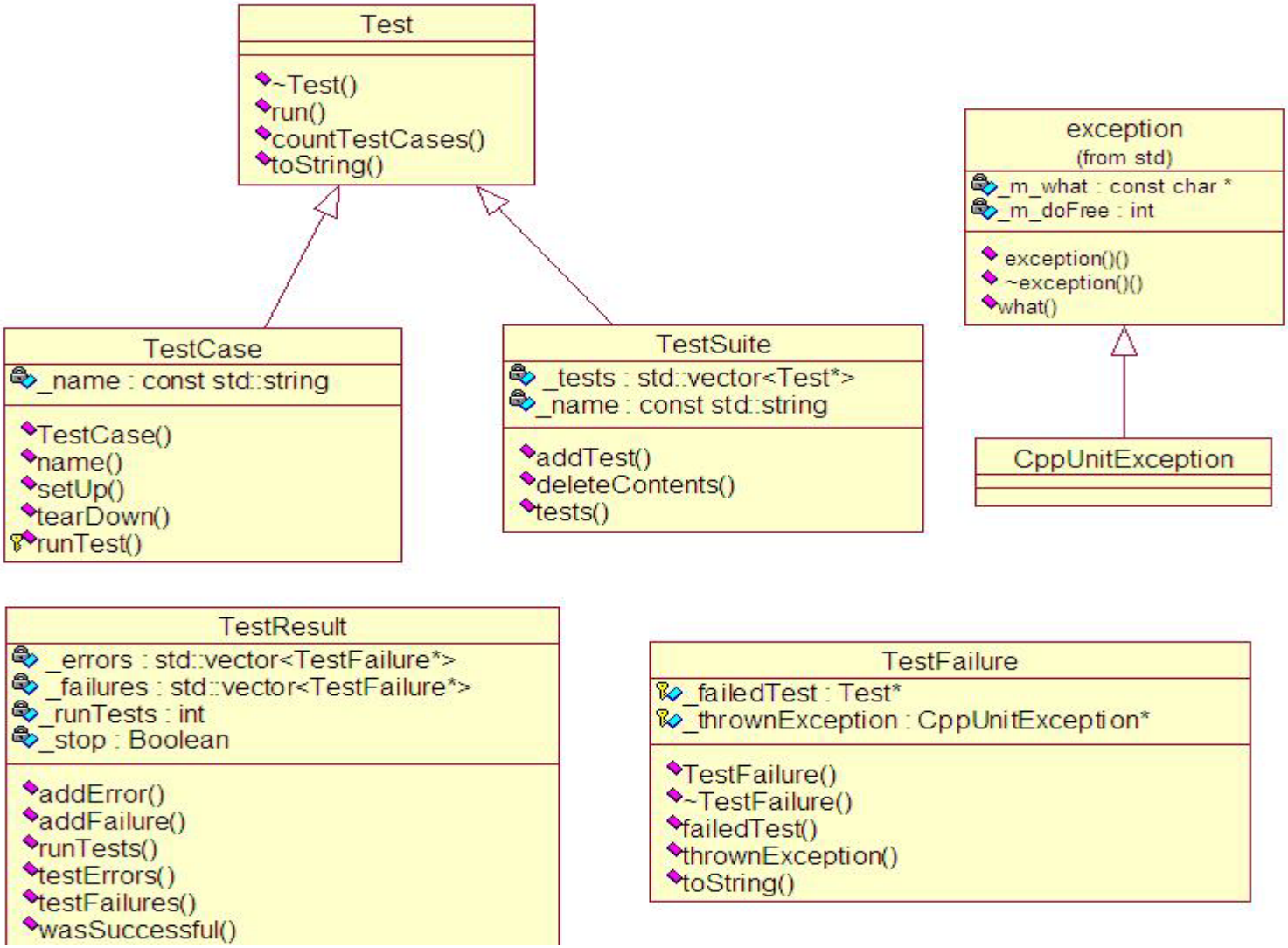
Android 与 JUnit

- Android、JUnit 深入浅出（一）——JUnit 初步解析
- Android、JUnit 深入浅出（二）——JUnit 例子分析
- Android、JUnit 深入浅出（三）——JUnit 深入解析（上）
- Android、JUnit 深入浅出（三）——JUnit 深入解析（下）荐
- Android、JUnit 深入浅出（四）——AndroidTestRunner荐
- Android、JUnit 深入浅出（五）——AndroidTest 例子分析
- Android、JUnit 深入浅出（六）——如何运行单元测试？
- android.app.instrumentation 解析
- android.test.InstrumentationTestRunner 解析荐
- An instrumention test runner is not specifiedNEW
- Android、JUnit 深入浅出（七）——总结篇NEW

Android、JUnit 深入浅出（一）——JUnit 初步解析

Android SDK 1.5 已经将 JUnit 包含进来了，但是一直没有去深入了解，以前在使用一些 C++的开源库中学习过与 CPPUnit，简要分析过其主要框架，如下：



这次在学习 Android SDK 1.6 中的例子程序 APIDemoes 中的过程中,看到了一个 test 文件夹，似乎就是使用了 JUnit，于是就开始学习 Android 中如何使用 JUnit。APIDemoes\test 文件夹下的测试代码相对比较复杂，我们会循序渐进的学习，逐步深入的理解，在后面为大家详细解析如何阅读这些测 试代码，本篇幅就初步解析 JUnit。

什么是 JUnit ？

JUnit 是采用测试驱动开发的方式，也就是说在开发前先写好测试代码，主要用来说明被测试的代码会被如何使用，错误处理等；然后开始写代码，并在测试代码中逐步测试这些代码，直到最后在测试代码中完全通过。

现简要说 JUnit 的 4 大功能

1. 管理测试用例。修改了哪些代码，这些代码的修改会对哪些部分有影响，通过 JUnit 将这次的修改做个完整测试。这也就 JUnit 中所谓的 TestSuite。
2. 定义测试代码。这也就是 JUnit 中所谓的 TestCase，根据源代码的测试需要定义每个 TestCase，并将 TestCase 添加到相应的 TestSuite 方便管理。
3. 定义测试环境。在 TestCase 测试前会先调用“环境”配置，在测试中使用，当然也可以在测试用例中直接定义测试环境。
4. 检测测试结果。对于每种正常、异常情况下的测试，运行结果是什么、结果是否是我们预期的等都需要有个明确的定义，JUnit 在这方面提供了强大的功能。

以上部分与我们平常使用 IDE 调试的过程是完全一样的，只不过是增加了测试用例管理、测试结果检测等功能，提高了单元的效率，保证了单元测试的完整性，明确了单元测试的目标。

以上 4 大功能，在 JUnit 的框架中是如何实现的了，在下一篇幅 JUnit 例子分析中，通过一个简要的例子，详细说明 4 大功能是如何实现的。

相关文章

- Android、JUnit 深入浅出（三）——JUnit 深入解析（上）
- Android、JUnit 深入浅出（二）——JUnit 例子分析
- Android、JUnit 深入浅出（七）——总结篇
- An instrumention test runner is not specified
- android.test.InstrumentationTestRunner 解析

Android、JUnit 深入浅出（二）——JUnit 例子分析

在前一篇文章 JUnit 初步解析中，我们简要了解了 JUnit 的主要功能：

1. 管理测试用例；
2. 定义测试代码；
3. 定义测试环境；
4. 检测测试结果；

结合主要功能，举个简单的例子分析如下：

源代码：

```
public class SampleCalculator
{
public int add(int augend , int addend)
{return augend + addend ;}

public int subtration(int minuend , int subtrahend)
```

```
{ return minuend  -  subtrahend ;}

}
```

测试用例（TestCase）：

```
import junit.framework.TestCase;
public class TestSample extends TestCase
{
   private int a;
   private int b;
   private int r1,r2;
   void setUp() /*开始测试当前用例－初始化测试环境*/
   {
      a = 50;
      b = 20;
      r1 = 70;
      r2 = 30;
   }

   void tearDown()/*当期用例测试结束*/
   {}

   public void testAdd()/*测试 SampleCalculator 类的 Add 函数*/
   {
      SampleCalculator calculator = new SampleCalculator();
      int result = calculator.add(a , b);
      assertEquals(r1 , result);/*检测测试结果*/
   }

   public void testSubtration()/*测试 SampleCalculator 类的 Subtration 函
数*/
   {
      SampleCalculator calculator = new SampleCalculator();
      int result = calculator.subtration(a , b);
      assertEquals(r2 , result);/*检测测试结果*/
   }
}
```

以上 TestSample 测试用例中就对 SampleCalculator 进行了完整的单元测试，并对测试结果做了预期说明。

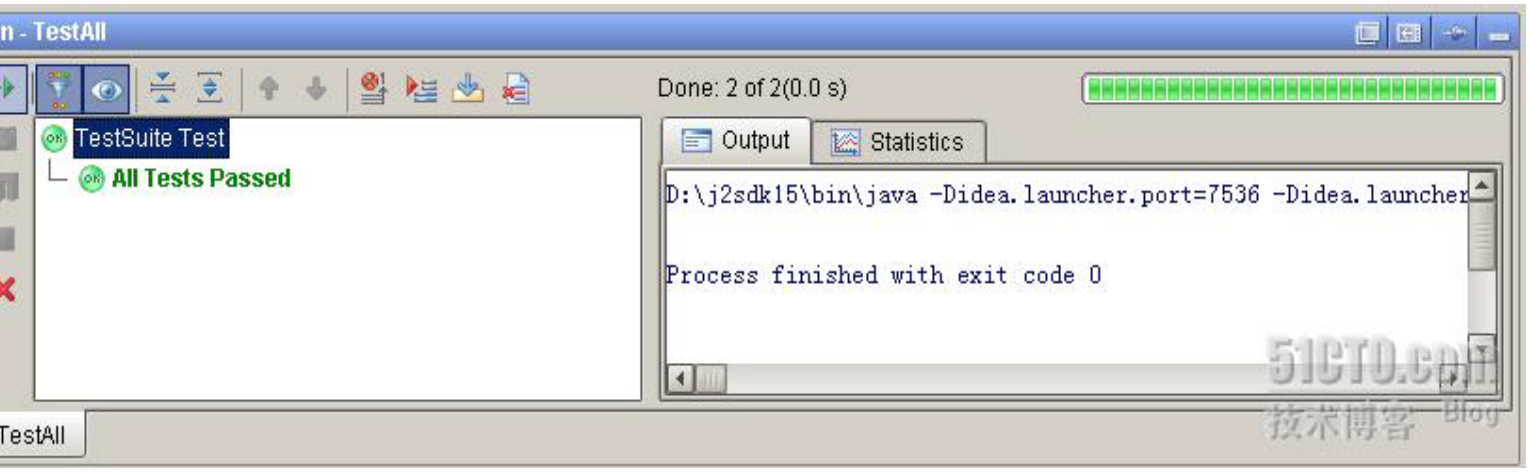
TestCase 的管理

```
import junit.framework.Test;
import junit.framework.TestSuite;
public class TestAll
{
```

```
    public static Test suite()
    {
        TestSuite suite = new TestSuite(" TestSuite Test");
        suite.addTestSuite(TestSample.class);
        return suite;
    }
}
```

以上就将 TestSample 增加到“TestSuite Test”中，将来在选择测试用例的过程中只要选择了 TestSuite Test，TestSample 就将加入当前测试中。如果将来 SampleCalculator 增加了其他功能，只需要在 TestSample 增加相应的 测试代码。

最后需要说明的：对 TestCase 的管理，是完全界面化的，JUnit 会自动产生 UI 界面，运行以上测试的例子，JUnit 的界面如下：



可能还需要下载 JUint packeage，最后送给大家一句话：大胆尝试下，你会发现编程真的可以如此“美好”。

相关文章

- Android、JUnit 深入浅出（三）——JUnit 深入解析（上）
- Android、JUnit 深入浅出（一）——JUnit 初步解析
- Android、JUnit 深入浅出（七）——总结篇
- An instrumention test runner is not specified
- android.test.InstrumentationTestRunner 解析

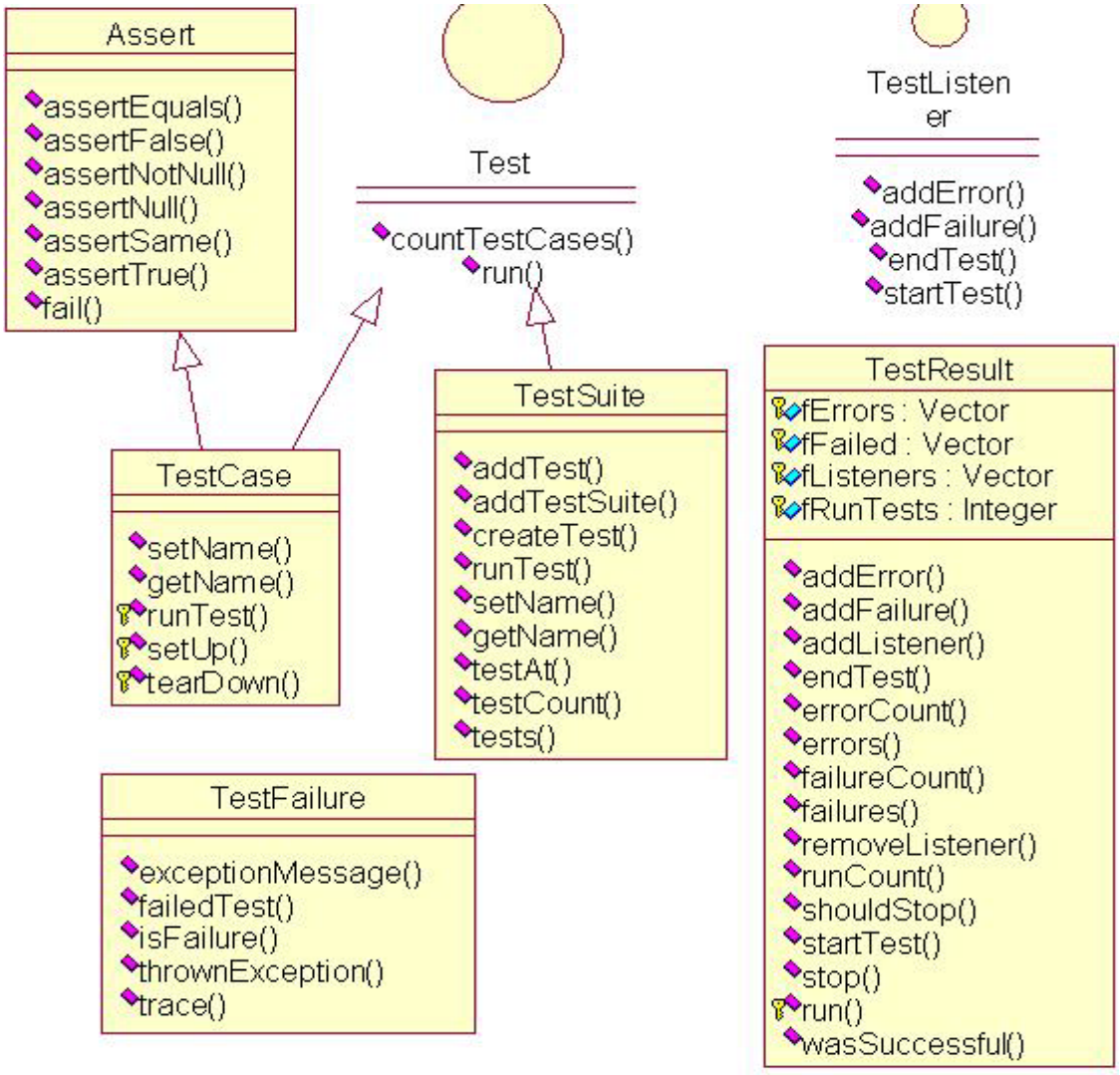
Android、JUnit 深入浅出（三）——JUnit 深入解析（上）

通过前面 2 篇文章的 学习，我们对 JUnit 有了初步的认识，下面我们将深入的解析 JUnit 数据包。整个 JUnit 的数据包应该是很强大的，但是一般来说，不一定每个工程都 需要这些数据包，而是在 JUnit 部分数据包的基础上扩展出自己的数据包，Android SDK 中也不例外。至于 JUnit 完整的包，这里我们就不详细分析了，我们这里只解析 Android SDK 中包含的那些 JUnit 数据包，以及 Android SDK 在 JUnit 的基础上扩展的一些数据包，如下：

SDK	功能说明
junit.framework	JUnit 测试框架
junit.runner	实用工具类支持 JUnit 测试框架
android.test	Android 对 JUnit 测试框架的扩展包
android.test.mock	Android 的一些辅助类
android.test.suitebuilder	实用工具类，支持类的测试运行

在这些包中最为重要的是：junit.framework、android.test，其中前者是 JUnit 的核心包，后者是 Andoid SDK 在 Junit.framework 的基础上扩展出来的包，我们将重点解析这 2 个包。

首先解析 junit.framework 包，结构如下：



通过这张图，大家就可以比较清晰的看到 JUnit 的主要框架，再回去看下上篇文章的例子，对前面的例子感觉明白多了。做个简要的总结，如下：

1. TestSuit：TestSuite 是测试用例的集合；
2. TestCase：定义运行多个测试用例；
3. TestResult：收集一个测试案例的结果，测试结果分为失败和错误，如果未能预计的断言就是失败，错误就像一个 ArrayIndexOutOfBoundsException 异常而导致的无法预料的问题；
4. TestFailure：测试失败时捕获的异常；
5. Assert：断言的方法集，当断言失败时显示信息；

TestCase 与 TestSuite 之间的关系，有些类似于图元对象与容器对象之间的关系，在面向对象的语言 C++、JAVA 中较常见，在这里就不多说了。

举个简单的例子，并简要说明过程

第一步：实现 TestCase

1. 继承父类 TestCase；
2. 定义一下变量在测试中使用；
3. 在 setUp()中初始化这些变量；
4. 在 tearDown()中清理这些变量；

```
public class MathTest extends TestCase{
    protected double fValue1;
    protected double fValue2;
    protected void setUp(){
        fValue1= 2.0;
        fValue2= 3.0;
    }
}
```

5. 编写测试单元代码；

```
public void testAdd() {
    double result= fValue1 + fValue2;
    assertTrue(result == 5.0);
}
```

6. 运行测试用例，这里有 2 种方法可以使用：

- 静态类型：覆盖 runTes()和定义测试函数。最常用的就是采用 java 的匿名类，如下：

```
TestCase test= new MathTest("add"){
    public void runTest() { testAdd();}
};
test.run();
```

- 动态类型：使用反射来实现 runTest，它动态地发现并调用的方法，在这种情况下，测试案例的名字对应的测试方法来运行，如下：TestCase= new MathTest("testAdd");

```
test.run();
```

相比之下，第 2 种更符合面向对象的思维。

第二步：将 TestCase 添加到 TestSuilt

```
TestSuite suite= new TestSuite();
suite.addTest(new MathTest("testAdd"));
```

由于 TestSuite 可以自动从 TestCase 中提取测试单元并运行，也可以用如下方法：

```
TestSuite suite= new TestSuite(MathTest.class);
```

一个测试用例就完成了,想要更加详细的了解 junit.framework,还是到 Android SDK 中仔细阅读。

总结说明

看了这些代码，再仔细看下 JUnit 的结构图，是不是感觉更加清晰了，下一篇幅我们将深入解析 android.test 包。

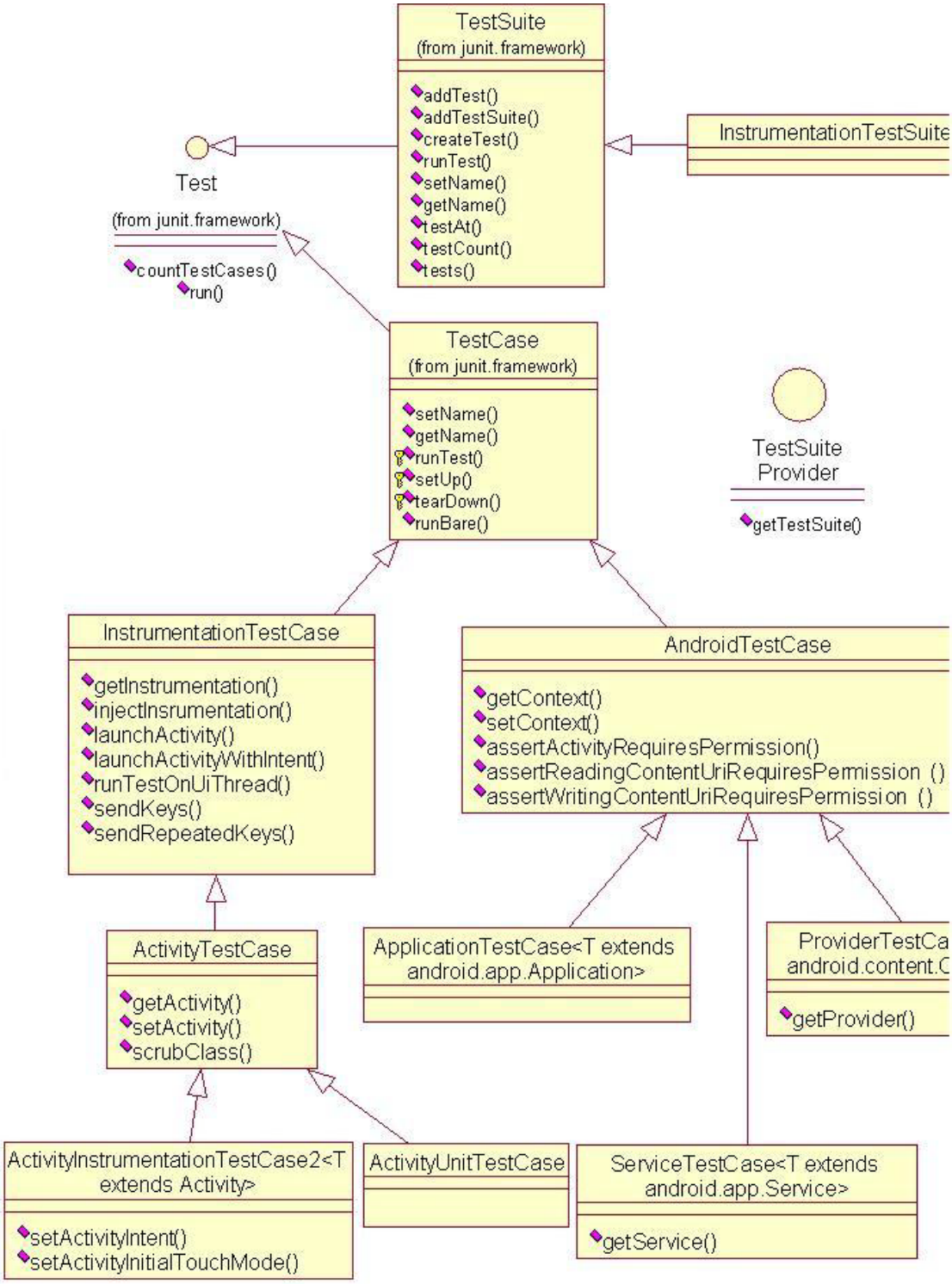
相关文章

- Android、JUnit 深入浅出（二）——JUnit 例子分析
- Android、JUnit 深入浅出（一）——JUnit 初步解析
- Android、JUnit 深入浅出（七）——总结篇
- An instrumention test runner is not specified
- android.test.InstrumentationTestRunner 解析

Android、JUnit 深入浅出（三）——JUnit 深入解析（下）

前面我们学习了 junit.framework 包，本篇幅我们开始学习 android.test 包，了解 Android SDK 是如何扩展 junit.framework 包。

首先整理 android.test 包的结构，如下图所示：



在前面的篇幅中说明过：TestCase 与 TestSuite 之间的关系，有些类似于图元对象与容器对象之间的关系。有 了这个思想，顺着这张的主线图从上往下看：首先看到的是 TestCase 类，接着看到 InstrumentationTestCase、 AndroidTestCase，最后看到 ApplicationTestCase，ProviderTestCase2，ServiceTestCase，ActivityTestCase。这 不就是 Android 系统中 4 大组成元素：Activity、Provider、Service、BroadCast 中的前 3 个，然后我们在看下 Android SDK 中的说明：

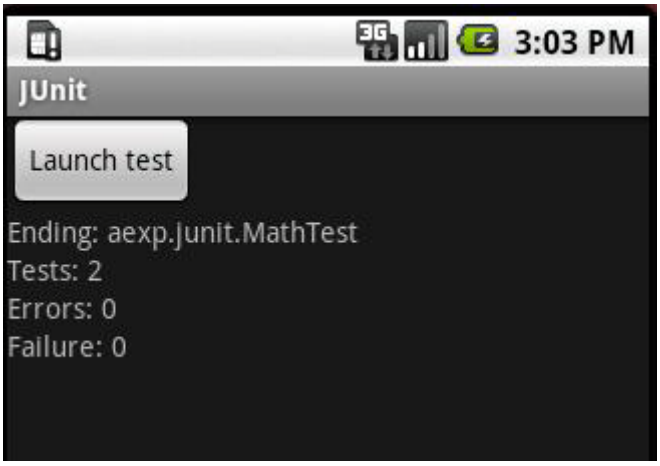
类	说明
AndroidTestCase	如果你要访问资源或其他东西依赖于 Activity 的环境，在这个类的基础上扩展。
ActivityInstrumentationTestCase2 <T extends Activity>	这个类提供了一个单一的活动功能测试
ApplicationTestCase <T extends Application>	提供了一个框架，可以在受控环境中测试 Application 类
ProviderTestCase2 <T extends ContentProvider>	提供了一个框架，可以在受控环境中测试 ContentProvider 类
ServiceTestCase <T extends Service>	提供了一个框架，可以在受控环境中测试 ServiceTest 类。

这些类就不在这里说明，应该与我们平时使用 Activity、Service、Provider 基本一样。下面举个例子，来说明如何使用这些类，我们就将上一篇 J2SE 测试例子，在 Android 中实现：

```
//MathTest.java
import android.test.AndroidTestCase;
import android.util.Log;
public class MathTest extends AndroidTestCase
{
   protected double fValue1;
   protected double fValue2;
   protected double fRe;
   static final String LOG_TAG = “MathTest”;
   protected void setUp() {
      fValue1= 2.0;
      fValue2= 3.0;
      fRe = 5.0;
   }

   public void testAdd()
   {
      Log.d( LOG_TAG, “testAdd” );
      assertTrue( LOG_TAG+”1”, ( (fValue1 + fValue2 ) == fRe ) );
   }
}

//ExampleSuite.java
import junit.framework.TestSuite;
public class ExampleSuite extends TestSuite
{
   public ExampleSuite()
   {
```

```
        addTestSuite( MathTest.class );
    }
}
```

与上一篇的例子比较后发现，除了引用的包、父类不一样以外，其他部分是完全一样的，在 Android AVD 上运行程序后看到的结果如下：



单击按钮“Launch test”运行测试用例，输出测试结果，我们的测试都通过了。这个测试界面是我们自己编写的单元测试的界面，下一篇幅我们将重点介绍这部分。

总结说明

这里我们简要学习了 android.test 包，其实这个的内容远不止这些，包含：3 个 Interface、20 个 class、2 个 Error，列举如下：

```
Interfaces（3个）
   PerformanceTestCase
   PerformanceTestCase.Intermediates
   TestSuiteProvider

 Classes（20个）
   InstrumentationTestCase
   AndroidTestCase

   //Application（Activity、Provider、Service）
   ActivityTestCase
   ActivityInstrumentationTestCase<T extends Activity>
   ActivityInstrumentationTestCase2<T extends Activity>
   ActivityUnitTestCase<T extends Activity>
   SingleLaunchActivityTestCase<T extends Activity>
   ApplicationTestCase<T extends Application>
   ProviderTestCase<T extends ContentProvider>
   ProviderTestCase2<T extends ContentProvider>
   ServiceTestCase<T extends Service>

   MoreAsserts
   ViewAsserts

   AndroidTestRunner
   SyncBaseInstrumentation
   TouchUtils
   InstrumentationTestRunner
   InstrumentationTestSuite
   IsolatedContext
   RenamingDelegatingContext

Errors（2个）
  AssertionFailedError
  ComparisonFailure
```

android.test 包深入的学习，只有到 Android SDK 中去仔细阅读了。通过学习 android.test，让我们对 Android 系统的组成元素：Activity、Provider、Service 有了更加深入的了解。

相关文章

- Android、JUnit 深入浅出（七）——总结篇
- An instrumention test runner is not specified
- Android、JUnit 深入浅出（六）——如何运行单元测试？
- Android、JUnit 深入浅出（五）——AndroidTest 例子分析
- Android、JUnit 深入浅出（四）——AndroidTestRunner

Android、JUnit 深入浅出（四）——AndroidTestRunner

随着学习的深入，发现包在前面的篇幅中，我们忽略了 android.test 包中一个重要的类 AndroidTestRunner，这个类是 android.test 包的核心类，下面为大家详细说明，并补充说明一些相关的内容。

junit.framework 包中的 TestListener 接口

这个接口的函数，列举如下：

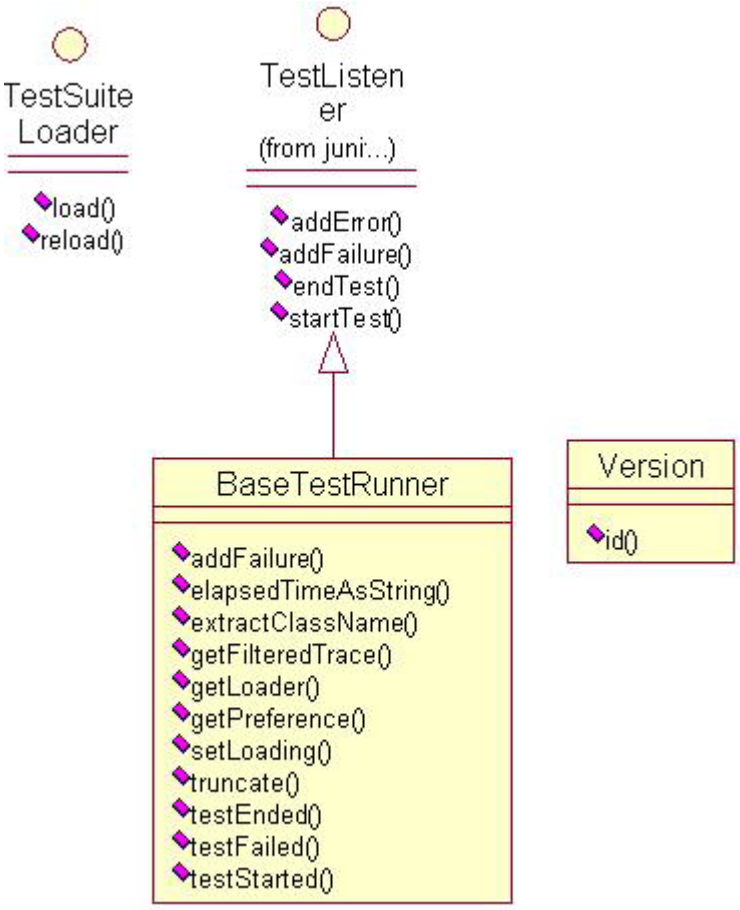
Public Methods	
String	exceptionMessage ()
Test	failedTest () Gets the failed test.
boolean	isFailure ()
Throwable	thrownException () Gets the thrown exception.
String	toString () Returns a short description of the failure
String	trace ()

与这个接口，相关的类就只用 TestResult，相关接口如下：

synchronized void	addListener (TestListener listener) Registers a TestListener
synchronized void	removeListener (TestListener listener) Unregisters a TestListener

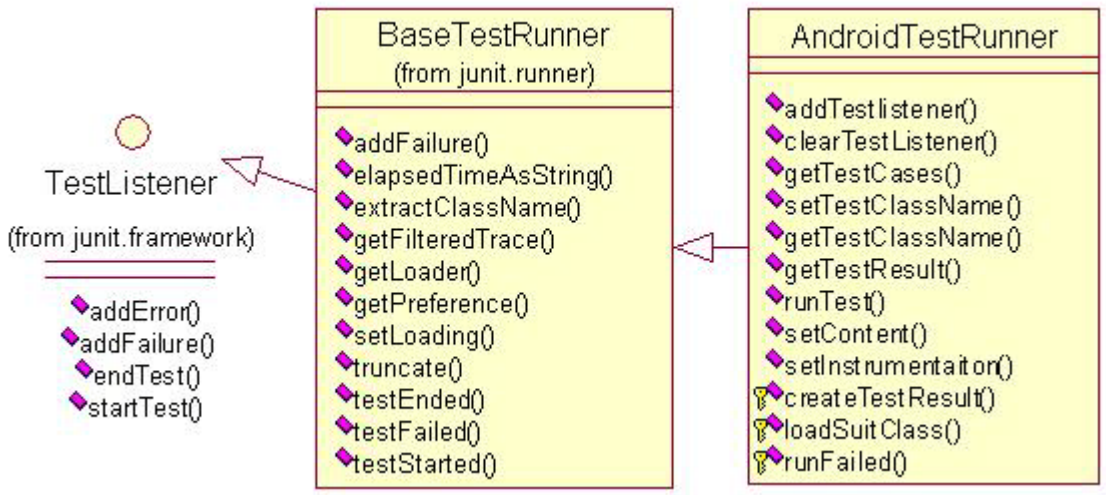
看到这里就应该知道如何使用了，具体的使用在下一篇幅例子中说明。

junit.runner 包，结构如下：



这是一个对 junit.framework 的辅助包，包主要就是 BaseTestRunner 类，其实现了 TestListener 接口，主要功能是：对测试过程中 Error、Failure 的检查。
有了这些补充说明，下面学习 android.test 包中一个重要的类 AndroidTestRunner。

AndroidTestRunner 类结构，如下图所示：



其主要接口函数，列举如下：

Public Methods	
void	addTestListener (TestListener testListener)
void	clearTestListeners ()
List<TestCase>	getTestCases ()
String	getTestClassName ()
TestResult	getTestResult ()
void	runTest (TestResult testResult)
void	runTest ()
void	setContext (Context context)
void	setInstrumentaiton (Instrumentation instrumentation)

看到 setContext(Context context)这个函数的这个参数 Context context，总算让我看到 junit 与 Android 的结合点了，在看下其他几个函数，我们会发现，这个类是 android.test 的核心控制类， 大家心中的疑惑顿时就没有了。列举一个简要的例子，如下：

```
AndroidTestRunner testRunner = new AndroidTestRunner();
testRunner.setTest( new ExampleSuite() );
testRunner.addTestListener( this );
testRunner.setContext( parentActivity );
testRunner.runTest();
```

通过 AndroidTestRunner 控制整个测试，并与我们的 Activity 向结合，具体的使用在下一篇幅中详细说明。

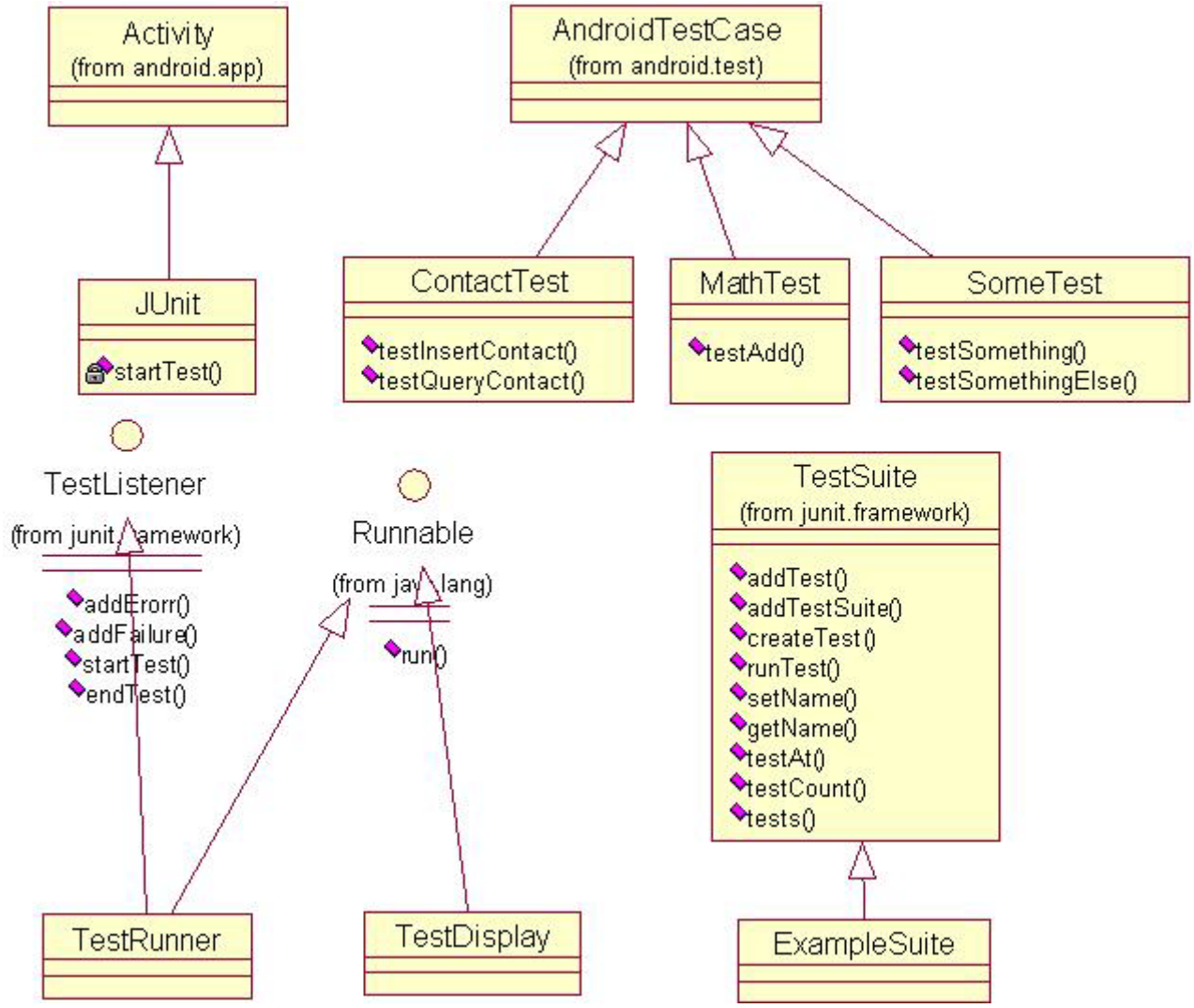
相关文章

- Android、JUnit 深入浅出（七）——总结篇
- An instrumention test runner is not specified
- Android、JUnit 深入浅出（六）——如何运行单元测试？
- Android、JUnit 深入浅出（五）——AndroidTest 例子分析
- Android、JUnit 深入浅出（三）——JUnit 深入解析（下）

Android、JUnit 深入浅出（五）——AndroidTest 例子分析

前面我们学习了 android.test 包中的大部分类，是该通过学习具体的例子将前面的知识融会贯通，让我们的理解更加深刻，例子程序代码下载地址，下载后添加 Eclipes 的工程中，边看这篇文章边阅读例子程序的代码。

首先分析整个工程的结构图，如下：



AndroidTestCase，Testsuite 在前面的篇幅中已经学习过了，ContestTest、MathTest、SomeTest、ExampleSuite 在前面的例子中已经为大家介绍了，这里我们主要说明整个程序是如何运行的？

核心类代码简要列举，如下：

```
public class JUnit extends Activity {
    static final String LOG_TAG = “junit”;
    Thread testRunnerThread = null;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button launcherButton = (Button)findViewById( R.id.launch_button );
        launcherButton.setOnClickListener( new View.OnClickListener() {
        public void onClick( View view ) {startTest();}
        } );
    }
    private synchronized void startTest()
    {
        if( ( testRunnerThread != null ) &&!testRunnerThread.isAlive() )
        testRunnerThread = null;
        if( testRunnerThread == null )
        {
            testRunnerThread = new Thread( new TestRunner( this ) );
            testRunnerThread.start();
        }
        else
        {
            Toast.makeText(this, “Test is still running”,
 Toast.LENGTH_SHORT).show();
        }

class TestRunner implements Runnable,TestListener
{
   static final String LOG_TAG = “TestRunner”;
   int testCounter;
   int errorCounter;
   int failureCounter;
   ……;
   Activity parentActivity;

   public TestRunner( Activity parentActivity )
   {this.parentActivity = parentActivity;}

   public void run()
   {
      testCounter = 0;
      errorCounter = 0;
      failureCounter = 0;
      ……….;

      Log.d( LOG_TAG, “Test started” );
      /*整个代码的核心*/
      AndroidTestRunner testRunner = new AndroidTestRunner();
      testRunner.setTest( new ExampleSuite() );
```

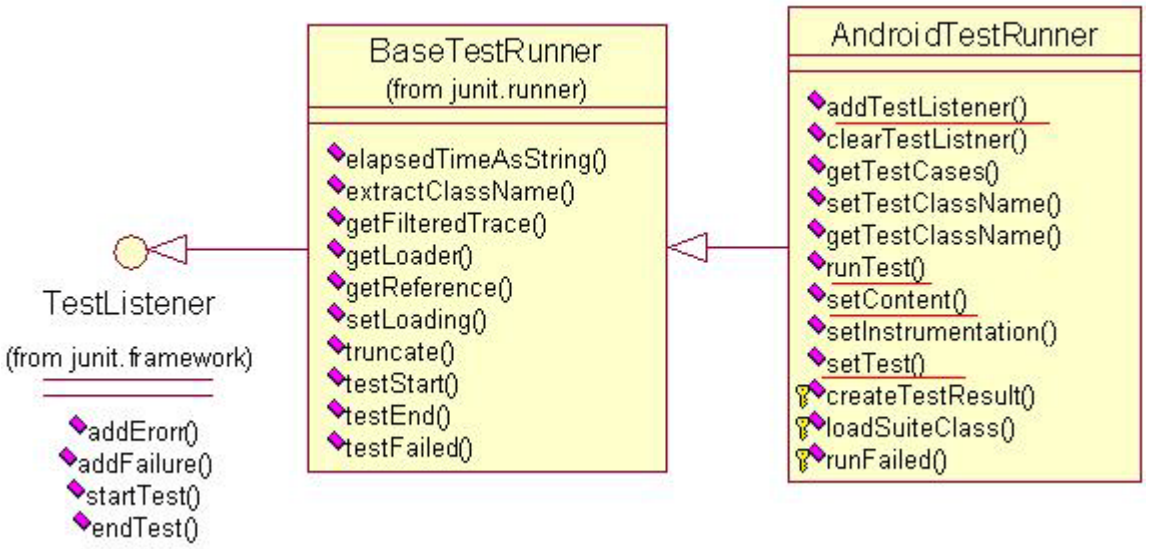
```
        testRunner.addTestListener( this );
        testRunner.setContext( parentActivity );
        testRunner.runTest();
        Log.d( LOG_TAG, “Test ended” );
    }

    // TestListener
    public void addError(Test test, Throwable t)
    {
        Log.d( LOG_TAG, “addError: “+test.getClass().getName() );
        Log.d( LOG_TAG, t.getMessage(), t );
        ++errorCounter;
        ……;
    }

    public void addFailure(Test test, AssertionFailedError t)
    {
        Log.d( LOG_TAG, “addFailure: “+test.getClass().getName() );
        Log.d( LOG_TAG, t.getMessage(), t );
        ++failureCounter;
        ……;
    }

    public void endTest(Test test)
    {
        Log.d( LOG_TAG, “endTest: “+test.getClass().getName() );
        …..;
    }

    public void startTest(Test test)
    {
        Log.d( LOG_TAG, “startTest: “+test.getClass().getName() );
        ++testCounter;
        ……;
    }

}
```

通过将源工程中的代码简单整理后，就可以看到 TestRunner 这个工作者线程（window 中的术语，没有界面的线程）的作用，这让我们对 TestListener 有了更加深入的了解。

整个程序的核心代码，如下：

```
public void run()
{
    ……;
    /*整个代码的核心*/
    AndroidTestRunner testRunner = new AndroidTestRunner();
    testRunner.setTest( new ExampleSuite() );
    testRunner.addTestListener( this );
    testRunner.setContext( parentActivity );
    testRunner.runTest();
    …….;
}
```

AndroidTestRunner 这个核心类，在前面的篇幅中我们已经学习过，再次回忆下这张图（在大脑中留下深刻的记忆，后面会经常使用）：

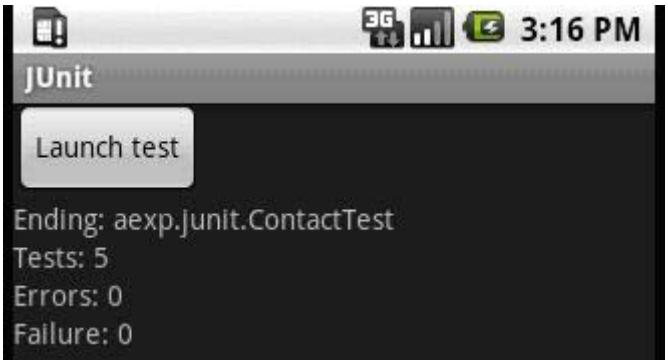


红色划线部分代表例子程序代码中使用的 AndroidTestRunner 类的函数。这里使用单独线程的主要作用就 是：testRunner.runTest();会占用大量的时间，如果直接在 UI 线程中运行会阻滞 UI 线程，导致界面停止反应，这对用户的操作会有很大 的影响。

如何将 TestRunner 中的测试信息显示在界面上？

在前面的 Snake 例子程序中介绍过：Android SDK 为我们提供了 Handler， 通过 Handler 与一个线程的消息队列相关联，发送和处理信息。在这个例子中使用了 Activity 类的 runOnUiThread (Runnable action)函数，这个函数的主要功能：在 UI 线程中运行指定的操作，如果当前线程是 UI 线程，然后采取行动立即执行；如果当前线程不是 UI 线程，发送 消息到 UI 线程的事件队列。

整个程序就介绍完了，运行程序后的界面如下：



在这里需要特殊说明的是：打开 AndroidManifest.xml 文件，发现<application>有个以前没有见过的标记，如下：

```
<application android:label=”@string/app_name”>
   <uses-library android:name=”android.test.runner” />
   <activity android:name=”.JUnit” android:label=”@string/app_name”>
      <intent-filter>
         <action android:name=”android.intent.action.MAIN” />
         <category android:name=”android.intent.category.LAUNCHER” />
      </intent-filter>
   </activity>
</application>
```

user-library 元素说明：指定一个共享库，应用程序需要连接的。默认情况下会连接所有的 Android 库，然而一些软件包（如地图和 AWT）是不会自动连接独立的库，以确定哪些库需要包含这些特定的包代码文件。

总结说明

这个例子已经学习完了，虽然它比较简单，但是让我们清晰的了解如何使用 AndroidTestRunner，后面我们将继续介绍一些复杂的例子，更加深入的学习。

相关文章

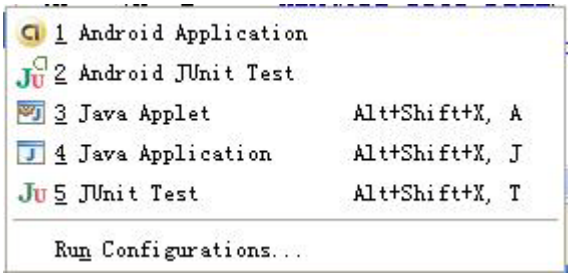
- Android、JUnit 深入浅出（七）——总结篇
- An instrumention test runner is not specified
- Android、JUnit 深入浅出（六）——如何运行单元测试？
- Android、JUnit 深入浅出（四）——AndroidTestRunner
- Android、JUnit 深入浅出（三）——JUnit 深入解析（下）

Android、JUnit 深入浅出（六）——如何运行单元测试？

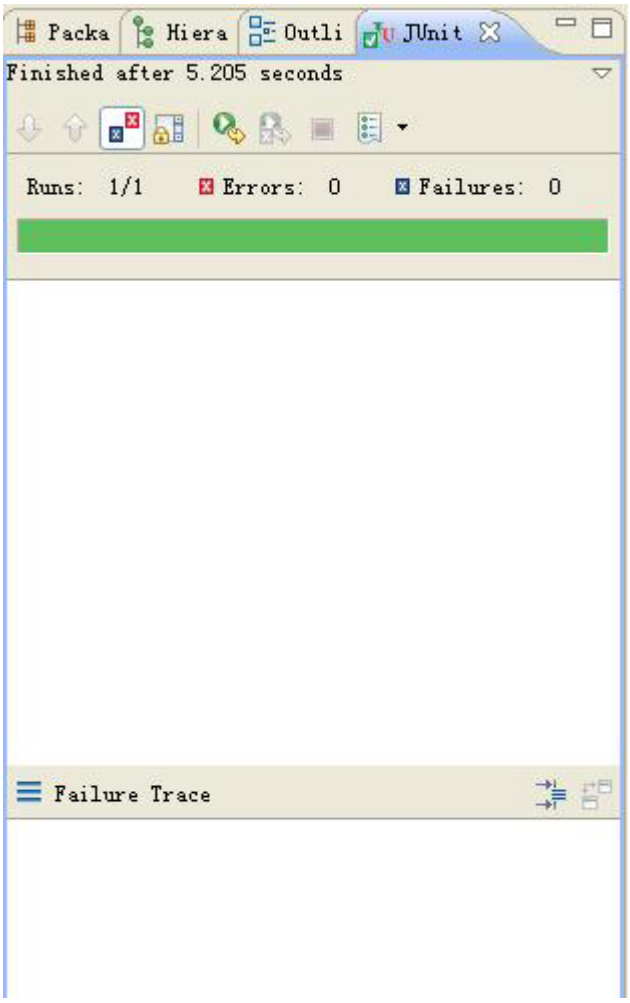
前面我们学习了很多 JUnit 的程序，在 Android ADT 插件中已经为我们提供了很多这方面的功能，方便我们进行单元测试。但是如何进行单元测试，例如在 ApiDemos\test 提供的测试例子程序如何 运行,对于我们这些初学者来说有些茫然，我也是在网上查找了不少这方面的的资料学习，才知道如何运行测试单元，因此在这里总结说明与大家分享。总结起来， 大概有 4 种不同的方法：

使用 ADT 运行测试单元

在 Eclipes 中选择工程，单击右键，在 Run as/Debug as 子菜单选项中选择 Android JUnit Test，如下：



单击运行后，应用程序将启动，在 Eclipes 中会出现个新的面板 JUnit，如下：

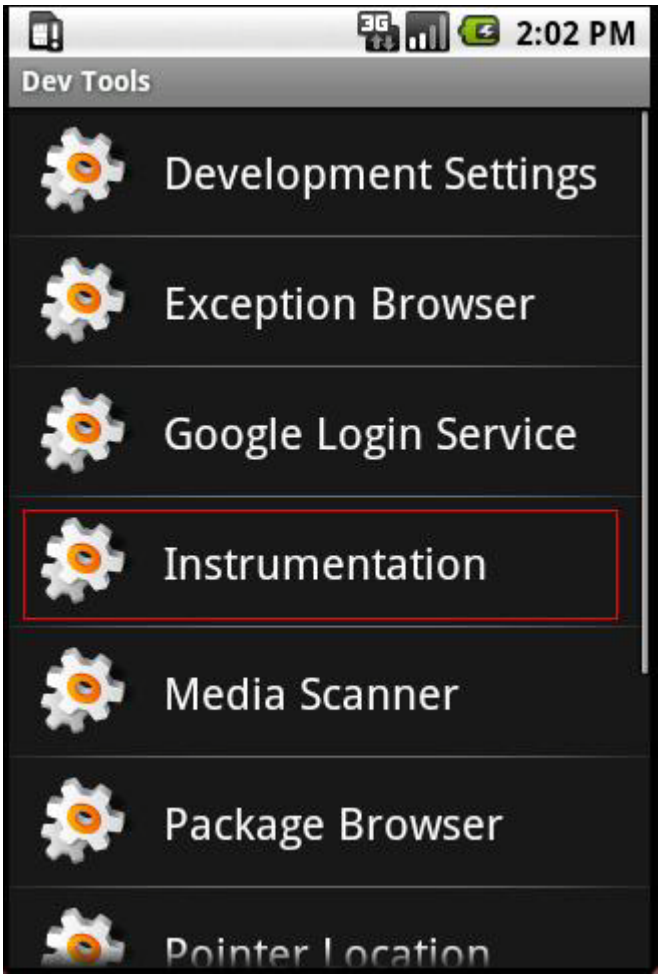


这个界面上就显示了测试的结果，这种方法操作比较简单，但是要想自己写单元测试就的必须深入的去了解后面 2 中运行的方法。

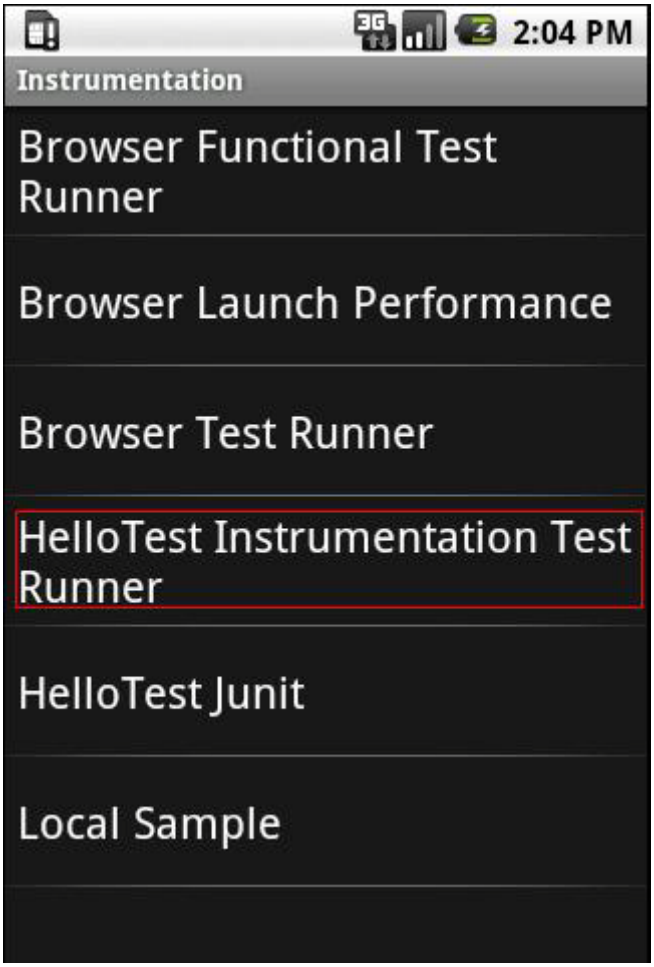
通过 AVD 运行测试单元

运行 AVD，选择 Dev Tool 

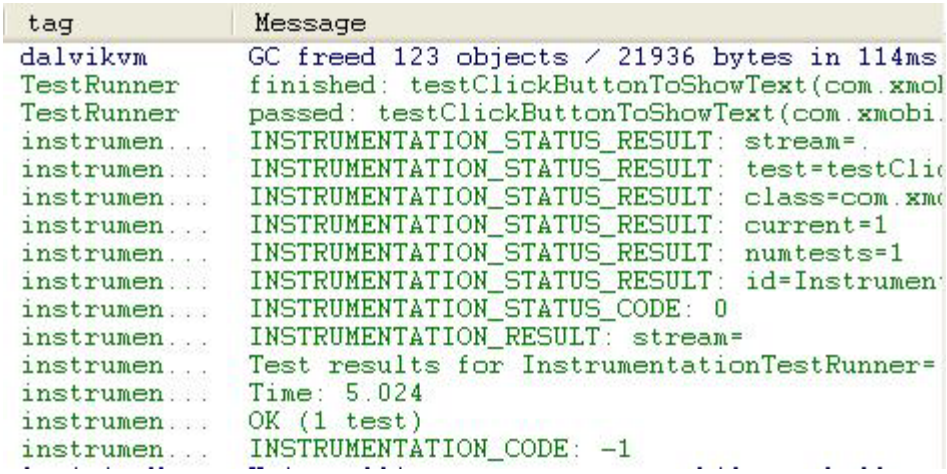
，当前界面如下：



双击 Instrumentation 后，界面如下：



测试例子开始运行，在 LogCat 中查看运行过程输出的信息，界面如下：



这些信息就是测试例子输出的信息。

通过 adb shell 命令运行测试单元

这种方法应该是为 linux 程序员设置的，完全的命令行，使用起来相对比较麻烦，命令格式如下：

```
usage: am [start|broadcast|instrument|profile]
       am start -D INTENT
       am broadcast INTENT
       am instrument [-r] [-e <ARG_NAME> <ARG_VALUE>] [-p <PROF_FILE>]
                [-w] <COMPONENT>
       am profile <PROCESS> [start <PROF_FILE>|stop]

       INTENT is described with:
                [-a <ACTION>] [-d <DATA_URI>] [-t <MIME_TYPE>]
                [-c <CATEGORY> [-c <CATEGORY>] ...]
                [-e|--es <EXTRA_KEY> <EXTRA_STRING_VALUE> ...]
                [--ez <EXTRA_KEY> <EXTRA_BOOLEAN_VALUE> ...]
                [-e|--ei <EXTRA_KEY> <EXTRA_INT_VALUE> ...]
                [-n <COMPONENT>] [-f <FLAGS>] [<URI>]
```

按照上面的命令行格式，输入：*adb shell am instrument -w com.xmobileapp.hello/android.test.InstrumentationTestRunner*

运行后的界面如下：

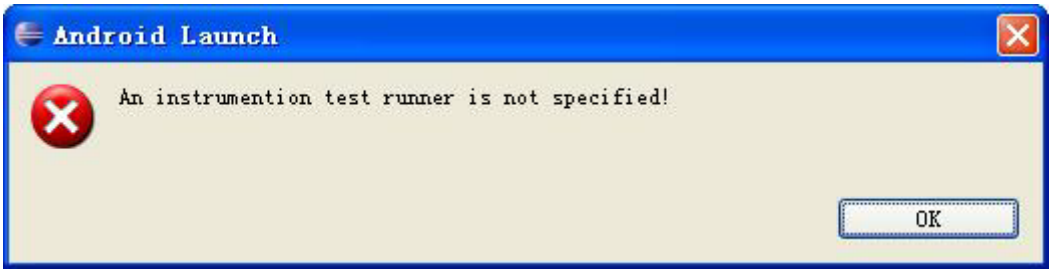
```
G:\android\android-sdk-windows-1.6\tools>adb shell am instrument -w com.xmobilea
pp.hello/android.test.InstrumentationTestRunner

com.xmobileapp.hello.tests.HelloTest:.
Test results for InstrumentationTestRunner=.
Time: 5.162

OK (1 test)
```

测试结果的信息与第 2 种方法中的 logCat 中输出的信息是完全一样的。

错误提示说明

在运行测试例子的过程中，也会遇到了不少的错误提示，总结如下：

单击“Android JUnit Test”运行后，出现“Android Launch”错误提示，如下：



这个错误的原因估计是：AndroidManifest.xml 配置错误，关于这个问题的分析说明，请阅读 An instrumention test runner is not specified 篇幅中的说明。

使用 adb shell am 命令运行，出现“Error = Unable to find instrumentation info for……..”错误信息，如下：

具体的原因是什么还不知道，如果有谁知道的，在评论中说明下。

总结说明

看了这些，对 instrument 是不是有很大的困惑，下一篇幅我们将学习 Android SDK 中的 instrument。为了方便大家学习上面的运行测试单元的方法，一个简单的例子供大家下载，这个例子中包含一个 Activity（Hello）以及对这个 Activity（HelloTest）的单元测试 2 部分，大家可以使用上面介绍的方法来启动单元测试。

相关文章

- Android、JUnit 深入浅出（七）——总结篇
- An instrumention test runner is not specified
- Android、JUnit 深入浅出（五）——AndroidTest 例子分析
- Android、JUnit 深入浅出（四）——AndroidTestRunner
- Android、JUnit 深入浅出（三）——JUnit 深入解析（下）

android.app.instrumentation 解析

已经在 Android SDK 中学习了很多关于 JUnit 的内容，但是感觉一直有几个问题没有解决（不知道大家是否有同样的感受）JUnit 的测试都自动化的，完全是不需要任何操作的，有 2 个问题我一直都还没有找到答案，这 2 个问题如下：

1. JUnit 可以用来测试 Android 的基本组成元素：Activity、Service、Provider，那么我们如何控制这些基本元素运行的，这需要 Android 系统提供一些底层操作的接口才可以做到。
2. 如何模拟界面操作，比如说：单击界面上的按钮、选择菜单等。

对于第二个问题，我在 Cview 中似乎找到了答案，Cview 中提供一些界面操作的函数，例如：performClick()，看来 Android SDK 还是提供了一些这方面的函数来模拟各种操作。对于第一个问题，本篇对这个问题幅详细说明下。

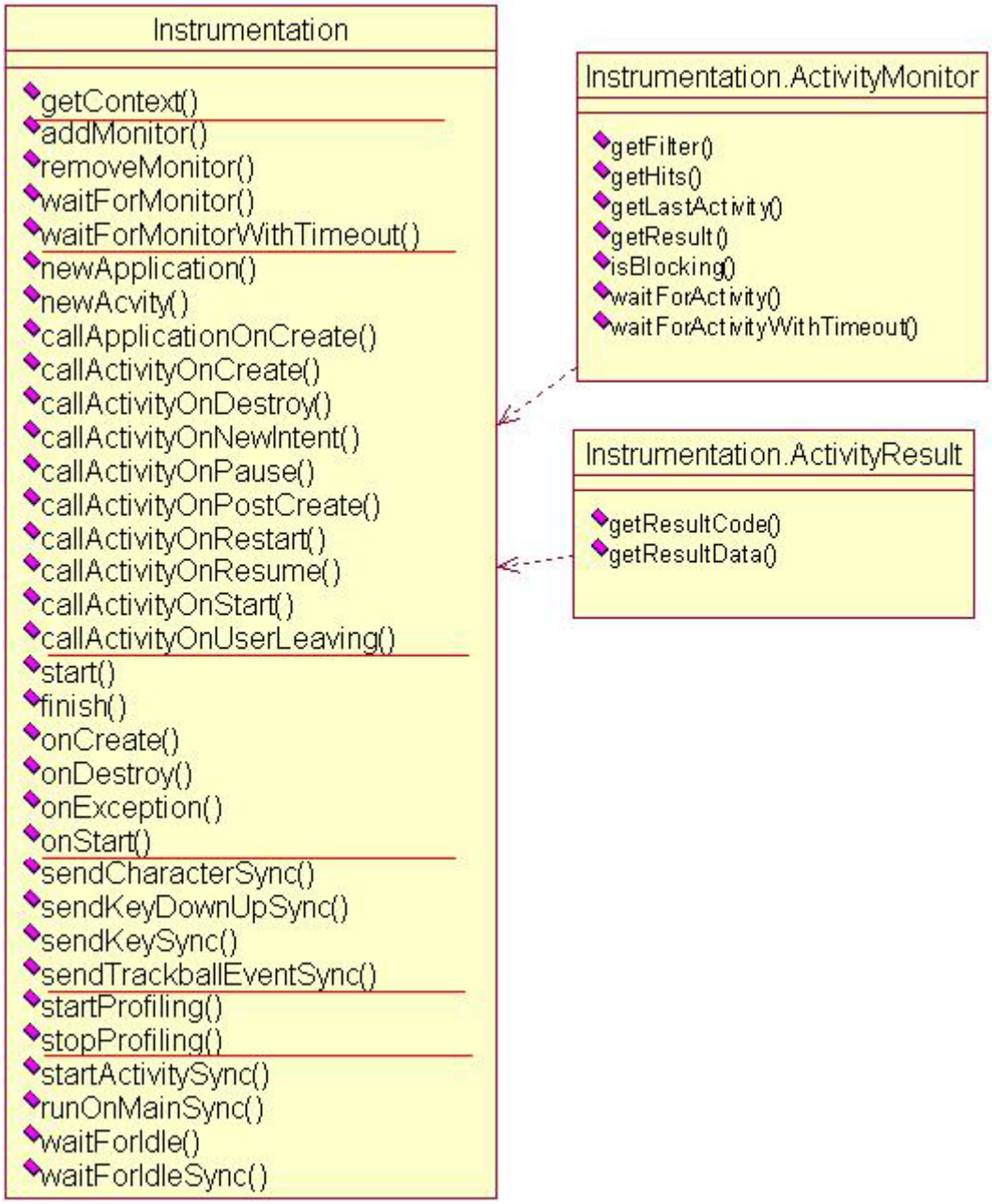
首先我们先想下，如果要实现对 Activity 的测试需要些什么？首先需要创建 Activity（这个 Android SDK 中已经提供了），然后可以控制 Activity 的生命周期，这些是最基本的。在 Android SDK 中说明过 Activity 的创建是异步的，也就是说当我们调用 startActivty()后，这个函数立刻返回，而不是等待 Activity 创建 好后才返回，这就需要一些同步方面的操作。将前面的概括起来就是 2 个方面：控制 Activity 的生命周期与同步方面的问题。Android SDK 为我们提供了 instrument，在 android.test 包中 InstrumentationTestCase 类有个函数 getInstrumentation()以及通过 adb shell am instrument 启动测试例子的方法中，都提到了 instrument，下面就对 instrumentation 详细说明。

什么是 Instrumentation?

Instrumentation 是执行 application instrumentation 代码的基类。当应用程序运行的时候 instrumentation 处于开启，Instrumentation 将在任何应用 程序运行前初始化，可以通过它监测系统与应用程序之间的交互。Instrumentation implementation 通过的 AndroidManifest.xml 中的<instrumentation>标签进行描述。

Instrumentation 似乎有些类似与 window 中的“钩子（Hook）函数”，在系统与应用程序之间安装了个“窃听器”。

android.app 包中 Instrumentation 类结构，如下图所示



其包含有 2 个内部类：ActivityMoniter、ActivityResult

- ActivityMoniter：有关特定的 Intent 的监视。一个 ActivityMoniter 类的实例通过函数 addMonitor（Instrumentation.ActivityMonitor）添加到当前 instrumentation 中，一旦添加后，每 当启动一个新的 Activity，ActivityMoniter 就会检测，如果匹配，其 hit count 计数更新等其他操作。 一个 ActivityMonitor 也可以用来寻找一个 Activity，通过 waitForActivity（）方 法，这个函数将返直到匹配的活动被创建。
- ActivityResult：一个活动执行的结果说明，返回到原来的活动。

然后我们看下 Instrumentation 类的函数，列举其主要函数如下：

- 添加、删除

ActivityMoniter;

addMonitor (Instrumentation.ActivityMonitor monitor)
Add a new Instrumentation.ActivityMonitor that will be checked whenever an activity is started.

addMonitor (IntentFilter filter, Instrumentation.ActivityResult result, boolean block)
creates an intent filter matching Instrumentation.ActivityMonitor for you and returns it.

addMonitor (String cls, Instrumentation.ActivityResult result, boolean block)
creates a class matching Instrumentation.ActivityMonitor for you and returns it.

removeMonitor (Instrumentation.ActivityMonitor monitor)
Remove an Instrumentation.ActivityMonitor that was previously added

- **Application、Activity 的创建与生命周期的控制；**

newActivity (ClassLoader cl, String className, Intent intent)
Perform instantiation of the process's Activity object.

newActivity (Class<?> clazz, Context context, IBinder token, Application applicatio
Perform instantiation of an Activity object.

newApplication (Class<?> clazz, Context context)
Perform instantiation of the process's Application object.

newApplication (ClassLoader cl, String className, Context context)
Perform instantiation of the process's Application object.

callActivityOnCreate (Activity activity, Bundle icicle)
Perform calling of an activity's onCreate(Bundle) method.

callActivityOnDestroy (Activity activity)

callActivityOnNewIntent (Activity activity, Intent intent)
Perform calling of an activity's onNewIntent(Intent) method.

callActivityOnPause (Activity activity)
Perform calling of an activity's onPause() method.

callActivityOnPostCreate (Activity activity, Bundle icicle)
Perform calling of an activity's onPostCreate(Bundle) method.

callActivityOnRestart (Activity activity)
Perform calling of an activity's onRestart() method.

callActivityOnRestoreInstanceState (Activity activity, Bundle savedInstanceState)
Perform calling of an activity's onRestoreInstanceState(Bundle) method.

callActivityOnResume (Activity activity)
Perform calling of an activity's onResume() method.

callActivityOnSaveInstanceState (Activity activity, Bundle outState)
Perform calling of an activity's onPause() method.

callActivityOnStart (Activity activity)
Perform calling of an activity's onStart() method.

- **控制 Instrumentation 的运行；**

start () Create and start a new thread in which to run instrumentation.
finish (int resultCode, Bundle results) Terminate instrumentation of the application.
onCreate (Bundle arguments) Called when the instrumentation is starting, before any application code has been loaded.
onDestroy () Called when the instrumented application is stopping, after all of the normal application cleanup has occurred.
onException (Object obj, Throwable e) This is called whenever the system captures an unhandled exception that was thrown by the application.
onStart () Method where the instrumentation thread enters execution.

- **发送按键、滚动球等事件消息到当前窗口；**

sendCharacterSync (int keyCode) Higher-level method for sending both the down and up key events for a particular character key code.
sendKeyDownUpSync (int key) Sends an up and down key event sync to the currently focused window.
sendKeySync (KeyEvent event) Send a key event to the currently focused window/view and wait for it to be processed.
sendPointerSync (MotionEvent event) Dispatch a pointer event.
sendStatus (int resultCode, Bundle results) Provide a status report about the application.
sendStringSync (String text) Sends the key events corresponding to the text to the app being instrumented.
sendTrackballEventSync (MotionEvent event) Dispatch a trackball event.

- **同步方面的操作；**

1. 创建一个 Activity 直到 Activity 开始运行；
2. 在主线程中执行一个调用，主线程被阻滞直到调用结束 ；
3. 当主线程空闲的时候（没有消息等待处理）执行一个调用；
4. 同步等待主线程处于空闲期

startActivitySync (Intent intent)	Start a new activity and wait for it to begin running before returning.
runOnMainSync (Runnable runner)	Execute a call on the application's main thread, blocking until it is complete.
waitForIdle (Runnable recipient)	Schedule a callback for when the application's main thread goes idle (has no more events to process).
waitForIdleSync ()	Synchronously wait for the application to be idle.

看了这些，我们在结合前面说的 2 个方面：制 Activity 的生命周期与同步方面的问题，在 Instrumentation 类中都实现了，尤其是些同步操作方面的。

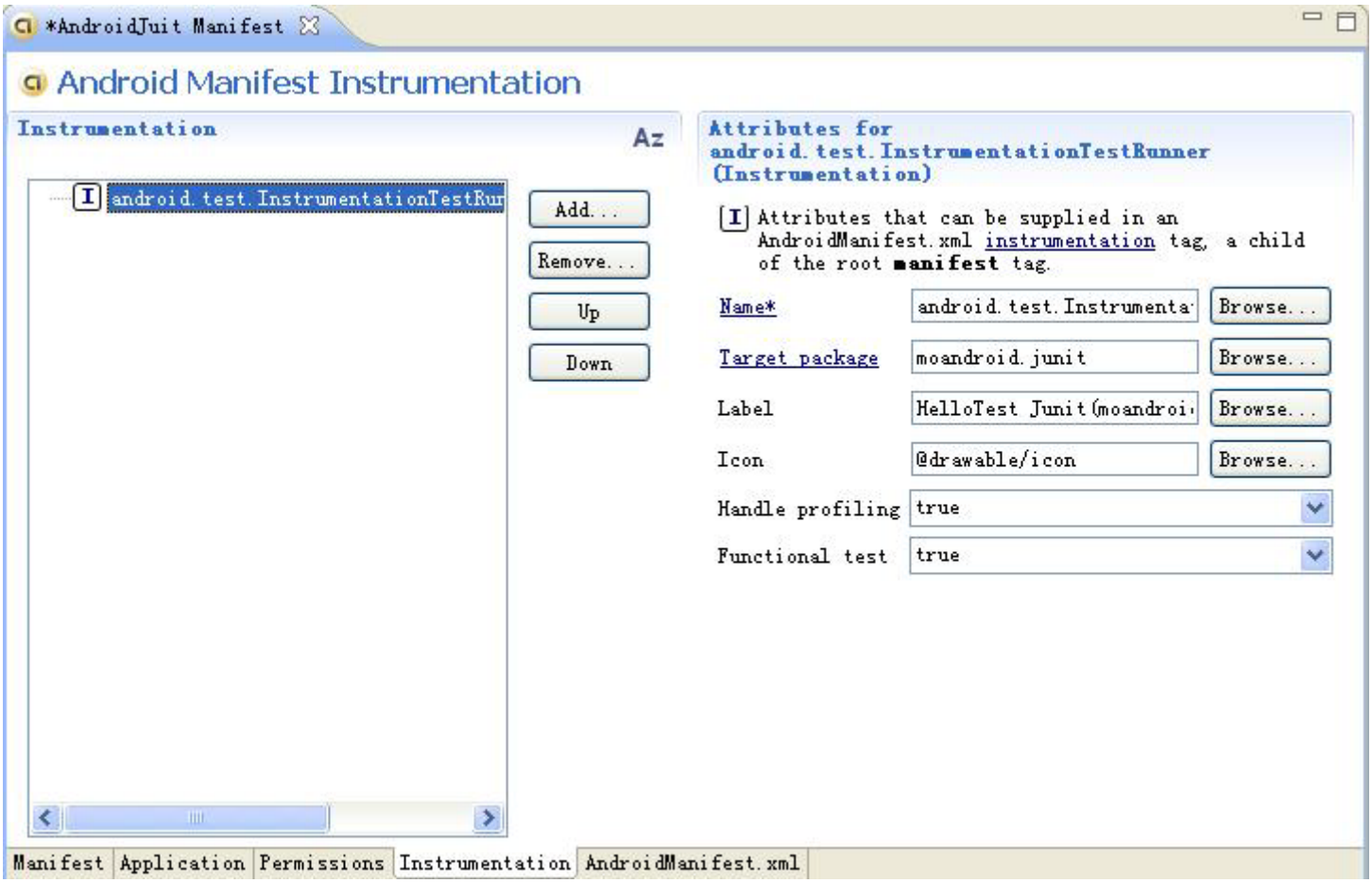
Instrumentation 简单使用的例子

```
public class HelloTest extends InstrumentationTestCase
{
  Hello mActivityTested;
  public HelloTest() {}

  //@Override
  protected void setUp() throws Exception
  {
    super.setUp();
    Intent intent = new Intent();
    intent.setClassName("com.xmobileapp.hello", Hello.class.getName());
    intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
    mActivityTested = (Hello) getInstrumentation().startActivitySync(intent);
  }

 //@Override
  protected void tearDown() throws Exception
  {
    mActivityTested.finish();
    super.tearDown();
  }
}
```

在这里我们使用 InstrumentationTestCase 类的 getInstrumentation()函数获取 Instrumentation 对象，通过 Instrumentation 的 startActivitySync()函数启动一个 Activity，直到 Avtivity 启动后返回。

AndroidManifest.xml 中的<instrumentation>标签说明

打开 AndroidManifest.xml 文件后，将会看到 AndroidManifest.xml 编辑工具，通过这个工具我们可以很方面的在 AndroidManifest.xml 添加元素



选择 Instrumentation 面板后，我们就可以编辑 Instrumentation 标签了，界面如何操作在这里就不说明了，主要说明属性值的意义：

属性	说明
Name	Instrumentation 子类的类名，这应该是一个完全限定类名（如，“com.example.project.StringInstrumentation”）。但作为一个速记，如果名称的第一个字符是“.”，包路径默认与<manifest>元素中的一样。没有默认设置，该名称必须被指定。
Target package	将要测试的应用程序，通过包名来区别应用程序，
Label	一个 Instrumentation 类的可读标签。该标签可以设置为原始字符串或一个字符串资源的引用。
Icon	代表 Instrumentation 类的图标，此属性必须设置为一个 drawable 资源引用。
Handle profiling	是否将 profiling 打开，默认设置为否
Function test	是否将 Instrumentation 当作函数单元测，默认设置为否

总结说明

学习完了 Instrumentation，心中的很多问题都没有了，还需要到 Android SDK 中去深入的学习这个类，才能更好的测试 Activity 对象。

相关文章

- android.test.InstrumentationTestRunner 解析
- Android、JUnit 深入浅出（三）——JUnit 深入解析（上）
- Android、JUnit 深入浅出（二）——JUnit 例子分析
- Android、JUnit 深入浅出（一）——JUnit 初步解析
- Android 下如何调试程序？

android.test.InstrumentationTestRunner 解析

在学习 Android、JUnit 的过程中，随着学习的深入，发现相关的内容越来越多，将这些类按照继承关系整理如下：

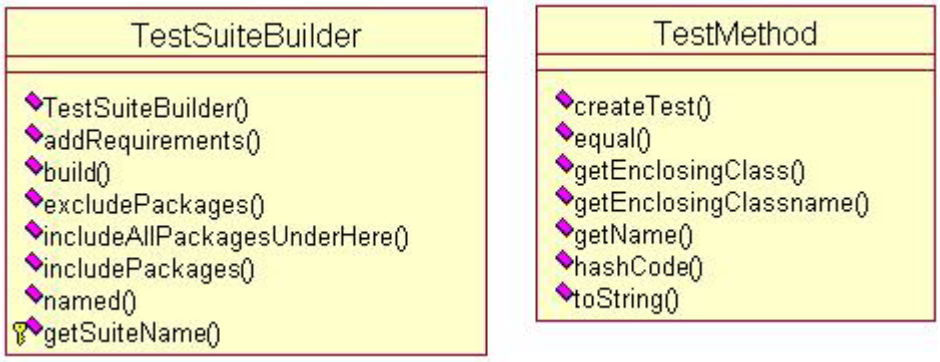
- Test—TestCase—AndroidTestCase
- Test—TestCase—InstrumentationTestCase
- Test—TestSuite—InstrumentationTestSuite
- TestListener—–BaseTestRunner—AndroidTestRunner
- Instrumentation—InstrumentationTestRunner

前 4 条路线是 Android 在 JUnit 框架上的扩展，最后一条线是一条重要的线，用一句话来说明就是：这是 Android 在 JUnit 框架的基础上锦上添花。在

android.app.instrumentation 篇 幅的学习中，我们对 instrumentation 有了一定的了解，本篇幅我们将介绍最后一个类 InstrumentationTestRunner。在 学习这个类前，我们先补充一些知识：在前面学习了这么多，但是在我们的测试例子中却没有看到核心类 InstrumentationTestSuite (这个类相当于测试单元中的容器，所有的 TestCase 都需要添加带 TestSuite 中加以管理),这是为什么了？因为在 Android SDK 中对这部分有着一层很深的“包装”，正是有了这个中间层，所以我们没有看到 TestSuite 这个容器，下面开始介绍这个中间层。

android.test.suitebuilder

包的名称似乎就在告诉我们这个包的作用：suite 产生器，其包结构如下：



从这个图上似乎没有看到我们想要的，下面我们再仔细看下 TestSuiteBuilder 这个类：

	TestSuiteBuilder(Class clazz) The given name is automatically prefixed with the package containing the tests to be run.
	TestSuiteBuilder(String name, ClassLoader classLoader)
TestSuiteBuilder	addRequirements(List<Predicate<TestMethod>> predicates) Exclude tests that fail to satisfy all of the given predicates.
final TestSuiteBuilder	addRequirements(Predicate...<TestMethod> predicates) Exclude tests that fail to satisfy all of the given predicates.
final TestSuite	build() Call this method once you've configured your builder as desired.
TestSuiteBuilder	excludePackages(String... packageNames) Exclude all tests in the given packages and all sub-packages, unless otherwise specified.
final TestSuiteBuilder	includeAllPackagesUnderHere() Include all junit tests that satisfy the requirements in the calling class' package and all sub-packages.
TestSuiteBuilder	includePackages(String... packageNames) Include all tests that satisfy the requirements in the given packages and all sub-packages, unless otherwise specified.
TestSuiteBuilder	named(String newSuiteName) Override the default name for the suite being built.

通过上面的关键字：**Exclude**(排除；不包括在内)，**Include**(包括，包含)，**the given packages and all sub-package**(给定的包和所有子包)，这些都让我们感觉到 TestSuiteBuilder 类的主要作用是：将包添加或排除在当前的单元测试中。大家是不想起来了[如何启动单元测试](#)篇幅中介绍的启动命令：

```
adb shell am instrument -w
com.xmobileapp.hello/android.test.InstrumentationTestRunner
```

这条命令的实际作用是是将 com.xmobileapp.hello 添加到当前单元测试中，在这里我们在列举一些这样的命令，如下：

运行某个 TestCase：

```
adb shell am instrument -w -e class com.android.foo.FooTest
com.android.foo/android.test.InstrumentationTestRunner
```

运行一个 TestCase 中的某个功能：

```
adb shell am instrument -w -e class com.android.foo.FooTest#testFoo
com.android.foo/android.test.InstrumentationTestRunner
```

同时测试多个 TestCase：

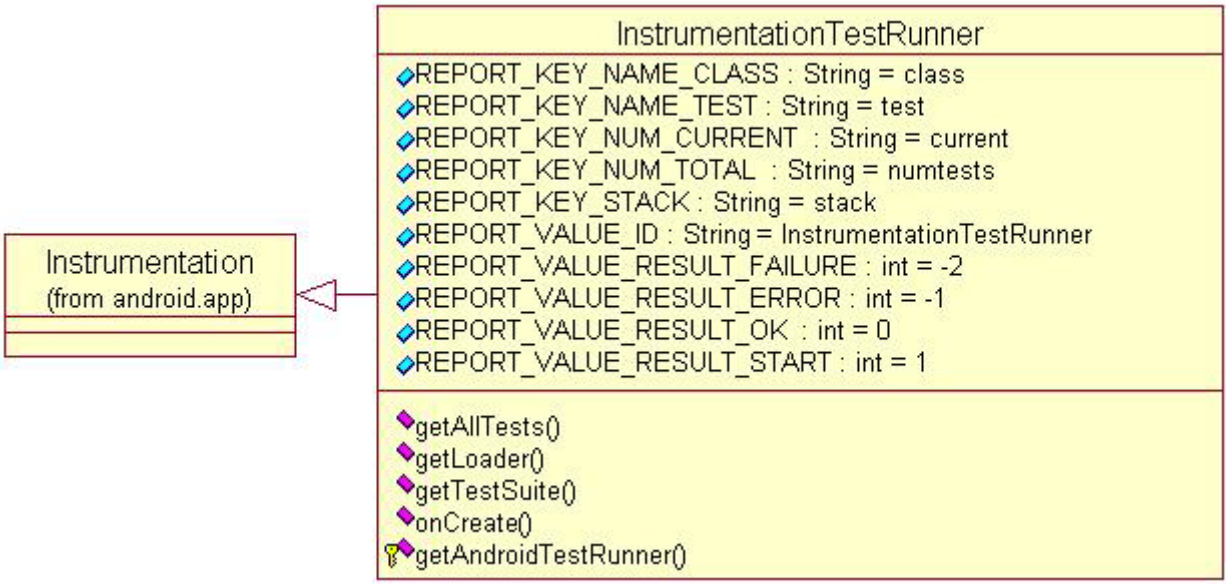
```
adb shell am instrument -w -e class
com.android.foo.FooTest,com.android.foo.TooTest
com.android.foo/android.test.InstrumentationTestRunner
```

看了这些命令，再结合 TestSuiteBuilder 的函数，我想大家就明白了一个重要的问题：在 AndroidManifest.XML 文件 Instrumentation 的属性（如下图所示）中为什么没有任何与 TestSuite 相关的说明?

```
<instrumentation
android:name="android.test.InstrumentationTestRunner"
android:targetPackage="com.example.android.apis"
android:label="Tests for Api Demos."/>
```

单元测试的配置已经取代了 TestCase 的管理,所以 TestSuite 的功能就弱化了很多，TestSuiteBuilder 就是在我们提供当 前测试的测试范围的配置，例如：是否将某个 TestCase 添加到当前测试中。TestSuiteBuilder 类的函数 builder 就根据的我们的配 置产生 TestSuite。看到这里我们的的疑惑就少了很多，下面我们继续介绍 InstrumentationTestRunner 类。

InstrumentationTestRunner 类结构，如下图所示：



主要函数接口列举如下：

TestSuite	getAllTests () Override this to define all of the tests to run in your package.
ClassLoader	getLoader () Override this to provide access to the class loader of your package.
TestSuite	getTestSuite ()
void	onCreate (Bundle arguments) Called when the instrumentation is starting, before any application code has been loaded.
void	onStart () Method where the instrumentation thread enters execution.

InstrumentationTestRunner 典型的使用过程：

1. 编写测试用例，测试用例基本上是从以下类继承的：
   - l ActivityInstrumentationTestCase
   - ActivityUnitTestCase
   - AndroidTestCase
   - ApplicationTestCase
   - InstrumentationTestCase
   - ProviderTestCase
   - ServiceTestCaseSingleLaunchActivityTestCase
2. 在 AndroidManifest.xml 中定义一个 instrumentation 并 targetPackage 属性中说明被测试的包名称;
3. 运行 instrumentation 使用“adb shell am instrument -w”，运行所有测试（除性能测试）;
4. 运行 instrumentation 使用“adb shell am instrument -w”，并添加额外的命令“-e func true’” 来运行所有的功能测试。这谢测试继承至测 InstrumentationTestCase;
5. 运行 instrumentation 使用“adb shell am instrument -w”，并添加额外的命令“-e unit true” 来运行所有的单元测试。这些测试那些非 InstrumentationTestCase 从继承的类;
6. 运行 instrumentation 使用“adb shell am instrument -w”，并添加额外的命令“-e class” 来运行某个单独的 TestCase。

看了上面的命令，我们在看下 ApiDemos\test\…\AllTests.java 中的一些注释，如下：

```
/**
 * A test suite containing all tests for ApiDemos.
 *
 * To run all suites found in this apk:
 * $ adb shell am instrument -w \
 *   com.example.android.apis.tests/android.test.InstrumentationTestRunner
 *
 * To run just this suite from the command line:
 * $ adb shell am instrument -w \
 *   -e class com.example.android.apis.AllTests \
 *   com.example.android.apis.tests/android.test.InstrumentationTestRunner
 *
 * To run an individual test case, e.g.
 * {@link com.example.android.apis.os.MorseCodeConverterTest}:
 * $ adb shell am instrument -w \
 *   -e class com.example.android.apis.os.MorseCodeConverterTest \
 *   com.example.android.apis.tests/android.test.InstrumentationTestRunner
 *
 * To run an individual test, e.g.
 * {@link com.example.android.apis.os.MorseCodeConverterTest#testCharacterS()}:
 * $ adb shell am instrument -w \
 *   -e class com.example.android.apis.os.MorseCodeConverterTest#testCharacterS \
 *   com.example.android.apis.tests/android.test.InstrumentationTestRunner
*/
```

大家就应该明白这些注释说的是什么意思了吧，这篇文章的目的也就达到了。

下面介绍一个简要的例子：

```
public class ApiDemosRunner extends InstrumentationTestRunner
{
    @Override
    public TestSuite getAllTests()
    {
        Log.i(" ApiDemosRunner",  "ApiDemosRunner::getAllTests()" );
        return new TestSuiteBuilder(ApiDemosRunner.class)
            .includeAllPackagesUnderHere()
            .build();
    }

    @Override
    public ClassLoader getLoader()
    {
        return ApiDemosRunner.class.getClassLoader();
    }
```

}
看了这段代码，大家就明白 TestSuiteBuilder 的主要作用了，这里我们需要说明的：万变不离其中，整个测试的核心还是 TestSuite，只 不过 Android SDK 在此基础上增加了 TestSuiteBuilder，是我们对 TestCase 的管理更加方便。
InstrumentationTestRunner 类相对比较简单，看了上面的例子，以后按照这种方法使用就可以了，这里就不在详细说明。

总结说明

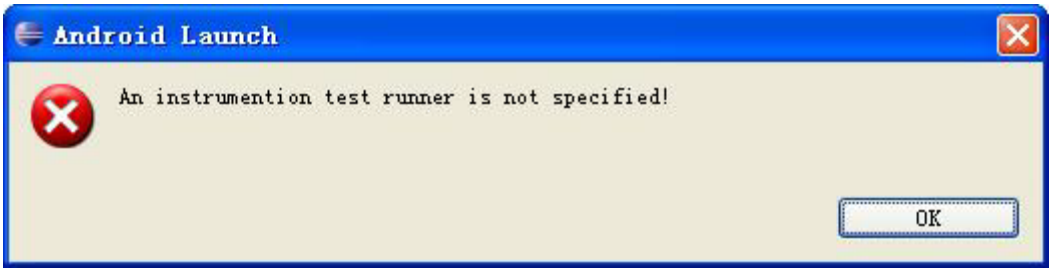
Android SDK 在单元测试方面的封装比较完美，我们几乎不需要写太多的代码就可以完成单元测试。

相关文章

- android.app.instrumentation 解析
- Android、JUnit 深入浅出（三）——JUnit 深入解析（上）
- Android、JUnit 深入浅出（二）——JUnit 例子分析
- Android、JUnit 深入浅出（一）——JUnit 初步解析
- Android 下如何调试程序？

An instrumention test runner is not specified

单击“Android JUnit Test”运行后，出现“**An instrumention test runner is not specified** ”错误提示，如下：



同时，在程序的 console 面板中会输出如下信息：
ERROR: Application does not specify a android.test.InstrumentationTestRunner instrumentation or does not declare uses-library android.test.runner。
出现错误的原因可能是：AndroidManifest.xml 配置错误。那么在 AndroidManifest.xml 到底需要配置哪些内容呢，下面一一为大家说明：

1. 在<application>增加引用 android.test.runner 的声明，如下图所示：

```
<uses-library android:name="android.test.runner"/>
```

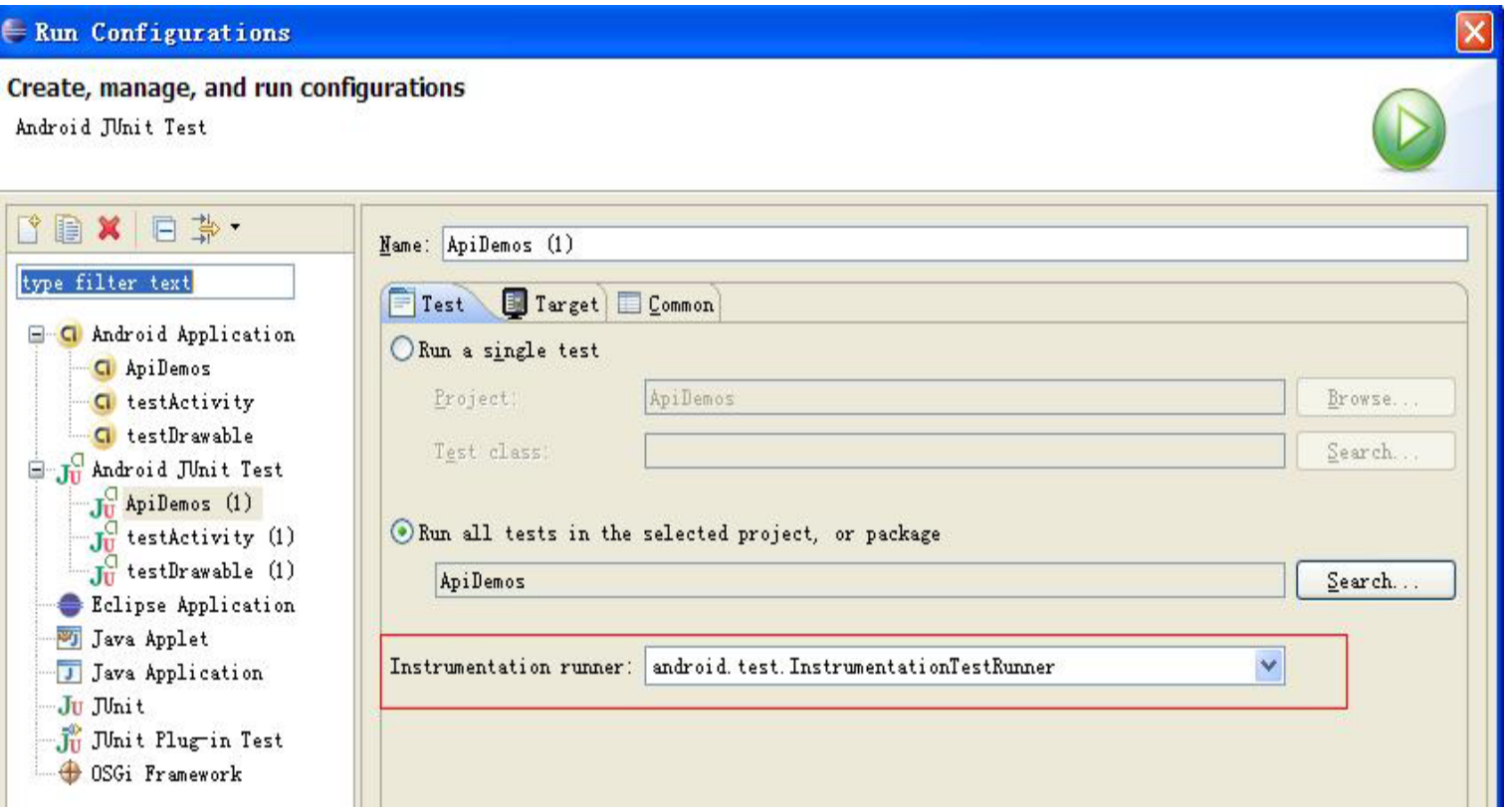
2. 然后在<manifest>中增加 instrumentation 的信息说明，如下图所示：

```
<instrumentation
    android:targetPackage="com.example.android.apis"
    android:name="com.example.android.apis.test.ApiDemosRunner"
    android:label="ApiDemoes JUnit Test Sample"/>
```

3. 最后还需要增加 uses-permission 声明，如下图所示：

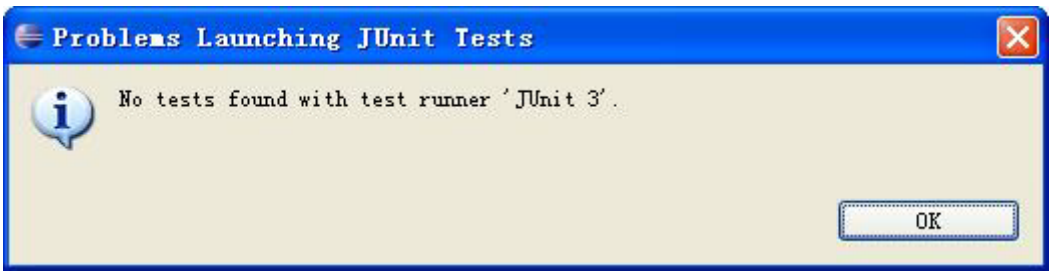
```
<uses-permission android:name="android.permission.RUN_INSTRUMENTATION"/>
```

根据自己的程序，在 AndroidManifest.xml 文件中配置上面的信息。如果上面的信息配置正确，鼠标右键单击工程，选择 Run As\Run configurations，在 Android JUnit Test 选项中选择工程，将会看到下面这个界面：



在 Instrumentation runner 后的列表框选项中，我们看到 android.test.InstrmentationTestRunner，并处于当前选择状态。如果这个没 有选择框中没有任何选项，就说明 AndroidManifest.xml 配置有问题。

如果你确信自己的配置没有问题，而且上面的界面显示也没有任何问题，但是运行却出现下面这个错误提示：



这个提示出现的原因，在 JUnit 的 FAQ 中有对这个错误的说明，这里我也不知道是不是这个原因导致的错误，JUnit FAQ 中是这么说明的：

Why do I get the warning "AssertionFailedError: **No tests found in XXX**" when I run my test?
Make sure you have more or more method annotated(有注解的) with **@Test**.
For example:

```
@Test
public void testSomething() {}
```

这个错误在提示我们：当前的 test runner 没有任何 Test。在我们的 package 中包含有不少的 TestCase，一切也都是按照 Android SDK 中的设置来编写的代码，出现这个错误实在是莫名其妙，但是有一点可以说明的，当前这个单元测试有问题。

即使是 Android SDK 中的例子程序运行也存在这个问题，我运行 ApiDemos\test 中的 JUnit 测试单元，也一直存在这样那样的问题。在学习到了前面的这些知识后，按照 Android SDK 中的规范，自己写个 JUnit 测试程序，步骤如下：

首先，从 InstrmentationRunner 继承，实现我们的自己的 Instrument

```
public class MoandroidTestRunner extends InstrumentationTestRunner
{
    @Override
    public TestSuite getAllTests()
    {

    }

    @Override
    public ClassLoader getLoader()
    {
        return MoandroidTestRunner.class.getClassLoader();
    }
}
```

然后，将这个 TestCase 添加到 TestSuite 中，并将 TestSuite 返回到 MoandroidTestRunner

```
public TestSuite getAllTests()
{
    InstrumentationTestSuite suite = new InstrumentationTestSuite(this);
    suite.addTestSuite(ForwardingTest.class);
    return suite;

    /*—更加简单的方法
    return new TestSuiteBuilder(ApiDemosRunner.class)
        .includeAllPackagesUnderHere()
        .build();
    */
}
```

最后，按照前面说的步骤，配置 AndroidManifest.xml 文件

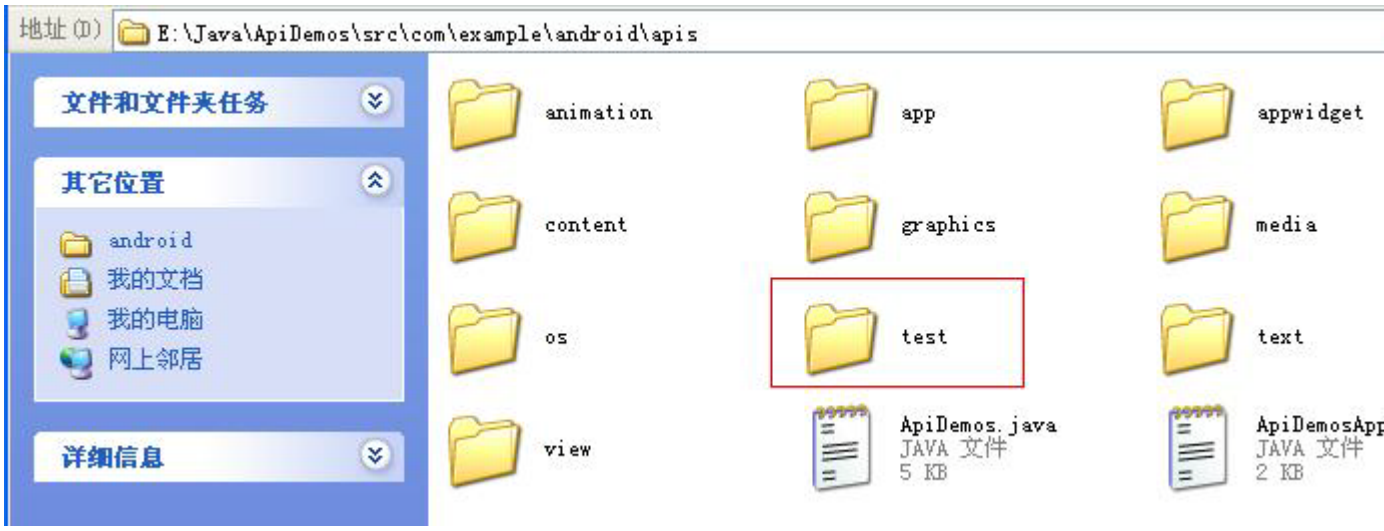
这里需要说明的，我们创建了自己的 Instrumentation，在配置要将 Instrumentation 的 android:name 属性设置为我们自己的 MoandroidTestRunner（别忘了还需要加上包名）。

```
<instrumentation
    android:targetPackage="com.example.android.apis"
    android:name="com.example.android.MoandroidTestRunner"
    android:label="ApiDemoes JUnit Test Sample"/>
```

按照上面的方法，创建 JUnit 测试单元后，分别在 Eclipes 中、Dev Tool 中 Instrumentation、adb shell am -w 命令启动，应该就可以看到测试运行了。看来这多的例子程序，实际上 JUnit 单元测试的核心就是：TestCase、TestSuite，无论 Android SDK 再此基础上怎么扩展，只要我们能了解这个核心，所有的变化也就是锦上添花而已。

这里我将 ApiDemos 中的测试单元修改后，上传博客与大家分享，下载测试例子程序，主要修改说明如下：

- 去掉了其自身包含的 test 子文件，在 src\com\example\android\apis 包下创建一个 test 子包，其包含所有的测试代码，如下图所示：



- 手工修改配置文件 AndroidManifest.xml

在 Eclipes 中选择工程，单击右键，在 Run as/Debug as 子菜单选项中选择 Android JUnit Test，JUnit 测试界面就会显示测试过程了。

补充说明

在 Android SDK 的 troubleshooting 页面中也说明了一个可能出现的错误，原文如下：
If you are developing on Eclipse/ADT, you can add JUnit test classes to your application. However, you may get an error when trying to run such a class as a JUnit test:
Error occurred during initialization of VM
java/lang/NoClassDefFoundError: java/lang/ref/FinalReference
这个错误的主要原因是 android.jar 并没有完全包含 Junit.* 类，只是包含了部分类，至于如何修改这个错误，还是到 Android SDK 中去查看具体的办法吧。

总结说明

在学习的过程中，不断动手写例子程序，发现问题后查阅资料，并把这些问题总结下来与大家分享。也是通过这些问题让我对 JUint 有了比较深入的了解，尤其是 Android 是如何扩展 JUnit。如果你按照上面的方法编写 JUnit 测试例子，运行还是有问题，在博客上留言，并将程序发到我们的邮箱 **moandroid@gmail.com** 中，我们会尽量帮你解决。

相关文章

- Android、JUnit 深入浅出（七）——总结篇

- Android、JUnit 深入浅出（六）——如何运行单元测试？
- Android、JUnit 深入浅出（五）——AndroidTest 例子分析
- Android、JUnit 深入浅出（四）——AndroidTestRunner
- Android、JUnit 深入浅出（三）——JUnit 深入解析（下）

Android、JUnit 深入浅出（七）——总结篇

在学习 Android、JUnit 的过程中，随着学习的深入，将 Android、JUnit 的类按照继承关系整理如下：

- Test—TestCase—**AndroidTestCase**
- Test—TestCase—**InstrumentationTestCase**
- Test—TestSuite—**InstrumentationTestSuite**
- TestListener——BaseTestRunner—**AndroidTestRunner**
- Instrumentation—**InstrumentationTestRunner**

上面的 5 条路线，也是我们不断学习的过程，对于前 4 条路线感觉自己解析的都比较清楚，最后一条路线似乎说的不是很清楚，后来我又查看了不少这方面的资料，对 Instrumentation 再次说明下。

每个 Android 应用程序运行在自己的进程，Instrumentation 杀死当前应用程序，并重新启动应用程序（restarts the process with Instrumentation）。Instrumentation 提供给我们一个应用程序上下文的 Handle，通过这个 Handle 我们可以洞察应用程序，从而验证测试断言，我们还可以通过它来写一些比界面测试更加底层的测试用例。需要强调说明的是：Instrumentation 不能捕获 UI 方面的 bugs。

Android 在 JUnit 的基础上扩展出来的、与 Instrumentation 有关的 3 个类：

类	描述
InstrumentationTestCase	它扩展了 JUnit 中的 TastCase，并提供了一个接口 getInstrumentation() 获取 Instrumentation 类。这个可以根据自己的需求来扩展这个类，比如说：测试中可能会启动某个 Avtivity 和发送按键事件，为此完成测 试，instrumentation 需要将其注入到 TastCase 中。
InstrumentationTestRunner	它是 Instrumentation 的基础上扩展的，它将自己注入到每个测试用例本身，测试用例需要分组到一个适当的 InstrumentationTestRunner 运行起来。
InstrumentationTestSuite	它扩展了 JUnit TestSuite，其主要作用是保证每个 TestCase 在运行前，Instrumentation 能注入到 TestCase 中，InstrumentationTestRunner 中需要使用 InstrumentationTestSuite。

以上说明来自网页 Instrumentation Testing（英文的），在这里推荐给大家阅读。

JUnit 的使用心得

JUnit 是采用测试驱动开发的方式，也就是说在开发前先写好测试代码，主要用来说明被测试的代码会被如何使用，错误处理等；然后开始写代码，并在测试代码中逐步测试这些代码，直到最后在测试代码中完全通过。

看了是否感觉有些不符合程序员的思维习惯（先写代码然后在调试），的确这也是 JUnit 是对程序员思维习惯的“颠覆”。在这里我自己也感觉，好像很 难做到，为什么？在一匹“马”没有完全设计好前，怎么规定这匹“马”将来会如何跑？而且即使把“马”将来会如何“跑”定义好了，在实现的时候“马”被改变 了怎么办？说到底还是：一个人不能同时具有 2 个角色，否则自己有时候就不知道当前是哪个角色！

说到这里，我就说明下，我自己**对 JUnit “错误”的使用方法**，这也许与 JUnit 测试驱动开发的目的相矛盾，但是的确可以有效地减少 bug。JUnit 从核心来说就是将源代码与测试代码完全分开，将测试代码作为一个单独的程序。前面介绍的方法，都将源代码与 测试代码合为一体，由于源代码的重要性大于测试代码的重要性，所以测试代码经常有不完整、结构不清晰等问题，这样程序员的单元测试也就不完整。JUnit 就是被我用来做完整的单元测试，对当前的部分代码，测试其在每种“环境”下的运行结果。

后记说明

历时半个月的学习，总算把 Android、JUnit 深入解析篇写完了，还是比较全面的介绍了 Android 中使用 JUnit 的方方面面，在写博客的过程中，我也尽量把自己遇到的每个问题，在这里与大家分享，并尽量把每个问题解析清楚。如果你看了之后，感觉有什么不足、错误、遗漏的地方，欢迎大家在博客中留言。

相关文章

- An instrumention test runner is not specified
- Android、JUnit 深入浅出（六）——如何运行单元测试？
- Android、JUnit 深入浅出（五）——AndroidTest 例子分析
- Android、JUnit 深入浅出（四）——AndroidTestRunner
- Android、JUnit 深入浅出（三）——JUnit 深入解析（下）