CTS测试的环境搭建及使用

修订记录

|  |  |  |
| --- | --- | --- |
| **日期** | **修订版本** | **修改描述** |
| 2021/10/13 | V1.0 | 初始版本 |
|  |  |  |

本文档旨在通过“手把手”的教程，帮助项目组成员快速掌握CTS测试的操作流程，能够学会用例的代码编写、编译、测试以及分析报告。

一、CTS简述

1. 基本介绍

CTS是“兼容性测试套件”的英文简称，CTS是一个**自动化测试工具**，也是一套**单元测试**，它运行在**桌面**机器上，并直接在附加设备或模拟器上执行测试用例。它的目的是在集成到工程师构建设备的日常工作流程（例如通过连续构建系统）中。其目的是尽早发现不兼容性，并确保软件在整个开发过程中保持兼容性，保证开发的应用在所有兼容Android设备上正常运行。

CTS的源码包含在“Android 开源项目 (AOSP)”中，可以通过编译源码获得，也可以从谷歌的官网单独下载二进制文件。具体的使用方法详见：<https://source.android.google.cn/compatibility/cts>

1. 组成

* CTS Trade Federation：在桌面设备上运行，并管理测试执行情况。Trade Federation简称 Tradefed，内部仍然是通过adb跟设备通信的，并通过对若干adb命令的组织封装，实现了一键自动化功能。
* 测试用例：在目标设备上运行，采用 Java 语言编写为 JUnit 测试，并打包为 \*.apk 文件，由CTS Trade Federation管理（比如安装、执行等）

1. 工作流程



4、工作原理

基本原理就是利用测试APK调用对应的系统接口，保证系统对应的接口按照谷歌的要求返回对应的值。

5、作用

通过CTS 测试，确保了APP与Android Framework之间有一致的调用接口（API），这使得APP开发者编写的同一款程序可以运行在不同系统版本（向前兼容）、不同硬件平台、不同厂商制造的不同设备上。

二、CTS环境搭建

注意：CTS 目前支持 64 位 Linux 和 Mac OS 主机。CTS 无法在Windows 操作系统上运行。在虚拟机和ubuntu下都测试通过，但是虚拟机上无法在终端显示正在测试情况，会打印出null的信息，但是在纯linux环境ubuntu下是正常的。

本文档以Ubuntu18.04为操作系统，针对CCS2.0+项目(Android P)书写。

1. 准备一台PC，硬件上硬盘空间不低于500G，内存不低于16G。下载Android源码，安装编译依赖的所有工具链。这本身不是CTS的范畴，不多赘述。
2. Android源码编译通过

先编过源码，然后才好编译CTS。

1. 编译CTS

（1）source ./build/envsetup.sh

（2）lunch（选择33） mek\_8q-eng

（3）make cts

对于CCS2.0+之外的项目，“lunch mek\_8q-eng”请改成你们项目使用的构建目标。

1. CTS源码结构

cts // Android兼容性测试套件标准

apps // cts测试所使用的apk源码

   ├─CtsVerifier

       ├─jni //mk,cpp C++代码

       ├─res //项目资源放置并且编译应用程序的地方

       ├─src //有关verifier的源码

       ├─tests //src目录对应的测试用例

build // cts套件的编译配置

common // cts套件的通用代码（包括设备端的检查锁屏、外设、wifi、phone，）

development // cts套件兼容之前使用eclipseADT开发apk的脚本

hostsidetests // cts套件新增host测试项

libs // cts套件使用的第三方库

├─ deviceutillegacy

├─ json  json库文件

├─ runner   包含Makefile的文件来运行tests/core/runner目录下的核心测试

├─ view

├─ testserver  http测试服务

├─ vogar-expect  是Java的命令行测试实用工具

├─ wrappedgtest

suite // cts套件

tests // cts套件进行的测试项，这个目录包含Java测试用例，测试不同类型的东西，主要是从Java应用程序中获得的特性。

tools // cts套件使用的测试工具源码 ，此目录包含可供您使用的不同类型的工具。

5、编译好cts后生成的文件位置

CTS测试套件涉及到的目录有：

（1）Package CTS : out/host/linux-x86/cts/android-cts.zip

（2）Test packages : mydroid/out/host/linux-x86/cts/android-cts/testcases

（3）Test tools: mydroid/out/host/linux-x86/cts/android-cts/tools

（4）Test results : mydroid/out/host/linux-x86/cts/android-cts/results

（5）Cts make file : mydroid/build/core/tasks/cts.mk

三、编写CTS测试代码

CTS测试使用 JUnit 和 Android测试API来编写测试代码。cts测试代码编写就是单元测试代码编写，在测试方法中，调用对应的接口，通过判断接口返回值与预期返回值是否一致，来判别接口是否成功调用。

已经熟悉JUnit和安卓插桩测试的用户可以跳过这一节。

1. JUnit介绍

Junit是一个可编写重复测试的简单框架，是基于Xunit架构的单元测试框架的实例。JUnit4是JUnit框架有史以来的最大改进，其主要目标便是利用JDK 5的Annotation特性简化测试用例的编写。

1. 测试类和测试方法

新建一个测试类，里面包含一些测试方法。在方法上面添加注解“@Test”，示例如下：

**public** **class** CalculatorTest {

    @Test

**public** **void** test1() {

        assertThat(resultAdd, is(equalTo(2d)));

    }

    @Test

**public** **void** test2() {

        assertThat(resultAdd, is(equalTo(3d)));

    }

}

每个用“@Test”标注的方法就是一个用例，每个用例执行结果只有两种：OK、fail。一个正确的用例，如果执行fail，就暗示代码逻辑可能有bug。

处于规范性，每个测试方法命名最好以test为前缀。例：testDoAudioNext（）。判断是否调用成功：

（1） 调用的接口是**有返回值**的方法，可以比较返回值的不同来判断是否调用成功。

（2）调用的接口是**无返回值**的方法，可以根据调用前后某些特征的不同，来判别接口是否调用成功。

常用的**断言**方法有：assert（）；assertTrue（）/assertFalse（）；assertEquals（）/assertNotEquals（）；assertNull（）/assertNotNull（）；assertSame()/assertNotSame（）。

1. 执行

可以执行单个测试方法，也可以执行整个测试类，甚至可以一次执行多个测试类。这种一次包含多个测试类称为“测试套件”，用@Suite.SuiteClasses()注解创建。

Android Studio原生支持单元测试。用鼠标选中代码区方法名，右击-run，执行单个测试方法；用鼠标选中代码区类名(或选中左侧树状导航-类文件名)，右击-run，执行整个测试类中的所有case；

1. @Before @BeforeClass @After @AfterClass

如果当前测试类的每个测试方法，都存在测试前的初始化工作以及测试后的清除工作，比如在接口测试前面做一些前置的参数赋值，数据库操作等等，在测试后做参数还原或销毁，数据库的还原恢复等，那么按照正常思路，我们可以在每个@Test方法的开头实现初始化代码，末尾实现清楚代码。但是这样就存在大量重复。因此引入@Before、@After、@BeforeClass、@AfterClass注解。

简单来说，@Before、@After会在每一个被@Test注解的方法前、后分别执行，比如有5个被@Test注解的方法，那么@Before、@After会被执行5次。被@Before、@After注解的方法一般起到初始化和释放的作用，这么做就保证了每一次@Test执行，他们都是相互独立，互不干扰的。

而@BeforeClass、@AfterClass从头至尾仅会执行一次。对于那些较耗资源的、只需要初始化一次的，建议采用这种方法。

举例来说，假设class CalculatorTest 里面有test1、test2、test3三个测试方法，那么执行顺序大致是：

@BeforeClass-@Before-test1-@After-@Before-test2-@After-@Before-test3-@After-@AfterClass。

然而test1、test2、test3这些case之间的顺序是不确定的。因为从理论上讲，每个case都是独立的个体，测试结果不应该依赖case的执行顺序。

1. @Test的执行顺序

一般来说，被@Test注解的方法是随机顺序执行的，因为单元测试，本身它们就应该是互不干扰可以独立运行的。但是某些特殊情况下，想让它按照一定顺序执行的话，可以使用@FixMethodOrder(MethodSorters.NAME\_ASCENDING),放在JUnit测试类定义的外面，这样一来就会按照方法名称的升序来执行。此时命好方法的名字，就可以轻松实现既定顺序执行。

三种执行顺序选择

MethodSorters.JVM：按照JVM得到的顺序执行，即按照代码顺序执行。

MethodSorters.NAME\_ASCENDING：按照方法名字顺序执行。

MethodSorters.DEFAULT：按照默认顺序执行，以确定的但是不可预期的顺序执行。

1. AndroidManifest.xml

前面介绍过，CTS用例会打包成apk，是在Android设备终端上执行的，仅凭JUnit显然做不到这一点。所以CTS用例自身也是一个Android项目，需要有一个“AndroidManifest.xml”。在AndroidManifest.xml中添加一个instrumentation标签，指定正在使用的测试运行程序为AndroidJUnitRunner：

<!--     This is a self-instrumenting test package.-->

**<instrumentation** android:name="android.support.test.runner.AndroidJUnitRunner"

        android:targetPackage="com.hsae.autocore.cts"

        android:label="CTS tests of autocore component"**>**

**<meta-data** android:name="listener"

            android:value="com.android.cts.runner.CtsTestRunListener" **/>**

**</instrumentation>**

因为cts只测试对外暴露的接口，故targetPackage是项目自身的包名，而不是测试目标apk包名。其它配置保持不变。

1. AndroidTest.xml

在本文档末尾给出一个AndroidTest.xml的示例。开发者把示例拷贝到自己的项目根目录即可。只需一点修改：

test-file-name的value值修改为你的apk名，比如我现在给AutoCore设计CTS用例，那么我的apk是CtsAutoCoreTestCases.apk。

package的value值修改为你的项目包名。例如我现在是“com.hsae.autocore.cts”。

其它属性的含义无需深究，大部分从命名上也能猜出来。

1. Android.mk

跟普通app大同小异，添加一句：

LOCAL\_COMPATIBILITY\_SUITE := cts

以及引入单元测试依赖的类库。

在文档末尾给出一个范例。

1. 构建

将写好的测试代码放到源码服务器<source\_dir>/cts/tests/tests目录下。

如果没有执行过make cts，按照第二章(CTS环境搭建)执行命令即可。

如果执行过make cts，那么只需make你的代码模块。

编译好后，生成apk在mydroid/out/host/linux-x86/cts/android-cts/testcases目录下。

1. AndroidStudio的环境配置

尽管代码是在源码环境下用make脚本编译的，但是为了开发方便，我们用Android Studio作为开发环境，需要把AndroidStudio的环境配置好，如下几点：

（1）新建一个普通的APP项目，与正常的应用开发没有区别。

（2）修改build.gradle,引入需要使用到的单元测试库，如下图



当然，有些不是必须的，不过你全部引入就是了。

（3）添加AndroidTest.xml和Android.mk文件。

四、CTS运行流程

1. 在服务器上将测试代码编译完成。
2. 在Linux系统下使用命令运行cts
3. 使用adb连接车机，使用adb devices查看并确认待测设备状态；
4. 执行source build/envsetup、lunch <target\_product>
5. 确保adb只连接了一台设备, 然后运行cts-tradefed，进入cts命令控制台
6. 假设你要执行CtsAutoCoreTestCases里的所有case，运行命令：

run cts --abi arm64-v8a -m CtsAutoCoreTestCases --skip-device-info --skip-preconditions --skip-all-system-status-check --disable-reboot

1. 运行完成后，会自动生成测试结果报告。如果运行速度非常缓慢，请参考本文最后一节。

run cts -m CtsAutoCoreTestCases命令里的-m是module的意思，即执行指定module里的所以case。有时你只希望执行里面的一个类，请添加—test参数，比如：

run cts -m CtsAutoCoreTestCases --test com.hsae.autocore.cts.InstrumentTest

再如果，你只希望执行一个类里面的一个方法，参考：

run cts -m CtsAutoCoreTestCases --test com.hsae.xx.cts.XTest#test\_isAccOn

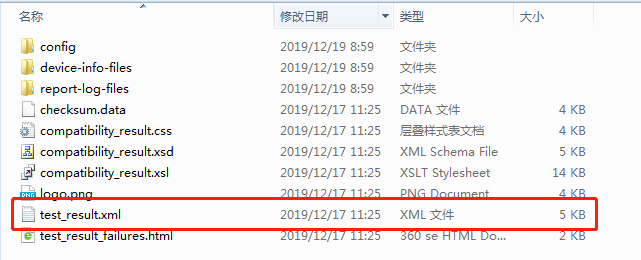
--skip-device-info ：跳过收集设备相关信息的步骤

--skip-preconditions ：跳过前提条件以缩减运行时间，从而对新测试执行迭代开发。这会绕过验证和设置设备配置（例如推送媒体文件或检查 Wi-Fi 连接）。

除了按模块执行，你还可以按计划分类执行。总的来说，命令是非常灵活的。深入介绍run cts命令不是本文的目的，读者请自行查看官方文档。

五、CTS测试报告

（1）测试任务执行完成后，会在./android-cts/results文件夹中生成对应的测试结果及./android-cts/log文件夹中有对应的日志文件。

（2）失败用例单独输出到一个html格式的文件，可以用任何浏览器直接打开查看。test\_result.xml文件包含全部测试结果，用支持xml的浏览器打开，也可以用专门的xml解析工具打开，这类工具很多。大部分浏览器出于安全限制，禁止本地加载XML，甚至有些浏览器本身就不支持XML Schema。目前仅知道IE 浏览器可以正常打开。

（3）打开test\_result.xml文件可以查看已执行的测试计划的详细信息，如：CTS计划名称、执行开始和结束时间，提供通过、失败、超时或无法执行的测试数量的汇总摘要，如下图所示：



# 附录

## 可能遇到的问题

1. 编译报错：./tests/tests/telecom2/res/.gitignore路径下缺少文件，但是该文件是空文件，添加之后可以编译通过
2. 错误信息：

build/core/base\_rules.mk:238: error: external/googletest/googletest: MODULE.TARGET.STATIC\_LIBRARIES.libgtest already defined by external/googletest/googletest

分析：源码服务器是大家共用的。有人修改了环境变量。进一步分析发现，有人在~/.bashrc里添加了：

export NDK=/home/work/xwltest/android-ndk-r21/

export PATH=${PATH}:$NDK

export NDK\_ROOT=/home/work/xwltest/android-ndk-r21/

解决：执行unset NDK\_ROOT即可解决。

1. 找不到'FindClass'方法

cts/hostsidetests/jvmti/base/jni/cts\_agent.cpp:33:37: error: use of undeclared identifier 'FindClass'

分析：'FindClass'函数是art/test/ti-agent/ jni\_binder.h提供的，在CCS2.0的代码里，确实存在这个函数，但是到了CCS2.0+，方法名改成了‘GetClass’，而CTS里面没有跟着改过来，因此报错。

解决：打开报错文件，将'FindClass'  改成“GetClass”即可。

1. 执行run cts命令后，命令直接返回，没有输出。

分析：这是正常现象，run cts命令不会阻塞当前的命令窗口，而是把命令发送到某个队列然后立即返回。命令会在后台执行，output输出到当前控制台上。

如果继续输入下一条命令，会排队，等前面的命令执行结束。

1. run cts执行过程中卡住，有时一个非常简单的case，理论应该很快就可以跑结束，实际要等十几分钟，在中间的某些阶段，界面不动，似乎是有什么阻塞了。导致开发阶段我们调试case代码极为困难。
2. 记住不要用虚拟机，要用物理机。
3. 如果你的车机su命令有密码，卡住是因为工具内部执行了su，然后su命令阻塞等你输入密码。命令调用是在工具里，你没法输入密码，所以就一直阻塞着。

比方说，阻塞很久后打印下面异常：

11-26 16:48:47 W/NativeDevice: Failed to query MAC address for 1e12700e8288de07

11-26 16:48:47 W/NativeDevice: com.android.ddmlib.ShellCommandUnresponsiveException

实际上是执行了“su root cat /sys/class/net/wlan0/address”，su命令阻塞，PC端一直等待直到超时失败打印异常。

解决：请联系项目经理，提供专用于CTS测试的版本，去掉密码。

1. run cts时，实际上执行的不仅仅是指定的case，还有许多例外的检查，比如查询设备信息等。这些步骤对我们没用，而且会拖慢执行速度，加上三个参数跳过去：-skip-device-info --skip-preconditions --skip-all-system-status-check。
2. 开发阶段没有物理条件、没有安装好Ubuntu的机器、切adb也不方便。

解决：须知cts本质上还是调用am instrument命令执行的。我们可以自行安装好apk，然后在Android Shell里面执行：

am instrument -w -r -e debug false -e class 'com.hsae.xxx.cts.XXTest' com.hsae.autocore.cts/android.support.test.runner.AndroidJUnitRunner

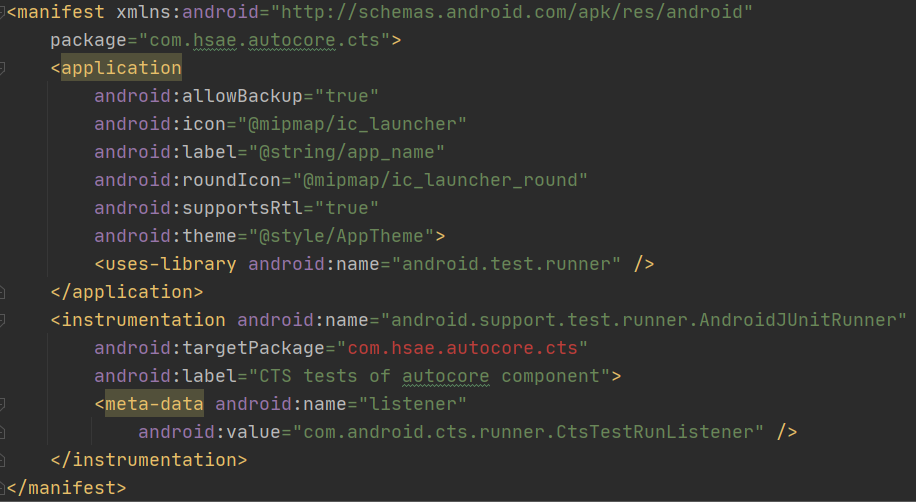
但是这仅作为开发者在开发阶段方便调试的临时手法，正式还是要用cts命令执行，输出报告。

## 补充点

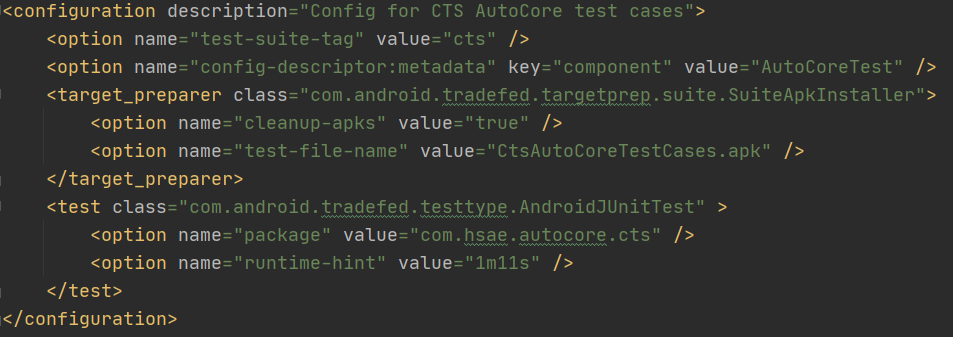
* 如果需要提交Google确认版本测试结果，我们采取下载官方CTS测试包进行测试并提交测试结果，不会采取源码编译的方法。（测试结果提交方只承认官方下载的测试包测试出来的结果）
* 执行官方CTS测试包，请按官方要求，设置好Android设备，包括恢复出厂设置、语言设置为英语、打开GPS等。具体要求见<https://source.android.google.cn/compatibility/cts/setup>
* CTS的版本有 v1和v2，对于 Android 6.0 及更低版本，使用 CTS v1；对于 Android 7.0 或更高版本，使用 CTS v2。本文档针对CCS2.0+,安卓版本是9.0，故使用的CTS v2。以后不会有基于安卓6.0的项目，故开发者没有必要了解CTS v1。但是在网络上查阅CTS的资料时，许多网上的资料都是基于V1的，开发者应该有能力区别。

# 案例

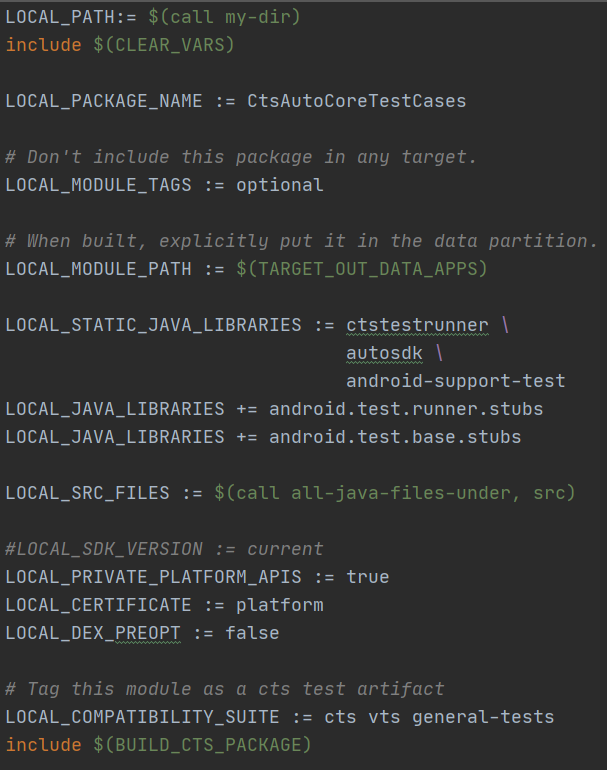
以中间件(AutoCore)为例，AndroidMainfest.xml文件完整内容如下：



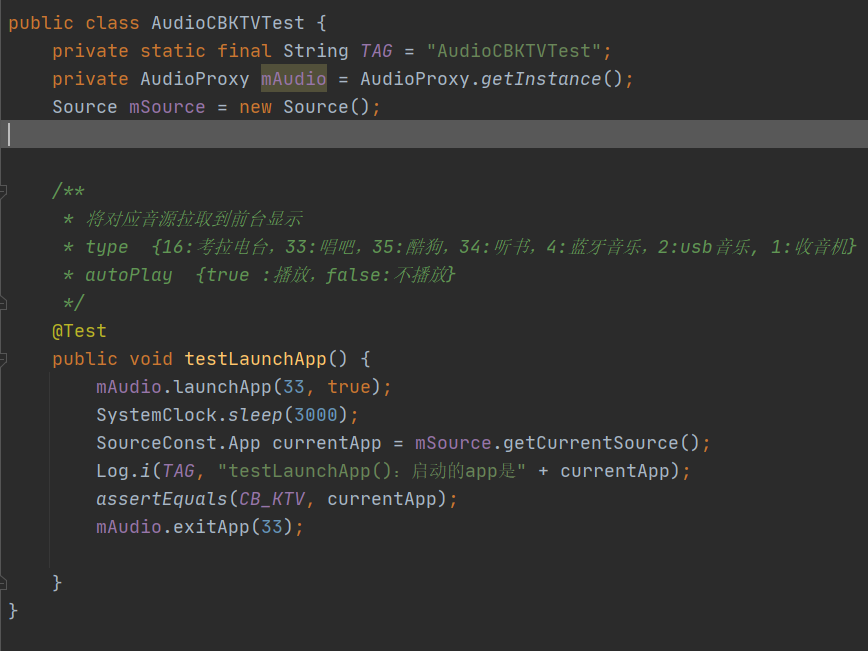
AndroidTest.xml内容如下：



Android.mk文件内容如下:



测试类大致这样写：



读者亦可参考源码路径cts/tests/tests下谷歌提供的大量用例，甚至可以用里面的案例复制粘贴，修改为自己的项目，在此基础上开发。