# Android 单元测试入门

**Start**

简单了解一下，如何依赖 Junit 进行 Java 代码的单元测试。依旧如何借助 Robolectric 进行 Android 方面的单元测试，主要是 Context 的获取。最后就网络请求的单元测试，简单叙述一下。

**JUnit**

* 依赖

**testImplementation** 'junit:junit:4.12'

* example

**public** **class** Tools {

**private** **static** final String TAG = "Tools";

**public** **static** String getCurrentTime() {

SimpleDateFormat simpleDateFormat = **new** SimpleDateFormat("yyyy年MM月dd日 HH:mm:ss", Locale.CHINA);

Date curDate = **new** Date(System.currentTimeMillis());

**return** simpleDateFormat.format(curDate);

}

**public** **static** String getCurrentTime(long tempStap) {

SimpleDateFormat simpleDateFormat = **new** SimpleDateFormat("yyyy年", Locale.CHINA);

Date curDate = **new** Date(tempStap);

**return** simpleDateFormat.format(curDate);

}

}

**public** **class** **ToolsUnitTest** {

@BeforeClass

**public** **static** **void** **setUp**() {

System.out.println( ToolsUnitTest.class.getSimpleName()+ "=====单元测试开始");

}

@AfterClass

**public** **static** **void** **end**() {

System.out.println( ToolsUnitTest.class.getSimpleName()+ "=====单元测结束");

}

@Test(expected = NullPointerException.class)

**public** **void** **getCurrentTimeTest**() {

assertNotEquals("1111",Tools.getCurrentTime());

// 起码在 2019 年，这条测试是可以通过的

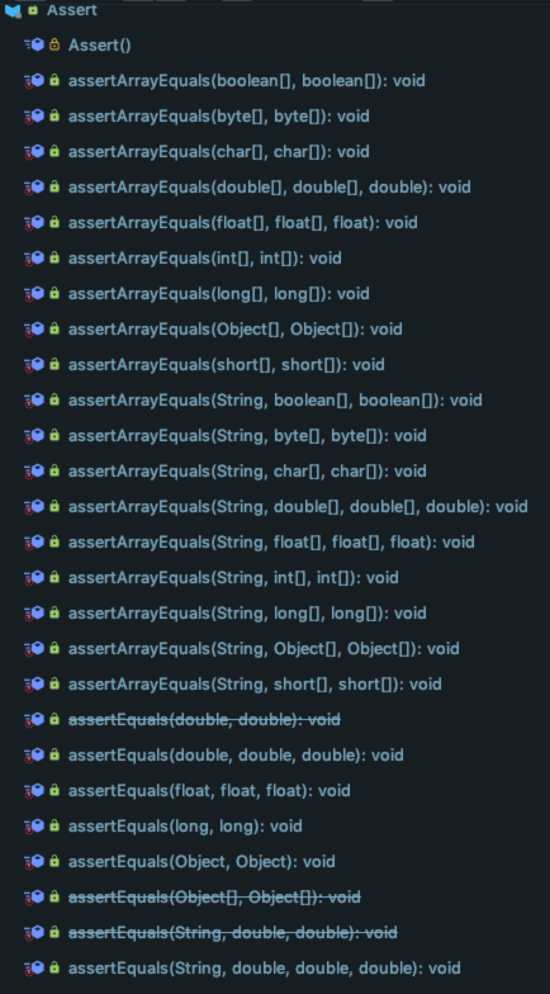
assertEquals("2019年",Tools.getCurrentTime(System.currentTimeMillis()));

}

}

执行顺序： @BeforeClass –> @Before –> @Test –> @After –> @AfterClass

* 可用 API



都是 Assert 的静态方法， **对有返回值的方法，用断言非常好用** ，你甚至可以测试异常

* 高级用法
  + @RunWith(Parameterized.class) 参数化
  + assertThat用法
  + @Rule用法
* 问题来了

**public** **static** **String** getAppVersion(Context mContext) { ... }

Android 中的 context 怎么搞 ？

**Robolectric**

* 配置
  + 依赖
  + testImplementation 'androidx.test:core:1.2.0'
  + testImplementation 'androidx.test:rules:1.2.0'
  + testImplementation 'androidx.test.espresso:espresso-core:3.2.0'
  + testImplementation 'org.robolectric:robolectric:4.3'
  + 允许 robolectric 读取 assets、resources 和 manifests，在 build.gradle 中添加

**一定要添加以下配置，否则将导致单元测试运行异常**

**一定要添加以下配置，否则将导致单元测试运行异常**

**一定要添加以下配置，否则将导致单元测试运行异常**

**testOptions** {

**unitTests** {

**includeAndroidResources** = true

}

}

* + 在 gradle.properties 中添加
  + android.enableUnitTestBinaryResources=**true**
* example
* @RunWith(RobolectricTestRunner::class)
* @Config(sdk = [27])
* **class** **RobolectricUnitTest** {
* @Test
* **fun** **assertContext**() {
* **val** context = ApplicationProvider.getApplicationContext<Context>()
* **val** version = AppUtils.getAppVersion(context)
* // 单元测试，也可以打印日志
* println("version ==$version")
* assertEquals("1.0", version)
* }
* **companion** **object** {
* @BeforeClass
* **fun** **setup**() {
* // for Fresco
* SoLoader.setInTestMode()
* }
* }
* }

单元测试也可以用 Kotlin 写，:grimacing::grimacing::grimacing:

**Context 是什么 ？**

@RunWith(RobolectricTestRunner.class)

@Config(sdk = 28, manifest = Config.NONE)

**public** **class** **ContextTest** {

@Test

**public** **void** **useContextTest**() **throws** Utils.SystemUtilsException {

Context context = ApplicationProvider.getApplicationContext();

System.out.println("context ===" + context.getClass().getName());

assertEquals(Constants.PACKAGE\_NAME, Utils.getPackageName(context));

assertEquals(Constants.PACKAGE\_VERSION, Utils.getPackageVersionName(context));

}

@Test

**public** **void** **screenInfoTest**() {

Context context = ApplicationProvider.getApplicationContext();

Resources resources = context.getResources();

DisplayMetrics displayMetrics = resources.getDisplayMetrics();

**float** density = displayMetrics.density;

**float** width = displayMetrics.widthPixels;

**float** height = displayMetrics.heightPixels;

System.out.println("density==" + density);

System.out.println("width ==" + width);

System.out.println("height ==" + height);

}

}

-output

density==1.0

width ==320.0

height ==470.0

* 可能遇到的问题

Application 中某些方法无法被初始化，比如 Fresco，详见

[***GitHub issue***](https://github.com/facebook/stetho/issues/440)

[***Fresco issue***](https://github.com/facebook/fresco/issues/2060)

及解决方案。

**Robolectric 很强大，可以在不使用真机及模拟器的情况下，进行 UI 测试**

Robolectric 可以认为是一个虚拟的模拟器

**网络测试**

网络测试有以下困难

* 请求结果是异步返回的，无法直接进行断言，需要进行同步转换
* 后端会做 cookie 和 headers 的校验，Robolectric 只是模拟器，请求时没有这些信息。   
  对 okhttp 添加 intercept ，拦截返回值，进行任意返回结果，测试后续逻辑。
* simple

@Test

**public** **void** **netTest**() {

**final** Retrofit mRetrofit = initRetrofit();

GankApi mGankApi = mRetrofit.create(GankApi.class);

Observable<GankAndroid> mAndroidObservable = mGankApi.getData("10/1");

mAndroidObservable.subscribe(gankAndroid -> {

doAssert(gankAndroid.getResults().get(0));

}, throwable -> System.out.println("fail"));

}

**private** **void** **doAssert**(GankAndroid.ResultsEntity entity) {

assertEquals("5d423ff19d2122031ea52264", entity.get\_id());

assertEquals("web", entity.getSource());

assertEquals("Android", entity.getType());

assertEquals("潇湘剑雨", entity.getWho());

}

这个就可以对网络请求回来的数据，进行各种业务逻辑的测试了。

但是也可以添加 interceptor ,创建特定的返回结果

* 添加 Interceptor 拦截返回结果，方便测试后续逻辑。

@Test

public **void** netWithInterceptTest() {

String json = "404";

SimpleIntercept errorIntercept = **new** SimpleIntercept(json, 404);

// SimpleIntercept successIntercept = new SimpleIntercept(json, HTTP\_OK);

**final** Retrofit mRetrofit = initRetrofit(errorIntercept);

GankApi mGankApi = mRetrofit.create(GankApi.**class**);

Observable<GankAndroid> mAndroidObservable = mGankApi.getData("10/1");

mAndroidObservable.subscribe(gankAndroid -> {

},

throwable -> {

// for example

assertEquals("HTTP 404 404", throwable.getMessage());

});

}

SimpleIntercept

**public** **class** **SimpleIntercept** **implements** **Interceptor** {

**private** **static** **final** String APPLICATION\_JSON = "application/json";

**private** String expectedResult;

**private** **int** code;

**public** **SimpleIntercept**(@NonNull String expectedResult, **int** code) {

**this**.expectedResult = expectedResult;

**this**.code = code;

}

@Override

**public** Response **intercept**(Chain chain) {

Response response = **new** Response.Builder()

.code(code)

.message(expectedResult)

.request(chain.request())

.protocol(Protocol.HTTP\_1\_1)

.body(ResponseBody.create(MediaType.parse(APPLICATION\_JSON), expectedResult))

.addHeader("content-type", APPLICATION\_JSON)

.build();

**return** response;

}

}