测试的分类

测试分类方式

A、代码了解程度：

a、 黑箱测试

b、白箱测试

B、按照测试维度（粒度）

a、单元测试: Java中单元就是类。

b、模块测试

c、集成测试

d、系统测试

C、按照测试力度

a、冒烟测试 ： 一种低强度、系统完整性的测试

b、压力测试：按照系统设计流量进行满负荷运行试验。

monkey测试

要配置好adb环境

在命令提示窗口里通过:命令ADB SHELL -----> monkey -p 应用包名 次数

android单元测试

1、建立工程写一个独立的目标待测试类

2、在AndroidManifest.xml清单里添加测试工具及要测试的包名。

在<uses-sdk/>下添加

```
<instrumentation  android:name="android.test.InstrumentationTestRunner"
      指定要测试的包   android:targetPackage="com.itheima.Tunit">
</instrumentation>
     在<application>
     <uses-library  android:name="android.test.runner"/>
   </application>
```

如图：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.itheima.Tunit"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk
        android:minSdkVersion="11"
        android:targetSdkVersion="17" />
    <instrumentation android:name="android.test.InstrumentationTestRunner"
        android:targetPackage="com.itheima.Tunit"></instrumentation>
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.itheima.Tunit.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <uses-library android:name="android.test.runner"/>
    </application>
```

测试工具包

要测试的包名

3、写一个测试类要继承AndroidTestCase类　与被测试类不要在同一包下

```
public class MyTest extends AndroidTestCase
```

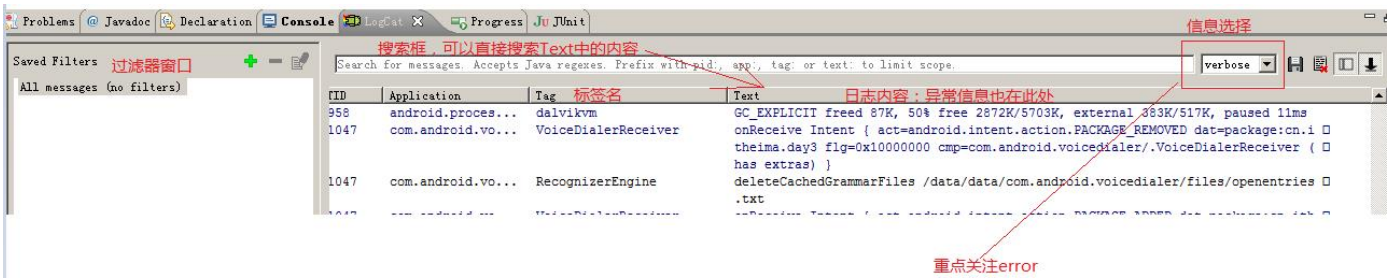
4、写一个测试方法，这个方法名最好用自己的名字来定义。

```
package sssss;

import android.test.AndroidTestCase;

public class MyTest extends AndroidTestCase{
    public void testAdd(){
        Test t = new Test();
        int add = t.add1(1, 1);
        assertEquals(2, add);
    }
}
```

这方法名尤其要注意

5、运行测试

LogCat日志工具



五种级别

1、V：verbose: 所有信息，包括溶于信息

2、D：　debug : 调试信息及其以上等级信息

3、I：info：　一般信息及其以上信息

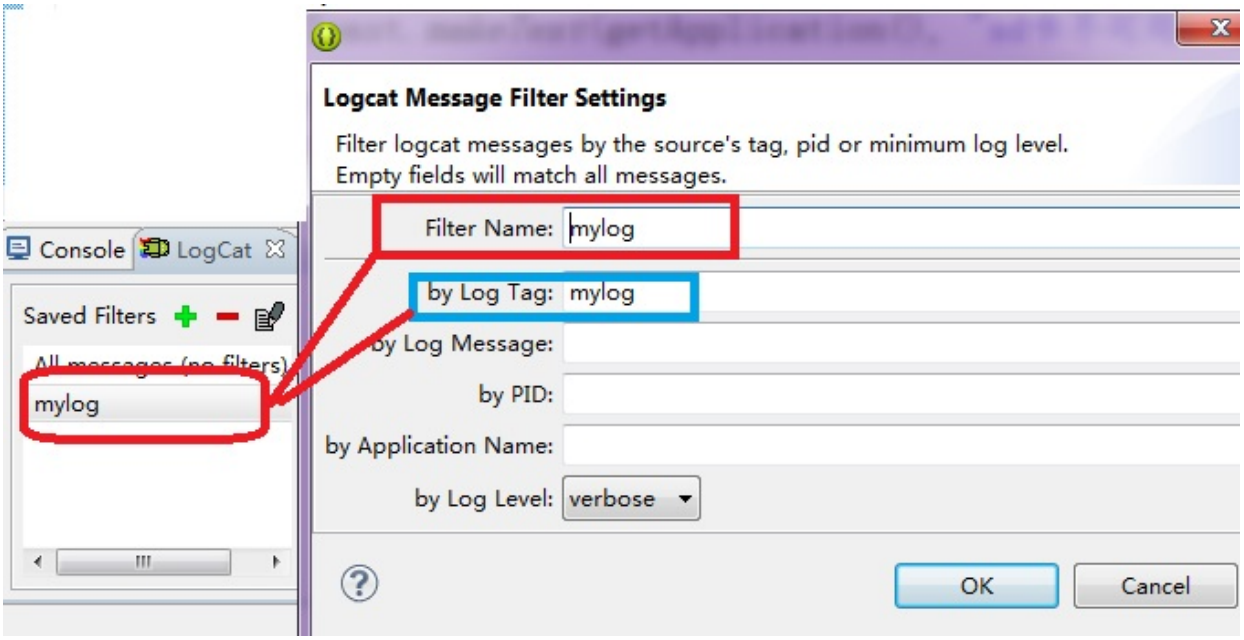
4、W：warn : 警告信息及其以上信息

5、 E：error:异常信息（红色，重点关注）

信息等级递增

Log类输出日志

1、通过在过滤器中添加信息过滤方式



2、代码中写入 Log.e("mylog", e.getMessage()); 输出e级错误信息

```
System.out.println("syso输出的信息");
Log.v("mylog", "v级信息");
Log.d("mylog", "d级信息");
Log.i("mylog", "i级信息");
Log.w("mylog", "w级信息");
try{
    System.out.println(1/0);
}catch (Exception e) {
    Log.e("mylog", "e级信息:" + e.getMessage());
}
```

标签，对于过滤器的tag

注意此处

Context上下文的概念

Android系统的角度来理解：Context是一个场景，代表与操作系统的交互的一种过程。从程序的角度上来理解：Context是个抽象类，而Activity、Service、Application等都是该类的一个实现。Activity、Service、Application都是继承自ContextWrapper，而ContextWrapper内部会包含一basecontext，由这个base context去实现了绝大多数的方法。

Context:上下文环境。

a，列举Context的作用

获取相关运行环境＼资源 ＼调度Activity ＼获取服务。

加载资源、启动一个新的Activity、获取系统服务、获取内部文件（夹）路径、创建View操作时等都需要Context的参与

b，说出Context和Acitivty之间的关系，继承关系

Activity、Service、Application都是Context的子类

内部存储数据的存储

1、内部存储空间的路径手写就是：

/data/data/应用包名/files/文件名.xxx;

2、利用上下文获取路径：

getApplication().getFileStreamPath("文件名.xxx")；加getPath()方法将路径文件转字符串

3、自动获取文件路径：

FileInputStream fis = openFileInput("文件名.xxx") 读出文件

FileOutputStream fos = openFileOutput("文件名.xxx", 文件的权限参数)；

写到文件里

文件的权限参数

0代表 文件私有权限 Context.MODE_PRIVATE 不可重写每次新建文件覆盖原来的内容

1代表 Context.MODE_WORLD_READABLE 已过时

2代表 Context.MODE_WORLD_WRITEABLE 已过时

Context.MODE_APPEND 这个的对应数值较大，建议手写MODE_APPEND 不会每次新建文件，写的内容可以追加写入

SD卡的数据存取读

SD卡路径：

外部SD卡的路径：/mnt/sdcard/文件名.xxx

1）File file = new File(Environment.getExternalStorageDirectory(), "cun123");

外部存储的私有路径：/mnt/sdcard/android/data/包名/files

2）File file = new File(this.getExternalFilesDir(Environment.DIRECTORY_DOWNLOADS), "cun123");

读写文件：

BufferedReader reader=new BufferedReader(new FileReader(file)); 读文件

BufferedWriter writer==new BufferedWriter(new FileWriter(file)); 写文件但不追加写入

BufferedWriter writer==new BufferedWriter(new FileWriter(file, true)); 追加写入文件

1、sd卡的文件路径手写：

"mnt/sdcard/文件名.xxx"; 模拟器默认的路径

2、通过外界环境去获取路径自动获取sd卡的文件路径：

File f = Environment.getExternalStorageDirectory(); 返回一个文件夹路径

String str = Environment.getExternalStorageDirectory().getPath()+"/文件名.xxx";

3、SD卡处于只读的候必须改变读写权限，模拟器的sdcard处于只读状态，进命令行进行操作

在命令提示窗口里通过:命令ADB SHELL -----> cd mnt ------> chmod 075 或者 shell下, mount -o remount rw/

4、在SD卡中进行文件的读写前需要添加一个外置设备文件操作权限

如里没有加权限会报一个 /mnt/sdcard/test3.txt: open failed: EACCES (Permission denied) 错误

添加步骤：AndroidManifest.xml清单文件---〉Permission ----〉uses-permission ----〉android.permission.WRITE_EXTERNAL_STORAGE

5、SD卡读写前还需要做一个SD卡是否挂载到系统的判断（明明安装了SD卡但由于某种原因系统并为识别到SD卡的存在，如果不做判断可能导致应用异常崩溃）

判断的方法：`if (Environment.MEDIA_MOUNTED.equals(Environment.getExternalStorageState())) {进行写入或读取操作}`

如果外挂设备安装与外挂设置状态相同那么就可以读写SD卡文件否则提示SD卡不可用。

6、SD卡内存大小的获取，获取数据用于判断下载文件是否能装入SD卡中

获取SD卡所有文件夹：

file1 = Environment.getExternalStorageDirectory(); 外部环境.外部存储目录()

SD卡总大小 long space = file1.getTotalSpace(); 获取的是字节数

SD卡可用空间大小 long space2 = file1.getFreeSpace();

通过Formatter.formatFileSize(getBaseContext(), space)方法可以获取对应的GB/MB数

txt.setText("SD卡总大小"+Formatter.formatFileSize(this, space)+"sd卡可用"+Formatter.formatFileSize(this, space2)); 将获取的SD卡存储容量返回到文本框中

Android中文件权限的作用：权限用于限制、禁止其他应用读取我们的应用下的文件，除非应用主动开发读写权限Android极其重视用户隐私，所有对于文件操作等可能暴露用户隐私的风险操作，默认禁止。

SharedPreference类　通过键值对的形式存取配置文件

作用:主要用于读取保存用户信息,以XML形式存储,

默认存储在:/data/data/应用包名/Sared_Prefs/文件名.xml;

haredPreferences对象本身只能获取数据而不支持存储和修改，存储修改是通过Editor对象实现。实现SharedPreferences存储的步骤如下：

一、根据Context获取SharedPreferences对象　二、利用edit()方法获取Editor对象。

三、通过Editor对象存储key-value键值对数据。　四、通过commit()方法提交数据。

获取SharedPreferences的两种方式:

调用Context对象的getSharedPreferences()方法　调用Activity对象的getPreferences()方法

两种方式的区别:

调用Context对象的getSharedPreferences()方法获得的SharedPreferences对象可以被同一应用程序下的其他组件共享.调用Activity对象的getPreferences()方法获得的SharedPreferences对象只能在该Activity中使用.

haredPreferences的四种操作模式:

MODE_PRIVATE　MODE_APPEND　MODE_WORLD_READABLE　MODE_WORLD_WRITEABLE

MODE_PRIVATE为默认操作模式,代表该文件是私有数据,只能被应用本身访问,在该模式下,写入的内容会覆盖原文件的内容　MODE_APPEND：模式会检查文件是否存在,存在就往文件追加内容,否则就创建新文件.　MODE_WORLD_READABLE和MODE_WORLD_WRITEABLE用来控制其他应用是否有权限读写该文件MODE_WORLD_READABLE：表示当前文件可以被其他应用读取.MODE_WORLD_WRITEABLE：表示当前文件可以被其他应用写入.

将用户信息保存到XML文件中：　通过Editor类实现写入到文件的操作

```
SharedPreferences  p  =  getSharedPreferences("info",  0);
      Editor  e  =  p.edit();
      e.putString("name"  ,   "张三"); 以键值对存入
      e.putString("mima"  ,   1234585);
      e.commit();
```

从XML文件中获取用户信息:获取的方式是键值对方式根据键来获得值

```
SharedPreferences  sp  =  getSharedPreferences("infox",  0);
            sp.getString("name",  ""));
            sp.getString("mima",  ""));
```

关于EditText组件上内容的获取和设置:

UI组件ID.getText().toString(); 获取内容

UI组件ID.setText(""); 设置内容

XML文件的生成和解析

xml文件通过XmlSerializer生成：XmlSerializer将数据序列化到xml文件，该类是Android提供的工具

```
//1、获取XML文件生成工具对象
XmlSerializer parser = Xml.newSerializer();
try{
//2、获取通过openFileOutput（）方法输入流，第一个参数为文件名，第二个参数为文件权限
//生成文件路径:    /date/date/应用包名/files/文件名
FileOutputStream fis = openFileOutput("xinxi", MODE_APPEND);
parser.setOutput(fis, "utf-8");
//是null的原因：这个参数是标签的命名空间Namespaces解决相同文件相同命名的标签冲突
//只有一个绝对唯一的文件存在的时间可以不给命名空间参数直接给个null
//通过工具对象调用开始文档startDocument(null,true)，结束文档endDocument();
//开始标签startTag（null,"标签名"）  结束标签endTag(null, "标签名"); 成对出现添加
parser.startDocument(null,true);
parser.startTag(null, "users");
parser.startTag(null, "user");
parser.startTag(null, "name");
parser.text("xiaoming");
parser.endTag(null, "name");
parser.startTag(null, "mima");
parser.text("123456789");
parser.endTag(null, "mima");
parser.startTag(null, "id");
parser.text("110120119");
parser.endTag(null, "id");
parser.endTag(null, "user");
parser.startTag(null, "user");
parser.startTag(null, "name");
```

```
        parser.text("xiaoming");
        parser.endTag(null, "name");
        parser.startTag(null, "mima");
        parser.text("123456789");
        parser.endTag(null, "mima");
        parser.startTag(null, "id");
        parser.text("110120119");
        parser.endTag(null, "id");
        parser.endTag(null, "user");
        parser.endTag(null, "users");
        parser.endDocument();
        //最后关闭流
        fis.close();
        fag = true;
        }catch (Exception e) {
            Log.e("mylog",e.getMessage());
        }
```

使用XmlPullParser解析xml文件：　通过XML类解析

```
        //1、获取解析器对象，通过XML类.newPullParser()方法
        XmlPullParser parser = Xml.newPullParser();
        StringBuffer sb = new StringBuffer();    //同物取出拿到的数据
        try{
//2、通过解析器对象.setInput（）方法获取输出流对象，如果用字节流需要指定字符编码，如果是用字节流可以不用指定编码
        parser.setInput(openFileInput("xinxi"), "utf-8");
//3、通过解析对象.getEventType()方法，获取文档事件类型。
        int eventType = parser.getEventType();
        int n = 1;
        //4、通过循环获取数据，条件：如果事件不是文档结束
        while(eventType!=parser.END_DOCUMENT){
            //5、如果 事件是标签开始
            if(eventType==parser.START_TAG){
                //6、如果根标签名与解析器对象获取的标签名相同  就获取到第一判断第一组标签名
                if("user".equals(parser.getName())){
                    sb.append("第"+n+"个用户信息："+"\n");
                    n++;
        //7、如果根标签下的某个标签与获取的标签名相同，就通过解析器对象.nextText()方法获取该标签里的数据
                }if("name".equals(parser.getName())){
                    sb.append("用户名："+parser.nextText()+"\n");
                }
                if("mima".equals(parser.getName())){
                    sb.append("密码："+parser.nextText()+"\n");
                }


                if("id".equals(parser.getName())){
                    sb.append("ID："+parser.nextText()+"\n");
                }
            }
            //最后让事件继续向下一个走。
            eventType = parser.next();
        }
        }catch (Exception e) {
            Log.e("mylog", e.getMessage());
        }
        text.setText(sb.toString());
```

toast　用法：Toast.makeText(this, "已经写入成功请不要重复写入", 0).show();

Android系统中一种消息框类型

当视图显示给用户，在应用程序中显示为浮动。和Dialog不一样的是，它永远不会获得焦点，无法被点击。用户将可能是在中间键入别的东西。Toast类的思想就是尽可能不引人注意，同时还向用户显示信息，希望他们看到。而且Toast显示的时间有限，Toast会根据用户设置的显示时间后自动消失。