宝贵建议请发送至：wangzhenyang@itcast.cn

黑马程序员 itheima.com -编程，始于黑马

Android 课程同步笔记

Alpha 0.01 版

Android-基础功能

1.Android Junit

1.1 常见的测试

在介绍 Android Junit 前先介绍一下常见的测试分类。

- 根据是否知道源码分类

  黑盒测试：不知道源码，是以用户的角度，从输入数据与输出数据的对应关系出发进行测试的。

  白盒测试：知道源码,又称结构测试、透明盒测试、逻辑驱动测试或基于代码的测试。

- 根据测试粒度分类

  方法测试： FunctionTest，粒度最低，测试单个方法

  单元测试： JunitTest 方法里面会调用其他的方法

  联调测试： IntegrationTest 后台与前台集成，各模块之间的集成测试

- 根据测试次数分类：

  冒烟测试：顾名思义，把设备点冒烟的，随机的去点 Android 下提供给我们一种冒烟测试的功能：猴子测试，在命令行输入 adb shell ,进入 Linux 内核

  测试整个系统:monkey 1000(事件的数量)

  测试某个程序：monkey -p 包名事件的数量

  压力测试： PressureTest ,给后台用的,主体向被观察者布置一定量任务和作业，借以观察个体完成任务的行为。

1.2Android Junit

在实际开发中，开发 android 软件的过程需要不断地进行测试。而使用 Junit 测试框架，则是正规 Android 开发的必用技术，在 Junit 中可以得到组件，可以模拟发送事件和检测程序处理的正确性。

下面以一个 Android 为例来介绍 Android Junit 的使用方法。

1 可以新建一个 Android Application Project 也可以直接用存在的工程。

2 在工程中新建一个类 MyTest 继承 android.test.AndroidTestCase

在该类中可以写测试方法，跟普通 java 工程使用 junit 类似。执行的时候右击要执行的方法名->Run as->Android Junit Test 。

3 在 Android 清单文件中追加 2 条信息，如下图高亮部分。

Tips :`android:targetPackage` 的值可以是任意项目的任意包名，而 `instrumentation` 标签中的 `android:name` 属性值是由测试框架提供的，值必须为 *`android.test.InstrumentationTestRunner`*。同样的 `uses-library` 属性的值也是由测试框架提供的，必须为 *`android.test.runner`*。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.itheima.junit"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="18" />
    <instrumentation
        android:targetPackage="com.itheima.junit"
        android:name="android.test.InstrumentationTestRunner"
        ></instrumentation>
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <uses-library android:name="android.test.runner"/>
        <activity
            android:name="com.itheima.junit.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>
    </application>

</manifest>
```

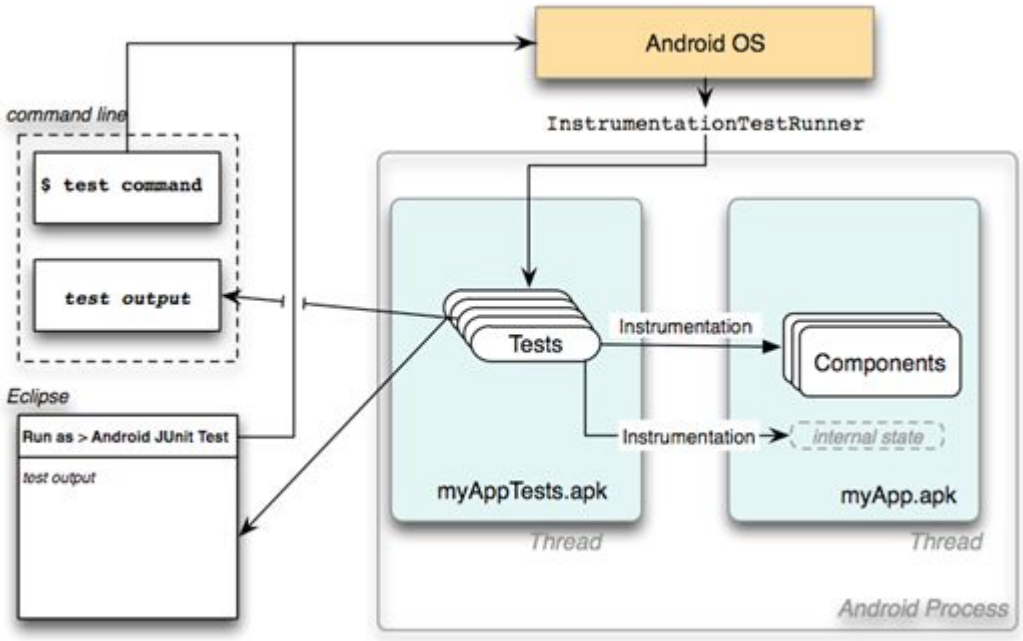
1.3Android Junit 拓展知识

Tips：拓展知识并不是必须要求掌握的，心有余力之时可以作为进一步提升的参考。

Android 测试环境的核心是一个 Instrumentation 框架，在这个框架下，你的测试应

用程序可以精确控制应用程序。使用 Instrumentation， 你可以在主程序启动之前，创建模拟的系统对象，如 Context；控制应用程序的多个生命周期；发送 UI 事件给应用程序；在执行期间检查程序状态。 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<instrumentation>元素来指定要测试的应用程序。这个元素的特性指明了要测试的应用程序包名，以及告诉 Android 如何运行测试程序。下面的图片概要的描述了 Android 的测试环境：



在 Android 中，测试程序也是 Android 程序，因此，它和被测试程序的书写方式有很多相同的地方。SDK 工具能帮助你同时创建主程序工程及它的测试工程。你可以通过 Eclipse 的 ADT 插件或者命令行来运行 Android 测试。Eclipse 的 ADT 提供了大量的工具来创建测试用例，运行以及查看结果。

Android 提供了基于 JUnit 测试框架的测试 API 来书写测试用例和测试程序。另外，Android 还提供了强大的 Instrumentation 框架，允许测试用例访问程序的状态及运行时对象。

1.3.1 Android Junit 中的主要核心 API

JUnit TestCase 类

继承自 JUnit 的 TestCase，不能使用 Instrumentation 框架。但这些类包含访问系统对象（如 Context）的方法。使用 Context，你可以浏览资源，文件，数据库等等。基类是 AndroidTestCase，一般常见的是它的子类，和特定组件关联。

子类有：

1、ApplicationTestCase——测试整个应用程序的类。它允许你注入一个模拟的 Context 到应用程序中，在应用程序启动之前初始化测试参数，并在应用程序结束之后销毁之前检查应用程序。

2、 ProviderTestCase2——测试单个 ContentProvider 的类。因为它要求使用 MockContentResolver，并注入一个 IsolatedContext，因此 Provider 的测试是与 OS 孤立的。

3、ServiceTestCase——测试单个 Service 的类。你可以注入一个模拟的 Context 或模拟的 Application（或者两者），或者让 Android 为你提供 Context 和 MockApplication。

Instrumentation TestCase 类

继承自 JUnit TestCase 类，并可以使用 Instrumentation 框架，用于测试 Activity。使用 Instrumentation，Android 可 以向程序发送事件来自动进行 UI 测试，并可以精确控制 Activity 的启动，监测 Activity 生命周期的状态。

基类是 InstrumentationTestCase。它的所有子类都能发送按键或触摸事件给 UI。子类还可以注入一个模拟的 Intent。

子类有：

1、ActivityTestCase——Activity 测试类的基类。

2、SingleLaunchActivityTestCase——测试单个 Activity 的类。它能触发一次 setup()和 tearDown()，而不是每个方法调用时都触发。如果你的测试方法都是针对同一个 Activity 的话，那就使用它吧。

3、SyncBaseInstrumentation——测试 Content Provider 同步性的类。它使用 Instrumentation 在启动测试同步性之前取消已经存在的同步对象。

4、ActivityUnitTestCase——对单个 Activity 进行单一测试的类。使用它，你可以注入模拟的 Context 或 Application，或者两者。它用于对 Activity 进行单元测试。

不同于其它的 Instrumentation 类，这个测试类不能注入模拟的 Intent。

5、ActivityInstrumentationTestCase2——在正常的系统环境中测试单个 Activity 的类。你不能注入一个模拟的 Context，但你可以注入一个模拟的 Intent。另外，你还可以在 UI 线程（应用程序的主线程）运行测试方法，并且可以给应用程序 UI 发送按键及触 摸事件。

Assert 类

Android 还继承了 JUnit 的 Assert 类，其中，有两个子类，MoreAsserts 和 ViewAsserts。

1、MoreAsserts 类包含更多强大的断言方法，如 assertContainsRegex(String, String)，可以作正则表达式的匹配。

2、ViewAsserts 类包含关于 Android View 的有用断言方法，如 assertHasScreenCoordinates(View, View, int, int) ,可以测试 View 在可视区域的特定 X、Y 位置。这些 Assert 简化了 UI 中几何图形和对齐方式的测试。

Mock 对象类

Android 有一些类可以方便的创建模拟的系统对象，如 Application，Context，Content Resolver 和 Resource。Android 还在一些测试类中提供了一些方法来创建模拟的 Intent。因为这些模拟的对象比实际对象更容易使 用，因此，使用它们能简化依赖注入。你可以在 android.test 和 android.test.mock 中找到这些类。

它们是：

1、IsolatedContext——模拟一个 Context，这样应用程序可以孤立运行。与此同时，还有大量的代码帮助我们完成与 Context 的通信。这个类在单元测试时很有用。

2、RenamingDelegatingContext——当修改默认的文件和数据库名时，可以委托大多数的函数到一个存在的、常规的 Context 上。使用这个类来测试文件和数据库与正常的系统 Context 之间的操作。

3、MockApplication,MockContentResolver,MockContext,MockDialogInterface,MockPackageManager,MockResources ——创建模拟的系统对象的类。它们只暴露那些对对象

的管理有用的方法。这些方法的默认实现只是抛出异常。你需要继承这些类并重写这些方法。

Instrumentation TestRunner

Android 提供了自定义的运行测试用例的类，叫做 InstrumentationTestRunner。这个类控制应用程序处于测试环境中，在同一个进程中运行测试 程序和主程序，并且将测试结果输出到合适的地方。IntrumentationTestRunner 在运行时对整个测试环境的控制能力的关键是使用 Instrumentation。注意，如果你的测试类不使用 Instrumentation 的话，你也可以使用这个 TestRunner。

当你运行一个测试程序时，首先会运行一个系统工具叫做 Activity Manager。Activity Manager 使用 Instrumentation 框架来启动和控制 TestRunner，这个 TestRunner 反过来又使用 Intrumentation 来关闭任何主程序的实例，然后启动测试程序及主程序（同一个进程中）。这就能确保测试程序与主程序间的直接交互。

1.3.2 在测试环境中工作

对 Android 程序的测试都包含在一个测试程序里，它本身也是一个 Android 应用程序。测试程序以单独的 Android 工程存在，与正常的 Android 程序有着相同的文件和文件夹。测试工程通过在 manifest 文件中指定要测试的应用程序。

每个测试程序包含一个或多个针对特定类型组件的测试用例。测试用例里定义了测试应用程序某些部分的测试方法。当你运行测试程序，Android 会在相同进程里加载主程序，然后触发每个测试用例里的测试方法。

1.3.3 测试工程

为了开始对一个 Android 程序测试，你需要使用 Android 工具创建一个测试工程。工具会创建工程文件夹、文件和所需的子文件夹。工具还会创建一个 manifest 文件，指定被测试的应用程序。

1.3.4 测试用例

一个测试程序包含一个或多个测试用例，它们都继承自 Android TestCase 类。选择一个测试用例类取决于你要测试的 Android 组件的类型以及你要做什么样的测试。一个测试程序可以测试不同的组件，但每个测试用例类设计时只能测试单一类型的组件。

一些 Android 组件有多个关联的测试用例类。在这种情况下，在可选择的类间，你需要判断你要进行的测试类型。例如，对于 Activity 来说，你有两个选择，ActivityInstrumentationTestCase2 和 ActivityUnitTestCase。

ActivityInstrumentationTestCase2 设计用于进行一些功能性的测试，因此，它在一个正常的系统环境中测试 Activity。你可以注入模拟的 Intent，但不能是模拟的 Context。一般来说，你不能模拟 Activity 间的依赖关系。相比而言，ActivityUnitTestCase 设计用于单元测试，因此，它在一个孤立的系统环境中测试 Activity。换句话说，当你使用这个测试类时，Activity 不能与其它 Activity 交互。

作为一个经验法则，如果你想测试 Activity 与 Android 的交互的话，使用 ActivityInstrumentationTestCase2。如果你想对一个 Activity 做回归测试的话，使用 ActivityUnitTestCase。

1.3.5 测试方法

每个测试用例类提供了可以建立测试环境和控制应用程序的方法。例如，所有的测试用例类都提供了 JUnit 的 setUp()方法来搭建测试环境。另外，你可以添加方法来定义单独的测试。当你运行测试程序时，每个添加的方法都会运行一次。如果你重写了 setUp()方法，它会在每个方法运行前运行。相似 的，tearDown()方法会在每个方法之后运行。

测试用例类提供了大量的对组件启动和停止控制的方法。由于这个原因，在运行测试之前，你需要明确告诉 Android 启动一个组件。例如，你可以使用 getActivity()来启动一个 Activity。在整个测试用例期间，你只能调 用这个方法一次，或者每个测试方法一次。甚至你可以在单个测试方法中，调用它的 finishing()来销毁 Activity，然后再调用 getActivity()重新启动一个。

1.3.6 运行测试并查看结果

编译完测试工程后，你就可以使用系统工具 Activity Manager 来运行测试程序。你给 Activity Manager 提供了 TestRunner 的名（一般是 InstrumentationTestRunner，在程序中指定）；名包括被测试程序的包名和 TestRunner 的名。Activity Manager 加载并启动你的测试程序，杀死主程序的任何实例，然后在测试程序的同一个进程里加载主程序，然后传递测试程序的第一个测试用例。这个时 候，TestRunner 会接管这些测试用例，运行里面的每个测试方法，直到所有的方法运行结束。

如果你使用 Eclipse，结果会在 JUnit 的面板中显示。如果你使用命令行，将输出到 STDOUT 上。

1.3.7 测试什么？

除了一些功能测试外，这里还有一些你应该考虑测试的内容：

- Activity 生命周期事件：你应该测试 Activity 处理生命周期事件的正确性。例如，一个 Activity 应该在 pause 或 destroy 事件 时保存它的状态。记住一点的是屏幕方向的改变也会引发当前 Activity 销毁，因此，你需要测试这种偶然情况确保不会丢失应用程序状态。
- 数据库操作：你应该确保数据库操作能正确处理应用程序状态的变化。使用 android.test.mock 中的模拟对象。
- 屏幕大小和分辨率：在发布程序之前，确保在所有要运行的屏幕大小和分辨率上测试通过。你可以使用 AVD 来测试，或者使用真实的目标设备进行测试。

1.3.8 UI 线程中测试

Activity 运行在程序的 UI 线程里。一旦 UI 初始化后，例如在 Activity 的 onCreate()方法后，所有与 UI 的交互都必须运行在 UI 线程里。当你正常运行程序时，它有权限可以访问这个线程，并且不会出现什么特别的事情。当你运行测试程序时，这一点发生了变化。在带有 instrumentation 的类里，你可以触发方法在 UI 线程里运行。其它的测试用例类不允许这么做。为了一个完整的测试方法都在 UI 线程里运行，你可以使用@UIThreadTest 来声明线程。注意，这将会在 UI 线程里运行方法里所有的语句。不与 UI 交 互的方法不允许这么做；例如，你不能触发 Instrumentation.waitForIdleSync()。

如果让方法中的一部分代码运行在 UI 线程的话，创建一个匿名的 Runnable 对象，把代码放到 run()方法中，然后把这个对象传递给 appActivity.runOnUiThread()，在这里，

appActivity 就是你要测试的 app 对象。

2.Android 中的 LogCat 和 Log

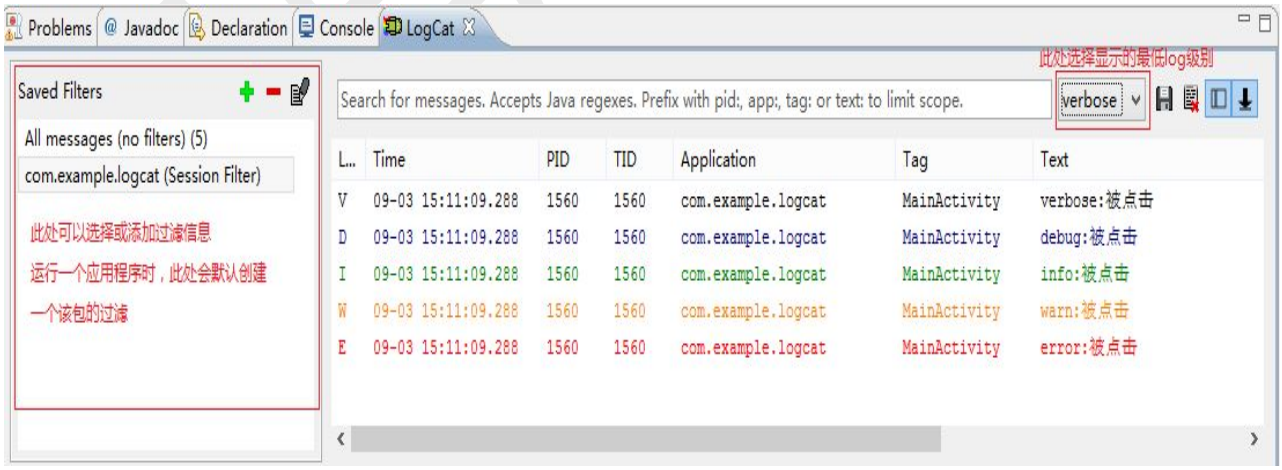
Android 的 Logcat 用于显示系统的调试信息，Log 是 **android.util.Log** 包下的类，用于日志的输出。下面内容将分别介绍 LogCat 和 Log 的使用，同时补充了在 Android 系统中调用 LogCat 日志的相关知识作为拓展内容。

2.1 LogCat 的使用

Android LogCat 的获取有两种方式：1、DDMS 提供的 LogCat 视图 2、通过 adb 命令行

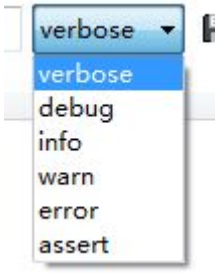
DDMS 提供的 LogCat 视图如下

如果该视图没有打开，点击 window->show view->other->android->Logcat 来进行选择。



视图的左侧可以选择或者添加过滤信息，运行一个应用程序时，此处会默认创建一个该包的过滤。视图的右上角区域用于选择 LogCat 的 log 级别，共有 verbose、debug、info、

warn、error、assert6 个可选项。如图所示：



。

该视图的主体部分是 log 的详细信息，包括错误级别（Level）、时间（Time）、进程 ID（PID）、线程 ID（TID）、应用程序包名（Application）、标签（Tag）、日志正文（Text）。

Tips：其中的 TID 并不等同于 Java 中的 `Thread.currentThread().getId()`,而是我们 Linux 中的 Thread ID，跟 PID 相同。

重要提示：Level 的错误级别有 V（verbose）、D(debug)、I（info）、W（warn）、E（error）。

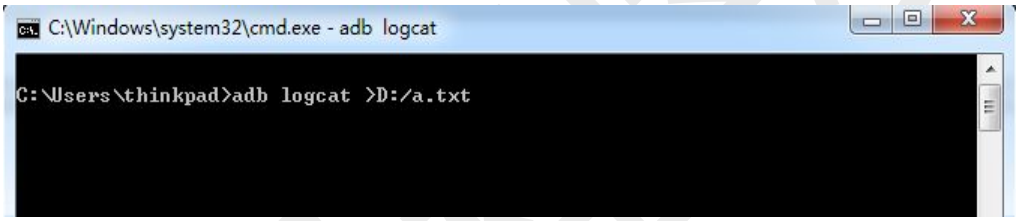
错误级别	意义	严重等级	显示颜色
verbose	所有信息都显示	最低	黑色
debug	显示调试信息	第四严重	蓝色
info	普通信息	第三严重	绿色
warn	警告信息	第二严重	黄色
error	错误信息	最高	红色

通过命令行调用 LogCat

1、将 LogCat 信息显示在控制台中

在控制台中输入 adb logcat 然后按回车键即可看到 LogCat 信息，如果需要终止按 Ctrl+C 键即可。

2、将 LogCat 信息保存在文件中



执行 adb logcat >D:/a.txt 则将日志输出到 D:/a.txt 文件中。按 Ctrl+C 键终止日志的输出。

Tips：上面介绍的只是 adb logcat 命令的最简单用法，其实该命令还有多种可选参数供选择，这里就不再详细说明。

2.2 Log 的使用

android.util.Log 常用的方法有以下 **5** 个：**Log.v() Log.d() Log.i()** Log.w() 以及 **Log.e()** 。根据首字母对应 **VERBOSE**，**DEBUG**,INFO, WARN，ERROR。

1、Log.v 的调试颜色为**黑色**的，任何消息都会输出，这里的 v 代表 verbose 啰嗦的意思，平时使用就是 Log.v("","");

2、Log.d 的输出颜色是**蓝色**的，仅输出 debug 调试的意思，但他会输出上层的信息，过滤起来可以通过 DDMS 的 Logcat 标签来选择.

3、Log.i 的输出为**绿色**，一般提示性的消息 information，它不会输出 Log.v 和 Log.d 的信息，但会显示 i、w 和 e 的信息

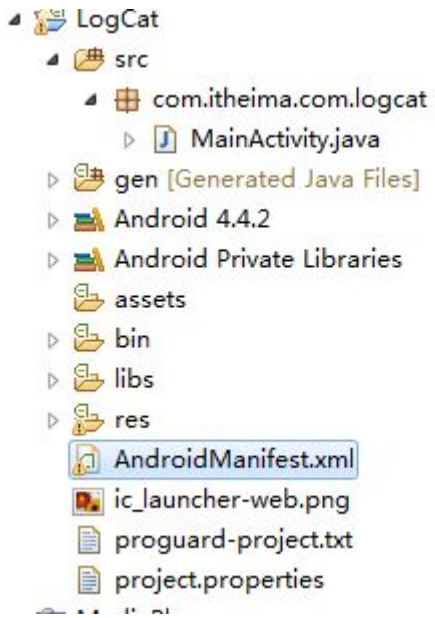
4、Log.w 的意思为**橙色**，可以看作为 warning 警告，一般需要我们注意优化 Android 代码，同时选择它后还会输出 Log.e 的信息。

5、Log.e 为红色，可以想到 error 错误，这里仅显示红色的错误信息，这些错误就需要我们认真的分析，查看栈的信息了。

2.3Android 程序获取 LogCat 信息

下面通过创建一个 Android 工程来演示如何在代码中实时获取 LogCat 信息。

1 创建一个新工程，这里工程名为 LogCat



在这个工程中使用默认的 MainActivity.java 类和默认的布局文件。

2 修改布局文件

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity" >
   <Button
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="获取 LogCat"
       android:id="@+id/bt_click"
       />
   <TextView
       android:id="@+id/tv_show"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:hint="显示日志"
       />

</LinearLayout>
```

3 修改 MainActivity.java 代码

代码第一部分：

```
package com.itheima.com.logcat;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import android.app.Activity;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends Activity {
    protected static final String TAG = "MyTest";
    private Button btn;
    private TextView tv_show;
    private Handler handler = new Handler() {
        public void handleMessage(android.os.Message msg) {
            tv_show.setText((String) msg.obj);
        };
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        btn = (Button) findViewById(R.id.bt_click);
        tv_show = (TextView) findViewById(R.id.tv_show);
```

代码第二部分：

```
btn.setOnClickListener(new View.OnClickListener() {

        @Override
        public void onClick(View v) {
            Log.v(TAG, "这是 verbose 信息");
            Log.d(TAG, "这是 debug 信息");
            Log.i(TAG, "这是 info 信息");
            Log.w(TAG, "这是 warn 信息");
            Log.e(TAG, "这是 error 信息");

            /** 开启线程用于监听 log 输出的信息 **/
            new Thread(new Runnable() {

                @Override
                public void run() {

                    Process mLogcatProc = null;
                    BufferedReader reader = null;
                    try {
                        /*
                         * 通过执行命令行获取 LogCat 信息
                         */
                        mLogcatProc = Runtime.getRuntime().exec(new
String[] { "logcat", TAG + ":v *:s" });
                        /*
                         * 获取进程输出流对象
                         */
                        reader = new BufferedReader(new
InputStreamReader(mLogcatProc.getInputStream()));
                        String line = null;
                        StringBuilder sb = new StringBuilder();
```

代码第三部分（完）：

```
while ((line = reader.readLine()) != null) {
                            sb.append(line);
                            Message msg = Message.obtain();
                            msg.obj = sb.toString();
                            handler.sendMessage(msg);
                        }
                        reader.close();
                        /*
                         * 通过发送消息，通知主线程修改 TextView 对象
                         * 因此这个操作是在子线程中进行的，而 Android 应用中子线程是无法修改 UI（UI 的修改操作必须在
                         * 主线程中进行
                         * ，因此 Android 提供了 Handler 机制，让子线程发送消息给主线程，然后由主线程修改 UI）.
                         */

                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
            }).start();
        }
    });
  }

}
```

4 将项目运行在模拟器上，并点击按钮

运行结果如图所示：

3.Android 系统中文件的权限

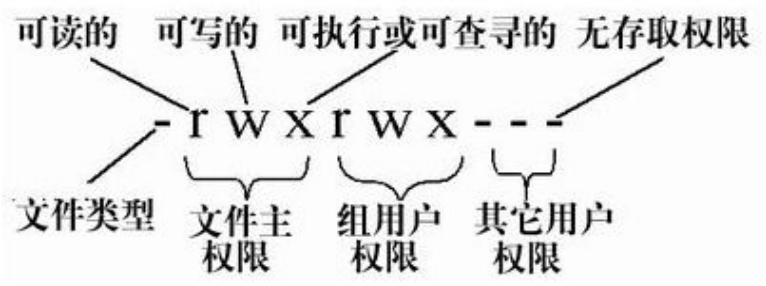
安卓是基于 linux 开发的 ,因此介绍 android 系统文件的权限 ,需要从介绍 linux 说起。

linux 系统权限简介：

一个文件一共有三个组别：用户、群组、其它

其中每个组包含三种权限：读 r、写 w、执行 x

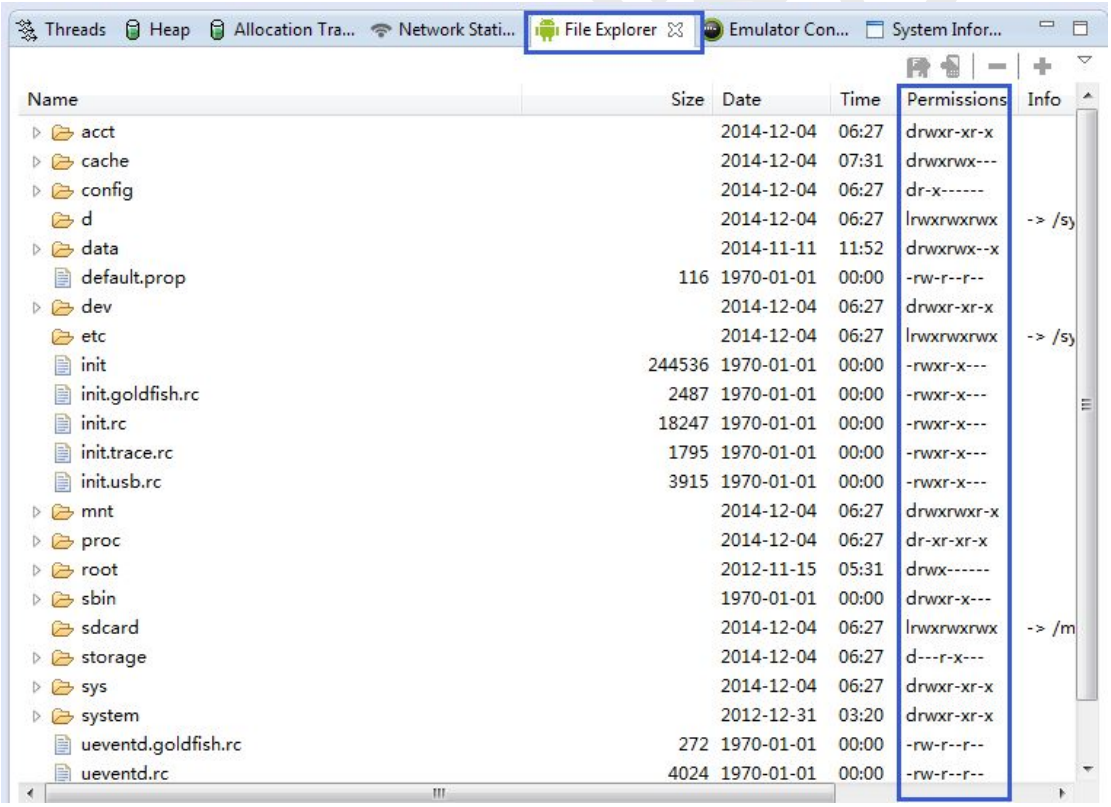
也就是说一个文件共有 9 个权限属性。



从左往右一到三位是 [用户] ，四到六位是 [群组] ，七到九位是 [其它]

举例 通过 DDMS 的 File Explorer 查看文件信息 ,可以看到某个文件的权限为 :rw-r--rwx

他的意思就是 [用户] 对其享有读写权限， [群组] 享有读权限， [其它] 享有读写执行权限。



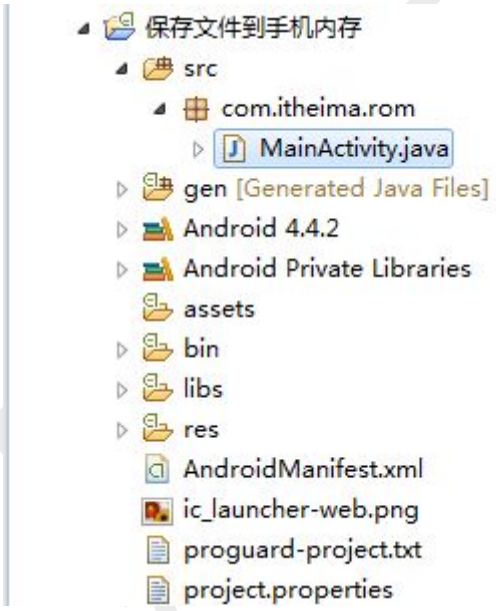
4.Android 下的文件存储

在 Android 系统中我们常用的数据存储方式有 4 种。1、存储中手机内存中（ROM）2、存储在 SD 卡中 3、存储在 SharedPreferences 中 4、存储在 SQLite 数据库中。在本文档中只介绍前 3 种数据存储方式，而 SQLite 将在下一篇中做详细说明。

4.1 保存文件到手机内存

通过一个模拟用户登录的案例来介绍如何将文件保存到手机内存中。

1 新创建一个 Android 工程，如图。



2 使用默认的布局文件盒默认的 Activity。修改布局文件。

在该布局文件中采用了 LinearLayout 布局。

布局文件第一部分：

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity" >
   <EditText
       android:layout_marginTop="10dp"
       android:layout_height="wrap_content"
       android:layout_width="match_parent"
       android:hint="请输入用户名"
       android:id="@+id/et_username"
       />
   <EditText
       android:layout_marginTop="10dp"
       android:layout_height="wrap_content"
       android:layout_width="match_parent"
       android:hint="请输入密码"
       android:inputType="textPassword"
       android:id="@+id/et_pwd"
       />
```

Tips： 属性说明，在以后的文档中对新出现的属性都会进行详细介绍，而已经使用过的属性则不再重复介绍。

- 上面布局文件中的 android:orientation 属性在 LinearLayout 布局中必须指定，有两个可选项：vertical 和 horizontal，分别代表垂直布局和水平布局。
- android:layout_marginTop="10dp" 代表该组件头部距离上一个组件的间隔为 10dp。

布局文件第二部分（完）：

```
<LinearLayout
        android:layout_marginTop="10dp"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:orientation="horizontal"
        android:gravity="right"
        >
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="登陆"
            android:onClick="login"
            />
        <CheckBox
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="保存密码"
            android:layout_marginRight="10dp"
            android:id="@+id/cb"
            />

    </LinearLayout>
</LinearLayout>
```

Tips：上面的 LinearLayout 布局中嵌套了 LinearLayout 组件。第二个 LinearLayout 布局采用水平方向。

- android:gravity="right"*的意思是在当前容器内的子元素右靠起的方式布局。*

3 编写 Activity 代码实现登录功能

Activity 主要功能是完成用户的登陆过程，在该过程中需要将用户的数据保存到手机内存（ROM 而不是 RAM）中。根据分层设计的思想，将有关文件的读、写操作封装为一个

工具类来实现。该工具类在下一步会详细列出，这里先引用。

MainActivity.java 代码第一部分：

```
package com.itheima.rom;

import android.annotation.SuppressLint;
import android.app.Activity;
import android.os.Bundle;
import android.text.TextUtils;
import android.view.View;
import android.widget.CheckBox;
import android.widget.EditText;
import android.widget.Toast;

import com.itheima.rom.service.SaveFileService;

public class MainActivity extends Activity {
    private EditText et_username;
    private EditText et_pwd;
    private CheckBox cb;
    /*
     * 为了方便演示，因此将用户名和密码设置为常量
     */
    private static final String PWD = "123456";
    private static final String USERNAME = "wzy";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        /*
         * 初始化页面元素
         */
        et_username = (EditText) findViewById(R.id.et_username);
        et_pwd = (EditText) findViewById(R.id.et_pwd);
        cb = (CheckBox) findViewById(R.id.cb);
```

MainActivity.java 代码第二部分：

```
        /*
         * 在应用界面打开的时候查看手机内存中是否保存有用户的密码信息
         * 如果有则进行数据的回显。
         */
        String user = SaveFileService.findUser(this);
        if (user!=null) {
            String[] split = user.split(":");
            et_username.setText(split[0]);
            et_pwd.setText(split[1]);
        }
    }

    public void login(View view){
        String userName = et_username.getText().toString();
        String pwd = et_pwd.getText().toString();
        boolean checked = cb.isChecked();
        /*
         * 用户名和密码如果为空，则提示用户。
         */
        if (TextUtils.isEmpty(userName)) {
            Toast.makeText(this, "用户名不能为空！",
Toast.LENGTH_SHORT).show();
            return ;
        }
        if (TextUtils.isEmpty(pwd)) {
            Toast.makeText(this, "密码不能为空！",
Toast.LENGTH_SHORT).show();
            return ;
        }
        /*
         * 如果用户选择了保存密码,则将用户名和密码保存在手机内存中
         * 如果没有选择就将文件删除
         */
```

MainActivity.java 代码第三部分（完）：

```
        if (USERNAME.equals(userName)&&PWD.equals(pwd)) {
            if (checked) {
                SaveFileService.saveFile(this, userName, pwd);
            }else {//删除用户文件
                SaveFileService.deleteFile(this);
            }
            Toast.makeText(this, "恭喜您，登陆成功！",
Toast.LENGTH_SHORT).show();
        }else {
            Toast.makeText(this, "对不起，登陆失败！",
Toast.LENGTH_SHORT).show();
        }
    }

}
```

4 创建一个新包，包名为 com.itheima.rom.service,然后创建 SafeFileService.java 类

Tips： 这里的*Service 类并不是 Android 中的 Service 类，而只是对业务逻辑的抽取而命名的。在以后的学习中会遇到大量的*Service，这些 Service 则是 Android API 中很重要的一部分。有关 Service 会在以后的文档中作详细的介绍。

SafeFileService.java 代码第一部分：

```
package com.itheima.rom.service;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;

import android.content.Context;
```

SafeFileService.java 代码第二部分：

```
/**
 * {@http://www.itheima.com}
 * @author wzy Dec 4, 2014
 * 用于操作手机内存文件的工具类
 */
public class SaveFileService {
    //将数据保存在指定文件中
    private static final String FILE_NAME = "info.txt";
    /**
     *
     * @param context
     * @param username
     * @param pwd
     * @return boolean
     * 保存用户名和密码于文件中
     */
    public static boolean saveFile(Context context,String
username,String pwd){
        /*
         *content.getFilesDir()返回的路径为:/data/data/当前包名/files
         * 比如下面这句代码返回的路径为:
/data/data/com.itheima.rom/files
         */
        File file = new File(context.getFilesDir(), FILE_NAME);
        try {
            FileWriter fw = new FileWriter(file);
            fw.write(username+":"+pwd);
            fw.close();
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
        return true;
    }
```

SafeFileService.java 代码第三部分（完）：

```
    /**
     * @param context
     * @return
     * 删除用户文件
     */
    public static boolean deleteFile(Context context){
        File file = new File(context.getFilesDir(), FILE_NAME);
        try {
            return file.delete();
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }
    /**
     * @param context
     * @return 返回用户保存的 username:pwd
     *
     */
    public static String findUser(Context context){
        File file = new File(context.getFilesDir(), FILE_NAME);
        //如果文件不存在则返回 null
        if (!file.exists()) {
            return null;
        }
        String result = null;
        try {
            BufferedReader reader = new BufferedReader(new
FileReader(file));
            result = reader.readLine();
            reader.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return result ;
    }
}
```

5 将项目部署到 AVM 上，并进行测试

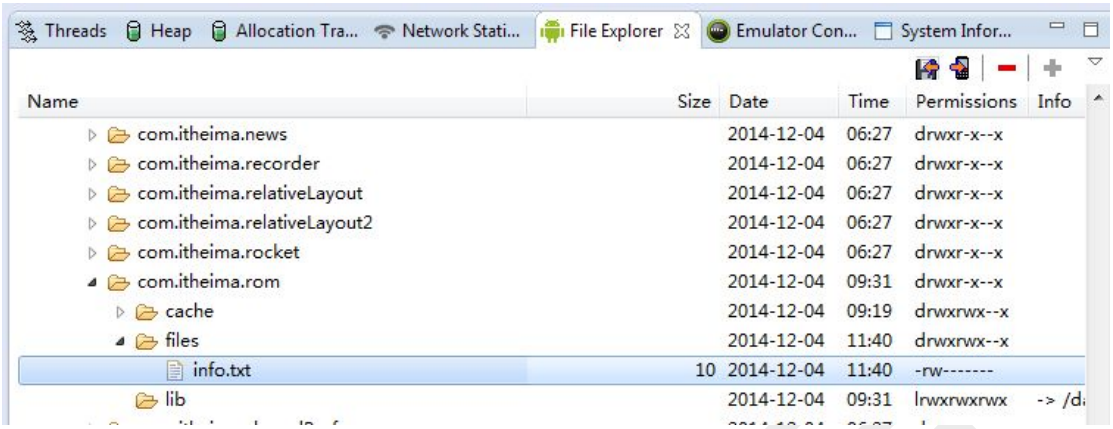
运行截图 1：

输入用户名（wzy）和密码（123456），不选择保存密码选项：

发现登录成功了。然后退出当前应用并在此打开该程序的界面的时候发现用户名和密码没用回显，这结果是正确的因为我们上次登录的时候没有勾选“保存密码”选项。这一次我们登录前勾选“保存密码”选项，然后再退出程序，重新打开应用界面，发现用户名和密码已经成功回显。

Tips： 在该案例中我们将用户名和密码以文件的形式保存在内存中，并且用户名和密码只是用“:”分隔开，这是严重 bug 的设计，如果用户的用户名或者密码中有“:”字符，那么该程序就无法获取正确的答案。之所以这么做是因为这个项目只是单纯为了演示如何将用户信息保存在手机内存中。

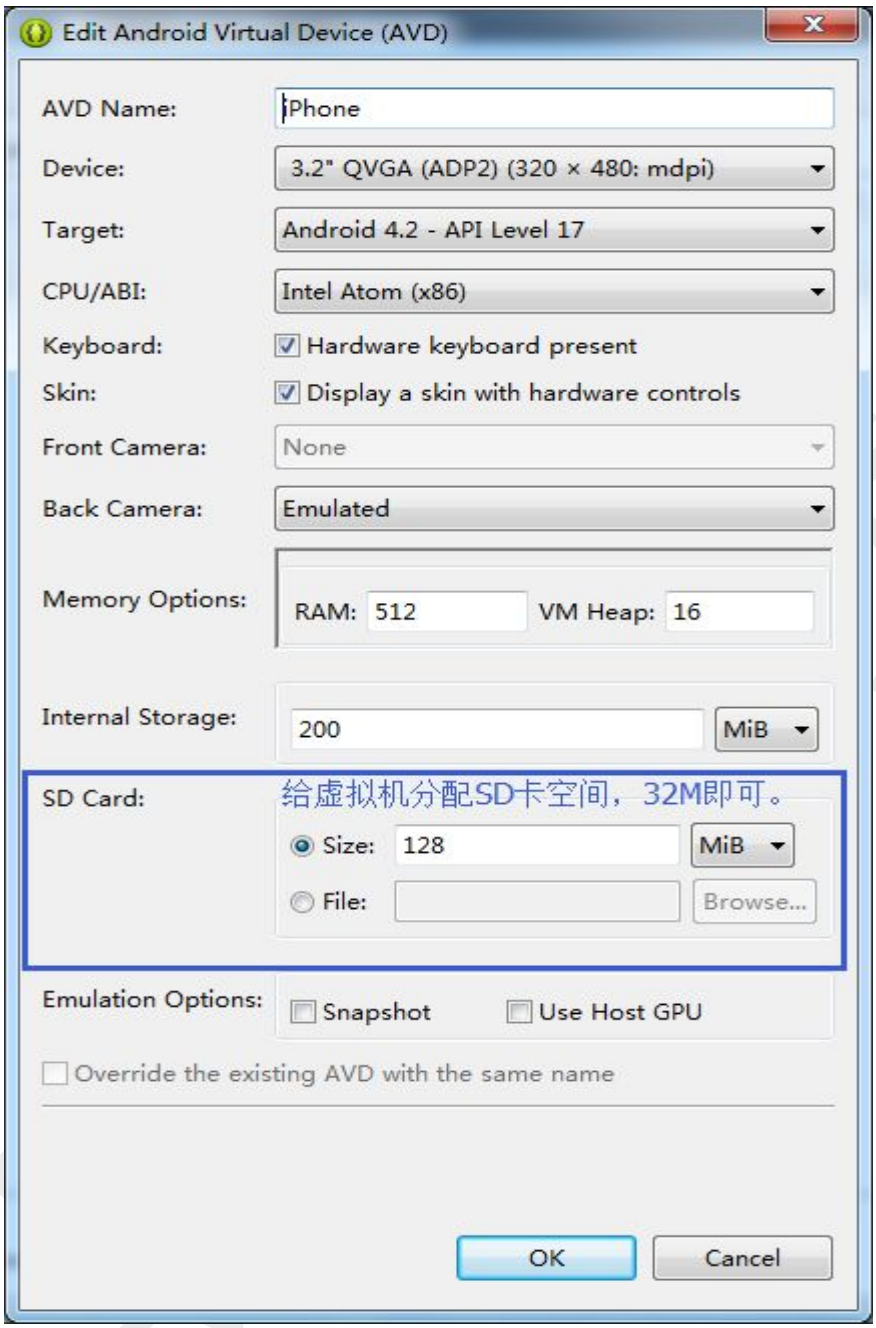
该案例会在 data/data/com.itheima.rom/files 文件夹中创建一个 info.txt 文件。如下图。



4.2 保存文件到 SD 卡

在该案例中我们依然采用 4.1 章节的案例。只需要对 SafeFileService.java 文件进行修改，将文件的保存路径改为 SD 卡。如何在 SD 卡上读、写文件是我们这章节的重点的内容。

注意：如果想将数据文件保存到 SD 卡的前提是得为你的 Android 虚拟机创建一个 SD 卡，为了演示我们分配 32M 内存空间即可。分配太多会影响虚拟机的启动时间。下图演示了如何分配 SD 卡空间。



1 将 4.1 章节中的 SaveFileService 类进行修改

也可以重新复制一下这个类，并起名为 SaveFileServiceSD，然后只需将 MainActivity 类中的 SaveFileService 全部替换为 SaveFileServiceSD，这样我们的两个文件都将保存下来，本人就是这么干的。

修改该类中的 saveFile 方法，这也是重点内容。

```
public static boolean saveFile(Context context,String username,String
pwd){
        /*
         * 监测当前设备是否已经安装好 SDCard
         * 如果没有安装好则返回
         */
        String state = Environment.getExternalStorageState();
        if (!state.equals(Environment.MEDIA_MOUNTED)) {
            return false;
        }
        /*
         * Environment.getExternalStorageDirectory()返回的路径为:
/sdcard
         * 比如下面这句代码返回的路径为: /sdcard
         * 其实/sdcard 只是一个引用地址，真正的地址为/mnt/sdcard
         */
        File file = new
File(Environment.getExternalStorageDirectory(), FILE_NAME);
        try {
            FileWriter fw = new FileWriter(file);
            fw.write(username+":"+pwd);
            fw.close();
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
        return true;
    }
```

Tips:上面这个方法对 SDCard 进行了写入文件的操作，因此需要需要在清单文件中添加权限:

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

注意:**如果不加该权限，保存文件会失败，但是很遗憾的是这时候系统并没有报任何异常。**

修改该类中的 saveFile 方法。

代码修改起来很简单，只需要修改一个地方即可。在第一个方法中已经展示了如何判断 SDCard 是否处于可用状态，处于节省篇幅的考虑以后的方法中就不再做判断，如果是在实际开发中，所有关于 SDCard 的读、写操作都建议大家进行判断。

```
public static boolean deleteFile(Context context){
      File file = new
File(Environment.getExternalStorageDirectory(), FILE_NAME);
      try {
         return file.delete();
      } catch (Exception e) {
         e.printStackTrace();
         return false;
      }
   }
```

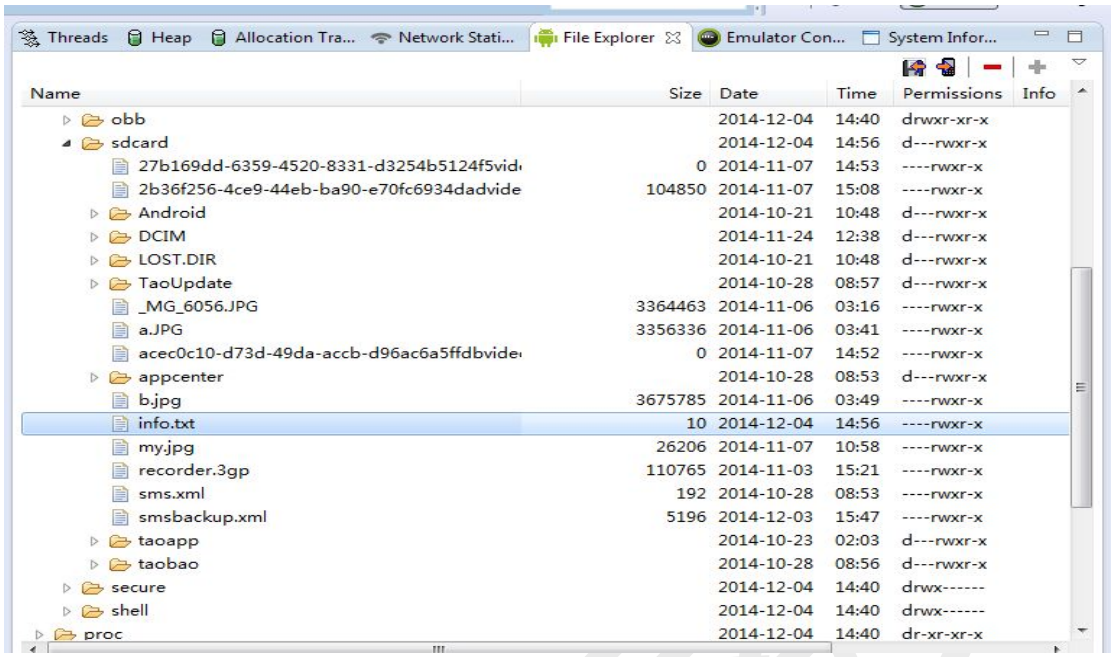
修改该类中的 findUser 方法。

同样的只需要修改一个地方即可。

```
public static String findUser(Context context){
       File file = new
File(Environment.getExternalStorageDirectory(), FILE_NAME);
       //如果文件不存在则返回 null
       if (!file.exists()) {
          return null;
       }
       String result = null;
       try {
          BufferedReader reader = new BufferedReader(new
FileReader(file));
          result = reader.readLine();
          reader.close();
       } catch (Exception e) {
          e.printStackTrace();
       }
       return result ;
   }
```

2 运行该应用于模拟器上。输入正确的用户名和密码，并勾选“保存密码”选项。然后打开 DDMS 的 File Explorer 窗口。可以看到 sdcard 目录下产生了 info.txt 文件。

当取消“保存密码”选项后，在点击登录，然后再打开 DDMS 的 File Explorer 窗口，发现该文件已经被删除了。



Tips：当我们的应用界面处于打开状态时，我们重新部署该程序到模拟器上经常会有类似如下异常产生。产生这样的异常很好解决，直接在的模拟器中将该应用退出。然后再部署一般就可以解决问题。

```
com.itheima.rom.MainActivity activity launch
[2014-12-04 23:00:19 - 保存文件到手机内存] Automatic Target Mode: using
existing emulator 'emulator-5554' running compatible AVD 'iPhone'
[2014-12-04 23:00:19 - 保存文件到手机内存] Application already
deployed. No need to reinstall.
[2014-12-04 23:00:19 - 保存文件到手机内存] Starting activity
com.itheima.rom.MainActivity on device emulator-5554
[2014-12-04 23:00:19 - 保存文件到手机内存] ActivityManager: Starting:
Intent { act=android.intent.action.MAIN
cat=[android.intent.category.LAUNCHER]
cmp=com.itheima.rom/.MainActivity }
[2014-12-04 23:00:19 - 保存文件到手机内存] ActivityManager: Warning:
Activity not started, its current task has been brought to the front
```

4.3 使用 SharedPreferences

SharedPreferences 是 Android 平台上一个轻量级的存储类，主要是保存一些常用的配置，它提供了 Android 平台常规的 Long、Int、String 字符串型的保存，它是什么样的处理方式呢?SharedPreferences 类似过去 Windows 系统上的 ini 配置文件，但是它分为多种权限，可以全局共享访问，整体效率来看不是特别的高，对于常规的轻量级而言比 SQLite 要好不少，如果真的存储量不大可以考虑自己定义文件格式。xml 处理时 Dalvik 会通过自带底层的本地 XML Parser 解析，比如 XMLpull 方式，这样对于内存资源占用比较好。

这种方式应该是用起来最简单的 Android 读写外部数据的方法了。他以一种简单、 透明的方式来保存一些用户个性化设置的字体、颜色、位置等参数信息。一般的应用程序都会提供“设置”或者“首选项”的这样的界面，那么这些设置最后就可以通过 Preferences 来保存，而程序员不需要知道它到底以什么形式保存的，保存在了什么地方。当然，如果你愿意保存其他的东西，也没有什么限制。只是在性能上不知道会有什么问题。

在 Android 系统中该文件保存在：/data/data/PACKAGE_NAME /shared_prefs 目录下。

下面通过修改 4.1 章节中的案例来演示如何使用 SharedPreferences。在此案例中对用户数据的操作就不再需要 SaveFileService 类，直接在 MainActivity 类中使用 SharedPreferences API 即可。由于操作 SharedPreferences 不需要权限，因此清单文件中关于写 SDCard 的权限可以删除，当然放在那里不删除也是可以的。

修改后的 MainActivity 如下：

MainActivity.java 代码第一部分：

```
package com.itheima.rom;
import android.annotation.SuppressLint;
import android.app.Activity;
import android.content.SharedPreferences;
import android.content.SharedPreferences.Editor;
import android.os.Bundle;
import android.text.TextUtils;
import android.view.View;
import android.widget.CheckBox;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends Activity {
    private EditText et_username;
    private EditText et_pwd;
    private CheckBox cb;
    //声明一个 SharedPreferences 对象
    private SharedPreferences sp;
    /*
     * 为了方便演示，因此将用户名和密码设置为常量
     */
    private static final String PWD = "123456";
    private static final String USERNAME = "wzy";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        /*
         * 第二个参数代表的是创建该文件的访问范围（权限），通常并建议选择
MODE_PRIVATE，该值为 0，
         * 意思是只有当前应用可以访问该文件。而还有 2 个可选项
MODE_WORLD_READABLE 和 MODE_WORLD_WRITEABLE
         * 值分别为 1 和 2 已经废除，因为这两种方式可以允许其他应用来访问此文
件，这是很不安全的。
         */
        sp = getSharedPreferences("info", MODE_PRIVATE);
```

MainActivity.java 代码第二部分：

```
      et_username = (EditText) findViewById(R.id.et_username);
       et_pwd = (EditText) findViewById(R.id.et_pwd);
       cb = (CheckBox) findViewById(R.id.cb);
       /*
        * 从 sp 中获取用户信息，用户数据的回显
        * 第二个参数为默认返回值，也就是当要查找的 key-value 不存在时，返
回的数据
        */
       String username = sp.getString("username", "");
       String pwd = sp.getString("pwd", "");
       et_username.setText(username);
       et_pwd.setText(pwd);
   }

   public void login(View view){
       String userName = et_username.getText().toString();
       String pwd = et_pwd.getText().toString();
       boolean checked = cb.isChecked();
       /*
        * 用户名和密码如果为空，则提示用户。
        */
       if (TextUtils.isEmpty(userName)) {
           Toast.makeText(this, "用户名不能为空！",
Toast.LENGTH_SHORT).show();
           return ;
       }
       if (TextUtils.isEmpty(pwd)) {
           Toast.makeText(this, "密码不能为空！",
Toast.LENGTH_SHORT).show();
           return ;
       }
       /*
        * 如果用户选择了保存密码,则将用户名和密码保存在手机内存中
        * 如果没有选择就将文件删除
        */
```

MainActivity.java 代码第三部分：

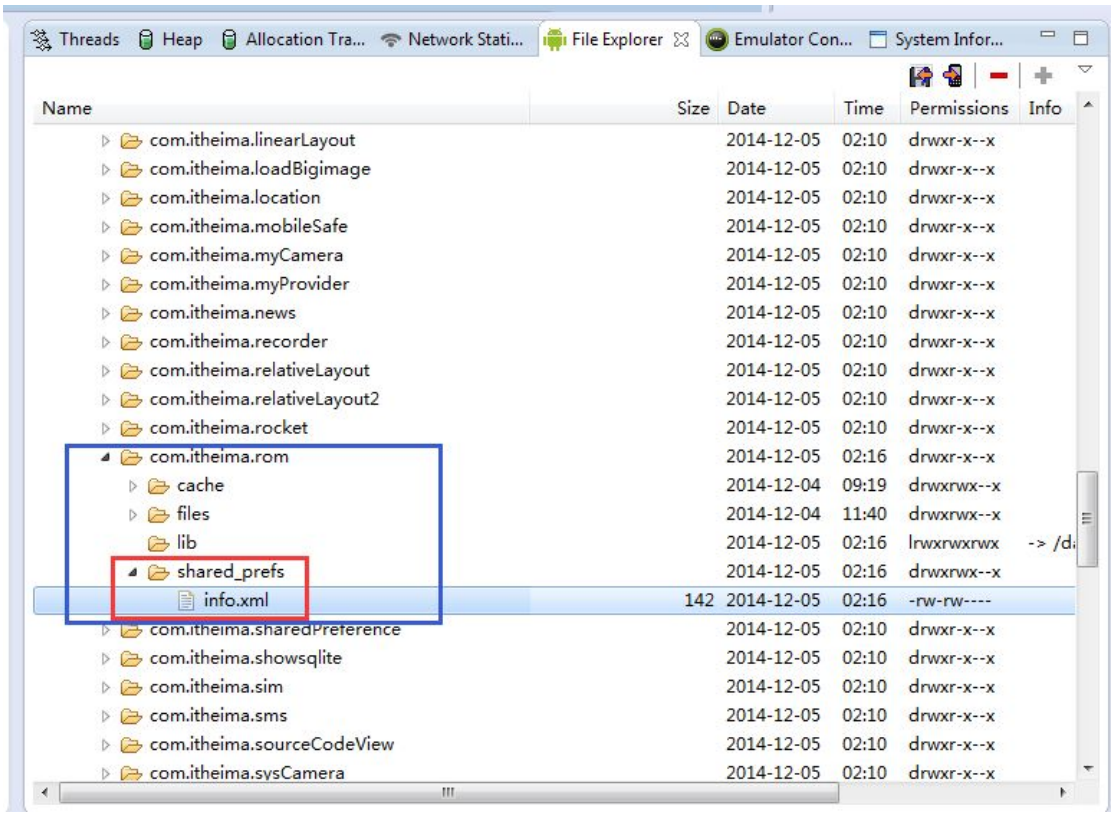
```
if (USERNAME.equals(userName)&&PWD.equals(pwd)) {
          if (checked) {
              /*
               * 在对 sp 进行写、修改需要获取 Editor 对象
               */
              Editor editor = sp.edit();
              editor.putString("username", userName);
              editor.putString("pwd", pwd);
              /*
               * 此处非常重要，执行完修改或者写操作后只有调用 sp 的 commit
方法，数据才会被保存下来。
               */
              editor.commit();
          }else {//删除用户文件
              Editor editor = sp.edit();
              /*
               * 删除该 sp 中的所有数据
               */
              editor.clear();
              editor.commit();
          }
          Toast.makeText(this, "恭喜您，登陆成功！",
Toast.LENGTH_SHORT).show();
       }else {
          Toast.makeText(this, "对不起，登陆失败！",
Toast.LENGTH_SHORT).show();
       }
   }

}
```

运行该程序，实现效果跟 4.1、4.2 章节是相同的，因此效果图就不再展示。

新技能get：运行上面程序后系统会自动创建一个文件：

/data/data/com.itheima.rom/shared_prefs/info.xml

文件目录结构如下图。



5.获取 SD 卡和内存的空间信息

本章节将通过案例演示在 Android 中如何获取 SDCard 和手机内存的总空间和可用空间等信息。

1 创建一个新的 Android 工程，工程名字为《获取存储空间大小》，包名为：com.itheima.storageSize

这里使用默认生成的布局文件和 Activity 类。

2 修改布局文件 activity_main.xml

布局文件第一部分：

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity" >
   <LinearLayout
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:layout_marginTop="10dp"
       >
       <TextView
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:text="可用空间"
           />
       <EditText
           android:layout_width="match_parent"
           android:layout_height="wrap_content"
           android:hint="显示可用空间"
           android:id="@+id/et_availSize"
           />
   </LinearLayout>
   <LinearLayout
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:layout_marginTop="10dp"
       >
       <TextView
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:text="总空间"
           />
```

布局文件第二部分（完）：

```
 <EditText
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:hint="显示总空间"
          android:id="@+id/et_TotalSize"
          />
   </LinearLayout>
   <LinearLayout
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:layout_marginTop="10dp"
       >
     <Button
         android:layout_width="0dp"
         android:layout_weight="1"
         android:layout_height="wrap_content"
         android:text="获取 SD 卡信息"
         android:onClick="sdCardInfo"
         />
     <Button
         android:layout_width="0dp"
         android:layout_weight="1"
         android:layout_height="wrap_content"
         android:text="获取内存信息"
         android:onClick="memeoryInfo"
         />
   </LinearLayout>

</LinearLayout>
```

3 编写业务代码

业务代码第一部分：

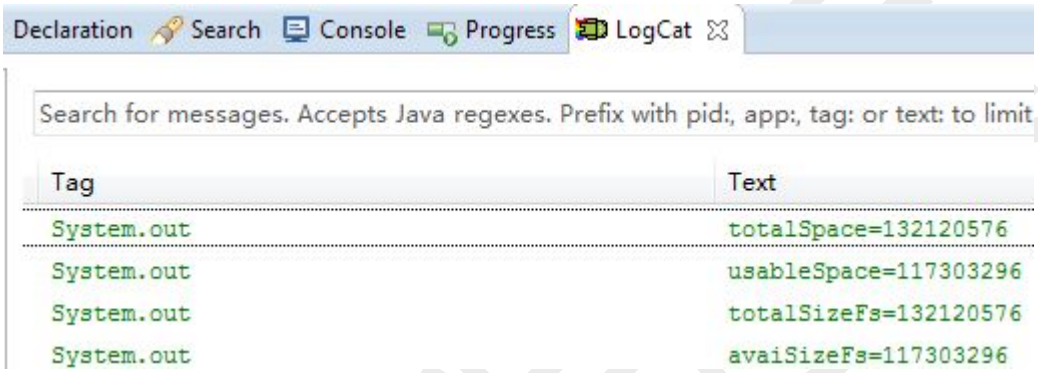
```
package com.itheima.storageSize;

import java.io.File;

import android.app.Activity;
import android.os.Bundle;
import android.os.Environment;
import android.os.StatFs;
import android.text.format.Formatter;
import android.view.View;
import android.widget.EditText;

public class MainActivity extends Activity {

    private EditText et_avaiSize;
    private EditText et_totalSize;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        /*
         * 初始化布局中的控件对象
         */
        et_avaiSize = (EditText) findViewById(R.id.et_availSize);
        et_totalSize = (EditText) findViewById(R.id.et_TotalSize);
    }
    /**
     * 获取 SDCard 内存信息
     * @param view
     */
    public void sdCardInfo(View view){
        File storageDirectory =
Environment.getExternalStorageDirectory();
        /*
         * Returns the total size in bytes of the partition containing
this path.
         *  Returns 0 if this path does not exist.
         *  返回该路径下文件的总大小，单位是字节。如果该路径不存在则返回 0
         */
```

业务代码第二部分：

```
long totalSpace = storageDirectory.getTotalSpace();
        long usableSpace = storageDirectory.getUsableSpace();
        /*
         * Formats a content size to be in the form of bytes, kilobytes,
megabytes, etc
         * 将返回的大小格式化为 byte、KB、MB、等等。
         */
        String totalSpaceString = Formatter.formatFileSize(this,
totalSpace);
        String usableSpaceString = Formatter.formatFileSize(this,
usableSpace);
        /*
         * Retrieve overall information about the space on a filesystem.
This is a wrapper for Unix statfs().
         */
        StatFs fs = new StatFs(storageDirectory.getAbsolutePath());
        /*
         * The total number of blocks on the file system. This corresponds
to the Unix statfs.f_blocks field.
         *
         */
        long blockCount = fs.getBlockCount();
        int blockSize = fs.getBlockSize();
        int availableBlocks = fs.getAvailableBlocks();
        long totalSizeFs = blockCount*blockSize;
        long avaiSizeFs = blockSize*availableBlocks;
        String totalSizeFsString = Formatter.formatFileSize(this,
totalSizeFs);
        String avaiSizeFsString = Formatter.formatFileSize(this,
avaiSizeFs);
        et_totalSize.setText(totalSizeFsString);
        et_avaiSize.setText(avaiSizeFsString);
        System.out.println("totalSpace="+totalSpace);
        System.out.println("usableSpace="+usableSpace);
        System.out.println("totalSizeFs="+totalSizeFs);
        System.out.println("avaiSizeFs="+avaiSizeFs);
```

Tips：在上述方法中，我们使用了两种方法分别计算 SDCard 的内存信息。其中第一种方

法是 JDK API 提供的，第二种方法是 Android API 提供的。这两种方法获取到的总容量和可用容量信息在日志中输出见下图。发现得到的结果是一样的。在我们 Android 的开发中自己比较推荐使用第二种方法。因为第二种方法 Google 工程师专门针对 Android 系统设计的。相对更加的适用，在看 Android 源码的时候我们也能发现 Android 系统自己计算内存容量的时候使用的就是第二种方法。



业务代码第三部分（完）：

```
    }
    /**
     * 获取手机内部存储信息
     * @param view
     */
    public void memeoryInfo(View view){
        File filesDir = getFilesDir();
        long totalSpace = filesDir.getTotalSpace();
        long usableSpace = filesDir.getUsableSpace();
        String totalSpaceStr = Formatter.formatFileSize(this,
totalSpace);
        String usableSpaceStr = Formatter.formatFileSize(this,
usableSpace);
        et_totalSize.setText(totalSpaceStr);
        et_avaiSize.setText(usableSpaceStr);
    }

}
```

4 运行该程序，分别获取 SD 卡信息和内存信息。效果如图。

新技能get：上面布局文件中出现了如下的属性：

```
android:layout_width="0dp"        android:layout_weight="1"
```

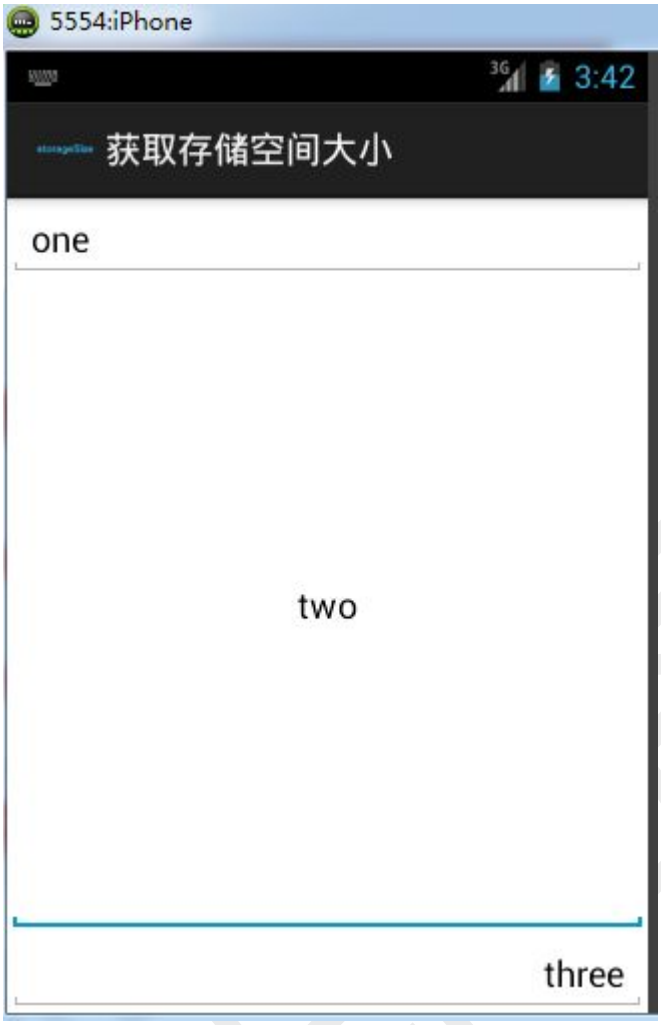
在 LinearLayout 的 android:orientation="horizontal"（或者默认）的情况下，如果 android:layout_width="0dp" 那么就必须添加 android:layout_weight="1"属性，属性值是 float 类型的。下面将重点说明该属性的真实含义。

5.1 android:layout_weight 属性详解

Layout_weight 属性的作用：它是用来分配剩余空间的一个属性，你可以设置他的权重。通过几个小例子来说明该属性的用法，比如有如下布局文件：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<EditText
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:gravity="left"
    android:text="one"/>
<EditText
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:gravity="center"
    android:layout_weight="1.0"
    android:text="two"/>
    <EditText
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:gravity="right"
    android:text="three"/>
</LinearLayout>
```

运行结果是：



看上面代码发现只有 Button2 使用了 Layout_weight 属性 ,并赋值为了 1 ,而 Button1 和 Button3 没有设置 Layout_weight 这个属性，根据 API，可知，他们默认是 0。

下面我就来讲，Layout_weight 这个属性的真正的意思：**Android 系统先按照你设置的 3 个 Button 高度 Layout_height 值 wrap_content,给你分配好他们 3 个的高度，然后会把剩下来的屏幕空间全部赋给 Button2,因为只有他的权重值是 1，这也是为什么 Button2 占了那么大的一块空间。**

通过上面的例子我相信大家对该属性已经有了一定的了解，那么再看下面的例子，相信大家会得到进一步的了解。

布局文件（完）：

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="20dp"
    android:orientation="horizontal" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:background="#ff0000"
        android:text="1"
        android:textColor="@android:color/white" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="2"
        android:background="#cccccc"
        android:text="2"
        android:textColor="@android:color/black" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="3"
        android:background="#ddaacc"
        android:text="3"
        android:textColor="@android:color/black" />

</LinearLayout>
```

运行效果如下：

- 对于上面运行效果的分析：三个文本框的属性 layout_width= “wrap_content ”时，系统先给 3 个 TextView 分配他们的宽度值 wrap_content（宽度足以包含他们的内容 1,2,3 即可），然后会把剩下来的屏幕空间按照 1:2:3 的比列分配给 3 个 textview，所以就出现了上面的图像。

- 修改上面的布局文件，当 layout_width= “fill_parent” 时，如果分别给三个 TextView 设置他们的 Layout_weight 为 1、2、2 的话，就会出现下面的效果：

我们会发现 1 的权重小，反而分的多了，这是为什么呢？我们可以简单的理解为当 layout_width= “fill_parent” 时，weight 值越小权重越大，优先级越高，而其真正的原因是 layout_width="fill_parent"的原因造成的。依照上面理解我们来分析：

系统先给 3 个 textview 分配他们所要的宽度 fill_parent，也就是说每一都是填满他的

父控件，这里就是屏幕的宽度那么这时候的剩余空间=1 个 parent_width-3 个 parent_width=-2 个 parent_width (parent_width 指的是屏幕宽度)。

那么第一个 TextView 的实际所占宽度应该=parent_width + 他所占剩余空间的权重比列 1/5 * 剩余空间大小（-2 parent_width）=3/5parent_width, 同理第二个 TextView 的实际所占宽度=parent_width+2/5*(-2parent_width)=1/5parent_width; 第三个 TextView 的实际所占宽度=parent_width + 2/5*(-2parent_width)=1/5parent_width；所以就是 3:1:1 的比列显示了。

6.XML 文件的读取和序列化

在 Android 平台上可以使用 Simple API for XML(SAX) 、Document Object Model(DOM)和 Android 自带的 pull 解析器解析 XML 文件。

1. SAX 解析 XML 文件

SAX 是一个解析速度快并且占用内存少的 xml 解析器，非常适合用于 Android 等移动设备。 SAX 解析 XML 文件采用的是事件驱动，也就是说，它并不需要解析完整个文档，在按内容顺序解析文档的过程中，SAX 会判断当前读到的字符是否合法 XML 语法中的某部分，如果符合就会触发事件。所谓事件，其实就是一些回调（callback）方法，这些方法(事件)定义在 ContentHandler 接口。

2. DOM 解析 XML 文件

DOM 解析 XML 文件时，会将 XML 文件的所有内容读取到内存中，然后允许您使用 DOM API 遍历 XML 树、检索所需的数据。使用 DOM 操作 XML 的代码看起来比较直观，并且，在某些方面比基于 SAX 的实现更加简单。但是，因为 DOM 需要将 XML 文件的所有内容读取到内存中，所以内存的消耗比较大，特别对于运行 Android 的移动设备来说，因为设备的资源比较宝贵，所以建议还是采用 SAX 来解析 XML 文件，当然，如果 XML 文件的内容比较小采用 DOM 是可行的。

3.Pull 解析器解析 XML 文件

Pull 解析器的运行方式与 SAX 解析器相似。它提供了类似的事件，如：开始元素和结束元素事件，使用 parser.next()可以进入下一个元素并触发相应事件。事件将作为数值代码被发送，因此可以使用一个 switch 对感兴趣的事件进行处理。当元素开始解析时，调用 parser.nextText()方法可以获取下一个 Text 类型元素的值。

4.SAX 和 PULL 区别

SAX 解析器的工作方式是自动将事件推入事件处理器进行处理，因此你不能控制事件的处理主动结束；而 Pull 解析器的工作方式为允许你的应用程序代码主动从解析器中获取事件，正因为是主动

获取事件，因此可以在满足了需要的条件后不再获取事件，结束解析。

你随便找个 sax 和 pull 的例子比较一下就可以发现，pull 是一个 while 循环，随时可以跳出，而 sax 不是，sax 是只要解析了，就必须解析完成。

Tips： 在本文档中我们将通过一个案例来重点讨论 Android 自带的 pull 解析器的使用。首先介绍在 Android 中 XML 的序列化。

演示 XML 的序列化

创建一个工程，使用其默认的布局文件和 Activity 类。布局文件如下：

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:orientation="vertical"
     >
    <Button
        android:layout_height="60dp"
        android:layout_width="wrap_content"
        android:text="生成XML 文件"
        android:onClick="makeXML"
        />
</LinearLayout>
```

Activity 类比较简单，出于篇幅考虑，这里只给出核心方法。

makeXML 方法：

```
public void makeXML(View v) {
	File file = null;
	try {
	    XmlSerializer serializer = Xml.newSerializer();
	    //将生成的文件保存在 SDCard 中
	    String path =
Environment.getExternalStorageDirectory().getAbsolutePath() +
"/sms.xml";
	    file = new File(path);
	    FileOutputStream outputStream = new FileOutputStream(file);
	    serializer.setOutput(outputStream, "utf-8");
	    serializer.startDocument("utf-8", true);
	    serializer.startTag(null, "message");
	    serializer.startTag(null, "sms");
	    serializer.startTag(null, "address");
	    serializer.text("这是测试地址");
	    serializer.endTag(null, "address");
	    serializer.endTag(null, "sms");
	    serializer.startTag(null, "body");
	    serializer.text("这是测试短信内容。");
	    serializer.endTag(null, "body");
	    serializer.startTag(null, "number");
	    serializer.text("132111111111");
	    serializer.endTag(null, "number");
	    serializer.endTag(null, "message");
	    serializer.endDocument();
	} catch (Exception e) {
	    Toast.makeText(this, "xml 生成失败",
Toast.LENGTH_LONG).show();
	}
	Toast.makeText(this, "xml 生成了" + file.getAbsolutePath(),
Toast.LENGTH_LONG).show();
    }
```

Tips:由于此方法中对 SDCard 进行了写操作，因此必须加上如下权限：

android.permission.WRITE_EXTERNAL_STORAGE。

运行上面程序后发现在/mnt/sdcard 下面多了一个 sms.xml 文件。将该文件导出到电脑中

打开查看如下内容：

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<message>
  <sms>
    <address>这是测试地址</address>
  </sms>

  <body>这是测试短信内容。</body>
  <number>132111111111</number>

  </message>
```

演示 XML 的 Pull 解析

已知 weather.xml 文件中存储着天气信息，使用 pull 解析 xml，将天气信息解析出来并显示。weather.xml 内容如下：

```
<?xml version='1.0' encoding='utf-8' standalone='yes'?>
<weather>
    <city id="1">
    <name>北京</name>
    <pm>200</pm>
    <wind>4</wind>
    <temp>21-30</temp>
    </city>
</weather>
```

布局文件比较简单，出于节省篇幅的考虑就不再给出。直接给出 java 代码。首先是 Weather.java 类，该类主要用户对数据的封装，这里只给出该类的属性。

```
    private String id;//city id
    private String name;//city name
    private String pm;//pm2.5 指数
    private String wind;//风力
    private String temp;//温度
```

创建 PullService.java 类，该类用于完成解析 XML 的业务逻辑。

```
public class PullService {
    public static Weather pullWeather(InputStream in) throws
Exception{
        Weather weather = null;
        XmlPullParser parser = Xml.newPullParser();
        parser.setInput(in, "utf-8");
        //开始循环遍历 xml 文件，直到文件的结尾
        int type = parser.next();
        while(type!=XmlPullParser.END_DOCUMENT){
            if (type==XmlPullParser.START_TAG) {
                //如果开始标签是 weather 标签则实例化 weather
                if ("weather".equals(parser.getName())) {
                    weather = new Weather();
                }else if ("city".equals(parser.getName())) {
                    //获取节点下的属性值
                    String id = parser.getAttributeValue(0);
                    weather.setId(id);
                }else if ("name".equals(parser.getName())) {
                    String name = parser.nextText();
                    weather.setName(name);
                }else if ("pm".equals(parser.getName())) {
                    String pm = parser.nextText();
                    weather.setPm(pm);
                }else if ("wind".equals(parser.getName())) {
                    String wind = parser.nextText();
                    weather.setWind(wind);
                }else if ("temp".equals(parser.getName())) {
                    String temp = parser.nextText();
                    weather.setTemp(temp);
                }
            }
            type = parser.next();
        }
        in.close();
        return weather;
    }
}
```

将 weather.xml 文件拷贝到 src 目录下，MainActivity.java 中修改 pullXML 方法

```
public void pullXML(View v) {
      InputStream in =
this.getClass().getClassLoader().getResourceAsStream("weather.x
ml");
      try {
         Weather weather = PullService.pullWeather(in);
         Toast.makeText(this,weather.toString() , 1).show();
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
```

运行结果如图：

7. 案例-学生管理系统

该案例的重点在于 xml 的解析和序列化，同时该案例的布局方案也是值得学习的知识点。系统运行如下图：

需求说明：1、点击添加按钮，将文本框中的数据添加到页面

2、点击保存数据按钮，将页面中的所有数据保存到 xml 文件中

3、点击清空数据，将页面中的所有数据清空，xml 中的数据不作处理

4、点击恢复数据，将 xml 文件中的数据展示在页面

使用工程默认的布局文件（第一部分）：

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:orientation="vertical"
     >
    <TextView
        android:layout_marginTop="10dp"
        android:textColor="#0000ff"
        android:gravity="center_horizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/studentsys"
        />
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        >
        <LinearLayout
            android:layout_height="wrap_content"
            android:layout_width="0dp"
            android:orientation="vertical"
            android:layout_weight="1"
            >
            <TextView
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:text="姓名"
                />
            <EditText
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:id="@+id/et_username"
                />
        </LinearLayout>
```

布局文件第二部分：

```
        <LinearLayout
           android:layout_height="wrap_content"
           android:layout_width="0dp"
           android:orientation="vertical"
           android:layout_weight="1"
           >
           <TextView
               android:layout_width="match_parent"
               android:layout_height="wrap_content"
               android:text="年龄"
               />
           <EditText
               android:layout_width="match_parent"
               android:layout_height="wrap_content"
               android:inputType="number"
               android:id="@+id/et_age"
               />
       </LinearLayout>
        <LinearLayout
           android:layout_height="wrap_content"
           android:layout_width="0dp"
           android:orientation="vertical"
           android:layout_weight="1"
           >
           <TextView
               android:layout_width="match_parent"
               android:layout_height="wrap_content"
               android:text="性别"
               />
           <EditText
               android:layout_width="match_parent"
               android:layout_height="wrap_content"
               android:id="@+id/et_sex"
               />
       </LinearLayout>
```

布局文件第三部分：

```
<Button
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="添加"
            android:id="@+id/bt_saveUser"
            android:onClick="saveUser"
            />
    </LinearLayout>

    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        >
         <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:id="@+id/ll_list"
        ></LinearLayout>
    </ScrollView>
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        >
        <Button
            android:layout_width="0dp"
            android:layout_weight="1"
            android:layout_height="wrap_content"
            android:text="保存数据"
            android:onClick="saveData"
            android:id="@+id/bt_saveData"
            />
```

布局文件第四部分（完）：

```
 <Button
           android:layout_width="0dp"
           android:layout_weight="1"
           android:layout_height="wrap_content"
           android:text="恢复数据"
           android:onClick="revalData"
           android:id="@+id/bt_revalData"
           />
       <Button
           android:layout_width="0dp"
           android:layout_weight="1"
           android:layout_height="wrap_content"
           android:text="清空数据"
           android:id="@+id/bt_clearData"
           android:onClick="clearData"
           />
           </LinearLayout>

</LinearLayout>
```

创建 User 类，其属性信息和在 XML 中的格式分别如下：

```
public class User {
    private String username;
    private int age;
    private String sex;
..........省略 setter 和 getter 方法
```

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<users>
  <user>
    <name>www</name>
    <age>12</age>
     <sex>ww</sex>

   </user>

  </users>
```

为了方便演示，这里将核心业务写在默认的 MainActivity 类中：

代码第一部分：

```
public class MainActivity extends Activity {
    private EditText et_username;
    private EditText et_sex;
    private EditText et_age;
    private Button bt_saveUser;
    private Button bt_saveData;
    private Button bt_clearData;
    private Button bt_revalData;
    private LinearLayout ll_list;
    private static final String ENCODING = "utf-8";
    private static final String FILE_PATH = "users.xml";
```

代码第二部分：

该部分代码主要对布局文件中元素进行了初始化操作。

```
@Override
protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      et_username = (EditText) findViewById(R.id.et_username);
      et_age = (EditText) findViewById(R.id.et_age);
      et_sex = (EditText) findViewById(R.id.et_sex);
      bt_clearData = (Button) findViewById(R.id.bt_clearData);
      bt_revalData = (Button) findViewById(R.id.bt_revalData);
      bt_saveData = (Button) findViewById(R.id.bt_saveData);
      bt_saveUser = (Button) findViewById(R.id.bt_saveUser);
      ll_list = (LinearLayout) findViewById(R.id.ll_list);

   }
```

代码第三部分：

```
// 保存用户信息到界面视图中，注意：此时并没有保存到 xml 文件中
    public void saveUser(View view) {
        //获取到用户输入的信息
        String age = et_age.getText().toString();
        String sex = et_sex.getText().toString();
        String username = et_username.getText().toString();
        //动态创建 LinearLayout 容器，用于容纳用户信息
        LinearLayout layout = new LinearLayout(this);
        layout.setOrientation(LinearLayout.HORIZONTAL);
        //创建 3 个 TextView 对象用户显示用户信息
        TextView tv_username = new TextView(this);
        tv_username.setText("\t\t" + username + "\t\t");
        //将用户信息添加 到 LinearLayout 容器中
        layout.addView(tv_username);
        TextView tv_age = new TextView(this);
        tv_age.setText("\t\t" + age);
        layout.addView(tv_age);
        TextView tv_sex = new TextView(this);
        tv_sex.setText("\t\t" + sex + "\t\t");
        layout.addView(tv_sex);
        //将动态创建的 LinearLayout 容器放到界面 LinearLayout 容器中
        ll_list.addView(layout, 0);
    }
```

代码第四部分：saveData（）方法，在界面点击添加按钮的业务逻辑。

```
public void saveData(View view) {
        // 获取 LinearLayout 的所有孩子节点
        int childCount = ll_list.getChildCount();
        List<User> users = new ArrayList<User>();
        for (int i = 0; i < childCount; i++) {
            //从 LinearLayout 容器中遍历其子元素，其子元素也是
LinearLayout
            LinearLayout layout = (LinearLayout)
ll_list.getChildAt(i);
            //获取子元素 LinearLayout 中的 3 个 TextView
            TextView tv_username = (TextView) layout.getChildAt(0);
            TextView tv_age = (TextView) layout.getChildAt(1);
            TextView tv_sex = (TextView) layout.getChildAt(2);
```

```
            String username =
tv_username.getText().toString().trim();
            int age =
Integer.valueOf(tv_age.getText().toString().trim());
            String sex = tv_sex.getText().toString().trim();
            User user = new User();
            user.setAge(age);
            user.setSex(sex);
            user.setUsername(username);
            users.add(user);
        }
        //调用序列化 xml 的方法
        boolean flag = seriaXML(users);
        if (flag) {
            Toast.makeText(this, "数据已经保存成功", 0).show();
        }else {
            Toast.makeText(this, "数据保存失败", 0).show();
        }
    }
```

代码第五部分（核心代码）：seriaXML(List<User> users)方法的实现

```
/*
     * 将集合中的数据序列化到 xml 文件中
     */
    private boolean seriaXML(List<User> users) {
        //创建 Xml 序列化对象
        XmlSerializer serializer = Xml.newSerializer();
        try {
            //创建文件，用于保存 xml 数据 getFilesDir()放回的路径为
            //data/data/com.itheima.studentManager/files
            File file = new File(getFilesDir(), FILE_PATH);
            //获取该文件输出流
            FileOutputStream outputStream = new
FileOutputStream(file);
            //给序列化期设置输出流和字符编码
            serializer.setOutput(outputStream, ENCODING);
            //开始编辑 xml 文档，并设置 xml 文档的编码
            serializer.startDocument(ENCODING, true);
```

```
            //创建 users 标签，第一个参数是命名空间，我们设置为 null
            serializer.startTag(null, "users");
            for (User user : users) {
                //有开始标签就一定记得有结束标签 开始与结束是对称的关系
                serializer.startTag(null, "user");
                serializer.startTag(null, "name");
                serializer.text(user.getUsername());
                serializer.endTag(null, "name");
                serializer.startTag(null, "age");
                serializer.text(user.getAge() + "");
                serializer.endTag(null, "age");
                serializer.startTag(null, "sex");
                serializer.text(user.getSex());
                serializer.endTag(null, "sex");
                serializer.endTag(null, "user");
            }
            serializer.endTag(null, "users");
            //完成 xml 的编辑，并输出到文件中
            serializer.endDocument();
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
        return true;
    }
```

代码第六部分：实现清除数据功能，对应的是 clearData()方法，这里的业务逻辑比较简单，只需要清楚界面的数据就行，其实就是将页面 LinearLayout 容器里面的所有子元素清除即可。

```
/*
     * 将页面的数据清除掉
     */
    public void clearData(View view) {
        ll_list.removeAllViews();
    }
```

代码第七部分：实现恢复数据功能，也就是点击恢复数据按钮后将 xml 文件中的数据显示在界面。

```
public void revalData(View view) {
        //自定义方法，该方法完成了将数据从 xml 文件到集合的功能
        List<User> users = pullXML();
        //在界面显示 xml 中的数据前，先清空当前页面中的老数据
        clearData(view);
        for(User user : users){
            String age = user.getAge()+"";
            String sex = user.getSex();
            String username = user.getUsername();
            //动态创建一个 LinearLayout
            LinearLayout layout = new LinearLayout(this);
            //设置布局方式为水平布局
            layout.setOrientation(LinearLayout.HORIZONTAL);
            //动态创建 3 个 TextView，用于显示用户的数据，并将 TextView
添加到 LinearLayout 容器中
            TextView tv_username = new TextView(this);
            tv_username.setText("\t\t" + username + "\t\t");
            layout.addView(tv_username);
            TextView tv_age = new TextView(this);
            tv_age.setText("\t\t" + age);
            layout.addView(tv_age);
            TextView tv_sex = new TextView(this);
            tv_sex.setText("\t\t" + sex + "\t\t");
            layout.addView(tv_sex);
            //请动态创建的 LinearLayout 容器添加到布局文件中的
LinearLayout 容器中，第二个参数是指放置的位置
            //在这里我们添加到第 0 个位置，也就是最上面，这样保证了最下面
的数据在最上面
            ll_list.addView(layout, 0);
        }
    }
```

代码第八部分（核心功能）：实现 pullXML()方法，也就是将 xml 文件中的数据用 pull 方式解析出来，并封装在 List<User>集合中。

```
private List<User> pullXML(){
        //创建一个集合
        List<User> users = null;
        //实例化一个 pull 解析器
        XmlPullParser parser = Xml.newPullParser();
        //获取到 xml 数据文件
        File file = new File(getFilesDir(), FILE_PATH);
        FileInputStream inputStream;
        try {
            //获取到该 xml 文件的流对象
            inputStream = new FileInputStream(file);
            //给 pull 解析器设置流对象和编码格式
            parser.setInput(inputStream, ENCODING);
            //开始解析第一个事件类型
            int type = parser.next();
            User user = null;
            //如果不是文档的结束就一直循环
            while(type!=XmlPullParser.END_DOCUMENT){
                //遇到标签的开始事件
                if (type==XmlPullParser.START_TAG) {
                    //如果标签的名字等于 users
                    if ("users".equals(parser.getName())) {
                        //实例化集合对象
                        users = new ArrayList<User>();
                        //如果遇到 user 标签
                    }else if ("user".equals(parser.getName())) {
                        //实例化 user 对象
                        user = new User();
                        //如果遇到 name 标签
                    }else if ("name".equals(parser.getName())) {
                        //给当前 user 设置相应的属性
                        user.setUsername(parser.nextText());
                    }else if ("age".equals(parser.getName())) {

    user.setAge(Integer.valueOf(parser.nextText()));
                    }else if ("sex".equals(parser.getName())) {
                        user.setSex(parser.nextText());
                    }
```

```
//遇到标签的结束事件
            }else if (type==XmlPullParser.END_TAG) {
                //如果是以 user 结束的标签
                if ("user".equals(parser.getName())) {
                    //将当前 user 对象添加到 users 集合中
                    users.add(user);
                }
            }
            //进行下一个事件循环
            type = parser.next();
        }
        //关闭流对象
        inputStream.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
    return users;
}
```

至此，所有功能均实现完毕！