



13.2 文件存储

- Android使用的是基于Linux的文件系统
- 程序开发人员可以建立和访问程序自身的私有文件
- 也可以访问保存在资源（res）目录中的原始文件和XML文件
- 还可以在SD卡等外部存储设备中保存与读取文件





13.2 文件存储

- 文件主要用于存储大容量的数据
- 采用java.io.*库所提供的I/O接口读写文件。
- 只有本地文件可以被访问
 - 优点：可以存储大容量的数据
 - 缺点：文件更新或是格式改变可能会导致大量的编程工作





13.2 文件存储

- 13.2.1 内部存储：
 - Android系统允许应用程序创建仅能够自身访问的私有文件，文件保存在设备的内部存储器上，在Linux系统下的/data/data/<package name>/files目录中
 - Android系统不仅支持标准Java的IO类和方法，还提供了能够简化读写流式文件过程的函数
 - 主要介绍两个函数
 - openFileOutput()
 - openFileInput()





13.2 文件存储

- 13.2.1 内部存储：
 - openFileOutput()函数
 - 用于写入数据，如果指定的文件不存在，则创建一个新的文件
 - 语法格式

```
public FileOutputStream openFileOutput(String name, int mode)
```

- 第1个参数是文件名称，这个参数不能包含描述路径的斜杠
- 第2个参数是操作模式
- 函数的返回值是FileOutputStream类型





13.2 文件存储

- 13.2.1 内部存储：
 - openFileOutput()函数
 - Android系统支持四种文件操作模式

模式	说明
MODE_PRIVATE	私有模式，缺陷模式，文件仅能够被文件创建程序访问，或具有相同UID的程序访问。
MODE_APPEND	追加模式，如果文件已经存在，则在文件的结尾处添加新数据。
MODE_WORLD_READABLE	全局读模式，允许任何程序读取私有文件。
MODE_WORLD_WRITEABLE	全局写模式，允许任何程序写入私有文件。





13.2 文件存储

- 13.2.1 内部存储：

- openFileOutput()函数

使用openFileOutput()函数建立新文件的示例代码如下

```
String FILE_NAME = "fileDemo.txt";
FileOutputStream fos =
    openFileOutput(FILE_NAME, Context.MODE_PRIVATE);
String text = "Some data";
fos.write(text.getBytes());
fos.flush();
fos.close();
```

- 第1行代码定义了建立文件的名称fileDemo.txt
- 第2行代码使用openFileOutput()函数以私有模式建立文件
- 第4行代码调用write()函数将数据写入文件
- 第5行代码调用flush()函数将所有剩余的数据写入文件
- 第6行代码调用close()函数关闭FileOutputStream





13.2 文件存储

- 13.2.1 内部存储：
 - openFileOutput()函数
 - 为了提高文件系统的性能，一般调用write()函数时，如果写入的数据量较小，系统会把数据保存在数据缓冲区中，等数据量累积到一定程度时再一次性的写入文件中
 - 由上可知，在调用close()函数关闭文件前，务必要调用flush()函数，将缓冲区内所有的数据写入文件





13.2 文件存储

- 13.2.1 内部存储：
 - openFileInput()函数
 - openFileInput()函数用于打开一个与应用程序联系的私有文件输入流
 - 当文件不存在时抛出FileNotFoundException 异常
 - openFileInput()函数的语法格式如下

```
public FileInputStream openFileInput (String name)
```

- 参数是文件名称，同样不允许包含描述路径的斜杠





13.2 文件存储

- 13.2.1 内部存储：
 - openFileInput()函数
 - openFileInput ()函数打开已有文件的示例代码如下

```
String FILE_NAME = "fileDemo.txt";
FileInputStream fis = openFileInput(FILE_NAME);

byte[] readBytes = new byte[fis.available()];
while(fis.read(readBytes) != -1){
    //对读入的数据进行处理
}
```

- 上面的两部分代码在实际使用过程中会遇到错误提示，因为文件操作可能会遇到各种问题而最终导致操作失败，因此代码应该使用try/catch捕获可能产生的异常





13.2 文件存储

- 13.2.1 内部存储：示例
 - FileDemo示例用来演示在内部存储器上进行文件写入和读取,如图





13.2 文件存储

- 13.2.1 内部存储：示例核心代码FileDemo (write)

```
/*
 * 这里定义的是一个文件保存的方法，写入到文件中，所以是输出流
 */
public void save(String filename, String filecontent) throws Exception {
    //这里我们使用私有模式，创建出来的文件只能被本应用访问，还会覆盖原文件
    FileOutputStream output = mContext.openFileOutput(filename, Context.MODE_PRIVATE);
    output.write(filecontent.getBytes()); //将String字符串以字节流的形式写入到输出流中
    output.close(); //关闭输出流
}

/*
 * 这里定义的是文件读取的方法
 */
public String read(String filename) throws IOException {
    //打开文件输入流
    FileInputStream input = mContext.openFileInput(filename);
    byte[] temp = new byte[1024];
    StringBuilder sb = new StringBuilder("");
    int len = 0;
    //读取文件内容：
    while ((len = input.read(temp)) > 0) {
        sb.append(new String(temp, 0, len));
    }
    //关闭输入流
    input.close();
    return sb.toString();
}
```





13.2 文件存储

- 13.2.1 内部存储：
 - fileDemo.txt文件

[-] edu.hrbeu.InternalFileDemo		2009-07-16	02:28	drwxr-xr-x
[-] files		2009-07-16	02:31	drwxrwx---x
fileDemo.txt	9	2009-07-16	03:24	-rw-rw----
[+] lib		2009-07-16	02:28	drwxr-xr-x

- fileDemo.txt从文件权限上进行分析，“-rw-rw---”表明文件仅允许文件创建者和同组用户读写，其他用户无权使用
- 文件的大小为9个字节，保存的数据为 “Some data”





13.2 文件存储

- 13.2.2 外部存储：
 - Android的外部存储设备指的是SD卡（Secure Digital Memory Card），是一种广泛使用于数码设备上的记忆卡



- 不是所有的Android手机都有SD卡，但Android系统提供了对SD卡的便捷的访问方法





13.2 文件存储

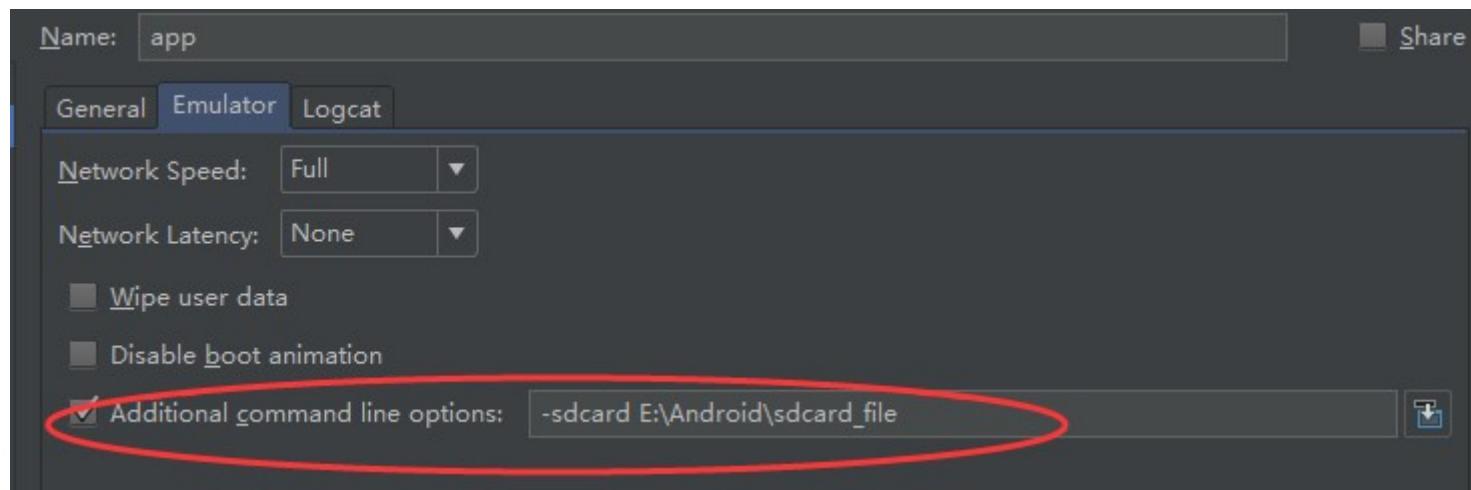
- 13.2.2 外部存储
 - SD卡适用于保存**大尺寸**的文件或者是一些无需设置访问权限的文件，可以保存录制的大容量的视频文件和音频文件等
 - SD卡使用的是FAT (File Allocation Table) 的文件系统，**不支持访问模式和权限控制**，但可以通过Linux文件系统的文件访问权限的控制保证文件的私密性
 - Android模拟器支持SD卡，但模拟器中没有缺省的SD卡，开发人员须在模拟器中手工添加SD卡的映像文件





13.2 文件存储

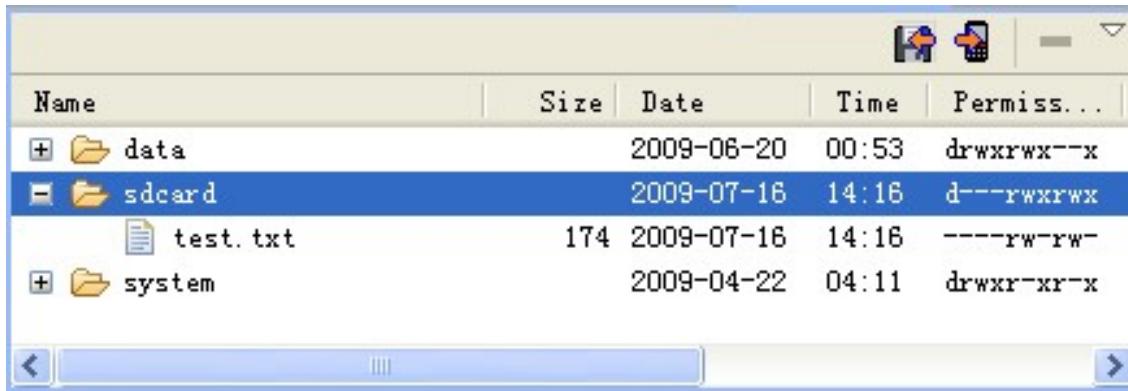
- 13.2.2 外部存储
 - 如果希望Android模拟器启动时能够自动加载指定的SD卡，还需要在模拟器的“运行设置”（Run Configurations）中添加SD卡加载命令
 - SD卡加载命令中只要指明映像文件位置即可
 - SD卡加载命令





13.2 文件存储

- 13.2.2 外部存储
 - 测试SD卡映像是否正确加载
 - 在模拟器启动后，使用FileExplorer向SD卡中随意上传一个文件，如果文件上传成功，则表明SD卡映像已经成功加载
 - 向SD卡中成功上传了一个测试文件test.txt，文件显示在/sdcard目录下





13.2 文件存储

- 13.2.2 外部存储
 - 编程访问SD卡
 - 首先需要检测系统的/sdcard目录是否可用
 - 如果不可用，则说明设备中的SD卡已经被移除，在Android模拟器则表明SD卡映像没有被正确加载
 - 如果可用，则直接通过使用标准的Java.io.File类进行访问
 - 将数据保存在SD卡
 - FileDemo2示例说明了如何将数据保存在SD卡





13.2 文件存储

- 13.2.2 外部存储
 - 下图是FileDemo2示例的用户界面





13.2 文件存储

- 13.2.2 外部存储
 - FileDemo2示例中，在每次点击“保存到SD卡”按钮后，都会在SD卡中生产一个新文件，文件名各不相同
 - SD卡中生产的文件

Name	Size	Date	Time	Permiss...
+ data		2009-06-20	00:53	drwxrwx--x
- sdcard		2009-07-16	15:17	d---rwxrwx
SdcardFile-1247755731168.txt	193	2009-07-16	14:48	-----rw-rw-
SdcardFile-1247755731369.txt	193	2009-07-16	14:48	-----rw-rw-
SdcardFile-1247755774895.txt	187	2009-07-16	14:49	-----rw-rw-
SdcardFile-1247756083152.txt	187	2009-07-16	14:54	-----rw-rw-
SdcardFile-1247756146017.txt	187	2009-07-16	14:55	-----rw-rw-
SdcardFile-1247756147340.txt	187	2009-07-16	14:55	-----rw-rw-
test.txt	174	2009-07-16	14:16	-----rw-rw-
+ system		2009-04-22	04:11	drwxr-xr-x





13.2 文件存储

- 13.2.2 外部存储
 - FileDemo2示例与FileDemo示例的核心代码比较相似
 - FileDemo2示例与FileDemo示例的不同之处，在下文中的注释中会加以说明。





13.2 文件存储

- 13.2.2 外部存储：FileDemo2示例核心代码

```
//往SD卡写入文件的方法
public void saveFileToSD(String filename, String filecontent) throws Exception {
    //如果手机已插入sd卡,且app具有读写sd卡的权限
    if (Environment.getExternalStorageState().equals(Environment.MEDIA_MOUNTED)) {
        filename = Environment.getExternalStorageDirectory().getCanonicalPath() + "/" + filename;
        //这里就不要用openFileOutput了,那个是往手机内存中写数据的
        FileOutputStream output = new FileOutputStream(filename);
        output.write(filecontent.getBytes());
        //将String字符串以字节流的形式写入到输出流中
        output.close();
        //关闭输出流
    } else Toast.makeText(context, "SD卡不存在或者不可读写", Toast.LENGTH_SHORT).show();
}

//读取SD卡中文件的方法
//定义读取文件的方法:
public String readFromSD(String filename) throws IOException {
    StringBuilder sb = new StringBuilder("");
    if (Environment.getExternalStorageState().equals(Environment.MEDIA_MOUNTED)) {
        filename = Environment.getExternalStorageDirectory().getCanonicalPath() + "/" + filename;
        //打开文件输入流
        FileInputStream input = new FileInputStream(filename);
        byte[] temp = new byte[1024];

        int len = 0;
        //读取文件内容:
        while ((len = input.read(temp)) > 0) {
            sb.append(new String(temp, 0, len));
        }
        //关闭输入流
        input.close();
    }
    return sb.toString();
}
```





13.2 文件存储

- 13.2.2 外部存储：FileDemo2示例核心代码

```
<!-- 在SDCard中创建与删除文件权限 -->
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
<!-- 往SDCard写入数据权限 -->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

- PS：需要注意，在Android6.0及以上版本，权限管理更加严格，需要动态申请SD卡权限，如下：

```
private static final int REQUEST_EXTERNAL_STORAGE = 1;
private static String[] PERMISSIONS_STORAGE = {
    "android.permission.READ_EXTERNAL_STORAGE",
    "android.permission.WRITE_EXTERNAL_STORAGE" };
public static void verifyStoragePermissions(Activity activity) {
    try {
        //检测是否有权限
        int permission = ActivityCompat.checkSelfPermission(activity,
            "android.permission.WRITE_EXTERNAL_STORAGE");
        if (permission != PackageManager.PERMISSION_GRANTED) {
            // 没有权限就去申请权限，会弹出对话框
            ActivityCompat.requestPermissions(activity, PERMISSIONS_STORAGE, REQUEST_EXTERNAL_STORAGE);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```





13.2 文件存储

- 13.2.3 资源文件
 - 程序开发人员可以将程序开发阶段已经准备好的原始格式文件和XML文件分别存放在`/res/raw`和`/res/xml`目录下，供应用程序在运行时进行访问
 - 原始格式文件可以是任何格式的文件，例如视频格式文件、音频格式文件、图像文件和数据文件等等，在应用程序编译和打包时，`/res/raw`目录下的所有文件都会保留原有格式不变





13.2 文件存储

- 13.2.3 资源文件
 - /res/xml目录下的XML文件，一般用来保存格式化的数据，在应用程序编译和打包时会将XML文件转换为高效的二进制格式，应用程序运行时会以特殊的方式进行访问





13.2 文件存储

- 13.2.3 资源文件：如何读取原始格式文件？
 - 首先调用getResource()函数获得资源对象
 - 然后通过调用资源对象的openRawResource()函数，以二进制流的形式打开文件
 - 在读取文件结束后，调用close()函数关闭文件流

```
InputStream input = this.getResources().openRawResource(R.raw.filename);

byte[] reader = new byte[inputStream.available()];
while (inputStream.read(reader) != -1) {}
txt_text.setText(new String(reader, "utf-8")); //获得Context资源

input.close(); //关闭输入流
```





13.2 文件存储

- 13.2.3 资源文件
 - /res/xml目录下的XML文件会转换成高效的二进制格式
 - 在程序运行时读取/res/xml目录下的XML文件
 - 在/res/xml目录下创建一个名为toys.xml的文件
 - XML文件定义了多个<toy>元素，每个<toy>元素都包含2个属性name和price，表示姓名和价格

```
<toys>
<toy name = "Winnie" price = "100" />
<toy name = "Teddy" price = "125" />
<toy name = "BearBear" price = "120" />
</toys>
```





13.2 文件存储

- 13.2.3 资源文件
 - 读取XML格式文件
 - 首先通过调用资源对象的getXml()函数，获取到XML解析器XmlPullParser
(XmlPullParser是Android平台标准的XML解析器，这项技术来自一个开源的XML解析API项目XMLPULL)

```
XmlPullParser parser = resources.getXml(R.xml.toys);
//通过资源对象的getXml()函数获取到XML解析器
while (parser.next() != XmlPullParser.END_DOCUMENT) {
    String toys = parser.getName(); //getName()函数获得元素的名称
    //查看元素名是否匹配
    if ((toys != null) && toys.equals("toy")) {
        .... /*逐个元素地读取属性名和属性值（本例中的toys.xml共有3个
               元素），并通过分析属性名获取到正确的属性值*/
    }
}
```



ANDROID



13.2 文件存储

- 13.2.3 资源文件

- 如何获取元素个数

```
int count = parser.getAttributeCount();
```

- 如何获得属性名和属性值

```
String attrName = parser.getAttributeName(i); //获得属性名  
String attrValue = parser.getAttributeValue(i); //获得属性值
```

- 如何分析已获得的属性名和属性值

```
//通过分析属性名获取到正确的属性值  
if ((attrName != null) && attrName.equals("name")) {  
    name = attrValue;  
}  
else if ((attrName != null) && attrName.equals("price")) {  
    price = attrValue;  
}
```





13.2 文件存储

- 13.2.3 资源文件
 - XmlPullParser的XML事件类型

事件类型	说明
START_TAG	读取到标签开始标志
TEXT	读取文本内容
END_TAG	读取到标签结束标志
END_DOCUMENT	文档末尾

- 读取文件过程中遇到END_DOCUMENT时停止分析

