



**SUN YAT-SEN UNIVERSITY**

中山大學



# 手机应用平台软件开发

---

## 9、数据存储(一)

---



## Android有4种数据存取方式

- SharedPreferences

轻量级键-值方式存储，以XML文件方式保存。

- 文件

采用java.io.\*库所提供有I/O接口，读写文件。

- SQLite数据库

轻量级嵌入式内置数据库。

- ContentProvider

封装各种数据源（文件、数据库、网络），共享给多个应用。

---



### 应用场景

- 保存播放位置：用手机播放器播放音乐，希望重启播放器时，播放器能从上次停止的那首曲目开始播放；
  - 自动登录：记住登录用户名（密码）
  - 其他应用.....
-



# SharedPreferences

SUN YAT-SEN UNIVERSITY

- 用来存储简单数据的工具类，与Cookie的概念相似，通过用键值对的方式把简单的数据存储在应用程序的私有目录（data/data/<packagename>/shared\_prefs/）下指定的xml文件中。
  - 保存现场：保存用户所作的修改或者自定义参数设定，当再次启动程序后回复上次退出时的状态。
  - 应用程序有少量的数据要保存，而且这些数据的格式很简单，都是普通的字符串、数值等（属性设置、较简单的参数设置）。
-



# SharedPreferences

SUN YAT-SEN UNIVERSITY

- 将NVP（Name/Value Pair，名称/值对）保存在Android的文件系统中（XML文件），完全屏蔽的对文件系统的操作过程。
  - NVP举例：(姓名,张三), (性别,男), (年龄,30), ...
  - 开发人员仅是通过调用SharedPreferences的API对NVP进行保存和读取.
-



# SharedPreferences

SUN YAT-SEN UNIVERSITY

- SharedPreferences本身仅仅为一个接口，程序无法直接创建SharedPreferences实例，只能通过Context提供的getSharedPreferences(String name, **int** mode)方法来获取SharedPreferences的实例；
  - 第二个参数表示支持的3种数据访问模式（读写权限）
    1. 私有（MODE\_PRIVATE）：仅创建程序可读、写
    2. 全局读（MODE\_WORLD\_READABLE）：创建程序可读写，其他程序可读不可写
    3. 全局写（MODE\_WORLD\_WRITEABLE）：创建程序和其他程序都可写，但不可读！

(后两种模式可组合，用+号或|号.)
  - 支持的数据格式：boolean, float, int, long, String
-



# SharedPreferences

SUN YAT-SEN UNIVERSITY

## SharedPreferences接口主要方法

- `contains (String key)` : 检查是否已存在key这个关键字。
  - `getAll()`: 返回preferences所有的数据（Map）。
  - `getBoolean(String key, boolean defValue)`: 获取Boolean型数据
  - `getFloat(String key, float defValue)`: 获取Float型数据
  - `getInt(String key, int defValue)`: 获取Int型数据
  - `getLong(String key, long defValue)`: 获取Long型数据
-



# SharedPreferences

SUN YAT-SEN UNIVERSITY

SharedPreferences接口本身并没有提供写入数据的能力，而是通过SharedPreferences的内部接口，SharedPreferences调用edit（）即可获得它所对应的Editor对象，该对象主要提供以下方法

- Clear（），清空SharedPreferences里所有数据；
  - put\*\*\*（String key，\*\*\* Value）；
  - Remove（String key）：删除SharedPreferences里指定的key对应的项；
  - commit（）：当Editor编辑完成后，调用该方法提交修改；
-





### 第1步：定义访问模式

- 下面的代码将访问模式定义为私有模式

```
public static int MODE = MODE_PRIVATE;
```

- 访问模式可组合：既可以全局读，也可以全局写，将两种模式组合（+号或|号）成下面的方式：

```
public static int MODE = Context.MODE_WORLD_READABLE  
+ Context.MODE_WORLD_WRITEABLE;
```



# SharedPreferences

SUN YAT-SEN UNIVERSITY

## 第2步：定义SharedPreferences的名称

- 该名称与Android文件系统中保存的XML文件同名。
- (保存在:/data/data/<package name>/shared\_prefs/)
- 相同名称的NVP内容，都会保存在同一个文件中。

```
public static final String PREFERENCE_NAME = "SaveSetting";
```



### 第3步：创建SharedPreferences对象

- 将访问模式和名称作为参数，传递到

getSharedPreferences()函数，并获得SharedPreferences

对象

```
SharedPreferences sharedPreferences =  
getSharedPreferences(PREFERENCE_NAME,MODE);
```

其中文件名称，不需要加后缀.xml，系统会自动加上。

---



### 第4步：修改与保存

- 通过SharedPreferences.Editor类进行修改
- 调用commit()函数保存修改内容

支持数据类型：整型、布尔型、浮点型和长整型等

```
1. SharedPreferences.Editor editor = sharedPreferences.edit();  
2. editor.putString("Name", "Tom");  
3. editor.putInt("Age", 20);  
4. editor.putFloat("Height", (float)163.00 );  
5. editor.commit();
```



### 第5步：读取数据

➤ （先调用getSharedPreferences()函数获得对象）

➤ 通过get<Type>()函数获取NVP

1. 第1个参数是NVP的名称(Name)

2. 第2个参数是在无法获取到数值的时候使用的缺省值

```
1. SharedPreferences sharedPreferences =getSharedPreferences(PREFERENCE_NAME,
    MODE);
2. String name = sharedPreferences.getString("Name","Default Name");
3. int age = sharedPreferences.getInt("Age", 20);
```



# SharedPreferences

SUN YAT-SEN UNIVERSITY

```
public class SharedPreferencesDemo extends Activity {
    SharedPreferences preferences;
    SharedPreferences.Editor editor;
    EditText name,pwd;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_shared_preferences_demo);
        // 获取只能被本应用程序读、写的SharedPreferences对象
        preferences = getSharedPreferences("demo", MODE_WORLD_READABLE);
        editor = preferences.edit();
        Button save = (Button) findViewById(R.id.save);
        Button read = (Button) findViewById(R.id.read);
        name=(EditText)findViewById(R.id.name);
        pwd=(EditText)findViewById(R.id.password);
        read.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                // 读取字符串数据
                String sName = preferences.getString("name", null);
                String sPwd = preferences.getString("pwd",null);
                if (sName==null) { //相关处理
                }
                else
                {
                    name.setText(sName);
                    pwd.setText(sPwd);}}});
        save.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                editor.putString("name", name.getText().toString());
                editor.putString("pwd", pwd.getText().toString());
                // 提交所有存入的数据
                editor.commit();
            }});
    }
}
```



### SharedPreferences实现方式

- 用Preferences来存取数据，这些数据究竟被保存在手机的什么地方？
  - 每安装一个应用程序时，SharedPreferences文件就保存在  
`/data/data/<package name>/shared_prefs/`  
目录下，其中的就是数据文件。
-



### SharedPreferences调试方式

- 如何读取程序SharedPreferences的数据？
- 可通过DDMS的FileExplorer查看/data/data下的数据，Android为每个应用程序建立了与包同名的目录，用来保存应用程序产生的数据，这些数据包括文件、SharedPreferences文件和数据库等

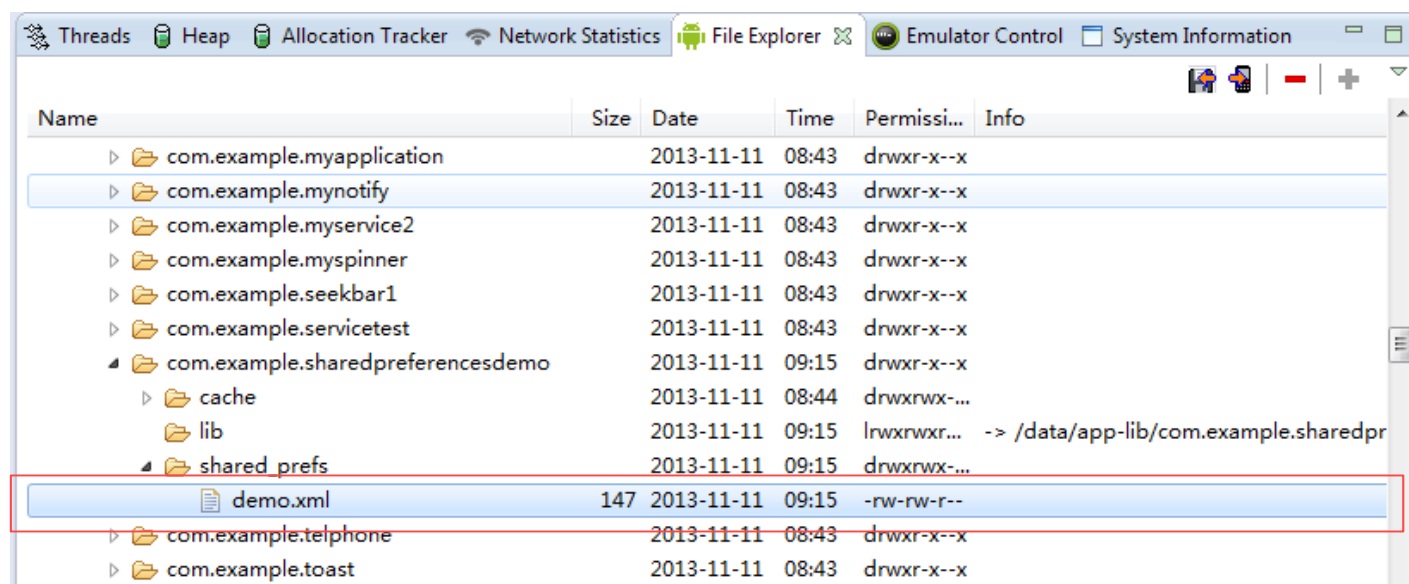




# SharedPreferences

SUN YAT-SEN UNIVERSITY

com.example.sharedpreferencesdemo的程序，其shared\_prefs目录下生成了一个名为demo.xml的文件



Name	Size	Date	Time	Permissi...	Info
com.example.myapplication		2013-11-11	08:43	drwxr-x--x	
com.example.mynotify		2013-11-11	08:43	drwxr-x--x	
com.example.myservice2		2013-11-11	08:43	drwxr-x--x	
com.example.myspinner		2013-11-11	08:43	drwxr-x--x	
com.example.seekbar1		2013-11-11	08:43	drwxr-x--x	
com.example.servicetest		2013-11-11	08:43	drwxr-x--x	
com.example.sharedpreferencesdemo		2013-11-11	09:15	drwxr-x--x	
cache		2013-11-11	08:44	drwxrwx--x	
lib		2013-11-11	09:15	lrwxrwxr--	-> /data/app-lib/com.example.sharedpr
shared_prefs		2013-11-11	09:15	drwxrwx--x	
demo.xml	147	2013-11-11	09:15	-rw-rw-r--	
com.example.telphone		2013-11-11	08:43	drwxr-x--x	
com.example.toast		2013-11-11	08:43	drwxr-x--x	

该文件就是保存SharedPreferences的文件，文件大小为147字节，在Linux下的权限为“-rw-rw-r--”，文件名取决于之前的代码：

```
preferences = getSharedPreferences("demo", MODE_WORLD_READABLE)
```

---



### 文件权限说明

- 在Linux系统中，文件权限描述符每3个字符分别描述了创建者（第1—3字符）、同组用户（第4—6字符）和其他用户（第7—9字符）对文件的操作限制（权限）；
  - 每字符意义：x表示可执行，r表示可读，w表示可写，d表示目录，-表示普通文件(无操作权限)；
  - “-rw-rw-r--”表示demo.xml可以被创建者、同组用户读取和写入操作，其他用户进行可以读，但不可写；
  - 产生这样的文件权限与程序人员设定的SharedPreferences的访问模式有关，“-rw-rw-r--”的权限是“MODE\_WORLD\_READABLE”的结果；
-



### 数据文件格式

demo.xml文件是以XML格式保存的信息，内容如下

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>  
<map>  
  <string name="pwd">John 123456789</string>  
  <string name="name">John</string>  
</map>
```



### 文件共享

如何可以获取其他应用保存的SharedPreferences数据？

- 需要在创建该SharedPreferences的应用程序中写明：可被其它程序读取的权限（访问模式设置为全局读或全局写）；
  - 获取其它程序的Context，Context代表了访问该Android应用的全局信息接口，而Android应用的包名正是该应用的唯一标识，因此可以根据Android应用的包名获取相应的Context；
  - 访问者需要确切知道每个数据的名称和数据类型，用以正确读取数据
-



# SharedPreferences

SUN YAT-SEN UNIVERSITY

## ■ SharedPreferencesDemo示例的核心代码

```
1.  public static final String PREFERENCE_PACKAGE =  
        "com.example.sharedpreferencesdemo";  
2.  public static int MODE = Context.MODE_WORLD_READABLE  
        + Context.MODE_WORLD_WRITEABLE;  
3.  public static final String PREFERENCE_NAME = "demo";  
4.  public void onCreate(Bundle savedInstanceState) {  
5.      Context c = null;  
6.      try {  
7.          //获取SimplePreferenceDemo示例的Context  
8.          c = this.createPackageContext(PREFERENCE_PACKAGE,  
                Context.CONTEXT_IGNORE_SECURITY);  
9.      } catch (NameNotFoundException e) {e.printStackTrace();}  
10. //将正确的SharedPreferences名称传递给函数  
11.     SharedPreferences sharedPreferences =  
        c.getSharedPreferences( PREFERENCE_NAME, MODE );
```

(新) 定义要访问的应用的名称

第1-2步: 定义模式、名称

(新) 创建要访问的应用的上下文

第3步: 得到上下文的SP



# SharedPreferences

SUN YAT-SEN UNIVERSITY

## 示例

```
String name = sharedPreferences.getString("Name", "Tom");  
    int age = sharedPreferences.getInt("Age", 20);  
    float height = sharedPreferences.getFloat("Height", 1.75f);  
}
```

第5步:  
读NVP

访问共享设置的额外步骤

- 第1行定义要访问的应用的包名
- 第8行代码调用了createPackageContext()获取到了要访问的应用的上下文Context
  1. 第1个参数是要访问的应用的包名称
  2. 第2个参数Context.CONTEXT\_IGNORE\_SECURITY表示忽略所有可能产生的安全问题。
  3. 这段代码可能引发异常，因此必须放在try/catch中。



# SharedPreferences

SUN YAT-SEN UNIVERSITY

访问共享设置的额外步骤（续）

- 在代码第11行，通过Context得到要访问应用的SharedPreferences对象。
    1. （有别于访问自己配置过程的第3步）
    2. 同样在getSharedPreferences()函数中，需要将正确的SharedPreferences名称、访问模式传递给函数。
-



# SharedPreferences

SUN YAT-SEN UNIVERSITY

- 现有某Activity专门用于手机属性设置 那么应该如何做呢？
- 根据已学知识，使用：Activity + Preference 组合，前者用于界面构建 后者用于设置数据存放，但是这会比较繁琐。
- 因为每个设置选项都要建立与其对应的Preference，所以现在有更好的选择PreferenceActivity 从名字应该可以看出其实 Activity与 Preference 的混合物；







# SharedPreferences

SUN YAT-SEN UNIVERSITY

## PreferenceActivity类

实现一个程序参数设置的UI界面，管理SharedPreferences

- 使用SharedPreferences以键值对的形式保存数据
  - 可以通过SharedPreferences获取PreferenceActivity设置的值
-



# PreferenceActivity

SUN YAT-SEN UNIVERSITY

- 步骤1：编写preference XML，在res/xml/下加入preference XML文件，

main.xml

```
<?xml version="1.0" encoding="utf-8" ?>
```

<!-- preference的组织方式有PreferenceScreen和PreferenceCategory，PreferenceCategory是带层次组织关系，在后面的例子体验，而PreferenceScreen就是最平白和基础的方式 -->

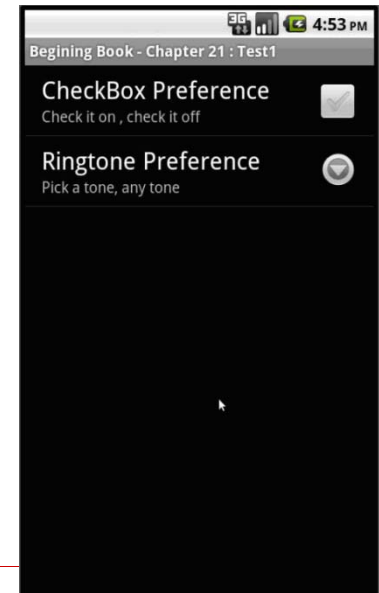
```
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android" >
```

<!-- 有四个基本组件，这个例子学习两个。里面的内容对照图，很容易理解，RingtonePreference是选择铃声，这里给出两个选择，系统默认的铃声和无声 -->

```
<CheckBoxPreference android:key="checkbox"
    android:title="CheckBox Preference"
    android:summary="Check it on , check it off"/>
```

```
<RingtonePreference android:key="ringtone"
    android:title="Ringtone Preference"
    android:showDefault="true"
    android:showSilent="true"
    android:summary="Pick a tone, any tone"/>
```

```
</PreferenceScreen>
```





# PreferenceActivity

SUN YAT-SEN UNIVERSITY

- **步骤2:** 在java源代码中调用该xml, 生成相应的preference界面

```
public class MyActivity extends PreferenceActivity{  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        addPreferencesFromResource(R.xml.main);}
```

与以往layout方式一样, 代码很简单。和以往界面最大的区别是, 用户对checkbox进行选择或者对ringtone进行选择后, 选择结果是被保留的, 当退出activity后再次进入, 上次选项依然保留。利用preference, 而无须人工对数据进行保存和读取。

---



### ➤ 步骤3：获取preference的数据

除了在preference的界面中保存数据，也希望能够读出数据，以便可以用在其他的activity中使用。若另一个activity和这些数据关联，而且希望能够实时进行同步，在onResume()中对preference保留的数据进行读取，而后进行相关的更新。

首先要获取preferences，然后通过键值对的获取方式根据key获取数值，在xml中checkbox的key为“checkbox”，值的类型为布尔值，而ringtone的值为String；

```
protected void onResume() {  
    //在另一个Activity中  
    super.onResume();  
    SharedPreferences prefs =PreferenceManager.getDefaultSharedPreferences(this) ;  
    System.out.println(new Boolean(prefs.getBoolean("checkbox",false)).toString());  
    //false表示没有查到checkbox这个key的返回值  
    System.out.println(prefs.getString("ringtone","<unset>"));  
    //<unset>表示没有查到ringtone这个key的返回值  
}
```

---



# 获取preferences的三种方式对比

SUN YAT-SEN UNIVERSITY

1. `PreferenceManager.getDefaultSharedPreferences(context);`
2. `getSharedPreferences(name, mode);`
3. `getPreferences(mode);`

- 通过**Android**的偏好管理器来获取其所管理的**preferences**。本方法实际上调用第二个方法  
`context.getSharedPreferences(getDefaultSharedPreferencesName(context),getDefaultSharedPreferencesMode())`,参数为`getSharedPreferences(<package_name>_preferences,model);`  
`getDefaultSharedPreferencesName(context)` 就是调用`context.getPackageName() + "_preferences"`, 其  
Preference功能范围在当前项目包下, 超出此范围Preference功能 就无效了;
  - **Context**类中的方法, 可以指定file name 以及 mode, 可以获取应用级别的preferences();
  - **Activity**类中的方法, 只需指定mode, 可以获取同一Activity中的preference, 本方法Preference功能范围只在同一  
activity中的preference, 如果`getPreferences(mode)`方法不是和`addPreferencesFromResource(R.xml.main)`在同  
一个activity 中的话, Preference功能也是无效的, 这个方法默认使用当前类不带包名的类名作为文件的名称;
-



### Preference数据的操作

在上面的例子中，**preference**的数值会被保留，而且可以在其他的**activity**中读取。

如果需要清除数据，可以通过**remove()**清除某个名字的**prefernece**，**clear()**清除

所有的**preferences**。我们可以通过**edit()**获取**preferences**的**editor**，进而进行编

辑，修改后，通过**commit()**将修改值保存。

---

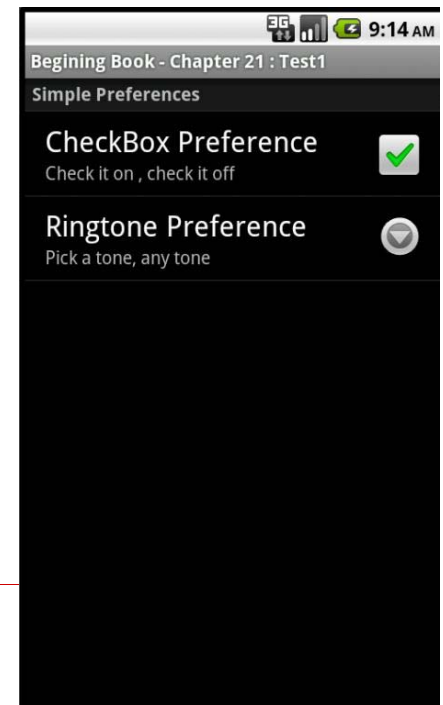


# Preference XML

SUN YAT-SEN UNIVERSITY

Preference的XML可以通过PreferenceCategory来进行组织。PreferenceCategory可以将几个组建组合在一起，并加上标题。用前面的xml例子，通过PreferenceCategory来进行组织。和之前的例子比较，将checkbox和ringtone组织成为一个category，并加上"simple Preferences"的标题。

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
    <PreferenceCategory android:title="Simple Preferences">
        <CheckBoxPreference android:key="checkbox" ..../>
        <RingtonePreference android:key="ringtone" ... />
    </PreferenceCategory>
</PreferenceScreen>
```





# Preference XML

SUN YAT-SEN UNIVERSITY

`PreferenceCategory`也可以嵌套`PreferenceScreen`，`PreferenceScreen`中的内容，将通过另一屏来显示，在上面例子后面添加一个嵌套了`PreferenceScreen`的`PreferenceCategory`。整个`PreferenceScreen`作为一个组件出现，点击后新的一屏，由`PreferenceScreen`定义。通过这个关系，可以组织自己的`preference`架构。

---





# Preference XML

SUN YAT-SEN UNIVERSITY

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<PreferenceScreen
```

```
  xmlns:android="http://schemas.android.com/apk/res/android">
```

```
    <PreferenceCategory android:title="Simple Preferences">
```

```
      <CheckBoxPreference android:key="checkbox" ..../>
```

```
      <RingtonePreference android:key="ringtone" ... />
```

```
    </PreferenceCategory>
```

```
    <PreferenceCategory android:title="Detail Screens">
```

```
      <PreferenceScreen android:title="Detail"
```

```
        android:summary="Additional preference help in another page">
```

```
        <CheckBoxPreference android:key="checkbox2"
```

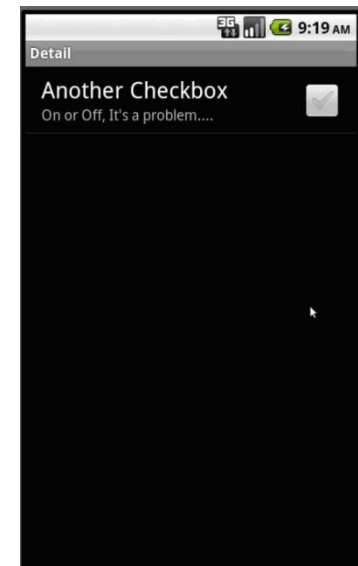
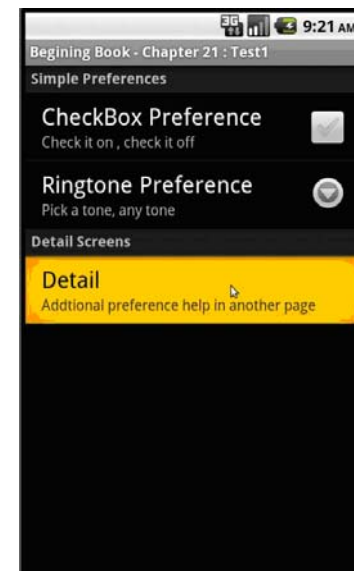
```
          android:title="Another Checkbox"
```

```
          android:summary="On or Off, It's a problem...."/>
```

```
        </PreferenceScreen>
```

```
      </PreferenceCategory>
```

```
    </PreferenceScreen>
```





# Preference XML

SUN YAT-SEN UNIVERSITY

采用另外的两个组件**EditText**和**List**，这两种都是以弹框的方式

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen ...>
    .....
    <PreferenceCategory android:title="Other Preferences">
        <EditTextPreference android:key="text"
            android:title="Text Entry Dialog"
            android:summary="Click to pop up a field for entry"
            android:dialogTitle="Enter something useful"/>
        <ListPreference android:key="list"
            android:title="Selection Dialog"
            android:summary="Click to pop yo a list for select"
            android:entries="@array/cities"
            android:entryValues="@array/airport_codes"
            android:dialogTitle="Choose a City"/>
    </PreferenceCategory>
</PreferenceScreen>
```

在**list**中由两个引用**@array/cities**和**@array/airport\_codes**，在资源文件中定义此两**array**。内容如下：

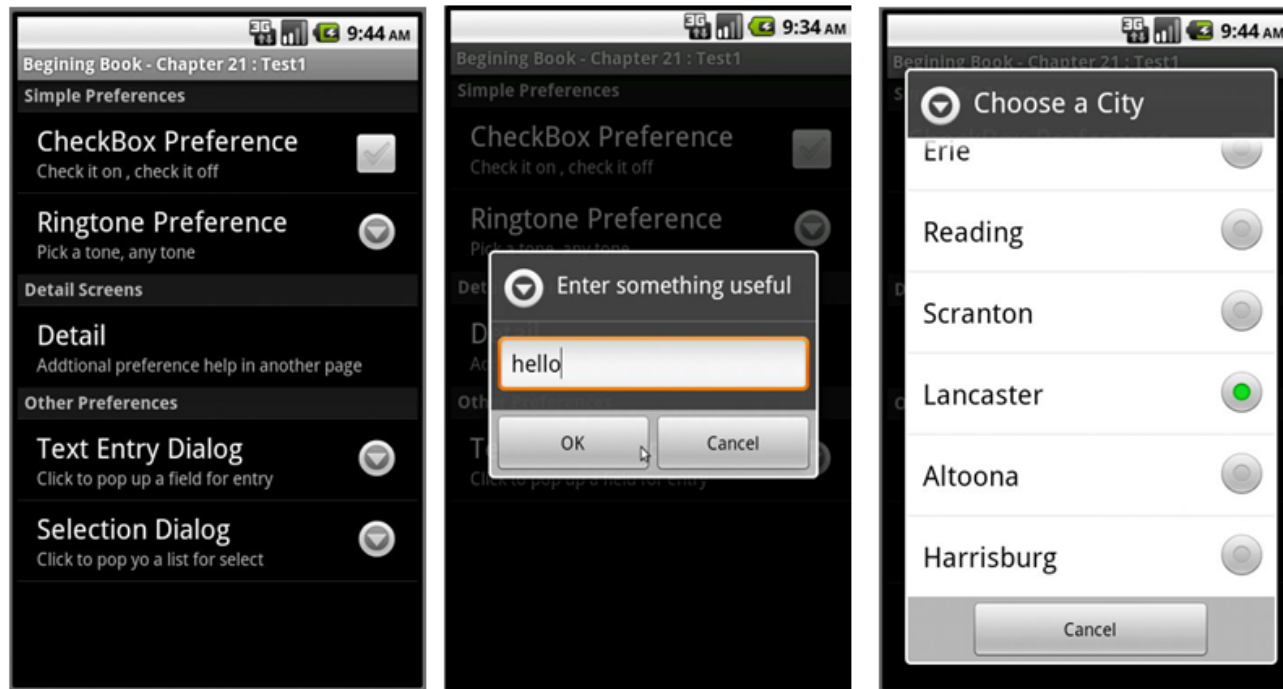
```
<resource>
    <string-array>
        <item>Pittsburgh</item>
        <item>Allentown/Bethlehem</item>
        <item>Erie</item>
        ... ..
    </string-array>
    <string-array name="airport_codes">
        <item>PHL</item>
        <item>PIT</item>
        <item>ABE</item>
        ... ..
    </string-array>
</resources>
```



# Preference XML

SUN YAT-SEN UNIVERSITY

运行结果如下图所示。对于**EditText**，键值对保存的值是**String**，即用户在输入框中输入的内容。**List**的情况稍微复杂一些，在设置**ListPreference**的属性有一个**entries**，这是在**List**显示给用户看的内容，如果去获取**preference**的值，在这个例子为 `prefs.getString( "list" , "<unset>" );`，则返回在**entryValues**对应的数值，例如选择了第一个item: **Pittsburgh**，则获取的值为**PHL**。**entries**和**entryValues**是一一对应的。





# 文件存储

SUN YAT-SEN UNIVERSITY

- 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
-



# 文件存储

SUN YAT-SEN UNIVERSITY

- 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
    - 1. 优点：可以存储大容量的数据
    - 2. 缺点：文件更新或是格式改变可能会导致大量的编程工作
-



## 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
    1. `openFileOutput()`
    2. `openFileInput()`
-



## 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)
```

1. 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
  2. 第2个参数是操作模式
  3. 函数的返回值是FileOutputStream类型
-



### 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，缺省模式，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。

- 默认的写入操作会覆盖源文件的内容，若要把新写入的内容附加在原文件的内容之后，可以指定模式为Context.MODE\_APPEND。
- 默认使用openFileOutput方法打开的文件只能被其调用的应用程序使用，其它应用程序将无法读取这个文件。





### openFileOutput()函数

使用openFileOutput()函数建立新文件的示例代码

```
1. String FILE_NAME = "fileDemo.txt";  
2. FileOutputStream fos = openFileOutput(FILE_NAME,Context.MODE_PRIVATE)  
3. String text = "Some data";  
4. fos.write(text.getBytes());  
5. fos.flush();  
6. 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



## openFileOutput()函数（获得FileInputStream）

- 为了提高文件系统的性能，一般在调用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
  - 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
-



## FileOutputStream对象的Write方法

- `write(byte[] buffer);` 将字节数组buffer写入输出流
- `write(int b);` 将整数b写入输出流;

## FileInputStream对象的Read方法

- `read ()` 从此输入流中读取一个字节;
  - `read (byte[] b)` 从输入流中将最多b.length字节的数据读入一个byte数组中;
  - `Read (byte[] b,int off,int len)` 从此输入流中将最多len个字节数据读入byte数组中
-



## 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



### openFileInput()函数

➤ openFileInput ()函数打开已有文件的示例代码

```
1. String FILE_NAME = "fileDemo.txt";  
2. FileInputStream fis = openFileInput(FILE_NAME);  
3.  
4. byte[] readBytes = new byte[fis.available()];  
5. while(fis.read(readBytes) != -1){  
6.     //对读入的数据进行处理  
7. }
```

➤ 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

---



# FileNotFoundException

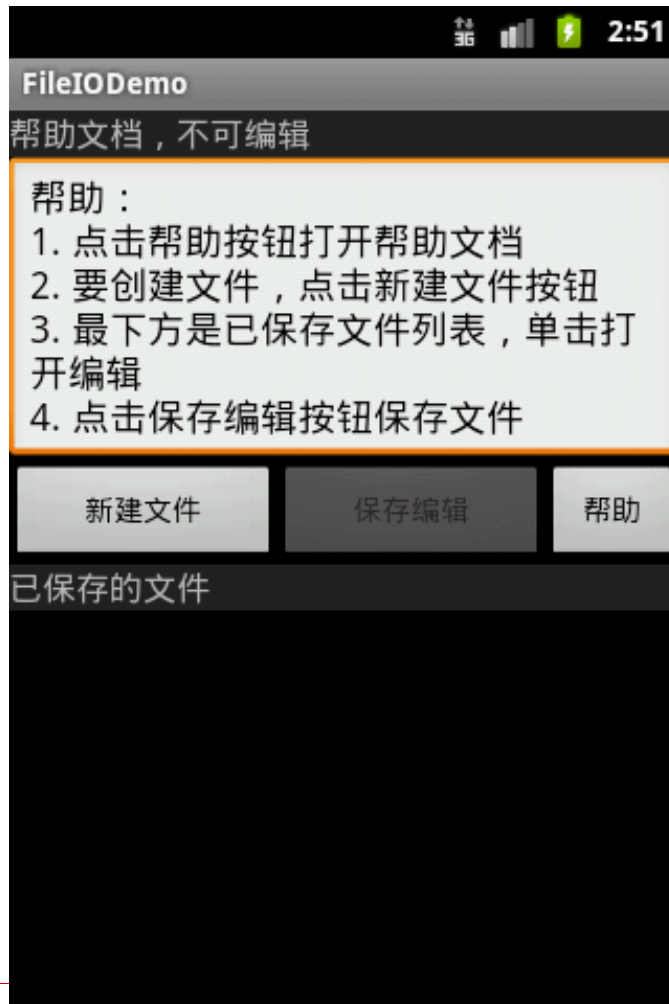
SUN YAT-SEN UNIVERSITY

- 当程序试图打开指定路径名表示的文件失败时，抛出此异常。
  - 在不存在具有指定路径名的文件时，此异常将由 [FileInputStream](#)、[FileOutputStream](#) 和 [RandomAccessFile](#) 构造方法抛出。如果该文件存在，但是由于某些原因不可访问，比如试图打开一个只读文件进行写入，此时这些构造方法仍然会抛出该异常。
-



# Files示例

SUN YAT-SEN UNIVERSITY





# Files示例

SUN YAT-SEN UNIVERSITY







## Files示例代码说明—savefile方法

SUN YAT-SEN UNIVERSITY

```
protected void savefile() throws IOException {  
    FileOutputStream fos = new FileOutputStream(mTextFile);  
    fos.write(et.getText().toString().getBytes());  
    fos.flush();  
    fos.close();  
}
```

该方法用于保存文件。保存文件的过程就是先使用FileOutputStream创建输出流，然后获取待写入到文件中的数据并写入文件中。FileOutputStream写文件的方法是使用write()方法，使用flush()方法保证输出流写入完成，最后使用close()方法关闭输出流，文件保存完毕。



## Files示例代码说明—helpdoc方法

SUN YAT-SEN UNIVERSITY

```
private void helpdoc() throws IOException{
    save.setClickable(false);
    save.setEnabled(false);
    tw.setText("帮助文档，不可编辑");
    String myString = null;
    InputStream is= getApplicationContext().getContentResolver()
        .openInputStream(Uri.parse("android.resource://" +
            "com.android.example.fileiodemo/" + R.raw.help));
    BufferedInputStream bis = new BufferedInputStream(is);
    ByteBuffer baf = new ByteBuffer(8192);
    int current = 0;
    while((current = bis.read()) != -1) {
        baf.append((byte)current);
    }
    myString = new String(baf.toByteArray(),"GBK");
    et.setText(myString);
}
```

显示该程序的帮助文档。



## Files示例代码说明—readfile方法

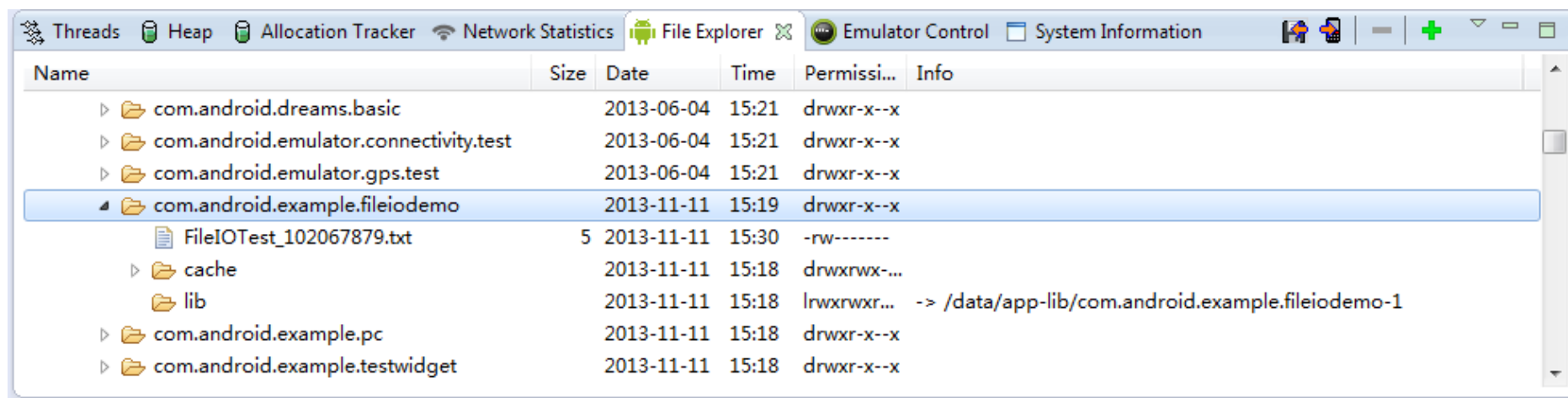
SUN YAT-SEN UNIVERSITY

```
private void readfile(File file) throws IOException{
    mTextFile = file;
    tw.setText("正在编辑文件" + mTextFile.getName());
    if(!mTextFile.exists()){
        Log.v("duanhong", "创建文件");
        if(!mTextFile.createNewFile()){
            Log.v("duanhong", "创建文件失败");
            return;
        }
    }
    String myString = null;
    InputStream is = new FileInputStream(mTextFile); //创建写入流
    BufferedInputStream bis = new BufferedInputStream(is);
    ByteBuffer baf = new ByteBuffer(8192);
    int current = 0;
    while((current = bis.read()) != -1) {
        baf.append((byte)current);
    }
    myString = new String(baf.toByteArray());
    et.setText(myString);
}
```

用于打开文件。该例中文件打开需要的步骤是使用FileInputStream得到待打开文件的输入流，然后从输入流中读出所包含的数据内容并显示到文本框中。



### ➤ FileIOTest\*\*\*.txt文件



Name	Size	Date	Time	Permissi...	Info
▶ com.android.dreams.basic		2013-06-04	15:21	drwxr-x--x	
▶ com.android.emulator.connectivity.test		2013-06-04	15:21	drwxr-x--x	
▶ com.android.emulator.gps.test		2013-06-04	15:21	drwxr-x--x	
▶ com.android.example.fileiodemo		2013-11-11	15:19	drwxr-x--x	
FileIOTest_102067879.txt	5	2013-11-11	15:30	-rw-----	
▶ cache		2013-11-11	15:18	drwxrwx---	
lib		2013-11-11	15:18	lrwxrwxr...	-> /data/app-lib/com.android.example.fileiodemo-1
▶ com.android.example.pc		2013-11-11	15:18	drwxr-x--x	
▶ com.android.example.testwidget		2013-11-11	15:18	drwxr-x--x	

- fileDemo.txt从文件权限上进行分析，“-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5个字节；
-



## 外部存储

- 当通过Context的openFileInput和OpenFileOutput打开文件输入和输出流时，程序所打开的都是应用程序的data目录中的文件，大小有限。
  - 为了扩充大小，可以在SD卡上对文件进行读写；
-



## 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
-



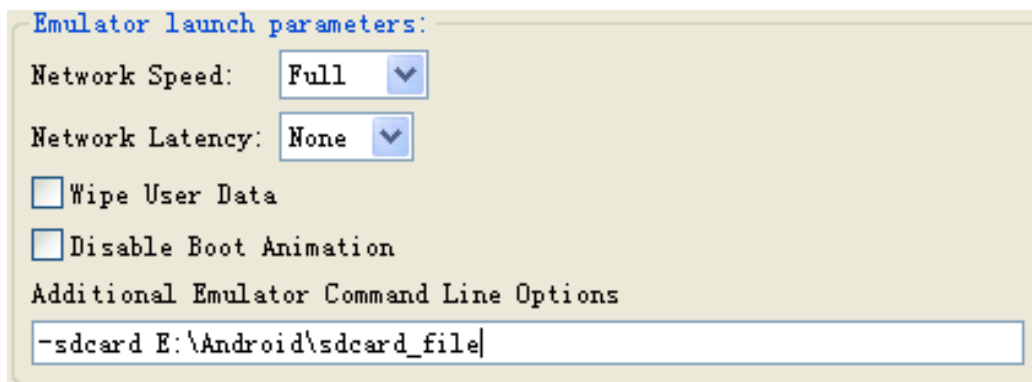
## 外部存储

- 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
-



### 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







### 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

Name	Size	Date	Time	Permiss...
+ data		2009-06-20	00:53	drwxrwx--x
- sdcard		2009-07-16	14:16	d---rwxrwx
test.txt	174	2009-07-16	14:16	----rw-rw-
+ system		2009-04-22	04:11	drwxr-xr-x



## 外部存储

### ➤ 编程访问SD卡

1. 需要检测SD卡是否可用，若不可用，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

- `Environment.getExternalStorageState().equals(Environment.MEDIA_MOUNTED)`
- 或判断/mnt/sdcard/路径是否存在，等价于是否插卡；

2. 若可用，则直接通过使用标准的Java.io.File类进行访问

- `Environment.getExternalStorageDirectory();` // 获取SD卡的目录

### ➤ 将数据保存在SD卡

SDcardFileDemo示例说明了如何将数据保存在SD卡

---



### 外部存储

- 配置AndroidManifest.xml文件，添加对SD卡的读写权限

<!-- 在SD卡中创建与删除文件权限 -->

<uses-permission android:name="android.permission.MOUNT\_UNMOUNT\_FILESYSTEMS"/>

<!-- 向SD卡写入数据权限 -->

<uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE"/>

---



### 外部存储

➤ 下图是SDcardFileDemo示例的用户界面

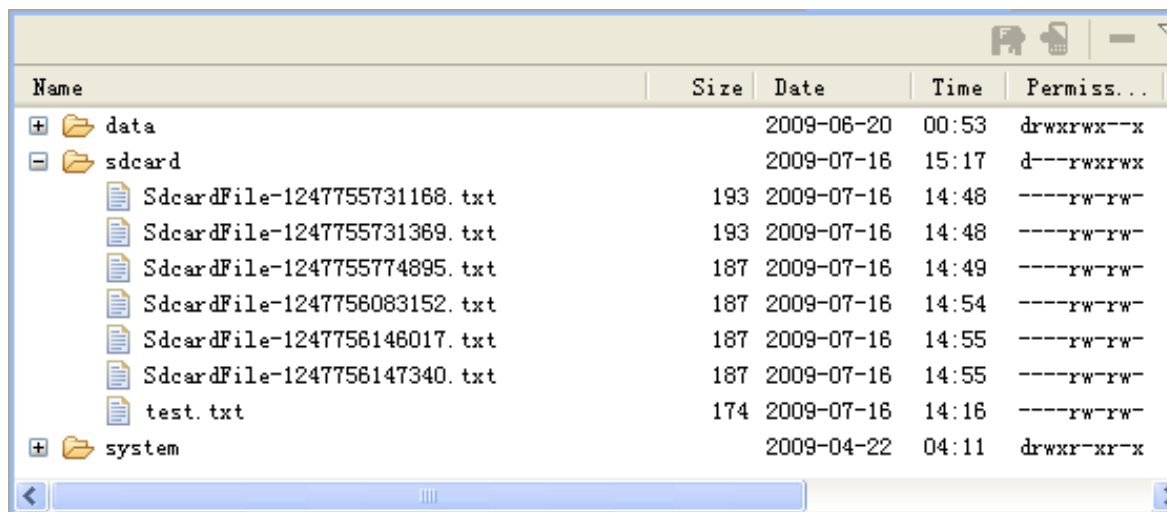
1. 通过“生产随机数列”按钮生产10个随机小数
2. 通过“写入SD卡”按钮将生产的数据保存在SD卡的目录下





### 外部存储

- SDcardFileDemo示例运行后，在每次点击“写入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



## 外部存储

### ■ 是SDcardFileDemo示例的核心代码

```
1. private static String randomNumbersString = "";
2. OnClickListener writeButtonListener = new OnClickListener() {
3.     @Override
4.     public void onClick(View v) {
5.         String fileName = "SdcardFile-"+System.currentTimeMillis()+".txt";
6.         //通过当前时间来为文件名命名，避免重复命名
7.         File dir = new File("/sdcard/");
8.         if (dir.exists() && dir.canWrite()) {//sdcard目录存在性检查
9.             File newFile = new File(dir.getAbsolutePath() + "/" + fileName);
10.            //使用“绝对目录+文件名”的形式表示新建立的文件
11.            FileOutputStream fos = null;
12.            try {
13.                newFile.createNewFile();
14.                if (newFile.exists() && newFile.canWrite()) {
15.                    //文件存在性和可写入性进行检查
16.                    fos = new FileOutputStream(newFile);
17.                    fos.write(randomNumbersString.getBytes());
```



### 外部存储

```
15.                TextView labelView = (TextView)findViewById(R.id.label);
16.                labelView.setText(fileName + "文件写入SD卡");
17.            }
18.        } catch (IOException e) {
19.            e.printStackTrace();
20.        } finally {
21.            if (fos != null) {
22.                try{
23.                    fos.flush();
24.                    fos.close();
25.                }
26.                catch (IOException e) { }
27.            }
28.        }
29.    }
30. }
31. };
```



## 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
-





## 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



## 资源文件

### ➤ 如何读取原始格式文件

1. 首先调用`getResource()`函数获得资源对象
2. 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
3. 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
```



## 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
  1. 在/res/xml目录下创建一个名为toys.xml的文件
  2. 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>
```



## 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  
    if ((toys != null) && toys.equals("toy")) { //查看元素名是否匹配  
        ..... //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  
    }  
}
```



## 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;  
}
```



## 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

---

# Questions?



ANDROID