Adnroid的数据存储和访问

Android的数据存储有五种：文件、 SharedPreferences（参数）、SQLite数据库、内容提供者（Content provider）、网络。

## 一：文件：

使用File进行存储操作主要使用到以下的

**①:public abstract FileInputStream openFileInput (String name)**

这个主要是打开文件，返回FileInputStream

**②:public abstract FileOutputStream openFileOutput (String name, int mode)**

这个主要是写入文件，如果该文件不存在，直接进行创建，返回FileOutputStream

Mode(主要有以下的四种的模式)

Context.MODE\_PRIVATE：新内容覆盖原内容

Context.MODE\_APPEND：新内容追加到原内容后

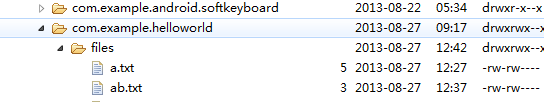
Context.MODE\_WORLD\_READABLE：允许其他应用程序读取

Context.MODE\_WORLD\_WRITEABLE：允许其他应用程序写入，会覆盖原数据。

**③:FileInputStream(获取文件输入流)与FileOutputStream （获取文件输出流）这两类在JavaIO 操作中很常见**

接下来进行操作 保存成功之后讲文件保存在当前应该程序的包名下的files/（可以改变存储的其他路径）

在data –> data 🡪com.android.|文件. -- > files 中



程序：



简单的layout设计就不多介绍了，只需知道些代码就够了

**public** **class** MainActivity **extends** Activity {

**private** EditText fileName;

**private** EditText fileContent;

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

**private** FileService fileService;

@Override

**protected** **void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(R.layout.*activity\_main*);

fileService = **new** FileService(getApplicationContext());

Button button = (Button)**this**.findViewById(R.id.*button*);

fileName = (EditText)**this**.findViewById(R.id.*editText1*);

fileContent = (EditText)**this**.findViewById(R.id.*editText2*);

button.setOnClickListener(**new** View.OnClickListener() {

**public** **void** onClick(View v) {

String name = fileName.getText().toString();

String content = fileContent.getText().toString();

**try** {

fileService.save(name,content);

Log.*i*(*TAG*, "success");

Toast.*makeText*(getApplicationContext(), R.string.*succue*, 1).show();

} **catch** (Exception e) {

e.printStackTrace();

Toast.*makeText*(getApplicationContext(), R.string.*fale*, 1).show();

Log.*i*(*TAG*, "-----fals-------");

} }

});

}

FileSerice程序

**public** **class** FileService {

**private** Context context;

**public** FileService(Context context) {

**this**.context = context;

}

**public** **void** save(String name, String content) **throws** Exception{

// **TODO** Auto-generated method stub

FileOutputStream fos = context.openFileOutput(name, Context.*MODE\_PRIVATE*);

fos.write(content.getBytes());

String str = look(name);

fos.close();

}

**public** String look(String name) **throws** Exception{

FileInputStream fis = context.openFileInput(name);

ByteArrayOutputStream baos = **new** ByteArrayOutputStream();

**byte**[] buffer = **new** **byte**[1024];

**int** len = 0;

**while**((len = fis.read(buffer)) > 0){

baos.write(buffer, 0, len);

}

**return** baos.toByteArray().toString();

}

从此输入流中将最多 len 个字节的数据读入一个 byte 数组中。如果 len 不为 0，则在输入可用之前，该方法将阻塞；否则，不读取任何字节并返回 0。

## 二：把文件放入SDcard中

**在程序中访问SDCard，你需要申请访问SDCard的权限。**

在AndroidManifest.xml中加入访问SDCard的权限如下:

<!-- 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"/>

主要程序：

**if** (Environment.*getExternalStorageState*().equals(

Environment.*MEDIA\_MOUNTED*)) {//判断SD是否存在和是否成功挂载

File fileAdd = Environment.*getExternalStorageDirectory*();

//由于每个版本的sdcard存放的位置不同，所以最好用上面的，读取文件位置

// File fileAdd1 = new File("/mnt/sdcard");

File saveSD = **new** File(fileAdd, "wl.txt");

FileOutputStream fos = **new** FileOutputStream(saveSD);

fos.write("Hello".getBytes());

Log.*i*("TAG", "succuss");

fos.close();

} **else** {

Log.*i*("TAG", "sd is not found！");

}

## 三：pull解析XML文件

除了可以使用 SAX或DOM解析XML文件之外，大家也可以使用Android内置的Pull解析器解析XML文件。 Pull解析器是一个开源的java项目，既可以用于android，也可以用于JavaEE。如果用在javaEE需要把其jar文件放入类路径中，因为Android已经集成进了Pull解析器，所以无需添加任何jar文件。android系统本身使用到的各种xml文件，其内部也是采用Pull解析器进行解析的。 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。跟SAX不同的是， Pull解析器产生的事件是一个数字，而非方法，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类型节点的值。

1. 解析的person.xml

<?xml version=*"1.0"* encoding=*"utf-8"*?><!-- xml开始标志 start document -->

<persons><!-- 节点startTag -->

<!-- node Type?Element node/Text node -->

<person id=*"1"*>

<name>Tom</name><!-- Tom是Text -->

<age>18</age>

</person>

<person id=*"2"*>

<name>Sam</name>

<age>20</age>

</person>

</persons>

<!-- hidden 结束标志符 end document -->

1. 建person类，应用JavaBean
2. 建解析类ParserXml类

**public** **class** ParseXMLService {

**public** List<Person> getXML(InputStream xml) **throws** Exception {

List<Person> persons = **null**;

Person person = **null**;

// 1.找到xml文件,由于用pull解析xml，而其中android有pull解析器

XmlPullParser xmlPullParser = Xml.*newPullParser*();

// XmlPullParser xmlPullParser1 =

// XmlPullParserFactory.newInstance().newPullParser();

// 2:通过输出流来得到文件的地址

xmlPullParser.setInput(xml, "utf-8");

// 3:通过getEventType来得到xml文件中状态

**int** event = xmlPullParser.getEventType();

**while** (event != XmlPullParser.*END\_DOCUMENT*) {

**switch** (event) {

// 4:读取版本信息的头文件，查看文件属性

**case** XmlPullParser.*START\_DOCUMENT*:

persons = **new** ArrayList<Person>();

**break**;

// 5：读取Start\_Tag节点

**case** XmlPullParser.*START\_TAG*:

**if**("person".equals(xmlPullParser.getName())){

person = **new** Person();

// 6：读取第一个person的属性id

**int** id = Integer.*parseInt*(xmlPullParser.getAttributeValue(0));

person.setId(id);

}

**if**("name".equals(xmlPullParser.getName())){

// 7：读取下一个节点Text的信息

String name = xmlPullParser.nextText();

person.setName(name);

}**else** **if**("age".equals(xmlPullParser.getName())){

**int** age = Integer.*parseInt*(xmlPullParser.nextText());

person.setAge(age);

}

**break**;

**case** XmlPullParser.*END\_TAG*:

persons.add(person);

**break**;

}

// 8：可以进入下一个节点，并触发相应事件

event = xmlPullParser.next();

}

xml.close();

**return** persons;

}

**public** **void** saveParseXml(List<Person> persons, OutputStream os) **throws** IllegalArgumentException, IllegalStateException, IOException{

// 1：序列化读取xml

XmlSerializer serializer = Xml.*newSerializer*();

// XmlPullParserFactory.newInstance().newSerializer();

// 2：输出流读出文件

serializer.setOutput(os, "utf-8");

// 3:按面向对象的方式来写入节点

serializer.startDocument("utf-8", **true**);

serializer.startTag(**null**, "persons");

**for**(Person person: persons){

serializer.startTag(**null**, "person");

serializer.attribute(**null**, "id", person.getId().toString());

serializer.startTag(**null**, "name");

serializer.text(person.getName());

serializer.endTag(**null**, "name");

serializer.startTag(**null**, "age");

serializer.text(person.getAge()+"");

serializer.endTag(**null**,"age");

serializer.endTag(**null**, "person");

}

serializer.endTag(**null**, "persons");

serializer.endDocument();

os.flush();

os.close();

}

4. 测试类Test

**public** **void** getReaderXml() **throws** Exception{

InputStream is = **this**.getClass().getClassLoader().getResourceAsStream("person.xml");

ParseXMLService pxs = **new** ParseXMLService();

List<Person> persons= pxs.getXML(is);

**for**(Person person : persons){

Log.*i*("TAG", person.toString());

}

}

**public** **void** saveXmlFile() **throws** Exception{

File xmlFile = **new** File(getContext().getFilesDir(), "test.xml");

//data/data/包名/fiter/test.xml

OutputStream os = **new** FileOutputStream(xmlFile);

ParseXMLService pxs = **new** ParseXMLService();

List<Person> persons= **new** ArrayList<Person>();

Person person1 = **new** Person();

person1.setId(2);

person1.setName("sum");

person1.setAge(19);

Person person2 = **new** Person();

person2.setId(4);

person2.setName("thik");

person2.setAge(20);

persons.add(person1);

persons.add(person2);

pxs.saveParseXml(persons, os);

os.flush();

os.close();

}

## 四：SharedPreferences共享参数类

**一、作用**

主要用于存放软件的配置参数等信息。sharedPreferences用于存取和修改软件配置参数数据的接口，由getSharedPreferences(String, int)函数返回。任何具体的参数，都有一个单独的该类实例向所有客户端共享。修改参数必须通过SharedPreferences.Editor 对象，以确保这些参数在被提交到外存的时候它们的值处于一致的状态和控制之下。该类暂不支持多进程操作，但是以后将提供该功能。

### Activity中

**protected** **void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(R.layout.*activity\_main*);

editText1 = (EditText)**this**.findViewById(R.id.*editText1*);

editText2 = (EditText)**this**.findViewById(R.id.*editText2*);

ps = **new** PreferencesService(**this**.getApplicationContext());

Map<String, String> map = ps.getMyPreferences();

editText1.setText(map.get("name"));

editText2.setText(map.get("age"));

}

@SuppressLint("ShowToast")

**public** **void** save(View v){

Log.*i*(*TAG*, "---run----");

ps.savePreference(editText1.getText().toString(),editText2.getText().toString());

Toast.*makeText*(getApplicationContext(), R.string.*succue*, 1).show();

}

### ****三、存放参数实例源码：****

**public** **void** savePreference(String name, String age) {

SharedPreferences sharedPreferences = context.getSharedPreferences("my\_perferences", context.*MODE\_PRIVATE*);

Editor edit = sharedPreferences.edit();

edit.putString("name", name);

edit.putInt("age", **new** Integer(age));

edit.commit();

}

### 四、读取参数实例源码**：**

**public** Map<String,String> getMyPreferences(){

SharedPreferences sp = context.getSharedPreferences("my\_perferences", context.*MODE\_PRIVATE*);

Map<String,String> perence =**new** HashMap<String, String>();

perence.put("name", sp.getString("name", ""));

perence.put("age",String.*valueOf*(sp.getInt("age", 0)));

**return** perence;

}

**五、归纳**

通过以上类的介绍和实例源码分析，可以总结出一般步骤：

存放：

1.获得SharedPreferences 的实例对象，通过getSharedPreferences（）传递文件名和模式；

2.获得Editor 的实例对象，通过SharedPreferences 的实例对象的edit（）方法；

3.存入数据，利用Editor 对象的putXXX（）方法；

4.提交修改的数据，利用Editor 对象的commit（）方法。

读取：

1.获得SharedPreferences 的实例对象，通过getSharedPreferences（）传递文件名和模式；

2.读取数据，通过SharedPreferences 的实例对象的getXXX（）方法。

如果是window软件通常我们会采用ini文件进行保存，如果是j2se应用，我们会采用properties属性文件或者xml进行保存。如果是Android应用，我们最适合采用什么方式保存软件配置参数呢？Android平台给我们提供了一个SharedPreferences类，它是一个轻量级的存储类，特别适合用于保存软件配置参数。使用SharedPreferences保存数据，其背后是用xml文件存放数据，文件存放在**/data/data/<package name>/shared\_prefs目录**