# 基础代码知识

## 1. Toast吐司

Toast.*makeText*(getApplicationContext(), "来了", 1).show();

## 2. XmlPullParser xml解析用到

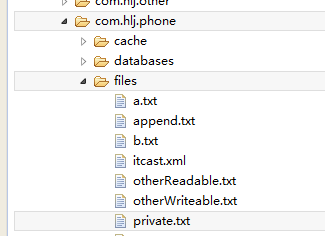
XmlSerializer xml导入用到

# 1.file文件存储

## 1.文件操作模式，私有

FileOutputStream outStream = context.openFileOutput(filename, Context.*MODE\_PRIVATE*);

outStream.write(content.getBytes());



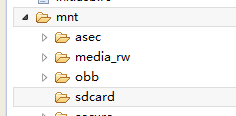
### **私有模式** ①只能被创建这个文件的当前应用访问 ②若文件不存在会创建文件；若创建的文件已存在则会覆盖掉原来的文件 Context.MODE\_PRIVATE = 0;

### **追加模式** ①私有的 ②若文件不存在会创建文件；若文件存在则在文件的末尾进行追加内容 Context.MODE\_APPEND = 32768;

### **可读模式** ①创建出来的文件可以被其他应用所读取 Context.MODE\_WORLD\_READABLE=1;

### **可写模式** ①允许其他应用对其进行写入。 Context.MODE\_WORLD\_WRITEABLE=2

## 2.取得sd卡文件路径



File filePathStorage = **new** File("/mnt/sdcard",filename);

File file = **new** File(Environment.*getExternalStorageDirectory*(), filename);

outStream.write(content.getBytes());

## 3.判断sd卡是不是存在

**if**(Environment.*getExternalStorageState*().equals(Environment.*MEDIA\_MOUNTED*)){

## 3.文件存储和读取代码

文件存储：

File file = **new** File(Environment.*getExternalStorageDirectory*(), filename);

outStream.write(content.getBytes());

文件读取

FileInputStream inStream =**new** FileInputStream(filePathStorage);

ByteArrayOutputStream outStream = **new** ByteArrayOutputStream();

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

**int** len = 0;

**while**((len = inStream.read(buffer)) != -1){

outStream.write(buffer, 0, len);

}

**byte**[] data = outStream.toByteArray();

**return** **new** String(data);

# 2.xml解析

## 1.src下面解析

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

## 2.file文件目录下解析

InputStream = context.openFileInput(xmlName);

// <包>/files 相当于是/data/data/com.hlj.phone/files/xmlAnalyze.xml

## 3.xml解析代码 XmlPullParser

**public** **static** List<XmlPerson> getPersons(InputStream xml) **throws** Exception{

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

XmlPerson person = **null**;

XmlPullParser pullParser = Xml.*newPullParser*();

pullParser.setInput(xml, "UTF-8");//为Pull解析器设置要解析的XML数据

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

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

**switch** (event) {

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

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

**break**;

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

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

**int** id = **new** Integer(pullParser.getAttributeValue(0));

person = **new** XmlPerson();

person.setId(id);

}

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

String name = pullParser.nextText();

person.setName(name);

}

**if**("age".equals(pullParser.getName())){

**int** age = **new** Integer(pullParser.nextText());

person.setAge(age);

}

**break**;

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

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

persons.add(person);

person = **null**;

}

**break**;

}

event = pullParser.next();

}

**return** persons;

}

## 4.导入xml文件内容

// <包>/files 相当于是/data/data/com.hlj.phone/files/xmlAnalyze.xml

File xmlFile = **new** File(context.getFilesDir(), "xmlAnalyze.xml");

**public** **static** **void** save(List<XmlPerson> persons, OutputStream out) **throws** Exception{

//xml 可被序列化的

XmlSerializer serializer = Xml.*newSerializer*();

serializer.setOutput(out, "UTF-8");

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

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

**for**(XmlPerson 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().toString());

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

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

}

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

serializer.endDocument();

out.flush();

out.close();

}

# 3. SharedPreferences

## 1.保存数据

**public** **void** save(String name, Integer valueOf) {

//这里就我们使用的参数进行保存吗，

//下面这个就是创建文件 name是不用指定扩展名的，系统会直接xml是他的扩展名

//一般用户这个是不需要被其他应用访问的，所以直接设置为MODE.PRIVATE

SharedPreferences preferences =context.getSharedPreferences("shareprefernces",Context.*MODE\_PRIVATE*);

//现在我们要开始存放数据了，先得到编辑器的对象

Editor editor = preferences.edit();

//第一个是参数所定义的名称，只要是字符串随便选即可，随便选择，

//第二个是我么的参数

editor.putString("name",name);

editor.putInt("age",valueOf);

//现在我们的参数是存放在内存中，还没有放到文件中,使用方法将它放到文件中,

//将数据交回文件中

editor.commit();

}

## 2.取得数据

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

/\*\*

\* 获取各项配置参数

\*/

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

SharedPreferences preferences =context.getSharedPreferences("shareprefernces",Context.*MODE\_PRIVATE*);

//第一个我们的参数，第二个是如果没有这个参数的时候，默认返回的是blank或者自己写入

canshu.put("name",preferences.getString("name", "dage"));

canshu.put("age",String.*valueOf*(preferences.getInt("age",0)));

**return** canshu;

}

# 4.sqlite数据库

## 1.第一步，建立一个数据库的配置文件

**public** **class** SqliteDatabeseService **extends** SQLiteOpenHelper {

//调用父类的构造器

//第二个指定我们的数据库的名称 ，如果非要给这个数据库加一个后缀，.db

//第三个是游标工程，就是用来产生游标对象，我们这里传入一个null

//第四个是我们的数据库的 版本号，我们建议是 一

**public** SqliteDatabeseService(Context context) {

**super**(context,"sqlite.db",**null**,1);

// **TODO** Auto-generated constructor stub

}

**public** **void** onCreate(SQLiteDatabase db) {//是在数据库每一次被创建的时候调用的

db.execSQL("CREATE TABLE person(personid integer primary key autoincrement, name varchar(20), phone VARCHAR(12),amount integer)");

}

////这个方法是在数据库的版本号变化的时候调用的,假如上面的版本变成2时候，SQL语句会额外的加入一条语句(如下面)

**public** **void** onUpgrade(SQLiteDatabase db, **int** oldVersion, **int** newVersion) {

//更改表格

// db.execSQL("ALTER TABLE person ADD amount integer");

}

}

## 2.增

**public** **void** saveDb(SqlitePerson person){

//这样就可以取得数据库的操作实例了

//这样构造完成之后我们的另一个参数id是自动的调节的，第一个次是一

SQLiteDatabase db = servise.getWritableDatabase();

////通过问号进行赋值，

ContentValues values = **new** ContentValues();

values.put("name", person.getName());

values.put("phone", person.getPhone());

values.put("amount", person.getAmount());

db.insert("person",**null**,values);

}

**public** **void** saveexecSQL(SqlitePerson person){

SQLiteDatabase db = servise.getWritableDatabase();

db.execSQL("insert into person(name,phone,amount) values(?,?,?)",

**new** Object[]{person.getName(),person.getPhone(), person.getAmount()});

db.close();

}

## 2。删

**public** **void** deleteexecSQL(Integer id){

SQLiteDatabase db = servise.getWritableDatabase();

db.execSQL("delete from person where personid=?",**new** Object[]{id} );

db.close();

}

## 3.改

**public** **void** updateexecSQL(SqlitePerson person){

SQLiteDatabase db = servise.getWritableDatabase();

db.execSQL("update person set name=?,phone=?,amount=? where personid =? ",

**new** Object[]{person.getName(),person.getPhone(),person.getAmount(),person.getPersonid()});

db.close();

## 4.查

**public** SqlitePerson findrawQuery(Integer id){

SQLiteDatabase db = servise.getReadableDatabase();

Cursor cursor = db.rawQuery("select \* from person where personid=?", **new** String[]{id.toString()});

**if**(cursor.moveToFirst()){

Integer personid = cursor.getInt(cursor.getColumnIndex("personid"));

**int** amount = cursor.getInt(cursor.getColumnIndex("amount"));

String name = cursor.getString(cursor.getColumnIndex("name"));

String phone = cursor.getString(cursor.getColumnIndex("phone"));

**return** **new** SqlitePerson(name, phone,personid, amount);

}

cursor.close();

**return** **null**;

}

# 5 ContentProvider

## 1.清单文件中进行配置

<!-- 第二个是我们的内容提供这的标志，唯一标示，就像我们访问一个东西必须知道它的标识-表明我们的身份-写入即可 -->

<!-- 防止其他应用访问不到 android:exported="true--> <provider android:name=*"com.hlj.datastoragecontentprovider.PersonProvider"*

android:authorities=*"com.hlj.providers.personprovider"*

android:exported=*"true"*> </provider>

## 2. 内容提供者

**private** SqliteDatabeseService dbOpenHelper;

// 首先需要对Uri匹配判断MATCHER常量(翻译匹配机)，不匹配的时候-1返回，如果正确，就返回常量

**private** **static** **final** UriMatcher *MATCHER* = **new** UriMatcher(UriMatcher.*NO\_MATCH*);

**private** **static** **final** **int** *PERSONS* = 1;

//删除的时候用到PERSON

**private** **static** **final** **int** *PERSON* = 2;

**static**{

*MATCHER*.addURI("com.hlj.providers.personprovider", "person", *PERSONS*);

*MATCHER*.addURI("com.hlj.providers.personprovider", "person/#", *PERSON*);

}

/\*\*

\* onCreate

\* 只会被调用一次，我们可以进行初始化操作

\*/

@Override

**public** **boolean** onCreate() {

dbOpenHelper = **new** SqliteDatabeseService(**this**.getContext());

**return** **true**;

}

@Override

**public** Uri insert(Uri uri, ContentValues values) {

SQLiteDatabase db = dbOpenHelper.getWritableDatabase();

**switch** (*MATCHER*.match(uri)) {

**case** *PERSONS*:

// 下面的name 是其中过得一个字段的名称当values是空值的时候可以用的上 ,可以参考之前的sqlite保存可以看到

**long** rowid = db.insert("person", "name", values);//主键值

// content://com.hlj.provides.personprovider/person/10

//返回添加的记录所代表的uri 下面的是系统提供的方法

//Uri insertUri = Uri.parse("com.hlj.providers.personprovider/person/"+rowid);

Uri insertUri = ContentUris.*withAppendedId*(uri, rowid);

**this**.getContext().getContentResolver().notifyChange(uri, **null**);//发出数据变化通知

**return** insertUri;

**default**:

**throw** **new** IllegalArgumentException("this is Unknown Uri:"+ uri);

}

}

## 3.其他应用进行测试

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

Uri uri = Uri.*parse*("content://com.hlj.providers.personprovider/person");

ContentResolver resolver = **this**.getContext().getContentResolver();

ContentValues values = **new** ContentValues();

values.put("name", "laoli");

values.put("phone", "186010383");

values.put("amount", "500");

resolver.insert(uri, values);

}

# 6.listview

解释：根据列表的适配器类型，列表分为三种，ArrayAdapter，SimpleAdapter和SimpleCursorAdapter，其中以ArrayAdapter最为简单，只能展示一行字。SimpleAdapter有最好的扩充性，可以自定义出各种效果。 SimpleCursorAdapter可以认为是SimpleAdapter对数据库的简单结合，可以方面的把数据库的内容以列表的形式展示出来。

## 1. ArrayAdapter

## 2. **SimpleCursorAdapter，SimpleAdapter对数据库的简单结合**

### 3. simpleAdapter的扩展性最好，可以定义各种各样的布局出来

## 4.自定义的适配器

# 7.网络

# 8.activity 传递参数

Intent intent = **new** Intent(**this**, ReceiveParameterTransmitActivity.**class**);//激活组件,显示意图:明确指定了组件名称的意图叫显示意图

Bundle bundle = **new** Bundle();

**int** parametertwo = Integer.*parseInt*(parametertwoEditText.getText().toString());

bundle.putString("parameterone",parameteroneEditText.getText().toString());

bundle.putInt("parametertwo",parametertwo );

intent.putExtras(bundle);

startActivity(intent);

## 2.接受参数

Intent intent = getIntent();//获取用于激活它的意图对象

Bundle bundle = intent.getExtras();

String parameterone = bundle.getString("parameterone");

**int** parametertwo = bundle.getInt("parametertwo");

parameteronetext.setText(parameterone);

parametertwotext.setText(parametertwo+"");

# 9.activity启动模式

## 1.模式的配置，清单文件中

<activity android:name=*"com.hlj.activitylaunchmode.ActivityLaouchModeActivity"* > </activity>

<!-- singleTop -->

<activity android:name=*"com.hlj.activitylaunchmode.SingleTopActivity"*

android:launchMode=*"singleTop"* ></activity>

<!-- singleTask -->

<activity android:name=*"com.hlj.activitylaunchmode.SingleTaskActivity"*

android:launchMode=*"singleTask"* ></activity>

## 2. standard

每次启用一个activity都会重新创建一个新实例页面，

\* 测试方法，每次点击这个按钮，上面的内容就会发生变化，点击返回之后回到上个activity，也就是之前的实例

## 3. \* SingleTop

\* 每次启动会创建一个实例页面，如果栈顶有则打开自己本身，如果栈顶没有则重新创建，

## 4. SingleTask

\* 每次启动会创建一个实例，如果在整个栈的过程中产生之后，则不再产生，会打开自己本身，同时在自己之上的实例都会被删除。

## 5. singleinstance

\* 所有应用公用一个实例，不管谁激活都会进入同一个应用中去（）

\* 特别说明：这个没有进行实践

# 10.intent

## 1.配置文件中进行设置

<intent-filter >

<!-- 必须加下面这个 category android:name="android.intent.category.DEFAULT-->

<category android:name=*"android.intent.category.DEFAULT"*/>

<action android:name=*"com.hlj.actionName1"*/>

<action android:name=*"com.hlj.actionName2"*/>

<category android:name=*"com.hlj.categoryName1"*/>

<category android:name=*"com.hlj.catagoryName2"*/>

<data android:scheme=*"hlj"* android:host=*"www.mxzd.com"* android:path=*"/person"*/>

<data android:mimeType=*"image/\*"*/>

</intent-filter>

## 2.跳转

/\*\*

\* (没设数据参数的情况下)只要Intent中的Action和Category都出现在Intent-Filter中，就能与之匹配，否则匹配失败

\*/

Intent intent = **new** Intent();//隐式意图激活Activity

intent.setAction("com.hlj.actionName1");

intent.addCategory("com.hlj.categoryName1");

//只有数据

//intent.setData(Uri.parse("hlj://www.mxzd.com/person"));

//类型匹配 ,不匹配数据，只匹配类型会失败。如果都有数据和类型，

//则使用setDataAndType方法，因为只匹配类型setType的时候会将数据清空

//intent.setType("image/jpeg");

/\*有数据和类型\*/

intent.setDataAndType(Uri.*parse*("hlj://www.mxzd.com/person"), "image/jpeg");

startActivity(intent);//方法内部为Intent添加了android.intent.category.DEFAULT类别

# 11.activity生命周期

1.启动Activity：系统会先调用onCreate方法，然后调用onStart方法，最后调用onResume，Activity进入运行状态。

2.当前Activity被其他Activity覆盖其上或被锁屏：系统会调用onPause方法，暂停当前Activity的执行。

3.当前Activity由被覆盖状态回到前台或解锁屏：系统会调用onResume方法，再次进入运行状态。

4.当前Activity转到新的Activity界面或按Home键回到主屏，自身退居后台：系统会先调用onPause方法，然后调用onStop方法，进入停滞状态。

5.用户后退回到此Activity：系统会先调用onRestart方法，然后调用onStart方法，最后调用onResume方法，再次进入运行状态。

6.当前Activity处于被覆盖状态或者后台不可见状态，即第2步和第4步，系统内存不足，杀死当前Activity，而后用户退回当前Activity：再次调用onCreate方法、onStart方法、onResume方法，进入运行状态。

7.用户退出当前Activity：系统先调用onPause方法，然后调用onStop方法，最后调用onDestory方法，结束当前Activity。

但是知道这些还不够，我们必须亲自试验一下才能深刻体会，融会贯通

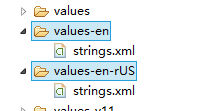
# 11.service服务

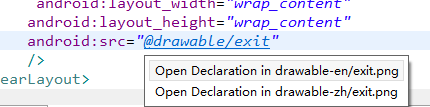
# 12播放器

mediaPlayer = **new** MediaPlayer();

照相机 Camera

# 13.软件国际化





# 14 屏幕适配

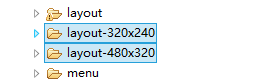
/\*\*

\* 屏幕适配。工作量巨大，

\* 适配是根据安卓设备的分辨率，自动选择 xml 的布局文件的。

\* 所以说工作量巨大

\*/



# 15.样式和主题

<!-- 样式和主题 StyleAndTheme -->

<!-- 主题样式，窗口的全屏显示 android:theme="@style/itcastTheme" -->

<activity android:name=*"com.hlj.styleandtheme.StyleAndThemeActivity"*

android:theme=*"@style/hljTheme"*

> </activity>

<style name=*"hljTheme"*>

<!-- 没有标题 -->

<item name=*"android:windowNoTitle"*>true</item>

<!-- 窗口全屏显示 下面的问号为引用上面的 android:windowNoTitle 值 ，一般情况下二者都是ture，或者都是false-->

<item name=*"android:windowFullscreen"*>?android:windowNoTitle</item>

<item name=*"android:textSize"*>18sp</item>

<item name=*"android:textColor"*>#FFFFFF</item>

</style>

## 16.状态栏的通知

**int** icon = android.R.drawable.*stat\_notify\_chat*;

//第一是图片，第二是概要，第三个是时间

Notification notification = **new** ~~Notification~~(icon, tickerText, System.*currentTimeMillis*());

//点击之后打电话

Intent intent = **new** Intent(Intent.*ACTION\_CALL*, Uri.*parse*("tel:18842637651"));

PendingIntent pendingIntent = PendingIntent.*getActivity*(**this**, 10, intent, 0);

// 拉下来出现的 第一个是我们的标题，第二个使我们的内容，第三个就是我们点击之后出现的

notification.~~setLatestEventInfo~~(**this**, title, content, pendingIntent);

//通知了，会有一个声音提醒,下面这个就是android系统中自带的声音

notification.defaults = Notification.*DEFAULT\_SOUND*;

//这时我们的通知栏出现了但是点击之后还是存在，这时我需要增加一个代码，进行取消了它 设置一个flag自动废除

notification.flags = Notification.*FLAG\_AUTO\_CANCEL*;

//接受通知还需要一个通知管理者，第一个个id是随便定义的

NotificationManager manager = (NotificationManager) getSystemService(Context.*NOTIFICATION\_SERVICE*);

manager.notify(100, notification);

17.