1.JSON

**JSON数据获取**

InputStreamReader isr = new InputStreamReader(getAssets().open(“json文件”),”UTF-8”);

BufferedReader br = new BufferedReader(isr);

StringBuilder builder = new StringBuilder();

While( (line = br.readLine()) != null){

Builder.append(line);

}

br.close();

isr.close();

JSONObject root = new JSONObject(builder.toString());

//获取不同类型的数据

String string = root.getString(“key”);

JSONArray array =root.getJSONArray(“key”);

int value = root.getInt(“key”);

**JSON数据创建**

{ “key”:”value”,

“arraykey”:[

{ “key1”:”value1”, “key2”:”value2”}

{ “key1”:”value1”, “key2”:”value2”}

]

}

JSONObject root = new JSONObject();

root.put(“key”,”value”); //单行数据

//数组

JSONObject line1 = new JSONObject();

line1.put(“key1”,”value1”); line1.put(“key2”,”value2”);

JSONObject line2 = new JSONObject();

line2.put(“key1”,”value1”); line2.put(“key2”,”value2”);

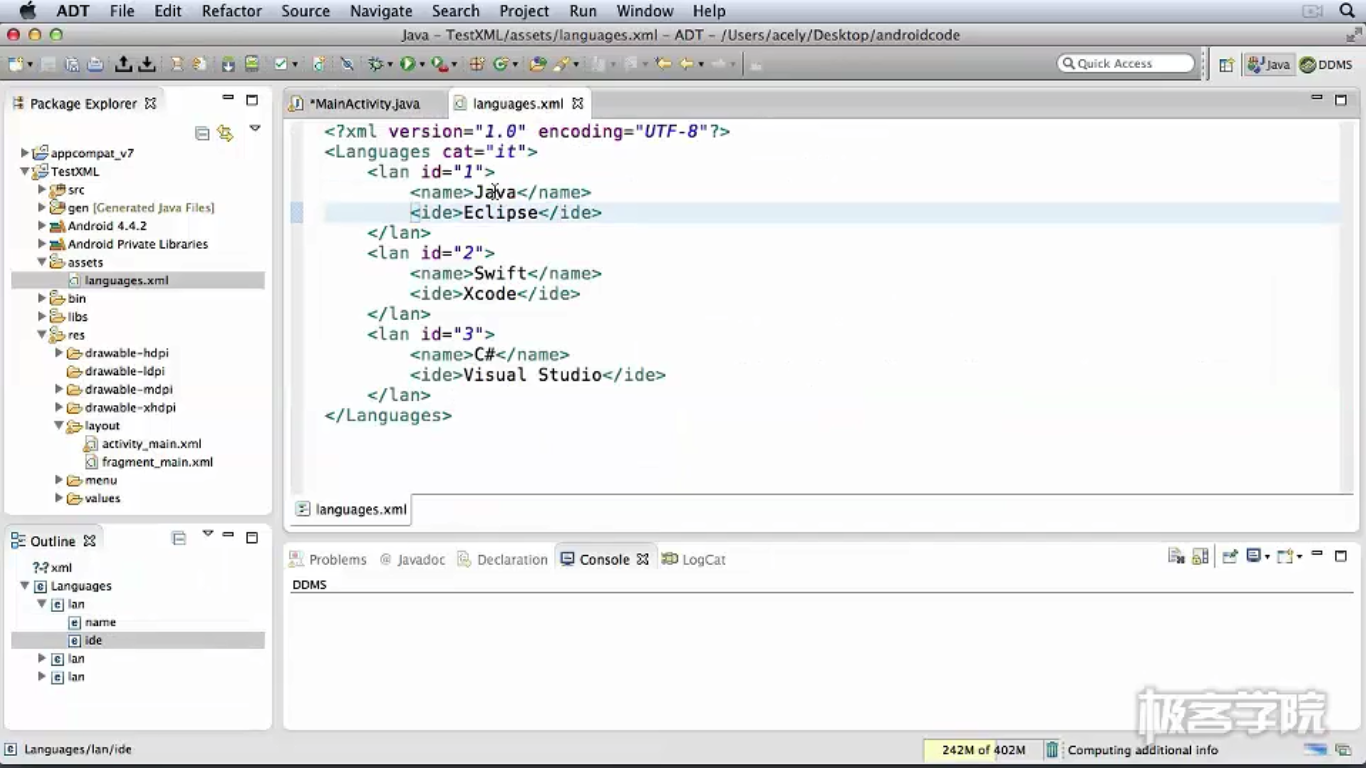
JSONArray array = new JSONArray();

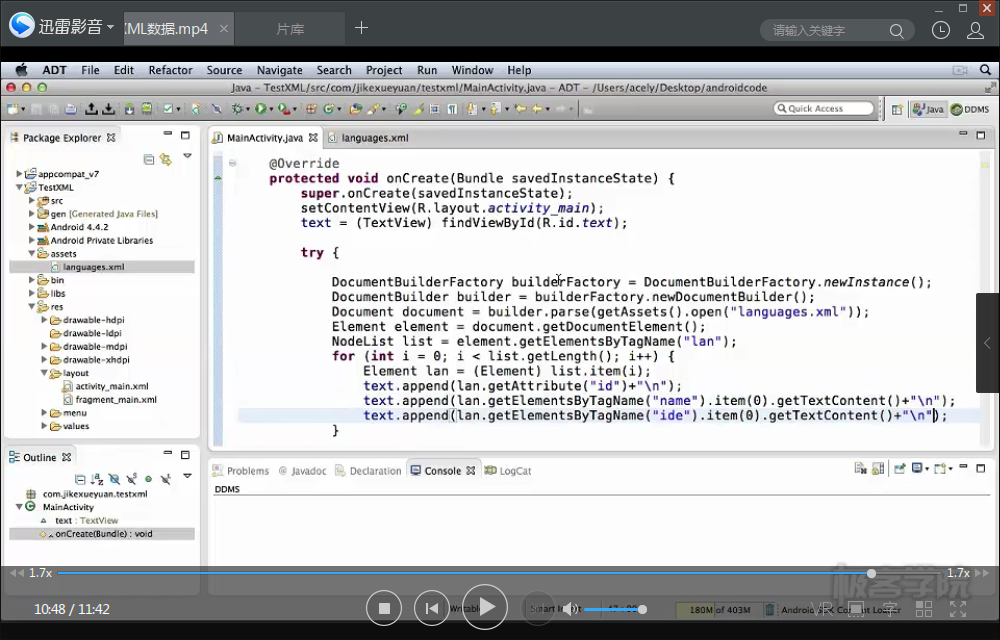
array.put(line1); array.put(line2);

root.put(“arraykey”,array);

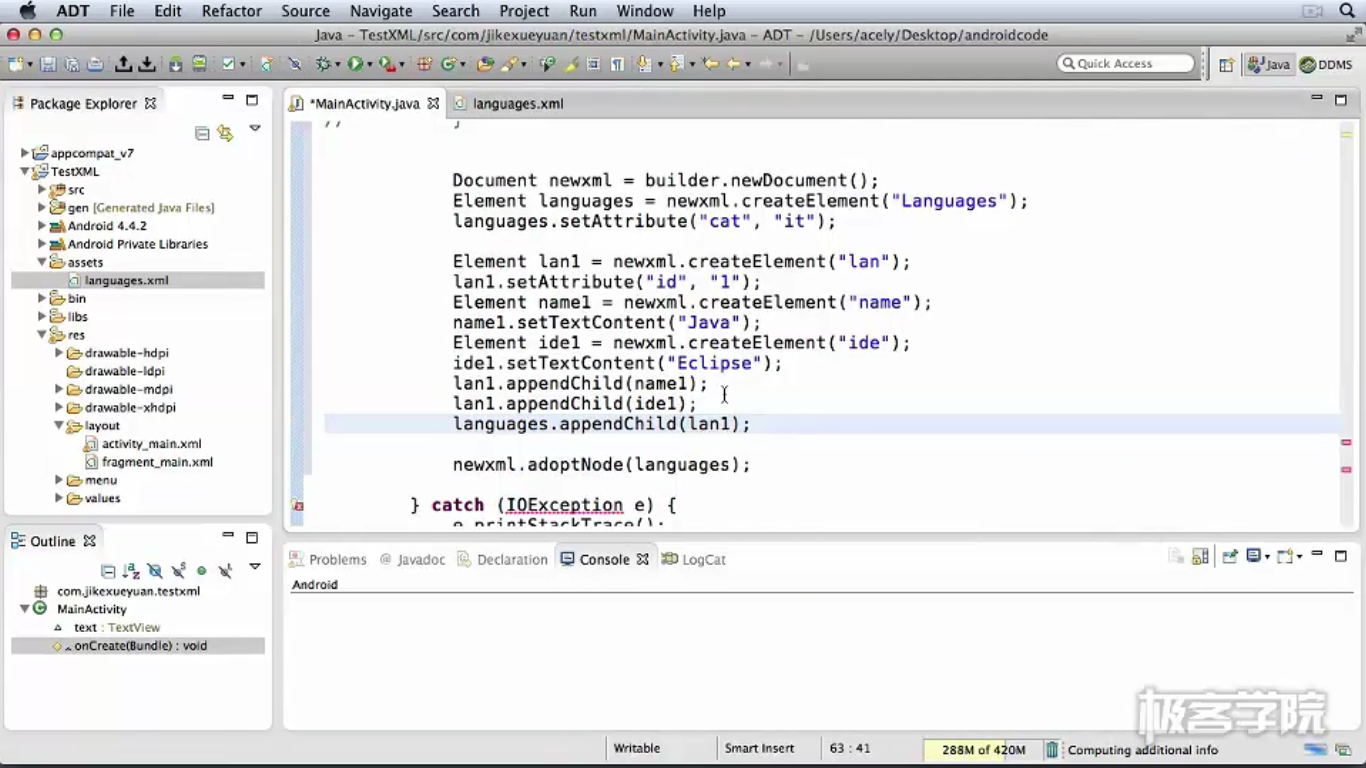
2.XML

**读取与解析**



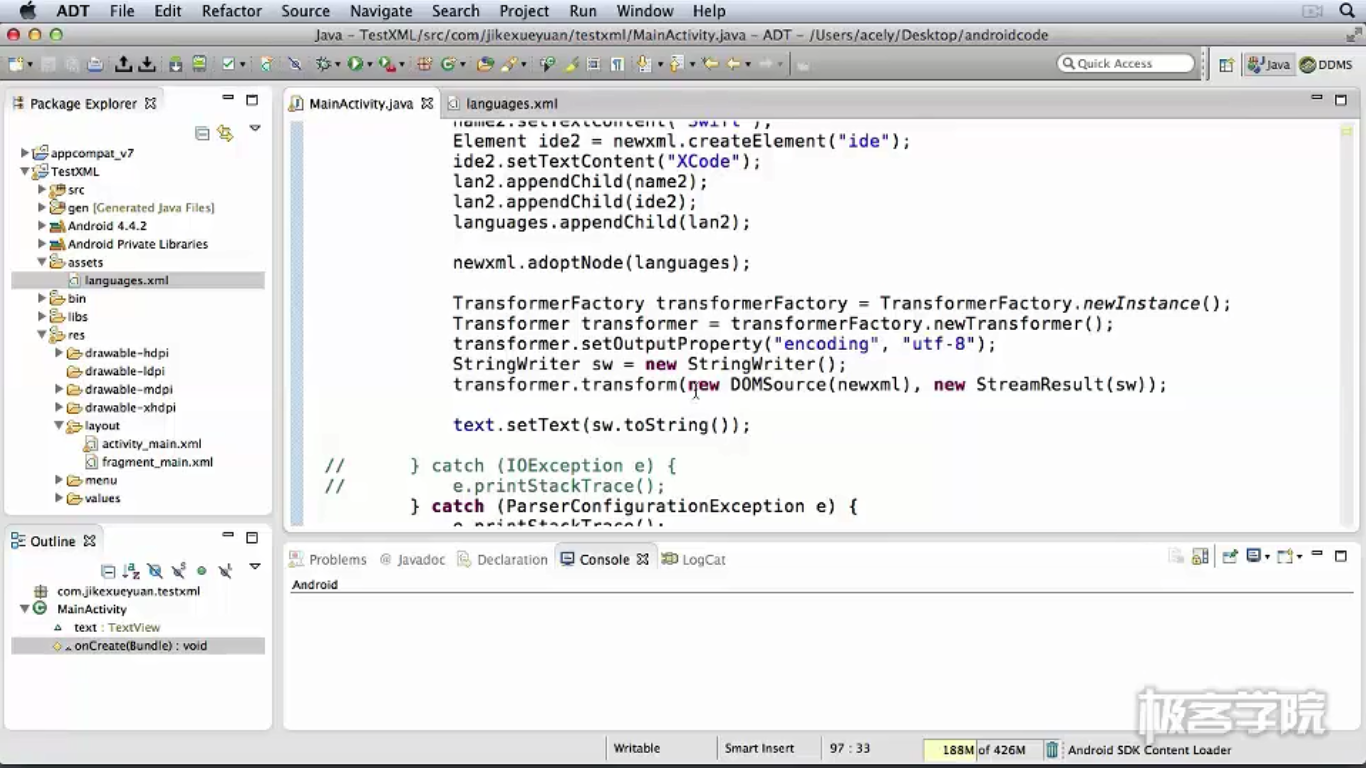


**创建**



Newxml.appendChild(Language);

格式转换输出



3. SQLite

*SQLite* is an Open Source database. SQLite supports standard relational database features like SQL syntax, transactions and prepared statements. The database requires limited memory at runtime (approx. 250 Kbytes) which makes it a good candidate from being embedded into other runtimes.

SQLiteOpenHelper

To create and upgrade a database in your Android application you create a subclass of the SQLiteOpenHelper class. In the constructor of your subclass you call the super() method of SQLiteOpenHelper, specifying the database name and the current database version.

**public** MySQLiteHelper(Context context) {  
 **super**(context, ***DATABASE\_NAME***, **null**, ***DATABASE\_VERSION***);  
}

In this class you need to override the following methods to create and update your database.

* onCreate() - is called by the framework, if the database is accessed but not yet created.
* onUpgrade() - called, if the database version is increased in your application code. This method allows you to update an existing database schema or to drop the existing database and recreate it via the onCreate() method.

The SQLiteOpenHelper class provides the getReadableDatabase() andgetWriteableDatabase() methods to get access to an SQLiteDatabase object; either in read or write mode.

The database tables should use the identifier \_id for the primary key of the table. Several Android functions rely on this standard.

|  |  |
| --- | --- |
|  | It is good practice to create a separate class per table. This class defines staticonCreate() and onUpgrade() methods. These methods are called in the corresponding methods of SQLiteOpenHelper. This way your implementation ofSQLiteOpenHelper stays readable, even if you have several tables. |

SQLiteDatabase

SQLiteDatabase is the base class for working with a SQLite database in Android and provides methods to open, query, update and close the database.

More specifically SQLiteDatabase provides the insert(), update() and delete() methods.

In addition it provides the execSQL() method, which allows to execute an SQL statement directly.

The object ContentValues allows to define key/values. The key represents the table column identifier and the value represents the content for the table record in this column.ContentValues can be used for inserts and updates of database entries.

Queries can be created via the rawQuery() and query() methods or via theSQLiteQueryBuilder class .

**rawQuery()**directly accepts an SQL select statement as input.

Cursor cursor = getReadableDatabase().

rawQuery("select \* from todo where \_id = ?", new String[] { id });

**SQLiteQueryBuilder** is a convenience class that helps to build SQL queries.

**query()**provides a structured interface for specifying the SQL query.

return database.query(DATABASE\_TABLE,

new String[] { KEY\_ROWID, KEY\_CATEGORY, KEY\_SUMMARY, KEY\_DESCRIPTION },

null, null, null, null, null);

|  | |
| --- | --- |
| **Parameter** | **Comment** |
| String dbName | The table name to compile the query against. |
| String[] columnNames | A list of which table columns to return. Passing "null" will return all columns. |
| String whereClause | Where-clause, i.e. filter for the selection of data, null will select all data. |
| String[] selectionArgs | You may include ?s in the "whereClause"". These placeholders will get replaced by the values from the selectionArgs array. |
| String[] groupBy | A filter declaring how to group rows, null will cause the rows to not be grouped. |
| String[] having | Filter for the groups, null means no filter. |
| String[] orderBy | Table columns which will be used to order the data, null means no ordering. |

If a condition is not required you can pass null, e.g. for the group by clause.

The "whereClause" is specified without the word "where", for example a "where" statement might look like: "\_id=19 and summary=?".

If you specify placeholder values in the where clause via ?, you pass them as the selectionArgs parameter to the query.

Cursor

A query returns a Cursor object. A Cursor represents the result of a query and basically points to **one row** of the query result. This way Android can buffer the query results efficiently; as it does not have to load all data into memory.

To get the number of elements of the resulting query use the getCount() method.

To move between individual data rows, you can use the moveToFirst() and moveToNext() methods. The isAfterLast() method allows to check if the end of the query result has been reached.

Cursor provides typed get\*() methods, e.g. getLong(columnIndex),getString(columnIndex) to access the column data for the current position of the result. The "columnIndex" is the number of the column you are accessing.

Cursor also provides the getColumnIndexOrThrow(String) method which allows to get the column index for a column name of the table.

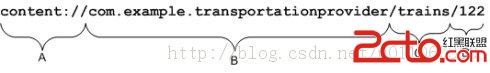
A Cursor needs to be closed with the close() method call.

ContentProvider

**一、相关ContentProvider概念解析：**

1、ContentProvider简介  
在[Android](http://www.2cto.com/kf/yidong/Android/)官方指出的Android的数据存储方式总共有五种，分别是：Shared Preferences、网络存储、文件存储、外储存储、SQLite。但是我们知道一般这些存储都只是在单独的一个应用程序之中达到一个数据的共享，有时候我们需要操作其他应用程序的一些数据，例如我们需要操作[系统](http://www.2cto.com/os/" \t "_blank)里的媒体库、通讯录等，这时我们就可能通过ContentProvider来满足我们的需求了。

2、Uri介绍

为系统的每一个资源给其一个名字，比方说通话记录。  
1）、每一个ContentProvider都拥有一个公共的URI，这个URI用于表示这个ContentProvider所提供的数据。   
2）、Android所提供的ContentProvider都存放在android.provider包中。 将其分为A，B，C，D 4个部分：  
  
  
A：标准前缀，用来说明一个Content Provider控制这些数据，无法改变的；"content://"  
B：URI 的标识，用于唯一标识这个ContentProvider，外部调用者可以根据这个标识来找到它。它定义了是哪个Content Provider提供这些数据。对于第三方应用程序，为了保证URI标识的唯一性，它必须是一个完整的、小写的类名。这个标识在 元素的 authorities属性中说明：一般是定义该ContentProvider的包.类的名称  
C：路径（path），通俗的讲就是你要操作的[数据库](http://www.2cto.com/database/" \t "_blank)中表的名字，或者你也可以自己定义，记得在使用的时候保持一致就可以了；"content://com.bing.provider.myprovider/tablename"  
D：如果URI中包含表示需要获取的记录的ID；则就返回该id对应的数据，如果没有ID，就表示返回全部； "content://com.bing.provider.myprovider/tablename/#" #表示数据id。

**PS：**

路径（path）可以用来表示我们要操作的数据，路径的构建应根据业务而定，如下:  
1、要操作person表中id为10的记录，可以构建这样的路径:/person/10  
2、要操作person表中id为10的记录的name字段， person/10/name  
3、要操作person表中的所有记录，可以构建这样的路径:/person  
4、要操作xxx表中的记录，可以构建这样的路径:/xxx  
5、当然要操作的数据不一定来自数据库，也可以是文件、xml或网络等其他存储方式，如下:  
要操作xml文件中person节点下的name节点，可以构建这样的路径：/person/name  
6、如果要把一个字符串转换成Uri，可以使用Uri类中的parse()方法，如下：Uri uri = Uri.parse("content://com.bing.provider.personprovider/person")

**4、UriMatcher类使用介绍**

因为Uri代表了要操作的数据，所以我们经常需要解析Uri，并从Uri中获取数据。Android系统提供了两个用于操作Uri的工具类，分别为UriMatcher和ContentUris 。  
UriMatcher类用于匹配Uri，它的用法如下：  
首先第一步把你需要匹配Uri路径全部给注册上，如下：

|  |  |
| --- | --- |
|  | //常量UriMatcher.NO\_MATCH表示不匹配任何路径的返回码  UriMatcher  sMatcher = new UriMatcher(UriMatcher.NO\_MATCH);  //如果match()方法匹配<content://com.bing.procvide.personprovider/person>路径，返回匹配码为1  sMatcher.addURI("com.bing.procvide.personprovider", "person", 1);//添加需要匹配uri，如果匹配就会返回匹配码  //如果match()方法匹配<content://com.bing.provider.personprovider/person/230>路径，返回匹配码为2  sMatcher.addURI("com.bing.provider.personprovider", "person/#", 2);//#号为通配符  switch (sMatcher.match(Uri.parse("<content://com.ljq.provider.personprovider/person/10>"))) {     case 1     case 2     default://不匹配       break;  } |

注册完需要匹配的Uri后，就可以使用sMatcher.match(uri)方法对输入的Uri进行匹配，如果匹配就返回匹配码，匹配码是调用addURI()方法传入的第三个参数，假设匹配content://com.ljq.provider.personprovider/person路径，返回的匹配码为1   
**5、ContentUris类使用介绍**

ContentUris类用于操作Uri路径后面的ID部分，它有两个比较实用的方法：  
withAppendedId(uri, id)用于为路径加上ID部分：

|  |  |
| --- | --- |
|  | Uri uri = Uri.parse("<content://com.bing.provider.personprovider/person>")  Uri resultUri = ContentUris.withAppendedId(uri, 10);  //生成后的Uri为：<content://com.bing.provider.personprovider/person/10> |

parseId(uri)方法用于从路径中获取ID部分：

|  |  |
| --- | --- |
|  | Uri uri = Uri.parse("<content://com.ljq.provider.personprovider/person/10>")  long personid = ContentUris.parseId(uri);//获取的结果为:10 |

**6、使用ContentProvider共享数据**

1）ContentProvider类主要方法：  
public boolean onCreate()

public Uri insert(Uri uri, ContentValues values)

public int delete(Uri uri, String selection, String[] selectionArgs)

public int update(Uri uri, ContentValues values, String selection, String[] selectionArgs)

public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)：该方法用于供外部应用从ContentProvider中获取数据。  
public String getType(Uri uri)：该方法用于返回当前Url所代表数据的MIME类型。  
  
2）如果操作的数据属于集合类型，那么MIME类型字符串应该以vnd.android.cursor.dir/开头，  
  
例如：要得到所有person记录的Uri为content://com.bing.provider.personprovider/person，那么返回的MIME类型字符串应该为："vnd.android.cursor.dir/person"。  
  
3）如果要操作的数据属于非集合类型数据，那么MIME类型字符串应该以vnd.android.cursor.item/开头，  
  
例如：得到id为10的person记录，Uri为content://com.bing.provider.personprovider/person/10，那么返回的MIME类型字符串为："vnd.android.cursor.item/person"。

**7、ContentResolver操作ContentProvider中的数据**

ContentProvider使你数据库中数据能够被其他程序访问，但能访问不能任意方式都能访问，只能通过规定的方式，这中方式就是通过ContentResolver来实现

1）当外部应用需要对ContentProvider中的数据进行添加、删除、修改和查询操作时，可以使用ContentResolver 类来完成，要获取ContentResolver 对象，可以使用Activity提供的getContentResolver()方法。

2）ContentResolver 类提供了与ContentProvider类相同签名的四个方法  
  
使用ContentResolver对ContentProvider中的数据进行添加、删除、修改和查询操作：

|  |  |
| --- | --- |
|  | ContentResolver resolver =  getContentResolver();  Uri uri = Uri.parse("<content://com.bing.provider.personprovider/person>");  //添加一条记录  ContentValues values = new ContentValues();  values.put("name", "bingxin");  values.put("age", 25);  resolver.insert(uri, values);  //获取person表中所有记录  Cursor cursor = resolver.query(uri, null, null, null, "personid desc");  while(cursor.moveToNext()){     Log.i("ContentTest", "personid="+ cursor.getInt(0)+ ",name="+ cursor.getString(1));  }  //把id为1的记录的name字段值更改新为zhangsan  ContentValues updateValues = new ContentValues();  updateValues.put("name", "zhangsan");  Uri updateIdUri = ContentUris.withAppendedId(uri, 1);  resolver.update(updateIdUri, updateValues, null, null);  //删除id为2的记录  Uri deleteIdUri = ContentUris.withAppendedId(uri, 2);  resolver.delete(deleteIdUri, null, null); |

**8、监听ContentProvider中数据的变化**  
  
如果ContentProvider的访问者需要知道ContentProvider中的数据发生变化，可以在ContentProvider发生数据变化时调用getContentResolver().notifyChange(uri, null)来通知注册在此URI上的访问者，例子如下：

|  |  |
| --- | --- |
|  | public class PersonContentProvider extends ContentProvider {     public Uri insert(Uri uri, ContentValues values) {        db.insert("person", "personid", values);        getContext().getContentResolver().notifyChange(uri, null);     }  } |

如果ContentProvider的访问者需要得到数据变化通知，必须使用ContentObserver对数据（数据采用uri描述）进行监听，当监听到数据变化通知时，系统就会调用ContentObserver的onChange()方法：

|  |  |
| --- | --- |
|  | getContentResolver().registerContentObserver(Uri.parse("<content://com.ljq.providers.personprovider/person>"),         true, new PersonObserver(new Handler()));  public class PersonObserver extends ContentObserver{     public PersonObserver(Handler handler) {        super(handler);     }     public void onChange(boolean selfChange) {        //此处可以进行相应的业务处理     }  } |

**二、ContentProvider的实现过程**

1、定义一个CONTENT\_URI常量，提供了访问ContentProvider的标识符。

|  |  |
| --- | --- |
|  | public static final Uri CONTENT\_URI = Uri.parse("<content://com.example.codelab.transportationprovider>"); |

2、定义一个类，继承ContentProvider。

|  |  |
| --- | --- |
|  | public class FirstContentProvider extends ContentProvider { |
|  | public static final UriMatcher uriMatcher;        static {                       uriMatcher = new UriMatcher(UriMatcher.NO\_MATCH);                       uriMatcher.addURI(Book.AUTHORITY, "item", Book.ITEM);                       uriMatcher.addURI(Book.AUTHORITY, "item/#", book.ITEM\_ID);                } | |

}  
addURI()方法是用来增加其他URI匹配路径的，

第一个参数传入标识ContentProvider的AUTHORITY字符串。

第二个参数传入需要匹配的路径，这里的#号为通配符，代表匹配任意数字，另外还可以用\*来匹配任意文本。

第三个参数必须传入一个大于零的匹配码，用于match()方法对相匹配的URI返回相对应的匹配码。 例如：sMatcher.addURI(“com.test.provider.personprovider”, “person”, 1);如果match()方法匹配content://com.test.provider.personprovider/person路径，返回匹配码为1。

3、实现query,insert,update,delete,getType和onCreate方法。   
4、在AndroidManifest.xml当中进行声明。

|  |  |
| --- | --- |
|  | <!-- android:name是完成ContentProvider类的全称               android:authorities是和FirstProvidermetaData中的常量AUTHORITY的值一样，否则会报错           -->          <provider android:name="com.bj.FirstContentProvider" android:authorities="com.bj.firstcontentprovider"></provider>  **三、实例**  E:\Android Project\TestCP\app\src\main\java\com\testcp\administrator\testcp  n峨� 夗z{O畫鈛觊wk�'Z喎� |