手机应用平台软件开发

9、数据存储(二)

SQLite数据库

- SQLite是一个开源的嵌入式关系数据库
- 支持SQL92语法，允许开发者使用SQL语句操作数据库中的数据；
- 在2000年由D. Richard Hipp发布
- 广泛应用在很多应用程序中，包括迅雷、360、金山词霸等，在Android中也内置了完整支持的SQLite数据库。

SQLite数据库特点

- 更加适用于嵌入式系统，嵌入到使用它的应用程序中
- 占用非常少，运行高效可靠，可移植性好
- 提供了零配置（zero-configuration）运行模式
- SQLite数据库不仅提高了运行效率，而且屏蔽了数据库使用和管理的复杂性，程序仅需要进行最基本的数据操作，其他操作可以交给进程内部的数据库引擎完成

SQLite是一个轻量级的软件库

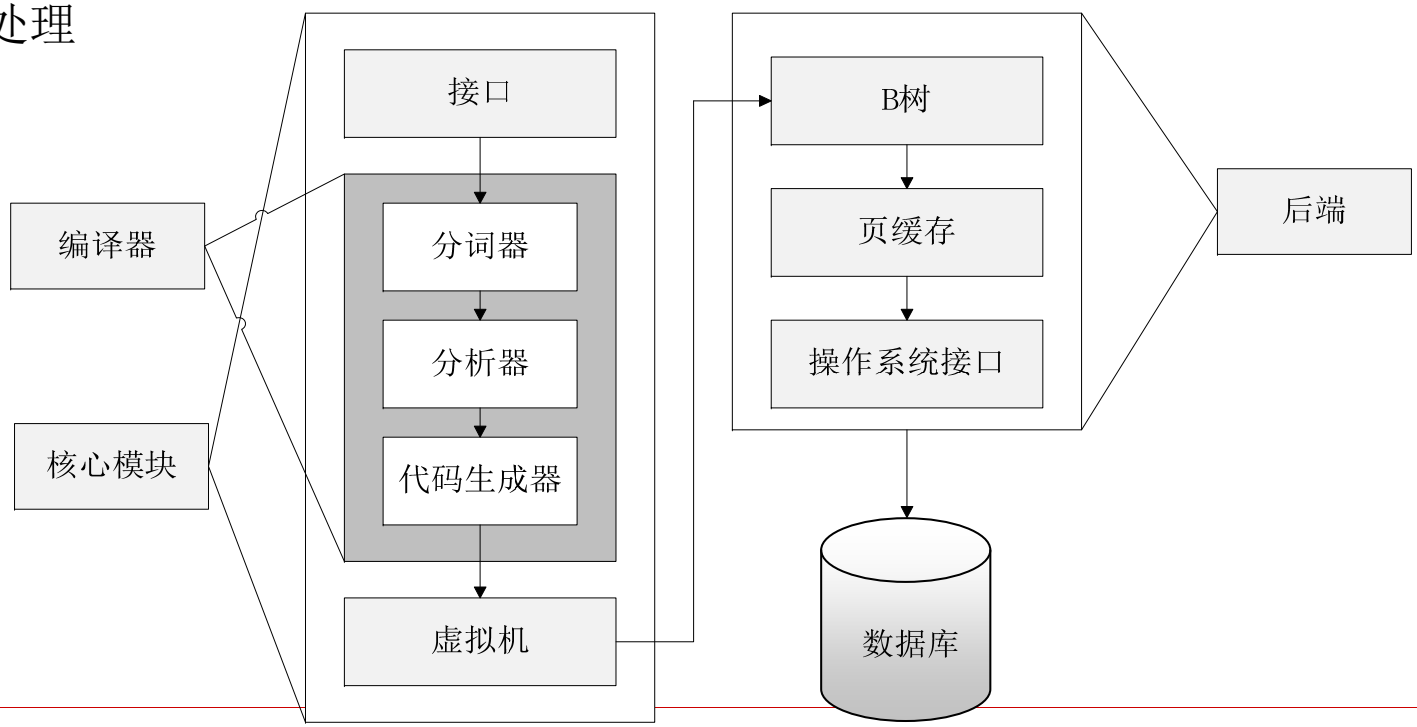
- 原子量性
- 坚固性
- 独立性
- 耐久性
- 体积大小只用几千字节
- 一些SQL 的指令只是部分支持（例如：不支持ALTER TABLE）
- 打开/创建一个SQLite数据库时，其实仅打开一个文件准备读写；

SQLite数据库

- 这些数据库和其中的数据是应用程序所私有的
- 不应该用其来存储文件
- 如果要将其同其他应用程序共享，则必须把应用程序变为一个ContentProvider（后面将讲到）

SQLite数据库

- SQLite数据库采用了模块化设计，由8个独立的模块构成，这些独立模块又构成了三个主要的子系统，模块将复杂的查询过程分解为细小的工作进行处理



SQLite数据库

- SQLite数据库具有很强的移植性，可以运行在Windows，Linux，BSD，Mac OS X和一些商用Unix系统，比如Sun的Solaris，IBM的AIX
- SQLite数据库也可以工作在许多嵌入式操作系统下，例如QNX，VxWorks，Palm OS，Symbian和Windows CE
- SQLite的核心大约有3万行标准C代码，模块化的设计使这些代码更加易于理解

SQLite数据库

- Android SDK包含了若干有用的SQLite数据库管理类大多都存在于android.database.sqlite包中
- SQLite3支持 NULL、INTEGER、REAL（浮点数字）、TEXT(字符串文本)和BLOB(二进制对象)数据类型，虽然它支持的类型虽然只有五种，但实际上sqlite3也接受varchar(n)、char(n)、decimal(p,s) 等数据类型，只不过在运算或保存时会转成对应的五种数据类型。
- 数据库保存在：Android设备/data/data/package_name/databases 文件夹中

创建SQLite数据库

- 配置数据库属性

  重要的属性包括：

  - 版本
  - 本地化（local）
  - 线程安全锁

```
mDatabase.setVersion(1);
mDatabase.setLocale(Locate.getDefault());
mDatabase.setLockingEnable(true);
```

创建SQLite数据库

- 使用应用程序上下文创建SQLite数据库
  - 首先调用openOrCreateDatabases()函数创建数据库对象
  - 然后执行SQL命令建立数据库中的表和直接的关系

```
1.  SQLiteDatabase mDatabase;
2.  mDatabase = Context.openOrCreateDatabase(
3.                  “my_sqlite_database.db”,
4.                  SQLiteDatabase.CREATE_IF_NECESSARY,
5.                  null);
```

创建表

- demoSQLite.execSQL(String sql);
- Pid能够自动增长的唯一id号，并作为表的主键
- content为文本字段

```
1. String sql=“create table demo ( Pid integer primary key
   autoincrement, content text)”;
2. demoSQLite.execSQL(sql);
```

数据操作

- Android提供了一个名为SQLiteDatabase的类，该类封装了一些操作数据库的API。可以调用SQLiteDatabase的静态方法：
- SQLiteDatabase类的公共函数insert()、delete()、update()和query()，封装了执行的添加、删除、更新和查询功能的SQL命令

添加数据

- 首先构造ContentValues对象，然后调用ContentValues对象的put()方法，将每个属性的值写入到ContentValues对象中，最后使用SQLiteDatabase对象的insert()函数，将ContentValues对象中的数据写入指定的数据库表中
- insert()函数的返回值是新数据插入的位置，即ID值。

  ContentValues类是一个数据承载容器，主要用来向数据库表中添加一条数据

采用ContentValues添加数据

```
// Create a new row of values to insert.
ContentValues newValues = new ContentValues();
// Assign values for each row.
newValues.put(COLUMN_NAME, newValue);
[ ... Repeat for each column ... ]
 // Insert the row into your table
myDatabase.insert(DATABASE_TABLE, null, newValues);
```

添加数据

1、Insert方法（利用ContentValues对象）

① ContentValues values = new ContentValues();

② Values.put(String key,Stringf Values);

③ demoSQLite.insert(String table,String nullColumnHack,ContentValues Values);

其中，table是表名，nullColumnHack指的是不允许插入一个完全的空行，若nullColumnHack是一个空值，要明确指定为null值。Values指的是ContentValues对象;

```
① ContentValues values = new ContentValues();
② Values.put(content,"重要数据" );
③ demoSQLite.insert(demo,null, Values);
```

- 添加数据

2、使用execSQL方法

```
① String sql="insert into demo(content) values ("data")"

② demoSQLite.execSQL(sql);
```

修改数据

使用**update**方法

Update(String table, ContentValues values, String whereClause, String[] whereArgs);

1. table是表名;
2. values指的是想要更新的数据;
3. whereClause：满足whereClause子句的记录将会被更新;
4. whereArgs：为whereClause子句传入的参数;

```
ContentValues values=new ContentValues();
Values.put("content","更新数据");
demoSQLite.update("person_inf",values,"_id>?",new Integer[] {20});
```

更新person_inf表中所有主键大于20的人的人名

删 除

- 删除表中的数据

使用delete方法删除数据

Delete (String table,String whereClause,String[] whereArgs);

1. table是表名；
2. whereClause指的是删除的条件；
3. whereArgs用于为whereClause子句传入参数。

- 删 除
  - 删除某个表

删除某个指定的表用execSQL方法实现

```
1. String sql=“drop table demo”
2. demoSQLite.execSQL(sql);
```

- 关闭数据库

```
demoSQLite.close();
```

辅助类

- 数据操作是指对数据的添加、删除、查找和更新的操作
- 数据操作的最佳实践是创建一个辅助类，由它来封装所有对数据库的复杂访问，对于调用代码而言它是透明的。因此创建DBAdapter的辅助类，由它创建、打开、关闭和使用SQLite数据库。

数据操作

为了使DBAdapter类支持对数据的添加、删除、更新和查找等功能，在DBAdapter类中增加下面的这些函数

- insert(People people)用来添加一条数据
- queryAllData()用来获取全部数据
- queryOneData(long id)根据id获取一条数据
- deleteAllData()用来删除全部数据
- deleteOneData(long id)根据id删除一条数据
- updateOneData(long id , People people)根据id更新一条数据

数据操作

```
public class DBAdapter {
    public long insert(People people) {}
    public long deleteAllData() { }
    public long deleteOneData(long id) { }
    public People[] queryAllData() {}
    public People[] queryOneData(long id) { }
    public long updateOneData(long id , People people){ }
    private People[] ConvertToPeople(Cursor cursor){}}
```

- ConvertToPeople(Cursor cursor)是私有函数，作用是将查询结果转换为用来存储数据自定义的People类对象
- People类的包含四个公共属性，分别为ID、Name、Age和Height，对应数据库中的四个属性值

数据操作

People类的代码如下

```
public class People {
	public int ID = -1;
	public String Name;
	public int Age;
	public float Height;
	@Override
	public String toString(){
		String result = "";
		result += "ID：" + this.ID + "，";
		result += "姓名：" + this.Name + "，";
		result += "年龄：" + this.Age + "，  ";
		result += "身高：" + this.Height + "，";
		return result;
	}
}
```

数据操作—添加

```
1. public long insert(People people) {
2.         ContentValues newValues = new ContentValues();
3.         newValues.put(KEY_NAME, people.Name);
4.         newValues.put(KEY_AGE, people.Age);
5.         newValues.put(KEY_HEIGHT, people.Height);
6.         return db.insert(DB_TABLE, null, newValues);
7. }
```

- 第3行代码向ContentValues对象newValues中添加一个名称/值对，put()函数的第1个参数是名称，第2个参数是值
- 在第6行代码的insert()函数中，第1个参数是数据表的名称，第2个参数是在NULL时的替换数据，第3个参数是需要向数据库表中添加的数据

数据操作—删除

删除数据比较简单，只需要调用当前数据库对象的delete()函数，并指明表名称和删除条件即可

```
1.  public long deleteAllData() {
2.          return db.delete(DB_TABLE, null, null);
3.  }
4.  public long deleteOneData(long id) {
5.          return db.delete(DB_TABLE,  KEY_ID + "=" + id, null);
6.  }
```

- delete()函数第1个参数是数据库表名，第2个参数是删除条件
- 在第2行代码中，删除条件为null，表示删除表中的所有数据
- 第5行代码指明了需要删除数据的id值，因此deleteOneData()函数仅删除一条数据，此时delete()函数的返回值表示被删除的数据的数量

数据操作—更新

- 更新数据同样要使用ContentValues对象
- 首先构造ContentValues对象
- 然后调用put()函数将属性的值写入到ContentValues对象
- 最后使用SQLiteDatabase对象的update()函数，并指定数据的更新条件

数据操作—更新

```
1. public long updateOneData(long id , People people){
2.         ContentValues updateValues = new ContentValues();
3.         updateValues.put(KEY_NAME, people.Name);
4.         updateValues.put(KEY_AGE, people.Age);
5.         updateValues.put(KEY_HEIGHT, people.Height);
6.         return db.update(DB_TABLE, updateValues,  KEY_ID + "=" + id, null);
7. }
```

- 在代码的第6行中，update()函数
  - 第1个参数表示数据表的名称，
  - 第3个参数是更新条件。
  - 返回值表示数据库表中被更新的数据数量

数据操作—查询

- 在Android系统中，数据库查询结果的返回值并不是数据集合的完整拷贝，而是返回数据集的指针，这个指针就是Cursor类
- Cursor类支持在查询的数据集合中多种方式移动，并能够获取数据集合的属性名称和序号

数据操作—查询

- Cursor类的方法和说明

函　数	说　明
moveToFirst	将指针移动到第一条数据上
moveToNext	将指针移动到下一条数据上
moveToPrevious	将指针移动到上一条数据上
getCount	获取集合的数据数量
getColumnIndexOrThrow	返回指定属性名称的序号，如果属性不存在则产生异常
getColumnName	返回指定序号的属性名称
getColumnNames	返回属性名称的字符串数组
getColumnIndex	根据属性名称返回序号
moveToPosition	将指针移动到指定的数据上
getPosition	返回当前指针的位置

数据操作—查询

- 从Cursor中提取数据
- 在提取Cursor数据中的数据前，推荐测试Cursor中的数据数量，避免在数据获取中产生异常

```
if (resultCounts == 0 || !cursor.moveToFirst()){
            return null;
    }
```

数据操作—查询

- 从Cursor中提取数据
- 从Cursor中提取数据使用类型安全的get<Type>()函数，函数的输入值为属性的序号，为了获取属性的序号，可以使用getColumnIndex()函数获取指定属性的序号

```
cursor.getString(cursor.getColumnIndex(KEY_NAME));
```

数据库存储

数据操作—查询示例

```
private People[] ConvertToPeople(Cursor cursor){
      int resultCounts = cursor.getCount();
      if (resultCounts == 0 || !cursor.moveToFirst()){
                  return null;}
      People[] peoples = new People[resultCounts];
      for (int i = 0 ; i<resultCounts; i++){
                  peoples[i] = new People();
                  peoples[i].ID = cursor.getInt(0);
                  peoples[i].Name =
cursor.getString(cursor.getColumnIndex(KEY_NAME));
                  peoples[i].Age = cursor.getInt(cursor.getColumnIndex(KEY_AGE));
                  peoples[i].Height =
cursor.getFloat(cursor.getColumnIndex(KEY_HEIGHT));
                  cursor.moveToNext();}
      return peoples;
}
```

把数据库的记录加载到对象数组中！

数据操作—查询

- 查询操作：SQLiteDatabase类的query()函数

```
Cursor android.database.sqlite.SQLiteDatabase.query(String table,
        String[] columns, String selection, String[] selectionArgs,
        String groupBy, String having, String orderBy)
```

位置	类型+名称	说明
1	String table	表名称
2	String[] columns	返回的属性列名称
3	String selection	查询条件
4	String[] selectionArgs	如果在查询条件中使用的问号，则需要定义替换符的具体内容
5	String groupBy	分组方式
6	String having	定义组的过滤器
7	String orderBy	排序方式

数据操作—查询

例1：根据id查询数据的代码

```
public People[] getOneData(long id) {
    Cursor results =  db.query(DB_TABLE, new String[] { KEY_ID, KEY_NAME, KEY_AGE,
KEY_HEIGHT}, KEY_ID + "=" + id, null, null, null, null);
    return ConvertToPeople(results);}
```

例2：根据id查询全部数据的代码

```
public People[] getAllData() {
    Cursor results = db.query(DB_TABLE, new String[] { KEY_ID, KEY_NAME, KEY_AGE,
KEY_HEIGHT}, null, null, null, null, null);
    return ConvertToPeople(results);}
```

数据操作

- 数据操作是指对数据的添加、删除、查找和更新的操作
- 通过执行SQL命名完成数据操作，但推荐使用Android提供的专用类和方法，这些类和方法更加简洁、易用

SQLiteOpenHelper的使用方法

SQLiteOpenHelper是一个辅助类，主要用来管理数据库的创建和版本

。 可以通过继承这个类，实现它的一些方法来对数据库进行一些操作

。 所有继承了这个类的类都必须实现下面这样的一个构造方法：

```
public DatabaseHelper(Context context, String name,
CursorFactory factory, int version)
```

1. 第一个参数：Context类型，上下文对象；
2. 第二个参数：String类型，数据库的名称
3. 第三个参数：CursorFactory类型
4. 第四个参数：int类型，数据库版本

ContentProvider

- Android四大组件之一；
- ContentProvider（数据提供者）是在应用程序间共享数据的一种接口机制（也可以看成数据共享标准）
- 提供了更为高级的数据共享方法，应用程序可以指定需要共享的数据，而其他应用程序则可以在不知数据来源、路径的情况下，对共享数据进行查询、添加、删除和更新等操作
- 许多Android系统的内置数据通过ContentProvider提供给用户使用，例如通讯录、音视频文件和图像文件等

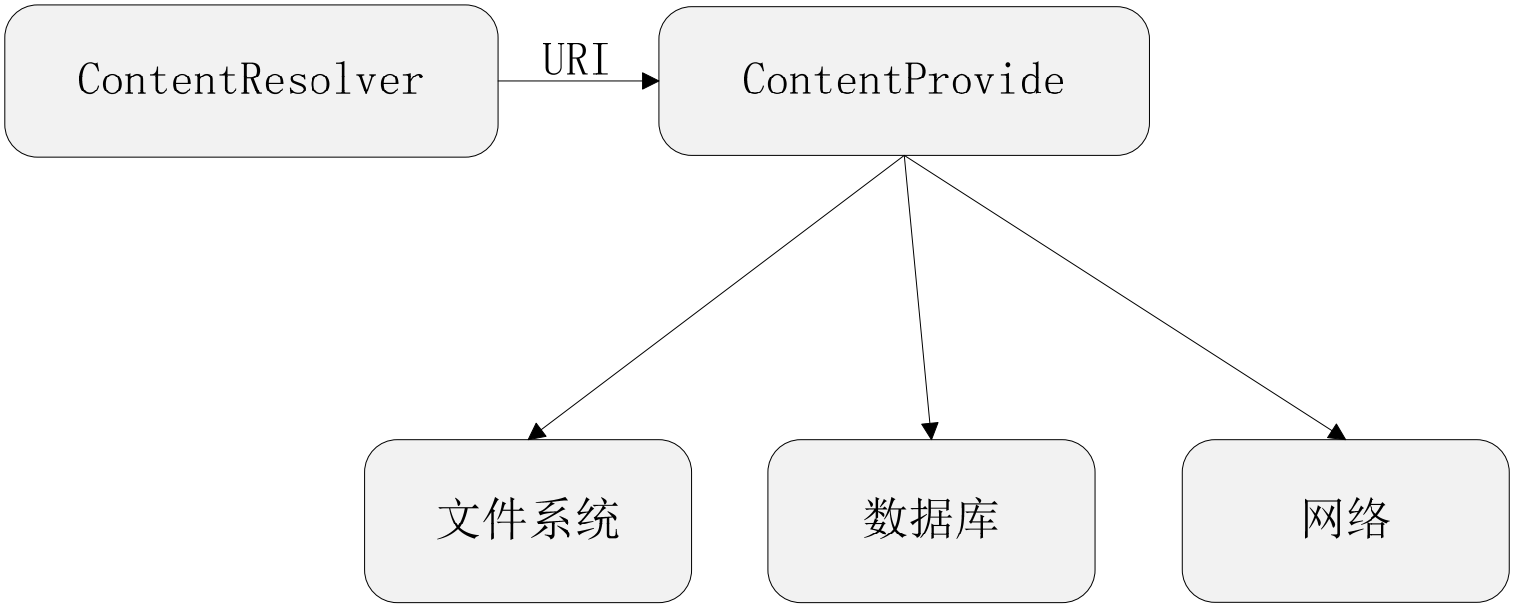
创建ContentProvider基本过程

1. 使用数据库、文件系统或网络实现底层存储功能

2. 在继承ContentProvider的类中实现基本数据操作的接口函数，包括添加、删除、查找和更新等功能

ContentProvider

- 调用者不能够直接调用ContentProvider的接口函数
- 需要使用ContentResolver对象，通过URI间接调用ContentProvider



ContentProvider

- 程序开发人员使用ContentResolver对象与ContentProvider进行交互，而ContentResolver则通过URI确定需要访问的ContentProvider的数据集
- 在发起一个请求的过程中，Android首先根据URI确定处理这个查询的ContentResolver，然后初始化ContentResolver所有需要的资源，这个初始化的工作是Android系统完成的，无需程序开发人员参与
- 一般情况下只有一个ContentResolver对象，但却可以同时与多个ContentProvider进行交互

ContentProvider

- ContentProvider完全屏蔽了数据提供组件的数据存储方法
- 在使用者看来，数据提供者通过ContentProvider提供了一组标准的数据操作接口，却无法得知数据提供者的数据存储方式

ContentProvider

- 数据提供者可以使用SQLite数据库存储数据，也可以通过文件系统或SharedPreferences存储数据，甚至是使用网络存储的方法，这些内容对数据使用者都是不可见
- 同时也正是因为屏蔽数据的存储方法，很大程度上简化的ContentProvider的使用难度，使用者只要调用ContentProvider提供的接口函数，就可完成所有的数据操作

ContentProvider

- ContentProvider的数据模式类似于数据库的数据表，每行是一条记录，每列具有相同的数据类型
- 每条记录都包含一个长型的字段_ID，唯一标识该记录
- ContentProvider可以提供多个数据集，调用者使用URI对不同的数据集的数据进行操作
- ContentProvider数据模型

_ID	NAME	AGE	HEIGHT
1	Tom	21	1.81
2	Jim	22	1.78

ContentProvider

- URI是通用资源标志符（Uniform Resource Identifier），用来定位任何远程或本地的可用资源
- ContentProvider使用的URI语法结构

  content://<authority>/<data_path>/<id>

  - content://是通用前缀，表示该URI用于ContentProvider定位资源，无需修改
  - <authority>是授权者名称，用来唯一确定由哪一个ContentProvider提供资源，一般由类的小写全称组成

ContentProvider

- <data_path>是数据路径，用来确定请求的是哪个数据集
  - 如果ContentProvider仅提供一个数据集，数据路径可以省略
  - 如果ContentProvider提供多个数据集，数据路径则必须指明
  - 数据集的数据路径可以写成多段格式，例如/people/girl和/people/boy。
- <id>是数据编号，用来唯一确定数据集中的一条记录，用来匹配数据集中_ID字段的值

ContentProvider

- 如果请求的数据并不只限于一条数据，则<id>可以省略
  - 请求整个people数据集的URI应写为

content://edu.sysu.peopleprovider/student

  - 请求people数据集中第3条数据的URI应写为

content://edu.sysu.peopleprovider/student/3

ContentProvider

- 系统样例
  1. Content://media/internal/images 返回设备上所有图片；
  2. Content://contacts/people/5 返回联系人中ID为5的联系人记录；
  3. Content：//contacts/people 返回设备中所有联系人信息；
- 系统提供的简便方法
  1. Uri uri=ContentUris.withAppendedId(People.CONTENT_URI,10)
  2. Uri uri=Uri.withAppendedPath(Contacts.CONTENT_URI,”10”);

ContentProvider

- 数据查询（两种方法参数一样，返回都是**Cursor**）

  1. ContentResolver.query()方法；
  2. Activity.managedQuery()方法；

  Cursor=Query（Uri uri, String[] projection, String selection, String selectionArgs,String sortOrder）；

  Projection 返回的数据字段；selection where的条件子句；selectionArgs where字句的参数；

- 样例

  1. String columns[]=new String[] {People.NAME,People.number };
  2. Uri contacts = People.CONTENT_URI；
  3. Cursor cur=managedQuery(contacts,columns,null,null,null);

ContentProvider

- 数据修改

  ContentResolver.update（）方法

  Int i=Update（Uri uri，ContentValues values，String where，String[] selectionArgs ）

  Values 表示要修改的数据；where 可选参数 表示where条件字句；selectionArgs是可选参数，表示where子句参数。

- 样例(修改移动电话及号码)

```
Uri uri=ContentUris.withAppendedId(People.CONTENT_URI,12)
ContentValues values=new ContentValues();
Values.put(People.Phones.TYPE,People.Phones.TYPE_MOBILE);
Values.put(People.Phones.NUMBER,"12345678901");
getContentResolver().update(uri,values,null,null);
```

ContentProvider

- 数据添加

  ContentResolver.insert（）方法

  Uri uri=insert（Uri uri，ContentValues values）

  返回值为新的uri，可通过这个uri获得包含着条记录的Cursor对象。

- 样例(修改移动电话及号码)

```
ContentValues values=new ContentValues（）；
Values.put（people.NAME,“张三"）;
Uri uri=getContentResolver().insert(People.CONTENT_URI，values)；
```

ContentProvider

- 系统样例
  1. Content://media/internal/images 返回设备上所有图片；
  2. Content://contacts/people/5 返回联系人中ID为5的联系人记录；
  3. Content：//contacts/people 返回设备中所有联系人信息；
- 系统提供的简便方法
  1. Uri uri=ContentUris.withAppendedId(People.CONTENT_URI,10)
  2. Uri uri=Uri.withAppendedPath(Contacts.CONTENT_URI,”10”);

创建数据提供者

- 程序开发人员通过继承ContentProvider类可以创建一个新的数据提供者，过程可以分为三步
  1. 继承ContentProvider，并重载六个函数
  2. 声明CONTENT_URI，实现UriMatcher
  3. 注册ContentProvider

第1步：继承ContentProvider，并重载六个函数

- delete():删除数据集
- insert()：添加数据集
- qurey()：查询数据集
- update()：更新数据集
- onCreate()：初始化底层数据集和建立数据连接等工作
- getType()：返回指定URI的MIME数据类型
  1. 若URI是单条数据，则返回的MIME数据类型应以vnd.android.cursor.item开头
  2. 若URI是多条数据，则返回的MIME数据类型应以vnd.android.cursor.dir/开头

创建数据提供者

- 第1步：继承ContentProvider，并重载六个函数

  新建立的类继承ContentProvider后，Eclipse会提示程序开发人员需要重载部分代码，并自动生成需要重载的代码框架

```
import android.content.*;
import android.database.Cursor;
import android.net.Uri;
public class PeopleProvider extends ContentProvider{
    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {
        // TODO Auto-generated method stub
        return 0;}
      @Override
    public String getType(Uri uri) {
        // TODO Auto-generated method stub
        return null;
    }
    @Override
    public Uri insert(Uri uri, ContentValues values) {
        // TODO Auto-generated method stub
        return null;
    }
    @Override
    public boolean onCreate() {
        // TODO Auto-generated method stub
        return false;
    }
      @Override
    public Cursor query(Uri uri, String[] projection, String selection,
            String[] selectionArgs, String sortOrder) {
        // TODO Auto-generated method stub
        return null;
    }
    @Override
    public int update(Uri uri, ContentValues values, String selection,
            String[] selectionArgs) {
        // TODO Auto-generated method stub
        return 0;
    }
}
```

创建数据提供者

- 第2步：声明CONTENT_URI，实现UriMatcher
  - 在新构造的ContentProvider类中，通过构造一个UriMatcher，判断URI是单条数据还是多条数据
  - 为了便于判断和使用URI，一般将URI的授权者名称和数据路径等内容声明为静态常量，并声明CONTENT_URI

创建数据提供者

- 第2步：声明CONTENT_URI和构造UriMatcher的代码

```
1.  public static final String AUTHORITY = "edu.sysu.peopleprovider";
2.  public static final String PATH_SINGLE = "people/#";
3.  public static final String PATH_MULTIPLE = "people";
4.  public static final String CONTENT_URI_STRING = "content://" + AUTHORITY
    + "/" + PATH_MULTIPLE;
5.  public static final Uri  CONTENT_URI = Uri.parse(CONTENT_URI_STRING);
6.  private static final int MULTIPLE_PEOPLE = 1;
7.  private static final int SINGLE_PEOPLE = 2;
8.
9.  private static final UriMatcher uriMatcher;
10.static {
11.       uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
12.       uriMatcher.addURI(AUTHORITY, PATH_SINGLE, MULTIPLE_PEOPLE);
13.       uriMatcher.addURI(AUTHORITY, PATH_MULTIPLE, SINGLE_PEOPLE);
14.}
```

创建数据提供者

- 第1行代码声明了URI的授权者名称
- 第2行代码声明了单条数据的数据路径
- 第3行代码声明了多条数据的数据路径
- 第4行代码声明了CONTENT_URI的字符串形式
- 第5行代码则正式声明了CONTENT_URI
- 第6行代码声明了多条数据的返回代码
- 第7行代码声明了单条数据的返回代码
- 第9行代码声明了UriMatcher
- 第10行到第13行的静态构造函数中，声明了UriMatcher的匹配方式和返回代码

创建数据提供者

- 其中第11行UriMatcher的构造函数中，UriMatcher.NO_MATCH表示URI无匹配时的返回代码
- 第12行的addURI()函数用来添加新的匹配项，语法如下

```
public void  addURI  (String authority, String path, int code)
```

  - authority表示匹配的授权者名称
  - path表示数据路径
  - #可以代表任何数字
  - code表示返回代码

创建数据提供者

- 第2步：声明CONTENT_URI，实现UriMatcher
  - 使用UriMatcher时，则可以直接调用match()函数，对指定的URI进行判断，代码如下

```
switch(uriMatcher.match(uri)){
        case MULTIPLE_PEOPLE:
                //多条数据的处理过程
                break;
        case SINGLE_PEOPLE:
                //单条数据的处理过程
                break;
        default:
                throw new IllegalArgumentException("不支持的URI:" + uri);
}
```

创建数据提供者

- 第3步：注册ContentProvider
  1. 在完成ContentProvider类的代码实现后，需要在AndroidManifest.xml文件中进行注册
  2. 注册ContentProvider使用<provider>标签，代码如下

```
<application android:icon="@drawable/icon" android:label="@string/app_name">
    <provider android:name = ".PeopleProvider"
              android:authorities = "edu.sysu.peopleprovider"/>
</application>
```

  3. 在上面的代码中，注册了一个授权者名称为edu.sysu.peopleprovider的ContentProvider，其实现类是PeopleProvider

使用数据提供者

- 使用ContentProvider是通过Android组件都具有的ContentResolver对象，通过URI进行数据操作
- 程序开发人员只需要知道URI和数据集的数据格式，则可以进行数据操作，解决不同应用程序之间的数据共享问题
- 每个Android组件都具有一个ContentResolver对象，获取ContentResolver对象的方法是调用getContentResolver()函数

```
ContentResolver resolver = getContentResolver();
```

使用数据提供者

- 查询操作
  - 在获取到ContentResolver对象后，程序开发人员则可以使用query()函数查询目标数据
  - 下面的代码是查询ID为2的数据

```
String KEY_ID = "_id";
String KEY_NAME = "name";
String KEY_AGE = "age";
String KEY_HEIGHT = "height";
Uri uri = Uri.parse(CONTENT_URI_STRING + "/" + "2";
Cursor cursor = resolver.query(uri,
    new String[] {KEY_ID, KEY_NAME, KEY_AGE, KEY_HEIGHT}, null, null, null);
```

  - 在URI中定义了需要查询数据的ID，在query()函数并没有额外声明查询条件

使用数据提供者

- 查询操作
  - 如果需要获取数据集中的全部数据，则可直接使用CONTENT_URI，此时ContentProvider在分析URI时将认为需要返回全部数据
  - ContentResolver的query()函数与SQLite数据库的query()函数非常相似，语法结构如下

```
Cursor query(Uri uri, String[] projection, String selection, String[]
selectionArgs, String sortOrder)
```

    - uri定义了查询的数据集
    - projection定义了从数据集返回哪些数据项
    - selection定义了返回数据的查询条件

使用数据提供者

- 添加操作
  - 向ContentProvider中添加数据有两种方法
    - 一种是使用insert()函数，向ContentProvider中添加一条数据
    - 另一种是使用bultInsert()函数，批量的添加数据

使用数据提供者

- 添加操作
  - 例1：如何使用insert()函数添加单条数据

```
1. ContentValues values = new ContentValues();
2. values.put(KEY_NAME, "Tom");
3. values.put(KEY_AGE, 21);
4. values.put(KEY_HEIGHT, );
5. Uri newUri = resolver.insert(CONTENT_URI, values);
```

  - 例2：如何使用bultInsert()函数添加多条数据

```
1. ContentValues[] arrayValues = new ContentValues[10];
2. //实例化每一个ContentValues
3. int count = resolver.bultInsert(CONTENT_URI, arrayValues);
```

使用数据提供者

- 删除操作
  - 删除操作需要使用delete()函数
  - 如果需要删除单条数据，则可以在URI中指定需要删除数据的ID
  - 如果需要删除多条数据，则可以在selection中声明删除条件

使用数据提供者

- 删除操作
  - 例1：如何删除ID为2的数据

```
Uri uri = Uri.parse(CONTENT_URI_STRING + "/" + "2");
int result = resolver.delete(uri, null, null);
```

  - 例2：在selection将删除条件定义为ID大于4的数据

```
String selection = KEY_ID + ">4";
int result = resolver.delete(CONTENT_URI, selection, null);
```

使用数据提供者

- 更新操作
  - 更新操作需要使用update()函数，参数定义与delete()函数相同，同样可以在URI中指定需要更新数据的ID，也可以在selection中声明更新条件
  - 例：如何更新ID为7的数据

```
ContentValues values = new ContentValues();
values.put(KEY_NAME, "Tom");
values.put(KEY_AGE, 21);
values.put(KEY_HEIGHT, );
Uri uri = Uri.parse(CONTENT_URI_STRING + "/" + "7");
int result = resolver.update(uri, values, null, null);
```

Questions?