第8章 数据存储与访问

杨刚
中国人民大学

本章学习目标：

- 掌握SharedPreferences的使用方法
- 掌握各种文件存储的区别与适用情况
- 了解SQLite数据库的特点和体系结构
- 掌握SQLite数据库的建立和操作方法
- 理解ContentProvider的用途和原理
- 掌握ContentProvider的创建与使用方法

8.1 SharedPreferences

- SharedPreferences是一种轻量级的数据保存方式
- 通过SharedPreferences开发人员可以将名称/值对（Name/Value Pair）保存在Android的文件系统中，而且SharedPreferences完全屏蔽了对文件系统的操作过程
- 使用SharedPreferences类创建的命名映射，可以在会话之间持久化，并在同一个应用程序沙箱中运行的程序组件之间共享。

8.1 SharedPreferences

- 存储的SharedPreferencesy是以XML文件的格式方式自动保存的，可以利用开发环境中DDMS在file explorer中的/data/data/程序包名/shared_prefs中来查看

8.1SharedPreference

- SharedPreferences支持3种访问操作模式
  - 私有（MODE_PRIVATE）：仅创建SharedPreferences的程序有权限对其进行读取或写入
  - 全局读（MODE_WORLD_READABLE）：不仅创建程序可以对其进行读取或写入，其它应用程序也具有读取操作的权限，但没有写入操作的权限
  - 全局写（MODE_WORLD_WRITEABLE）：所有程序都可以对其进行写入操作，但没有读取操作的权限
  - 后两种模式可以组合

8.1 SharedPreferences

- 定义SharedPreferences的访问模式，如下定义访问模式为私有

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

- 有的时候需要将SharedPreferences的访问模式设定为即可以全局读，也可以全局写，
-

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

8.1 SharedPreferences

- 除了定义SharedPreferences的访问模式，还要定义其名称，这个名称也是SharedPreferences在Android文件系统中保存的文件名称
- 一般将SharedPreferences名称声明为字符串常量，这样可以在代码中多次使用

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

- 使用SharedPreferences时需要将访问模式和SharedPreferences名称作为参数传递到getSharedPreferences()函数，则可获取到SharedPreferences实例

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

8.1 SharedPreferences

- 在获取到SharedPreferences实例后，可以通过SharedPreferences.Editor类对SharedPreferences进行修改，最后调用commit()函数保存修改内容
- SharedPreferences广泛支持各种基本数据类型，包括整型、布尔型、浮点型和长型等

```
sharedPreferences = getSharedPreferences(PREFS_READ_WRITE, Context.MODE_W
                    ORLD_READABLE + Context.MODE_WORLD_WRITEABLE);

Editor prefsPrivateEditor = sharedPreferences.edit();

prefsPrivateEditor.putString(KEY_READ_WRITE, read_writeField.getText().toString());

booleanresult = prefsPrivateEditor.commit();
```

8.1 SharedPreferences

- 建议使用Apply方法，调用它会安全地异步写入Shared Preference Editor。异步保证了安全性。
- 对比commit方法

```
editor.apply();
```

8.1 SharedPreferences

- 如果需要从已经保存的SharedPreferences中读取数据，同样是调用getSharedPreferences()函数，并在函数第1个参数中指明需要访问的SharedPreferences名称，最后通过get<Type>()函数获取保存在SharedPreferences中的NVP
- get<Type>()函数的第1个参数是键的名称
- 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);
4  float height = sharedPreferences.getFloat("Height",1.81f);
```

8.1 SharedPreferences

- 访问其他应用程序的SharedPreferences需要的3个条件：
  - 访问模式为全局读或全局写
  - 需要知道共享者的包名和SharedPreferences名称，通过Context获取对象
  - 需要知道每个数据的名称和数据类型以读取数据

```
Context otherAppsContext = createPackageContext("cn.itcast.action",
                                        Context.CONTEXT_IGNORE_SECURITY);
SharedPreferences sharedPreferences =
            otherAppsContext.getSharedPreferences("itcast",
                                                Context.MODE_WORLD_READABLE);

String name = sharedPreferences.getString("name", "");
int age = sharedPreferences.getInt("age", 0);
```

8.2 文件存储

- 8.2.1 内部存储
  - 虽然SharedPreferences能够为开发人员简化数据存储和访问过程，但直接使用文件系统保存数据仍然是Android数据存储中不可或缺的组成部分
  - Android使用Linux的文件系统，开发人员可以建立和访问程序自身建立的私有文件，也可以访问保存在资源目录中的原始文件和XML文件，还可以将文件保存在TF卡等外部存储设备中

8.2 文件存储

- 8.2.1 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Android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
  - 主要使用两个函数
    - openFileOutput()
    - openFileInput()

8.2 文件存储

- 8.2.1 内部存储
  - openFileOutput()函数
    - openFileOutput()函数为写入数据做准备而打开文件
    - 如果指定的文件存在，直接打开文件准备写入数据
    - 如果指定的文件不存在，则创建一个新的文件
    - openFileOutput()函数的语法格式如下

      public FileOutputStream openFileOutput(String name, int mode)
      - 第1个参数是文件名称，这个参数不可以包含描述路径的斜杠
      - 第2个参数是操作模式，Android系统支持四种文件操作模式
    - 函数的返回值是FileOutputStream类型

8.2 文件存储

- 8.2.1 内部存储
  - openFileOutput()函数
    - 四种文件操作模式

模式	说明
MODE_PRIVATE	私有模式，文件仅能够被创建文件的程序访问，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。

8.2 文件存储

- 8.2.1 内部存储
  - openFileOutput()函数
    - 当调用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，等数据量积攒到一定程度时再将数据一次性写入文件，因此，在调用close()函数关闭文件前，必须要调用flush()函数，将缓冲区内所有的数据写入文件，否则可能会导致部分数据丢失

8.2 文件存储

- 8.2.1 内部存储
  - openFileInput()函数
    - openFileInput()函数为读取数据做准备而打开文件
    - openFileInput()函数的语法格式如下

      public FileInputStream openFileInput (String name)
      - 第1个参数也是文件名称，同样不允许包含描述路径的斜杠
      - 使用openFileInput()函数打开已有文件，并以二进制方式读取数据的示例代码如下

```
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  }
```

8.2 文件存储

- 8.2.2 外部存储
  - 程序在模拟器中运行前，还必须在AndroidManifest.xml中注册两个用户权限，分别是加载卸载文件系统的权限和向外部存储器写入数据的权限
  - AndroidManifest.xml的核心代码如下

```
1  <uses-permission
   android:name="android.permission.MOUNT_UNMOUNT_
   FILESYSTEMS"> </uses-permission>
2  <uses-permission
   android:name="android.permission.WRITE_EXTERNAL_S
   TORAGE"> </uses-permission>
```

8.2 文件存储

- 8.2.3 资源文件
  - 在/res/raw和/res/xml目录中保存着原始格式文件和XML文件，这些文件是程序开发阶段在工程中保存的文件
  - 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、图像文件或数据文件等等
  - 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。而/res/xml目录下一般用来保存格式化数据的XML文件，则会在编译和打包时将XML文件转换为二进制格式，用以降低存储器空间占用和提高访问效率，在应用程序运行的时候会以特殊的方式进行访问

8.2 文件存储

8.2.3 资源文件

- /res/xml目录下的XML文件与其它资源文件有所不同，程序开发人员不能够以流的方式直接读取，其主要原因在于Android系统为了提高读取效率，减少占用的存储空间，将XML文件转换为一种高效的二进制格式
- 在程序运行时读取/res/xml目录下的XML文件
  - 首先在/res/xml目录下创建一个名为people.xml的文件
  - XML文件定义了多个<person>元素，每个<person>元素都包含三个属性name、age和height，分别表示姓名、年龄和身高
- /res/xml/people.xml文件代码如下

```
1  <people>
2      <person name="李某某" age="21" height="1.81" />
3      <person name="王某某" age="25" height="1.76" />
4      <person name="张某某" age="20" height="1.69" />
5  </people>
```

8.3 数据库存储

- 8.3.1 SQLite数据库
  - SQLite是一个2000年由D.Richard Hipp发布的开源嵌入式关系数据库
  - 普通数据库的管理系统比较庞大和复杂，会占用了较多的系统资源
  - 轻量级数据库SQLite的特点
    - 比传统数据库更适合用于嵌入式系统
    - 占用资源少，运行高效可靠，可移植性强
    - 提供了零配置（zero-configuration）运行模式

8.3 数据库存储

- 8.3.1 SQLite数据库
  - 遵循ACID要求
    - 原子性：一个交易（transaction）中的所有操作，要么全部完成，要么全部不完成，不会结束在中间某个环节。事务在执行过程中发生错误，会被回滚（Rollback）到事务开始前的状态，就像这个事务从来没有执行过一样。
    - 一致性：在事务开始之前和事务结束以后，数据库的完整性没有被破坏。这表示写入的资料必须完全符合所有的默认规则，这包含资料的精确度、串联性以及后续数据库可以自发性地完成预定的工作。

8.3 数据库存储

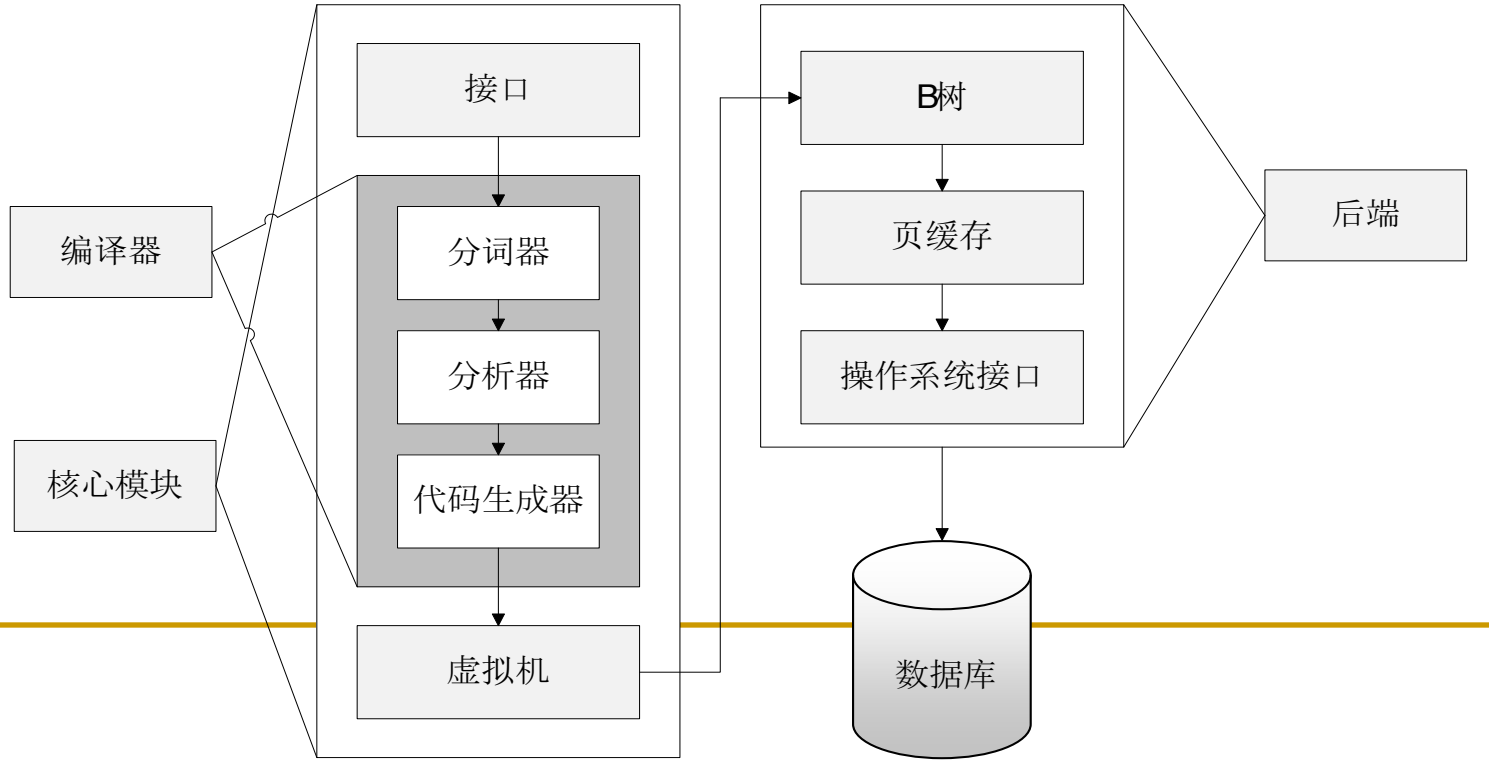
- 8.3.1 SQLite数据库
  - 遵循ACID要求
    - 隔离性：当两个或者多个事务并发访问（此处访问指查询和修改的操作）数据库的同一数据时所表现出的相互关系。事务隔离分为不同级别，包括读未提交（Read uncommitted）、读提交（read committed）、可重复读（repeatable read）和串行化（Serializable）。
    - 持久性：在事务完成以后，该事务对数据库所作的更改便持久地保存在数据库之中，并且是完全的。

8.3 数据库存储

- 8.3.1 SQLite数据库
  - SQLite数据库的优势
    - 可以嵌入到使用它的应用程序中
      - 提高了运行效率
      - 屏蔽了数据库使用和管理的复杂性
    - 客户端和服务器在同一进程空间运行
      - 完全不需要进行网络配置和管理
      - 减少了网络调用所造成的额外开销
    - 简化了数据库的管理过程
      - 应用程序更加易于部署和使用
      - 只需要把SQLite数据库正确编译到应用程序中

8.3 数据库存储

- 8.3.1 SQLite数据库
  - SQLite数据库采用了模块化设计，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



8.3 数据库存储

- 8.3.1 SQLite数据库
  - 接口
    - 由SQLite C API组成，因此无论是应用程序、脚本，还是库文件，最终都是通过接口与SQLite交互
  - 编译器
    - 在编译器中，分词器和分析器对SQL语句进行语法检查，然后把SQL语句转化为便于底层处理的分层数据结构，这种分层的数据结构称为“语法树”
    - 然后把语法树传给代码生成器进行处理，生成一种用于SQLite的汇编代码，最后由虚拟机执行

8.3 数据库存储

- 8.3.1 SQLite数据库
  - 虚拟机
    - SQLite数据库体系结构中最核心的部分是虚拟机，也称为虚拟数据库引擎（Virtual Database Engine，VDBE）
    - 与Java虚拟机相似，虚拟数据库引擎用来解释并执行字节代码
    - 虚拟数据库引擎的字节代码由128个操作码构成，这些操作码主要用以对数据库进行操作，每一条指令都可以完成特定的数据库操作，或以特定的方式处理栈的内容

8.3 数据库存储

- 8.3.1 SQLite数据库
  - 后端
    - 后端由B-树、页缓存和操作系统接口构成，B-树和页缓存共同对数据进行管理
    - B-树的主要功能就是索引，它维护着各个页面之间复杂的关系，便于快速找到所需数据
    - 页缓存的主要作用是通过操作系统接口在B-树和磁盘之间传递页面

8.3 数据库存储

- 8.3.1 SQLite数据库
  - 移植性
    - 可以运行在Windows、Linux、BSD、Mac OS和一些商用Unix系统，比如Sun的Solaris或IBM的AIX
    - 嵌入式操作系统下，比如QNX、VxWorks、Palm OS、Symbin和Windows CE
  - SQLite的核心大约有3万行标准C代码，因为模块化的设计使这些代码非常易于理解

8.3 数据库存储

- 8.3.3 建库操作
  - 在代码中动态建立数据库是比较常用的方法
  - 例如在程序运行过程中，当需要进行数据库操作时，应用程序会首先尝试打开数据库，此时如果数据库并不存在，程序则会自动建立数据库，然后再打开数据库
  - 在编程实现时，一般将所用对数据库的操作都封装在一个类中，因此只要调用这个类，就可以完成对数据库的添加、更新、删除和查询等操作

8.3 数据库存储

- 8.3.3 建库操作
  - 程序开发人员不应直接调用onCreate()和onUpgrade()函数，而应由SQLiteOpenHelper类来决定何时调用这两个函数
  - SQLiteOpenHelper类的getWritableDatabase()函数和getReadableDatabase()函数是可以直接调用的函数
  - getWritableDatabase()函数用来建立或打开可读写的数据库实例，一旦函数调用成功，数据库实例将被缓存，在需要使用数据库实例时就可以调用这个方法获取数据库实例，务必在不使用时调用close()函数关闭数据库
  - 如果保存数据库文件的磁盘空间已满，调用getWritableDatabase()函数则无法获得可读写的数据库实例，这时可以调用getReadableDatabase()函数，获得一个只读的数据库实例

8.3 数据库存储

- 8.3.3建库操作
  - 如果不希望使用SQLiteOpenHelper类，也可以直接使用SQL命令建立数据库，方法是：
    - 先调用openOrCreateDatabases()函数创建数据库实例
    - 然后调用execSQL()函数执行SQL命令，完成数据库和数据表的建立过程，其示例代码如下

```
1    private static final String DB_CREATE = "create table " +
2          DB_TABLE + " (" + KEY_ID + " integer primary key autoincrement, " +
3          KEY_NAME+ " text not null, " + KEY_AGE+ " integer," + KEY_HEIGHT
     + " float);";
4    public void create() {
5          db.openOrCreateDatabases(DB_NAME, context.MODE_PRIVATE, null)
6          db.execSQL(DB_CREATE);
7    }
```

8.3 数据库存储

8.3.4 数据操作

- 数据操作指的是对数据的添加、删除、查找和更新操作
- 虽然程序开发人员完全可以通过执行SQL命令完成数据操作，但这里仍然推荐使用Android提供的专用类和方法，这些类和方法的使用更加简洁、方便

- 为了使DBAdapter类支持数据添加、删除、更新和查找等功能，在DBAdapter类中增加下面的函数
  - 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

8.3 数据库存储

- 8.3.4 数据操作
  - SQLiteDatabase类的公有函数insert()、delete()、update()和query()，封装了执行添加、删除、更新和查询功能的SQL命令

8.3 数据库存储

- 8.3.4 数据操作
  - 添加功能
    - 首先构造一个ContentValues实例，然后调用ContentValues实例的put()方法，将每个属性的值写入到ContentValues实例中，最后使用SQLiteDatabase实例的insert()函数，将ContentValues实例中的数据写入到指定的数据表中
    - 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

8.3 数据库存储

- 8.3.4 数据操作
  - 删除功能
    - 删除数据比较简单，只需要调用当前数据库实例的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
5    public long deleteOneData(long id) {
6            return db.delete(DB_TABLE,  KEY_ID + "=" + id, null);
7    }
```

8.3 数据库存储

- 8.3.4 数据操作
  - 更新功能
    - 更新数据同样要使用ContentValues实例，首先构造ContentValues实例，然后调用put()函数将属性值写入到ContentValues实例中，最后使用SQLiteDatabase的update()函数，并指定数据的更新条件

```
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
7            return db.update(DB_TABLE, updateValues,  KEY_ID + "=" + id, null);
8    }
```

8.3 数据库存储

- 8.3.4 数据操作
  - 查询功能
    - 介绍查询功能前，先要介绍一下Cursor类
    - 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类支持在查询结果的数据集合中以多种方式移动，并能够获取数据集合的属性名称和序号，具体的方法和说明可以参考下表

8.3 数据库存储

8.3.4 数据操作

Cursor类的公有方法

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

8.3 数据库存储

- 8.3.4 数据操作
  - 从Cursor中提取数据可以参考ConvertToPeople()函数的实现方法
  - 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
  - 从Cursor中提取数据使用类型安全的get<Type>()函数，函数的参数是属性的序号，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

8.3 数据库存储

8.3.4 数据操作

- 要进行数据查询就需要调用SQLiteDatabase类的query()函数，query()函数的语法如下

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

- query()函数的参数说明

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通配符(?)，则需要在这里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

8.3 数据库存储

- 8.3.4 数据操作
  - 根据id查询数据的代码

```
1    public People[] getOneData(long id) {
2        Cursor results =  db.query(DB_TABLE, new String[] { KEY_ID,
     KEY_NAME, KEY_AGE, KEY_HEIGHT}, KEY_ID + "=" + id, null, null,
     null, null);
3        return ConvertToPeople(results);
4    }
```

8.3 数据库存储

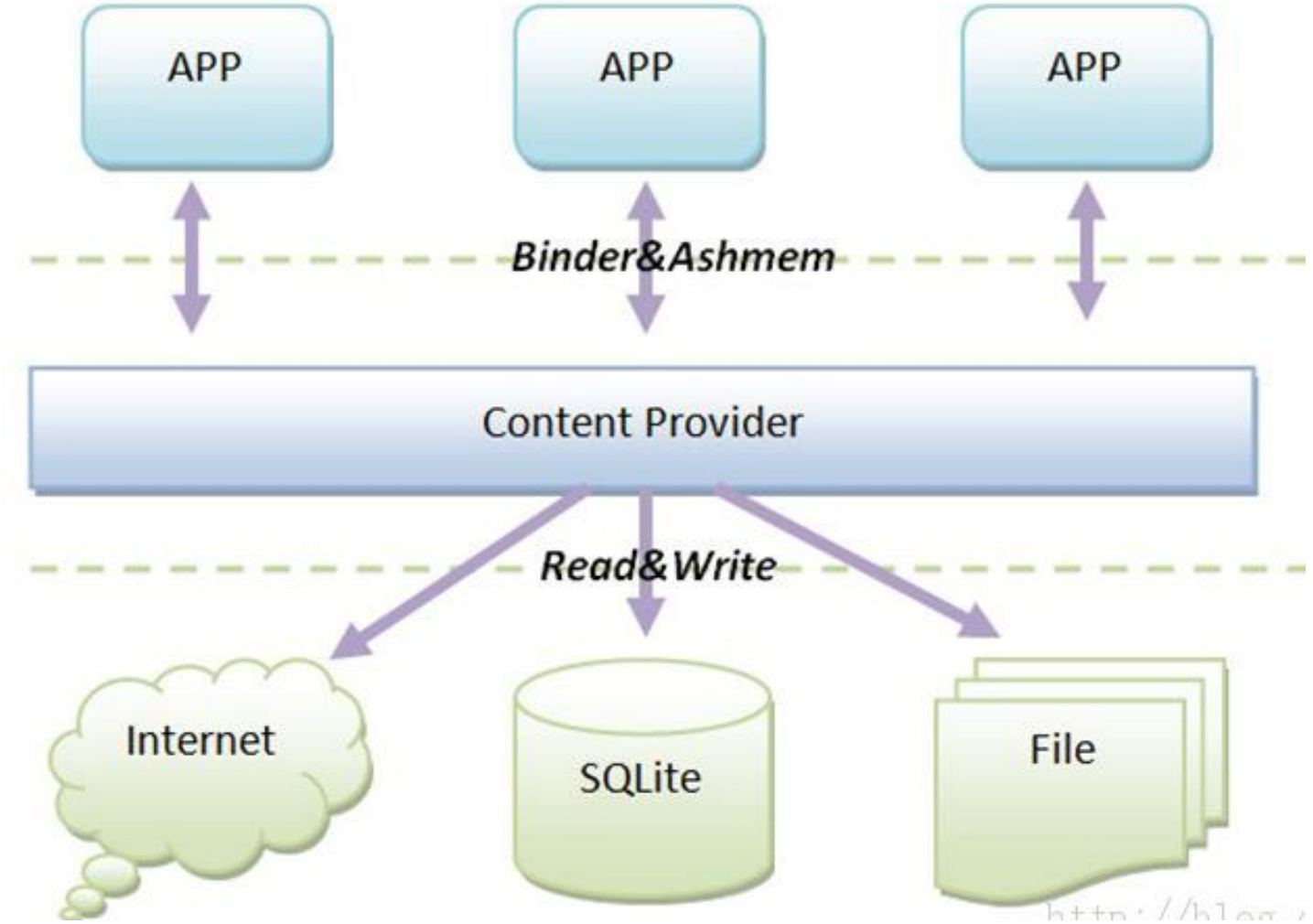
- 8.3.4 数据操作
  - 查询全部数据的代码

```
1    public People[] getAllData() {
2        Cursor results = db.query(DB_TABLE, new String[] { KEY_ID,
     KEY_NAME, KEY_AGE, KEY_HEIGHT}, null, null, null, null, null);
3        return ConvertToPeople(results);
4    }
```

8.4 数据分享

- 8.4.1 ContentProvider
  - Android应用程序运行在不同的进程空间中，因此不同应用程序的数据是不能够直接访问的
  - ContentProvider（ContentProvider）是应用程序之间共享数据的一种接口机制，可以根据需要指定需要共享的数据，而其它应用程序则可在不知道数据来源、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。
  - Android已经为常见的一些数据提供了默认的ContentProvider，例如通讯录、音视频文件和图像文件等

8.4 数据分享


APP
APP
APP
Binder&Ashmem
Content Provider
Read&Write
Internet
SQLite
File

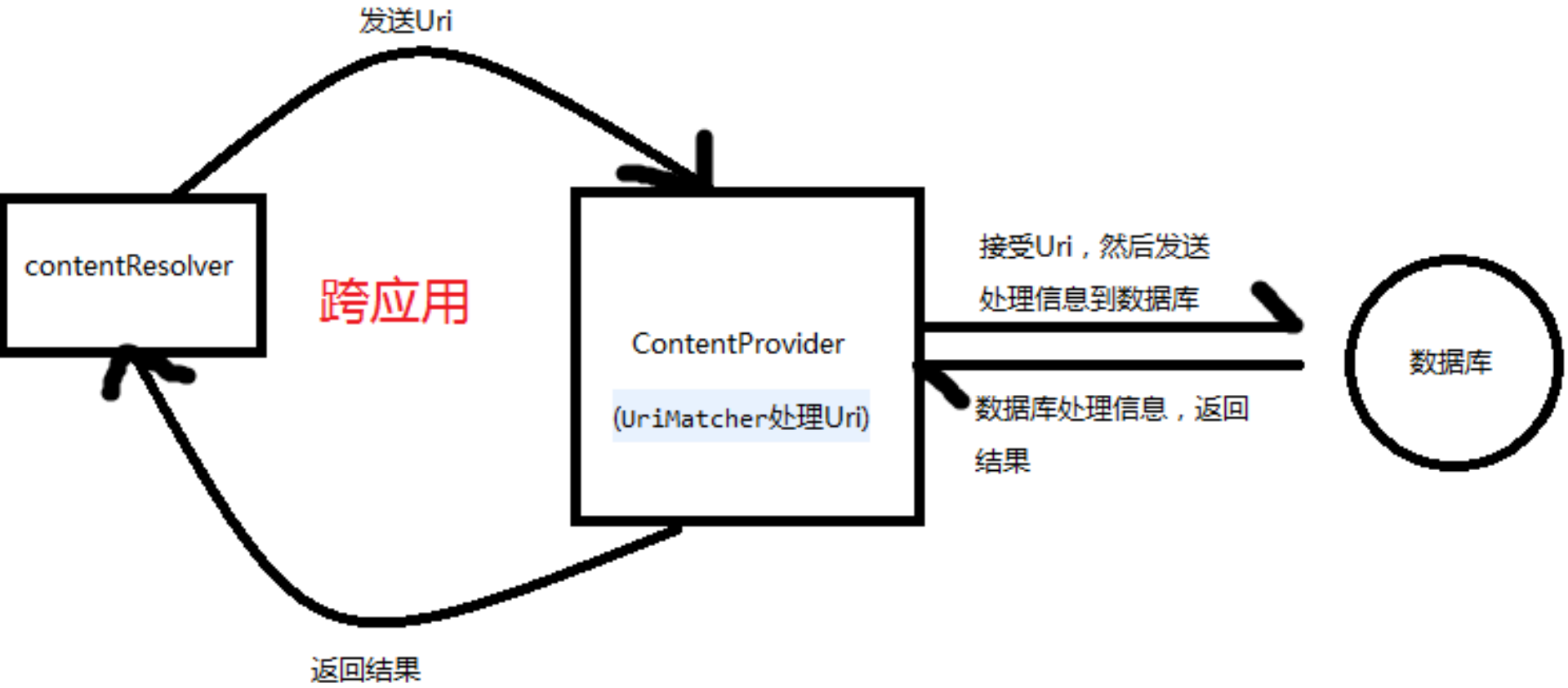
8.4 数据分享

- 8.4.1 ContentProvider
  - 无论数据的来源是什么，ContentProvider都会认为是一种表，然后把数据组织成表格；
  - ContentProvider提供的方法
    - 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

8.4 数据分享

- 8.4.1 ContentProvider
  - 每个ContentProvider都有一个公共的URI，这个URI用于表示这个ContentProvider所提供的数据。Android所提供的ContentProvider都存放在android.provider包当中；
  - 调用者需要使用ContentResolver对象，通过URI间接调用ContentProvider

8.4 数据分享


发送Uri
contentResolver
跨应用
ContentProvider
(UriMatcher处理Uri)
接受Uri，然后发送
处理信息到数据库
数据库处理信息，返回
结果
数据库
返回结果

8.4 数据分享

- 8.4.1 ContentProvider
  - 在发起一个请求的过程中，Android系统根据URI确定处理这个查询的ContentProvider，然后初始化ContentProvider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ContentProvider对象，但却可以同时与多个ContentResolver进行交互

8.4 数据分享

8.4.1 ContentProvider

- ContentProvider完全屏蔽了数据提供组件的数据存储方法
- 在程序开发人员看来，数据提供者通过ContentProvider提供了一组标准的数据操作接口，但却无需知道数据提供者的内部数据的存储方法
- 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，这些数据的存储方法和存储设备对数据使用者都是不可见的
- 同时，也正是这种屏蔽模式，很大程度上简化的ContentProvider的使用方法，使用者只要调用ContentProvider提供的接口函数，即可完成所有的数据操作，而数据存储方法则是ContentProvider设计者需要考虑的问题

8.4 数据分享

- 8.4.1 ContentProvider
  - ContentProvider的数据集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
  - 如下表所示。每条记录都包含一个长型的字段_ID，用来唯一标识每条记录
  - ContentProvider可以提供多个数据集，调用者使用URI对不同数据集的数据进行操作
  - ContentProvider数据集

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

8.4 数据分享

8.4.1 ContentProvider

- 如果ContentProvider仅提供一个数据集，数据路径则是可以省略的
- 但如果ContentProvider提供多个数据集，数据路径则必须指明具体是哪一个数据集
  - 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
- 如果请求的数据并不只限于一条数据，则<id>是可以省略，例如
  - 请求整个people数据集的URI应写为

```
content://com.example.peopleprovider/people
```

  - 请求people数据集中第3条数据的URI则应写为

```
content://com.example.peopleprovider/people/3
```

8.4 数据分享

- 8.4.2 创建ContentProvider
  - 程序开发人员通过继承ContentProvider类可以创建一个新的ContentProvider，过程可以分为以下几步
    - 声明CONTENT_URI，实现UriMatcher
    - 继承ContentProvider
      - Public class MyContentProvider extends ContentProvider
    - 实现ContentProvider的所有方法(query、insert、update、delete、getType、onCreate)
    - 在AndroidMinifest.xml中进行声明，注册ContentProvider

8.4 数据分享

- 8.4.3 使用ContentProvider
  - 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

```
1    String KEY_ID = "_id";
2    String KEY_NAME = "name";
3    String KEY_AGE = "age";
4    String KEY_HEIGHT = "height";
5
6    Uri uri = Uri.parse(CONTENT_URI_STRING + "/" + "2";
7    Cursor cursor = resolver.query(uri,
8        new String[] {KEY_ID, KEY_NAME, KEY_AGE, KEY_HEIGHT}, null,
     null, null);
```

8.4 数据分享

- 8.4.3 使用ContentProvider
  - 查询操作
    - 从上面的代码不难看出，在URI中定义了需要查询数据的ID后，在query()函数中则没有必要再加入其他的查询条件
    - 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

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

    - uri定义了查询的数据集，projection定义了从应返回数据的哪些属性，selection定义了返回数据的查询条件

8.4 数据分享

- 8.4.3 使用ContentProvider
  - 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
    - 下面的代码说明了如何使用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, 1.81f);
5
6    Uri newUri = resolver.insert(CONTENT_URI, values);
```

8.4 数据分享

- 8.4.3 使用ContentProvider
  - 添加操作
    - 下面的代码说明了如何使用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);
```

8.4 数据分享

- 8.4.3 使用ContentProvider
  - 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
    - 下面代码说明了如何删除ID为2的数据

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

    - 也可以在selection将删除条件定义为ID大于4的数据

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

8.4 数据分享

- 8.4.3 使用ContentProvider
  - 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
    - 下面代码说明了如何更新ID为7的数据

```
1    ContentValues values = new ContentValues();
7    values.put(KEY_NAME, "Tom");
8    values.put(KEY_AGE, 21);
9    values.put(KEY_HEIGHT, 1.81f);
2
3    Uri uri = Uri.parse(CONTENT_URI_STRING + "/" + "7");
4    int result = resolver.update(uri, values, null, null);
```