# 内容提供者

* 四大组件之一
* 应用的数据库是不允许其他应用访问的
* 内容提供者的作用就是让别的应用访问到你的数据库
* 通常，我们都通过内容提供者访问系统的各种数据库（短信，联系人…）

# 自定义内容提供者：

* 首先，要有内容提供（自己创建一个数据库，作为内容提供出去的）
* 定义一个类，继承android.content.ContentProvider
* 重写6个方法
  + boolean onCreate()：内容提供者创建时调用
  + String getType(Uri uri)：基本没什么用，使用来返回。调用者所查询到的数据的mime类型的。这个方法是给其他应用调用的
    - Uri：内容提供者的主机名，也就是地址，调用者通过这个uri，指定要调用那个内容提供者
    - 但是数据库中的数据是没有mime类型的。不过我们可以自定义一个mime类型返回回去。
  + Uri insert(Uri uri, ContentValues values)：增。用来往数据库里插入数据，这个方法是给其他应用调用的。
    - uri：内容提供者的主机名，也就是地址，调用者通过这个uri，指定要调用那个内容提供者。由其他应用传入
    - values：由其他应用传入，用于封装要插入的数据
    - 返回一个uri，可以把插入后返回的主键，拼接在uri后面
  + int delete(Uri uri, String selection, String[] selectionArgs)：删。用来删除数据库的数据，这个方法是给其他应用调用的
    - uri：内容提供者的主机名，也就是地址，调用者通过这个uri，指定要调用那个内容提供者。由其他应用传入
    - selection：删除的where条件，由其他应用传入
    - selectionArgs：填充where条件站位符的，由其他应用传入
    - 返回一个int，就是删除后的影响行数
  + int update(Uri uri, ContentValues values, String selection, String[] selectionArgs)：改。用来修改数据库的数据，这个方法是给其他应用调用的
    - uri：内容提供者的主机名，也就是地址，调用者通过这个uri，指定要调用那个内容提供者。由其他应用传入
    - values：修改后的新数据，封装在ContentValues，由其他应用传入
    - selection：修改的where条件，由其他应用传入
    - selectionArgs：填充where条件占位符的，由其他应用传入
    - 返回一个int，就是修改后的影响条数
  + Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)：查。用来查询数据库中的数据。这个方法是给其他应用调用的
    - uri：uri：内容提供者的主机名，也就是地址，调用者通过这个uri，指定要调用那个内容提供者。由其他应用传入
    - projection：查询的字段，如果是null就查所有字段，由其他应用传入
    - selection：where条件，由其他应用传入
    - selectionArgs：填充where条件占位符的，由其他应用传入
    - sortOrder：排序条件，由其他应用传入
    - 返回结果集

在清单文件中配置

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"utf-8"*?>  <manifest xmlns:android=*"http://schemas.android.com/apk/res/android"*  package=*"com.example.mycontentprovider"*  android:versionCode=*"1"*  android:versionName=*"1.0"* >  <instrumentation android:name=*"android.test.InstrumentationTestRunner"*  android:targetPackage=*"com.example.mycontentprovider"*></instrumentation>  <uses-sdk  android:minSdkVersion=*"8"*  android:targetSdkVersion=*"17"* />  <application  android:allowBackup=*"true"*  android:icon=*"@drawable/ic\_launcher"*  android:label=*"@string/app\_name"*  android:theme=*"@style/AppTheme"* >    <provider  android:name=*"com.example.mycontentprovider.MyContentProvider"*  android:authorities=*"com.example.mycontentprovider"*  android:exported=*"true"*></provider>  </application>  </manifest> |

# 访问自定义内容提供者

* 拿到ContentResolver，内容解析者。ContextWrapper# getContentResolver();
* 往自定义内容提供者所在项目的数据库插入数据
  + 通过ContentResolver# insert(Uri url, ContentValues values)调用内容提供者的insert(Uri uri, ContentValues values)
    - uri：内容提供者的主机名，内容提供者在清单文件中配置的时候有个authorities属性，就是这个属性值，要有”content://”前缀。通过uri找到内容提供者
    - values：要插入的数据，封装到这里
* 把自定义内容提供者所在项目的数据库中的数据删除
  + 通过ContentResolver# delete(url, where, selectionArgs)调用内容提供者的delete(Uri uri, String selection, String[] selectionArgs)
    - uri：内容提供者的主机名，内容提供者在清单文件中配置的时候有个authorities属性，就是这个属性值，要有”content://”前缀。通过uri找到内容提供者
    - where：删除的条件
    - selectionArgs：填充条件占位符的
    - 也会返回影响行数
* 把自定义内容提供者所在项目的数据库中的数据修改
  + 通过ContentResolver#update(uri, values, where, selectionArgs)调用内容提供者的update(Uri uri, ContentValues values, String selection, String[] selectionArgs)
    - uri：内容提供者的主机名，内容提供者在清单文件中配置的时候有个authorities属性，就是这个属性值，要有”content://”前缀。通过uri找到内容提供者
    - where：修改的条件
    - selectionArgs：填充条件占位符的
    - 也会返回影响行数
* 查询出自定义内容提供者所在项目的数据库中的数据
  + 通过ContentResolver# query(uri, projection, selection, selectionArgs, sortOrder);调用内容提供者的query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)
    - uri：内容提供者的主机名，内容提供者在清单文件中配置的时候有个authorities属性，就是这个属性值，要有”content://”前缀。通过uri找到内容提供者
    - projection：查询的字段，null就查询全部
    - selection：查询条件
    - selectionArgs：填充查询条件的
    - sortOrder：排序
    - 会返回结果集

# 使用uri携带路径（要查询的表名）：

* Uri不仅可以携带调用者指定的内容提供者的主机名，还可以携带要查询数据库的那张表
  + content://主机名/表名

# 使用uri匹配器判断携带的路径（表名）：

* 在内容提供者中，创建uri匹配器：android.content.UriMatcher
  + private static UriMatcher um = new UriMatcher(UriMatcher.NO\_MATCH);
    - 如果匹配规则中，没有一条是跟调用者传入的uri匹配，那么匹配码返回-1
* 在静态代码块中，添加匹配规则

|  |
| --- |
| //创建uri匹配器。如果用户传入的uri没有一条跟我们定义的匹配规则匹配，匹配码就返回-1  **private** **static** UriMatcher *um* = **new** UriMatcher(UriMatcher.*NO\_MATCH*);    //添加匹配规则  **static** {  /\*  \* authority: 内容提供者的主机名，与我们定义的内容提供者的主机名一致  \* path：路径（表名）  \* code：匹配码，如果用户传入的uri与这一条匹配，就返回这个匹配码  \*/  // 用户传入的uri：content://com.example.mycontentprovider2/person，与这条匹配  *um*.addURI("com.example.mycontentprovider2", "person", 1);  // 用户传入的uri：content://com.example.mycontentprovider2/person2，与这条匹配  *um*.addURI("com.example.mycontentprovider2", "person2", 2);  } |

* 在insert方法中，对用户传入的uri，用匹配器进行匹配

|  |
| --- |
| @Override  **public** Uri insert(Uri uri, ContentValues values) {    //把用户传入的uri与匹配器进行匹配，会返回一个匹配码  **if**(*um*.match(uri) == 1) {  db.insert("person", **null**, values);  } **else** **if**(*um*.match(uri) == 2) {  db.insert("person2", **null**, values);  } **else** {  **throw** **new** IllegalAccessError("没有匹配的uri");  }    //可以把插入后返回的主键，拼接在uri后面。这里就不拼接  **return** uri;  } |

# Uri通过路径携带数据

* 路径除了可以携带表名，还可以携带其他数据
* 添加匹配规则的时候，进行规定。”#”:可以携带任何数字，“\*”：可以携带任何文本
* 携带的数据可以用来做很多事情，例如：作为查询条件..
* 一般来书：数字携带过来是做id使用的，但是没有任何规定一定要做id。做其他什么的都可以

|  |
| --- |
| //创建uri匹配器。如果用户传入的uri没有一条跟我们定义的匹配规则匹配，匹配码就返回-1  **private** **static** UriMatcher *um* = **new** UriMatcher(UriMatcher.*NO\_MATCH*);    //添加匹配规则  **static** {  /\*  \* authority: 内容提供者的主机名，与我们定义的内容提供者的主机名一致  \* path：路径（表名）  \* code：匹配码，如果用户传入的uri与这一条匹配，就返回这个匹配码  \*/  // 用户传入的uri：content://com.example.mycontentprovider2/person，与这条匹配  *um*.addURI("com.example.mycontentprovider2", "person", 1);  // 用户传入的uri：content://com.example.mycontentprovider2/person2，与这条匹配  *um*.addURI("com.example.mycontentprovider2", "person2", 2);    //路径path除了可以携带表名，也可以携带其他数据（“\*”: 任何文本，“#:任何数字”）  // 用户传入的uri：content://com.example.mycontentprovider2/person2/abd，与这条匹配  *um*.addURI("com.example.mycontentprovider2", "person/\*", 3);  // 用户传入的uri：content://com.example.mycontentprovider2/person2/123，与这条匹配  *um*.addURI("com.example.mycontentprovider2", "person/#", 4);  } |

* 在insert方法中，对用户传入的uri，用匹配器进行匹配

|  |
| --- |
| @Override  **public** Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder) {  Cursor cursor = **null**;  **if**(*um*.match(uri) == 1) {  cursor = db.query("person", projection, selection, selectionArgs, **null**, **null**, sortOrder, **null**);  } **else** **if**(*um*.match(uri) == 2) {  cursor = db.query("person2", projection, selection, selectionArgs, **null**, **null**, sortOrder, **null**);  } **else** **if**(*um*.match(uri) == 4) { //如果携带的是数字，就作为id查询  //把uri末尾携带的数字取出来  **long** id = ContentUris.*parseId*(uri);  System.*out*.println(id);  cursor = db.query("person", projection, "\_id = ?", **new** String[] {id+""}, **null**, **null**, sortOrder, **null**);  } **else** {  **throw** **new** IllegalAccessError("没有匹配的uri");  }  //返回结果集  **return** cursor;  } |

* 用户调用内容提供者进行查询

|  |
| --- |
| **public** **void** query(View v) {  //通过内容提供者查询出MyContentProvider项目的数据库中的数据  //拿到ContentResolver  ContentResolver cr = getContentResolver();    //要查询的字段，null查全部  String[] projection = **null**;  //where条件  String selection = **null**;  //填充where条件的  String[] selectionArgs = **null**;  //排序  String sortOrder = **null**;    //查询  //它会通过uri，调用指定的内容提供者的query方法  Cursor cursor = cr.query(Uri.*parse*("content://com.example.mycontentprovider3/person/1"), projection, selection, selectionArgs, sortOrder);    //对结果集进行处理得到想要的数据  **while**(cursor.moveToNext()) {  String name = cursor.getString(cursor.getColumnIndex("name"));  System.*out*.println(name);  }    } |

## getType的作用：

* 告诉调用者，所查询到的数据的mime类型
* 这个方法是给其他应用调用的

|  |
| --- |
| @Override  **public** String getType(Uri uri) {  **if**(*um*.match(uri) == 1) {  //查询的是多条数据,要用：vnd.android.cursor.item开头  **return** "vnd.android.cursor.item/person"; // 告诉调用者，查询到的是person表中的多条数据  } **else** **if**(*um*.match(uri) == 4) {  **return** "vnd.android.cursor.dir/person"; //告诉调用者，查询到的是person表中的单条数据  }  **return** **null**;  } |

# 获取系统短信

* 系统短信的数据库，自然是在系统短信应用中：data/data/com.android.providers.telephony
* 系统短信是放在sms表里面的，表里的其中四个字段
  + body：短信内容
  + date：短信时间
  + address：对方号码
  + type：发送还是接收， 1是收到，2是发送
* 系统短信的内容提供者的地址：content://sms
  + 匹配规则：
    - content://sms：不加路径，表示操作所有短信
    - content://sms/inbox: 收件箱短信
    - content://sms/sent: 已发送短信

….

* 读取系统短信，和写入短信，是需要权限的
  + 读取权限：android.permission.READ\_SMS
  + 写入权限：android.permission.WRITE\_SMS
* 内容提供者的地址和匹配规则，可以去源码里找

## 短信备份：

* 从短信数据库获取到所有短信
* 封装到一个javaben集合中
* 在把集合里的数据，用XmlSerializer序列化器生成到xml文件中

# 插入短信：

# 获取系统练习人

* 联系人的内容提供者，在：con.android.providers.contacts
* 联系人的数据库也在这
* 内容提供者的地址：
  + 在源码中是这样定义的：contacts;com.android.contacts
  + 分号：左边的以前定义的，右边是新的
  + 左边的在高版本不能用，右边的高低版本都能用。所以用右边的
  + “content://com.android.contacts”
  + 匹配规则：
    - content://com.android.contacts/raw\_contacts: 操作raw\_contacts表
    - content://com.android.contacts/data: 操作data表
* 读取联系人和写入联系人也需要权限：
  + 读取：android.permission.READ\_CONTACTS
  + 写入：android.permission.WRITE\_CONTACTS
* 联系人数据库中我们要看三个表：
  + raw\_contacts：存放联系人id，有三行数据。因为有三个联系人
    - contact\_id：联系人id
  + data：存放联系人详细信息。有多行数据，每行存放一个联系人的一个信息（例如：邮箱存放一行，电话存放一行）
    - raw\_contact\_id: 联系人id，通过id，知道每行数据是属于哪个联系人的
    - mimetype\_id: 信息类型，去mimetypes表中，查看对应的类型。根据这个类型。知道这条信息是什么（名字，还是邮箱）
    - data1: 信息
  + mimetypes：存放信息的类型
    - id: 类型的id
    - mimetype:类型的字符串
* 获取过程：
  + 先去raw\_contacts表，查询联系人id
  + 在通过id，去date表查询对应id的联系人信息
    - 在这里，如果要查询mimetype\_id字段，是查询不到的（会报bug，没有这个字段）
    - 因为：在他的内容提供者中，有个人性化的设计
    - 在查询的时候，是进行多表查询。把mimetypes表中的mimetype字段直接查询出来的
    - 所以，查询的时候，传mimetype\_id，但是在mimetypes表中根本没有这个字段，所以会包bug
    - 解决: 查询的字段改成mimetype
  + 在通过查询出来的mimetype，进行判断，是哪个类型。就存到Javabean的哪个变量中去

# 插入联系人：

* 要操作两张表：
  + raw\_contacts
    - 插入联系人id
    - 要插入联系人id，就要先遍历出有多少个联系人，然后id在+1
    - 这里，要遍历的字段就不是联系人id了，而是主键
    - 因为：如果一个联系人被删掉。它会把联系人id置为null，但是在data表中的数据还是存在的
    - 所以：如果用联系人id字段进行计算，那么，如果一个联系人id为null，就不会被算进去。然后你又插入了一个和这个为null的联系人id一样的联系联系人id。那么，在data表中的数据，就会冲突
    - Google这样做是为了方便云同步。
  + Data：
    - 因为每一个联系人的每一条信息，在数据库中都占一行
    - 所以，如果要插入三个信息（名字， 邮箱， 手机），就要插入三行数据
    - 每一行都要操作是三个字段，在查询的时候，mimetype\_id字段是不存在的，所以用mimetype字段。插入的时候也是一样。这样他会直接操作mimetypes表中的mimetype字段

# 内容观察者

* 当数据库数据改变时，内容提供者会发出通知，在内容提供者的uri上注册一个内容观察者，就可以收到数据改变的通知

# 内容提供者者中发送数据改变通知

* 系统的内容提供者，已经写好了发送通知的代码
* 自定义的内容提供者，需要我们自己发送通知
* 就一行代码，在内容提供者中，insert方法一执行，就说明数据库内容改变。这时候就发送通知

|  |
| --- |
| ContentResolver cr = getContext().getContentResolver()  /\*  \* 发送通知  \* uri: 要发送到哪个uri中，如果观察者注册的uri与这个uri匹配，那么观察者就会收到通知  \* observer：可以指定发送到哪个观察者中  \*/  cr.notifyChange(uri, **null**); |