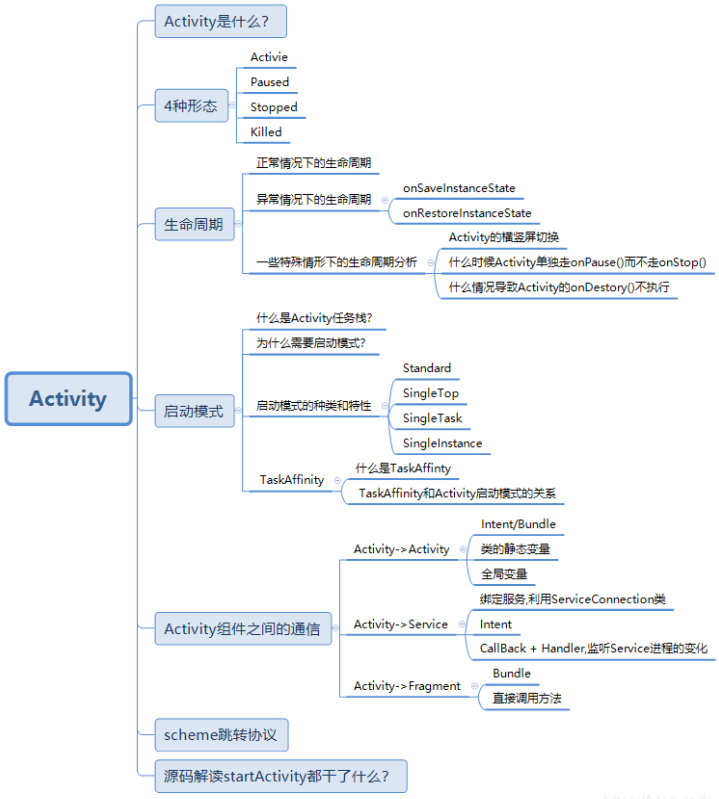
Android知识整理

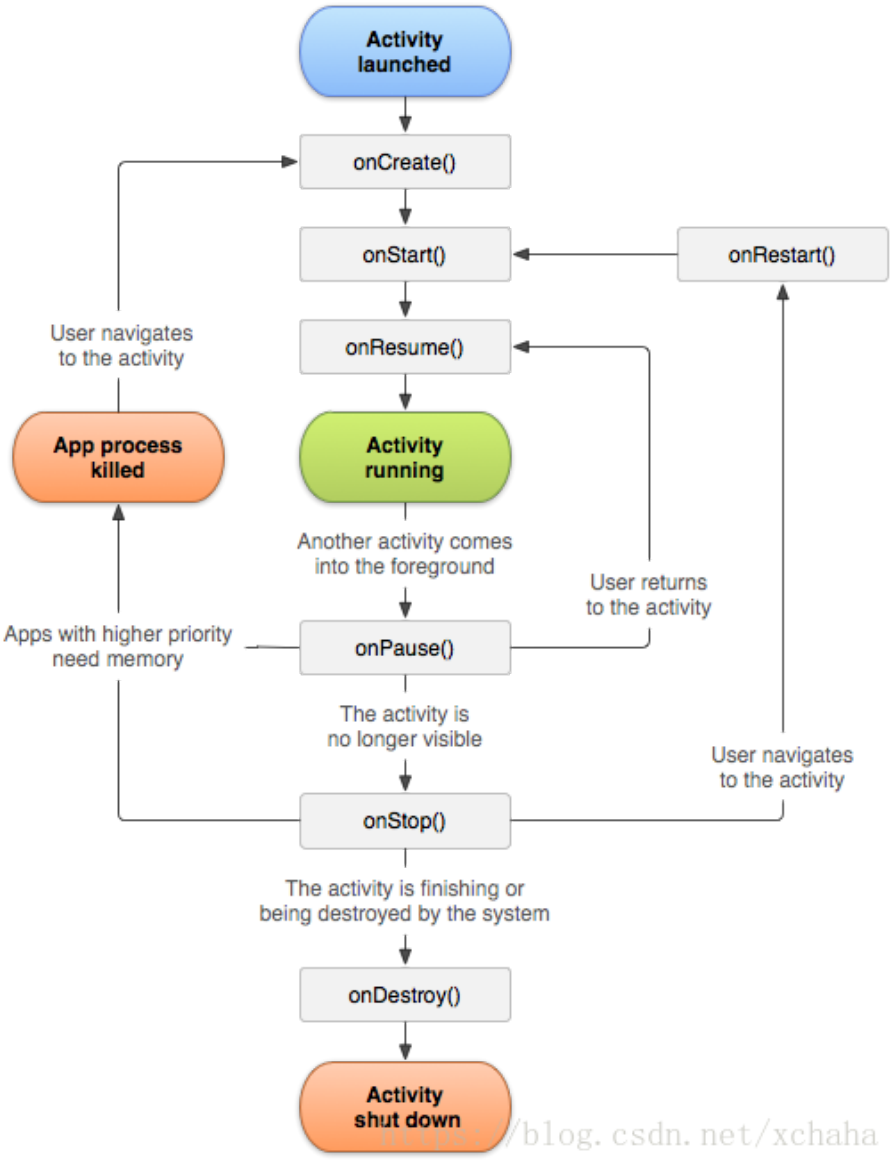
Android 集成Git、Android Studio提交代码

四大组件

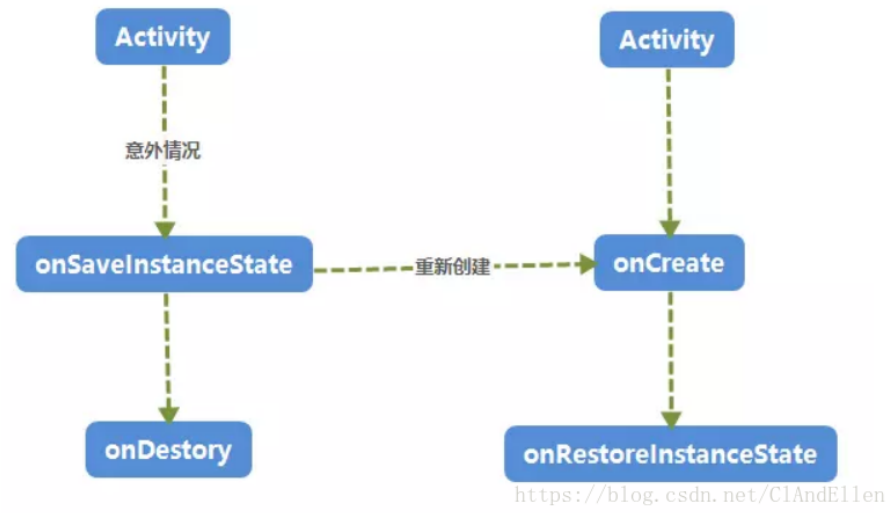
Activity



正常情况下的生命周期：



异常情况下的生命周期(两种)：



1. 情况1：资源相关系统配置发生改变导致Activty被杀死重新创建：手机横竖屏切换、语言切换等等。（可以通过设置"android:configChanges"或相关属性达到不受到情况1的影响）

2. 情况2：资源内存不足导致低优先级的Activity被杀死；（内存不足）

什么时候Activity不执行onDestory()

栈里面的第一个没有销毁的activity会执行ondestroy方法，其他的不会执行。

> 比如说：从mainactivity跳转到activity-A（或者继续从activity-A再跳转到activity-B），这时候，从后台强杀，只会执行mainactivity的onDestroy方法，activity-A（以及activity-B）的onDestroy方法都不会执行；

进程优先级

> 前台>可见>服务>后台>空

前台：与当前用户正在交互的Activity所在的进程。

可见：Activity可见但是没有在前台所在的进程。

服务：Activity在后台开启了Service服务所在的进程。

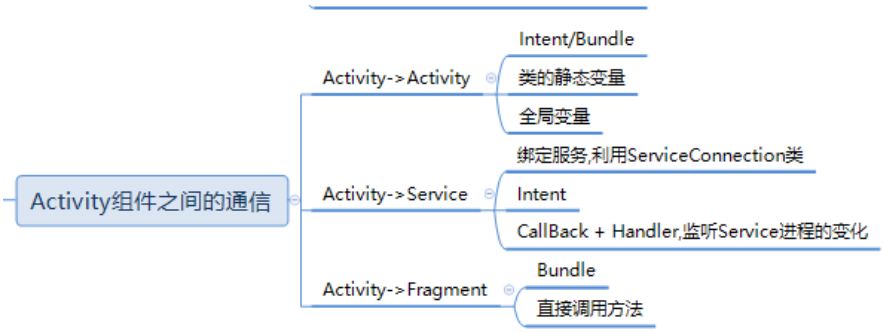
后台：Activity完全处于后台所在的进程。

Activity的启动模式有哪些?

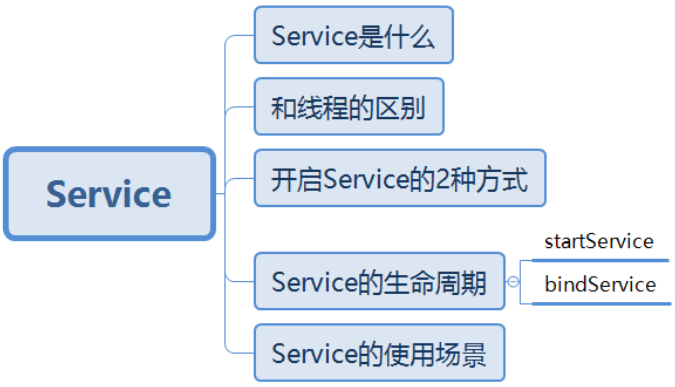
> Activity的启动模式有4种，分别是：standard,singleTop,singleTask和singleInstance。

1. 系统默认的启动模式:Standard,每次启动一个Activity都会重新创建一个新的实例，不管这个实例是否存在。
2. 栈顶复用模式：SingleTop:在这种模式下，如果新的Activity已经位于任务栈的栈顶，那么此Activity不会被重新创建，同时它的onNewIntent方法被回调，通过此方法的参数我们可以取出当前请求的信息。
3. 栈内复用模式：SingleTask:这是一种单例实例模式，在这种模式下，只要Activity在一个栈中存在，那么多次启动此Activity都不会重新创建实例，和singleTop一样，系统也会回调其onNewIntent。
4. 单实例模式：SingleInstance:这是一种加强的singleTask模式，它除了具有singleTask模式所有的特性外，还加强了一点，那就是具有此种模式的Activity只能单独位于一个任务栈中，换句话说，比如Activity A是singleInstance模式，当A启动后，系统会为它创建一个新的任务栈，然后A独自在这个新的任务栈中，由于栈内复用的特性，后续的请求均不会创建新的Activity,除非这个独特的任务栈被系统销毁了。

进程之间的通信

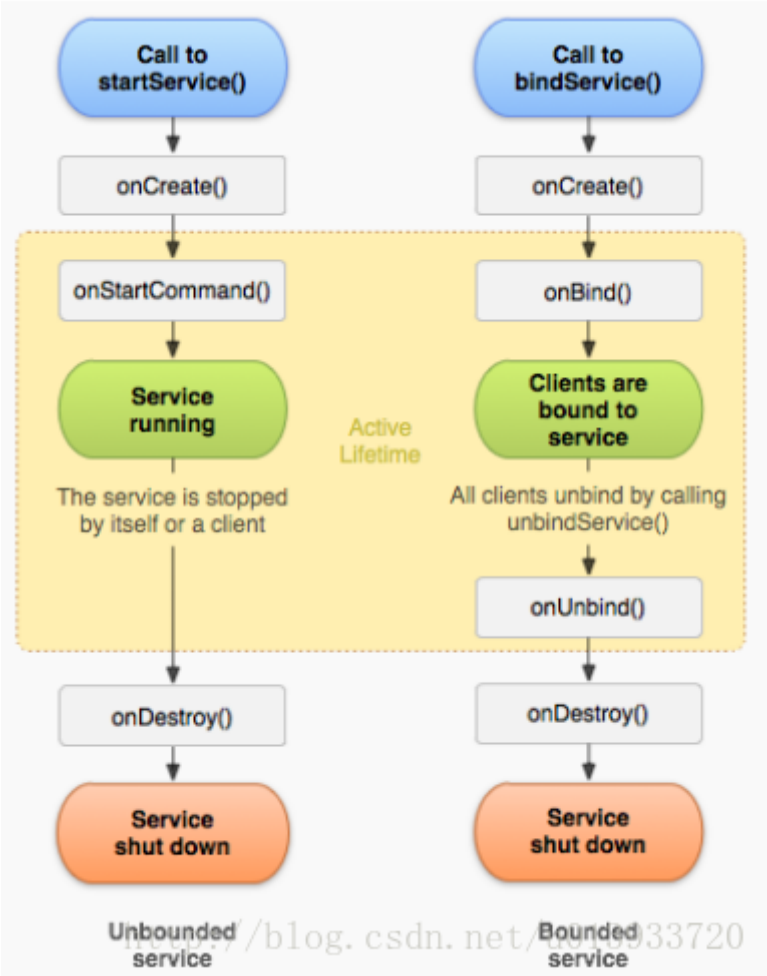


Service



https://blog.csdn.net/ClAndEllen

生命周期：



a. 被启动的服务的生命周期：如果一个Service被某个Activity 调用Context.startService 方法启动，那么不管是否有Activity使用bindService绑定或unbindService解除绑定到该Service，该Service都在后台运行。如果一个Service被startService 方法多次启动，那么onCreate方法只会调用一次，onStart将会被调用多次（对应调用startService的次数），并且系统只会创建Service的一个实例（因此你应该知道只需要一次stopService调用）。该Service将会一直在后台运行，而不管对应程序的Activity是否在运行，直到被调用stopService，或自身的stopSelf方法。当然如果系统资源不足，android系统也可能结束服务。

b. 被绑定的服务的生命周期：如果一个Service被某个Activity 调用 Context.bindService 方法绑定启动，不管调用 bindService 调用几次，onCreate 方法都只会调用一次，同时onStart方法始终不会被调用。当连接建立之后，Service将会一直运行，除非调用Context.unbindService 断开连接或者之前调用 bindService 的 Context 不存在了（如Activity被finish的时候），系统将会自动停止Service，对应onDestroy将被调用。

c. 被启动又被绑定的服务的生命周期：如果一个Service又被启动又被绑定，则该Service将会一直在后台运行。并且不管如何调用，onCreate始终只会调用一次，对应startService调用多少次，Service的onStart便会调用多少次。调用unbindService将不会停止Service，而必须调用 stopService 或 Service的 stopSelf 来停止服务。

d. 当服务被停止时清除服务：当一个Service被终止（1、调用stopService；2、调用stopSelf；3、不再有绑定的连接（没有被启动））时，onDestroy方法将会被调用，在这里你应当做一些清除工作，如停止在Service中创建并运行的线程。
特别注意：

1、你应当知道在调用 bindService 绑定到Service的时候，你就应当保证在某处调用 unbindService 解除绑定（尽管 Activity 被 finish 的时候绑定会自动解除，并且Service会自动停止）；

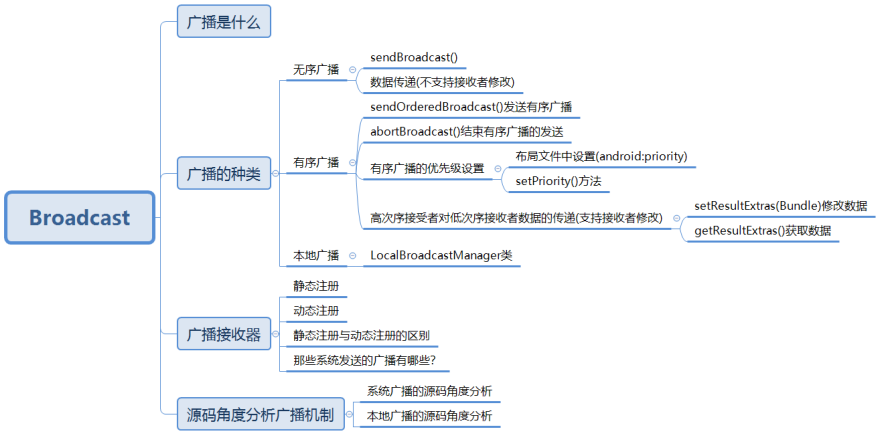
2、你应当注意 使用 startService 启动服务之后，一定要使用 stopService停止服务，不管你是否使用bindService；

3、同时使用 startService 与 bindService 要注意到，Service 的终止，需要unbindService与stopService同时调用，才能终止 Service，不管 startService 与 bindService 的调用顺序，如果先调用 unbindService 此时服务不会自动终止，再调用 stopService 之后服务才会停止，如果先调用 stopService 此时服务也不会终止，而再调用 unbindService 或者 之前调用 bindService 的 Context 不存在了（如Activity 被 finish 的时候）之后服务才会自动停止；

4、当在旋转手机屏幕的时候，当手机屏幕在“横”“竖”变换时，此时如果你的 Activity 如果会自动旋转的话，旋转其实是 Activity 的重新创建，因此旋转之前的使用 bindService 建立的连接便会断开（Context 不存在了），对应服务的生命周期与上述相同。

5、在 sdk 2.0 及其以后的版本中，对应的 onStart 已经被否决变为了 onStartCommand，不过之前的 onStart 任然有效。这意味着，如果你开发的应用程序用的 sdk 为 2.0 及其以后的版本，那么你应当使用 onStartCommand 而不是 onStart。

Broadcast



https://blog.csdn.net/ClAndEllen

广播的使用场景

a.同一app内有多个进程的不同组件之间的消息通信。
b.不同app之间的组件之间消息的通信。

广播种类（3种）

1. 无序广播
   context.sendBroadcast(Intent)方法发送的广播，不可被拦截，当然发送的数据，接收者是不能进行修改的。
2. 有序广播
   context.sendOrderBroadcast(Intent)方法发送的广播，可被拦截，而且接收者是可以修改其中要发送的数据，修改和添加都是可以的，这就意味着优先接收者对数据修改之后，下一个接收者接受的数据是上一个接收者已经修改了的，这必须明白。
3. 本地广播
   localBroadcastManager.sendBroadcast(Intent)，只在app内传播。

广播接收器

广播接收器是专门用来接收广播信息的，它可分为静态注册和动态注册：

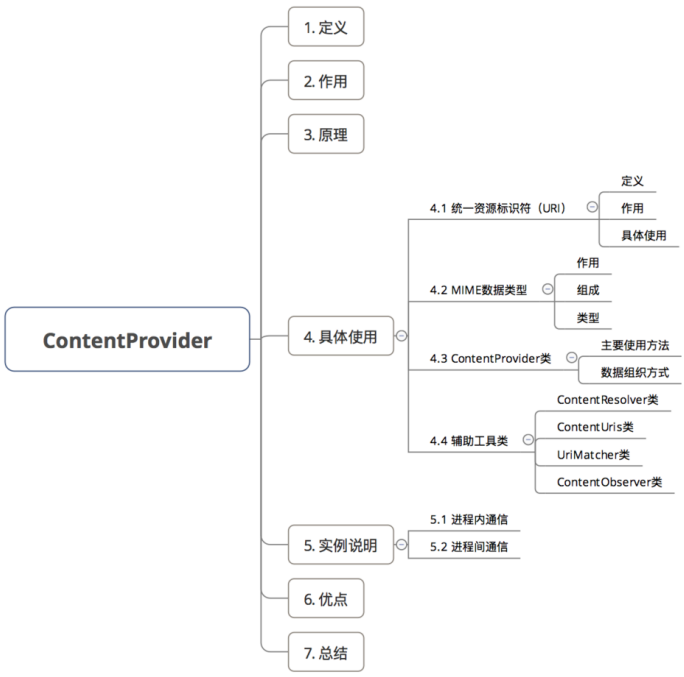
1. 静态注册
   - 首先你要创建一个广播接收器类;
   - 在AndroidManifest.xml文件中注册;
2. 动态注册
   - 新建一个类，让它继承自BroadcastReceiver,并重写父类的onReceive()方法;
   - 代码中创建intentFilter:

```
intentFilter = new IntentFilter();
intentFilter.addAction("android.net.conn.CONNECTIVITY_CHANGE");
networkChangeReceiver = new NetWorkChangeReceiver();
registerReceiver(networkChangeReceiver, intentFilter);//注册广播接收器
```

优点：动态注册的广播接收器可以自由地控制注册与注销，在灵活性方面有很大优势；
缺点：必须要在程序启动之后才能接收到广播，因为注册的逻辑是写在onCreate()方法中的。那么有没有广播能在程序未启动的情况下就能接收到广播呢？静态注册的广播接收器就可以做到。

ContentProvider

参考链接2： https://blog.csdn.net/carson_ho/article/details/76101093

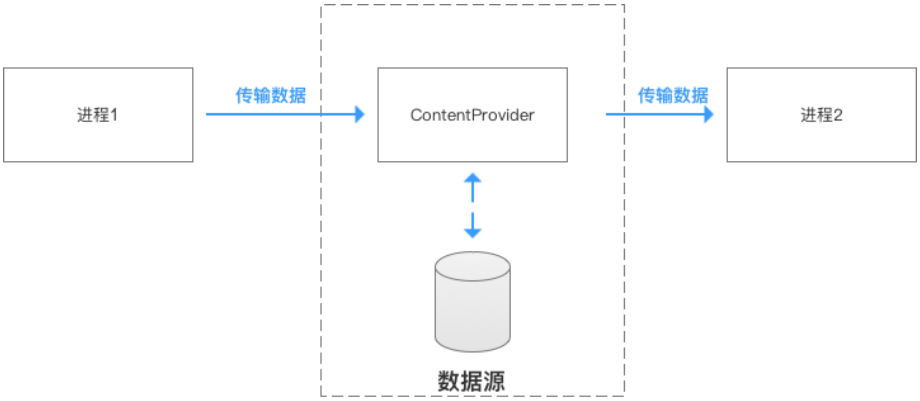


什么是ContentProvider：

```
是Android的四大组件之一；
主要用于不同的应用程序之间实现数据共享功能；
```

作用

进程间 进行数据交互 & 共享，即跨进程通信



注：
1. ContentProvider = 中间者角色（搬运工），真正 存储&操作数据的数据源还是原来存储数据的方式（数据库、文件、xml或网络）
2. 数据源可以：数据库（如Sqlite）、文件、XML、网络等等

什么是ContentResolver：

是数据调用者，ContentProvider将数据发布出来，通过ContentResolver对象结合Uri进行调用,通过ContentResolver对象可以调用ContentProvider的增删改查;

什么是Uri：

Uri（通用资源标识符 Universal Resource Identifer），代表数据操作的地址，每一个ContentProvider发布数据时都会有唯一的地址。
比如：content：//（固定写法）+com.android.contacts(包名，可变)+/contacts（path路径）

MIME数据类型

- 作用：指定某个扩展名的文件用某种应用程序来打开。如指定.html文件采用text应用程序打开、指定.pdf文件采用flash应用程序打开；

ContentProvider类

组织数据方式

- ContentProvider主要以 表格的形式 组织数据,同时也支持文件数据，只是表格形式用得比较多;
- 每个表格中包含多张表，每张表包含行 & 列，分别对应记录 & 字段,同数据库;

主要方法

- 进程间共享数据的本质是：添加、删除、获取 & 修改（更新）数据；
- 所以ContentProvider的核心方法也主要是上述4个作用：

```
<-- 4个核心方法 -->
  public Uri insert(Uri uri, ContentValues values)
  // 外部进程向 ContentProvider 中添加数据

  public int delete(Uri uri, String selection, String[]
selectionArgs)
  // 外部进程 删除 ContentProvider 中的数据

  public int update(Uri uri, ContentValues values, String
selection, String[] selectionArgs)
  // 外部进程更新 ContentProvider 中的数据

  public Cursor query(Uri uri, String[] projection, String
selection, String[] selectionArgs,  String sortOrder)
  // 外部应用 获取 ContentProvider 中的数据

// 注：
  // 1. 上述4个方法由外部进程回调，并运行在ContentProvider进程的
Binder线程池中（不是主线程）
 // 2. 存在多线程并发访问，需要实现线程同步
   // a. 若ContentProvider的数据存储方式是使用SQLite & 一个，则
不需要，因为SQLite内部实现好了线程同步，若是多个SQLite则需要，因为
SQL对象之间无法进行线程同步
  // b. 若ContentProvider的数据存储方式是内存，则需要自己实现线程
同步

<-- 2个其他方法 -->
public boolean onCreate()
// ContentProvider创建后 或 打开系统后其它进程第一次访问该
ContentProvider时 由系统进行调用
// 注：运行在ContentProvider进程的主线程，故不能做耗时操作

public String getType(Uri uri)
// 得到数据类型，即返回当前 Url 所代表数据的MIME类型
```

- Android为常见的数据（如通讯录、日程表等）提供了内置了默认的ContentProvider
- 但也可根据需求自定义ContentProvider，但上述6个方法必须重写；
- ContentProvider类并不会直接与外部进程交互，而是通过 ContentResolver 类;

ContentResolver类

作用

统一管理不同 ContentProvider间的操作:
1. 即通过 URI 即可操作 不同的ContentProvider 中的数据;
2.外部进程通过 ContentResolver类 从而与ContentProvider类进行交互;

为什么要使用通过ContentResolver类从而与ContentProvider类进行交互，而不直接访问ContentProvider类？

一般来说，一款应用要使用多个ContentProvider，若需要了解每个ContentProvider的不同实现从而再完成数据交互，操作成本高 & 难度大，所以再ContentProvider类上加多了一个 ContentResolver类对所有的ContentProvider进行统一管理。

Android 提供了3个用于辅助ContentProvide的工具类：

```
ContentUris
UriMatcher
ContentObserver
```

ContentUris类

作用：操作 URI
具体使用
核心方法有两个：withAppendedId（） &parseId（）

```
// withAppendedId（）作用：向URI追加一个id
Uri uri = Uri.parse("content://cn.scu.myprovider/user")
Uri resultUri = ContentUris.withAppendedId(uri, 7);
// 最终生成后的Uri为：content://cn.scu.myprovider/user/7
// parseId（）作用：从URL中获取ID
Uri uri = Uri.parse("content://cn.scu.myprovider/user/7")
long personid = ContentUris.parseId(uri);
//获取的结果为:7
```

UriMatcher类

作用:

- 在ContentProvider 中注册URI
- 根据 URI 匹配 ContentProvider 中对应的数据表
  具体使用:

```
// 步骤1：初始化UriMatcher对象
    UriMatcher matcher = new
UriMatcher(UriMatcher.NO_MATCH);
    //常量UriMatcher.NO_MATCH  = 不匹配任何路径的返回码
    // 即初始化时不匹配任何东西

// 步骤2：在ContentProvider 中注册URI（addURI（））
    int URI_CODE_a = 1;
```

```
    int URI_CODE_b = 2;
    matcher.addURI("cn.scu.myprovider", "user1",
URI_CODE_a);
    matcher.addURI("cn.scu.myprovider", "user2",
URI_CODE_b);
    // 若URI资源路径 = content://cn.scu.myprovider/user1 ，
则返回注册码URI_CODE_a
    // 若URI资源路径 = content://cn.scu.myprovider/user2 ，
则返回注册码URI_CODE_b

// 步骤3：根据URI 匹配 URI_CODE，从而匹配ContentProvider中相应
的资源（match（））

@Override
    public String getType(Uri uri) {
      Uri uri = Uri.parse("
content://cn.scu.myprovider/user1");

      switch(matcher.match(uri)){
     // 根据URI匹配的返回码是URI_CODE_a
     // 即matcher.match(uri) == URI_CODE_a
      case URI_CODE_a:
        return tableNameUser1;
        // 如果根据URI匹配的返回码是URI_CODE_a，则返回
ContentProvider中的名为tableNameUser1的表
      case URI_CODE_b:
        return tableNameUser2;
        // 如果根据URI匹配的返回码是URI_CODE_b，则返回
ContentProvider中的名为tableNameUser2的表
    }
}
```

ContentObserver类

定义：内容观察者

- 作用：观察 Uri引起 ContentProvider 中的数据变化 & 通知外界（即访问该数据访问者）
  当ContentProvider 中的数据发生变化（增、删 & 改）时，就会触发该 ContentObserver类
  具体使用:

```
// 步骤1：注册内容观察者ContentObserver
    getContentResolver().registerContentObserver（uri）；
    // 通过ContentResolver类进行注册，并指定需要观察的URI

// 步骤2：当该URI的ContentProvider数据发生变化时，通知外界（即访
问该ContentProvider数据的访问者）
    public class UserContentProvider extends
ContentProvider {
      public Uri insert(Uri uri, ContentValues values) {
```

```
        db.insert("user", "userid", values);
        getContext().getContentResolver().notifyChange(uri,
null);
        // 通知访问者
    }
}

// 步骤3：解除观察者
 getContentResolver().unregisterContentObserver（uri）；
    // 同样需要通过ContentResolver类进行解除
```

创建自定义ContentProvider的步骤：

```
1.使用SQLite技术，创建好数据库和数据表
2.新建类继承ContentProvider
3.重写6个抽象方法
4.创建UriMatcher，定义Uri规则
5.在Manifest中注册provider
6.ContentResolver对ContentProvider中共享的数据进行增删改查操作
```

四大组件学习心得

```
1. 更加深入的理解了四大组件的实现机制，以及很多之前不清楚的知识，基础知识得以充实；
2. 其中对ContentProvider这边还是有点模糊，还需要自己练习一下；
3. 最深的是Activity的异常生命周期，之前工作的时候就遇到过类似情况，由于都不是专业的Android开发，语言切换导致走到了异常生命周期，当时那个项目排查此问题花费的很长时间；
4. Android基础还是需要多多看；
```