个人资料

ToYueXinShangWan

访问： 106850次
积分： 1851
等级： BLOG 4
排名： 第13183名

原创： 78篇 转载： 20篇
译文： 2篇 评论： 30条

文章搜索

文章分类

业内状况 (1)
设计模式 (9)
C++ (1)
JAVA (3)
Android 工具类 (3)
Android 系统编译 (0)
Android 应用开发 (22)
泷果网答题 (2)
Android Open (5)
算法 (2)

文章存档

2015年12月 (1)
2015年11月 (1)
2015年10月 (1)
2015年05月 (1)
2014年12月 (2)

展开

阅读排行

Android线程间通信机制 (16710)

Unity3D中基于订阅者模式实现事件机制 云计算行业圆桌论坛 【征文】Hadoop十周年特别策划——我与Hadoop不得不说的故事

Android跨进程通信的四种方式

2013-03-06 09:53 9047人阅读 评论(6) 收藏 举报

由于android系统中应用程序之间不能共享内存。因此，在不同应用程序之间交互数据（跨进程通讯）就稍微麻烦一些。在android SDK中提供了4种用于跨进程通讯的方式。这4种方式正好对应于android系统中4种应用程序组件：Activity、Content Provider、Broadcast和Service。其中Activity可以跨进程调用其他应用程序的Activity；Content Provider可以跨进程访问其他应用程序中的数据（以Cursor对象形式返回），当然，也可以对其他应用程序的数据进行增、删、改操 作；Broadcast可以向android系统中所有应用程序发送广播，而需要跨进程通讯的应用程序可以监听这些广播；Service和Content Provider类似，也可以访问其他应用程序中的数据，但不同的是，Content Provider返回的是Cursor对象，而Service返回的是Java对象，这种可以跨进程通讯的服务叫AIDL服务。

完整示例请参阅本文提供的源代码。

方式一：访问其他应用程序的**Activity**

Activity既可以在进程内（同一个应用程序）访问，也可以跨进程访问。如果想在同一个应用程序中访问Activity，需要指定Context对象和Activity的Class对象，代码如下：

```
01.  Intent intent = new  Intent(this , Test.class );
02.  startActivity(intent);
```

```
[java]
01.  Intent intent = new Intent(this, Test.class);
02.  startActivity(intent);
```

Activity的跨进程访问与进程内访问略有不同。虽然它们都需要Intent对象，但跨进程访问并不需要指定Context对象和Activity的 Class对象，而需要指定的是要访问的Activity所对应的Action（一个字符串）。有些Activity还需要指定一个Uri（通过 Intent构造方法的第2个参数指定）。

在android系统中有很多应用程序提供了可以跨进程访问的Activity，例如，下面的代码可以直接调用拨打电话的Activity。

```
01.  Intent callIntent = new  Intent(Intent.ACTION_CALL, Uri.parse("tel:12345678" );
02.  startActivity(callIntent);
```

```
[java]
01.  Intent callIntent = new Intent(Intent.ACTION_CALL, Uri.parse("tel:12345678");
02.  startActivity(callIntent);
```

执行上面的代码后，系统会自动拨号，界面如图1所示。

Android跨进程通信的四种 (9044)
Android加载网络图片学习 (7822)
自定义圆角风格dialog的 (6980)
android listview利用多线 (4954)
AlarmManager (4862)
Android ListView下拉刷新 (4807)
Android Activity之间切换时 (2383)
BlowFish算法加密解密 (2203)
基于百度地图API开发问题 (2116)

评论排行

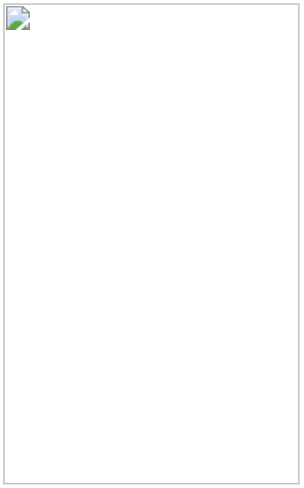
android listview利用多线 (7)
Android跨进程通信的四种 (6)
Android加载网络图片学习 (4)
Android 监视桌面 (3)
Android线程间通信机制 (3)
引导页 聚焦效果 (2)
AlarmManager (1)
什么是Service以及描述下 (1)
自定义圆角风格dialog的 (1)
想要成为游戏开发者？他 (1)

推荐文章

* HDFS如何检测并删除多余副本块
* Project Perfect让Swift在服务器端跑起来－让Perfect更Rails (五)
* 数据库性能优化之SQL语句优化
* Animation动画详解(七)——ObjectAnimator基本使用
* 机器学习系列(7)_机器学习路线图（附资料）
* 大数据三种典型云服务模式

最新评论

Android线程间通信机制
qq_29474489: 好 很好 非常好
Android 监视桌面
老曾哥: 楼主我运行你的代码出现java.net.SocketException: socket failed...
Android加载网络图片学习过程
chen214123158: 必须赞一个
自定义圆角风格dialog的Activity
aaadaaaada: 没实现,浪费我时间
备忘录模式
靳雯: 。。。你现在不是已经月薪上万了么，最近怎样，还在新浪？
Android跨进程通信的四种方式
吴大维David: 好文章，讲得很清晰
Android加载网络图片学习过程
life614: 受教了，楼主写的很好
android listview利用多线程异步l
hestersmile: MyConnection肿么写啊，大神求教啊
Android加载网络图片学习过程
Celia_CiCi: 赞赞
Android线程间通信机制
eagle1054: 看了两遍，有点点懂了，要多看几遍才能动（我是初学者）



在调用拨号程序的代码中使用了一个Intent.ACTION_CALL常量，该常量的定义如下：

```
01.  public  static  final  String ACTION_CALL = "android.intent.action.CALL" ;
```

```
[java]
01.  public static final String ACTION_CALL = "android.intent.action.CALL";
```

这个常量是一个字符串常量，也是我们在这节要介绍的跨进程调用Activity的关键。如果在应用程序中要共享某个Activity，需要为这个 Activity指定一个字符串ID，也就是Action。也可以将这个Action看做这个Activity的key。在其他的应用程序中只要通过这个 Action就可以找到与Action对应的Activity，并通过startActivity方法来启动这个Activity。

下面先来看一下如何将应用程序的Activity共享出来，读者可按如下几步来共享Activity：

1. 在AndroidManifest.xml文件中指定Action。指定Action要使用<action>标签，并在该标签的android:name属性中指定Action

2. 在AndroidManifest.xml文件中指定访问协议。在指定Uri（Intent类的第2个参数）时需要访问协议。访问协议需要使 用<data>标签的android:scheme属性来指定。如果该属性的值是“abc”，那么Uri就应该是“abc://Uri的主体 部分”，也就是说，访问协议是Uri的开头部分。

3. 通过getIntent().getData().getHost()方法获得协议后的Uri的主体部分。这个Host只是个称谓，并不一定是主机名。读者可以将其看成是任意的字符串。

4. 从Bundle对象中获得其他应用程序传递过来的数据。

5. 这一步当然是获得数据后做进一步的处理了。至于如何处理这些数据，就得根据具体的需求决定了。

下面来根据这些步骤共享一个Activity。首先建立一个android工程（ActionActivity），工程的主Activity是Main。在 本例中我们会共享这个Main类。首先打开AndroidManifest.xml文件，添加一个<activity>标签，并重新定义了 Main的相应属性。AndroidManifest.xml文件的内容如下：

```
01.  <!--  重新配置Main  -->
02.  <activity android:name=".Main"  android:label="@string/app_name" >
03.      <intent-filter>
04.          <action android:name="net.blogjava.mobile.MYACTION"  />
05.          <data android:scheme="info"  />
06.          <category android:name="android.intent.category.DEFAULT"  />
07.      </intent-filter>
08.  </activity>
```

```
[java]
01.  <!--  重新配置Main  -->
02.  <activity android:name=".Main" android:label="@string/app_name">
03.      <intent-filter>
04.          <action android:name="net.blogjava.mobile.MYACTION" />
05.          <data android:scheme="info" />
06.          <category android:name="android.intent.category.DEFAULT" />
07.      </intent-filter>
08.  </activity>
```

在配置AndroidManifest.xml时要注意，不能在同一个<activity>中配置多个动作，否则会覆盖MAIN动作以使该程序无法正常启动（虽然其他应用程序调用Main是正常的）。

从上面的代码可以看出，<action>标签的android:name属性值是 net.blogjava.mobile.MYACTION，这就是

Main自定义的动作。<data>标签指定了Url的协议。如果指定 了<data>标签的android:scheme属性值（info），则在调用Main时需要使用如下的URL:

```
info://任意字符串
```

[java]

```
info://任意字符串
```

一般<category>标签的android:name属性值可以设成android.intent.category.DEFAULT。

下面来看看如何在Main类的onCreate方法中获得其他应用程序传递过来的数据。

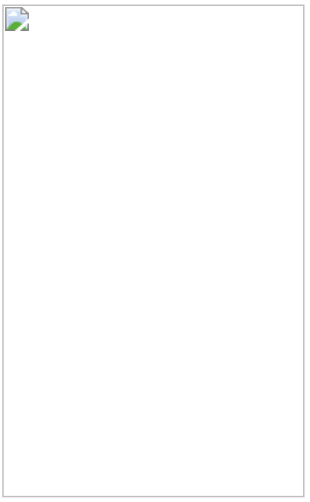
```
package  net.blogjava.mobile.actionactivity;
... ...
public  class  Main extends  Activity implements  OnClickListener
{
    private  EditText editText;
    @Override
    public  void  onClick(View view)
    {
        //  单击按钮，会显示文本框中的内容（以Toast信息框形式显示）
        Toast.makeText(this , editText.getText().toString(), Toast.LENGTH_LONG)
                .show();
    }
    @Override
    public  void  onCreate(Bundle savedInstanceState)
    {
        super .onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button button = (Button) findViewById(R.id.button);
        button.setOnClickListener(this );
        editText = (EditText) findViewById(R.id.edittext);
        //  获得其他应用程序传递过来的数据
        if  (getIntent().getData() != null )
        {
            //  获得Host，也就是info://后面的内容
            String host = getIntent().getData().getHost();
            Bundle bundle = getIntent().getExtras();
            //  其他的应用程序会传递过来一个value值，在该应用程序中需要获得这个值
            String value = bundle.getString("value" );
            //  将Host和Value组合在一下显示在EditText组件中
            editText.setText(host + ":"  + value);
            //  调用了按钮的单击事件，显示Toast信息提示框
            onClick(button);
        }
    }
}
```

[java]

```
package net.blogjava.mobile.actionactivity;
... ...
public class Main extends Activity implements OnClickListener
{
    private EditText editText;
    @Override
    public void onClick(View view)
    {
        //  单击按钮，会显示文本框中的内容（以Toast信息框形式显示）
        Toast.makeText(this, editText.getText().toString(), Toast.LENGTH_LONG)
                .show();
    }
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button button = (Button) findViewById(R.id.button);
        button.setOnClickListener(this);
        editText = (EditText) findViewById(R.id.edittext);
        //  获得其他应用程序传递过来的数据
        if (getIntent().getData() != null)
        {
```

```
            //  获得Host，也就是info://后面的内容
            String host = getIntent().getData().getHost();
            Bundle bundle = getIntent().getExtras();
            //  其他的应用程序会传递过来一个value值，在该应用程序中需要获得这个值
            String value = bundle.getString("value");
            //  将Host和Value组合在一下显示在EditText组件中
            editText.setText(host + ":" + value);
            //  调用了按钮的单击事件，显示Toast信息提示框
            onClick(button);
        }
    }
}
```

从上面的程序可以看出，首先通过getIntent().getData()来判断其他的应用程序是否传递了Uri（getData方法返回了一个Uri 对象）。如果运行该程序，Uri为null，因此，不会执行if语句里面的代码。当其他的应用程序传递了Uri对象后，系统会执行if语句里面的代码。当 运行ActionActivity后，在文本框中输入“Running”，单击“显示文本框的内容”按钮，会显示如图2所示的Toast提示信息框。

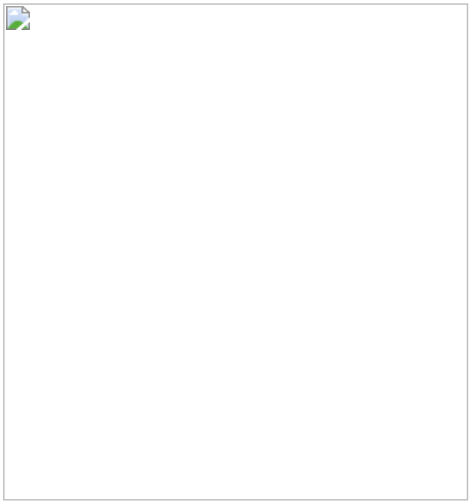


下面来看一下其他的应用程序是如何调用ActionActivity中的Main。新建一个android工程（InvokeActivity），并添加一个按钮，按钮的单击事件方法代码如下：

```
public  void  onClick(View view)
{
    //  需要使用Intent类的第2个参数指定Uri
    Intent intent = new  Intent("net.blogjava.mobile.MYACTION" , Uri
            .parse("info://调用其他应用程序的Activity" ));
    //  设置value属性值
    intent.putExtra("value" , "调用成功" );
    //  调用ActionActivity中的Main
    startActivity(intent);
}
```

```
[java]
public void onClick(View view)
{
    //  需要使用Intent类的第2个参数指定Uri
    Intent intent = new Intent("net.blogjava.mobile.MYACTION", Uri
            .parse("info://调用其他应用程序的Activity"));
    //  设置value属性值
    intent.putExtra("value", "调用成功");
    //  调用ActionActivity中的Main
    startActivity(intent);
}
```

在运行InvokeActivity之前，先要运行ActionActivity以便在android模拟器中安装该程序。然后单击InvokeActivity中的按钮，就会显示如图3所示的效果。



当然，也可以使用startActivityForResult方法来启动其他应用程序的Activity，以便获得Activity的返回值。例如，可以将ActionActivity中Main类的onClick代码修改为下面的形式。

```
01.  public  void  onClick(View view)
02.  {
03.      Toast.makeText(this , editText.getText().toString(), Toast.LENGTH_LONG).show();
04.      Intent intent = new  Intent();
05.      //  设置要返回的属性值
06.      intent.putExtra("result" , editText.getText().toString());
07.      //  设置返回码和Intent对象
08.      setResult(2 , intent);
09.      //  关闭Activity
10.      finish();
11.  }
```

```
[java]
01.  public void onClick(View view)
02.  {
03.      Toast.makeText(this, editText.getText().toString(), Toast.LENGTH_LONG).show();
04.      Intent intent = new Intent();
05.      //  设置要返回的属性值
06.      intent.putExtra("result", editText.getText().toString());
07.      //  设置返回码和Intent对象
08.      setResult(2, intent);
09.      //  关闭Activity
10.      finish();
11.  }
```

然后在InvokeActivity中使用下面的代码来调用Main。

```
01.  intent = new  Intent("net.blogjava.mobile.MYACTION" , Uri
02.          .parse("info://调用其他应用程序的Activity" ));
03.  //  传递数据
04.  intent.putExtra("value" , "调用成功" );
05.  startActivityForResult(intent, 1 );              //  1为请求码
```

```
[java]
01.  intent = new Intent("net.blogjava.mobile.MYACTION", Uri
02.          .parse("info://调用其他应用程序的Activity"));
03.  //  传递数据
04.  intent.putExtra("value", "调用成功");
05.  startActivityForResult(intent, 1);              //  1为请求码
```

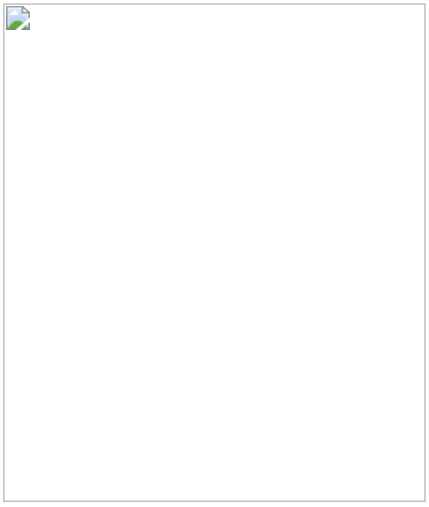
要想接收Activity返回的值，需要覆盖onActivityResult事件方法，代码如下：

```
01.  @Override
02.  protected  void  onActivityResult(int  requestCode, int  resultCode, Intent data)
03.  {
04.      Toast.makeText(this , "返回值: "  + data.getExtras().getString("result" ),
```

```
            Toast.LENGTH_LONG).show();
}
```

```
[java]
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
    Toast.makeText(this, "返回值：" + data.getExtras().getString("result"),
            Toast.LENGTH_LONG).show();
}
```

当单击InvokeActivity中的相应按钮后，并且Main关闭后，会显示如图4所示的Toast信息提示框。



从本节的介绍可以看出，跨进程访问Activity（访问其他应用程序中的Activity）主要是通过一个Action来完成的，如果要传递数据，还需 要指定一个Uri。当然，传递数据也可以通过Intent来完成。传递数据的过程可以是双向的。如果要想从调用的Activity中返回数据，就需要使用 startActivityForResult方法来启动Activity了。

方式二：**Content Provider**

Android应用程序可以使用文件或SqlLite数据库来存储数据。Content Provider提供了一种在多个应用程序之间数据共享的方式（跨进程共享数据）。应用程序可以利用Content Provider完成下面的工作

1. 查询数据
2. 修改数据
3. 添加数据
4. 删除数据

虽然Content Provider也可以在同一个应用程序中被访问，但这么做并没有什么意义。Content Provider存在的目的向其他应用程序共享数据和允许其他应用程序对数据进行增、删、改操作。

Android系统本身提供了很多Content Provider，例如，音频、视频、联系人信息等等。我们可以通过这些Content Provider获得相关信息的列表。这些列表数据将以Cursor对象返回。因此，从Content Provider返回的数据是二维表的形式。

对于访问Content Provider的程序，需要使用ContentResolver对象。该对象需要使用getContentResolver方法获得，代码如下：

```
ContentResolver cr = getContentResolver();
```

```
[java]
ContentResolver cr = getContentResolver();
```

与Activity一样，Content Provider也需要与一个URI对应。每一个Content Provider可以控制多个数据集，在这种情况下，每一个数据集会对应一个单独的URI。所有的URI必须以“content://”开头。

为了程序更容易维护，也为了简化程序代码，一般将URI定义成一个常量。例如，下面的常量表示系统的联系人电话号码。

```
android.provider.Contacts.Phones.CONTENT_URI
```

```
[java]
```

```
android.provider.Contacts.Phones.CONTENT_URI
```

下面来看一下编写Content Provider的具体步骤。

1. 编写一个继承于android.content.ContentProvider的子类。该类是ContentProvider的核心类。在该类中会实现 query、insert、update及delete方法。实际上调用ContentResolver类的这4个方法就是调用ContentProvider类中与之要对应的方法。在本文中只介绍query。至于insert、update、delete和query的用法类 似。也是通过Uri传递参数，然后在这些方法中接收这些参数，并做进一步地处理。
2. 在AndroidManifest.xml文件中配置ContentProvider。要想唯一确定一个ContentProvider，需要指定这个 ContentProvider的URI，除此之外，还需要指定URI所对应的ContentProvider类。这有些象Servlet的定义，除了要 指定Servlet对应的Web地址，还要指定这个地址所对应的Servlet类。

现在来看一下Uri的具体格式，先看一下如图5所示的URI。

下面对图5所示的URI的4个部分做一下解释。

A：Content Provider URI的固定前缀，也就是说，所有的URI必须以content://开头。

B：URI中最重要的部分。该部分是Content Provider的唯一标识。对于第三方应用程序来说，该部分最后使用完整的类名（包名+类名），以确保URI的唯一性。该部分需要在 AndroidManifest.xml文件中<provider>标签中定义，代码如下：

```
<provider name=".TransportationProvider" authorities="com.example.transportationprovider"
          . . .  >
```

```
[java]
<provider name=".TransportationProvider" authorities="com.example.transportationprovider"
          . . .  >
```

C：这部分是URI的路径（path）。表示URI中各种被请求的数据。这部分是可选的， 如果Content Provider仅仅提供一种请求的数据，那么这部分可以省略。如果Content Provider要提供多种请求数据。就需要添加多个路径，甚至是子路径。例如，“land/bus”、“land/train”、“sea/ship” 就指定了3种可能提供的数据。

D：这部分也是可选的。如果要传递一个值给Content Provider，可以通过这部分传递。当然，如果不需要传值，这部分也可以省略，省略后的URI如下所示：

```
content://com.example.transportationprovider/trains
```

```
[java]
content://com.example.transportationprovider/trains
```

本例利用了《基于 android SDK1.5的英文电子词典的实现》一文中实现的电子词典程序。通过ContentProvider，将电子词典的查词功能共享成Cursor对象。这样 其他的应用程序就可以通过ContentProvider来查词英文单词了。关于英文词典的具体实现细节，读者可以通过如下的地址查看《基于android SDK1.5的英文电子词典的实现》一文。

```
http://www. androidsdn.com/article/show/111
```

```
[java]
http://www.ophonesdn.com/article/show/111
```

在电子词典程序中需要一个DictionaryContentProvider类，该类是ContentProvider的子类。在该类中实现了query方法，并根据不同的URI来返回不同的结果。让我们先看一下DictionaryContentProvider类，然后再对这些代码做一些解 释。

```
... ...
public  class  DictionaryContentProvider extends  ContentProvider
{
    private  static  UriMatcher uriMatcher;
    private  static  final String AUTHORITY = "net.blogjava.mobile.dictionarycontentprovid
    private  static  final  int  SINGLE_WORD = 1 ;
    private  static  final  int  PREFIX_WORDS = 2 ;
    public  static  final  String DATABASE_PATH = android.os.Environment
    .getExternalStorageDirectory().getAbsolutePath()
    + "/dictionary" ;
    public  static  final  String DATABASE_FILENAME = "dictionary.db" ;
    private  SQLiteDatabase database;
    static
    {
        //  添加访问ContentProvider的Uri
        uriMatcher = new  UriMatcher(UriMatcher.NO_MATCH);
        uriMatcher.addURI(AUTHORITY, "single" , SINGLE_WORD);
        uriMatcher.addURI(AUTHORITY, "prefix/*" , PREFIX_WORDS);
    }
    //  该方法在Activity的onCreate方法之前调用
    @Override
    public  boolean  onCreate()
    {
        database = openDatabase();
        return  true ;
    }
    //  在本例中只实现了query方法，其他的方法（insert、update和delete）与query方法的实现
    //  类似
    @Override
    public  Cursor query(Uri uri, String[] projection, String selection,
            String[] selectionArgs, String sortOrder)
    {
        Cursor cursor = null ;
        switch  (uriMatcher.match(uri))
        {
            case  SINGLE_WORD:
                //  查找指定的单词
                cursor = database.query("t_words" , projection, selection,
                        selectionArgs, null , null , sortOrder);
                break ;
            case  PREFIX_WORDS:
                String word = uri.getPathSegments().get(1 );
                //  查找以指定字符串开头的单词集合
                cursor = database
                        .rawQuery(
                                "select english as _id, chinese from t_words where english
                                new  String[]
                                { word + "%"  });
                break ;

            default :
                throw  new  IllegalArgumentException("<"  + uri + ">格式不正确." );
        }
        return  cursor;
    }
    ... ...
}
```

[java]

```
... ...
public class DictionaryContentProvider extends ContentProvider
{
    private static UriMatcher uriMatcher;
    private static final String AUTHORITY = "net.blogjava.mobile.dictionarycontentprovider
    private static final int SINGLE_WORD = 1;
    private static final int PREFIX_WORDS = 2;
    public static final String DATABASE_PATH = android.os.Environment
    .getExternalStorageDirectory().getAbsolutePath()
    + "/dictionary";
    public static final String DATABASE_FILENAME = "dictionary.db";
    private SQLiteDatabase database;
    static
    {
        //  添加访问ContentProvider的Uri
        uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
```

```
        uriMatcher.addURI(AUTHORITY, "single", SINGLE_WORD);
        uriMatcher.addURI(AUTHORITY, "prefix/*", PREFIX_WORDS);
    }
    //  该方法在Activity的onCreate方法之前调用
    @Override
    public boolean onCreate()
    {
        database = openDatabase();
        return true;
    }
    //  在本例中只实现了query方法，其他的方法（insert、update和delete）与query方法的实现
    //  类似
    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
            String[] selectionArgs, String sortOrder)
    {
        Cursor cursor = null;
        switch (uriMatcher.match(uri))
        {
            case SINGLE_WORD:
                //  查找指定的单词
                cursor = database.query("t_words", projection, selection,
                        selectionArgs, null, null, sortOrder);
                break;
            case PREFIX_WORDS:
                String word = uri.getPathSegments().get(1);
                //  查找以指定字符串开头的单词集合
                cursor = database
                        .rawQuery(
                                "select english as _id, chinese from t_words where english
                                new String[]
                                { word + "%" });
                break;

            default:
                throw new IllegalArgumentException("<" + uri + ">格式不正确.");
        }
        return cursor;
    }
    ... ...
}
```

关于DictionaryContentProvider类的代码需要做如下的解释。

1. 在DictionaryContentProvider类的开头定义的AUTHORITY是访问ContentProvider的URI的前半部分。
2. 访问ContentProvider的URI的后半部分由uriMatcher.addURI(...)方法指定。该方法的第1个参数就是AUTHORITY（Uri的前半部分），第2个参数是Uri的后半部分，第3个参数是与第2个参数值对应的代码。当其他的应用程序通过Uri访问 ContentProvider时。系统解析Uri后，将addURI方法的第2个参数值转换成与之对应的代码（第3个参数值）。
3. addURI的第2个参数值可以使用通配符。例如，prefix/*中的*表示所有字符。prefix/abc、prefix/xxx都会匹配成功。
4. 访问ContentProvider的URI是addURI的第1个和第2个参数值的组件，例如，按着DictionaryContentProvider中设置的两个URI，可以分别匹配下面的两个URI。

```
content://net.blogjava.mobile.dictionarycontentprovider/single
content://net.blogjava.mobile.dictionarycontentprovider/prefix/wo
```

```
[java]
content://net.blogjava.mobile.dictionarycontentprovider/single
content://net.blogjava.mobile.dictionarycontentprovider/prefix/wo
```

要注意的是，访问ContentProvider的URI必须以“content://”开头。

5. 在query方法中建议使用SQLiteDatabase对象的query方法查询。因为query方法的参数正好和DictionaryContentProvider类中的query方法的参数对应，这样使用起来比较方便。
6. 由于安装了ContentProvider的应用程序会先调用ContentProvider的onCreate方法（该方法会在

Activity的 onCreate方法之前调用），因此，只需要将打开或复制数据库的方法（openDatabase）放在DictionaryContentProvider类中，并在onCreate方法中调用即可。

7．在DictionaryContentProvider类中只实现了query方法。在该方法中判断了其他应用程序发送的是哪一个Uri。并进行相应的处理。这两个Uri一个是查询指定单词的，另外一个是查询以某个字符串开头的所有单词的（用于显示单词列表）。

下面在AndroidManifest.xml文件中配置DictionaryContentProvider类。

```
<provider android:name="DictionaryContentProvider"
            android:authorities="net.blogjava.mobile.dictionarycontentprovider"  />
```

[java]

```
<provider android:name="DictionaryContentProvider"
            android:authorities="net.blogjava.mobile.dictionarycontentprovider" />
```

OK，现在来看看应用程序如何调用ContentProvider。调用ContentProvider的关键是使用getContentResolver方法来获得一个ContentResolver对象，并通过ContentResolver对象的query方法来 访问ContentProvider。

首先来定义两个访问ContentProvider的常量。

```
public  final String DICTIONARY_SINGLE_WORD_URI = "content://net.blogjava.mobile.dictionar
public  final String DICTIONARY_PREFIX_WORD_URI = "content://net.blogjava.mobile.dictionar
```

[java]

```
public final String DICTIONARY_SINGLE_WORD_URI = "content://net.blogjava.mobile.dictionary
public final String DICTIONARY_PREFIX_WORD_URI = "content://net.blogjava.mobile.dictionary
```

然后在查询按钮的单击事件中编写如下的代码来查询单词。

```
public  void  onClick(View view)
{
    Uri uri = Uri.parse(DICTIONARY_SINGLE_WORD_URI);
    //  通过ContentProvider查询单词，并返回Cursor对象，然后的操作就和直接从数据中获得
    //  Cursor对象后的操作是一样的了
    Cursor cursor = getContentResolver().query(uri, null , "english=?" ,
            new  String[]{ actvWord.getText().toString() }, null );
    String result = "未找到该单词." ;
    if  (cursor.getCount() > 0 )
    {
        cursor.moveToFirst();
        result = cursor.getString(cursor.getColumnIndex("chinese" ));
    }
    new  AlertDialog.Builder(this ).setTitle("查询结果" ).setMessage(result)
            .setPositiveButton("关闭" , null ).show();

}
```

[java]

```
public void onClick(View view)
{
    Uri uri = Uri.parse(DICTIONARY_SINGLE_WORD_URI);
    //  通过ContentProvider查询单词，并返回Cursor对象，然后的操作就和直接从数据中获得
    //  Cursor对象后的操作是一样的了
    Cursor cursor = getContentResolver().query(uri, null, "english=?",
            new String[]{ actvWord.getText().toString() }, null);
    String result = "未找到该单词.";
    if (cursor.getCount() > 0)
    {
        cursor.moveToFirst();
        result = cursor.getString(cursor.getColumnIndex("chinese"));
    }
    new AlertDialog.Builder(this).setTitle("查询结果").setMessage(result)
            .setPositiveButton("关闭", null).show();

}
```

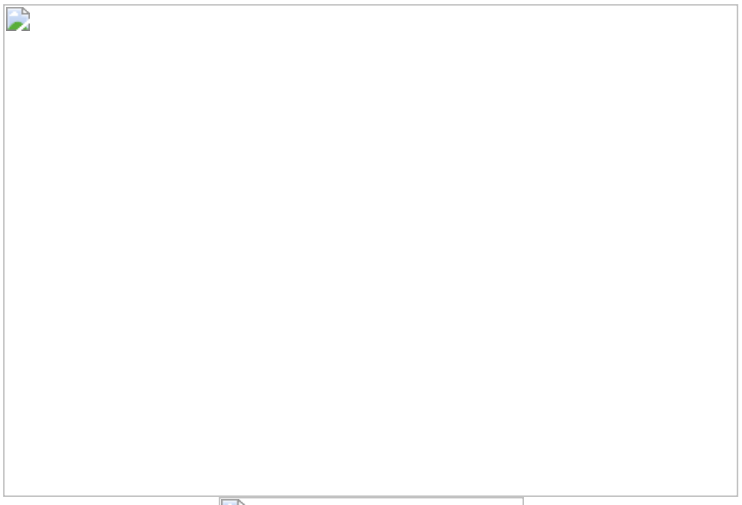
下面是显示单词列表的代码。

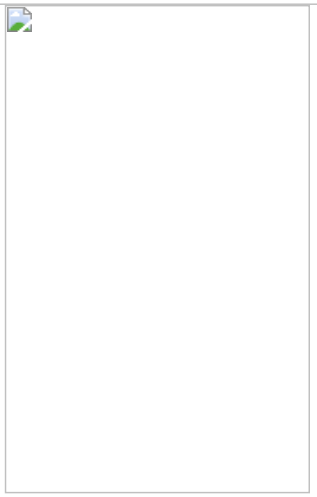
```
01.  public  void  afterTextChanged(Editable s)
02.  {
03.      if  ("" .equals(s.toString()))
04.          return ;
05.      Uri uri = Uri.parse(DICTIONARY_PREFIX_WORD_URI + "/"  + s.toString());
06.      //  从ContentProvider中获得以某个字符串开头的所有单词的Cursor对象
07.      Cursor cursor = getContentResolver().query(uri, null , null , null , null );
08.      DictionaryAdapter dictionaryAdapter = new  DictionaryAdapter(this ,
09.              cursor, true );
10.      actvWord.setAdapter(dictionaryAdapter);
11.  }
```

```
[java]

01.  public void afterTextChanged(Editable s)
02.  {
03.      if ("".equals(s.toString()))
04.          return;
05.      Uri uri = Uri.parse(DICTIONARY_PREFIX_WORD_URI + "/" + s.toString());
06.      //  从ContentProvider中获得以某个字符串开头的所有单词的Cursor对象
07.      Cursor cursor = getContentResolver().query(uri, null, null, null, null);
08.      DictionaryAdapter dictionaryAdapter = new DictionaryAdapter(this,
09.              cursor, true);
10.      actvWord.setAdapter(dictionaryAdapter);
11.  }
```

现在来运行本例，会看到如图6所示的界面。当查询单词时会显示如图7所示的单词列表，查询出结果后，会显示如图8所示的界面。





方式三：广播（**Broadcast**）

广播是一种被动跨进程通讯的方式。当某个程序向系统发送广播时，其他的应用程序只能被动地接收广播数据。这就象电台进行广播一样，听众只能被动地收听，而不能主动与电台进行沟通。

在应用程序中发送广播比较简单。只需要调用sendBroadcast方法即可。该方法需要一个Intent对象。通过Intent

对象可以发送需要广播的数据。

先建一个android工程：sendbroadcast。在XML布局文件中放两个组件：EditText和Button，当单击按钮后，会弹出显示 EditText组件中文本的对话框，关闭对话框后， 会使用sendBroadcast方法发送消息，并将EditText组件的文本通过Intent对象发送出去。完整的代码如下：

```
01. package  net.blogjava.mobile.sendbroadcast;
02. ... ...
03. public  class  Main extends  Activity implements  OnClickListener
04. {
05.     private  EditText editText;
06.     @Override
07.     public  void  onClick(View view)
08.     {
09.         new  AlertDialog.Builder(this ).setMessage(editText.getText().toString())
10.                 .setPositiveButton("确定" , null ).show();
11.         //  通过Intent类的构造方法指定广播的ID
12.         Intent intent = new  Intent("net.blogjava.mobile.MYBROADCAST" );
13.         //  将要广播的数据添加到Intent对象中
14.         intent.putExtra("text" , editText.getText().toString());
15.         //  发送广播
16.         sendBroadcast(intent);
17.     }
18.     ... ...
19. }
```

```
[java]
01. package net.blogjava.mobile.sendbroadcast;
02. ... ...
03. public class Main extends Activity implements OnClickListener
04. {
05.     private EditText editText;
06.     @Override
07.     public void onClick(View view)
08.     {
09.         new AlertDialog.Builder(this).setMessage(editText.getText().toString())
10.                 .setPositiveButton("确定", null).show();
11.         //  通过Intent类的构造方法指定广播的ID
12.         Intent intent = new Intent("net.blogjava.mobile.MYBROADCAST");
13.         //  将要广播的数据添加到Intent对象中
14.         intent.putExtra("text", editText.getText().toString());
15.         //  发送广播
16.         sendBroadcast(intent);
17.     }
18.     ... ...
19. }
```

发送广播并不需要在AndroidManifest.xml文件中注册，但接收广播必须在AndroidManifest.xml文件中注册receiver。下面来编写一个接收广播的应用程序。首先建立一个android工程：receiver。然后编写一个MyReceiver类，该类是 BroadcastReceiver的子类，代码如下：

```
01. package  net.blogjava.mobile.receiver;
02. ... ...
03. public  class  MyReceiver extends  BroadcastReceiver
04. {
05.     //  当sendbroadcast发送广播时，系统会调用onReceive方法来接收广播
06.     @Override
07.     public  void  onReceive(Context context, Intent intent)
08. {
09.     //  判断是否为sendbroadcast发送的广播
10.         if  ("net.blogjava.mobile.MYBROADCAST" .equals(intent.getAction()))
11.         {
12.             Bundle bundle = intent.getExtras();
13.             if  (bundle != null )
14.             {
15.                 String text = bundle.getString("text" );
16.                 Toast.makeText(context, "成功接收广
    播: " + text, Toast.LENGTH_LONG).show();
17.             }
18.         }
19.     }
```

```
}
```

[java]

```
package net.[illegible]le.receiver;
... ...
public class [illegible]xtends BroadcastReceiver
{
    //  当sendbroadcast发送广播时，系统会调用onReceive方法来接收广播
    @Override
    public void onReceive(Context context, Intent intent)
{
    //  判断是否为sendbroadcast发送的广播
        if ("net.blogjava.mobile.MYBROADCAST".equals(intent.getAction()))
        {
            Bundle bundle = intent.getExtras();
            if (bund[illegible]l)
            {
                = bundle.getString("text");
                xt(context, "成功接收广
                NG).show();
```

nReceive方法来接收广播，并通过intent.getAction()方法返回广播的
定的字符串。然后就可以从Bundle对象中获得相应的数据了。
中注册receiver，代码如下：

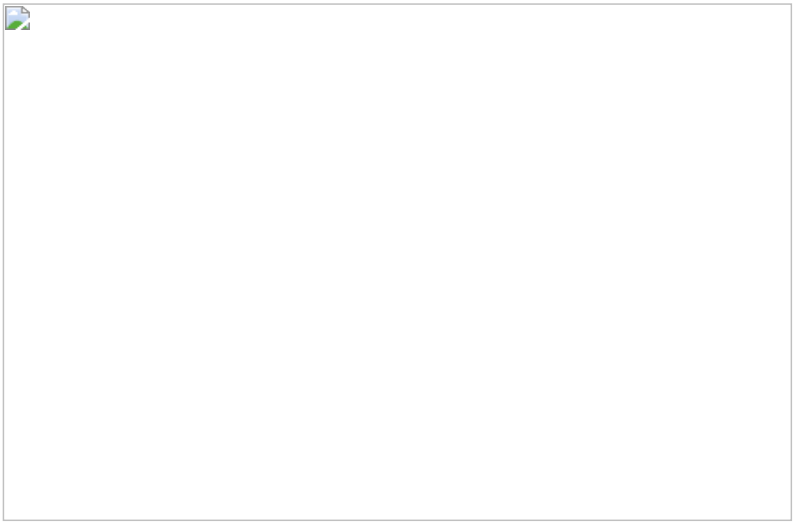
```
<!--  注册receiver -->
<receiver android:name="MyReceiver" >
    <intent-filter>
        <action android:name="net.blogjava.mobile.MYBROADCAST"  />
    </intent-filter>
</receiver>
```

[java]

```
<!--  注册receiver -->
<receiver android:name="MyReceiver">
    <intent-filter>
        <action android:name="net.blogjava.mobile.MYBROADCAST" />
    </intent-filter>
</receiver>
```

在注册MyReceiver类时需要使用<receiver>标签，android:name属性指定MyReceiver类，<action>标签的android:name指定了广播的ID。

首先运行receiver程序，然后就可以关闭receiver程序了。接收广播并不依赖于程序的状态。就算程序关闭了，仍然可以接收广播。然后再启动 sendbroadcast程序。并在文本框中输入“android”，然后单击按钮，会弹出一个显示文本框内容的对话框，如图9所示。当关闭对话框后，会 显示一个Toast信息提示框，这个信息框是由receiver程序弹出的。如图10所示。



方式四：**AIDL**服务

服务（Service）是android系统中非常重要的组件。Service可以脱离应用程序运行。也就是说，应用程序只起到一个启动Service的作用。一但Service被启动，就算应用程序关闭，Service仍然会在后台运行。

android系统中的Service主要有两个作用：后台运行和跨进程通讯。后台运行就不用说了，当Service启动后，就可以在Service对象中 运行相应的业务代码，而这一切用户并不会察觉。而跨进程通讯是这一节的主题。如果想让应用程序可以跨进程通讯，就要使用我们这节讲的AIDL服 务，AIDL的全称是Android Interface Definition Language，也就是说，AIDL实际上是一种接口定义语言。通过这种语言定义接口后，Eclipse插件（ODT）会自动生成相应的Java代码接 口代码。下面来看一下编写一个AIDL服务的基本步骤。

1. 在Eclipse工程的package目录中建立一个扩展名为aidl的文件。package目录就是Java类所在的目录。该文件的语法类似于Java代码。aidl文件中定义的是AIDL服务的接口。这个接口需要在调用AIDL服务的程序中访问。
2. 如果aidl文件的内容是正确的，Eclipse插件会自动生成一个Java接口文件（*.java）。
3. 建立一个服务类（Service的子类）。
4. 实现由aidl文件生成的Java接口。
5. 在AndroidManifest.xml文件中配置AIDL服务，尤其要注意的是，<action>标签的android:name属性值就是客户端要引用该服务的ID，也就是Intent类构造方法的参数值。

现在我们来编写一个AIDL服务，首先建立一个android工程：aidlservice。在aidlservice工程中有一个Main类，在Main类所有的目录建立一个IMyService.aidl文件，内容如下：

```
01.  package  net.blogjava.mobile.aidlservice;
02.  interface  IMyService
03.  {
04.      String getValue();  //  为AIDL服务的接口方法，调用AIDL服务的程序需要调用该方法
05.  }
```

```
[java]
01.  package net.blogjava.mobile.aidlservice;
02.  interface IMyService
03.  {
04.      String getValue();  //  为AIDL服务的接口方法，调用AIDL服务的程序需要调用该方法
05.  }
```

在保存IMyService.aidl文件后，ODT会在gen目录下产生一个IMyService.java文件，读者可以不必管这个文件中的内容，也 不需要修改该文件的内容。这个文件是由ODT自动维护的，只要修改了IMyService.aidl文件的内容，IMyService.java文件的内 容就会随之改变。

然后建立一个MyService类，该类是Service的子类，代码如下：

```
01.  package  net.blogjava.mobile.aidlservice;
02.  ... ...
03.  public  class  MyService extends  Service
04.  {
05.      //  IMyService.Stub类是根据IMyService.aidl文件生成的类，该类中包含了接口方法（getValue）
06.      public  class  MyServiceImpl extends  IMyService.Stub
07.      {
```

```
        @Override
        public  String getValue() throws  RemoteException
        {
            return  "从AIDL服务获得的值." ;
        }
    }
    @Override
    public  IBinder onBind(Intent intent)
{
//  该方法必须返回MyServiceImpl类的对象实例
        return  new  MyServiceImpl();
    }
}
```

[java]

```
package net.blogjava.mobile.aidlservice;
... ...
public class MyService extends Service
{
    //  IMyService.Stub类是根据IMyService.aidl文件生成的类，该类中包含了接口方法（getValue）
    public class MyServiceImpl extends IMyService.Stub
    {
        @Override
        public String getValue() throws RemoteException
        {
            return "从AIDL服务获得的值.";
        }
    }
    @Override
    public IBinder onBind(Intent intent)
{
//  该方法必须返回MyServiceImpl类的对象实例
        return new MyServiceImpl();
    }
}
```

最后需要在AndroidManifest.xml文件中配置MyService类，代码如下：

```
<!--  注册服务 -->
<service android:name=".MyService" >
    <intent-filter>
        <!--  指定调用AIDL服务的ID  -->
        <action android:name="net.blogjava.mobile.aidlservice.IMyService"  />
    </intent-filter>
</service>
```

[java]

```
<!--  注册服务 -->
<service android:name=".MyService">
    <intent-filter>
        <!--  指定调用AIDL服务的ID  -->
        <action android:name="net.blogjava.mobile.aidlservice.IMyService" />
    </intent-filter>
</service>
```

下面来看看如何调用这个AIDL服务。首先建立一个android工程：aidlclient。然后将aidlservice工程中自动生成的 IMyService.java文件复制到aidlclient工程中。在调用AIDL服务之前需要先使用bindService方法绑定AIDL服务。 bindService方法需要一个ServiceConnection对象。ServiceConnection有一个 onServiceConnected方法，当成功绑定AIDL服务且，该方法被调用。并通过service参数返回AIDL服务对象。下面是调用 AIDL服务的完成代码。

```
package  net.blogjava.mobile.aidlclient;
... ...
public  class  Main extends  Activity implements  OnClickListener
{
private  IMyService myService = null ;
//  创建ServiceConnection对象
    private  ServiceConnection serviceConnection = new  ServiceConnection()
    {
```

```
        @Override
        public  void  onServiceConnected(ComponentName name, IBinder service)
        {
            // 获得AIDL服务对象
            myService = IMyService.Stub.asInterface(service);
            try
            {
                //  调用AIDL服务对象中的getValue方法，并以对话框中显示该方法的返回值
                new  AlertDialog.Builder(Main.this ).setMessage(
                        myService.getValue()).setPositiveButton("确定" , null )
                        .show();
            }
            catch  (Exception e)
            {
            }
        }
        @Override
        public  void  onServiceDisconnected(ComponentName name)
        {
        }
    };
    @Override
    public  void  onClick(View view)
{
    //  绑定AIDL服务
        bindService(new  Intent("net.blogjava.mobile.aidlservice.IMyService" ),
                serviceConnection, Context.BIND_AUTO_CREATE);
    }
    ... ...
}
```

[java]

```
package net.blogjava.mobile.aidlclient;
... ...
public class Main extends Activity implements OnClickListener
{
private IMyService myService = null;
//  创建ServiceConnection对象
    private ServiceConnection serviceConnection = new ServiceConnection()
    {
        @Override
        public void onServiceConnected(ComponentName name, IBinder service)
        {
            // 获得AIDL服务对象
            myService = IMyService.Stub.asInterface(service);
            try
            {
                //  调用AIDL服务对象中的getValue方法，并以对话框中显示该方法的返回值
                new AlertDialog.Builder(Main.this).setMessage(
                        myService.getValue()).setPositiveButton("确定", null)
                        .show();
            }
            catch (Exception e)
            {
            }
        }
        @Override
        public void onServiceDisconnected(ComponentName name)
        {
        }
    };
    @Override
    public void onClick(View view)
{
    //  绑定AIDL服务
        bindService(new Intent("net.blogjava.mobile.aidlservice.IMyService"),
                serviceConnection, Context.BIND_AUTO_CREATE);
    }
    ... ...
}
```

在编写AIDL服务和客户端时要注意如下两点：

1. AIDL服务中的onBind方法必须返回AIDL接口对象（MyServiceImpl对象）。该对象也是onServiceConnected事件方法的第2个参数值。
2. bindService方法的第1个参数是Intent对象，该对象构造方法的参数需要指定AIDL服务的ID，也就是在

AndroidManifest.xml文件中<service>标签的<action>子标签的android:name属性 的值。

现在先运行aidlservice程序，以便安装AIDL服务，然后运行aidlclient程序，并单击按钮，会显示如图11所示的对话框。对话框中的信息就是AIDL服务接口中getValue方法的返回值。

总结

本文介绍了4种跨进程通讯的方式：Activity、ContentProvider、Broadcast和AIDL Service。其中Activity可以跨进程调用其他应用程序的Activity；ContentProvider可以访问其他应用程序返回的 Cursor对象；Broadcast采用的是被动接收的方法，也就是说，客户端只能接收广播数据，而不能向发送广播的程序发送信息。AIDL Service可以将程序中的某个接口公开，这样在其他的应用程序中就可以象访问本地对象一样访问AIDL服务对象了。这4种跨进程通讯的方式可以应用在 不同的场合，例如，在需要显示可视化的界面时可以用Activity，需要返回记录集时可以用ContentProvider。至于在应用程序中具体要用 到哪一种或几种方式进行跨进程通讯，读者可以根据实际情况进行选择。

查看评论

5楼 吴大维David 2015-03-21 14:31发表

好文章，讲得很清晰

4楼 北极的冰箱 2014-07-30 16:20发表


好文章。

3楼 qiyongqiang2012 2014-07-08 21:51发表

真是不赖

2楼 imyfriend 2014-03-22 17:44发表


那请问binder，共享内存这些东西算不算是android跨进程通信的方式呢？

1楼 imyfriend 2014-03-19 14:35发表


这对吗？请问楼主。这是android的四大组件吧

Re: ToYueXinShangWan 2014-03-21 17:38发表

回复imyfriend：虽然上面有四大组件相关描述，但这不是介绍的四大组件，

如题 Android跨进程通信的四种方式。
当然对。

您还没有登录,请[登录]或[注册]

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题 Hadoop AWS 移动游戏 Java Android iOS Swift 智能硬件 Docker OpenStack
VPN Spark ERP IE10 Eclipse CRM JavaScript 数据库 Ubuntu NFC WAP jQuery
BI HTML5 Spring Apache .NET API HTML SDK IIS Fedora XML LBS Unity
Splashtop UML components Windows Mobile Rails QEMU KDE Cassandra CloudStack
FTC coremail OPhone CouchBase 云计算 iOS6 Rackspace Web App SpringSide
Maemo Compuware 大数据 aptech Perl Tornado Ruby Hibernate ThinkPHP HBase
Pure Solr Angular Cloud Foundry Redis Scala Django Bootstrap