

使用Messenger实现进程间通信

1. 原理说明

Messenger作为信使，它在不同进程中传递Message对象，从而实现不同进程间信息的传递。

Messenger实现了Parcelable接口，从而可以被序列化传递。

Messenger构造方法：

|  |
| --- |
| [Messenger](https://developer.android.google.cn/reference/android/os/Messenger.html#Messenger%28android.os.Handler%29)([Handler](https://developer.android.google.cn/reference/android/os/Handler.html) target)  Create a new Messenger pointing to the given Handler.  创建一个指向给定Handler（信息处理方式）的Messenger（对象） |
| [Messenger](https://developer.android.google.cn/reference/android/os/Messenger.html#Messenger%28android.os.IBinder%29)([IBinder](https://developer.android.google.cn/reference/android/os/IBinder.html) target)  Create a Messenger from a raw IBinder, which had previously been retrieved with [getBinder()](https://developer.android.google.cn/reference/android/os/Messenger.html#getBinder%28%29).  通过IBinder创建Messenger对象，该IBinder是从Messenger对象中getBinder（）方法中恢复而来。 |

Messenger常用的方法

|  |  |
| --- | --- |
| [IBinder](https://developer.android.google.cn/reference/android/os/IBinder.html) | [getBinder](https://developer.android.google.cn/reference/android/os/Messenger.html" \l "getBinder%28%29)()  Retrieve the IBinder that this Messenger is using to communicate with its associated Handler.  恢复IBinder，用来和与其关联的Handler通信 |
| void | send(Message message)  Send a Message to this Messenger's Handler.  发送信息到和Messenger相关的Handler（也就是信息处理方式）。 |

Message构造方法：

|  |
| --- |
| [Message](https://developer.android.google.cn/reference/android/os/Message.html#Message%28%29)()  Constructor (but the preferred way to get a Message is to call [Message.obtain()](https://developer.android.google.cn/reference/android/os/Message.html#obtain%28%29)).一般通过该静态方法实现Message的实例化。 |

Message的属性

|  |  |
| --- | --- |
| public int | [arg1](https://developer.android.google.cn/reference/android/os/Message.html#arg1)  arg1 and arg2 are lower-cost alternatives to using [setData()](https://developer.android.google.cn/reference/android/os/Message.html#setData%28android.os.Bundle%29) if you only need to store a few integer values. 可以代替setData（）方法存储简单的整形数据。 |
| public int | [arg2](https://developer.android.google.cn/reference/android/os/Message.html#arg2)  arg1 and arg2 are lower-cost alternatives to using [setData()](https://developer.android.google.cn/reference/android/os/Message.html#setData%28android.os.Bundle%29) if you only need to store a few integer values. |
| public [Object](https://developer.android.google.cn/reference/java/lang/Object.html) | [obj](https://developer.android.google.cn/reference/android/os/Message.html#obj)  An arbitrary object to send to the recipient.  一个被发送给接受者的任意对象。 |
| public [Messenger](https://developer.android.google.cn/reference/android/os/Messenger.html) | [replyTo](https://developer.android.google.cn/reference/android/os/Message.html#replyTo)  Optional Messenger where replies to this message can be sent. 发送Message的Messenger，也就是获取message的载体。 |

Message的方法

|  |  |
| --- | --- |
| [Bundle](https://developer.android.google.cn/reference/android/os/Bundle.html) | [getData](https://developer.android.google.cn/reference/android/os/Message.html#getData%28%29)()  Obtains a Bundle of arbitrary data associated with this event, lazily creating it if necessary. 获取Message中的数据 |
| Handler | getTarget()  Retrieve the a Handler implementation that will receive this message.获得接收该Message的Handler对象 |
| static Message | obtain(Handler h, int what, Object obj)  Same as obtain(), but sets the values of the target, what, and obj members. |
| static Message | obtain()  Return a new Message instance from the global pool.  从全局线程中获取一个Message的实例 |
| void | setData(Bundle data)  Sets a Bundle of arbitrary data values.设置一组数据 |

以上是线程通信过程中需要使用一些基本方法和类。下面把Service当做服务端，MainActivity当做客户端进行实例分析。

1：Service只接受来自MainActivity的数据。

Service需要处理的事情：1）创建一个处理Message的Handler类（方法多样）；2）创建一个Handler作为参数的Messenger的实例；3) 在OnBinder（）方法中返回IBinder-该对象需要messenger. getBinder()获得。代码如下：

|  |
| --- |
| */\*\*  \* 1：class  \* 创建一个Handler类，接受消息  \*/* private static class MessengerHandler extends Handler{  private static final String *TAG* = "MyService";   @Override  public void handleMessage(Message msg) {  switch (msg.what){  case *MSG\_FROM\_CLIENT*:  // msg.getData().getString("msg")获取数据  Log.*i*(*TAG*,"this is the msg from client :"+msg.getData().getString("msg"));   break;   default:  super.handleMessage(msg);  }  } }  */\*\**  *2：  \* 创建Messenger对象，该对象持有Handler对象  \*/* private final Messenger mMessenger = new Messenger(new MessengerHandler());  public MyService() { }  //3: @Override public IBinder onBind(Intent intent) { //返回IBinder  return mMessenger.getBinder(); } |

客户端可以发送数据：

采用类似于Service绑定的方式开启服务，并与之通信。具体实现方式：

1）创建ServiceConnection，并复写其中方法，实例化化一个Messenger；2）再需要复写的方法中添加Messenger发送数据的内容；3）绑定服务，并且在程序退出时，解除绑定。示例代码如下

|  |
| --- |
| private ServiceConnection connection = new ServiceConnection() {  @Override  public void onServiceConnected(ComponentName componentName, IBinder iBinder) {  //信使，全局变量，通过IBinder创建Messenger实例  messenger = new Messenger(iBinder);  //信息载体，通过obtain()进行实例化  Message message = Message.*obtain*(null,MyService.*MSG\_FROM\_CLIENT*);  //信息内容，即数据  Bundle bundle = new Bundle() ;  bundle.putString("msg","你好，我来自美丽的星星，yeah！");  message.setData(bundle);     try {  messenger.send(message);  } catch (RemoteException e) {  e.printStackTrace();  }  }   @Override  public void onServiceDisconnected(ComponentName componentName) {   } };  @Override protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  setContentView(R.layout.*activity\_main*);   Intent intent = new Intent(this,MyService.class);  //绑定服务  bindService(intent,connection, Context.*BIND\_AUTO\_CREATE*); }  @Override protected void onDestroy() {  super.onDestroy(); //退出后，自动销毁  unbindService(connection);  } |

2：Service向来自MainActivity发送数据

在数据通信过程中，MainActivity需要获取来自Service的数据，因此Service需要对外发送数据。

第一步，Service需要对外发送数据。1）在Handler中的获取Messenger对象，因为Handler中操控的是Message，而Message中有个replyTo的属性就是获取传递信息的Messenger。2）再通过obtain()方法获取实例化新的Message信息数据的包装；3）然后通过Messenger的sendData()方法发送数据。具体代码如下：

|  |
| --- |
| //获取Messenger的实例  Messenger messenger = msg.replyTo;  //获取信息载体Message  Message message = Message.obtain(null, MyService.MSG\_FROM\_SERVICE);  //数据组装，并且发送  //数据 Bundle bundle = new Bundle();  //数据key ="reply", value = "我很开心" bundle.putString("reply","我很开心"); message.setData(bundle); try {  messenger.send(message); } catch (RemoteException e) {  e.printStackTrace(); } |

第二步，就是在MainActivity中接收信息。具体操作步骤如下：

1）创建Handler的实例（可以是内部类），该类中接收消息；2）创建Messenger，通过new Messenger(Handler)的方式创建该类；3）在Connection中的方法中将Message的replyTo属性值设为新创建的Messenger。

代码如下：

|  |
| --- |
| */\*\*  \* 1：class  \* 信息处理的Handler的子类  \*  \*/* private static final class ClientMessengerHandler extends Handler{  @Override  public void handleMessage(Message msg) {  super.handleMessage(msg);  switch (msg.what){  case MyService.*MSG\_FROM\_SERVICE*:  Log.*i*("TAG","来此服务端的消息回复："+msg.getData().getString("reply"));  break;  }   } }  //2：  private Messenger receiverMessenger = new Messenger(new ClientMessengerHandler());  //3:其中被两行8注释的部分  private ServiceConnection connection = new ServiceConnection() {  @Override  public void onServiceConnected(ComponentName componentName, IBinder iBinder) {  //信使  messenger = new Messenger(iBinder);  //信息载体  Message message = Message.*obtain*(null,MyService.*MSG\_FROM\_CLIENT*);  //信息内容，即数据  Bundle bundle = new Bundle() ;  bundle.putString("msg","你好，我来自美丽的星星，yeah！");  message.setData(bundle);   //8888888888888888888，未接收数据而修改信息回应的信使  message.replyTo = receiverMessenger ;  //8888888888888888888  try {  messenger.send(message);  } catch (RemoteException e) {  e.printStackTrace();  }  }   @Override  public void onServiceDisconnected(ComponentName componentName) {   } }; |

2综上所述，在服务器方回复消息时，首先通过Message的replyTo属性获取信使（Messenger），从而发送消息；接收方获取信息时，将信使（Messenger）赋值给Message的replyTo属性值。应用部分就讲解到这里。