原创 Android Handler Message总结

2013-01-24 09:14:55 Carserdadi 阅读数 114130 ☆ 收藏 更多

当应用程序启动时，会开启一个主线程（也就是UI线程），由她来管理UI，监听用户点击，来响应用户并分发事件等。所以一般在主线程中不要执行比较耗时的操作，如联网下载数据等，否则出现ANR错误。所以就将这些操作放在子线程中，但是由于AndroidUI线程是不安全的，所以只能在主线程中更新UI。Handler就是用来 子线程和创建Handler的线程进行通信的。

Handler的使用分为两部分：

一部分是创建Handler实例，重载handleMessage方法，来处理消息。

```
mProgressHandler = new Handler()
        {
            public void handleMessage(Message msg)
            {
                super.handleMessage(msg);
            }
        };
```

登录后复制

当然，也可继承自Handler，同样要实现handleMessage(Message msg)方法。

```
class MyHandler extends Handler {
        public MyHandler() {
        }

        // 子类必须重写此方法,接受数据
        @Override
        public void handleMessage(Message msg) {
            // TODO Auto-generated method stub
            Log.d("MyHandler", "handleMessage......");
            super.handleMessage(msg);

        }
    }

```

另一部分是分发Message 或者Runable对象到Handler所在的线程中，一般Handler在主线程中。

Handler中分发消息的一些方法
post(Runnable)
postAtTime(Runnable,long)
postDelayed(Runnable long)
sendEmptyMessage(int what)
sendMessage(Message)
sendMessageAtTime(Message,long)
sendMessageDelayed(Message,long)

handler本身不仅可以发送消息，还可以用post的方式添加一个实现Runnable接口的匿名对象到消息队列中，在目标收到消息后就可以回调的方式在自己的线程中执行run的方法体。

Message

```
        Message message = Message.obtain();
        message.arg1 = 1;
        message.arg2 = 2;
        message.obj = "Demo";
        message.what = 3;
        Bundle bundleData = new Bundle();
        bundleData.putString("Name", "Lucy");
        message.setData(bundleData);
```

Message 可以传递的参数有：

1. arg1 arg2 整数类型，是setData的低成本替代品。传递简单类型

2. Object 类型 obj

3. what 用户自定义的消息代码，这样接受者可以了解这个消息的信息。每个handler各自包含自己的消息代码，所以不用担心自定义的消息跟其他handlers有冲突。

4.其他的可以通过Bundle进行传递

Message可以通过new Message构造来创建一个新的Message,但是这种方式很不好，不建议使用。最好使用Message.obtain()来获取Message实例,它创建了消息池来处理的。

公共构造器

public **Message()**

构造器（但是获取Message对象的最好方法是调用Message.obtain()）。

如下这些通过Message.obtain方式获取Message实例，参数中传递了Handler,发送该消息时不再使用handler.sendMessage这种方式。使用message.sendToTarget();不过归根到底都是调用Handler.sendMessage进行发送消息。Message类中保存Handler实例。

public static Message **obtain** (Handler h, int what, int arg1, int arg2, Object obj)

与obtain()一样，但是设置了target, what, arg1, arg2和obj的值。

参数

h	设置的target值
what	设置的what值
arg1	设置的arg1值
arg2	设置的arg2值
obj	设置的obj值

返回值

从全局池中分配的一个Message对象。

public static Message **obtain** (Handler h, int what, Object obj)

与obtain()一样，但是设置了target, what和obj的值。

参数

h 设置的target值

what 设置的what值

obj 设置的obj值

返回值

从全局池中分配的一个Message对象。

public static Message**obtain** (Handler h, int what)

与obtain()一样，但是设置了target和what的值。

参数

h target的值

what what的值

返回值

从全局池中分配的一个Message对象。

public static Message **obtain** (Handler h)

与obtain()一样，但是设置了target的值

参数

h 消息对象的target成员的值

返回值

从全局池中分配的一个Message对象。

public static Message **obtain** (Handler h, Runnable callback)

与obtain(Handler)一样，但是设置回调函数，在Message返回时调用。

参数

h 消息对象的target成员的值

callback 当消息处理时会调用的回调函数

返回值

从全局池中分配的一个Message对象。

public static Message **obtain** ()

从全局池中返回一个新的Message实例。在大多数情况下这样可以避免分配新的对象。

public static Message **obtain** (Handler h, int what, int arg1, int arg2)

与obtain()一样，但是设置了target, what, arg1和arg2的值

参数

h 设置的targe值

what 设置的what值

arg1 设置的arg1值

arg2　　设置的arg2值

返回值

从全局池中分配的一个Message对象。

public static Message **obtain** (Message obj)

同obtain()，但是从一个已存在的消息中拷贝值（包括它的目标）。

参数

orig　　要拷贝的源消息

返回值

从全局池中分配的一个Message对象。

public Bundle **peekData** ()

与getData()相似，但是并不延迟创建Bundle。如果Bundle对象不存在返回null。更多信息见getData()。

参考

getData()

setData(Bundle)

public void **recyle** ()

向全局池中返回一个Message实例。一定不能在调用此函数后再使用Message——它会立即被释放。

public void **sendToTarget** ()

向Handler发送此消息，getTarget()方法可以获取此Handler。如果这个字段没有设置会抛出个空指针异常。

public void **setData** (Bundle data)

设置一个任意数据值的Bundle对象。如果可以，使用arg1和arg2域发送一些整型值以减少消耗。

参考

getData()

peekData()

public void **setTarget** (Handler target)

设置将接收此消息的Handler对象。

PS:

线程安全和线程不安全

线程安全就是多线程访问时，采用了加锁机制，当一个线程访问该类的某个数据时，进行保护，其他线程不能进行访问直到该线程读取完，其他线程才可使用。不会出现数据不一致或者数据污染。

线程不安全就是不提供数据访问保护，有可能出现多个线程先后更改数据造成所得到的数据是脏数据