综述

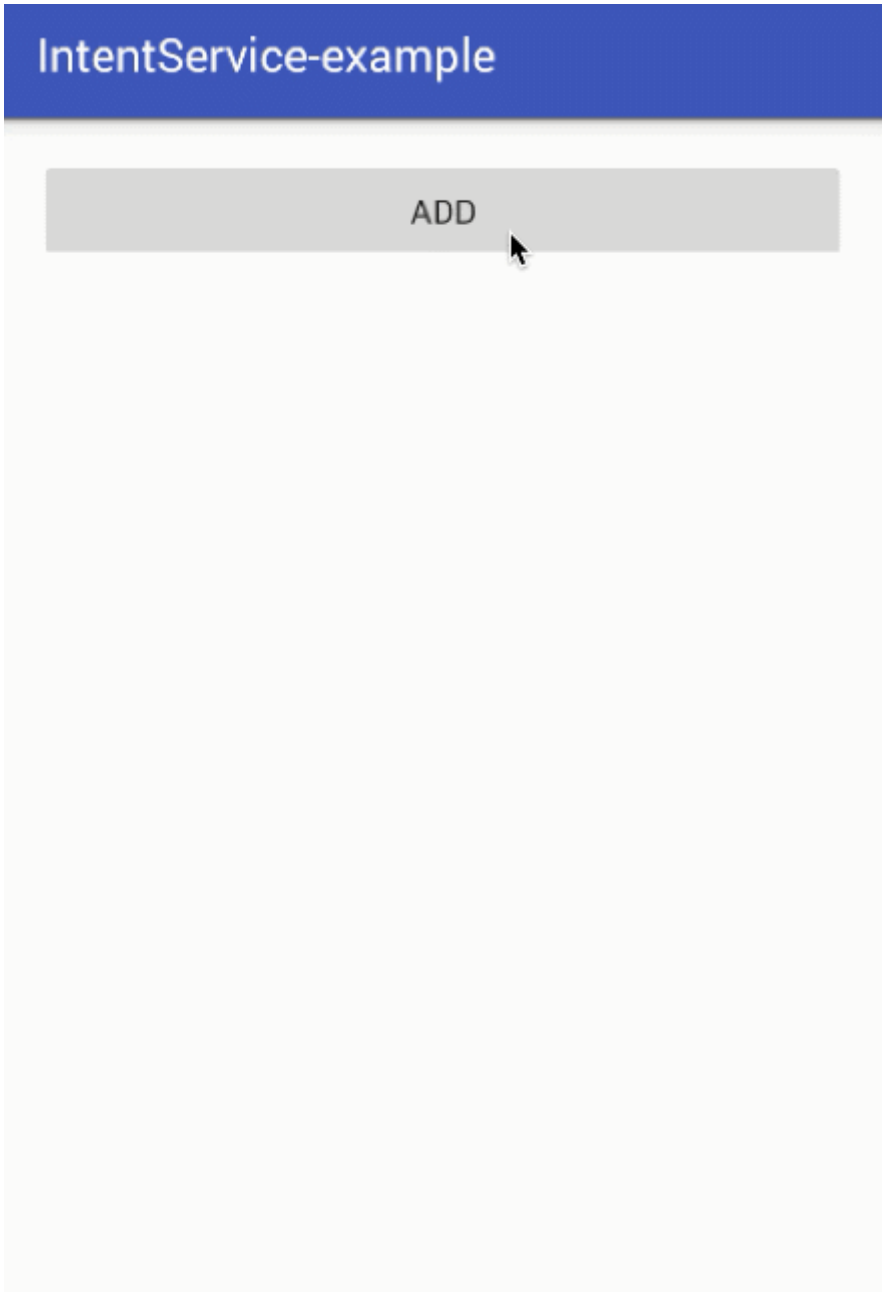
我们都知道Service是作为后台服务运行再程序中的。但是Service他依然是运行在主线程中的，所以我们依然不能在Service中进行耗时的操作。所以当我们在Service处理时，我们需要在Service中开启一个子线程，并且在子线程中运行。当然为了简化我们的操作，在Android中为我们提供了IntentService来进行这一处理，下面我们就来看一下这个IntentService用法以及它的工作原理。

用法简介

IntentService它继承自Service，一来说我们开启一个Service可以通过startService和bindService两个方式进行开启一个服务，但是对于IntentService我们采用startService方法进行开启服务，对于为什么要这么做，在后面会进行分析讲解。下面我们来看一下如何使用这个IntentService的。

效果演示

在这里我们做一个倒计时的程序，以毫秒为单位。这里先看一下效果演示。



代码分析

在这里我们使用到了开源框架EventBus，对于EventBus的使用可以参考这篇文章。由于我们用到这EventBus,首先我们创建一个实体类，在EventBus中进行发送，接收处理。

```
package com.example.ljd.intentservice;

public class Counter {

    public Counter(int progress,int tag) {
        this.progress = progress;
        this.tag = tag;
    }

    public int progress;      //进度显示

    public int tag;           //TextView的Tag
}12345678910111213
```

下面我们看一下IntentService中的代码。

```
package com.example.ljd.intentservice;

import android.app.IntentService;
import android.content.Intent;
import android.content.Context;

import org.greenrobot.eventbus.EventBus;


public class MyIntentService extends IntentService {

    private static final String ACTION_COUNTER =
"com.example.ljd.intentservice.action.COUNTER";

    private static final String EXTRA_SEC =
"com.example.ljd.intentservice.extra.SECOND";

    private static final String EXTRA_TAG =
"com.example.ljd.intentservice.extra.TAG";

    private static final int SLEEP_TIME = 1;

    public MyIntentService() {
        super("MyIntentService");
    }


    public static void startDownload(Context context, int second,int tag) {
        Intent intent = new Intent(context, MyIntentService.class);
        intent.setAction(ACTION_COUNTER);
        intent.putExtra(EXTRA_SEC, second);
        intent.putExtra(EXTRA_TAG,tag);
        context.startService(intent);
    }
```

```
    @Override
    protected void onHandleIntent(Intent intent) {
        if (intent != null) {
            final String action = intent.getAction();
            if (ACTION_COUNTER.equals(action)) {
                final int second = intent.getIntExtra(EXTRA_SEC, 0);
                final int tag = intent.getIntExtra(EXTRA_TAG,0);
                handleActionFoo(second,tag);
            }
        }
    }

    private void handleActionFoo(int sec,int tag) {
        int millis = sec * 1000;
        Counter counter = new Counter(0,tag);

        for (int i = millis;i >= 0; i-=SLEEP_TIME){
            counter.progress = i;
            EventBus.getDefault().post(counter);
            try {
                Thread.sleep(SLEEP_TIME);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

    }

}12345678910111213141516171819202122232425262728293031323334353637383940414243444546474849505152535455565758596061
```

在上面的handleActionFoo方法中进行我们的耗时任务。然后我们在看一下Activity中的代码。

```
package com.example.ljd.intentservice;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.TextView;

import org.greenrobot.eventbus.EventBus;
import org.greenrobot.eventbus.Subscribe;
import org.greenrobot.eventbus.ThreadMode;

import java.util.Random;

public class MainActivity extends AppCompatActivity implements
View.OnClickListener{

    private int mTextViewTag;
    private Button mAddButton;
    private LinearLayout mContainerLinear;


    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        EventBus.getDefault().register(this);
        mTextViewTag = 0;
        mAddButton = (Button) findViewById(R.id.add_btn);
        mContainerLinear = (LinearLayout) findViewById(R.id.linear_container);
        mAddButton.setOnClickListener(this);
    }

    @Subscribe(threadMode = ThreadMode.MAIN)
    public void OnEventProgress(Counter counter){
        TextView textView =
(TextView)mContainerLinear.findViewWithTag(counter.tag);
        textView.setText(counter.progress + "ms");
    }

    @Override
    protected void onDestroy() {
        EventBus.getDefault().unregister(this);
        super.onDestroy();
    }


    @Override
    public void onClick(View v) {
        switch (v.getId()){
            case R.id.add_btn:
                //生成1~3之间的随机数
                Random random = new Random();
                int num = random.nextInt(3)%(3) + 1;

                TextView textView = new TextView(this);
                textView.setTag(mTextViewTag);
                textView.setText(num * 1000 + "ms");
                mContainerLinear.addView(textView);
                MyIntentService.startDownload(this,num,mTextViewTag);
                mTextViewTag++;
                break;
            default:
                break;
        }
    }
}12345678910111213141516171819202122232425262728293031323334353637383940414243444546474849505152535455565758596061626364656
```

最后是布局代码。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
```

```
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.ljd.intentservice.MainActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">
        <Button
            android:id="@+id/add_btn"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="add"/>
    </LinearLayout>

    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent">
        <LinearLayout
            android:id="@+id/linear_container"
            android:orientation="vertical"
            android:layout_width="match_parent"
            android:layout_height="wrap_content">
        </LinearLayout>
    </ScrollView>

</LinearLayout>12345678910111213141516171819202122232425262728293031323334353536
```

对于上面代码实现起来都是非常的简单，在这里就不在进行详细介绍。

IntentService工作原理分析

其实对于IntentService的工作原理也不复杂，既然在IntentService中能够进行耗时操作，也就是说在这个IntentService中必然也创建了一个子线程，在Android中我们称为工作者线程。然后在这个工作者线程中进行我们的任务。在分析IntentService之前，我们先看一下HandlerThread。

HandlerThread

其实HandlerThread就是一个工作者线程，在这里看一下HandlerThread的源码。

```
package android.os;

public class HandlerThread extends Thread {
    int mPriority;
    int mTid = -1;
    Looper mLooper;

    public HandlerThread(String name) {
        super(name);
        mPriority = Process.THREAD_PRIORITY_DEFAULT;
    }

    public HandlerThread(String name, int priority) {
        super(name);
        mPriority = priority;
```

```
    }

    protected void onLooperPrepared() {
    }

    @Override
    public void run() {
        mTid = Process.myTid();
        Looper.prepare();
        synchronized (this) {
            mLooper = Looper.myLooper();
            notifyAll();
        }
        Process.setThreadPriority(mPriority);
        onLooperPrepared();
        Looper.loop();
        mTid = -1;
    }

    public Looper getLooper() {
        if (!isAlive()) {
            return null;
        }

        // If the thread has been started, wait until the looper has been
created.
        synchronized (this) {
            while (isAlive() && mLooper == null) {
                try {
                    wait();
                } catch (InterruptedException e) {
                }
            }
        }
        return mLooper;
    }

    public boolean quit() {
        Looper looper = getLooper();
        if (looper != null) {
            looper.quit();
            return true;
        }
        return false;
    }

    public boolean quitSafely() {
        Looper looper = getLooper();
        if (looper != null) {
            looper.quitSafely();
            return true;
        }
        return false;
    }

    public int getThreadId() {
        return mTid;
    }
```

```
}12345678910111213141516171819202122232425262728293031323334353637383940414243444546474849505152535455565758596061626364656667686970717273
```

看过上篇文章Android的消息机制——Handler的工作过程就很容易理解这个HandlerThread了。还记的我们在上篇文章的最后，新建了一个包含Looper的子线程。而这个HandlerThread也就是一个包含Looper的子线程。所以当我们需要创建一个包含Looper的线程时直接使用HandlerThread即可。对于HandlerThread有以下几点需要说明一下。

1. 在构造方法中设置线程优先级的时候，使用的Process是android.os包中的而不是java.lang包内的。
2. 如果在Looper开启消息循环之前我们进行一些设置，我们可以继承HandlerThread并且重写onLooperPrepared方法。
3. 通过getLooper方法我们获取HandlerThread的Looper对象时，有可能Looper还未创建完成。所以在getLooper中未创建Looper是进行了线程等待操作，在创建完Looper以后在返回Looper对象。

IntentService

下面我们再看一下IntentService。

```
package android.app;

import android.annotation.WorkerThread;
import android.content.Intent;
import android.os.Handler;
import android.os.HandlerThread;
import android.os.IBinder;
import android.os.Looper;
import android.os.Message;

public abstract class IntentService extends Service {
    private volatile Looper mServiceLooper;
    private volatile ServiceHandler mServiceHandler;
    private String mName;
    private boolean mRedelivery;

    private final class ServiceHandler extends Handler {
        public ServiceHandler(Looper looper) {
            super(looper);
        }

        @Override
        public void handleMessage(Message msg) {
            onHandleIntent((Intent)msg.obj);
            stopSelf(msg.arg1);
        }
    }

    public IntentService(String name) {
        super();
        mName = name;
    }

    public void setIntentRedelivery(boolean enabled) {
        mRedelivery = enabled;
    }
```

```
    @Override
    public void onCreate() {
        // TODO: It would be nice to have an option to hold a partial wakelock
        // during processing, and to have a static startService(Context, Intent)
        // method that would launch the service & hand off a wakelock.

        super.onCreate();
        HandlerThread thread = new HandlerThread("IntentService[" + mName +
"]");
        thread.start();

        mServiceLooper = thread.getLooper();
        mServiceHandler = new ServiceHandler(mServiceLooper);
    }

    @Override
    public void onStart(Intent intent, int startId) {
        Message msg = mServiceHandler.obtainMessage();
        msg.arg1 = startId;
        msg.obj = intent;
        mServiceHandler.sendMessage(msg);
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        onStart(intent, startId);
        return mRedelivery ? START_REDELIVER_INTENT : START_NOT_STICKY;
    }

    @Override
    public void onDestroy() {
        mServiceLooper.quit();
    }

    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }

    @WorkerThread
    protected abstract void onHandleIntent(Intent intent);
}12345678910111213141516171819202122232425262728293031323334353637383940414243444546474849505152535455565758596061626364656667686970717273747576778
```

我们看一下这个IntentService的构造是不是很简单。在这里主要看一下onCreate和onStart方法即可。在onCreate中，我们开启了一个HandlerThread线程，之后获取HandlerThread线程中的Looper，并通过这个Looper创建了一个Handler。然后在onStart方法中通过这个Handler将intent与startId作为Message的参数进行发送到消息队列中，然后交由Handler中的handleMessage中进行处理。由于在onStart方法是在主线程内运行的，而Handler是通过工作者线程HandlerThread中的Looper创建的。所以也就是在主线程中发送消息，在工作者接收到消息后便可以进行一些耗时的操作。

我们在看一下handleMessage中的操作，在handleMessage中调用onHandleIntent方法，他是一个抽象方法，所以在我们的Service中复写onHandleIntent方法并且将耗时的操作写在onHandleIntent方法内即可。当执行完onHandleIntent后通过stopSelf来停止服务，这样就不用我们手动停止服务了。所以也就回答了我们上面那个为什么要使用startService而不用onBind来开启一个IntentService。

总结

从我们的示例和源码分析中可以看出来。对于通过IntentService来执行任务，他是串行的。也就是说只有在上一个任务执行完以后才会执行下一个任务。因为Handler中将消息插入消息队列，而队列又是先进先出的数据结构。所以只有在上个任务执行完成以后才能够获取到下一个任务进行操作。在这里也就说明了对于高并发的任务同过IntentService是不合适。