现在假设有个场景，需要你在子线程里面跑个耗时操作，怎么搞? 不BB，上代码，首先定义一个子线程，里面需要有个Handler:

public class MThread extends Thread {

//需要有个Handler

private Handler handler;

@Override

public void run() {

Looper.prepare(); //创建Looper

handler = new Handler(); //创建Handler，这时候Handler会取当前线程的Looper

Looper.loop(); //轮询

}

//对外提供自己的Handler

public Handler getHandler() {

return handler;

}

}

创建好了子线程后，我们开始使用

MThread mThread = new MThread(); //创建线程

mThread.start(); //让线程跑起来

mThread.getHandler().sendMessage(new Message()); //获取这个线程的Handler，并发消息

这个时候我们发现大概率会抛出空指针异常，提示mThread.getHandler()是个null，为什么呢，因为我们在mThread.start()之后，立刻去获取它的Handler，此时可能MThread.run()还没跑完，或者说里面的那句 handler = new Handler()还没执行到，而且这样使用也太麻烦了，so，HandlerThread出场了，我们直接看它的源码:

public class HandlerThread extends Thread {

int mPriority;

int mTid = -1;

Looper mLooper; //提供一个当前线程的Looper

private @Nullable Handler mHandler; //提供一个Handler

public HandlerThread(String name) {

super(name);

mPriority = Process.THREAD\_PRIORITY\_DEFAULT;

}

public HandlerThread(String name, int priority) {

super(name);

mPriority = priority;

}

//这个是在Looper创建好后回调的，可以在这里面直接获取Looper

protected void onLooperPrepared() {

}

@Override

public void run() {

mTid = Process.myTid();

Looper.prepare(); //这里创建了Looper

synchronized (this) { //这里加锁了

mLooper = Looper.myLooper(); //给mLooper赋值

notifyAll(); //唤醒其他等待"this"锁的线程

}

Process.setThreadPriority(mPriority);

onLooperPrepared(); //回调

Looper.loop(); //轮询

mTid = -1;

}

//获取Looper

public Looper getLooper() {

if (!isAlive()) { //如果当前线程已经不存在了，就返回null，变相避免了死锁

return null;

}

synchronized (this) { //加锁，跟创建Looper那里相呼应

while (isAlive() && mLooper == null) { //如果线程还存活并且mLooper为空，也就是说run()里面还没跑完

try {

wait(); //就等着，等待run()里面的那个notifyAll()

} catch (InterruptedException e) {

}

}

}

return mLooper; //返回

}

//获取Handler

public Handler getThreadHandler() {

if (mHandler == null) {

mHandler = new Handler(getLooper()); //这里也调用了getLooper()，所以如果mLooper还没创建完，也会卡在这里

}

return mHandler;

}

//退出，会删除MessageQueue里面的全部消息

public boolean quit() {

Looper looper = getLooper();

if (looper != null) {

looper.quit();

return true;

}

return false;

}

//安全退出，只会删除MessageQueue里面的将来才执行的消息，比如在时间A调用了quitSafely，那么MessageQueue里面执行时间msg.when>=A的才会被回收，其他保留

public boolean quitSafely() {

Looper looper = getLooper();

if (looper != null) {

looper.quitSafely();

return true;

}

return false;

}

public int getThreadId() {

return mTid;

}

}

**问:如果先getLooper()在mThread.start()呢，会死锁吗？比如:**

MThread thread = new MThread();

thread.getLooper(); //这个调完后，他会synchronized(this)持有this锁，并且卡在wait()那里等待notify

thread.start(); //这个调完了，它的run()里面的synchronized(this)进不去，卡在那里等待this锁，这样跟上面就形成了死锁

答: 不会，因为thread.getLooper()的时候，如果线程还没start()，那么isAlive()就是false，他就会立刻返回null，所以不会。

***问: 但是线程具有不确定性，如果我先start()线程，然后等到它的跑到run()里面的synchronized(this)，然后立刻去thread.getLooper()，此时就跳过了if(isAlive())检测 但是此时如果getLooper()里面的synchronized()先获取到了this呢?流程如下:***

public void run() {

mTid = Process.myTid();

Looper.prepare(); // (1先跑到了这里)

synchronized (this) { //(4那么这里跑不到，等着，卡住了)

mLooper = Looper.myLooper(); //给mLooper赋值

notifyAll(); //唤醒其他等待"this"锁的线程

}

Process.setThreadPriority(mPriority);

onLooperPrepared(); //回调

Looper.loop(); //轮询

mTid = -1;

}

//获取Looper

public Looper getLooper() {

if (!isAlive()) { //(2跑到了这里)

return null;

}

synchronized (this) { //(3 又跑到了这里，先获取到this锁)

while (isAlive() && mLooper == null) { //(5跑到这了，俩条件符合，等)

try {

wait(); (6)

} catch (InterruptedException e) {

}

}

}

return mLooper;

}

答:不会，因为上面代码还有个(6)，6是干啥的？等待，对，但是除了等待，wait()还会释放锁！！，所以一旦wait()，那么run()里面就继续执行了，所以不存在死锁，死锁发生的前提是:大于等于两把锁！！