

昵称: [樱桃小锤子](#)
园龄: [3年5个月](#)
粉丝: [23](#)
关注: [2](#)
[+加关注](#)

<	2011年8月						>
日	一	二	三	四	五	六	
31	1	2	3	4	5	6	
7	8	9	10	11	12	13	
14	15	16	17	18	19	20	
21	22	23	24	25	26	27	
28	29	30	31	1	2	3	
4	5	6	7	8	9	10	

搜索

找找看

谷歌搜索

常用链接

- [我的随笔](#)
- [我的评论](#)
- [我的参与](#)
- [最新评论](#)
- [我的标签](#)

我的标签

- [Android\(6\)](#)
- [C++\(5\)](#)
- [Qt\(5\)](#)
- [科普\(3\)](#)
- [NDK\(2\)](#)
- [Python\(2\)](#)

随笔档案(11)

- [2013年12月 \(1\)](#)
- [2013年4月 \(2\)](#)
- [2013年3月 \(1\)](#)
- [2012年12月 \(2\)](#)
- [2011年10月 \(1\)](#)
- [2011年8月 \(4\)](#)

最新评论

Qt那点事儿（一）

第一回 **Signal**和**Slot**是同步的还是异步的？

我们知道Qt以他的signal和slot机制独步天下。但大家在用的时候有没有注意过，signal和slot之间是异步的，还是同步的呢？为此我问过不少使用Qt的道友。有人说是同步的，有人说是异步的，也有人说要看当时你的人品。:(# \$ % ^ & *

为此贫道，特别做了以下几个测试：

First,在main()主函数里，设置两个基于QObject为父类的对象a和b，a触发signal,b接受signal。请看具体案例：



```
1 class MyTestA : public QObject
2 {
3     Q_OBJECT
4 public:
5     void emitSignal()
6     {
7         signalMyTestA();
8     }
9
10 public slots:
11     void slotMyTestA()
12     {
13         qDebug() << "slotMyTestA is called.";
14     }
15 signals:
16     void signalMyTestA();
17 };
18
19 class MyTestB : public QObject
20 {
21     Q_OBJECT
22 public slots:
23     void slotMyTestB()
24     {
25         qDebug() << "slotMyTestB is called.";
26     }
```

1. Re:Qt那点事儿（一）
整个求知的过程值得我们学习
--Tony.Works

阅读排行榜

- 1. QwebKit使用心得(4944)
- 2. Qt那点事儿（三） 论父对象与子对象的关系(4700)
- 3. Qt那点事儿（一）(3204)
- 4. Qt那点事儿（二）(2416)
- 5. 傲娇Android二三事之天不长地不久的Bitmap.compress(1963)

评论排行榜

- 1. 傲娇Android二三事之天不长地不久的Bitmap.compress(10)
- 2. Qt那点事儿（一）(9)
- 3. Qt那点事儿（二）(8)
- 4. 傲娇Android二三事之操蛋的开发日记(第一回)(8)
- 5. NDK开发笔记(一) NDK的安装(7)

推荐排行榜

- 1. NDK开发笔记(一) NDK的安装(5)
- 2. 傲娇Android二三事之诡谲异异的图片加载(5)
- 3. Qt那点事儿（一）(4)
- 4. 傲娇Android二三事之天不长地不久的Bitmap.compress(3)
- 5. 傲娇Android二三事之操蛋的开发日记(第一回)(2)

```
27 signals:
28     void signalMyTestB();
29 };
30
31 int main(int argc, char *argv[])
32 {
33     QApplication app(argc, argv);
34
35     MyTestA a;
36     MyTestB b;
37     QObject::connect(&a,SIGNAL(signalMyTestA()),&b,SLOT(slotMyTestB()));
38
39     a.emitSignal();
40
41     return app.exec();
42 }
```



在slotMyTestB的函数里打个断点，看一下调用堆栈(call stack)。

是同步调用的，某些道友开始拈胡微笑，实践出真知啊。

	Name
	MyTest.exe!MyTestB::slotMyTestB() Line 34
	MyTest.exe!MyTestB::qt_metacall(QMetaObject::Call _c=
	QtCored4.dll!QMetaObject::metacall(QObject * object=0:
	QtCored4.dll!QMetaObject::activate(QObject * sender=0
	MyTest.exe!MyTestA::signalMyTestA() Line 87 + 0x13 b
	MyTest.exe!MyTestA::emitSignal() Line 17


此时只见东方黑云滚滚，电闪雷鸣，又有道友开始度劫了。突然一度劫道友横眉冷对，拿起拂尘刷刷的改写了上面的代码。只见此道友把a对象挪到了一个新线程中（MyTestC创建的），而b对象仍然在主线程中。然后a对象触发信号。

```
1 class MyTestA : public QObject
2 {
3     Q_OBJECT
4 public:
5     void emitSignal()
6     {
7         signalMyTestA();
8     }
9
10 public slots:
11     void slotMyTestA()
12     {
13         qDebug()<<"slotMyTestA is called.";
14     }
15 signals:
16     void signalMyTestA();
17 };
18
19 class MyTestB : public QObject
20 {
21     Q_OBJECT
22 public slots:
```

[illegible]

```
73     int main(int argc, char *argv[])
74     {
75         QApplication app(argc, argv);
76
77
78         MyTestB b;
79         g_pMyTestB = &b;
80         MyTestC c;
81
82         c.start();
83
84         return app.exec();
85     }
```

说时迟，那时快。在一道紫雷劈下之际，按下了F5。只见，此时的调用堆栈显示，

Name
 MyTest.exe!MyTestB::slotMyTestB() Line 34
MyTest.exe!MyTestB::qt_metacall(QMetaObject::Call _c=InvokeMetaMethod, int _id=1, void * *_a=0x0058f390) Line 148
QtCored4.dll!QMetaObject::metacall(QObject * object=0x003afe34, QMetaObject::Call cl=InvokeMetaMethod, int idx=5, void * *)
QtCored4.dll!QMetaCallEvent::placeMetaCall(QObject * object=0x003afe34) Line 535 + 0x19 bytes
QtCored4.dll!QObject::event(QEvent * e=0x0058f3d0) Line 1217 + 0x14 bytes
QtGuid4.dll!QApplicationPrivate::notify_helper(QObject * receiver=0x003afe34, QEvent * e=0x0058f3d0) Line 4462 + 0x11 bytes
QtGuid4.dll!QApplication::notify(QObject * receiver=0x003afe34, QEvent * e=0x0058f3d0) Line 3862 + 0x10 bytes
QtCored4.dll!QCoreApplication::notifyInternal(QObject * receiver=0x003afe34, QEvent * event=0x0058f3d0) Line 731 + 0x15 bytes
QtCored4.dll!QCoreApplication::sendEvent(QObject * receiver=0x003afe34, QEvent * event=0x0058f3d0) Line 215 + 0x39 bytes
QtCored4.dll!QCoreApplicationPrivate::sendPostedEvents(QObject * receiver=0x00000000, int event_type=0, QThreadData * data)
QtCored4.dll!qt_internal_proc(HWND__ * hwnd=0x00041e2a, unsigned int message=1025, unsigned int wp=0, long lp=0) Line 49
user32.dll!767e62fa()
[Frames below may be incorrect and/or missing, no symbols loaded for user32.dll]
user32.dll!767e6d3a()
user32.dll!767e6ce9()
user32.dll!767e77c4()
user32.dll!767e788a()
QtCored4.dll!QEventDispatcherWin32::processEvents(QFlags<enum QEventLoop::ProcessEventsFlag> flags={...}) Line 813
QtGuid4.dll!QGuiEventDispatcherWin32::processEvents(QFlags<enum QEventLoop::ProcessEventsFlag> flags={...}) Line 1170 +
QtCored4.dll!QEventLoop::processEvents(QFlags<enum QEventLoop::ProcessEventsFlag> flags={...}) Line 150
QtCored4.dll!QEventLoop::exec(QFlags<enum QEventLoop::ProcessEventsFlag> flags={...}) Line 201 + 0x2d bytes
QtCored4.dll!QCoreApplication::exec() Line 1008 + 0x15 bytes
QtGuid4.dll!QApplication::exec() Line 3737
MyTest.exe!main(int argc=1, char * * argv=0x00586490) Line 20 + 0x6 bytes
MyTest.exe!WinMain(HINSTANCE__ * instance=0x012f0000, HINSTANCE__ * prevInstance=0x00000000, char * __formal=0x004

奇迹出现了，居然变成异步调用了。只见东方天空万道金光射下，在阵阵仙乐声中，传来朗朗之声："贫道尘事已了，再无牵挂"。

难道Qt真的是靠人品的，或者Qt莫不是也是修仙道友，不日也将飞升。

在吾等众人膜拜加疑惑之时，只见飞升前辈，留下一条偈语。内事不决问百度，外事不决问谷歌。

吾等众人立刻搜寻，恍然大悟。

原来**signal**和**slot**是异步调用还是同步调用，取决于对**connect**的设定。其实**connect**还有一个参数(**Qt::ConnectionType**),是它决定了是同步还是异步。以下是ConnectionType的定义

Constant	Value	Description
Qt::AutoConnection	0	(default) If the signal is emitted from a different thread than the receiving object, the signal is queued, behaving as Qt::QueuedConnection. Otherwise, the slot is invoked directly, behaving as Qt::DirectConnection. The type of connection is determined when the signal is emitted.
Qt::DirectConnection	1	The slot is invoked immediately, when the signal is emitted.
Qt::QueuedConnection	2	The slot is invoked when control returns to the event loop of the receiver's thread. The slot is executed in the receiver's thread.
Qt::BlockingQueuedConnection	4	Same as QueuedConnection, except the current thread blocks until the slot returns. This connection type should only be used where the emitter and receiver are in different threads. Note: Violating this rule can cause your application to deadlock.
Qt::UniqueConnection	0x80	Same as AutoConnection, but the connection is made only if it does not duplicate an existing connection. i.e., if the same signal is already connected to the same slot for the same pair of objects, then the connection will fail. This connection type was introduced in Qt 4.6.
Qt::AutoCompatConnection	3	The default type when Qt 3 support is enabled. Same as AutoConnection but will also cause warnings to be output in certain situations. See Compatibility Signals and Slots for further information.

只不过，平常它有一个默认值Qt::AutoConnection,我们忽略了它。这时有道友问道，为何在AutoConnection模式下，有时是同步，有时是异步，莫非Auto就是人品代名词。

非也，其实Auto是这样规定的，

当**sender**和**receiver**在同一线程时，就是同步模式，而在不同线程时，则是异步模式。

众人皆曰善。

就在众人弹冠相庆之时，突然一道类似眼镜发出的寒光闪过，一个黑影渐渐清晰了起来。

他居然就是.....

青春永驻，十二年如一日的柯南君，他招牌式的磁性声音给众道友一晴天霹雳，“诸位以为这就是全部的真相吗？”

接着他刷刷的又改写了代码，在主线程中生成a,b两个对象，而a对象在新线程（MyTestC创建的）中触发信号。


```
1  class MyTestA : public QObject
2  {
3      Q_OBJECT
4  public:
5      void emitSignal()
6      {
7          signalMyTestA();
8      }
9
10 public slots:
11     void slotMyTestA()
12     {
13         qDebug()<<"slotMyTestA is called.";
14     }
15 signals:
16     void signalMyTestA();
17 };
18
19 class MyTestB : public QObject
20 {
21     Q_OBJECT
22 public slots:
23     void slotMyTestB()
24     {
25         qDebug()<<"slotMyTestB is called.";
26     }
27 signals:
```

```

28     void signalMyTestB();
29 };
30
31 extern MyTestB *g_pMyTestB;
32 extern MyTestA *g_pMyTestA;
33 class MyTestC : public QThread
34 {
35     Q_OBJECT
36 public:
37
38     void run()
39     {
40         g_pMyTestA->emitSignal();
41         exec();
42
43
44     }
45
46 public slots:
47     void slotMyTestC()
48     {
49         qDebug()<<"slotMyTestC is called.";
50     }
51 signals:
52     void signalMyTestC();
53
54
55 };
56 ///////////////////////////////////////////////////
57 MyTestB *g_pMyTestB = NULL;
58 MyTestA *g_pMyTestA = NULL;
59 int main(int argc, char *argv[])
60 {
61     QApplication app(argc, argv);
62
63     MyTestA a;
64     g_pMyTestA = &a;
65     MyTestB b;
66     g_pMyTestB = &b;
67
68     QObject::connect(&a,SIGNAL(signalMyTestA()),&b,SLOT(slotMyTestB()));
69
70     MyTestC c;
71
72     c.start();
73
74     return app.exec();
75 }

```

在众人疑惑的眼光中，此君淡定的按下了F5。只见调用堆栈(call stack)显示

	Name
	MyTest.exe!MyTestB::slotMyTestB() Line 34
	MyTest.exe!MyTestB::qt_metacall(QMetaObject::Call _c=InvokeMetaMethod, int _id=1, void ** _a=0x0058f390) Line 148
	QtCored4.dll!QMetaObject::metacall(QObject * object=0x003afe34, QMetaObject::Call cl=InvokeMetaMethod, int idx=5, void **
	QtCored4.dll!QMetaCallEvent::placeMetaCall(QObject * object=0x003afe34) Line 535 + 0x19 bytes
	QtCored4.dll!QObject::event(QEvent * e=0x0058f3d0) Line 1217 + 0x14 bytes
	QtGuid4.dll!QApplicationPrivate::notify_helper(QObject * receiver=0x003afe34, QEvent * e=0x0058f3d0) Line 4462 + 0x11 bytes
	QtGuid4.dll!QApplication::notify(QObject * receiver=0x003afe34, QEvent * e=0x0058f3d0) Line 3862 + 0x10 bytes
	QtCored4.dll!QCoreApplication::notifyInternal(QObject * receiver=0x003afe34, QEvent * event=0x0058f3d0) Line 731 + 0x15 bytes
	QtCored4.dll!QCoreApplication::sendEvent(QObject * receiver=0x003afe34, QEvent * event=0x0058f3d0) Line 215 + 0x39 bytes
	QtCored4.dll!QCoreApplicationPrivate::sendPostedEvents(QObject * receiver=0x00000000, int event_type=0, QThreadData * da
	QtCored4.dll!qt_internal_proc(HWND__ * hwnd=0x00041e2a, unsigned int message=1025, unsigned int wp=0, long lp=0) Line 49
	user32.dll!767e62fa()
	[Frames below may be incorrect and/or missing, no symbols loaded for user32.dll]
	user32.dll!767e6d3a()
	user32.dll!767e6ce9()
	user32.dll!767e77c4()
	user32.dll!767e788a()
	QtCored4.dll!QEventDispatcherWin32::processEvents(QFlags<enum QEventLoop::ProcessEventsFlag> flags={...}) Line 813
	QtGuid4.dll!QGuiEventDispatcherWin32::processEvents(QFlags<enum QEventLoop::ProcessEventsFlag> flags={...}) Line 1170 +
	QtCored4.dll!QEventLoop::processEvents(QFlags<enum QEventLoop::ProcessEventsFlag> flags={...}) Line 150
	QtCored4.dll!QEventLoop::exec(QFlags<enum QEventLoop::ProcessEventsFlag> flags={...}) Line 201 + 0x2d bytes
	QtCored4.dll!QCoreApplication::exec() Line 1008 + 0x15 bytes
	QtGuid4.dll!QApplication::exec() Line 3737
	MyTest.exe!main(int argc=1, char ** argv=0x00586490) Line 20 + 0x6 bytes
	MyTest.exe!WinMain(HINSTANCE__ * instance=0x012f0000, HINSTANCE__ * prevInstance=0x00000000, char * __formal=0x004

众人皆惊呼，“Impossible”。a和b明明是属于一个线程的，为何会异步调用。此时我们熟悉的语录，又在耳边回响,是"我相信真相只有一个！！!"这句话吗？No，只见柯南君，优雅地挥了挥手，"Nothing impossible",从口中缓缓滑出。

。众人皆扑街，“有屁快放”。

此时柯南君缓缓从口袋中，摸出一张纸，抛向空中，然后转身离去。只见随风飘落的纸面上面摘录了这么一段Qt源代码，在Auto模式下，如果要同步调用，不仅要求sender和receiver是同一线程，而且sender触发的时候，所在的线程也要和receiver一致。

```
1 // determine if this connection should be sent immediately or
2 // put into the event queue
3 if ((c->connectionType == Qt::AutoConnection
4      && (currentThreadData != sender->d_func()->threadData
5         || receiver->d_func()->threadData != sender->d_func()->threadData))
6      || (c->connectionType == Qt::QueuedConnection)) {
7     queued_activate(sender, signal_absolute_index, c, argv ? argv : empty_argv);
8     continue;
9 } else if (c->connectionType == Qt::BlockingQueuedConnection) {
10     blocking_activate(sender, signal_absolute_index, c, argv ? argv : empty_argv);
11     continue;
12 }<br><br>摘自qobject.cpp
```

z众人皆惊，原来在Auto模式下，如果sender的触发时所处的线程和receiver不同，也会是异步调用。此时道友齐声向柯南喊道“这是全部的真相了吗”？柯南转过头来笑而不语，渐渐又消失在黑暗中。“有多少无耻可以重来”漂上了众人的心头。望着远处的雨后阳光，一个大大的问号也出现在众人头顶,"Qt你到底有多无耻？？？"。众人又陷入了沉思。

欲知后事如何，请听下回分解。

此篇已在CSDN同时发布

标签: C++, Qt

绿色通道:

好文要顶

关注我

收藏该文

与我联系





樱桃小锤子
关注 - 2
粉丝 - 23
[+加关注](#)

« 上一篇: QwebKit使用心得

» 下一篇: Qt那点事儿 (二)

posted on 2011-08-03 12:38 樱桃小锤子 阅读(3204) 评论(9) 编辑 收藏

评论:

#1楼 2011-08-03 12:55 | conarena

园子里 也来QT大神了啊

支持(0) 反对(0)

#2楼[楼主] 2011-08-03 13:08 | 樱桃小锤子

@conarena

大神不敢当，只是把平时开发时碰到的一些问题和心得共大家分享，共同提高

支持(0) 反对(0)

#3楼 2011-08-03 13:10 | bestcomy

还真有玩这玩意儿的,虽然还未入道,但是帮顶!

支持(0) 反对(0)

#4楼[楼主] 2011-08-03 13:24 | 樱桃小锤子

@bestcomy

谢谢，其实，现在在Mac和Linux上用它的人满多的

支持(0) 反对(0)

#5楼 2011-08-03 13:46 | 微加幸福

引用

bestcomy: 还真有玩这玩意儿的,虽然还未入道,但是帮顶!

支持(0) 反对(0)

#6楼 2011-08-04 10:07 | yofly

哈哈，博主看来尘缘未了啊。

支持(0) 反对(0)

#7楼 2011-09-02 22:38 | Morya
奉献几分香火之力
顶

支持(0) 反对(0)

#8楼 2012-05-15 15:41 | Ranger98
楼主太强了，喜欢这种本土化的讲解方法，有兴趣出书的话我一定捧场

支持(0) 反对(0)

#9楼 2014-06-04 20:27 | Tony.Works
整个求知的过程值得我们学习

支持(0) 反对(0)

[刷新评论](#) [刷新页面](#) [返回顶部](#)

注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问](#)网站首页。

- 【[免费课程](#)】[移动端开发框架入门](#)
- 【[推荐](#)】[50万行VC++源码：大型组态工控、电力仿真CAD与GIS源码库](#)
- [融云](#)，免费为你的App加入IM功能——让你的App“聊”起来！！



- 最新**IT**新闻：
- [刷大墙/送财神/送春联 网贷平台进军农村也是拼了！](#)
 - [阿里与工商总局掐架 受伤的是美国股民](#)
 - [为啥手机厂商一窝蜂地烧Hi-Fi，做耳机？](#)
 - [你在淘宝上的买买买，能让阿里巴巴给你的信用打几分呢？](#)
 - [亚马逊将分拆AWS云计算服务](#)
- » [更多新闻...](#)



- 最新知识库文章：
- [大数据架构和模式（五）——对大数据问题应用解决方案模式并选择实现它的产品](#)
 - [大数据架构和模式（四）——了解用于大数据解决方案的原子模式和复合模式](#)
 - [大数据架构和模式（三）——理解大数据解决方案的架构层](#)
 - [大数据架构和模式（二）——如何知道一个大数据解决方案是否适合您的组织](#)

· [大数据架构和模式（一）——大数据分类和架构简介](#)
» [更多知识库文章...](#)