# QT基础面试题

**1:QT信号槽机制的优缺点 :**

1）QT信号槽机制的引用精简了程序员的代码量 (不用写回调函数)  
2）QT的信号可以对应多个槽（但他们的调用顺序随机），也可以多个槽映射一个信号   
3）QT的信号槽的建立和解除绑定十分自由   
4）信号槽同真正的回调函数比起来时间的耗损还是很大的，所有在嵌入式实时系统中应当慎用   
5）信号槽的参数限定很多例如不能携带模板类参数，不能出现宏定义等等

**2、多线程情况下, Qt中的信号槽分别在什么线程中执行, 如何控制?**

可以通过connect函数的第五个参数来控制, 信号槽执行时所在的线程

1)**自动连接**(AutoConnection)，默认的连接方式，如果信号与槽，也就是**发送者与接受者在同一线程**，等同于直接连接；如果发送者与接受者处在不同线程，等同于队列连接。  
2)**直接连接**(DirectConnection)，当信号发射时，槽函数立即直接调用。无论槽函数所属对象在哪个线程，槽函数总在发送者所在线程执行，即**槽函数和信号发送者在同一线程**  
3)**队列连接**(QueuedConnection)，当控制权回到接受者所在线程的事件循环时，槽函数被调用。槽函数在接受者所在线程执行，即**槽函数与信号接受者在同一线程**

**3、描述QT中的文件流(QTextStream)和数据流(QDataStream)的区别, 他们都能帮助我们完成一些什么事情.**

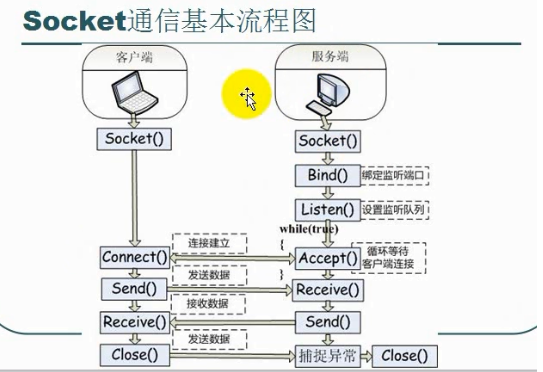
QTextStream – 文本流, 操作轻量级数据(int, double, QString), 数据写入文件中之后以文本的方式呈现。   
QDataStream – 数据流, 通过数据流可以操作各种数据类型, 包括类对象, 存储到文件中数据可以还原到内存。   
QTextStream, QDataStream可以操作磁盘文件, 也可以操作内存数据, 通过流对象可以将数据打包到内存, 进行数据的传输.

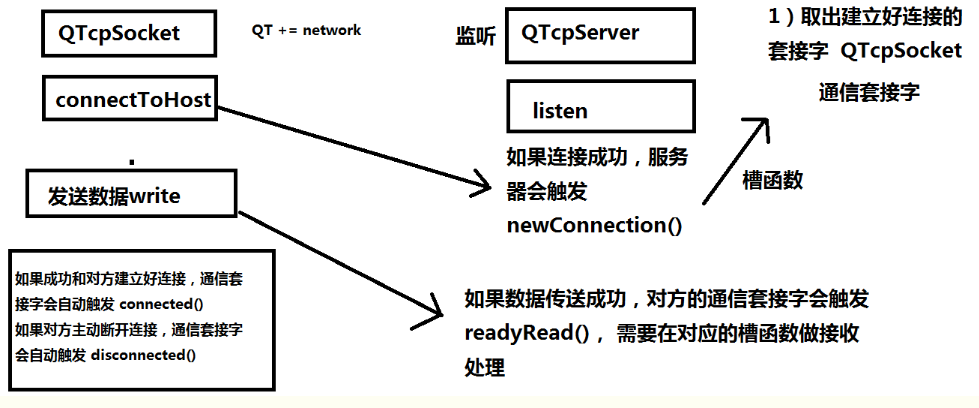
**4、描述Qt下Tcp通信的整个流程**

[具体解释](https://www.cnblogs.com/yuweifeng/p/9382841.html) ：<https://www.cnblogs.com/yuweifeng/p/9382841.html>

QT如果要进行网络编程首先需要在.pro中添加如下代码：QT += network

服务器端:   
1. 创建用于监听的套接字   
2. 给套接字设置监听   
3. 如果有连接到来, 监听的套接字会发出信号newConnected   
4. 接收连接, 通过nextPendingConnection()函数, 返回一个QTcpSocket类型的套接字对象(用于通信)   
5. 使用用于通信的套接字对象通信   
1>. 发送数据: write   
2>. 接收数据: readAll/read   
客户端:   
1. 创建用于通信的套接字   
2. 连接服务器: connectToHost   
3. 连接成功与服务器通信   
1>. 发送数据: write   
2>. 接收数据: readAll/read





**6、描述QT下udp通信的整个流程**

QT下udp通信服务器端和客户端的关系是对等的, 做的处理也是一样的.   
1. 创建套接字对象   
2. 如果需要接收数据, 必须绑定端口   
3. 发送数据: writeDatagram   
4. 接收数据: readDatagram

**7： 描述QT下多线程的两种使用方法, 以及注意事项**

第一种方法:   
1. 创建一个类从QThread类派生   
2. 在子线程类中重写 run 函数, 将处理操作写入该函数中   
3. 在主线程中创建子线程对象, 启动子线程, 调用start()函数   
第二种方法:   
1. 将业务处理抽象成一个业务类, 在该类中创建一个业务处理函数   
2. 在主线程中创建一QThread类对象   
3. 在主线程中创建一个业务类对象   
4. 将业务类对象移动到子线程中   
5. 在主线程中启动子线程   
6. 通过信号槽的方式, 执行业务类中的业务处理函数   
多线程使用注意事项:   
\* 1. 业务对象, 构造的时候不能指定父对象   
\* 2. 子线程中不能处理ui窗口(ui相关的类)   
\* 3. 子线程中只能处理一些数据相关的操作, 不能涉及窗口

**2. 新建Ｑt 项目**

　　具体过程见我的另一篇随笔，如何新建ＱＴ项目。

　　有两种创建图形化程序界面的方法，一、通过系统自动生成的.ui文件，二、通过代码实现。

　　方法一可通过鼠标拖拉布局界面，较方便，但在设计较复杂的界面时最好还是使用Ｑt提供的布局管理器。

**４、ＱＴ项目框架分析**

**４.1** 每一个工程都有一个执行的入口函数，此项目中的main.cpp中的 main()函数就是此工程的入口。

main.cpp

[复制代码](javascript:void(0);)

#include "dialog.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

Dialog w;

w.show();

return a.exec();

}

[复制代码](javascript:void(0);)

　　如上所示，其中：

　　１) #include "dialog.h"　包含了程序中要完成功能的Dialog类的定义，在Dialog类中封装完成所需要的功能。

　　　　注意：程序中使用哪个类，就要引用包含该类定义的头文件。如　dialog.h中使用QLabel、QLineEdit、QPushButton必须包含头文件

　　　　#include <QLabel>　#include <QLineEdit>　#include <QPushButton>

２) #include <QApplication>：Application类的定义。

　  ３)  int main(int argc, char \*argv[]):　应用程序的入口，几乎在所有使用Ｑt的情况下，main()函数只需要在将控制权交给Ｑt库之前执行初始化，然后Ｑt 库通过事件向程序告知用户的行为。所有的Ｑt程序中都　必须有且只有一个main()函数。

　  ４) QApplication a(argc, argv): a 是这个程序的　QApplication对象。

*在每一个使用Ｑt图形化应用程序中都必须使用一个QApplication 对象。ＱApplication管理了各种和样的图形化应用程序的广泛资源、基本设置、控制流及事件处理等。*

在任何的窗口部件被使用之前必须创建QApplication 对象。它在这里被创建并且处理这些命令行变量。所有被Qt识别的命令行参数都将从argv中被移去（并且 argc 也因此而减少）。

　　５) w.show()：当创建一个窗口部件的时候，默认它是不可见的，必须调用show()函数使它变为可见。、

　　６) return a.exec()：程序进入消息循环，等待可能的输入进行响应。这里就是main()函数将控制权交给Qt, Ｑt完成事件处理工作，当应用程序退出的时候，exec()函数的值就会返回。在exec()函数中，Qt接收并处理用户和系统的事件并且将它们传递给适当的窗口部件。

**4.2  dialog.h 头文件**

在类Dialog中的定义中，Q\_OBJECT宏的作用是启动Ｑt元对象系统的一些特性（如支持信号和槽等），它必须放到类定义的私有区。

private slots:　表示下面的函数为槽函数。

dialog.h

[复制代码](javascript:void(0);)

#ifndef DIALOG\_H

#define DIALOG\_H

#include <QDialog>

#include <QLabel>

#include <QLineEdit>

#include <QPushButton>

#include <QGridLayout>

const static double PI = 3.1416;

class Dialog : public QDialog

{

Q\_OBJECT

public:

Dialog(QWidget \*parent = 0);

~Dialog();

private:

QLabel \*label1, \*label2;

QLineEdit \*lineEdit;

QPushButton \*button;

private slots:

void showArea();

};

#endif // DIALOG\_H

[复制代码](javascript:void(0);)

**４.3 dialog.cpp**

　　在该文件中初始化部件，创建布局管理器，将部件加到布局管理器中，并且连接

[复制代码](javascript:void(0);)

#include "dialog.h"

Dialog::Dialog(QWidget \*parent)

: QDialog(parent)

{  
　　//初始化部件

label1 = new QLabel(this);

label1->setText(tr("请输入圆的半径："));

lineEdit = new QLineEdit(this);

label2 = new QLabel(this);

button = new QPushButton(this);

button->setText(tr("显示对应的圆的面积"));  
　　//创建布局管理器，将部件加到布局管理器中

QGridLayout \*mainLayout = new QGridLayout(this);

mainLayout->addWidget(label1,0,0);

mainLayout->addWidget(lineEdit,0,1);

mainLayout->addWidget(label2,1,0);

mainLayout->addWidget(button,1,1);  
　　// 连接信号与槽  
　　 //分别是 点击和文本内容改变信号

connect(button,SIGNAL(clicked()),this,SLOT(showArea()));

connect(lineEdit,SIGNAL(textChanged(QString)),this,SLOT(showArea()));

}

Dialog::~Dialog()

{

delete label1,label2,lineEdit,button;

}

//定义槽函数

void Dialog::showArea()

{

bool ok;

QString tempStr;

QString valueStr = lineEdit->text();

int valueInt = valueStr.toInt(&ok);

double area = valueInt \* valueInt \* PI;

label2->setText(tempStr.setNum(area));

}

[复制代码](javascript:void(0);)

**４.4** 上面三部分为这个工程代码的分析，介绍了主程序中完成图形化界面程序所需的编写的代码，即窗口部件的初始化，以及窗口的使用。dialog.h dialog.cpp主要告诉如何自定义一个对话框Ｄialog类（继承自ＱDialog)即定义窗口部件，以及如何将窗口部件的信号与处理事件的槽函数进行绑定。

　　通过上面的分析我们可以创建一个简单的计算圆面积的图形应用程序。

**５、概念解析**

**５.1　信号和槽机制（Signal & Slot）**

 　　Qt提供了信号和槽机制用于完成界面操作的响应，是完成任意两个Ｑt对象之间的通信机制。其中，信号会在某个特定的情况或动作下被触发，槽是等同于接收并处理信号的函数。

　　每个Ｑt对象都包含若干个预定义的信号和若干个预定义的槽，当某一个特定的事件发生时，一个信号被发送，与信号相关的槽则会响应信号并完成相应的处理。当一个类被继承时，该类的信号和槽也同时被继承，也可以根据需要自定义信号和槽。

**１. 信号与槽的连接方式**

　　　　１）一个信号可以与另一个信号相连

　　　　connect(Object1,SIGNAL(signal1),Object2,SLOT(signal2));  
　　　　　表示Object1的信号１可以触发Object2的信号2的发送。  
　　　２）一个信号可以与多个槽相连　  
　　　　connect(Object1,SIGNAL(signal1),Object2,SLOT(slot2));  
　　　　connect(Object1,SIGNAL(signal1),Object3,SLOT(slot3));  
　　　3）一个槽可以响应多个信号  
　　　　connect(Object1,SIGNAL(signal1),Object2,SLOT(slot2));  
　　　　connect(Object3,SIGNAL(signal3),Object2,SLOT(slot2));  
　　　常用的连接方式为  
　　　　connect(Object1,SIGNAL(signal),Object2,SLOT(slot));  
　　　　其中，signal为Object1的信号，slot为Object2的槽  
　　　SIGNAL()和SLOT()是Ｑt定义的两个宏，它们返回其参数的Ｃ语言风格的字符串(const char\*)。如下所示，两者等同。  
　　　　connect(Object1,SIGNAL(signal),Object2,SLOT(slot));  
　　　　connect(Object1,“signal”,Object2,“slot”);  
　　**２.信号与槽机制的优点**

**１）类型安全。**需要关联的信号和槽的签名必须是等同的，即信号的参数类型和参数个数与接收该信号的槽的参数类型和参数个数相同。  
　　　　　　但是槽的参数个数可以少于信号的参数个数，但缺少的参数必须是信号参数的最后一个或几个参数。如信号和槽的参数签名不符，编译器就会报错。  
　　　　**２）松散耦合。**信号与槽机制减弱了Ｑt对象的耦合度。  
　　　　　　激发信号的Ｑt对象无须知道是哪个对象的槽接收它发出的信号，它只需在适当的时候发送适当的信号即可，它不需要关心它发出的信号有没有被接收到，以及哪个对象的哪个槽接收到该信号。  
　　　　　　对象的槽也不需要知道哪些信号关联了自己，而一旦关联信号和槽，Ｑt就保证了适合的槽得到了调用。即使关联的对象在运行时被删除程序也不会崩溃。  
　　　　注意：一个类若要支持信号和槽，就必须从QObject或QObject的子类继承。Ｑt的信号和槽机制不支持对模板的使用。  
　　　　**３）信号与槽机制的效率**  
　　　　　　信号与槽机制增强了对象间通信的灵活性，然而损失发一些性能。同回调函数相比，信号和槽机制运行速度有些慢。通常，传递一个信号来调用槽函数将会比直接调用非虚函数运行速度慢10倍。原因如下：  
　　　　　　a. 需要定位接收信号的对象。  
　　　　　　b. 安全地遍历所有的关联（如一个信号关联多个槽的情况）。  
　　　　　　c. 编组（marshal）／解组（unmarshal）传递的参数。  
　　　　　　d. 多线程的时候，信号可能需要排队等待。  
　　　　　然而，与创建对象的new操作和删除对象的delete操作相比，信号和槽的运行代价只是它们很少的一部分。信号和槽机制导致的这点性能损耗，对实时应用程序是可以忽略的；同信号和槽提供的灵活性和简便性相比，这点性能损耗是值得的。  
  
**５.２　Qt5元对象系统**  
　　Ｑt 元对象系统提供了对象间的通信机制（信号和槽）、运行时类型信息和动态属性系统的支持，是标准Ｃ＋＋的一个，它使Ｑt能够更好地实现ＧＵＩ图形用户界面编程。  
　　Ｑt的元对象系统不支持Ｃ＋＋模板，尽管模板扩展了标准Ｃ＋＋的功能，但是元对象系统提供了模板无法提供的一些特性。  
　　**Ｑt元对象系统基于以下三个事实。**  
　　**１）基类QObject:** 任何需要使用元对象系统功能的类必须继承自QObject。  
　　**２）Q\_OBJECT宏：**Q\_OBJECT宏必须出现在类的私有声明区，用于启动元对象的特性。  
　　**３）元对象编译器　（Ｍeta-Object Complier, moc):** 为QObject子类实现元对象特性提供必要的代码实现。  
  
**５.3 布局管理器**  
　　在设计较复杂的ＧＵＩ用户界面时，仅通过指定窗口部件的父子关系以期达到加载和排列窗口部件的方法是行不通的，最好的办法是**使用Ｑt提供的布局管理器。**

QGridLayout \*mainLayout = new QGridLayout(this);　　　//(a)

mainLayout->addWidget(label1,0,0); 　　　　//(b)

mainLayout->addWidget(lineEdit,0,1);

mainLayout->addWidget(label2,1,0);

mainLayout->addWidget(button,1,1);

setLayout(mainLayout); 　　　　　　　　　　　　　　　　//(c)

　　其中：

　　(a) QGridLayout \*mainLayout = new QGridLayout(this) : 创建一个网格布局管理器对象　mainLayout, 并且 this 指出父窗口。

　　(b) mainLayout->addWidget(...) : 分别将控件label1等放置在该布局管理器中，还可以在创建布局管理器对象时不必指明父窗口。

　　(c) QWidget::setLayout(...) : 将布局管理器添加到对应的窗口部件对象中。因为这里的主窗口就是父窗口，所以直接调用　setLayout(mainLayout)即可。

Qt的网络模块性能比较差，而且难以提升，这是硬伤，除非Qt把现有的架构推倒重来。  
此外，Qt的UI运行效率与wxWidgets，Windows的MFC，Linux的GTK+也都没法比。  
主要原因之一是Qt的信号槽这个核心机制，给开发带来便利，但因此也丧失了一些性能。Qt的信号槽调用涉及链表操作，事件处理，还包括最伤性能的互斥锁，等等，相比直接回调方式。多出100多行代码，按官方说法，信号槽调用比直接回调慢了10倍左右。可是估计一旦遇到锁竞争，恐怕远远不只10倍了吧。Qt的UI与网络模块都严重依赖信号槽机制。  
不过，相对于Java、C#之类，Qt毕竟是C++，运行效率自然要胜出很多。  
总体上看，牺牲一点性能，换来漂亮的界面，更高的开发效率，大多时候还是蛮值的。

QT主要是跨平台特性，如果不想跨平台，windows下用WPF就好了，不要提MFC好么？  
嵌入式平台首选Qt啊，开发效率把别的框架爆的渣都不剩。运行效率？我没见到有多差。

