第3章-信号与槽

author：岳石磊 copyright：科林明伦 内部资料禁止外泄

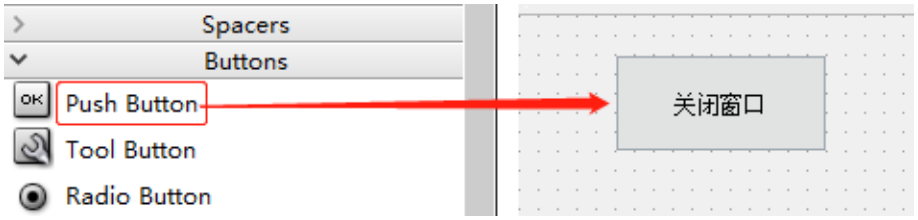
1. 基本概念

信号(signal)和槽(slot)是 Qt 框架引以为豪的机制之一，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。
信号：特定条件下发射的事件。例如 pushButton 最常见的信号是鼠标单击时发射一个 clicked信号。
槽：对信号响应的函数。槽就是一个类成员函数可以是任何属性的（public、protected、private），可以带有参数、也可以被直接调用，槽函数与一般的函数的区别：槽函数可以与一个信号关联，当信号被发射时，关联的槽函数被自动执行。
当某个事件发生之后它就会发射一个信号，如果有对象对这个信号感兴趣，将信号和自己的一个函数（称为槽（slot））绑定来处理这个信号，这个槽函数就会执行，也就是回调。所以槽的本质是一个类成员函数。
所有使用信号与槽的类中，必须有 `Q_OBJECT` 这个宏。

2. 添加信号与槽

2.1 方式一

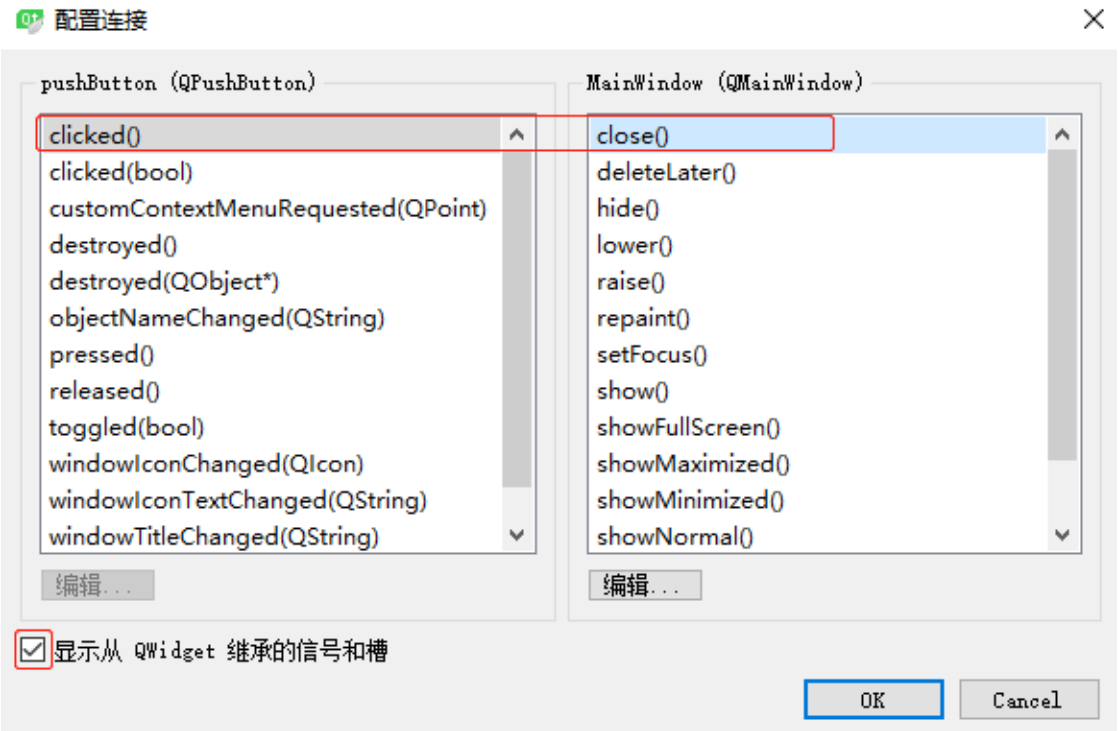
在UI界面编辑器中，我们手动从窗口添加一个信号与槽，假设我们想点击一下窗口上的按钮使窗口关闭程序退出。



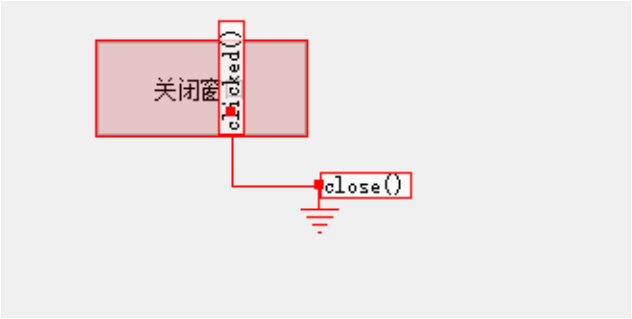
切换到第二个工具按钮【Edit Signal/Slots】在按钮中鼠标左键按下不释放并拖拽后释放，

在弹出的弹出框中选择，要添加的信号与槽，如果当前类中并未找到所需要的信号、槽函数，可以勾

选【显示从QWidget继承的信号与槽】。



最后点击【OK】就添加好了。



这里涉及到 clicked 和 close，分别为事件的触发函数 和 事件的处理函数，发送者为QPushButton 对象 接收者为 MainWindow 对象。

绝大多数的信号与槽，都是通过 QObject::connect 函数进行绑定关联的，在上例中我们添加的也不例外。添加完信号与槽之后，我们需要重新构建下，然后跳转到 ui_mainwindow.h的 Ui_MainWindow::setupUi 中，会看到这样一行代码

```
QObject::connect(Button, SIGNAL(clicked()), MainWindow, SLOT(close()));
```

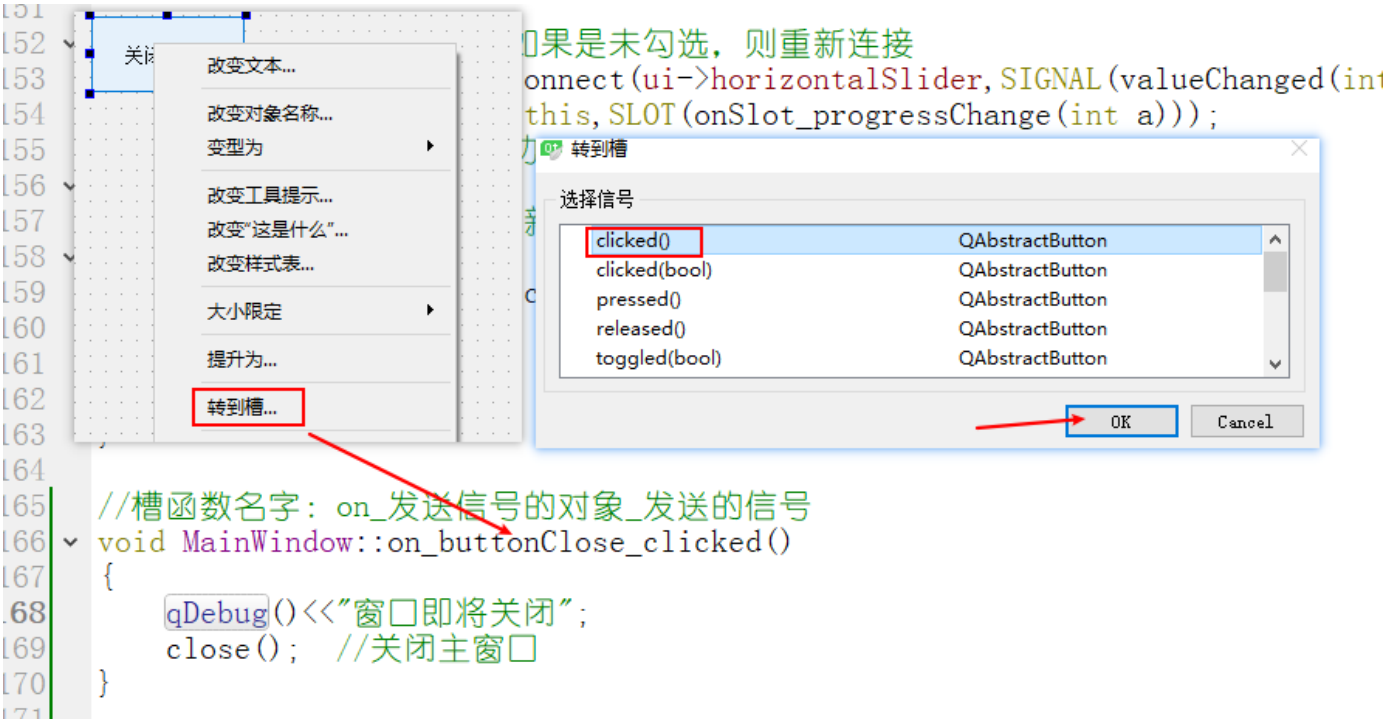
进行的关联。

```
void setupUi(QMainWindow *MainWindow)
{
    if (MainWindow->objectName().isEmpty())
        MainWindow->setObjectName(QStringLiteral("MainWindow"));
    MainWindow->resize(757, 627);
    centralWidget = new QWidget(MainWindow);
    centralWidget->setObjectName(QStringLiteral("centralWidget"));
    statusBar->setObjectName(QStringLiteral("statusBar"));
    MainWindow->setStatusBar(statusBar);

    retranslateUi(MainWindow);
    QObject::connect(pushButton, SIGNAL(clicked()), MainWindow, SLOT(close()));
```

当点击按钮时，触发信号QPushButton::clicked -> QMainWindow::close，这样主窗口就会被关闭了。

2.2 方式二

还有一种方便快捷的添加方式，选中某个组件->右键 ->【转到槽】，择一个信号，如：clicked，会在按钮的父窗口 mainwindow 中**自动生成**一个槽，也就是说这个信号的处理函数，由 Mainwindow 中自定义函数处理。



自动生成的槽函数名字 on_buttonClose_clicked() 是有规则的，void on_<object_name>_<signal_name>(<signal_parameters>);。

object_name 指的是触发信号组件的对象名。

signal_name 指的是信号名。

signal_parameters 槽函数参数一般与信号参数一致。 这里信号是无参的，所以槽函数也是无参。

信号槽的绑定与槽函数名字有对应关系，如果我们随意将 object_name 手动修改了，那么程序运行时会报错如下：

QMetaObject::connectSlotsByName: No matching signal for xxx 即 这个槽没有匹配的信号，这个也是便捷带来的弊端。

这种方式添加的信号槽并未直接用到connect函数关联绑定，而是在ui_mainwindow.h 的 setupUi的

QMetaObject::connectSlotsByName(MainWindow);中按name规则进行绑定的。

```
void setupUi(QMainWindow *MainWindow)
{
    if (MainWindow->objectName().isEmpty())
        MainWindow->setObjectName(QStringLiteral("MainWindow"));
    MainWindow->resize(757, 627);
    centralWidget = new QWidget(MainWindow);
    centralWidget->setObjectName(QStringLiteral("centralWidget"));
    pushButton = new QPushButton(centralWidget);
    pushButton->setObjectName(QStringLiteral("pushButton"));
    pushButton->setGeometry(QRect(380, 10, 111, 51));
    buttonClose = new QPushButton(centralWidget);

    retranslateUi(MainWindow);
    QObject::connect(pushButton, SIGNAL(clicked()), MainWindow, SLOT(close()));

    QMetaObject::connectSlotsByName(MainWindow);
} // setupUi
```

3. 自定义信号与槽

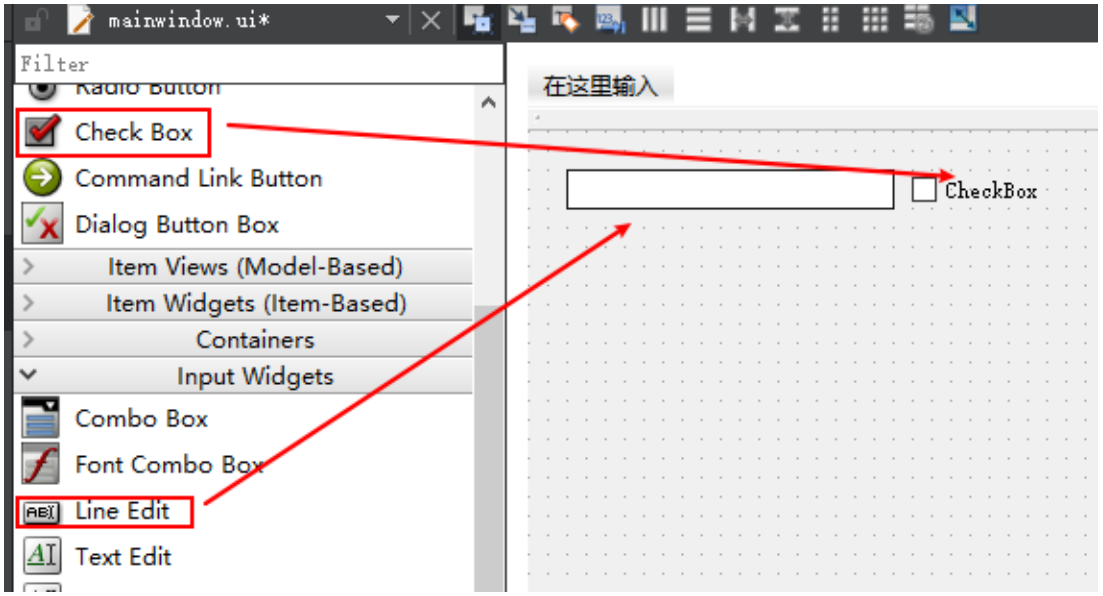
在QMetaObject::connectSlotsByName(MainWindow); 执行后，会自动匹配特定命名规则的槽函数，也就是说我们可以手动添加一个能被自动调用的槽函数。但这种方式并不是"长久之计"，
如果我们手动指定信号与槽的绑定连接，需要用到函数 QObject::connect。
常用的函数原型包含下面两个：

```
//方式一：(常用推荐)
QObject::connect(
    const QObject *sender,  //发出信号的对象
    const char *signal,     //发送对象发出的具体信号
    const QObject *receiver,//接收信号的对象
    const char *method,     //接收对象在接收到信号之后所需要调用的函数
    Qt::ConnectionType type = Qt::AutoConnection //连接类型
)

//方式二：
QMetaObject::Connection QObject::connect(
    const QObject *sender,
    PointerToMemberFunction signal,
    const QObject *receiver,
    PointerToMemberFunction method,
    Qt::ConnectionType type = Qt::AutoConnection
)
```

方式一指定信号和槽分别用宏 SIGNAL(a) 和 SLOT(a)，这两个宏最终将信号槽转换为字符串。
connect中信号与槽需要指定 **函数名，参数列表形参类型但不包含其变量名，不需要返回值。**

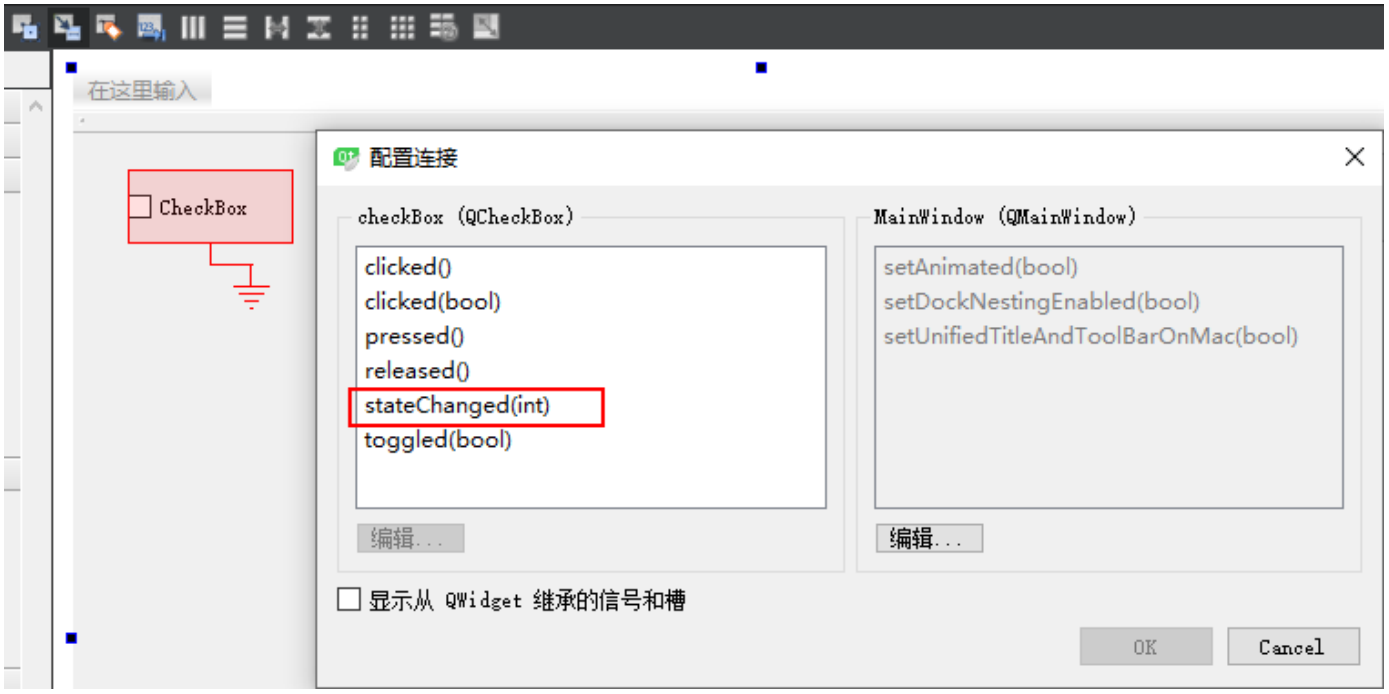
3.1 CheckBox

在窗口中添加一个【Check Box】按钮 和 【Line Edit】单行文本编辑框，当勾选【Check Box】 时设置编辑框不可编辑，取消勾选则恢复可编辑状态。



需要用到信号为 【Check Box】 的勾选状态，我们在信号和槽编辑框的弹出框中，查看其有哪些信号以及 哪个信号使我们所需要的。状态改变信号为：`stateChanged(int)`，记住这个信号**函数名及参数。**

注意：我们并不是要在这里添加，因为并没有合适的槽函数，而是需要我们自定义。



在`MainWindow`主窗口中添加自定义的槽函数，**在槽函数上需要有** 访问修饰符 `slots:` **进行修饰。返回类型一般为** void **，函数名自定义，参数与信号函数一致。**

```
private slots:    // slots :qt中的关键字，下方修饰槽函数
    void onSlots_StateChanged(int) ;
```

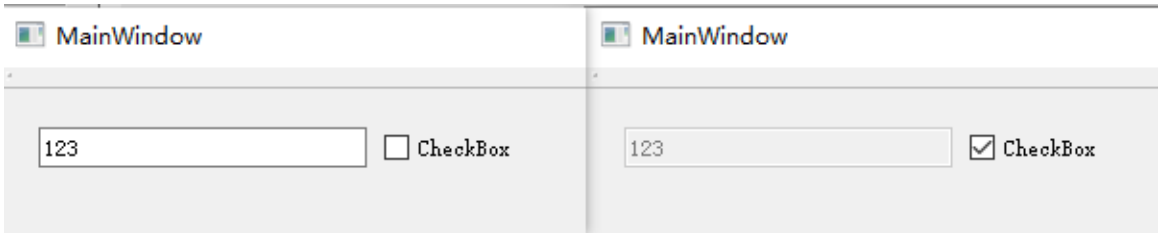
```
void MainWindow::onSlots_StateChanged(int a){
    qDebug()<<a;
    if(a==0){          //未勾选
        ui->lineEdit->setDisabled(false);    //编辑框可编辑
    }else if(a==2){  //勾选
        ui->lineEdit->setDisabled(true);     //编辑框不可编辑
    }
}
```

参数 a 默认传递两个值： 0： 表明勾选框未勾选，2：表明勾选框已勾选。

最后一个步是在使用之前进行 绑定连接， 在构造函数中添加如下代码：

```
QObject::connect(ui->checkBox,SIGNAL(stateChanged(int)),this,SLOT(onSlots_StateChanged(int)));
```

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    QObject::connect(ui->checkBox,SIGNAL(stateChanged(int)),
                     this,
                     SLOT(onSlots_StateChanged(int)));
```



参数状态 0、2 缺少 1。原来 1值代表的是 半选状态。

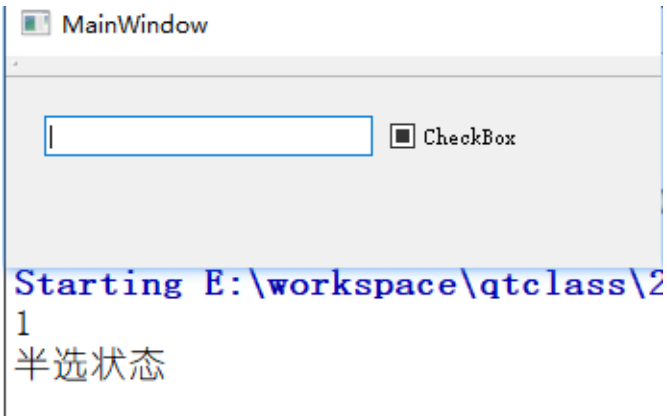
Constant	Value	Description
Qt::Unchecked	0	The item is unchecked.
Qt::PartiallyChecked	1	The item is partially checked. Items in hierarchical models may be partially checked if some, but not all, of their children are checked.
Qt::Checked	2	The item is checked.

我们开启三态，使用函数 `setTristate`。

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    QObject::connect(ui->checkBox,SIGNAL(stateChanged(int)),
                     this,
                     SLOT(onSlots_StateChanged(int)));


    ui->checkBox->setTristate(true);  //设置三态
```

在槽函数中，增加判断分支：

```
void MainWindow::onSlots_StateChanged(int a){
    qDebug()<<a;
    if(a==0){          //未勾选
        ui->lineEdit->setDisabled(false);    //编辑框可编辑
    }else if(a==2){  //勾选
        ui->lineEdit->setDisabled(true);     //编辑框不可编辑
    }else if(a==1){
        qDebug()<<"半选状态";
    }
}
```



3.2 自定义QMessageBox

对于QMessageBox中提供的静态函数，其只是支持已经定义好的一些按钮，却不支持自定义按钮。

```
QMessageBox::information(nullptr,"title","text",QMessageBox::Ok|QMessageBox::Cancel);
```

尝试着手动构建 `QMessageBox` 对话框，在`MainWindow` 构造函数中，new QMessageBox ，并添加标题、文本和按钮。

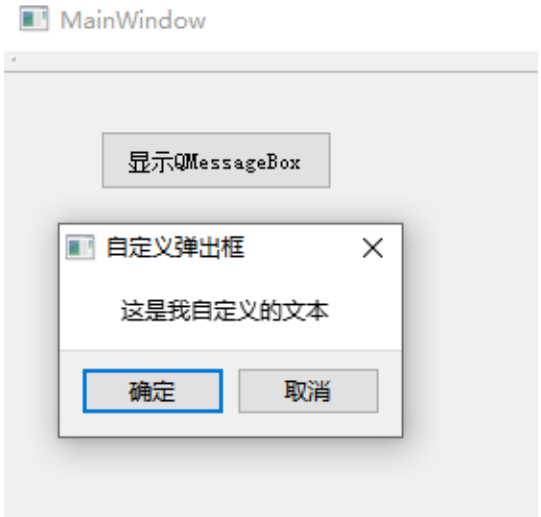
```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    pMsg = new QMessageBox;
    pMsg->setWindowTitle("自定义弹出框");  //设置弹出框的标题
    pMsg->setText("这是我自定义的文本");

    pOk     = pMsg->addButton("确定",QMessageBox::AcceptRole);
    pCancel = pMsg->addButton("取消",QMessageBox::RejectRole);
```

在窗口增加一个按钮，用于控制显示对话框，

在按钮信号 `clicked` 对应的槽函数中添加如下代码：`show`用于显示已经创建好的窗口。

```
void MainWindow::on_showQMsg_clicked()
{
    pMsg->show();   //显示弹出框
}
```

点击按钮，如下：



当点击【确定】【取消】要如何确定，到底是点击了哪个按钮了呢？当然是通过信号与槽机制了。

当 addButton 添加按钮返回对应的指针，我们需要存储一下。

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    pMsg = new QMessageBox;
    pMsg->setWindowTitle("自定义弹出框");  //设置弹出框的标题
    pMsg->setText("这是我自定义的文本");

    pOk     = pMsg->addButton("确定",QMessageBox::AcceptRole);
    pCancel = pMsg->addButton("取消",QMessageBox::RejectRole);
```

```
private:
    Ui::MainWindow *ui;

    QMessageBox* pMsg;     //这是一个弹出框的指针
    QPushButton * pOk;     //添加【确定】返回的QPushButton指针
    QPushButton * pCancel;//添加【取消】返回的QPushButton指针
```

我们在寻找，在QMessageBox 中是否存在着与点击按钮有关的信号?
转定义到 QMessageBox 类中，搜索signal 找到 Q_SIGNALS ，其下方修饰的就是信号 buttonClicked

```
277  class Q_WIDGETS_EXPORT QMessageBox : public QDialog
278  {
279      Q_OBJECT
280
281  Q_SIGNALS:
282      void buttonClicked(QAbstractButton *button);
283
284  #ifdef Q_QDOC
285  public Q_SLOTS:
```

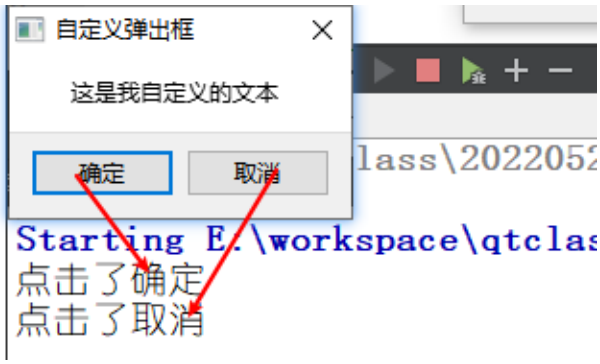
Find: signal 搜索
Replace with:

在MainWindow 中添加自定义的槽函数， 注意参数应当与信号保持一致。

```
private slots:   // slots :qt中的关键字，下方修饰槽函数
    void onSlots_clickButton(QAbstractButton *);
```

```
void MainWindow::onSlots_clickButton(QAbstractButton * button){
    if(pOk ==  button)
        qDebug()<<"点击了确定";
    else if(pCancel == button)
        qDebug()<<"点击了取消";
}
```

判断点击了哪个按钮，通过参数中的 `button` 指针与 `pOk` 和 `pCancel` 作比较可判断出。



3.3 自定义信号

自定义信号一般由 signals关键字在其上方修饰，且前面**不能有访问修饰符**修饰，没有返回值，但可以有参数，信号就是函数的声明，**只需声明，无需定义**，信号可以重载。

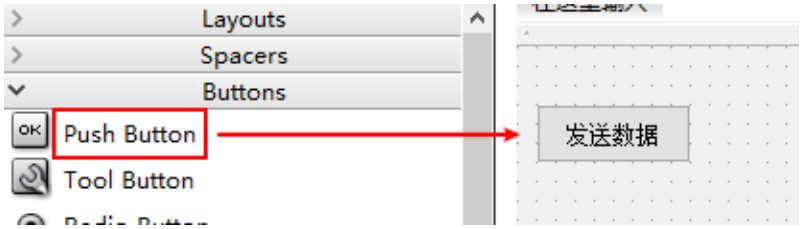
```
signals:            // ok
Q_SIGNALS:          // ok
public Q_SIGNALS:   //Error: Signals cannot have access specifier   （信号不能有访问修饰符）
public signals:     //Error: Signals cannot have access specifier
```

自定义槽函数 需要使用 slots 关键字或 Q_SLOTS 宏在上方修饰，且前面必须要有**访问修饰符**修饰(public、protected、private 三者任一），没有返回值，**参数类型、数量、顺序 一般与信号函数一致**，既要函数声明也要完成定义。

```
public slots:  // ok
public Q_SLOTS:// ok
slots:         //Error: Missing access specifier for slots
Q_SLOTS:       //Error: Missing access specifier for slots
```

槽函数和普通的 C++成员函数没有很大的区别。它们也可以使 virtual 的；可以被重写。

在主窗口中添加按钮并更名为【发送数据】，在这个按钮的槽函数中去 发射一个信号，



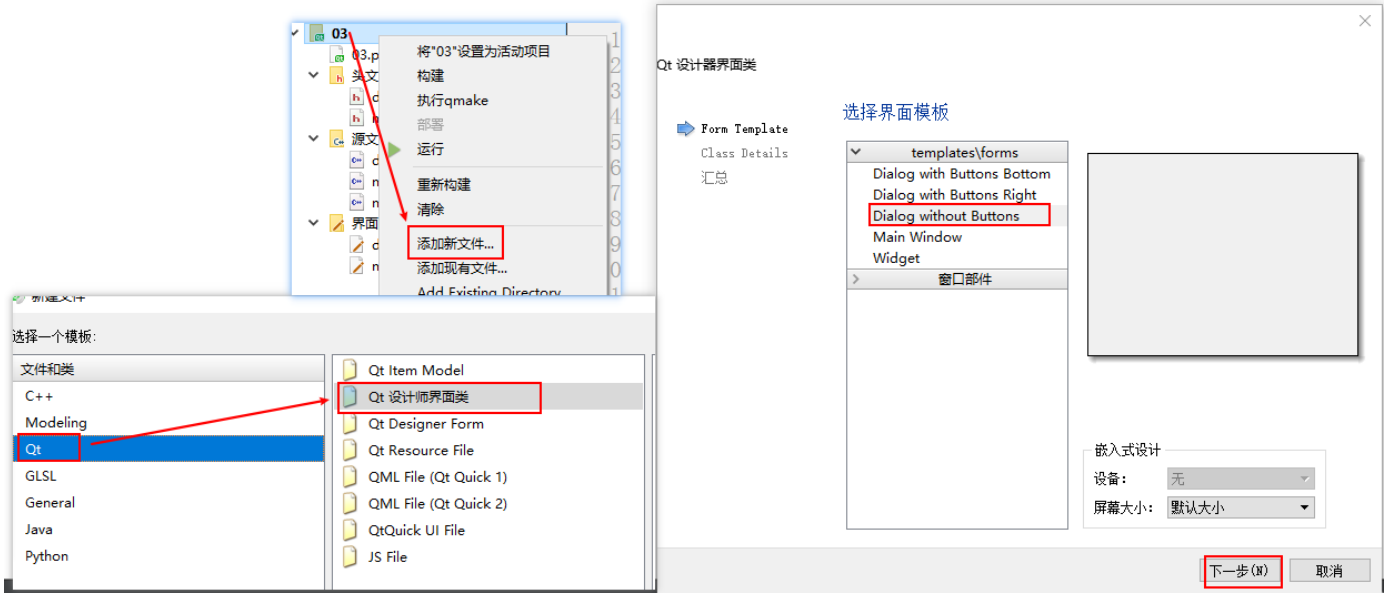
在MainWindow中自定义一个信号：参数为 int,int,QString。

```
class MainWindow : public QMainWindow
{
    Q_OBJECT

signals:  //signals 修饰信号的关键字，下面的函数被认为是信号，前面没有访问修饰符修饰，
    //槽函数修饰时，slots前面一定要有访问修饰符
    void onSign_sendData(int,int,QString);    //信号函数只需要声明，不需要定义
```

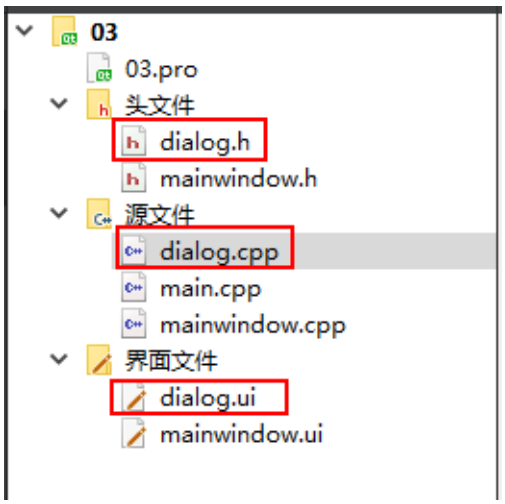
在按钮点击信号的槽函数中，增加如下代码，其中 emit 为宏，标识此处为发射一个信号。

```
void MainWindow::on_button_sendData_clicked()
{
    int a = 10;
    int b =20;
    QString str="加法";
    emit onSign_sendData(a,b,str);    //发射一个信号
}
```

接下来准备接收者：
新建一个对话框窗口，项目->【右键】->【添加新文件】



会增加 【dialog.h】【dialog.cpp】【dialog.ui】。



在 dialog 窗口上添加Label组件，用于显示接收到的数据

在 dialog 中增加自定义的槽函数，并在源文件中定义。

```
class Dialog : public QDialog
{
    Q_OBJECT

public:
    explicit Dialog(QWidget *parent = 0);
    ~Dialog();

public slots:
    //槽函数和信号参数应当保持一致
    void onSlot_RecvData(int,int,QString);    //槽函数不但要声明而且一定要定义出来
```

```
void Dialog::onSlot_RecvData(int a,int b,QString str){
    //将文字显示到窗口的label标签上
    ui->label->setText(str+"结果为："+QString::number(a+b));
    this->show();  //显示dialog 窗口
}
```

定义槽接收者对象，并绑定自定义信号与槽。

```cpp
#include "mainwindow.h"
#include <QApplication>
#include "ui_mainwindow.h"
#include"dialog.h"

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    MainWindow w;
    w.show();

    Dialog dia;

    //自定义的信号和槽进行绑定连接
    QObject::connect(&w,SIGNAL(onSign_sendData(int,int,QString)),
                     &dia,SLOT(onSlot_RecvData(int ,int ,QString)));
```

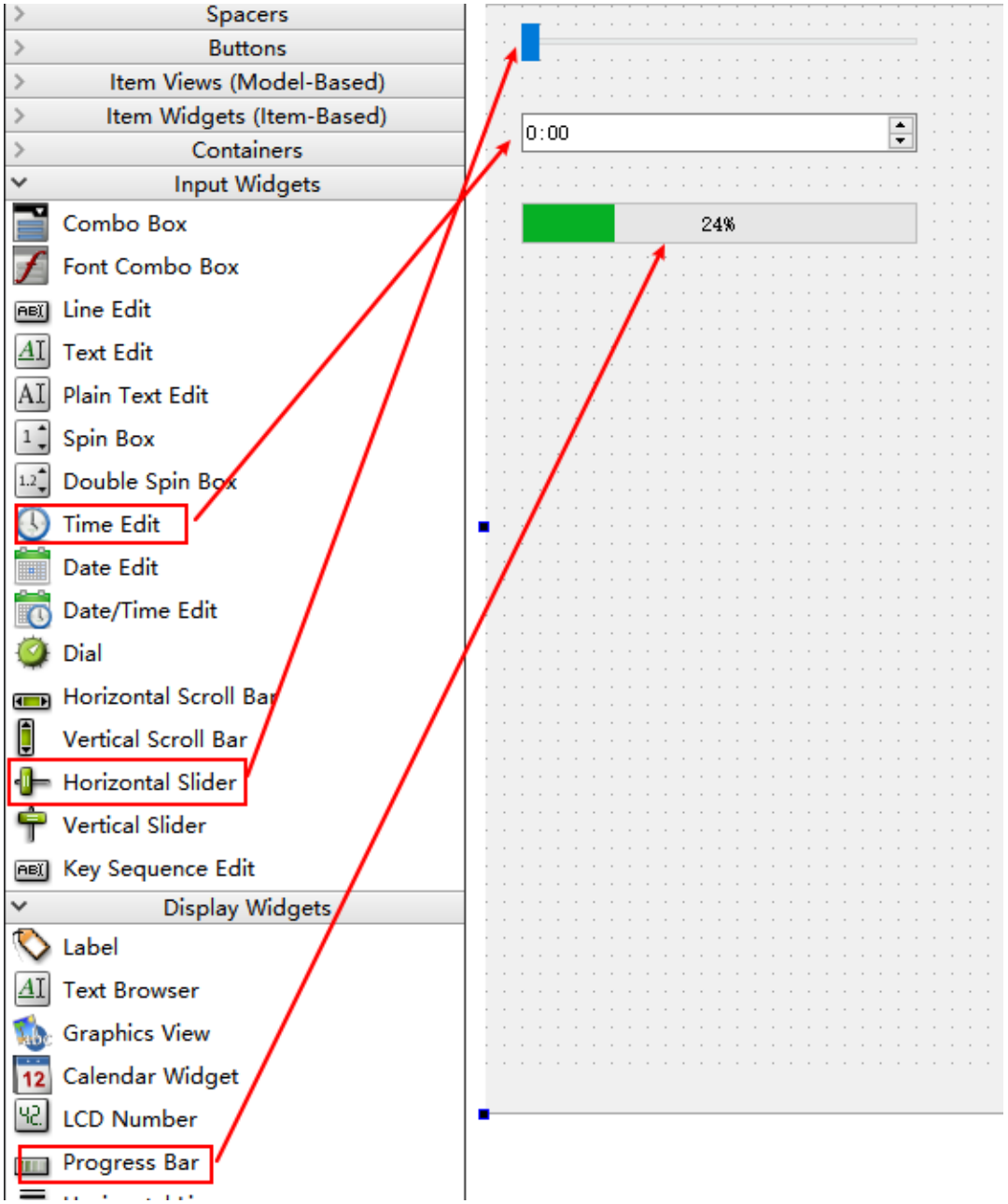
测试：

```cpp
void MainWindow::on_button_sendData_clicked()
{
    int a = 10;
    int b =20;
    QString str="加法";
    emit onSign_sendData(a,b,str);    //发射一个信号
}
```

4.信号与槽多对多

4.1 一个信号连接多个槽

在窗口上添加【Horizontal Slider】水平滑块 、【Time Edit】时间编辑框、【Progress Bar】进度条三个组件，通过水平滑块控制其他两个组件。



通过信号-槽的形式，信号的发出者为 水平滑块，信号选择为 `valueChanged(int)` 译为：滑块有操作导致其值改变时，就是发射这个信号。

对于接收者来说，我们选定的是`Mainwindow`，让其中的两个槽函数分别控制 时间编辑框 和 进度条 显示对应的值。

在MainWindow 中添加两个槽函数：`onSlots_changeTime` 、`onSlots_changeProgress`。

```
class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
private slots:  //修饰槽函数的关键字:  slots 前面需要访问修饰符修饰
    void onSlots_changeTime    (int value);
    void onSlots_changeProgress(int value);
```

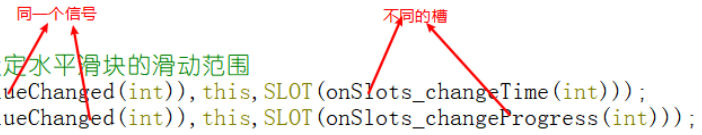
```
void MainWindow::onSlots_changeTime(int value){
    QTime time(0,0);                                    //初始为0时
    time = time.addSecs(3600*24*value/100);             //按照比例增加秒数

    ui->timeEdit->setDisplayFormat("hh-mm:ss");  //设置显示的格式
    ui->timeEdit->setTime(time);                        //设置时间
}
void MainWindow::onSlots_changeProgress(int value){
    qDebug()<<"value = "<<value;
    ui->progressBar->setValue(value);  //设置进度条的值
}
```

注意信号-槽的参数为int，水平滑块滑动值的范围默认为 0~99，但我们想要改变其范围 0~100，在构造函数中，通过函数 setRange 设定范围：

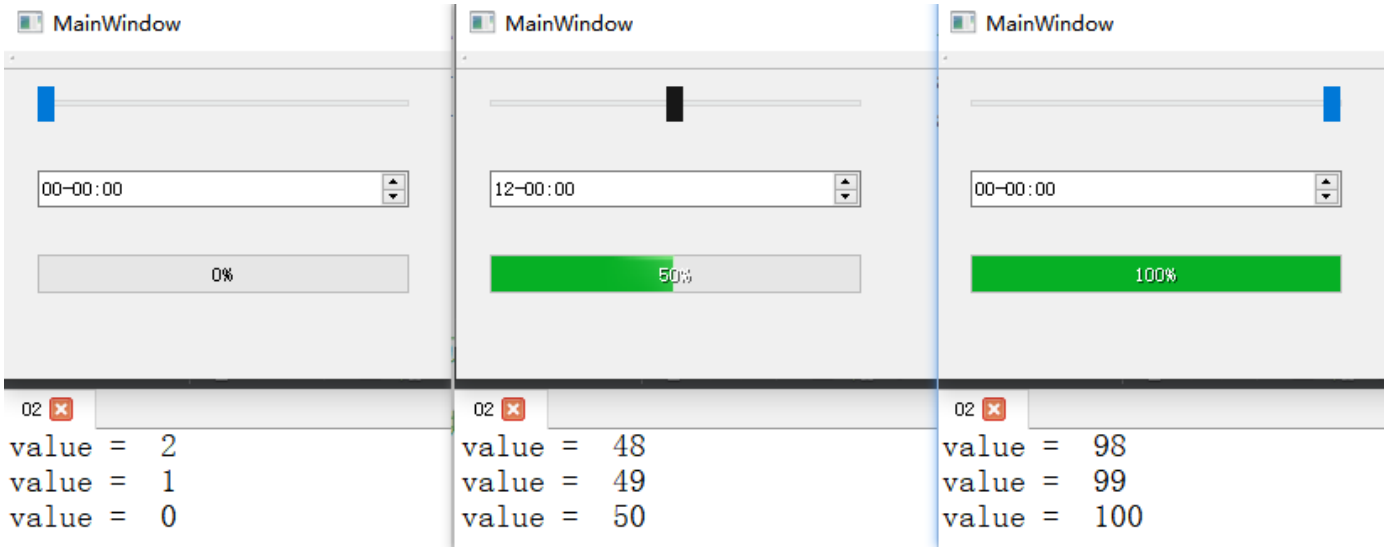
```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    ui->horizontalSlider->setRange(0,100);  //重新设定水平滑块的滑动范围
```

最后绑定连接：

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    ui->horizontalSlider->setRange(0,100);  //重新设定水平滑块的滑动范围
    QObject::connect(ui->horizontalSlider,SIGNAL(valueChanged(int)),this,SLOT(onSlots_changeTime(int)));
    QObject::connect(ui->horizontalSlider,SIGNAL(valueChanged(int)),this,SLOT(onSlots_changeProgress(int)));
```



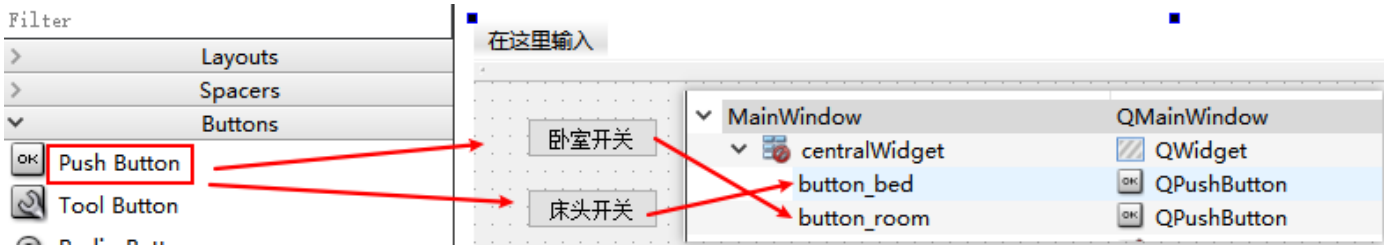
结果如下：



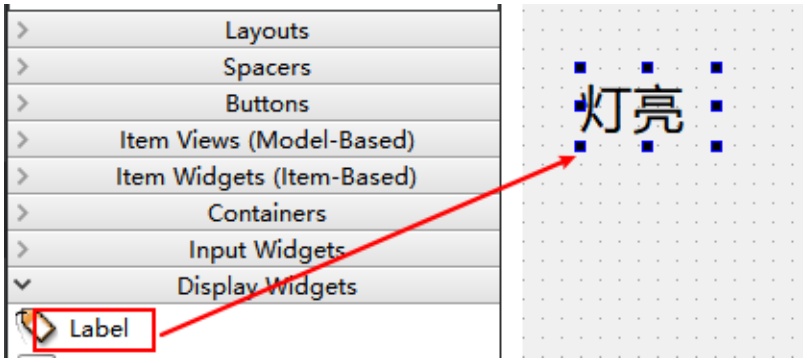
多个槽按照绑定的顺序依次去调用。

4.2 多个信号连接一个槽

在主窗口添加两个【Push Button】并分别更名为 【卧室开关】【床头开关】。对象名称分别更名为【button_bed】【button_room】



这里模拟一下“灯的双控开关”，所谓的灯用之前添加好的【Dialog】弹出框模拟。在其上添加【Label】组件文字显示“灯亮”。



这里信号有两个【button_bed】-> `clicked` 和 【button_room】->`clicked`。槽的接收者为

【Dialog】，自定义槽函数为onSlot_showOrHide。

```
class Dialog : public QDialog
{
    Q_OBJECT

public:
    explicit Dialog(QWidget *parent = 0);
    ~Dialog();

private:
    Ui::Dialog *ui;

public slots:
    void onSlot_showOrHide();
```

```
void Dialog::onSlot_showOrHide(){
    //判断窗口是否显示
    if(this->isVisible())  this->hide();  //隐藏窗口
    else this->show();  //重新显示窗口
}
```

接下来就是在主函数中绑定连接了，我们发现 主窗口的中成员 ui 其为私有的成员属性，在类外主函数中并不能直接使用。

```
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    Dialog dia;

    //多个信号 连接同一个槽
    QObject::connect(w.ui->button_bed,SIGNAL(clicked()),
                     &dia,SLOT(onSlot_showOrHide()));
    QObject::connect(w.ui->button_room,SIGNAL(clicked()),
                     &dia,SLOT(onSlot_showOrHide()));
```

```
class MainWindow : public QMainWindow
{

private:
    Ui::MainWindow *ui;
```

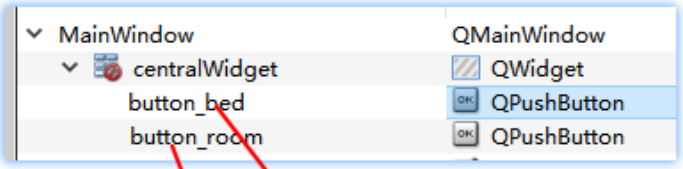
所以我们需要提供公共的UI接口函数。

```
class MainWindow : public QMainWindow
{

private:
    Ui::MainWindow *ui;
public:
    Ui::MainWindow * GetUi();

};
```

```
Ui::MainWindow * MainWindow::GetUi(){
    return ui;
}
```

于是主函数中改为调用 GetUi函数:

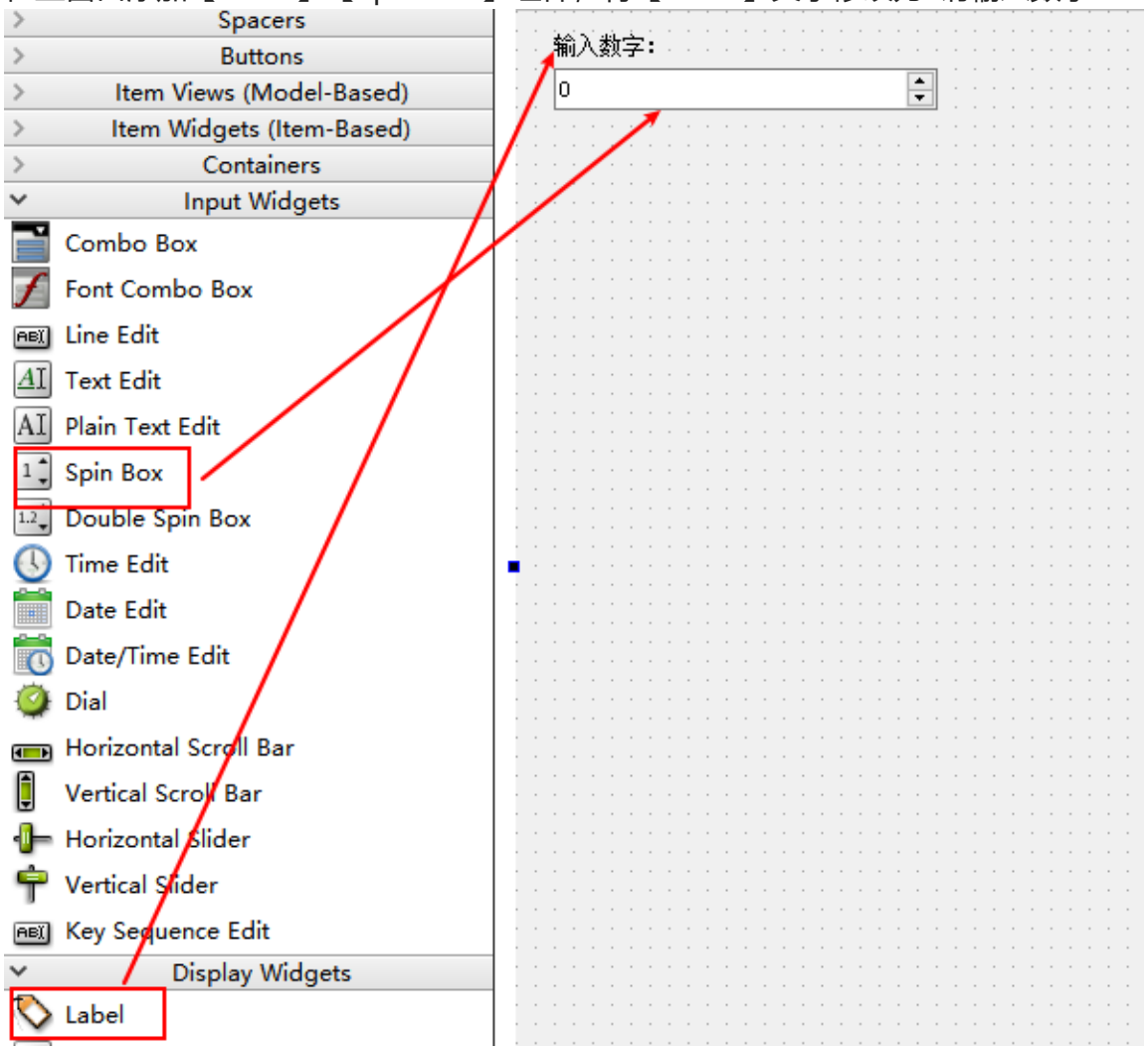
```
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    Dialog dia;



    //多个信号 连接同一个槽
    QObject::connect(w.GetUi()->button_bed, SIGNAL(clicked()),
                     &dia, SLOT(onSlot_showOrHide()));
    QObject::connect(w.GetUi()->button_room, SIGNAL(clicked()),
                     &dia, SLOT(onSlot_showOrHide()));
```



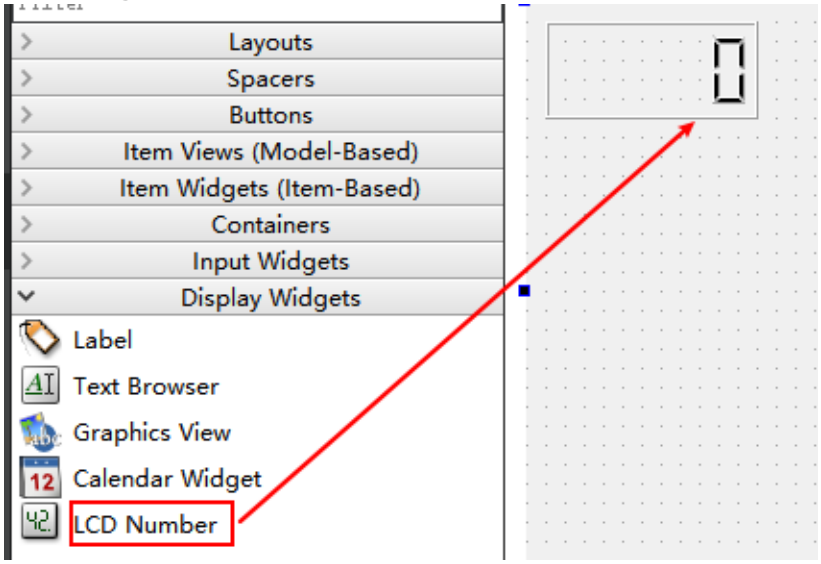
只要任意一个信号发出（点击任何一个按钮），这个槽就会被调用（Dialog会显示或隐藏）。

4.3 一个信号连接一个信号

在主窗口添加【Label】【Spin Box】组件，将【Label】文字修改为“请输入数字：”



在 Dialog 窗口上增加 【LCD Number】组件，同样是用于显示数字。



这个例子中，希望通过 MainWindow ->【Spin Box】 更改数字 ——> Dialog ->【LCD Number】上显示对应的数字。

我们选择的信号为：【Spin Box】->`valueChanged(int)` 译为：当输入框中数值改变时，会发射这个信号。

槽为：Dialog 窗口中自定义的槽函数 `onSlot_recvInt(int)`，注意参数要与信号保持一致。

```
class Dialog : public QDialog
{
    Q_OBJECT

public:
    explicit Dialog(QWidget *parent = 0);
    ~Dialog();

private:
    Ui::Dialog *ui;

public slots:
    void onSlot_recvInt(int);
```

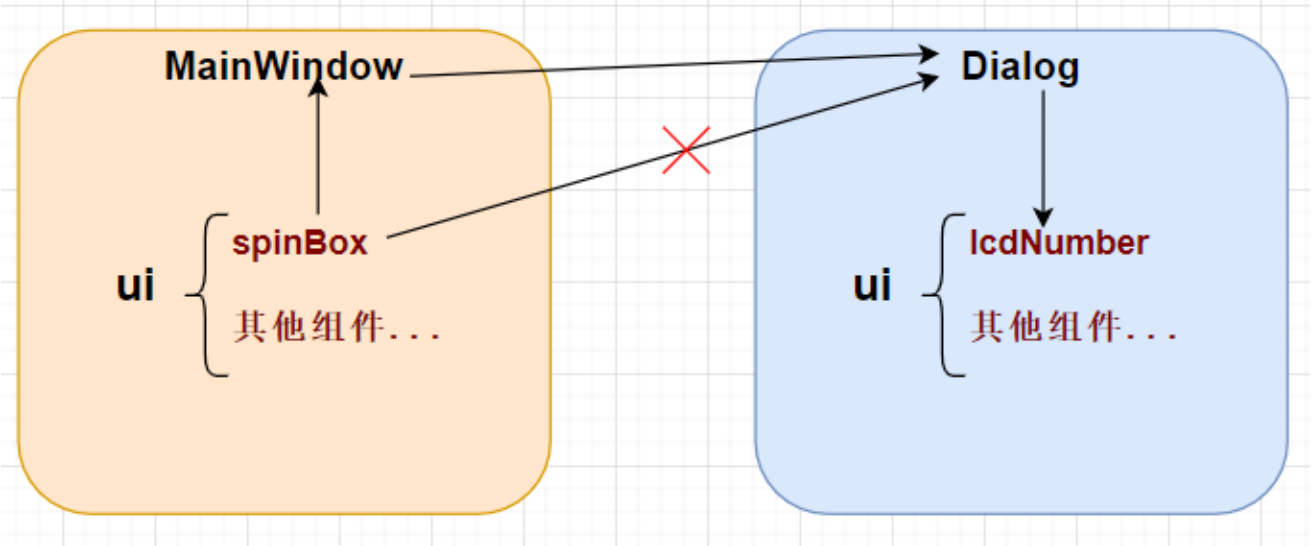
```
void Dialog::onSlot_recvInt(int a){
    //在 LCD 组件上显示数字
    ui->lcdNumber->display(a);
}
```

接下来进行绑定连接，这个连接的操作应当在主函数中，假设我们做一个约束条件，为了窗口安全考虑，主窗口中的 `ui` 其私有的属性不能修改，也不允许提供公共的`GetUi`接口函数。总之就只能在类内部使用。

```
private:
    Ui::MainWindow *ui;
public:
    Ui::MainWindow * GetUi();

};
```

此时在主函数中绑定连接，因为`ui`访问权限问题，而无法连接。此时我们需要一个“中转”，这个中转则为 `MainWindow`。

```
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    Dialog dia;

    QObject::connect(w.ui->spinBox,SIGNAL(valueChanged(int)),&dia,SLOT(onSlot_recvInt(int)));
```



在MainWindow 中添加一个自定义信号`void onSignal_sendInt(int)`。

```
class MainWindow : public QMainWindow
{
    Q_OBJECT

signals:  //前面不要有任何访问修饰符
    void onSignal_sendInt(int);    //自定义的信号
```

于是此时的连接顺序就变为： spinBox->`valueChanged(int)`（信号） ——> MainWindow->`onSignal_sendInt(int)` （信号）——> dialog->`onSlot_recvInt(int)`（槽）。

第一处连接在主窗口的构造函数中：

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    //spinBox 的信号  与 主窗口的信号进行绑定连接
    QObject::connect(ui->spinBox,SIGNAL(valueChanged(int)),this,SIGNAL(onSignal_sendInt(int)));
```

第二处连接在main 函数中：

```
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    Dialog dia;
    //主窗口的信号  与 Dialog 的槽进行绑定连接
    QObject::connect(&w,SIGNAL(onSignal_sendInt(int)),&dia,SLOT(onSlot_recvInt(int)));
```

5. 断开连接

在 `QObject::connection()` 建立连接时，会返回一个 `QMetaObject::Connection` ，其可以用于判断是否连接成功。

在MainWindow中增加一个成员属性，`m_con` 用于承接 时间组件和水平滑块之间的连接信息。

```
class MainWindow : public QMainWindow
{
    Q_OBJECT

private:
    QMetaObject::Connection m_con;
```

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    ui->horizontalSlider->setRange(0,100);  //重新设定水平滑块的滑动范围
    QObject::connect(ui->horizontalSlider,SIGNAL(valueChanged(int)),this,SLOT(onSlots_changeProgress(int)));
    m_con = QObject::connect(ui->horizontalSlider,SIGNAL(valueChanged(int)),this,SLOT(onSlots_changeTime(int)));
    if(m_con){
        qDebug()<<"连接成功";
    }else{
        qDebug()<<"连接失败";

    }
```

正常连接成功：

```
43      ui->horizontalSlider->setRange(0,100);
44      QObject::connect(ui->horizontalSlider,SI
45      m_con = QObject::connect(ui->horizontalS
46      if(m_con){
47          qDebug()<<"连接成功";
48      }else{
49          qDebug()<<"连接失败";
50
51      }
52
53
54
```

```
应用程序输出
02
Starting E:\workspace\qtclass\20220705\02\build-02-D
连接成功
```

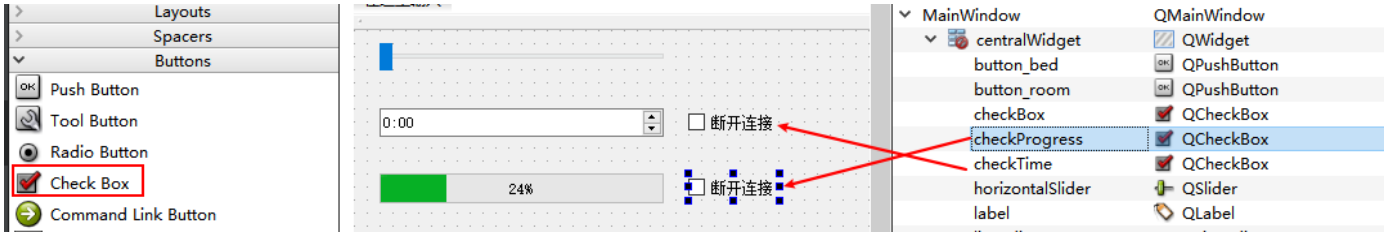
当我们故意在槽函数增加一个形参名，则绑定会报错：

```
    ui->horizontalSlider->setRange(0,100);  //重新设定水平滑块的滑动范围
    QObject::connect(ui->horizontalSlider,SIGNAL(valueChanged(int)),this,SLOT(onSlots_changeProgress(int)));
    m_con = QObject::connect(ui->horizontalSlider,SIGNAL(valueChanged(int)),this,SLOT(onSlots_changeTime(int a)));
    if(m_con){
        qDebug()<<"连接成功";
    }else{
        qDebug()<<"连接失败";

    }
```

```
应用程序输出
02
Starting E:\workspace\qtclass\20220705\02\build-02-Desktop_Qt_5_6_2_MinGW_32bit-Debug\debug\02.exe...
连接成功
E:\workspace\qtclass\20220705\02\build-02-Desktop_Qt_5_6_2_MinGW_32bit-Debug\debug\02.exe exited with code 0

Starting E:\workspace\qtclass\20220705\02\build-02-Desktop_Qt_5_6_2_MinGW_32bit-Debug\debug\02.exe...
QObject::connect: No such slot MainWindow::onSlots_changeTime(int a) in ..\02\mainwindow.cpp:45
QObject::connect:  (sender name:   'horizontalSlider')
QObject::connect:  (receiver name: 'MainWindow')
连接失败
```

除此之外，连接信息还可以用于取消连接。

在主窗口上增加两个 【Check Box】并更改文字为 “断开连接”分别用于控制 水平滑块 和 时间组件、进度条之间是否连接。



对于这两个【Check Box】分别添加信号与槽，信号为 `valueChanged(int)`。槽需要自己定义出来。如下：

```
class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
private slots:  //修饰槽函数的关键字：  slots 前面需要访问修饰符修饰
    void onSlot_checkTime(int);
    void onSlot_checkProgress(int);
```

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    QObject::connect(ui->checkTime,SIGNAL(stateChanged(int)),this,SLOT(onSlot_checkTime(int)));
    QObject::connect(ui->checkProgress,SIGNAL(stateChanged(int)),this,SLOT(onSlot_checkProgress(int)));

}
```

如果要取消信号与槽的连接，第一种方式使用 `QObject::disconnect()` 参数为 建立连接时返回的 `QMetaObject::Connection` 连接信息。

第二种方式 QObject::disconnect() 参数为 连接时 connect 的前4个参数。

```
void MainWindow::onSlot_checkTime(int a){
    if(a==2){                    方式一:
        QObject::disconnect(m_con);  //取消连接
    }else{  //重新连接上
        m_con = QObject::connect(ui->horizontalSlider,SIGNAL(valueChanged(int)),this,SLOT(onSlots_changeTime(int)));
    }
}
void MainWindow::onSlot_checkProgress(int a){
    if(a==2){
        //取消连接      方式二:
        QObject::disconnect(ui->horizontalSlider,SIGNAL(valueChanged(int)),this,SLOT(onSlots_changeProgress(int)));
    }else{  //重新连接上
        QObject::connect(ui->horizontalSlider,SIGNAL(valueChanged(int)),this,SLOT(onSlots_changeProgress(int)));
    }
}
```