

知识点1【Qt的概述】（了解）

---

知识点2【Qt工程的创建】

---

知识点3【第一个Qt工程】

---

知识点4【第一个Qt按钮】

---

知识点5【对象树】

---

知识点6【Qt的坐标体系】

---

知识点7【信号和槽机制】（重要）

---

案例1：系统的信号和槽

---

案例2：自定义信号和槽函数

---

定义信号：

---

定义槽函数：

---

信号和槽之间的参数传递

---

信号和槽的注意：

---

知识点8【lambda表达式】（了解）

---

知识点9【QMainWindow窗口】（了解）

---

1、创建菜单栏（QMenuBar）

---

2、创建菜单（QMenu）

---

3、创建菜单项（QAction）

---

4、让菜单项动起来

---

5、设置菜单项的快捷方式

---

6、添加一个分隔符

---

7、创建工具栏QToolBar

---

## 8、创建状态栏

## 9、创建中心部件

## 10、创建铆接部件

### 知识点10【资源文件】（了解）

给菜单项添加图标 :代表的是资源文件

### 知识点11【UI文件的使用】

### 知识点12【对话框】

#### 1、模态对话框 和 非模态对话框

#### 2、消息对话框QMessageBox

错误对话框:

信息对话框

询问对话框

#### 3、文件对话框QFileDialog

#### 4、颜色对话框

#### 5、字体对话框QFontDialog

### 知识点13【布局管理】

#### 1、水平布局QHBoxLayout

#### 2、垂直布局QVBoxLayout

#### 3、栅格布局QGridLayout

### 知识点14【登录器】

#### 1、界面布局

#### 2、输入框的密码模式

#### 3、获取输入框的内容

#### 4、设置输入框的内容

5、创建新的页面

6、页面的跳转

7、从新页面 回到 主页面

8、完整代码

#### 知识点15【常用控件】

1、QLabel 标签

1、显示文本

2、显示图片

3、显示动画

2、单选框QRadioButton

3、复选框QCheckBox

4、下拉列表框QComboBox

5、列表控件QListWidget

6、树控件QTreeWidget

7、表格控件QTableWidget

8、工具盒子QToolBox

9、自定义控件

10、tab widget容器

11、stackedWidget栈容器（局部更新页面）

#### 知识点16【Qt的事件】

1、Qt事件的概述

2、重写QLabel的鼠标事件

3、Qevent事件分发器

4、事件过滤器

### 知识点17【定时器】

- 1、定时器事件 触发定时
- 2、定时器对象 触发定时 (方便控制定时器)
- 3、定时器静态函数 触发定时 用于延时

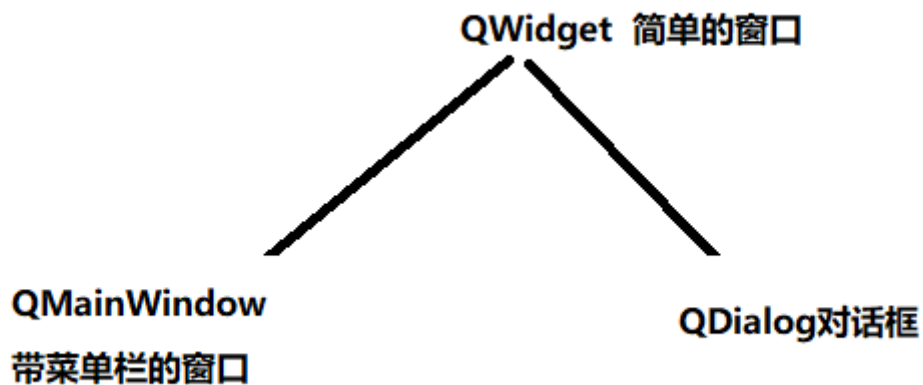
### 知识点18【Qt绘制背景图片】

- 1、绘制背景图片
- 2、切换背景图片

## 知识点1【Qt的概述】（了解）

Qt下载位置: <http://download.qt.io/archive/qt/>

## 知识点2【Qt工程的创建】



Qt Widgets Application

Location

Kits

Details

汇总

类信息

指定您要创建的源码文件的基本类信息。

类名(C):

基类(B):

头文件(H):

源文件(S):

创建界面(G): ☐ 去掉✓

界面文件(F):

下一步(N)

取消

## main.cpp

```
1 #include "widget.h"
2 #include <QApplication>
3
4 int main(int argc, char *argv[])
5 {
6     //定义一个应用程序 对象a
7     QApplication a(argc, argv);
8
9     //定义一个主窗口 w
10    Widget w;
11    //默认窗口为隐藏 需要调用show显示 hide隐藏
12    w.show();
13
14    //exec主事件循环 （等待用户操作界面 并响应）
15    return a.exec();
16 }
```

```
1 #include "widget.h"
2
3 Widget::Widget(QWidget *parent)
4 : QWidget(parent)
5 {
6     //设置窗口 构造函数中设计窗口
7 }
8
9 Widget::~Widget()
10 {
11
12 }
```

widget.h

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>

class Widget : public QWidget
{
    Q_OBJECT    //让当前类Widget支持信号和槽机制（界面动起来）

public:
    Widget(QWidget *parent = 0);
    ~Widget();
};

#endif // WIDGET_H
```

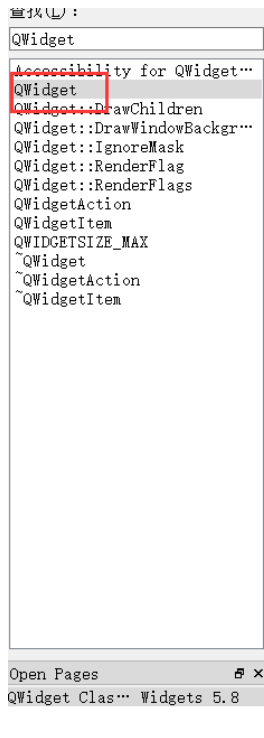
## QPushButton Class

The `QPushButton` widget provides a command button. [More...](#)

Header: `#include <QPushButton>`  
qmake: `QT += widgets`  
Inherits: `QAbstractButton` 父类  
Inherited By: `QCommandLinkButton` 子类  
• List of all members, including inherited members

## 知识点3 【第一个Qt工程】

设置窗口标题：



## Public Slots

```

bool    close()
void    hide()
void    lower()
void    raise()
void    repaint()
void    setDisabled(bool disable)
void    setEnabled(bool)
void    setFocus()
void    setHidden(bool hidden)
void    setStyleSheet(const QString &styleSheet)
virtual void setVisible(bool visible)
void    setWindowModified(bool)
void    setWindowTitle(const QString &)
void    show()

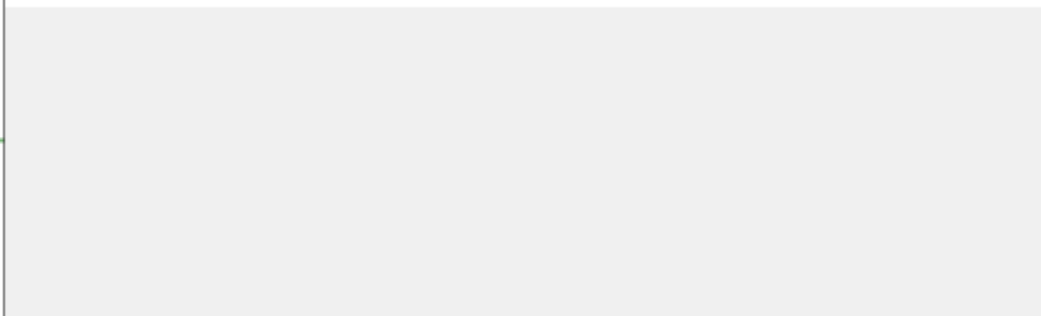
```

```

#include "widget.h"
#include <QString>
Widget::Widget(QWidget *parent)
    : QWidget(parent) 主窗口的构造函数中
{
    //修改窗口的标题
    this->setWindowTitle("第一个窗口");
    //设置窗口的大小
    this->resize(800,600);
    //固定窗口大小
    //this->setFixedSize(QSize(800,600));
    this->setFixedSize(800,600);
}

```

第一个窗口

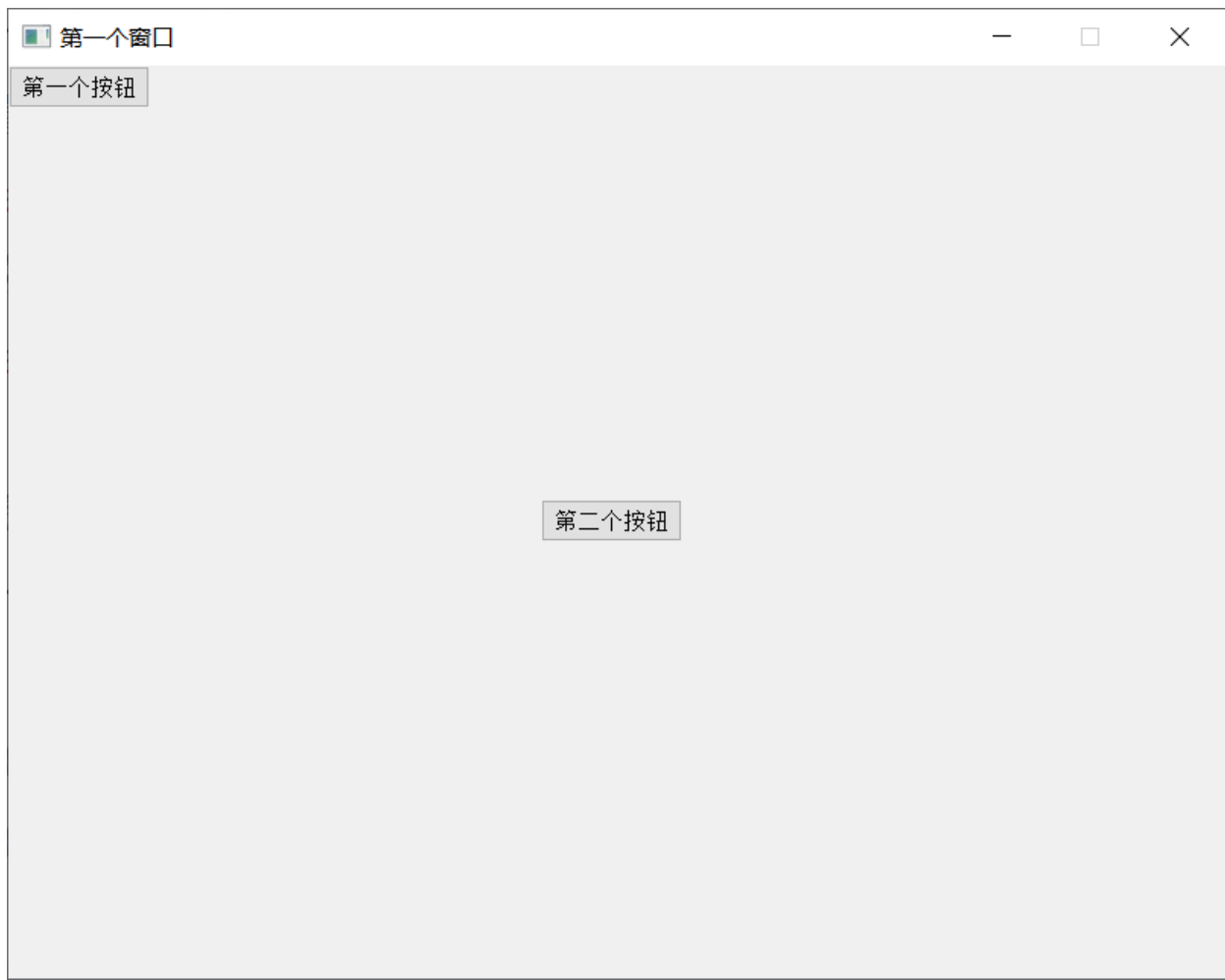


## 知识点4 【第一个Qt按钮】

widget构造函数中：

```
1  Widget::Widget(QWidget *parent)
2      : QWidget(parent)
3  {
4      //修改窗口的标题
5      this->setWindowTitle("第一个窗口");
6      //设置窗口的大小
7      this->resize(800,600);
8      //固定窗口大小
9      //this->setFixedSize(QSize(800,600));
10     this->setFixedSize(800,600);
11
12     //创建一个按钮控件
13     QPushButton *btn1 = new QPushButton;
14     //给按钮设置文本
15     btn1->setText("第一个按钮");
16     //设置按钮的父对象（在哪个窗口显示按钮 按钮就认哪个窗口为父对象）
17     btn1->setParent(this);
18
19     //创建一个按钮控件
20     QPushButton *btn2 = new QPushButton("第二个按钮",this);
21     //移动按钮
22     btn2->move( 30 ,30);
23     btn2->move( this->width()/2-btn2->width()/2,this->height()/2-btn2->heigh
ht()/2 );
24 }
```





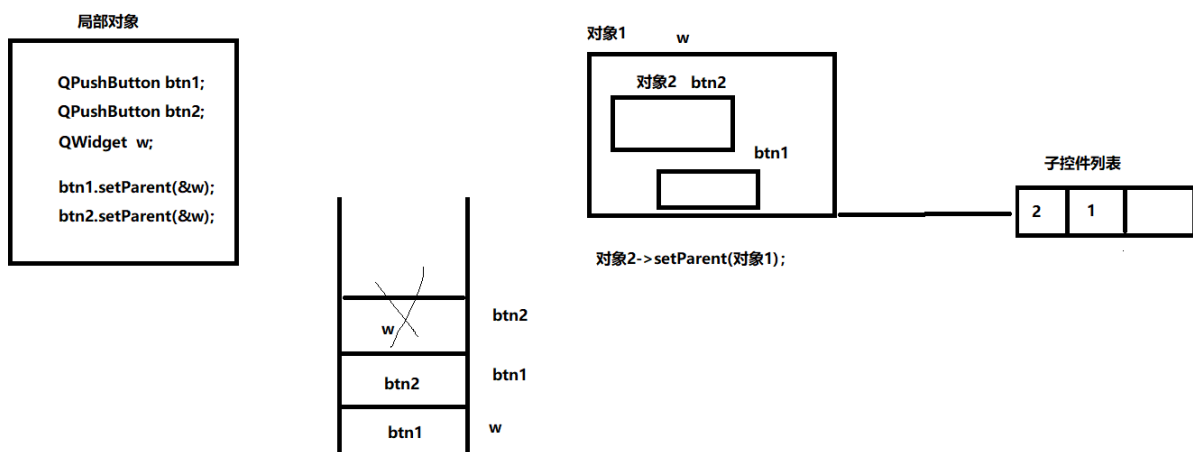
## 知识点5 【对象树】

在创建 QObject 对象时，可以提供一个其父对象，我们创建的这个 QObject 对象会自动添加到其父对象的 children() 列表。□ 当父对象析构的时候，这个列表中的所有对象也会被析构。（注意，这里的父对象并不是继承意义上的父类！）

QWidget 继承自 QObject，因此也继承了这种对象树关系。一个孩子自动地成为父组件的一个子组件

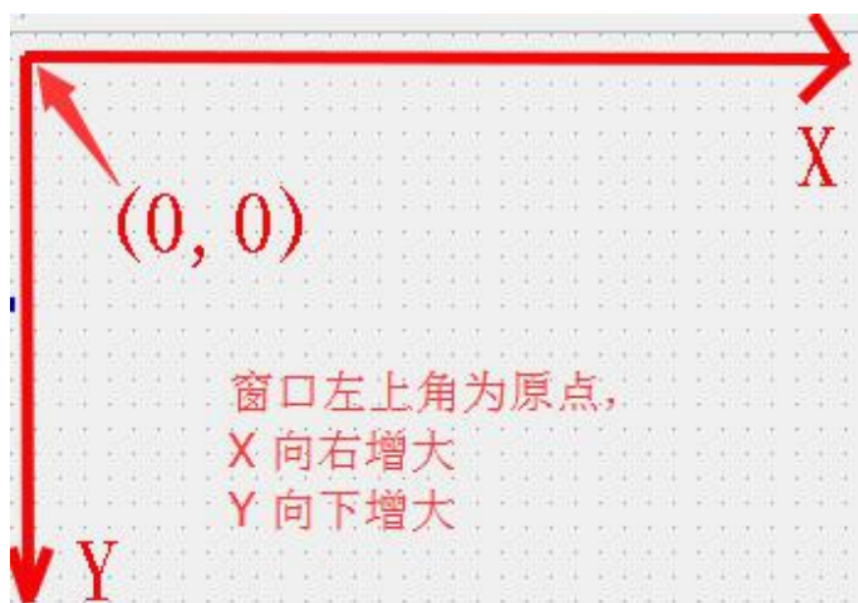
我们也可以自己删除子对象，它们会自动从其父对象列表中删除

综上所述：所有的控件尽量在 **堆区** 创建



## 知识点6 【Qt的坐标体系】

以左上角为原点 (0,0) , X 向右增加, Y 向下增加

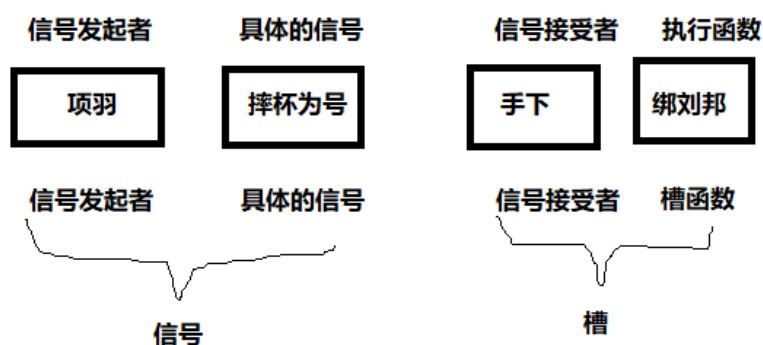


对于嵌套窗口, 其坐标是相对于父窗口来说的。

## 知识点7 【信号和槽机制】 (重要)

项羽绑刘邦

开会作战会议: 项羽 摔杯为号 手下 绑刘邦

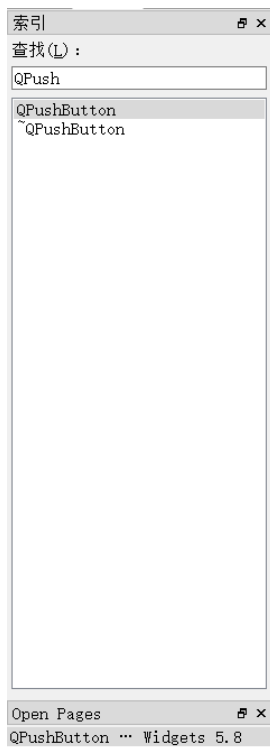


建立 信号 和 槽 之间的关系, 一旦信号发出, 槽函数 立即执行 (异步)

建立信号和槽的关系: `connect( 信号的发起者, 信号, 信号的接收者, 槽函数 )`

### 案例1: 系统的信号和槽

查找QPushButton的信号



## Contents

Properties  
Public Functions  
Reimplemented Public Functions  
Public Slots  
Protected Functions  
Reimplemented Protected Functions  
Detailed Description

无信号 去父类找

## QPushButton Class

The QPushButton widget provides a command button. [More...](#)

Header: `#include <QPushButton>`

qmake: `QT += widgets`

Inherits: [QAbstractButton](#)

Inherited By: [QCommandLinkButton](#)

- [List of all members, including inherited members](#)


## Contents

Properties  
Public Functions  
Public Slots  
Signals  
Protected Functions  
Reimplemented Protected Functions  
Detailed Description

进去

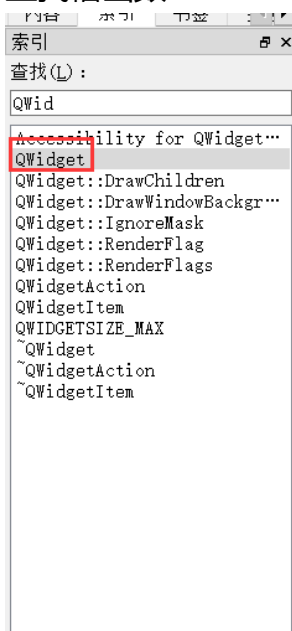
## QAbstractButton Class

# Signals

```
void clicked(bool checked = false)   
void pressed()  
void released()  
void toggled(bool checked)
```

- 3 signals inherited from QWidget
- 2 signals inherited from QObject

查找槽函数：



Qt 5.8 Qt Widgets C++ Classes QWidget

## Contents

Public Types  
Properties  
Public Functions  
Reimplemented Public Functions  
Public Slots  槽函数  
Signals  
Static Public Members  
Protected Functions  
Reimplemented Protected Functions  
Protected Slots

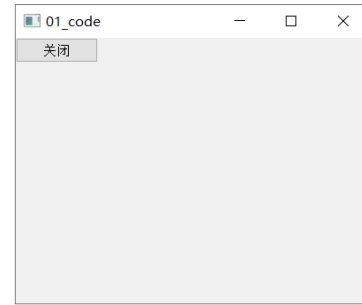
# Public Slots

```
bool close()  
void hide()  
void lower()  
void raise()  
void repaint()  
void setDisabled(bool disable)
```

```

#include "widget.h"
#include <QPushButton>
Widget::Widget(QWidget *parent)
    : QWidget(parent)
{
    //设置主窗口大小
    this->resize(400,300);
    //创建一个按钮
    QPushButton *btn = new QPushButton("关闭", this);
    #if 1
        // 建立信号和槽的关系(Qt5的方式)
        connect( btn, &QPushButton::clicked ,this, &Widget::close );
    #else
        // 建立信号和槽的关系(Qt4的方式) SIGNAL将（）里面当成字符串看待 不会判断信号是否存在
        connect( btn, SIGNAL(clicked()) ,this, SLOT(close()) );
    #endif
}

```



## 案例2：自定义信号和槽函数

### 定义信号：

在关键字signals下实现：

```

class Teacher : public QWidget
{
    Q_OBJECT
public:
    explicit Teacher(QWidget *parent = 0);

signals:
    //信号的返回值类型为void 可以有参（重载），不能实现函数体
    void hungry(void);

public slots:
};

```

### 定义槽函数：

在关键字public slots下实现：

```

3
4 #include <QWidget>
5 #include <QDebug>
6 class Student : public QWidget
7 {
8     Q_OBJECT
9 public:
10     explicit Student(QWidget *parent = 0);
11
12 signals:
13
14 public slots:
15     //槽函数返回值类型为void 可以有参数（重载） 必须实现函数体
16     void treat(void)
17     {
18         qDebug()<<"请老师吃饭了"<<endl;
19     }
20 };
21

```

```

4 #include <QWidget>
5 #include <QDebug>
6 class Teacher : public QWidget
7 {
8     Q_OBJECT
9 public:
10     explicit Teacher(QWidget *parent = 0);
11
12 signals:
13     //信号的返回值类型为void 可以有参（重载），不能实现函数体
14     void hungry(void);
15
16 public slots:
17     //下课的槽函数
18     void classOver(void)
19     {
20         qDebug()<<"下课了"<<endl;
21         //emit 发出老师饿了信号
22         emit this->hungary();
23     }
24 };
25

```

widget.cpp的构造函数中

```

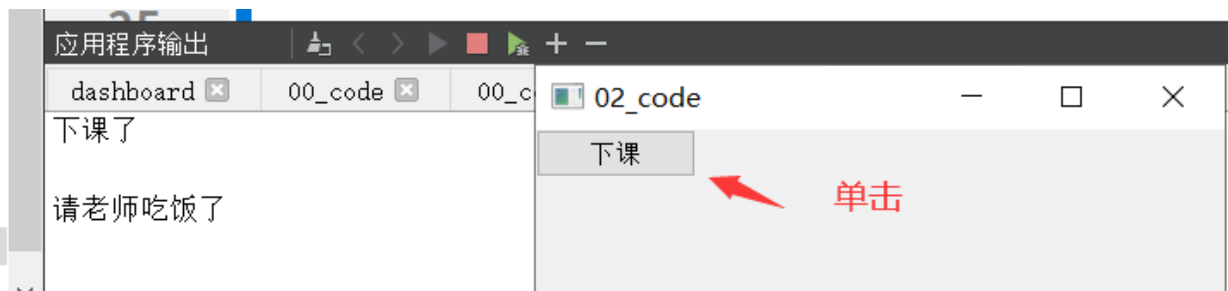
1 #include "widget.h"
2 #include "teacher.h"
3 #include "student.h"
4 #include <QPushButton>
5 Widget::Widget(QWidget *parent)
6 : QWidget(parent)

```

```

7 {
8     this->resize(400,300);
9     //实例化老师的控件
10    Teacher *tea = new Teacher(this);
11    //实例化学生的控件
12    Student *stu = new Student(this);
13
14    //建立老师和学生的关系
15    #if 1
16        //connect(tea,饿了,stu,请客吃放);
17        connect(tea, &Teacher::hungury ,stu, &Student::treat);
18    #else
19
20    #endif
21    //创建一个按钮
22    QPushButton *btn = new QPushButton("下课", this);
23    connect(btn, &QPushButton::clicked, tea, &Teacher::classOver );
24 }
25
26 Widget::~Widget()
27 {
28
29 }

```



### 信号和槽之间的参数传递

```

5 class Teacher : public QWidget
6 {
7     Q_OBJECT
8 public:
9     explicit Teacher(QWidget *parent = 0);
10
11 signals:
12     //信号的返回值类型为void 可以有参（重载），不能实现函数体
13     void hungry(void);
14     void hungry(QString foodName);
15 }

```

```

class Student : public QWidget
{
    Q_OBJECT
public:
    explicit Student(QWidget *parent = 0);

signals:

public slots:
    //槽函数返回值类型为void 可以有参数（重载） 必须实现函数体
    void treat(void)
    {
        qDebug()<<"请老师吃饭了"<<endl;
    }
    void treat(QString foodName)
    {
        qDebug()<<"请老师吃了"<<foodName<<endl;
    }
};

```

widget.cpp构造函数中(Qt5的方式)

```

//建立老师和学生的关系
#if 0
//connect(tea,饿了,stu,请客吃放);
void (Teacher:: *p1)() = &Teacher::hungury;
void (Student:: *p2)() = &Student::treat;
connect(tea, p1 ,stu, p2);
#else
//当信号 或 槽 发生了重载 使用函数指针 做出 区分
void (Teacher:: *p1)(QString) = &Teacher::hungury;
void (Student:: *p2)(QString) = &Student::treat;
connect(tea, p1 ,stu, p2);
#endif
//创建一个按钮
QPushButton *btn = new QPushButton("下课", this);
connect(btn, &QPushButton::clicked, tea, &Teacher::classOver );

```

teacher.h



```

12 signals:
13     //信号的返回值类型为void 可以有参（重载），不能实现函数体
14     void hungry(void);
15     void hungry(QString foodName);
16
17 public slots:
18     //下课的槽函数
19     void classOver(void)
20     {
21         qDebug() << "下课了" << endl;
22         //emit 发出老师饿了信号
23
24         //| emit this->hungry();
25
26         emit this->hungry("锅包肉");
27     }
28 };
29
30

```

widget.cpp构造函数中(Qt4的方式)

```

3 //实例化老师的控件
4 Teacher *tea = new Teacher(this);
5 //实例化学生的控件
6 Student *stu = new Student(this);
7
8 //建立老师和学生的关系
9
10 #if 0
11     connect( tea, SIGNAL(hungry()), stu, SLOT(treat()) );
12 #else
13     connect( tea, SIGNAL(hungry(QString)), stu, SLOT(treat(QString)) );
14 #endif
15
16 //创建一个按钮
17 QPushButton *btn = new QPushButton("下课", this);
18 connect(btn, &QPushButton::clicked, tea, &Teacher::classOver );
19
20 }
21
22 ~Widget()
23 {
24
25 }

```

## 信号和槽的注意:

一个信号 连接 多个槽

多个信号 连接 一个槽

disconnect取消信号和槽的连接

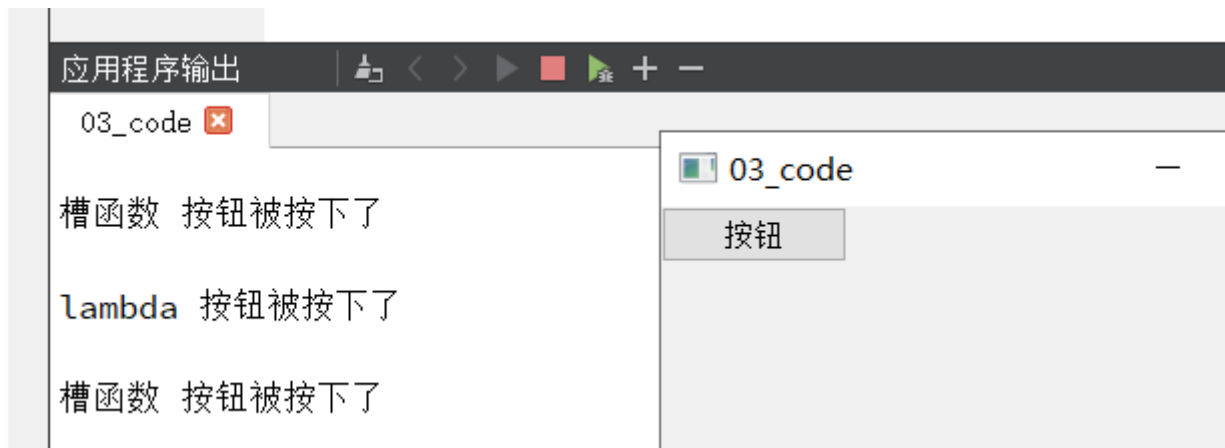
槽函数的参数 一定要 小于等于 信号的参数个数。

## 知识点8 【lambda表达式】（了解）

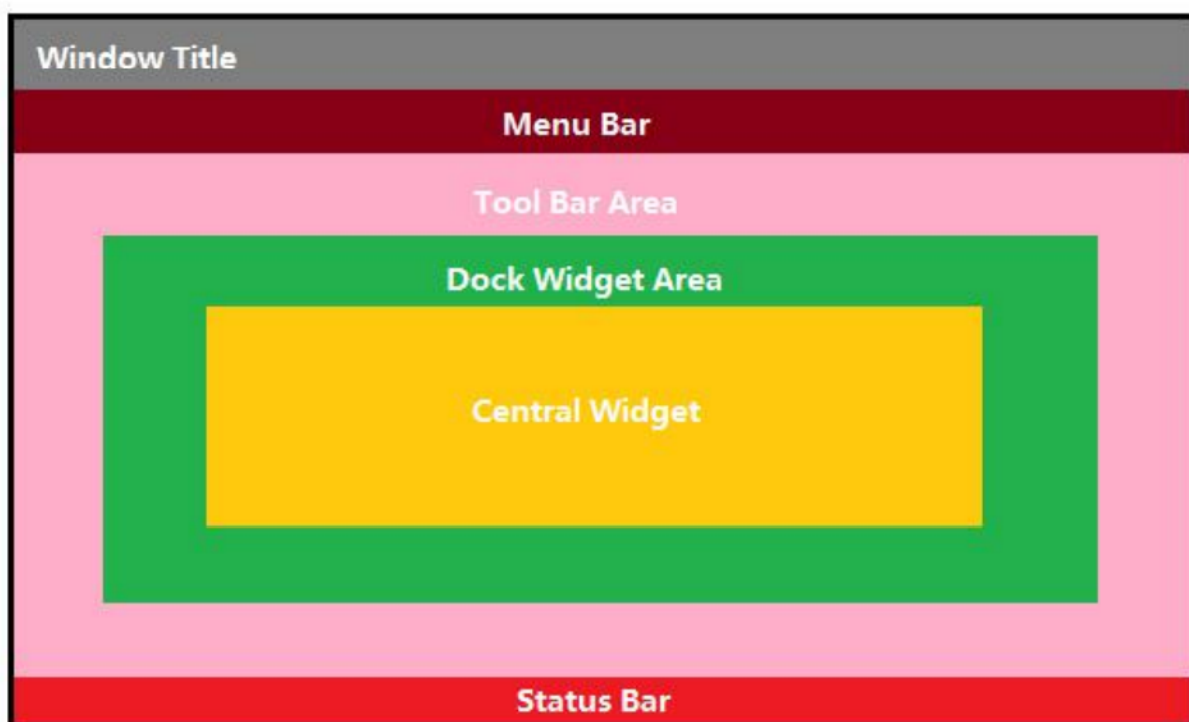
```
✓ class Widget : public QWidget
{
    Q_OBJECT

public:
    Widget(QWidget *parent = 0);
    ~Widget();
public slots:
    void printMsg(void);
};
```

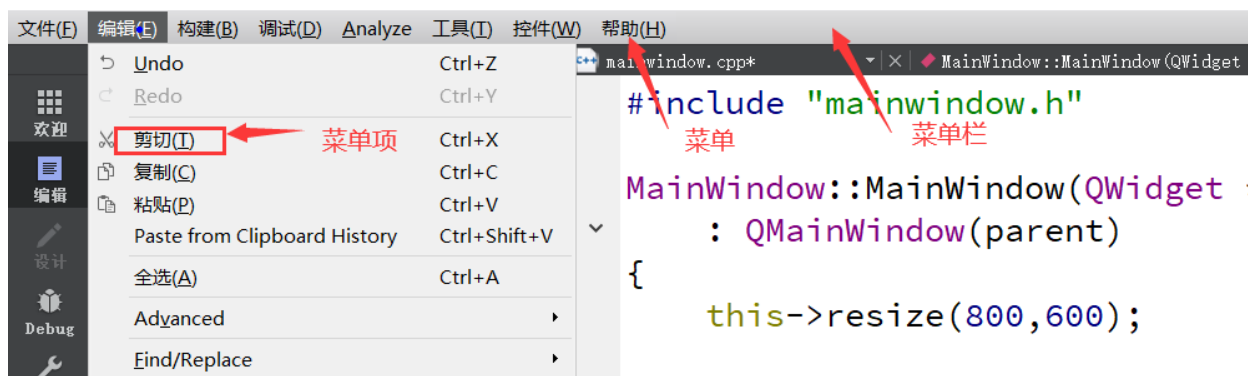
```
1 Widget::Widget(QWidget *parent)
2   : QWidget(parent)
3   {
4     this->resize(400,300);
5     QPushButton *btn = new QPushButton("按钮", this);
6
7     //lambda 表达式
8     connect(btn, &QPushButton::clicked, [&]() {
9         qDebug() << "lambda 按钮被按下了" << endl;
10    } );
11
12    connect(btn, &QPushButton::clicked, this, &Widget::printMsg );
13 }
14 void Widget::printMsg()
15 {
16     qDebug() << "槽函数 按钮被按下了" << endl;
17 }
```



## 知识点9 【QMainWindow窗口】（了解）



QMainWindow带菜单栏的窗口



### 1、创建菜单栏（QMenuBar）

```
1 //创建 菜单栏
2 QMenuBar *menuBar = new QMenuBar(this);
3 //将菜单栏 添加到主窗口的特殊位置
4 this->setMenuBar(menuBar);
```

## 2、创建菜单 (QMenu)

```
1 //创建菜单
2 QMenu *file = new QMenu("文件", this);
3 QMenu *edit = new QMenu("编辑", this);
4 //将菜单 放入 菜单栏中
5 menuBar->addMenu(file);
6 menuBar->addMenu(edit);
```

## 3、创建菜单项 (QAction)

```
1 //创建菜单项
2 QAction *New = new QAction("新建", this);
3 QAction *Open = new QAction("打开", this);
4 //将菜单项 添加到菜单中
5 file->addAction(New);
6 file->addAction(Open);
7 edit->addActions(QList<QAction *>()<<New<<Open);
```

## 4、让菜单项动起来

```
1 connect(New, &QAction::triggered, [&]() {
2     qDebug()<<"新建文件了"<<endl;
3 } );
4 connect(Open, &QAction::triggered, [&]() {
5     qDebug()<<"打开文件了"<<endl;
6 } );
```

## 5、设置菜单项的快捷方式

```
1 New->setShortcut(QKeySequence(Qt::CTRL+Qt::Key_N));
2 Open->setShortcut(tr("Ctrl+o"));
```

```
QKeySequence(QKeySequence::Print);
QKeySequence(tr("Ctrl+P"));
QKeySequence(tr("Ctrl+p"));
QKeySequence(Qt::CTRL + Qt::Key_P);
```

## 6、添加一个分隔符

```
1 //将菜单项 添加到菜单中
2 file->addAction(New);
```

```
3 //添加一个分隔符
4 file->addSeparator();
5 file->addAction(Open);
```

## 7、创建工具栏QToolBar

```
1 QToolBar *toolBar = new QToolBar(this);
2 //将工具栏 放入主窗口的特定位置
3 this->addToolBar(toolBar);
4 //将菜单项 放入工具栏中
5 toolBar->addAction(New);
6 toolBar->addAction(Open);
7 //工具栏默认可以浮动、可以停靠四周
8 //不允许工具栏浮动false
9 toolBar->setFloatable(false);
10 //只允许工具栏停靠左右
11 toolBar->setAllowedAreas(Qt::LeftToolBarArea|Qt::RightToolBarArea);
```

## 8、创建状态栏

```
1 QStatusBar *statusBar = new QStatusBar(this);
2 //将状态栏 添加到 主窗口中
3 this->setStatusBar(statusBar);
4 //在状态栏上 添加左侧信息
5 QLabel *label1= new QLabel("左侧提示信息", this);
6 statusBar->addWidget(label1);
7 //在状态栏上 添加右侧信息
8 QLabel *label2= new QLabel("右侧提示信息", this);
9 statusBar->addPermanentWidget(label2);
```

## 9、创建中心部件

```
1 //将文本框作为中心部件
2 QTextEdit *textEdit = new QTextEdit("这是中心部件", this);
3 this->setCentralWidget(textEdit);
```

## 10、创建铆接部件

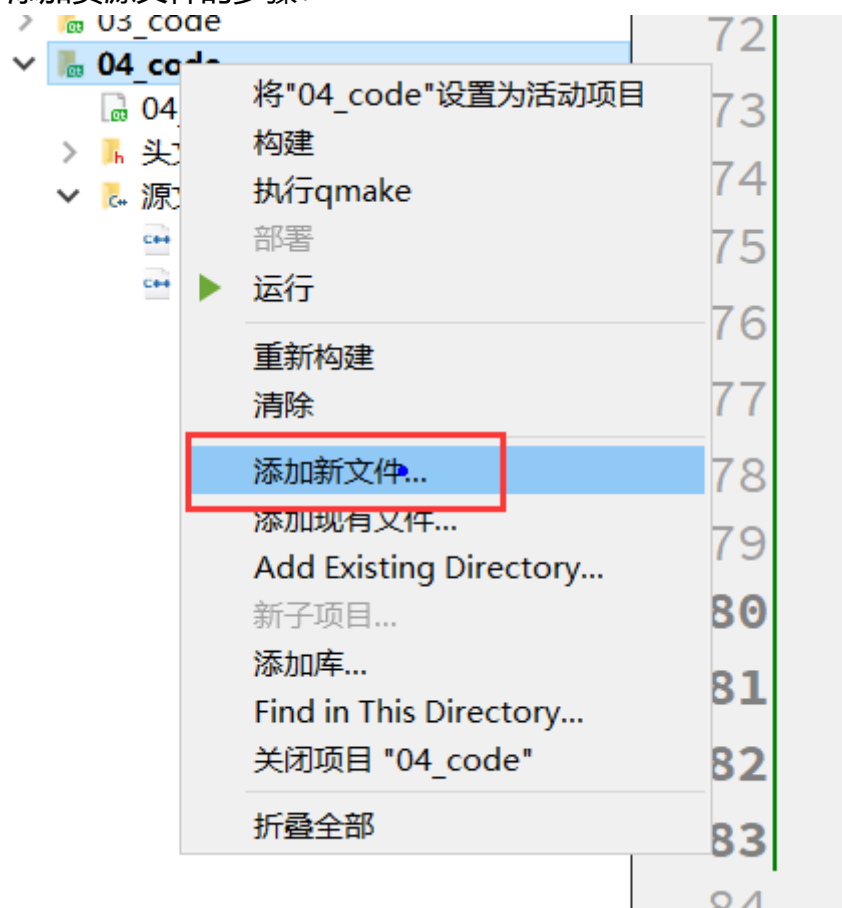
```
1 QDockWidget *dock = new QDockWidget("我是铆接部件",this);
2 //将铆接部件 添加到主窗口中
3 this->addDockWidget(Qt::AllDockWidgetAreas, dock);
```

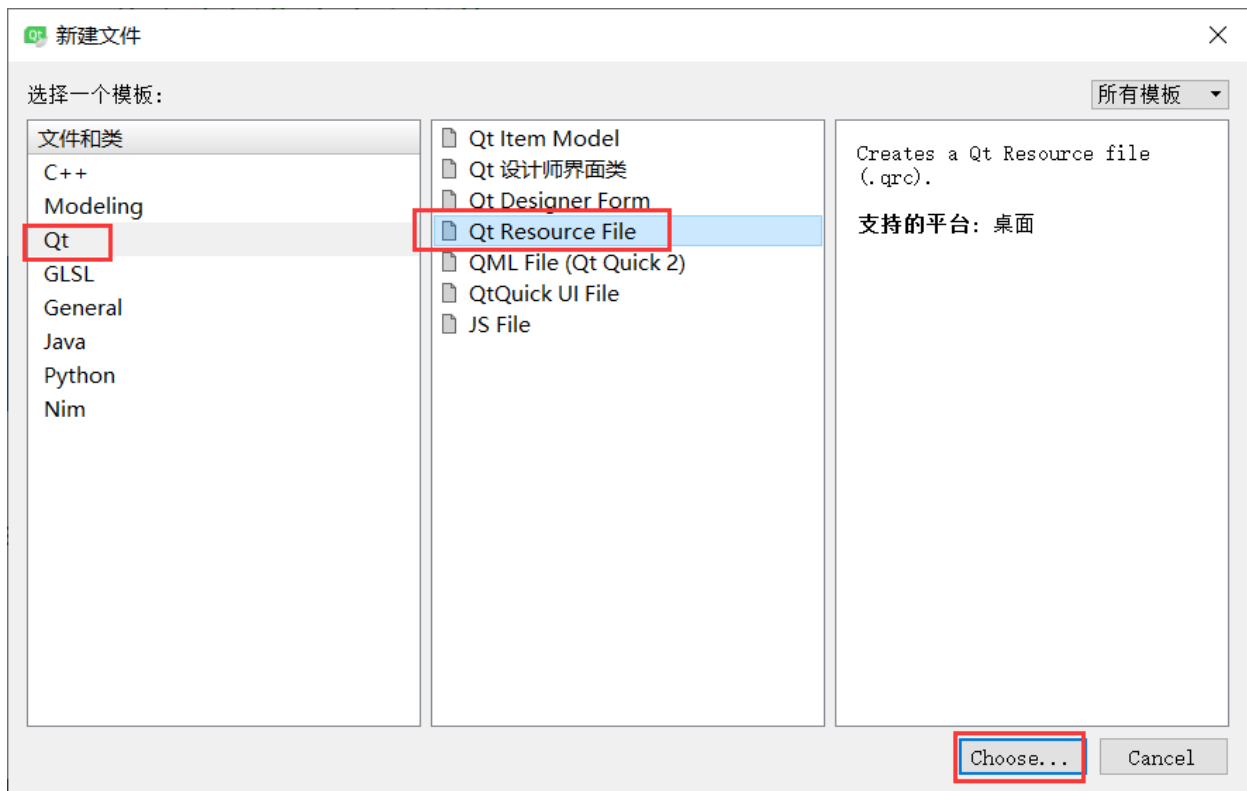
# 知识点10【资源文件】（了解）

以添加图片资源为例：

« bk2103 > code > c++ > day09 > 04_code >			▼	(
名称		修改日期		
image		2021/8/23 11:		
04_code.pro		2021/8/23 9:3		
04_code.pro.user		2021/8/23 9:3		
main.cpp		2021/8/23 9:3		
mainwindow.cpp		2021/8/23 11:		
mainwindow.h		2021/8/23 9:3		

添加资源文件的步骤：





×

## Qt Resource File

Location  
Summary

### Location

可以任意

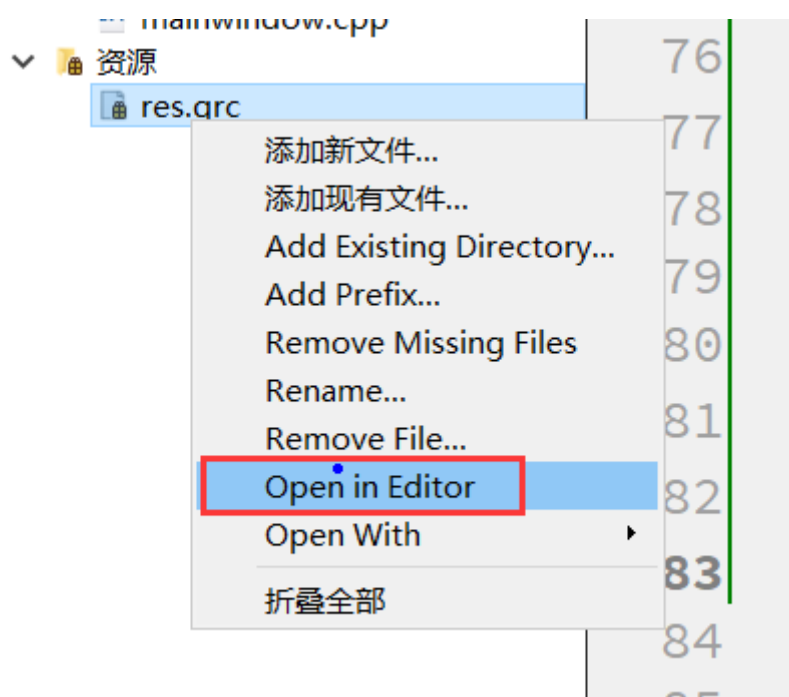
名称:

路径:

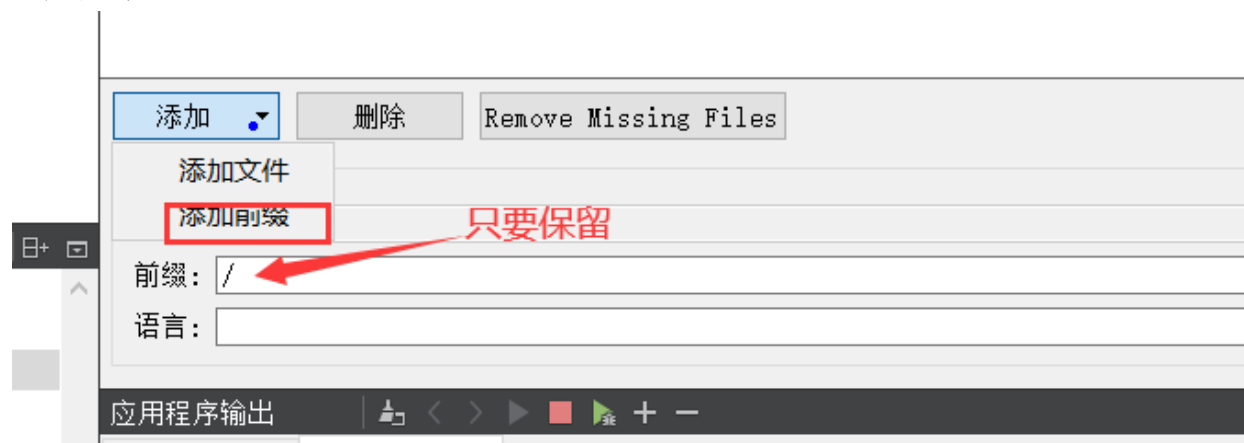
下一步(N)

取消

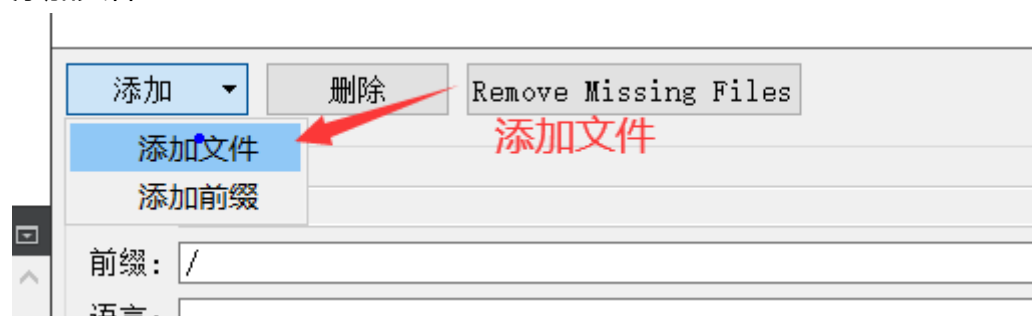
如果单击了其他页面 需要回到资源管理页面



添加前缀:

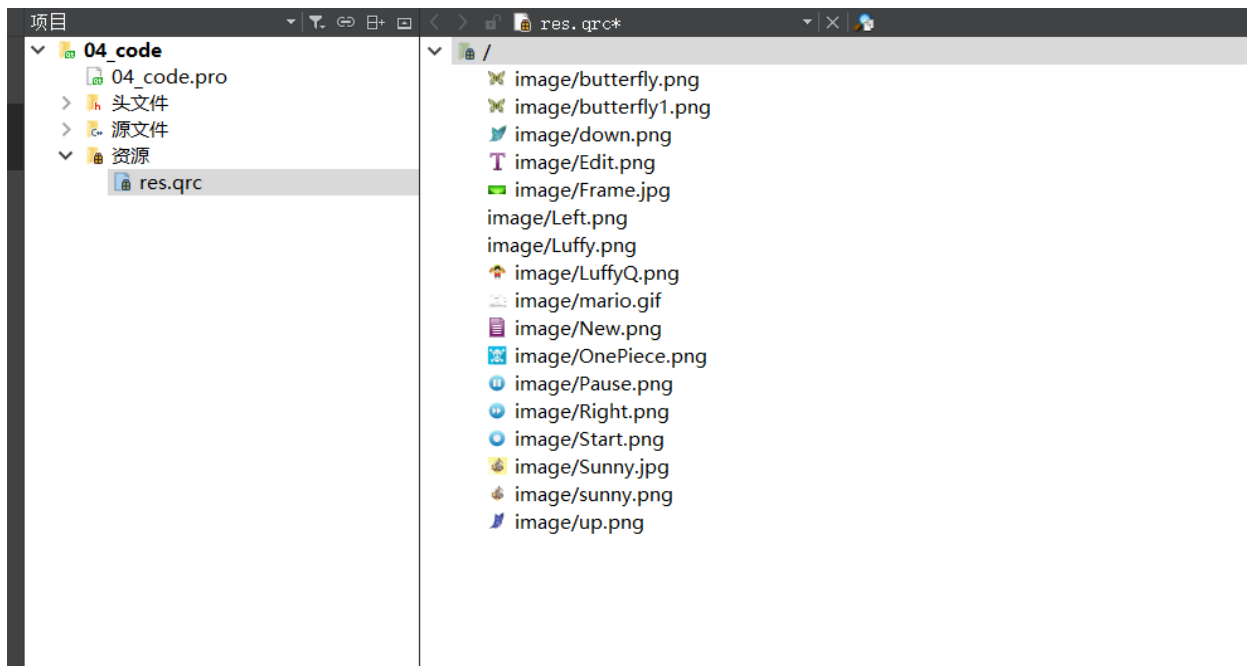
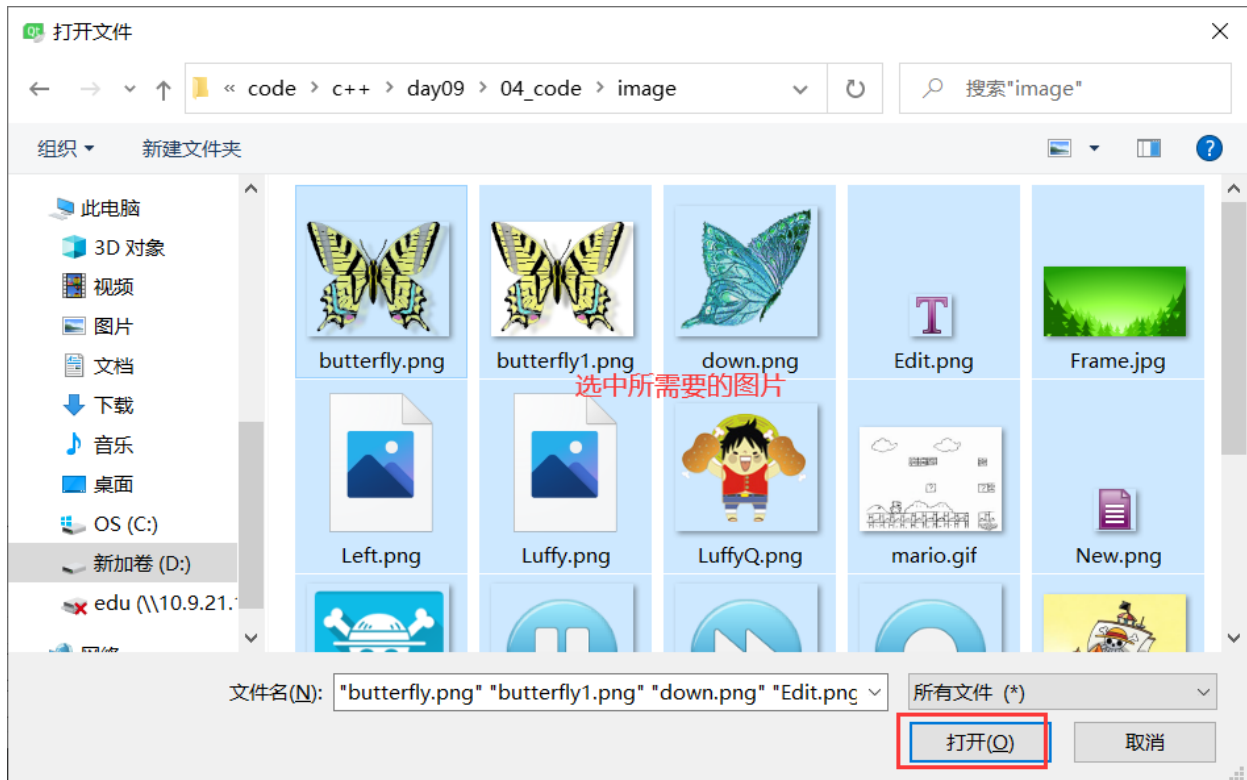


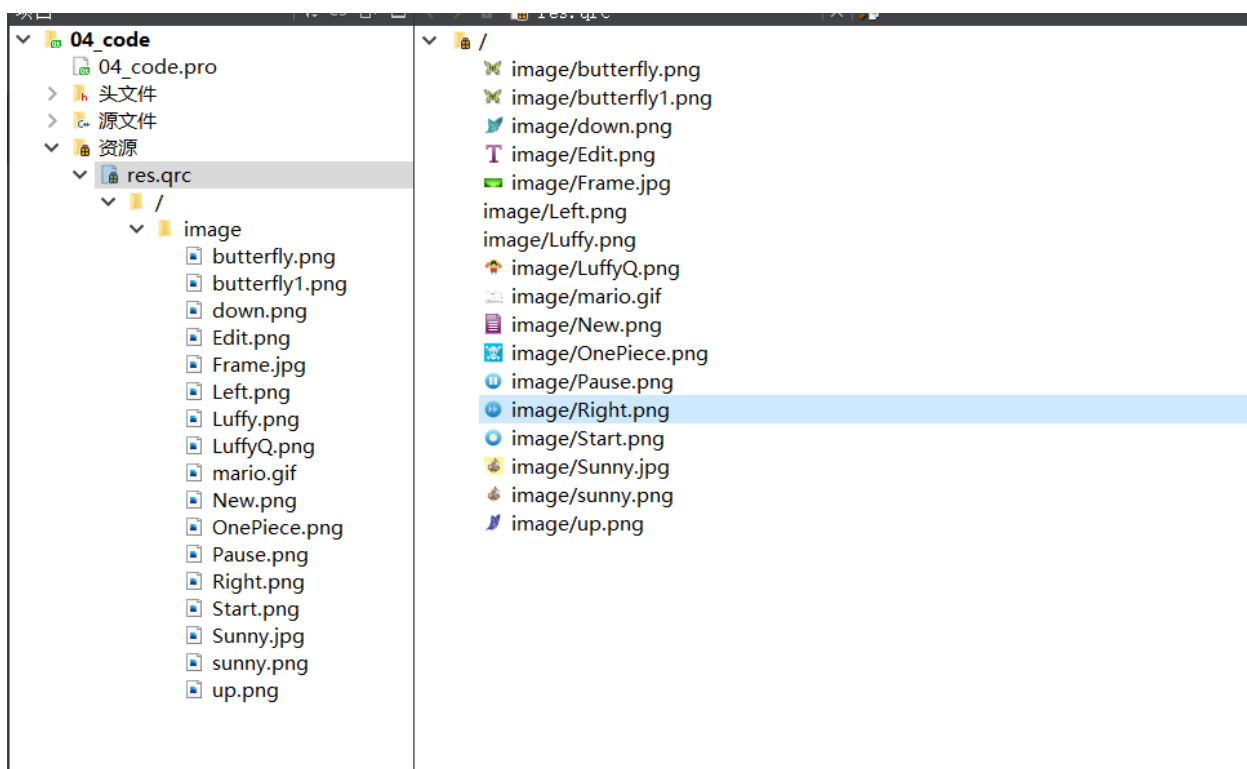
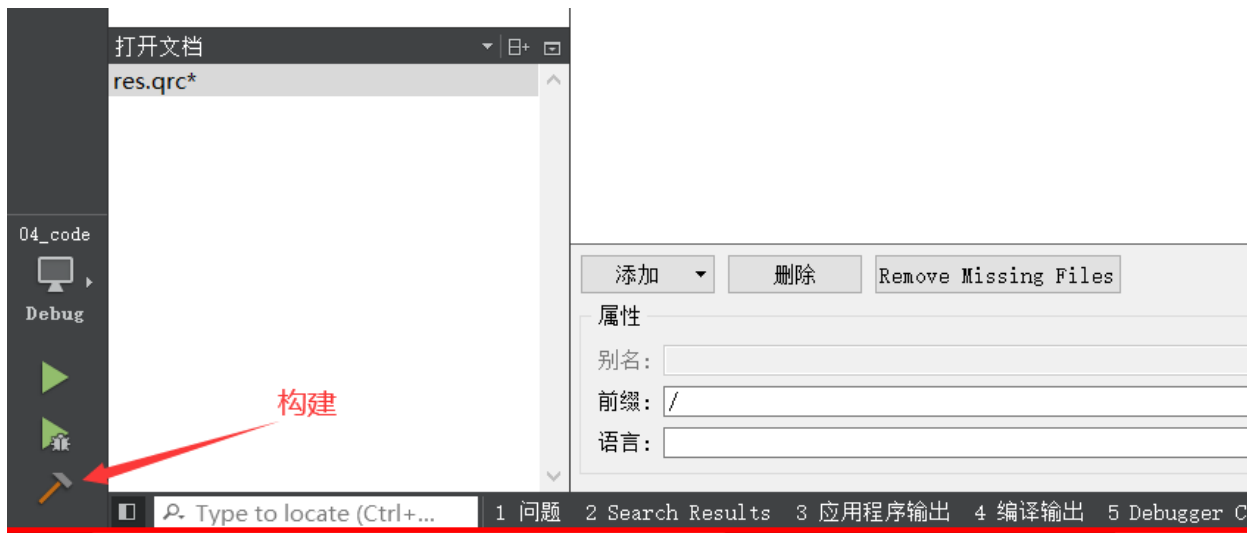
添加文件



选择图片







给菜单项添加图标 :代表的是资源文件

```
1 New->setIcon(QIcon(":/image/New.png"));
2 Open->setIcon(QIcon(":/image/Edit.png"));
```

## 知识点11 【UI文件的使用】

✕

← Qt Widgets Application

Location  
Kits  
Details  
汇总

### 类信息

指定您要创建的源码文件的基本类信息。

类名(C):

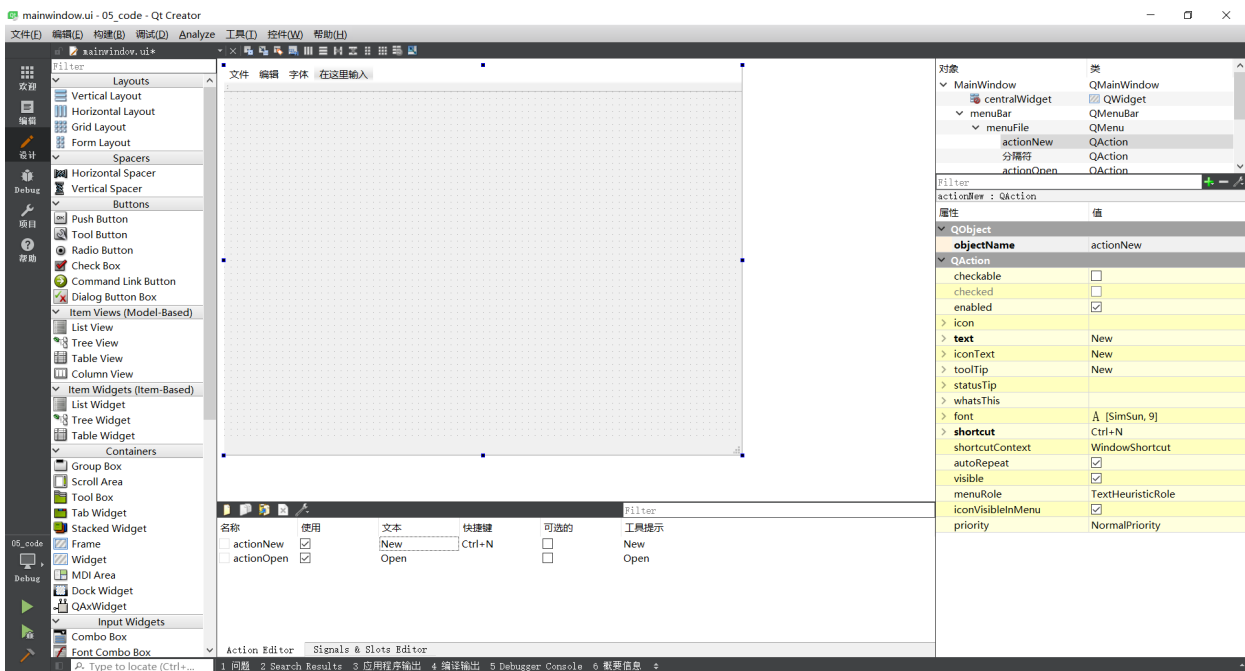
基类(B):

头文件(H):

源文件(S):

创建界面(G): ☒ ← 勾上

界面文件(F):



## 知识点12【对话框】

### 1、模态对话框 和 非模态对话框

**模态**对话框，就是会阻塞同一应用程序中其它窗口的输入

QDialog::exec()、QDialog::open()

**非模态**对话框，不会阻塞同一应用程序中其它窗口的输入

QDialog::show()

### 2、消息对话框QMessageBox

模态对话框，用于显示信息、询问问题等。

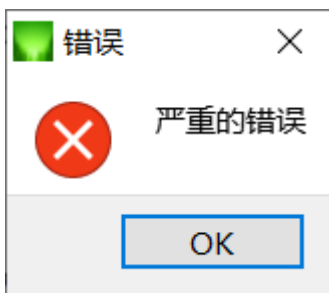
## Static Public Members

```
void about(QWidget *parent, const QString &title, const QString &text)
void aboutQt(QWidget *parent, const QString &title = QString())
StandardButton critical(QWidget *parent, const QString &title, const QString &text, StandardButtons buttons = Ok,
                        StandardButton defaultButton = NoButton)
StandardButton information(QWidget *parent, const QString &title, const QString &text, StandardButtons buttons = Ok,
                           StandardButton defaultButton = NoButton)
StandardButton question(QWidget *parent, const QString &title, const QString &text, StandardButtons buttons =
                        StandardButtons( Yes | No ), StandardButton defaultButton = NoButton)
StandardButton warning(QWidget *parent, const QString &title, const QString &text, StandardButtons buttons = Ok,
                       StandardButton defaultButton = NoButton)

• 5 static public members inherited from QWidget
• 10 static public members inherited from QObject
```

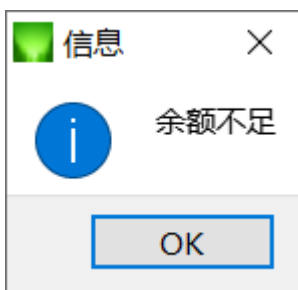
### 错误对话框:

```
1 QMessageBox::critical(this, "错误", "严重的错误");
```



### 信息对话框

```
1 QMessageBox::information(this, "信息", "余额不足");
```



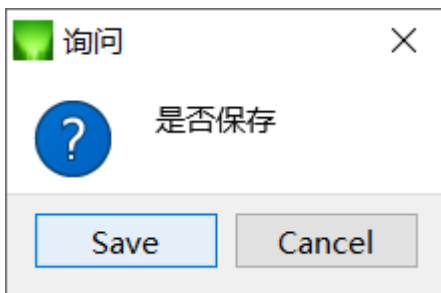
### 询问对话框

```
1 #if 0
2 QMessageBox::StandardButton ret;
3 ret = QMessageBox::question(this, "询问", "是否保存");
4 if(ret == QMessageBox::Yes)
5 {
6     qDebug() << "正在下载中" << endl;
7 }
8 else if(ret == QMessageBox::No)
9 {
10    qDebug() << "取消下载" << endl;
11 }
```

```

12 #endif
13 #if 1
14 QMessageBox::StandardButton ret;
15 ret = QMessageBox::question(this, "询问", "是否保存", QMessageBox::Save | QMe
ssageBox::Cancel);
16 if(ret == QMessageBox::Save)
17 {
18     qDebug()<<"正在下载中"<<endl;
19 }
20 else if(ret == QMessageBox::Cancel)
21 {
22     qDebug()<<"取消下载"<<endl;
23 }
24 #endif

```

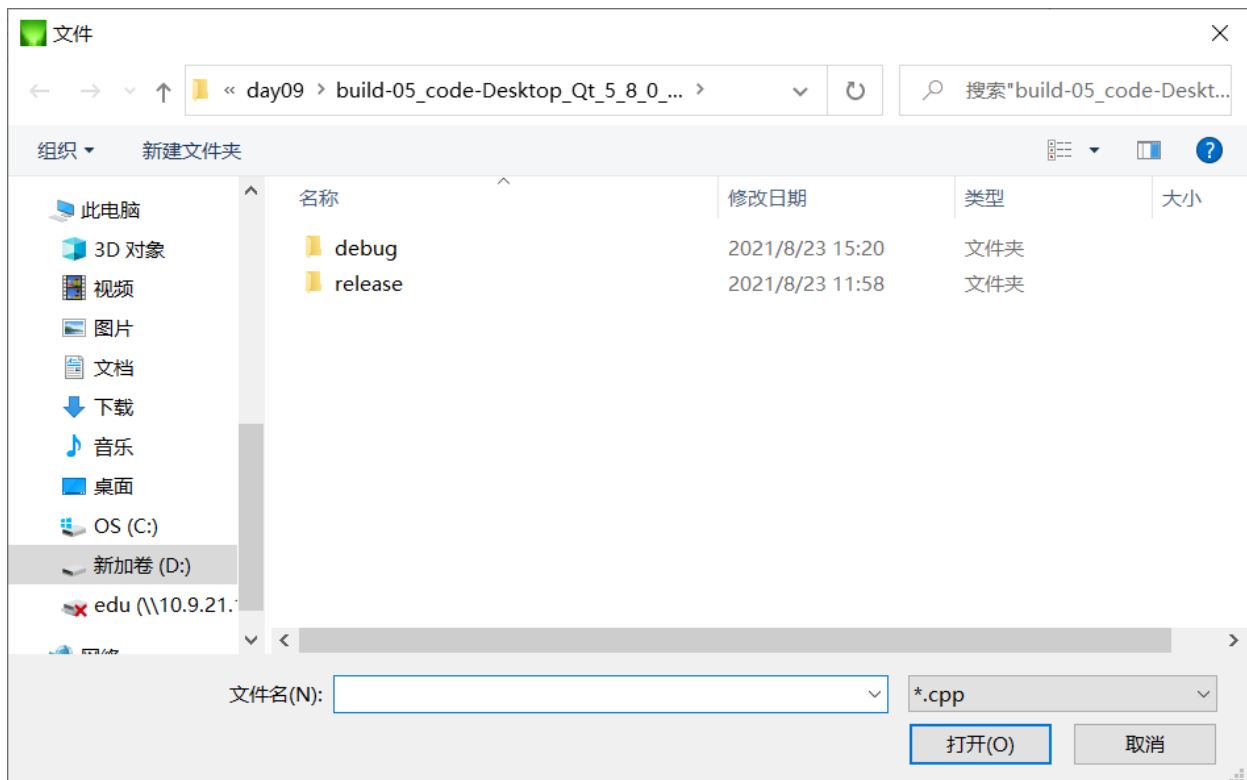


### 3、文件对话框QFileDialog

```

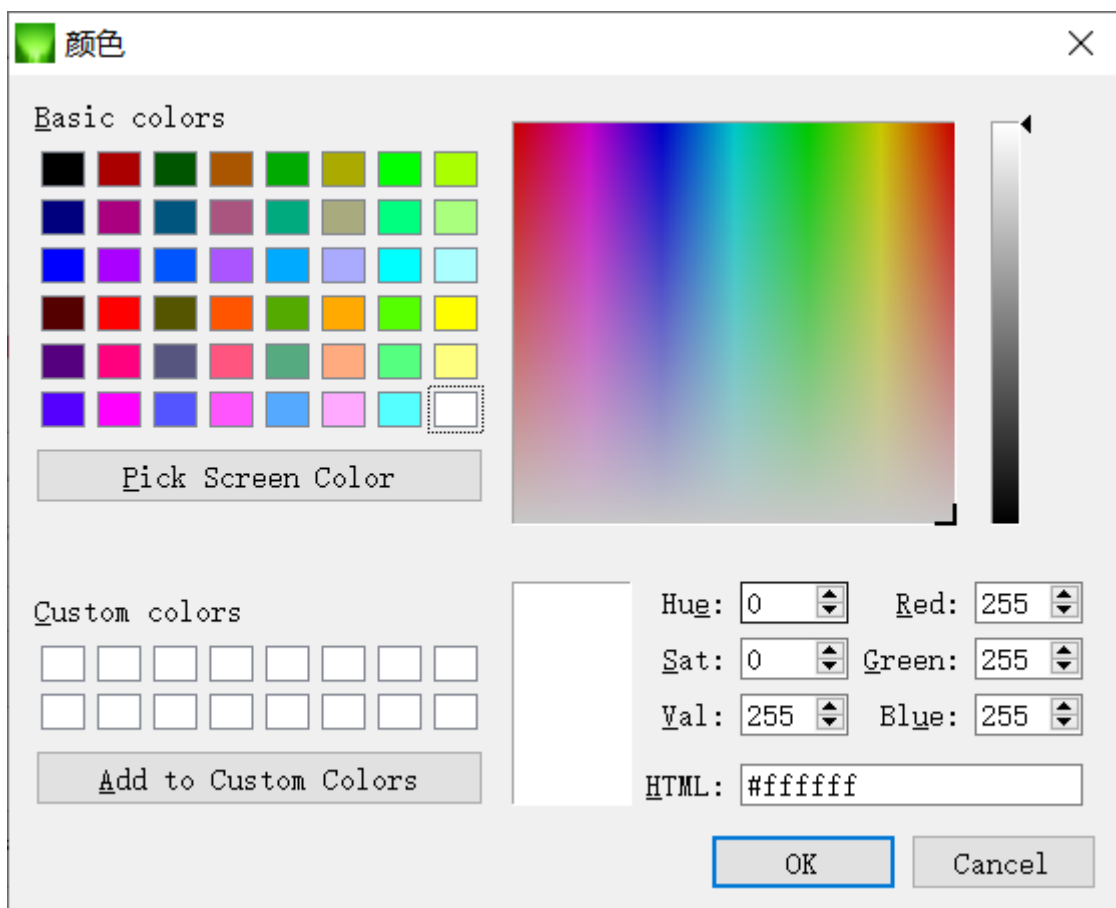
1 QString fileName = QFileDialog::getOpenFileName(this, "文件", "./", "*.cpp");
2 qDebug()<<fileName<<endl;

```



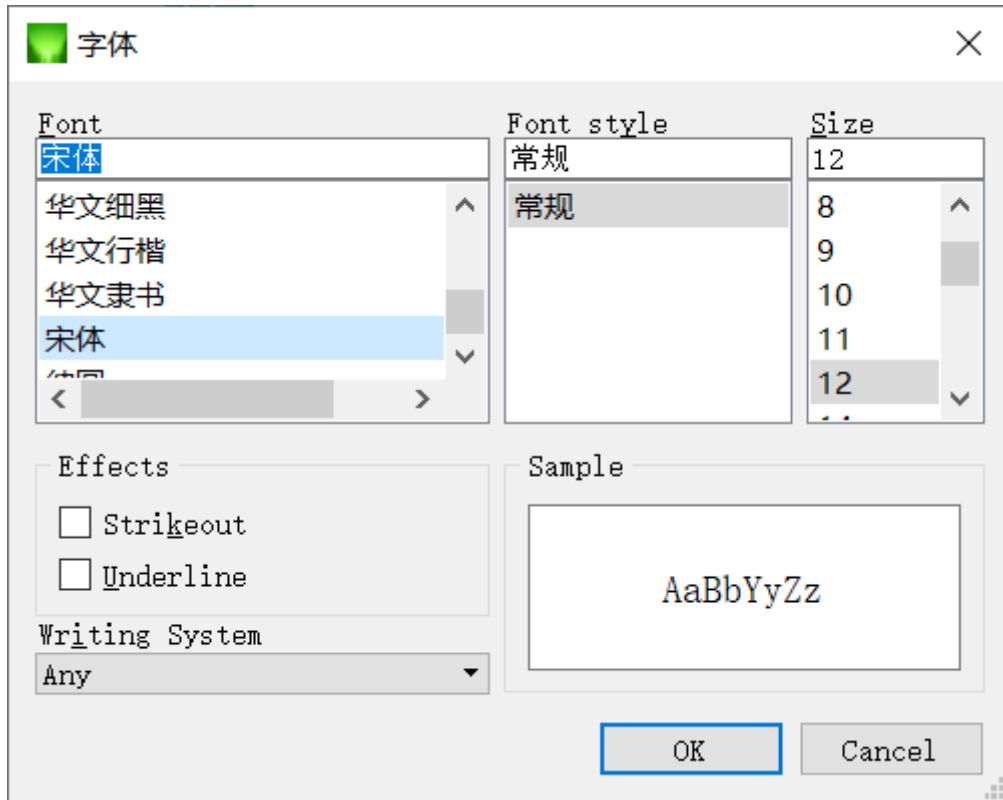
## 4、颜色对话框

```
1 QColor color = QColorDialog::getColor(Qt::white, this, "颜色");
2 qDebug() << color.red() << " " << color.green() << " " << color.blue() << endl;
```



## 5、字体对话框QFontDialog

```
1 bool yes=true;
2 QFont font = QFontDialog::getFont(&yes, QFont("宋体"), this, "字体");
3 qDebug()<<font.family()<<" "<<font.pointSize()<<endl;
```



## 知识点13【布局管理】

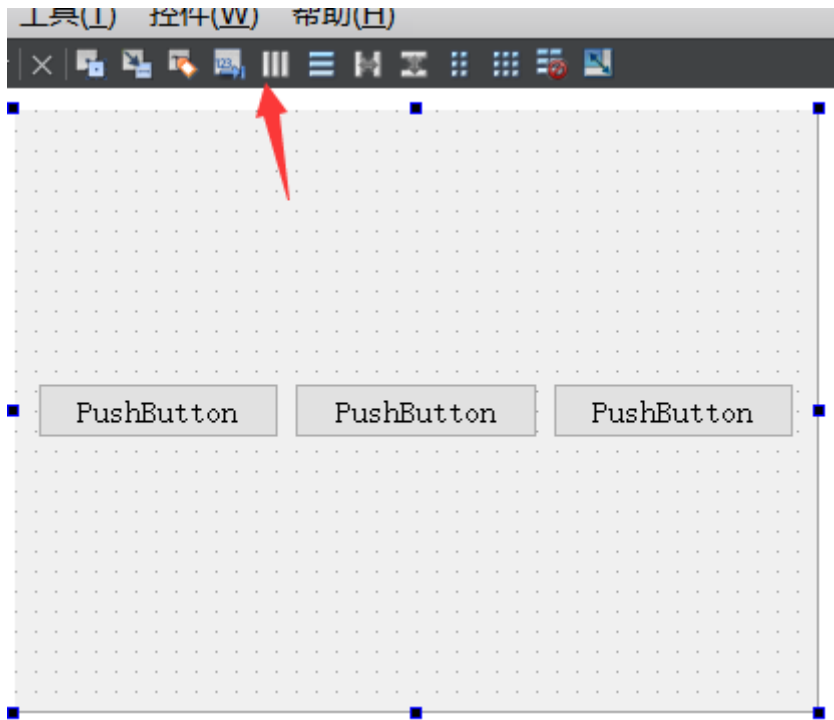
Qt 提供的布局中以下三种是我们最常用的：

QHBoxLayout：按照水平方向从左到右布局；

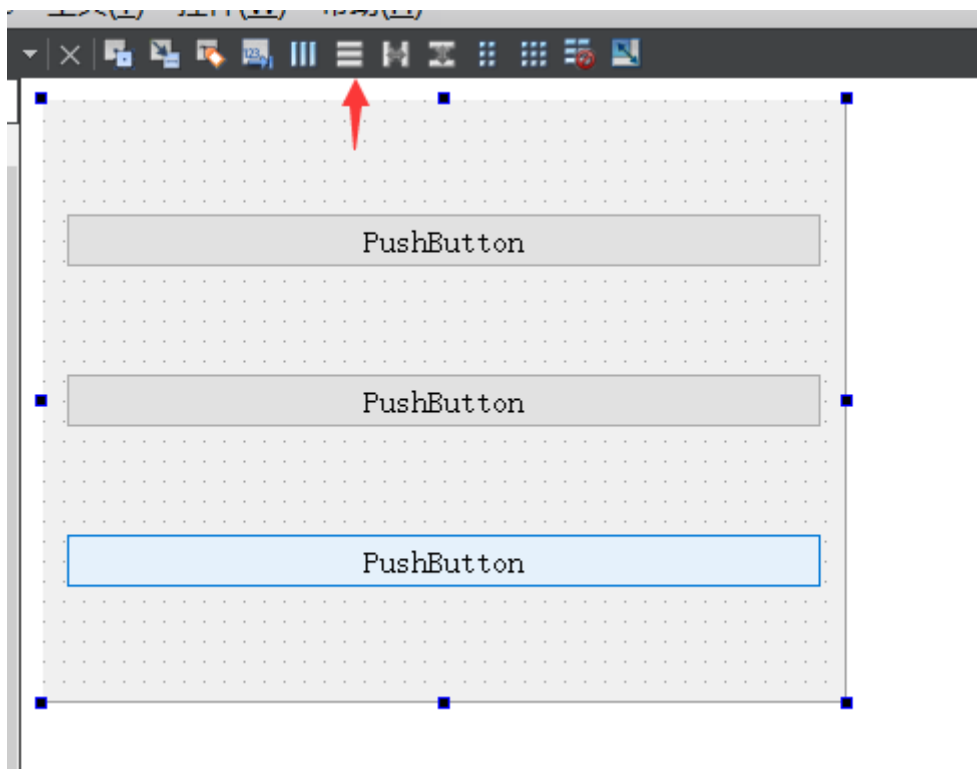
QVBoxLayout：按照竖直方向从上到下布局；

QGridLayout：在一个网格中进行布局，类似于 HTML 的 table

### 1、水平布局QHBoxLayout

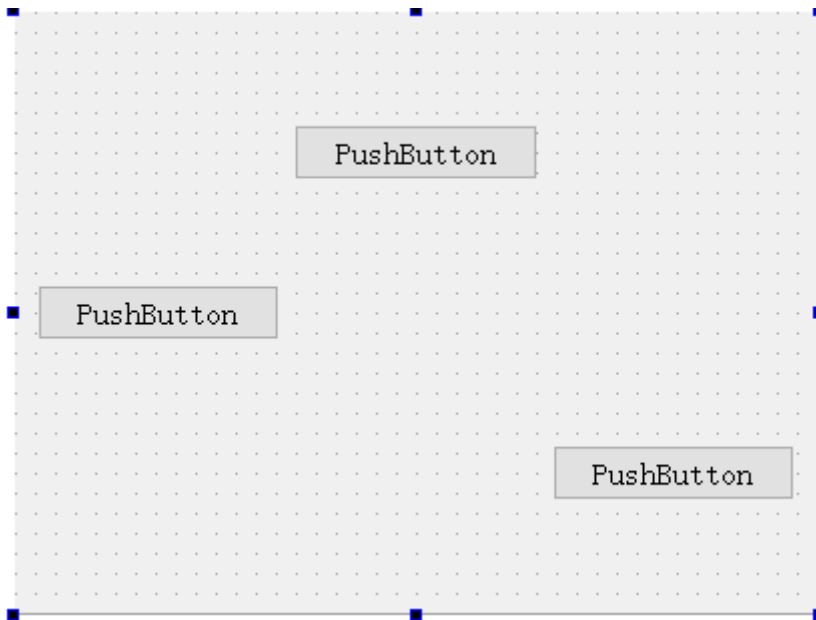


## 2、垂直布局QVBoxLayout



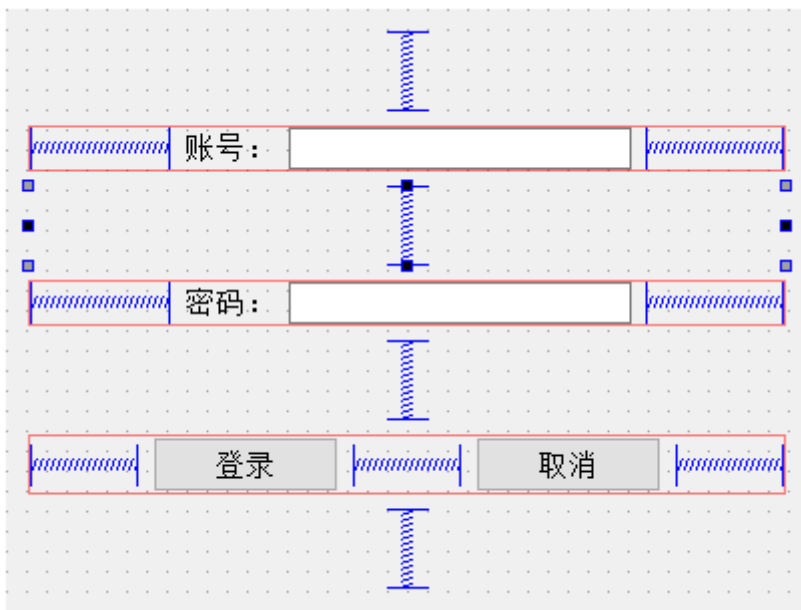
## 3、栅格布局QGridLayout



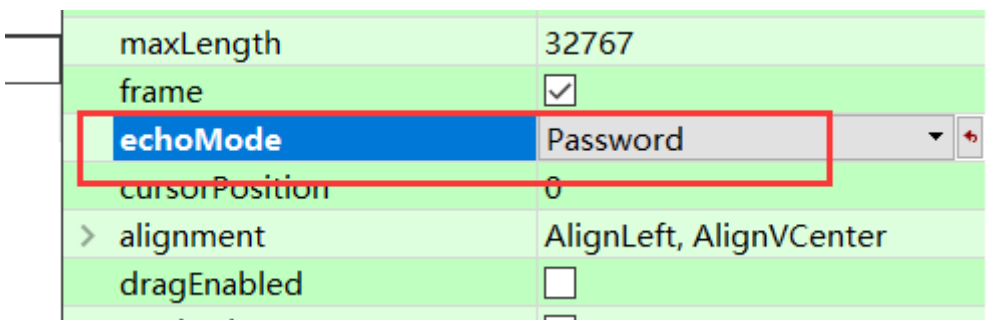


## 知识点14 【登录器】

### 1、界面布局



### 2、输入框的密码模式



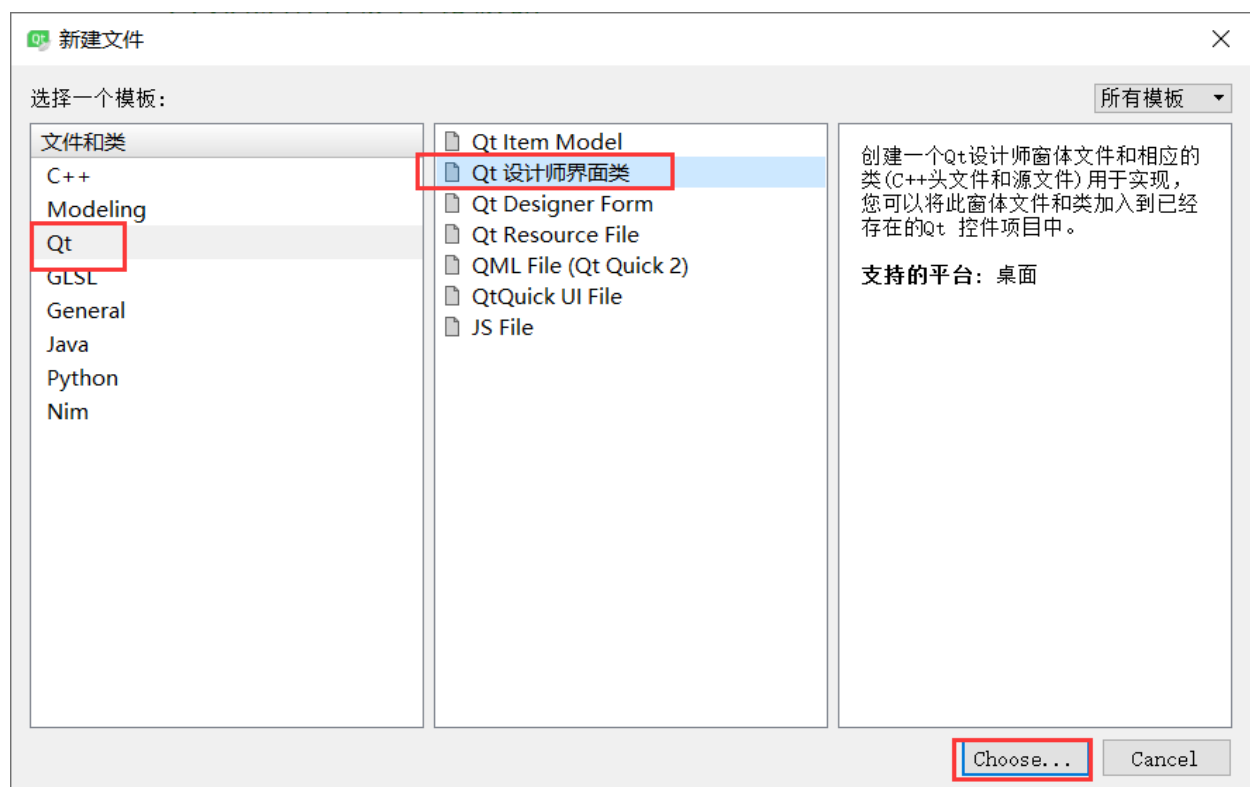
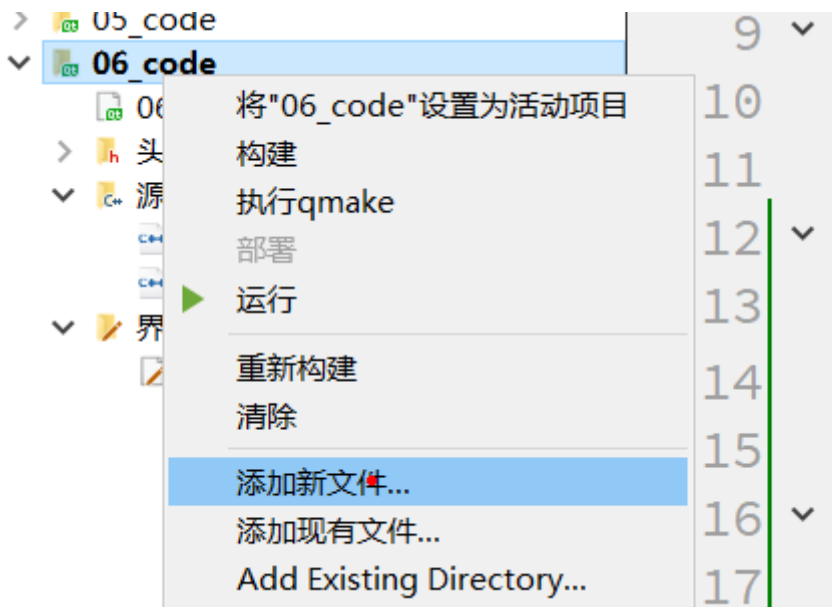
### 3、获取输入框的内容

```
1 QString usr = ui->lineEditUser->text();
2 QString pwd = ui->lineEditPwd->text();
```

## 4、设置输入框的内容

```
1 //将输入框中的用户名及密码清空
2 ui->lineEditUser->setText("");
3 ui->lineEditPwd->setText("");
```

## 5、创建新的页面



×

## Qt 设计器界面类

Form Template  
Class Details  
汇总

### 选择界面模板

templates\forms

Dialog with Buttons Bottom  
Dialog with Buttons Right  
Dialog without Buttons  
Main Window  
**Widget**

> 窗口部件

嵌入式设计  
设备: 无  
屏幕大小: 默认大小

下一步(N)

取消

×

## Qt 设计器界面类

Form Template  
Class Details  
汇总

### 选择类名

类

任意

类名(C): Form

头文件(H): form.h

源文件(S): form.cpp

界面文件(F): form.ui

路径(P): D:\work\bk2103\code\c++\day09\06\_code 浏览...

下一步(N)

取消

## 6、页面的跳转

实例化一个新页面的对象

```
private:
    Ui::Widget *ui;
    Form *newWidget;
};
```

```
1 //创建一个新页面
2 this->newWidget = new Form();
```

//创建一个新页面

```
this->newWidget = new Form();
```

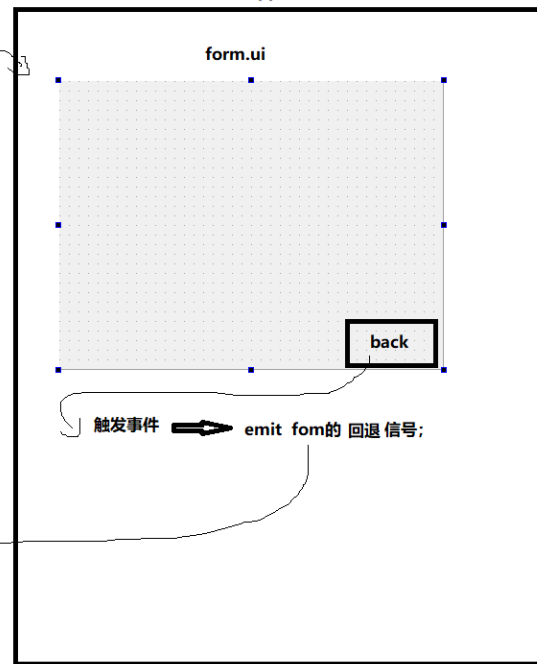
```
connect( ui->loginBtn, &QPushButton::clicked,[&]() {
    //获取用户名以及密码
    QString usr = ui->lineEditUser->text();
    QString pwd = ui->lineEditPwd->text();
    if(usr == "root" && pwd=="123456")
    {
        qDebug()<<"登录成功"<<endl;
        //页面跳转
        //旧页面隐藏
        this->hide();
        //新页面显示
        newWidget->show();
    }
}
```

## 7、从新页面 回到 主页面

widget.cpp构造函数中



form.cpp 构造函数中



## 8、完整代码

widget.cpp的构造函数中

```

1  #include "widget.h"
2  #include "ui_widget.h"
3  #include <QPushButton>
4  #include <QLineEdit>
5  #include <QDebug>
6  #include <QMessageBox>
7  #include "form.h"
8  Widget::Widget(QWidget *parent) :
9      QWidget(parent),
10     ui(new Ui::Widget)
11 {
12     ui->setupUi(this);
13
14     //创建一个新页面
15     this->newWidget = new Form();
16
17     connect( ui->loginBtn, &QPushButton::clicked,[&]() {
18         //获取用户名以及密码
19         QString usr = ui->lineEditUser->text();
20         QString pwd = ui->lineEditPwd->text();
21         if(usr == "root" && pwd=="123456")
22         {
23             qDebug()<<"登录成功"<<endl;

```

```
24 //页面跳转
25 //旧页面隐藏
26 this->hide();
27 //新页面显示
28 newWidget->show();
29 }
30 else
31 {
32 QMessageBox::critical(this, "错误", "用户名或密码错误");
33 //将输入框中的用户名及密码清空
34 ui->lineEditUser->setText("");
35 ui->lineEditPwd->setText("");
36 }
37 } );
38
39 //监听新窗口的 myBack信号
40 connect(newWidget, &Form::myBack, [&]() {
41 //新窗口隐藏
42 newWidget->hide();
43 //主窗口显示
44 this->show();
45 } );
46 }
47
48 Widget::~Widget()
49 {
50 delete ui;
51 }
```

widget.h

```

1  #include <QWidget>
2  | #include "form.h"
3  v namespace Ui {
4  class Widget;
5  }
6
7  v class Widget : public QWidget
8  {
9      Q_OBJECT
10
11  public:
12      explicit Widget(QWidget *parent = 0);
13      ~Widget();
14
15  private:
16      Ui::Widget *ui;
17      | Form *newWidget;
18  };

```

form.h

```

v namespace Ui {
class Form;
}

v class Form : public QWidget
{
    Q_OBJECT

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

    private:
        Ui::Form *ui;
    signals:
        | void myBack(void);
};

```

form.cpp的构造函数中

form.cpp的构造函数中

```

#include "form.h"
#include "ui_form.h"
#include <QPushButton>
Form::Form(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Form)
{
    ui->setupUi(this);
    //给主窗口发送回退信号
#if 0
    connect( ui->backBtn, &QPushButton::clicked, [&]() {
        emit this->myBack();
    } );
#else
    connect( ui->backBtn, &QPushButton::clicked, this, &Form::myBack );
#endif
}

```

## 知识点15 【常用控件】

### 1、QLabel 标签

显示文本、显示图片、显示动画

```
1 setText("hello");
```

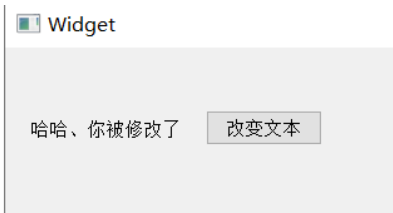
#### 1、显示文本



```

void Widget::on_pushButton_clicked()
{
    ui->label->setText("哈哈、你被修改了");
}

```



#### 2、显示图片



```

void Widget::on_pushButton_2_clicked()
{
    QString imgPath[2]={":/image/down.png", ":/image/up.png"};
    static int x=0;
    QPixmap pix(imgPath[flag]);
    ui->label_2->move( x+=10 ,ui->label_2->pos().y());
    if(x>this->width())
        x=0;
    ui->label_2->setPixmap(pix);

    flag = !flag;
}

```

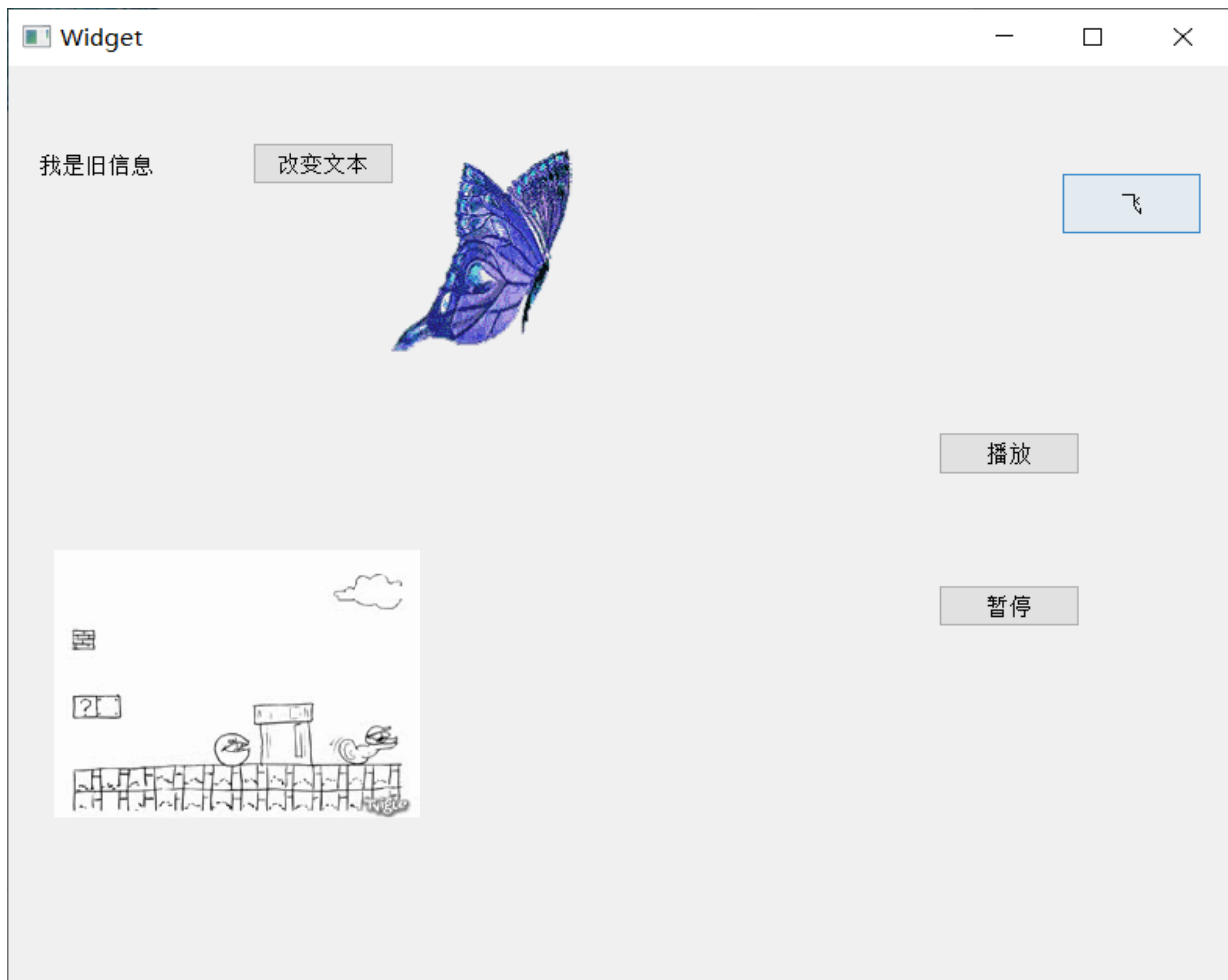


### 3、显示动画

```

1  QMovie *m = new QMovie(":/image/mario.gif");
2  ui->label_3->setMovie(m);
3
4  connect( ui->playBtn, &QPushButton::clicked, [=]() {
5      m->start(); // 播放动画
6  } );
7
8  connect( ui->pause, &QPushButton::clicked, [=]() {
9      m->stop(); // 暂停动画
10 } );

```



## 2、单选框QRadioButton



```
1 //让单选框动起来
2 connect(ui->radioButton, &QRadioButton::clicked, [=]() {
3     qDebug() << ui->radioButton->text() << endl;
4 } );
```

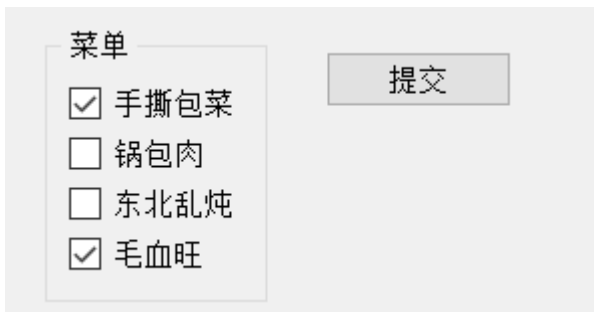
## 3、复选框QCheckBox

```
1 static QStringList strList;
2
3 connect( ui->checkBox, &QCheckBox::stateChanged, [=](int state) {
4     //state 2选中 0未选中
5     if(state == 2)
6         strList.append(ui->checkBox->text()); ;
```

```

7 } );
8 connect( ui->checkBox_2, &QCheckBox::stateChanged, [=](int state){
9     //state 2选中 0w未选中
10     if(state == 2)
11         strList.append(ui->checkBox_2->text());
12 } );
13 connect( ui->checkBox_3, &QCheckBox::stateChanged, [=](int state){
14     //state 2选中 0w未选中
15     if(state == 2)
16         strList.append(ui->checkBox_3->text());
17 } );
18 connect( ui->checkBox_4, &QCheckBox::stateChanged, [=](int state){
19     //state 2选中 0w未选中
20     if(state == 2)
21         strList.append(ui->checkBox_4->text());
22 } );
23
24 connect( ui->pushButton, &QPushButton::clicked, [=]() {
25     qDebug() << "选中的菜单: " << strList << endl;
26 } );

```



## 4、下拉列表框QComboBox

```

1 void (QComboBox::*p)(const QString &) =&QComboBox::currentIndexChanged;
2 connect( ui->comboBox, p, [=](const QString &text){
3     qDebug() << "选中的车型为:" << text << endl;
4 } );

```



## 5、列表控件QListWidget

```

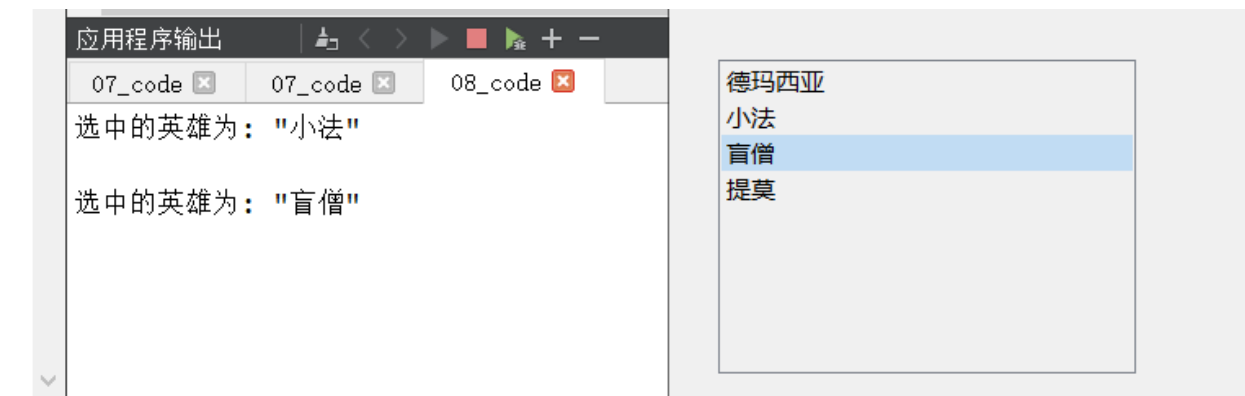
1 //添加一项数据

```

```

2 ui->listWidget->addItem("德玛西亚");
3 //添加 多项数据
4 QStringList list;
5 list<<"小法"<<"盲僧"<<"提莫";
6 ui->listWidget->addItems(list);
7
8 connect( ui->listWidget, &QListWidget::itemDoubleClicked,[=](QListWidgetItem *item){
9     qDebug()<<"选中的英雄为:"<<item->text()<<endl;
10 } );
11 //设置背景透明
12 ui->listWidget->setStyleSheet("background-color:transparent");

```

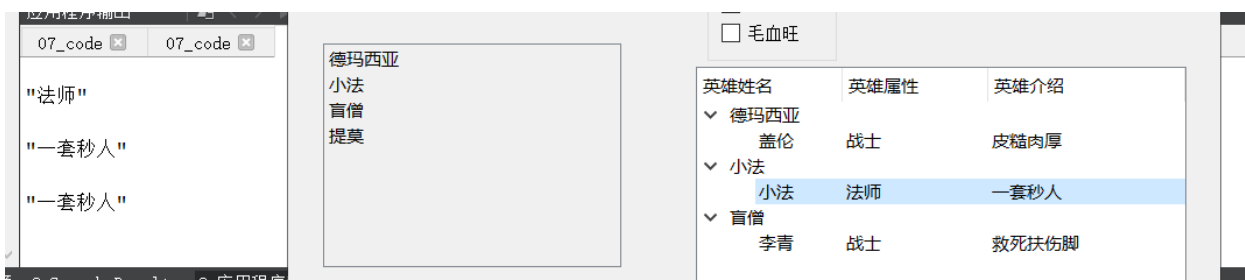


## 6、树控件QTreeWidget

```

1 connect( ui->treeWidget, &QTreeWidget::itemClicked,[=](QTreeWidgetItem *item, int column){
2     qDebug()<<item->text(column)<<endl;
3 } );

```



## 7、表格控件QTableWidget

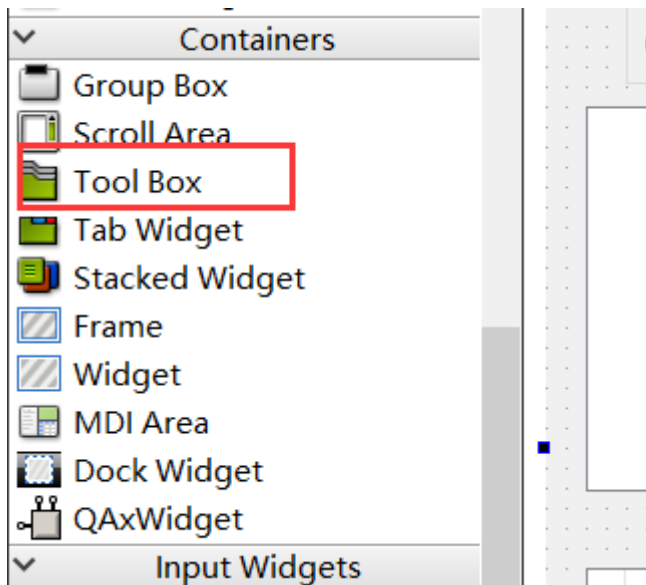
```

1 connect( ui->tableWidget, &QTableWidget::cellClicked, [=](int row, int col){
2     qDebug()<< ui->tableWidget->item(row,col)->text()<<endl;
3 } );
4

```

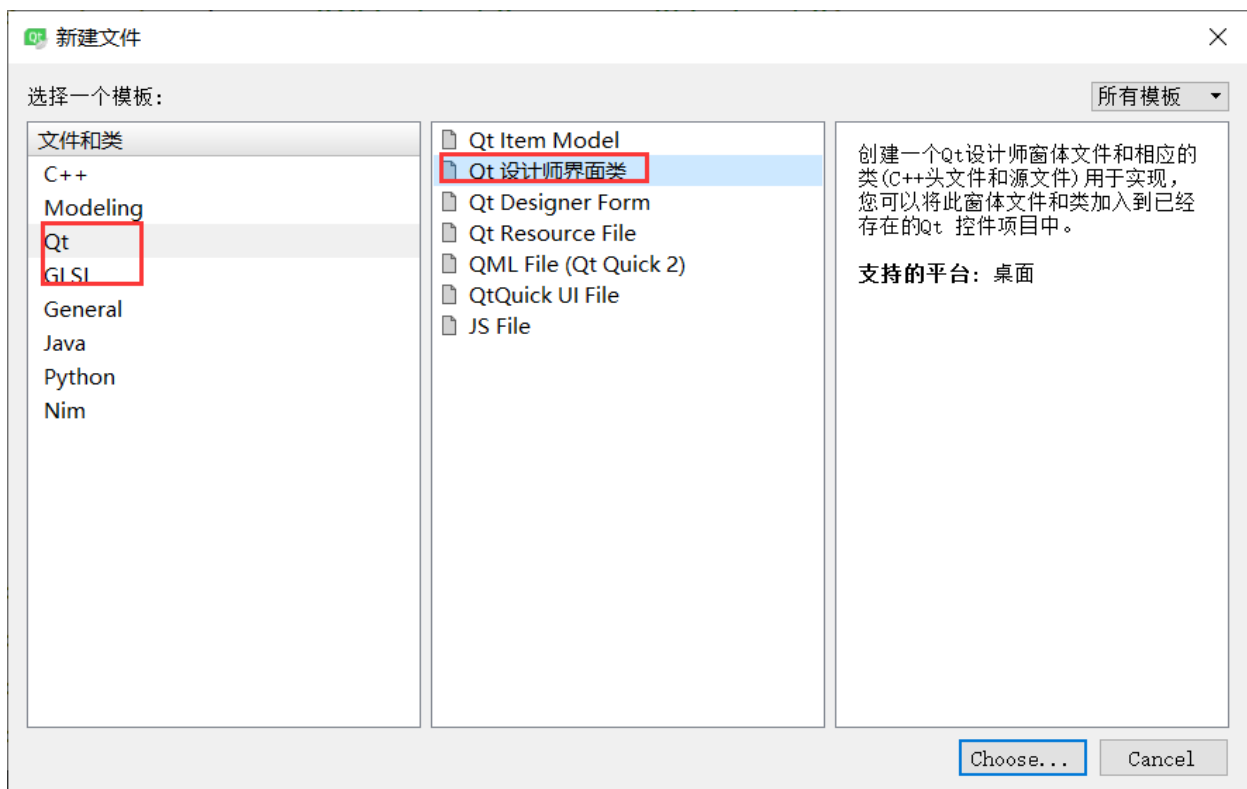
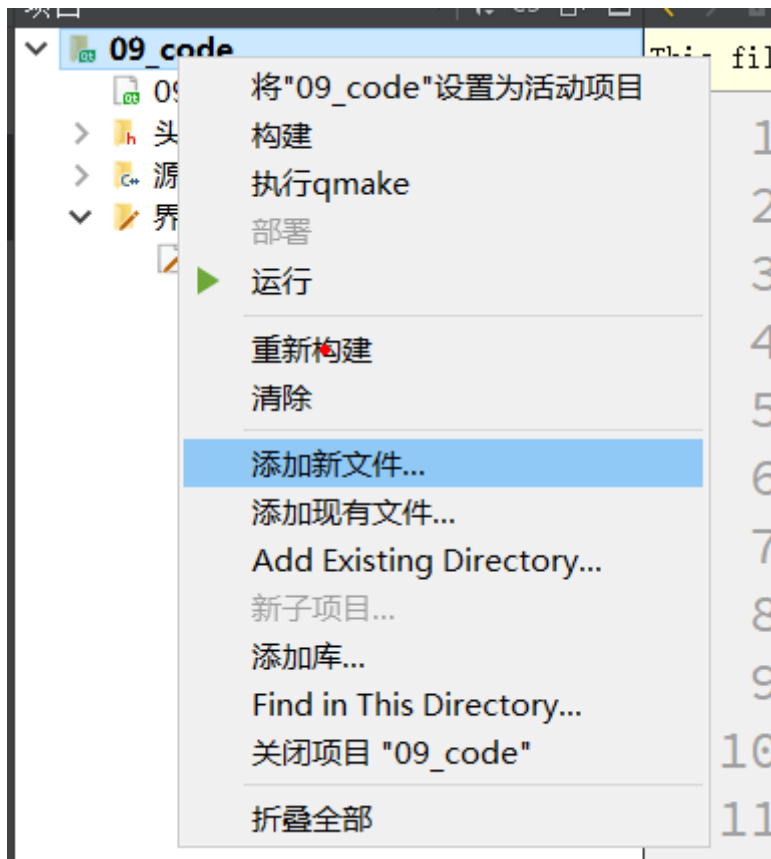
	姓名	性别	年龄	地址
1	刘备	男	18	蜀
2	诸葛亮	男	18	蜀
3	关羽	男	18	蜀
4	z赵云	男	18	蜀

## 8、工具箱QToolBox



## 9、自定义控件

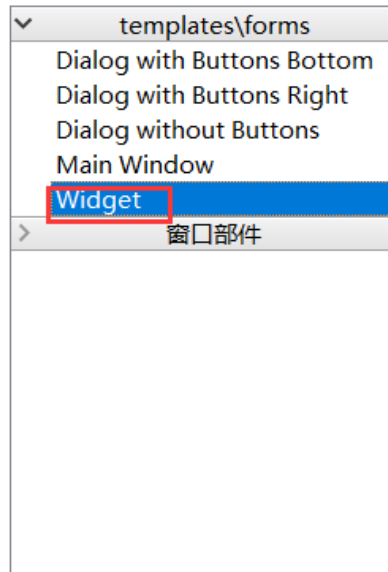
有些控件 需要反复使用 可以事先 将这些控件制作成 自定义控件，在使用处 直接提升。



## Qt 设计器界面类

### 选择界面模板

Form Template  
Class Details  
汇总



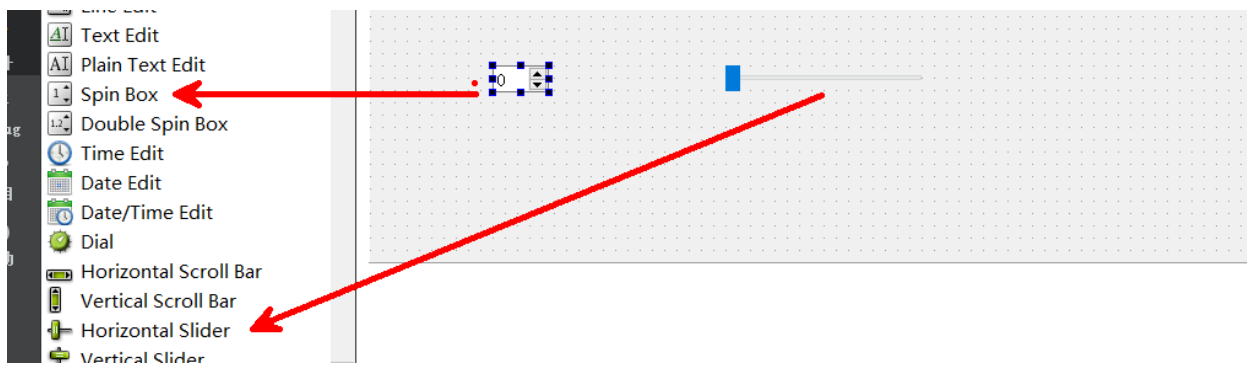
嵌入式设计

设备： 无

屏幕大小： 默认大小

下一步(N) >

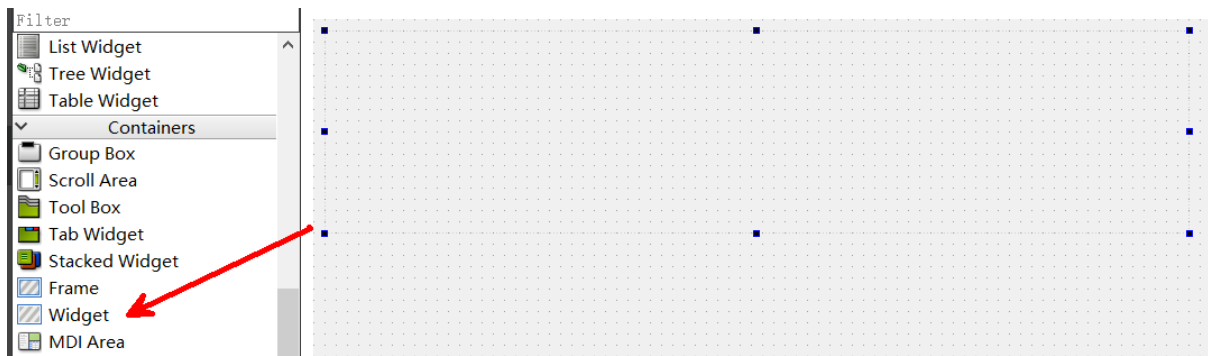
取消



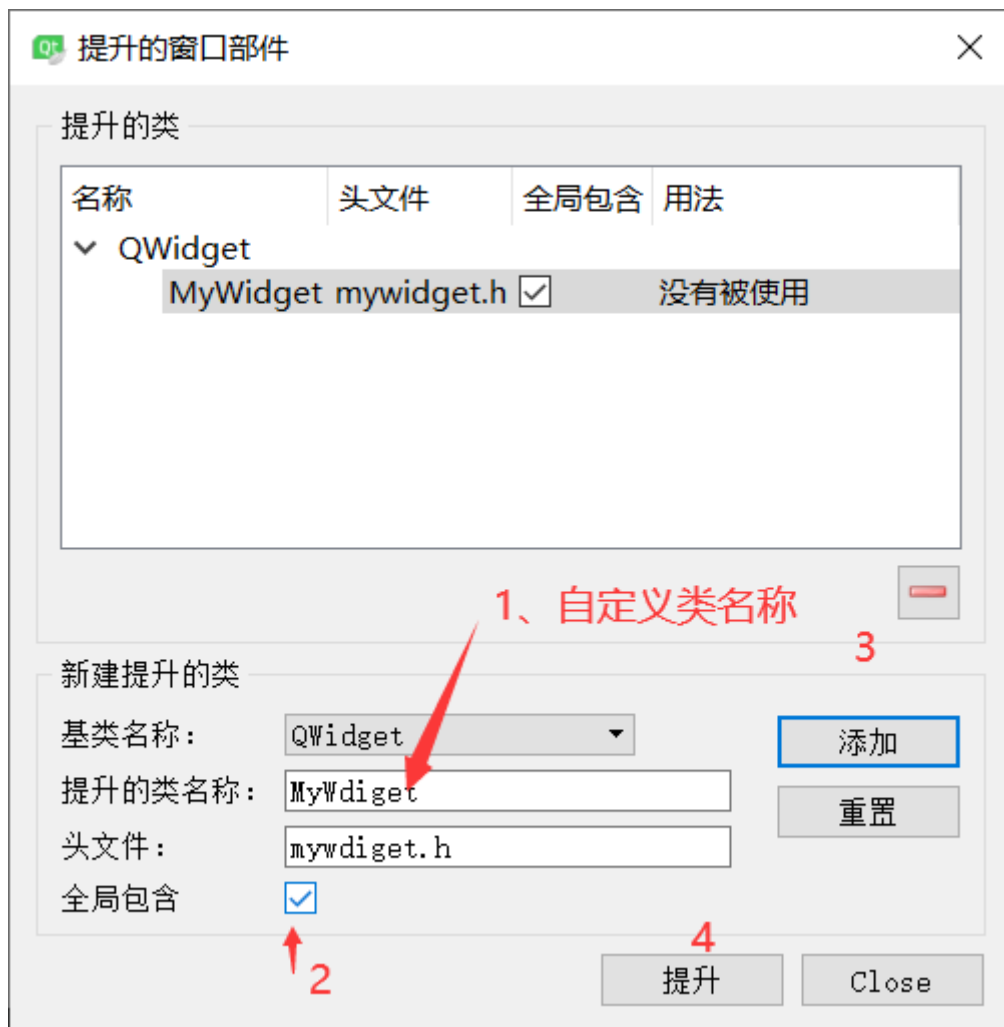
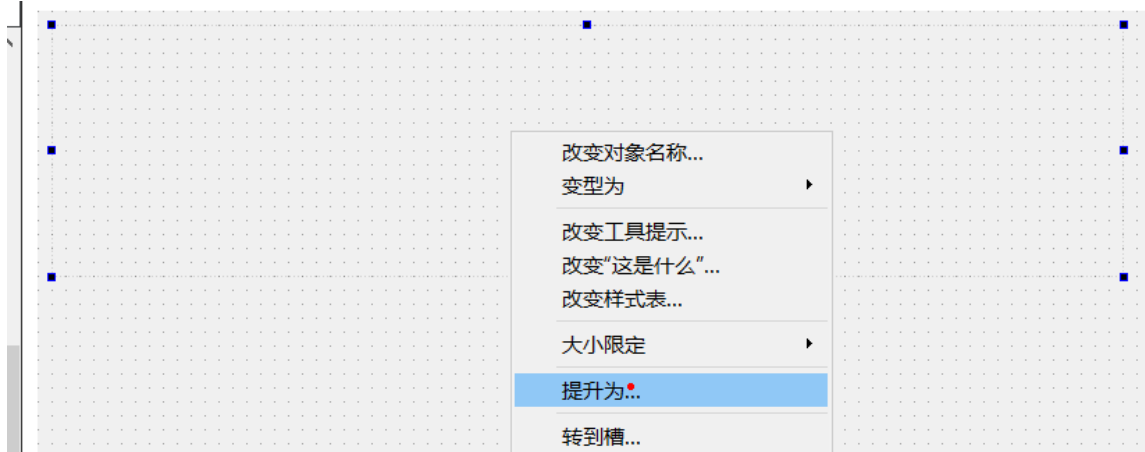
mywidget.ui



widget.ui 先放置一个widget容器



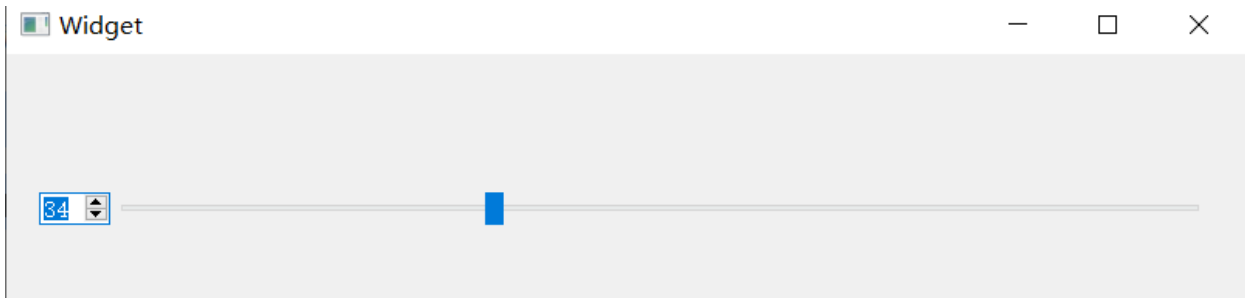
提升容器 自定义的控制件





## 在MyWidget构造函数中 让自定义控件 动起来

```
1 //改变spinBox数据 让进度条QSlider发生变化
2 void (QSpinBox:: *p1)(int) = &QSpinBox::valueChanged;
3 connect(ui->spinBox, p1, ui->horizontalSlider, &QSlider::setValue);
4
5 //拖动进度条QSlider 改变spinBox数据
6 connect(ui->horizontalSlider, &QSlider::valueChanged, ui->spinBox, &QSpinBox::setValue);
```



## 给自定义控件提供接口

```
class MyWidget : public QWidget
{
    Q_OBJECT

public:
    explicit MyWidget(QWidget *parent = 0);
    ~MyWidget();
    void mySetValue(int value);

private:
    Ui::MyWidget *ui;
};
```

```
23
24 void MyWidget::mySetValue(int value)
25 {
26     ui->horizontalSlider->setValue(50);
27 }
28
```

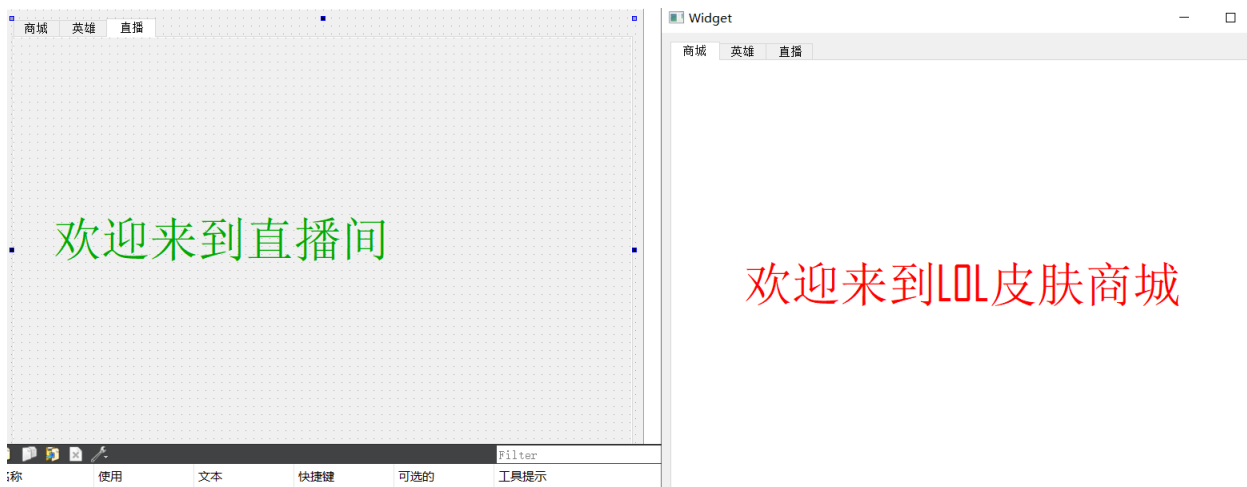
```

// 构造函数
Widget::Widget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Widget)
{
    ui->setupUi(this);

    connect( ui->pushButton, &QPushButton::clicked, [=]() {
        ui->widget->mySetValue(50);
    } );
}

```

## 10、tab widget容器



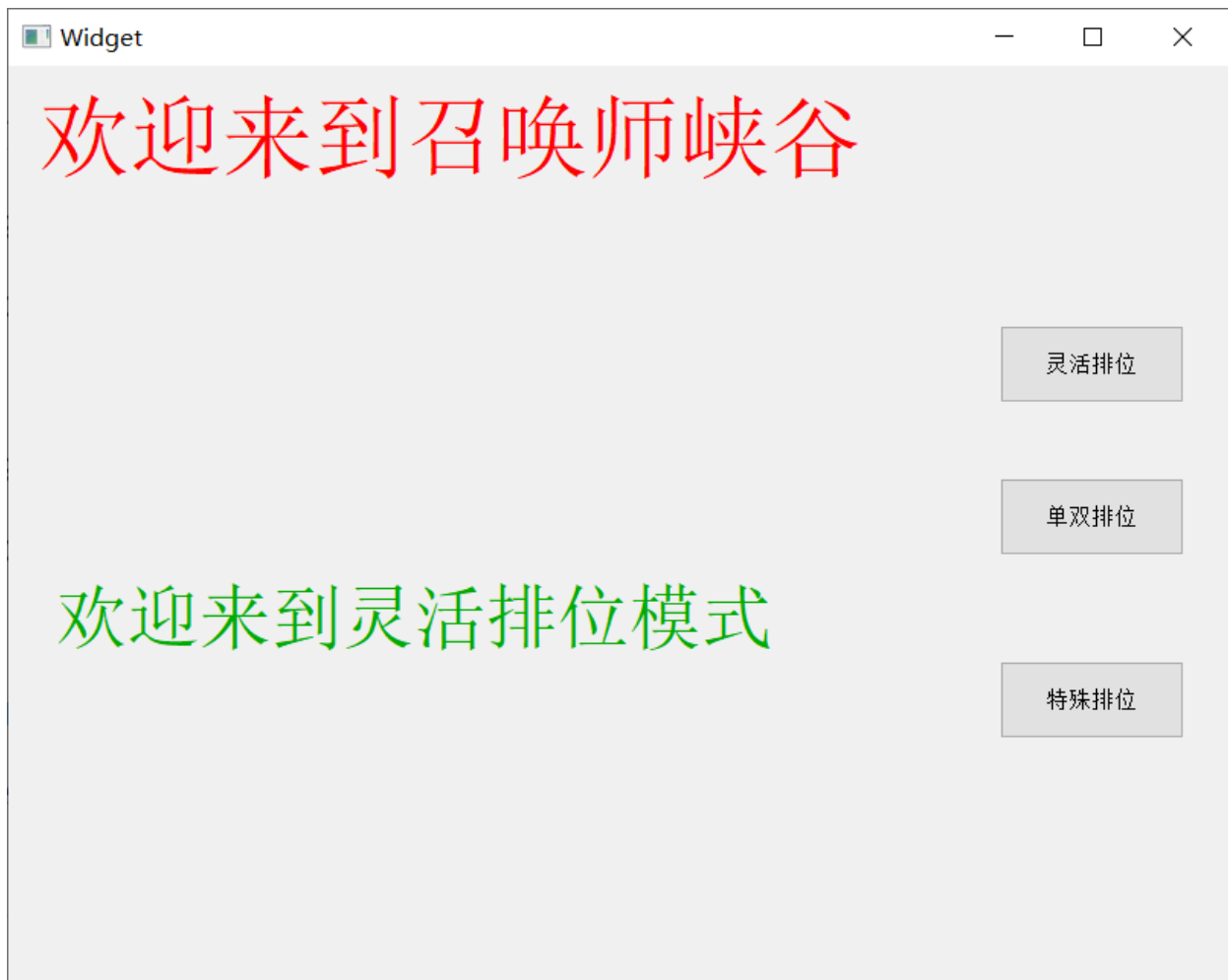
## 11、stackedWidget栈容器（局部更新页面）

widget.cpp构造函数中

```

1 //设置栈容器的默认页码
2 ui->stackedWidget->setCurrentIndex(0);
3
4 //动起来 切换页码
5 connect(ui->pushButton, &QPushButton::clicked, [=]() {
6     ui->stackedWidget->setCurrentIndex(0);
7 } );
8 connect(ui->pushButton_2, &QPushButton::clicked, [=]() {
9     ui->stackedWidget->setCurrentIndex(1);
10 } );
11 connect(ui->pushButton_3, &QPushButton::clicked, [=]() {
12     ui->stackedWidget->setCurrentIndex(2);
13 } );

```



## 知识点16 【Qt的事件】

### 1、Qt事件的概述

Qt 中所有事件类都继承于 `QEvent`

`event()`函数并不 直接处理事件，而是按照事件对象的类型分派给特定的事件处理函数。

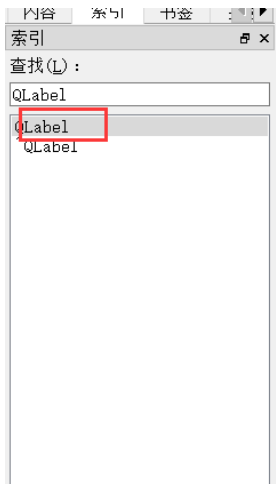
在所有组件的父类 `QWidget` 中，定义了很多事件处理的回调函数，如

- `keyPressEvent()`
- `keyReleaseEvent()` 虚函数
- `mouseDoubleClickEvent()`
- `mouseMoveEvent()`
- `mousePressEvent()`
- `mouseReleaseEvent()` 等。

这些函数都是 `protected virtual` 的，也就是说，我们可以在子类中重新实现

这些函数。下面来看一个例子：

### 2、重写QLabel的鼠标事件



## Contents

[Properties](#)  
[Public Functions](#)  
[Reimplemented Public Functions](#)  
[Public Slots](#)  
[Signals](#)  
[Reimplemented Protected Functions](#)  
[Detailed Description](#)

virtual void	<a href="#">changeEvent</a> (QEvent * <i>ev</i> )
virtual void	<a href="#">contextMenuEvent</a> (QContextMenuEvent * <i>ev</i> )
virtual bool	<a href="#">event</a> (QEvent * <i>e</i> )
virtual void	<a href="#">focusInEvent</a> (QFocusEvent * <i>ev</i> )
virtual bool	<a href="#">focusNextPrevChild</a> (bool <i>next</i> )
virtual void	<a href="#">focusOutEvent</a> (QFocusEvent * <i>ev</i> )
virtual void	<a href="#">keyPressEvent</a> (QKeyEvent * <i>ev</i> )
virtual void	<a href="#">mouseMoveEvent</a> (QMouseEvent * <i>ev</i> )
virtual void	<a href="#">mousePressEvent</a> (QMouseEvent * <i>ev</i> )
virtual void	<a href="#">mouseReleaseEvent</a> (QMouseEvent * <i>ev</i> )
virtual void	<a href="#">paintEvent</a> (QPaintEvent *)

mylabel.h

```

class MyLabel : public QLabel
{
    Q_OBJECT
public:
    explicit MyLabel(QWidget *parent = 0);

    //重写鼠标按下事件
    virtual void mousePressEvent(QMouseEvent *ev);
    //重写鼠标移动事件
    virtual void mouseMoveEvent(QMouseEvent *ev);

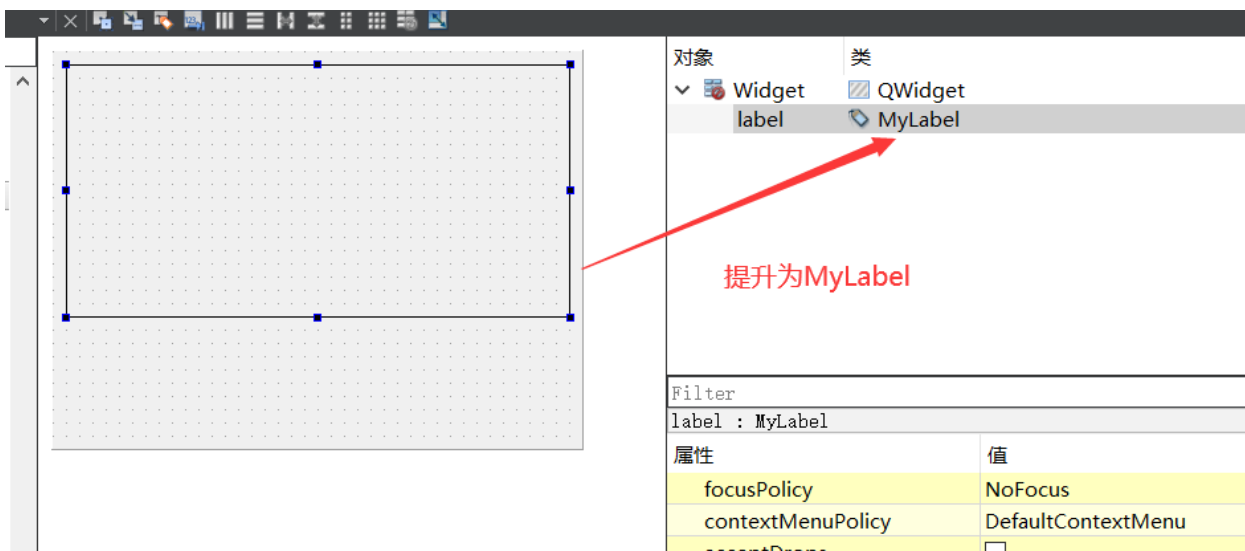
```

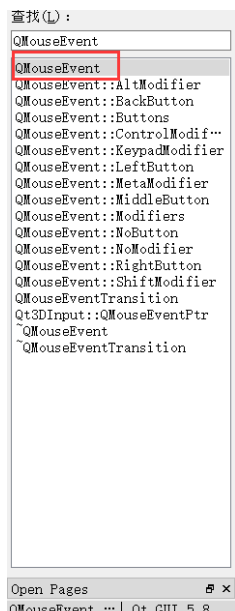
mylabel.cpp

```

8
9 void MyLabel::mousePressEvent(QMouseEvent *ev)
10 {
11     qDebug() << "鼠标按下了" << endl;
12 }
13
14 void MyLabel::mouseMoveEvent(QMouseEvent *ev)
15 {
16     qDebug() << "鼠标移动了" << endl;
17 }
18

```





```
Qt::MouseButton button, Qt::KeyboardModifiers modifier
QMouseEvent(Type type, const QPointF &localPos, const Q
Qt::MouseButton button, Qt::MouseButton buttons, Qt::Ke
QMouseEvent(Type type, const QPointF &localPos, const Q
Qt::MouseButton button, Qt::MouseButton buttons, Qt::Ke
Qt::MouseEventSource source)

Qt::MouseButton button() const
Qt::MouseButton buttons() const
Qt::MouseEventFlags flags() const
QPoint globalPos() const
int globalX() const
int globalY() const
const QPointF & localPos() const
QPoint pos() const
const QPointF & screenPos() const
Qt::MouseEventSource source() const
const QPointF & windowPos() const
int x() const
int y() const
```

按个按键按下

全局位置 从主窗口(0,0)开始计算

局部位置, 从label控制的左上角 (0,0) 开始计算

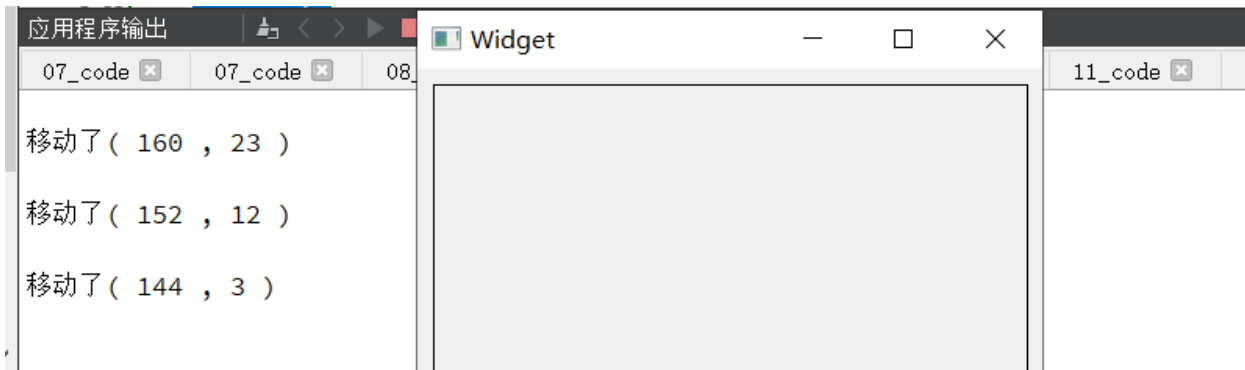
2 public functions inherited from QInputEvent

```
1 #include "mylabel.h"
2 #include <QDebug>
3 #include <QMouseEvent>
4 MyLabel::MyLabel(QWidget *parent) : QLabel(parent)
5 {
6     //设置鼠标的追踪模式
7     this->setMouseTracking(true);
8 }
9
10 //形参QMouseEvent *ev中ev保存了鼠标的信息（左键、右键、滚轮、位置）
11 void MyLabel::mousePressEvent(QMouseEvent *ev)
12 {
13     if(ev->button() == Qt::LeftButton)
14     {
15         qDebug()<<"左键按下了("<<ev->x()<<","<<ev->y()<<")"<<endl;
16     }
17     else if(ev->button() == Qt::RightButton)
18     {
19         qDebug()<<"右键按下了("<<ev->x()<<","<<ev->y()<<")"<<endl;
20     }
21 }
22
23 void MyLabel::mouseMoveEvent(QMouseEvent *ev)
```

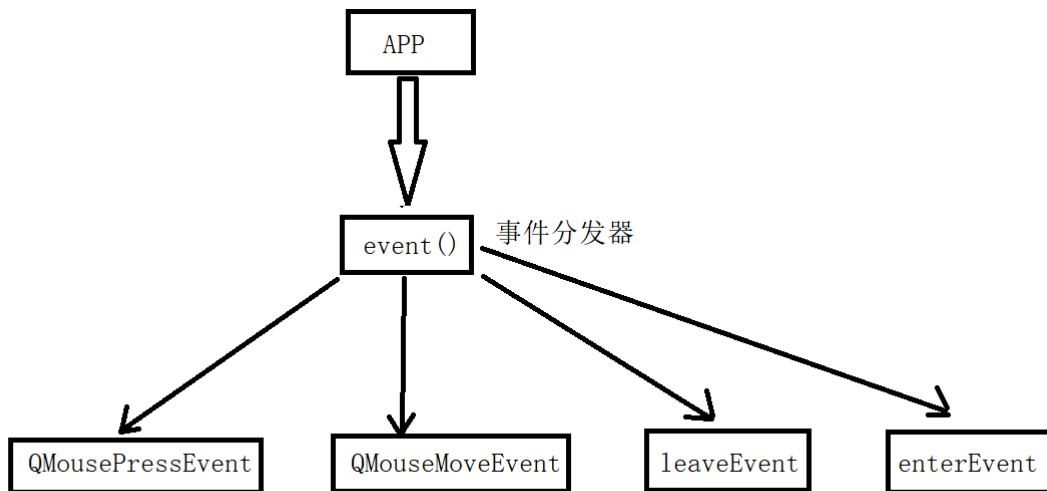
```

24 {
25     qDebug() << "移动了(" << ev->x() << ", " << ev->y() << ")" << endl;
26 }

```



### 3、Qevent事件分发器



event()函数主要用于事件的分发。所以，如果你希望在事件分发之前做一些操作，就可以重写这个 event()函数。

如果传入的事件已被识别并且处理，则需要返回 true，否则返回false.

如果返回值是 true，那么 Qt 会认为这个事件已经处理完毕，不会再将这个事件发送给其它对象，而是会继续处理事件队列中的下一事件.

virtual bool	<u>event</u> (QEvent *event)
--------------	------------------------------

```

class MyLabel : public QLabel
{
    Q_OBJECT
public:
    explicit MyLabel(QWidget *parent = 0);

    //重写鼠标按下事件
    virtual void mousePressEvent(QMouseEvent *ev);
    //重写鼠标移动事件
    virtual void mouseMoveEvent(QMouseEvent *ev);

    //重写事件分发器
    virtual bool event(QEvent *event);

signals:

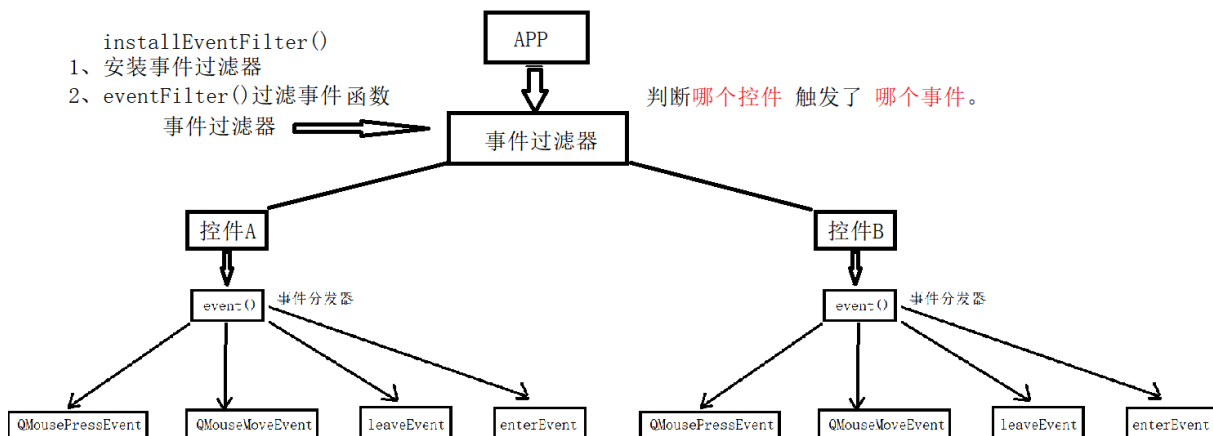
```

```

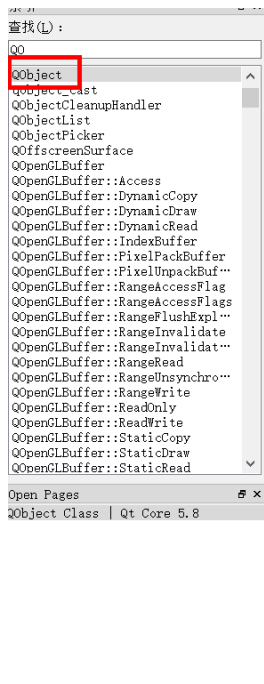
28 bool MyLabel::event(QEvent *event)
29 {
30     //处理关心的事件（鼠标按下事件）
31     if( event->type() == QEvent::MouseButtonPress )
32     {
33         QMouseEvent *ev = (QMouseEvent *)event;
34         qDebug() << "事件分发器中 鼠标按下了 (" << ev->x() << ", " << ev->y() << ")" << endl;
35         return true; //表示处理完毕 不继续分发
36     }
37
38     //不关心的事件 统一由父类的event分发器处理
39     return QLabel::event(event);
40 }

```

## 4、事件过滤器







```
virtual    QObject()
           bool    blockSignals(bool block)
           const QObjectList & children() const
           QMetaObject::Connection connect(const QObject *sender, const char *signal, const char *method, Qt::Con
           bool    disconnect(const char *signal = Q_NULLPTR, const QObject *receiver = Q_NULLPTR
           bool    disconnect(const QObject *receiver, const char *method = Q_NULLPTR) const
           void    dumpObjectInfo()
           void    dumpObjectTree()
           QList<QByteArray> dynamicPropertyNames() const
           virtual bool event(QEvent *e)
           virtual bool eventFilter(QObject *watched, QEvent *event)
               T findChild(const QString &name = QString(), Qt::FindChildOptions options = Qt::
           QList<T> findChildren(const QString &name = QString(), Qt::FindChildOptions options = Q
           QList<T> findChildren(const QRegExp &regExp, Qt::FindChildOptions options = Qt::FindChi
           QList<T> findChildren(const QRegularExpression &re, Qt::FindChildOptions options = Qt::
           bool    inherits(const char *className) const
           void    installEventFilter(QObject *filterObj)
           bool    isWidgetType() const
           bool    isWindowType() const
           void    killTimer(int id)
           virtual const QMetaObject * metaObject() const
```

事件过滤函数

安装事件过滤器

virtual bool

eventFilter(QObject \*watched, QEvent \*event)

类的构造函数中 安装事件过滤器

```
1 //安装事件过滤器
2 this->installEventFilter(this);
```

重写事件过滤器

```
1 头文件声明
2 //重写事件分发器
3 virtual bool eventFilter(QObject *watched, QEvent *event);
```

```
1 .cpp 重写事件分发器
2 bool MyLabel::eventFilter(QObject *watched, QEvent *event)
3 {
4     //对当前控件this 感兴趣
5     if(watched == this)
6     {
7         //感兴趣的是鼠标按下事件
8         if( event->type() == QEvent::MouseButtonPress )
9         {
10             QMouseEvent *ev = (QMouseEvent *)event;
11             qDebug()<<"事件过滤器中 鼠标按下了("<<ev->x()<<","<<ev->y()<<")"<<endl;
12             return true;//表示处理完毕 不继续分发
13         }
14     }
15 }
```

```

16 //其他过滤事件 交给父类
17 return QLabel::eventFilter(watched, event);
18 }

```

## 知识点17 【定时器】

### 1、定时器事件 触发定时

virtual void	<code>timerEvent(QTimerEvent *event)</code>
-----------------	---

```

class Widget : public QWidget
{
    Q_OBJECT

public:
    explicit Widget(QWidget *parent = 0);
    ~Widget();
    //重写定时器 事件
    virtual void timerEvent(QTimerEvent *event);

private:
    Ui::Widget *ui;
    //定义两个定时器 标示
    int t1;
    int t2;
};

```

```

#include <QDebug>
Widget::Widget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Widget)
{
    ui->setupUi(this);

    //启动定时器事件 会返回定时器的标示
    t1 = this->startTimer(1000); //1000ms
    t2 = this->startTimer(2000); //1000ms
}

```

```

void Widget::timerEvent(QTimerEvent *event)
{
    if(event->timerId() == t1)
    {
        static int time1 = 0;
        ui->label->setText(QString::number(time1));
        time1++;
    }
    else if(event->timerId() == t2)
    {
        static int time2 = 0;
        ui->label_2->setText(QString::number(time2));
        time2++;
    }
}

```



## 2、定时器对象 触发定时（方便控制定时器）

```
1 #include <QTimer>
```

widget.cpp的构造函数中

```

1 //创建一个定时器对象
2 QTimer *timer = new QTimer(this);
3 //定时器对象 监听 超时信号的 到来
4 connect( timer, &QTimer::timeout, [=]() {
5     static int num = 0;
6     ui->label_3->setText(QString::number(num));
7     num++;
8 } );

```

```

9 //启动定时
10 connect( ui->pushButton, &QPushButton::clicked,[=]() {
11     timer->start(1000); //1000ms
12 } );
13 //停止计时
14 connect( ui->pushButton_2, &QPushButton::clicked,[=]() {
15     timer->stop(); //停止计时
16 } );

```



### 3、定时器静态函数 触发定时 用于延时

#### Static Public Members

```

void singleShot(int msec, const QObject *receiver, const char *member)
void singleShot(int msec, Qt::TimerType timerType, const QObject *receiver, const char *member)
void singleShot(int msec, const QObject *receiver, PointerToMemberFunction method)
void singleShot(int msec, Qt::TimerType timerType, const QObject *receiver, PointerToMemberFunction method)
void singleShot(int msec, Functor functor)
void singleShot(int msec, Qt::TimerType timerType, Functor functor)
void singleShot(int msec, const QObject *context, Functor functor)
void singleShot(int msec, Qt::TimerType timerType, const QObject *context, Functor functor)
void singleShot(std::chrono::milliseconds msec, const QObject *receiver, const char *member)
void singleShot(std::chrono::milliseconds msec, Qt::TimerType timerType, const QObject *receiver, const char *member)
• 11 static public members inherited from QObject

```

```

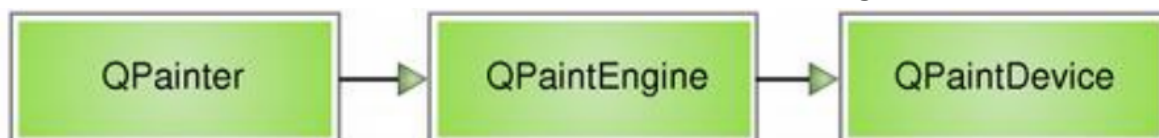
1 //QTimer的静态函数 只会触发一次 加入延时5秒
2 QTimer::singleShot(5000, [=]() {
3     ui->label_4->setText("哈哈，梁哥来了!!!!");
4 });

```



## 知识点18 【Qt绘制背景图片】

整个绘图系统基于 QPainter, QPainterDevice 和 QPaintEngine 三个类完成。



如果想在窗口上 绘制图片需要重载widget的绘图事件

```
1 void QWidget::paintEvent(QPaintEvent *event)
```

窗口加载、update都会调用 paintEvent

### 1、绘制背景图片

```
1 //窗口加载、update都会执行绘图事件
2 void Widget::paintEvent(QPaintEvent *event)
3 {
4     //定义一个画家 在主窗口this上 作画
5     QPainter *painter = new QPainter(this);
6
7     //开始画
8     QPixmap pix;
9     //加载图片
10    pix.load(":/image/Frame.jpg");
11    //修改图片大小
12    pix.scaled(this->width(), this->height());
13    painter->drawPixmap(0,0,this->width(),this->height(), pix);
14
15    //绘制局部图片
```

```

16  pix.load(":/image/butterfly.png");
17  //修改图片大小
18  pix.scaled(50, 50);
19  painter->drawPixmap(50,50,50,50, pix);
20  }

```

## 2、切换背景图片

构造函数：

```

1  //单击按钮切换背景 图片 update调用 就会执行 绘图事件
2  connect( ui->pushButton, &QPushButton::clicked, [=]() {
3      this->update();
4  } );

```

绘图事件：

```

1  //窗口加载、update都会执行绘图事件
2  void Widget::paintEvent(QPaintEvent *event)
3  {
4      QString img[4]={"./image/Frame.jpg", "./image/butterfly.png", "./image/LuffyQ.png", "./image/Sunny.jpg"};
5      int n = sizeof(img)/sizeof(img[0]);
6      static int index = 0;
7
8      //定义一个画家 在主窗口this上 作画
9      QPainter *painter = new QPainter(this);
10
11     //开始画
12     QPixmap pix;
13     //加载图片
14     pix.load(img[index]);
15     //修改图片大小
16     pix.scaled(this->width(), this->height());
17     painter->drawPixmap(0,0,this->width(),this->height(), pix);
18
19     index++;
20     if(index == 4)
21         index=0;
22 }

```

Widget



切换