



北京大学  
PEKING UNIVERSITY

# Qt 编程

# 示例8 常用组件介绍

演示QToolBox, QListWidget, QStackedWidget

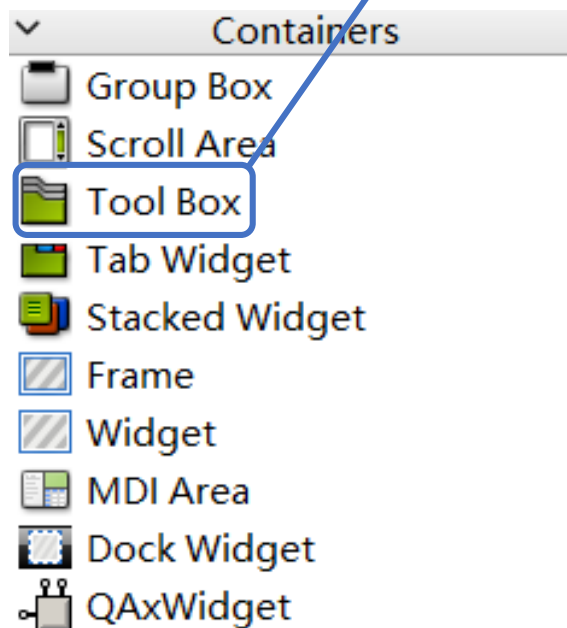
# 示例8 常用组件介绍

- QToolBox简介

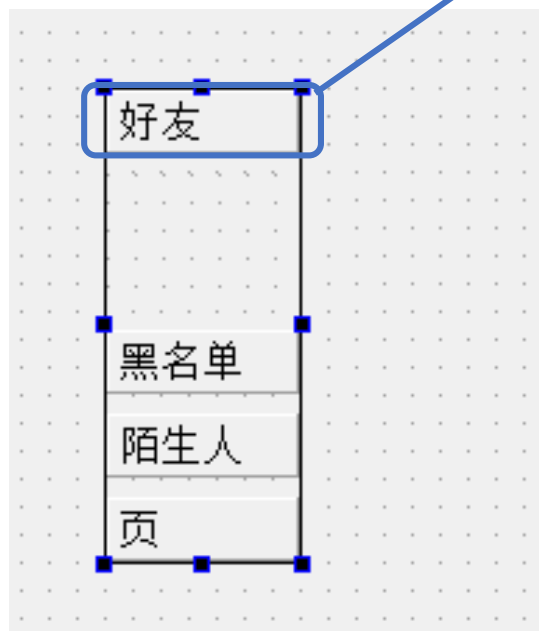
1. QToolBox类提供了一个列（选项卡式的）部件条目
2. QToolBox可以在一个tab列上显示另外一个，  
并且当前的item显示在当前的tab下面  
每个tab都在tab列中有一个索引位置  
tab的item是一个QWidget

## 示例8 常用组件介绍

在使用QToolBox时可以在设计中选择该组件，生成一个QToolBox类的实例对象



这里就是tab的item，是一个QWidget对象。



QToolBox对象

QToolBox对象也可以在MyWidget中声明，具体细节可参考：  
<https://blog.csdn.net/liang19890820/article/details/52439711>

# 示例8 常用组件介绍

注意QLabel是QWidget的子类

```
QLabel * pLabel = new QLabel();  
pLabel->setText ("good");  
ui->toolBox->addItem(pLabel, "hello");
```

设置组件pLabel显示的文本是"good"

在设计QToolBox时，也可以在主界面MyWidget的设计中声明一个新的QWidget的对象作为item，并用addItem函数插入QToolBox对象中

设置此item下面显示文本"hello"，除了文本之外也可在下面显示QIcon等对象

## 示例8 常用组件介绍

可以看到比起之前多了一个显示"hello"的item



这里的"good"不是item，而是我们设置的"hello"这个item条目的text内容

# 示例8 常用组件介绍

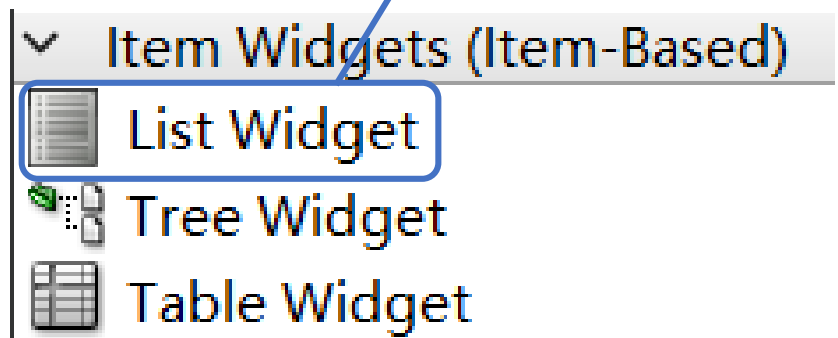
- QListWidget简介

1. QListWidget类列表框控件用来加载并显示多个列表项

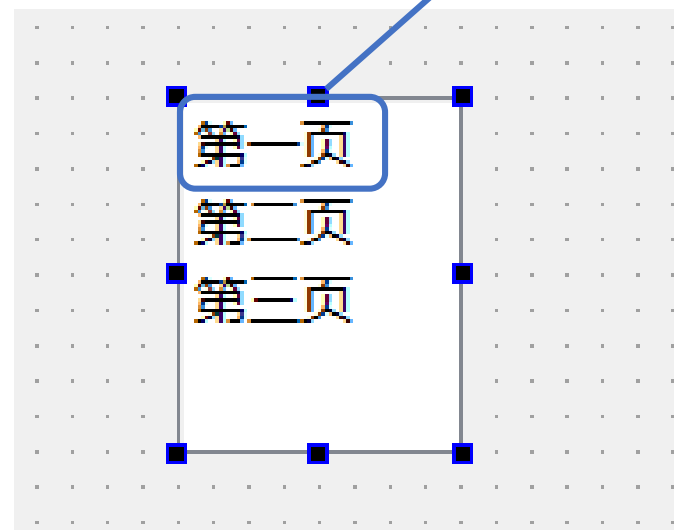
2. QListWidget对象中可以添加QListWidgetItem类型作为列表项，QListWidgetItem类的设置与之前的QToolBox类似

## 示例8 常用组件介绍

在使用QListWidget时可以在  
**设计**中选择该组件，生成一个  
QListWidget类的实例对象



这里是QListWidgetItem  
类的对象



QListWidgetItem对象

若想学习QListWidget和后面要学的QStackedWidget对象更多细节可参考：  
<https://blog.csdn.net/l09711/article/details/7315979>



# 示例8 常用组件介绍

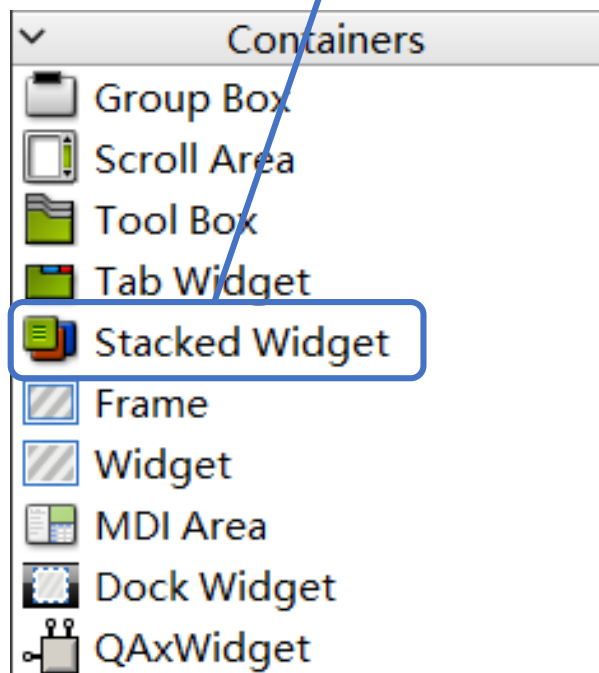
- QStackedWidget简介

1.QStackedWidget可实现**同一界面**切换不同的窗口，直接切换就可以显示不同子窗口的内容

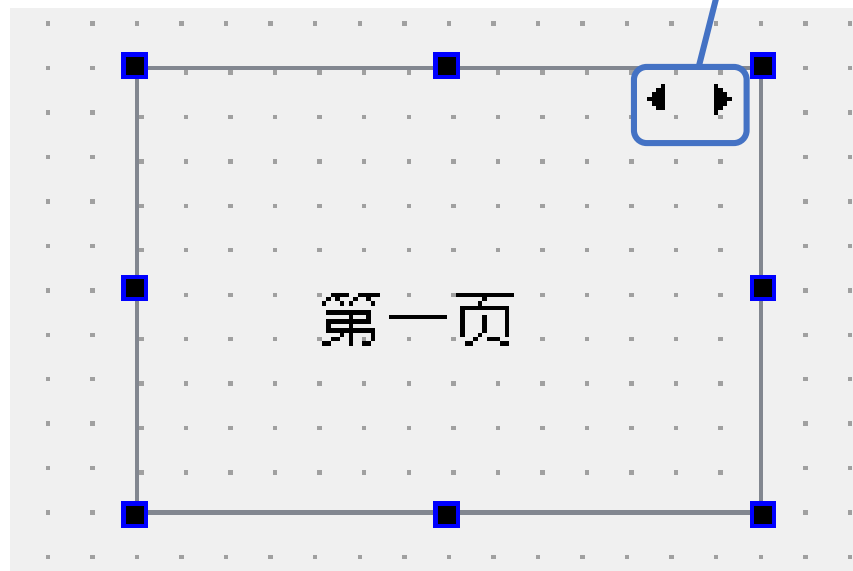
2.QStackedWidget类往往可以和QListWidget类共同完成界面的切换

## 示例8 常用组件介绍

和之前类似，可以直接在QtCreator的**设计**中插入QStackedWidget对象



目前只展示了QStackedWidget对象的一个部件，可以点击这里来切换显示的部件



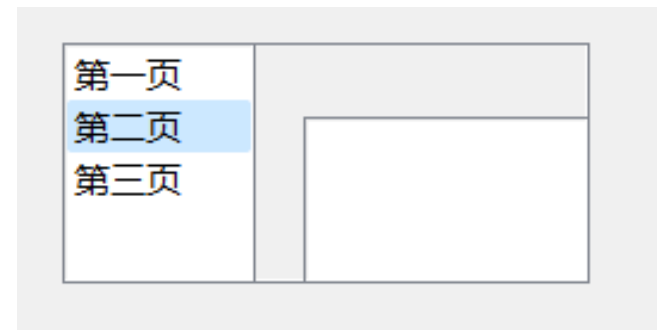
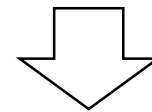
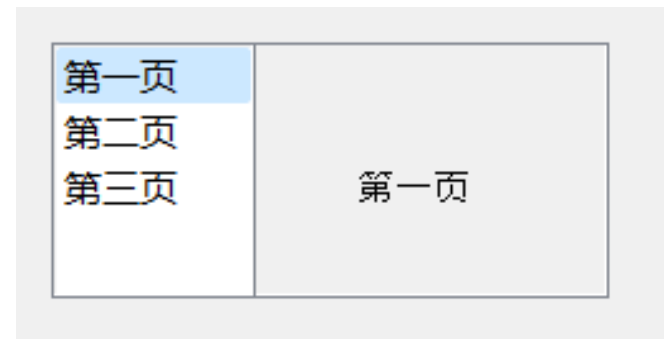
QStackedWidget对象

## 示例8 常用组件介绍

QStackedWidget对象切换部件比较常用的一种方法是和QListWidget对象的列表项进行**关联**，通过QListWidget对象列表项的选择切换组件

```
QListWidget * list = new QListWidget(this);
QStackedWidget * stack = new QStackedWidget (this);
connect(list,SIGNAL(currentRowChanged(int))
        ,stack,SLOT(setCurrentIndex(int)));
```

这里利用了之前学的**connect**函数，在QListWidget对象**切换**列表项的时候，发出信号并被QStackedWidget对象**捕获**，实行对应的**槽函数**（这里为显示对应的组件/页面）



# 示例9 常用按钮介绍

演示不同的按钮，按钮数组，按钮带menu

# 示例9 常用按钮介绍

- QPushButton

QPushButton是最基本的按钮，一般在点击（click）时发出信号触发槽函数，建立

槽映射的方法可参照示例5~7

The image shows a Qt IDE environment with a C++ source file and a corresponding Qt Designer window.

**Source Code (C++):**

```
69 void MyWidget::on_pushButton_toggled(bool checked)
70 {
71     qDebug() << tr("按钮是否按下: ") << checked;
72 }
73
74 void MyWidget::on_btSummary_clicked()
75 {
76     QString s = "选中的checkbox:";
77     for(int i = 0; i < 3; ++i)
78         if( ckSports[i]->isChecked())
79             s += ckSports[i]->objectName()+" ";
80 }
```

**Qt Designer Window (MyWidget):**

- A text label "nihao" is highlighted with a blue box. A blue arrow points from this box to the `tr("按钮是否按下: ")` string in the code.
- Below the label are two groups of controls:
  - 复选框 (Checkboxes):** Three checkboxes labeled "跑步", "踢球", and "游泳".
  - 单选框 (Radio Buttons):** Three radio buttons labeled "很好", "一般", and "不好".
- At the bottom, there is a "TextLabel" widget.

**Application Output (应用程序输出):**

```
myframe x mybutton x
"按钮是否按下: " true
"按钮是否按下: " false
00:50:35: D:\QTcode\3-9\build-mybutton-Desktop_Qt_5_12_12_M
00:50:37: Starting D:\QTcode\3-9\build-mybutton-Desktop_Qt_
QMetaObject::connectSlotsByName: No matching signal for on_
QMetaObject::connectSlotsByName: No matching signal for on_
"按钮是否按下: " true
```

A blue arrow points from the output line `"按钮是否按下: " true` to the `checked` parameter in the `on_pushButton_toggled` function signature.

# 示例9 常用按钮介绍

- QRadioButton

QRadioButton是单选按钮，除了点击后可发出信号触发槽函数外，也会记录是否选中了这个按钮

```
MyWidget::MyWidget(QWidget *parent) :  
    QWidget(parent),  
    ui(new Ui::MyWidget)  
{  
  
    ui->setupUi(this);  
  
    ui->pushBtn1->setText(tr("&nihao"));  
    // 这样便指定了Alt+N为加速键  
  
    ui->pushBtn2->setText(tr("帮助(&H)"));  
    ui->pushBtn2->setIcon(  
        QIcon("../mybutton/images/help.png"));  
  
    rbStatus[0] = ui->rbStatusGood;  
    rbStatus[1] = ui->rbStatusSoso;  
    rbStatus[2] = ui->rbStatusBad;  
    for(int i = 0; i < 3; ++i)  
        connect(rbStatus[i], SIGNAL(clicked()), this, SLOT(on_rbStatus_clicked()));  
}
```



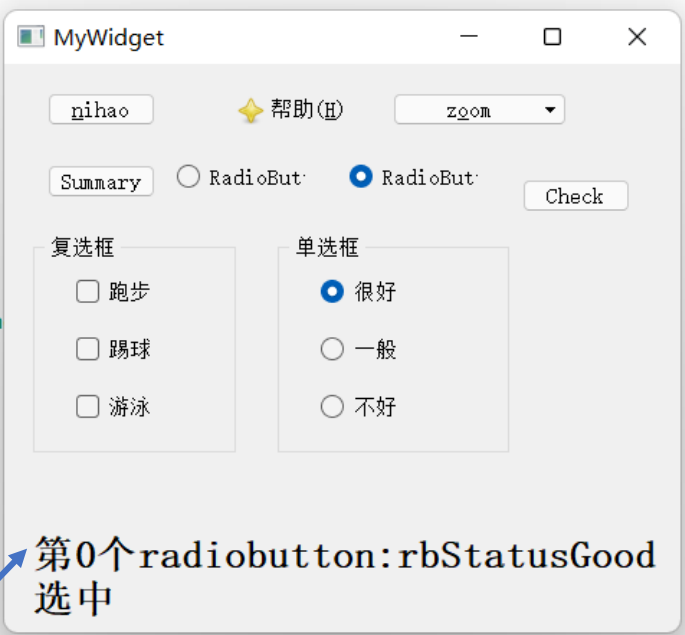
# 示例9 常用按钮介绍

- QRadioButton

在点击QRadioButton对象后，  
触发了槽函数，打印了该按钮  
的名字并且还对主界面利用  
setText函数进行文本的更新

```
82         if( rbStatus[i]->isChecked())
83             s += rbStatus[i]->objectName() + " ";
84         ui->lbInfo->setText(s);
85
86
87     }
88
89
90 void MyWidget::on_rbStatus_clicked()
91 {
92     //QList<QCheckBox *> allCheckboxes= parentWidget->findChildren<QCheckBox *>();
93     QRadioButton * rb = (QRadioButton*)
94         this->sender();
95     std::string objName =
96         rb->objectName().toStdString();
97     QString str = QString::fromStdString(objName);
98     qDebug() << str; //打印事件源头对象名称
99
100     for(int i = 0; i < 3; ++i) {
101         if( rb == rbStatus[i])
102             ui->lbInfo->
103
104     }
105 }
106
```

第0个radiobutton:rbStatusGood选中

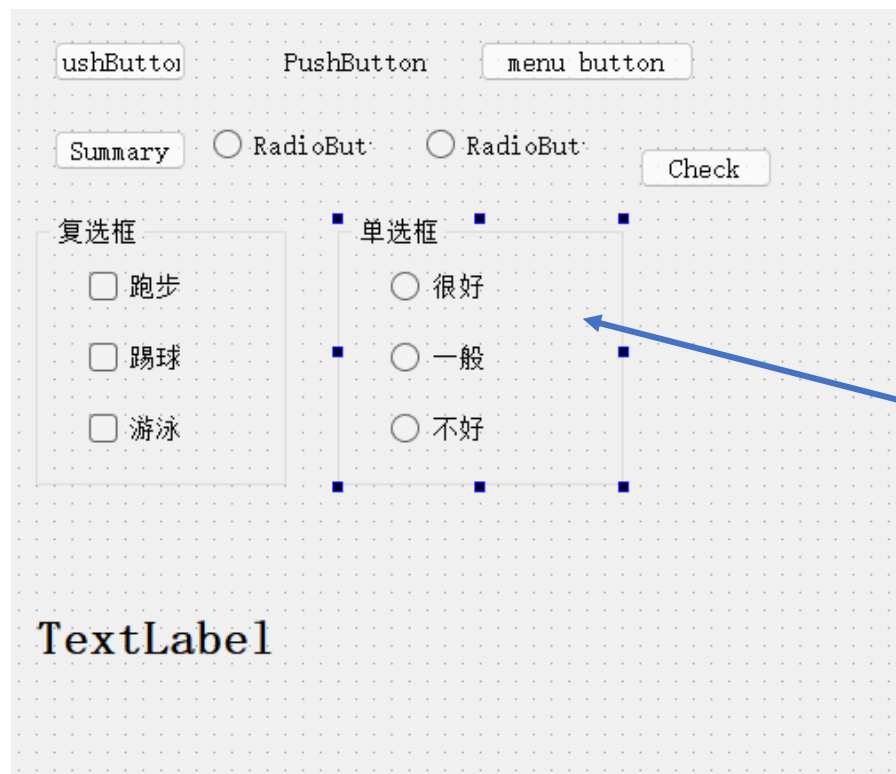


```
应用程序输出
myframe
mybutton
"rbStatusSoso"
"rbStatusGood"
00:57:32: D:\QTcode\3-9\build-mybutton-Desktop_Qt_5_12_12_MinGW_32_bit-Debug\debug\mybutton.exe exited with co
00:57:34: Starting D:\QTcode\3-9\build-mybutton-Desktop_Qt_5_12_12_MinGW_32_bit-Debug\debug\mybutton.exe ...
QMetaObject::connectSlotsByName: No matching signal for on_rbStatus_clicked()
QMetaObject::connectSlotsByName: No matching signal for on_ckSports_clicked(bool)
"rbStatusGood"
```

# 示例9 常用按钮介绍

- QRadioButton

注意：这里是一个  
QRadioButton对象数  
组，数组在对象  
QGroupBox中，**所有**  
QRadioButton对象只  
能有一个**被选中**



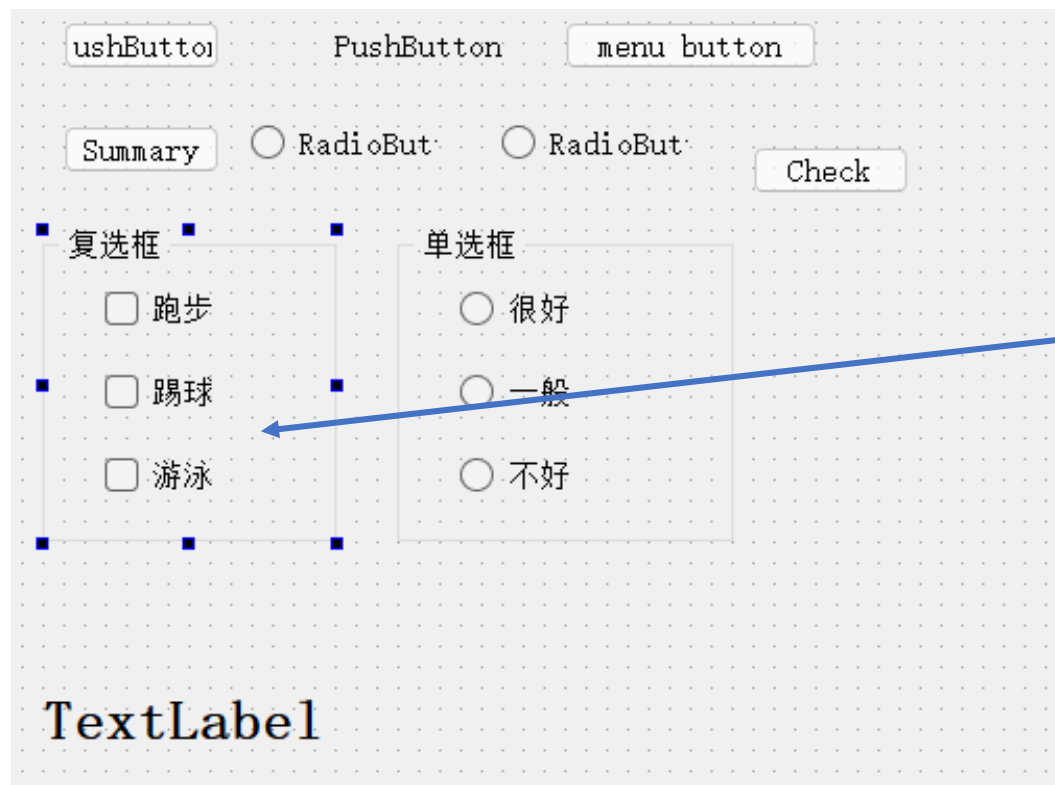
对象	类
MyWidget	QWidget
btCheck	QPushButton
btSummary	QPushButton
groupBox	QGroupBox
ckRun	QCheckBox
ckSocer	QCheckBox
ckSwim	QCheckBox
groupBox_2	QGroupBox
rbStatusBad	QRadioButton
rbStatusGood	QRadioButton
rbStatusSoso	QRadioButton
lblInfo	QLabel
pushBtn1	QPushButton
pushBtn2	QPushButton
pushBtn3	QPushButton
radioButton_4	QRadioButton
radioButton_5	QRadioButton



# 示例9 常用按钮介绍

- QCheckBox

复选框和单选框用法类似，不过复选框 QCheckBox 对象数组中的所有对象都是可以同时选中的



对象	类
MyWidget	QWidget
btCheck	QPushButton
btSummary	QPushButton
groupBox	QGroupBox
ckRun	QCheckBox
ckSocer	QCheckBox
ckSwim	QCheckBox
groupBox_2	QGroupBox
rbStatusBad	QRadioButton
rbStatusGood	QRadioButton
rbStatusSoso	QRadioButton
lblInfo	QLabel
pushBtn1	QPushButton
pushBtn2	QPushButton
pushBtn3	QPushButton

# 示例9 常用按钮介绍

- QMenu

这里为了给按钮提供菜单功能，先声明一个QMenu类的对象，这个类顾名思义是菜单类

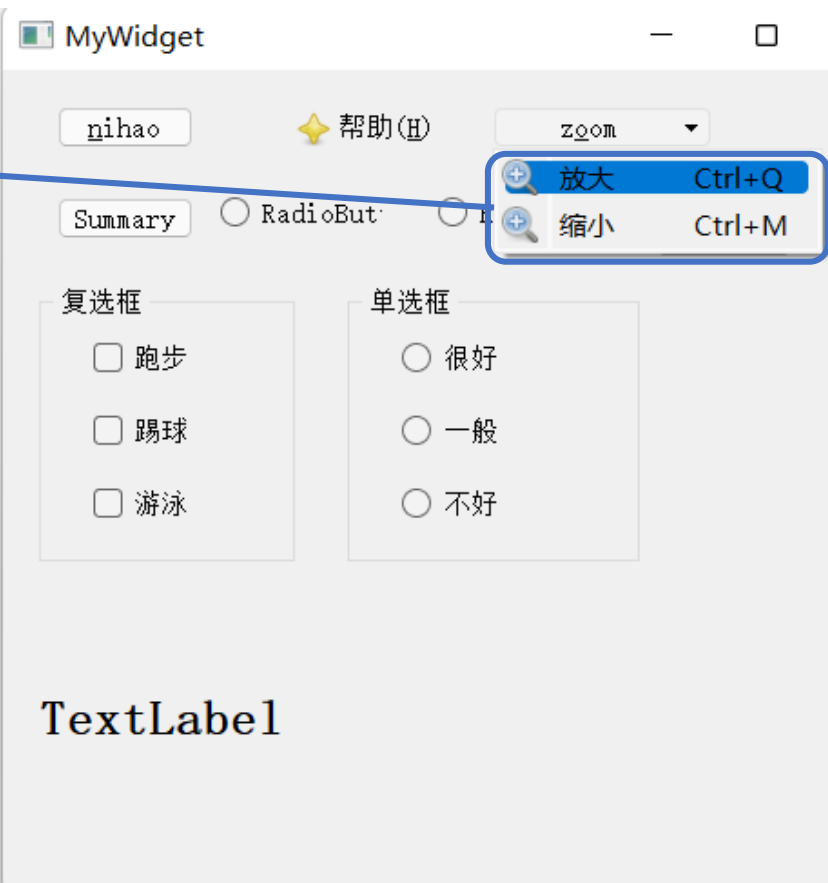
```
ui->pushBtn3->setText(tr("zoom"));
QMenu *menu = new QMenu(this);

act[0] = new QAction(QIcon("../mybutton/images/zoom-in.png"), tr("放大"), this);
act[0]->setShortcut(tr("Ctrl+Q"));
act[0]->setStatusTip(tr("放大一下"));

act[1] = new QAction(QIcon("../mybutton/images/zoom-in.png"), tr("缩小"), this);
act[1]->setShortcut(tr("Ctrl+M"));
act[1]->setStatusTip(tr("缩小一下"));

for(int i = 0; i < 2; ++i) {
    menu->addAction(act[i]);
    connect(act[i], SIGNAL(triggered()), this, SLOT(menuAction()));
}

//menu->addAction(QIcon("../mybutton/images/zoom-in.png"), tr("缩小"));
ui->pushBtn3->setMenu(menu);
```



# 示例9 常用按钮介绍

- QMenu

act数组是之前声明好的QAction类，这里设置了放大和缩小两种行为。

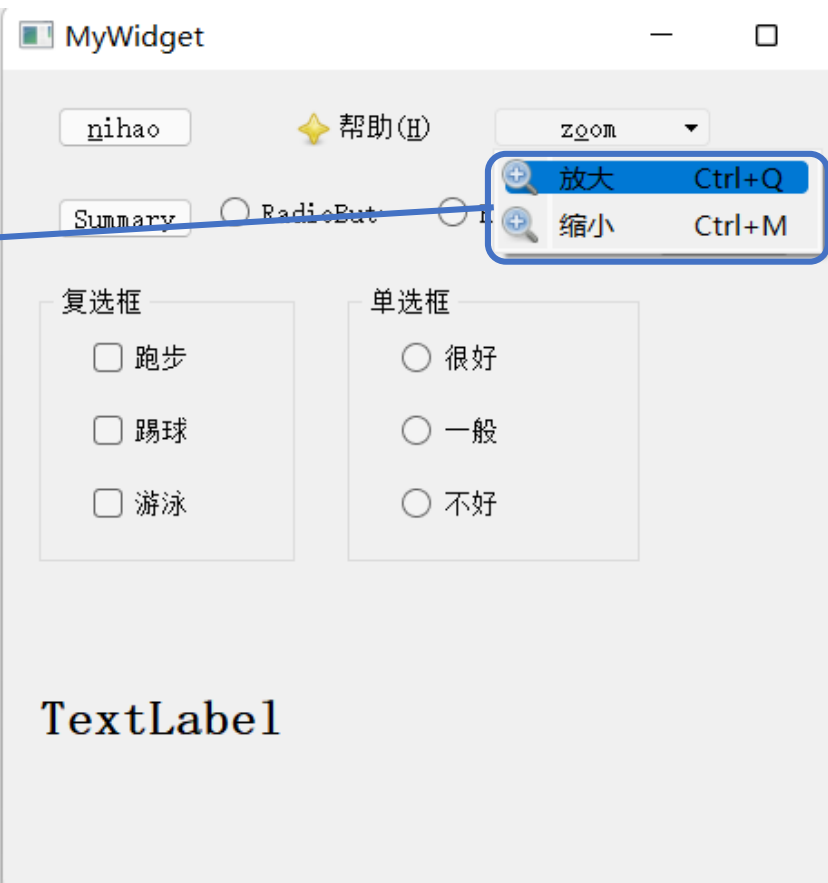
```
ui->pushBtn3->setText(tr("zoom"));
QMenu *menu = new QMenu(this);

act[0] = new QAction(QIcon("../mybutton/images/zoom-in.png"), tr("放大"), this);
act[0]->setShortcut(tr("Ctrl+Q"));
act[0]->setStatusTip(tr("放大一下"));

act[1] = new QAction(QIcon("../mybutton/images/zoom-in.png"), tr("缩小"), this);
act[1]->setShortcut(tr("Ctrl+M"));
act[1]->setStatusTip(tr("缩小一下"));

for(int i = 0; i < 2; ++i) {
    menu->addAction(act[i]);
    connect(act[i], SIGNAL(triggered()), this, SLOT(menuAction()));
}

//menu->addAction(QIcon("../mybutton/images/zoom-in.png"), tr("缩小"));
ui->pushBtn3->setMenu(menu);
```



# 示例9 常用按钮介绍

- QMenu

设置好后便把act数组利用addAction函数添加到QMenu对象中，并利用connect把menu的action和槽函数

SLOT(menuAction())关联

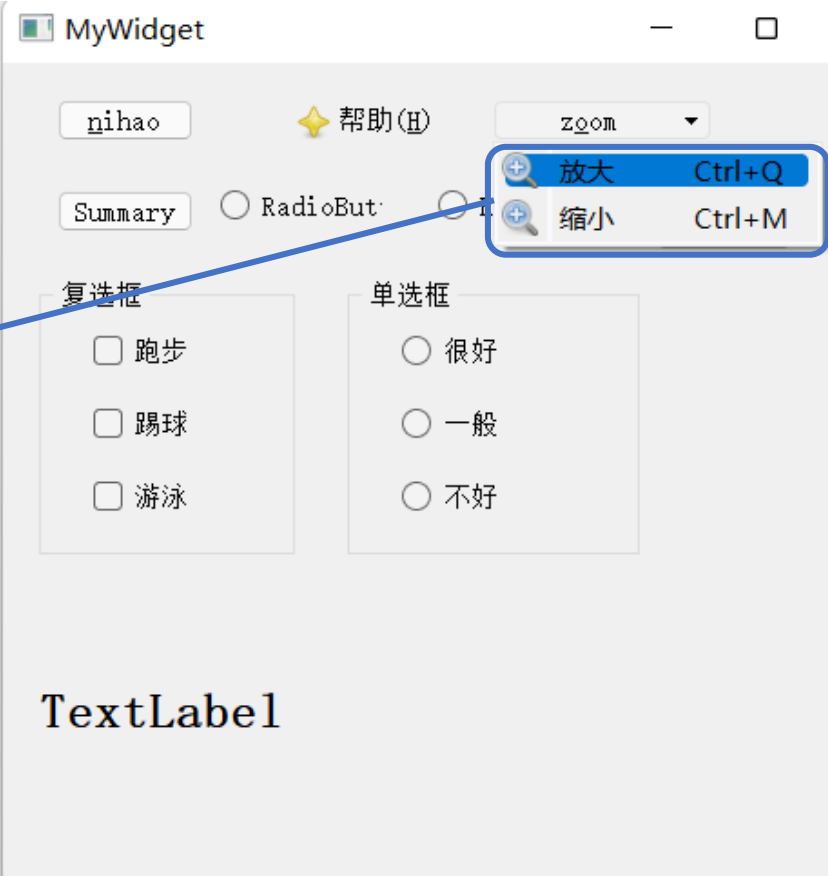
```
ui->pushBtn3->setText(tr("zoom"));
QMenu *menu = new QMenu(this);

act[0] = new QAction(QIcon("../mybutton/images/zoom-in.png"), tr("放大"), this);
act[0]->setShortcut(tr("Ctrl+Q"));
act[0]->setStatusTip(tr("放大一下"));

act[1] = new QAction(QIcon("../mybutton/images/zoom-in.png"), tr("缩小"), this);
act[1]->setShortcut(tr("Ctrl+M"));
act[1]->setStatusTip(tr("缩小一下"));

for(int i = 0; i < 2; ++i) {
    menu->addAction(act[i]);
    connect(act[i], SIGNAL(triggered()), this, SLOT(menuAction()));
}

//menu->addAction(QIcon("../mybutton/images/zoom-in.png"), tr("缩小"));
ui->pushBtn3->setMenu(menu);
```



# 示例9 常用按钮介绍

- QMenu

最后只需要把已经设置好的QMenu对象利用setMenu函数和按钮关联即可

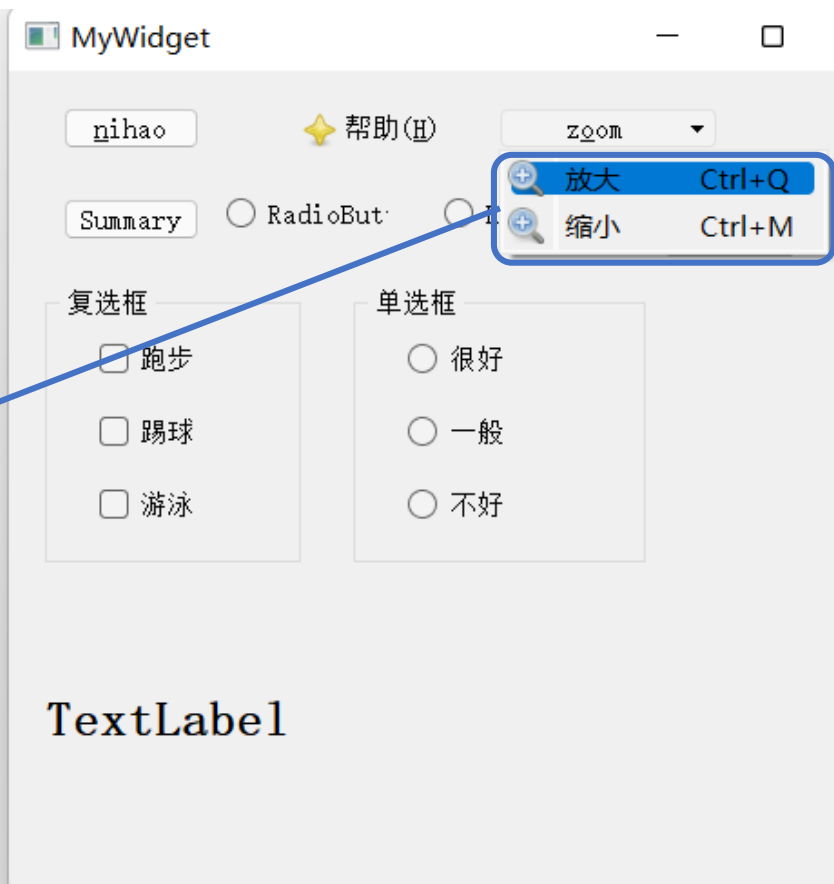
```
ui->pushBtn3->setText(tr("zoom"));
QMenu *menu = new QMenu(this);

act[0] = new QAction(QIcon("../mybutton/images/zoom-in.png"), tr("放大"), this);
act[0]->setShortcut(tr("Ctrl+Q"));
act[0]->setStatusTip(tr("放大一下"));

act[1] = new QAction(QIcon("../mybutton/images/zoom-in.png"), tr("缩小"), this);
act[1]->setShortcut(tr("Ctrl+M"));
act[1]->setStatusTip(tr("缩小一下"));

for(int i = 0; i < 2; ++i) {
    menu->addAction(act[i]);
    connect(act[i], SIGNAL(triggered()), this, SLOT(menuAction()));
}

//menu->addAction(QIcon("../mybutton/images/zoom-in.png"), tr("缩小"));
ui->pushBtn3->setMenu(menu);
```



# 示例10 常用输入框介绍

演示不同的输入框

# 示例10 常用输入框介绍

- QLineEdit简介

1. **QLineEdit**是一个单行文本输入框，带有撤销、剪切、粘贴以及拖拽等功能。

2. 通过改变输入框的echoMode()，同时也可以设置为一个“只写”字段，用于输入密码等。

3. 文本的长度可以被限制为maxLength()，可以使用一个validator()或inputMask()来任意限制文本。

# 示例10 常用输入框介绍

我们产生的界面如下：

显示模式:

输入掩码:

输入验证:

自动完成:

每一个输入框都是一个  
**QLineEdit**类的实例对象

ObjectName从上到下分别为：

lineEdit1

lineEdit2

lineEdit3

lineEdit4



# 示例10 常用输入框介绍

4种echo mode 后面会进一步介绍

```
4  #include <QCompleter>
5
6  MyWidget::MyWidget(QWidget *parent) :
7      QWidget(parent),
8      ui(new Ui::MyWidget)
9  {
10     ui->setupUi(this);
11
12     //QLineEdit 有4种echo mode
13
14     // 新建验证器，指定范围为100-999
15     QIntValidator *validator = new QIntValidator(100, 999, this);
16     rx.setRegex(QString::fromLatin1("?\\d{1,3}"));
17     validator =
18     RegExpValidator(rx, this);
19     //使用验证器
20     ui->setValidator(validator);
21
22     wordList;
23     Qt << "Qt Creator" << tr("你好");
24
25     QCompleter *completer =
26         new QCompleter(wordList, this); // 新建自动完成器
27     completer->setCaseSensitivity(
28         Qt::CaseInsensitive); // 设置大小写不敏感
29     ui->lineEdit4->setCompleter(completer);
30
31 }
```

常量	值	描述
QLineEdit::Normal	0	正常显示输入的字符，默认选项
QLineEdit::NoEcho	1	不显示任何输入，常用于密码类型，其密码长度都需要保密的时候
QLineEdit::Password	2	显示平台相关的密码掩码字符，而不是实际的字符输入
QLineEdit::PasswordEchoOnEdit	3	在编辑的时候显示字符，负责显示密码类型

# 示例10 常用输入框介绍



经过输入的界面如下：

显示模式:

Echo mode: Password

输入掩码:

Echo mode: Normal

输入验证:

自动完成:

输入掩码这一行的object应用了InputMask

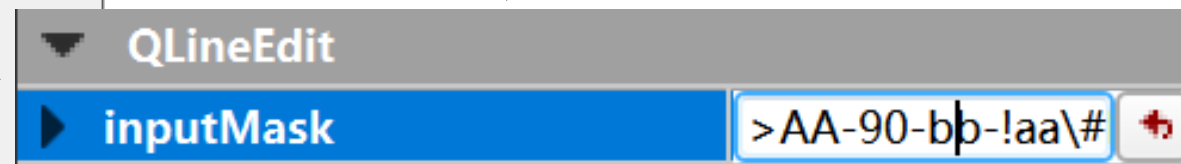
QLineEdit::setInputMask(const QString & inputMask)方法可以使Edit控件只允许输入自定义的格式字符串，**inputMask**参数设置格式化字符串的掩码。

# 示例10 常用输入框介绍



比如这里ui设置了inputMask, 第一位A表示只接受输入ASCII character A-Z, a-z.

其他类似, 详细内容请查看文档



输入掩码这一行的object应用了InputMask

QLineEdit::setInputMask(const QString & inputMask)方法可以使Edit控件只允许输入自定义的格式字符串, inputMask参数设置格式化字符串的掩码。

参考: <https://blog.csdn.net/xgbing/article/details/7776422>

## 示例10 常用输入框介绍

```
MyWidget::MyWidget(QWidget *parent) :  
    QWidget(parent)  
{  
    ui->setupUi(this);  
  
    //QLineEdit 有4种echo mode  
  
    // 新建验证器, 指定范围为100-999  
    //  QValidator *validator = new QIntValidator(100, 999, this);  
  
    QRegExp rx("-?\\d{1,3}"); //用正则表达式规定只能输入最多3个数字或者-  
    QValidator *validator =  
        new QRegExpValidator(rx, this);  
    // 在行编辑器中使用验证器  
    ui->lineEdit3->setValidator(validator);  
  
    QStringList wordList;  
    wordList << "Qt" << "Qt Creator" << tr("你好");  
    QCompleter *completer =  
        new QCompleter(wordList, this); // 新建自动完成器  
    completer->setCaseSensitivity(  
        Qt::CaseInsensitive); // 设置大小写不敏感  
    ui->lineEdit4->setCompleter(completer);  
}
```

通过validator函数也可以进行限制, 这里限制对话框的输入只能为数字

QCompleter能实现自动填充功能, 方便用户输入, 一般和QLineEdit搭配起来使用.

# 示例10 常用输入框介绍

输入验证:

自动完成:

Qt  
Qt Creator

```
MyWidget::MyWidget(QWidget *parent) :  
    QWidget(parent)  
{  
    ui->setupUi(this);  
  
    //QLineEdit 有4种echo mode  
  
    // 新建验证器, 指定范围为100-999  
    //  QValidator *validator = new QIntValidator(100, 999, this);  
  
    QRegExp rx("-?\\d{1,3}"); //用正则表达式规定只能输入最多3个数字或者-  
    QValidator *validator =  
        new QRegExpValidator(rx, this);  
    // 在行编辑器中使用验证器  
    ui->lineEdit3->setValidator(validator);  
  
    QStringList wordList;  
    wordList << "Qt" << "Qt Creator" << tr("你好");  
    QCompleter *completer =  
        new QCompleter(wordList, this); // 新建自动完成器  
    completer->setCaseSensitivity(  
        Qt::CaseInsensitive); // 设置大小写不敏感  
    ui->lineEdit4->setCompleter(completer);  
}
```

QCompleter能实现自动填充功能, 方便用户输入, 一般和QLineEdit搭配起来使用.

# 示例10 常用输入框介绍

```
38 void MyWidget::on_lineEdit2_returnPressed()  
39 {  
40     ui->lineEdit3->setFocus();  
41     qDebug() << ui->lineEdit2->text();  
42     qDebug() << ui->lineEdit2->displayText();  
43 }  
44  
45
```

设置回车键按下的槽

输入完按下回车效果如下:

```
// 回车键按下信号的槽  
  
// 让lineEdit3获得焦点  
// 输出lineEdit2的内容  
// 输出lineEdit2显示的内容
```



控制台输出:

```
Starting C:\DISKD\qtthing\booksample\03\3-10\build-mylineedit-Desktop_Qt_5_6_1_MinGW_32bit-Debug\debug  
\mylineedit.exe...
```

```
"SD-23--fd#2"  
"SD-23-**-fd#2"
```

# 示例11 Spinbox和Datetime edit

---

演示Spinbox和Datetime edit的使用

# 示例11 Spinbox和Datetime edit

介绍QAbstractSpinBox类：

一句话概括：QAbstractSpinBox类提供了一个选择框和一个行编辑来显示值。 类似右图：



QObject、QPaintDevice

----- QWidget 所有用户接口对象的基类

----- QAbstractSpinBox 旋转框和行编辑来显示值

多个子类：

----- QDateTimeEdit

用于编辑日期和时间的小部件

----- QDateEdit

小部件，编辑日期

----- QTimeEdit

小部件，编辑时间

----- QDoubleSpinBox

旋转框小部件，需要double

----- QSpinBox

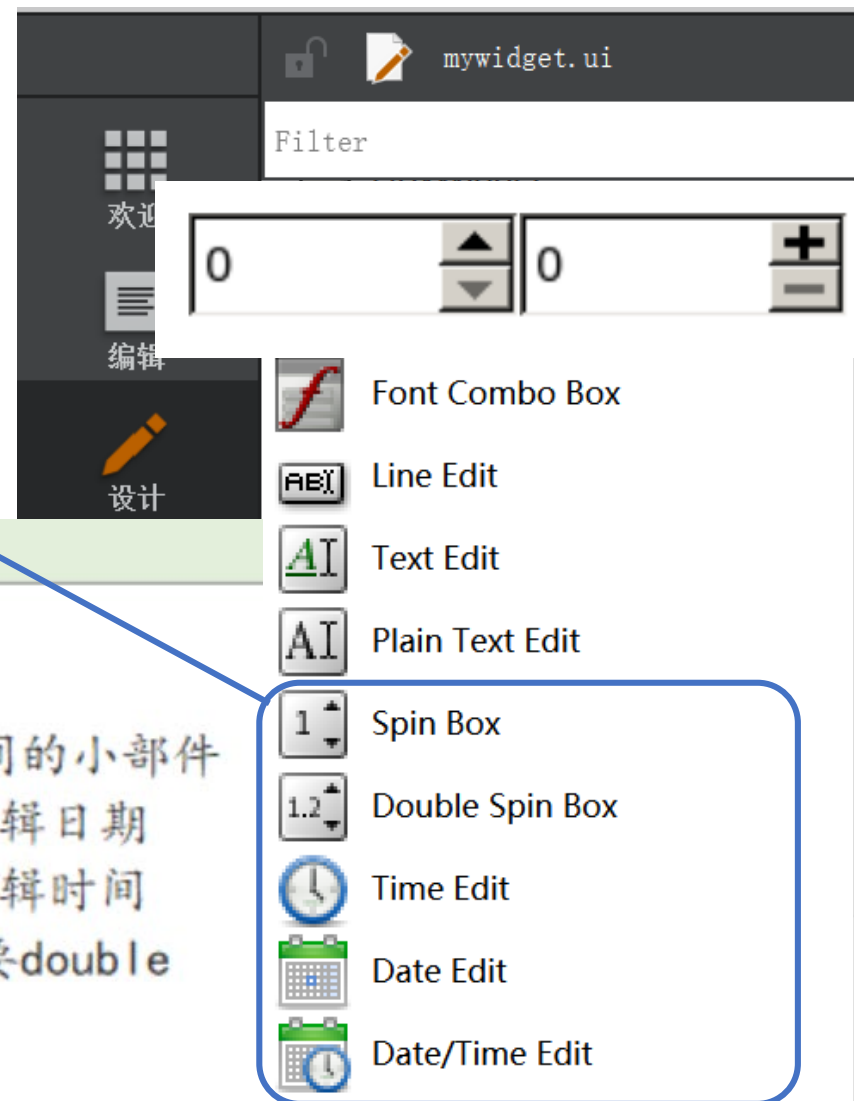
自旋box小部件

接下来逐一讲解



# 示例11 Spinbox和Datetime edit

可以在UI设计界面来选择建立一个  
DateTimeEdit或者SpinBox对象



QObject、QPaintDevice

----- QWidget 所有用户接口对象的基类

----- QAbstractSpinBox 旋转框和行编辑来显示值

多个子类:

----- QDateTimeEdit 用于编辑日期和时间的小部件

----- QDateEdit 小部件, 编辑日期

----- QTimeEdit 小部件, 编辑时间

----- QDoubleSpinBox 旋转框小部件, 需要double

----- QSpinBox 自旋box小部件

# 示例11 Spinbox和Datetime edit

首先我们的程序产生的界面如下：



均是QDateTimeEdit类的实例对象,用于编辑日期和时间



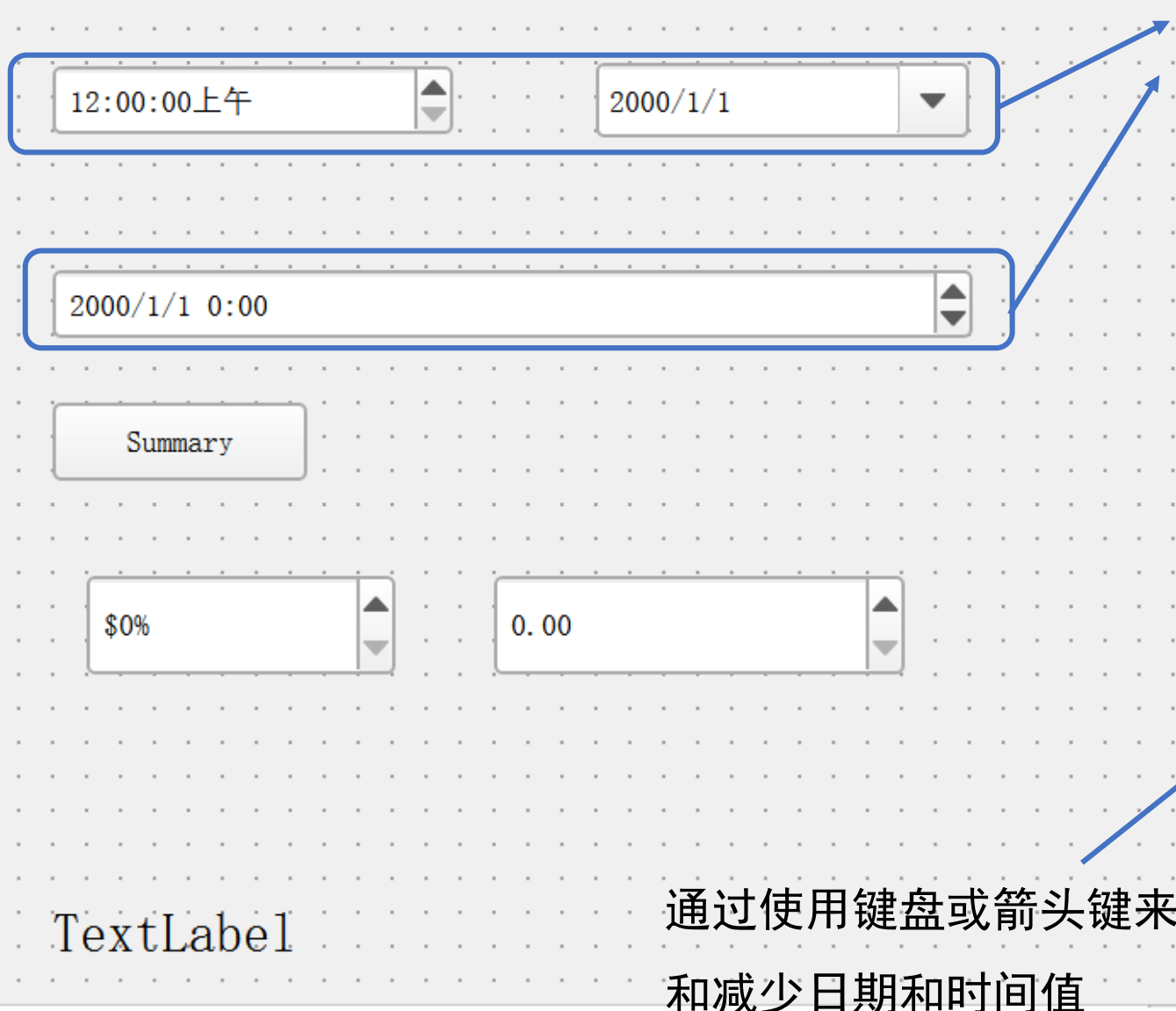
不同的是左上角只编辑时间，  
右上角只编辑日期，下面的日期时间都包含

Summary



TextLabel

# 示例11 Spinbox和Datetime edit

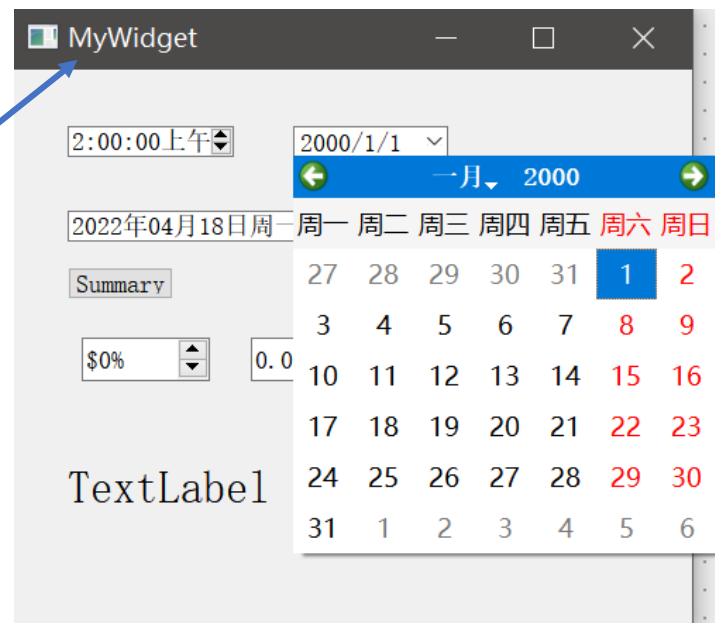


均是QDateTimeEdit类的实例

对象,用于编辑日期和时间

不同的是左上角只编辑时间,  
右上角只编辑日期, 下面的日  
期时间都包含

通过使用键盘或箭头键来增加  
和减少日期和时间值



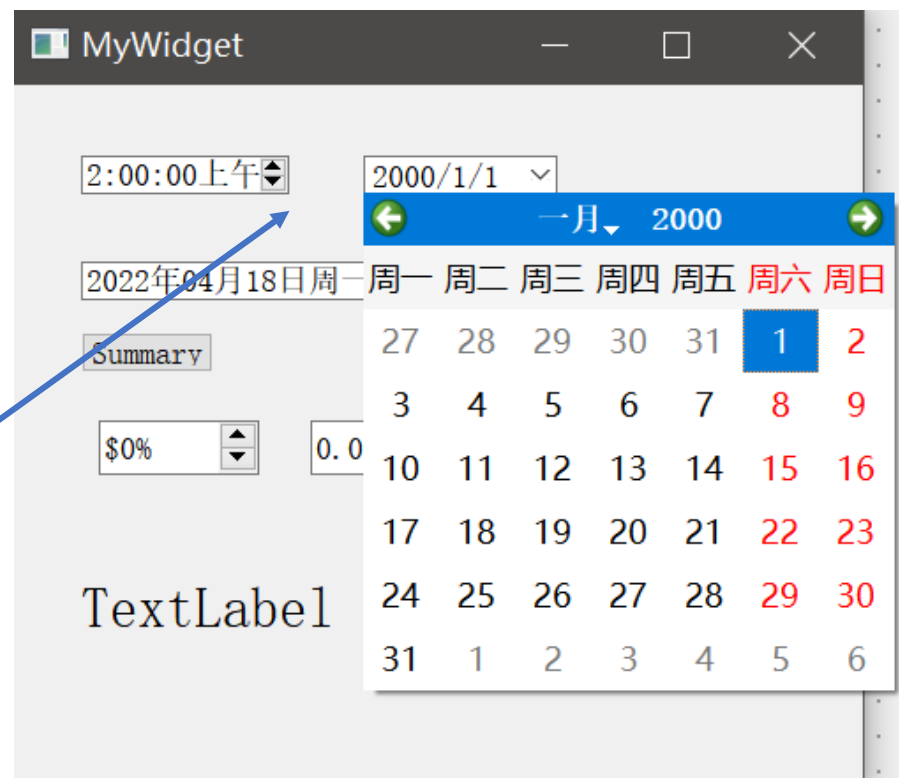
# 示例11 Spinbox和Datetime edit

拓展：此外还可以设置日期、时间显示格式、选择弹出日历、设置日期时间改变时的槽函数等等。这些均与之前讲过的设置类似，详细可参考：

<https://doc.qt.io/qt-5/qdatetimeedit.html>

<https://blog.csdn.net/liang19890820/article/details/52387275>

通过使用键盘或箭头键来增加和减少日期和时间值



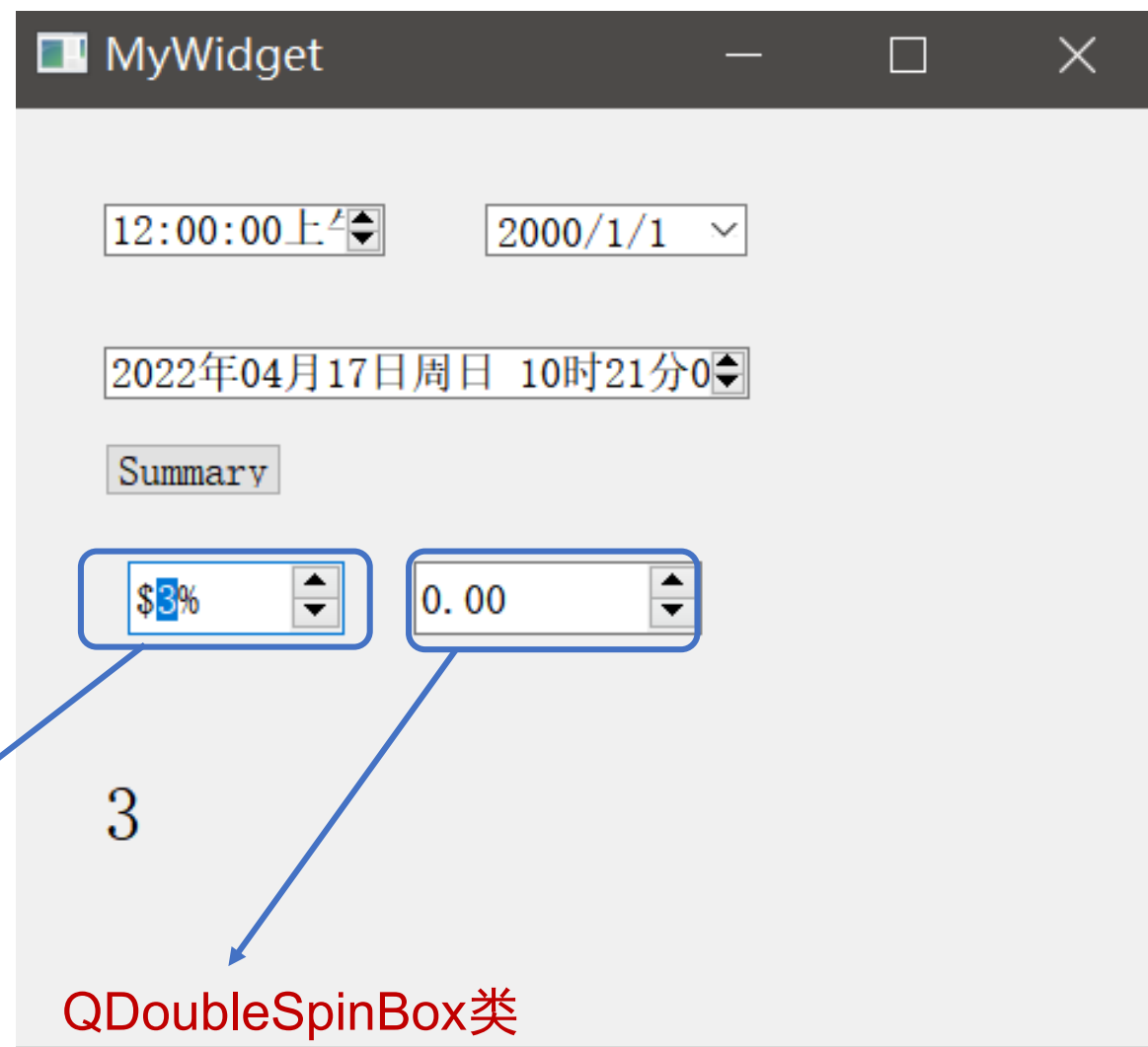
# 示例11 Spinbox和Datetime edit

SpinBox介绍:

QSpinBox旨在处理整数和离散值集（例如月份名称）；使用QDoubleSpinBox来处理浮点值。

QSpinBox允许用户通过单击上/下按钮或按键盘上的上/下按钮来选择一个值，以增加/减少当前显示的值。

QSpinBox类



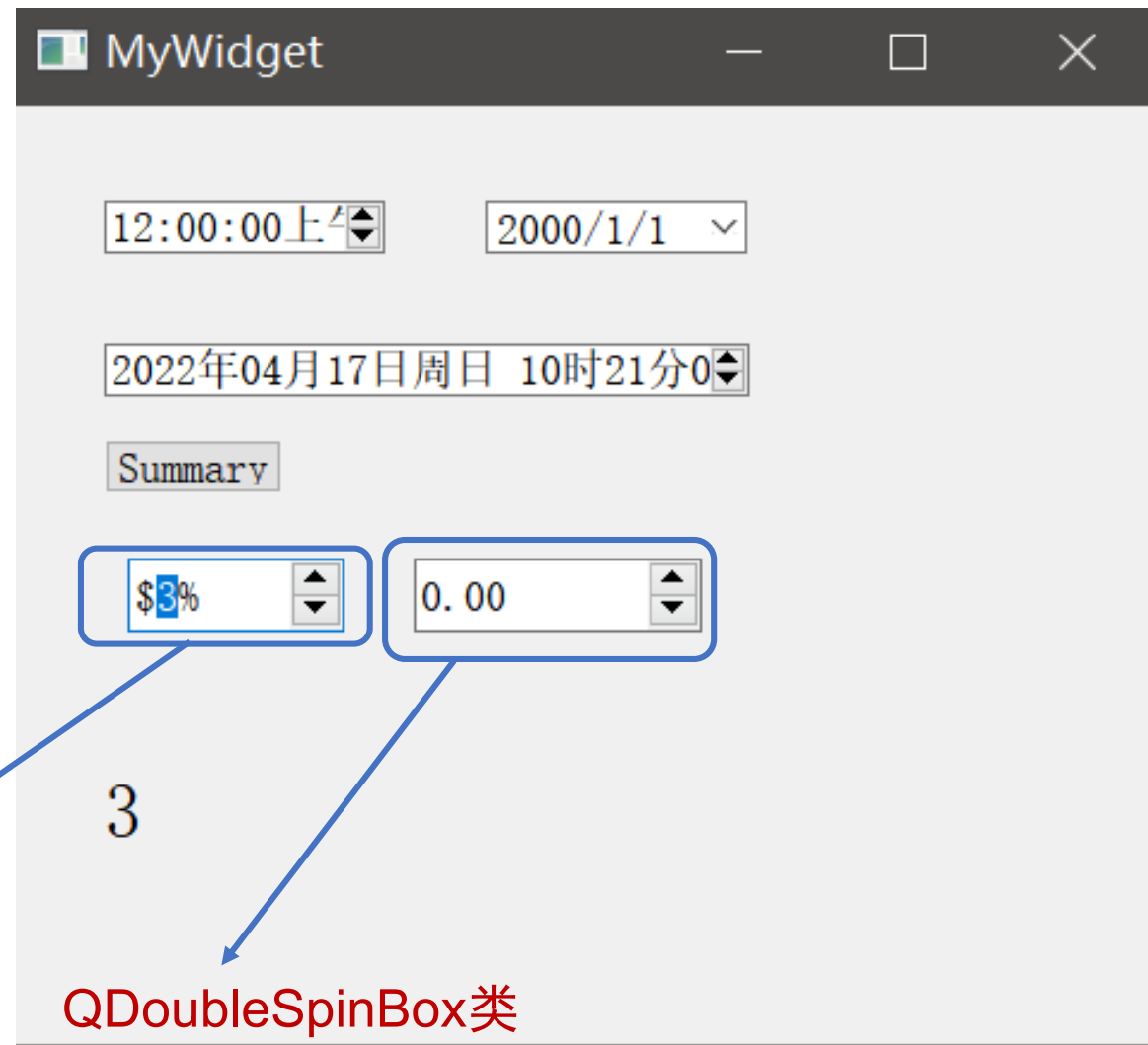
# 示例11 Spinbox和Datetime edit

每次值更改时，QSpinBox都会发出两个  
valueChanged()信号，一个信号提供整数，另一个  
信号提供QString。QString带有数值及其前后缀  
（在这里前缀就是“\$”，后缀就是“%”）。

可以设定相应的槽函数

QSpinBox类

QDoubleSpinBox类

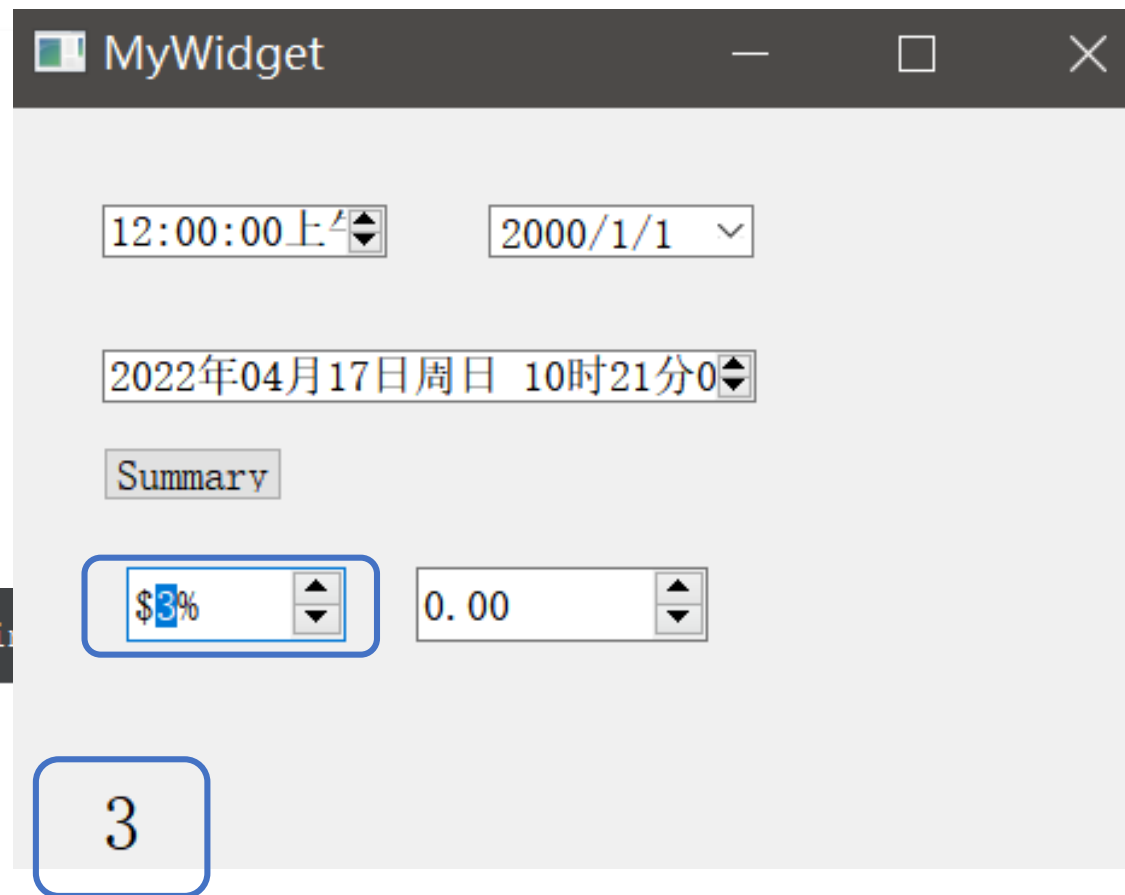


# 示例11 Spinbox和Datetime edit

每次值更改时，QSpinBox都会发出两个valueChanged()信号，一个信号提供整数，另一个信号提供QString。QString带有数值及其前后缀（在这里前缀就是“\$”，后缀就是“%”）。

可以设定相应的槽函数

```
< > mywidget.cpp MyWidget::on_spinBox_valueChanged(int arg1)
17 MyWidget::~MyWidget()
18 {
19     delete ui;
20 }
21
22 void MyWidget::on_spinBox_valueChanged(int arg1)
23 {
24     ui->lbInfo->setText(QString::number(arg1));
25 }
26
```



在这里就设置了spinbox值改变的槽函数，将改变后的值打印在textlabel处

# 示例11 Spinbox和Datetime edit

拓展：QT提供了多个函数接口可以实现设置步长、字符串到数值的转换、开启循环、默认值等等功能。

详细请参考：

<https://doc.qt.io/qt-5/qspinbox.html>

<https://blog.csdn.net/YinShiJiaW/article/details/104896416>



# 示例12 Slider和Dial

---

演示Slider和Dial的使用

# 示例12 Slider和Dial

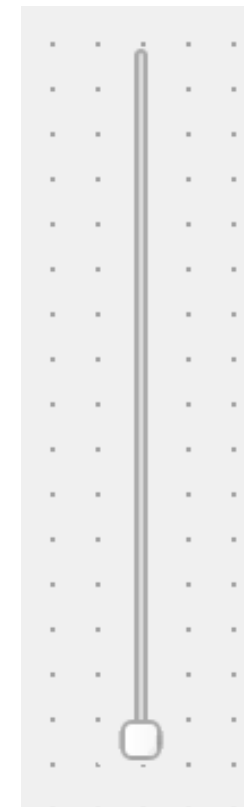
介绍QSlider类：

**QSlider**部件提供了一个垂直或水平滑动条。

滑块是一个用于控制有界值的典型部件。它允许用户沿水平或垂直方向移动滑块，并将滑块所在的位置转换成一个合法范围内的值。

常用的函数是setValue()，用来设置滑块的当前值

大致效果如右所示：



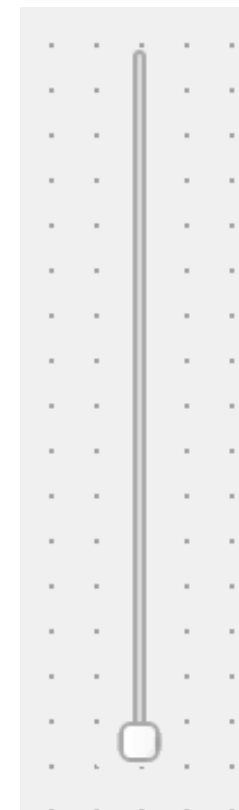
# 示例12 Slider和Dial

介绍Qslider类：

常用的函数是`setValue()`，用来设置滑块的当前值

Value改变时会发出`valueChanged`信号

当tracking参数（一个可选参数）设置为True时，改变value会随之改变Slider的position（就是那个可拖动的块）



MyWidget.cpp

```
20 void MyWidget::on_verticalSlider_valueChanged(int value)
21 {
22     ui->dial->setValue(value);
23     ui->horizontalSlider->setValue(value);
24 }
25
```

后面会讲

这行语句约束了改变verticalSlider会同时改变horizontalSlider

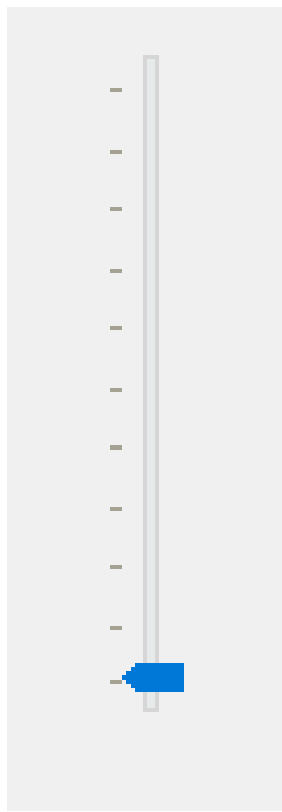
# 示例12 Slider和Dial

QSlider类是QAbstractSlider类的子类，QSlider类又增加了刻度可选功能TickPosition

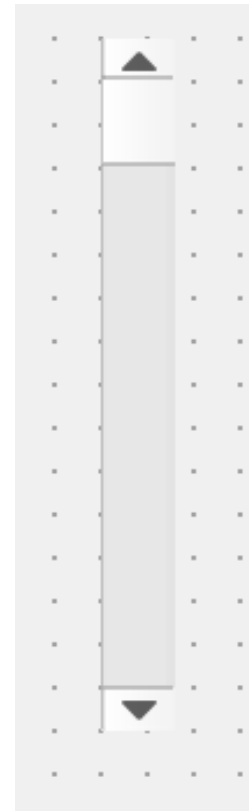
QSlider:



加了Tick的QSlider:



QAbstractSlider:

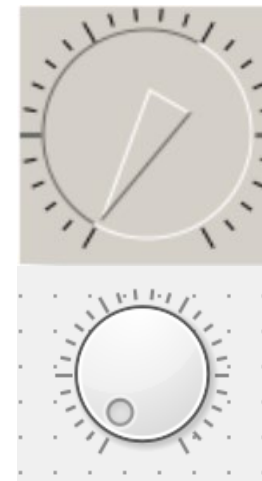


# 示例12 Slider和Dial

介绍QDial类：

和QSlider类似，**Qdial也是QAbstractSlider的子类。**

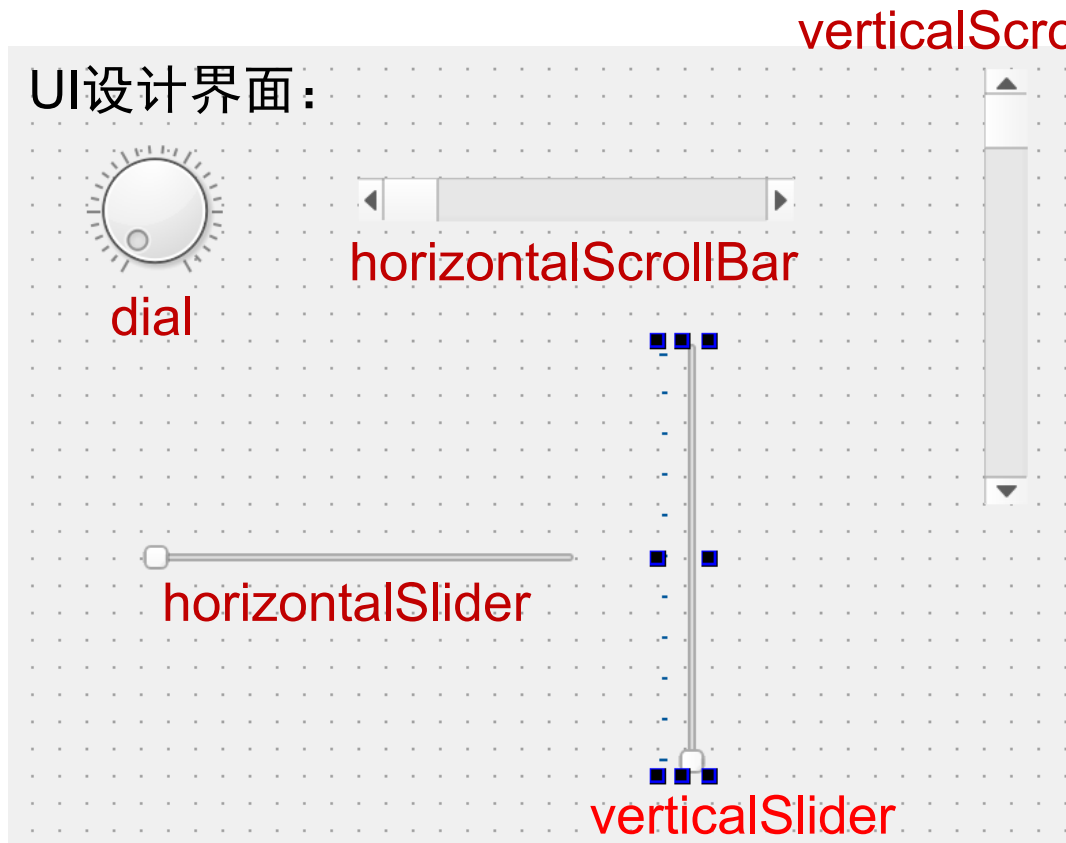
QDial类提供了四舍五入的范围控制（如速度及或者电位计），有一定刻度的圆形控件



关于信号：

- 1.移动滑块时，sliderMoved信号发出
- 2.没有禁用跟踪tracking属性时，valueChanged信号一起发出
- 3.按下/释放鼠标按钮时，转盘发出SliderPressed和SliderReleased信号

# 示例12 Slider和Dial

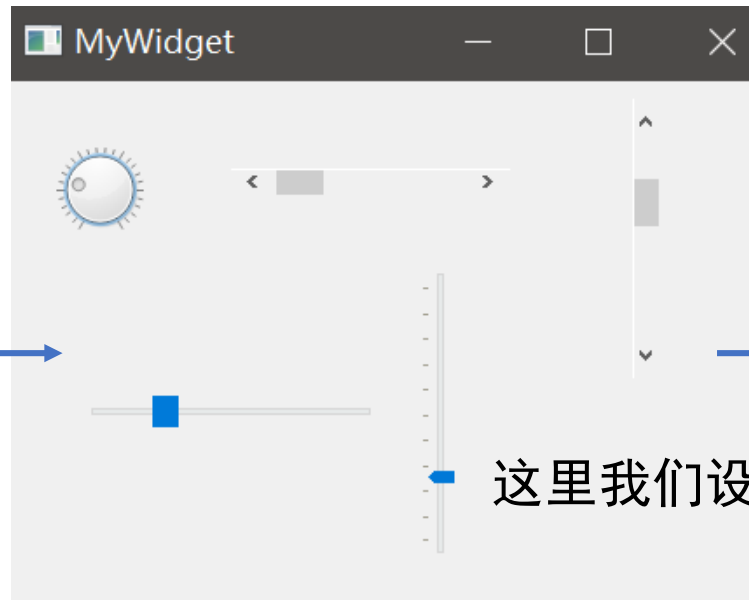
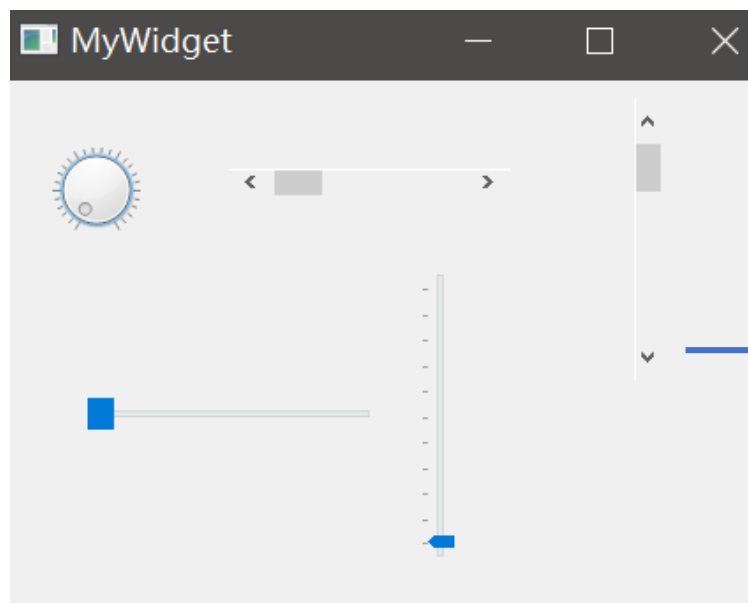


这里我们设置了槽映射: (具体方法可以看示例6)

当dial被鼠标移动时, 发出sliderMoved信号, 然后对应的三个接收者都会调用setValue槽函数, 效果就是部件**同步移动**

发送者	信号	接收者	槽
dial	sliderMoved(int)	verticalScrollBar	setValue(int)
dial	sliderMoved(int)	horizontalSlider	setValue(int)
dial	sliderMoved(int)	verticalSlider	setValue(int)

# 示例12 Slider和Dial



这里我们设置了槽映射：（具体方法可以看示例6）

当dial被鼠标移动时，发出sliderMoved信号，然后对应的三个接收者都会调用setValue槽函数，效果就是

部件**同步移动**

发送者	信号	接收者	槽
dial	sliderMoved(int)	verticalScrollBar	setValue(int)
dial	sliderMoved(int)	horizontalSlider	setValue(int)
dial	sliderMoved(int)	verticalSlider	setValue(int)

# 示例12 Slider和Dial

拓展：更多内容可参考：

<https://doc.qt.io/qt-5/qdial.html>

<https://doc.qt.io/qt-5/qslider.html>