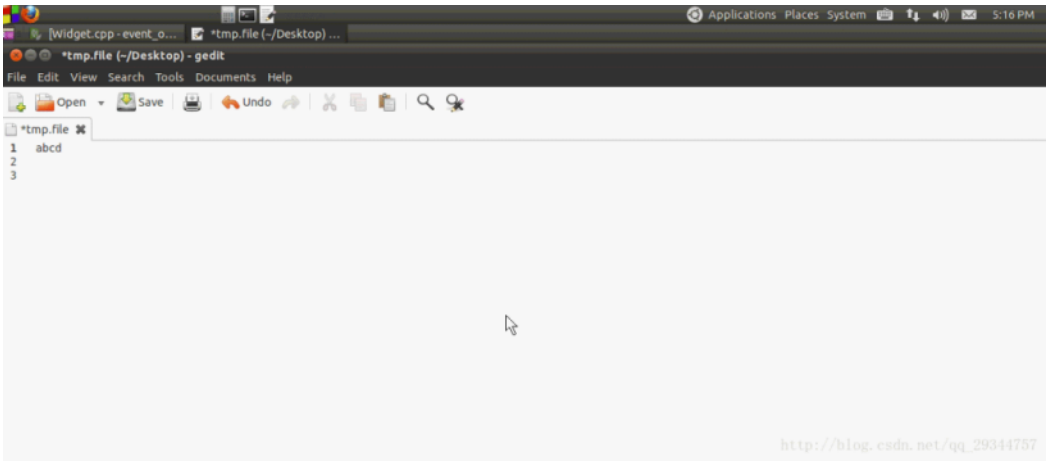
Qt中的事件处理机制

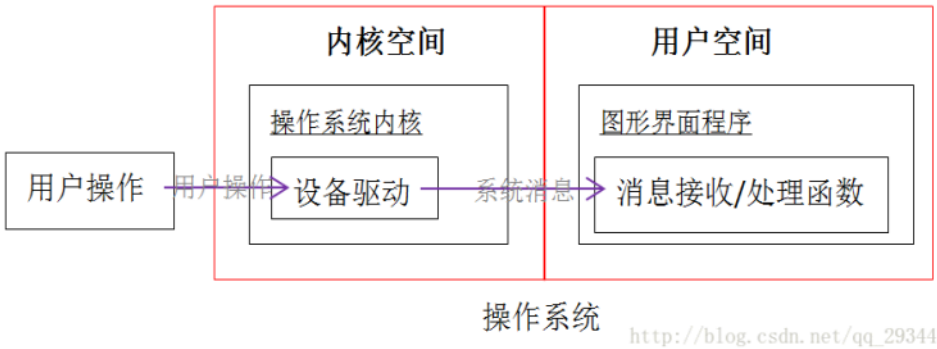
2019年4月8日　　16:24

以ubuntu上的gedit文本编辑器为例，当我们修改了文件内容，在不加以保存的前提下关闭窗口，编辑器就会弹窗提示，并让用户做出选择：



要实现这样的功能？如何做？

一开始自然会想到Qt中的信号和槽机制：程序在合适的地方将该信号与某个槽函数连接，当用户点击关闭按钮的瞬间会产生信号，进而槽函数被调用，实现弹窗操作。但是，对于关闭操作，Qt并没有产生相应的信号，即信号与槽机制不可使用。那么要实现这个功能，就得从Qt的事件处理模型说起。



它反映了包括Qt在内的GUI应用程序的消息处理模型：

(1) 用户操作界面，被操作系统内核空间中的设备驱动程序感知

(2) 设备驱动程序向操作系统的用户空间的GUI应用程序发出系统消息

(3) GUI应用程序将系统消息转换为信号，进而触发槽函数

在GUI应用程序中，一个核心且关键的操作就是将系统消息转换为信号，涉及到Qt的事件处理机制：

(1) Qt平台将系统底层发来的消息转换为Qt事件，并将其Qt事件产生后立即被分发到QWidget对象

(2) QWidget对象中的event(QEvent*)函数对事件进行处理，即根据不同的事件，调用不同的事件处理函数

(3) 在事件处理函数中发送Qt中预定义的对应事件的Qt信号，进而调用到信号关联的槽函数

下面将通过两个阶段，对Qt中的事件处理机制的处理流程进行分析，上述的问题的解答也涵盖其中。

1．事件处理函数

Qt事件是一个QEvent对象，用于描述程序内部(如定时器超时)和外部发生的动作，任何QObject对象都具备事件处理能力。

从上图也可以明显看出，Qt信号是由继承自QObject的QWidget对象中的event()函数调用的事件处理函数发来的。以按钮QPushButton为例，用户对按钮的点击操作，所对应的的事件处理函数是谁?

在Qt Creator开发环境中可以看出，QPushButton继承自QAbstractButton类，在QAbstractButton中有针对鼠标操作的三个事件处理函数:

```
void mousePressEvent(QMouseEvent *e);    //按下鼠标按钮操作
void mouseReleaseEvent(QMouseEvent *e); //松开鼠标按钮操作
void mouseMoveEvent(QMouseEvent *e);     //移动鼠标操作
```

显然，按钮按下操作产生信号是由上面的mousePressEvent()产生的。为验证上述的事件处理流程机制，我们自定义一个继承于QPushButton的MyQPushButton类:

(1) 定义MyQPushButton类

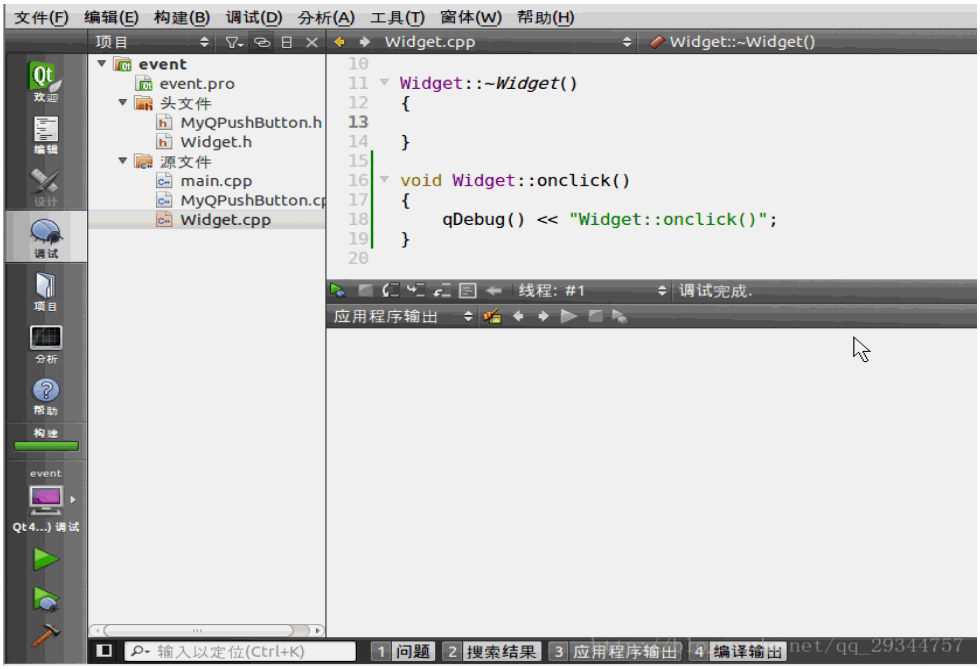
```
//MyQPushButton.h
#ifndef MYQPUSHBUTTON_H
#define MYQPUSHBUTTON_H
#include <QPushButton>
#include <QEvent>

class MyQPushButton : public QPushButton
{
public:
    MyQPushButton(QWidget *parent=0);
};

#endif // MYQPUSHBUTTON_H

//MyQPushButton.cpp
#include "MyQPushButton.h"
#include <QDebug>

MyQPushButton::MyQPushButton(QWidget *parent) : QPushButton(parent)
{
}
```

(2) 在Widget对象中使用MyQPushButton类

```
//Widget.h
#include <QPushButton>
#include "MyQPushButton.h"
#include <QDebug>
class Widget : public QWidget
{
```

```
    Q_OBJECT

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

public slots:
    //定义按钮的槽函数，用于响应按钮按下信号
    void onclick();
};

#endif // WIDGET_H

//Widget.cpp
#include "Widget.h"

Widget::Widget(QWidget *parent) : QWidget(parent), btn(this)
{
    //初始化按钮的相关设置，并映射按钮按下和槽函数
    btn.setText("1");
    btn.move(20, 20);
    connect(&btn, SIGNAL(clicked()), this, SLOT(onclick()));
}

Widget::~Widget()
{

}

//按钮按下的槽函数
void Widget::onclick()
{
    qDebug() << "Widget::onclick()";
}

//main.cpp
#include "Widget.h"

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

    return a.exec();

}
```

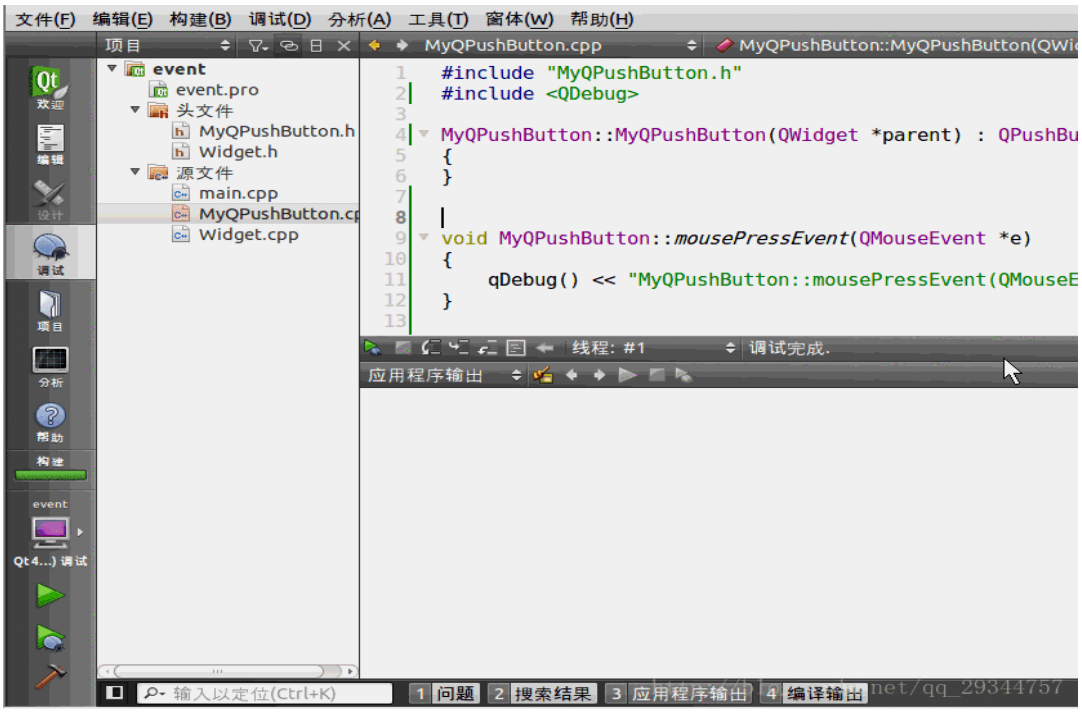
运行:



在MyQPushButton类中重写产生信号的mousePressEvent()函数:

```
//MyQPushButton.h
class MyQPushButton : public QPushButton
{
public:
    MyQPushButton(QWidget *parent=0);
    void mousePressEvent(QMouseEvent *e);
};
//MyQPushButton.cpp
void MyQPushButton::mousePressEvent(QMouseEvent *e)
{
    qDebug() << "MyQPushButton::mousePressEvent(QMouseEvent *e)";
}
```

运行:



(1) 重写后的mousePressEvent()确实被调用，打印出了"MyQPushButton::mousePressEvent(QMouseEvent *e)"

(2) 重写后的mousePressEvent()函数没有发出信号，所以与信号绑定的槽没有被调用

运行效果验证了上面讲的系统消息到信号的转换我的观点是正确的。

(3)另外，注意到一点，没有重写mousePressEvent()函数前，点击按钮，按钮会发生颜色深浅变化，但是重写后，颜色不会发生改变，这说明：在事件处理函数中还需要对UI上的Widget做出相应变化操作。可在代码中加上setDown()函数，该函数会设置按钮被按下的颜色样式，还会触发按下信号，进而槽函数得到调用。

```
void MyQPushButton::mousePressEvent(QMouseEvent *e)
{
    qDebug() << "MyQPushButton::mousePressEvent(QMouseEvent *e)";
    e->accept();
setDown(true);
}
```

回到一开始的问题，一个Widget窗口被鼠标点击关闭前需要弹窗，在Widget类的声明有函数：

```
virtual void closeEvent(QCloseEvent *);
```

索性去看了一下Qt源码qwidget.cpp对该虚函数的实现：

```
void QWidget::closeEvent(QCloseEvent *event)
{
    event->accept();
}
```

该函数只是调用了QEvent类中的accept()函数，它是QEvent的关键成员函数之一。QEvent的关键成员函数有：

```
inline bool isAccepted() const { return m_accept; } //判断当前时间是否被处理
inline void accept() { m_accept = true; }           //接收者已经处理完毕当前事件
inline void ignore() { m_accept = false; }          //接收者忽略当前事件，事件有可能被传递给其父组件
```

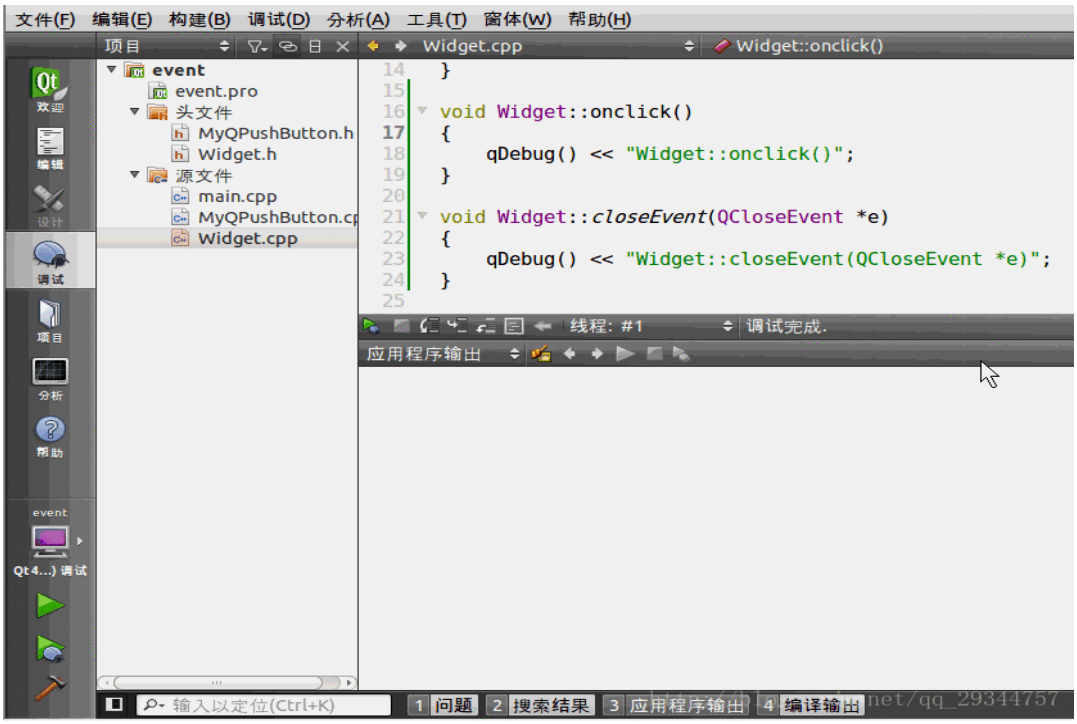
显然QWidget::closeEvent()将事件标志位已经处理完毕，即不是忽略(导致窗口不关闭)也不是忽略(父组件会处理)，所以自然就是响应关闭操作了。

所以要实现窗口关闭前弹窗提示，只需要重写closeEvent()函数即可。是不是这样，测试一下：

```
//Widget.h
class Widget : public QWidget
{
    Q_OBJECT
MyQPushButton btn;
public:
    Widget(QWidget *parent = 0);
    ~Widget();

//重写父类的关闭窗口函数
    void closeEvent(QCloseEvent *e);

public slots:
    void onclick();
};

//Widget.cpp
void Widget::closeEvent(QCloseEvent *e)
{
```

```
    qDebug() << "Widget::closeEvent(QCloseEvent *e)";
}
```

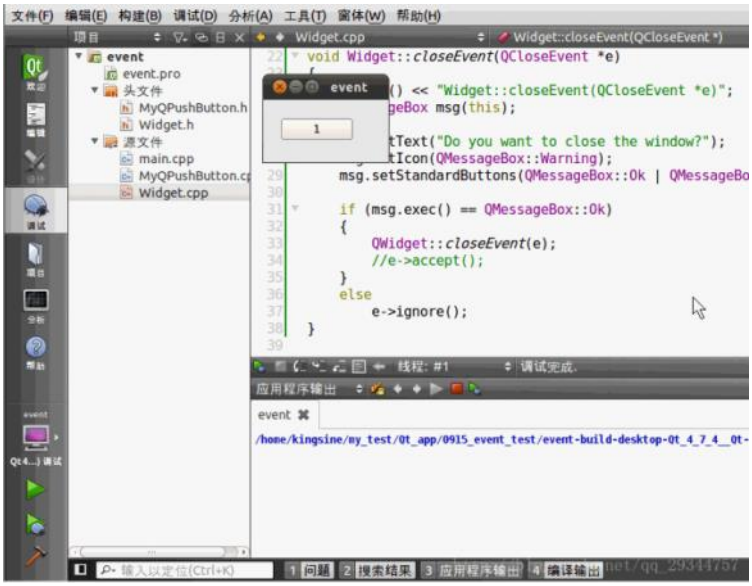
运行：



在窗口被关闭前确定打印出了”Widget::closeEvent(QCloseEvent *e)”。那么弹窗效果在重写函数closeEvent()中实现即可：

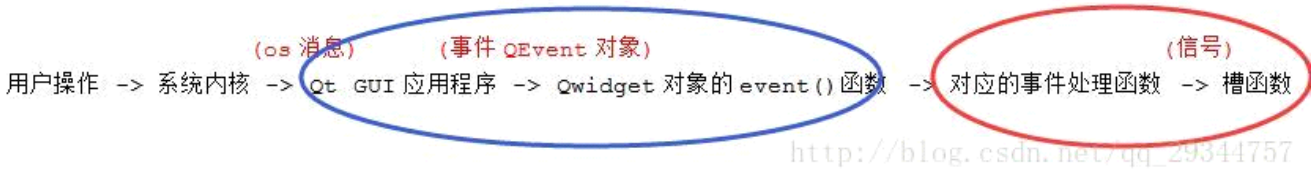
```
void Widget::closeEvent(QCloseEvent *e)
{
    qDebug() << "Widget::closeEvent(QCloseEvent *e)";
    QMessageBox msg(this);
    msg.setText("Do you want to close the window");
    msg.setIcon(QMessageBox::Warning);
    msg.setStandardButtons(QMessageBox::Ok | QMessageBox::Cancel);

   if (msg.exec() == QMessageBox::Ok)
   {
       QWidget::closeEvent(e);
       //e->accept();
   }
   else
       e->ignore();
   }
```

运行：



2. 事件对象QEvent到QWidget对象

用Qt中的每一个窗口和窗口内组件都是继承自QWidget，QWidget又继承自QObject类，使得每个Widget都具有事件处理能力。针对每一种系统消息，每个QObject都有默认的事件操作(忽略也是一种事件操作)。当我们不想使用Qt默认实现的针对某事件的操作时候就可以重写对应的事件处理函数。这个在前面已经实现了，即验证了下图中的红色圈部分。



下来验证的是蓝色圈部分的事件处理流程：

(1) 定义一个继承自QLineEdit的类MyQLineEdit：

```
//MyQLineEdit.h
#ifndef MYQLINEEDIT_H
#define MYQLINEEDIT_H
#include <QLineEdit>
#include <QtGui/QWidget>
#include <QDebug>
#include <QEvent>

class MyQLineEdit : public QLineEdit
{
    Q_OBJECT
public:
    MyQLineEdit(QWidget* parent = 0);
};

#endif // MYQLINEEDIT_H

//MyQLineEdit.cpp
#include "MyQLineEdit.h"

MyQLineEdit::MyQLineEdit(QWidget* parent) : QLineEdit(parent)
{
}
```

(2) 定义Widget并使用MyQlineEdit

```
//Widget.h
#define WIDGET_H
#include <QtGui/QWidget>
#include "MyQLineEdit.h"
#include <QLineEdit>

class Widget : public QWidget
{
    Q_OBJECT

MyQLineEdit line_edit;

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

#endif // WIDGET_H

//Widget.cpp
#include "Widget.h"

Widget::Widget(QWidget *parent) : QWidget(parent), line_edit(this)
{
}

Widget::~Widget()
{}
```

(3)main函数

```
//main.cpp
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    Widget w;
w.resize(240, 132);
    w.show();

return a.exec();
}
```

运行:

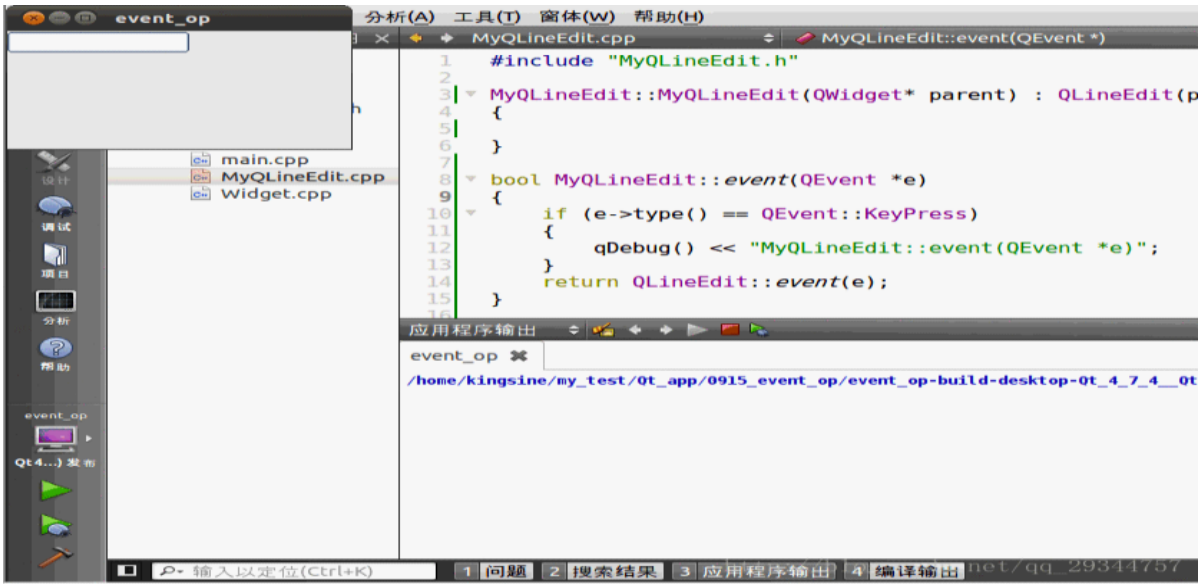
输入框是供用户键盘输入数据的，也就是会有对应的键盘事件。在QLineEdit的声明中可看到键盘事件处理函数:

```
void keyPressEvent(QKeyEvent *);
```

该函数肯定是通过QLineEdit的event()函数调用的，所以在MyQLineEdit重写这两个函数：

```
//MyQLineEdit.h
class MyQLineEdit : public QLineEdit
{
    Q_OBJECT
public:
    MyQLineEdit(QWidget* parent = 0);
    bool event(QEvent *e);
    void keyPressEvent(QKeyEvent* e);
};
//MyQLineEdit.cpp
bool MyQLineEdit::event(QEvent *e)
{
    if (e->type() == QEvent::KeyPress)
    {
        qDebug() << "MyQLineEdit::event(QEvent *e)";
    }
    //交由QLineEdit类中的event()处理
    return QLineEdit::event(e);
}

void MyQLineEdit::keyPressEvent(QKeyEvent* e)
{
    qDebug() << "MyQLineEdit::keyPressEvent(QKeyEvent* e)";

//交由QLineEdit类中的keyPressEvent()处理
    QLineEdit::keyPressEvent(e);
}
```

运行结果：



每次键盘输入字符，都先打印”MyQLineEdit::event(QEvent e)”，然后打印”*MyQLineEdit::keyPressEvent(QKeyEvent* e)*”，同样证实观点。

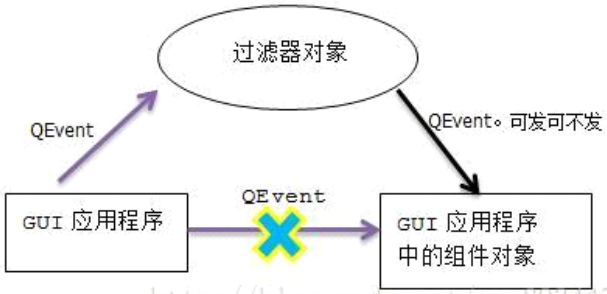
在GUI应用程序将QEvent事件发生给Widget对象过程中，有一个十分关键的角色存在，事件过滤器。顾名思义，事件过滤器可以将GUI程序发来QEvent事件拦截。

3．事件过滤器

下来介绍事件过滤器。定义在父类容器类型组件中的Qt事件过滤器可以对容器内的其他组件收到的事件进行监控，任何QObject对象都可以被作为事件过滤器的过滤目标。

没有安装事件过滤器的父类容器组件对象：

安装事件过滤器的父类容器组件对象：



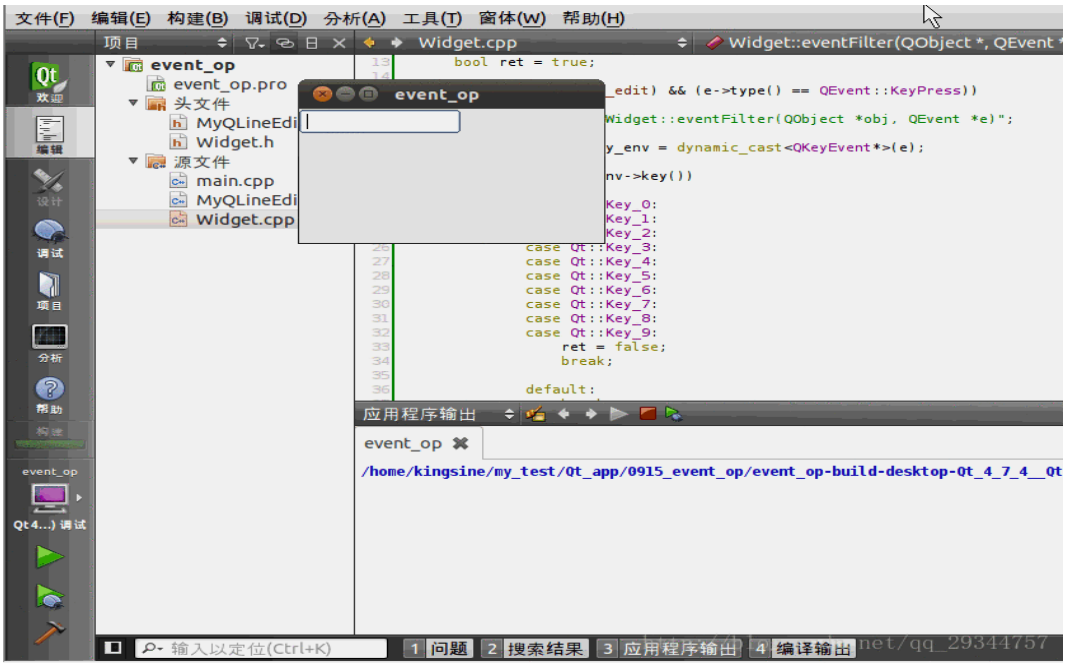
定义的事件过滤器对象需要重写eventFilter()函数，该函数是过滤操作的具体实现，组件再通过installEventFilter()函数安装事件过滤器。安装过滤器后，过滤器会在组件之前接收到事件，事件过滤器能够决定是否将事件转发到组件对象。

基于前面的例子，过滤器组件用于过滤输入的非数字字符：

```
//Widget.h
class Widget : public QWidget
{
    Q_OBJECT
MyQLineEdit line_edit;

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

//过滤器的用于实现过滤操作的函数
    bool eventFilter(QObject *obj, QEvent *e);
};

//Widget.cpp
//构造函数中为MyLineEdit安装过滤器
Widget::Widget(QWidget *parent) : QWidget(parent), line_edit(this)
{
    line_edit.installEventFilter(this);
}

//过滤器的具体过来实现。该函数返回true表示过滤器不将事件对象发回目标组件，反之则发回
bool Widget::eventFilter(QObject *obj, QEvent *e)
{
    bool ret = true;
```

```
//确认监测的目标组件和监测的事件
    if ((obj == &line_edit) && (e->type() == QEvent::KeyPress))
    {
        qDebug() << "Widget::eventFilter(QObject *obj, QEvent *e)";

//强制类型转换事件
        QKeyEvent *key_env = dynamic_cast<QKeyEvent*>(e);

//数字字符可发回目标组件，非数字字符不发回
        switch (key_env->key())
        {
            case Qt::Key_0:
            case Qt::Key_1:
            case Qt::Key_2:
            case Qt::Key_3:
            case Qt::Key_4:
            case Qt::Key_5:
            case Qt::Key_6:
            case Qt::Key_7:
            case Qt::Key_8:
            case Qt::Key_9:
                ret = false;
                break;

            default:
                break;
        }
    }
    else    //非监测目标或者非监测的事件，调用父类的默认过滤函数，即不过滤
        ret = QWidget::eventFilter(obj, e);

    return ret;
}
```

运行:



4. 小结

(1) Qt中的事件(Event)和Qt中的信号(SIGNAL)是两种不同的东西，事件由Object对象(包括继承于

Object对象的QWidth对象)处理，对应的组件由对应的对象处理，处理操作

中其中一个就是触发信号，另一个操作是改变界面对应的显示效果。

(2) Qt GUI程序有着严格的事件处理流程，本文通过两个示例验证这一观点。

(3) Qt事件被处理函数处理完毕够可能还会传递给父类对象，关键看处理函数在调用的是标志已经处理完的accept()还是不做处理的ignore()函数，后者会将事件传递给父类对象。

(4) 事件过滤器用于对目标组件接收的目标事件进行监控，它能决定是否将事件发回目标组件对象。被监测的组件对象需要通过installEventFilter()函数安装事件过滤器。