安装事件过滤器（installEventFilter），过滤子控件事件，截获控件按键、鼠标事件

2019年4月7日 星期日 22:07

Qt的事件模型一个强大的功能是一个QObject对象能够监视发送其他QObject对象的事件，在事件到达之前对其进行处理。

假设我们有一个CustomerInfoDialog控件，由一些QLineEdit控件组成。我们希望使用Space键得到下一个QLineEdit的输入焦点。一个最直接的方法是继承QLineEdit重写keyPressEvent()函数，当点击了Space键时，调用focusNextChild()：

```
void MyLineEdit::keyPressEvent(QKeyEvent *event)
{
   if (event->key() == Qt::Key_Space) {
     focusNextChild();
   } else {
     QLineEdit::keyPressEvent(event);
   }
}
```

这个方法有一个最大的缺点：如果我们在窗体中使用了很多不同类型的控件（QComboBox，QSpinBox等等），我们也要继承这些控件，重写它们的keyPressEvent()。一个更好的解决方法是让CustomerInfoDialog监视其子控件的键盘事件，在监视代码处实现以上功能。这就是事件过滤的方法。实现一个事件过滤包括两个步骤：

1. 在目标对象上调用installEventFilter()，注册监视对象。
2. 在监视对象的eventFilter()函数中处理目标对象的事件。

注册监视对象的位置是在CustomerInfoDialog的构造函数中：

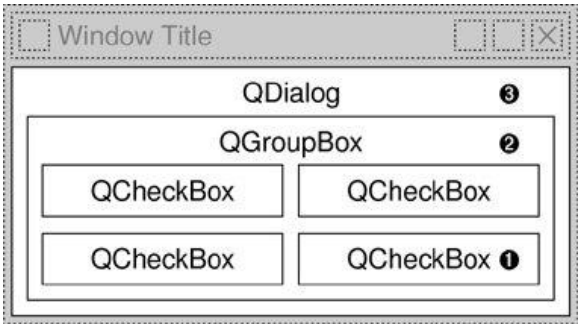
```
CustomerInfoDialog::CustomerInfoDialog(QWidget *parent)
   : QDialog(parent)
{
   ...
   firstNameEdit->installEventFilter(this);
   lastNameEdit->installEventFilter(this);
   cityEdit->installEventFilter(this);
   phoneNumberEdit->installEventFilter(this);
}
```

事件过滤器注册后，发送到firstNameEdit, lastNameEdit, cityEdit, phoneNumberEdit控件的事件首先到达CustomerInfoDialog::eventFilter()函数，然后在到达最终的目的地。
下面是eventFilter()函数的代码：

```
bool CustomerInfoDialog::eventFilter(QObject *target, QEvent *event)
{
   if (target == firstNameEdit || target == lastNameEdit
```

```
        || target == cityEdit || target == phoneNumberEdit) {
    if (event->type() == QEvent::KeyPress) {
        QKeyEvent *keyEvent = static_cast<QKeyEvent *>(event);
        if (keyEvent->key() == Qt::Key_Space) {
            focusNextChild();
            return true;
        }
    }
  }
  return QDialog::eventFilter(target, event);
}
```

首先，我们看是目标控件是否为QLineEdit，如果事件为键盘事件，把QEvent转换为QKeyEvent，确定被敲击的键。如果为Space键，调用focusNextChild()，把焦点交给下一个控件，返回true通知Qt已经处理了这个事件，如果返回false，Qt将会把事件传递给目标控件，把一个空格字符插入到QLineEdit中。
如果目标控件不是QLineEdit，或者事件不是Space敲击事件，把控制权交给基类QDialog的eventFilter()。目标控件也可以是基类QDialog正在监视的控件。（在Qt4.1中，QDialog没有监视的控件，但是Qt的其他控件类，如QScrollArea，监视一些它们的子控件）
Qt的事件处理有5种级别：
1. 重写控件的事件处理函数：如重写keyPressEvent()，mousePressEvent()和paintEvent()，这是最常用的事件处理方法，我们已经看到过很多这样的例子了。
2. 重写QObject::event()，在事件到达事件处理函数时处理它。在需要改变Tab键的惯用法时这样做。也可以处理那些没有特定事件处理函数的比较少见的事件类型（例如，QEvent::HoverEnter）。我们重写event()时，必须要调用基类的event()，由基类处理我们不需要处理的那些情况。
3. 给QObject对象安装事件过滤器：对象用installEventFilter()后，所有达到目标控件的事件都首先到达监视对象的eventFilter()函数。如果一个对象有多个事件过滤器，过滤器按顺序激活，先到达最近安装的监视对象，最后到达最先安装的监视对象。
4. 给QApplication安装事件过滤器，如果qApp（唯一的QApplication对象）安装了事件过滤器，程序中所有对象的事件都要送到eventFilter()函数中。这个方法在调试的时候非常有用，在处理非活动状态控件的鼠标事件时这个方法也很常用。
5. 继承QApplication，重写notify()。Qt调用QApplication::nofity()来发送事件。重写这个函数是在其他事件过滤器处理事件前得到所有事件的唯一方法。通常事件过滤器是最有用的，因为在同一时间，可以有任意数量的事件过滤器，但是notify()函数只有一个。
许多事件类型，包括鼠标，键盘事件，是能够传播的。如果事件在到达目标对象的途中或者由目标对象处理掉，事件处理的过程会重新开始，不同的是这时的目标对象是原目标对象的父控件。这样从父控件再到父控件，知道有控件处理这个事件或者到达了最顶级的那个控件。
图7.2显示了一个键盘事件在一个对话框中从子控件到父控件的传播过程。当用户敲击一个键盘，时间首先发送到有焦点的控件上（这个例子中是QCheckBox）。如果QCheckBox没有处理这个事件，Qt把事件发送到QGroupBox中，如果仍然没有处理，则最后发送到QDialog中。
Figure 7.2. Event propagation in a dialog



==========

Qt事件处理（三）——事件过滤器

目录

前言

`Qt`事件处理（二）——事件处理链中事件的处理过程是**先判断发生事件，控件本身是否对事件进行处理**，然后**传递给父对象**。但是事件还有另外一种处理方法，安装事件过滤器。可以认为事件产生的时候先在父控件中对事件进行处理，然后选择是否传递给子对象进行处理。

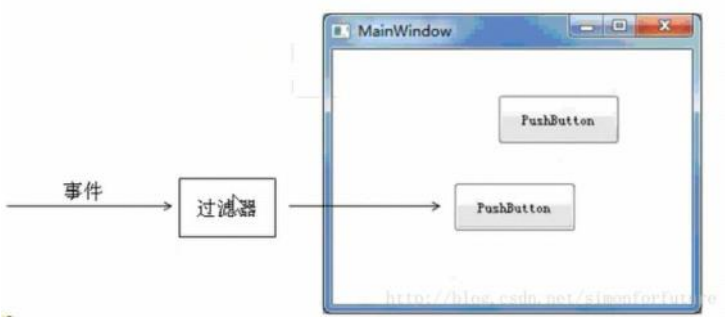
事件过滤器

事件过滤器函数

- `installEventFilter()`和`eventFilter()`

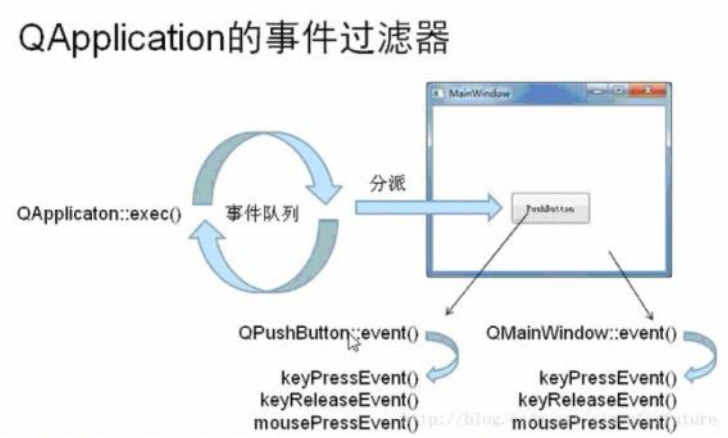
安装事件过滤器的位置

- 在`QObject`中安装事件过滤器



在这张图片中，`MainWindow`中有些控件安装了事件过滤器，有些空间没有安装事件过滤器。对应安装了事件过滤器的，我们可以**重写**`eventFilter()`来在`MainWindow()`中处理事件。前提是我们指定控件`installEventFilter()`安装在`MainWindow()`中。
注意：我们可以对**一个控件安装多个过滤器**，以后有需求继续补充。

- 在`QApplication`中安装事件过滤器



如果给`QApplication`安装事件过滤器，相当于在**所有的控件**对事件进行处理之前，已经在`QApplication`中对事件进行了处理。

`QApplication()`中也可以安装多个事件过滤器。安装的事件过滤器，最后安装的先执行。

```
#include "dialog.h"
#include <QApplication>
class DemoFilter
{
public:
    bool eventFilter(QObject *, QEvent *);

};

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

a.installEventFilter(new DemoFilter);

return a.exec();
}
```

- 子类化`QApplication`并且重新实现`notify()`

我们也可以在子类化`QApplication`，也就是继承`QApplication`，然后重写`notify()`函数来对事件进行处理。

```
class MyApplication : public QApplication
{
    bool notify(QObject*, QEvent*)
    {
   }

};
```

代码

建立三个`QLineEdit`，默认我们按**回车**键来对跳转到下一个`QLineEdit`，但是现在我们修改为按**空格**键来跳转到下一个`QLineEdit`。

给对应的`QObject`安装事件过滤器

- main.cpp

```
#include "dialog.h"
#include <QApplication>
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    Dialog w;
    w.show();

return a.exec();
}
```

- dialog.h

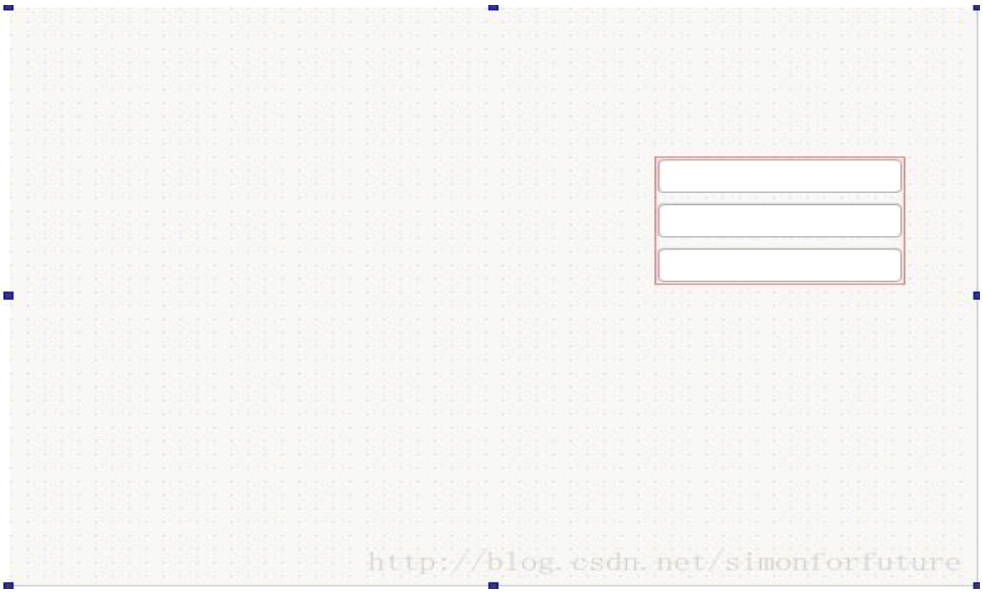
```
#ifndef DIALOG_H
#define DIALOG_H
#include <QDialog>

namespace Ui {
class Dialog;
}

class Dialog : public QDialog
{
    Q_OBJECT

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

bool eventFilter(QObject *, QEvent *);
private:
    Ui::Dialog *ui;
};

#endif // DIALOG_H
```

- dialog.cpp

```
#include "dialog.h"
#include "ui_dialog.h"
```

```
#include <QKeyEvent>

// Dialog is Filter
Dialog::Dialog(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::Dialog)
{
    ui->setupUi(this);

    ui->nameLineEdit->installEventFilter(this);
    ui->idLineEdit->installEventFilter(this);
    ui->phoneNumberLineEdit->installEventFilter(this);
}

Dialog::~Dialog()
{
    delete ui;
}

// target is the object which trigger event
bool Dialog::eventFilter(QObject *target, QEvent *event)
{
    if(target == ui->nameLineEdit ||
            target == ui->idLineEdit ||
            target == ui->phoneNumberLineEdit)
    {
        // QEvent
        if(event->type() == QEvent::KeyPress)
        {
            // QKeyEvent
            QKeyEvent *keyEvent = (QKeyEvent*)event;
            if(keyEvent->key() == Qt::Key_Space)
            {
                focusNextChild();
                return true;
            }
        }
    }
    // QDialog's filter and Dialog's filter
    // if the target has other event, use default event.
    return QDialog::eventFilter(target, event);
}
```

- dialog.ui



这三个窗口的`objectName`分别是`nameLineEdit`，`phoneNumberLineEdit`，`idLineEdit`。

总结

事件处理的方法如下：

1. 重新实现事件处理器
2. 重新实现`QObject::event()`
3. 在`QObject`中安装事件过滤器
4. 在`QApplication`对象中安装事件过滤器（全局事件处理）
5. 子类化`QApplication`并且重新实现`notify()`（全局事件处理）

感觉来说，方法1~方法3都比较适合做事件处理。