DEVBEAN TECH WORLD

Colorful Technologies...

Qt 学习之路 2（21）：事件过滤器

豆子 | 2012年10月15日 | Qt 学习之路 2 | 37条评论

有时候，对象需要查看、甚至要拦截发送到另外对象的事件。例如，对话框可能想要拦截按键事件，不让别的组件接收到；或者要修改回车键的默认处理。

通过前面的章节，我们已经知道，Qt 创建了 QEvent 事件对象之后，会调用 QObject 的 event() 函数处理事件的分发。显然，我们可以在 event() 函数中实现拦截的操作。由于 event() 函数是 protected 的，因此，需要继承已有类。如果组件很多，就需要重写很多个 event() 函数。这当然相当麻烦，更不用说重写 event() 函数还得小心一堆问题。好在 Qt 提供了另外一种机制来达到这一目的：事件过滤器。

QObject 有一个 eventFilter() 函数，用于建立事件过滤器。这个函数的签名如下：

```
1  virtual bool QObject::eventFilter ( QObject * watched, QEvent * event );
```

这个函数正如其名字显示的那样，是一个“事件过滤器”。所谓事件过滤器，可以理解成一种过滤代码。想想做化学实验时用到的过滤器，可以将杂质留到滤纸上，让过滤后的液体溜走。事件过滤器也是如此：它会检查接收到的事件。如果这个事件是我们感兴趣的类型，就进行我们自己的处理；如果不是，就继续转发。这个函数返回一个 bool 类型，如果你想将参数 event 过滤出来，比如，不想让它继续转发，就返回 true，否则返回 false。事件过滤器的调用时间是目标对象（也就是参数里面的 watched 对象）接收到事件对象之前。也就是说，如果你在事件过滤器中停止了某个事件，那么，watched 对象以及以后所有的事件过滤器根本不会知道这么一个事件。

我们来看一段简单的代码：

```
1  class MainWindow : public QMainWindow
2   {
3   public:
4       MainWindow();
5   protected:
6       bool eventFilter(QObject *obj, QEvent *event);
7   private:
8       QTextEdit *textEdit;
9   };
10
11  MainWindow::MainWindow()
12  {
13      textEdit = new QTextEdit;
14      setCentralWidget(textEdit);
15
16      textEdit->installEventFilter(this);
17  }
18
19  bool MainWindow::eventFilter(QObject *obj, QEvent *event)
20  {
21      if (obj == textEdit) {
22          if (event->type() == QEvent::KeyPress) {
23              QKeyEvent *keyEvent = static_cast<QKeyEvent *>(event);
24              qDebug() << "Ate key press" << keyEvent->key();
25              return true;
26          } else {
27              return false;
28          }
29      } else {
30          // pass the event on to the parent class
31          return QMainWindow::eventFilter(obj, event);
32      }
33  }
```

MainWindow 是我们定义的一个类。我们重写了它的 eventFilter() 函数。为了过滤特定组件上的事件，首先需要判断这个对象是不是我们感兴趣的组件，然后判断这个事件的类型。在上面的代码中，我们不想让 textEdit 组件处理键盘按下的事件。所以，首先我们找到这个组件，如果这个事

件是键盘事件，则直接返回 true，也就是过滤掉了这个事件，其他事件还是要继续处理，所以返回 false。对于其它的组件，我们并不保证是不是还有过滤器，于是最保险的办法是调用父类的函数。

eventFilter() 函数相当于创建了过滤器，然后我们需要安装这个过滤器。安装过滤器需要调用 QObject::installEventFilter() 函数。这个函数的签名如下：

```
1  void QObject::installEventFilter ( QObject * filterObj )
```

这个函数接受一个 QObject * 类型的参数。记得刚刚我们说的， eventFilter() 函数是 QObject 的一个成员函数，因此，任意 QObject 都可以作为事件过滤器（问题在于，如果你没有重写 eventFilter() 函数，这个事件过滤器是没有任何作用的，因为默认什么都不会过滤）。已经存在的过滤器则可以通过 QObject::removeEventFilter() 函数移除。

我们可以向一个对象上面安装多个事件处理器，只要调用多次 installEventFilter() 函数。如果一个对象存在多个事件过滤器，那么，最后一个安装的会第一个执行，也就是后进先执行的顺序。

还记得我们前面的那个例子吗？我们使用 event() 函数处理了 Tab 键：

```
1  bool CustomWidget::event(QEvent *e)
2  {
3      if (e->type() == QEvent::KeyPress) {
4          QKeyEvent *keyEvent = static_cast<QKeyEvent *>(e);
5          if (keyEvent->key() == Qt::Key_Tab) {
6              qDebug() << "You press tab.";
7              return true;
8          }
9      }
10     return QWidget::event(e);
11 }
```

现在，我们可以给出一个使用事件过滤器的版本：

```
1  bool FilterObject::eventFilter(QObject *object, QEvent *event)
2  {
3      if (object == target && event->type() == QEvent::KeyPress) {
4          QKeyEvent *keyEvent = static_cast<QKeyEvent *>(event);
5          if (keyEvent->key() == Qt::Key_Tab) {
6              qDebug() << "You press tab.";
7              return true;
8          } else {
9              return false;
10         }
11     }
12     return false;
13 }
```

事件过滤器的强大之处在于，我们可以为整个应用程序添加一个事件过滤器。记得，installEventFilter() 函数是 QObject 的函数， QApplication 或者 QCoreApplication 对象都是 QObject 的子类，因此，我们可以向 QApplication 或者 QCoreApplication 添加事件过滤器。这种全局的事件过滤器将会在所有其它特性对象的事件过滤器之前调用。尽管很强大，但这种行为会严重降低整个应用程序的事件分发效率。因此，除非是不得不使用的情况，否则的话我们不应该这么做。

注意，如果你在事件过滤器中 delete 了某个接收组件，务必将函数返回值设为 true。否则，Qt 还是会将事件分发给这个接收组件，从而导致程序崩溃。

事件过滤器和被安装过滤器的组件必须在同一线程，否则，过滤器将不起作用。另外，如果在安装过滤器之后，这两个组件到了不同的线程，那么，只有等到二者重新回到同一线程的时候过滤器才会有效。

Comments (37)

chsin 2013年1月1日

“事件过滤器和被安装过滤器的组件必须在同一线程，否则，过滤器将不起作用。另外，如果在安装过滤器之后，这两个组件到了不同的线程，那么，只有等到二者重新回到同一线程的时候过滤器才会有效。” 对此不甚理解， 希望豆子能举个例子

回复