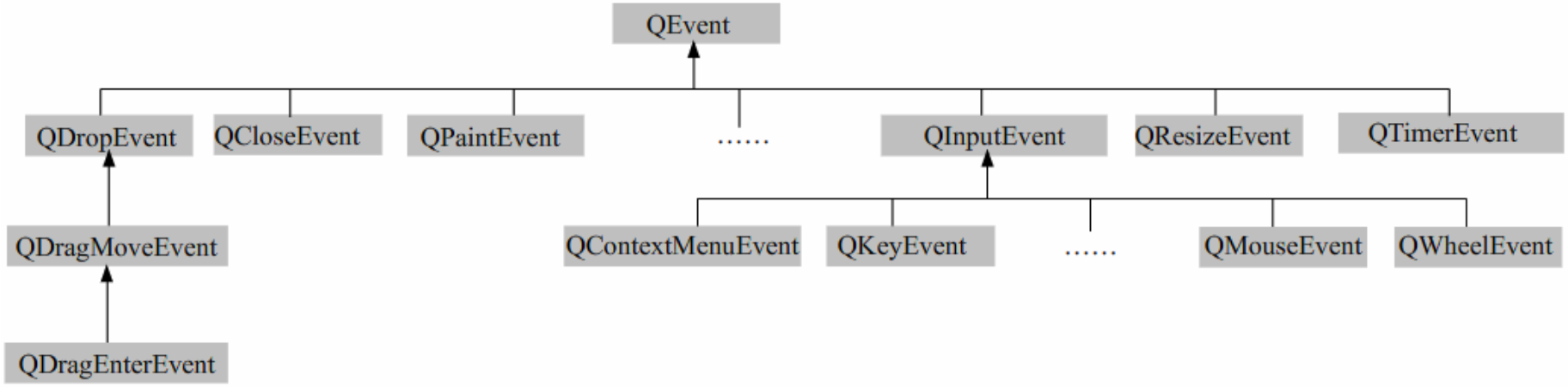
事件系统

软件学院
马玲 张圣林

在Qt中，事件作为一个对象，继承自QEvent类，常见的有键盘事件QKeyEvent、鼠标事件QMouseEvent和定时器事件QTimerEvent等，它们与QEvent类的继承关系如图所示。



主 要 内 容

- Qt中的事件
- 鼠标事件和滚轮事件
- 键盘事件
- 定时器事件与随机数
- 小结

Qt中的事件

事件是对各种应用程序需要知道的由应用程序内部或者外部产生的事情或者动作的通称。在Qt中使用一个对象来表示一个事件，它继承自QEvent类。

事件与信号并不相同，比如我们使用鼠标点击了一下界面上的按钮，那么就会产生鼠标事件QMouseEvent（不是按钮产生的），而因为按钮被按下了，所以它会发出clicked()单击信号（是按钮产生的）。这里一般只关心按钮的单击信号，而不用考虑鼠标事件，但是如果要设计一个按钮，或者当鼠标点击按钮时让它产生别的效果，那么就要关心鼠标事件了。可以看到，事件与信号是两个不同层面的东西，它们的发出者不同，作用也不同。在Qt中，任何QObject的子类的实例都可以接收和处理事件。

常见事件：鼠标事件、键盘事件、定时事件、上下文菜单事件、关闭事件、拖放事件、绘制事件等。

- 事件的处理
- 事件的传递

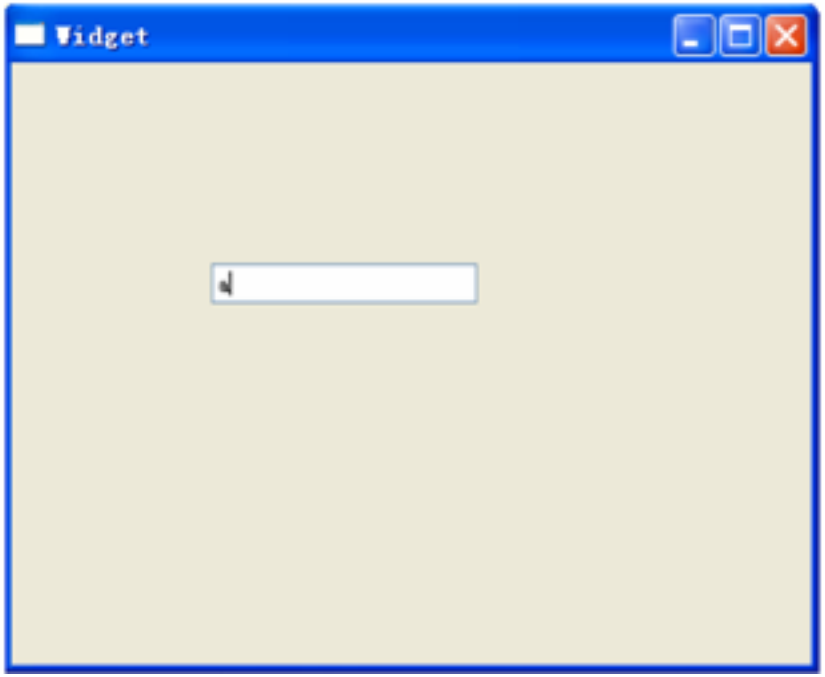
事件的处理

- 方法一：重新实现部件的paintEvent()，mousePressEvent()等事件处理函数。这是最常用也的一种方法，不过它只能用来处理特定部件的特定事件。例如前一章实现拖放操作，就是用的这种方法。
- 方法二：重新实现notify()函数。这个函数功能强大，提供了完全的控制，可以在事件过滤器得到事件之前就获得它们。但是，它一次只能处理一个事件。
- 方法三：向QApplication对象上安装事件过滤器。因为一个程序只有一个QApplication对象，所以这样实现的功能与使用notify()函数是相同的，优点是可以同时处理多个事件。
- 方法四：重新实现event()函数。QObject类的event()函数可以在事件到达默认的事件处理函数之前获得该事件。
- 方法五：在对象上安装事件过滤器。使用事件过滤器可以在一个界面类中同时处理不同子部件的不同事件。

在实际编程中，最常用的是方法一，其次是方法五。

重新实现事件处理函数

例如：使用自定义的Widget作为主窗口（继承自QWidget），然后在上面放置一个自定义的MyLineEdit（继承自QLineEdit）。



- 在MyLineEdit中添加键盘按下事件处理函数声明：

```
protected:
    void keyPressEvent(QKeyEvent *event);
```

- 事件处理函数的定义：

```
void MyLineEdit::keyPressEvent(QKeyEvent *event)    // 键盘按下事件
{
    qDebug() << tr("MyLineEdit键盘按下事件");
    QLineEdit::keyPressEvent(event);   // 执行QLineEdit类的默认事件处理
    event->ignore();                   // 忽略该事件
}
```

- 在Widget中添加键盘按下事件处理函数声明：

```
protected:
    void keyPressEvent(QKeyEvent *event);
```

- 事件处理函数的定义：

```
void Widget::keyPressEvent(QKeyEvent *event)
{
    qDebug() << tr("Widget键盘按下事件");
}
```

从这个例子中可以看到，事件是先传递给指定窗口部件的，这里确切的说应该是先传递给获得焦点的窗口部件的。但是如果该部件忽略掉该事件，那么这个事件就会传递给这个部件的父部件。在重新实现事件处理函数时，一般要调用父类的相应的事件处理函数来实现默认的操作。

安装事件过滤器

在MyLineEdit中添加函数声明：

```
bool event(QEvent *event);
```

该函数定义：

```
bool MyLineEdit::event(QEvent *event)  // 事件
{
   if(event->type() == QEvent::KeyPress)
      qDebug() << tr("MyLineEdit的event()函数");
   return QLineEdit::event(event);  //执行QLineEdit类event()函数的默认操作
}
```

在MyLineEdit的event()函数中使用了QEvent的type()函数来获取事件的类型，如果是键盘按下事件QEvent::KeyPress，则输出信息。因为event()函数具有bool型的返回值，所以在该函数的最后要使用return语句，这里一般是返回父类的event()函数的操作结果。

在Widget中进行事件过滤器函数的声明：

bool *eventFilter*(QObject *obj, QEvent *event);

在widget.cpp文件中的构造函数的最后添上一行代码：

lineEdit->installEvent 装事件过滤器

下面是事件过滤器函

bool Widget::*eventFi* 件过滤器

{

if(obj == lineEdi 件

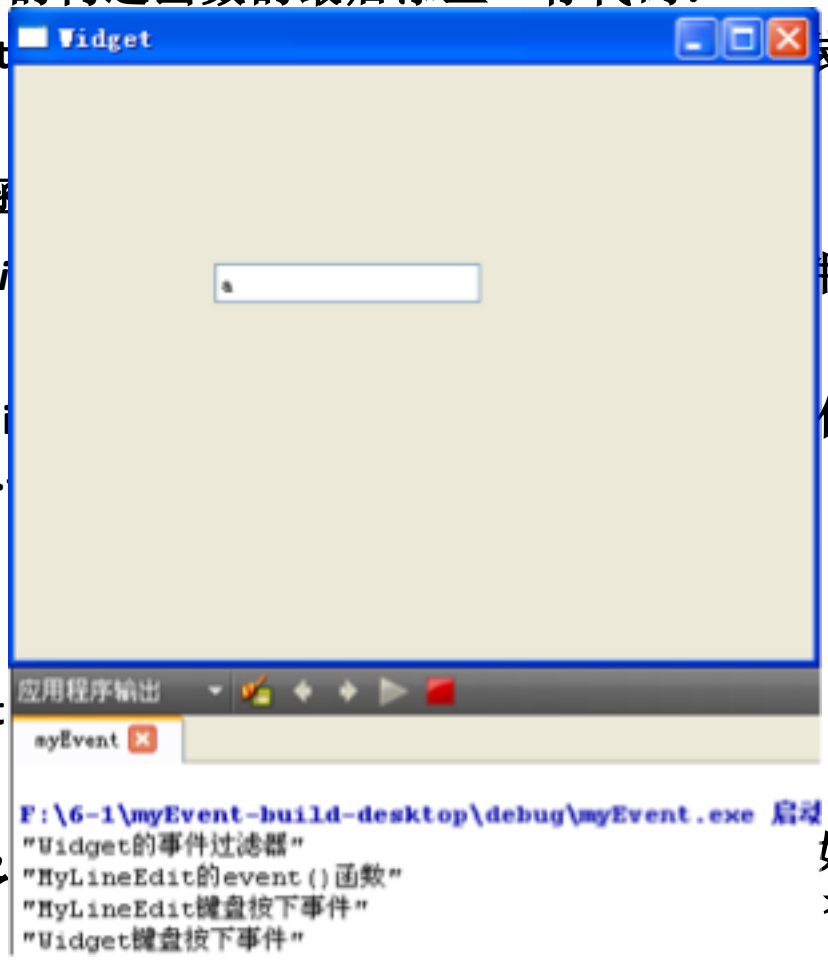
if(event->

qDebug()

}

return QWidget

}

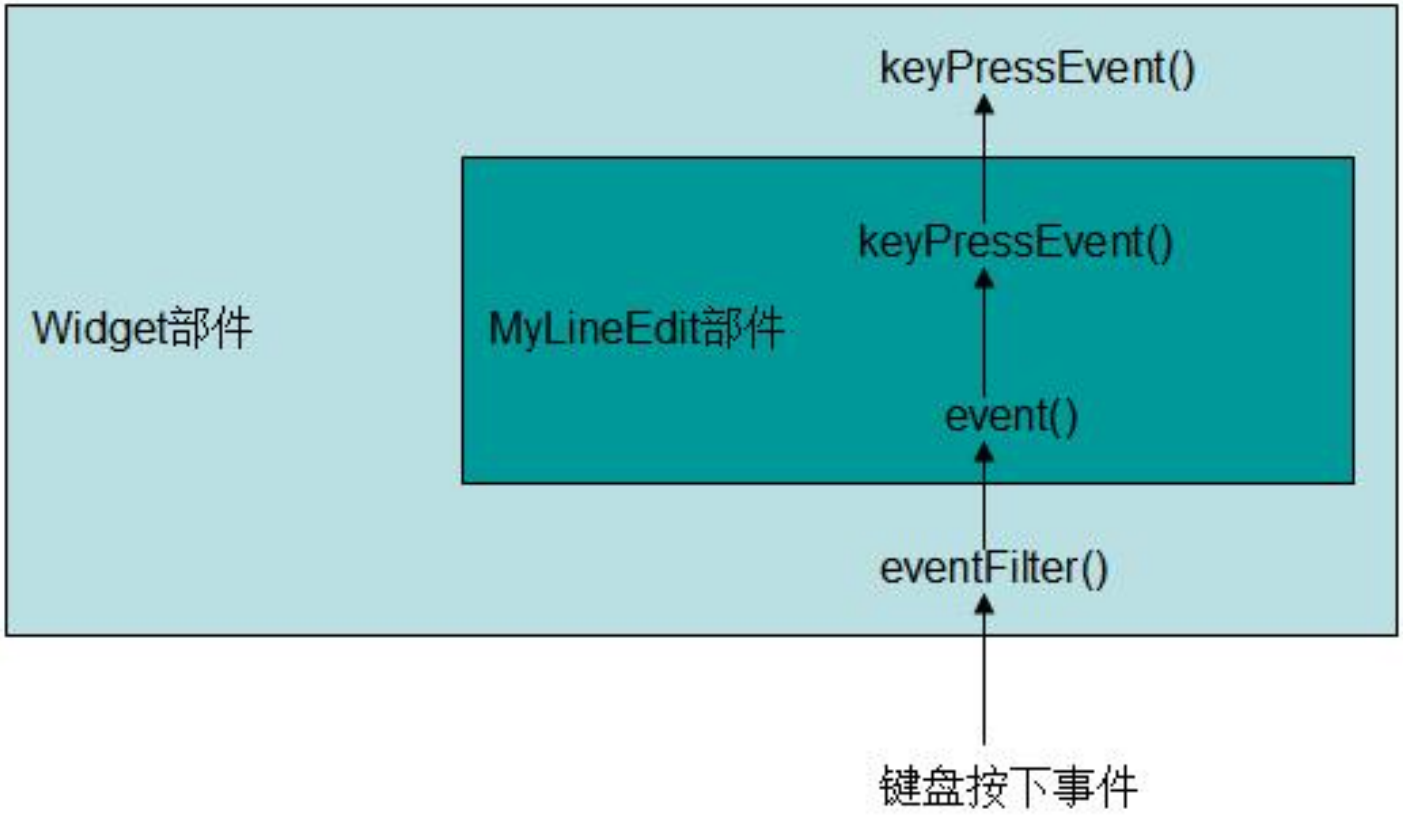


在事件过滤器中， 如果是，再判断事件类型。最后返回了 果。

事件的传递

在每
它会
发生
对象

从前
然后
意，
过滤



数，
收
类的

器，
要注
事件

鼠标事件和滚轮事件

QMouseEvent类用来表示一个鼠标事件，当在窗口部件中按下鼠标或者移动鼠标指针时，都会产生鼠标事件。利用QMouseEvent类可以获知鼠标是哪个键按下了，还有鼠标指针的当前位置等信息。通常是重定义部件的鼠标事件处理函数来进行一些自定义的操作。

QWheelEvent类用来表示鼠标滚轮事件，在这个类中主要是获取滚轮移动的方向和距离。

下面来看一个实际的例子，这个例子要实现的效果是：可以在界面上按着鼠标左键来拖动窗口，双击鼠标左键来使其全屏，按着鼠标右键则使指针变为一个自定义的图片，而使用滚轮则可以放大或者缩小编辑器中的内容。

```
void Widget::mousePressEvent(QMouseEvent *event) // 鼠标按下事件
{
   if(event->button() == Qt::LeftButton){      // 如果是鼠标左键按下
      QCursor cursor;
      cursor.setShape(Qt::ClosedHandCursor);
      QApplication::setOverrideCursor(cursor); // 使鼠标指针暂时改变形状
      offset = event->globalPos() - pos();   // 获取指针位置和窗口位置的差值
   }
   else if(event->button() == Qt::RightButton){ // 如果是鼠标右键按下
      QCursor cursor(QPixmap("../linux.png"));
      QApplication::setOverrideCursor(cursor);// 使用自定义的图片作为鼠标指针
   }
}
```

在鼠标按下事件处理函数中，先判断是哪个按键按下，如果是鼠标左键，那么就更改指针的形状，并且存储当前指针位置与窗口位置的差值。这里使用了globalPos()函数来获取鼠标指针的位置，这个位置是指针在桌面上的位置，因为窗口的位置就是指的它在桌面上的位置。另外，还可以使用QMouseEvent类的pos()函数获取鼠标指针在窗口中的位置。如果是鼠标右键按下，那么就将指针显示为我们自己的图片。

```
void Widget::mouseMoveEvent(QMouseEvent *event) // 鼠标移动事件
{
   if(event->buttons() & Qt::LeftButton){     // 这里必须使用buttons()
   QPoint temp;
   temp = event->globalPos() - offset;
   move(temp);// 使用鼠标指针当前的位置减去差值，就得到了窗口应该移动的位置
   }
}
```

在鼠标移动事件处理函数中，先判断是否是鼠标左键按下，如果是，那么就使用前面获取的差值来重新设置窗口的位置。因为在鼠标移动时，会检测所有按下的键，而这时使用QMouseEvent的button()函数无法获取哪个按键被按下，只能使用buttons()函数，所以这里使用buttons()和Qt::LeftButton进行按位与的方法来判断是否是鼠标左键按下。

```
void Widget::mouseReleaseEvent(QMouseEvent *event) // 鼠标释放事件
{
   QApplication::restoreOverrideCursor();       // 恢复鼠标指针形状
}
```

在鼠标释放函数中进行了恢复鼠标形状的操作，这里使用的restoreOverrideCursor()函数要和前面的setOverrideCursor()函数配合使用。

```
void Widget::mouseDoubleClickEvent(QMouseEvent *event) // 鼠标双击事件
{
    if(event->button() == Qt::LeftButton){          // 如果是鼠标左键按下
        if(windowState() != Qt::WindowFullScreen)    // 如果现在不是全屏
            setWindowState(Qt::WindowFullScreen);    // 将窗口设置为全屏
        else setWindowState(Qt::WindowNoState);      // 否则恢复以前的大小
    }
}
```

在鼠标双击事件处理函数中使用setWidowState()函数来使窗口处于全屏状态或者恢复以前的大小。

```
void Widget::wheelEvent(QWheelEvent *event)   // 滚轮事件
{
   if(event->delta() > 0){                    // 当滚轮远离使用者时
      ui->textEdit->zoomIn();                 // 进行放大
   }else{                                     // 当滚轮向使用者方向旋转时
      ui->textEdit->zoomOut();                // 进行缩小
   }
}
```

在滚轮事件处理函数中，使用QWheelEvent类的delta()函数获取了滚轮移动的距离，每当滚轮旋转一下，默认的是15度，这时delta()函数就会返回15*8即整数120。当滚轮向远离使用者的方向旋转时，返回正值；当向着靠近使用者的方向旋转时，返回负值。这样便可以利用这个函数的返回值来判断滚轮的移动方向，从而进行编辑器中内容的放大或者缩小操作。

键盘事件

QKeyEvent类用来描述一个键盘事件。当键盘按键被按下或者被释放时，键盘事件便会被发送给拥有键盘输入焦点的部件。

QKeyEvent的key()函数可以获取具体的按键，需要特别说明的是，回车键在这里是Qt::Key_Return；键盘上的一些修饰键，比如Ctrl和Shift等，这里需要使用QKeyEvent的modifiers()函数来获取它们。例如：

```
void Widget::keyPressEvent(QKeyEvent *event)      // 键盘按下事件
{
   if(event->modifiers() == Qt::ControlModifier){ // 是否按下Ctrl键
      if(event->key() == Qt::Key_M)               // 是否按下M键
         setWindowState(Qt::WindowMaximized);   // 窗口最大化
   }
   else QWidget::keyPressEvent(event);
}

void Widget::keyReleaseEvent(QKeyEvent *event)    // 按键释放事件
{
   // 其他操作
}
```

定时器事件与随机数

QTimerEvent类用来描述一个定时器事件。对于一个QObject的子类，只需要使用int QObject::startTimer (int interval)函数来开启一个定时器，这个函数需要输入一个以毫秒为单位的整数作为参数来表明设定的时间，它返回一个整型编号来代表这个定时器。当定时器溢出时就可以在timerEvent()函数中获取该定时器的编号来进行相关操作。

编程中更多的是使用QTimer类来实现一个定时器，它提供了更高层次的编程接口，比如可以使用信号和槽，还可以设置只运行一次的定时器。所以在以后的章节中，如果使用定时器，那么一般都是使用的QTimer类。

关于随机数，在Qt中是使用qrand()和qsrand()两个函数实现的。

通过ID使用定时器

使用QTimerEvent的timerId()函数来获取定时器的编号，然后判断是哪一个定时器并分别进行不同的操作。

在构造函数中：

```
id1 = startTimer(1000);              // 开启一个1秒定时器，返回其ID
id2 = startTimer(2000); id3 = startTimer(3000);
```

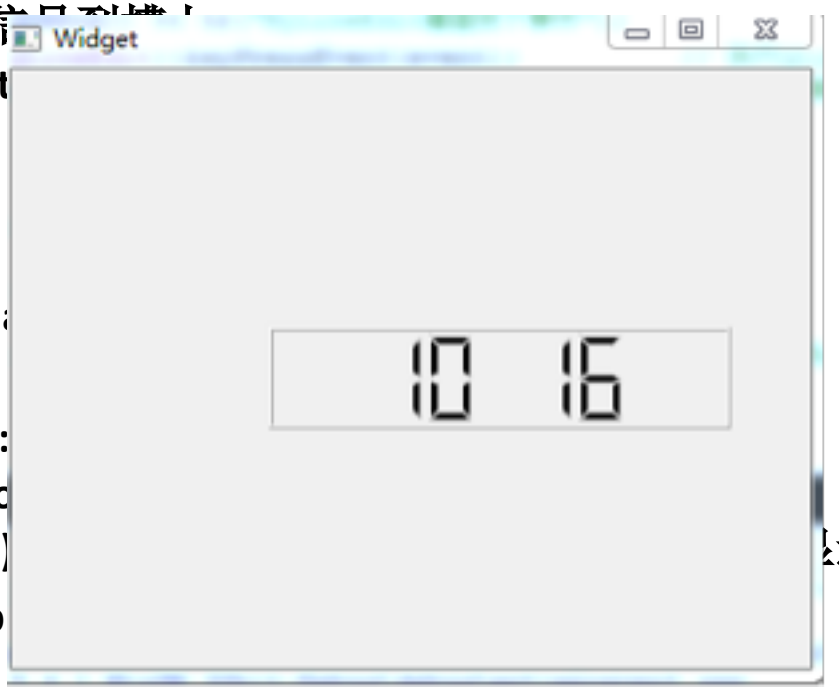
下面是定时器事件函数的定义：

```
void Widget::timerEvent(QTimerEvent *event)
{
    if (event->timerId() == id1) {      // 判断是哪个定时器
        qDebug() << "timer1”; }
    else if (event->timerId() == id2) {
        qDebug() << "timer2”; }
    else {
        qDebug() << "timer3”; }
}
```

通过信号和槽实现定时器

在构造函数中：

```
QTimer *timer = new QTimer(this);        // 创建一个新的定时器
// 关联定时器的溢出信号到槽
connect(timer,SIGNAL(t
timer->start(1000);
```



溢出处理：

```
void Widget::timerUpd
{
   QTime time = QTime:
   QString text = time.to
   if((time.second() % 2)                                    示为空格
   ui->lcdNumber->disp
}
```

这里在构造函数中开启了一个1秒的定时器，当它溢出时就会发射timeout()信号，这时就会执行我们的定时器溢出处理函数。在槽里我们获取了当前的时间，并且将它转换为可以显示的字符串。

随机数

构造函数里添加一行代码：

```
qsrand(QTime(0, 0, 0).secsTo(QTime::currentTime()));
```

然后在timerUpdate()函数里面添加如下代码：

```
int rand = qrand() % 300;          // 产生300以内的正整数
ui->lcdNumber->move(rand, rand);
```

在使用qrand()函数产生随机数之前，一般要使用qsrand()函数为其设置初值，如果不设置初值，那么每次运行程序，qrand()都会产生相同的一组随机数。为了每次运行程序时，都可以产生不同的随机数，我们要使用qsrand()设置一个不同的初值。这里使用了QTime类的secsTo()函数，它表示两个时间点之间所包含的秒数，比如代码中就是指从零点整到当前时间所经过的秒数。当使用qrand()要获取一个范围内的数值时，一般是让它与一个整数取余，比如这里与300取余，就会使所有生成的数值在0-299之间（包含0和299）。

小结

学习完本章，要掌握基本的事件的处理方法，包括重新实现事件处理函数和使用事件过滤器。对于定时器和随机数的知识，它们在实现一些特殊效果以及动画和游戏中会经常使用到，要学会使用。