1. QEvent

当事件产生被发送到对应的窗口之后，窗口并不会直接处理这个事件，而是对这些事件进行细分，然后根据事件的类型再次进行分发（相当于公司接了个项目，对项目进行查分之后分发给各个职能部门，由各个部门进行模块的开发），对应的事件处理器函数得到这个分发的事件之后就开始处理这个事件。

关于窗口事件的分发，对应一个事件分发器，叫做 `event`

```C++
[override virtual protected] bool QWidget::event(QEvent *event);
```

通过事件分发器的函数原型可以得知，关于事件类型的判断是基于参数完成的，这个参数是一个 `QEvent` 类型的对象，下面来看一下这个类中常用的一些API函数:

```C++
void QEvent::accept();
```

- 该函数的作用是让窗口接受传递过来的事件， 事件不会向上层窗口（父窗口）传递 。

```C++
void QEvent::ignore();
```

- 该函数的作用是让窗口忽略传递过来的事件， 事件被传递给父窗口（向上传递） 。

```C++
bool QEvent::isAccepted() const;
void QEvent::setAccepted(bool accepted);
```

- 设置传递过来的事件是被接受还是被忽略
  - `setAccepted(true) == accept()`
  - `setAccepted(false) == ignore()`

```C++
QEvent::Type QEvent::type() const;
```

- 得到传递的窗口的事件的类型，返回值是一个枚举类型，内容很多可以自己查帮助文档，简单的贴个图：

enum QEvent:: Type

This enum type defines the valid event types in Qt. The event types and the specialized classes for each type are as follows:

Constant（事件的类型）	Value	Description（管理这个事件的子事件类）
QEvent::None	0	Not an event.
QEvent::ActionAdded	114	A new action has been added (QActionEvent).
QEvent::ActionChanged	113	An action has been changed (QActionEvent).
QEvent::ActionRemoved	115	An action has been removed (QActionEvent).
QEvent::ActivationChange	99	A widget's top-level window activation state has changed.
QEvent::ApplicationActivate	121	This enum has been deprecated. Use ApplicationStateChange instead.
QEvent::ApplicationActivated	ApplicationActivate	This enum has been deprecated. Use ApplicationStateChange instead.
QEvent::ApplicationDeactivate	122	This enum has been deprecated. Use ApplicationStateChange instead.
QEvent::ApplicationFontChange	36	The default application font has changed.
QEvent::ApplicationLayoutDirectionChange	37	The default application layout direction has changed.
QEvent::ApplicationPaletteChange	38	The default application palette has changed.
QEvent::ApplicationStateChange	214	The state of the application has changed.
QEvent::ApplicationWindowIconChange	35	The application's icon has changed.
QEvent::ChildAdded	68	An object gets a child (QChildEvent).
QEvent::ChildPolished	69	A widget child gets polished (QChildEvent).
QEvent::ChildRemoved	71	An object loses a child (QChildEvent).
QEvent::Clipboard	40	The clipboard contents have changed.

2. 事件分发器

在不需要人为干预的情况下，事件分发器会自主的完成相关事件的分发，下面来还原一下事件分发器的分发流程，以下是这个函数的部分源码展示：

C++

```cpp
bool QWidget::event(QEvent *ev)
{
    switch(ev->type())
    {
    // 鼠标移动
    case QEvent::MouseMove:
        mouseMoveEvent((QMouseEvent*)event);
        break;
    // 鼠标按下
    case QEvent::MouseButtonPress:
        mousePressEvent((QMouseEvent*)event);
        break;
    // 鼠标释放
    case QEvent::MouseButtonRelease:
        mouseReleaseEvent((QMouseEvent*)event);
        break;
    // 鼠标双击
    case QEvent::MouseButtonDblClick:
        mouseDoubleClickEvent((QMouseEvent*)event);
        break;
    // 键盘按键被按下事件
    case QEvent::KeyPress:
        break;
        ...
        ...
        ...
    default:
        break;
    }
}
```

可以直观的看到事件分发器在对事件进行判定之后会调用相关的事件处理器函数，这样事件就被最终处理掉了。

如果我们不想让某些触发的事件进入到当前窗口中，可以在事件分发器中进行拦截，拦截之前先来了解一下事件分发器函数的返回值：

1. 如果传入的事件已被识别并且处理，则需要返回 `true`，否则返回 `false`。如果返回值是 `true`，那么 Qt 会认为这个事件已经处理完毕，不会再将这个事件发送给其它对象，而是会继续处理事件队列中的下一事件。
2. 在`event()`函数中，调用事件对象的 `accept()` 和 `ignore()` 函数是没有作用的，不会影响到事件的传播。

也就是说如果想过滤某个事件，只需要在判断出这个事件之后直接返回 true 就可以了。

下面来举个例子，在窗口中过滤掉鼠标按下的事件:

C++

```cpp
bool MainWindow::event(QEvent *ev)
{
    if(ev->type() == QEvent::MouseButtonPress ||
            ev->type() == QEvent::MouseButtonDblClick)
    {
        // 过滤调用鼠标按下的事件
        return true;
    }
    return QWidget::event(ev);
}
```

这样窗口就再也收不到鼠标的单击和双击事件了，对于这两个事件以外的其他事件是没有任何影响的，因为在重写的事件分发器函数的最后调用了父类的事件分发器函数

C++

```cpp
return QWidget::event(ev);
```

这样就能保证其他事件按照默认的分发流程进行分发，并最终被窗口处理掉。