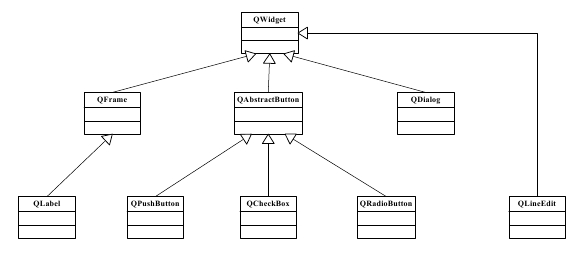
<http://blog.csdn.net/hmsiwtv/article/details/7562015>

QWidget 类代表一般的窗口，其他窗口类都是从 QWidget 类继承出来的。而 QWidget 类则同时继承了 QObject 类 和 QPaintDevice 类，也就是说，窗口类都是 Qt 对象类。这里的 QPaintDevice 类则是所有可绘制的对象的基类。

       常用窗口类的继承关系如图所示：



       构造

       QWidget 类的构造函数如下：

**[plain]** [view plain](http://blog.csdn.net/hmsiwtv/article/details/7562015) [copy](http://blog.csdn.net/hmsiwtv/article/details/7562015)

1. QWidget(QWidget \*parent = 0, Qt::WindowFlags f = 0);

       其中参数 parent 指向父窗口，如果这个参数为 0，则窗口就成为一个顶级窗口

       参数 f 是构造窗口的标志，主要用于控制窗口的类型和外观等，有以下常用值。

       1）Qt::FramelessWindowHint：没有边框的窗口。

       2）Qt::WindowStaysOnTopHint：总是最上面的窗口。

       3）Qt::CustomizeWindowHint：自定义窗口标题栏，以下标志必须与这个标志一起使用才有效，否则窗口将有默认的标题栏。

       4）Qt::WindowTitleHint：显示窗口标题栏。

       5）Qt::WindowSystemMenuHint：显示系统菜单。

       6）Qt::WindowMinimizeButtonHint：显示最小化按钮。

       7）Qt::WindowMaximizeButtonHint：显示最大化按钮。

       8）Qt::WindowMinMaxbuttonHint：显示最小化按钮和最大化按钮。

       9）Qt::WindowCloseButtonHint：显示关闭按钮。

       独立窗口

       窗口构造的时候如果有 Qt::Window 标志，那么它就是一个独立窗口，否则就是一个依附于其他独立窗口的窗口部件。顶级窗口一定是独立窗口，但独立窗口不一定是顶级的，它可以有父窗口，当父窗口被析构时它也会随之被析构。独立窗口一般有自己的外边框和标题栏，可以有移动、改变大小等操作。

       一个窗口是否为独立窗口可用下面的成员函数来判断：

**[plain]** [view plain](http://blog.csdn.net/hmsiwtv/article/details/7562015) [copy](http://blog.csdn.net/hmsiwtv/article/details/7562015)

1. bool isWindow() const;     // 判断是否为独立窗口

       下面这个函数可以得到窗口部件所在的独立窗口。

**[plain]** [view plain](http://blog.csdn.net/hmsiwtv/article/details/7562015) [copy](http://blog.csdn.net/hmsiwtv/article/details/7562015)

1. QWidget \*window() const;      // 所得所在的独立窗口

        当然，如果窗口本身就是独立窗口，那么得到的就是自己。

        而下面这个函数可以得到窗口的父窗口：

**[plain]** [view plain](http://blog.csdn.net/hmsiwtv/article/details/7562015) [copy](http://blog.csdn.net/hmsiwtv/article/details/7562015)

1. QWidget \*parentWidget() const;    // 得到父窗口

        窗口标题

        WindowTitle 属性表示窗口的标题，与之相关的成员函数如下：

**[plain]** [view plain](http://blog.csdn.net/hmsiwtv/article/details/7562015) [copy](http://blog.csdn.net/hmsiwtv/article/details/7562015)

1. QString windowTitle() const;    // 获得窗口标题
2. void setWindowTitle(const QString &text);    // 设置窗口标题为 text

       几何参数

       这里的几何参数指的是窗口的大小和位置。一个窗口有两套几何参数，一套是窗口外边框所占的矩形区域，另一套是窗口客户区所占的矩形区域。所谓窗口客户区就是窗口中去除边框和标题栏用来显示内容的区域。

       这两套几何参数分别由两个 QRect 型的属性代表，相关的成员函数如下：

**[plain]** [view plain](http://blog.csdn.net/hmsiwtv/article/details/7562015) [copy](http://blog.csdn.net/hmsiwtv/article/details/7562015)

1. const QRect &geometry() const;                 // 获取客户区几何参数
2. void setGeometry(int x, int y, int w, int h);    // 设置客户取几何参数
3. void setGeometry(const QRect &rect);         // 设置客户区几何参数
4. QRect frameGeometry() const;                  // 获取外边框几何参数

       这里虽然没有直接设置外边框几何参数的函数，但客户区几何参数变化之后，外边框的几何参数也会随之变化。设置几何参数可能会使窗口的位置及大小发生变化，这时会发送窗口移动事件 QMoveEvent，如果大小有变化，还会发送窗口改变大小事件 QResizeEvent，事件的处理函数分别是 moveEvent 和 resizeEvent。注意这里的坐标都是相对于父窗口的，因此移动一个窗口并不导致它的所有部件都接收到移动事件。

注意：不要在 moveEvent 或 resizeEvent 两个事件处理函数中设置几何参数，否则将导致无限循环。

       窗口的几何参数也可以由用户的操作改变，这时也会发送相应的事件。

       为了方便使用，与几何参数相关的成员函数还有以下这些：

**[plain]** [view plain](http://blog.csdn.net/hmsiwtv/article/details/7562015) [copy](http://blog.csdn.net/hmsiwtv/article/details/7562015)

1. QPoint pos() const;     // 获得窗口左上角的坐标(外边框几何参数)
2. QSize size() const;      // 窗口大小 （客户区几何参数）
3. int x() const;                  // 窗口左上角横坐标 （外边框几何参数）
4. int y() const;                  // 窗口左上角纵坐标 （外边框几何参数）
5. int height() const;        // 窗口高度 （客户区几何参数）
6. int width() const;          // 窗口宽度 （客户区几何参数）

       可以看出，坐标全部是外边框几何参数，而大小全部是客户区几何参数。要获得外边框的大小需要用下面这个成员函数：

**[plain]** [view plain](http://blog.csdn.net/hmsiwtv/article/details/7562015) [copy](http://blog.csdn.net/hmsiwtv/article/details/7562015)

1. QSize frameSize() const;    // 窗口大小 （外边框几何参数）

       改变这些属性可以用下面这些成员函数：

**[plain]** [view plain](http://blog.csdn.net/hmsiwtv/article/details/7562015) [copy](http://blog.csdn.net/hmsiwtv/article/details/7562015)

1. void move(int x, int y);    // 将窗口左上角移动到坐标（x,  y）处；
2. void move(const QPoint &pos);     // 将窗口左上角移动到 pos 处；
3. void resize(int w, int h);     // 将窗口的宽度改为 w， 高度改为 h
4. void resize(const QSize &size);     // 将窗口的大小改为  size

       同样，这里 move 函数用的是外边框几何参数，而 resize 函数用的是客户区几何参数。

       还有一个属性比较特殊，相关的成员函数如下：

**[plain]** [view plain](http://blog.csdn.net/hmsiwtv/article/details/7562015) [copy](http://blog.csdn.net/hmsiwtv/article/details/7562015)

1. QRect rect() const;     // 获取窗口区域

       它获得的坐标都是相对于窗口自己的客户区的，也就是说横纵坐标永远为 0。

注意： 对于一个窗口部件来说，它的两套几何参数是一致的。

       可见性与隐藏

       可见性指的是窗口是否显示在屏幕上的属性。被其他窗口暂时遮挡住的窗口也属于可见的。可见性由窗口的 visible 属性表示，与之相关的成员函数如下：

**[plain]** [view plain](http://blog.csdn.net/hmsiwtv/article/details/7562015) [copy](http://blog.csdn.net/hmsiwtv/article/details/7562015)

1. bool isVisible() const;    // 判断窗口是否可见
2. bool isHidden() const;   // 判断窗口是否隐藏
3. virtual void setVisible(bool visible);   // 设置窗口是否隐藏
4. void setHidden(bool hidden);    // 等价于 setvisible(!hidedn);

        visible 属性为 true 时表示窗口可见，为 false 时表示窗口不可见。这里要注意的是，setVisible 函数实际上设置的是窗口是否隐藏，而不是可见性。可见性与隐藏有如下关系。

        1）隐藏的窗口一定是不可见的。

        2）非隐藏的窗口在它的父窗口可见的情况下也是可见的。

        3）非隐藏的顶级窗口是可见的。

        setVisible 和 setHidden 同时也是槽，它们一般并不直接使用，而是使用以下几个槽：

**[plain]** [view plain](http://blog.csdn.net/hmsiwtv/article/details/7562015) [copy](http://blog.csdn.net/hmsiwtv/article/details/7562015)

1. void show();     // 显示窗口，等价于 setVisible(true);
2. void hide();       // 隐藏窗口，等价于 setHidden(true);

        当窗口显示时，将发送 QShowEvent 事件；当窗口隐藏时，将发送 QHideEvent 事件。事件的处理函数分别是 showEvent 和 hideEvent。

        窗口状态

        独立窗口有正常、全屏、最大化、最小化几种状态，与之相关的成员函数如下：

**[plain]** [view plain](http://blog.csdn.net/hmsiwtv/article/details/7562015) [copy](http://blog.csdn.net/hmsiwtv/article/details/7562015)

1. bool isMinimized() const;     // 判断窗口是否为最小化
2. bool isMaximized() const;    // 判断窗口是否为最大化
3. bool isFullScreen() const;   // 判断窗口是否为全屏
4. void showMinimized();         // 以最小化方式显示窗口，这是一个槽
5. void showMaximized();        // 以最大化方式显示窗口，这是一个槽
6. void showFullScreen();        // 以全屏方式显示窗口，这是一个槽
7. void showNormal();              // 以正常方式显示窗口，这是一个槽

        注意后 4 个函数同时也是槽。全屏方式与最大化的区别在于：全屏方式下窗口的边框和标题栏消失，客户区占据整个屏幕。窗口的各种状态仅对独立窗口有效，对窗口部件来说没有意义。

        另外还有一个 windowState 属性和窗口状态有关，相关的成员函数如下：

**[plain]** [view plain](http://blog.csdn.net/hmsiwtv/article/details/7562015) [copy](http://blog.csdn.net/hmsiwtv/article/details/7562015)

1. Qt::WindowStates windowState() const;                         // 获取窗口状态
2. void setWindowState(Qt::WindowStates windowState);      // 设置窗口状态

        这里的 Qt::WindowStates 类型有以下几个取值。

        1）Qt::WindowNoState：无标志，正常状态。

        2）Qt::WindowMinimized：最小化状态。

        3）Qt::WindowMaxmized：最大化状态。

        4）Qt::WindowFullScreen：全屏状态。

        5）Qt::WindowActive：激活状态。

        这里取值可以用 “按位或” 的方式组合起来使用。

        需要注意的是，调用 setWindowState 函数将使窗口变为隐藏状态。

        使能

        处于使能状态的窗口才能处理键盘和鼠标等输入事件，反之，处于禁用状态的窗口不能处理这些事件。窗口是否处于使能状态由属性 enabled 表示，相关成员函数如下：

**[plain]** [view plain](http://blog.csdn.net/hmsiwtv/article/details/7562015) [copy](http://blog.csdn.net/hmsiwtv/article/details/7562015)

1. bool isEnabled() const;     // 获得窗口的使能装态
2. void setEnabled(bool enable);  // 设置窗口的使能状态，这是一个槽
3. void setDisabled(bool disabled);     // 等价于 setEnabled(!disable)，这是一个槽

        其中两个设置属性的函数同时也是槽。窗口的使能状态也可能影响外观，比如处于禁用状态的按钮文本本身为灰色。

        使能状态和窗口的可见性有相似的逻辑：禁用一个窗口同 时会使它的所有子窗口成为禁用状态。

        激活状态

        当有多个独立窗口同时存在时，只有一个窗口能够处于激活状态。系统产生的键盘、鼠标等输入事件将被发送给处于激活状态的窗口。一般来说，这样的窗口会被提升到堆叠层次的最上面，除非其他窗口有总在最上面的属性。与激活状态相关的成员函数如下：

**[plain]** [view plain](http://blog.csdn.net/hmsiwtv/article/details/7562015) [copy](http://blog.csdn.net/hmsiwtv/article/details/7562015)

1. bool isActiveWindow() const;   // 判断窗口所在的独立窗口是否激活
2. void activateWindow();    //  设置窗口所在的独立窗口为激活状态

注意：这里操作的其实不是窗口本身，而是窗口所在的独立窗口，因为窗口部件时没有激活状态的概念的。

        焦点

        焦点用来控制同一个独立窗口内哪一个部件可以接受键盘事件，同一时刻只能有一个部件获得焦点。与焦点有关的成员函数如下：

**[plain]** [view plain](http://blog.csdn.net/hmsiwtv/article/details/7562015) [copy](http://blog.csdn.net/hmsiwtv/article/details/7562015)

1. bool hasFocus() const;                     // 判断窗口是否获得焦点
2. void setFocus();                             // 使窗口获得焦点，这是一个槽
3. void clearFocus();                          // 使窗口失去焦点
4. QWidget \*focusWidget() const;        // 得到窗口内获得焦点的子窗口

        setFocus 函数同时又是一个槽。窗口部件得到焦点以后，别忘了还需要它所在的独立窗口处于激活状态才能得到键盘事件。

        一个窗口获得焦点，同时意味着另一个窗口失去焦点。当窗口获得或失去焦点时，将发送 QFocusEvent 事件，它有两个处理函数：forceInEvent 和 focusOutEvent，分别对应获得焦点和失去焦点。

        值得一提的是 editFocus 属性，这是一个专门用于嵌入式系统的属性。因为嵌入式系统通常键盘较小，没有专门用于切换焦点的 Tab 键，所以上下方向键被用来切换焦点。如果一个窗口部件设置 editFocus 属性为 true，则上下方向键就不再用于切换焦点，而是发送给这个窗口。与这个属性相关的成员函数如下:

**[plain]** [view plain](http://blog.csdn.net/hmsiwtv/article/details/7562015) [copy](http://blog.csdn.net/hmsiwtv/article/details/7562015)

1. bool hasEditfocus() const;     // 判断窗口是否有 editFocus 属性
2. void QWidget::setEditFocus(bool enable);     // 设置窗口的 editFocus 属性

        捕获键盘和鼠标事件

        窗口部件即使获得焦点，也不一定能获得按键事件，因为其他窗口可能会捕获键盘事件。捕获了键盘事件的窗口将得到所有键盘事件，而其他窗口将完全得到不到键盘事件，直到捕获了键盘事件的窗口释放键盘事件。与键盘事件捕获相关的成员函数如下：

**[plain]** [view plain](http://blog.csdn.net/hmsiwtv/article/details/7562015) [copy](http://blog.csdn.net/hmsiwtv/article/details/7562015)

1. void grabKeyboard();           // 捕获键盘事件
2. void releaseKeyboard();     // 释放键盘事件

        类似的还有鼠标事件的捕获和释放，其成员函数如下：

**[plain]** [view plain](http://blog.csdn.net/hmsiwtv/article/details/7562015) [copy](http://blog.csdn.net/hmsiwtv/article/details/7562015)

1. void grabMouse();          // 捕获鼠标事件
2. void releaseMouse();    // 释放鼠标事件

        对键盘事件和鼠标事件的捕获是相互独立的。这里要注意两点：一是如果有另外一个窗口进行了捕获操作，则当前处于捕获状态的窗口将失去对事件的捕获；二是只有可见的窗口才能进行输入事件捕获。

        以下的成员函数能够得到应用程序中正在捕获键盘或鼠标事件的窗口：

**[plain]** [view plain](http://blog.csdn.net/hmsiwtv/article/details/7562015) [copy](http://blog.csdn.net/hmsiwtv/article/details/7562015)

1. QWidget \*keyboardGrabber();      // 得到正在捕获键盘事件的窗口
2. QWidget \*mouseGrabber();        // 得到正在捕获鼠标事件的窗口

        这两个函数是静态函数。

        布局

        属性 layout 代表窗口的顶级布局，相关的成员函数如下：

**[plain]** [view plain](http://blog.csdn.net/hmsiwtv/article/details/7562015) [copy](http://blog.csdn.net/hmsiwtv/article/details/7562015)

1. QLayout \*layout() const;                   // 获得顶级布局
2. void setLayout(QLayout \*layout);        // 设置顶级布局

        字体

        font 属性表示所用的字体，相关的成员函数如下：

**[plain]** [view plain](http://blog.csdn.net/hmsiwtv/article/details/7562015) [copy](http://blog.csdn.net/hmsiwtv/article/details/7562015)

1. const QFont &font() const;         // 获得字体
2. void setFont(const QFont &);    // 设置字体

       如果没有为窗口设置字体，则窗口自动使用父窗口的字体，顶级窗口则使用应用程序的默认字体。

       信号

       当窗口要被析构时会发射以下信号：

**[plain]** [view plain](http://blog.csdn.net/hmsiwtv/article/details/7562015) [copy](http://blog.csdn.net/hmsiwtv/article/details/7562015)

1. void destoryed(QObject \*obj = 0);

        这是一个从 QOjbect 类继承过来的信号。QObject 对象析构时，先发射这个信号，然后才析构它的所有子对象。

        槽

        在前面的介绍中已经提及了 QWidget 类的许多槽，这里将介绍其他常用的槽。

        下面的槽可以关闭窗口：

**[plain]** [view plain](http://blog.csdn.net/hmsiwtv/article/details/7562015) [copy](http://blog.csdn.net/hmsiwtv/article/details/7562015)

1. bool close();

        当这个槽被调用时，首先向这个窗口发送一个关闭事件，如果事件被接受，则窗口隐藏，如果被拒绝，则什么也不做。如果窗口设置了 Qt::WA\_QuitOnClose 属性，则窗口对象会被析构，大多数类型的窗口都默认设置了这个属性。

         这个槽的返回值表示关闭事件是否被接受，也就是窗口是否真的被关闭了。

         下面的槽可以提升或降低窗口所在的堆叠层次：

**[plain]** [view plain](http://blog.csdn.net/hmsiwtv/article/details/7562015) [copy](http://blog.csdn.net/hmsiwtv/article/details/7562015)

1. void lower();         // 降低窗口到最下面
2. void raise();        // 提升窗口到最上面

         事件

         QWidget 类能够处理类型丰富的事件，这里将介绍一些常用的事件处理函数。

         窗口事件：

**[plain]** [view plain](http://blog.csdn.net/hmsiwtv/article/details/7562015) [copy](http://blog.csdn.net/hmsiwtv/article/details/7562015)

1. virtual void closeEvent(QCloseEvent \*event);    // 关闭
2. virtual void showEvent(QShowEvent \*event);    //  显示
3. virtual void hideEvent(QHideEvent \*event);        // 隐藏
4. virtual void moveEvent(QMoveEvent \*evnet);     // 移动
5. virtual void resizeEvent(QResizeEvent \*event);  // 改变大小

         这里通过 QMoveEvent 类的以下成员函数可以获得窗口的旧坐标和新坐标：

**[plain]** [view plain](http://blog.csdn.net/hmsiwtv/article/details/7562015) [copy](http://blog.csdn.net/hmsiwtv/article/details/7562015)

1. const QPoint &oldPos() const;     // 旧坐标
2. const QPoint &newPos() constl   // 新坐标

         通过 QResizeEvent 类的以下成员函数可以获得窗口的旧大小和新大小：

**[plain]** [view plain](http://blog.csdn.net/hmsiwtv/article/details/7562015) [copy](http://blog.csdn.net/hmsiwtv/article/details/7562015)

1. const QSize &oldSize() const;     // 旧大小
2. const QSize &newSize() const;   // 新大小

         键盘事件：

**[plain]** [view plain](http://blog.csdn.net/hmsiwtv/article/details/7562015) [copy](http://blog.csdn.net/hmsiwtv/article/details/7562015)

1. virtual void keyPressEvent(QKeyEvent \*event);   // 键按下
2. virtual void keyReleaseEvent(QKeyEvent \*event);  // 键松开

         这里通过  QKeyEvent 类的成员函数可以获得关于按键的一些信息，如：

**[plain]** [view plain](http://blog.csdn.net/hmsiwtv/article/details/7562015) [copy](http://blog.csdn.net/hmsiwtv/article/details/7562015)

1. int key() const;     // 得到键值

          鼠标事件：

**[plain]** [view plain](http://blog.csdn.net/hmsiwtv/article/details/7562015) [copy](http://blog.csdn.net/hmsiwtv/article/details/7562015)

1. virtual void mousePressEvent(QMouseEvent \*event);                               // 鼠标键按下
2. virtual void mouseReleaseEvent(QMouseEvent \*event);                            // 鼠标键松开
3. virtual void mouseDoubleCllckEvent(QMouseEvent \*event);                       // 鼠标键双击
4. virtual void mouseMoveEvent(QMouseEvent \*event);                               // 鼠标移动
5. virtual void enterEvent(QEvent \*event);                                                // 鼠标进入窗口
6. virtual void leaveEvent(QEvent \*event);                                                // 鼠标离开窗口
7. virtual void wheelEvent(QWheelEvent \*event);                                       // 鼠标滚轮移动

         这里通过 QMouseEvent 事件的成员函数可获得关于鼠标的信息，如：

**[plain]** [view plain](http://blog.csdn.net/hmsiwtv/article/details/7562015) [copy](http://blog.csdn.net/hmsiwtv/article/details/7562015)

1. const QPoint &pos() const;                                                             // 得到鼠标坐标（相对于接收事件的窗口）
2. int x()  const;                                                                              // 得到鼠标横坐标（相对于接收事件的窗口）
3. int y() const;                                                                               // 得到鼠标纵坐标（相对于接收事件的窗口）
4. const QPoint &globalPos() const;                                                     // 得到鼠标坐标（全局坐标）
5. int globalX() const;                                                                        // 得到鼠标横坐标 （全局坐标）
6. int globalY() const;                                                                        // 得到鼠标纵坐标 （全局坐标）
7. Qt::MouseButton button() const;                                                       // 得到引起事件的鼠标键
8. Qt::MouseButtons buttons() const;                                                    // 得到事件发生时的鼠标键状态

         其中，Qt::MouseButton 是一个枚举类型，有以下常用取值。

         1）Qt::NoButton：无键。

         2）Qt::LeftButton：左键。

         3）Qt::RightButton：右键。

         4）Qt::MidButton：中键。

         注意，对于鼠标移动事件 QMouseEvent 和 button 函数总是返回 Qt::NoButton，而 buttons 函数返回值则是 Qt::MouseButton 类型的 “按位或” 组合，它能反映事件发生时鼠标键的按下状态。

          QWheelEvent 类代表滚轮事件，它有一套与 QMountEvent 类几乎相同的成员函数，但少一个 button 函数，多以下两个函数：

**[plain]** [view plain](http://blog.csdn.net/hmsiwtv/article/details/7562015) [copy](http://blog.csdn.net/hmsiwtv/article/details/7562015)

1. int delta() const;    // 获得滚轮转动的角度
2. Qt::Orientation orientationI() const;    // 获得滚轮转动的方向

          其中 Qt::Orientation 是一个枚举类型，它有以下取值。

          1）Qt::Horizontal：横向。

          2）Qt::Vertical：纵向。

          焦点事件：

**[plain]** [view plain](http://blog.csdn.net/hmsiwtv/article/details/7562015) [copy](http://blog.csdn.net/hmsiwtv/article/details/7562015)

1. virtual void focusInEvent(QFocusEvent \*event);    // 获得焦点
2. virtual void focusOutEvent(QFocusEvent \*event);  // 时取焦点

          这些事件处理函数都没有返回值，因此如果要接受或拒绝和一个事件要调用 QEvent 类的成员函数，如：

**[plain]** [view plain](http://blog.csdn.net/hmsiwtv/article/details/7562015) [copy](http://blog.csdn.net/hmsiwtv/article/details/7562015)

1. event->accept();    // 接受事件
2. event->ignore();    // 拒绝事件

          事件被拒绝后的结果视具体情况而定，比如关闭事件被拒绝后，窗口将不会被关闭，而键盘、鼠标等输入事件被拒绝后会向上传播到父窗口