

QWidget 类

QWidget 是所有用户界面的基类

Header:	#include <QWidget>
qmake:	QT += widgets
Inherits:	QObject and QPaintDevice
Inherited By:	QAbstractButton , QAbstractSlider , QAbstractSpinBox , QCalendarWidget , QComboBox , QDesktopWidget , QDialog , QDialogButtonBox , QDockWidget , QFocusFrame , QFrame , QGroupBox , QKeySequenceEdit , QLineEdit , QMacCocoaViewContainer , QMacNativeWidget , QMainWindow , QMdiSubWindow , QMenu , QMenuBar , QOpenGLWidget , QProgressBar , QRubberBand , QSizeGrip , QSplashScreen , QSplitterHandle , QStatusBar , QTabBar , QTabWidget , QToolBar , and QWizardPage

- [List of all members, including inherited members](#)
- [Obsolete members](#)

Public Types|公共类型

enum	RenderFlag { DrawWindowBackground, DrawChildren, IgnoreMask }
flags	RenderFlags

Properties|函数

<ul style="list-style-type: none">• acceptDrops : bool• accessibleDescription : QString• accessibleName : QString• autoFillBackground : bool• baseSize : QSize• childrenRect : const QRect• childrenRegion : const QRegion	<ul style="list-style-type: none">• minimumWidth : int• modal : const bool• mouseTracking : bool• normalGeometry : const QRect• palette : QPalette• pos : QPoint• rect : const QRect
--	--

<ul style="list-style-type: none"> • <u>contextMenuPolicy</u> : Qt::ContextMenuPolicy • <u>cursor</u> : QCursor • <u>enabled</u> : bool • <u>focus</u> : const bool • <u>focusPolicy</u> : Qt::FocusPolicy • <u>font</u> : QFont • <u>frameGeometry</u> : const QRect • <u>frameSize</u> : const QSize • <u>fullScreen</u> : const bool • <u>geometry</u> : QRect • <u>height</u> : const int • <u>inputMethodHints</u> : Qt::InputMethodHints • <u>isActiveWindow</u> : const bool • <u>layoutDirection</u> : Qt::LayoutDirection • <u>locale</u> : QLocale • <u>maximized</u> : const bool • <u>maximumHeight</u> : int • <u>maximumSize</u> : QSize • <u>maximumWidth</u> : int • <u>minimized</u> : const bool • <u>minimumHeight</u> : int • <u>minimumSize</u> : QSize • <u>minimumSizeHint</u> : const QSize 	<ul style="list-style-type: none"> • <u>size</u> : QSize • <u>sizeHint</u> : const QSize • <u>sizeIncrement</u> : QSize • <u>sizePolicy</u> : QSizePolicy • <u>statusTip</u> : QString • <u>styleSheet</u> : QString • <u>tabletTracking</u> : bool • <u>toolTip</u> : QString • <u>toolTipDuration</u> : int • <u>updatesEnabled</u> : bool • <u>visible</u> : bool • <u>whatsThis</u> : QString • <u>width</u> : const int • <u>windowFilePath</u> : QString • <u>windowFlags</u> : Qt::WindowFlags • <u>windowIcon</u> : QIcon • <u>windowModality</u> : Qt::WindowModality • <u>windowModified</u> : bool • <u>windowOpacity</u> : double • <u>windowTitle</u> : QString • <u>x</u> : const int • <u>y</u> : const int
--	--

- 1 property inherited from [QObject](#)

Public Functions|公共函数

	<u>QWidget</u> (QWidget *parent = nullptr, Qt::WindowFlags f = ...)
virtual	<u>~QWidget</u> ()
bool	<u>acceptDrops</u> () const

QString	<u>accessibleDescription()</u> const
QString	<u>accessibleName()</u> const
QList<QAction *>	<u>actions()</u> const
void	<u>activateWindow()</u>
void	<u>addAction()</u> (QAction * <i>action</i>)
void	<u>addActions()</u> (QList<QAction *> <i>actions</i>)
void	<u>adjustSize()</u>
bool	<u>autoFillBackground()</u> const
QPalette::ColorRole	<u>backgroundRole()</u> const
QBackingStore *	<u>backingStore()</u> const
QSize	<u>baseSize()</u> const
QWidget *	<u>childAt()</u> (int <i>x</i> , int <i>y</i>) const
QWidget *	<u>childAt()</u> (const QPoint & <i>p</i>) const
QRect	<u>childrenRect()</u> const
QRegion	<u>childrenRegion()</u> const
void	<u>clearFocus()</u>
void	<u>clearMask()</u>
QMargins	<u>contentsMargins()</u> const
QRect	<u>contentsRect()</u> const
Qt::ContextMenuPolicy	<u>contextMenuPolicy()</u> const

QCursor	<u>cursor()</u> const
WId	<u>effectiveWinId()</u> const
void	<u>ensurePolished()</u> const
Qt::FocusPolicy	<u>focusPolicy()</u> const
QWidget *	<u>focusProxy()</u> const
QWidget *	<u>focusWidget()</u> const
const QFont &	<u>font()</u> const
QFontInfo	<u>fontInfo()</u> const
QFontMetrics	<u>fontMetrics()</u> const
QPalette::ColorRole	<u>foregroundRole()</u> const
QRect	<u>frameGeometry()</u> const
QSize	<u>frameSize()</u> const
const QRect &	<u>geometry()</u> const
void	<u>getContentsMargins()</u> (int * <i>left</i> , int * <i>top</i> , int * <i>right</i> , int * <i>bottom</i>) const
QPixmap	<u>grab()</u> (const QRect & <i>rectangle</i> = QRect(QPoint(0, 0), QSize(-1, -1)))
void	<u>grabGesture()</u> (Qt::GestureType <i>gesture</i> , Qt::GestureFlags <i>flags</i> = ...)
void	<u>grabKeyboard()</u>
void	<u>grabMouse()</u>
void	<u>grabMouse()</u> (const QCursor & <i>cursor</i>)
int	<u>grabShortcut()</u> (const QKeySequence & <i>key</i> , Qt::ShortcutContext <i>context</i> = Qt::WindowShortcut)

QGraphicsEffect *	<u>graphicsEffect()</u> const
QGraphicsProxyWidget *	<u>graphicsProxyWidget()</u> const
bool	<u>hasEditFocus()</u> const
bool	<u>hasFocus()</u> const
virtual bool	<u>hasHeightForWidth()</u> const
bool	<u>hasMouseTracking()</u> const
bool	<u>hasTabletTracking()</u> const
int	<u>height()</u> const
virtual int	<u>heightForWidth(int w)</u> const
Qt::InputMethodHints	<u>inputMethodHints()</u> const
virtual QVariant	<u>inputMethodQuery</u> (Qt::InputMethodQuery <i>query</i>) const
void	<u>insertAction</u> (QAction * <i>before</i> , QAction * <i>action</i>)
void	<u>insertActions</u> (QAction * <i>before</i> , QList<QAction *> <i>actions</i>)
bool	<u>isActiveWindow()</u> const
bool	<u>isAncestorOf</u> (const QWidget * <i>child</i>) const
bool	<u>isEnabled()</u> const
bool	<u>isEnabledTo</u> (const QWidget * <i>ancestor</i>) const
bool	<u>isFullScreen()</u> const
bool	<u>isHidden()</u> const

bool	<u>isMaximized()</u> const
bool	<u>isMinimized()</u> const
bool	<u>isModal()</u> const
bool	<u>isVisible()</u> const
bool	<u>isVisibleTo()</u> (const QWidget * <i>ancestor</i>) const
bool	<u>isWindow()</u> const
bool	<u>isWindowModified()</u> const
QLayout *	<u>layout()</u> const
Qt::LayoutDirection	<u>layoutDirection()</u> const
QLocale	<u>locale()</u> const
QPoint	<u>mapFrom()</u> (const QWidget * <i>parent</i> , const QPoint & <i>pos</i>) const
QPoint	<u>mapFromGlobal()</u> (const QPoint & <i>pos</i>) const
QPoint	<u>mapFromParent()</u> (const QPoint & <i>pos</i>) const
QPoint	<u>mapTo()</u> (const QWidget * <i>parent</i> , const QPoint & <i>pos</i>) const
QPoint	<u>mapToGlobal()</u> (const QPoint & <i>pos</i>) const
QPoint	<u>mapToParent()</u> (const QPoint & <i>pos</i>) const
QRegion	<u>mask()</u> const
int	<u>maximumHeight()</u> const
QSize	<u>maximumSize()</u> const
int	<u>maximumWidth()</u> const

int	<u>minimumHeight()</u> const
QSize	<u>minimumSize()</u> const
virtual QSize	<u>minimumSizeHint()</u> const
int	<u>minimumWidth()</u> const
void	<u>move()</u> (const QPoint &)
void	<u>move()</u> (int x, int y)
QWidget *	<u>nativeParentWidget()</u> const
QWidget *	<u>nextInFocusChain()</u> const
QRect	<u>normalGeometry()</u> const
void	<u>overrideWindowFlags()</u> (Qt::WindowFlags flags)
const QPalette &	<u>palette()</u> const
QWidget *	<u>parentWidget()</u> const
QPoint	<u>pos()</u> const
QWidget *	<u>previousInFocusChain()</u> const
QRect	<u>rect()</u> const
void	<u>releaseKeyboard()</u>
void	<u>releaseMouse()</u>
void	<u>releaseShortcut()</u> (int id)
void	<u>removeAction()</u> (QAction *action)

void	<u>render</u> (QPaintDevice * <i>target</i> , const QPoint & <i>targetOffset</i> = QPoint(), const QRegion & <i>sourceRegion</i> = QRegion(), QWidget::RenderFlags <i>renderFlags</i> = ...)
void	<u>render</u> (QPainter * <i>painter</i> , const QPoint & <i>targetOffset</i> = QPoint(), const QRegion & <i>sourceRegion</i> = QRegion(), QWidget::RenderFlags <i>renderFlags</i> = ...)
void	<u>repaint</u> (int <i>x</i> , int <i>y</i> , int <i>w</i> , int <i>h</i>)
void	<u>repaint</u> (const QRect & <i>rect</i>)
void	<u>repaint</u> (const QRegion & <i>rgn</i>)
void	<u>resize</u> (const QSize & <i>ℓ</i>)
void	<u>resize</u> (int <i>w</i> , int <i>h</i>)
bool	<u>restoreGeometry</u> (const QByteArray & <i>geometry</i>)
QByteArray	<u>saveGeometry</u> () const
void	<u>scroll</u> (int <i>dx</i> , int <i>dy</i>)
void	<u>scroll</u> (int <i>dx</i> , int <i>dy</i> , const QRect & <i>r</i>)
void	<u>setAcceptDrops</u> (bool <i>on</i>)
void	<u>setAccessibleDescription</u> (const QString & <i>description</i>)
void	<u>setAccessibleName</u> (const QString & <i>name</i>)
void	<u>setAttribute</u> (Qt::WidgetAttribute <i>attribute</i> , bool <i>on</i> = true)
void	<u>setAutoFillBackground</u> (bool <i>enabled</i>)
void	<u>setBackgroundRole</u> (QPalette::ColorRole <i>role</i>)
void	<u>setBaseSize</u> (const QSize & <i>ℓ</i>)
void	<u>setBaseSize</u> (int <i>basew</i> , int <i>baseh</i>)

void	<u>setContentsMargins</u> (int <i>left</i> , int <i>top</i> , int <i>right</i> , int <i>bottom</i>)
void	<u>setContentsMargins</u> (const QMargins & <i>margins</i>)
void	<u>setContextMenuPolicy</u> (Qt::ContextMenuPolicy <i>policy</i>)
void	<u>setCursor</u> (const QCursor &)
void	<u>setEditFocus</u> (bool <i>enable</i>)
void	<u>setFixedHeight</u> (int <i>h</i>)
void	<u>setFixedSize</u> (const QSize & <i>s</i>)
void	<u>setFixedSize</u> (int <i>w</i> , int <i>h</i>)
void	<u>setFixedWidth</u> (int <i>w</i>)
void	<u>setFocus</u> (Qt::FocusReason <i>reason</i>)
void	<u>setFocusPolicy</u> (Qt::FocusPolicy <i>policy</i>)
void	<u>setFocusProxy</u> (QWidget * <i>w</i>)
void	<u>setFont</u> (const QFont &)
void	<u>setForegroundRole</u> (QPalette::ColorRole <i>role</i>)
void	<u>setGeometry</u> (const QRect &)
void	<u>setGeometry</u> (int <i>x</i> , int <i>y</i> , int <i>w</i> , int <i>h</i>)
void	<u>setGraphicsEffect</u> (QGraphicsEffect * <i>effect</i>)
void	<u>setInputMethodHints</u> (Qt::InputMethodHints <i>hints</i>)
void	<u>setLayout</u> (QLayout * <i>layout</i>)
void	<u>setLayoutDirection</u> (Qt::LayoutDirection <i>direction</i>)

void	<u>setLocale</u> (const QLocale & <i>locale</i>)
void	<u>setMask</u> (const QBitmap & <i>bitmap</i>)
void	<u>setMask</u> (const QRegion & <i>region</i>)
void	<u>setMaximumHeight</u> (int <i>maxh</i>)
void	<u>setMaximumSize</u> (const QSize &)
void	<u>setMaximumSize</u> (int <i>maxw</i> , int <i>maxh</i>)
void	<u>setMaximumWidth</u> (int <i>maxw</i>)
void	<u>setMinimumHeight</u> (int <i>minh</i>)
void	<u>setMinimumSize</u> (const QSize &)
void	<u>setMinimumSize</u> (int <i>minw</i> , int <i>minh</i>)
void	<u>setMinimumWidth</u> (int <i>minw</i>)
void	<u>setMouseTracking</u> (bool <i>enable</i>)
void	<u>setPalette</u> (const QPalette &)
void	<u>setParent</u> (QWidget * <i>parent</i>)
void	<u>setParent</u> (QWidget * <i>parent</i> , Qt::WindowFlags <i>f</i>)
void	<u>setShortcutAutoRepeat</u> (int <i>id</i> , bool <i>enable</i> = true)
void	<u>setShortcutEnabled</u> (int <i>id</i> , bool <i>enable</i> = true)
void	<u>setSizeIncrement</u> (const QSize &)
void	<u>setSizeIncrement</u> (int <i>w</i> , int <i>h</i>)
void	<u>setSizePolicy</u> (QSizePolicy)

void	<u>setSizePolicy</u> (QSizePolicy::Policy <i>horizontal</i> , QSizePolicy::Policy <i>vertical</i>)
void	<u>setStatusTip</u> (const QString &)
void	<u>setStyle</u> (QStyle * <i>style</i>)
void	<u>setTabletTracking</u> (bool <i>enable</i>)
void	<u>setToolTip</u> (const QString &)
void	<u>setToolTipDuration</u> (int <i>msec</i>)
void	<u>setUpdatesEnabled</u> (bool <i>enable</i>)
void	<u>setWhatsThis</u> (const QString &)
void	<u>setWindowFilePath</u> (const QString & <i>filePath</i>)
void	<u>setWindowFlag</u> (Qt::WindowType <i>flag</i> , bool <i>on</i> = true)
void	<u>setWindowFlags</u> (Qt::WindowFlags <i>type</i>)
void	<u>setWindowIcon</u> (const QIcon & <i>icon</i>)
void	<u>setWindowModality</u> (Qt::WindowModality <i>windowModality</i>)
void	<u>setWindowOpacity</u> (qreal <i>level</i>)
void	<u>setWindowRole</u> (const QString & <i>role</i>)
void	<u>setWindowState</u> (Qt::WindowStates <i>windowState</i>)
void	<u>setupUi</u> (QWidget * <i>widget</i>)
QSize	<u>size</u> () const
virtual QSize	<u>sizeHint</u> () const
QSize	<u>sizeIncrement</u> () const

QSizePolicy	<u>sizePolicy()</u> const
void	<u>stackUnder</u> (QWidget * <i>w</i>)
QString	<u>statusTip()</u> const
QStyle *	<u>style()</u> const
QString	<u>styleSheet()</u> const
bool	<u>testAttribute</u> (Qt::WidgetAttribute <i>attribute</i>) const
QString	<u>toolTip()</u> const
int	<u>toolTipDuration()</u> const
bool	<u>underMouse()</u> const
void	<u>ungrabGesture</u> (Qt::GestureType <i>gesture</i>)
void	<u>unsetCursor()</u>
void	<u>unsetLayoutDirection()</u>
void	<u>unsetLocale()</u>
void	<u>update</u> (int <i>x</i> , int <i>y</i> , int <i>w</i> , int <i>h</i>)
void	<u>update</u> (const QRect & <i>rect</i>)
void	<u>update</u> (const QRegion & <i>rgn</i>)
void	<u>updateGeometry()</u>
bool	<u>updatesEnabled()</u> const
QRegion	<u>visibleRegion()</u> const
QString	<u>whatsThis()</u> const

int	width() const
WId	winId() const
QWidget *	window() const
QString	windowFilePath() const
Qt::WindowFlags	windowFlags() const
QWindow *	windowHandle() const
QIcon	windowIcon() const
Qt::WindowModality	windowModality() const
qreal	windowOpacity() const
QString	windowRole() const
Qt::WindowStates	windowState() const
QString	windowTitle() const
Qt::WindowType	windowType() const
int	x() const
int	y() const

Reimplemented Public Functions|重写公共函数

virtual QPaintEngine *	paintEngine() const override
------------------------	--

- 34 public functions inherited from [QObject](#)
- 14 public functions inherited from [QPaintDevice](#)

Public Slots|公共槽

bool	<u>close</u> ()
void	<u>hide</u> ()
void	<u>lower</u> ()
void	<u>raise</u> ()
void	<u>repaint</u> ()
void	<u>setDisabled</u> (bool <i>disable</i>)
void	<u>setEnabled</u> (bool)
void	<u>setFocus</u> ()
void	<u>setHidden</u> (bool <i>hidden</i>)
void	<u>setStyleSheet</u> (const QString & <i>styleSheet</i>)
virtual void	<u>setVisible</u> (bool <i>visible</i>)
void	<u>setWindowModified</u> (bool)
void	<u>setWindowTitle</u> (const QString &)
void	<u>show</u> ()
void	<u>showFullScreen</u> ()
void	<u>showMaximized</u> ()
void	<u>showMinimized</u> ()
void	<u>showNormal</u> ()
void	<u>update</u> ()

- 1 public slot inherited from [QObject](#)

Signals|信号

void	customContextMenuRequested (const QPoint & <i>pos</i>)
void	windowIconChanged (const QIcon & <i>icon</i>)
void	windowTitleChanged (const QString & <i>title</i>)

- 2 signals inherited from [QObject](#)

Static Public Members|静态公共成员

QWidget *	createWindowContainer (QWidget * <i>window</i> , QWidget * <i>parent</i> = nullptr, Qt::WindowFlags <i>flags</i> = ...)
QWidget *	find (WId <i>id</i>)
QWidget *	keyboardGrabber ()
QWidget *	mouseGrabber ()
void	setTabOrder (QWidget * <i>first</i> , QWidget * <i>second</i>)

- 10 static public members inherited from [QObject](#)

Protected Functions|事件函数

virtual void	actionEvent (QActionEvent * <i>event</i>)
virtual void	changeEvent (QEvent * <i>event</i>)
virtual void	closeEvent (QCloseEvent * <i>event</i>)
virtual void	contextMenuEvent (QContextMenuEvent * <i>event</i>)
void	create (WId <i>window</i> = 0, bool <i>initializeWindow</i> = true, bool <i>destroyOldWindow</i> = true)
void	destroy (bool <i>destroyWindow</i> = true, bool <i>destroySubWindows</i> = true)
virtual void	dragEnterEvent (QDragEnterEvent * <i>event</i>)

virtual void	<u>dragLeaveEvent</u> (QDragLeaveEvent * <i>event</i>)
virtual void	<u>dragMoveEvent</u> (QDragMoveEvent * <i>event</i>)
virtual void	<u>dropEvent</u> (QDropEvent * <i>event</i>)
virtual void	<u>enterEvent</u> (QEvent * <i>event</i>)
virtual void	<u>focusInEvent</u> (QFocusEvent * <i>event</i>)
bool	<u>focusNextChild</u> ()
virtual bool	<u>focusNextPrevChild</u> (bool <i>next</i>)
virtual void	<u>focusOutEvent</u> (QFocusEvent * <i>event</i>)
bool	<u>focusPreviousChild</u> ()
virtual void	<u>hideEvent</u> (QHideEvent * <i>event</i>)
virtual void	<u>inputMethodEvent</u> (QInputMethodEvent * <i>event</i>)
virtual void	<u>keyPressEvent</u> (QKeyEvent * <i>event</i>)
virtual void	<u>keyReleaseEvent</u> (QKeyEvent * <i>event</i>)
virtual void	<u>leaveEvent</u> (QEvent * <i>event</i>)
virtual void	<u>mouseDoubleClickEvent</u> (QMouseEvent * <i>event</i>)
virtual void	<u>mouseMoveEvent</u> (QMouseEvent * <i>event</i>)
virtual void	<u>mousePressEvent</u> (QMouseEvent * <i>event</i>)
virtual void	<u>mouseReleaseEvent</u> (QMouseEvent * <i>event</i>)
virtual void	<u>moveEvent</u> (QMoveEvent * <i>event</i>)
virtual bool	<u>nativeEvent</u> (const QByteArray & <i>eventType</i> , void * <i>message</i> , long * <i>result</i>)

virtual void	paintEvent (QPaintEvent * <i>event</i>)
virtual void	resizeEvent (QResizeEvent * <i>event</i>)
virtual void	showEvent (QShowEvent * <i>event</i>)
virtual void	tabletEvent (QTabletEvent * <i>event</i>)
virtual void	wheelEvent (QWheelEvent * <i>event</i>)

Reimplemented Protected Functions|重写事件函数

virtual bool	event (QEvent * <i>event</i>) override
virtual void	initPainter (QPainter * <i>painter</i>) const override
virtual int	metric (QPaintDevice::PaintDeviceMetric <i>m</i>) const override

- 9 protected functions inherited from [QObject](#)
- 1 protected function inherited from [QPaintDevice](#)

Protected Slots|私有槽

void	updateMicroFocus ()
------	-------------------------------------

Macros|

QWIDGETSIZE_MAX

Additional Inherited Members

- 1 public variable inherited from [QObject](#)
- 2 protected variables inherited from [QObject](#)
- 1 protected type inherited from [QPaintDevice](#)

Detailed Description|详细描述

The [QWidget](#) class is the base class of all user interface objects. # QWidget 类是所有用户界面对象的基类;

The widget is the atom of the user interface: it receives mouse, keyboard and other events from the window system, and paints a representation of itself on the screen. Every widget is rectangular, and they are sorted in a Z-order. A widget is clipped by its parent and by the widgets in front of it.

widget 是用户界面的基础: 它从窗口系统中获取鼠标、键盘和其他事件, 并且在屏幕上绘制窗口。每一个 widget 都是长方形的并且在 Z-order (Z 轴) 上分类。一个 widget 处于其父窗口和子窗口之间。

A widget that is not embedded in a parent widget is called a window. Usually, windows have a frame and a title bar, although it is also possible to create windows without such decoration using suitable [window flags](#)). In Qt, [QMainWindow](#) and the various subclasses of [QDialog](#) are the most common window types.

一个没有 parent widget 的 widget 将直接出现的桌面上。通常一个窗口包含一个主体 Frame 和一个标题栏, 有时也可以创建一个没有标题栏的无边框窗口。在 Qt 中 QMainWindow 和各种各样的 QDialog 是最常用的窗口类型。

Every widget's constructor accepts one or two standard arguments: # 每一个 widget 构造器接受一到两个参数;

1. QWidget *parent = 0 is the parent of the new widget. If it is 0 (the default), the new widget will be a window. If not, it will be a child of *parent*, and be constrained by *parent's* geometry (unless you specify [Qt::Window](#) as window flag).

QWidget *parent = None 参数是新 widget 的父 widget。如果没有设定则新 widget 将出现的桌面上; 若设定了 parent 参数, 新 widget 将作为父 widget 的子 widget 并且包含在父 widget 的结构中除非你特别设置了 Qt::Window 做为新的 window flag;

2. Qt::WindowFlags f = 0 (where available) sets the window flags; the default is suitable for almost all widgets, but to get, for example, a window without a window system frame, you must use special flags.

Qt:: WindowFlags f = 0 用来设置窗口样式; 默认的风格适合大多数 widgets, 当然你也可以设置一个特使的窗口样式;

[QWidget](#) has many member functions, but some of them have little direct functionality; for example, [QWidget](#) has a font property, but never uses this itself. There are many subclasses which provide real functionality, such as [QLabel](#), [QPushButton](#), [QListWidget](#), and [QTabWidget](#).

QWidget 有许多成员函数但是有些函数几乎没有直接的功能: 例如, QWidget 有一个 font(字体)属性, 但是确从来不会自己使用该属性。QWidget 有许多象 QLabel, QPushButton, QListWidget, QTabWidget 子类, 这些子类提供一些真正的功能函数;

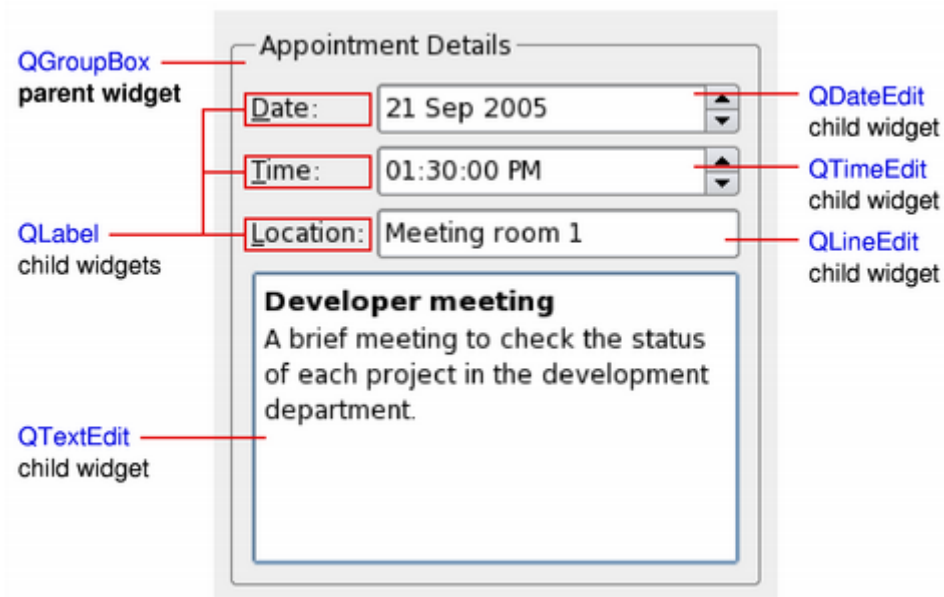
Top-Level and Child Widgets

A widget without a parent widget is always an independent window (top-level widget). For these widgets, [setWindowTitle\(\)](#) and [setWindowIcon\(\)](#) set the title bar and icon respectively.

一个没有 parent widget 的 widget 的部件会成为一个独立的窗口(顶层窗口)。对这些窗口来说使用 [setWindowTitle\(\)](#) 和 [setWindowIcon\(\)](#) 用来设置标题、图标是有效的；

Non-window widgets are child widgets, displayed within their parent widgets. Most widgets in Qt are mainly useful as child widgets. For example, it is possible to display a button as a top-level window, but most people prefer to put their buttons inside other widgets, such as [QDialog](#).

非顶层窗口 widget 都是子 widget，显示在他们的父 widget 中。Qt 大多数的 widget 是作为子 widget 使用的。例如，可以将 Button 作为顶层窗口但是大多数时候它都是放在其他 widget 中使用的如 QDialog。



The diagram above shows a [QGroupBox](#) widget being used to hold various child widgets in a layout provided by [QGridLayout](#). The [QLabel](#) child widgets have been outlined to indicate their full sizes.

像上面显示的那样用一个 [QGroupBox](#) widget 使用 [QGridLayout](#) 布局装载多种子 widget。一个 [QLabel](#) 子部件在外部指示它的大小。

If you want to use a [QWidget](#) to hold child widgets you will usually want to add a layout to the parent [QWidget](#). See [Layout Management](#) for more information.

你经常想要添加一种布局方式在父 [QWidget](#) 中来安置各种子 widget。可以查阅 [Layout Management](#) 来获取更多信息

Composite Widgets

When a widget is used as a container to group a number of child widgets, it is known as a composite widget. These can be created by constructing a widget with the required visual properties - a [QFrame](#), for example - and adding child widgets to it, usually managed by a layout. The above diagram shows such a composite widget that was created using Qt Designer.

当一个 widget 包含了一系列的子 widget 是就被认为是一个复合 widget。可以通过构造一个符合可视需求的 widget 来装载子 widget 例如 QFrame，通常通过布局管理来放置子 widget。上图显示了 Qt Designer 的一个复合 widget。

Composite widgets can also be created by subclassing a standard widget, such as [QWidget](#) or [QFrame](#), and adding the necessary layout and child widgets in the constructor of the subclass. Many of the [examples provided with Qt](#) use this approach, and it is also covered in the Qt [Tutorials](#).

复合 widget 可被创建归类为标准 widget 就像 QWidget 或 QFrame 一样，添加必要的布局和子 widget 来构造一个子类。有许多 Qt 例程使用这种方法，在 Qt Tutorials 有许多教程。

Custom Widgets and Painting

Since [QWidget](#) is a subclass of [QPaintDevice](#), subclasses can be used to display custom content that is composed using a series of painting operations with an instance of the [QPainter](#) class. This approach contrasts with the canvas-style approach used by the [Graphics View Framework](#) where items are added to a scene by the application and are rendered by the framework itself.

QWidget 是 QPaintDevice 的子类，故它可以用来显示定制性的内容，通过构建 QPainter 实例使用系列的绘画操作。构建 QPainter 实例的方法和 canvas-style 方法用来构建 Graphics View Framework(视图框架)，在框架中将元素添加到应用程序并显示出来。

Each widget performs all painting operations from within its [paintEvent\(\)](#) function. This is called whenever the widget needs to be redrawn, either as a result of some external change or when requested by the application.

每一个 widget 都依赖 paintEvent() 函数来执行绘画操作。无论是因为外部改变还是应用程序请求需要重绘都会调用 paintEvent()。

The [Analog Clock example](#) shows how a simple widget can handle paint events.

Size Hints and Size Policies

When implementing a new widget, it is almost always useful to reimplement [sizeHint\(\)](#) to provide a reasonable default size for the widget and to set the correct size policy with [setSizePolicy\(\)](#).

当构建一个新的 widget 是，它总会重新实现 sizeHint()，提供 widget 必要的默认大小并且使用 setSizePolicy() 设置正确的大小策略。

By default, composite widgets which do not provide a size hint will be sized according to the space requirements of their child widgets.

The size policy lets you supply good default behavior for the layout management system, so that other widgets can contain and manage yours easily. The default size policy indicates that the size hint represents the preferred size of the widget, and this is often good enough for many widgets.

那些没有提供大小设置的复合窗口会默认依据子 widget 所需要的空间来设置其大小。Size policy 会为你的布局管理系统提供一个良好默认方式，这样其所包含的子 widget 就能很好布局且易于管理。默认的 Size policy 一种优先的 widget 大小，这对大多数 widget 已经足够好用了。

Note: The size of top-level widgets are constrained to 2/3 of the desktop's height and width. You can [resize\(\)](#) the widget manually if these bounds are inadequate.

贴士：顶层 widget 的大小被限制在桌面 2 高/3 宽。如果大小不够你可以使用 [resize\(\)](#)来重新设置。

Events

Widgets respond to events that are typically caused by user actions. Qt delivers events to widgets by calling specific event handler functions with instances of [QEvent](#) subclasses containing information about each event.

widget 响应通常由用户操作引起。Qt 通过调用特定的事件处理函数来将事件传递给 widget，其中包含每个事件信息的 QEvent 的实例。

If your widget only contains child widgets, you probably do not need to implement any event handlers. If you want to detect a mouse click in a child widget call the child's [underMouse\(\)](#) function inside the widget's [mousePressEvent\(\)](#).

如果你的 widget 仅包好子 widget，那么可能不需要任何事件处理。如果你想获取鼠标点击子 widget，调用 [mousePressEvent\(\)](#)下的 [underMouse\(\)](#) The [Scribble example](#) implements a wider set of events to handle mouse movement, button presses, and window resizing.

Scribble example 实例显示了更多的事件处理鼠标移动、按钮按压和窗口大小调整。

You will need to supply the behavior and content for your own widgets, but here is a brief overview of the events that are relevant to [QWidget](#), starting with the most common ones:

你需要为你的 widget 提供行为和内容，这里有一个与 QWidget 相关的事件的简要概述，先从最常见的开始：

- [paintEvent\(\)](#) is called whenever the widget needs to be repainted. Every widget displaying custom content must implement it. Painting using a [QPainter](#) can only take place in a [paintEvent\(\)](#) or a function called by a [paintEvent\(\)](#).
[paintEvent\(\)](#)，在 widget 需要重绘是调用。每个显示订制内容的 widget 都必须调用。每个绘制 [QPainter\(\)](#)都要在 [paintEvent\(\)](#)上或调用其实现
- [resizeEvent\(\)](#) is called when the widget has been resized.
[resizeEvent\(\)](#)在 widget 调整大小是调用；
- [mousePressEvent\(\)](#) is called when a mouse button is pressed while the mouse cursor is inside the widget, or when the widget has grabbed the mouse using [grabMouse\(\)](#). Pressing the mouse without releasing it is effectively the same as calling [grabMouse\(\)](#).
[mousePressEvent\(\)](#)当鼠标指针在 widget 内且鼠标按钮按下时调用或者当 widget 使用 [grabMouse\(\)](#)捕获了鼠标操作。在不释放鼠标的情况下按下鼠标实际上与 [grabMouse\(\)](#)是等效的；

- [mouseReleaseEvent\(\)](#) is called when a mouse button is released. A widget receives mouse release events when it has received the corresponding mouse press event. This means that if the user presses the mouse inside *your* widget, then drags the mouse somewhere else before releasing the mouse button, *your* widget receives the release event. There is one exception: if a popup menu appears while the mouse button is held down, this popup immediately steals the mouse events.
[mouseReleaseEvent\(\)](#)在鼠标按键释放时调用。当 **widget** 接收到鼠标按压事件时，会接收相应的鼠标释放事件。这意味着如果在 **widget** 内接收到鼠标按压事件然后在释放前将鼠标拖到其他地方释放，**widget** 也可以接收鼠标释放事件。这里有一个例外：如果当鼠标按下时出现了弹出菜单，该菜单会立即窃取鼠标事件，**release event** 将不被接收。
- [mouseDoubleClickEvent\(\)](#) is called when the user double-clicks in the widget. If the user double-clicks, the widget receives a mouse press event, a mouse release event, (a mouse click event,) a second mouse press, this event and finally a second mouse release event. (Some mouse move events may also be received if the mouse is not held steady during this operation.) It is *not possible* to distinguish a click from a double-click until the second click arrives. (This is one reason why most GUI books recommend that double-clicks be an extension of single-clicks, rather than trigger a different action.)
[mouseDoubleClickEvent\(\)](#)当鼠标在 **widget** 内双击时调用。当你双击鼠标时，先获取两个鼠标单击事件（如果不能保持鼠标稳定还会获取鼠标移动事件），无法在第二次单击事件到达前区分单击还是双击。这也是为什么大多数 GUI 书籍将双击当做单击的延伸而不是区分为不同的时间

Widgets that accept keyboard input need to reimplement a few more event handlers:

widget 接收键盘输入需要重新实现一些事件处理函数：

- [keyPressEvent\(\)](#) is called whenever a key is pressed, and again when a key has been held down long enough for it to auto-repeat. The **Tab** and **Shift+Tab** keys are only passed to the widget if they are not used by the focus-change mechanisms. To force those keys to be processed by your widget, you must reimplement [QWidget::event\(\)](#).
[keyPressEvent\(\)](#)当一个键被按下时调用，当键被长时间按下则自动重复调用。如果为使用 **focus-change** 机制 **Tab** 和 **Shift+Tab** 将只会传递给 **widget**。如果要使用这些键，你必须重写 [QWidget::event\(\)](#)。
- [focusInEvent\(\)](#) is called when the widget gains keyboard focus (assuming you have called [setFocusPolicy\(\)](#)). Well-behaved widgets indicate that they own the keyboard focus in a clear but discreet way.
[focusInEvent\(\)](#) **widget** 获取键盘焦点(假设已经设置了 [setFocusPolicy\(\)](#))时调用。
- [focusOutEvent\(\)](#) is called when the widget loses keyboard focus.
[focusOutEvent\(\)](#)当 **widget** 丢失键盘焦点时调用

You may be required to also reimplement some of the less common event handlers:

您可能还需要重新实现一些不太常见的事件处理程序：

- [mouseMoveEvent\(\)](#) is called whenever the mouse moves while a mouse button is held down. This can be useful during drag and drop operations. If you call [setMouseTracking\(true\)](#), you get mouse move events even when no buttons are held down. (See also the [Drag and Drop](#) guide.)
mouseMoveEvent()当鼠标有一个键按下且鼠标移动时调用。这个可以用在拖动和放置功能中。如果你设置 [setMouseTracking\(true\)](#)，即使没有按住鼠标按钮也会触发该事件。（详细请查看 [Drag](#) 和 [Drop](#) 说明）
- [keyReleaseEvent\(\)](#) is called whenever a key is released and while it is held down (if the key is auto-repeating). In that case, the widget will receive a pair of key release and key press event for every repeat. The **Tab** and **Shift+Tab** keys are only passed to the widget if they are not used by the focus-change mechanisms. To force those keys to be processed by your widget, you must reimplement [QWidget::event\(\)](#).
keyReleaseEvent()当键盘键释放或持续按键被释放(键盘键自动填充)时触发。在这种情况下，widget 会收到一个 key release(键释放) 和 key press (键按压)事件。如果没有改变 focus-change mechanisms 的话，Tab 和 Shift+Tab 只会收到一个 key press(键按压)事件。如果想强制 widget 处理这些键(Tab 和 Shift+Tab 等)，你必须重写 QWidget::Event()。
- [wheelEvent\(\)](#) is called whenever the user turns the mouse wheel while the widget has the focus.
wheelEvent()当 widget 设置了焦点，并滚动鼠标滚轮是触发。
- [enterEvent\(\)](#) is called when the mouse enters the widget's screen space. (This excludes screen space owned by any of the widget's children.)
enterEvent()当鼠标进入 widget 在屏幕上显示的空间是触发。（这时就禁用了 widget 的子 widget 触发该事件）
- [leaveEvent\(\)](#) is called when the mouse leaves the widget's screen space. If the mouse enters a child widget it will not cause a [leaveEvent\(\)](#).
leaveEvent()当鼠标离开 widget 的屏幕空间时触发。如果鼠标进入子 widget 将不会触发 leaveEvent()事件。
- [moveEvent\(\)](#) is called when the widget has been moved relative to its parent.
moveEvent()当 widget 相对于其父 widget 移动时触发。
- [closeEvent\(\)](#) is called when the user closes the widget (or when [close\(\)](#) is called).
closeEvent()当用户关闭 widget （或使用 close()函数）时触发。

There are also some rather obscure events described in the documentation for [QEvent::Type](#). To handle these events, you need to reimplement [event\(\)](#) directly.

在 [Qevent::Type](#) 的文档中还有一些其他的事件详细描述。若要使用这些事件，你需要明确的重写事件内容：

The default implementation of [event\(\)](#) handles **Tab** and **Shift+Tab** (to move the keyboard focus), and passes on most of the other events to one of the more specialized handlers above.

event()默认执行 Tab 和 Shift+Tab（去更改键盘焦点），并将大部分其他事件传递给上面的一个更专业的处理程序。

Events and the mechanism used to deliver them are covered in [The Event System](#).

事件和其传递机制都包含在 [The Event System](#) 中。

Groups of Functions and Properties

Context	Functions and Properties
Window functions 窗口函数	show() , hide() , raise() , lower() , close() .
Top-level windows 顶层窗口函数	windowModified() , windowTitle() , windowIcon() , isActiveWindow() , activateWindow() , minimized() , showMinimized() , maximized() , showMaximized() , fullScreen() , showFullScreen() , showNormal() .
Window contents 窗口内容函数	update() , repaint() , scroll() .
Geometry 位置&大小函数	pos() , x() , y() , rect() , size() , width() , height() , move() , resize() , sizePolicy() , sizeHint() , minimumSizeHint() , updateGeometry() , layout() , frameGeometry() , geometry() , childrenRect() , childrenRegion() , adjustSize() , mapFromGlobal() , mapToGlobal() , mapFromParent() , mapToParent() , maximumSize() , minimumSize() , sizeIncrement() , baseSize() , setFixedSize()
Mode 模式	visible() , isVisibleTo() , enabled() , isEnabledTo() , modal() , isWindow() , mouseTracking() , updatesEnabled() , visibleRegion() .
Look and feel 外观函数	style() , setStyle() , styleSheet() , cursor() , font() , palette() , backgroundRole() , setBackgroundRole() , fontInfo() , fontMetrics() .
Keyboard focus functions 键盘焦点函数	focus() , focusPolicy() , setFocus() , clearFocus() , setTabOrder() , setFocusProxy() , focusNextChild() , focusPreviousChild() .
Mouse and keyboard grabbing 键盘&鼠标抓取	grabMouse() , releaseMouse() , grabKeyboard() , releaseKeyboard() , mouseGrabber() , keyboardGrabber() .
Event handlers 事件处理	event() , mousePressEvent() , mouseReleaseEvent() , mouseDoubleClickEvent() , mouseMoveEvent() , keyPressEvent() , keyReleaseEvent() , focusInEvent() , focusOutEvent() , wheelEvent() , enterEvent() , leaveEvent() , paintEvent() , moveEvent() , resizeEvent() , closeEvent() , dragEnterEvent() , dragMoveEvent() , dragLeaveEvent() , dropEvent() , childEvent() , showEvent() , hideEvent() , customEvent() . changeEvent() ,
System functions 系统函数	parentWidget() , window() , setParent() , winId() , find() , metric() .
Context menu 菜单函数	contextMenuPolicy() , contextMenuEvent() , customContextMenuRequested() , actions()

Context	Functions and Properties
Interactive help 交互提示	setToolTip() , setWhatsThis()

Widget Style Sheets|Widget 样式表

In addition to the standard widget styles for each platform, widgets can also be styled according to rules specified in a [style sheet](#). This feature enables you to customize the appearance of specific widgets to provide visual cues to users about their purpose. For example, a button could be styled in a particular way to indicate that it performs a destructive action.

除了每个平台标准 widget 样式外，widget 还能依据样式表指定的规则进行样式化。这个规则使你能订制美化的 widget 样式，从而为用户提供更适
合其用途的外观。例如，一个按钮可以通过特定的样式化来指出其破坏的功能。

The use of widget style sheets is described in more detail in the [Qt Style Sheets](#) document.

widget 样式表的详细描述在 Qt Style Sheet 文档中。

Transparency and Double Buffering|透明度和双缓冲技术

Since Qt 4.0, [QWidget](#) automatically double-buffers its painting, so there is no need to write double-buffering code in [paintEvent\(\)](#) to avoid flicker.

自从 Qt 4.0 起 QWidget 自动使用双缓冲构图，所以不用在 paintEvent() 写双缓冲代码来避免闪烁。

Since Qt 4.1, the [Qt::WA_ContentsPropagated](#) widget attribute has been deprecated. Instead, the contents of parent widgets are propagated by default to each of their children as long as [Qt::WA_PaintOnScreen](#) is not set. Custom widgets can be written to take advantage of this feature by updating irregular regions (to create non-rectangular child widgets), or painting with colors that have less than full alpha component. The following diagram shows how attributes and properties of a custom widget can be fine-tuned to achieve different effects.

自从 Qt 4.0 起，Qt::WA_ContentsPropagated widget 属性已经被弃用。取而代之，只要 Qt::WA_PaintOnScreen 未设置，父 widget 的属性默认传
递给子 widget。定制 widget 可以通过更新不规则的区域（创建非矩形的子部件）来利用这个特性，或者用小于完全 alpha 组件的颜色进行绘画。下
图展示了如何对定制 widget 的属性和属性进行微调，以获得不同的效果。



In the above diagram, a semi-transparent rectangular child widget with an area removed is constructed and added to a parent widget (a [QLabel](#) showing a pixmap). Then, different properties and widget attributes are set to achieve different effects:

在上面的图中，一个带有区域移除的半透明矩形子 `widget` 被构造并添加到父 `widget`（一个显示像素图的 `QLabel`）。然后，将不同的属性和小部件属性设置为不同的效果：

- The left widget has no additional properties or widget attributes set. This default state suits most custom widgets using transparency, are irregularly-shaped, or do not paint over their entire area with an opaque brush.

左边的小部件没有附加的属性或 `widget` 属性集。这个默认状态适用于大多数自定义 `widget`，使用透明，是不规则形状的，或者不使用不透明的笔刷绘制整个区域

- The center widget has the [autoFillBackground](#) property set. This property is used with custom widgets that rely on the widget to supply a default background, and do not paint over their entire area with an opaque brush.

这个中心 `widget` 有自动补全属性集。这个属性与自定义 `widget` 一起使用，这些 `widget` 依赖于 `widget` 来提供默认的背景，并且不使用不透明的笔刷绘制整个区域。

- The right widget has the [Qt::WA_OpaquePaintEvent](#) widget attribute set. This indicates that the widget will paint over its entire area with opaque colors. The widget's area will initially be *uninitialized*, represented in the diagram with a red diagonal grid pattern that shines through the overpainted area. The `Qt::WA_OpaquePaintArea` attribute is useful for widgets that need to paint their own specialized contents quickly and do not need a default filled background.

右边的 `widget` 有 `Qt::WA_OpaquePaintEvent` 属性集。这表明 `widget` 将用不透明的颜色覆盖整个区域。`widget` 的区域最初是未初始化的，在图中表示一个红色的对角网格模式，它可以在覆盖过的区域中发光。`Qt::WA_OpaquePaintEvent` 属性对于那些需要快速绘制自己的专业内容的小部件非常有用，并且不需要一个默认的填充背景。

To rapidly update custom widgets with simple background colors, such as real-time plotting or graphing widgets, it is better to define a suitable background color (using `setBackgroundRole()` with the `QPalette::Window` role), set the `autoFillBackground` property, and only implement the necessary drawing functionality in the widget's `paintEvent()`.

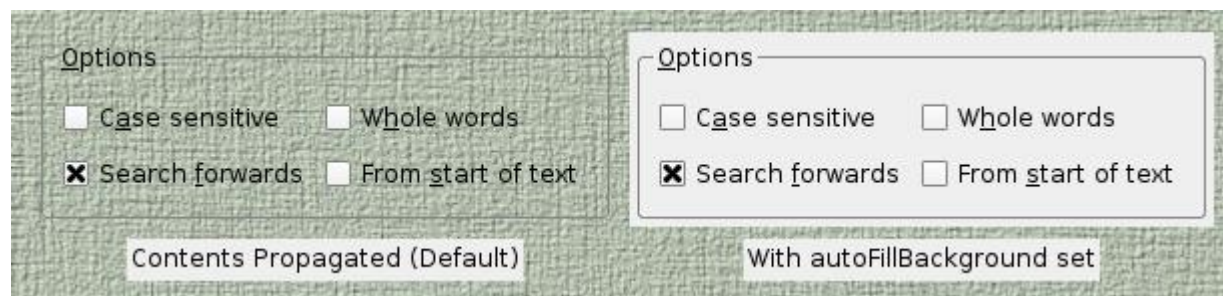
快速简单的修改 widget 的背景色，例如使用实时绘图或绘制 widget，最好的做法是使用 `setBackgroundRole()` 和 `QPalette::Window` 函数并设置 `autoFillBackground` 属性，在 `paintEvent()` 中执行必要的绘制函数。

To rapidly update custom widgets that constantly paint over their entire areas with opaque content, e.g., video streaming widgets, it is better to set the widget's `Qt::WA_OpaquePaintEvent`, avoiding any unnecessary overhead associated with repainting the widget's background.

If a widget has both the `Qt::WA_OpaquePaintEvent` widget attribute *and* the `autoFillBackground` property set, the `Qt::WA_OpaquePaintEvent` attribute takes precedence. Depending on your requirements, you should choose either one of them.

Since Qt 4.1, the contents of parent widgets are also propagated to standard Qt widgets. This can lead to some unexpected results if the parent widget is decorated in a non-standard way, as shown in the diagram below.

为了快速更新自定义 widget，这些 widget 以不透明的内容在整个区域内进行绘图，例如，流媒体 widget，最好设置的 `Qt::WA_OpaquePaintEvent`，避免与重新绘制 widget 的背景相关的任何不必要的开销。如果一个 widget 有 `Qt::WA_OpaquePaintEvent` 属性和 `autoFillBackground` 属性，`Qt::WA_OpaquePaintEvent` 属性优先。根据您的需求，您应该选择其中一个。自 Qt 4.1 以来，父 widget 的属性也被传播到标准 Qt widget。如果父 widget 以非标准的方式进行装饰，这可能会导致一些意想不到的结果，如下图所示。



The scope for customizing the painting behavior of standard Qt widgets, without resorting to subclassing, is slightly less than that possible for custom widgets. Usually, the desired appearance of a standard widget can be achieved by setting its `autoFillBackground` property.

在不使用子类化的情况下，定制标准 Qt widget 的绘画行为的范围比定制 widget 的范围要小一些。通常，标准 widget 的期望外观可以通过设置它的 `autoFillBackground` 属性来实现。

Creating Translucent Windows|创建半透明窗口

Since Qt 4.5, it has been possible to create windows with translucent regions on window systems that support compositing.

自从 Qt 4.5 以来，可以在支持合成的窗口系统上创建具有半透明区域的窗口

To enable this feature in a top-level widget, set its `Qt::WA_TranslucentBackground` attribute with `setAttribute()` and ensure that its background is painted with non-opaque colors in the regions you want to be partially transparent.

为了在顶层 widget 中启用该特性，请将其 `Qt::WA_TranslucentBackground` 属性设置为 `setAttribute()`，并确保其背景在您希望部分透明的区域中被涂上不透明的颜色。

Platform notes: # 平台说明：

- X11: This feature relies on the use of an X server that supports ARGB visuals and a compositing window manager.

X11: 这个特性依赖于使用支持 ARGB 视觉效果 of X 服务器和一个合成窗口管理器。

- Windows: The widget needs to have the `Qt::FramelessWindowHint` window flag set for the translucency to work.

Windows: 小部件需要有 `Qt::FramelessWindowHint` 窗口标志，用于半透明工作。

Native Widgets vs Alien Widgets|本地 widget&外部 widget

Introduced in Qt 4.4, alien widgets are widgets unknown to the windowing system. They do not have a native window handle associated with them. This feature significantly speeds up widget painting, resizing, and removes flicker.

在 Qt 4.4 中引入的外部 widget 是窗口系统未知的 widget。它们没有与之相关联的本地窗口句柄。这个特性极大地加快了 widget 的绘制、调整大小和消除闪烁。

Should you require the old behavior with native windows, you can choose one of the following options:

如果您需要使用本机 windows 的旧风格，您可以选择以下选项之一：

1. Use the `QT_USE_NATIVE_WINDOWS=1` in your environment.

设置 `QT_USE_NATIVE_WINDOWS=1`；

2. Set the `Qt::AA_NativeWindows` attribute on your application. All widgets will be native widgets.

设置 `Qt::AA_NativeWindows` 属性，使所有 widget 都使用本地风格；

3. Set the `Qt::WA_NativeWindow` attribute on widgets: The widget itself and all of its ancestors will become native (unless `Qt::WA_DontCreateNativeAncestors` is set).

设置 `Qt::WA_NativeWindow` 属性：所有 widget 都将为本地风格除非 `Qt::WA_DontCreateNativeAncestors` 属性；

4. Call `QWidget::winId` to enforce a native window (this implies 3).
调用 `QWidget::winId` 强制执行本机窗口;
5. Set the `Qt::WA_PaintOnScreen` attribute to enforce a native window (this implies 3).
设置 `Qt::WA_PaintOnScreen` 属性来强制执行本机窗口;
- See also `QEvent`, `QPainter`, `QGridLayout`, and `QBoxLayout`.

Member Type Documentation|成员类型文档

enum `QWidget::RenderFlag`

flags `QWidget::RenderFlags`

This enum describes how to render the widget when calling `QWidget::render()`.

这个 enum 描述了在调用 `QWidget::render()` 时如何呈现 widget

Constant 属性	Value 值	Description 描述
<code>QWidget::DrawWindowBackground</code>	0x1	If you enable this option, the widget's background is rendered into the target even if <code>autoFillBackground</code> is not set. By default, this option is enabled. # 如果你激活了这个选项, widget 的背景将被转换为 target, 无论 <code>autoFillBackground</code> 是否被设置。这个选项默认是激活的;
<code>QWidget::DrawChildren</code>	0x2	If you enable this option, the widget's children are rendered recursively into the target. By default, this option is enabled. # 如果你激活了这个选项, widget 的子 widget 将被递归的输入的 target 中去。该选项默认激活;
<code>QWidget::IgnoreMask</code>	0x4	If you enable this option, the widget's <code>QWidget::mask()</code> is ignored when rendering into the target. By default, this option is disabled. # widget 的 <code>QWidget::mask()</code> 在渲染到目标时被忽略, 默认该选项关闭;

This enum was introduced or modified in Qt 4.3. # 这个 enum 的在 Qt 4.3 被引进并修改。

The RenderFlags type is a typedef for [QFlags<RenderFlag>](#). It stores an OR combination of RenderFlag values.
RenderFlags 类型是 QFlags<RenderFlag>。它存储了 RenderFlag 值的 OR 组合。

Property Documentation|属性文档

acceptDrops : bool

This property holds whether drop events are enabled for this widget # 这个属性用来设置拖放事件是否适用于此小部件

Setting this property to true announces to the system that this widget *may* be able to accept drop events.
设置此属性为 True 意味着向系统宣称此小部件可能会接受拖放事件
If the widget is the desktop ([windowType\(\)](#) == [Qt::Desktop](#)), this may fail if another application is using the desktop; you can call `acceptDrops()` to test if this occurs.
如果此窗口为顶层窗口(`windowType() == Qt::Desktop`)可以适用 `acceptDrops()`函数去验证设置是否工作。
Warning: Do not modify this property in a drag and drop event handler.
警告: 不要在 drag 和 drop event handler 中修改此项属性

By default, this property is false. # 此属性默认为 False

Access functions:

bool	<code>acceptDrops()</code>	const	# 获得属性值, 测试是否启用属性
void	<code>setAcceptDrops</code>	(bool on)	# 设置属性值

See also [Drag and Drop](#).

```
例子:
# !/usr/bin/env python
# -*- coding: utf-8 -*-
"""
```

Created on Fri Jul 6 16:04:02 2018

@author: 肖建斌

@Function: 连接拖放事件处理

"""

import sys

from PyQt5.QtWidgets import *

from PyQt5.QtGui import *

from PyQt5.QtCore import *

class DragTextButton(QPushButton):

拖动改变文字按钮

def __init__(self, title, parent):

super(DragTextButton, self).__init__(title, parent)

self.setAcceptDrops(True) # 启用控件 Drop 事件

def dragEnterEvent(self, e):

重新实现 dragEnterEvent() 方法，设置接收的数据类型为<明文本格式>

if e.mimeData().hasFormat('text/plain'):

e.accept()

else:

e.ignore()

def dropEvent(self, e):

重新实现 dropEvent() 方法，定义当 drop 事件到达时，将采取何种操作。此处为修改控件内容

self.setText(e.mimeData().text())

class DragMoveButton(QPushButton):

拖动改变位置按钮

def __init__(self, title, parent):

```

        super(DragMoveButton, self).__init__(title, parent)

def mousePressEvent(self, event):
    # 重写鼠标按下事件
    QPushButton.mousePressEvent(self, event)

def mouseMoveEvent(self, event):
    if (event.button() == Qt.LeftButton): # 如果按下鼠标左键
        return
    data = QMimeData()
    # QMimeData 在拖拽中非常有用，可以用来保存拖拽操作附带的信息，
    # 比如字符串、文件或者图片，同时也可以用来验证其所保存的信息格式，
    # 并以此来判断是否可接收
    drag = QDrag(self) # 创建一个拖动类
    drag.setMimeData(data)
    drag.setHotSpot(event.pos() - self.rect().topLeft())
    dropAction = drag.exec_(Qt.MoveAction)

class DragOpneFileButton(QPushButton):
    # 拖动获取文件名按钮
    def __init__(self, title, parent):
        super(DragOpneFileButton, self).__init__(title, parent)
        self.setAcceptDrops(True)

    def dragEnterEvent(self, event):
        if (event.mimeData().hasFormat("text/uri-list")):
            event.acceptProposedAction()

    def dropEvent(self, event):
        urls = event.mimeData().urls()[0] # 文件路径 QUrl
        if urls.isEmpty(): # 验证路径是否为空

```



```

        return
        fileName = urls.fileName() # 获取文件名
        self.setText(fileName)

class Example(QWidget):
    def __init__(self, parent=None):
        super(Example, self).__init__(parent)
        self.initUI()

    def initUI(self):
        edit = QLineEdit("", self)
        # QLineEdit 控件内置支持拖拽（drag）操作，使用 setDragEnabled(True)激活
        edit.setDragEnabled(True)
        edit.move(30, 65)

        changeTextButton = DragTextButton('拖放改变内容按钮', self)
        changeTextButton.move(30, 95)

        self.moveButton = DragMoveButton('可拖动按钮', self)
        self.moveButton.move(30, 125)
        self.setAcceptDrops(True) # 按钮在主界面拖动，设置主界面接受拖动事件

        openFileButton = DragOpneFileButton('拖动文件打开按钮', self)
        openFileButton.setGeometry(300, 300, 200, 200)

        self.setWindowTitle('简单拖放测试界面')
        self.setGeometry(300, 300, 600, 600)
        self.show()

    def dropEvent(self, event):
        position = event.pos() # 放置终点位置

```

```
self.moveButton.move(position) # 将部件位置
event.setDropAction(Qt.MoveAction) # 设置部件放置事件响应类型
event.accept() # 接受事件
```

```
def dragEnterEvent(self, event):
    event.accept() # 拖动, 接受事件
```

```
if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())
```

accessibleDescription : [QString](#)

This property holds the widget's description as seen by assistive technologies

The accessible description of a widget should convey what a widget does. While the [accessibleName](#) should be a short and concise string (e.g. **Save**), the description should give more context, such as **Saves the current document**.

This property has to be [localized](#).

By default, this property contains an empty string and Qt falls back to using the tool tip to provide this information.

Access functions:

QString	<code>accessibleDescription()</code> const
void	<code>setAccessibleDescription(const QString &<i>description</i>)</code>

See also [QWidget::accessibleName](#) and [QAccessibleInterface::text\(\)](#).

accessibleName : [QString](#)

This property holds the widget's name as seen by assistive technologies

This is the primary name by which assistive technology such as screen readers announce this widget. For most widgets setting this property is not required. For example for [QPushButton](#) the button's text will be used. It is important to set this property when the widget does not provide any text. For example a button that only contains an icon needs to set this property to work with screen readers. The name should be short and equivalent to the visual information conveyed by the widget.

This property has to be [localized](#).
By default, this property contains an empty string.

Access functions:

QString	<code>accessibleName() const</code>
void	<code>setAccessibleName(const QString &name)</code>

See also [QWidget::accessibleDescription](#) and [QAccessibleInterface::text\(\)](#).

autoFillBackground : bool

This property holds whether the widget background is filled automatically # 用来设置部件背景是否自动填充;

If enabled, this property will cause Qt to fill the background of the widget before invoking the paint event. The color used is defined by the [QPalette::Window](#) color role from the widget's [palette](#).
如果启用了这个属性将导致 Qt 在 paintEvent 前自动填充部件背景色。颜色将由 Qpalette 定义;
In addition, Windows are always filled with [QPalette::Window](#), unless the WA_OpaquePaintEvent or WA_NoSystemBackground attributes are set. This property cannot be turned off (i.e., set to false) if a widget's parent has a static gradient for its background.
此外, Windows 窗口总是充满各种 QPalette, 除非设置 WA_OpaquePaintEvent 或 WA_NoSystemBackground, 否则这个属性无法被关闭;

Warning: Use this property with caution in conjunction with [Qt Style Sheets](#). When a widget has a style sheet with a valid background or a border-image, this property is automatically disabled.
警告: 这个属性与 QStyleSheet()同时使用应特别注意, 当 StyleSheet 设置了 background 获取 border-image 时自动填充属性设置无效。

By default, this property is false. # 该属性默认为 False

This property was introduced in Qt 4.1.

Access functions:

bool	autoFillBackground() const	# 查看背景自动填充状态
void	setAutoFillBackground(bool <i>enabled</i>)	# 设置背景自动填充属性

See also [Qt::WA_OpaquePaintEvent](#), [Qt::WA_NoSystemBackground](#), and [Transparency and Double Buffering](#).

```
例子：
# !/usr/bin/env python
# -*- coding: utf-8 -*-

import sys
import os
from PyQt5.QtWidgets import *
from PyQt5.QtGui import *
from PyQt5.QtCore import *

class Example(QWidget):
    def __init__(self, parent=None):
        super(Example, self).__init__(parent)
        self.initUI()

    def initUI(self):
        image = '\\'.join([os.getcwd(), 'Background.jpg']) # 背景图地址
        self.setWindowTitle('简单测试界面')
        self.setGeometry(300, 300, 600, 600)
        self.setAutoFillBackground(True) # 使用 QPalette 设置背景时必须设置该属性为 True
        palette1 = QPalette() # 调用调色板
```

```
# palette1.setColor(self.backgroundRole(), QColor(192,253,123)) # 设置背景颜色
palette1.setBrush(self.backgroundRole(), QBrush(QPixmap(image))) # 设置背景图片
self.setPalette(palette1) # 应用调色板
self.show()
```

```
if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())
```

baseSize : [QSize](#)

This property holds the base size of the widget # 该属性设置小部件的基础尺寸；

The base size is used to calculate a proper widget size if the widget defines [sizeIncrement\(\)](#).

如果部件定义了 [sizeIncrement\(\)](#)，该基础用于计算一个合适的部件大小；

By default, for a newly-created widget, this property contains a size with zero width and height.

该属性默认值为 [baseWidht=0](#)，[baseHight=0](#)。

Access functions:

QSize	baseSize() const	# 获取基础部件大小
void	setBaseSize (const QSize &)	# 使用 Qsiz() 设置基础部件大小
void	setBaseSize (int basew, int baseh)	# 使用 (int basew, int baseh) 设置基础部件大小

See also [setSizeIncrement\(\)](#).

childrenRect : const [QRect](#)

This property holds the bounding rectangle of the widget's children # 该属性包含所有子部件的矩形边界;

Hidden children are excluded. # 排除隐藏的子部件

By default, for a widget with no children, this property contains a rectangle with zero width and height located at the origin.

对于没有子部件的部件，该属性默认为一个宽度和高度均为 0 在原点的矩形;

Access functions:

QRect	childrenRect() const	# 获取属性值，该值为只读 QRect()
-------	----------------------	-----------------------

See also [childrenRegion\(\)](#) and [geometry\(\)](#).

childrenRegion : const [QRegion](#)

This property holds the combined region occupied by the widget's children # 这个属性包含小部件的子元素所占用的合并区域

Hidden children are excluded. # 排除隐藏的子部件

By default, for a widget with no children, this property contains an empty region. # 默认部件没有子级，这个属性为一个包含一个空的区域

Access functions:

QRegion	childrenRegion() const	# 获取属性值，该值为只读 QRegion()
---------	------------------------	-------------------------

See also [childrenRect\(\)](#), [geometry\(\)](#), and [mask\(\)](#).

contextMenuPolicy : Qt::ContextMenuPolicy

how the widget shows a context menu # 该属性设置部件如何显示其右键菜单

The default value of this property is `Qt::DefaultContextMenu`, which means the `contextMenuEvent()` handler is called. Other values are `Qt::NoContextMenu`, `Qt::PreventContextMenu`, `Qt::ActionsContextMenu`, and `Qt::CustomContextMenu`. With `Qt::CustomContextMenu`, the signal `customContextMenuRequested()` is emitted.

这个属性的默认值为 `Qt::DefaultContextMenu`，这意味着 `contextMenuEvent()` 事件处理是激活状态。

Constant	Value	Description
Qt::NoContextMenu	0	the widget does not feature a context menu, context menu handling is deferred to the widget's parent. # 这个 widget 没有上下文菜单，菜单的处理归于父 widget;
Qt::PreventContextMenu	4	the widget does not feature a context menu, and in contrast to NoContextMenu, the handling is <i>not</i> deferred to the widget's parent. This means that all right mouse button events are guaranteed to be delivered to the widget itself through <code>QWidget::mousePressEvent()</code> , and <code>QWidget::mouseReleaseEvent()</code> . # 这个 widget 没有菜单，且其上下文菜单不会归于其父 widget。这就意味着所有鼠标、键盘事件都会通过 <code>QWidget::mousePressEvent()</code> 和 <code>QWidget::mouseReleaseEvent()</code> 传递给 widget 本身;
Qt::DefaultContextMenu	1	the widget's <code>QWidget::contextMenuEvent()</code> handler is called. # widget 的 <code>QWidget::contextMenuEvent()</code> 函数被调用;
Qt::ActionsContextMenu	2	the widget displays its <code>QWidget::actions()</code> as context menu. # widget 显示 <code>QWidget::actions()</code> 作为上下文菜单;
Qt::CustomContextMenu	3	the widget emits the <code>QWidget::customContextMenuRequested()</code> signal. # widget 发送 <code>QWidget::customContextMenuRequested()</code> 信号;

Access functions:

Qt::ContextMenuPolicy	<code>contextMenuPolicy()</code> const	# 查看菜单显示方式
-----------------------	--	------------

```
void setContextMenuPolicy(Qt::ContextMenuPolicy policy) # 设置菜单显示方式
```

See also [contextMenuEvent\(\)](#), [customContextMenuRequested\(\)](#), and [actions\(\)](#).

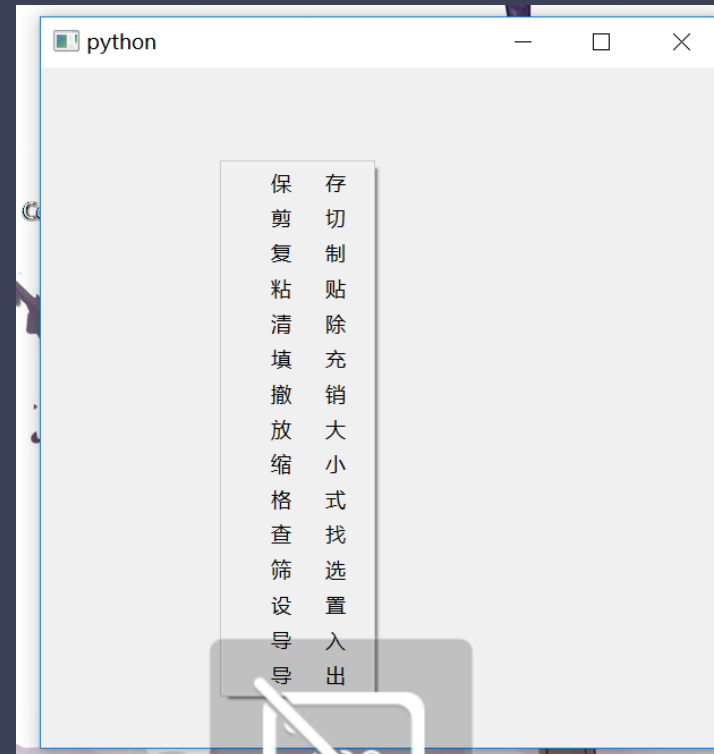
例子:

```
# !/usr/bin/env python
# -*- coding: utf-8 -*-
"""
Created on Fri Jul 6 16:04:02 2018
@author: 肖建斌
@Function: 菜单
"""

import sys
from PyQt5.QtWidgets import *
from PyQt5.QtGui import *
from PyQt5.QtCore import *

class Example(QWidget):
    def __init__(self, parent=None):
        super(Example, self).__init__(parent)
        self.initUI()
        self.show()

    def initUI(self):
        self.setGeometry(300, 300, 600, 600)
        self.setContextMenuPolicy(Qt.ActionsContextMenu) # 右键菜单模式
        name = ('保存', '剪切', '复制', '粘贴', '清除',
                '填充', '撤销', '放大', '缩小', '格式',
                '查找', '筛选', '设置', '导入', '导出')
        for i in range(len(name)):
            action = QAction((' '*6).join(name[i]), self) # 设置功能
```




```
self.addAction(action) # 加入右键菜单
```

```
if __name__ == '__main__':  
    app = QApplication(sys.argv)  
    ex = Example()  
    sys.exit(app.exec_())
```












cursor : [QCursor](#)

This property holds the cursor shape for this widget # 该属性保存鼠标指针形状

The mouse cursor will assume this shape when it's over this widget. See the [list of predefined cursor objects](#) for a range of useful shapes.

An editor widget might use an I-beam cursor:

当鼠标经过部件时鼠标指针会变成指定的形状。

```
setCursor(Qt. ArrowCursor); # 鼠标指针形状   
setCursor(Qt. UpArrowCursor); # 鼠标指针形状   
setCursor(Qt. CrossCursor); # 鼠标指针形状   
setCursor(Qt. WaitCursor); # 鼠标指针形状   
setCursor(Qt. IBeamCursor); # 鼠标指针形状   
setCursor(Qt. SizeVerCursor); # 鼠标指针形状   
setCursor(Qt. SizeHorCursor ); # 鼠标指针形状   
setCursor(Qt. SizeBDiagCursor); # 鼠标指针形状   
setCursor(Qt. SizeFDiagCursor ); # 鼠标指针形状   
setCursor(Qt. SizeAllCursor); # 鼠标指针形状   
setCursor(Qt. BlankCursor ); # 一个不可见的鼠标指针通常在需要隐藏鼠标时使用  
setCursor(Qt. SplitVCursor ); # 鼠标指针形状 
```

```
setCursor(Qt::SplitHCursor); # 鼠标指针形状
setCursor(Qt::PointingHandCursor); # 鼠标指针形状
setCursor(Qt::ForbiddenCursor); # 鼠标指针形状
setCursor(Qt::OpenHandCursor); # 鼠标指针形状
setCursor(Qt::ClosedHandCursor); # 鼠标指针形状
setCursor(Qt::WhatsThisCursor); # 鼠标指针形状
setCursor(Qt::BusyCursor); # 鼠标指针形状
setCursor(Qt::DragMoveCursor); # 在拖动物品时通常使用的光标
setCursor(Qt::DragCopyCursor); # 在拖动并复制时使用的光标
setCursor(Qt::DragLinkCursor); # 在拖拽一个项目时通常使用的光标来链接到它
setCursor(Qt::BitmapCursor); # 一个指定位图光标
```

If no cursor has been set, or after a call to `unsetCursor()`, the parent's cursor is used.
如果没有设置鼠标指针或者在 `unsetCursor()`之前设置，则子部件会使用父部件的指针形状

By default, this property contains a cursor with the `Qt::ArrowCursor` shape. # 系统默认使用 `Qt::ArrowCursor` 鼠标指针形状
Some underlying window implementations will reset the cursor if it leaves a widget even if the mouse is grabbed. If you want to have a cursor set for all widgets, even when outside the window, consider `QApplication::setOverrideCursor()`.
即使鼠标被捕获，一些底层窗口实现也可以重新设置光标。如果您想为所有小部件设置一个光标，即使在窗口之外，可以考虑使用 `QApplication::setOverrideCursor()`

Access functions:

QCursor	<code>cursor()</code> const	# 查看鼠标指针形状
void	<code>setCursor(const QCursor &)</code>	# 设置鼠标指针形状 <code>setCursor(Qt::ArrowCursor)</code>
void	<code>unsetCursor()</code>	# 取消已设置的鼠标指针形状恢复父窗口光标或默认光标

See also `QApplication::setOverrideCursor()`.

`enabled` : bool

This property holds whether the widget is enabled # 该属性确定部件是否可激活

In general an enabled widget handles keyboard and mouse events; a disabled widget does not. An exception is made with [QAbstractButton](#).
通常来说一个启用的 widget 可以处理鼠标键盘事件；禁用的 widget 却不能。QabstractButton 是一个例外。
Some widgets display themselves differently when they are disabled. For example a button might draw its label grayed out. If your widget needs to know when it becomes enabled or disabled, you can use the [changeEvent\(\)](#) with type [QEvent::EnabledChange](#).
有一些 widget 在禁用时会有不同的显示方式。例如一个 button 在禁用时显示为灰色。如果你要知道一个 widget 何时启用或禁用，你可以适用 [changeEvent\(\)](#)和 [QEvent::EnabledChange](#)。
Disabling a widget implicitly disables all its children. Enabling respectively enables all child widgets unless they have been explicitly disabled. It is not possible to explicitly enable a child widget which is not a window while its parent widget remains disabled.
禁用 widget 会隐式的禁用其所有的子 widget。启用 widget 会隐式启用其所有子 widget，除非该子 widget 有显式的禁用。在非父 widget 为非窗口时且父 widget 为禁用时，不可能显式启用其子 widget。

By default, this property is true. # 默认这个属性为 True

Access functions:

bool	isEnabled() const	# 查看激活属性
void	setEnabled(<i>bool</i>)	# 设置激活属性

See also [isEnabledTo\(\)](#), [QKeyEvent](#), [QMouseEvent](#), and [changeEvent\(\)](#).

focus : const bool

This property holds whether this widget (or its focus proxy) has the keyboard input focus.
这个属性保存 widget（或它的焦点代理）是否具有键盘输入焦点。

By default, this property is false. # 默认该属性为 False

Note: Obtaining the value of this property for a widget is effectively equivalent to checking whether [QApplication::focusWidget\(\)](#) refers to the widget.
贴士：获取这属性的值等同于检查 QApplication::focusWidget()是否连接到该 widget。

Access functions:

bool	hasFocus() const	# 查看 widget 是否获取键盘或鼠标焦点
------	------------------	-------------------------

See also [setFocus\(\)](#), [clearFocus\(\)](#), [setFocusPolicy\(\)](#), and [QApplication::focusWidget\(\)](#).

focusPolicy : Qt::FocusPolicy

This property holds the way the widget accepts keyboard focus.

这个属性保存 widget 接收键盘焦点的方式。

The policy is [Qt::TabFocus](#) if the widget accepts keyboard focus by tabbing, [Qt::ClickFocus](#) if the widget accepts focus by clicking, [Qt::StrongFocus](#) if it accepts both, and [Qt::NoFocus](#) (the default) if it does not accept focus at all.

Constant	Value	Description
Qt::TabFocus	0x1	the widget accepts focus by tabbing. # 接收键盘 Tab 焦点；
Qt::ClickFocus	0x2	the widget accepts focus by clicking. # 接收鼠标单击焦点；
Qt::StrongFocus	TabFocus ClickFocus 0x8	the widget accepts focus by both tabbing and clicking. On macOS this will also be indicate that the widget accepts tab focus when in 'Text/List focus mode'. # 接收 Tab 和鼠标单击两个焦点。；
Qt::WheelFocus	StrongFocus 0x4	like Qt::StrongFocus plus the widget accepts focus by using the mouse wheel. # 与 Qt::StrongFocus 类似，widget 接收鼠标滚轮焦点；
Qt::NoFocus	0	the widget does not accept focus. # widget 不接收焦点；默认为此模式

You must enable keyboard focus for a widget if it processes keyboard events. This is normally done from the widget's constructor. For instance, the [QLineEdit](#) constructor calls [setFocusPolicy\(Qt::StrongFocus\)](#).

如果想处理键盘事件，你必须启用一个 widget 的键盘焦点。这通常是从小部件的构造函数中完成的。例如，QLineEdit 构造函数调用 setFocusPolicy(Qt::StrongFocus)。

If the widget has a focus proxy, then the focus policy will be propagated to it.

如果小部件有一个焦点代理，那么焦点策略将被传播到它。

贴士：设置焦点可以让应用更便捷。比如当你打开百度主页或其他带有编辑框的页面时，不需要先用鼠标点击编辑框就可以直接输入文字等信息到其中。这就是由于搜索框设置了焦点。当前有焦点事件的窗口只能有一个。

Access functions:

Qt::FocusPolicy	focusPolicy() const	# 查看焦点状态
void	setFocusPolicy(Qt::FocusPolicy policy)	# 设置焦点类型

See also [focusInEvent\(\)](#), [focusOutEvent\(\)](#), [keyPressEvent\(\)](#), [keyReleaseEvent\(\)](#), and [enabled](#).

font : [QFont](#)

This property holds the font currently set for the widget. # 这个属性保存当前为 widget 设置的字体。

This property describes the widget's requested font. The font is used by the widget's style when rendering standard components, and is available as a means to ensure that custom widgets can maintain consistency with the native platform's look and feel. It's common that different platforms, or different styles, define different fonts for an application.

这个属性描述了 widget 的请求字体。widget 的样式使用该字体，并且可以作为一种方法来确保定制 widget 能够与本机平台的外观和感觉保持一致。不同的平台，或者不同的样式，为应用程序定义不同的字体是很常见的。

When you assign a new font to a widget, the properties from this font are combined with the widget's default font to form the widget's final font. You can call [fontInfo\(\)](#) to get a copy of the widget's final font. The final font is also used to initialize [QPainter](#)'s font.

当您为 widget 分配新字体时，该字体的属性与 widget 的默认字体组合在一起，形成 widget 的最终字体。你可以调用 fontInfo()来获得小部件的最终字体的拷贝。最后的字体也用于初始化 QPaint 的字体。

The default depends on the system environment. [QApplication](#) maintains a system/theme font which serves as a default for all widgets. There may also be special font defaults for certain types of widgets. You can also define default fonts for widgets yourself by passing a custom font and the name of a widget to [QApplication::setFont\(\)](#). Finally, the font is matched against Qt's font database to find the best match.

默认情况取决于系统环境。QApplication 维护一个系统/主题字体，它是所有 widget 的默认值。对于某些类型的 widget，也可能有特殊的字体默认值。您还可以通过传递自定义字体和 widget 的名称到 QApplication::setFont()来定义 widget 的默认字体。最后，字体与 Qt 的字体数据库匹配，以找到最匹配的字体。

[QWidget](#) propagates explicit font properties from parent to child. If you change a specific property on a font and assign that font to a widget, that property will propagate to all the widget's children, overriding any system defaults for that property. Note that fonts by default don't propagate to windows (see [isWindow\(\)](#)) unless the [Qt::WA_WindowPropagation](#) attribute is enabled.

QWidget 在父 widget 和子 widget 之间显式传递字体属性。 如果您改变字体上的特定属性并将该字体分配给 widget，该属性将传递到所有 widget 的子 widget，覆盖该属性的任何系统默认值。请注意，默认情况下，字体不会传递到 windows（参见 isWindow()），除非 Qt::WA_WindowPropagation 属性被启用。

QWidget's font propagation is similar to its palette propagation. # QWidget 字体属性的传递类似于颜色的传递。

The current style, which is used to render the content of all standard Qt widgets, is free to choose to use the widget font, or in some cases, to ignore it (partially, or completely). In particular, certain styles like GTK style, Mac style, and Windows Vista style, apply special modifications to the widget font to match the platform's native look and feel. Because of this, assigning properties to a widget's font is not guaranteed to change the appearance of the widget. Instead, you may choose to apply a [styleSheet](#).

当前的样式用于呈现所有标准 Qt widget 的内容，可以自由选择使用 widget 字体，或者在某些情况下，忽略它（部分或完全）。特别地，某些样式，如 GTK 样式、Mac 风格和 Windows Vista 风格，对 widget 字体进行特殊的修改，以匹配平台的本机外观和感觉。正因为如此，将属性分配给小部件的字体并不能保证改变小部件的外观。相反，您可以选择应用样式表。

Note: If [Qt Style Sheets](#) are used on the same widget as setFont(), style sheets will take precedence if the settings conflict.

贴士：如果 Qt 样式表作用于 setFont()相同的 widget，样式表属性会优先使用。

Access functions:

const QFont &	font() const	# 查看字体
void	setFont(const QFont &)	# 设置字体，QFont

See also [fontInfo\(\)](#) and [fontMetrics\(\)](#).

frameGeometry : const [QRect](#)

geometry of the widget relative to its parent including any window frame

See the [Window Geometry](#) documentation for an overview of geometry issues with windows.
By default, this property contains a value that depends on the user's platform and screen geometry.

Access functions:

QRect	frameGeometry() const
-------	-----------------------

See also [geometry\(\)](#), [x\(\)](#), [y\(\)](#), and [pos\(\)](#).

frameSize : const [QSize](#)

This property holds the size of the widget including any window frame. # 该属性保存 widget 的大小，包含任何窗口区域。

By default, this property contains a value that depends on the user's platform and screen geometry.

默认情况下，这个属性包含一个依赖于用户的平台和屏幕几何的值。

Access functions:

QSize	frameSize() const
-------	-------------------

fullScreen : const bool

This property holds whether the widget is shown in full screen mode. # 该属性保存 widget 是否在全屏模式下显示

A widget in full screen mode occupies the whole screen area and does not display window decorations, such as a title bar.

全屏模式下的 widget 占据整个屏幕区域，不显示窗口装饰，比如标题栏。

By default, this property is false. # 默认 False

Access functions:

bool	isFullScreen() const	# 设置窗口是否全屏
------	----------------------	------------

See also [windowState\(\)](#), [minimized](#), and [maximized](#).

geometry : [QRect](#)

This property holds the geometry of the widget relative to its parent and excluding the window frame.

这个属性包含 widget 的几何形状，相对于它的父 widget，并且不包括窗口框架。

When changing the geometry, the widget, if visible, receives a move event ([moveEvent\(\)](#)) and/or a resize event ([resizeEvent\(\)](#)) immediately. If the widget is not currently visible, it is guaranteed to receive appropriate events before it is shown.

当改变几何图形时，如果可见，widget 将立即接收移动事件（moveEvent()）和/或调整大小事件（resizeEvent()）。如果小部件目前不可见，它将保证在显示之前接收适当的事件。

The size component is adjusted if it lies outside the range defined by [minimumSize\(\)](#) and [maximumSize\(\)](#).

如果大小组件位于 minimumSize() 和 maximumSize()定义的范围之外，则调整大小组件。

Warning: Calling setGeometry() inside [resizeEvent\(\)](#) or [moveEvent\(\)](#) can lead to infinite recursion.

警告：在 resizeEvent() 或 moveEvent()中调用 setGeometry()可以导致无限递归。

See the [Window Geometry](#) documentation for an overview of geometry issues with windows.

请参阅 Window Geometry 文档，了解关于 windows 的几何问题的概述。

By default, this property contains a value that depends on the user's platform and screen geometry.

默认情况下，这个属性包含一个依赖于用户的平台和屏幕几何的值。

Access functions:

const QRect &	geometry() const	# 获取 widget 的几何特征，QRect
void	setGeometry (int x, int y, int w, int h)	# 设置 widget 的几何特征
void	setGeometry (const QRect &)	# 设置 widget 的几何特征

See also [frameGeometry\(\)](#), [rect\(\)](#), [move\(\)](#), [resize\(\)](#), [moveEvent\(\)](#), [resizeEvent\(\)](#), [minimumSize\(\)](#), and [maximumSize\(\)](#).

height : const int

This property holds the height of the widget excluding any window frame. # 这个属性包含 widget 的高度，不包括任何窗口框架。

See the [Window Geometry](#) documentation for an overview of geometry issues with windows.

请参阅 Window Geometry 文档，了解关于 windows 的几何问题的概述。

Note: Do not use this function to find the height of a screen on a [multiple screen desktop](#). Read [this note](#) for details.

贴士：不要使用这个函数来在多个屏幕桌面上找到屏幕的高度。请阅读这篇文章的详细内容。

By default, this property contains a value that depends on the user's platform and screen geometry.

默认情况下，这个属性包含一个依赖于用户的平台和屏幕几何的值。

Access functions:

int	height()	const	# 窗口高度
-----	----------	-------	--------

See also [geometry](#), [width](#), and [size](#).

inputMethodHints : Qt::InputMethodHints

What input method specific hints the widget has. # 具体的输入方法提示 widget 有什么。

This is only relevant for input widgets. It is used by the input method to retrieve hints as to how the input method should operate. For example, if the [Qt::ImhFormattedNumbersOnly](#) flag is set, the input method may change its visual components to reflect that only numbers can be entered.
这只会与输入 widget 有关。输入方法使用它来检索关于输入方法应该如何操作的提示。例如，如果 Qt::ImhFormattedNumbersOnly 标志被设置，输入方法可能会改变它的可视组件，以反映只能输入数字。

Warning: Some widgets require certain flags in order to work as intended. To set a flag, do w->setInputMethodHints(w->inputMethodHints()|f) instead of w->setInputMethodHints(f).
警告：有些 widget 需要特定的标志才能按预期工作。要设置一个标识，w->setInputMethodHints(w->inputMethodHints()|f)取代 w->setInputMethodHints(f)。

Note: The flags are only hints, so the particular input method implementation is free to ignore them. If you want to be sure that a certain type of characters are entered, you should also set a [QValidator](#) on the widget.
贴士：标记只是提示，因此特定的输入方法实现可以自由地忽略它们。如果您想要确保输入了某种类型的字符，您还应该在 widget 上设置 QValidator。

The default value is [Qt::ImhNone](#). # 默认为 Qt::ImhNone
This property was introduced in Qt 4.6. # 这个属性是在 Qt 4.6 中引入的。

Constant	Value	Description
Qt::ImhNone	0x1	hints. #不配置，系统默认

Qt::ImhHiddenText	0x1	<p>The input method should not show the characters while typing. This is automatically set when setting QLineEdit::echoMode to Password. Note that setting ImhHiddenText does not change the echo mode.</p> <p># 输入法在输入后不显示字符。这个当 QLineEdit::echoMode 设置为 Password 是自动设置。注意，设置 ImhHiddenText 并不改变返回模式；</p>
Qt::ImhSensitiveData	0x2	<p>Typed text should not be stored by the active input method in any persistent storage like predictive user dictionary.</p> <p># 这个类型不会被任何活动输入方式存储起来，例如预测用户行为的字典；</p>
Qt::ImhNoAutoUppercase	0x4	<p>The input method should not try to automatically switch to upper case when a sentence ends.</p> <p># 当一个句子结束后，输入法不会自动切换到大写字母；</p>
Qt::ImhPreferNumbers	0x8	<p>Numbers are preferred (but not required).</p> <p># 更适合输入数字(但并不强制)；</p>
Qt::ImhPreferUppercase	0x10	<p>Upper case letters are preferred (but not required).</p> <p># 更适合输入大写字母(但并不强制)；</p>
Qt::ImhPreferLowercase	0x20	<p>Lower case letters are preferred (but not required).</p> <p># 更适合输入小写字母(但并不强制)；</p>
Qt::ImhNoPredictiveText	0x40	<p>Do not use predictive text (i.e. dictionary lookup) while typing.</p> <p># 在输入时不要使用预测性文本（例如字典查找）；</p>
Qt::ImhDate	0x80	<p>The text editor functions as a date field.</p> <p># 日期类文本编辑功能；</p>
Qt::ImhTime	0x100	<p>The text editor functions as a time field.</p> <p># 时间类文本编辑功能；</p>
Qt::ImhPreferLatin	0x200	<p>Latin characters are preferred (but not required).</p> <p># 更适合输入拉丁字母(但并不强制)；</p>

Qt::ImhMultiLine	0x400	Multiple lines can be entered into the text field. # 输入的文本带有下划线;
Qt::ImhNoEditMenu	0x800	Do not use built-in edit menu. This flag was introduced in Qt 5.11. # 不要构建编辑菜单。这个标志的介绍在 Qt 5.11 中;
Qt::ImhNoTextHandles	0x1000	Do not use built-in text cursor and selection handles. This flag was introduced in Qt 5.11. # 不要构建 text cursor 和 selection handles。这个介绍在 Qt 5.11 中;

Access functions:

Qt::InputMethodHints	inputMethod() const	# 查看输入提示设置
void	setInputMethodHints(Qt::InputMethodHints hints)	# 设置输入提示

See also [inputMethodQuery\(\)](#).

isActiveWindow : const bool

This property holds whether this widget's window is the active window. # 这个属性这个表示 widget 的窗口是否是活动窗口。

The active window is the window that contains the widget that has keyboard focus (The window may still have focus if it has no widgets or none of its widgets accepts keyboard focus).

活动窗口是包含有键盘焦点的 widget 的窗口（如果窗口没有 widget，或者它的 widget 没有接受键盘焦点，窗口可能仍然会有焦点）。

When popup windows are visible, this property is true for both the active window *and* for the popup.

当弹出窗口可见时，这个属性对于活动窗口和弹出窗口是等价的;

By default, this property is false. # 默认这个属性为 False

Access functions:

bool	isActiveWindow() const	# 指定是否为活动窗口
------	------------------------	-------------

See also [activateWindow\(\)](#) and [QApplication::activeWindow\(\)](#).

layoutDirection : Qt::LayoutDirection

This property holds the layout direction for this widget. # 该属性保存 widget 的布局方向。

By default, this property is set to [Qt::LeftToRight](#). # 默认该属性为 Qt::LeftToRight。

When the layout direction is set on a widget, it will propagate to the widget's children, but not to a child that is a window and not to a child for which `setLayoutDirection()` has been explicitly called. Also, child widgets added *after* `setLayoutDirection()` has been called for the parent do not inherit the parent's layout direction.

当在 widget 上设置布局方向是，该属性会传递给所有子 widget 上，如果该 widget 是窗口或者显式调用了 `setLayoutDirection()` 不会首到影响。此外，子 widget 的 `setLayoutDirection()` 也被调用。

This method no longer affects text layout direction since Qt 4.7. # 自 Qt4.7 以来这种方法不再影响文本布局方向。

Constant	Value	Description
Qt::LeftToRight	0	Left-to-right layout. # 从左到右布局
Qt::RightToLeft	1	Right-to-left layout. # 从右到左布局
Qt::LayoutDirectionAuto	2	Automatic layout. # 依据内容自动布局

Access functions:

Qt::LayoutDirection	<code>layoutDirection() const</code>	# 查看布局方向
void	<code>setLayoutDirection(Qt::LayoutDirection direction)</code>	# 设置布局方向
void	<code>unsetLayoutDirection()</code>	# 清除布局方向

See also [QApplication::layoutDirection](#).

locale : [QLocale](#)

This property holds the widget's locale # 该属性保存 widget 的地区信息。
用于定义用户语言、国家（或地区）以及其他在用户界面中可见的与语言和国家相关的特性，比如日期表示、货币表示等；

As long as no special locale has been set, this is either the parent's locale or (if this widget is a top level widget), the default locale.

If the widget displays dates or numbers, these should be formatted using the widget's locale.
只要没有设置特定的场所，这要么是父语言环境，要么是（如果这个 widget 是顶层 widget），则是缺省语言环境。如果 widget 显示日期或数字，则应该使用 widget 的地区信息进行格式化。

This property was introduced in Qt 4.3. # 该属性在 Qt 4.3 中引进

Access functions:

QLocale	locale() const	# 查看地区属性
void	setLocale(const QLocale &locale)	# 设置地区属性
void	unsetLocale()	# 清除地区属性

See also [QLocale](#) and [QLocale::setDefault\(\)](#).

maximized : const bool

This property holds whether this widget is maximized. # 该属性指定 widget 是否被最大化。

This property is only relevant for windows. # 该属性仅对窗口有效。

Note: Due to limitations on some window systems, this does not always report the expected results (e.g., if the user on X11 maximizes the window via the window manager, Qt has no way of distinguishing this from any other resize). This is expected to improve as window manager protocols evolve.

贴士：由于某些窗口系统的限制，这并不总是返回预期的结果（例如，如果 X11 的用户通过窗口管理器最大化窗口，Qt 无法将其与其他大小的调整大小区分开来）。随着窗口管理器协议的发展，预计这将会改进。

By default, this property is false. # 该属性默认为 False。

Access functions:

bool	isMaximized() const	# 指定是否被最大化
------	---------------------	------------

See also [windowState\(\)](#), [showMaximized\(\)](#), [visible](#), [show\(\)](#), [hide\(\)](#), [showNormal\(\)](#), and [minimized](#).

maximumHeight : int

This property holds the widget's maximum height in pixels # 该属性保存 widget 最大高度的像素值。

This property corresponds to the height held by the [maximumSize](#) property. # 该属性对应于 maximumSize 属性所持有的高度。
By default, this property contains a value of 16777215. # 该属性默认值为 16777215

Note: The definition of the QWIDGETSIZE_MAX macro limits the maximum size of widgets.

贴士：QWIDGETSIZE_MAX 宏的定义限制了 widget 的最大尺寸

Access functions:

int	maximumHeight() const	# 查看最大高度
void	setMaximumHeight(int maxh)	# 设置最大高度

See also [maximumSize](#) and [maximumWidth](#).

maximumSize : [QSize](#)

This property holds the widget's maximum size in pixels # 该属性保存 widget 最大尺寸的像素值

The widget cannot be resized to a larger size than the maximum widget size. # widget 不能将大小调整的超过最大大小

By default, this property contains a size in which both width and height have values of 16777215. # 默认值为 QSize(16777215, 16777215)。

Note: The definition of the QWIDGETSIZE_MAX macro limits the maximum size of widgets.

贴士: QWIDGETSIZE_MAX 宏的定义限制了 widget 的最大尺寸

Access functions:

QSize	maximumSize() const	# 查看最大尺寸
void	setMaximumSize(const QSize &)	# 设置最大尺寸, QSize()
void	setMaximumSize(int maxw, int maxh)	# 设置最大尺寸, wigth, heighth

See also [maximumWidth](#), [maximumHeight](#), [minimumSize](#), and [sizeIncrement](#).

maximumWidth : int

This property holds the widget's maximum width in pixels # 该属性保存 widget 最大宽度的像素值。

This property corresponds to the width held by the [maximumSize](#) property. # 该属性对应于 maximumSize 属性所持有的宽度度。
By default, this property contains a value of 16777215. # 该属性默认值为 16777215。

Note: The definition of the QWIDGETSIZE_MAX macro limits the maximum size of widgets.

贴士: QWIDGETSIZE_MAX 宏的定义限制了 widget 的最大尺寸

Access functions:

int	maximumWidth() const	# 查看最大宽度
void	setMaximumWidth(int maxw)	# 设置最大宽度

See also [maximumSize](#) and [maximumHeight](#).

minimized : const bool

This property holds whether this widget is minimized (iconified) # 该属性指定 widget 是否被最小化。

This property is only relevant for windows. # 该属性仅对窗口有效。

By default, this property is false. # 默认值为 False

Access functions:

bool	isMinimized() const	# 指定是否最小化
------	---------------------	-----------

See also [showMinimized\(\)](#), [visible](#), [show\(\)](#), [hide\(\)](#), [showNormal\(\)](#), and [maximized](#).

minimumHeight : int

This property holds the widget's minimum height in pixels # 该属性保存 widget 最小高度的像素值

This property corresponds to the height held by the [minimumSize](#) property. # 该属性对应于 minimumSize 属性所持有的高度。

By default, this property has a value of 0. # 该属性默认为 0

Access functions:

int	minimumHeight() const	# 查看最小高度
-----	-----------------------	----------

void	setMinimumHeight(int <i>minh</i>)	# 设置最小高度
------	------------------------------------	----------

See also [minimumSize](#) and [minimumWidth](#).

minimumSize : QSize

This property holds the widget's minimum size # 该属性指定 widget 的最小尺寸。

The widget cannot be resized to a smaller size than the minimum widget size. The widget's size is forced to the minimum size if the current size is smaller.

widget 不能将大小调整的超过最小尺寸。如果当前尺寸比最小尺寸更小，widget 的尺寸将强制调整到最小尺寸。

The minimum size set by this function will override the minimum size defined by [QLayout](#). In order to unset the minimum size, use a value of `QSize(0, 0)`.

由这个函数设置的最小尺寸将覆盖 [QLayout](#) 定义的最小尺寸。为了取消最小尺寸，使用 `QSize(0, 0)` 的值。

By default, this property contains a size with zero width and height. # 默认 `QSize(0, 0)`

Access functions:

QSize	<code>minimumSize() const</code>	# 查看最小尺寸
void	<code>setMinimumSize(const QSize &)</code>	# 设置小尺寸, <code>QSize()</code>
void	<code>setMinimumSize(int minw, int minh)</code>	# 设置最小尺寸, <code>wigth, heighth</code>

See also [minimumWidth](#), [minimumHeight](#), [maximumSize](#), and [sizeIncrement](#).

minimumSizeHint : const [QSize](#)

This property holds the recommended minimum size for the widget # 该属性保存 widget 的推荐最小尺寸

If the value of this property is an invalid size, no minimum size is recommended. # 如果该属性的值为无效大小，则不建议使用最小尺寸

The default implementation of `minimumSizeHint()` returns an invalid size if there is no layout for this widget, and returns the layout's minimum size otherwise. Most built-in widgets reimplement `minimumSizeHint()`.

如果这个 widget 没有布局，那么 `minimumSizeHint()` 的默认实现会返回无效大小，否则返回布局的最小尺寸。大多数内置小部件重新实现 `minimumSizeHint()`。

[QLayout](#) will never resize a widget to a size smaller than the minimum size hint unless [minimumSize\(\)](#) is set or the size policy is set to `QSizePolicy::Ignore`. If [minimumSize\(\)](#) is set, the minimum size hint will be ignored.

QLayout 绝不会设置 widget 的尺寸小于最小推荐尺寸除非 `minimumSize()` 或者将尺寸策略设置为 `QSizePolicy::Ignore`。如果设置了 `minimumSize()`，那么最小大小提示将被忽略。

Access functions:

virtual QSize	<code>minimumSizeHint()</code> const	# 设置最小推荐大小
---------------	--------------------------------------	------------

See also [QSize::isValid\(\)](#), [resize\(\)](#), [setMinimumSize\(\)](#), and [sizePolicy\(\)](#).

minimumWidth : int

This property holds the widget's minimum width in pixels # 该属性保存 widget 最小宽度的像素值。

This property corresponds to the width held by the [minimumSize](#) property. # 该属性对应于 `minimumSize` 属性所持有的宽度。
By default, this property has a value of 0. # 该属性默认值为 0

Access functions:

int	<code>minimumWidth()</code> const	# 查看最小宽度
void	<code>setMinimumWidth(int minw)</code>	# 设置最小宽度

See also [minimumSize](#) and [minimumHeight](#).

modal : const bool

This property holds whether the widget is a modal widget # 该属性指定 widget 是否为模态 widget

This property only makes sense for windows. A modal widget prevents widgets in all other windows from getting any input.
该属性仅对窗口有效(`parent=NULL`)。模态 widget 会阻止其它 widget 获取输入信息。

By default, this property is false. # 默认为 False

Access functions:

bool	<code>isModal()</code> const	# 指定是否为模态
------	------------------------------	-----------

See also [isWindow\(\)](#), [windowModality](#), and [QDialog](#).

mouseTracking : bool

This property holds whether mouse tracking is enabled for the widget # 该属性指定是否为 widget 启用鼠标追踪

If mouse tracking is disabled (the default), the widget only receives mouse move events when at least one mouse button is pressed while the mouse is being moved.

如果鼠标追踪是禁用的(默认), 当鼠标至少有一个键被按下且移动时, widget 只会接收鼠标移动事件

If mouse tracking is enabled, the widget receives mouse move events even if no buttons are pressed.

如果启用了鼠标追踪, 即使没有按键 widget 也会接收鼠标移动事件。

Access functions:

bool	<code>hasMouseTracking() const</code>	# 获取属性值
void	<code>setMouseTracking(bool enable)</code>	# 设置属性值

See also [mouseMoveEvent\(\)](#).

normalGeometry : const QRect

This property holds the geometry of the widget as it will appear when shown as a normal (not maximized or full screen) top-level widget

该属性保存 widget 的为正常(非最大或全屏)顶层窗口时的几何形状

For child widgets this property always holds an empty rectangle. # 对子 widget 该属性通常为空的矩形

By default, this property contains an empty rectangle. # 默认此属性值为空的矩形

Access functions:

QRect	<code>normalGeometry() const</code>	# 顶层窗口正常状态的 QRect()
-------	-------------------------------------	---------------------

See also [QWidget::windowState\(\)](#) and [QWidget::geometry](#).

palette : [QPalette](#)

This property holds the widget's palette # 该属性保存 widget 的调色板

This property describes the widget's palette. The palette is used by the widget's style when rendering standard components, and is available as a means to ensure that custom widgets can maintain consistency with the native platform's look and feel. It's common that different platforms, or different styles, have different palettes.

这个属性描述 widget 的调色板。面板在呈现标准组件时被 widget 的样式使用，并且可以作为一种方法来确保定制 widget 能够与本机平台的外观和感觉保持一致。不同的平台，或不同的风格，有不同的调色板，这是很常见的。

When you assign a new palette to a widget, the color roles from this palette are combined with the widget's default palette to form the widget's final palette. The palette entry for the widget's background role is used to fill the widget's background (see [QWidget::autoFillBackground](#)), and the foreground role initializes [QPainter](#)'s pen.

当您为 widget 分配一个新调色板时，这个调色板中的颜色角色将与 widget 的默认调色板结合起来，形成 widget 的最终面板。widget 的后台角色的调色板条目被用来填充 widget 的背景（参见 [QWidget::autoFillBackground](#)），前景角色初始化 QPainter 画笔。

The default depends on the system environment. [QApplication](#) maintains a system/theme palette which serves as a default for all widgets. There may also be special palette defaults for certain types of widgets (e.g., on Windows Vista, all classes that derive from [QMenuBar](#) have a special default palette). You can also define default palettes for widgets yourself by passing a custom palette and the name of a widget to [QApplication::setPalette\(\)](#). Finally, the style always has the option of polishing the palette as it's assigned (see [QStyle::polish\(\)](#)).

默认情况取决于系统环境。QApplication 维护一个系统/主题调色板，它是所有 widget 的默认值。对于某些类型的 widget，也可能有特殊的调色板默认值（例如，在 Windows Vista 上，从 QMenuBar 派生的所有类都有一个特殊的默认调色板）。您还可以通过将一个定制调色板和一个 widget 的名称传递给 QApplication::setPalette() 来定义 widget 的默认调色板。最后，风格总是可以选择在指定的调色板上进行修正（参见 [QStyle::polish\(\)](#)）。

[QWidget](#) propagates explicit palette roles from parent to child. If you assign a brush or color to a specific role on a palette and assign that palette to a widget, that role will propagate to all the widget's children, overriding any system defaults for that role. Note that palettes by default don't propagate to windows (see [isWindow\(\)](#)) unless the [Qt::WA_WindowPropagation](#) attribute is enabled.

QWidget 从父 widget 显式传递调色板角色到子 widget。如果您将画笔或颜色分配给调色板上的特定角色，并将该调色板分配给 widget，该角色将传播到所有 widget 的子节点，覆盖该角色的任何系统默认值。注意，默认情况下，调色板不会传播到 windows（参见 [isWindow\(\)](#)），除非 Qt::WA_WindowPropagation 属性被启用。

[QWidget](#)'s palette propagation is similar to its font propagation. # QWidget 调色板属性的传递类似于字体属性的传递。

The current style, which is used to render the content of all standard Qt widgets, is free to choose colors and brushes from the widget palette, or in some cases, to ignore the palette (partially, or completely). In particular, certain styles like GTK style, Mac style, and Windows Vista style, depend on third party APIs to render the content of widgets, and these styles typically do not follow the palette. Because of this, assigning roles to a widget's palette is not guaranteed to change the appearance of the widget. Instead, you may choose to apply a [styleSheet](#). You can refer to our Knowledge Base article [here](#) for more information.

当前的样式用于呈现所有标准 Qt widget 的内容，可以自由地从 widget 调色板中选择颜色和画笔，或者在某些情况下，忽略调色板（部分或完全）。特别是，某些样式，如 GTK 样式、Mac 风格和 Windows Vista 风格，依赖于第三方 API 来呈现小部件的内容，而这些样式通常不遵循调色板。正因为如此，将角色分配给 widget 的面板并不能保证改变小部件的外观。相反，您可以选择应用样式表。您可以参考我们的知识库文章获取更多信息。

Warning: Do not use this function in conjunction with [Qt Style Sheets](#). When using style sheets, the palette of a widget can be customized using the "color", "background-color", "selection-color", "selection-background-color" and "alternate-background-color".

警告：不要在使用 Qt Style Sheets 时使用该功能。当使用样式表时，widget 调试板可以使用"color", "background-color", "selection-color", "selection-background-color" 和 "alternate-background-color"。

Access functions:

const QPalette &	palette() const	# 查看调色板
void	setPalette(const QPalette &)	# 设置调色板

See also [QApplication::palette\(\)](#) and [QWidget::font\(\)](#).

pos : [QPoint](#)

This property holds the position of the widget within its parent widget # 该属性保存 widget 的在其父 widget 的位置

If the widget is a window, the position is that of the widget on the desktop, including its frame.

如果该 widget 是窗口，那就是 widget 在桌面上的位置。

When changing the position, the widget, if visible, receives a move event ([moveEvent\(\)](#)) immediately. If the widget is not currently visible, it is guaranteed to receive an event before it is shown.

当 widget 的相对位置改变(widget 可见)，widget 会立即收到一个移动事件（moveEvent()）。如果这个 widget 目前还不可见，那么它将被保证在显示之前接收移动事件。

By default, this property contains a position that refers to the origin. # 默认位置在原点

Warning: Calling `move()` or `setGeometry()` inside `moveEvent()` can lead to infinite recursion.

警告：在 `moveEvent()` 中调用 `move()` 或 `setGeometry()` 将造成无限递归。

See the [Window Geometry](#) documentation for an overview of geometry issues with windows.

请参阅 [Window Geometry](#) 文档，了解关于 windows 的几何问题的概述。

Access functions:

QPoint	<code>pos() const</code>	# 查看位置属性
void	<code>move(int x, int y)</code>	# 移动位置, x y
void	<code>move(const QPoint &)</code>	# 移动位置, QPoint()

See also [frameGeometry](#), [size](#), [x\(\)](#), and [y\(\)](#).

rect : const [QRect](#)

This property holds the internal geometry of the widget excluding any window frame

这个属性包含 widget 的内部几何形状，不包括任何窗口框架

The rect property equals `QRect(0, 0, width\(\), height\(\))`. # 这个 rect 相当于 `QRect(0, 0, width\(\), height\(\))`。

See the [Window Geometry](#) documentation for an overview of geometry issues with windows.

请参阅 [Window Geometry](#) 文档，了解关于 windows 的几何问题的概述。

By default, this property contains a value that depends on the user's platform and screen geometry. # 该属性默认值依赖于平台和屏幕尺寸

Access functions:

QRect	<code>rect() const</code>	# 查看 widget 的内部几何形状
-------	---------------------------	---------------------

See also [size](#).

size : [QSize](#)

This property holds the size of the widget excluding any window frame # 这个属性包含 widget 的尺寸，不包括任何窗口框架

If the widget is visible when it is being resized, it receives a resize event ([resizeEvent\(\)](#)) immediately. If the widget is not currently visible, it is guaranteed to receive an event before it is shown.

如果 widget 是可见的，当其大小调整使 widget 将会立即收到一个调整事件（[resizeEvent\(\)](#)）。如果 widget 不可见，将会在可见前收到该事件。

The size is adjusted if it lies outside the range defined by [minimumSize\(\)](#) and [maximumSize\(\)](#).

如果尺寸超出 [minimumSize\(\)](#) 和 [maximumSize\(\)](#)定义的范围，将会自动调整。

By default, this property contains a value that depends on the user's platform and screen geometry. # 该属性默认值依赖于平台和屏幕尺寸

Warning: Calling [resize\(\)](#) or [setGeometry\(\)](#) inside [resizeEvent\(\)](#) can lead to infinite recursion.

警告：在 [resizeEvent\(\)](#)中调用 [resize\(\)](#) 或 [setGeometry\(\)](#)将会导致无线循环。

Note: Setting the size to `QSize(0, 0)` will cause the widget to not appear on screen. This also applies to windows.

贴士：将大小设置为 `QSize(0, 0)`将导致 widget 不会出现在屏幕上。这也适用于 windows 系统。

Access functions:

QSize	size() const	# 查看尺寸
void	resize (int <i>w</i> , int <i>h</i>)	# 调整尺寸，(w, h)
void	resize (const <i>QSize</i> &)	# 调整尺寸， QSize()

See also [pos](#), [geometry](#), [minimumSize](#), [maximumSize](#), [resizeEvent\(\)](#), and [adjustSize\(\)](#).

sizeHint : const [QSize](#)

This property holds the recommended size for the widget # 该属性保存 widget 的推荐尺寸

If the value of this property is an invalid size, no size is recommended. # 如果该属性是无效的，将不会有推荐尺寸

The default implementation of [sizeHint\(\)](#) returns an invalid size if there is no layout for this widget, and returns the layout's preferred size otherwise.

如果这个 widget 没有布局的话，[sizeHint\(\)](#)的默认实现返回无效的大小，否则返回布局的首选大小。

Access functions:

virtual QSize	sizeHint() const	# 查看属性值
---------------	------------------	---------

See also [QSize::isValid\(\)](#), [minimumSizeHint\(\)](#), [sizePolicy\(\)](#), [setMinimumSize\(\)](#), and [updateGeometry\(\)](#).

sizeIncrement : [QSize](#)

This property holds the size increment of the widget # 该属性保存 widget 的尺寸增量

When the user resizes the window, the size will move in steps of `sizeIncrement().width()` pixels horizontally and `sizeIncrement().height()` pixels vertically, with [baseSize\(\)](#) as the basis. Preferred widget sizes are for non-negative integers *i* and *j*:

当用户对窗口进行调整时，大小将在 `sizeIncrement().width()` 像素水平和 `sizeIncrement().height()` 像素垂直，以 `baseSize()` 为基础。首选的 widget 大小用于非负整数 *i* 和 *j*:

```
width = baseSize().width() + i * sizeIncrement().width();
height = baseSize().height() + j * sizeIncrement().height();
```

Note that while you can set the size increment for all widgets, it only affects windows.

注意，虽然您可以为所有小部件设置大小增量，但它只会影响窗口。

By default, this property contains a size with zero width and height. # 默认职位 width=0, height=0

Warning: The size increment has no effect under Windows, and may be disregarded by the window manager on X11.

警告：大小增量在 Windows 下没有影响，并且可能会被窗口管理器在 X11 上忽略。

Access functions:

QSize	sizeIncrement() const	# 查看尺寸增量
void	setSizeIncrement(const QSize &)	# 设置尺寸增量, QSize()
void	setSizeIncrement(int w, int h)	# 设置尺寸增量, (w, h)

See also [size](#), [minimumSize](#), and [maximumSize](#).

sizePolicy : [QSizePolicy](#)

This property holds the default layout behavior of the widget # 该属性持有 widget 的默认布局行为

If there is a [QLayout](#) that manages this widget's children, the size policy specified by that layout is used. If there is no such [QLayout](#), the result of this function is used.

如果有一个 [QLayout](#) 管理这个子 widget，那么使用该布局所指定的大小策略。如果没有这样的 [QLayout](#)，则使用该函数的结果。

The default policy is Preferred/Preferred, which means that the widget can be freely resized, but prefers to be the size [sizeHint\(\)](#) returns. Button-like widgets set the size policy to specify that they may stretch horizontally, but are fixed vertically. The same applies to lineedit controls (such as [QLineEdit](#), [QSpinBox](#) or an editable [QComboBox](#)) and other horizontally orientated widgets (such as [QProgressBar](#)). [QToolButton](#)'s are normally square, so they allow growth in both directions. Widgets that support different directions (such as [QSlider](#), [QScrollBar](#) or [QHeader](#)) specify stretching in the respective direction only. Widgets that can provide scroll bars (usually subclasses of [QScrollArea](#)) tend to specify that they can use additional space, and that they can make do with less than [sizeHint\(\)](#).

默认策略是首选/首选项，这意味着 widget 可以自由调整，但更喜欢大小为 [sizeHint\(\)](#) 返回。按钮类的 widget 设置大小策略，以指定它们可以水平伸缩，但垂直固定。这同样适用于行编辑控件（如 [QLineEdit](#)、[QSpinBox](#) 或可编辑的 [QComboBox](#)）和其他水平导向的 widget（如 [QProgressBar](#)）。[QToolButton](#) 通常是方形的，所以它们允许两个方向的增长。支持不同方向的 widget（如 [QSlider](#)、[QScrollBar](#) 或 [QHeader](#)）只在各自的方向上指定拉伸可以提供滚动条（通常是 [QScrollArea](#) 的子类）的 widget 倾向于指定它们可以使用额外的空间，并且它们可以用小于 [sizeHint\(\)](#) 的方式来完成。

Access functions:

QSizePolicy	sizePolicy() const	# 查看属性
void	setSizePolicy(QSizePolicy)	# 设置属性
void	setSizePolicy (QSizePolicy::Policy <i>horizontal</i> , QSizePolicy::Policy <i>vertical</i>)	# 设置属性

See also [sizeHint\(\)](#), [QLayout](#), [QSizePolicy](#), and [updateGeometry\(\)](#).

statusTip : [QString](#)

This property holds the widget's status tip # 这个属性保存 widget 的状态提示

By default, this property contains an empty string. # 默认该属性为空

Access functions:

QString	statusTip() const	# 查看状态提示
---------	-----------------------------------	----------

void	<code>setStatusTip(const QString &)</code>	# 设置状态提示
------	--	----------

See also [toolTip](#) and [whatsThis](#).

styleSheet : [QString](#)

This property holds the widget's style sheet # 该属性保存 widget 样式表

The style sheet contains a textual description of customizations to the widget's style, as described in the [Qt Style Sheets](#) document. Since Qt 4.5, Qt style sheets fully supports [macOS](#).

样式表包含对小部件样式的定制的文本描述，如 [Qt Style Sheets](#) 文档中所描述的那样。自从 Qt 4.5，Qt 样式表完全支持 macOS。

Warning: Qt style sheets are currently not supported for custom [QStyle](#) subclasses. We plan to address this in some future release.

警告：Qt 样式表目前不支持定制 [QStyle](#) 子类。我们计划在未来的版本中解决这个问题。

This property was introduced in Qt 4.2. # 这个属性是在 Qt 4.2 中引入的。

Access functions:

QString	<code>styleSheet()</code> const	# 查看属性
void	<code>setStyleSheet(const QString &styleSheet)</code>	# 设置样式表

See also [setStyle\(\)](#), [QApplication::styleSheet](#), and [Qt Style Sheets](#).

tabletTracking : bool

This property holds whether tablet tracking is enabled for the widget # 该属性保存 widget 是否支持平板追踪

If tablet tracking is disabled (the default), the widget only receives tablet move events when the stylus is in contact with the tablet, or at least one stylus button is pressed, while the stylus is being moved.

如果平板电脑追踪功能被禁用（默认），当触控笔与平板电脑接触时，widget 只会收到平板电脑的移动事件，或者至少有一个触控笔按钮被按下，而触控笔正在被移动。

If tablet tracking is enabled, the widget receives tablet move events even while hovering in proximity. This is useful for monitoring position as well as the auxiliary properties such as rotation and tilt, and providing feedback in the UI.

如果启用了平板电脑跟踪功能，widget 就会在接近的时候接收到平板电脑的移动事件。这对于监视位置和辅助属性，如旋转和倾斜，以及在 UI 中提供反馈是很有用的。

This property was introduced in Qt 5.9. # 该属性在 Qt 5.9 中说明

Access functions:

bool	hasTabletTracking() const	# 获取属性
void	setTabletTracking(bool enable)	# 设置属性

See also [tabletEvent\(\)](#).

toolTip : [QString](#)

This property holds the widget's tooltip # 该属性保存 widget 的提示信息

Note that by default tooltips are only shown for widgets that are children of the active window. You can change this behavior by setting the attribute [Qt::WA_AlwaysShowToolTips](#) on the *window*, not on the widget with the tooltip.

注意，默认的工具提示只显示在活跃窗口的子 widget 中。你可以通过设置属性 Qt::WA_AlwaysShowToolTips 在窗口中显示工具提示来改变这种行为，而不是在工具提示的 widget 上。

If you want to control a tooltip's behavior, you can intercept the [event\(\)](#) function and catch the [QEvent::ToolTip](#) event (e.g., if you want to customize the area for which the tooltip should be shown).

如果您想要控制工具提示的行为，您可以拦截 event()函数并捕获 QEvent::ToolTip 事件（例如，如果您想要定制工具提示应该显示的区域）。

By default, this property contains an empty string. # 默认该属性为空字符

Access functions:

QString	toolTip() const	# 查看小贴士
void	setToolTip(const QString &)	# 设置小贴士

See also [QToolTip](#), [statusTip](#), and [whatsThis](#).

toolTipDuration : int

This property holds the widget's tooltip duration # 小贴士持续时长

Specifies how long time the tooltip will be displayed, in milliseconds. If the value is -1 (default) the duration is calculated depending on the length of the tooltip.

指定工具提示将在内显示多长时间（毫秒数）。如果值是-1（默认值），则根据工具提示的长度计算持续时间。

This property was introduced in Qt 5.2. # 该属性在 Qt 5.2 中介绍

Access functions:

int	toolTipDuration() const	# 查看贴士出现时长
void	setToolTipDuration(int msec)	# 设置贴士出现毫秒数

See also [toolTip](#).

updatesEnabled : bool

This property holds whether updates are enabled # 该属性保证 widget 是否可改变

An updates enabled widget receives paint events and has a system background; a disabled widget does not. This also implies that calling [update\(\)](#) and [repaint\(\)](#) has no effect if updates are disabled.

启用更新的 widget 接收绘画事件并具有系统背景；禁用的 widget 不会。这也意味着，如果更新被禁用，调用 update()和 repaint()没有效果。

By default, this property is true. # 默认为 True

setUpdatesEnabled() is normally used to disable updates for a short period of time, for instance to avoid screen flicker during large changes. In Qt, widgets normally do not generate screen flicker, but on X11 the server might erase regions on the screen when widgets get hidden before they can be replaced by other widgets. Disabling updates solves this.

启用 setUpdatesEnabled()通常用于在短时间内禁用更新，例如在大的更改期间避免屏幕闪烁。在 Qt 中，widget 通常不会产生屏幕闪烁，但是在 X11 上，当小部件被隐藏起来之前，服务器可能会擦除屏幕上的区域，然后它们就可以被其他小部件替换掉。禁用更新解决了这个。

Example:

```
setUpdatesEnabled(false);
bigVisualChanges();
setUpdatesEnabled(true);
```

Disabling a widget implicitly disables all its children. Enabling a widget enables all child widgets *except* top-level widgets or those that have been explicitly disabled. Re-enabling updates implicitly calls [update\(\)](#) on the widget.

禁用 widget 会隐式地禁用它的所有子 widget。启用 widget 可以支持除顶级 widget 或已显式禁用的所有 widget。重新启用更新在 widget 上隐式地调用 update()。

Access functions:

bool	updatesEnabled() const	# 查看属性
void	setUpdatesEnabled(bool enable)	# 设置属性

See also [paintEvent\(\)](#).

visible : bool

This property holds whether the widget is visible # 该属性保存 widget 是否可见

Calling setVisible(true) or [show\(\)](#) sets the widget to visible status if all its parent widgets up to the window are visible. If an ancestor is not visible, the widget won't become visible until all its ancestors are shown. If its size or position has changed, Qt guarantees that a widget gets move and resize events just before it is shown. If the widget has not been resized yet, Qt will adjust the widget's size to a useful default using [adjustSize\(\)](#). Calling setVisible(false) or [hide\(\)](#) hides a widget explicitly. An explicitly hidden widget will never become visible, even if all its ancestors become visible, unless you show it.

如果 widget 的所有父 widget 都在窗口可见，调用 setVisible(True)或 show()将 widget 设置为可见状态。如果一个祖 widget 是不可见的，那么这个 widget 在所有的祖 widget 都显示出来之前是不会被看到的。如果它的大小或位置发生了变化，Qt 保证 widget 在显示之前移动并调整大小。如果 widget 还没有设置大小，Qt 将调用 adjustSize()调整 widget 的大小到有用的默认值。调用 setVisible(False) 或 hide()显式隐藏一个小部件。一个显式隐藏的 widget 永远不会被看到，即使它的所有祖先都是可见的，除非你显式使其可见的它。

A widget receives show and hide events when its visibility status changes. Between a hide and a show event, there is no need to waste CPU cycles preparing or displaying information to the user. A video application, for example, might simply stop generating new frames.

当它的可见性状态发生变化时，**widget** 接收显示状态改变事件。在隐藏和显示事件之间，不需要浪费 CPU 周期来准备或向用户显示信息。例如，一个视频应用程序可能会停止生成新的帧。

A widget that happens to be obscured by other windows on the screen is considered to be visible. The same applies to iconified windows and windows that exist on another virtual desktop (on platforms that support this concept). A widget receives spontaneous show and hide events when its mapping status is changed by the window system, e.g. a spontaneous hide event when the user minimizes the window, and a spontaneous show event when the window is restored again.

一个 **widget** 碰巧被屏幕上的其他窗口所遮挡，被认为是可见的。这同样适用于在另一个虚拟桌面（虚拟机）的窗口和最小化窗口。当窗口系统改变了它的映射状态时，**widget** 会接收自发的显示/隐藏改变事件，例如当用户最小化窗口时自动隐藏事件，以及当窗口恢复时的自发显示事件。

You almost never have to reimplement the `setVisible()` function. If you need to change some settings before a widget is shown, use [showEvent\(\)](#) instead. If you need to do some delayed initialization use the Polish event delivered to the [event\(\)](#) function.

您几乎不需要重写 `setVisible()` 函数。如果您需要在小部件显示之前更改一些设置，则重写 `showEvent()` 来代替。如果您需要进行一些延迟的初始化，那么将使用 `Polish` 事件交付给 `event()` 函数处理。

Access functions:

bool	<code>isVisible() const</code>	# 获取属性值
virtual void	<code>setVisible(bool <i>visible</i>)</code>	# 设置 widget 可见性

See also [show\(\)](#), [hide\(\)](#), [isHidden\(\)](#), [isVisibleTo\(\)](#), [isMinimized\(\)](#), [showEvent\(\)](#), and [hideEvent\(\)](#).

whatsThis : [QString](#)

This property holds the widget's What's This help text. # 这个属性保存 widget 的帮助文本

By default, this property contains an empty string. # 该属性默认为空字符

Access functions:

QString	<code>whatsThis() const</code>	# 获取属性值	
void	<code>setWhatsThis(const <i>QString</i> &)</code>	# 设置 widget 帮助文档	

See also [QWhatsThis](#), [QWidget::toolTip](#), and [QWidget::statusTip](#).

width : const int

This property holds the width of the widget excluding any window frame # 该属性保存非窗口框架的 widget 的宽度的像素值

See the [Window Geometry](#) documentation for an overview of geometry issues with windows.

请参阅 [Window Geometry](#) 文档，了解关于 windows 的几何问题的概述。

Note: Do not use this function to find the width of a screen on a [multiple screen desktop](#). Read [this note](#) for details.

贴士：不要使用该函数去查找屏幕的宽度。请阅读 [this note](#) 的详细内容。

By default, this property contains a value that depends on the user's platform and screen geometry. # 该属性默认值依赖于平台和屏幕尺寸

Access functions:

int	width() const	# 获取 widget 的宽度
-----	---------------	-----------------

See also [geometry](#), [height](#), and [size](#).

windowFilePath : [QString](#)

This property holds the file path associated with a widget # 该属性保存与 widget 相关联的文件路径

This property only makes sense for windows. It associates a file path with a window. If you set the file path, but have not set the window title, Qt sets the window title to the file name of the specified path, obtained using [QFileInfo::fileName\(\)](#).

If the window title is set at any point, then the window title takes precedence and will be shown instead of the file path string.

这个属性只对 `windows(parent=None)` 有意义。它将一个文件路径与一个窗口关联起来。如果你设置了文件路径，但是没有设置窗口标题，Qt 将窗口标题设置为指定路径的文件名，使用 `QFileInfo::fileName()` 获得文件名。如果窗口标题设置在任何一点，那么窗口标题将优先使用为窗口名而非文件名。

Additionally, on [macOS](#), this has an added benefit that it sets the [proxy icon](#) for the window, assuming that the file path exists.

If no file path is set, this property contains an empty string.

此外，在 macOS 上，如果存在文件路径，那么它将为窗口设置代理图标，这还有一个额外的好处。如果未设置该属性为空字符串。

By default, this property contains an empty string. # 该属性默认为空字符串

This property was introduced in Qt 4.4. # 该属性自 Qt 4.4 引入

Access functions:

QString	windowFilePath() const	# 查看顶层窗口文件路径
void	setWindowFilePath(const QString &filePath)	# 设置顶层窗口文件路径

See also [windowTitle](#) and [windowIcon](#).

windowFlags : Qt::WindowFlags

Window flags are a combination of a type (e.g. [Qt::Dialog](#)) and zero or more hints to the window system (e.g. [Qt::FramelessWindowHint](#)). If the widget had type [Qt::Widget](#) or [Qt::SubWindow](#) and becomes a window ([Qt::Window](#), [Qt::Dialog](#), etc.), it is put at position (0, 0) on the desktop. If the widget is a window and becomes a [Qt::Widget](#) or [Qt::SubWindow](#), it is put at position (0, 0) relative to its parent widget.

窗口标志是一种类型的组合（例如 [Qt::Dialog](#)）和对窗口系统的零个或多个提示（例如 [Qt::FramelessWindowHint](#)）。如果 widget 有 [Qt::Widget](#) 类型或 [Qt::SubWindow](#) 并成为一个窗口（[Qt::Window](#), [Qt::Dialog](#) 等等），它会放置桌面(0, 0)处；如果 widget 是一个窗口，并且变成了 [Qt::Widget](#) 或 [Qt::SubWindow](#)，它被放在父 widget 的(0, 0)位置。

Constant	Value	Description
Qt::Widget	0x00000000	This is the default type for QWidget . Widgets of this type are child widgets if they have a parent, and independent windows if they have no parent. See also Qt::Window and Qt::SubWindow . # 这是 QWidget 的默认类型。当有父 widget 时，这种类型的 widget 是子 widget，如果它们没有父 widget，则是独立窗口。参见 Qt::Window and Qt::SubWindow .
Qt::Window	0x00000001	Indicates that the widget is a window, usually with a window system frame and a title bar, irrespective of whether the widget has a parent or not. Note that it is not possible to unset this flag if the widget does not have a parent. # 表示 widget 是一个窗口，通常有一个窗口系统框架和一个标题栏，不管这个 widget 是否有父 widget。注意，如果 widget 没有父 widget，则不可能取消设置此标志。

Qt::Dialog	0x00000002 Window	<p>Indicates that the widget is a window that should be decorated as a dialog (i.e., typically no maximize or minimize buttons in the title bar). This is the default type for QDialog. If you want to use it as a modal dialog, it should be launched from another window, or have a parent and used with the QWidget::windowModality property. If you make it modal, the dialog will prevent other top-level windows in the application from getting any input. We refer to a top-level window that has a parent as a <i>secondary</i> window.</p> <p># 表示小部件是一个应该作为对话框进行修饰的窗口（例如通常在标题栏没有最大化或最小化按钮）。这是 QDialog 的默认类型。如果您想将它用作模态对话框，那么它应该从另一个窗口启动，或者有一个父 widget，并与 QWidget::windowModality 属性一起使用。如果您使它成为模态，对话框将阻止应用程序中的其他顶层窗口获得任何输入。我们指的是一个顶级窗口，它的父 widget 是第二个窗口。</p>
Qt::Sheet	0x00000004 Window	<p>Indicates that the window is a sheet on macOS. Since using a sheet implies window modality, the recommended way is to use QWidget::setWindowModality(), or QDialog::open(), instead.</p> <p># 表示窗口是 macOS 上的一张表。因为使用一个表意味着窗口模式，推荐使用 QWidget::setWindowModality() 或 QDialog::open() 来代替。</p>
Qt::Drawer	Sheet Dialog	<p>Indicates that the widget is a drawer on macOS.</p> <p># 显示小部件是 macOS 上的一个下拉窗口。</p>
Qt::Popup	0x00000008 Window	<p>Indicates that the widget is a pop-up top-level window, i.e. that it is modal, but has a window system frame appropriate for pop-up menus.</p> <p># 表示 widget 是一个弹出式顶层窗口，也就是说它是模态的，但是有一个适合弹出菜单的窗口系统框架。</p>
Qt::Tool	Popup Dialog	<p>Indicates that the widget is a tool window. A tool window is often a small window with a smaller than usual title bar and decoration, typically used for collections of tool buttons. If there is a parent, the tool window will always be kept on top of it. If there isn't a parent, you may consider using Qt::WindowStaysOnTopHint as well. If the window system supports it, a tool window can be decorated with a somewhat lighter frame. It can also be combined with Qt::FramelessWindowHint. On macOS, tool windows correspond to</p>

		<p>the NSPanel class of windows. This means that the window lives on a level above normal windows making it impossible to put a normal window on top of it. By default, tool windows will disappear when the application is inactive. This can be controlled by the Qt::WA_MacAlwaysShowToolWindow attribute.</p> <p># 表明 widget 是一个工具栏窗口。工具栏窗口通常是一个小窗口，它的标题栏和装饰比通常要小，通常用于工具按钮的集合。如果有父 widget，工具窗口将始终保持在它的顶部。如果没有父 widget，你可以考虑使用 Qt::WindowStaysOnTopHint。如果窗口系统支持它，工具窗口可以用更轻的框架来装饰。它经常与 Qt::FramelessWindowHint (无标题栏模式) 一起使用。在 macOS 上，工具窗口对应于 NSPanel 类的 windows。这意味着，窗口的位置比普通窗口高，因此不可能在上面放置一个普通的窗口。默认情况下，当应用程序处于非活动状态时，工具窗口将会消失。这可以由 Qt::WA_MacAlwaysShowToolWindow 属性来控制。</p>
Qt::ToolTip	Popup Sheet	<p>Indicates that the widget is a tooltip. This is used internally to implement tooltips.</p> <p># 表明小部件是一个工具提示。这是在内部用来实现工具提示的。</p>
Qt::SplashScreen	ToolTip Dialog	<p>Indicates that the window is a splash screen. This is the default type for QSplashScreen.</p> <p># 表面 widget 是一个启动桌面。它是 QSplashScreen 的默认属性。</p>
Qt::Desktop	0x00000010 Window	<p>Indicates that this widget is the desktop. This is the type for QDesktopWidget.</p> <p># 表示这个 widget 是桌面。这是 QDesktopWidget 的类型。</p>
Qt::SubWindow	0x00000012	<p>Indicates that this widget is a sub-window, such as a QMdiSubWindow widget.</p> <p># 表示 widget 是底层窗口，就像 QmdiSubWindow 一样。</p>
Qt::ForeignWindow	0x00000020 Window	<p>Indicates that this window object is a handle representing a native platform window created by another process or by manually using native code.</p> <p># 表明这个视窗对象是一个句柄，表示由另一个进程创建的本机平台窗口，或者手工使用本机代码。</p>
Qt::CoverWindow	0x00000040 Window	<p>Indicates that the window represents a cover window, which is shown when the application is minimized on some platforms.</p> <p># 表示窗口代表一个封面窗口，当应用程序在某些平台上被最小化时显示。</p>

There are also a number of flags which you can use to customize the appearance of top-level windows. These have no effect on other windows:

还有一些标记，您可以使用它们来定制顶级窗口的外观。这些对其他窗口没有影响：

Constant	Value	Description
Qt::MSWindowsFixedSizeDialogHint	0x00000100	<p>Gives the window a thin dialog border on Windows. This style is traditionally used for fixed-size dialogs.</p> <p># 在窗口上给窗口一个小的对话框。这种风格传统上用于固定大小的对话框。</p>
Qt::MSWindowsOwnDC	0x00000200	<p>Gives the window its own display context on Windows.</p> <p># 在窗口中为窗口提供自己的显示上下文。</p>
Qt::BypassWindowManagerHint	0x00000400	<p>This flag can be used to indicate to the platform plugin that “all” window manager protocols should be disabled. This flag will behave different depending on what operating system the application is running on and what window manager is running. The flag can be used to get a native window with no configuration set.</p> <p># 这个标志可以用来指示平台插件，“所有”窗口管理器协议都应该被禁用。根据应用程序运行的操作系统和窗口管理器的运行情况，此标志将表现不同。可以使用该标志来获得没有配置集的本机窗口。</p>
Qt::X11BypassWindowManagerHint	BypassWindowManagerHint	<p>Bypass the window manager completely. This results in a borderless window that is not managed at all (i.e., no keyboard input unless you call <code>QWidget::activateWindow()</code> manually).</p> <p># 完全绕过窗口管理器。这导致了一个没有边界的无边界窗口（即没有键盘输入，除非你手动调用 <code>QWidget::activateWindow()</code>）。</p>
Qt::FramelessWindowHint	0x00000800	<p>Produces a borderless window. The user cannot move or resize a borderless window via the window system. On X11, the result of the flag is dependent on the window manager and its ability to understand Motif and/or NETWM hints. Most existing modern window managers can handle this.</p>

		# 产生一个无边框的窗口。用户不能通过窗口系统移动或调整无边界窗口的大小。在 X11 上，标志的结果取决于窗口管理器及其理解主题和/或 NETWM 提示的能力。大多数现有的现代窗口管理器都可以处理这个问题。
Qt::NoDropShadowWindowHint	0x40000000	Disables window drop shadow on supporting platforms. # 在支持平台上禁用窗口投影。

The CustomizeWindowHint flag is used to enable customization of the window controls. This flag must be set to allow the WindowTitleHint, WindowSystemMenuHint, WindowMinimizeButtonHint, WindowMaximizeButtonHint and WindowCloseButtonHint flags to be changed.

CustomizeWindowHint 标志用于启用窗口控件的定制，必须设置 WindowTitleHint, WindowSystemMenuHint, WindowMinimizeButtonHint, WindowMaximizeButtonHint 和 WindowCloseButtonHint

Constant	Value	Description
Qt::CustomizeWindowHint	0x02000000	Turns off the default window title hints. # 隐藏窗口标题
Qt::WindowTitleHint	0x00001000	Gives the window a title bar. # 给窗口一个标题栏
Qt::WindowSystemMenuHint	0x00002000	Adds a window system menu, and possibly a close button (for example on Mac). If you need to hide or show a close button, it is more portable to use WindowCloseButtonHint. # 添加一个窗口系统菜单，可能还有一个关闭按钮（例如在 Mac 上）。如果你需要隐藏或显示一个关闭按钮，它使用 WindowCloseButtonHint 更方便。
Qt::WindowMinimizeButtonHint	0x00004000	Adds a minimize button. On some platforms this implies Qt::WindowSystemMenuHint for it to work.

		# 添加一个最小化按钮。在有些平台上意味着 Qt::WindowSystemMenuHint 是工作的。
Qt::WindowMaximizeButtonHint	0x00008000	Adds a maximize button. On some platforms this implies Qt::WindowSystemMenuHint for it to work. # 添加一个最大化按钮。在有些平台上意味着 Qt::WindowSystemMenuHint 是工作的。
Qt::WindowMinMaxButtonsHint	WindowMinimizeButtonHint WindowMaximizeButtonHint	Adds a minimize and a maximize button. On some platforms this implies Qt::WindowSystemMenuHint for it to work. # 添加最大化和最小化按钮。在有些平台上意味着 Qt::WindowSystemMenuHint 是工作的。
Qt::WindowCloseButtonHint	0x08000000	Adds a close button. On some platforms this implies Qt::WindowSystemMenuHint for it to work. # 添加一个关闭按钮。在有些平台上意味着 Qt::WindowSystemMenuHint 是工作的。
Qt::WindowContextHelpButtonHint	0x00010000	Adds a context help button to dialogs. On some platforms this implies Qt::WindowSystemMenuHint for it to work. # 添加一个帮助按钮。在有些平台上意味着 Qt::WindowSystemMenuHint 是工作的。
Qt::MacWindowToolBarButtonHint	0x10000000	On macOS adds a tool bar button (i.e., the oblong button that is on the top right of windows that have toolbars). # 在 macOS 上添加一个工具栏按钮（例如在有工具栏的窗口右上方的长方形按钮。
Qt::WindowFullscreenButtonHint	0x80000000	On macOS adds a fullscreen button. # 在 macOS 平台上为窗口添加一个全屏按钮
Qt::BypassGraphicsProxyWidget	0x20000000	Prevents the window and its children from automatically embedding themselves into a QGraphicsProxyWidget if the

		<p>parent widget is already embedded. You can set this flag if you want your widget to always be a toplevel widget on the desktop, regardless of whether the parent widget is embedded in a scene or not.</p> <p># 如果父部件已经嵌入，则防止窗口及其子女自动嵌入到 QGraphicsProxyWidget 中。如果你想保证你的为 widget 一直为顶层桌面窗口无论父 widget 是否嵌入到场景中，可以设置各种标志。</p>
Qt::WindowShadeButtonHint	0x00020000	<p>Adds a shade button in place of the minimize button if the underlying window manager supports it.</p> <p># 如果底层窗口管理器支持最小化按钮，则添加一个阴影按钮。</p>
Qt::WindowStaysOnTopHint	0x00040000	<p>Informs the window system that the window should stay on top of all other windows. Note that on some window managers on X11 you also have to pass Qt::X11BypassWindowManagerHint for this flag to work correctly.</p> <p># 通知窗口系统，窗口应该保持在所有其他窗口的顶部。请注意，在 X11 的某些窗口管理器上，您还必须传递 Qt::X11BypassWindowManagerHint 使该标志正常工作。</p>
Qt::WindowStaysOnBottomHint	0x04000000	<p>Informs the window system that the window should stay on bottom of all other windows. Note that on X11 this hint will work only in window managers that support _NET_WM_STATE_BELOW atom. If a window always on the bottom has a parent, the parent will also be left on the bottom. This window hint is currently not implemented for macOS.</p> <p># 通知窗口系统，窗口应该保持在所有其他窗口的底部。请注意，在 X11 中，这个提示只适用于支持 _NET_WM_STATE_BELOW 窗口</p>

		<p>管理器。如果一个窗口总是在底部有一个父 widget，那么父 widget 也会留在底部。这个窗口提示目前还没有为 macOS 实现。</p>
Qt::WindowTransparentForInput	0x00080000	<p>Informs the window system that this window is used only for output (displaying something) and does not take input. Therefore input events should pass through as if it wasn't there.</p> <p># 通知窗口系统，这个窗口仅用于输出（显示某些东西），并且不接受输入。因此，输入事件被忽略，就好像它不存在一样。</p>
Qt::WindowOverridesSystemGestures	0x00100000	<p>Informs the window system that this window implements its own set of gestures and that system level gestures, like for instance three-finger desktop switching, should be disabled.</p> <p># 通知窗口系统，这个窗口实现了它自己的一组手势，并且系统级的手势，比如三指桌面切换，应该被禁用。</p>
Qt::WindowDoesNotAcceptFocus	0x00200000	<p>Informs the window system that this window should not receive the input focus.</p> <p># 通知窗口系统，该窗口不接收输入焦点。</p>
Qt::MaximizeUsingFullscreenGeometryHint	0x00400000	<p>Informs the window system that when maximizing the window it should use as much of the available screen geometry as possible, including areas that may be covered by system UI such as status bars or application launchers. This may result in the window being placed under these system UIs, but does not guarantee it, depending on whether or not the platform supports it. When the flag is enabled the user is responsible for taking QScreen::availableGeometry() into account, so that any UI elements in the application that require user interaction are not covered by system UI.</p>

		<div># 通知窗口系统，当最大化窗口时，它应该尽可能多地使用可用的屏幕几何图形，包括可能被系统 UI 覆盖的区域，比如状态栏或应用程序启动器。这可能会导致窗口被置于这些系统 UI 之下，但不能保证是否有效，这取决于平台是否支持它。当启用了标志时，用户负责将 <code>QScreen::availableGeometry()</code> 考虑在内，这样应用程序中任何需要用户交互的 UI 元素都不会被系统 UI 所覆盖。</div>
<code>Qt::WindowType_Mask</code>	<code>0x000000ff</code>	<div>A mask for extracting the window type part of the window flags. # 用于提取窗口标志的窗口类型部分的掩码。</div>

Note: This function calls [setParent\(\)](#) when changing the flags for a window, causing the widget to be hidden. You must call [show\(\)](#) to make the widget visible again.

贴士：这个函数在改变窗口的标志时调用 `setParent()`，导致 `widget` 被隐藏。您必须调用 `show()` 来使 `widget` 再次可见。

Access functions:

<code>Qt::WindowFlags</code>	<code>windowFlags() const</code>	# 查看窗口类型标志
<code>void</code>	<code>setWindowFlags(Qt::WindowFlags type)</code>	# 设置窗口类型标志

See also [windowType\(\)](#), [setWindowFlag\(\)](#), and [Window Flags Example](#).

例如：

```
self.setWindowFlags(Qt.FramelessWindowHint | Qt.WindowCloseButtonHint) # 设置无边框，有关闭按钮的窗口
```

windowIcon : [QIcon](#)

This property holds the widget's icon # 该属性保存 widget 的图标

This property only makes sense for windows. If no icon has been set, `windowIcon()` returns the application icon ([QApplication::windowIcon\(\)](#)).

该属性仅对顶层窗口(`parent=None`)有效，如果图标没有设置，`windowIcon()`返回应用程序图标([QApplication::windowIcon\(\)](#))。

Access functions:

QIcon	<code>windowIcon()</code> const	# 获取图标
void	<code>setWindowIcon(const QIcon &icon)</code>	# 设置图标, QIcon()

Notifier signal:

void	<code>windowIconChanged(const QIcon &icon)</code>	# 窗口的图标发生变化时, 会发出此信号, 并将新图标作为参数
------	---	---------------------------------

See also [windowTitle](#).

windowModality : Qt::WindowModality

This property holds which windows are blocked by the modal widget # 这个属性保存了哪个窗口被模态小部件阻塞

This property only makes sense for windows. A modal widget prevents widgets in other windows from getting input. The value of this property controls which windows are blocked when the widget is visible. Changing this property while the window is visible has no effect; you must [hide\(\)](#) the widget first, then [show\(\)](#) it again.

这个属性只对顶层窗口(parent=None)有意义。模态 widget 防止其他窗口中的小部件获得输入。当 widget 可见时, 该属性的值控制哪些窗口被阻塞。在窗口可见时改变该属性没有效果;您必须先 [hide\(\)](#) widget, 然后再 [show\(\)](#)。

By default, this property is [Qt::NonModal](#). # 属性默认值 Qt::NonModal

This property was introduced in Qt 4.1. # 在 Qt 4.1 中介绍

Constant	Value	Description
Qt::NonModal	0	The window is not modal and does not block input to other windows. # 窗口不是模态的, 也不会阻止对其他窗口的输入。
Qt::WindowModal	1	The window is modal to a single window hierarchy and blocks input to its parent window, all grandparent windows, and all siblings of its parent and grandparent windows. # 窗口是一个单一窗口层次结构的模态, 它阻止对其母窗口的输入、所有祖父母窗口, 以及其父节点和祖父母窗口的所有兄弟节点。
Qt::ApplicationModal	2	The window is modal to the application and blocks input to all windows. # 窗口时应用程序模态的, 阻止所有其他窗口获取输入。

Access functions:

Qt::WindowModality	windowModality() const	# 查看模态属性
void	setWindowModality (Qt::WindowModality <i>windowModality</i>)	# 设置模态属性

See also [isWindow\(\)](#), [QWidget::modal](#), and [QDialog](#).

windowModified : bool

This property holds whether the document shown in the window has unsaved changes # 该属性保存窗口中显示的文档是否有未保存的更改

A modified window is a window whose content has changed but has not been saved to disk. This flag will have different effects varied by the platform. On [macOS](#) the close button will have a modified look; on other platforms, the window title will have an '*' (asterisk).

The window title must contain a "[*]" placeholder, which indicates where the '*' should appear. Normally, it should appear right after the file name (e.g., "document1.txt[*] - Text Editor"). If the window isn't modified, the placeholder is simply removed.

修改后的窗口是它的内容发生了变化，但是没有保存到磁盘。这个标志将会有不同的效果。在 macOS 上，关闭按钮将会有有一个修改的外观；在其他平台上，窗口标题将有一个 '*'（星号）。窗口标题必须包含一个 "[*]" 占位符，它表示 "*" 出现的位置。通常，它应该出现在文件名之后（例如："document1.txt[*] - Text Editor"）。

Note that if a widget is set as modified, all its ancestors will also be set as modified. However, if you call `setWindowModified(false)` on a widget, this will not propagate to its parent because other children of the parent might have been modified.

注意，如果一个 widget 被设置为修改，那么它的所有祖 widget 也将被设置为修改。但是，如果您在小部件上调用 `setWindowModified(false)`，那么它将不会传播到其父 widget，因为其他子 widget 可能已经被修改了。

Access functions:

bool	isWindowModified() const	# 查看属性
void	setWindowModified (<i>bool</i>)	# 设置属性

See also [windowTitle](#), [Application Example](#), [SDI Example](#), and [MDI Example](#).

windowOpacity : double

This property holds the level of opacity for the window. # 该属性保存 widget 的透明度

The valid range of opacity is from 1.0 (completely opaque) to 0.0 (completely transparent). # 透明度取值在 1.0（不透明）到 0.0（完全透明）这件

By default the value of this property is 1.0. # 默认值为 1.0（不透明）

This feature is available on Embedded Linux, [macOS](#), Windows, and X11 platforms that support the Composite extension.

该功能可在支持复合扩展的 Embedded Linux， macOS， Windows 和 X11 平台上使用。

Note: On X11 you need to have a composite manager running, and the X11 specific _NET_WM_WINDOW_OPACITY atom needs to be supported by the window manager you are using.

贴士：在 X11 上，您需要运行一个复合管理器并且 X11 特定的 _NET_WM_WINDOW_OPACITY 元素需要由您正在使用的窗口管理器支持。

Warning: Changing this property from opaque to transparent might issue a paint event that needs to be processed before the window is displayed correctly. This affects mainly the use of QPixmap::grabWindow(). Also note that semi-transparent windows update and resize significantly slower than opaque windows.

警告：将该属性从不透明转到透明，需要在真正显示前处理绘画事件(paint event)。这将影响掉 QPixmap::grabWindow()的使用。还要注意的，半透明的窗口更新和调整的速度比不透明的窗口要慢得多。

Access functions:

qreal	windowOpacity() const	# 查看透明度
void	setWindowOpacity(qreal level)	# 设置透明度

See also [setMask\(\)](#).

windowTitle : QString

This property holds the window title (caption) # 该属性保存 widget 的标题

This property only makes sense for top-level widgets, such as windows and dialogs. If no caption has been set, the title is based of the [windowFilePath](#). If neither of these is set, then the title is an empty string.

该属性仅对顶层窗口（如窗口、对话框）有效。如果 widget 的标题没有设置则标题依赖 windowFilePath 设置。如果两者都没有设置，那么标题就是空字符串。

If you use the [windowModified](#) mechanism, the window title must contain a "[*]" placeholder, which indicates where the '*' should appear. Normally, it should appear right after the file name (e.g., "document1.txt[*] - Text Editor"). If the [windowModified](#) property is false (the default), the placeholder is simply removed.

如果您使用了 windowModified 机制，窗口标题必须包含一个"*"占位符，它表示"*"出现的位置。通常，它应该出现在文件名之后（例如："document1.txt[*] - Text Editor"）。如果的 windowModified 属性是 False（默认值），占位符就会被删除。

On some desktop platforms (including Windows and Unix), the application name (from [QGuiApplication::applicationDisplayName](#)) is added at the end of the window title, if set. This is done by the QPA plugin, so it is shown to the user, but isn't part of the windowTitle string.

Access functions:

QString	<code>windowTitle() const</code>	# 获取窗口标题
void	<code>setWindowTitle(const QString &)</code>	# 设置窗口标题

Notifier signal:

void	<code>windowTitleChanged(const QString &title)</code>
------	---

See also [windowIcon](#), [windowModified](#), and [windowFilePath](#).

x : const int

This property holds the x coordinate of the widget relative to its parent including any window frame # 保存 widget 相对于父 widget 的 X 坐标

See the [Window Geometry](#) documentation for an overview of geometry issues with windows.

参阅 Window Geometry 文档获取窗口几何功能概述。

By default, this property has a value of 0. # 默认为 0

Access functions:

int	<code>x() const</code>	# 获取 X 坐标
-----	------------------------	-----------

See also [frameGeometry](#), [y](#), and [pos](#).

y : const int

This property holds the y coordinate of the widget relative to its parent and including any window frame # 保存 widget 相对于父 widget 的 Y 坐标

See the [Window Geometry](#) documentation for an overview of geometry issues with windows.

参阅 Window Geometry 文档获取窗口几何功能概述。

By default, this property has a value of 0. # 默认为 0

Access functions:

int	y()	const	# 获取 Y 坐标
-----	-----	-------	-----------

See also [frameGeometry](#), [x](#), and [pos](#).

Member Function Documentation|成员函数文档

QWidget::QWidget([QWidget](#) *parent = nullptr, Qt::WindowFlags f = ...)

Constructs a widget which is a child of *parent*, with widget flags set to *f*. # 构建一个 widget，父 widget = parent，widget 标记=f

If *parent* is 0, the new widget becomes a window. If *parent* is another widget, this widget becomes a child window inside *parent*. The new widget is deleted when its *parent* is deleted.

如果 parent=None，则新的 widget 将成为一个窗口。如果 parent=其他 widget，这个 widget 就变成了父 widget 中的子窗口。当它的 parent 被删除时，新的 widget 将被删除。

The widget flags argument, *f*, is normally 0, but it can be set to customize the frame of a window (i.e. *parent* must be 0). To customize the frame, use a value composed from the bitwise OR of any of the [window flags](#).

If you add a child widget to an already visible widget you must explicitly show the child to make it visible.

widget 标志参数，f，通常为 None，但可以设置为自定义窗口的框架（即 parent=None）。要自定义框架，请使用由位或任何窗口标志组成的值。如果您将一个子 widget 添加到一个已经可见的 widget 中，那么您必须显式地显示子 widget 使其可见。

Note that the X11 version of Qt may not be able to deliver all combinations of style flags on all systems. This is because on X11, Qt can only ask the window manager, and the window manager can override the application's settings. On Windows, Qt can set whatever flags you want.

请注意，X11 版本的 Qt 可能无法向系统上传递样式标志的所有组合。这是因为在 X11 中，Qt 只能询问窗口管理器，而窗口管理器可以覆盖应用程序的设置。在 Windows 上，Qt 可以设置任何你想要的标志。

See also [windowFlags](#).

[virtual] QWidget::~~QWidget()

Destroys the widget. # 销毁所有的 widget

All this widget's children are deleted first. The application exits if this widget is the main widget.

所有的子 widget 也会销毁。若无主窗口应用程序关闭

[virtual protected] void QWidget::actionEvent([QActionEvent](#) *event)

This event handler is called with the given *event* whenever the widget's actions are changed.

每当 widget 的 QAction 发生改变时，都会调用这个事件处理程序。

See also [addAction\(\)](#), [insertAction\(\)](#), [removeAction\(\)](#), [actions\(\)](#), and [QActionEvent](#).

[QList](#)<[QAction](#) *> QWidget::actions() const

Returns the (possibly empty) list of this widget's actions.

返回 widget 的 QAction() 列表（可能为[]）

See also [contextMenuPolicy](#), [insertAction\(\)](#), and [removeAction\(\)](#).

void QWidget::activateWindow()

Sets the top-level widget containing this widget to be the active window. # 将包含这个 widget 的顶层 widget 设置为活动窗口

An active window is a visible top-level window that has the keyboard input focus. # 顶层窗口可见且可接受键盘输入

This function performs the same operation as clicking the mouse on the title bar of a top-level window. On X11, the result depends on the Window Manager. If you want to ensure that the window is stacked on top as well you should also call [raise\(\)](#). Note that the window must be visible, otherwise `activateWindow()` has no effect.

这个函数执行与在顶级窗口标题栏上单击鼠标相同的操作。在 X11 上，结果取决于窗口管理器。如果您想要确保窗口也在顶部，您也应该调用 [raise\(\)](#)。注意，窗口必须是可见的，否则 `activateWindow()` 无效。

On Windows, if you are calling this when the application is not currently the active one then it will not make it the active window. It will change the color of the taskbar entry to indicate that the window has changed in some way. This is because Microsoft does not allow an application to interrupt what the user is currently doing in another application.

在 Windows 上，如果您在应用程序当前不是活动的时候调用它，那么它就不会使它成为活动窗口。它将改变任务栏条目的颜色，以指示窗口在某种程度上发生了变化。这是因为微软不允许应用程序中断用户当前在另一个应用程序中所做的事情。

See also [isActiveWindow\(\)](#), [window\(\)](#), [show\(\)](#), and [QWindowsWindowFunctions::setWindowActivationBehavior\(\)](#).

`void QWidget::addAction(QAction *action)`

Appends the action *action* to this widget's list of actions. # 为 widget 添加 [QAction\(\)](#)

All [QWidget](#)s have a list of [QActions](#), however they can be represented graphically in many different ways. The default use of the [QAction](#) list (as returned by [actions\(\)](#)) is to create a context [QMenu](#).

A [QWidget](#) should only have one of each action and adding an action it already has will not cause the same action to be in the widget twice.

The ownership of *action* is not transferred to this [QWidget](#).

所有 [QWidget](#)s 都有一个 [QAction](#) 的列表，但是它们可以用许多不同的方式表示出来。[QAction](#) list 的默认用法（由 [action\(\)](#) 返回）是创建一个上下文 [QMenu](#)。[QWidget](#) 应该只拥有一个动作，并添加一个已经拥有的动作，不会在小部件中造成相同的动作两次。动作的所有权没有转移到这个 [QWidget](#)。

See also [removeAction\(\)](#), [insertAction\(\)](#), [actions\(\)](#), and [QMenu](#).

`void QWidget::addActions(QList<QAction> *actions)`

Appends the actions *actions* to this widget's list of actions. # 为 widget 添加一个 [QAction](#) list

See also [removeAction\(\)](#), [QMenu](#), and [addAction\(\)](#).

`void QWidget::adjustSize()`

Adjusts the size of the widget to fit its contents. # 改变 widget 的尺寸以适应内容

This function uses [sizeHint\(\)](#) if it is valid, i.e., the size hint's width and height are ≥ 0 . Otherwise, it sets the size to the children rectangle that covers all child widgets (the union of all child widget rectangles).

For windows, the screen size is also taken into account. If the [sizeHint\(\)](#) is less than (200, 100) and the size policy is [expanding](#), the window will be at least (200, 100). The maximum size of a window is 2/3 of the screen's width and height.

如果它是有效的，这个函数使用 [sizeHint\(\)](#)。大小提示的宽度和高度等于 0。另外，它将大小设置为涵盖所有子小部件（所有子部件矩形的联合）的子矩形。对于 **windows** 来说，屏幕的大小也被考虑在内。如果 [sizeHint\(\)](#) 小于 (200, 100)，并且尺寸策略正在扩展，那么窗口将至少是 (200, 100)。窗口的最大大小是屏幕宽度和高度的 2/3。

See also [sizeHint\(\)](#) and [childrenRect\(\)](#).

QPalette::ColorRole QWidget::backgroundRole() const

Returns the background role of the widget. # 返回 widget 的背景

The background role defines the brush from the widget's [palette](#) that is used to render the background.

If no explicit background role is set, the widget inherits its parent widget's background role.

[backgroundRole\(\)](#) 定义了用于 widget 的 QPalette 中的 brush。如果没有设定明确的 [backgroundRole\(\)](#)，widget 将继承其父部件的 [backgroundRole\(\)](#)。

See also [setBackgroundRole\(\)](#) and [foregroundRole\(\)](#).

[QBackingStore](#) *QWidget::backingStore() const

Returns the [QBackingStore](#) this widget will be drawn into. # 返回 QBackingStore 这个小部件将被拖入。

This function was introduced in Qt 5.0. # 在 Qt 5.0 中引进

[virtual protected] void QWidget::changeEvent([QEvent](#) *event)

This event handler can be reimplemented to handle state changes.

The state being changed in this event can be retrieved through the *event* supplied. # 在这个事件中被更改的状态可以通过所提供的事件来检索。

Change events # 更改事件包括

include: [QEvent::ToolBarChange](#), [QEvent::ActivationChange](#), [QEvent::EnabledChange](#), [QEvent::FontChange](#), [QEvent::StyleChange](#), [QEvent::PaletteChange](#), [QEvent::WindowTitleChange](#), [QEvent::IconTextChange](#), [QEvent::ModifiedChange](#), [QEvent::MouseTrackingChange](#), [QEvent::ParentChange](#), [QEvent::WindowStateChange](#), [QEvent::LanguageChange](#), [QEvent::LocaleChange](#), [QEvent::LayoutDirectionChange](#), [QEvent::ReadOnlyChange](#).

[QWidget](#) *QWidget::childAt(int x, int y) const

Returns the visible child widget at the position (x, y) in the widget's coordinate system. If there is no visible child widget at the specified position, the function returns 0.

返回可见的子 widget 的在父 widget 中的坐标 (x, y)。如果在指定位置没有可见的子部件，则函数返回 0

[QWidget](#) *QWidget::childAt(const [QPoint](#) &p) const

This is an overloaded function. # 这是一个重载函数

Returns the visible child widget at point p in the widget's own coordinate system. # 返回可见的子 widget 的在父 widget 中的坐标 QPoint。

void QWidget::clearFocus()

Takes keyboard input focus from the widget. # 清除 widget 上的键盘输入焦点

If the widget has active focus, a [focus out event](#) is sent to this widget to tell it that it has lost the focus.

This widget must enable focus setting in order to get the keyboard input focus, i.e. it must call [setFocusPolicy\(\)](#).

如果小部件有活动焦点，则将焦点丢失事件发送到这个 widget，告诉它它已经失去了焦点。这个小部件必须启用焦点设置，以便获得键盘输入焦点，也就是说，它必须调用 [setFocusPolicy\(\)](#)。

See also [hasFocus\(\)](#), [setFocus\(\)](#), [focusInEvent\(\)](#), [focusOutEvent\(\)](#), [setFocusPolicy\(\)](#), and [QApplication::focusWidget\(\)](#).

void QWidget::clearMask()

Removes any mask set by [setMask\(\)](#). # 清除 setMask() 的设置

See also [setMask\(\)](#).

[slot] bool QWidget::close()

Closes this widget. Returns true if the widget was closed; otherwise returns false. # 关闭窗口，如果成功则返回 True

First it sends the widget a [QCloseEvent](#). The widget is [hidden](#) if it [accepts](#) the close event. If it [ignores](#) the event, nothing happens. The default implementation of [QWidget::closeEvent\(\)](#) accepts the close event.

首选 close() 项 widget 发送一个 QcloseEvent。如果 accept() 则将 widget 隐藏(hidden)，如果 ignores()，则不进行任何动作。默认重新 QWidget::closeEvent() 来接收关闭事件。

If the widget has the [Qt::WA_DeleteOnClose](#) flag, the widget is also deleted. A close event is delivered to the widget no matter if the widget is visible or not.

The [QApplication::lastWindowClosed\(\)](#) signal is emitted when the last visible primary window (i.e. window with no parent) with the [Qt::WA_QuitOnClose](#) attribute set is closed. By default this attribute is set for all widgets except transient windows such as splash screens, tool windows, and popup menus.

如果 widget 有 Qt::WA_DeleteOnClose，widget 会被清除。无论 widget 是否可见，都会将 close 事件发送给 widget。

QApplication::lastWindowClosed() 信号在最后一个可见的主窗口（parent=None）与 Qt::WA_QuitOnClose 属性集关闭时发出。默认情况下，这个属性是为所有小部件设置的，除了临时窗口，如闪屏、工具窗口和弹出菜单。

[virtual protected] void QWidget::closeEvent([QCloseEvent](#) *event)

This event handler is called with the given *event* when Qt receives a window close request for a top-level widget from the window system.

By default, the event is accepted and the widget is closed. You can reimplement this function to change the way the widget responds to window close requests. For example, you can prevent the window from closing by calling [ignore\(\)](#) on all events.

当 Qt 收到来自窗口系统的顶层 widget 的窗口关闭请求时，这个事件处理器会被调用。默认情况下，事件被接受，widget 关闭。您可以重写这个功能，以改变 widget 响应窗口关闭请求的方式。例如，您可以在所有事件上调用 ignore() 来防止窗口关闭。

Main window applications typically use reimplementations of this function to check whether the user's work has been saved and ask for permission before closing. For example, the [Application Example](#) uses a helper function to determine whether or not to close the window:

主窗口应用程序通常使用这个函数的重新实现来检查用户的工作是否已经被保存，并在关闭之前请求获得许可。例如，应用程序示例使用一个助手函数来确定是否关闭窗口：

```
void MainWindow::closeEvent(QCloseEvent *event)
{
```

```
if (maybeSave()) {  
    writeSettings();  
    event->accept();  
} else {  
    event->ignore();  
}  
}
```

See also [event\(\)](#), [hide\(\)](#), [close\(\)](#), [QCloseEvent](#), and [Application Example](#).

[QMargins](#) QWidget::contentsMargins() const

The contentsMargins function returns the widget's contents margins. # 返回 widget 的外边框

This function was introduced in Qt 4.6. # 在 Qt 4.6 中介绍

See also [getContentsMargins\(\)](#), [setContentsMargins\(\)](#), and [contentsRect\(\)](#).

[QRect](#) QWidget::contentsRect() const

Returns the area inside the widget's margins. # 返回小部件的边缘区域

See also [setContentsMargins\(\)](#) and [getContentsMargins\(\)](#).

[virtual protected]void QWidget::contextMenuEvent([QContextMenuEvent](#) *event)

This event handler, for event *event*, can be reimplemented in a subclass to receive widget context menu events.

The handler is called when the widget's [contextMenuPolicy](#) is [Qt::DefaultContextMenu](#).

The default implementation ignores the context event. See the [QContextMenuEvent](#) documentation for more details.

这个事件处理程序，对于事件事件，可以在一个子类中重写，以接收小部件上下文菜单事件。当小部件的 [contextMenuPolicy](#) 是 [Qt::DefaultContextMenu](#) 时，就会调用处理程序。默认的实现忽略上下文事件。有关更多细节，请参见 [QContextMenuEvent](#) 文档。

See also [event\(\)](#), [QContextMenuEvent](#), and [customContextMenuRequested\(\)](#).

```
[protected]void QWidget::create(WId window = 0, bool initializeWindow = true, bool destroyOldWindow = true)
```

Creates a new widget window. # 创建一个窗口

The parameter *window* is ignored in Qt 5. Please use [QWindow::fromWinId\(\)](#) to create a [QWindow](#) wrapping a foreign window and pass it to [QWidget::createWindowContainer\(\)](#) instead.

参数 window 在 Qt 5 中忽略。请使用 [QWindow::fromWinId\(\)](#) 创建一个 [QWindow](#)，通过他把外部窗口嵌入尽量取代 [QWidget::createWindowContainer\(\)](#)。

Initializes the window (sets the geometry etc.) if *initializeWindow* is true. If *initializeWindow* is false, no initialization is performed. This parameter only makes sense if *window* is a valid window.

如果 initializeWindow=True 则初始化窗口。如果窗口是一个有效的窗口，那么这个参数是有意义的。

Destroys the old window if *destroyOldWindow* is true. If *destroyOldWindow* is false, you are responsible for destroying the window yourself (using platform native code).

如果 destroyOldWindow=True 则销毁旧窗口。如果 destroyOldWindow=False，你就要手动销毁旧窗口。

The [QWidget](#) constructor calls `create(0,true,true)` to create a window for this widget.

QWidget 构造器调用 `create(0,true,true)` 去构建一个新的窗口。

See also [createWindowContainer\(\)](#) and [QWindow::fromWinId\(\)](#).

```
[static]QWidget *QWidget::createWindowContainer(QWindow *window, QWidget* parent = nullptr, Qt::WindowFlags flags = ...)
```

Creates a [QWidget](#) that makes it possible to embed *window* into a [QWidget](#)-based application.

创建一个 [QWidget](#)，使其能够将窗口嵌入到基于 [QWidget](#) 的应用程序中。

The window container is created as a child of *parent* and with window flags *flags*. # 窗口容器是作为父 widget 创建的，并且带有窗口标志(flags)。

Once the window has been embedded into the container, the container will control the window's geometry and visibility. Explicit calls to [QWindow::setGeometry\(\)](#), [QWindow::show\(\)](#) or [QWindow::hide\(\)](#) on an embedded window is not recommended.

一旦窗口被嵌入到容器中，容器将控制窗口的几何形状和可见性。不推荐在嵌入的窗口显式调用 `QWindow::setGeometry()`, `QWindow::show()` 或 `QWindow::hide()`。

The container takes over ownership of *window*. The window can be removed from the window container with a call to `QWindow::setParent()`. The window container is attached as a native child window to the toplevel window it is a child of. When a window container is used as a child of a `QAbstractScrollArea` or `QMdiArea`, it will create a native window for every widget in its parent chain to allow for proper stacking and clipping in this use case. Creating a native window for the window container also allows for proper stacking and clipping. This must be done before showing the window container. Applications with many native child windows may suffer from performance issues.

容器接管了窗口的所有权。调用 `QWindow::setParent()` 可以从窗口容器中取出窗口。窗口容器作为一个本机子窗口附加到顶层窗口。当一个窗口容器被用作 `QAbstractScrollArea` 或 `QMdiArea` 的子元素时，它将其母链中的每个小部件创建一个本机窗口，以便在这个用例中适当地堆叠和剪裁。为窗口容器创建一个本机窗口也允许适当的堆叠和剪裁。这必须在显示窗口容器之前完成。带有许多本机子窗口的应用程序可能会遇到性能问题。

The window container has a number of known limitations: # 窗口容器有一些已知的局限性

- Stacking order; The embedded window will stack on top of the widget hierarchy as an opaque box. The stacking order of multiple overlapping window container instances is undefined.

堆栈顺序;嵌入的窗口将堆叠在小部件层次结构的顶部，作为一个不透明的盒子。多个重叠窗口容器实例的叠加顺序没有定义。

- Rendering Integration; The window container does not interoperate with `QGraphicsProxyWidget`, `QWidget::render()` or similar functionality.

呈现集成;窗口容器不能与 `QgraphicsProxyWidget`, `QWidget::render()` 或类似功能进行互操作。

- Focus Handling; It is possible to let the window container instance have any focus policy and it will delegate focus to the window via a call to `QWindow::requestActivate()`. However, returning to the normal focus chain from the `QWindow` instance will be up to the `QWindow` instance implementation itself. For instance, when entering a Qt Quick based window with tab focus, it is quite likely that further tab presses will only cycle inside the QML application. Also, whether `QWindow::requestActivate()` actually gives the window focus, is platform dependent.

集中处理;可以让窗口容器实例有任何焦点策略，它可以通过对 `QWindow::requestActivate()` 的调用将焦点委托给窗口。然而，从 `QWindow` 实例返回普通焦点链将取决于 `QWindow` 实例实现本身。例如，当以 `tab` 键进入 Qt Quick 的窗口时，很有可能进一步的 `tab` 按压只会在 QML 应用程序中循环。另外，`QWindow::requestActivate()` 实际上给予窗口焦点，是依赖于平台的。

- Using many window container instances in a `QWidget`-based application can greatly hurt the overall performance of the application.

在基于 `QWidget` 的应用程序中使用许多窗口容器实例会极大地损害应用程序的整体性能。

[signal]void QWidget::customContextMenuRequested(const QPoint &pos)

This signal is emitted when the widget's [contextMenuPolicy](#) is [Qt::CustomContextMenu](#), and the user has requested a context menu on the widget. The position *pos* is the position of the context menu event that the widget receives. Normally this is in widget coordinates. The exception to this rule is [QAbstractScrollArea](#) and its subclasses that map the context menu event to coordinates of the [viewport\(\)](#).

当 widget 的 contextMenuPolicy 是 Qt::CustomContextMenu 时，这个信号就会发出，并且用户已经在 widget 上请求了一个上下文菜单。位置 pos 是 widget 接收到的上下文菜单事件的位置，通常这是在 widget 的坐标范围内。这个规则的例外是 QAbstractScrollArea 及其子类，它将上下文菜单事件映射到 viewport() 的坐标。

See also [mapToGlobal\(\)](#), [QMenu](#), and [contextMenuPolicy](#).

[protected]void QWidget::destroy(bool *destroyWindow* = true, bool *destroySubWindows* = true)

Frees up window system resources. Destroys the widget window if *destroyWindow* is true.

destroy() calls itself recursively for all the child widgets, passing *destroySubWindows* for the *destroyWindow* parameter. To have more control over destruction of subwidgets, destroy subwidgets selectively first.

This function is usually called from the [QWidget](#) destructor.

释放窗口系统资源。若 destroyWindow=True，则销毁 widget 窗口。destroy()递归地调用所有的子 widget。destroySubWindows 是 destroyWindow 的参数。要对 widget 的销毁有更精确的控制，首先要有选择地销毁子 widget。这个函数通常来自 QWidget 析构函数。

[virtual protected]void QWidget::dragEnterEvent([QDragEnterEvent](#) **event*)

This event handler is called when a drag is in progress and the mouse enters this widget. The event is passed in the *event* parameter.

If the event is ignored, the widget won't receive any [drag move events](#).

See the [Drag-and-drop documentation](#) for an overview of how to provide drag-and-drop in your application.

当鼠标进入 widget 并拖动时，这个事件处理器会被调用。在事件中传递 event 参数。如果事件被忽略，widget 不会接收任何拖动移动事件。

请参阅 Drag-and-drop documentation，了解如何在应用程序中提供拖放操作。

See also [QDrag](#) and [QDragEnterEvent](#).

[virtual protected]void QWidget::dragLeaveEvent([QDragLeaveEvent](#) **event*)

This event handler is called when a drag is in progress and the mouse leaves this widget. The event is passed in the *event* parameter.

See the [Drag-and-drop documentation](#) for an overview of how to provide drag-and-drop in your application.

当鼠标离开这个 widget 且拖动正在进行时，这个事件处理器会被调用。事件中传递 event 参数。

请参阅 Drag-and-drop documentation，了解如何在应用程序中提供拖放操作。

See also [QDrag](#) and [QDragLeaveEvent](#).

[virtual protected]void QWidget::dragMoveEvent([QDragMoveEvent](#) *event)

This event handler is called if a drag is in progress, and when any of the following conditions occur: the cursor enters this widget, the cursor moves within this widget, or a modifier key is pressed on the keyboard while this widget has the focus. The event is passed in the *event* parameter.

See the [Drag-and-drop documentation](#) for an overview of how to provide drag-and-drop in your application.

如果一个拖动正在进行中，那么这个事件处理程序就会被调用，并且当出现以下情况时：光标进入这个 widget，光标在这个 widget 中移动，或者在键盘上按下一个修饰键，而这个 widget 有焦点。事件中传递 event 参数。

请参阅 [Drag-and-drop documentation](#)，了解如何在应用程序中提供拖放操作。

See also [QDrag](#) and [QDragMoveEvent](#).

[virtual protected]void QWidget::dropEvent([QDropEvent](#) *event)

This event handler is called when the drag is dropped on this widget. The event is passed in the *event* parameter.

See the [Drag-and-drop documentation](#) for an overview of how to provide drag-and-drop in your application.

这个事件处理程序在这个 widget 上的拖动后放下时被调用。事件中传递 event 参数。

请参阅 [Drag-and-drop documentation](#)，了解如何在应用程序中提供拖放操作。

See also [QDrag](#) and [QDropEvent](#).

WId QWidget::effectiveWinId() const

Returns the effective window system identifier of the widget, i.e. the native parent's window system identifier.

返回 widget 的有效窗口系统标识符，即主窗口系统标识符。

If the widget is native, this function returns the native widget ID. Otherwise, the window ID of the first native parent widget, i.e., the top-level widget that contains this widget, is returned.

如果这个 widget(parent=None)，这个函数返回给 widge 系统 ID。否则，返回父 widget 的窗口 ID，即包含这个 widget 的顶层窗口 ID。

Note: We recommend that you do not store this value as it is likely to change at run-time.

注意：我们建议您不要存储这个值，因为它可能在运行时发生变化。

This function was introduced in Qt 4.4. # 该功能在 Qt 4.4 中介绍

See also [nativeParentWidget\(\)](#).

void QWidget::ensurePolished() const

Ensures that the widget and its children have been polished by [QStyle](#) (i.e., have a proper font and palette).

[QWidget](#) calls this function after it has been fully constructed but before it is shown the very first time. You can call this function if you want to ensure that the widget is polished before doing an operation, e.g., the correct font size might be needed in the widget's [sizeHint\(\)](#) reimplementation. Note that this function *is* called from the default implementation of [sizeHint\(\)](#).

Polishing is useful for final initialization that must happen after all constructors (from base classes as well as from subclasses) have been called.

确保小部件及其 widget 经过 Qstyle 美化（即：有适当的字体和调色板）。QWidget 在它被完全构造但还未显示时调用这个函数。如果您想要确保 widget 在执行操作之前是经过美化，那么您可以调用这个函数，例如，在 widget 的 sizeHint() 重写中可能需要更改字体大小。注意，这个函数是从 sizeHint() 的默认实现中调用的。样式化对于最终的初始化是很有用的，在所有的构造函数（从基类和子类）被调用之后，必须进行最终的初始化。

If you need to change some settings when a widget is polished, reimplement [event\(\)](#) and handle the [QEvent::Polish](#) event type.

如果您需要在一个小部件被样式化时更改一些设置，则重新实现 event() 并处理 QEvent::Polish 事件类型。

Note: The function is declared const so that it can be called from other const functions (e.g., [sizeHint\(\)](#)).

注意：这个函数被声明为常量，这样它就可以从其他常量函数调用（例如，sizeHint()）。

See also [event\(\)](#).

[virtual protected]void QWidget::enterEvent([QEvent](#) *event)

This event handler can be reimplemented in a subclass to receive widget enter events which are passed in the *event* parameter.

An event is sent to the widget when the mouse cursor enters the widget.

这个事件处理器可以在子类中重写，以接收在事件参数中传递 widget 进入事件。当鼠标光标进入 widget 时，会将事件发送给 widget。

See also [leaveEvent\(\)](#), [mouseMoveEvent\(\)](#), and [event\(\)](#).

[override virtual protected]bool QWidget::event([QEvent](#) *event)

Reimplemented from [QObject::event\(\)](#). # 自 QObject::event() 重写

This is the main event handler; it handles event *event*. You can reimplement this function in a subclass, but we recommend using one of the specialized event handlers instead.

这是主事件处理程序；它处理事件的事件。您可以在子类中重新实现这个功能，但是我们建议使用一个专门的事件处理程序

Key press and release events are treated differently from other events. event() checks for Tab and Shift+Tab and tries to move the focus appropriately. If there is no widget to move the focus to (or the key press is not Tab or Shift+Tab), event() calls [keyPressEvent\(\)](#).

键按压和键弹起被以不同的事件进行对待。event()检查 Tab 和 Shift+Tab 来移动焦点。如果没有 widget 将焦点转移到（或者按键不是 Tab 或 Shift+Tab），event()调用 keyPressEvent()。

Mouse and tablet event handling is also slightly special: only when the widget is [enabled](#), event() will call the specialized handlers such as [mousePressEvent\(\)](#); otherwise it will discard the event.

鼠标和平板事件处理也有点特殊：只有当 widget 启用时，event()才会调用诸如 mousePressEvent()之类的专门处理程序;否则它将丢弃事件。

This function returns true if the event was recognized, otherwise it returns false. If the recognized event was accepted (see [QEvent::accepted](#)), any further processing such as event propagation to the parent widget stops.

如果事件被接收返回 True，否则返回 False。如果被认可的事件被接受（参见 [Qevent::accepted](#)），任何进一步的处理，如事件传播到 widget 都停止。

See

also [closeEvent\(\)](#), [focusInEvent\(\)](#), [focusOutEvent\(\)](#), [enterEvent\(\)](#), [keyPressEvent\(\)](#), [keyReleaseEvent\(\)](#), [leaveEvent\(\)](#), [mouseDoubleClickEvent\(\)](#), [mouseMoveEvent\(\)](#), [mousePressEvent\(\)](#), [mouseReleaseEvent\(\)](#), [moveEvent\(\)](#), [paintEvent\(\)](#), [resizeEvent\(\)](#), [QObject::event\(\)](#), and [QObject::timerEvent\(\)](#).

[static] [QWidget](#) *QWidget::find(WId id)

Returns a pointer to the widget with window identifier/handle id. # 返回带有窗口标识/句柄 id 的 widget 的指针。

The window identifier type depends on the underlying window system, see qwindowdefs.h for the actual definition. If there is no widget with this identifier, 0 is returned.

窗口标识符类型依赖于底层窗口系统，参见 qwindowdefs.h 代表实际的定义。如果没有这个标识符的 widget，则返回 0。

[virtual protected] void QWidget::focusInEvent([QFocusEvent](#) *event)

This event handler can be reimplemented in a subclass to receive keyboard focus events (focus received) for the widget. The event is passed in the event parameter

A widget normally must [setFocusPolicy\(\)](#) to something other than [Qt::NoFocus](#) in order to receive focus events. (Note that the application programmer can call [setFocus\(\)](#) on any widget, even those that do not normally accept focus.)

The default implementation updates the widget (except for windows that do not specify a [focusPolicy\(\)](#)).

这个事件处理程序可以在子类中重新实现，以接收 `widget` 的键盘焦点事件（焦点接收）。事件传递 `event` 参数。一个 `widget` 通常必须 `setFocusPolicy()` 到 `Qt::NoFocus` 来接收焦点事件。（注意，应用程序编程人员可以在任何小部件上调用 `setFocus()`，即使是那些通常不接受焦点的部件。）默认的实现更新 `widget`（除了没有指定 `focusPolicy()` 的窗口）。

See also [focusOutEvent\(\)](#), [setFocusPolicy\(\)](#), [keyPressEvent\(\)](#), [keyReleaseEvent\(\)](#), [event\(\)](#), and [QFocusEvent](#).

[protected] `bool QWidget::focusNextChild()`

Finds a new widget to give the keyboard focus to, as appropriate for **Tab**, and returns true if it can find a new widget, or false if it can't.

找到一个新的 `widget`，让键盘聚焦到，适合于 **Tab**，如果它能找到一个新的 `widget`，则返回 `true`。

See also [focusPreviousChild\(\)](#).

[virtual protected] `bool QWidget::focusNextPrevChild(bool next)`

Finds a new widget to give the keyboard focus to, as appropriate for **Tab** and **Shift+Tab**, and returns true if it can find a new widget, or false if it can't.

找到一个新的 `widget`，焦点切换适用于 **Tab** 和 **Shift+Tab**，如果它能找到一个新的 `widget`，则返回 `true`。

If *next* is true, this function searches forward, if *next* is false, it searches backward. # 如果 *next* 返回 `True` 程序向前查找 `widget`，否则向后

Sometimes, you will want to reimplement this function. For example, a web browser might reimplement it to move its "current active link" forward or backward, and call `focusNextPrevChild()` only when it reaches the last or first link on the "page".

有时，您将希望重新实现这个功能。例如，web 浏览器可能会重新实现它，以将其“当前活动链接”向前或向后移动，并且只有当它到达“page”上的最后或第一个链接时才调用 `focusNextPrevChild()`。

Child widgets call `focusNextPrevChild()` on their parent widgets, but only the window that contains the child widgets decides where to redirect focus. By reimplementing this function for an object, you thus gain control of focus traversal for all child widgets.

子 `widget` 在它们的父 `widget` 上调用 `focusNextPrevChild()`，但是只有包含子 `widget` 的窗口决定了哪里重定向焦点。通过重新实现对象的这个功能，您就可以获得所有子 `widget` 的焦点遍历的控制。通过重新实现对象的这个功能，您就可以获得所有子部件的焦点遍历的控制。

See also [focusNextChild\(\)](#) and [focusPreviousChild\(\)](#).

[virtual protected] `void QWidget::focusOutEvent(QFocusEvent *event)`

This event handler can be reimplemented in a subclass to receive keyboard focus events (focus lost) for the widget. The events is passed in the *event* parameter.

A widget normally must [setFocusPolicy\(\)](#) to something other than [Qt::NoFocus](#) in order to receive focus events. (Note that the application programmer can call [setFocus\(\)](#) on any widget, even those that do not normally accept focus.)

The default implementation updates the widget (except for windows that do not specify a [focusPolicy\(\)](#)).

这个事件处理程序可以在子类中重新实现，以接收 widget 的焦点事件（焦点丢失）。事件传递 event 参数。一个 widget 通常必须 [setFocusPolicy\(\)](#) 到 [Qt::NoFocus](#)，以便接收焦点事件。（注意，应用程序编程人员可以在任何 widget 上调用 [setFocus\(\)](#)，即使是那些通常不接受焦点的部件。）默认的实现更新小部件（除了没有指定 [focusPolicy\(\)](#) 的窗口）。

See also [focusInEvent\(\)](#), [setFocusPolicy\(\)](#), [keyPressEvent\(\)](#), [keyReleaseEvent\(\)](#), [event\(\)](#), and [QFocusEvent](#).

[protected] bool QWidget::focusPreviousChild()

Finds a new widget to give the keyboard focus to, as appropriate for **Shift+Tab**, and returns true if it can find a new widget, or false if it can't.

找到一个新的 widget，以使键盘聚焦，适合于 Shift+Tab，如果它能找到新的 widget，则返回 true;如果它不能，则返回 false。

See also [focusNextChild\(\)](#).

[QWidget](#) *QWidget::focusProxy() const

Returns the focus proxy, or 0 if there is no focus proxy. # 返回一个焦点代理或 0(没有焦点代理)

See also [setFocusProxy\(\)](#).

[QWidget](#) *QWidget::focusWidget() const

Returns the last child of this widget that [setFocus](#) had been called on. For top level widgets this is the widget that will get focus in case this window gets activated

This is not the same as [QApplication::focusWidget\(\)](#), which returns the focus widget in the currently active window.

返回这个 widget 的最后一个子 widget，setFocus 已经被调用了。对于顶层窗口，这是一个 widget，它会得到焦点，以防这个窗口被激活

[QFontInfo](#) QWidget::fontInfo() const

Returns the font info for the widget's current font. Equivalent to `QFontInfo(widget->font())`. # 返回 widget 当前字体的字体信息。

See also [font\(\)](#), [fontMetrics\(\)](#), and [setFont\(\)](#).

[QFontMetrics](#) QWidget::fontMetrics() const

Returns the font metrics for the widget's current font. Equivalent to `QFontMetrics(widget->font())`. # 返回当前 widget 的字体大小

See also [font\(\)](#), [fontInfo\(\)](#), and [setFont\(\)](#).

QPalette::ColorRole QWidget::foregroundRole() const

Returns the foreground role. # 返回 widget 的前景

The foreground role defines the color from the widget's [palette](#) that is used to draw the foreground.

If no explicit foreground role is set, the function returns a role that contrasts with the background role.

前景角色定义了用于绘制前景的 widget 调色板中的颜色。如果没有设定明确的前景角色，则该函数返回与背景角色对比的角色。

See also [setForegroundRole\(\)](#) and [backgroundRole\(\)](#).

void QWidget::getContentsMargins(int *left, int *top, int *right, int *bottom) const

Returns the widget's contents margins for *left*, *top*, *right*, and *bottom*. # 返回 widget 的内边距(left, top, right, bottom)

See also [setContentsMargins\(\)](#) and [contentsRect\(\)](#).

[QPixmap](#) QWidget::grab(const [QRect](#) &rectangle = QRect(QPoint(0, 0), QSize(-1, -1)))

Renders the widget into a pixmap restricted by the given *rectangle*. If the widget has any children, then they are also painted in the appropriate positions.

If a rectangle with an invalid size is specified (the default), the entire widget is painted.

This function was introduced in Qt 5.0.

Note: This function can be invoked via the meta-object system and from QML. See [Q_INVOKABLE](#).

将 widget 呈现为受给定矩形限制的像素图。如果 widget 有任何子部件，那么它们也会被绘制在适当的位置。如果指定了一个无效大小的长方形（默认值），则会绘制整个 widget。注意：这个函数可以通过元对象系统和 QML 来调用。参阅 [Q_INVOKABLE](#)。

See also [render\(\)](#) and [QPixmap](#).

void QWidget::grabGesture(Qt::GestureType *gesture*, Qt::GestureFlags *flags* = ...)

Subscribes the widget to a given *gesture* with specific *flags*. # 参阅给予特殊窗口标志的
This function was introduced in Qt 4.6.

See also [ungrabGesture\(\)](#) and [QGestureEvent](#).

void QWidget::grabKeyboard()

Grabs the keyboard input. # 获取键盘输入

This widget receives all keyboard events until [releaseKeyboard\(\)](#) is called; other widgets get no keyboard events at all. Mouse events are not affected. Use [grabMouse\(\)](#) if you want to grab that.

The focus widget is not affected, except that it doesn't receive any keyboard events. [setFocus\(\)](#) moves the focus as usual, but the new focus widget receives keyboard events only after [releaseKeyboard\(\)](#) is called.

If a different widget is currently grabbing keyboard input, that widget's grab is released first.

这个 widget 接收所有的键盘事件，直到调用 [releaseKeyboard\(\)](#) 为止，阻止其他 widget 获取键盘输入。鼠标事件不受影响。如果你想要抓住它，请使用 [grabMouse\(\)](#)。焦点 widget 不受影响，只是它没能接收任何键盘事件。[setFocus\(\)](#) 像往常一样移动焦点，但是新的焦点 widget 只在发布了 [releaseKeyboard\(\)](#) 之后才会收到键盘事件。如果一个不同的 widget 目前正在抓取键盘输入，那么这个 widget 的抓取将首先被释放。

See also [releaseKeyboard\(\)](#), [grabMouse\(\)](#), [releaseMouse\(\)](#), and [focusWidget\(\)](#).

void QWidget::grabMouse()

Grabs the mouse input. # 获取鼠标输入

This widget receives all mouse events until [releaseMouse\(\)](#) is called; other widgets get no mouse events at all. Keyboard events are not affected. Use [grabKeyboard\(\)](#) if you want to grab that.

widget 接收所有的鼠标事件直到 [releaseMouse\(\)](#) 被调用。其他 widget 不能获取鼠标事件。键盘事件无影响。若想抓取键盘事件用 [grabKeyboard\(\)](#)

Warning: Bugs in mouse-grabbing applications very often lock the terminal. Use this function with extreme caution, and consider using the `-nograd` command line option while debugging.

警告：鼠标抓取应用程序中的 **bug** 通常会锁定终端。使用这个函数非常谨慎，并在调试时考虑使用 `-nograd` 命令行选项。

It is almost never necessary to grab the mouse when using Qt, as Qt grabs and releases it sensibly. In particular, Qt grabs the mouse when a mouse button is pressed and keeps it until the last button is released.

在使用 Qt 时，几乎没有必要抓住鼠标，因为 Qt 会抓取并释放它。特别地，当鼠标按下鼠标按钮时，Qt 就会抓取鼠标，直到最后一个按钮被释放。

Note: Only visible widgets can grab mouse input. If `isVisible()` returns false for a widget, that widget cannot call `grabMouse()`.

注意：只有可将 widget 可以抓取鼠标输入。如果 `isVisible()=False`，则不能调用 `grabMouse()`

Note: On Windows, `grabMouse()` only works when the mouse is inside a window owned by the process. On [macOS](#), `grabMouse()` only works when the mouse is inside the frame of that widget.

在 Windows 上 `grabMouse()` 只有当鼠标在进程所拥有的窗口时工作。在 macOS 中，只有当鼠标在这个 widget 的框架内时工作

See also [releaseMouse\(\)](#), [grabKeyboard\(\)](#), and [releaseKeyboard\(\)](#).

void QWidget::grabMouse(const [QCursor](#) &cursor)

This function overloads [grabMouse\(\)](#). # 重新实现 `grabMouse()`

Grabs the mouse input and changes the cursor shape. # 抓取鼠标输入并更改鼠标指针

The cursor will assume shape *cursor* (for as long as the mouse focus is grabbed) and this widget will be the only one to receive mouse events until [releaseMouse\(\)](#) is called().

光标将 [QCursor](#) 形状（只要鼠标的焦点被抓取），这个 widget 将是唯一接收鼠标事件的 widget，直到 `releaseMouse()` 被调用。

Warning: Grabbing the mouse might lock the terminal.

警告：抓取鼠标将锁定终端

Note: See the note in [QWidget::grabMouse\(\)](#). # 参阅 `QWidget::grabMouse()`

See also [releaseMouse\(\)](#), [grabKeyboard\(\)](#), [releaseKeyboard\(\)](#), and [setCursor\(\)](#).

int QWidget::grabShortcut(const [QKeySequence](#) &key, Qt::ShortcutContext context = Qt::WindowShortcut)

Adds a shortcut to Qt's shortcut system that watches for the given *key* sequence in the given *context*. If the *context* is [Qt::ApplicationShortcut](#), the shortcut applies to the application as a whole. Otherwise, it is either local to this widget, [Qt::WidgetShortcut](#), or to the window itself, [Qt::WindowShortcut](#).

为 Qt 的快捷系统添加了一个快捷方式，它可以在给定的上下文中观察给定的键序列。如果上下文是 Qt::ApplicationShortcut，这个快捷方式适用于整个应用程序。否则，它是这个 widget 的本地，Qt::WidgetShortcut，或者窗口本身，Qt::WindowShortcut。

Constant	Value	Description
Qt::WidgetShortcut	0	The shortcut is active when its parent widget has focus. # 该其 widget 是键盘焦点时激活该快捷键；
Qt::WidgetWithChildrenShortcut	3	The shortcut is active when its parent widget, or any of its children has focus. Children which are top-level widgets, except pop-ups, are not affected by this shortcut context. # 当它的父 widget 或它的任何一个子 widget 都有焦点时，快捷键激活。除了弹出窗口之外，顶级窗口的子 widget 不会受到这个快捷上下文的影响；
Qt::WindowShortcut	1	The shortcut is active when its parent widget is a logical subwidget of the active top-level window. # 当其父 widget 是活动顶层窗口的逻辑子 widget 时，快捷键激活；
Qt::ApplicationShortcut	2	The shortcut is active when one of the applications windows are active. # 整个应用程序在窗口活动时，快捷键激活；

If the same *key* sequence has been grabbed by several widgets, when the *key* sequence occurs a [QEvent::Shortcut](#) event is sent to all the widgets to which it applies in a non-deterministic order, but with the "ambiguous" flag set to true.

如果相同的快捷键被几个 widget 捕获，当关键序列发生 Qevent::Shortcut 事件被发送到它应用于非确定性顺序的所有 widget，但是将“模糊”的标志设置为 true。

Warning: You should not normally need to use this function; instead create [QActions](#) with the shortcut key sequences you require (if you also want equivalent menu options and toolbar buttons), or create [QShortcuts](#) if you just need key sequences. Both [QAction](#) and [QShortcut](#) handle all the

event filtering for you, and provide signals which are triggered when the user triggers the key sequence, so are much easier to use than this low-level function.

警告：您通常不需要使用这个函数;相反，用你需要的快捷键创建 [QAction](#)（如果你想要同样的菜单选项和工具栏按钮），或者在你只需要关键序列时创建 [QShortcut](#)。[QAction](#) 和 [QShortcut](#) 都为您处理所有的事件过滤，并提供当用户触发快捷键时触发的信号，因此比这个低级功能更容易使用。

See also [releaseShortcut\(\)](#) and [setShortcutEnabled\(\)](#).

[QGraphicsEffect](#) *QWidget::graphicsEffect() const

The graphicsEffect function returns a pointer to the widget's graphics effect. # [graphicsEffect](#) 函数返回一个指向 widget 的 graphics 效果的指针

If the widget has no graphics effect, 0 is returned. # 如果 widget 没有 graphics 影响则返回 0

This function was introduced in Qt 4.6. # 该功能于 Qt 4.6 引用

See also [setGraphicsEffect\(\)](#).

[QGraphicsProxyWidget](#) *QWidget::graphicsProxyWidget() const

Returns the proxy widget for the corresponding embedded widget in a graphics view; otherwise returns 0.

返回代理 widget 相对于嵌入式 widget 的视觉效果;否则返回 0。

This function was introduced in Qt 4.5. # 该功能于 Qt 4.5 引用

See also [QGraphicsProxyWidget::createProxyForChildWidget\(\)](#) and [QGraphicsScene::addWidget\(\)](#).

bool QWidget::hasEditFocus() const

Returns true if this widget currently has edit focus; otherwise false. # 如果这个 widget 当前有编辑焦点，则返回 True;否则 False。

This feature is only available in Qt for Embedded Linux. # 该特性仅适用于嵌入式 Linux 的 Qt

See also [setEditFocus\(\)](#) and [QApplication::keypadNavigationEnabled\(\)](#).

`[virtual]bool QWidget::hasHeightForWidth() const`

Returns true if the widget's preferred height depends on its width; otherwise returns false. # 如果 widget 的首选高度取决于它的宽度，则返回 True

This function was introduced in Qt 5.0. # 该功能于 Qt 5.0 引入

`[virtual]int QWidget::heightForWidth(int w) const`

Returns the preferred height for this widget, given the width w. # 首选高度取决于宽度，返回宽度 w

If this widget has a layout, the default implementation returns the layout's preferred height. if there is no layout, the default implementation returns -1 indicating that the preferred height does not depend on the width.

如果这个 widget 有一个布局，默认的实现会返回布局的首选高度。如果没有布局，默认的实现返回-1 表示首选高度不依赖于宽度。

`[slot]void QWidget::hide()`

Hides the widget. This function is equivalent to [setVisible\(false\)](#). # 隐藏 widget，相对于 [setVisible\(False\)](#)

Note: If you are working with [QDialog](#) or its subclasses and you invoke the [show\(\)](#) function after this function, the dialog will be displayed in its original position.

注意：如果您使用 [QDialog](#) 或它的子类，并且在这个函数之后调用 [show\(\)](#) 函数，对话框将显示在它的原始位置。

See also [hideEvent\(\)](#), [isHidden\(\)](#), [show\(\)](#), [setVisible\(\)](#), [isVisible\(\)](#), and [close\(\)](#).

`[virtual protected]void QWidget::hideEvent(QHideEvent *event)`

This event handler can be reimplemented in a subclass to receive widget hide events. The event is passed in the *event* parameter.

Hide events are sent to widgets immediately after they have been hidden.

该事件处理可以在子类中重新来接收隐藏事件。事件中传递 *event* 参数。隐藏事件在被隐藏之后立即发送到 widget。

Note: A widget receives spontaneous show and hide events when its mapping status is changed by the window system, e.g. a spontaneous hide event when the user minimizes the window, and a spontaneous show event when the window is restored again. After receiving a spontaneous hide event, a widget is still considered visible in the sense of [isVisible\(\)](#).

注意：当窗口系统改变了它的映射状态时，`widget` 会接收自发的显示和隐藏事件，例如当用户最小化窗口时自动隐藏事件，以及当窗口恢复时的自发显示事件。在接收到自发的隐藏事件之后，在 `isVisible()` 的意义上，`widget` 仍然被认为是可见的。

See also [visible](#), [event\(\)](#), and [QHideEvent](#).

[override virtual protected] **void** QWidget::initPainter([QPainter](#) **painter*) const

Initializes the *painter* pen, background and font to the same as the given widget's. This function is called automatically when the painter is opened on a [QWidget](#). # 初始化绘图笔、背景和字体，与给定的 `widget` 相同。当在 `QWidget` 上打开 `QPainter` 时，这个函数就会自动调用。

[virtual protected] **void** QWidget::inputMethodEvent([QInputMethodEvent](#) **event*)

This event handler, for event *event*, can be reimplemented in a subclass to receive Input Method composition events. This handler is called when the state of the input method changes.

这个事件处理程序，是事件的 `event`，可以在子类中重新实现以接收输入方法组合事件。当输入方法的状态发生变化时，就调用此处理程序。

Note that when creating custom text editing widgets, the [Qt::WA_InputMethodEnabled](#) window attribute must be set explicitly (using the [setAttribute\(\)](#) function) in order to receive input method events.

请注意，在创建定制文本编辑 `widget` 时，必须显式地设置 `Qt::WA_InputMethodEnabled` 窗口属性（使用 `setAttribute()` 函数）来接收输入方法事件。

The default implementation calls `event->ignore()`, which rejects the Input Method event. See the [QInputMethodEvent](#) documentation for more details. # 默认的实现调用 `event->ignore()`，它阻止 Input Method 事件。有关详细信息参见 `QInputMethodEvent` 文档。

See also [event\(\)](#) and [QInputMethodEvent](#).

[virtual] [QVariant](#) QWidget::inputMethodQuery(`Qt::InputMethodQuery` *query*) const

This method is only relevant for input widgets. It is used by the input method to query a set of properties of the widget to be able to support complex input method operations as support for surrounding text and reconversions.

这种方法只适用于输入 `widget`。它被输入方法用于查询 `widget` 的一组属性，以便能够支持复杂的输入方法操作，以支持周围的文本和重新转换。
query specifies which property is queried. # *query* 指定查询那些属性

See also [inputMethodEvent\(\)](#), [QInputMethodEvent](#), [QInputMethodQueryEvent](#), and [inputMethodHints](#).

void QWidget::insertAction([QAction](#) **before*, [QAction](#) **action*)

Inserts the action *action* to this widget's list of actions, before the action *before*. It appends the action if *before* is 0 or *before* is not a valid action for this widget. # 在 widget 的 QAction 列表值的 QAction 前插入一个 QAction。如果 QAction 列表为空或无效则添加一个 QAction。

A [QWidget](#) should only have one of each action. # QWidget 会获得最少一个 QAction

See also [removeAction\(\)](#), [addAction\(\)](#), [QMenu](#), [contextMenuPolicy](#), and [actions\(\)](#).

void QWidget::insertActions([QAction](#) *before, [QList<QAction](#) *> actions)

Inserts the actions *actions* to this widget's list of actions, before the action *before*. It appends the action if *before* is 0 or *before* is not a valid action for this widget. # 在 widget 的 QAction 列表值的 QAction 前插入一个 QAction 列表。如果 QAction 列表为空或无效则添加一个 QAction 列表。

A [QWidget](#) can have at most one of each action. # QWidget 会获得最少一个 QAction

See also [removeAction\(\)](#), [QMenu](#), [insertAction\(\)](#), and [contextMenuPolicy](#).

bool QWidget::isAncestorOf(const [QWidget](#) *child) const

Returns true if this widget is a parent, (or grandparent and so on to any level), of the given *child*, and both widgets are within the same window;

otherwise returns false. # 如果这个 widget 是父 widget（或祖 widget，等等）并且两个小部件都在同一个窗口中，那么返回 True;否则返回 False。

bool QWidget::isEnabledTo(const [QWidget](#) *ancestor) const

Returns true if this widget would become enabled if *ancestor* is enabled; otherwise returns false.

This is the case if neither the widget itself nor every parent up to but excluding *ancestor* has been explicitly disabled.

`isEnabledTo(0)` returns false if this widget or any of its ancestors was explicitly disabled.

The word ancestor here means a parent widget within the same window.

Therefore `isEnabledTo(0)` stops at this widget's window, unlike [isEnabled\(\)](#) which also takes parent windows into considerations.

如果该 widget 启用了，则返回 True;否则返回 False。如果不是 widget 本身，也不是所有的父 widget 都被显式禁用，那么就会出现这种情况。如果这个 widget 或它的祖 widget 显式禁用的话，`isEnabledTo(0)`返回 False。“祖 widget”这个词在同一个窗口中意味着父 widget。因此，与 `isEnabled()`不同，`isEnabledTo(0)`停在这个 widget 的窗口，这也将父 widget 考虑在内。

See also [setEnabled\(\)](#) and [enabled](#).

bool QWidget::isHidden() const

Returns true if the widget is hidden, otherwise returns false. # 如果 widget 被隐藏则返回 True

A hidden widget will only become visible when [show\(\)](#) is called on it. It will not be automatically shown when the parent is shown.

隐藏的 widget 只有调用 show()时才可见。当父 widget 显示时它不会自动显示。

To check visibility, use [!isVisible\(\)](#) instead (notice the exclamation mark). # 检查可见性使用 isVisible()

isHidden() implies [!isVisible\(\)](#), but a widget can be not visible and not hidden at the same time. This is the case for widgets that are children of widgets that are not visible.

isHidden()不同于 isVisible(), 但是 widget 不可能同时既不可见也不隐藏。这是 widget 的情况, 这些 widget 是不可见的 widget 的子 widget。

Widgets are hidden if: # widget 会隐藏如果:

- they were created as independent windows, # 它们被当做独立窗口创建
- they were created as children of visible widgets, # 它们被当做可见 widget 的子 widget
 - [hide\(\)](#) or [setVisible\(false\)](#) was called. # hide() 或者 setVisible(False)被调用

bool QWidget::isVisibleTo(const [QWidget](#) *ancestor) const

Returns true if this widget would become visible if *ancestor* is shown; otherwise returns false.

如果祖 widget 被显示且该 widget 可见, 返回 True

The true case occurs if neither the widget itself nor any parent up to but excluding *ancestor* has been explicitly hidden.

This function will still return true if the widget is obscured by other windows on the screen, but could be physically visible if it or they were to be moved.

如果 widget 本身或任何父 widget 都没有显式隐藏, 那么返回 True。如果 widget 被屏幕上的其他窗口遮住, 这个函数仍然会返回 True, 尽管如果它或它们被移动, 它可能会在物理上可见。

isVisibleTo(0) is identical to [isVisible\(\)](#). # isVisibleTo(0) 等价 isVisible()

See also [show\(\)](#), [hide\(\)](#), and [isVisible\(\)](#).

bool QWidget::isWindow() const

Returns true if the widget is an independent window, otherwise returns false. # 如果 widget 是一个独立窗口返回 True

A window is a widget that isn't visually the child of any other widget and that usually has a frame and a [window title](#).

A window can have a [parent widget](#). It will then be grouped with its parent and deleted when the parent is deleted, minimized when the parent is minimized etc. If supported by the window manager, it will also have a common taskbar entry with its parent.

[QDialog](#) and [QMainWindow](#) widgets are by default windows, even if a parent widget is specified in the constructor. This behavior is specified by the [Qt::Window](#) flag.

窗口是一个 widget，它不是任何其他 widget 的子 widget，它通常有一个布局框架和一个窗口标题。一个窗口可以有一个 [parentWidget\(\)](#)。然后，它将其 [parentWidget\(\)](#) 进行分组，并在 [parentWidget\(\)](#) 被删除时删除，当 [parentWidget\(\)](#) 被最小化时最小化等等。如果被窗口管理器支持，它也将有一个与其 [parentWidget\(\)](#) 一起的通用任务栏条目。[QDialog](#) 和 [QMainWindow](#) 都是默认独立窗口，在构造函数中指定其父 widget 是无效的。这种行为是由 [Qt::Window](#) flag 指定的。

See also [window\(\)](#), [isModal\(\)](#), and [parentWidget\(\)](#).

[virtual protected] void QWidget::keyPressEvent([QKeyEvent](#) *event)

This event handler, for event *event*, can be reimplemented in a subclass to receive key press events for the widget.

A widget must call [setFocusPolicy\(\)](#) to accept focus initially and have focus in order to receive a key press event.

If you reimplement this handler, it is very important that you call the base class implementation if you do not act upon the key.

这个事件处理程序，处理事件的 *event*，可以在子类中重新实现，以接收 widget 的键盘按压事件。一个 widget 必须调用 [setFocusPolicy\(\)](#) 来接受焦点，并已经接收到键盘焦点来处理键盘按压事件。如果你重写该事件，在没有按指定键将调用基类处理。

The default implementation closes popup widgets if the user presses the key sequence for [QKeySequence::Cancel](#) (typically the Escape key).

Otherwise the event is ignored, so that the widget's parent can interpret it.

Note that [QKeyEvent](#) starts with [isAccepted\(\)](#) == true, so you do not need to call [QKeyEvent::accept\(\)](#) - just do not call the base class implementation if you act upon the key.

对于弹窗，按下 [QKeySequence::Cancel](#)（通常为 Esc 键）将默认关闭。否则，按其他键将会忽略，那是 widget 的父窗口会处理弹窗。注意，[QKeyEvent](#) 从 [isAccepted\(\)](#) == True 开始，所以您不需要调用 [QKeyEvent::accept\(\)](#)，如果您按键操作，就不要调用基类实现。

See also [keyPressEvent\(\)](#), [setFocusPolicy\(\)](#), [focusInEvent\(\)](#), [focusOutEvent\(\)](#), [event\(\)](#), [QKeyEvent](#), and [Tetrix Example](#).

[virtual protected]void QWidget::keyReleaseEvent([QKeyEvent](#) *event)

This event handler, for event *event*, can be reimplemented in a subclass to receive key release events for the widget.

A widget must [accept focus](#) initially and [have focus](#) in order to receive a key release event.

If you reimplement this handler, it is very important that you call the base class implementation if you do not act upon the key.

这个事件处理程序，处理事件的 *event*，可以在子类中重新实现，用以 *widget* 处理按键弹起事件。*widget* 必须已经接收到焦点才能处理此事件。

The default implementation ignores the event, so that the widget's parent can interpret it. # 默认重新忽略该事件

Note that [QKeyEvent](#) starts with `isAccepted() == true`, so you do not need to call [QKeyEvent::accept\(\)](#) - just do not call the base class implementation if you act upon the key.

注意，[QKeyEvent](#) 从 `isAccept() == True` 开始，所以您不需要调用 [QKeyEvent::accept\(\)](#)，如果您按键操作，就不要调用基类实现。

See also [keyPressEvent\(\)](#), [QEvent::ignore\(\)](#), [setFocusPolicy\(\)](#), [focusInEvent\(\)](#), [focusOutEvent\(\)](#), [event\(\)](#), and [QKeyEvent](#).

[static][QWidget](#) *QWidget::keyboardGrabber()

Returns the widget that is currently grabbing the keyboard input. # 返回当前抓取键盘输入的 *widget*

If no widget in this application is currently grabbing the keyboard, 0 is returned. # 如果应用程序中没有 *widget* 在抓取键盘输入则返回 0

See also [grabMouse\(\)](#) and [mouseGrabber\(\)](#).

[QLayout](#) *QWidget::layout() const

Returns the layout manager that is installed on this widget, or 0 if no layout manager is installed. # 返回 *widget* 内的布局管理器。若无则返回 0

The layout manager sets the geometry of the widget's children that have been added to the layout.

布局管理器设置 *widget* 内在布局内的子 *widget* 的几何构型

See also [setLayout\(\)](#), [sizePolicy\(\)](#), and [Layout Management](#).

[virtual protected]void QWidget::leaveEvent([QEvent](#) *event)

This event handler can be reimplemented in a subclass to receive widget leave events which are passed in the *event* parameter. A leave event is sent to the widget when the mouse cursor leaves the widget.

这个事件可以在子类中重写，使 widget 可以接收到离开事件。leave event 在鼠标离开 widget 是发出。

See also [enterEvent\(\)](#), [mouseMoveEvent\(\)](#), and [event\(\)](#).

[slot]void QWidget::lower()

Lowers the widget to the bottom of the parent widget's stack. # 窗口被其他窗口堆叠下置

After this call the widget will be visually behind (and therefore obscured by) any overlapping sibling widgets.

在这个调用之后，widget 将会在可视的放置在其他窗口后面（被遮盖）。

See also [raise\(\)](#) and [stackUnder\(\)](#).

[QPoint](#) QWidget::mapFrom(const [QWidget](#) *parent, const [QPoint](#) &pos) const

Translates the widget coordinate *pos* from the coordinate system of *parent* to this widget's coordinate system. The *parent* must not be 0 and must be a parent of the calling widget.

将 widget 坐标 pos 从父 widget 的坐标系统转换为这个小部件的坐标系统。因此父 widget 参数 parent != None。

See also [mapTo\(\)](#), [mapFromParent\(\)](#), [mapFromGlobal\(\)](#), and [underMouse\(\)](#).

[QPoint](#) QWidget::mapFromGlobal(const [QPoint](#) &pos) const

Translates the global screen coordinate *pos* to widget coordinates. # 将全屏坐标 pos 转换为 widget 坐标。

See also [mapToGlobal\(\)](#), [mapFrom\(\)](#), and [mapFromParent\(\)](#).

[QPoint](#) QWidget::mapFromParent(const [QPoint](#) &pos) const

Translates the parent widget coordinate *pos* to widget coordinates. # 将父 widget 坐标 pos 转换为 widget 坐标

Same as [mapFromGlobal\(\)](#) if the widget has no parent.

See also [mapToParent\(\)](#), [mapFrom\(\)](#), [mapFromGlobal\(\)](#), and [underMouse\(\)](#).

[QPoint](#) QWidget::mapTo(const [QWidget](#) *parent, const [QPoint](#) &pos) const

Translates the widget coordinate *pos* to the coordinate system of *parent*. The *parent* must not be 0 and must be a parent of the calling widget.

转换 widget 的坐标 pos 到 parent 坐标。(parent != None)

See also [mapFrom\(\)](#), [mapToParent\(\)](#), [mapToGlobal\(\)](#), and [underMouse\(\)](#).

[QPoint](#) QWidget::mapToGlobal(const [QPoint](#) &pos) const

Translates the widget coordinate *pos* to global screen coordinates. For example, `mapToGlobal(QPoint(0,0))` would give the global coordinates of the top-left pixel of the widget.

将 widget 坐标 pos 转换为全屏坐标。例如，`mapToGlobal(QPoint(0, 0))`将给出 widget 左上角的全屏坐标。

See also [mapFromGlobal\(\)](#), [mapTo\(\)](#), and [mapToParent\(\)](#).

[QPoint](#) QWidget::mapToParent(const [QPoint](#) &pos) const

Translates the widget coordinate *pos* to a coordinate in the parent widget.

给出 widget 点 pos 的相对于父 widget 的坐标

Same as [mapToGlobal\(\)](#) if the widget has no parent. # 如果 widget 没有父 widget 则给出相对全屏的坐标

See also [mapFromParent\(\)](#), [mapTo\(\)](#), [mapToGlobal\(\)](#), and [underMouse\(\)](#).

[QRegion](#) QWidget::mask() const

Returns the mask currently set on a widget. If no mask is set the return value will be an empty region.

返回 widget 当前的 mask。如果 mask 为空则返回空的区域

See also [setMask\(\)](#), [clearMask\(\)](#), [QRegion::isEmpty\(\)](#), and [Shaped Clock Example](#).

[override virtual protected]int QWidget::metric(QPaintDevice::PaintDeviceMetric m) const

Reimplemented from [QPaintDevice::metric\(\)](#). # 重写自 [QPaintDevice::metric\(\)](#)

Internal implementation of the virtual [QPaintDevice::metric\(\)](#) function.

m is the metric to get.

[virtual protected]void QWidget::mouseDoubleClickEvent([QMouseEvent](#) *event)

This event handler, for event *event*, can be reimplemented in a subclass to receive mouse double click events for the widget.

The default implementation calls [mousePressEvent\(\)](#).

事件处理程序，可以在子类中重写用于接收鼠标双击事件。默认调用 [mousePressEvent\(\)](#)

Note: The widget will also receive mouse press and mouse release events in addition to the double click event. It is up to the developer to ensure that the application interprets these events correctly.

注意：除了双击事件之外，**widget** 还将接收鼠标按压和鼠标释放事件。由开发人员来确保应用程序正确地解释这些事件。

See also [mousePressEvent\(\)](#), [mouseReleaseEvent\(\)](#), [mouseMoveEvent\(\)](#), [event\(\)](#), and [QMouseEvent](#).

[static][QWidget](#) *QWidget::mouseGrabber()

Returns the widget that is currently grabbing the mouse input. # 返回当前正在抓取鼠标的 **widget**

If no widget in this application is currently grabbing the mouse, 0 is returned. # 如果当前应用没有抓取鼠标的 **widget** 则返回 0

See also [grabMouse\(\)](#) and [keyboardGrabber\(\)](#).

[virtual protected]void QWidget::mouseMoveEvent([QMouseEvent](#) *event)

This event handler, for event *event*, can be reimplemented in a subclass to receive mouse move events for the widget.

If mouse tracking is switched off, mouse move events only occur if a mouse button is pressed while the mouse is being moved. If mouse tracking is switched on, mouse move events occur even if no mouse button is pressed.

事件处理程序，可以在子类中重写以处理鼠标移动事件。如果鼠标跟踪被关闭，鼠标移动事件只有在鼠标一个键被按键且移动时发送。如果鼠标追踪开启无论是否按键，都会发送

[QMouseEvent::pos\(\)](#) reports the position of the mouse cursor, relative to this widget.

For press and release events, the position is usually the same as the position of the last mouse move event, but it might be different if the user's hand shakes. This is a feature of the underlying window system, not Qt.

`QMouseEvent::pos()` 提供鼠标相对 `widget` 的位置。通过按压和释放事件，这个位置通常与最后一个鼠标移动事件的位置相同，但是如果用户的手抖动，它可能会有所不同。这是底层窗口系统的一个特性，而不是 Qt。

If you want to show a tooltip immediately, while the mouse is moving (e.g., to get the mouse coordinates with [QMouseEvent::pos\(\)](#) and show them as a tooltip), you must first enable mouse tracking as described above. Then, to ensure that the tooltip is updated immediately, you must call [QToolTip::showText\(\)](#) instead of [setToolTip\(\)](#) in your implementation of `mouseMoveEvent()`.

当鼠标移动时，如果您想要显示一个工具提示（例如，为了得到与 `QMouseEvent::pos()` 的鼠标坐标，并将其显示为工具提示），您必须首先启用如上所述的鼠标跟踪。然后，为了确保工具提示立即被更新，您必须调用 `QToolTip::showText()`，而不是在 `mouseMoveEvent()` 中使用 `setToolTip()`。

See also [setMouseTracking\(\)](#), [mousePressEvent\(\)](#), [mouseReleaseEvent\(\)](#), [mouseDoubleClickEvent\(\)](#), [event\(\)](#), [QMouseEvent](#), and [Scribble Example](#).

```
[virtual protected]void QWidget::mousePressEvent(QMouseEvent *event)
```

This event handler, for event *event*, can be reimplemented in a subclass to receive mouse press events for the widget.

If you create new widgets in the `mousePressEvent()` the [mouseReleaseEvent\(\)](#) may not end up where you expect, depending on the underlying window system (or X11 window manager), the widgets' location and maybe more.

The default implementation implements the closing of popup widgets when you click outside the window. For other widget types it does nothing.

事件处理程序，可以在子类中重写以处理鼠标按压事件。如果你在 `widget` 中创建了 `mousePressEvent()`，`mouseReleaseEvent()` 可能不会结束，根据底层窗口系统（或 X11 窗口管理器），`widget` 的位置，可能更多。对弹窗来说当你点击窗口外的区域是会关闭窗口。

See also [mouseReleaseEvent\(\)](#), [mouseDoubleClickEvent\(\)](#), [mouseMoveEvent\(\)](#), [event\(\)](#), [QMouseEvent](#), and [Scribble Example](#).

```
[virtual protected]void QWidget::mouseReleaseEvent(QMouseEvent *event)
```

This event handler, for event *event*, can be reimplemented in a subclass to receive mouse release events for the widget.

事件处理程序，可以在子类中重写以处理鼠标释放事件。

See also [mousePressEvent\(\)](#), [mouseDoubleClickEvent\(\)](#), [mouseMoveEvent\(\)](#), [event\(\)](#), [QMouseEvent](#), and [Scribble Example](#).

```
void QWidget::move(int x, int y)
```

This is an overloaded function. # 这是一个重载函数

This corresponds to `move(QPoint(x, y))`. # 相当于 `move(QPoint(x, y))`

Note: Setter function for property `pos`. # 服务于属性 `pos`

`[virtual protected]void QWidget::moveEvent(QMoveEvent *event)`

This event handler can be reimplemented in a subclass to receive widget move events which are passed in the `event` parameter. When the widget receives this event, it is already at the new position.

事件处理程序，可以在子类中重写用以接收移动事件（通过参数 `event`）。当 `widget` 接收到该事件时，他的坐标已经移动。

The old position is accessible through `QMoveEvent::oldPos()`.

See also `resizeEvent()`, `event()`, `move()`, and `QMoveEvent`.

`[virtual protected]bool QWidget::nativeEvent(const QByteArray &eventType, void*message, long *result)`

This special event handler can be reimplemented in a subclass to receive native platform events identified by `eventType` which are passed in the `message` parameter.

这是一个特殊的事件处理函数，可以在子类中重写用以接收由 `eventType` 识别的本地平台事件。

In your reimplementation of this function, if you want to stop the event being handled by Qt, return true and set `result`. If you return false, this native event is passed back to Qt, which translates the event into a Qt event and sends it to the widget.

重写这个功能，如果你想要通过 Qt 终止该事件处理，返回 `True` 并设置 `result` 参数。如果你返回 `False`，这个本机事件会传回 Qt，它将事件转换为 Qt 事件并将其发送给 widget。

Note: Events are only delivered to this event handler if the widget is has a native Window handle.

Note: This function superseedes the event filter functions `x11Event()`, `winEvent()` and `macEvent()` of Qt 4.

注意：如果 `widget` 有一个本机窗口句柄，事件只传送到这个事件处理程序

注意：这个函数取代了 Qt 4 的事件过滤器函数 `x11Event()`、`winEvent()`和 `macEvent()`

Platform	Event Type Identifier	Message Type	Result Type
Windows	"windows_generic_MSG"	MSG *	LRESULT
macOS	"NSEvent"	NSEvent *	

[QWidget](#) *QWidget::nativeParentWidget() const

Returns the native parent for this widget, i.e. the next ancestor widget that has a system identifier, or 0 if it does not have any native parent.

为这个 widget 返回父窗口，也就是具有系统标识符的下一个祖 widget，如果没有任何父窗口，则为 0。

This function was introduced in Qt 4.4.

See also [effectiveWinId\(\)](#).

[QWidget](#) *QWidget::nextInFocusChain() const

Returns the next widget in this widget's focus chain. # 返回焦点链上的下一个 widget，当切换焦点时的下一个焦点

See also [previousInFocusChain\(\)](#).

void QWidget::overrideWindowFlags(Qt::WindowFlags flags)

Sets the window flags for the widget to *flags*, *without* telling the window system. # 为 widget 设置窗口标记 flags，不告诉窗口系统

Warning: Do not call this function unless you really know what you're doing.

警告：不要调用这个函数，除非你真正的知道你能用他干什么

See also [setWindowFlags\(\)](#).

[override virtual] [QPaintEngine](#) *QWidget::paintEngine() const

Reimplemented from [QPaintDevice::paintEngine\(\)](#). # 重写的 QPaintDevice::paintEngine()

Returns the widget's paint engine. # 返回 widget 的 paint engine

Note that this function should not be called explicitly by the user, since it's meant for reimplementation purposes only. The function is called by Qt internally, and the default implementation may not always return a valid pointer.

注意，这个函数不应该被用户显式地调用，因为它只用于重新实现目的。这个函数在内部被 Qt 调用，默认的实现可能并不总是返回一个有效的指针

[virtual protected] void QWidget::paintEvent([QPaintEvent](#) *event)

This event handler can be reimplemented in a subclass to receive paint events passed in *event*.

事件处理函数，可以在子类中重写去接收 event 绘图事件。

A paint event is a request to repaint all or part of a widget. It can happen for one of the following reasons:

一个绘画事件是重新绘制 widget 的全部或部分的请求。它可能发生在以下情况：

- [repaint\(\)](#) or [update\(\)](#) was invoked, # [repaint\(\)](#)或[update\(\)](#)被调用
- the widget was obscured and has now been uncovered, or # widget 被覆盖或者覆盖后出现
- many other reasons. # 其他的多种原因

Many widgets can simply repaint their entire surface when asked to, but some slow widgets need to optimize by painting only the requested region: [QPaintEvent::region\(\)](#). This speed optimization does not change the result, as painting is clipped to that region during event processing. [QListView](#) and [QTableView](#) do this, for example.

Qt also tries to speed up painting by merging multiple paint events into one. When [update\(\)](#) is called several times or the window system sends several paint events, Qt merges these events into one event with a larger region (see [QRegion::united\(\)](#)). The [repaint\(\)](#) function does not permit this optimization, so we suggest using [update\(\)](#) whenever possible.

When the paint event occurs, the update region has normally been erased, so you are painting on the widget's background.

当被要求时，widget 可以简单地重新绘制它们的整个表面，但是一些缓慢的 widget 需要通过只绘制被请求的区域：QPaintEvent::region()来进行优化。这种速度优化不会改变结果，因为在事件处理过程中，绘画被剪切到那个区域。例如，QListView 和 QTableView 会这样做。Qt 还试图通过将多个绘图事件合并成一个来加快绘画速度。当 update()被多次调用或窗口系统发送几个绘图事件时，Qt 将这些事件合并到一个更大的区域（参见 QRegion::united()）repaint()函数禁用这种优化，所以我们建议尽可能使用 update()。当绘图事件发生时，更新区域通常被擦除，所以您在 widget 的背景上进行绘画。

The background can be set using [setBackgroundRole\(\)](#) and [setPalette\(\)](#). # 背景可以通过 setBackgroundRole() 和 setPalette()设置

Since Qt 4.0, [QWidget](#) automatically double-buffers its painting, so there is no need to write double-buffering code in paintEvent() to avoid flicker.

自从 Qt 4.0 开始 QWidget 自动采用双缓冲绘图，不需要在 paintEvent()中写双缓冲代码去避免闪烁。

Note: Generally, you should refrain from calling [update\(\)](#) or [repaint\(\)](#) inside a paintEvent(). For example, calling [update\(\)](#) or [repaint\(\)](#) on children inside a paintEvent() results in undefined behavior; the child may or may not get a paint event.

注意：通常，你需要避免在 paintEvent()中调用 update() 或 repaint()。

Warning: If you are using a custom paint engine without Qt's backingstore, [Qt::WA_PaintOnScreen](#) must be set.

Otherwise, [QWidget::paintEngine\(\)](#) will never be called; the backingstore will be used instead.

警告：如果你想要用超出 Qt's backingstore 的自定义 paint engine，Qt::WA_PaintOnScreen 必须被设置。此外 QWidget::paintEngine()不会调用。

See also [event\(\)](#), [repaint\(\)](#), [update\(\)](#), [QPainter](#), [QPixmap](#), [QPaintEvent](#), and [Analog Clock Example](#).

[QWidget](#) *QWidget::parentWidget() const

Returns the parent of this widget, or 0 if it does not have any parent widget. # 返回该窗口的父窗口，如果无父窗口则返回 0

[QWidget](#) *QWidget::previousInFocusChain() const

The previousInFocusChain function returns the previous widget in this widget's focus chain. # 返回焦点链上的前一个 widget

This function was introduced in Qt 4.6. # 在 Qt 4.6 中引进

See also [nextInFocusChain\(\)](#).

[slot]void QWidget::raise()

Raises this widget to the top of the parent widget's stack. # 将窗口前置

After this call the widget will be visually in front of any overlapping sibling widgets. # 调用该函数会是将该窗口置于最前激活并可见

Note: When using [activateWindow\(\)](#), you can call this function to ensure that the window is stacked on top.

贴士：当使用 [activateWindow\(\)](#)，你可以使用该函数确保窗口置于桌面最前端

See also [lower\(\)](#) and [stackUnder\(\)](#).

void QWidget::releaseKeyboard()

Releases the keyboard grab. # 释放键盘抓取

See also [grabKeyboard\(\)](#), [grabMouse\(\)](#), and [releaseMouse\(\)](#).

void QWidget::releaseMouse()

Releases the mouse grab. # 释放鼠标抓取

See also [grabMouse\(\)](#), [grabKeyboard\(\)](#), and [releaseKeyboard\(\)](#).

void QWidget::releaseShortcut(int *id*)

Removes the shortcut with the given *id* from Qt's shortcut system. The widget will no longer receive [QEvent::Shortcut](#) events for the shortcut's key sequence (unless it has other shortcuts with the same key sequence).

从 Qt 的快捷系统中删除带有给定 ID 的快捷方式。widget 将不再接收该快捷键 QEvent::Shortcut 事件（除非它有其他的快捷键，具有相同的键序列）。

Warning: You should not normally need to use this function since Qt's shortcut system removes shortcuts automatically when their parent widget is destroyed. It is best to use [QAction](#) or [QShortcut](#) to handle shortcuts, since they are easier to use than this low-level function. Note also that this is an expensive operation.

警告：你通常不需要使用该功能 Qt 在父 widget 被销毁时自动清除快捷键。该功能最好用在 QAction 或 QShortcut 来处理快捷键，它们比这个低级函数更容易使用，还要注意的，这是一个很消耗资源的操作。

See also [grabShortcut\(\)](#) and [setShortcutEnabled\(\)](#).

void QWidget::removeAction([QAction](#) **action*)

Removes the action *action* from this widget's list of actions. # 从 widget 的 QAction 中移除特定的 action

See also [insertAction\(\)](#), [actions\(\)](#), and [insertAction\(\)](#).

void QWidget::render([QPaintDevice](#) **target*, const [QPoint](#) &*targetOffset* = QPoint(), const [QRegion](#) &*sourceRegion* = QRegion(), [QWidget::RenderFlags](#) *renderFlags* = ...)

Renders the *sourceRegion* of this widget into the *target* using *renderFlags* to determine how to render. Rendering starts at *targetOffset* in the *target*. For example:

使用 renderFlags 将这个小部件的 sourceRegion 渲染给 target，以决定如何渲染。渲染从 target 的 targetOffset 开始。例如：

```
QPixmap pixmap(widget->size());
```

```
widget->render(&pixmap);
```

If *sourceRegion* is a null region, this function will use [QWidget::rect\(\)](#) as the region, i.e. the entire widget.

如果 *sourceRegion* 是 null 区域，这个函数会使用 [QWidget::rect\(\)](#) 作为区域。

Ensure that you call [QPainter::end\(\)](#) for the *target* device's active painter (if any) before rendering. For example:

确保在渲染之前，你调用了 [QPainter::end\(\)](#)。例如：

```
QPainter painter(this);
```

```
...
```

```
painter.end();
```

```
myWidget->render(this);
```

Note: To obtain the contents of a [QOpenGLWidget](#), use [QOpenGLWidget::grabFramebuffer\(\)](#) instead.

贴士：要获得 [QOpenGLWidget](#) 的内容，请使用 [QOpenGLWidget::grabFramebuffer\(\)](#)

Note: To obtain the contents of a [QGLWidget](#) (deprecated), use [QGLWidget::grabFramebuffer\(\)](#) or [QGLWidget::renderPixmap\(\)](#) instead.

贴士：要获得 [QGLWidget](#) 的内容，请使用 [QGLWidget::grabFramebuffer\(\)](#) 或 [QGLWidget::renderPixmap\(\)](#)

This function was introduced in Qt 4.3. # 该功能从 Qt 4.3 引进

`void QWidget::render(QPainter *painter, const QPoint &targetOffset = QPoint(),`

`const QRegion &sourceRegion = QRegion(), QWidget::RenderFlags renderFlags = ...)`

This is an overloaded function. # 这是重写函数

Renders the widget into the *painter*'s [QPainter::device\(\)](#). # 通过 [QPainter::device\(\)](#) 渲染 widget

Transformations and settings applied to the *painter* will be used when rendering. # 应用于 painter 的转换和设置将在渲染时使用

Note: The *painter* must be active. On [macOS](#) the widget will be rendered into a [QPixmap](#) and then drawn by the *painter*.

提示：painter 必须是激活的。在 macOS 上 widget 会被 QPixmap 渲染然后在进行交给 painter

See also [QPainter::device\(\)](#).

[slot] `void QWidget::repaint()`

Repaints the widget directly by calling [paintEvent\(\)](#) immediately, unless updates are disabled or the widget is hidden.

立即调用 [paintEvent\(\)](#)重绘 widget, 除非重绘不可用或 widget 是隐藏的

We suggest only using [repaint\(\)](#) if you need an immediate repaint, for example during animation. In almost all circumstances [update\(\)](#) is better, as it permits Qt to optimize for speed and minimize flicker.

我们建议如非必要请勿使用 [repaint\(\)](#), 例如显示动画时。在大多数情况下使用 [update\(\)](#)更好因为它允许 Qt 对速度进行优化, 并最小化闪烁。

Warning: If you call [repaint\(\)](#) in a function which may itself be called from [paintEvent\(\)](#), you may get infinite recursion. The [update\(\)](#) function never causes recursion.

如果你在 [paintEvent\(\)](#)中调用了 [repaint\(\)](#)将会无限递归。[update\(\)](#)不会产生这种循环

See also [update\(\)](#), [paintEvent\(\)](#), and [setUpdatesEnabled\(\)](#).

void QWidget::repaint(int x, int y, int w, int h)

This is an overloaded function. # 这是一个重新函数

This version repaints a rectangle (x, y, w, h) inside the widget. # 在 widget 内重绘(x, y, w, h)区域

If w is negative, it is replaced with width() - x, and if h is negative, it is replaced with height() - y.

如果 w 是负数的, 则 w=width()-x,如果 h 是负数的 h=height()-y

void QWidget::repaint(const [QRect](#) &rect)

This is an overloaded function. # 这是个重写函数

This version repaints a rectangle *rect* inside the widget. # 重绘 widget 内的 [QRect](#)

void QWidget::repaint(const [QRegion](#) &rgn)

This is an overloaded function. # 这是一个重写函数

This version repaints a region *rgn* inside the widget. # 重绘 widget 内的 [QRegion](#)

void QWidget::resize(int w, int h)

This is an overloaded function. # 这是一个重写函数

This corresponds to `resize(QSize(w, h))`. # 相对于调整大小到 `QSize(w, h)`

Note: Setter function for property `size`. # 贴士：用于调整 `size` 属性

[virtual protected] void QWidget::resizeEvent([QResizeEvent](#) *event)

This event handler can be reimplemented in a subclass to receive widget resize events which are passed in the *event* parameter. When `resizeEvent()` is called, the widget already has its new geometry. The old size is accessible through [QResizeEvent::oldSize\(\)](#).

The widget will be erased and receive a paint event immediately after processing the resize event. No drawing need be (or should be) done inside this handler.

这个事件处理器可以在子类中重新实现，以接收在 `event` 参数中传递的 `widget` 的大小调整事件。当调用 `resizeEvent()` 时，`widget` 已经有了它的新几何图形。旧的尺寸可以通过 `QResizeEvent::oldSize()` 来访问。`widget` 将被擦除，并在处理调整大小事件后立即接收绘图事件。在这个处理程序中不需要（或者应该）做任何绘图。

See also [moveEvent\(\)](#), [event\(\)](#), [resize\(\)](#), [QResizeEvent](#), [paintEvent\(\)](#), and [Scribble Example](#).

bool QWidget::restoreGeometry(const [QByteArray](#) &geometry)

Restores the geometry and state of top-level widgets stored in the byte array *geometry*. Returns true on success; otherwise returns false.

If the restored geometry is off-screen, it will be modified to be inside the available screen geometry.

恢复几何形状并且保存顶层窗口的状态到 byte array *geometry*。如果成功返回 `True`。如果恢复的几何图形在屏幕外，它将被修改为在可用的屏幕空间中。

To restore geometry saved using [QSettings](#), you can use code like this: # 恢复几何形状(位置、大小)设置使用 `QSetting`，你可以写以下代码：

```
QSettings settings("MyCompany", "MyApp");
myWidget->restoreGeometry(settings.value("myWidget/geometry").toByteArray());
```

See the [Window Geometry](#) documentation for an overview of geometry issues with windows. # 参阅 Window Geometry 文档
Use [QMainWindow::restoreState\(\)](#) to restore the geometry and the state of toolbars and dock widgets.
使用 QMainWindow::restoreState()去存储几何形状、工具栏、widget 锚点等信息
This function was introduced in Qt 4.2. # 在 Qt 4.2 中介绍

See also [saveGeometry\(\)](#), [QSettings](#), [QMainWindow::saveState\(\)](#), and [QMainWindow::restoreState\(\)](#).

[QByteArray](#) QWidget::saveGeometry() const

Saves the current geometry and state for top-level widgets. # 保存顶层窗口当前的几何形状(pos and size)

To save the geometry when the window closes, you can implement a close event like this:

为了保存窗口关闭时的几何图形，您可以实现如下的关闭事件：

```
void MyWidget::closeEvent(QCloseEvent *event)
{
    QSettings settings("MyCompany", "MyApp");
    settings.setValue("geometry", saveGeometry());
    QWidget::closeEvent(event);
}
```

See the [Window Geometry](#) documentation for an overview of geometry issues with windows. # 参阅 Window Geometry 文档
Use [QMainWindow::saveState\(\)](#) to save the geometry and the state of toolbars and dock widgets.
使用 QMainWindow::restoreState()去存储几何形状、工具栏、widget 锚点等信息
This function was introduced in Qt 4.2. # 在 Qt 4.2 中介绍

See also [restoreGeometry\(\)](#), [QMainWindow::saveState\(\)](#), and [QMainWindow::restoreState\(\)](#).

void QWidget::scroll(int dx, int dy)

Scrolls the widget including its children *dx* pixels to the right and *dy* downward. Both *dx* and *dy* may be negative.

滚动小部件，包括它的子 widget dx 像素到右边和 dy 下边。dx 和 dy 都是负的

After scrolling, the widgets will receive paint events for the areas that need to be repainted. For widgets that Qt knows to be opaque, this is only the newly exposed parts. For example, if an opaque widget is scrolled 8 pixels to the left, only an 8-pixel wide stripe at the right edge needs updating.
在滚动之后，widget 将会收到需要重新绘制的区域的绘图事件。对于 Qt 知道不透明的 widget，这只是新暴露的部分。例如，如果一个不透明的 widget 向左滚动 8 个像素，那么在右边只有一个 8 像素宽的条纹需要更新。

Since widgets propagate the contents of their parents by default, you need to set the [autoFillBackground](#) property, or use [setAttribute\(\)](#) to set the [Qt::WA_OpaquePaintEvent](#) attribute, to make a widget opaque.

For widgets that use contents propagation, a scroll will cause an update of the entire scroll area.

由于 widget 在默认情况下增加了它们的父 widget 的内容，所以您需要设置 autoFillBackground 属性，或者使用 setAttribute() 来设置 Qt::WA_OpaquePaintEvent 属性，以使 widget 不透明。对于使用内容增加的 widget，滚动条将导致整个滚动区域的更新。

See also [Transparency and Double Buffering](#).

void QWidget::scroll(int dx, int dy, const [QRect](#) &r)

This is an overloaded function. # 这是一个重写函数

This version only scrolls *r* and does not move the children of the widget. # 这个版本只滚动 QRect，单不移动 widget 的子 widget。

If *r* is empty or invalid, the result is undefined. # 如果 QRect 为空或者无效，该函数无效

See also [QScrollArea](#).

void QWidget::setAttribute(Qt::WidgetAttribute *attribute*, bool *on* = true)

Sets the attribute *attribute* on this widget if *on* is true; otherwise clears the attribute. # 如果 on=True 设置状态 attribute，否则清除 attribute

See also [testAttribute\(\)](#).

void QWidget::setBackgroundRole(QPalette::ColorRole *role*)

Sets the background role of the widget to *role*. # 设置 widget 背景 role

The background role defines the brush from the widget's [palette](#) that is used to render the background.

背景 role 定义了用于渲染 widget 背景的调色板中的 brush。

If *role* is [QPalette::NoRole](#), then the widget inherits its parent's background role. # 如果 role=QPalette::NoRole, widget 继承父 widget 的背景 role
Note that styles are free to choose any color from the palette. You can modify the palette or set a style sheet if you don't achieve the result you want with setBackgroundRole().

注意, 样式可以自由地从调色板中选择任何颜色。如果您没有通过 setBackgroundRole()实现您想要的结果, 您可以修改调色板或设置样式表。

See also [backgroundRole\(\)](#) and [foregroundRole\(\)](#).

void QWidget::setBaseSize(int *basew*, int *baseh*)

This is an overloaded function. # 这是一个重写函数

This corresponds to [setBaseSize\(QSize\(basew, baseh\)\)](#). Sets the widgets base size to width *basew* and height *baseh*.

这相当于 setBaseSize(QSize(basew, baseh))。

Note: Setter function for property [baseSize](#). # 服务于 baseSize 属性

void QWidget::setContentsMargins(int *left*, int *top*, int *right*, int *bottom*)

Sets the margins around the contents of the widget to have the sizes *left*, *top*, *right*, and *bottom*. The margins are used by the layout system, and may be used by subclasses to specify the area to draw in (e.g. excluding the frame).

Changing the margins will trigger a [resizeEvent\(\)](#).

See also [contentsMargins\(\)](#), [contentsRect\(\)](#), and [getContentsMargins\(\)](#).

void QWidget::setContentsMargins(const [QMargins](#) &*margins*)

This is an overloaded function. # 这是一个重写函数

The [setContentsMargins](#) function sets the margins around the widget's contents. # setContentsMargins 在 widget 四周设置一个 margins

Sets the margins around the contents of the widget to have the sizes determined by *margins*. The margins are used by the layout system, and may be used by subclasses to specify the area to draw in (e.g. excluding the frame).

在 widget 四周设置 margins, 布局系统使用了 margins, 子类可以用来指定要绘制的区域(例如, 不包括框架)。

Changing the margins will trigger a [resizeEvent\(\)](#). # 更改 margins 会引发 `resizeEvent()`

This function was introduced in Qt 4.6. # 该功能自 Qt 4.6 引进

See also [contentsRect\(\)](#) and [getContentsMargins\(\)](#).

[slot]void QWidget::setEnabled(bool *disable*)

Disables widget input events if *disable* is true; otherwise enables input events. # 如果 `daiable=True` 输入事件无效

See the [enabled](#) documentation for more information.

See also [isEnabledTo\(\)](#), [QKeyEvent](#), [QMouseEvent](#), and [changeEvent\(\)](#).

void QWidget::setEditFocus(bool *enable*)

If *enable* is true, make this widget have edit focus, in which case [Qt::Key_Up](#) and [Qt::Key_Down](#) will be delivered to the widget normally; otherwise, [Qt::Key_Up](#) and [Qt::Key_Down](#) are used to change focus.

This feature is only available in Qt for Embedded Linux.

如果 `enable=True`, 使 widget 获取编辑焦点。在这种状态下 `Qt::Key_Up` 和 `Qt::Key_Down` 会被启用, 其他情况下这两个键用来改变焦点。该功能仅用于嵌入式 Linux

See also [hasEditFocus\(\)](#) and [QApplication::keypadNavigationEnabled\(\)](#).

void QWidget::setFixedHeight(int *h*)

Sets both the minimum and maximum heights of the widget to *h* without changing the widths. Provided for convenience.

锁定 widget 的高度为 *h*, 宽度不变。

See also [sizeHint\(\)](#), [minimumSize\(\)](#), [maximumSize\(\)](#), and [setFixedSize\(\)](#).

void QWidget::setFixedSize(const [QSize](#) &*s*)

Sets both the minimum and maximum sizes of the widget to *s*, thereby preventing it from ever growing or shrinking.

This will override the default size constraints set by [QLayout](#).

To remove constraints, set the size to [QWIDGETSIZE_MAX](#).

Alternatively, if you want the widget to have a fixed size based on its contents, you can call [QLayout::setSizeConstraint\(QLayout::SetFixedSize\)](#); # 锁定 widget 的大小为 QSize(w, h), 阻止 widget 的大小发生改变。这将覆盖 QLayout 设定的默认大小的限制并移除 QWIDGETSIZE_MAX 的设置。此外, 如果您希望 widget 基于其内容有一个固定大小, 您可以调用 QLayout::setSizeConstraint(QLayout::SetFixedSize)

See also [maximumSize](#) and [minimumSize](#).

void QWidget::setFixedSize(int w, int h)

This is an overloaded function. # 这是一个重写函数

Sets the width of the widget to *w* and the height to *h*. # 设置宽为 w 高为 h

void QWidget::setFixedWidth(int w)

Sets both the minimum and maximum width of the widget to *w* without changing the heights. Provided for convenience.

锁定 widget 的宽度, 高度不变

See also [sizeHint\(\)](#), [minimumSize\(\)](#), [maximumSize\(\)](#), and [setFixedSize\(\)](#).

void QWidget::setFocus(Qt::FocusReason reason)

Gives the keyboard input focus to this widget (or its focus proxy) if this widget or one of its parents is the [active window](#). The *reason* argument will be passed into any focus event sent from this function, it is used to give an explanation of what caused the widget to get focus. If the window is not active, the widget will be given the focus when the window becomes active.

如果 widget 是激活状态, 将键盘焦点给予 widget。参数 reason 将被发送到该函数, 它用来解释是什么导致了焦点的获得。如果窗口为激活, 将在激活时获取焦点。

First, a focus about to change event is sent to the focus widget (if any) to tell it that it is about to lose the focus. Then focus is changed, a focus out event is sent to the previous focus item and a focus in event is sent to the new item to tell it that it just received the focus. (Nothing happens if the focus in and focus out widgets are the same.)

首选, 焦点改变事件发送到焦点 widget 告诉它将会失去焦点。然后焦点改变, 焦点丢失事件发送到前一个焦点 widget, 焦点进入事件发送到新焦点 widget, 告诉它刚刚接收到焦点。(如果焦点进入 widget 和焦点退出 widget 是相同的则什么都不会发生)

Note: On embedded platforms, [setFocus\(\)](#) will not cause an input panel to be opened by the input method. If you want this to happen, you have to send a [QEvent::RequestSoftwareInputPanel](#) event to the widget yourself.

[setFocus\(\)](#) gives focus to a widget regardless of its focus policy, but does not clear any keyboard grab (see [grabKeyboard\(\)](#)).

Be aware that if the widget is hidden, it will not accept focus until it is shown.

注意：在嵌入式平台上，[setFocus\(\)](#)不会打开一个输入面板。如果您希望打开，您必须将 [QEvent::RequestSoftwareInputPanel](#) 事件发送给 widget。[setFocus\(\)](#)把焦点放在小部件上，不管它的焦点策略是什么，但是不清除任何键盘抓取（参见 [grabKeyboard\(\)](#)）。请注意，如果小部件是隐藏的，那么它将不会接受焦点，直到它被显示出来。

Warning: If you call [setFocus\(\)](#) in a function which may itself be called from [focusOutEvent\(\)](#) or [focusInEvent\(\)](#), you may get an infinite recursion.

警告：如果你在 [focusOutEvent\(\)](#) 或 [focusInEvent\(\)](#)中调用了 [setFocus\(\)](#)将会得到一个无限循环。

See

also [hasFocus\(\)](#), [clearFocus\(\)](#), [focusInEvent\(\)](#), [focusOutEvent\(\)](#), [setFocusPolicy\(\)](#), [focusWidget\(\)](#), [QApplication::focusWidget\(\)](#), [grabKeyboard\(\)](#), [grabMouse\(\)](#), [Keyboard Focus in Widgets](#), and [QEvent::RequestSoftwareInputPanel](#).

[slot]void QWidget::setFocus()

This is an overloaded function. # 这是一个覆写函数

Gives the keyboard input focus to this widget (or its focus proxy) if this widget or one of its parents is the [active window](#).

如果 widget 是激活状态，将键盘焦点给予 widget。参数 reason 将被发送到该函数，它用来解释是什么导致了焦点的获得。如果窗口为激活，将在激活时获取焦点。

void QWidget::setFocusProxy([QWidget](#) *w)

Sets the widget's focus proxy to widget w. If w is 0, the function resets this widget to have no focus proxy. # 将 widget 的焦点代理给 w

Some widgets can "have focus", but create a child widget, such as [QLineEdit](#), to actually handle the focus. In this case, the widget can set the line edit to be its focus proxy.

有些 widget 可以获取焦点，但是创造了一个子 widget 去实际回去焦点。例如 [QlineEdit](#)。在这种情况下这种 line edit 为其焦点代理

[setFocusProxy\(\)](#) sets the widget which will actually get focus when "this widget" gets it. If there is a focus proxy, [setFocus\(\)](#) and [hasFocus\(\)](#) operate on the focus proxy.

[setFocusProxy\(\)](#)设置那些实际去获取焦点的 widget。如果有一个焦点代理，[setFocus\(\)](#)和 [hasFocus\(\)](#)在焦点代理上操作。

See also [focusProxy\(\)](#).

void QWidget::setForegroundRole(QPalette::ColorRole *role*)

Sets the foreground role of the widget to *role*. # 设置前景 *role*

The foreground role defines the color from the widget's [palette](#) that is used to draw the foreground.

If *role* is [QPalette::NoRole](#), the widget uses a foreground role that contrasts with the background role.

Note that styles are free to choose any color from the palette. You can modify the palette or set a style sheet if you don't achieve the result you want with `setForegroundRole()`.

See also [foregroundRole\(\)](#) and [backgroundRole\(\)](#).

void QWidget::setGeometry(int *x*, int *y*, int *w*, int *h*)

This is an overloaded function. # 这是一个重写函数

This corresponds to [setGeometry\(QRect\(*x*, *y*, *w*, *h*\)\)](#). # 相当于 `setGeometry(QRect(x, y, w, h))`

Note: Setter function for property [geometry](#).

void QWidget::setGraphicsEffect([QGraphicsEffect](#) **effect*)

The `setGraphicsEffect` function is for setting the widget's graphics effect. # `setGraphicsEffect` 设置 widget 的图形特效

Sets *effect* as the widget's effect. If there already is an effect installed on this widget, [QWidget](#) will delete the existing effect before installing the new *effect*. # 将 *effect* 设置为 widget 的特效，如果已经存在特效将会删除后设置 *effect* 为特效

If *effect* is the installed effect on a different widget, `setGraphicsEffect()` will remove the effect from the widget and install it on this widget.

[QWidget](#) takes ownership of *effect*. # 如果 *effect* 设置在不同的 widget 上，`setGraphicsEffect()` 将会删除一个并在另一个上设置 *effect*

Note: This function will apply the effect on itself and all its children. # 贴士： *effect* 将会同时设置在 widget 及其子 widget 上

Note: Graphics effects are not supported for OpenGL-based widgets, such as [QGLWidget](#), [QOpenGLWidget](#) and [QQuickWidget](#).

贴士： 图形特效不支持 OpenGL-based widgets，例如 [QGLWidget](#)、[QOpenGLWidget](#) 和 [QQuickWidget](#)

This function was introduced in Qt 4.6.

See also [graphicsEffect\(\)](#).

[slot] `void QWidget::setHidden(bool hidden)`

Convenience function, equivalent to [setVisible\(!hidden\)](#). # 等同于 `setVisible(!hidden)`

See also [isHidden\(\)](#).

`void QWidget::setLayout(QLayout *layout)`

Sets the layout manager for this widget to *layout*. # 设置窗口布局管理器

If there already is a layout manager installed on this widget, [QWidget](#) won't let you install another. You must first delete the existing layout manager (returned by [layout\(\)](#)) before you can call `setLayout()` with the new layout.

如果 widget 已有布局，则不再接受新的布局。你必须先清除布局然后再使用 `setLayout()` 设置新布局

If *layout* is the layout manager on a different widget, `setLayout()` will reparent the layout and make it the layout manager for this widget.

Example: # 如果 layout 是另一个 widget 上的布局管理器，`setLayout()` 将重新调整布局，并使其成为这个 widget 的布局管理器。例如：

```
QVBoxLayout *layout = new QVBoxLayout;
layout->addWidget(formWidget);
setLayout(layout);
```

An alternative to calling this function is to pass this widget to the layout's constructor.

调用这个函数的另一种方法是将这个 widget 传递给布局的构造函数。

The [QWidget](#) will take ownership of *layout*. # widget 或获取 layout

See also [layout\(\)](#) and [Layout Management](#).

`void QWidget::setMask(const QBitmap &bitmap)`

Causes only the pixels of the widget for which *bitmap* has a corresponding 1 bit to be visible. If the region includes pixels outside the [rect\(\)](#) of the widget, window system controls in that area may or may not be visible, depending on the platform.

Note that this effect can be slow if the region is particularly complex.

从像素图的 **alpha** 通道提取可见部分。如果该区域很复杂将导致运行缓慢。

The following code shows how an image with an alpha channel can be used to generate a mask for a widget:

以下代码显示了如何用图片的 **alpha** 通道为 **widget** 设置 **mask**

```
QLabel topLevelLabel;  
QPixmap pixmap(":/images/tux.png");  
topLevelLabel.setPixmap(pixmap);  
topLevelLabel.setMask(pixmap.mask());
```

The label shown by this code is masked using the image it contains, giving the appearance that an irregularly-shaped image is being drawn directly onto the screen.

Masked widgets receive mouse events only on their visible portions. # Masked widgets 只在其可见部分接收鼠标事件

See also [mask\(\)](#), [clearMask\(\)](#), [windowOpacity\(\)](#), and [Shaped Clock Example](#).

void QWidget::setMask(const [QRegion](#) ®ion)

This is an overloaded function. # 这是一个重写函数

Causes only the parts of the widget which overlap *region* to be visible. If the region includes pixels outside the [rect\(\)](#) of the widget, window system controls in that area may or may not be visible, depending on the platform.

Note that this effect can be slow if the region is particularly complex. # 从像素图的 **alpha** 通道提取可见部分。如果该区域很复杂将导致运行缓慢。

See also [windowOpacity](#).

void QWidget::setMaximumSize(int *maxw*, int *maxh*)

This is an overloaded function. # 这是一个重写函数

This function corresponds to [setMaximumSize\(QSize\(*maxw*, *maxh*\)\)](#). Sets the maximum width to *maxw* and the maximum height to *maxh*.

Note: Setter function for property [maximumSize](#). # 锁定 widget 的最大宽度为 *maxw*，最大高度为 *maxh*。该函数设置 **maximumSize** 属性

void QWidget::setMinimumSize(int *minw*, int *minh*)

This is an overloaded function. # 这是一个重写函数

This function corresponds to [setMinimumSize\(QSize\(minw, minh\)\)](#). Sets the minimum width to *minw* and the minimum height to *minh*.

Note: Setter function for property [minimumSize](#). # 锁定 widget 的最小宽度为 minw，最小高度为 minh。该函数设置 maximumSize 属性

void QWidget::setParent([QWidget](#) **parent*)

Sets the parent of the widget to *parent*, and resets the window flags. The widget is moved to position (0, 0) in its new parent.

设置 widget 的父 widget 为 parent 并且移除 widget 的 flags。widget 会出现在新父 widget 的(0, 0)位置

If the new parent widget is in a different window, the reparented widget and its children are appended to the end of the [tab chain](#) of the new parent widget, in the same internal order as before. If one of the moved widgets had keyboard focus, `setParent()` calls [clearFocus\(\)](#) for that widget.

If the new parent widget is in the same window as the old parent, setting the parent doesn't change the tab order or keyboard focus.

If the "new" parent widget is the old parent widget, this function does nothing.

如果 widget 与新父 widget 在不同的窗口中，widget 及其子 widget 会位于新父 widget 的 Tab 焦点链的底部。如果其中一个移动 widget 有键盘焦点，`setParent()`调用该 widget 的 `clearFocus()`。如果位于同一个窗口中，不改变焦点链和焦点。如果 widget 和新父 widget 是同一个则忽略；

Note: The widget becomes invisible as part of changing its parent, even if it was previously visible. You must call [show\(\)](#) to make the widget visible again. # 注意：widget 改变其父元素的一部分变得不可见，即使它以前是可见的。您必须调用 `show()`来使小部件再次可见。

Warning: It is very unlikely that you will ever need this function. If you have a widget that changes its content dynamically, it is far easier to use [QStackedWidget](#).

警告：你不太可能需要这个函数。如果您有一个动态更改其内容的小部件，那么使用 `QStackedWidget` 就会容易得多。

See also [setWindowFlags\(\)](#).

void QWidget::setParent([QWidget](#) **parent*, Qt::WindowFlags *f*)

This is an overloaded function. # 这是一个重写函数

This function also takes widget flags, *f* as an argument. # 该功能设置 f widget 标志

void QWidget::setShortcutAutoRepeat(int *id*, bool *enable* = true)

If *enable* is true, auto repeat of the shortcut with the given *id* is enabled; otherwise it is disabled.

如果 `enable=True`，启用给定 ID 的快捷键自动重复

This function was introduced in Qt 4.2. # 在 Qt 4.2 时引进

See also [grabShortcut\(\)](#) and [releaseShortcut\(\)](#).

void QWidget::setShortcutEnabled(int *id*, bool *enable* = true)

If *enable* is true, the shortcut with the given *id* is enabled; otherwise the shortcut is disabled.

如果 `enable=True`，给定 ID 的快捷键可用

Warning: You should not normally need to use this function since Qt's shortcut system enables/disables shortcuts automatically as widgets become hidden/visible and gain or lose focus. It is best to use [QAction](#) or [QShortcut](#) to handle shortcuts, since they are easier to use than this low-level function.

警告：您通常不需要使用这个函数，因为 Qt 的快捷键系统会自动地启用/禁用快捷键，因为 widget 变得隐藏/可见，并获得或失去焦点。最好使用 [QAction](#) 或 [QShortcut](#) 来处理快捷键，因为它们比这个低级功能更容易使用。

See also [grabShortcut\(\)](#) and [releaseShortcut\(\)](#).

void QWidget::setSizeIncrement(int *w*, int *h*)

This is an overloaded function. # 这是一个重写函数

Sets the x (width) size increment to *w* and the y (height) size increment to *h*. # 将宽度增加 *w*，高度增加 *y*

Note: Setter function for property [sizeIncrement](#). # 服务于参数 `sizeIncrement`

void QWidget::setSizePolicy(QSizePolicy::Policy *horizontal*, QSizePolicy::Policy *vertical*)

This is an overloaded function. # 这是一个重写函数

Sets the size policy of the widget to *horizontal* and *vertical*, with standard stretch and no height-for-width.

设置 widget 尺寸策略为 horizontal and vertical

Note: Setter function for property [sizePolicy](#). # 服务于 sizePolicy 属性

See also [QSizePolicy::QSizePolicy\(\)](#).

void QWidget::setStyle([QStyle](#) *style)

Sets the widget's GUI style to *style*. The ownership of the style object is not transferred. # 设置 widget 的 GUI 样式

If no style is set, the widget uses the application's style, [QApplication::style\(\)](#) instead. # 如果样式未设置, 使用 QApplication::style()来代替

Setting a widget's style has no effect on existing or future child widgets. # 设置 widget 的样式对现有或未来的子 widget 没有影响。

Warning: This function is particularly useful for demonstration purposes, where you want to show Qt's styling capabilities. Real applications should avoid it and use one consistent GUI style instead.

警告: 这个函数对于演示目的特别有用, 您想要显示 Qt 的样式功能。真正的应用程序应该避免使用它, 而是使用一种一致的 GUI 样式。

Warning: Qt style sheets are currently not supported for custom [QStyle](#) subclasses. We plan to address this in some future release.

警告: Qt 样式表目前不支持自定义 QStyle 子类。我们计划在未来的版本中解决这个问题。

See also [style\(\)](#), [QStyle](#), [QApplication::style\(\)](#), and [QApplication::setStyle\(\)](#).

[static]void QWidget::setTabOrder([QWidget](#) *first, [QWidget](#) *second)

Puts the *second* widget after the *first* widget in the focus order. # 使 second widget 在 first widget 之后湖区 Tab 焦点

It effectively removes the *second* widget from its focus chain and inserts it after the *first* widget.

移除 second 在焦点链上的位置并将其插入到 first 之后

Note that since the tab order of the *second* widget is changed, you should order a chain like this:

因焦点 second 在 Tab 焦点链上的位置发生了改变, 焦点链将会变成这样:

```
setTabOrder(a, b); // a to b
setTabOrder(b, c); // a to b to c
setTabOrder(c, d); // a to b to c to d
```

not like this:

```
// WRONG
setTabOrder(c, d); // c to d
```



```
setTabOrder(a, b); // a to b AND c to d
setTabOrder(b, c); // a to b to c, but not c to d
```

If *first* or *second* has a focus proxy, `setTabOrder()` correctly substitutes the proxy. # 如果有焦点代理，同样适用

Note: Since Qt 5.10: A widget that has a child as focus proxy is understood as a compound widget. When setting a tab order between one or two compound widgets, the local tab order inside each will be preserved. This means that if both widgets are compound widgets, the resulting tab order will be from the last child inside *first*, to the first child inside *second*.

贴士：自 Qt 5.10 开始，widget 的子 widget 的作为焦点代理不在使用将会被当做复合 widget 使用。当在 *first* 和 *second* 设置 Tab 焦点切换时，他们自身的内部的焦点链不变。

See also [setFocusPolicy\(\)](#), [setFocusProxy\(\)](#), and [Keyboard Focus in Widgets](#).

void QWidget::setWindowFlag(Qt::WindowType *flag*, bool *on* = true)

Sets the window flag *flag* on this widget if *on* is true; otherwise clears the flag. # 如果 on=True，设置 flag 为 widget 的标志

This function was introduced in Qt 5.9. # 自 Qt 5.9 开始引进

See also [setWindowFlags\(\)](#), [windowFlags\(\)](#), and [windowType\(\)](#).

void QWidget::setWindowRole(const [QString](#) &*role*)

Sets the window's role to *role*. This only makes sense for windows on X11. # 设置窗口 role，进度 X11 系统有效

See also [windowRole\(\)](#).

void QWidget::setWindowState(Qt::WindowStates *windowState*)

Sets the window state to *windowState*. The window state is a OR'ed combination of [Qt::WindowState](#): [Qt::WindowMinimized](#), [Qt::WindowMaximized](#), [Qt::WindowFullScreen](#), and [Qt::WindowActive](#).

If the window is not visible (i.e. [isVisible\(\)](#) returns false), the window state will take effect when [show\(\)](#) is called. For visible windows, the change is immediate. For example, to toggle between full-screen and normal mode, use the following code:

设置窗口状态，该状态可以是 Qt::WindowState: Qt::WindowMinimized, Qt::WindowMaximized, Qt::WindowFullScreen, 和 Qt::WindowActive。如果窗口不可见，在窗口可见是工作。对可见窗口来说会立即改变。例如，使用以下代码，改变全屏和正常状态显示：

```
w->setWindowState(w->windowState() ^ Qt::WindowFullScreen);
```

In order to restore and activate a minimized window (while preserving its maximized and/or full-screen state), use the following:

为了恢复和激活最小化窗口（同时保持其最大化和/或全屏状态），请使用以下代码：

```
w->setWindowState(w->windowState() & ~Qt::WindowMinimized) | Qt::WindowActive);
```

Calling this function will hide the widget. You must call [show\(\)](#) to make the widget visible again.

调用该方法会隐藏窗口，你必须调用 [show\(\)](#) 来显示

Note: On some window systems [Qt::WindowActive](#) is not immediate, and may be ignored in certain cases.

When the window state changes, the widget receives a [changeEvent\(\)](#) of type [QEvent::WindowStateChange](#).

贴士：一些系统 [Qt::WindowActive](#) 不会立即工作，甚至会被忽略。当改变窗口状态是，widget 收到一个类型为 [QEvent::WindowStateChange](#) 的 [changeEvent\(\)](#)

See also [Qt::WindowState](#) and [windowState\(\)](#).

void QWidget::setupUi([QWidget](#) **widget*)

Sets up the user interface for the specified *widget*. # 为指定的 widget 设置用户界面

Note: This function is available with widgets that derive from user interface descriptions created using [uic](#).

贴士：这个功能可以从使用 [uic](#) 创建的用户界面描述中获得

See also [Using a Designer UI File in Your Application](#).

[slot]void QWidget::show()

Shows the widget and its child widgets. # 显示 widget

This is equivalent to calling [showFullScreen\(\)](#), [showMaximized\(\)](#), or [setVisible\(true\)](#), depending on the platform's default behavior for the window flags.

相同功能的有 [showFullScreen\(\)](#)、[showMaximized\(\)](#) 或 [setVisible\(true\)](#) 依赖于系统平台和窗口标志

See also [raise\(\)](#), [showEvent\(\)](#), [hide\(\)](#), [setVisible\(\)](#), [showMinimized\(\)](#), [showMaximized\(\)](#), [showNormal\(\)](#), [isVisible\(\)](#), and [windowFlags\(\)](#).

[virtual protected]void QWidget::showEvent([QShowEvent](#) **event*)

This event handler can be reimplemented in a subclass to receive widget show events which are passed in the *event* parameter.

事件处理函数，可以在子类中重写以处理显示事件

Non-spontaneous show events are sent to widgets immediately before they are shown. The spontaneous show events of windows are delivered afterwards.

Note: A widget receives spontaneous show and hide events when its mapping status is changed by the window system, e.g. a spontaneous hide event when the user minimizes the window, and a spontaneous show event when the window is restored again. After receiving a spontaneous hide event, a widget is still considered visible in the sense of [isVisible\(\)](#).

非自发的 **show** 事件在显示之前被发送到 **widget**。之后，**windows** 的自发显示事件会被交付。

注意：当窗口系统改变了它的映射状态时，**widget** 会接收自发的显示和隐藏事件，例如当用户最小化窗口时自动隐藏事件，以及当窗口恢复时的自发显示事件。在接收到自发的隐藏事件之后，在 [isVisible\(\)](#) 的意义上，**widget** 仍然被认为是可见的。

See also [visible](#), [event\(\)](#), and [QShowEvent](#).

[slot]void QWidget::showFullScreen()

Shows the widget in full-screen mode. # 全屏显示

Calling this function only affects [windows](#). # 该功能仅影响顶层窗口

To return from full-screen mode, call [showNormal\(\)](#). # 调用 [showNormal\(\)](#) 取消全屏状态

Full-screen mode works fine under Windows, but has certain problems under X. These problems are due to limitations of the ICCCM protocol that specifies the communication between X11 clients and the window manager. ICCCM simply does not understand the concept of non-decorated full-screen windows. Therefore, the best we can do is to request a borderless window and place and resize it to fill the entire screen. Depending on the window manager, this may or may not work. The borderless window is requested using MOTIF hints, which are at least partially supported by virtually all modern window managers.

全屏幕模式在 Windows 下运行良好，但在 X 下有一定的问题。这些问题是由于 ICCCM 协议的限制，它指定了 X11 客户端和窗口管理器之间的通信。ICCCM 根本没有全屏窗口的概念。因此，我们所能做的最好的事情就是请求一个无边界的窗口和位置，并调整大小以填充整个屏幕。根据窗口管理器的不同，这可能或可能不起作用。使用主题提示，可以使用无边界窗口，这至少在所有现代窗口管理器中得到了部分支持。

An alternative would be to bypass the window manager entirely and create a window with the [Qt::X11BypassWindowManagerHint](#) flag. This has other severe problems though, like totally broken keyboard focus and very strange effects on desktop changes or when the user raises other windows.

X11 window managers that follow modern post-ICCCM specifications support full-screen mode properly.

另一种方法是完全绕过窗口管理器，并创建一个带有 [Qt::X11BypassWindowManagerHint](#) 标志的窗口。不过，这也有其他严重的问题，比如完全中断的键盘焦点，以及对桌面变化的奇怪影响，或者当用户打开其他窗口时。遵循 post-ICCCM 规范的 X11 窗口管理器可以正确地支持全屏模式

See also [showNormal\(\)](#), [showMaximized\(\)](#), [show\(\)](#), [hide\(\)](#), and [isVisible\(\)](#).

[slot]void QWidget::showMaximized()

Shows the widget maximized. # 最大化显示窗口

Calling this function only affects [windows](#). # 仅影响顶层窗口

On X11, this function may not work properly with certain window managers. See the [Window Geometry](#) documentation for an explanation.

在 X11 中，这个函数可能不能正确地与某些窗口管理器正常工作。参阅 [Window Geometry](#) 获取详情

See also [setWindowState\(\)](#), [showNormal\(\)](#), [showMinimized\(\)](#), [show\(\)](#), [hide\(\)](#), and [isVisible\(\)](#).

[slot]void QWidget::showMinimized()

Shows the widget minimized, as an icon. # 最小化显示，仅显示一个图标

Calling this function only affects [windows](#). # 仅影响顶层窗口

See also [showNormal\(\)](#), [showMaximized\(\)](#), [show\(\)](#), [hide\(\)](#), [isVisible\(\)](#), and [isMinimized\(\)](#).

[slot]void QWidget::showNormal()

Restores the widget after it has been maximized or minimized. # 恢复窗口到设定大小

Calling this function only affects [windows](#). # 仅影响顶层窗口

See also [setWindowState\(\)](#), [showMinimized\(\)](#), [showMaximized\(\)](#), [show\(\)](#), [hide\(\)](#), and [isVisible\(\)](#).

void QWidget::stackUnder([QWidget](#) *w)

Places the widget under w in the parent widget's stack. # 将窗口堆叠在父窗口 w 下

To make this work, the widget itself and w must be siblings. # 为使此设置工作，widget 和 w 必须相关联

See also [raise\(\)](#) and [lower\(\)](#).

[QStyle](#) *QWidget::style() const

See also [QWidget::setStyle\(\)](#), [QApplication::setStyle\(\)](#), and [QApplication::style\(\)](#).

[virtual protected] void QWidget::tabletEvent([QTabletEvent](#) *event)

This event handler, for event *event*, can be reimplemented in a subclass to receive tablet events for the widget.

If you reimplement this handler, it is very important that you [ignore\(\)](#) the event if you do not handle it, so that the widget's parent can interpret it.

The default implementation ignores the event.

If tablet tracking is switched off, tablet move events only occur if the stylus is in contact with the tablet, or at least one stylus button is pressed, while the stylus is being moved. If tablet tracking is switched on, tablet move events occur even while the stylus is hovering in proximity of the tablet, with no buttons pressed.

这个事件处理程序，对于事件事件，可以在子类中重新实现，**widget** 接收的平板事件。如果您重新实现这个处理程序，那么如果您不处理它，那么 **ignore()** 事件是非常重要的，这样 **widget** 的父部件就可以解释它了。默认的实现忽略了事件。如果平板电脑的追踪被关闭，平板电脑的移动事件只会发生在触控笔与平板电脑接触的情况下，或者至少有一个触控笔按钮被按下，而触控笔正在被移动。如果平板电脑的追踪被打开，平板电脑的移动事件就会发生，即使触控笔在平板电脑附近徘徊，没有按下按钮。

See also [QEvent::ignore\(\)](#), [QEvent::accept\(\)](#), [event\(\)](#), [setTabletTracking\(\)](#), and [QTabletEvent](#).

bool QWidget::testAttribute(Qt::WidgetAttribute attribute) const

Returns true if attribute *attribute* is set on this widget; otherwise returns false. # 如果该 **widget** 设置了 **attribute** 属性则返回 True

See also [setAttribute\(\)](#).

bool QWidget::underMouse() const

Returns true if the widget is under the mouse cursor; otherwise returns false. # 如果 **widget** 在鼠标指针下返回真

This value is not updated properly during drag and drop operations. # 在拖放操作期间，该值不改变

See also [enterEvent\(\)](#) and [leaveEvent\(\)](#).

void QWidget::ungrabGesture(Qt::GestureType *gesture*)

Unsubscribes the widget from a given *gesture* type # 该给定的 *gesture* 下消耗 *widget*

This function was introduced in Qt 4.6. # 自 Qt 4.6 开始引进

See also [grabGesture\(\)](#) and [QGestureEvent](#).

[slot]void QWidget::update()

Updates the widget unless updates are disabled or the widget is hidden. # 更新窗口，除非更新被禁用或者 *widget* 被隐藏

This function does not cause an immediate repaint; instead it schedules a paint event for processing when Qt returns to the main event loop. This permits Qt to optimize for more speed and less flicker than a call to [repaint\(\)](#) does.

Calling `update()` several times normally results in just one [paintEvent\(\)](#) call.

Qt normally erases the widget's area before the [paintEvent\(\)](#) call. If the [Qt::WA_OpaquePaintEvent](#) widget attribute is set, the widget is responsible for painting all its pixels with an opaque color.

这个功能不会立即调用绘图事件。相反，当 Qt 返回到主事件循环时，它会安排一个绘图事件进行处理。这使得 Qt 能够以更快的速度和更少的闪烁来优化，而不是调用 `repaint()`。调用 `update()` 几次通常只会产生一个 `paintEvent()` 调用。Qt 通常会在 `paintEvent()` 调用之前擦除 *widget* 的区域。如果 *widget* 设置属性 `Qt::WA_OpaquePaintEvent`，*widget* 用不透明的颜色绘制所有像素。

See also [repaint\(\)](#), [paintEvent\(\)](#), [setUpdatesEnabled\(\)](#), and [Analog Clock Example](#).

void QWidget::update(int *x*, int *y*, int *w*, int *h*)

This is an overloaded function. # 这是一个重写函数

This version updates a rectangle (*x*, *y*, *w*, *h*) inside the widget. # 更新指定区域(*x*, *y*, *w*, *h*)

void QWidget::update(const [QRect](#) &*rect*)

This is an overloaded function.

This version updates a rectangle *rect* inside the widget. # 仅对可见 widget 有效

void QWidget::update(const QRegion &*rgn*)

This is an overloaded function. # 这是一个重写函数

This version repaints a region *rgn* inside the widget. # 更新指定区域 QRegion

void QWidget::updateGeometry()

Notifies the layout system that this widget has changed and may need to change geometry.

通知布局系统这个 widget 已经改变了，可能需要改变几何形状。

Call this function if the sizeHint() or sizePolicy() have changed. # 在 sizeHint() 或 sizePolicy() 被改变事件调用

For explicitly hidden widgets, updateGeometry() is a no-op. The layout system will be notified as soon as the widget is shown.

对于显式隐藏的 widget，updateGeometry() 是无效的。当 widget 显示时，布局系统将会被通知

[protected slot] void QWidget::updateMicroFocus()

Updates the widget's micro focus. # 更新 widget 的微焦点

QRegion QWidget::visibleRegion() const

Returns the unobscured region where paint events can occur. # 显示正在发生绘图事件的区域

For visible widgets, this is an approximation of the area not covered by other widgets; otherwise, this is an empty region.

对于可见的 widget，这是其他为 widget 没有覆盖的区域的近似值；否则，这是一个空区域。

The repaint() function calls this function if necessary, so in general you do not need to call it. # repaint() 会自动调用该函数，所以你不要调用

[virtual protected]void QWidget::wheelEvent([QWheelEvent](#) *event)

This event handler, for event *event*, can be reimplemented in a subclass to receive wheel events for the widget.

这是一个事件处理，你可以在子类中重写该函数来接收鼠标滚轮事件

If you reimplement this handler, it is very important that you [ignore\(\)](#) the event if you do not handle it, so that the widget's parent can interpret it. The default implementation ignores the event.

如果您重新实现这个处理程序，那么如果您不处理它，那么 [ignore\(\)](#) 事件是非常重要的，这样 widget 的父部件就可以解释它了。默认的实现忽略了事件。

See also [QEvent::ignore\(\)](#), [QEvent::accept\(\)](#), [event\(\)](#), and [QWheelEvent](#).

WId QWidget::winId() const

Returns the window system identifier of the widget. # 返回 widget 的窗口系统标记

Portable in principle, but if you use it you are probably about to do something non-portable. Be careful.

If a widget is non-native (alien) and winId() is invoked on it, that widget will be provided a native handle.

This value may change at run-time. An event with type [QEvent::WinIdChange](#) will be sent to the widget following a change in window system identifier.

原则上是可移植的，但如果你使用它，你可能会做一些不可移植的事情。小心些而已。如果一个 widget 是非本地的（外星人）和 winId()在它上被调用，那么这个 widget 将被提供一个本机句柄。这个值可能在运行时改变。在窗口系统标识符发生更改后，[QEvent::WinIdChange](#) 将被发送到 widget。

See also [find\(\)](#).

[QWidget](#) *QWidget::window() const

Returns the window for this widget, i.e. the next ancestor widget that has (or could have) a window-system frame. # 返回 widget 所在的窗口

If the widget is a window, the widget itself is returned. # 如果 widget 是顶层窗口，则返回他自身

Typical usage is changing the window title: # 典型的用法是用来改变窗口标题

```
aWidget->window()->setWindowTitle("New Window Title");
```

See also [isWindow\(\)](#).

[QWindow](#) *QWidget::windowHandle() const

If this is a native widget, return the associated [QWindow](#). Otherwise return null. # 如果 widget 是一个本机窗口返回 QWindow, 否则返回 Null

Native widgets include toplevel widgets, QGLWidget, and child widgets on which [winId\(\)](#) was called.

本地 widget 包括 toplevel widgets、QGLWidget 和 winId()的子 widget。

This function was introduced in Qt 5.0. # 自 Qt 5.0 开始引进

See also [winId\(\)](#).

[signal]void QWidget::windowIconChanged(const [QIcon](#) &icon)

This signal is emitted when the window's icon has changed, with the new *icon* as an argument.

发送一个窗口图标改变事件, 并设置 icon 为新窗口图标

This function was introduced in Qt 5.2. # 自 Qt 5.2 开始引进

Note: Notifier signal for property [windowIcon](#). # 向 windowIcon 属性发送信号

[QString](#) QWidget::windowRole() const

Returns the window's role, or an empty string. # 返回窗口 role, 或者空字符串

See also [setWindowRole\(\)](#), [windowIcon](#), and [windowTitle](#).

Qt::WindowStates QWidget::windowState() const

Returns the current window state. The window state is a OR'ed combination of [Qt::WindowState](#): [Qt::WindowMinimized](#), [Qt::WindowMaximized](#), [Qt::WindowFullScreen](#), and [Qt::WindowActive](#).

返回窗口当前状态。状态有 Qt::WindowState: Qt::WindowMinimized, Qt::WindowMaximized, Qt::WindowFullScreen, and Qt::WindowActive

See also [Qt::WindowState](#) and [setWindowState\(\)](#).

[signal]void QWidget::windowTitleChanged(const [QString](#) &*title*)

This signal is emitted when the window's title has changed, with the new *title* as an argument.

发送一个窗口标题改变事件，并设置 title 为新窗口标题

This function was introduced in Qt 5.2. # 自 Qt 5.2 开始引进

Note: Notifier signal for property [windowTitle](#). # 向 windowTitle 属性发送信号

Qt::WindowType QWidget::windowType() const

Returns the window type of this widget. This is identical to [windowFlags\(\)](#) & [Qt::WindowType_Mask](#).

返回 widget 的类型。该类型 windowFlags() & Qt::WindowType_Mask 设置相同

See also [windowFlags](#).

Macro Documentation

QWIDGETSIZE_MAX

Defines the maximum size for a [QWidget](#) object. # 定义 QWidget 最大化的尺寸

The largest allowed size for a widget is [QSize](#)(QWIDGETSIZE_MAX, QWIDGETSIZE_MAX), i.e. [QSize](#) (16777215,16777215).

最大 widget 的允许尺寸为 QSize(QWIDGETSIZE_MAX, QWIDGETSIZE_MAX)，例如，QSize (16777215,16777215)

See also [QWidget::setMaximumSize\(\)](#).