QMainWindow Class

QMainWindow 类提供了一个主应用程序窗口。更多的...

Header:	**#include < QMainWindow>**
CMake:	find_package(Qt6 REQUIRED COMPONENTS Widgets) target_link_libraries(mytarget PRIVATE Qt6::Widgets)
qmake:	QT += widgets
Inherits:	QWidget

- 所有成员的列表，包括继承的成员

公共类型

enum	**DockOption { AnimatedDocks, AllowNestedDocks, AllowTabbedDocks, ForceTabbedDocks, VerticalTabs, GroupedDragging }**
flags	**DockOptions**

特性

animated : booldockNestingEnabled : booldockOptions : DockOptionsdocumentMode : bool	**iconSize : QSizetabShape : QTabWidget::TabShapetoolButtonStyle : Qt::ToolButtonStyleunifiedTitleAndToolBarOnMac : bool**

公共方法

	QMainWindow(QWidget **parent* = nullptr, Qt::WindowFlags *flags* = Qt::WindowFlags())
virtual	**~QMainWindow()**
void	**addDockWidget**(Qt::DockWidgetArea *area*, QDockWidget **dockwidget*)

	QMainWindow(QWidget *parent = nullptr, Qt::WindowFlags flags = Qt::WindowFlags())
void	**addDockWidget**(Qt::DockWidgetArea *area*, QDockWidget **dockwidget*, Qt::Orientation *orientation*)
void	**addToolBar**(Qt::ToolBarArea *area*, QToolBar **toolbar*)
void	**addToolBar**(QToolBar **toolbar*)
QToolBar *	**addToolBar**(const QString &*title*)
void	**addToolBarBreak**(Qt::ToolBarArea *area* = Qt::TopToolBarArea)
QWidget *	**centralWidget**() const
Qt::DockWidgetArea	**corner**(Qt::Corner *corner*) const
virtual QMenu *	**createPopupMenu**()
QMainWindow::DockOptions	**dockOptions**() const
Qt::DockWidgetArea	**dockWidgetArea**(QDockWidget **dockwidget*) const
bool	**documentMode**() const
QSize	**iconSize**() const
void	**insertToolBar**(QToolBar **before*, QToolBar** toolbar*)
void	**insertToolBarBreak**(QToolBar **before*)
bool	**isAnimated**() const
bool	**isDockNestingEnabled**() const
QMenuBar *	**menuBar**() const
QWidget *	**menuWidget**() const
void	**removeDockWidget**(QDockWidget **dockwidget*)
void	**removeToolBar**(QToolBar **toolbar*)
void	**removeToolBarBreak**(QToolBar **before*)
void	**resizeDocks**(const QList<QDockWidget > &docks*, const QList &*sizes*, Qt::Orientation *orientation*)
bool	**restoreDockWidget**(QDockWidget **dockwidget*)
bool	**restoreState**(const QByteArray &*state*, int *version* = 0)
QByteArray	**saveState**(int *version* = 0) const
void	**setCentralWidget**(QWidget **widget*)
void	**setCorner**(Qt::Corner *corner*, Qt::DockWidgetArea *area*)
void	**setDockOptions**(QMainWindow::DockOptions *options*)

	QMainWindow(QWidget *parent = nullptr, Qt::WindowFlags flags = Qt::WindowFlags())
void	**setDocumentMode**(bool *enabled*)
void	**setIconSize**(const QSize &*iconSize*)
void	**setMenuBar**(QMenuBar **menuBar*)
void	**setMenuWidget**(QWidget **menuBar*)
void	**setStatusBar**(QStatusBar **statusbar*)
void	**setTabPosition**(Qt::DockWidgetAreas *areas*, QTabWidget::TabPosition *tabPosition*)
void	**setTabShape**(QTabWidget::TabShape *tabShape*)
void	**setToolButtonStyle**(Qt::ToolButtonStyle *toolButtonStyle*)
void	**splitDockWidget**(QDockWidget **first*, QDockWidget** second*, Qt::Orientation *orientation*)
QStatusBar *	**statusBar**() const
QTabWidget::TabPosition	**tabPosition**(Qt::DockWidgetArea *area*) const
QTabWidget::TabShape	**tabShape**() const
QList<QDockWidget *>	**tabifiedDockWidgets**(QDockWidget **dockwidget*) const
void	**tabifyDockWidget**(QDockWidget **first*, QDockWidget** second*)
QWidget *	**takeCentralWidget**()
Qt::ToolBarArea	**toolBarArea**(const QToolBar **toolbar*) const
bool	**toolBarBreak**(QToolBar **toolbar*) const
Qt::ToolButtonStyle	**toolButtonStyle**() const
bool	**unifiedTitleAndToolBarOnMac**() const

公共槽

void	**setAnimated(bool *enabled*)**
void	**setDockNestingEnabled**(bool *enabled*)
void	**setUnifiedTitleAndToolBarOnMac**(bool *set*)

信号

void	**iconSizeChanged(const QSize &*iconSize*)**
void	**tabifiedDockWidgetActivated**(QDockWidget **dockWidget*)
void	**toolButtonStyleChanged**(Qt::ToolButtonStyle *toolButtonStyle*)

重载protected function

virtual void	**contextMenuEvent(QContextMenuEvent **event*) override**
virtual bool	**event**(QEvent **event*) override

详细说明

Qt 主窗口框架

主窗口提供了用于构建应用程序的用户界面的框架。Qt 有 QMainWindow 及其related classes用于主窗口管理。QMainWindow 有自己的布局，您可以添加QToolBars,QDockWidget, 一个QMenuBar，和一个QStatusBar。布局有一个中心区域，可以被任何类型的小部件占据。您可以看到下面的布局图像。

创建主窗口组件

中央小部件通常是标准 Qt 小部件，例如QTextEdit或一个QGraphicsView。自定义小部件也可用于高级应用程序。您可以使用 来设置中央小部件`setCentralWidget()`。

主窗口具有单个 (SDI) 或多 (MDI) 文档界面。您可以使用 Qt 创建 MDI 应用程序QMdiArea作为中央小部件。

我们现在将检查可以添加到主窗口的每个其他小部件。我们提供了有关如何创建和添加它们的示例。

创建菜单

Qt 在中实现菜单QMenuQMainWindow 将它们保存在QMenuBar。QActions 被添加到菜单中，并将它们显示为菜单项。

您可以通过调用将新菜单添加到主窗口的菜单栏`menuBar()`，该函数返回QMenuBar对于窗口，然后添加一个菜单QMenuBar::addMenu()。

QMainWindow 带有一个默认菜单栏，但您也可以使用`setMenuBar()`. 如果您希望实现自定义菜单栏（即不使用QMenuBarwidget），您可以使用 来设置它`setMenuWidget()`。

如何创建菜单的示例如下：

```
void MainWindow::createMenus()
{
    fileMenu = menuBar()->addMenu(tr("&File"));
    fileMenu->addAction(newAct);
    fileMenu->addAction(openAct);
    fileMenu->addAction(saveAct);
```

createPopupMenu()当主窗口接收到上下文菜单事件时，该函数创建弹出菜单。默认实现会生成一个菜单，其中包含来自停靠小部件和工具栏的可检查操作。您可以重新createPopupMenu()实现自定义菜单。

创建工具栏

工具栏是在QToolBar班级。您可以使用 向主窗口添加工具栏addToolBar()。

您可以通过将工具栏分配给特定的位置来控制工具栏的初始位置。Qt::ToolBarArea。您可以通过插入工具栏分隔符来分割区域 - 将其视为文本编辑中的换行符 - 使用addToolBarBreak()或insertToolBarBreak()。您还可以限制用户的放置QToolBar::setAllowedAreas （） 和QToolBar::setMovable()。

可以使用 检索工具栏图标的大小iconSize()。尺寸取决于平台；您可以使用 设置固定大小setIconSize()。您可以使用 更改工具栏中所有工具按钮的外观setToolButtonStyle()。

工具栏创建的示例如下：

```
void MainWindow::createToolBars()
{
    fileToolBar = addToolBar(tr("File"));
    fileToolBar->addAction(newAct);
```

创建 Dock 小部件

Dock 小部件是在QDockWidget班级。停靠小部件是一个可以停靠到主窗口中的窗口。您可以使用 向主窗口添加停靠小部件addDockWidget()。

有四个停靠小部件区域，如下所示Qt::DockWidgetArea枚举：左、右、上、下。您可以指定哪个停靠小部件区域应占据与 重叠的角落setCorner()。默认情况下，每个区域只能包含一行（垂直或水平）停靠小部件，但如果您使用启用嵌套setDockNestingEnabled()，则可以在任一方向添加停靠小部件。

两个停靠小部件也可以堆叠在彼此之上。AQTabBar然后用于选择应显示哪些小部件。

我们给出了如何创建停靠小部件并将其添加到主窗口的示例：

```
QDockWidget *dockWidget = new QDockWidget(tr("Dock Widget"), this);
dockWidget->setAllowedAreas(Qt::LeftDockWidgetArea |
                            Qt::RightDockWidgetArea);
dockWidget->setWidget(dockWidgetContents);
addDockWidget(Qt::LeftDockWidgetArea, dockWidget);
```

状态栏

您可以使用 设置状态栏setStatusBar()，但会在第一次statusBar()调用时创建状态栏（返回主窗口的状态栏）。看QStatusBar有关如何使用它的信息。

存储状态

QMainWindow 可以存储其布局的状态`saveState()`；稍后可以使用 检索它`restoreState()`。它是存储的工具栏和停靠小部件的位置和大小（相对于主窗口的大小）。

也可以看看QMenuBar,QToolBar,QStatusBar,QDockWidget, Qt Widgets - 应用程序示例,Dock Widgets Example,MDI Example， 和Menus Example。

成员类型文档

枚举 QMainWindow:: DockOption 标志 QMainWindow:: DockOptions

该枚举包含指定对接行为的标志QMainWindow。

持续的	价值	描述
`QMainWindow::AnimatedDocks`	`0x01`	相同于animated财产。
`QMainWindow::AllowNestedDocks`	`0x02`	相同于dockNestingEnabled财产。
`QMainWindow::AllowTabbedDocks`	`0x04`	用户可以将一个停靠小部件放置在另一个停靠小部件的“顶部”。这两个小部件堆叠在一起，并出现一个选项卡栏，用于选择哪个小部件可见。
`QMainWindow::ForceTabbedDocks`	`0x08`	每个停靠区域都包含一堆选项卡式停靠小部件。换句话说，停靠小部件不能在停靠区域中彼此相邻放置。如果设置此选项，AllowNestedDocks 不起作用。
`QMainWindow::VerticalTabs`	`0x10`	主窗口两侧的两个垂直停靠区域垂直显示其选项卡。如果未设置此选项，所有停靠区域都会在底部显示其选项卡。暗示 AllowTabbedDocks。也可以看看setTabPosition()。
`QMainWindow::GroupedDragging`	`0x20`	拖动扩展坞的标题栏时，所有与其关联的选项卡都将被拖动。暗示AllowTabbedDocks。如果某些 QDockWidget 在允许的区域有限制，则效果不佳。（这个枚举值是在 Qt 5.6 中添加的。）

这些选项仅控制如何将停靠小部件放置在QMainWindow。他们不会重新排列停靠小部件以符合指定的选项。因此，应在将任何停靠小部件添加到主窗口之前设置它们。AnimatedDocks 和 VerticalTabs 选项除外，它们可以随时设置。

DockOptions 类型是QFlags 的类型定义。它存储 DockOption 值的 OR 组合。

属性文档

animated : bool

该属性保存操作停靠小部件和工具栏是否有动画

当将停靠小部件或工具栏拖动到主窗口上时，主窗口会调整其内容以指示停靠小部件或工具栏在放下时将停靠在何处。设置该属性会导致QMainWindow以流畅的动画方式移动其内容。清除此属性会导致内容捕捉到新位置。

默认情况下，已设置此属性。如果主窗口包含调整大小或重新绘制自身速度较慢的小部件，则可能会清除它。

设置此属性与设置AnimatedDocks选项使用setDockOptions()。

访问功能：

bool	**isAnimated() const**
void	**setAnimated**(bool *enabled*)

dockNestingEnabled : bool

该属性保存dock是否可以嵌套

如果此属性为`false`，则停靠区域只能包含单行（水平或垂直）的停靠小部件。如果此属性为`true`，则停靠小部件占用的区域可以沿任一方向分割以包含更多停靠小部件。

仅在包含大量停靠小部件的应用程序中才需要停靠嵌套。它为用户组织主窗口提供了更大的自由。然而，当将停靠窗口小部件拖动到主窗口上时，停靠嵌套会导致更复杂（且不太直观）的行为，因为可以通过多种方式将放置的停靠窗口小部件放置在停靠区域中。

设置此属性与设置AllowNestedDocks选项使用setDockOptions()。

访问功能：

bool	**isDockNestingEnabled() const**
void	**setDockNestingEnabled**(bool *enabled*)

dockOptions : DockOptions

该属性保存的对接行为QMainWindow

默认值为AnimatedDocks|AllowTabbedDocks。

访问功能：

QMainWindow::DockOptions	**dockOptions() const**
void	**setDockOptions**(QMainWindow::DockOptions *options*)

documentMode : bool

此属性保存选项卡式停靠小部件的选项卡栏是否设置为文档模式。

默认为 false。

访问功能：

bool	documentMode() const
void	**setDocumentMode**(bool *enabled*)

也可以看看QTabBar::documentMode。

iconSize : QSize

主窗口中工具栏图标的大小。

默认为GUI样式的默认工具栏图标大小。请注意，所使用的图标必须至少具有此大小，因为图标仅按比例缩小。

访问功能：

QSize	iconSize() const
void	**setIconSize**(const QSize &*iconSize*)

tabShape : QTabWidget::TabShape

此属性保存用于选项卡式停靠小部件的选项卡形状。

默认为QTabWidget::Rounded。

访问功能：

QTabWidget::TabShape	tabShape() const
void	**setTabShape**(QTabWidget::TabShape *tabShape*)

也可以看看setTabPosition()。

toolButtonStyle : Qt::ToolButtonStyle

此主窗口中工具栏按钮的样式。

要使工具按钮的样式遵循系统设置，请将此属性设置为Qt::ToolButtonFollowStyle。在 Unix 上，将使用桌面环境中的用户设置。在其他平台上，Qt::ToolButtonFollowStyle仅表示图标。

默认为Qt::ToolButtonIconOnly。

访问功能：

Qt::ToolButtonStyle	toolButtonStyle() const
void	**setToolButtonStyle**(Qt::ToolButtonStyle *toolButtonStyle*)

unifiedTitleAndToolBarOnMac : bool

该属性保存窗口是否使用 macOS 上的统一标题和工具栏外观

请注意，与 Qt 4 相比，Qt 5 实现有一些限制：

- 不支持在具有 OpenGL 内容的 Windows 中使用。这包括 QOpenGLWidget。
- 使用可固定或可移动的工具栏可能会导致绘制错误，因此不建议使用

访问功能：

bool	**unifiedTitleAndToolBarOnMac() const**
void	**setUnifiedTitleAndToolBarOnMac**(bool *set*)

成员函数文档

*[explicit]QMainWindow::QMainWindow(QWidget *parent = nullptr, Qt::WindowFlags flags = Qt::WindowFlags())*

使用给定的值构造一个 QMainWindow*parent*和指定的小部件*flags*。

QMainWindow 设置Qt::Window标记自身，因此将始终被创建为顶级小部件。

[virtual]QMainWindow::~QMainWindow()

销毁主窗口。

*void QMainWindow::addDockWidget(Qt::DockWidgetArea area, QDockWidget *dockwidget)*

添加给定的*dockwidget*到指定的*area*。

*void QMainWindow::addDockWidget(Qt::DockWidgetArea area, QDockWidget *dockwidget, Qt::Orientation orientation)*

添加*dockwidget*进入给定的*area*在指定的方向*orientation*。

*void QMainWindow::addToolBar(Qt::ToolBarArea area, QToolBar *toolbar)*

添加了*toolbar*进入指定的*area*在此主窗口中。这*toolbar*放置在当前工具栏块（即行）的末尾。如果主窗口已经管理*toolbar*那么它只会将工具栏移动到*area*。

也可以看看insertToolBar(),addToolBarBreak（），和insertToolBarBreak()。

*void QMainWindow::addToolBar(QToolBar *toolbar)*

这是一个过载功能。

相当于调用 addToolBar(Qt::TopToolBarArea,*toolbar*)

QToolBar QMainWindow::addToolBar(const QString &title)*

这是一个过载功能。

创建一个QToolBar对象，将其窗口标题设置为*title*，并将其插入顶部工具栏区域。

也可以看看setWindowTitle()。

void QMainWindow::addToolBarBreak(Qt::ToolBarArea area = Qt::TopToolBarArea)

添加一个工具栏中断给给定的*area*在所有其他存在的物体之后。

*QWidget *QMainWindow::centralWidget() const*

返回主窗口的中央小部件。`nullptr`如果尚未设置中央小部件，则此函数返回。

也可以看看setCentralWidget()。

*[override virtual protected]void QMainWindow::contextMenuEvent(QContextMenuEvent *event)*

重新实现：QWidget::contextMenuEvent（QContextMenuEvent *事件）。

Qt::DockWidgetArea QMainWindow::corner(Qt::Corner corner) const

返回占据指定的停靠小部件区域*corner*。

也可以看看setCorner()。

*[virtual]QMenu *QMainWindow::createPopupMenu()*

返回一个弹出菜单，其中包含主窗口中存在的工具栏和停靠小部件的可检查条目。如果不存在工具栏和停靠小部件，则此函数返回`nullptr`。

默认情况下，当用户激活上下文菜单时（通常是通过右键单击工具栏或停靠小部件），主窗口将调用此函数。

如果您想创建自定义弹出菜单，请重新实现此函数并返回新创建的弹出菜单。弹出菜单的所有权转移给调用者。

也可以看看addDockWidget(),addToolBar（），和menuBar()。

*Qt::DockWidgetArea QMainWindow::dockWidgetArea(QDockWidget *dockwidget) const*

返回Qt::DockWidgetArea为了*dockwidget*。如果*dockwidget*尚未添加到主窗口，该函数返回`Qt::NoDockWidgetArea`。

也可以看看addDockWidget(),splitDockWidget（），和Qt::DockWidgetArea。

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

重新实现：QWidget::event（QEvent *事件）。

[signal]void QMainWindow::iconSizeChanged(const QSize &iconSize)

当窗口中使用的图标的大小更改时，会发出此信号。传入新的图标大小*iconSize*。

您可以将此信号连接到其他组件，以帮助保持应用程序的外观一致。

也可以看看setIconSize()。

*void QMainWindow::insertToolBar(QToolBar **before***, **QToolBar** *toolbar*)*

插入*toolbar*进入所占据的区域*before*工具栏，使其出现在其前面。例如，在正常的从左到右布局操作中，这意味着*toolbar*将出现在由指定的工具栏的左侧*before*在水平工具栏区域中。

也可以看看insertToolBarBreak(),addToolBar（），和addToolBarBreak()。

*void QMainWindow::insertToolBarBreak(QToolBar *before)*

在由指定的工具栏之前插入工具栏分隔符*before*。

*QMenuBar *QMainWindow::menuBar() const*

返回主窗口的菜单栏。如果菜单栏不存在，此函数将创建并返回一个空菜单栏。

如果你想让Mac应用程序中的所有窗口共享一个菜单栏，就不要使用这个函数来创建，因为这里创建的菜单栏会有这个QMainWindow作为其父级。相反，您必须创建一个没有父级的菜单栏，然后可以在所有 Mac 窗口之间共享该菜单栏。这样创建一个无父菜单栏：

```
QMenuBar *menuBar = new QMenuBar(nullptr);
```

也可以看看setMenuBar()。

*QWidget *QMainWindow::menuWidget() const*

返回主窗口的菜单栏。如果尚未构造菜单栏，则此函数返回 null。

也可以看看setMenuWidget()。

*void QMainWindow::removeDockWidget(QDockWidget *dockwidget)*

删除*dockwidget*从主窗口布局中隐藏它。请注意，*dockwidget*没有被删除。

*void QMainWindow::removeToolBar(QToolBar *toolbar)*

删除*toolbar*从主窗口布局中隐藏它。请注意，*toolbar*没有被删除。

*void QMainWindow::removeToolBarBreak(QToolBar *before)*

删除先前插入的工具栏之前插入的工具栏分隔符*before*。

void QMainWindow::resizeDocks(const QList<QDockWidget > &docks, const QList< int> &sizes, Qt::Orientation orientation)*

调整列表中停靠小部件的大小*docks*到列表中相应的大小（以像素为单位）*sizes*。如果*orientation*是Qt::Horizontal，调整宽度，否则调整停靠小部件的高度。尺寸将进行调整，以尊重最大和最小尺寸，并且QMainWindow本身不会调整大小。任何额外/缺失的空间都会根据尺寸的相对权重分布在小部件之间。

例子：

```
    resizeDocks({blueWidget, yellowWidget}, {20 , 40}, Qt::Horizontal);
```

如果蓝色和黄色小部件嵌套在同一级别，它们将被调整大小，使得黄色小部件是蓝色小部件的两倍大

如果某些小部件分组在选项卡中，则每组仅应指定一个小部件。不在列表中的小部件可能会被更改以遵守约束。

*bool QMainWindow::restoreDockWidget(QDockWidget *dockwidget)*

恢复状态*dockwidget*如果它是在调用后创建的restoreState()。`true`如果状态恢复则返回；否则返回`false`.

也可以看看restoreState（） 和saveState()。

bool QMainWindow::restoreState(const QByteArray &state, int version = 0)

恢复*state*该主窗口的工具栏和停靠小部件。也恢复角落设置。这*version*数字与存储在中的数字进行比较*state*。如果它们不匹配，则主窗口的状态保持不变，并且该函数返回`false`；否则，状态恢复，并且该函数返回`true`。

恢复使用保存的几何图形QSettings，您可以使用这样的代码：

```
void MainWindow::readSettings()
{
    QSettings settings("MyCompany", "MyApp");
    restoreGeometry(settings.value("myWidget/geometry").toByteArray());
    restoreState(settings.value("myWidget/windowState").toByteArray());
}
```

也可以看看saveState(),QWidget::saveGeometry(),QWidget::restoreGeometry（），和restoreDockWidget()。

QByteArray QMainWindow::saveState(int version = 0) const

保存此主窗口工具栏和停靠小部件的当前状态。这包括可以使用以下命令设置的角设置setCorner()。这*version*数字作为数据的一部分存储。

这objectName属性用于识别每个QToolBar和QDockWidget。您应该确保此属性对于每个QToolBar和QDockWidget你添加到QMainWindow

要恢复保存的状态，请传递返回值并*version*号码至restoreState()。

要在窗口关闭时保存几何图形，您可以实现如下关闭事件：

```
void MyMainWindow::closeEvent(QCloseEvent *event)
{
    QSettings settings("MyCompany", "MyApp");
    settings.setValue("geometry", saveGeometry());
    settings.setValue("windowState", saveState());
    QMainWindow::closeEvent(event);
}
```

也可以看看restoreState(),QWidget::saveGeometry（），和QWidget::restoreGeometry()。

*void QMainWindow::setCentralWidget(QWidget *widget)*

设置给定的*widget*成为主窗口的中央小部件。

笔记：QMainWindow取得所有权*widget*指针并在适当的时候删除它。

也可以看看centralWidget()。

void QMainWindow::setCorner(Qt::Corner corner, Qt::DockWidgetArea area)

设置给定的停靠小部件*area*占据指定的*corner*。

也可以看看corner()。

*void QMainWindow::setMenuBar(QMenuBar *menuBar)*

将主窗口的菜单栏设置为*menuBar*。

笔记：QMainWindow取得所有权*menuBar*指针并在适当的时候删除它。

也可以看看menuBar()。

*void QMainWindow::setMenuWidget(QWidget *menuBar)*

将主窗口的菜单栏设置为*menuBar*。

QMainWindow取得所有权*menuBar*指针并在适当的时候删除它。

也可以看看menuWidget()。

*void QMainWindow::setStatusBar(QStatusBar *statusbar)*

将主窗口的状态栏设置为*statusbar*。

将状态栏设置为`nullptr`会将其从主窗口中删除。注意QMainWindow取得所有权*statusbar*指针并在适当的时候删除它。

也可以看看statusBar()。

void QMainWindow::setTabPosition(Qt::DockWidgetAreas areas, QTabWidget::TabPosition tabPosition)

设置给定停靠小部件的选项卡位置*areas*到指定的*tabPosition*。默认情况下，所有停靠区域都在底部显示其选项卡。

注：VerticalTabs停靠选项会覆盖此方法设置的选项卡位置。

也可以看看tabPosition（） 和setTabShape()。

*void QMainWindow::splitDockWidget(QDockWidget **first*, **QDockWidget** *second*, *Qt::Orientation orientation)*

分割所覆盖的空间*first*将小部件停靠为两部分，移动*first*将小部件停靠到第一部分，并将*second*将小部件停靠到第二部分。

这*orientation*指定空间如何划分：AQt::Horizontalsplit 将第二个停靠小部件放置在第一个停靠小部件的右侧；AQt::Verticalsplit 将第二个停靠小部件放置在第一个停靠小部件下方。

注意：如果*first*当前位于选项卡式停靠区域中，*second*将被添加为新选项卡，而不是作为相邻选项卡*first*。这是因为单个选项卡只能包含一个停靠小部件。

注：Qt::LayoutDirection影响分隔区域两部分中停靠小部件的顺序。当启用从右到左的布局方向时，停靠小部件的放置将相反。

也可以看看tabifyDockWidget(),addDockWidget（），和removeDockWidget()。

*QStatusBar *QMainWindow::statusBar() const*

返回主窗口的状态栏。如果状态栏不存在，此函数将创建并返回一个空状态栏。

也可以看看setStatusBar()。

QTabWidget::TabPosition QMainWindow::tabPosition(Qt::DockWidgetArea area) const

返回制表符位置*area*。

注：VerticalTabs停靠选项会覆盖此函数返回的选项卡位置。

也可以看看setTabPosition（） 和tabShape()。

*[signal]void QMainWindow::tabifiedDockWidgetActivated(QDockWidget *dockWidget)*

当通过选择选项卡激活选项卡式停靠小部件时，会发出此信号。传入激活的dock widget*dockWidget*。

也可以看看tabifyDockWidget（） 和tabifiedDockWidgets()。

*QList<QDockWidget > QMainWindow::tabifiedDockWidgets(QDockWidget **dockwidget) const*

返回与选项卡一起显示的停靠小部件*dockwidget*。

也可以看看tabifyDockWidget()。

*void QMainWindow::tabifyDockWidget(QDockWidget **first***, ***QDockWidget** second*)*

动作*second*将小部件停靠在顶部*first*停靠小部件，在主窗口中创建一个选项卡式停靠区域。

也可以看看tabifiedDockWidgets()。

QWidget *QMainWindow::takeCentralWidget()*

从此主窗口中删除中央小部件。

已删除小部件的所有权将传递给调用者。

Qt::ToolBarArea *QMainWindow::toolBarArea(const* *QToolBar* **toolbar) const*

返回Qt::ToolBarArea为了*toolbar*。如果*toolbar*尚未添加到主窗口，该函数返回`Qt::NoToolBarArea`。

也可以看看addToolBar(),addToolBarBreak（），和Qt::ToolBarArea。

*bool QMainWindow::toolBarBreak(*QToolBar* **toolbar) const*

返回之前是否有工具栏中断*toolbar*。

也可以看看addToolBarBreak（）和insertToolBarBreak()。

[signal]void QMainWindow::toolButtonStyleChanged(Qt::ToolButtonStyle toolButtonStyle)

当窗口中工具按钮的样式发生更改时，会发出此信号。新样式传入*toolButtonStyle*。

您可以将此信号连接到其他组件，以帮助保持应用程序的外观一致。

也可以看看setToolButtonStyle()。