QTextEdit Class

QTextEdit 类提供了一个用于编辑和显示纯文本和富文本的小部件。更多的...

Header:	**#include**
CMake:	find_package(Qt6 REQUIRED COMPONENTS Widgets) target_link_libraries(mytarget PRIVATE Qt6::Widgets)
qmake:	QT += widgets
Inherits:	QAbstractScrollArea
Inherited By:	QTextBrowser

- 所有成员的列表，包括继承的成员
- QTextEdit 是富文本处理 API的一部分。

公共类型

struct	**ExtraSelection**
flags	**AutoFormatting**
enum	**AutoFormattingFlag** { AutoNone, AutoBulletList, AutoAll }
enum	**LineWrapMode** { NoWrap, WidgetWidth, FixedPixelWidth, FixedColumnWidth }

特性

acceptRichText : **bool** **autoFormatting** : **AutoFormatting** **cursorWidth** : **int** **document** : **QTextDocument*** **documentTitle** : **QString** **html** : **QString** **lineWrapColumnOrWidth** : **int** **lineWrapMode** : **LineWrapMode** **markdown** : **QString**

overwriteMode : **bool** **placeholderText** : **QString** **plainText** : **QString** **readOnly** : **bool** **tabChangesFocus** : **bool** **tabStopDistance** : **qreal** **textInteractionFlags** : **Qt::TextInteractionFlags** **undoRedoEnabled** : **bool** **wordWrapMode** : **QTextOption::WrapMode**

公共函数

	QTextEdit(QWidget *parent = nullptr)
	QTextEdit(const QString &text, QWidget *parent = nullptr)
virtual	~QTextEdit()
bool	acceptRichText() const
Qt::Alignment	alignment() const
QString	anchorAt(const QPoint &pos) const
QTextEdit::AutoFormatting	autoFormatting() const
bool	canPaste() const
QMenu *	createStandardContextMenu()
QMenu *	createStandardContextMenu(const QPoint &position)
QTextCharFormat	currentCharFormat() const
QFont	currentFont() const
QTextCursor	cursorForPosition(const QPoint &pos) const
QRect	cursorRect(const QTextCursor &cursor) const
QRect	cursorRect() const
int	cursorWidth() const
QTextDocument *	document() const
QString	documentTitle() const
void	ensureCursorVisible()
QListQTextEdit::ExtraSelection	extraSelections() const
bool	find(const QString &exp, QTextDocument::FindFlags options = QTextDocument::FindFlags())
bool	find(const QRegularExpression &exp, QTextDocument::FindFlags options = QTextDocument::FindFlags())
QString	fontFamily() const
bool	fontItalic() const
qreal	fontPointSize() const
bool	fontUnderline() const
int	fontWeight() const
bool	isReadOnly() const
bool	isUndoRedoEnabled() const

	QTextEdit(QWidget **parent* = nullptr)
int	**lineWrapColumnOrWidth**() const
QTextEdit::LineWrapMode	**lineWrapMode**() const
virtual QVariant	**loadResource**(int *type*, const QUrl &*name*)
void	**mergeCurrentCharFormat**(const QTextCharFormat &*modifier*)
void	**moveCursor**(QTextCursor::MoveOperation *operation*, QTextCursor::MoveMode *mode* = QTextCursor::MoveAnchor)
bool	**overwriteMode**() const
QString	**placeholderText**() const
void	**print**(QPagedPaintDevice **printer*) const
void	**setAcceptRichText**(bool *accept*)
void	**setAutoFormatting**(QTextEdit::AutoFormatting *features*)
void	**setCurrentCharFormat**(const QTextCharFormat &*format*)
void	**setCursorWidth**(int *width*)
void	**setDocument**(QTextDocument **document*)
void	**setDocumentTitle**(const QString &*title*)
void	**setExtraSelections**(const QListQTextEdit::ExtraSelection &*selections*)
void	**setLineWrapColumnOrWidth**(int *w*)
void	**setLineWrapMode**(QTextEdit::LineWrapMode *mode*)
void	**setOverwriteMode**(bool *overwrite*)
void	**setPlaceholderText**(const QString &*placeholderText*)
void	**setReadOnly**(bool *ro*)
void	**setTabChangesFocus**(bool *b*)
void	**setTabStopDistance**(qreal *distance*)
void	**setTextCursor**(const QTextCursor &*cursor*)
void	**setTextInteractionFlags**(Qt::TextInteractionFlags *flags*)
void	**setUndoRedoEnabled**(bool *enable*)
void	**setWordWrapMode**(QTextOption::WrapMode *policy*)
bool	**tabChangesFocus**() const
qreal	**tabStopDistance**() const
QColor	**textBackgroundColor**() const

	QTextEdit(QWidget **parent* = nullptr)
QColor	**textColor**() const
QTextCursor	**textCursor**() const
Qt::TextInteractionFlags	**textInteractionFlags**() const
QString	**toHtml**() const
QString	**toMarkdown**(QTextDocument::MarkdownFeatures *features* = QTextDocument::MarkdownDialectGitHub) const
QString	**toPlainText**() const
QTextOption::WrapMode	**wordWrapMode**() const

重载的公共函数

virtual QVariant	**inputMethodQuery(Qt::InputMethodQuery *property*) const override**

公共槽

void	**append(const QString &*text*)**
void	**clear**()
void	**copy**()
void	**cut**()
void	**insertHtml**(const QString &*text*)
void	**insertPlainText**(const QString &*text*)
void	**paste**()
void	**redo**()
void	**scrollToAnchor**(const QString &*name*)
void	**selectAll**()
void	**setAlignment**(Qt::Alignment *a*)
void	**setCurrentFont**(const QFont &*f*)
void	**setFontFamily**(const QString &*fontFamily*)
void	**setFontItalic**(bool *italic*)
void	**setFontPointSize**(qreal *s*)

void	**setFontPointSize**(qreal *s*)
void	**append**(**const QString &*text*)**
void	**setFontUnderline**(bool *underline*)
void	**setFontWeight**(int *weight*)
void	**setHtml**(const QString &*text*)
void	**setMarkdown**(const QString &*markdown*)
void	**setPlainText**(const QString &*text*)
void	**setText**(const QString &*text*)
void	**setTextBackgroundColor**(const QColor &*c*)
void	**setTextColor**(const QColor &*c*)
void	**undo**()
void	**zoomIn**(int *range* = 1)
void	**zoomOut**(int *range* = 1)

信号

void	**copyAvailable**(**bool *yes*)**
void	**currentCharFormatChanged**(const QTextCharFormat &*f*)
void	**cursorPositionChanged**()
void	**redoAvailable**(bool *available*)
void	**selectionChanged**()
void	**textChanged**()
void	**undoAvailable**(bool *available*)

protected function

virtual bool	**canInsertFromMimeData**(**const QMimeData *source*) const**
virtual QMimeData *	**createMimeDataFromSelection**() const
virtual void	**insertFromMimeData**(const QMimeData *source*)

重载的protected function

virtual void	changeEvent(QEvent *e) override
virtual void	contextMenuEvent(QContextMenuEvent *event) override
virtual void	dragEnterEvent(QDragEnterEvent *e) override
virtual void	dragLeaveEvent(QDragLeaveEvent *e) override
virtual void	dragMoveEvent(QDragMoveEvent *e) override
virtual void	dropEvent(QDropEvent *e) override
virtual void	focusInEvent(QFocusEvent *e) override
virtual bool	focusNextPrevChild(bool *next*) override
virtual void	focusOutEvent(QFocusEvent *e) override
virtual void	inputMethodEvent(QInputMethodEvent *e) override
virtual void	keyPressEvent(QKeyEvent *e) override
virtual void	keyReleaseEvent(QKeyEvent *e) override
virtual void	mouseDoubleClickEvent(QMouseEvent *e) override
virtual void	mouseMoveEvent(QMouseEvent *e) override
virtual void	mousePressEvent(QMouseEvent *e) override
virtual void	mouseReleaseEvent(QMouseEvent *e) override
virtual void	paintEvent(QPaintEvent *event) override
virtual void	resizeEvent(QResizeEvent *e) override
virtual void	scrollContentsBy(int *dx*, int *dy*) override
virtual void	showEvent(QShowEvent *) override
virtual void	wheelEvent(QWheelEvent *e) override

详细说明

简介和概念

QTextEdit 是一种高级的所见即所得查看器/编辑器，支持使用 HTML 样式标签或 Markdown 格式的富文本格式。它经过优化，可以处理大型文档并快速响应用户输入。

QTextEdit 适用于段落和字符。段落是一个格式化的字符串，它被自动换行以适应小部件的宽度。默认情况下，阅读纯文本时，一个换行符表示一个段落。文档由零个或多个段落组成。段落中的单词按照段落的对齐方式对齐。段落之间用硬换行符分隔。段落中的每个字符都有自己的属性，例如字体和颜色。

QTextEdit可以显示图像、列表和表格。如果文本太大而无法在文本编辑的视口中查看，则会出现滚动条。文本编辑可以加载纯文本和富文本文件。富文本可以使用 HTML 4 标记的子集来描述；参考Supported HTML Subset页面了解更多信息。

如果您只需要显示一小段富文本使用QLabel。

Qt 中的富文本支持旨在提供一种快速、可移植且高效的方式来向应用程序添加合理的在线帮助设施，并为富文本编辑器提供基础。如果您发现 HTML 支持不足以满足您的需求，您可以考虑使用 Qt WebKit，它提供了功能齐全的 Web 浏览器小部件。

QTextEdit 上鼠标光标的形状是Qt::IBeamCursor默认情况下。可以通过以下方式更改viewport() 的光标属性。

使用 QTextEdit 作为显示小部件

QTextEdit 可以显示大型 HTML 子集，包括表格和图像。

可以使用设置或替换文本setHtml() 删除任何现有文本并将其替换为传入的文本setHtml （） 称呼。如果你打电话setHtml() 使用旧版 HTML，然后调用toHtml()，返回的文本可能有不同的标记，但会呈现相同的效果。可以使用以下命令删除整个文本clear()。

文本也可以使用设置或替换setMarkdown()，并且同样的警告适用：如果您随后调用toMarkdown()，返回的文本可能不同，但尽可能保留含义。可以解析带有一些嵌入 HTML 的 Markdown，具有与setHtml （） 有; 但toMarkdown() 只编写“纯”Markdown，没有任何嵌入的 HTML。

文本本身可以使用插入QTextCursor类或使用便利函数insertHtml(),insertPlainText(),append （） 或者paste()。QTextCursor还能够将复杂的对象（例如表格或列表）插入到文档中，并且它处理创建选择并对所选文本应用更改。

默认情况下，文本编辑会在空格处换行以适合文本编辑小部件。这setLineWrapMode() 函数用于指定您想要的换行类型，或者NoWrap如果你不想要任何包装。称呼setLineWrapMode() 设置固定像素宽度FixedPixelWidth，或字符列（例如80列）FixedColumnWidth与指定的像素或列setLineWrapColumnOrWidth()。如果您对小部件的宽度使用自动换行WidgetWidth，您可以指定是否在空格或任何地方中断setWordWrapMode()。

这find() 函数可用于查找并选择文本中的给定字符串。

如果您想限制 QTextEdit 中的段落总数，例如它在日志查看器中通常很有用，那么您可以使用QTextDocument的 MaximumBlockCount 属性。

只读键绑定

当 QTextEdit 以只读方式使用时，按键绑定仅限于导航，并且只能使用鼠标选择文本：

按键	行动
向上	向上移动一根线。
向下	向下移动一行。
左边	向左移动一个字符。
正确的	向右移动一个字符。
向上翻页	向上移动一页（视口）。
向下翻页	向下移动一页（视口）。
家	移动到文本的开头。
结尾	移至文本末尾。

按键	行动
Alt+滚轮	水平滚动页面（滚轮是鼠标滚轮）。
Ctrl+滚轮	缩放文本。
Ctrl+A	选择所有文本。

文本编辑也许能够提供一些元信息。例如，documentTitle() 函数将从 HTML 标记中返回文本`<title>`。

注意：仅当字体大小未设置为固定大小时，缩放 HTML 文档才有效。

使用 QTextEdit 作为编辑器

有关使用 QTextEdit 作为显示小部件的所有信息也适用于此处。

当前字符格式的属性设置为setFontItalic(),setFontWeight(),setFontUnderline(),setFontFamily(),setFontPointSize(),setTextColor() 和setCurrentFont()。当前段落的对齐方式设置为setAlignment()。

文本的选择由QTextCursor类，提供创建选择、检索文本内容或删除选择的功能。您可以使用以下命令检索与用户可见光标对应的对象textCursor() 方法。如果你想在 QTextEdit 中设置一个选择，只需在QTextCursor对象，然后使用该光标使光标成为可见光标setTextCursor()。可以使用以下命令将选择复制到剪贴板copy()，或使用以下命令剪切到剪贴板cut()。可以使用选择整个文本selectAll()。

当光标移动并且底层格式属性发生变化时，currentCharFormatChanged发出 () 信号以反映新光标位置处的新属性。

这textChanged每当文本发生变化时（由于setText() 或通过编辑器本身）。

QTextEdit 拥有一个QTextDocument可以使用以下方法检索对象document() 方法。您还可以使用设置自己的文档对象setDocument()。

QTextDocument提供了一个isModified() 函数，如果文本自加载以来或自上次使用 false 作为参数调用 setModified 以来已被修改，则该函数将返回 true。此外，它还提供了撤消和重做的方法。

拖放

QTextEdit 还支持自定义拖放行为。默认情况下，当用户将这些 MIME 类型的数据拖放到文档中时，QTextEdit 将插入纯文本、HTML 和富文本。重新实现canInsertFromMimeData() 和insertFromMimeData() 添加对其他 MIME 类型的支持。

例如，要允许用户将图像拖放到 QTextEdit 上，您可以通过以下方式实现这些功能：

```
bool TextEdit::canInsertFromMimeData( const QMimeData *source ) const
{
    if (source->hasImage())
        return true;
    else
        return QTextEdit::canInsertFromMimeData(source);
}
```

我们通过返回 true 添加对图像 MIME 类型的支持。对于所有其他 MIME 类型，我们使用默认实现。

```
void TextEdit::insertFromMimeData( const QMimeData *source )
{
    if (source->hasImage())
    {
        QImage image = qvariant_cast<QImage>(source->imageData());
        QTextCursor cursor = this->textCursor();
        QTextDocument *document = this->document();
        document->addResource(QTextDocument::ImageResource, QUrl("image"), image);
        cursor.insertImage("image");
    }
}
```

我们将图像从QVariant由 MIME 源保存并将其作为资源插入到文档中。

编辑按键绑定

实现编辑的按键绑定列表：

按键	行动
退格键	删除光标左侧的字符。
删除	删除光标右侧的字符。
Ctrl+C	将选定的文本复制到剪贴板。
Ctrl+插入	将选定的文本复制到剪贴板。
Ctrl+K	删除到行尾。
Ctrl+V	将剪贴板文本粘贴到文本编辑中。
Shift+插入	将剪贴板文本粘贴到文本编辑中。
Ctrl+X	删除选定的文本并将其复制到剪贴板。
Shift+删除	删除选定的文本并将其复制到剪贴板。
Ctrl+Z	撤消上次操作。
Ctrl+Y	重做上次操作。
左边	将光标向左移动一个字符。
Ctrl+左键	将光标向左移动一个单词。
正确的	将光标向右移动一个字符。
Ctrl+右	将光标向右移动一个字。
向上	将光标向上移动一行。
向下	将光标向下移动一行。
向上翻页	将光标向上移动一页。

向按键翻页	将光标向行移动一页。
家	将光标移至行首。
Ctrl+Home	将光标移动到文本的开头。
结尾	将光标移至行尾。
Ctrl+结束	将光标移至文本末尾。
Alt+滚轮	水平滚动页面（滚轮是鼠标滚轮）。

要选择（标记）文本，请按住 Shift 键，同时按任一移动键击，例如，*Shift+Right*将选择右侧的字符，*Shift+Ctrl+Right*将选择右侧的单词，等等。

可以参阅 QTextDocument,QTextCursor, Qt Widgets - 应用程序示例,Syntax Highlighter Example， 和 Rich Text Processing。

会员类型文档

枚举 QTextEdit:: AutoFormattingFlag 标志 QTextEdit:: AutoFormatting

持续的	价值	描述
`QTextEdit::AutoNone`	`0`	不要执行任何自动格式化。
`QTextEdit::AutoBulletList`	`0x00000001`	自动创建项目符号列表（例如，当用户在最左边的列中输入星号（“*”）或在现有列表项中按 Enter 键时。
`QTextEdit::AutoAll`	`0xffffffff`	应用所有自动格式。目前仅支持自动项目符号列表。

AutoFormatting 类型是QFlags 的类型定义。它存储 AutoFormattingFlag 值的 OR 组合。

enum QTextEdit::LineWrapMode

持续的	价值
`QTextEdit::NoWrap`	`0`
`QTextEdit::WidgetWidth`	`1`
`QTextEdit::FixedPixelWidth`	`2`
`QTextEdit::FixedColumnWidth`	`3`

属性文档

acceptRichText : bool

该属性保存文本编辑是否接受用户插入的富文本

当此属性设置为 false 时，文本编辑将仅接受用户的纯文本输入。例如通过剪贴板或拖放。

该属性的默认值为 true。

访问功能：

bool	**acceptRichText() const**
void	**setAcceptRichText**(bool *accept*)

autoFormatting : AutoFormatting

该属性保存启用的自动格式化功能集

该值可以是中值的任意组合AutoFormattingFlag枚举。默认为AutoNone。选择AutoAll启用所有自动格式化。

目前，唯一提供的自动格式化功能是AutoBulletList; Qt 的未来版本可能会提供更多功能。

访问功能：

QTextEdit::AutoFormatting	**autoFormatting() const**
void	**setAutoFormatting**(QTextEdit::AutoFormatting *features*)

cursorWidth : int

该属性指定光标的宽度（以像素为单位）。默认值为 1。

访问功能：

int	**cursorWidth() const**
void	**setCursorWidth**(int *width*)

*document : QTextDocument**

该属性保存文本编辑器的基础文档。

注意：编辑器*不拥有文档的所有权*，除非它是文档的父对象。所提供文档的父对象仍然是该对象的所有者。如果先前分配的文档是编辑器的子文档，那么它将被删除。

访问功能：

QTextDocument *	document() const
void	**setDocument**(QTextDocument **document*)

documentTitle : QString

该属性保存从文本解析的文档的标题。

默认情况下，对于新创建的空文档，此属性包含空字符串。

访问功能：

QString	documentTitle() const
void	**setDocumentTitle**(const QString &*title*)

html : QString

该属性为文本编辑的文本提供了 HTML 界面。

toHtml() 将文本编辑的文本返回为 html。

setHtml() 更改文本编辑的文本。任何先前的文本都将被删除，并且撤消/重做历史记录将被清除。输入文本被解释为 html 格式的富文本。currentCharFormat() 也会被重置，除非textCursor() 已位于文档的开头。

注意：调用者有责任确保文本在调用时正确解码。QString创建包含 HTML 的内容并将其传递给 setHtml()。

默认情况下，对于新创建的空文档，此属性包含用于描述没有正文文本的 HTML 4.0 文档的文本。

访问功能：

QString	toHtml() const
void	**setHtml**(const QString &*text*)

通知器信号：

void	textChanged()

可以参阅Supported HTML Subset和plainText。

lineWrapColumnOrWidth : int

此属性保存文本将被换行的位置（以像素或列为单位，具体取决于换行模式）

如果换行模式是FixedPixelWidth，该值是距离文本编辑左边缘应换行的像素数。如果换行模式是FixedColumnWidth，该值是距文本编辑左边缘的列号（在字符列中），文本应在该位置换行。

默认情况下，此属性包含值 0。

访问功能：

int	**lineWrapColumnOrWidth() const**
void	**setLineWrapColumnOrWidth**(int *w*)

可以参阅lineWrapMode。

lineWrapMode : LineWrapMode

该属性保存换行模式

默认模式是WidgetWidth这会导致单词在文本编辑的右边缘换行。换行发生在空白处，保持整个单词完整。如果您希望在单词中进行换行，请使用setWordWrapMode()。如果您设置换行模式FixedPixelWidth或者FixedColumnWidth你也应该打电话setLineWrapColumnOrWidth() 为您想要的宽度。

访问功能：

QTextEdit::LineWrapMode	**lineWrapMode() const**
void	**setLineWrapMode**(QTextEdit::LineWrapMode *mode*)

可以参阅lineWrapColumnOrWidth。

markdown : QString

该属性为文本编辑的文本提供了 Markdown 接口。

`toMarkdown()`将文本编辑的文本返回为“纯”Markdown，没有任何嵌入的 HTML 格式。一些功能QTextDocument支持（例如使用特定颜色和命名字体）无法用“纯”Markdown 表达，它们将被省略。

`setMarkdown()`更改文本编辑的文本。任何先前的文本都将被删除，并且撤消/重做历史记录将被清除。输入文本被解释为 Markdown 格式的富文本。

解析包含在*markdown*字符串的处理方式与setHtml; 但是，不支持 HTML 块内的 Markdown 格式。

解析器的某些功能可以通过以下方式启用或禁用*features*争论：

持续的	**描述**
`MarkdownNoHTML`	Markdown 文本中的任何 HTML 标签都将被丢弃

持续的	描述
`MarkdownDialectCommonMark`	解析器仅支持 CommonMark 标准化的功能
`MarkdownDialectGitHub`	解析器支持 GitHub 方言

默认为`MarkdownDialectGitHub`.

访问功能：

QString	**toMarkdown(QTextDocument::MarkdownFeatures *features* = QTextDocument::MarkdownDialectGitHub) const**
void	**setMarkdown**(const QString &*markdown*)

通知器信号：

void	**textChanged()**

可以参阅plainText,html,QTextDocument::toMarkdown（），和QTextDocument::setMarkdown()。

overwriteMode : bool

该属性保存用户输入的文本是否会覆盖现有文本

与许多文本编辑器一样，文本编辑器小部件可以配置为插入或用用户输入的新文本覆盖现有文本。

如果此属性为`true`，则现有文本将被新文本逐字符覆盖；否则，文本将插入到光标位置，替换现有文本。

默认情况下，此属性为`false`（新文本不会覆盖现有文本）。

访问功能：

bool	**overwriteMode() const**
void	**setOverwriteMode**(bool *overwrite*)

placeholderText : QString

该属性保存编辑器占位符文本

设置此属性使编辑器显示灰色的占位符文本，只要document（） 是空的。

默认情况下，该属性包含一个空字符串。

访问功能：

QString	**placeholderText() const**

QString	placeholderText() const
void	**setPlaceholderText**(const QString &*placeholderText*)

可以参阅document()。

plainText : QString

此属性将文本编辑器的内容保存为纯文本。

设置该属性后，以前的内容将被删除，撤消/重做历史记录将被重置。currentCharFormat() 也会被重置，除非 textCursor() 已位于文档的开头。

如果文本编辑有其他内容类型，则调用时它不会被纯文本替换toPlainText()。唯一的例外是不间断空格，*nbsp；*，将转换为标准空间。

默认情况下，对于没有内容的编辑器，此属性包含空字符串。

访问功能：

QString	**toPlainText**() const
void	**setPlainText**(const QString &*text*)

可以参阅html。

readOnly : bool

该属性保存文本编辑是否只读

在只读文本编辑中，用户只能浏览文本并选择文本；无法修改文本。

该属性的默认值是 false。

访问功能：

bool	**isReadOnly**() const
void	**setReadOnly**(bool *ro*)

tabChangesFocus : bool

该属性是否持有**Tab**改变焦点或被接受为输入

在某些情况下，文本编辑不应允许用户输入制表符或使用**Tab**键，因为这会破坏焦点链。默认为 false。

访问功能：

bool	**tabChangesFocus() const**
bool	**tabChangesFocus() const**
void	**setTabChangesFocus**(bool *b*)

tabStopDistance : qreal

该属性保存制表位距离（以像素为单位）

默认情况下，此属性包含 80 像素的值。

不要设置小于horizontalAdvance（） 的QChar::VisualTabCharacter字符，否则制表符将绘制不完整。

访问功能：

qreal	**tabStopDistance() const**
void	**setTabStopDistance**(qreal *distance*)

可以参阅QTextOption::ShowTabsAndSpaces和QTextDocument::defaultTextOption。

textInteractionFlags : Qt::TextInteractionFlags

指定小部件应如何与用户输入交互。

默认值取决于是否QTextEdit是只读还是可编辑，以及是否是QTextBrowser或不。

访问功能：

Qt::TextInteractionFlags	**textInteractionFlags() const**
void	**setTextInteractionFlags**(Qt::TextInteractionFlags *flags*)

undoRedoEnabled : bool

该属性保存是否启用撤消和重做。

仅当此属性为 true 并且存在可以撤消（或重做）的操作时，用户才能撤消或重做操作。

访问功能：

bool	**isUndoRedoEnabled() const**
void	**setUndoRedoEnabled**(bool *enable*)

wordWrapMode : QTextOption::WrapMode

该属性保存模式QTextEdit将在按单词换行文本时使用

默认情况下，该属性设置为QTextOption::WrapAtWordBoundaryOrAnywhere。

访问功能：

QTextOption::WrapMode	wordWrapMode() const
void	**setWordWrapMode**(QTextOption::WrapMode *policy*)

可以参阅QTextOption::WrapMode。

成员函数文档

*[explicit]QTextEdit::QTextEdit(QWidget *parent = nullptr)*

构造一个空的 QTextEdit 及其父级*parent*。

*[explicit]QTextEdit::QTextEdit(const QString &text, QWidget *parent = nullptr)*

构造一个带有父级的 QTextEdit*parent*。文本编辑将显示文本*text*。文本被解释为 html。

[virtual]QTextEdit::~QTextEdit()

析构函数。

Qt::Alignment QTextEdit::alignment() const

返回当前段落的对齐方式。

可以参阅setAlignment()。

QString QTextEdit::anchorAt(const QPoint &pos) const

返回位置处锚点的引用*pos*，如果该点不存在锚点，则为空字符串。

[slot]void QTextEdit::append(const QString &text)

附加一个新段落*text*到文本编辑的末尾。

注意：附加的新段落将具有与当前段落相同的字符格式和块格式，具体取决于光标的位置。

可以参阅currentCharFormat（） 和QTextCursor::blockFormat()。

*[virtual protected]bool QTextEdit::canInsertFromMimeData(const QMimeData *source) const*

`true`如果 MIME 数据对象的内容由该函数指定，则此函数返回*source*，可以解码并插入到文档中。例如，当在拖动操作期间鼠标进入该小部件并且需要确定是否可以接受拖放操作时调用它。

重新实现此函数以启用对其他 MIME 类型的拖放支持。

bool QTextEdit::canPaste() const

返回文本是否可以从剪贴板粘贴到文本编辑中。

*[override virtual protected]void QTextEdit::changeEvent(QEvent *e)*

重新实现：QFrame::changeEvent（QEvent *ev）。

[slot]void QTextEdit::clear()

删除文本编辑中的所有文本。

笔记：

- 撤消/重做历史记录也会被清除。
- currentCharFormat() 被重置，除非textCursor() 已位于文档的开头。

可以参阅cut(),setPlainText（），和setHtml()。

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

重新实现：QAbstractScrollArea::contextMenuEvent（QContextMenuEvent *e）。

显示使用创建的标准上下文菜单createStandardContextMenu()。

如果你不希望文本编辑有上下文菜单，你可以设置它contextMenuPolicy到Qt::NoContextMenu。如果您想自定义上下文菜单，请重新实现此函数。如果要扩展标准上下文菜单，请重新实现此函数，调用createStandardContextMenu()并扩展返回的菜单。

有关事件的信息在*event*目的。

```
void MyTextEdit::contextMenuEvent(QContextMenuEvent *event)
{
    QMenu *menu = createStandardContextMenu();
    menu->addAction(tr("My Menu Item"));
    //...
    menu->exec(event->globalPos());
    delete menu;
}
```

[slot]void QTextEdit::copy()

将任何选定的文本复制到剪贴板。

可以参阅copyAvailable()。

[signal]void QTextEdit::copyAvailable(bool yes)

当在文本编辑中选择或取消选择文本时，会发出此信号。

当选择文本时，将发出此信号*yes*设置为 true。如果没有选择任何文本或者如果取消选择所选文本，则发出此信号*yes*设置为假。

如果*yes*那么是真的copy() 可用于将所选内容复制到剪贴板。如果*yes*那么是假的copy（） 什么也没做。

可以参阅selectionChanged()。

[virtual protected]QMimeData *QTextEdit::createMimeDataFromSelection() const

此函数返回一个新的 MIME 数据对象来表示文本编辑当前选择的内容。当需要将选择封装到新的中时调用它QMimeData目的; 例如，当开始拖放操作时，或者当数据复制到剪贴板时。

如果您重新实现此函数，请注意返回的所有权QMimeData对象被传递给调用者。可以使用以下命令检索选择textCursor（） 功能。

QMenu *QTextEdit::createStandardContextMenu()

此函数创建标准上下文菜单，当用户用鼠标右键单击文本编辑时显示该菜单。它是从默认调用的 contextMenuEvent() 处理程序。弹出菜单的所有权转移给调用者。

我们建议您使用 createStandardContextMenu(QPoint) 版本，它将启用对用户单击位置敏感的操作。

QMenu QTextEdit::createStandardContextMenu(const *QPoint* &position*)

此函数创建标准上下文菜单，当用户用鼠标右键单击文本编辑时显示该菜单。它是从默认调用的 contextMenuEvent() 处理程序，它需要*position*在鼠标单击所在的文档坐标中。这可以启用对用户单击位置敏感的操作。弹出菜单的所有权转移给调用者。

QTextCharFormat QTextEdit::currentCharFormat() const

返回插入新文本时使用的字符格式。

可以参阅setCurrentCharFormat()。

[signal]void QTextEdit::currentCharFormatChanged(const *QTextCharFormat* &f)

如果当前字符格式已更改（例如由于光标位置的更改而引起），则会发出此信号。

新格式是*f*。

可以参阅setCurrentCharFormat()。

QFont QTextEdit::currentFont() const

返回当前格式的字体。

可以参阅setCurrentFont(),setFontFamily（），和setFontPointSize()。

QTextCursor QTextEdit::cursorForPosition(const *QPoint* &pos) const

返回一个QTextCursor在位置*pos*（在视口坐标中）。

[signal]void QTextEdit::cursorPositionChanged()

每当光标位置改变时，就会发出该信号。

QRect QTextEdit::cursorRect(const QTextCursor &cursor) const

返回一个矩形（在视口坐标中），其中包括*cursor*。

QRect QTextEdit::cursorRect() const

返回一个矩形（在视口坐标中），其中包含文本编辑的光标。

[slot]void QTextEdit::cut()

将选定的文本复制到剪贴板并将其从文本编辑中删除。

如果没有选定的文本，则不会发生任何事情。

可以参阅copy（） 和paste()。

*[override virtual protected]void QTextEdit::dragEnterEvent(QDragEnterEvent *e)*

重新实现：QAbstractScrollArea::dragEnterEvent（QDragEnterEvent *事件）。

*[override virtual protected]void QTextEdit::dragLeaveEvent(QDragLeaveEvent *e)*

重新实现：QAbstractScrollArea::dragLeaveEvent（QDragLeaveEvent *事件）。

*[override virtual protected]void QTextEdit::dragMoveEvent(QDragMoveEvent *e)*

重新实现：QAbstractScrollArea::dragMoveEvent（QDragMoveEvent *事件）。

*[override virtual protected]void QTextEdit::dropEvent(QDropEvent *e)*

重新实现：QAbstractScrollArea::dropEvent（QDropEvent *事件）。

void QTextEdit::ensureCursorVisible()

如有必要，通过滚动文本编辑确保光标可见。

QList[QTextEdit::ExtraSelection](https://doc-qt-io.translate.goog/qt-6/qtextedit-extraselection.html?_x_tr_sl=auto&_x_tr_tl=zh-CN&_x_tr_hl=zh-CN&_x_tr_pto=wapp) QTextEdit::extraSelections() const

返回先前设置的额外选择。

可以参阅setExtraSelections()。

bool QTextEdit::find(const QString &exp, QTextDocument::FindFlags options = QTextDocument::FindFlags())

查找字符串的下一个出现位置，*exp*，使用给定的*options*。返回`true`如果*exp*找到并更改光标以选择匹配项；否则返回`false`.

bool QTextEdit::find(const QRegularExpression &exp, QTextDocument::FindFlags options = QTextDocument::FindFlags())

这是一个过载功能。

查找与正则表达式匹配的下一个匹配项，*exp*，使用给定的*options*。这QTextDocument::FindCaseSensitively对于此重载，选项被忽略，使用QRegularExpression::CaseInsensitiveOption反而。

如果找到匹配项则返回`true`并更改光标以选择匹配项；否则返回`false`.

*[override virtual protected]void QTextEdit::focusInEvent(QFocusEvent *e)*

重新实现：QWidget::focusInEvent（QFocusEvent *事件）。

[override virtual protected]bool QTextEdit::focusNextPrevChild(bool next)

重新实现：QWidget::focusNextPrevChild（接下来是布尔值）。

*[override virtual protected]void QTextEdit::focusOutEvent(QFocusEvent *e)*

重新实现：QWidget::focusOutEvent（QFocusEvent *事件）。

QString QTextEdit::fontFamily() const

返回当前格式的字体系列。

可以参阅setFontFamily(),setCurrentFont（），和setFontPointSize()。

bool QTextEdit::fontItalic() const

返回true当前格式的字体是否为斜体；否则返回 false。

可以参阅setFontItalic()。

qreal QTextEdit::fontPointSize() const

返回当前格式字体的磅值。

可以参阅setFontFamily(),setCurrentFont（），和setFontPointSize()。

bool QTextEdit::fontUnderline() const

返回`true`当前格式的字体是否有下划线；否则返回 false。

可以参阅setFontUnderline()。

int QTextEdit::fontWeight() const

返回当前格式的字体粗细。

可以参阅setFontWeight(),setCurrentFont(),setFontPointSize（），和QFont::Weight。

*[override virtual protected]void QTextEdit::inputMethodEvent(QInputMethodEvent *e)*

重新实现：QWidget::inputMethodEvent（QInputMethodEvent *事件）。

[override virtual]QVariant QTextEdit::inputMethodQuery(Qt::InputMethodQuery property) const

重新实现：QWidget::inputMethodQuery(Qt::InputMethodQuery query) const。

*[virtual protected]void QTextEdit::insertFromMimeData(const QMimeData *source)*

该函数插入 MIME 数据对象的内容，由*source*，进入当前光标位置的文本编辑。每当由于剪贴板粘贴操作而插入文本时，或者当文本编辑接受来自拖放操作的数据时，都会调用它。

重新实现此函数以启用对其他 MIME 类型的拖放支持。

[slot]void QTextEdit::insertHtml(const QString &text)

方便插入的插槽*text*假定当前光标位置为 html 格式。

它相当于：

```
edit->textCursor().insertHtml(fragment);
```

注意：当将此功能与样式表一起使用时，样式表将仅应用于文档中的当前块。为了在整个文档中应用样式表，请使用QTextDocument::setDefaultStyleSheet（） 反而。

[slot]void QTextEdit::insertPlainText(const QString &text)

方便插入的插槽*text*在当前光标位置。

它相当于

```
edit->textCursor().insertText(text);
```

*[override virtual protected]void QTextEdit::keyPressEvent(QKeyEvent *e)*

重新实现：QAbstractScrollArea::keyPressEvent（QKeyEvent *e）。

*[override virtual protected]void QTextEdit::keyReleaseEvent(QKeyEvent *e)*

重新实现：QWidget::keyReleaseEvent（QKeyEvent *事件）。

[virtual invokable]QVariant QTextEdit::loadResource(int type, const QUrl &name)

加载给定指定的资源*type*和*name*。

这个函数是一个扩展QTextDocument::loadResource()。

注意：该函数可以通过元对象系统和 QML 调用。看Q_INVOKABLE。

可以参阅QTextDocument::loadResource()。

void QTextEdit::mergeCurrentCharFormat(const QTextCharFormat &modifier)

合并指定的属性*modifier*通过调用转换为当前字符格式QTextCursor::mergeCharFormat在编辑器的光标上。如果编辑器有一个选择，则属性*modifier*直接应用于选择。

可以参阅QTextCursor::mergeCharFormat()。

*[override virtual protected]void QTextEdit::mouseDoubleClickEvent(QMouseEvent *e)*

重新实现：QAbstractScrollArea::mouseDoubleClickEvent(QMouseEvent *e)。

*[override virtual protected]void QTextEdit::mouseMoveEvent(QMouseEvent *e)*

重新实现：QAbstractScrollArea::mouseMoveEvent(QMouseEvent *e)。

*[override virtual protected]void QTextEdit::mousePressEvent(QMouseEvent *e)*

重新实现：QAbstractScrollArea::mousePressEvent(QMouseEvent *e)。

*[override virtual protected]void QTextEdit::mouseReleaseEvent(QMouseEvent *e)*

重新实现：QAbstractScrollArea::mouseReleaseEvent(QMouseEvent *e)。

void QTextEdit::moveCursor(QTextCursor::MoveOperation operation, QTextCursor::MoveMode mode = QTextCursor::MoveAnchor)

通过执行给定的操作来移动光标*operation*。

如果*mode*是QTextCursor::KeepAnchor，光标选择它移动到的文本。这与用户按住 Shift 键并使用光标键移动光标时获得的效果相同。

可以参阅QTextCursor::movePosition()。

[override virtual protected]void QTextEdit::paintEvent(QPaintEvent *event)

重新实现：QAbstractScrollArea::paintEvent（QPaintEvent *事件）。

该事件处理程序可以在子类中重新实现以接收传入的绘制事件*event*。通常不需要在子类中重新实现这个函数QTextEdit。

注意：如果您创建QPainter，它必须运行在viewport()。

警告：不得在此函数的重新实现中修改基础文本文档。

[slot]void QTextEdit::paste()

将剪贴板中的文本粘贴到当前光标位置的文本编辑中。

如果剪贴板中没有文本，则不会发生任何事情。

改变这个函数的行为，即修改什么QTextEdit可以粘贴以及如何粘贴，重新实现虚拟canInsertFromMimeData（） 和insertFromMimeData（） 功能。

可以参阅cut（） 和copy()。

void QTextEdit::print(QPagedPaintDevice *printer) const

将文本编辑的文档打印到给定的便捷功能*printer*。这相当于直接在文档上调用 print 方法，只不过该函数还支持 QPrinter::Selection 作为打印范围。

可以参阅QTextDocument::print()。

[slot]void QTextEdit::redo()

重做上次操作。

如果没有要重做的操作，即撤消/重做历史记录中没有重做步骤，则不会发生任何情况。

可以参阅undo()。

[signal]void QTextEdit::redoAvailable(bool available)

每当重做操作可用时就会发出此信号（*available*为真）或不可用（*available*是假的）。

[override virtual protected]void QTextEdit::resizeEvent(QResizeEvent *e)

重新实现：QAbstractScrollArea::resizeEvent（QResizeEvent *事件）。

[override virtual protected]void QTextEdit::scrollContentsBy(int dx, int dy)

重新实现：QAbstractScrollArea::scrollContentsBy（int dx，int dy）。

[slot]void QTextEdit::scrollToAnchor(const QString &name)

滚动文本编辑，以便锚点具有给定的*name*是可见的；如果*name*为空，或已可见，或未找到。

[slot]void QTextEdit::selectAll()

选择所有文本。

可以参阅copy(),cut（），和textCursor()。

[signal]void QTextEdit::selectionChanged()

每当选择发生变化时就会发出此信号。

可以参阅copyAvailable()。

[slot]void QTextEdit::setAlignment(Qt::Alignment a)

将当前段落的对齐方式设置为*a*。有效的对齐方式是Qt::AlignLeft,Qt::AlignRight,Qt::AlignJustify和Qt::AlignCenter（水平居中）。

可以参阅alignment()。

void QTextEdit::setCurrentCharFormat(const QTextCharFormat &format)

设置插入新文本时使用的字符格式*format*通过致电QTextCursor::setCharFormat() 位于编辑器的光标上。如果编辑器有一个选择，则字符格式将直接应用于该选择。

可以参阅currentCharFormat()。

[slot]void QTextEdit::setCurrentFont(const QFont &f)

将当前格式的字体设置为*f*。

可以参阅currentFont(),setFontPointSize（），和setFontFamily()。

void QTextEdit::setExtraSelections(const QList[QTextEdit::ExtraSelection](https://doc-qt-io.translate.goog/qt-6/qtextedit-extraselection.html?_x_tr_sl=auto&_x_tr_tl=zh-CN&_x_tr_hl=zh-CN&_x_tr_pto=wapp) &selections)

此功能允许使用给定颜色临时标记文档中的某些区域，指定为*selections*。例如，这在编程编辑器中很有用，可以用给定的背景颜色标记整行文本以指示断点的存在。

可以参阅QTextEdit::ExtraSelection和extraSelections()。

[slot]void QTextEdit::setFontFamily(const QString &fontFamily)

将当前格式的字体系列设置为*fontFamily*。

可以参阅fontFamily（） 和setCurrentFont()。

[slot]void QTextEdit::setFontItalic(bool italic)

如果*italic*为 true，将当前格式设置为斜体；否则将当前格式设置为非斜体。

可以参阅fontItalic()。

[slot]void QTextEdit::setFontPointSize(qreal s)

将当前格式的磅值设置为*s*。

请注意，如果*s*为零或负数，则该函数的行为未定义。

可以参阅fontPointSize(),setCurrentFont（），和setFontFamily()。

[slot]void QTextEdit::setFontUnderline(bool underline)

如果*underline*为 true，将当前格式设置为下划线；否则将当前格式设置为无下划线。

可以参阅fontUnderline()。

[slot]void QTextEdit::setFontWeight(int weight)

将当前格式的字体粗细设置为给定的*weight*，其中使用的值在由QFont::Weight枚举。

可以参阅fontWeight(),setCurrentFont（），和setFontFamily()。

[slot]void QTextEdit::setPlainText(const QString &text)

将文本编辑的文本更改为字符串*text*。任何先前的文本都将被删除。

笔记：

- *text*被解释为纯文本。
- 撤消/重做历史记录也会被清除。
- currentCharFormat() 被重置，除非textCursor() 已位于文档的开头。

注意：属性的 Setter 函数plainText。

可以参阅toPlainText()。

[slot]void QTextEdit::setText(const QString &text)

设置文本编辑的*text*。文本可以是纯文本或 HTML，文本编辑将尝试猜测正确的格式。

使用setHtml（） 或者setPlainText() 直接避免文本编辑的猜测。

可以参阅toPlainText（） 和toHtml()。

[slot]void QTextEdit::setTextBackgroundColor(const QColor &c)

将当前格式的文本背景颜色设置为*c*。

可以参阅textBackgroundColor()。

[slot]void QTextEdit::setTextColor(const QColor &c)

将当前格式的文本颜色设置为*c*。

可以参阅textColor()。

void QTextEdit::setTextCursor(const QTextCursor &cursor)

设置可见*cursor*。

可以参阅textCursor()。

[override virtual protected]void QTextEdit::showEvent(QShowEvent *)

重新实现：QWidget::showEvent（QShowEvent *事件）。

QColor QTextEdit::textBackgroundColor() const

返回当前格式的文本背景颜色。

可以参阅setTextBackgroundColor()。

[signal]void QTextEdit::textChanged()

每当文档内容发生更改时都会发出此信号；例如，插入或删除文本时，或者应用格式时。

注意：属性的通知程序信号html。属性通知器信号markdown。

QColor QTextEdit::textColor() const

返回当前格式的文本颜色。

可以参阅setTextColor()。

QTextCursor QTextEdit::textCursor() const

返回一个副本QTextCursor代表当前可见的光标。请注意，返回游标的更改不会影响QTextEdit的光标；使用setTextCursor() 更新可见光标。

可以参阅setTextCursor()。

QString QTextEdit::toPlainText() const

QStringQTextEdit::toPlainText() const

以纯文本形式返回文本编辑的文本。

注意：属性的 getter 函数plainText。

可以参阅QTextEdit::setPlainText()。

[slot]void QTextEdit::undo()

撤消上次操作。

如果没有要撤消的操作，即撤消/重做历史记录中没有撤消步骤，则不会发生任何情况。

可以参阅redo()。

[signal]void QTextEdit::undoAvailable(bool available)

每当撤消操作可用时就会发出此信号（*available*为真）或不可用（*available*是假的）。

*[override virtual protected]void QTextEdit::wheelEvent(QWheelEvent *e)*

重新实现：QAbstractScrollArea::wheelEvent(QWheelEvent *e)。

[slot]void QTextEdit::zoomIn(int range = 1)

通过设置基本字体大小来放大文本*range*点变大并重新计算所有字体大小以成为新大小。这不会改变任何图像的大小。

可以参阅zoomOut()。

[slot]void QTextEdit::zoomOut(int range = 1)

通过设置基本字体大小来缩小文本*range*点变小并重新计算所有字体大小以成为新大小。这不会改变任何图像的大小。

可以参阅zoomIn()。