QLineEdit Class

QLineEdit 小部件是一个单行文本编辑器。更多的...

Header:	**#include < QLineEdit>**
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	**ActionPosition { LeadingPosition, TrailingPosition }**
enum	**EchoMode** { Normal, NoEcho, Password, PasswordEchoOnEdit }

特性

acceptableInput : const boolalignment : Qt::AlignmentclearButtonEnabled : boolcursorMoveStyle : Qt::CursorMoveStylecursorPosition : intdisplayText : const QStringdragEnabled : boolechoMode : EchoModeframe : boolhasSelectedText : const bool

inputMask : QStringmaxLength : intmodified : boolplaceholderText : QStringreadOnly : boolredoAvailable : const boolselectedText : const QStringtext : QStringundoAvailable : const bool

公共函数

	QLineEdit(QWidget *parent = nullptr)
	QLineEdit(const QString &*contents*, QWidget *parent = nullptr)
virtual	~QLineEdit()
void	addAction(QAction *action, QLineEdit::ActionPosition *position*)

	QLineEdit(QWidget **parent* = nullptr)
QAction *	**addAction**(const QIcon &*icon*, QLineEdit::ActionPosition *position*)
Qt::Alignment	**alignment**() const
void	**backspace**()
QCompleter *	**completer**() const
QMenu *	**createStandardContextMenu**()
void	**cursorBackward**(bool *mark*, int *steps* = 1)
void	**cursorForward**(bool *mark*, int *steps* = 1)
Qt::CursorMoveStyle	**cursorMoveStyle**() const
int	**cursorPosition**() const
int	**cursorPositionAt**(const QPoint &*pos*)
void	**cursorWordBackward**(bool *mark*)
void	**cursorWordForward**(bool *mark*)
void	**del**()
void	**deselect**()
QString	**displayText**() const
bool	**dragEnabled**() const
QLineEdit::EchoMode	**echoMode**() const
void	**end**(bool *mark*)
bool	**hasAcceptableInput**() const
bool	**hasFrame**() const
bool	**hasSelectedText**() const
void	**home**(bool *mark*)
QString	**inputMask**() const
void	**insert**(const QString &*newText*)
bool	**isClearButtonEnabled**() const
bool	**isModified**() const
bool	**isReadOnly**() const
bool	**isRedoAvailable**() const
bool	**isUndoAvailable**() const

	QLineEdit(QWidget *parent = nullptr)
int	**maxLength**() const
QString	**placeholderText**() const
QString	**selectedText**() const
int	**selectionEnd**() const
int	**selectionLength**() const
int	**selectionStart**() const
void	**setAlignment**(Qt::Alignment *flag*)
void	**setClearButtonEnabled**(bool *enable*)
void	**setCompleter**(QCompleter *c)
void	**setCursorMoveStyle**(Qt::CursorMoveStyle *style*)
void	**setCursorPosition**(int)
void	**setDragEnabled**(bool *b*)
void	**setEchoMode**(QLineEdit::EchoMode)
void	**setFrame**(bool)
void	**setInputMask**(const QString &*inputMask*)
void	**setMaxLength**(int)
void	**setModified**(bool)
void	**setPlaceholderText**(const QString &)
void	**setReadOnly**(bool)
void	**setSelection**(int *start*, int *length*)
void	**setTextMargins**(int *left*, int *top*, int *right*, int *bottom*)
void	**setTextMargins**(const QMargins &*margins*)
void	**setValidator**(const QValidator *v)
QString	**text**() const
QMargins	**textMargins**() const
const QValidator *	**validator**() const

重载的公共函数

virtual bool	**event(QEvent *e) override**
virtual QVariant	inputMethodQuery(Qt::InputMethodQuery *property*) const override
virtual QSize	minimumSizeHint() const override
virtual QSize	sizeHint() const override
virtual void	timerEvent(QTimerEvent *e) override

公共槽

void	**clear()**
void	copy() const
void	cut()
void	paste()
void	redo()
void	selectAll()
void	setText(const QString &)
void	undo()

信号

void	**cursorPositionChanged(int *oldPos*, int *newPos*)**
void	editingFinished()
void	inputRejected()
void	returnPressed()
void	selectionChanged()
void	textChanged(const QString &*text*)
void	textEdited(const QString &*text*)

protected function

QRect	**cursorRect() const**
virtual void	initStyleOption(QStyleOptionFrame *option) const

重载的protected function

virtual void	**changeEvent**(QEvent **ev*) **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 void	**focusOutEvent**(QFocusEvent **e*) override
virtual void	**inputMethodEvent**(QInputMethodEvent **e*) override
virtual void	**keyPressEvent**(QKeyEvent **event*) override
virtual void	**keyReleaseEvent**(QKeyEvent *) 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 *) override

详细说明

行编辑允许用户输入和编辑单行纯文本，并具有一系列有用的编辑功能，包括撤消和重做、剪切和粘贴以及拖放（请参阅setDragEnabled()）。

通过改变echoMode() 的行编辑，它也可以用作“只写”字段，用于输入密码等。

文本的长度可以限制为maxLength()。可以使用任意约束文本validator() 或inputMask （） ， 或两者。在同一行编辑上在验证器和输入掩码之间切换时，最好清除验证器或输入掩码以防止未定义的行为。

一个相关的类是QTextEdit它允许多行、富文本编辑。

您可以使用以下命令更改文本setText （） 或者insert()。文本检索为text(); 显示的文本（可能不同，请参见EchoMode) 检索为displayText()。可以选择文本setSelection （） 或者selectAll()，则选择可cut(),copy() 简易爆炸装置和paste()d．文本可以与setAlignment()。

当文本改变时textChanged() 信号被发射；当文本发生变化而不是通过调用时setText （） 这textEdited() 信号被发射；当光标移动时cursorPositionChanged() 信号被发射；当按下 Return 或 Enter 键时returnPressed() 信号被发射。

编辑完成后，由于行编辑失去焦点或按 Return/Enter 键editingFinished() 信号被发射。请注意，如果在没有进行任何更改的情况下失去焦点，editingFinished() 信号不会被发射。

请注意，如果编辑行上设置了验证器，则returnPressed()/editingFinished() 信号仅在验证器返回时才会发出QValidator::Acceptable。

默认情况下，QLineEdits 具有平台样式指南指定的框架；你可以通过拨打电话将其关闭setFrame（错误的）。

默认键绑定如下所述。行编辑还提供了一个上下文菜单（通常通过单击鼠标右键来调用），其中显示了其中一些编辑选项。

按键	行动
左箭头	将光标向左移动一个字符。
Shift+向左箭头	将文本向左移动并选择一个字符。
右箭头	将光标向右移动一个字符。
Shift+右箭头	将文本向右移动并选择一个字符。
家	将光标移至行首。
结尾	将光标移至行尾。
退格键	删除光标左侧的字符。
Ctrl+退格键	删除光标左侧的单词。
删除	删除光标右侧的字符。
Ctrl+删除	删除光标右侧的单词。
Ctrl+A	全选。
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行编辑中。
Shift+插入	将剪贴板文本粘贴到行编辑中。
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	重做上次撤消的操作。

任何其他代表有效字符的按键序列都会导致该字符被插入到行编辑中。

可以参阅QTextEdit,QLabel,QComboBox， 和Line Edits Example。

成员类型文档

enum QLineEdit::ActionPosition

此枚举类型描述行编辑应如何显示要添加的操作小部件。

持续的	价值	描述
`QLineEdit::LeadingPosition`	`0`	当使用布局方向时，小部件将显示在文本的左侧`Qt::LeftToRight`，而使用 时，小部件将显示在文本的右侧`Qt::RightToLeft`。
`QLineEdit::TrailingPosition`	`1`	使用布局方向时，小部件将显示在文本的右侧`Qt::LeftToRight`；使用 时，小部件将显示在文本的左侧`Qt::RightToLeft`。

可以参阅addAction(),removeAction（），和QWidget::layoutDirection。

enum QLineEdit::EchoMode

此枚举类型描述行编辑应如何显示其内容。

持续的	价值	描述
`QLineEdit::Normal`	`0`	显示输入的字符。这是默认设置。
`QLineEdit::NoEcho`	`1`	不要显示任何东西。这可能适用于甚至密码长度也应保密的密码。
`QLineEdit::Password`	`2`	显示与平台相关的密码掩码字符而不是实际输入的字符。
`QLineEdit::PasswordEchoOnEdit`	`3`	在编辑时显示输入的字符，否则显示与 一样的字符`Password`。

可以参阅setEchoMode（） 和echoMode()。

属性文件

[read-only]acceptableInput : const bool

该属性保存输入是否满足inputMask和验证器。

默认情况下，该属性为`true`。

访问功能：

bool	hasAcceptableInput() const

可以参阅setInputMask（） 和setValidator()。

alignment : Qt::Alignment

该属性保存行编辑的对齐方式。

这里允许水平和垂直对齐，Qt::AlignJustify将映射到Qt::AlignLeft。

默认情况下，此属性包含以下组合Qt::AlignLeft和Qt::AlignVCenter。

访问功能：

Qt::Alignment	alignment() const
void	**setAlignment**(Qt::Alignment *flag*)

可以参阅Qt::Alignment。

clearButtonEnabled : bool

该属性保存行编辑不为空时是否显示清除按钮。

如果启用，行编辑在包含某些文本时会显示尾随*清除按钮，否则行编辑不会显示清除按钮（默认）*。

访问功能：

bool	isClearButtonEnabled() const
void	**setClearButtonEnabled**(bool *enable*)

可以参阅addAction（） 和removeAction()。

cursorMoveStyle : Qt::CursorMoveStyle

该属性保存了该行编辑中光标的移动方式。

当该属性设置为Qt::VisualMoveStyle，线条编辑将使用视觉运动风格。无论文本的书写方向如何，按向左箭头键始终会导致光标向左移动。相同的行为适用于右箭头键。

当该财产是Qt::LogicalMoveStyle（默认），在 LTR 文本块内，按向左箭头键时增加光标位置，按向右箭头键时减少光标位置。如果文本块从右到左，则应用相反的行为。

访问功能：

Qt::CursorMoveStyle	cursorMoveStyle() const
void	**setCursorMoveStyle**(Qt::CursorMoveStyle *style*)

cursorPosition : int

该属性保存该行编辑的当前光标位置。

设置光标位置会在适当的时候导致重新绘制。

默认情况下，此属性包含值 0。

访问功能：

int	cursorPosition() const
void	**setCursorPosition**(int)

[read-only]displayText : const QString

该属性保存显示的文本。

如果echoMode是Normal这返回相同text(); 如果EchoMode是Password或者PasswordEchoOnEdit它返回一串与平台相关的密码掩码字符text().length() 的大小，例如“**”；如果EchoMode是NoEcho返回一个空字符串“”。

默认情况下，该属性包含一个空字符串。

访问功能：

QString	displayText() const

可以参阅setEchoMode(),text（），和EchoMode。

dragEnabled : bool

此属性保存如果用户在某些选定的文本上按下并移动鼠标，线条编辑是否开始拖动。

默认情况下禁用拖动。

访问功能：

bool	dragEnabled() const
void	**setDragEnabled**(bool *b*)

echoMode : EchoMode

该属性保存行编辑的回显模式。

回显模式确定如何向用户显示（或回显）行编辑中输入的文本。

最常见的设置是Normal，其中用户输入的文本逐字显示，但是QLineEdit还支持允许抑制或遮盖输入文本的模式：这些模式包括NoEcho,Password和PasswordEchoOnEdit。

小部件的显示以及复制或拖动文本的能力受此设置的影响。

默认情况下，该属性设置为Normal。

访问功能：

QLineEdit::EchoMode	echoMode() const
void	**setEchoMode**(QLineEdit::EchoMode)

可以参阅EchoMode和displayText()。

frame : bool

该属性保存线条编辑是否使用框架绘制自身。

如果启用（默认），线条编辑将在框架内绘制自身，否则线条编辑将在没有任何框架的情况下绘制自身。

访问功能：

bool	hasFrame() const
void	**setFrame**(bool)

[read-only]hasSelectedText : const bool

该属性保存是否有任何文本被选中。

`true`如果用户选择了部分或全部文本，则hasSelectedText() 返回；否则返回`false`.

默认情况下，该属性为`false`。

访问功能：

bool	hasSelectedText() const

可以参阅selectedText()。

inputMask : QString

该属性保存验证输入掩码。

如果未设置掩码，则 inputMask() 返回空字符串。

设置QLineEdit的验证掩码。验证器可以代替掩码使用，或者与掩码一起使用；看setValidator()。

解除遮罩并恢复正常QLineEdit通过传递空字符串（“”）进行操作。

输入掩码是输入模板字符串。它可以包含以下元素：

面具人物	**定义Category在此位置被认为有效的输入字符数**
元字符	各种特殊含义
分离器	所有其他字符都被视为不可变分隔符

下表显示了可在输入掩码中使用的掩码和元字符。

掩码	**意义**
A	所需字母类别的字符，例如 AZ、az。
a	允许但不要求的字母类别的字符。
N	所需字母或数字类别的字符，例如 AZ、az、0-9。
n	允许但不要求的字母或数字类别的字符。
X	需要任何非空白字符。
x	允许但不是必需的任何非空白字符。
9	所需数字类别的字符，例如 0-9。
0	允许但不要求的数字类别的字符。
D	Number 类别的字符，且必须大于零，例如 1-9
d	Number 类别的字符，允许但不要求大于零，例如 1-9。
#	数字类别的字符，或允许但不必需的加号/减号。
H	需要十六进制字符。AF，AF，0-9。
h	允许使用十六进制字符，但不是必需的。
B	需要二进制字符。0-1。
b	允许但不要求使用二进制字符。
元字符	意义
>	以下所有字母字符均为大写。
<	以下所有字母字符均为小写。

掩码	意义
!	关闭大小写转换。
;c	终止输入掩码并将*空白*字符设置为*c*。
[ ] { }	预订的。
\	用于\转义上面列出的特殊字符以将其用作分隔符。

创建或清除时，行编辑将填充输入掩码字符串的副本，其中元字符已被删除，并且掩码字符已替换为空白字符（默认情况下为*a*）space。

当设置输入掩码时，text() 方法返回行编辑内容的修改副本，其中所有*空白*字符已被删除。可以使用以下方式读取未修改的内容displayText()。

这hasAcceptableInput如果行编辑的当前内容不满足输入掩码的要求，() 方法将返回 false。

例子：

掩码	备注
000.000.000.000;_	IP地址; 空格是_.
HH:HH:HH:HH:HH:HH;_	MAC地址
0000-00-00	ISO 日期；空白是space
>AAAAA-AAAAA-AAAAA-AAAAA-AAAAA;#	许可证号; 空格#和所有（字母）字符都转换为大写。

要获得范围控制（例如，对于 IP 地址），请将掩码与validators。

访问功能：

QString	**inputMask() const**
void	**setInputMask**(const QString &*inputMask*)

可以参阅maxLength,QChar::isLetter(),QChar::isNumber（），和QChar::digitValue()。

maxLength : int

该属性保存文本的最大允许长度。

如果文本太长，则会在限制处被截断。

如果发生截断，任何选定的文本都将被取消选择，光标位置将设置为 0，并显示字符串的第一部分。

如果行编辑具有输入掩码，则该掩码定义最大字符串长度。

默认情况下，此属性包含值 32767。

访问功能：

int	maxLength() const
void	**setMaxLength**(int)

可以参阅inputMask。

modified : bool

该属性保存行编辑的内容是否已被用户修改。

修改后的标志永远不会被读取QLineEdit; 它的默认值为 false，并且每当用户更改行编辑的内容时就会更改为 true。

这对于需要提供默认值但一开始不知道默认值应该是什么（可能取决于表单上的其他字段）的事情很有用。在没有最佳默认值的情况下开始行编辑，当默认值已知时，如果 modded() 返回`false`（用户尚未输入任何文本），则插入默认值。

呼唤setText() 将修改标志重置为 false。

访问功能：

bool	isModified() const
void	**setModified**(bool)

placeholderText : QString

该属性保存行编辑的占位符文本。

只要行编辑为空，设置此属性就会使行编辑显示灰色的占位符文本。

通常，空行编辑会显示占位符文本，即使它具有焦点也是如此。但是，如果内容水平居中，则当行编辑具有焦点时，占位符文本不会显示在光标下方。

默认情况下，该属性包含一个空字符串。

访问功能：

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

可以参阅text()。

readOnly : bool

该属性保存行编辑是否为只读。

在只读模式下，用户仍然可以将文本复制到剪贴板，或拖放文本（如果echoMode（） 是Normal），但无法编辑。

QLineEdit在只读模式下不显示光标。

默认情况下，该属性为`false`。

访问功能：

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

可以参阅setEnabled()。

[read-only]redoAvailable : const bool

该属性保存重做是否可用。

一旦用户对行编辑中的文本执行了一项或多项撤消操作，重做就变得可用。

默认情况下，该属性为`false`。

访问功能：

bool	**isRedoAvailable() const**

[read-only]selectedText : const QString

该属性保存选定的文本。

如果没有选定的文本，则该属性的值为空字符串。

默认情况下，该属性包含一个空字符串。

访问功能：

QString	**selectedText() const**

可以参阅hasSelectedText()。

text : QString

该属性保存行编辑的文本。

设置此属性将清除选择、清除撤消/重做历史记录、将光标移动到行尾并重置modified属性为 false。使用 setText() 插入文本时，不会验证文本。

文本被截断为maxLength（） 长度。

默认情况下，该属性包含一个空字符串。

访问功能：

QString	**text() const**
void	**setText**(const QString &)

通知器信号：

void	**textChanged(const QString &*text*)**

可以参阅insert（） 和clear()。

[read-only]undoAvailable : const bool

该属性保存撤消是否可用。

一旦用户修改了行编辑中的文本，撤消操作就可用。

默认情况下，该属性为false。

访问功能：

bool	**isUndoAvailable() const**

成员函数文档

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

构造一个没有文本的行编辑。

最大文本长度设置为 32767 个字符。

这*parent*参数被发送到QWidget构造函数。

可以参阅setText（） 和setMaxLength()。

*[explicit]QLineEdit::QLineEdit(const QString &contents, QWidget *parent = nullptr)*

构造包含文本的行编辑*contents*。

光标位置设置为行尾，最大文本长度设置为 32767 个字符。

这*parent*并且参数被发送到QWidget构造函数。

可以参阅text（） 和setMaxLength()。

[virtual]QLineEdit::~QLineEdit()

破坏行编辑。

*void QLineEdit::addAction(QAction *action, QLineEdit::ActionPosition position)*

添加了*action*到操作列表*position*。

QAction QLineEdit::addAction(const QIcon &icon, QLineEdit::ActionPosition position)*

这是一个过载功能。

使用给定的创建一个新动作*icon*在*position*。

void QLineEdit::backspace()

如果未选择任何文本，则删除文本光标左侧的字符并将光标向左移动一位。如果选择任何文本，光标将移动到所选文本的开头，并删除所选文本。

可以参阅del()。

*[override virtual protected]void QLineEdit::changeEvent(QEvent *ev)*

重新实现：QWidget::changeEvent（QEvent *事件）。

[slot]void QLineEdit::clear()

清除行编辑的内容。

可以参阅setText（） 和insert()。

*QCompleter *QLineEdit::completer() const*

返回当前值QCompleter提供完成。

可以参阅setCompleter()。

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

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

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

如果你不希望行编辑有上下文菜单，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。

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

这*event*参数用于获取事件发生时鼠标光标所在的位置。

可以参阅setContextMenuPolicy()。

[slot]void QLineEdit::copy() const

将选定的文本复制到剪贴板（如果有的话）echoMode（） 是Normal。

可以参阅cut（） 和paste()。

QMenu *QLineEdit::createStandardContextMenu()

此函数创建标准上下文菜单，当用户用鼠标右键单击行编辑时显示该菜单。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。

void QLineEdit::cursorBackward(bool mark, int steps = 1)

将光标向后移动*steps*人物。如果*mark*为 true 时，移动的每个字符都会添加到选择中；如果*mark*为 false 则清除选择。

可以参阅cursorForward()。

void QLineEdit::cursorForward(bool mark, int steps = 1)

向前移动光标*steps*人物。如果*mark*为 true 时，移动的每个字符都会添加到选择中；如果*mark*为 false 则清除选择。

可以参阅cursorBackward()。

int QLineEdit::cursorPositionAt(const QPoint &pos)

返回该点下的光标位置*pos*。

[signal]void QLineEdit::cursorPositionChanged(int oldPos, int newPos)

每当光标移动时就会发出此信号。先前的位置由下式给出*oldPos*，以及新位置*newPos*。

可以参阅setCursorPosition（） 和cursorPosition()。

[protected]QRect QLineEdit::cursorRect() const

返回包含线条编辑光标的矩形。

void QLineEdit::cursorWordBackward(bool mark)

将光标向后移动一个字。如果*mark*确实如此，这个词也被选中了。

可以参阅cursorWordForward()。

void QLineEdit::cursorWordForward(bool mark)

将光标向前移动一个字。如果*mark*确实如此，这个词也被选中了。

可以参阅cursorWordBackward()。

[slot]void QLineEdit::cut()

将选定的文本复制到剪贴板并将其删除（如果有的话）echoMode（） 是Normal。

如果当前验证器不允许删除所选文本，则 cut() 将复制而不删除。

可以参阅copy(),paste（），和setValidator()。

void QLineEdit::del()

如果未选择任何文本，则删除文本光标右侧的字符。如果选择任何文本，光标将移动到所选文本的开头，并删除所选文本。

可以参阅backspace()。

void QLineEdit::deselect()

取消选择任何选定的文本。

可以参阅setSelection（） 和selectAll()。

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

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

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

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

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

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

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

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

[signal]void QLineEdit::editingFinished()

当按下 Return 或 Enter 键时，或者如果行编辑失去焦点并且自上次发出此信号以来其内容已更改，则会发出此信号。

请注意，如果有一个validator （） 或者inputMask() 在行 edit 上设置并按下 Enter/Return，只有当输入遵循inputMask() 和validator() 返回QValidator::Acceptable。

void QLineEdit::end(bool mark)

将文本光标移动到行尾，除非它已经在那里。如果*mark*为 true 时，文本将被选择到最后一个位置；否则，如果移动光标，则取消选择任何选定的文本。

可以参阅home()。

*[override virtual]bool QLineEdit::event(QEvent *e)*

重新实现：QWidget::event（QEvent *事件）。

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

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

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

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

void QLineEdit::home(bool mark)

将文本光标移动到行首，除非它已经在那里。如果*mark*为 true 时，文本将被选择到第一个位置；否则，如果移动光标，则取消选择任何选定的文本。

可以参阅end()。

*[virtual protected]void QLineEdit::initStyleOption(QStyleOptionFrame *option) const*

初始化*option*与此值QLineEdit。当子类需要时，此方法非常有用QStyleOptionFrame，但不想自己填写所有信息。

可以参阅QStyleOption::initFrom()。

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

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

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

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

[signal]void QLineEdit::inputRejected()

当用户按下不被认为是可接受的输入的键时，会发出此信号。例如，如果按键导致验证器的 validate() 调用返回 Invalid。另一种情况是尝试输入超出行编辑最大长度的更多字符时。

注意：在部分文本被接受但不是全部被接受的情况下，仍然会发出此信号。例如，如果设置了最大长度，并且粘贴时剪贴板文本比最大长度长。

void QLineEdit::insert(const QString &newText)

删除任何选定的文本，插入*newText*，并验证结果。如果有效，则将其设置为行编辑的新内容。

可以参阅setText（） 和clear()。

*[override virtual protected]void QLineEdit::keyPressEvent(QKeyEvent *event)*

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

转换给定的按键*event*进入行编辑操作。

如果按下 Return 或 Enter 键并且当前文本有效（或者可以是made valid由验证器），信号returnPressed() 被发射。

类的详细描述中列出了默认的键绑定。

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

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

[override virtual]QSize QLineEdit::minimumSizeHint() const

重新实现属性的访问函数：QWidget::minimumSizeHint。

返回行编辑的最小大小。

返回的宽度通常足以容纳至少一个字符。

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

重新实现：QWidget::mouseDoubleClickEvent（QMouseEvent *事件）。

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

重新实现：QWidget::mouseMoveEvent（QMouseEvent *事件）。

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

重新实现：QWidget::mousePressEvent（QMouseEvent *事件）。

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

重新实现：QWidget::mouseReleaseEvent（QMouseEvent *事件）。

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

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

[slot]void QLineEdit::paste()

在光标位置插入剪贴板的文本，删除任何选定的文本，前提是不进行行编辑read-only。

如果最终结果对当前无效validator， 什么都没发生。

可以参阅copy（） 和cut()。

[slot]void QLineEdit::redo()

如果重做，则重做最后一次操作available。

[signal]void QLineEdit::returnPressed()

当按下 Return 或 Enter 键时会发出此信号。请注意，如果有一个validator（） 或者inputMask() 设置在行编辑上，只有当输入遵循以下命令时才会发出 returnPressed() 信号inputMask() 和validator() 返回QValidator::Acceptable。

[slot]void QLineEdit::selectAll()

选择所有文本（即突出显示它）并将光标移动到末尾。当插入默认值时这非常有用，因为如果用户在单击小部件之前键入，则所选文本将被删除。

可以参阅setSelection（） 和deselect()。

[signal]void QLineEdit::selectionChanged()

每当选择发生变化时就会发出此信号。

可以参阅hasSelectedText（） 和selectedText()。

int QLineEdit::selectionEnd() const

在行编辑中选择后直接返回字符的索引，如果未选择文本，则返回 -1。

可以参阅selectedText(),selectionStart（）， 和selectionLength()。

int QLineEdit::selectionLength() const

返回选择的长度。

可以参阅selectedText(),selectionStart（），和selectionEnd()。

int QLineEdit::selectionStart() const

返回行编辑中第一个选定字符的索引；如果未选择任何文本，则返回 -1。

可以参阅selectedText(),selectionEnd（），和selectionLength()。

*void QLineEdit::setCompleter(QCompleter *c)*

设置此行编辑以提供来自完成者的自动完成，*c*。完成模式设置使用QCompleter::setCompletionMode()。

要使用QCompleter与一个QValidator或者QLineEdit::inputMask，您需要确保提供的模型QCompleter包含有效条目。您可以使用QSortFilterProxyModel以确保QCompleter的模型仅包含有效条目。

如果*c*== 0，setCompleter() 删除当前完成器，有效禁用自动完成。

可以参阅completer（） 和QCompleter。

void QLineEdit::setSelection(int start, int length)

从位置选择文本*start*并为*length*人物。允许负长度。

可以参阅deselect(),selectAll（），和selectedText()。

void QLineEdit::setTextMargins(int left, int top, int right, int bottom)

设置框架内文本周围的边距以具有尺寸*left*,*top*,*right*，和*bottom*。

可以参阅textMargins()。

可以参阅textMargins()。

void QLineEdit::setTextMargins(const QMargins &margins)

设置*margins*围绕框架内的文本。

可以参阅textMargins()。

*void QLineEdit::setValidator(const QValidator *v)*

将行编辑值的验证器设置为*v*。

行编辑的returnPressed（） 和editingFinished() 信号只有在以下情况下才会被发射：*v*验证行编辑的内容为Acceptable。用户可以将内容更改为任意Intermediate编辑期间的值，但将阻止将文本编辑为以下值*v*验证为Invalid。

这允许您限制编辑完成后最终输入的文本，同时让用户有足够的自由将文本从一种有效状态编辑为另一种有效状态。

如果*v*== 0，setValidator() 删除当前输入验证器。初始设置是没有输入验证器（即接受任何输入maxLength()）。

可以参阅validator(),hasAcceptableInput(),QIntValidator,QDoubleValidator， 和QRegularExpressionValidator。

[override virtual]QSize QLineEdit::sizeHint() const

重新实现属性的访问函数：QWidget::sizeHint。

返回小部件的建议大小。

返回的宽度（以像素为单位）通常足以容纳大约 15 到 20 个字符。

[signal]void QLineEdit::textChanged(const QString &text)

每当文本更改时就会发出此信号。这*text*参数是新文本。

不像textEdited()，当以编程方式更改文本时也会发出此信号，例如通过调用setText()。

注意：属性的通知程序信号text。

[signal]void QLineEdit::textEdited(const QString &text)

每当编辑文本时都会发出此信号。这*text*参数是新文本。

不像textChanged()，当以编程方式更改文本时，例如通过调用setText()。

QMargins QLineEdit::textMargins() const

返回小部件的文本边距。

可以参阅setTextMargins()。

*[override virtual]void QLineEdit::timerEvent(QTimerEvent *e)*

重新实现：QObject::timerEvent（QTimerEvent *事件）。

[slot]void QLineEdit::undo()

如果撤消，则撤消上一次操作available。取消选择任何当前选择，并将选择开始更新为当前光标位置。

*const QValidator *QLineEdit::validator() const*

返回指向当前输入验证器的指针，或者nullptr如果尚未设置验证器。

可以参阅setValidator()。