【文本编辑器】四、文档排版美化功能（上）

原创 Qt 学习 Qt 学习 2020-02-25

本篇将对文本的加粗、倾斜、下画线和文本对齐与颜色设置这些美化操作进行讲解，与这些操作对应的工具条按钮的可用（激活）状态设置也在实现这些文本操作后实现。

本篇目录

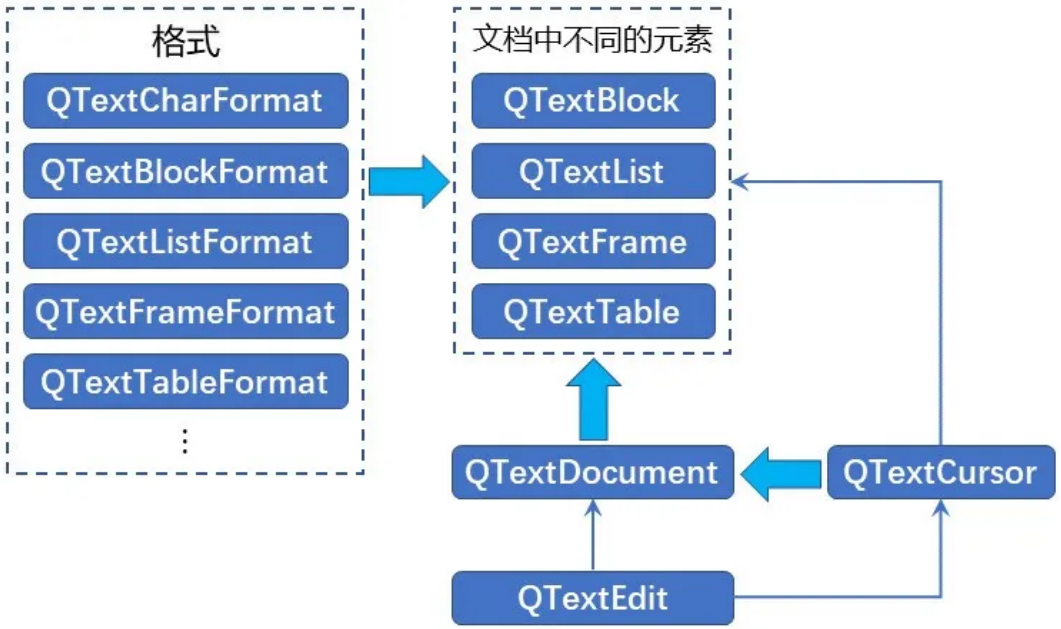
运行环境：

win 10 + Qt 5.12.5 + Qt Creator 4.10

1. 与文本编辑有关的类

任何一个文本编辑的程序都要用到 `QTextEdit` 作为输入文本的容器，在它里面输入可编辑文本由 `QTextDocument` 作为载体，而用来表示 `QTextDocument` 的元素的 `QTextBlock` 、 `QTextList` 、 `QTextFrame` 等是 `QTextDocument` 的不同表现形式，可以表示为字符串、段落、列表、表格或图片等。

在编写包含格式设置的文本编辑程序时，经常用到的 Qt 类及其之间的关系如下



文本编辑各类之间的划分与关系

每种元素都有自己的格式，这些格式则用的 `QTextCharFormat` 、 `QTextBlockFormat` 、 `QTextListFormat` 、 `QTextFrameFormat` 等类来描述与实现。例如， `QTextBlockFormat` 类对应于 `QTextBlock` 类， `QTextBlock` 类用于表示一块文本，通常可以理解为一个段落，但它并不仅指段落； `QTextBlockFormat` 类则表示这一块文本的格式，如缩进值，与四边的边距等。

从上图中可以看出用于表示编辑文本中的光标 `QTextCursor` 类是一个非常重要也经常会用到的类，它提供了对 `QTextDocument` 文档的修改接口，所有对文档格式的修改，说到底都与光标有关。例如，改变字符的格式，实际上指的是改变光标处字符的格式。又例

如，改变段落的格式，实际上指的是改变光标所在段落的格式。因此，所有对 `QTextDocument` 的修改都能够通过 `QTextCursor` 类实现， `QTextCursor` 类在文档编辑类程序中有着重要作用。

2. 加粗、倾斜与下画线

文本编辑窗口通过 `mergeFormatOnWordOrSelection()` 函数设置文本格式，在 "textedit.h" 头文件中声明

```
1  private:
2      void mergeFormatOnWordOrSelection(const QTextCharFormat &format);
```

并添加 `QTextCharFormat` 类的前置声明

```
1  class QTextCharFormat;
```

以及在 "textedit.cpp" 中添加如下头文件：

```
1  #include <QTextCharFormat>
2  #include <QTextCursor>
```

编写函数的实现如下

```
1  void TextEdit::mergeFormatOnWordOrSelection(const QTextCharFormat &format)
2  {
3      QTextCursor cursor = textEdit->textCursor();
4      if (!cursor.hasSelection())
5          cursor.select(QTextCursor::WordUnderCursor);
6      cursor.mergeCharFormat(format);
```

```
7       textEdit->mergeCurrentCharFormat(format);
8   }
```

通过调用 `QTextCursor::mergeCharFormat()` 函数，将参数 `format` 所表示的格式应用到光标所在处的文本字符上。

以上为基础，接下来实现“加粗”、“倾斜”和“下画线”功能，在 "textedit.h" 中添加对应的槽函数声明：

```
1   private slots:
2       void textBold();
3       void textItalic();
4       void textUnderline();
```

函数的实现代码如下：

```
 1   void TextEdit::textBold()
 2   {
 3       QTextCharFormat fmt;
 4       fmt.setFontWeight(actionTextBold->isChecked() ? QFont::Bold : QFont::Normal);
 5       mergeFormatOnWordOrSelection(fmt);
 6   }
 7
 8   void TextEdit::textItalic()
 9   {
10       QTextCharFormat fmt;
11       fmt.setFontItalic(actionTextItalic->isChecked());
12       mergeFormatOnWordOrSelection(fmt);
13   }
14
15   void TextEdit::textUnderline()
16   {
17       QTextCharFormat fmt;
18       fmt.setFontUnderline(actionTextUnderline->isChecked());
19       mergeFormatOnWordOrSelection(fmt);
20   }
```

其中，

- 调用 `QTextCharFormat::setFontWeight()` 函数设置粗细值，若检测到『加粗』按钮按下则设置字符的 `Weight` 值为 `QFont::Bold`；反之，则设为 `QFont::Normal`。

- 文字的粗细值由为 `QFont::Weight` 表示，他是一个整型值，取值范围可为 0~99，有 5 个预设值：`QFont::Light(25)`、`QFont::Normal(50)`、`QFont::DemiBold(63)`、`QFont::Bold(75)` 和 `QFont::Black(87)`，通常在 `QFont::Normal` 和 `QFont::Bold` 之间转换。

添加关联函数（加粗代码行）

```
1  void TextEdit::setupTextActions()
2  {
3      // 格式主菜单动作集
4      ...
5      QFont bold;                          // 定义加粗字体
6      bold.setBold(true);                  // 激活字体加粗
7      actionTextBold->setFont(bold);       // 将加粗功能赋给加粗动作
8      actionTextBold->setCheckable(true); // 设置为可以选中
9      connect(actionTextBold, &QAction::triggered, this, &TextEdit::textBold);
10
11     ...
12     QFont italic;                          // 定义倾斜字体
13     italic.setItalic(true);                // 激活字体倾斜
14     actionTextItalic->setFont(italic);     // 将倾斜功能赋给倾斜动作
15     actionTextItalic->setCheckable(true); // 设置为可以选中
16     connect(actionTextItalic, &QAction::triggered, this, &TextEdit::textItalic);
17
18     ...
19     QFont underline;                         // 定义字体下画线
20     underline.setUnderline(true);            // 激活字体下画线
21     actionTextUnderline->setFont(underline); // 将下画线功能赋给下画线动作
```

```
22      actionTextUnderline->setCheckable(true); // 设置为可以选中
23      connect(actionTextUnderline, &QAction::triggered, this, &TextEdit::textUnderline);
24
25      ...
26  }
```

运行程序，在文本编辑框中输入文本，实现了字体“加粗”、“倾斜”和“下画线”功能。不过此时将光标放在不同格式的字体上时，工具条按钮并没有随之产生相应的激活/非激活状态转变。

在 "textedit.h" 头文件中函数声明

```
1   private:
2       void fontChanged(const QFont &f);
3
4   private slots:
5       void currentCharFormatChanged(const QTextCharFormat &format);
```

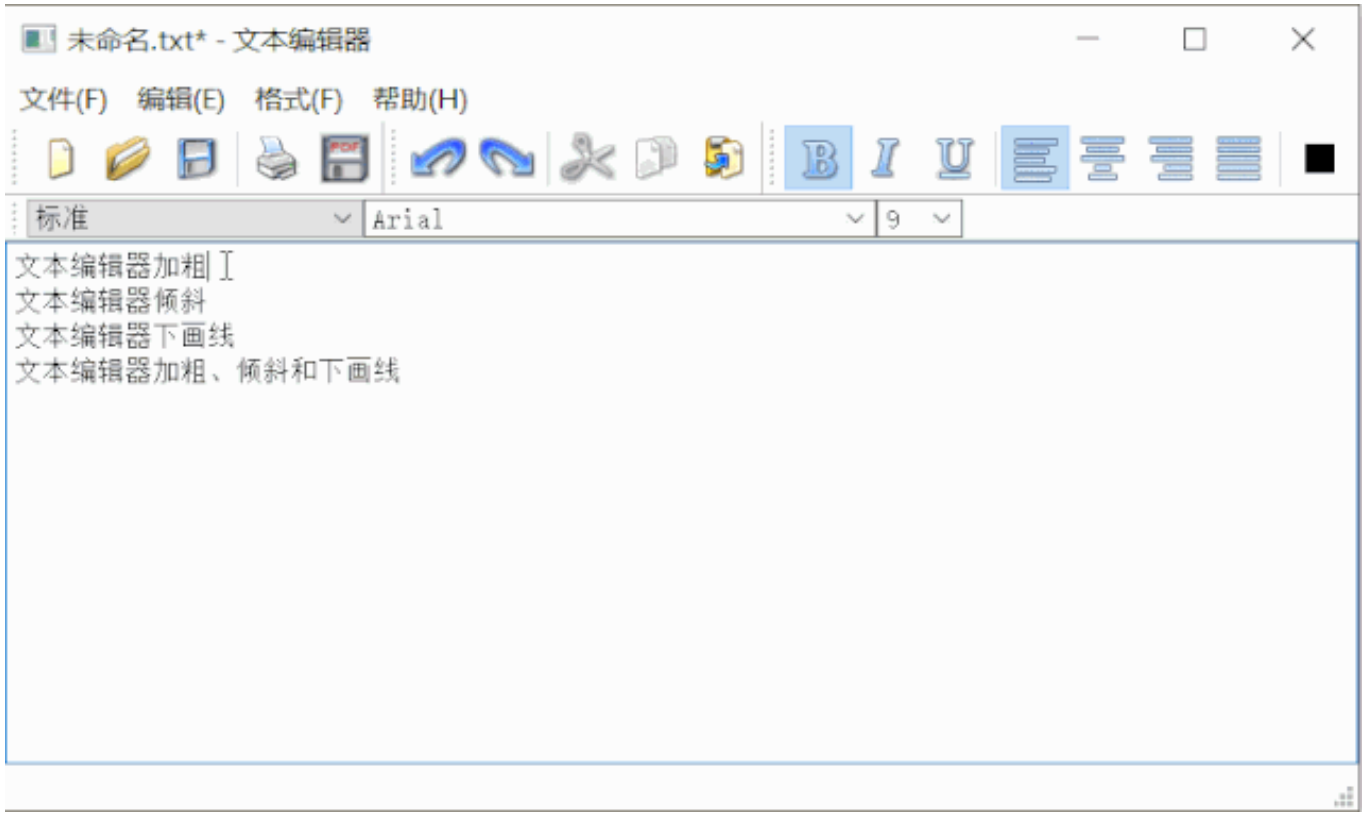
以及对应的实现函数代码

```
1   void TextEdit::fontChanged(const QFont &f)
2   {
3       actionTextBold->setChecked(f.bold());
4       actionTextItalic->setChecked(f.italic());
5       actionTextUnderline->setChecked(f.underline());
6   }
7
8   void TextEdit::currentCharFormatChanged(const QTextCharFormat &format)
9   {
10      fontChanged(format.font());
11  }
```

添加字体格式改变的 `connect` 函数

```
1   connect(textEdit, &QTextEdit::currentCharFormatChanged, this, &TextEdit::currentCharFormatChanged);
```

运行程序，在文本编辑框中输入三小段文本，并将其分别设置为“加粗”、“倾斜”和“下画线”字体，效果如下



注意：将光标放置在不同格式字体上时，对应的『加粗』『倾斜』和『下画线』按钮激活状态的变化

3. 段落对齐设置

按照上一节的实现逻辑，比较容易实现本文对齐和文本颜色操作功能，下面列出主要过程。

在 "textedit.h" 头文件中添加对齐槽函数的声明

```
1  private slots:
2      void textAlign(QAction *a);
```

实现代码如下

```
1  void TextEdit::textAlign(QAction *a)
2  {
3      if (a == actionAlignLeft)
4          textEdit->setAlignment(Qt::AlignLeft | Qt::AlignAbsolute);
5      else if (a == actionAlignCenter)
6          textEdit->setAlignment(Qt::AlignHCenter);
7      else if (a == actionAlignRight)
8          textEdit->setAlignment(Qt::AlignRight | Qt::AlignAbsolute);
9      else if (a == actionAlignJustify)
10         textEdit->setAlignment(Qt::AlignJustify);
11 }
```

添加关联函数

```
1  void TextEdit::setupTextActions()
2  {
3      // 格式主菜单动作集
4      ...
5
6      // 确保 alignLeft 始终位于 alignRight 的左侧
7      QActionGroup *alignGroup = new QActionGroup(this);
8      connect(alignGroup, &QActionGroup::triggered, this, &TextEdit::textAlign);
9
10     if (QApplication::isLeftToRight()) {
```

```
        alignGroup->addAction(actionAlignLeft);
        alignGroup->addAction(actionAlignCenter);
        alignGroup->addAction(actionAlignRight);
    } else {
        alignGroup->addAction(actionAlignRight);
        alignGroup->addAction(actionAlignCenter);
        alignGroup->addAction(actionAlignLeft);
    }
    alignGroup->addAction(actionAlignJustify);

    ...
}
```

上述过程实现了对齐功能，同样，此时的段落对齐按钮的激活状态不会随光标位置的改变而切换。添加实现相应功能的函数声明

```
private:
    void alignmentChanged(Qt::Alignment a);

private slots:
    void cursorPositionChanged();
```

函数实现代码

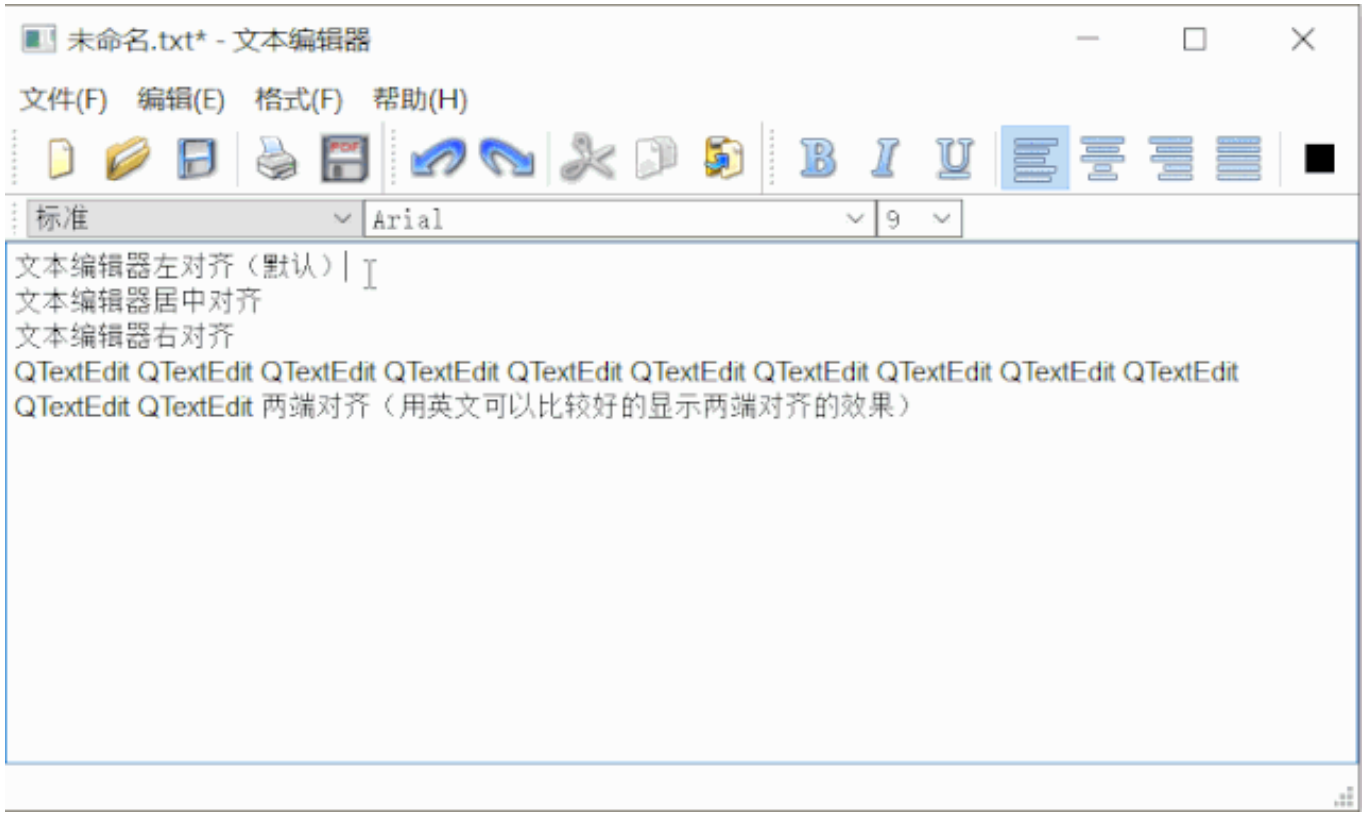
```
void TextEdit::alignmentChanged(Qt::Alignment a)
{
    if (a & Qt::AlignLeft)
        actionAlignLeft->setChecked(true);
    else if (a & Qt::AlignHCenter)
        actionAlignCenter->setChecked(true);
    else if (a & Qt::AlignRight)
        actionAlignRight->setChecked(true);
    else if (a & Qt::AlignJustify)
        actionAlignJustify->setChecked(true);
}

```

```
13  void TextEdit::cursorPositionChanged()
14  {
15      alignmentChanged(textEdit->alignment());
16  }
```

添加文本对齐功能的 `connect` 函数

```
1   connect(textEdit, &QTextEdit::cursorPositionChanged, this, &TextEdit::cursorPositionChanged);
```

运行程序，效果如下



4. 文本颜色设置

在 "textedit.h" 中添加颜色设置的函数声明

```
1  private slots:
2      void textColor();
3
4  private:
5      void colorChanged(const QColor &c);
```

函数实现代码如下

```
 1  void TextEdit::textColor()
 2  {
 3      QColor col = QColorDialog::getColor(textEdit->textColor(), this);
 4      if (!col.isValid())
 5          return;
 6      QTextCharFormat fmt;
 7      fmt.setForeground(col);
 8      mergeFormatOnWordOrSelection(fmt);
 9      colorChanged(col);
10  }
11
12  void TextEdit::colorChanged(const QColor &c)
13  {
14      QPixmap pix(16, 16);
15      pix.fill(c);
16      actionTextColor->setIcon(pix);
17  }
```

添加颜色标准对话框的声明

```
#include <QColorDialog>
```

其中，

- `QColor col = QColorDialog::getColor(textEdit->textColor(), this);` 使用了标准颜色对话框的方式，当单击触发颜色按钮时，弹出标准颜色对话框选择颜色。

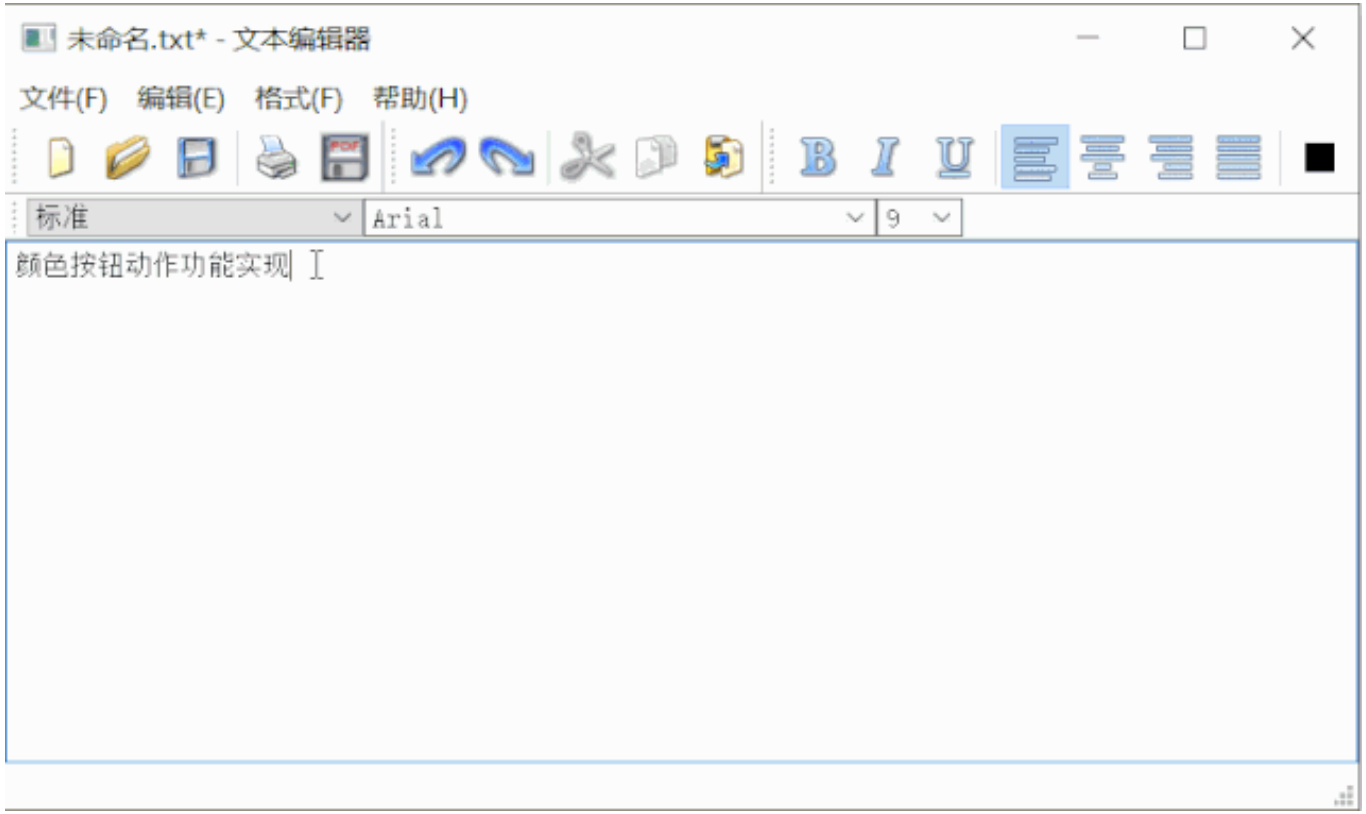
取消颜色按钮动作 `connect` 函数的注释

```
connect(actionTextColor, &QAction::triggered, this, &TextEdit::textColor);
```

将 `colorChanged()` 函数放入 `currentCharFormatChanged()` 函数，设置颜色按钮中显示的颜色（加粗代码行）

```
void TextEdit::currentCharFormatChanged(const QTextCharFormat &format)
{
    fontChanged(format.font());
    colorChanged(format.foreground().color());
}
```

运行程序，效果如下



小结

本篇实现了文本的加粗、倾斜、下画线和文本对齐与颜色设置操作，以及与这些操作对应的按钮可用（激活）状态设置。

相关阅读：

《【文本编辑器】一、界面设计》
《【文本编辑器】二、文件操作功能（上）》
《【文本编辑器】二、文件操作功能（下）》
《【文本编辑器】三、文本编辑功能》

《主要的窗体类和主窗体构成》