【文本编辑器】二、文件操作功能（上）

原创 Qt 学习 Qt 学习 2020-02-11

上一篇开发了文本编辑器界面，在这一篇以及接下来的几篇中，将对文本编辑器界面开发过程中涉及的所有动作集对应的功能项进行设计。本篇对文本编辑器的窗口标题设置、文件保存和关闭程序时的保存提醒功能进行讲解。此外，还包括过`QTextEdit`类、标准文件对话框和消息框的相关使用。

本篇目录

运行环境：

win 10 + Qt 5.12.5 + Qt Creator 4.10

1. 设置窗口标题

在《【文本编辑器】一、界面设计》中通过`setWindowTitle(tr("文本编辑器"));`设置窗口标题，不过这只是应用程序的名称。常见软件的标题条通常设置成"文件名 - 软件名"的形式，下面将按照这一标题形式设置文本编辑器的窗口标题。

在 "textedit.h" 文件中添加设置文本编辑器标题的函数和标题变量的声明，代码如下：

```
1  private:
2      void setCurrentFileName(const QString &fileName); // 设置当前文件标题
3      QString fileName; // 当前文件标题名
```

在源文件中添加函数`setCurrentFileName()`的实现

```
1  void TextEdit::setCurrentFileName(const QString &fileName)
2  {
3      this->fileName = fileName; // 当前文档名
4      textEdit->document()->setModified(false); // 默认文档未被修改
5
6      QString shownName; // 标题条显示的文档名
7      if (fileName.isEmpty())
8          shownName = "未命名.txt";
9      else
10         shownName = QFileInfo(fileName).fileName();
11     // 设置窗口标题，当文档被修改后使用星号 '*' 标记
12     setWindowTitle(tr("%1[*] - %2").arg(shownName, tr("文本编辑器")));
13     setWindowModified(false); // 默认标题中不显示星号*
14 }
```

上面的代码中，`textEdit->document()->setModified()`用于设置文档是否修改过，一般用于关闭时提示保存。`QFileInfo(fileName).fileName()`用于从文件路径中提取文件名，这里使用了`QFileInfo`类，需要添加头文件

```
1  #include <QFileInfo>
```

在构造函数中添加 `setCurrentFileName(QString())` 函数，创建窗口初始标题

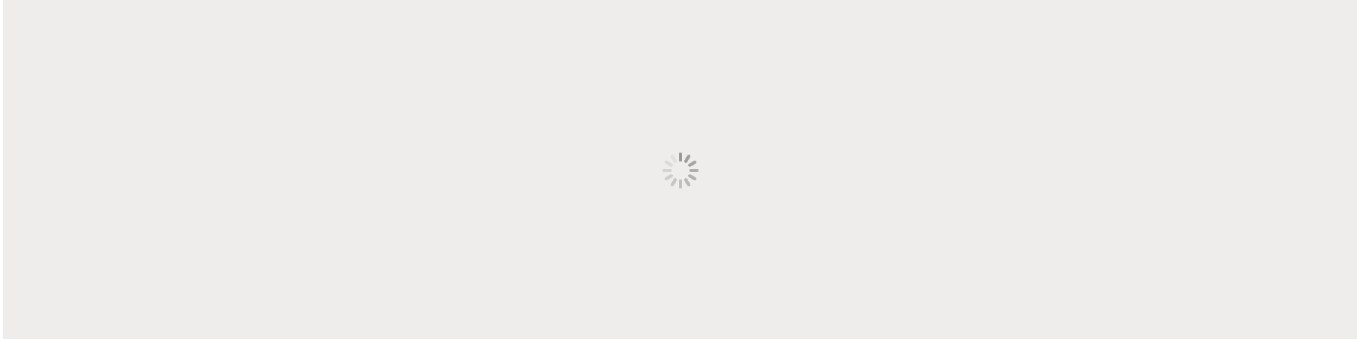
```
1   setCurrentFileName(QString());
```

运行程序后的效果如下

此时在文本编辑器中随意输入一些文字，窗口标题并没有如期出现星号 '*' 标记，在构造函数中再加上如下代码

```
1   connect(textEdit->document(), &QTextDocument::modificationChanged,
2           this, &QWidget::setWindowModified);
```

用来连接文本编辑窗口和窗口标题，文本编辑窗口的内容被更改时，使用 `QTextEdit` 类的 `document()` 函数获取 `QTextDocument` 类对象，发出文档被修改信号函数 `modificationChanged()` 。然后使用 `setWindowModified()` 函数设置窗口的更改状态标志（星号‘*’），如果参数为 `true` ，则在标题中设置了星号 '[*]' 标志的地方显示 '*' 号，表示该文件已被修改。运行程序后效果如下



2. 保存文件

保存文件功能分为『保存』或『另存为』两种操作，在 "textedit.h" 文件中添加函数声明：

```
1  private slots:
2      bool fileSave();
3      bool fileSaveAs();
```

在对应的源文件中添加头文件

```
1  #include <QFileDialog>
2  #include <QTextDocumentWriter>
```

文件保存的逻辑是先通过文件名判断文件是否符合保存的条件，符合则将文件保存至原路径下，否则则执行『另存为』操作。下面是保存函数的代码：

```
1  bool TextEdit::fileSave()
2  {
3      // 若未命名文件名或文件名者以 ':/' 开头，则直接执行另存为操作
4      if (fileName.isEmpty())
```

```
        return fileSaveAs();
    if (fileName.startsWith(QStringLiteral(":/")))
        return fileSaveAs();

    QTextDocumentWriter writer(fileName);
    bool success = writer.write(textEdit->document());
    // 在状态栏显示是否保存成功的提示信息
    if (success) {
        textEdit->document()->setModified(false);
        statusBar()->showMessage(tr("已写入 \"%1\"")
                                 .arg(QDir::toNativeSeparators(fileName)));
    } else {
        statusBar()->showMessage(tr("未能写入 \"%1\"")
                                 .arg(QDir::toNativeSeparators(fileName)));
    }
    return success;
}
```

代码中 `QTextDocumentWriter` 类提供了与格式无关的接口，用于将 `QTextDocument` 写入文件或其他设备。`QTextDocumentWriter writer(fileName);` 构造一个 `QTextDocumentWriter` 对象，该对象将使用指定的文档格式写入名称为 `fileName` 的文件。如果未提供格式，则 `QTextDocumentWriter` 将通过检查 `fileName` 的扩展名来检测文档格式。

另存为函数的实现流程是：新建保存形式的标准文件对话框；设置保存时默认的文件名后缀；将选择的另存为路径名赋给窗口标题设置函数。另存为函数的代码如下：

```
bool TextEdit::fileSaveAs()
{
    QFileDialog fileDialog(this, tr("另存为..."));  // 新建标准文件对话框
    fileDialog.setAcceptMode(QFileDialog::AcceptSave);
    QStringList mimeTypes;
    mimeTypes << "application/vnd.oasis.opendocument.text" << "text/html" << "text/plain";
    fileDialog.setMimeTypeFilters(mimeTypes); // 从 MIME 类型列表中设置文件对话框中使用的过滤器
    fileDialog.setDefaultSuffix("odt");       // 如果未指定其他后缀，则将此后缀添加到文件名中
    if (fileDialog.exec() != QDialog::Accepted)
        return false;
```

```
        const QString fn = fileDialog.selectedFiles().first(); // 包含对话框中所选文件的绝对路径的第一个字符串
        setCurrentFileName(fn);
        return fileSave();
    }
```

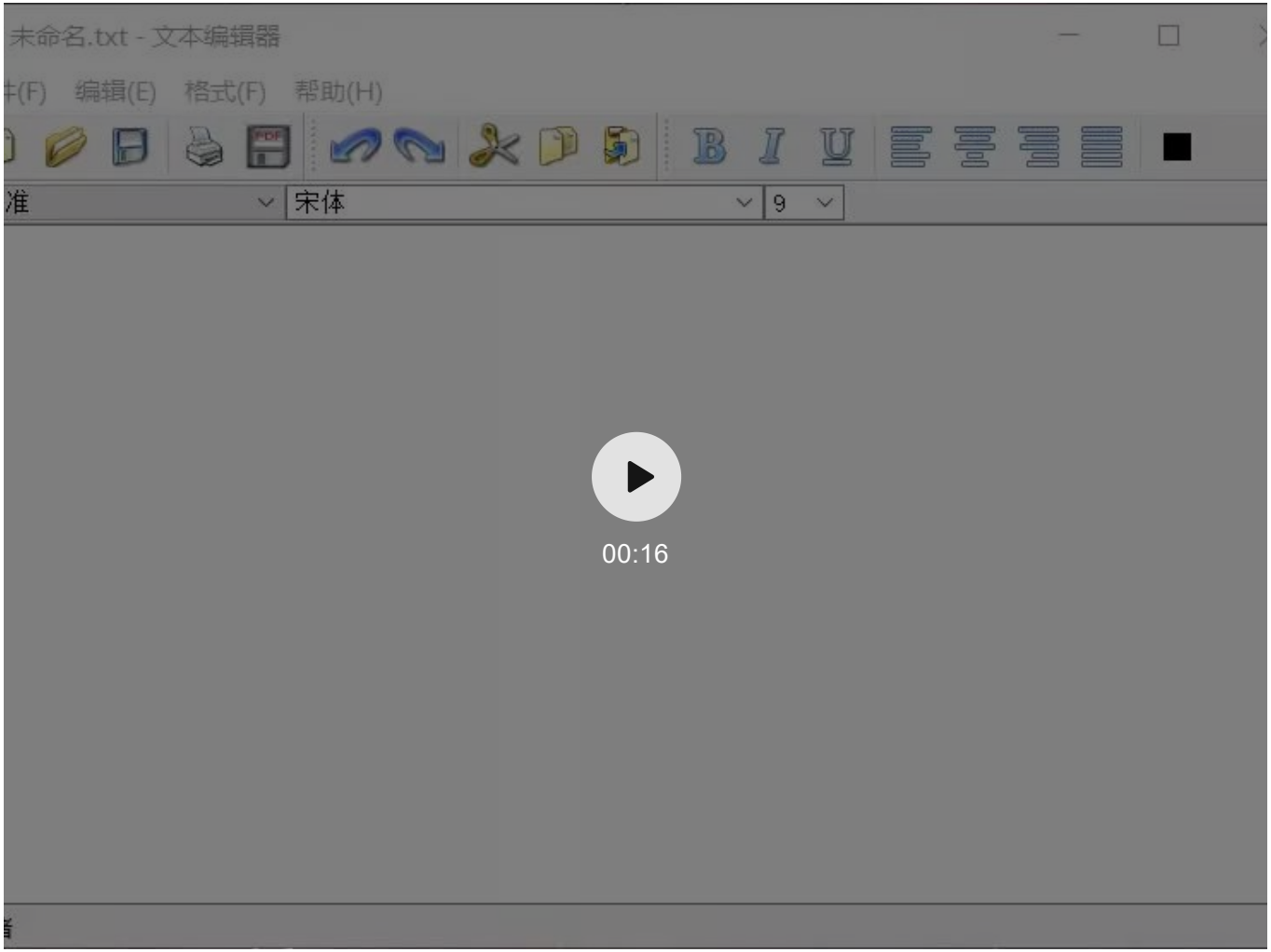
函数 `fileDialog.setAcceptMode()` 设置对话框模式，主要有两种

Constant	Value
QFileDialog::AcceptOpen	0
QFileDialog::AcceptSave	1

分别表示打开对话框和保存对话框，默认为 `AcceptOpen` 模式。取消 `setupFileActions()` 函数中对『保存』和『另存为』动作的连接函数 `connect()` 的注释

```
void TextEdit::setupFileActions()
{
    // 文件主菜单动作集
    ...
    connect(actionSave, &QAction::triggered, this, &TextEdit::fileSave);
    ...
    connect(actionSaveAs, &QAction::triggered, this, &TextEdit::fileSaveAs);
    ...
}
```

运行程序，点击『保存』或『另存为』或使用快捷键 "Ctrl+S"，效果如下



3. 关闭程序时的保存提醒

为了在关闭或退出应用程序时主动提醒用户保存，还需要设计“关闭”事件的关闭逻辑。这里通过一个 `maybeSave()` 函数来实现，在 "textedit.h" 文件中

```
private:
    bool maybeSave();
```

函数 `maybeSave()` 先判定文档是否被修改过，若未被修改过则直接退出，否则弹出保存提示对话框，再根据用户在对话框中的选择判断是否保存文件。函数的实现代码如下：

```
bool TextEdit::maybeSave()
{
    if (!textEdit->document()->isModified())
        return true;
    // 保存提醒消息框
    const QMessageBox::StandardButton ret =
        QMessageBox::warning(this, tr("文本编辑器"),
                             tr("文档已被修改，是否将其保存？"),
                             QMessageBox::Save | QMessageBox::Discard | QMessageBox::Cancel);
    // 根据对话框中的选择判断是否保存文件
    if (ret == QMessageBox::Save)
        return fileSave();
    else if (ret == QMessageBox::Cancel)
        return false;
    return true;
}
```

这里还需要添加相关的头文件

```
#include <QCloseEvent>
#include <QMessageBox>
```

在函数 `maybeSave()` 基础上就可以设计“关闭”事件，若 `maybeSave()` 函数返回值为真，则关闭文本编辑器，否则就忽略该事件。在 "textedit.h" 文件中声明关闭函数：

```
protected:
    void closeEvent(QCloseEvent *e) override;
```

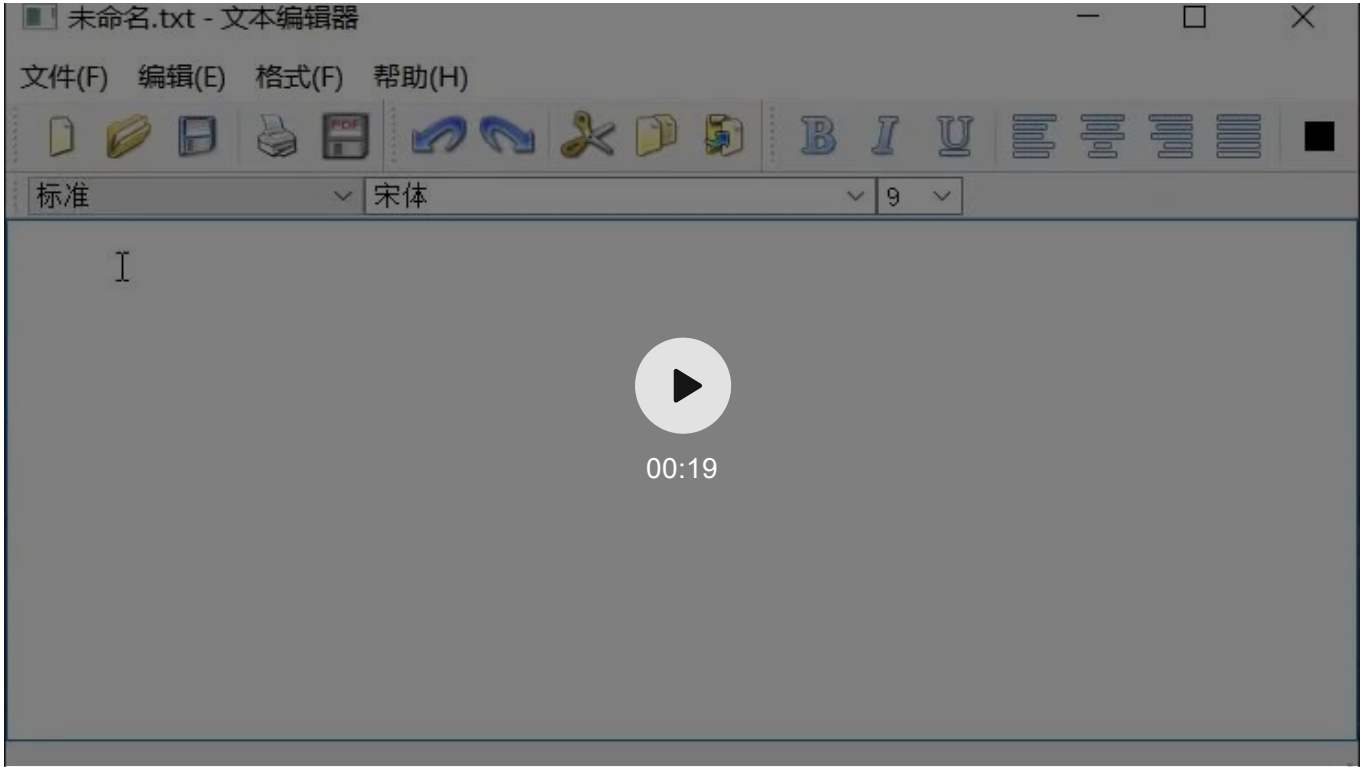
实现代码如下：

```
void TextEdit::closeEvent(QCloseEvent *e)
{
    if (maybeSave())
        e->accept();
    else
        e->ignore();
}
```

此外，将“文件”菜单中退出动作通过连接函数 `connect()` 关联到窗口关闭槽函数上，代码如下

```
connect(actionQuit, &QAction::triggered, this, &QWidget::close);
```

运行程序，在文本编辑器中输入任意内容，然后点击右上角的关闭或者依次单击『文件』『退出』，或者使用快捷键 "Ctrl+Q"，将会提示关闭提醒消息框，效果如下



小结

本篇介绍了文本编辑器的窗口标题设置、文件保存（另存为）和关闭/退出程序时的保存提醒功能，

相关阅读：

《【文本编辑器】一、界面设计》

《主要的窗体类和主窗体构成》

《Qt 模块简介》

《信号与槽》

《代码化 UI 设计》