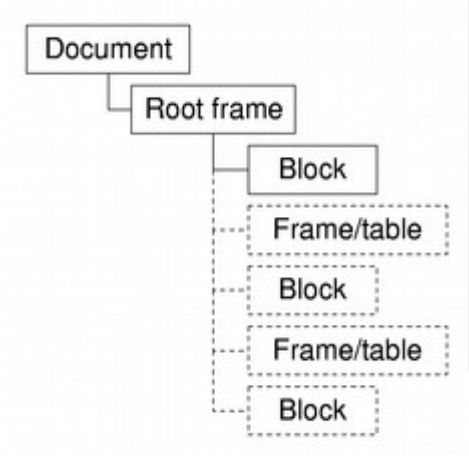
文本文件由 QTextDocument 类表示，它包含了文档的内部表示，它的结构信息，并保持修改跟踪，提供内置 undo/redo 操作支持。

文本文档的结构化表示呈现其内容为文本块，帧，表格和其他对象的层次结构。这些提供的逻辑结构的文件和描述它们的内容将如何显示。通常框架和表格用于组织其他结构，而文本块则包含真正的文本信息。

新元素的创建和插入可以通过使用 QTextCursor 以编程的方式实现，或者通过 QTextEdit 以用户可视化编辑的方式实现。元素可以在创建时指定一个特定的样式，或者是直接使用当前光标所在位置的样式。



基本结构

文档的“顶层”，决定显示的方式布局。每个文档总是包含一个根框架，而根框架总是包含至少一个文本块。

对于一些文字内容的文档，根框架通常包含块等元素的序列。

框架和表格的序列总是由文本块中的文件分开，即使文本块不包含任何信息。这确保了新的元素可以始终现有结构之间插入。

在本章中，我们来看看每一个富文本文档中使用的结构元素，勾勒出自己的特性和使用方法，并展示了如何检查它们的内容。文档编辑在 The QTextCursor Interface 文档中描述。

富文本文档

QTextDocument 对象包含所有构建丰富的文本文档所需的信息。文本文件可以用两种互补的方式进行访问：方便编辑器使用的线性缓存(linear buffer)和方便布局引擎使用的对象层次(object hierarchy)。在层次化文档模型中，对象用来描述可视元素，如框架、表格和列表。在更低的层次上，这些元素都有自己的描述属性，如文本风格和对齐方式。文档的线性描述则用于编辑和维护文档内容。

虽然 QTextEdit 提供了方便的富文本显示和编辑的功能，但文档也可以脱离任何一种编辑组件独立使用。例如：

```
QTextDocument *newDocument = new QTextDocument;
```

另外，也可以通过已有的文本组件获得：

```
QTextEdit *editor = new QTextEdit;
QTextDocument *editorDocument = editor->document();
```

这种灵活性使得应用程序能够同时操作多个文档，而不必包含多个文档组件，也不比要求文档必须存储为某种中间格式。

一个空的文档包含一个根框架，这个框架包含一个空的文本块。框架提供不同文档部分

的逻辑分割，同时也提供了在渲染时如何显示的属性。一个表格就是一个特化的框架，包含分布在不同行和列的多个单元。每个单元都能够包含更多的结构和文本。表格提供了灵活配置单元的管理和布局的特性。

文本块包含文本片段。每一个文本片段都有特定的文本和字符格式信息。文本属性在字符级别和块级别定义。在字符级别可以指定字体、颜色和大小。在块级别可以指定更高一级的行为，例如文本流方向、对齐方式和背景色。

文档结构并不是直接维护的，而是需要通过基于光标的接口进行编辑。文档光标接口会自动向根框架插入新的文档元素，并且确保这个元素在必要时有一个适合的空块。

我们可以通过以下方式访问到根框架：

```
QTextDocument *textDocument;
QTextFrame *root = textDocument->rootFrame();
```

当需要进行文档结构导航时，有时候可以从根框架开始。因为根框架提供了访问整个文档结构的能力。

文档元素

富文本文档通常包括一些通用的元素，例如段落、框架、表格和列表。这些在 QTextDocument 类中分别使用 QTextBlock，QTextFrame，QTextTable 和 QTextList 描述。不同于文档的其他元素，图片使用一种特殊的文本片段描述，这使得图片可以同普通文本混排。

文档的基本构建单位是 QTextBlock 和 QTextFrame。块本身就包含文本片段（QTextFragment），但是这不会直接影响到高层次的文档结构。

能够对其他文档元素分组的是 QTextObject 的子类。这些元素被分为两种类型：对文本块分组的是 QTextBlockGroup 的子类，对文本片段和其他元素分组的是 QTextFrame 的子类。

文本块

文本块由 QTextBlock 类提供。

文本块可以将具有不同字符样式的文本分组，用于表示文档段落。典型的文本块具有若干个不同样式的文本片段。当文本插入文档时，文本块被创建。在对文档进行编辑时，会增加更多的文本块。在块中，文档通过分割、合并、删除片段，有效地表现不同样式的文本。

一个文本块中的片段可以通过 QTextBlock::iterator 遍历：

```
QTextBlock::iterator it;
for (it = currentBlock.begin(); !(it.atEnd()); ++it) {
    QTextFragment currentFragment = it.fragment();
    if (currentFragment.isValid())
        processFragment(currentFragment);
}
```

文本块也可以用来表现列表项。因此，块能够在它们自己的字符样式包含块级样式信息，

例如列表项的符号等。块自己的样式由 QTextBlockFormat 描述，包括文本对齐方式，缩进和背景色。

虽然一个给定的文档可能包含复杂的结构，但是只要我们有一个文档中的文本块，我们就可以通过这个文本块对文档中所有文本块以编写顺序进行导航：

```
QTextBlock currentBlock = textDocument->begin();

while (currentBlock.isValid()) {
    processBlock(currentBlock);
    currentBlock = currentBlock.next();
}
```

当你需要导出文档中所有富文本内容时，这个方法就十分有用。因为它会忽略框架、表格以及其他文档结构。

QTextBlock 提供了比较运算符使操作更为简便：operator==() 和 operator!=() 用于比较两个块是否相同；operator<() 用于判断哪一个块在文档中的出现顺序靠前。

框架

框架由 QTextFrame 类提供。

文本框架用于组织文本块和子框架。这是一种比段落更大一级的文档结构。框架的格式决定它如何被显示和在页面中的位置。框架不是被插入文本流，就是浮在页面的左侧或者右侧。每一个文档都有一个根框架，包含了文档的所有结构。因此，除根框架之外，所有框架都有父框架。

既然文本块用于分割文档结构，那么，每一个框架都将至少包含一个文本块，零个或者多个子框架。我们可以通过 QTextFrame::iterator 来遍历所有子元素：

```
QDomElement frameElement = ...

QTextFrame::iterator it;
for (it = frame->begin(); !(it.atEnd()); ++it) {

    QTextFrame *childFrame = it.currentFrame();
    QTextBlock childBlock = it.currentBlock();

    if (childFrame)
        processFrame(frameElement, childFrame);
    else if (childBlock.isValid())
        processBlock(frameElement, childBlock);
}
```

注意，上面的代码，遍历器同时选出了文本块和框架，因此需要判断究竟是哪种元素。这能够让我们一个框架一个框架地导航文档，同时在需要的时候能够很方便地取出文本块。

QTextBlock::iterator 和 QTextFrame::iterator 两个类能够互补地取出文档中所需的结构。

表

表由 QTextTable 类提供。

表格是一个分布在行和列的单元的集合。每一个表格单元都是一个文档元素，拥有自己的字符样式，能够包含其他元素，例如框架和文本块。在表格构建，或者增加行或者列时，表格单元被自动创建。表格单元也可以在两个表格之间移动。

QTextTable 是 QTextFrame 的子类，因此表格在文档中被作为框架处理。如果我们需要处理文档中的每一个框架，我们需要对它们进行区分，从而分别处理：

```
QDomElement frameElement = ...

QTextFrame::iterator it;
for (it = frame->begin(); !(it.atEnd()); ++it) {

    QTextFrame *childFrame = it.currentFrame();
    QTextBlock childBlock = it.currentBlock();

    if (childFrame) {
        QTextTable *childTable = qobject_cast<QTextTable*>(childFrame);

        if (childTable)
            processTable(frameElement, childTable);
        else
            processFrame(frameElement, childFrame);

    } else if (childBlock.isValid())
        processBlock(frameElement, childBlock);
}
```

对于表格中已存在的单元，可以通过行和列进行遍历：

```
for (int row = 0; row < table->rows(); ++row) {
    for (int column = 0; column < table->columns(); ++column) {
        QTextTableCell tableCell = table->cellAt(row, column);
        processTableCell(tableCell);
    }
}
```

列表

列表由 QTextList 类提供。

列表是一系列按照一般方法格式化的文本块，同时有一个列表的修饰，例如一个点和列表项。列表可以嵌套。如果列表格式指定了非零缩进，列表就会有一定的缩进。

我们可以通过列表索引指定每一个列表项：

```
for (int index = 0; index < list->count(); ++index) {
```

```
    QTextBlock listItem = list->item(index);
    processListItem(listItem);
}
```

QTextList 实际上是 QTextBlockGroup 的子类，因此，它并不是将列表项当做子元素，而是提供另外的方法管理它们。这意味着，当我们遍历文档的所有文本块时，有可能它是一个列表中的一项。我们应当使用下面的代码进行区分

```
QTextFrame::iterator it;
for (it = frame->begin(); !(it.atEnd()); ++it) {

    QTextBlock block = it.currentBlock();

    if (block.isValid()) {

        QTextList *list = block.textList();

        if (list) {
            int index = list->itemNumber(block);
            processListItem(list, index);
        }
    }
}
```

图片

在 QTextDocument 中，图像当做一个文本片段，这个文本通过 Qt 资源机制指向图片的外部链接。图像通过光标接口创建，通过改变图像文本片段的样式进行修改：

```
if (fragment.isValid()) {
    QTextImageFormat newImageFormat = fragment.charFormat().toImageFormat();

    if (newImageFormat.isValid()) {
        newImageFormat.setName(":/images/newimage.png");
        QTextCursor helper = cursor;

        helper.setPosition(fragment.position());
        helper.setPosition(fragment.position() + fragment.length(),
                           QTextCursor::KeepAnchor);
        helper.setCharFormat(newImageFormat);
    }
}
```

表示图像的片段可以通过遍历包含图像的文本块的所有片段获得。