源码下载：https://github.com/QtExcel/QXlsx

1.加载QXlsx库

（1）将QXlsx源码文件夹拷贝到创建的Qt工程路径下（也可以是其它路径）

（2）在工程的.pro文件中添加下列代码（注意QXlsx文件夹的路径改成自己的）

```
include($$PWD/QXlsx/QXlsx.pri)              # QXlsx源代码
INCLUDEPATH += $$PWD/QXlsx
```

2.QXlsx库函数介绍

```
Document(const QString& xlsxName, QObject* parent = nullptr);
Document(QIODevice* device, QObject* parent = nullptr);

 //写数据
bool write(const CellReference &cell, const QVariant &value, const Format &format=Format());
bool write(int row, int col, const QVariant &value, const Format &format=Format());

 //读数据
QVariant read(const CellReference &cell) const;
QVariant read(int row, int col) const;

 //插入图片
int insertImage(int row, int col, const QImage &image);

 //获取图片
bool getImage(int imageIndex, QImage& img);
bool getImage(int row, int col, QImage& img);

 //获取图片数量
uint getImageCount();

 //插入图表
Chart *insertChart(int row, int col, const QSize &size);

```

```
 //合并单元格
bool mergeCells(const CellRange &range, const Format &format=Format());

 //取消合并单元格
bool unmergeCells(const CellRange &range);

 //设置单元格格式(列)
bool setColumnWidth(const CellRange &range, double width);
bool setColumnFormat(const CellRange &range, const Format &format);
bool setColumnHidden(const CellRange &range, bool hidden);
bool setColumnWidth(int column, double width);
bool setColumnFormat(int column, const Format &format);
bool setColumnHidden(int column, bool hidden);
bool setColumnWidth(int colFirst, int colLast, double width);
bool setColumnFormat(int colFirst, int colLast, const Format &format);
bool setColumnHidden(int colFirst, int colLast, bool hidden);

 //获取单元格格式(列)
double columnWidth(int column);
Format columnFormat(int column);
bool isColumnHidden(int column);

 //设置单元格格式(行)
bool setRowHeight(int row, double height);
bool setRowFormat(int row, const Format &format);
bool setRowHidden(int row, bool hidden);
bool setRowHeight(int rowFirst, int rowLast, double height);
bool setRowFormat(int rowFirst, int rowLast, const Format &format);
bool setRowHidden(int rowFirst, int rowLast, bool hidden);

 //获取单元格格式(行)
double rowHeight(int row);
Format rowFormat(int row);
bool isRowHidden(int row);

bool groupRows(int rowFirst, int rowLast, bool collapsed = true);
bool groupColumns(int colFirst, int colLast, bool collapsed = true);

bool addDataValidation(const DataValidation &validation);
```

```
64  bool addConditionalFormatting(const ConditionalFormatting &cf);
65
66  Cell *cellAt(const CellReference &cell) const;
67  Cell *cellAt(int row, int col) const;
68
69  bool defineName(const QString &name, const QString &formula,
70                  const QString &comment=QString(), const QString &scope=QString());
71
72  CellRange dimension() const;
73
74  QString documentProperty(const QString &name) const;
75  void setDocumentProperty(const QString &name, const QString &property);
76  QStringList documentPropertyNames() const;
77
78   //获取所有的工作表名称
79  QStringList sheetNames() const;
80
81   //添加工作表
82  bool addSheet(const QString &name = QString(),
    AbstractSheet::SheetType type = AbstractSheet::ST_WorkSheet);
83
84  //插入工作表
85  bool insertSheet(int index, const QString &name = QString(),
     AbstractSheet::SheetType type = AbstractSheet::ST_WorkSheet);
86
87  //选择工作表
88  bool selectSheet(const QString &name);
89  bool selectSheet(int index);
90
91   //重命名工作表
92  bool renameSheet(const QString &oldName, const QString &newName);
93
94   //拷贝工作表
95  bool copySheet(const QString &srcName, const QString &distName = QString());
96
97   //移动工作表
98  bool moveSheet(const QString &srcName, int distIndex);
99
100  //删除工作表
```

```
bool deleteSheet(const QString &name);

Workbook *workbook() const;
AbstractSheet *sheet(const QString &sheetName) const;
AbstractSheet *currentSheet() const;
Worksheet *currentWorksheet() const;

 //保存工作表
bool save() const;
bool saveAs(const QString &xlsXname) const;
bool saveAs(QIODevice *device) const;

// 复制一个xlsx文件的样式到另一个上
static bool copyStyle(const QString &from, const QString &to);

bool isLoadPackage() const;

 //加载xlsx文件
bool load() const;

 //改变图片
bool changeimage(int filenoinmidea,QString newfile);

 //设置单元格为自适应大小
bool autosizeColumnWidth(const CellRange &range);
bool autosizeColumnWidth(int column);
bool autosizeColumnWidth(int colFirst, int colLast);
bool autosizeColumnWidth(void);
```

3.创建excel文件

```
    Document xlsx;
    if(!xlsx.saveAs("1.xlsx")){
        qDebug()<<"创建失败";
    }
```

4.打开excel文件

```
    Document *m_xlsx;
    m_xlsx = new Document("1.xlsx",this);
    if(!m_xlsx->load()){
        qDebug()<<"打开失败";
    }
```

5.关闭excel文件

```
    delete m_xlsx;
    m_xlsx = nullptr;
```

6.写入excel

```
    m_xlsx->write("A1","1");
    m_xlsx->write("B1","2");
    m_xlsx->write("C1","3");
    m_xlsx->write("D1","4");

    m_xlsx->write(1,2,"xmr1");
    m_xlsx->write(2,2,"xmr1");
    m_xlsx->write(3,2,"xmr1");
    m_xlsx->write(4,2,"xmr1");

    m_xlsx->save();
```

7.读取excel

```
    int row = m_xlsx->dimension().rowCount();
    int col = m_xlsx->dimension().columnCount();

    for(int i = 0;i < row;i++){
```

```
        for(int j = 0;j < col;j++){
            qDebug()<<m_xlsx->read(i,j)<<m_xlsx->read(QString("%1%2").arg(char(64+i)).arg(j));
        }
    }

    //另存为
    m_xlsx->saveAs("2.excel");
```

8.查询所有的工作表

```
    QStringList list = m_xlsx->sheetNames();
```

9.在指定位置插入工作表，可设置工作表名称和类型

```
    //enum SheetType { ST_WorkSheet, ST_ChartSheet, ST_DialogSheet, ST_MacroSheet };
    QString sheet_name = "sheet2";
    AbstractSheet::SheetType type = AbstractSheet::ST_WorkSheet;
    int index = 1;
    m_xlsx->insertSheet(index,sheet_name,type);
    m_xlsx->save();
```

10.重命名工作表

```
    m_xlsx->renameSheet("sheet2","xmr");
    m_xlsx->save();
```

11.将指定的strName工作表拷贝为strNewName，如果strNewName已存在则拷贝失败

```
    m_xlsx->copySheet("strName","strNewName");
    m_xlsx->save();
```

12.移动工作表

根据输入的工作表名称，将工作表移动到指定位置，如果工作表不存在或移动到当前位置则失败，移动位置从0开始，如果大于sheet总数则移动到最后位置，如果小于0则移动到最开始位置

```
    m_xlsx->moveSheet("xmr2",0);
    m_xlsx->save();
```

13.删除工作表

```
m_xlsx->deleteSheet("xmr");
m_xlsx->save();
```

14.查询工作表隐藏或可见状态

```
//enum SheetState { SS_Visible,SS_Hidden, SS_VeryHidden };

Document xlsx(EXCEL_NAME);
AbstractSheet* sheet = xlsx.sheet(sheet_name);   // 根据名称获取工作表指针
int state = sheet->sheetState();
```

15.设置工作表隐藏或可见状态

```
//enum SheetState { SS_Visible,SS_Hidden, SS_VeryHidden };

Document xlsx(EXCEL_NAME);
AbstractSheet* sheet = xlsx.sheet(sheet_name);   // 根据名称获取工作表指针
sheet->setSheetState(AbstractSheet::SheetState(index));             // 修改工作表状态
xlsx.saveAs(EXCEL_NAME);
```

16.工作表中插入图表

演示在工作表中插入图表，这里演示了Qxlsx中所有图表类型；从源码中 void ChartPrivate::saveXmlChart(QXmlStreamWriter &writer) const函数看，部分ChartType类型还不支持，如CT_StockChart

```
    Document xlsx;                      // 创建一个Excel对象（默认有一个Sheet1）
    for(int i = 1; i < 10; i++)
    {
        xlsx.write(i, 1, i * i * i);     // 在第一列写入9个数据
        xlsx.write(i, 2, i * i);         // 在第二列写入9个数据
        xlsx.write(i, 3, i *i - 1);      // 写入第三列数据
    }
    qDebug() << CellReference(1, 2).toString();                    // 可将行列号转换为【字符串】单元格引用

    // 插入面积图
    xlsx.write(3, 4, "CT_AreaChart");  // 在图标左上角单元格中写入图标的类型
    Chart* areaChart = xlsx.insertChart(3, 3, QSize(300, 300));
    areaChart->setChartType(Chart::CT_AreaChart);
    areaChart->addSeries(CellRange("A1:C9"));

    // 插入3D面积图（在WPS中显示存在问题，office没有测试）
    xlsx.write(3, 10, "CT_Area3DChart");
    Chart *area3DChart = xlsx.insertChart(3, 9, QSize(300, 300));
    area3DChart->setChartType(Chart::CT_Area3DChart);
    area3DChart->addSeries(CellRange("A1:C9"));

    // 插入折线图
    xlsx.write(3, 16, "CT_LineChart");
    Chart* lineChart = xlsx.insertChart(3, 15, QSize(300, 300));
    lineChart->setChartType(Chart::CT_LineChart);
    lineChart->addSeries(CellRange("A1:C9"));

    // 插入3D折线图
    xlsx.write(23, 4, "CT_Line3DChart");
    Chart* line3DChart = xlsx.insertChart(23, 3, QSize(300, 300));
    line3DChart->setChartType(Chart::CT_Line3DChart);
    line3DChart->addSeries(CellRange("A1:C9"));

```

```
    // 插入股价图（貌似还不支持）
    xlsx.write(23, 10, "CT_StockChart");
    Chart* stockChart = xlsx.insertChart(23, 9, QSize(300, 300));
    stockChart->setChartType(Chart::CT_StockChart);
    stockChart->addSeries(CellRange("A1:C9"));

    // 插入雷达图（貌似还不支持）
    xlsx.write(23, 16, "CT_RadarChart");
    Chart* radarChart = xlsx.insertChart(23, 15, QSize(300, 300));
    radarChart->setChartType(Chart::CT_RadarChart);
    radarChart->addSeries(CellRange("A1:A9"));

    // 插入散点图（在WPS中效果和CT_LineChart一样）
    xlsx.write(43, 4, "CT_ScatterChart");
    Chart* scatterChart = xlsx.insertChart(43, 3, QSize(300, 300));
    scatterChart->setChartType(Chart::CT_ScatterChart);
    scatterChart->addSeries(CellRange("A1:A9"));              // 插入三个数据系列
    scatterChart->addSeries(CellRange("B1:B9"));
    scatterChart->addSeries(CellRange("C1:C9"));
    // 散点图不能以A1:C9这种方式同时选择三列数据，在WPS中会默认把第一列数据当做X轴数据，QXlsx中会直接舍弃掉第一列数据，
    // 由addSeries函数中if (d->chartType == CT_ScatterChart || d->chartType == CT_BubbleChart)可看出
//    scatterChart->addSeries(CellRange("A1:C9"));

    // 插入饼图
    xlsx.write(43, 10, "CT_PieChart");
    Chart* pieChart = xlsx.insertChart(43, 9, QSize(300, 300));     // 在第三行、三列的单元格右下角位置插入一个长宽为300的图表
    pieChart->setChartType(Chart::CT_PieChart);                     // 指定图表类型为【饼图】（支持16种类型图表）
    pieChart->addSeries(CellRange("A1:A9"));                        // 添加饼图的数据系列1（单元格引用字符串方式指定【第一列数据】）
    pieChart->addSeries(CellRange(CellReference(1, 2), CellReference(9, 2))); // 添加饼图数据2（通过CellReference指定【第二列数据】）
    pieChart->addSeries(CellRange(1, 3, 9, 3));                     // 添加饼图数据系列3（通过[开始行列号]和[结束行列号]指定【第三列数据】）

    // 插入3D饼图（这个图表在WPS中样式和CT_PieChart一样，没有表现出3D效果，无法设置三维旋转）
    xlsx.write(43, 16, "CT_Pie3DChart");
    Chart* pie3DChart = xlsx.insertChart(43, 15, QSize(300, 300));
```

```
    pie3DChart->setChartType(Chart::CT_Pie3DChart);
    pie3DChart->addSeries(CellRange("A1:A9"));

    // 插入圆环图
    xlsx.write(63, 4, "CT_DoughnutChart");
    Chart* doughnutChart = xlsx.insertChart(63, 3, QSize(300, 300));
    doughnutChart->setChartType(Chart::CT_DoughnutChart);
    doughnutChart->addSeries(CellRange("A1:A9"));

    // 插入柱状图
    xlsx.write(63, 10, "CT_BarChart");
    Chart* barChart = xlsx.insertChart(63, 9, QSize(300, 300));
    barChart->setChartType(Chart::CT_BarChart);
    barChart->addSeries(CellRange("A1:A9"));

    // 插入3D柱状图（在WPS中显示异常，不支持3D柱状图）
    xlsx.write(63, 16, "CT_Bar3DChart");
    Chart* bar3DChart = xlsx.insertChart(63, 15, QSize(300, 300));
    bar3DChart->setChartType(Chart::CT_Bar3DChart);
    bar3DChart->addSeries(CellRange("A1:A9"));

    // 插入饼图（还不支持）
    xlsx.write(83, 4, "CT_OfPieChart");
    Chart* ofPieChart = xlsx.insertChart(83, 3, QSize(300, 300));
    ofPieChart->setChartType(Chart::CT_OfPieChart);
    ofPieChart->addSeries(CellRange("A1:A9"));

    // 插入曲面图（还不支持）
    xlsx.write(83, 10, "CT_SurfaceChart");
    Chart* surfaceChart = xlsx.insertChart(83, 9, QSize(300, 300));
    surfaceChart->setChartType(Chart::CT_SurfaceChart);
    surfaceChart->addSeries(CellRange("A1:A9"));

    // 插入3D曲面图（还不支持）
    xlsx.write(83, 16, "CT_Surface3DChart");
    Chart* surface3DChart = xlsx.insertChart(83, 15, QSize(300, 300));
    surface3DChart->setChartType(Chart::CT_Surface3DChart);
    surface3DChart->addSeries(CellRange("A1:A9"));

```

```
    // 插入气泡图（还不支持）
    xlsx.write(103, 4, "CT_BubbleChart");
    Chart* bubbleChart = xlsx.insertChart(103, 3, QSize(300, 300));
    bubbleChart->setChartType(Chart::CT_BubbleChart);
    bubbleChart->addSeries(CellRange("A1:A9"));

    xlsx.saveAs(EXCEL_NAME);
```

17.插入图表Sheet，并绘制一个柱状图

```
    Document xlsx;
    for(int i = 1; i < 10; i++)
    {
        xlsx.write(i, 1, i * i);          // 在Sheet1中写入1列数据
    }

    xlsx.addSheet("Chart1", AbstractSheet::ST_ChartSheet);                // 插入一个名称为【Chart1】，类型为【图表】的Sheet
    Chartsheet* sheet = static_cast<Chartsheet*>(xlsx.currentSheet());    // 获取当前工作表，并将类型转换为Chartsheet*
    Chart* barChart = sheet->chart();            // 图表Sheet中会默认内置一个Chart，从这一步开始就和正常操作图表一样了
    barChart->setChartType(Chart::CT_BarChart); // 设置图表类型位柱状图
    barChart->addSeries(CellRange("A1:A9"), xlsx.sheet("Sheet1")); // 添加数据系列，数据位于Sheet1中的A1-A9

    xlsx.saveAs(EXCEL_NAME);
```

18.设置图表样式

1. 设置【图例】位置；
2. 设置图表【标题】；
3. 打开图表网格线；
4. 行列交换标头；
5. 设置插入的数据范围是否包含标题；
6. 插入图表，引用其它工作表数据。

```
    Document xlsx;

    for(int i = 1; i < 10; i++)
    {
        xlsx.write(1, i + 1, QString("Pos %1").arg(i));      // 写入列标题
        xlsx.write(2, i + 1, i * i * i);                     // 写入数据
        xlsx.write(3, i + 1, i * i);
    }
    // 写入行标题
    xlsx.write(2, 1, "Set 1");
    xlsx.write(3, 1, "Set 2");

    // 插入一个柱状图，并设置图例在【右边】
    xlsx.write(5, 4, "图例在右边");
    Chart* barChart1 = xlsx.insertChart(5, 3, QSize(300, 300));  // 插入图表
    barChart1->setChartType(Chart::CT_BarChart);
    barChart1->setChartLegend(Chart::Right);     // 设置图例在右边，可设置None：无图例, Left：左边, Right：右边，  Top：上边, Bottom：下边
    barChart1->setChartTitle("Test1");
    barChart1->addSeries(CellRange(1, 1, 3, 10), nullptr, true, true, false);

    // 插入一个柱状图，启动【主网格线】
    xlsx.write(5, 10, "图例在左边，启动主网格线");
    Chart* barChart2 = xlsx.insertChart(5, 9, QSize(300, 300));  // 插入图表
    barChart2->setChartType(Chart::CT_BarChart);
    barChart2->setChartLegend(Chart::Left);
    barChart2->setChartTitle("Test2");
    barChart2->setGridlinesEnable(true);   // 启动主网格线
    barChart2->addSeries(CellRange(1, 1, 3, 10), nullptr, true, true, false);

    // 插入一个柱状图，启动【次网格线】
    xlsx.write(5, 16, "图例在上边，启动次网格线");
    Chart* barChart3 = xlsx.insertChart(5, 15, QSize(300, 300));  // 插入图表
    barChart3->setChartType(Chart::CT_BarChart);
    barChart3->setChartLegend(Chart::Top);
    barChart3->setChartTitle("Test3");
    barChart3->setGridlinesEnable(false, true);  // 关闭主网格线，启动子网格线
    barChart3->addSeries(CellRange(1, 1, 3, 10), nullptr, true, true, false);
```

```
38
39      // 插入一个柱状图，【行列交换标头】
40      xlsx.write(25, 4, "图例在下边，行列交换标头");
41      Chart* barChart4 = xlsx.insertChart(25, 3, QSize(300, 300));  // 插入图表
42      barChart4->setChartType(Chart::CT_BarChart);
43      barChart4->setChartLegend(Chart::Bottom);
44      barChart4->setChartTitle("Test4");
45      barChart4->addSeries(CellRange(1, 1, 3, 10), nullptr, false, true, true);  // 参数5【true：以1列为1个数据系列，false：以1行为1个数据系列】
46
47      // 插入一个柱状图，【数据范围不包含标题】
48      xlsx.write(25, 10, "数据范围不包含标题");
49      Chart* barChart5 = xlsx.insertChart(25, 9, QSize(300, 300));  // 插入图表
50      barChart5->setChartType(Chart::CT_BarChart);
51      barChart5->setChartLegend(Chart::Right);
52      barChart5->setChartTitle("Test5");
53      // 参数1：添加数据系列范围；参数2：指定插入的数据位于哪个工作表（Sheet），默认为NULL，即当前工作表；
54      // 参数3，数据系列范围第一行是否为列标题，true：为标题；参数4，数据系列范围第1列是否为行标题，true：为标题；默认都不为标题
55      // 参数5：交换行列标头。
56      barChart5->addSeries(CellRange(1, 1, 3, 10));
57
58      // 插入一个柱状图，【数据范围包含列标题】
59      xlsx.write(25, 16, "数据范围包含列标题");
60      Chart* barChart6 = xlsx.insertChart(25, 15, QSize(300, 300));  // 插入图表
61      barChart6->setChartType(Chart::CT_BarChart);
62      barChart6->setChartLegend(Chart::Right);
63      barChart6->setChartTitle("Test6");
64      barChart6->addSeries(CellRange(1, 1, 3, 10), nullptr, true);
65
66      // 插入一个柱状图，【数据范围包含行标题】
67      xlsx.write(45, 4, "数据范围包含行标题");
68      Chart* barChart7 = xlsx.insertChart(45, 3, QSize(300, 300));  // 插入图表
69      barChart7->setChartType(Chart::CT_BarChart);
70      barChart7->setChartLegend(Chart::Right);
71      barChart7->setChartTitle("Test7");
72      barChart7->addSeries(CellRange(1, 1, 3, 10), nullptr, false, true);
73
74      // 添加一个工作表（Sheet2）,在Sheet2中插入图表，数据为Sheet1中的数据
```

```
    xlsx.addSheet("Sheet2");                                    // 添加一个工作表，当前工作表为Sheet2
    xlsx.write(3, 4, "插入图表，引用Sheet1数据");
    Chart* barChart8 = xlsx.insertChart(3, 3, QSize(300, 300));  // 插入图表
    barChart8->setChartType(Chart::CT_BarChart);
    barChart8->setChartLegend(Chart::Right);
    barChart8->setChartTitle("Test8");
    barChart8->addSeries(CellRange(1, 1, 3, 10), xlsx.sheet("Sheet1"));   // 添加数据系列范围，并指定为Sheet1中的数据

    xlsx.saveAs(EXCEL_NAME);                                    // 如果文件已经存在则覆盖
```

19.插入图片

```
    Document xlsx;
    QImage image1("://image/C++.PNG");
    QImage image2("://image/Qt.PNG");
    qDebug() << "插入图片："<<xlsx.insertImage(3, 3, image1);  // 在3行3列单元格右下角位置插入图片
    qDebug() << "插入图片："<<xlsx.insertImage(23, 3, image2);  // 在23行3列单元格右下角位置插入图片
    xlsx.saveAs(EXCEL_NAME);
```

20.打开Excel文件，并查询当前Sheet中图片数量

```
    Document xlsx(EXCEL_NAME);
    if(!xlsx.load())
    {
        QMessageBox::warning(this, "错误", QString("打开%1失败，可能是文件不存在！").arg(EXCEL_NAME));
        return;
    }

    uint count = xlsx.getImageCount();        // 查询当前Sheet中图片数量
    QMessageBox::about(this, "插入图片数", QString("共有%1张图片！").arg(count));
```

21.读取Excel中的图片（通过索引读取）

注意：这里索引从1开始，而不是从0开始（Qxlsx的一些小bug）

```
    Document xlsx(EXCEL_NAME);
    if(!xlsx.load())
    {
        QMessageBox::warning(this, "错误", QString("打开%1失败，可能是文件不存在！").arg(EXCEL_NAME));
        return;
    }

    QImage image;
    bool ret = xlsx.getImage(1, image);        // 读取当前Sheet中第1张图片(注意：索引是从1开始，而不是从0开始)
    if(ret)
    {
        ui->label->setPixmap(QPixmap::fromImage(image));    // 显示读取到的图片
    }
    else
    {
        QMessageBox::warning(this, "错误", "读取图片失败，可能是不存在！");
    }
```

22.读取Excel中的图片（通过行列号读取）

```
    Document xlsx(EXCEL_NAME);
    if(!xlsx.load())
    {
        QMessageBox::warning(this, "错误", QString("打开%1失败，可能是文件不存在！").arg(EXCEL_NAME));
        return;
    }

    QImage image;
    bool ret = xlsx.getImage(23, 3, image);        // 读取当前Sheet中第2张图片
    if(ret)
    {
```

```
        ui->label->setPixmap(QPixmap::fromImage(image));   // 显示读取到的图片
    }
    else
    {
        QMessageBox::warning(this, "错误", "读取图片失败，可能是不存在或位置错误！");
    }
```

23.设置单元格水平对齐

```
    Document xlsx(EXCEL_NAME);
    if(!xlsx.load())
    {
        QMessageBox::warning(this, "错误", QString("打开%1失败，可能是文件不存在！").arg(EXCEL_NAME));
        return;
    }

    Format format = xlsx.cellAt(8, 2)->format();                          // 获取单元格原有格式
    format.setHorizontalAlignment(Format::HorizontalAlignment(index));  // 设置单元格水平对齐格式
    xlsx.write(8, 2, xlsx.read(8, 2), format);                            // 将单元格原有内容、格式写入原来位置

    xlsx.saveAs(EXCEL_NAME);
```

24.设置单元格垂直对齐

```
    Document xlsx(EXCEL_NAME);
    if(!xlsx.load())
    {
        QMessageBox::warning(this, "错误", QString("打开%1失败，可能是文件不存在！").arg(EXCEL_NAME));
        return;
    }

```

```
    Format format = xlsx.cellAt(8, 2)->format();                    // 获取单元格原有格式
    format.setVerticalAlignment(Format::VerticalAlignment(index));  // 设置单元格垂直对齐格式
    xlsx.write(8, 2, xlsx.read(8, 2), format);                      // 将单元格原有内容、格式写入原来位置

    xlsx.saveAs(EXCEL_NAME);
```

25.合并单元格

```
    Document xlsx;

    // 在Excel中写入三组数据
    xlsx.write("B3", "hello");
    xlsx.write("C3", "123");
    xlsx.write("B8", 123456);
    xlsx.write("E8", "北风卷地白草折，胡天八月即飞雪");

    xlsx.mergeCells("B3:F6");                   // 可以通过【单元格引用】直接设置单元格合并(注意，有数据的单元格应该时第一个位置)
    xlsx.mergeCells(CellRange(8, 2, 20, 3));    // 通过【行列号】设置单元格合并

    Format format;
    format.setHorizontalAlignment(Format::AlignHCenter);  // 水平居中
    format.setVerticalAlignment(Format::AlignVCenter);    // 垂直居中
    xlsx.mergeCells("E8:F20", format);                    // 在设置单元格合并时可以设置单元格【格式】，如文本居中对齐
    xlsx.saveAs(EXCEL_NAME);
```

26.取消合并单元格

```
    Document xlsx(EXCEL_NAME);
    if(!xlsx.load())
    {
        QMessageBox::warning(this, "错误", QString("打开%1失败，可能是文件不存在！").arg(EXCEL_NAME));
```

```
        return;
    }

    xlsx.unmergeCells("B3:F6");    // 这里取消合并的范围【B3:F6】必须和之前合并单元格的【范围相同】，否则取消合并失败
    xlsx.saveAs(EXCEL_NAME);
```

27.设置字体样式

```
    Document xlsx;
    xlsx.write(1, 1, "默认样式");

    // 设置字体大小
    Format format;
    format.setFontSize(15);
    xlsx.write(1, 2, "字体大小15", format);
    qDebug() << xlsx.cellAt(1, 2)->format().fontSize();  // 获取当前单元格的字体大小

    // 设置字体斜体
    Format format1;
    format1.setFontItalic(true);
    xlsx.write(1, 3, "斜体", format1);
    qDebug() << xlsx.cellAt(1, 3)->format().fontItalic();  // 获取当前单元格的字体是否为斜体

    // 设置字体删除线
    Format format2;
    format2.setFontStrikeOut(true);
    xlsx.write(1, 4, "删除线", format2);
    qDebug() << xlsx.cellAt(1, 4)->format().fontStrikeOut();  // 获取当前单元格的字体是否有 删除线

    // 设置字体颜色
    Format format3;
    format3.setFontColor(Qt::red);
    xlsx.write(1, 5, "字体颜色", format3);
    qDebug() << xlsx.cellAt(1, 5)->format().fontColor();  // 获取当前单元格的字体颜色

```

```cpp
    // 设置字体加粗
    Format format4;
    format4.setFontBold(true);
    xlsx.write(1, 6, "字体加粗", format4);
    qDebug() << xlsx.cellAt(1, 6)->format().fontBold();  // 获取当前单元格的字体是否加粗

    // 设置字体特殊格式（上、下标）
    Format format5;
    format5.setFontScript(Format::FontScriptSub);   // 设置下标
    xlsx.write(1, 7, "字体下标", format5);
    format5.setFontScript(Format::FontScriptSuper); // 设置上标
    xlsx.write(1, 8, "字体上标", format5);
    qDebug() << xlsx.cellAt(1, 7)->format().fontScript();  // 获取当前单元格的字体的特殊格式

    // 设置下划线
    Format format6;
    format6.setFontUnderline(Format::FontUnderlineNone);
    xlsx.write(1, 9, "无下划线", format6);
    format6.setFontUnderline(Format::FontUnderlineSingle);
    xlsx.write(1, 10, "单下划线", format6);
    format6.setFontUnderline(Format::FontUnderlineDouble);
    xlsx.write(1, 11, "双下划线", format6);
    format6.setFontUnderline(Format::FontUnderlineSingleAccounting);
    xlsx.write(1, 12, "会计用单下划线", format6);
    format6.setFontUnderline(Format::FontUnderlineDoubleAccounting);
    xlsx.write(1, 13, "会计用双下划线", format6);
    qDebug() << xlsx.cellAt(1, 9)->format().fontUnderline();  // 获取当前单元格文本下划线格式

    // 设置字体轮廓
    Format format7;
    format7.setFontOutline(true);
    xlsx.write(1, 14, "字体轮廓", format7);
    qDebug() << xlsx.cellAt(1, 14)->format().fontOutline();  // 获取当前单元格是否打开字体轮廓

    // 设置字体类型
    Format format8;
    format8.setFontName("黑体");
```

```
65      xlsx.write(1, 15, "字体类型", format8);
66      qDebug() << xlsx.cellAt(1, 15)->format().fontName();  // 获取当前单元格字体类型
67
68      xlsx.saveAs(EXCEL_NAME);
```