

资料大全
2019年最新!

html5全集 JavaScript全集 WEB前端全集 Bootstrap全集 VUE全集
angular全集 node.js全集 实战项目全集 PHP项目实战全集 java全集
java项目全集 Python实战全集 Linux实战项目 500本电子书集合 2019年面试题库

大小: 7,152.32 GB

立即免费下载

Qt QStandardItemModel用法（超级详细）

< [上一页](#)[下一页](#) >

QStandardItemModel 是标准的以项数据（item data）为基础的标准数据模型类，通常与 QTableView 组合成 Model/View 结构，实现通用的二维数据的管理功能。

本节介绍 QStandardItemModel 的使用，主要用到以下 3 个类：

1. QStandardItemModel：基于项数据的标准数据模型，可以处理二维数据。维护一个二维的项数据数组，每个项是一个 QStandardItem 类的变量，用于存储项的数据、字体格式、对齐方式等。
2. QTableView：二维数据表视图组件，有多个行和多个列，每个基本显示单元是一个单元格，通过 setModel() 函数设置一个 QStandardItemModel 类的数据模型之后，一个单元格显示 QStandardItemModel 数据模型中的一个项。
3. QItemSelectionModel：一个用于跟踪视图组件的单元格选择状态的类，当在 QTableView 选择某个单元格，或多个单元格时，通过 QItemSelectionModel 可以获得选中的单元格的模型索引，为单元格的选择操作提供方便。

这几个类之间的关系是：QTableView 是界面视图组件，其关联的数据模型是 QStandardItemModel，关联的项选择模型是 QItemSelectionModel，QStandardItemModel 的数据管理的基本单元是 QStandardItem。

实例 samp5_3 演示 QStandardItemModel 的使用，其运行时界面如图 1 所示。

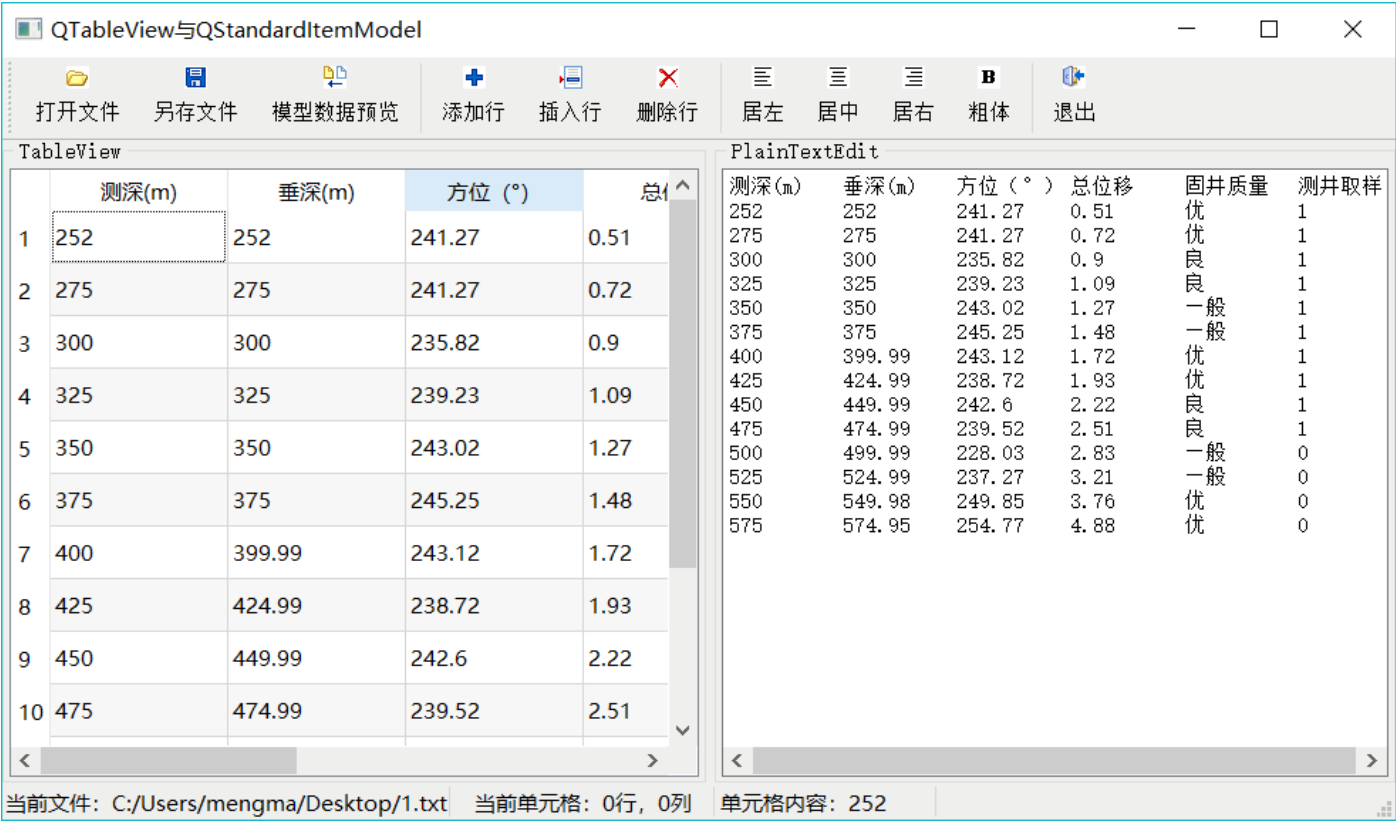


图 1 实例 samp5_3 的运行界面

- 该实例具有如下功能：
- 打开一个纯文本文件，该文件是规则的二维数据文件，通过字符串处理获取表头和各行各列的数据，导入到一个 QStandardItemModel 数据模型。
 - 编辑修改数据模型的数据，可以插入行、添加行、删除行，还可以在 QTableView 视图组件中直接修改单元格的数据内容。
 - 可以设置数据模型中某个项的不同角色的数据，包括文字对齐方式、字体是否粗体等。
 - 通过 QItemSelectionModel 获取视图组件上的当前单元格，以及选择单元格的范围，对选择的单元格进行操作。
 - 将数据模型的数据内容显示到 QPlainTextEdit 组件里，显示数据模型的内容，检验视图组件上做的修改是否与数据模型同步。
 - 将修改后的模型数据另存为一个文本文件。

界面设计与主窗口类定义

本实例的主窗口从 QMainWindow 继承而来，中间的 TableView 和 PlainTextEdit 组件采用水平分割条布局。在 Action 编辑器中创建如图 2 所示的一些 Action，并由 Action 创建主工具栏上的按钮，下方的状态栏设置了几个 QLabel 组件，显示当前文件名称、当前单元格行号、列号，以及相应内容。














名称	使用	文本	快捷键	可选的	工具提示
 actOpen	<input checked="" type="checkbox"/>	打开文件		<input type="checkbox"/>	打开文件
 actSave	<input checked="" type="checkbox"/>	另存文件		<input type="checkbox"/>	表格内容另存为文件
 actAppend	<input checked="" type="checkbox"/>	添加行		<input type="checkbox"/>	添加一行
 actInsert	<input checked="" type="checkbox"/>	插入行		<input type="checkbox"/>	插入一行
 actDelete	<input checked="" type="checkbox"/>	删除行		<input type="checkbox"/>	删除当前行
 actExit	<input checked="" type="checkbox"/>	退出		<input type="checkbox"/>	退出
 actMo...lData	<input checked="" type="checkbox"/>	模型数据预览		<input type="checkbox"/>	模型数据显示到文本框里
 actAlignLeft	<input checked="" type="checkbox"/>	居左		<input type="checkbox"/>	文字左对齐
 actAl...enter	<input checked="" type="checkbox"/>	居中		<input type="checkbox"/>	文字居中
 actAl...Right	<input checked="" type="checkbox"/>	居右		<input type="checkbox"/>	文字右对齐
 actFontBold	<input checked="" type="checkbox"/>	粗体		<input checked="" type="checkbox"/>	粗体字体

图 2 实例中创建的 Action

主窗口类 MainWindow 里新增的定义如下（省略了 UI 设计器生成的界面组件的槽函数的声明）：

```

01. #define FixedColumnCount 6    //文件固定 6 列
02. class MainWindow : public QMainWindow
03. {
04.     Q_OBJECT private:
05.     QLabel *LabCurFile;      //当前文件
06.     QLabel *LabCellPos;      //当前单元格行列号
07.     QLabel *LabCellText;    //当前单元格内容
08.     QStandardItemModel * theModel; //数据模型
09.     QItemSelectionModel *theSelection; //选择模型
10.     void iniModelFromStringList (QStringList&) ; //从 QStringList 初始化数据模型
11. public:
12.     explicit MainWindow(QWidget *parent = 0);
13. private slots:
14.     //当前选择单元格发生变化
15.     void on_currentChanged(const QModelIndex &current, const QModelIndex
        &previous);
16. private:
17.     Ui::MainWindow *ui;
18. };

```

这里定义了数据模型变量 theModel，项数据选择模型变量 theSelection。

定义的私有函数 iniModelFromStringList() 用于在打开文件时，从一个 QStringList 变量的内容创建数据模型。

自定义槽函数 on_currentChanged() 用于在 TableView 上选择单元格发生变化时，更新状态栏的信息显示，这个槽函数将会与项选择模型 theSelection 的 currentChanged() 信号关联。



QStandardItemModel的使用

系统初始化

在 MainWindow 的构造函数中进行界面初始化，数据模型和选择模型的创建，以及与视图组件的关联，信号与槽的关联等设置，代码如下：

```
01. MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent), ui (new
    Ui::MainWindow)
02. {
03.     ui->setupUi(this);
04.     setCentralWidget(ui->splitter);
05.     theModel = new QStandardItemModel (2, FixedColumnCount, this) ; //数据
    模型
06.     theSelection = new QItemSelectionModel (theModel) ;//选择模型
07.     connect(theSelection, SIGNAL(currentChanged(QModelIndex,QModelIndex)),
    this, SLOT(on_currentChanged(QModelIndex,QModelIndex)));
08.     ui->tableView->setModel (theModel) ; //设置数据模型
09.     ui->tableView->setSelectionModel(theSelection) ; //设置选择模型
10.     ui->tableView->
        >setSelectionMode(QAbstractItemView::ExtendedSelection);
11.     ui->tableView->setSelectionBehavior(QAbstractItemView::SelectItems);
12.     //创建状态栏组件，代码略
13. }
```

在构造函数里首先创建数据模型 theModel，创建数据选择模型时需要传递一个数据模型变量作为其参数。这样，数据选择模型 theSelection 就与数据模型 theModel 关联，用于表示 theModel 的项数据选择操作。

创建数据模型和选择模型后，为 TableView 组件设置数据模型和选择模型：

```
ui->tableView->setModel (theModel) ; //设置数据模型
ui->tableView->setSelectionModel (theSelection) ; //设置选择模型
```

构造函数里还将自定义的槽函数 on_currentChanged() 与 theSelection 的 currentChanged() 信号关联，用于界面上 tableView 选择单元格发生变化时，显示单元格的行号、列号、内容等信息，槽函数代码如下：

```
01. void MainWindow::on_currentChanged(const QModelIndex &current, const
    QModelIndex &previous)
02. { //选择单元格变化时的响应
03.     if (current.isValid())
04.     {
05.         LabCellPos->setText (QString::asprintf ("当前单元格: %d 行, %d 列",
            current.row(),current.column()));
06.         QStandardItem* aItem=theModel->itemFromIndex(current);
07.         this->LabCellText->setText ("单元格内容: "+aItem->text());
```

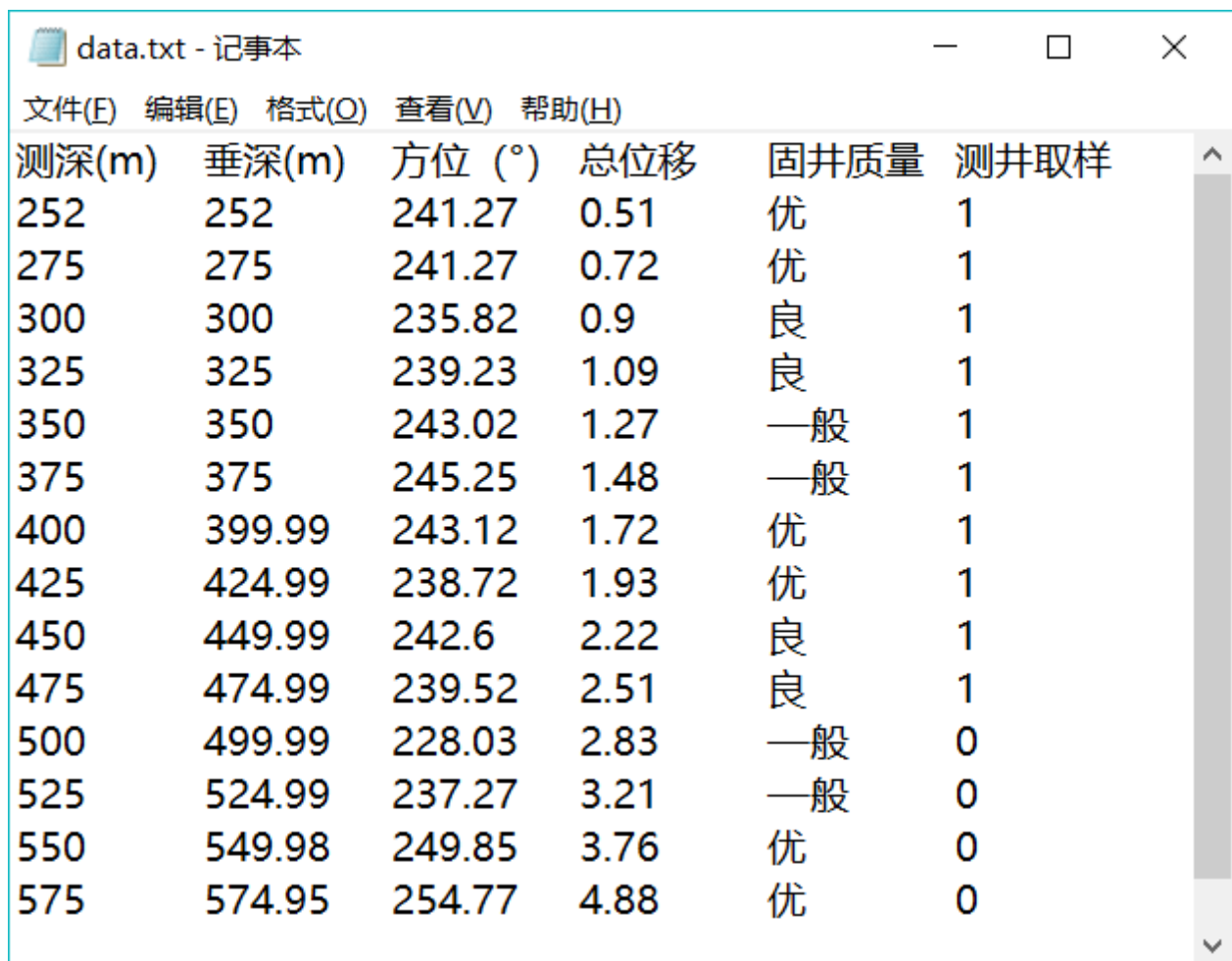
```
08.         QFont font=aItem->font();
09.         ui->actFontBold->setChecked(font.bold());
10.     }
11. }
```

从文本文件导入数据

QStandardItemModel 是标准的基于项数据的数据模型，以类似于二维数组的形式管理内部数据，适合于处理表格型数据，其显示一般采用 QTableView。

QStandardItemModel 的数据可以是程序生成的内存中的数据，也可以来源于文件。例如，在实际数据处理中，有些数据经常是以纯文本格式保存的，它们有固定的列数，每一列是一项数据，实际构成一个二维数据表。图 3 是本实例程序要打开的一个纯文本文件的内容，文件的第 1 行是数据列的文字标题，相当于数据表的表头，然后以行存储数据，以 TAB 键间隔每列数据。

当单击工具栏上的“打开文件”按钮时，需要选择一个这样的文件导入到数据模型，并在 tableView 上进行显示和编辑。图 3 的数据有 6 列，第 1 列是整数，第 2 至 4 列是浮点数，第 5 列是文字，第 6 列是逻辑型变量，“1”表示 true。



测深(m)	垂深(m)	方位 (°)	总位移	固井质量	测井取样
252	252	241.27	0.51	优	1
275	275	241.27	0.72	优	1
300	300	235.82	0.9	良	1
325	325	239.23	1.09	良	1
350	350	243.02	1.27	一般	1
375	375	245.25	1.48	一般	1
400	399.99	243.12	1.72	优	1
425	424.99	238.72	1.93	优	1
450	449.99	242.6	2.22	良	1
475	474.99	239.52	2.51	良	1
500	499.99	228.03	2.83	一般	0
525	524.99	237.27	3.21	一般	0
550	549.98	249.85	3.76	优	0
575	574.95	254.77	4.88	优	0

图 3 纯文本格式的数据文件

下面是“打开文件”按钮的槽函数代码：



```
01. void MainWindow::on_actOpen_triggered()
02. { //打开文件
03.     //QString str;
04.     QString curPath=QCoreApplication::applicationDirPath(); //获取应用程序的
    路径
05.     //调用打开文件对话框打开一个文件
06.     QString aFileName=QFileDialog::getOpenFileName(this, "打开一个文
    件", curPath, "并数据文件 (*.txt);;所有文件 (*.*)");
07.     if (aFileName.isEmpty())
08.         return; //如果未选择文件, 退出
09.
10.     QStringList fFileContent; //文件内容字符串列表
11.     QFile aFile(aFileName); //以文件方式读出
12.     if (aFile.open(QIODevice::ReadOnly | QIODevice::Text)) //以只读文本方式
    打开文件
13.     {
14.         QTextStream aStream(&aFile); //用文本流读取文件
15.         ui->plainTextEdit->clear(); //清空
16.         while (!aStream.atEnd())
17.         {
18.             QString str=aStream.readLine(); //读取文件的一行
19.             ui->plainTextEdit->appendPlainText(str); //添加到文本框显示
20.             fFileContent.append(str); //添加到 QStringList
21.         }
22.         aFile.close(); //关闭文件
23.
24.         this->LabCurFile->setText("当前文件: "+aFileName); //状态栏显示
25.         ui->actAppend->setEnabled(true); //更新Actions的enable属性
26.         ui->actInsert->setEnabled(true);
27.         ui->actDelete->setEnabled(true);
28.         ui->actSave->setEnabled(true);
29.
30.         iniModelFromStringList(fFileContent); //从StringList的内容初始化数据模
    型
31.     }
32. }
```

这段代码让用户选择所需要打开的数据文本文件，然后用只读和文本格式打开文件，逐行读取其内容，将每行字符串显示到界面上的 plainTextEdit 里，并且添加到一个临时的 QStringList 类型的变量 fFileContent 里。

然后调用自定义函数 iniModelFromStringList(), 用 fFileContent 的内容初始化数据模型。下面是 iniModelFromStringList() 函数的代码：

```
01. void MainWindow::iniModelFromStringList(QStringList& aFileContent)
02. { //从一个StringList 获取数据, 初始化数据Model
```




```

03.     int rowCnt=aFileContent.count(); //文本行数, 第1行是标题
04.     theModel->setRowCount(rowCnt-1); //实际数据行数
05.     //设置表头
06.     QString header=aFileContent.at(0); //第1行是表头
07.     //一个或多个空格、TAB等分隔符隔开的字符串, 分解为一个StringList
08.     QStringList
headerList=header.split(QRegExp("\\s+"),QString::SkipEmptyParts);
09.     theModel->setHorizontalHeaderLabels(headerList); //设置表头文字
10.
11.     //设置表格数据
12.     QString aText;
13.     QStringList tmpList;
14.     int j;
15.     QStandardItem *aItem;
16.     for (int i=1;i<rowCnt;i++)
17.     {
18.         QString aLineText=aFileContent.at(i); //获取数据区的一行
19.         //一个或多个空格、TAB等分隔符隔开的字符串, 分解为一个StringList
20.         QStringList
tmpList=aLineText.split(QRegExp("\\s+"),QString::SkipEmptyParts);
21.         for (j=0;j<FixedColumnCount-1;j++) //tmpList的行数等于
FixedColumnCount, 固定的
22.         { //不包含最后一列
23.             aItem=new QStandardItem(tmpList.at(j)); //创建item
24.             theModel->setItem(i-1,j,aItem); //为模型的某个行列位置设置Item
25.         }
26.
27.         aItem=new QStandardItem(headerList.at(j)); //最后一列是Checkable, 需要
设置
28.         //aItem=new QStandardItem(); //最后一列是Checkable, 设置
29.         aItem->setCheckable(true); //设置为Checkable
30.         //aItem->setTextAlignment(Qt::AlignHCenter);
31.         if (tmpList.at(j)=="0")
32.             aItem->setCheckState(Qt::Unchecked); //根据数据设置check状态
33.         else
34.             aItem->setCheckState(Qt::Checked);
35.         theModel->setItem(i-1,j,aItem); //为模型的某个行列位置设置Item
36.     }
37. }

```

传递来的参数 aFileContent 是文本文件所有行构成的 QStringList, 文件的每一行是 aFileContent 的一行字符串, 第 1 行是表头文字, 数据从第 2 行开始。

程序首先获取字符串列表的行数, 然后设置数据模型的行数, 因为数据模型的列数在初始化时已经设置了。



然后获取字符串列表的第 1 行，即表头文字，用 `QString::split()` 函数分割成一个 `QStringList`，设置为数据模型的表头标题。

`QString::split()` 函数根据某个特定的符号将字符串进行分割。例如，header 是数据列的标题，每个标题之间通过一个或多个 TAB 键分隔，其内容是：

```
测深 (m) 垂深 (m) 方位 (°) 总位移 (m) 固井质量 测井取样
```

那么通过上面的 `split()` 函数操作，得到一个字符串列表 `headerList`，其内容是：

```
测深 (m)
垂深 (m)
方位 (°)
总位移 (m)
固井质量
测井取样
```

也就是分解为一个 6 行的 `StringList`。然后使用此字符串列表作为数据模型，设置表头标题的函数 `setHorizontalHeaderLabels()` 的参数，就可以为数据模型设置表头了。

同样，在逐行获取字符串后，也采用 `split()` 函数进行分解，为每个数据创建一个 `QStandardItem` 类型的项数据 `item`，并赋给数据模型作为某行某列的项数据。

`QStandardItemModel` 以二维表格的形式保存项数据，每个项数据对应着 `QTableView` 的一个单元格。项数据不仅可以存储显示的文字，还可以存储其他角色的数据。

数据文件的最后一列是一个逻辑型数据，在 `tableView` 上显示时为其提供一个 `CheckBox` 组件，此功能通过调用 `QStandardItem` 的 `setCheckable()` 函数实现。

数据修改

当 `TableView` 设置为可编辑时，双击一个单元格可以修改其内容，对于使用 `CheckBox` 的列，改变 `CheckBox` 的勾选状态，就可以修改单元格关联项的选择状态。

在实例主窗口工具栏上有“添加行”、“插入行”、“删除行”按钮，它们实现相应的编辑操作，这些操作都是直接针对数据模型的，数据模型被修改后，会直接在 `TableView` 上显示出来。

添加行

“添加行”操作是在数据表的最后添加一行，其实现代码如下：

```
01. void MainWindow::on_actAppend_triggered()
02. { //在表格最后添加行
03.     QList<QStandardItem*> aItemList; //容器类
04.     QStandardItem *aItem;
05.     for(int i=0;i<FixedColumnCount-1;i++) //不包含最后1列
06.     {
```



```

07.         aItem=new QStandardItem("0"); //创建Item
08.         aItemList<<aItem;    //添加到容器
09.     }
10.     //获取最后一列的表头文字
11.     QString str=theModel->headerData(theModel->
    >columnCount()-1,Qt::Horizontal,Qt::DisplayRole).toString();
12.     aItem=new QStandardItem(str); //创建 "测井取样"Item
13.     aItem->setCheckable(true);
14.     aItemList<<aItem;    //添加到容器
15.
16.     theModel->insertRow(theModel->rowCount(),aItemList); //插入一行, 需要每
    个Cell的Item
17.     QModelIndex curIndex=theModel->index(theModel->rowCount()-1,0); //创建
    最后一行的ModelIndex
18.     theSelection->clearSelection(); //清空选择项
19.     theSelection->
    >setCurrentIndex(curIndex,QItemSelectionModel::Select); //设置刚插入的行为当前
    选择行
20. }

```

使用 QStandardItemModel::insertRow() 函数插入一行, 其函数原型是:

```
void insertRow(int row, const QList<QStandardItem *> fiitems)
```

其中, row 是一个行号, 表示在此行号之前插入一行, 若 row 等于或大于总行数, 则在最后添加一行。QList<QStandardItem *>&items 是一个 QStandardItem 类型的列表类, 需要为插入的一行的每个项数据创建一个 QStandardItem 类型的项, 然后传递给 insertRow() 函数。

在这段程序中, 为前 5 列创建 QStandardItem 对象时, 都使用文字 "0", 最后一列使用表头的标题, 并设置为 Checkable。创建完每个项数据对象后, 使用 insertRow() 函数在最后添加一行。

插入行

“插入行”按钮的功能是在当前行的前面插入一行, 实现代码与“添加行”类似。

删除行

“删除行”按钮的功能是删除当前行, 首先从选择模型中获取当前单元格的模型索引, 然后从模型索引中获取行号, 调用 removeRow(int row) 删除指定的行。

```

01. void MainWindow::on_actDelete_triggered()
02. { //删除行
03.     QModelIndex curIndex=theSelection->currentIndex (); //获取模型索引
04.     if (curIndex.row () ==theModel->rowCount () -1) //最后一行
05.         theModel->removeRow (curIndex.row () ); //删除最后一行
06.     else {
07.         theModel->removeRow (curIndex.row () ); //删除一行, 并重新设置当前选择
    行

```

```
08.         theSelection->setCurrentIndex (curIndex,
        QTableWidgetItem::Select);
09.     }
10. }
```

单元格格式设置

工具栏上有 3 个设置单元格文字对齐方式的按钮,还有一个设置字体粗体的按钮。当在 TableView 中选择多个单元格时,可以同时设置多个单元格的格式。例如,“居左”按钮的代码如下:

```
01. void MainWindow::on_actAlignLeft_triggered()
02. {    //设置文字居左对齐
03.     if (!theSelection->hasSelection())
04.         return;
05.     //获取选择的单元格的模型索引列表,可以是多选
06.     QModelIndexList selectedIndex=theSelection->selectedIndexes();
07.     for (int i=0;i<selectedIndex.count();i++)
08.     {
09.         QModelIndex aIndex=selectedIndex.at (i) ; //获取一个模型索引
10.         QStandardItem* aItem=theModel->itemFromIndex(aIndex);
11.         aItem->setTextAlignment (Qt::AlignLeft) ;//设置文字对齐方式
12.     }
13. }
```

QItemSelectionModel::selectedIndexes() 函数返回选择单元格的模型索引列表,然后通过此列表获取每个选择的单元格的模型索引,再通过模型索引获取其项数据,然后调用 QStandardItem::setTextAlignment() 设置一个项的对齐方式即可。

“居中”和“居右”按钮的代码与此类似。

“粗体”按钮设置单元格的字体是否为粗体,在选择单元格时,actFontBold 的 check 状态根据当前单元格的字体是否为粗体自动更新。actFontBold 的 triggered(bool) 的槽函数代码如下,与设置对齐方式的代码操作方式类似:

```
01. void MainWindow::on_actFontBold_triggered(bool checked)
02. { //设置字体粗体
03.     if (!theSelection->hasSelection())
04.         return;
05.     //获取选择单元格的模型索引列表
06.     QModelIndexList selectedIndex=theSelection->selectedIndexes();
07.     for (int i=0;i<selectedIndex.count();i++)
08.     {
09.         QModelIndex aIndex=selectedIndex.at(i); //获取一个模型索引
10.         QStandardItem* aItem=theModel->itemFromIndex(aIndex); //获取项数据
11.         QFont font=aItem->font(); //获取字体
12.         font.setBold(checked); //设置字体是否粗体
13.         aItem->setFont(font); //重新设置字体
```

```
14.     }  
15. }
```

数据另存为文件

在视图组件上对数据的修改都会自动更新到数据模型里，单击工具栏上的“模型数据预览”按钮，可以将数据模型的数据内容显示到 PlainTextEdit 里。

数据模型里的数据是在内存中的，工具栏上的“另存文件”按钮可以将数据模型的数据另存 为一个数据文本文件，同时也显示在 PlainTextEdit 里，其实现代码如下：

```
01. void MainWindow::on_actSave_triggered()  
02. { //保存为文件  
03.     QString curPath=QCoreApplication::applicationDirPath(); //获取应用程序的  
    路径  
04.     //调用打开文件对话框选择一个文件  
05.     QString aFileName=QFileDialog::getSaveFileName(this, tr("选择一个文  
    件"), curPath,  
06.             "井斜数据文件 (*.txt);;所有文件 (*.*)");  
07.  
08.     if (aFileName.isEmpty()) //未选择文件，退出  
09.         return;  
10.  
11.     QFile aFile(aFileName);  
12.     if (!(aFile.open(QIODevice::ReadWrite | QIODevice::Text |  
        QIODevice::Truncate)))  
13.         return; //以读写、覆盖原有内容方式打开文件  
14.  
15.     QTextStream aStream(&aFile); //用文本流读取文件  
16.  
17.     QStandardItem *aItem;  
18.     int i,j;  
19.     QString str;  
20.  
21.     ui->plainTextEdit->clear();  
22.  
23.     //获取表头文字  
24.     for (i=0;i<theModel->columnCount();i++)  
25.     {  
26.         aItem=theModel->horizontalHeaderItem(i); //获取表头的项数据  
27.         str=str+aItem->text()+"\t\t"; //以TAB见隔开  
28.     }  
29.     aStream<<str<<"\n"; //文件里需要加入换行符 \n  
30.     ui->plainTextEdit->appendPlainText(str);  
31.  
32.     //获取数据区文字  
33.     for ( i=0;i<theModel->rowCount();i++)
```

```
34.     {
35.         str="";
36.         for( j=0;j<theModel->columnCount()-1;j++)
37.         {
38.             aItem=theModel->item(i,j);
39.             str=str+aItem->text()+QString::asprintf("\t\t");
40.         }
41.
42.         aItem=theModel->item(i,j); //最后一列是逻辑型
43.         if (aItem->checkState()==Qt::Checked)
44.             str=str+"1";
45.         else
46.             str=str+"0";
47.
48.         ui->plainTextEdit->appendPlainText(str);
49.         aStream<<str<<"\n";
50.     }
51. }
```

[< 上一页](#)[下一页 >](#)

所有教程

[socket](#)[Python基础教程](#)[C#教程](#)[MySQL](#)[MySQL函数](#)[C语言入门](#)[C语言专题](#)[C语言编译器](#)[C语言编程实例](#)[GCC编译器](#)[数据结构](#)[C语言项目案例](#)[C++教程](#)[OpenCV](#)[Qt教程](#)[Unity 3D教程](#)[UE4](#)[STL](#)[Hibernate](#)[Mybatis](#)[Spring MVC](#)[TCP/IP](#)[JavaScript](#)[Spring Cloud](#)[Maven](#)[vi命令](#)[Android教程](#)[PHP](#)[Redis](#)[Spring Boot](#)[Linux](#)[Linux命令](#)[Shell脚本](#)[Java教程](#)[设计模式](#)[Spring](#)[Servlet](#)[Struts2](#)[Java Swing](#)[JSP教程](#)[CSS教程](#)[TensorFlow](#)[区块链](#)[Go语言教程](#)[Docker](#)[编程笔记](#)[资源下载](#)[关于我们](#)[汇编语言](#)[大数据](#)[云计算](#)[VIP视频](#)

优秀文章

C++指针作为函数参数（详解版）

Unity 3D Label控件

C# struct：结构体类型

Shell代码块重定向

Google App Engine是什么？

Android HttpURLConnection访问互联网资源

UserAgent（浏览器UA标识）是什么

使用SpringBoot构建其他形式的微服务

SpringWeb <form:input>标签：定义表单的文本框输入组件

Go语言冒泡排序

精美而实用的网站，提供C语言、C++、STL、Linux、Shell、Java、Go语言等教程，以及socket、GCC、vi、Swing、设计模式、JSP等专题。

Copyright ©2011-2018 biancheng.net, 陕ICP备15000209号

biancheng.net

