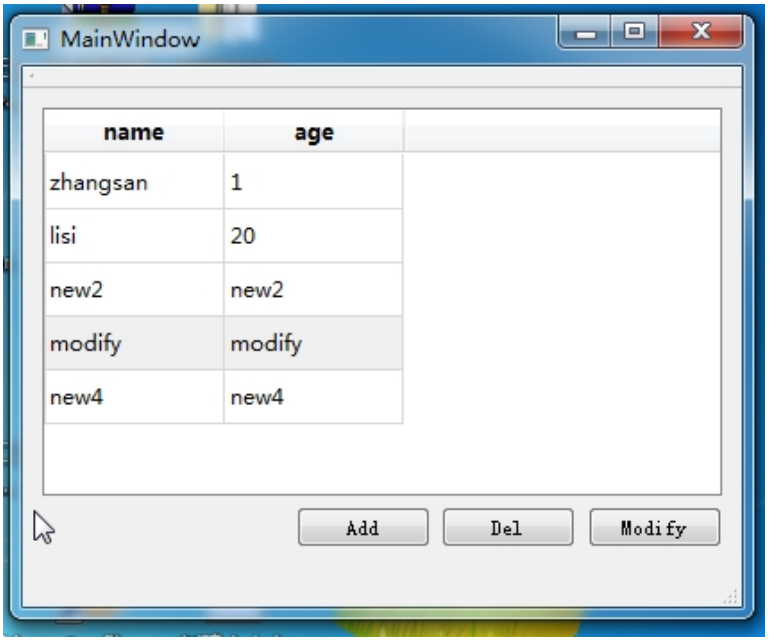
Qt学习笔记 TableWidget使用说明和增删改操作的实现 - lpxxn

看一下效果很简单的一个小功能



先说分部讲一下过程 再给出详细代码

添加数据

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    ui->tableWidget->setColumnCount(2);
    ui->tableWidget->setRowCount(2);
    ui->tableWidget->setHorizontalHeaderLabels(QStringList()<<"name"<<"age");
    ui->tableWidget->setSelectionBehavior(QAbstractItemView::SelectRows);  //整行选中的方式
    ui->tableWidget->setEditTriggers(QAbstractItemView::NoEditTriggers);   //禁止修改
    ui->tableWidget->setSelectionMode(QAbstractItemView::SingleSelection);  //设置为可以选中单个
    ui->tableWidget->setItem(0,0,new QTableWidgetItem("zhangsan"));
    ui->tableWidget->setItem(0,1,new QTableWidgetItem("1"));
    ui->tableWidget->verticalHeader()->setVisible(false);   //隐藏列表头

    ui->tableWidget->setItem(1,0,new QTableWidgetItem("lisi"));
```

```
    ui->tableWidget->setItem(1,1,new QTableWidgetItem("20"));
    ui->tableWidget->selectRow(0);
}
```

进行增删除修改操作

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QDebug>
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    ui->tableWidget->setColumnCount(2);
    ui->tableWidget->setRowCount(2);
    ui->tableWidget->setHorizontalHeaderLabels(QStringList()<<"name"<<"age");
    ui->tableWidget->setSelectionBehavior(QAbstractItemView::SelectRows);  //整行选中的方
式
    ui->tableWidget->setEditTriggers(QAbstractItemView::NoEditTriggers);   //禁止修改
    ui->tableWidget->setSelectionMode(QAbstractItemView::SingleSelection);  //设置为可以
选中单个
    ui->tableWidget->setItem(0,0,new QTableWidgetItem("zhangsan"));
    ui->tableWidget->setItem(0,1,new QTableWidgetItem("1"));
    ui->tableWidget->verticalHeader()->setVisible(false);   //隐藏列表头

    ui->tableWidget->setItem(1,0,new QTableWidgetItem("lisi"));
    ui->tableWidget->setItem(1,1,new QTableWidgetItem("20"));
    ui->tableWidget->selectRow(0);
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::on_tableWidget_currentItemChanged(QTableWidgetItem *current,
QTableWidgetItem *previous)
{

    if(previous!=Q_NULLPTR)
```

```
    {
        previous->setBackgroundColor(Qt::transparent);
    }
    if(current==Q_NULLPTR)return;
    current->setBackgroundColor(Qt::blue);
}

void MainWindow::on_btn_Add_clicked()
{
//    QAbstractItemModel *model = ui->tableWidget->model();
//     model->insertRow(model->rowCount());
    int cols=ui->tableWidget->columnCount();
    int rows=ui->tableWidget->rowCount();
    qDebug()<<rows;
    ui->tableWidget->insertRow(rows);
    for(int i=0;i<cols;i++)
    {
        ui->tableWidget->setItem(rows,i,new
QTableWidgetItem("new"+QString::number(rows)));
    }
    ui->tableWidget->selectRow(rows);
}

void MainWindow::on_btn_Del_clicked()
{
    QTableWidgetItem * item = ui->tableWidget->currentItem();
    if(item==Q_NULLPTR)return;
    ui->tableWidget->removeRow(item->row());
}

void MainWindow::on_btn_Modify_clicked()
{
    QModelIndex index =  ui->tableWidget->currentIndex();
    QList<QTableWidgetItem *> listItem = ui->tableWidget->selectedItems();
    if(listItem.count()==0)return;
    foreach (QTableWidgetItem * item, listItem) {
        item->setText("modify");
    }
//    QTableWidgetItem * item = ui->tableWidget->currentItem();
//    if(item==Q_NULLPTR)return;
//    item->setText("modify");
}
```

这里有一些对TableWidget的设置说明一下

1. 将表格变为禁止编辑

在默认情况下，表格里的字符是可以更改的，比如双击一个单元格，就可以修改原来的内容，如果想禁止用户的这种操作，让这个表格对用户只读，可以这样：

```
tableWidget->setEditTriggers(QAbstractItemView::NoEditTriggers);
```

QAbstractItemView.NoEditTriggers是QAbstractItemView.EditTrigger枚举中的一个，都是触发修改单元格内容的条件：

QAbstractItemView.NoEditTriggers	0	No editing possible. 不能对表格内容进行修改
QAbstractItemView.CurrentChanged	1	Editing start whenever current item changes.任何时候都能对单元格修改
QAbstractItemView.DoubleClicked	2	Editing starts when an item is double clicked.双击单元格
QAbstractItemView.SelectedClicked	4	Editing starts when clicking on an already selected item. 单击已选中的内容
QAbstractItemView.EditKeyPressed	8	Editing starts when the platform edit key has been pressed over an item.
QAbstractItemView.AnyKeyPressed	16	Editing starts when any key is pressed over an item.按下任意键就能修改
QAbstractItemView.AllEditTriggers	31	Editing starts for all above actions.以上条件全包括

2. 设置表格为整行选择

```
tableWidget->setSelectionBehavior(QAbstractItemView::SelectRows); //整行选中的方式
```

QAbstractItemView.SelectionBehavior枚举还有如下类型

Constant	Value	Description
QAbstractItemView.SelectItems	0	Selecting single items.选中单个单元格
QAbstractItemView.SelectRows	1	Selecting only rows.选中一行

QAbstractItemView.SelectColumns	2	Selecting only columns.选中一列

3.单个选中和多个选中的设置：
tableWidget->setSelectionMode(QAbstractItemView::ExtendedSelection); //设置为可以选中多个目标

该函数的参数还可以是：

QAbstractItemView.NoSelection　不能选择

QAbstractItemView.SingleSelection 选中单个目标

QAbstractItemView.MultiSelection　选中多个目标

QAbstractItemView.ExtendedSelection　QAbstractItemView.ContiguousSelection 的区别不明显，主要功能是正常情况下是单选，但按下Ctrl或Shift键后，可以多选

4. 表格表头的显示与隐藏

对于水平或垂直方法的表头，可以用以下方式进行 隐藏/显示 的设置：

```
    tableWidget->verticalHeader()->setVisible(false);   //隐藏列表头
    tableWidget->horizontalHeader()->setVisible(false); //隐藏行表头
```

5. 对表头文字的字体、颜色进行设置

```
    QTableWidgetItem *columnHeaderItem0 = tableWidget->horizontalHeaderItem(0); //获得水平方向表头的Item对象
    columnHeaderItem0->setFont(QFont("Helvetica")); //设置字体
    columnHeaderItem0->setBackgroundColor(QColor(0,60,10)); //设置单元格背景颜色
    columnHeaderItem0->setTextColor(QColor(200,111,30)); //设置文字颜色
```

6. 在单元格里加入控件：

QTableWidget不仅允许把文字加到单元格，还允许把控件也放到单元格中。比如，把一个下拉框加入单元格，可以这么做：

```
    QComboBox *comBox = new QComboBox();
    comBox->addItem("Y");
    comBox->addItem("N");
    tableWidget->setCellWidget(0,2,comBox);
```

6. 在单元格里加入控件：

```
    QWidget * widget=ui->tableWidget->cellWidget(i,0);//获得widget
    QComboBox *combox=(QComboBox*)widget;//强制转化为QComboBox
```

```
QString string=combox->currentText();
qDebug()<<string;
```

二. 对单元格的进行设置

1. 单元格设置字体颜色和背景颜色 及字体字符

```
QTableWidgetItem *item = new QTableWidgetItem("Apple");
item->setBackgroundColor(QColor(0,60,10));
item->setTextColor(QColor(200,111,100));
item->setFont(QFont("Helvetica"));
tableWidget->setItem(0,3,item);
```

另：如果需要对所有的单元格都使用这种字体，则可以使用 tableWidget->setFont(QFont("Helvetica"));

2. 设置单元格内文字的对齐方式

这个比较简单，使用newItem.setTextAlignment()函数即可，该函数的参数为单元格内的对齐方式，和字符输入顺序是自左相右还是自右向左。

水平对齐方式有：

Constant	Value	Description
Qt.AlignLeft	0x0001	Aligns with the left edge.
Qt.AlignRight	0x0002	Aligns with the right edge.
Qt.AlignHCenter	0x0004	Centers horizontally in the available space.
Qt.AlignJustify	0x0008	Justifies the text in the available space.

垂直对齐方式：

Constant	Value	Description
Qt.AlignTop	0x0020	Aligns with the top.
Qt.AlignBottom	0x0040	Aligns with the bottom.
Qt.AlignVCenter	0x0080	Centers vertically in the available space.

如果两种都要设置，只要用 Qt.AlignHCenter | Qt.AlignVCenter 的方式即可

3. 合并单元格效果的实现：

tableWidget->setSpan(0, 0, 3, 1) # 其参数为： 要改变单元格的 1行数 2列数 要合并的 3行数 4列数

4. 设置单元格的大小

首先，可以指定某个行或者列的大小

```
    tableWidget->setColumnWidth(3,200);
    tableWidget->setRowHeight(3,60);
```

还可以将行和列的大小设为与内容相匹配

```
    tableWidget->resizeColumnsToContents();
    tableWidget->resizeRowsToContents();
```

5. 获得单击单元格的内容

通过实现 itemClicked (QTableWidgetItem *) 信号的槽函数，就可以获得鼠标单击到的单元格指针，进而获得其中的文字信息

connect(tableWidget,SIGNAL(itemDoubleClicked(QTreeWidgetItem*,int)),this,SLOT(getItem(QTreeWidgetItem*,int)));

//将itemClicked信号与函数getItem绑定

6.QTableWidget要调整表格行宽主要涉及以下一个函数

resizeColumnsToContents(); 根据内容调整列宽
resizeColumnToContents(int col); 根据内容自动调整给定列宽
horizontalHeader()->setResizeMode 把给定列设置为给定模式
主要模式有Stretch和Fixed

7.

```
int row = rowCount();
 removeRow(row);//清除已有的行列
 setShowGrid(true);//显示表格线
 verticalHeader()->setVisible(false);//隐藏左边垂直
 QHeaderView *headerView = horizontalHeader();
 headerView->setMovable(false);//去除表头的移动
 headerView->resizeSection(0,284);//设置第一列宽
 headerView->resizeSection(1,127);//设置第二列宽
 headerView->setResizeMode(QHeaderView::Fixed);//列表不能移动
 headerView->setClickable(false);//不响应鼠标单击
 setEditTriggers(QTableWidget::NoEditTriggers);//不能编辑
 setSelectionBehavior(QTableWidget::SelectRows);//一次选中一行
 setSelectionMode(QAbstractItemView::SingleSelection);//只能单选
```

```
/*QScrollBar *scrollBar = horizontalScrollBar();
scrollBar->hide();*/
setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff);//去掉水平滚动条
setVerticalScrollMode(QAbstractItemView::ScrollPerItem);//垂直滚动条按项移动
setAutoScroll(false);//去掉自动滚动
```

作者：李鹏
出处：http://www.cnblogs.com/li-peng/

主要说Qt的以下几种容器

1.QList<T>

2.QLinkedList<T>

3.Map<T>

和一些常用的容器方法的使用

qSort

qCopy

qFind

1.QList<T>泛型集合是最常用的一种容器

看一下它的常用 操作

添加删除和两个迭代器

```
QListIterator和QMutableListIterator
```

```
#include <QCoreApplication>
#include<QList>
#include<QDebug>
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    QList<int> listInt;

    //添加
    for(int i =0;i<10;i++)
```

```
{
    listInt.append(i);
    //也可以这样添加元素
    //listInt<<i;
}
//删除
qDebug()<<"删除3";
listInt.removeAt(3);
//查询
foreach (int item, listInt) {
    qDebug()<<item;
}


qDebug()<<"Iterator";

//迭代器
QListIterator<int> iterator(listInt);
while(iterator.hasNext())
{

    qDebug()<<iterator.next();
    if(iterator.hasNext())
    qDebug()<<"the Next is :"<<iterator.peekNext();
}
//返转
iterator.toBack();
while(iterator.hasPrevious())
{
    qDebug()<<iterator.previous();
}
qDebug()<<"可变迭代器QMutableListIterator";
//可变的迭代器
QMutableListIterator<int> mutableiterator(listInt);
mutableiterator.insert(13);
mutableiterator.insert(14);
mutableiterator.insert(15);
while(mutableiterator.hasNext())
{
   int i=  mutableiterator.next();
   if(i==2||i==6)
   {
       mutableiterator.remove();
   }
```

```
    }

    //查询
    foreach (int item, listInt) {
        qDebug()<<item;
    }
    return a.exec();
}
```

2.QLinkedList<T>

QLinkedList<T>和QList<T>差不多，不同的一点是它是用迭代器做的访问项

也就是说QList<int> list只以通过这样访问它的内容list[i]而QLinkedList不可以只能用Iterator

性能上它要高于QList<T>

```
#include <QCoreApplication>
#include<QLinkedList>
#include<QDebug>

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    QLinkedList<int> link;
    link<<1<<2<<2<<3<<4;
    qDebug()<<"迭代器访问QLinkedListIterator";
    QLinkedListIterator<int> iterator(link);
    while(iterator.hasNext())
    {
        qDebug()<< iterator.next();
    }
    //删除第一个2
    link.removeOne(2);
    //添加两个3这两种方式一样
    link.push_back(3);
    link.append(3);
    //删除所有的3
    link.removeAll(3);
    qDebug()<<"普通访问foreach";
    foreach (int item, link) {
        qDebug()<< item;
```

```
    }

    qDebug()<<"迭代器QMutableLinkedListIterator";
    QMutableLinkedListIterator<int> mutableIter(link);

    while(mutableIter.hasNext())
    {
        int i= mutableIter.next();
        if(i==1)
        {
            mutableIter.insert(90);
        }
        if(i==4)
        {
            mutableIter.remove();
        }
        qDebug()<<i;
    }
    qDebug()<<"迭代器QMutableLinkedListIterator重新访问";
    mutableIter.toFront();
    while(mutableIter.hasNext())
    {
        int i= mutableIter.next();
        qDebug()<<i;
    }
    //mutable
    return a.exec();
}
```

```
a
```

3Map<T>

map类型是一个键值对 key/value组成 其它的和上边的两个集合没什么区别

```
#include <QCoreApplication>
#include<QMap>
#include<QDebug>
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
```

```
    QMap<int,QString> map;
    map.insert(1,"a");
    map.insert(2,"b");
    map.insert(3,"c");
    QMutableMapIterator<int,QString> mutableIte(map);
    while(mutableIte.hasNext())
    {
        mutableIte.next();
        qDebug()<<mutableIte.key()<<" "<<mutableIte.value();
    }
    return a.exec();
}
```

下边说一下常用的集合操作方法

qSort

qCopy

qFind

```
#include <QCoreApplication>
#include<QList>
#include<QDebug>
#include<QVector>
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    QList<int> listStrs;
    listStrs<<10<<5<<8<<2<<7;
    qSort(listStrs);
    foreach (int i, listStrs) {
        qDebug()<<i;
    }
    qDebug()<<"_____________________________";
    listStrs.clear();
    listStrs<<10<<5<<8<<2<<7;
    qSort(listStrs.begin()+1,listStrs.end()-1);
    foreach (int i, listStrs) {
        qDebug()<<i;
    }
```

```
    qDebug()<<"_____qCopy_____";
    QVector<int> newVec(5);
    qCopy(listStrs.begin(),listStrs.end(),newVec.begin());
    foreach (int i, newVec) {
        qDebug()<<i;
    }
    qDebug()<<"_____qFind_____";
    listStrs.clear();
    listStrs<<2<<5<<8<<2<<7;
    QList<int>::const_iterator iterFin=qFind(listStrs,2);
    if(iterFin!=listStrs.end())
    {
        qDebug()<<*iterFin;
    }
    else
    {
        qDebug()<<"notFound!";
    }
    return a.exec();
}
```

作者：李鹏