目录

1 Qt部件 **1**
1.1 connect . . . 1
1.2 ItemFlags . . . 1
1.3 QString . . . 1
1.4 tableView . . . 1
1.5 QFile . . . 2
1.6 QFileDialog . . . 2
1.7 QDir . . . 2

2 Qt5 布局管理 **3**
2.1 堆栈窗体 . . . 3
2.2 基本布局(了解即可) . . . 4

3 模型 **5**
3.1 标准基本对话框 . . . 5

4 TableView **6**
4.1 实现双击tableView触发事件 . . . 6
4.2 Model . . . 6
4.2.1 QAbstractItemModel(原型) . . . 6
4.2.2 QStandardItemModel . . . 7
4.2.3 QModelIndex . . . 8
4.2.4 QSelectionModel . . . 8
4.2.5 QSortFilterModel . . . 8

5 连接数据库 **9**
5.1 准备工作 . . . 9
5.2 建立数据库连接 . . . 9
5.3 访问数据库 . . . 9
5.3.1 QSqlQuery . . . 9

5.3.2 数据库语言 10

6 多线程 11

6.1 QMutex类 11
6.2 QMutexLocker类 11
6.3 QThread类 11
6.4 QSemaphore 11

7 Qt网络与通信 11

7.1 QAbstractSocket类 11
7.2 UDP协议传输 12
7.2.1 QUdpSocket类 12
7.3 TCP协议传输 12
7.4 QDataStream类 12
7.5 文件传输流程 12

1 Qt部件

1.1 connect

- 三种响应槽函数的方式, 见 文档

1.2 ItemFlags

- Qt::NoItemFlags()
- Qt::ItemIsSelectable()
- Qt::ItemIsEditable()
- Qt::ItemIsDragEnable()
- Qt::ItemIsDropEnable()
- Qt::ItemIsUserCheckable()
- Qt::ItemIsEnabled()
- Qt::ItemNeverHasChildren()

1.3 QString

- QString::arg(): 将字符内的'%N'转换为arg内的数据.

1.4 tableView

- `tableview->selectionModel()->seclectIndexes()`:返回一个QModelIndexList对象，实现返回一个选中对象的index.
- `setItem(row, column, QStandardItem)`:根据行列号更改数据.

1.5 QFile

- 文档
- 定义:`QFile *file = new QFile()`
- 新建文件:`QFile(filename`
- `:open(QIODeviceBase::OpenMode mode)`:打开文件,并选择对文件处理的方式.
- `qint64 QIODevice::write()`::写入文件.
-

1.6 QFileDialog

- `void QFileDialog::selectFile(const QString &filename)`:选择一个文件.
- `void QFileDialog::selectNameFilter(const QString &filter)`选择一个filter
- `getExistingDirectory(QWidget *parent = nullptr, const QString &caption = QString`

1.7 QDir

- 文档
- homepath():返回绝对地址.
- entryInfoList():返回QFileInfoList对象, 需要选择筛选目录.

2 Qt5 布局管理

2.1 堆栈窗体

QStackedWidget:

connect(list, SIGNAL(currentRowChanged(int)), stack, SLOT(setCurrentIndex(int)));

currentRowChanged:这个信号在用户选择或者改变了部件中当前项的时候被触发。它会发送两个参数，第一个参数是新的当前行的索引，第二个参数是旧的当前行的索引。

setCurrentIndex(int):是一个用于在 Qt 中切换 QStackedWidget 中当前显示页面的方法.

添加正则表达式:

```
QRegExp regexp("[A-Za-z]");
ui->lineEdit->syValidator(new QRegExpValidator(regExp,this));
```

QRegExp regexp(筛选的对象，大小写敏感程度，筛选方式)

2.2 基本布局(了解即可)

QFrame:Panel:是一种简单的边框样式，通常用于突出显示或者分隔不同的部分。

QFrame:Raised:凸起。

QFrame:Box:实心的边框。

QFrame:Plain:无阴影。

QFrame:Sunken:它表示一个凹陷的边框效果，通常用于突出显示或者分隔不同的部分。

QGridLayout:

setColumnStretch:是用于设置 QBoxLayout（包括 QHBoxLayout 和 QVBoxLayout）中
子部件列的拉伸因子的方法。

3 模型

[1]

3.1 标准基本对话框

QFileDialog://获取用户选择的文件名，保存的文件名，已存在的目录名，
文件名列表
getOpenFileName(),getSaveFileName(),getExistingDirectory(),getOpenFileNames
QColorDialog,QFontDialog://获取颜色值，字体
getColor(),getFont()
QInputDialog://标准输入对话框，下拉表条目输入框，int输入框，double类
型输入框
getText(),getItem(),getInt(),getDouble()
QMessageBox:
question,information,warning,critical,about,aboutQt

[1] u8"中文"可以避免乱码

4 TableView

- ShowGrid():显示网格.
- QStandardItemModel:是 Qt 中用于存储结构化数据的模型类之一。它提供了一个表格式的数据结构，可以在其中存储数据.
- setHorizontalHeaderLabels(model):用于设置表格视图的水平（列）表头标签.
- setModel（model）:将数据模型赋值给表格.
- setItem(int, int, new StandardItem):添加新数据进表格.
- setEditTriggers(state):设置可编辑性。
- QAbstractItemView:提供了一种通用的方法来管理模型中的数据.
- AllEditTriggers, NoEditTriggers:处于是否可编辑的状态.
- QModelIndexList: 用于存储QModelIndex对象的容器类.
- selectionModel()-selectedIndexes():可以获取选中的数据格.
- `T qobject_cast<T>(QObject* object)`:将父类指针安全地转换为子类指针，

4.1 实现双击tableView触发事件

```
connect(ui->tableView, &QTableView::doubleClicked, this, &MainWindow::openEditPage);
//选择发送对象tableView, 再选择触发事件QTableView::doubleClicked,最
后指定触发函数.
```

4.2 Model

4.2.1 QAbstractItemModel(原型)

主要用于自定义自己的数据结构

- 文档
- 行列相关:RowCount(),InsertRows(),removeRows().
- `bool QAbstractItemModel::setData(const QModelIndex &index, const QVariant &value, int role = Qt::EditRole)`
- `QAbstractItemModel::flags(const QModelIndex &index) const`:返回部件状态.
- headerData(), headerDataChanged():设置表头,见文档
-

4.2.2 QStandardItemModel

- 文档
- `appendRow(const QList<QStandardItem *> &items))`:添加一行数据;
- `item(int row, int column = 0)`:根据行列查询数据
- setItem(int row, int column, QStandardItem *item):根据行列设置,设置数据.
- setRowcount(int rows):将会遗弃比设置行号大的数据.
- setHorizontalHeaders(int row, QStandardItem *item):设置垂直或者平行表头.
- clear():清理所有表内数据.
- data():参考QAbstractItemModel.data():返回一个index指定的数据.
- 1

使用model创建实例:

```
QStandardItemModel model(4, 4);
for (int row = 0; row < model.rowCount(); ++row) {
    for (int column = 0; column < model.columnCount(); ++column)
    {
        QStandardItem *item = new QStandardItem(QString("row %0,
    column %1").arg(row).arg(column));
        model.setItem(row, column, item);
    }
}
```

4.2.3 QModelIndex

- 存储一个可被QAbstractItemModel使用的index.

4.2.4 QSelectionModel

- 文档
- isColumnSelected():

4.2.5 QSortFilterModel

筛选tableView的流程

1. 创建QSortFilterModel对象
2. 设置父model
3. 将tableView的model设置为QSortFilterModel对象
4. setFilterKeyColumn(),-1时为全选
5. 设置正则表达式，以及筛选对象等，使用setFilterRegExp.使用QRegExp对象

5 连接数据库

5.1 准备工作

1. lib/libmysql.dll放入msvc2013_64/bin中
2. Src\qtbase\src\plugins\sqldrivers\mysql\mysql.pro修改配置
   添加INLUDEPATH:mySQL文件下的INCLUDE目录
   添加LIB: 添加lib\libmysql.lib
   使用MSVC编译得到的dll文件复制到QT\QT5\5.14.1\msvc2017_64\plugins\sqldrivers

5.2 建立数据库连接

1. 设置数据库类型
2. 设置数据库主机名
3. 设置数据库用户名
4. 设置数据库密码
5. 打开连接

5.3 访问数据库

5.3.1 QSqlQuery

封装了访问修改执行数据库所需的一些函数.

声明:

```
1 QSqlQuery query("SELECT * FROM DATABASE");
```

- next(): 访问下一个record.
- prepare(const QString & query): 存储一条指令,遇到exec()才会执行.
- lastError()->TEXT():输出最后一条错误.

5.3.2 数据库语言

新增:

```
"INSERT INTO table (value1) VALUES (yourvalue)"
```

查找:

```
"SECLECT * FROM table WHERE **"
```

修改:

```
"UPDATE table SET value1 = yourvalue"
```

删除:

```
"DELETE FROM table WHERE **"
```

6 多线程

6.1 QMutex类

- mutex.lock():直到调用unlock为止，线程会被阻塞.
- mutex.tryLock():如果已经被锁定则立即返回.

6.2 QMutexLocker类

- 在析构函数中会解锁这个互斥量.

6.3 QThread类

- 文档
- started(),finished(): 将在线程创建时发出信号.
- run():将在创建后进行run()
- exit(),finish(): 终止线程.
- wait(): 将线程设置为阻塞态.
- sleep(): 将线程设置为休眠态.

6.4 QSemaphore

- 文档
- acquire():对对象使用P信号.
- release():对对象使用V信号.

7 Qt网络与通信

7.1 QAbstractSocket类

文档

7.2 UDP协议传输

7.2.1 QUdpSocket类

- 文档
- writeDatagram():指定端口和主机号后发送信息.
- hasPendingDatagrams(): 如果收到至少一条信息, 返回true.
- readDatagram():读取接收的数据.

7.3 TCP协议传输

7.4 QDataStream类

- 定义:

```
QDataStream::QDataStream(),QDataStream::QDataStream(QIODevice *d)
```

- 使用>>或者<<将数据传入一个类型内, DataStream内的数据对应减少类型的位数.

7.5 文件传输流程

传输方:

1. 建立连接, 使用ByteWritten的方式进行传输.
2. 选择文件后记录文件大小,文件名大小和文件名, 使用二进制数组传入socket.
3. 将剩下的文件使用read(qmin())的形式,分段读取并传送.(使用ByteWritten信号)
4. 在确认传输数据大小与已传输数据大小相等后完成传输.

接收方:

1. 建立连接, 使得QDateStream连接socket, 接收到二进制数据
2. 首先接收文件名字和文件大小等, 新建一个新的文件.
3. 接接收文件数据本体, 并分段写入文件, 同时记录文件传输进度.(使用readyRead信号)
4. 在确认文件需传送大小与已经传送大小相等时, 记录.[2]

[2]可能发生粘包, 需要拆包