**实训日志**

|  |  |  |
| --- | --- | --- |
| 学校：宝鸡文理学院 | 专业：电子科学与技术 | 学号：201691074036 |
| 日期；9月15日 | 星期：星期六 | 姓名：刘超飞 |
| 内容：（课程主要内容，代码/作业，收获，感悟，建议等）  课程主要内容：  >QT  >QT如何建立工程  >如何使用QT建立一个图形化的应用程序  >信号和槽触发机制  >QT下的网络编程  >什么是QT  QT是一套基于C++的应用程序开发套件(框架) QT拥有自己的SDK  QT具有跨平台性  我们使用QT：QT Creator(这是一个集成开发工具)  >QT如何创建工程：  >New Project>Application>Qt Widgets Application  配置工程路径，起一个工程名  选择 kits(编译工具)默认>next  选择基类为：QWidget >next>Finsh  >QT工程目录：  proname----------hello.pro  |  |------Headers  |  |------Sources  |  |------Forms  hello.pro:工程文件  Headers：存放头文件的文件夹  Sources：存放.cpp源文件的文件夹  Forms：存放ui文件的文件夹    >实现如何改变Label中的值：  Label:标签  QLabel：是一个标签类：可以显示字符也可以显示图片  PushButton：按钮  QPushButton：是一个按钮：可以点击触发事件  点击Button然后改变Label的文本：  /\*将ui文件中的label的文本清空\*/  ui->label->clear();  /\*将ui文件中的label的文本设置为My Frirst QT Project\*/  ui->label->setText("My Frirst QT Project ");  >QT建立一个图形化的应用程序  创建工程时，创建Qt Widgets Application  使用时：  -：有ui文件  -：没有ui文件  在类中定义图形化的成员：  先包含头文件：#include <QLabel>  #include <QHBoxLayout>  定义成员：  private：  QLabel \*lb;  QHBoxLayout \*hb;  在cpp中创建该成员  先包含头文件：#include <QLabel>  new一个lb：  再构造函数中：  lb = new QLabel(this);  this->lb = new QLabel;      再创建一个布局  this->hb = new QHBoxLayout;    将创建出啦的图形化成员实例添加到布局中  this->hb->addWidget(lb);    将布局加载到窗体中  setLayout(hb);  >QT的事件触发机制：  QT实现事件触发的机制我们叫做信号和槽机制。  如何实现信号触发机制：  第一种：转到槽  拖进去一个button，右键，选择go to slot  选择触发的信号，在槽函数中，写函数实现(想要触发的事件)    第二种：代码连接信号和槽  1.建立一个槽函数  .h文件中：  public slots：  槽函数的声明；  void change\_label\_text();  2.槽函数的实现：  .cpp文件中  void Widget::change\_label\_text()  {  槽函数的函数体；  }  3.连接信号和槽：  connect：用于连接信号和槽：  信号发出的对象：  发出的信号类型：  信号的接收对象：  接收到信号后触发的槽：  connect(信号发出的对象，SIGNAL(信号)，信号的接收对象，SLOT(想要触发的槽));  connect(bt, SIGNAL(clicked(bool)), this, SLOT(change\_label\_text()));    例子：不使用ui文件来实现改变label的内容  widget.h  #ifndef WIDGET\_H  #define WIDGET\_H  #include <QWidget> //QWidget基类的头文件  #include <QLabel> //QLabel的头文件  #include <QPushButton> //QPushButton的头文件  #include <QVBoxLayout> //QVBoxLayout的头文件：用于纵向布局  #include <QHBoxLayout> //QHBoxLayout的头文件：用于横向布局  class Widget : public QWidget //QWidget的子类：Widget的定义  {  Q\_OBJECT  public: //公开方法  Widget(QWidget \*parent = 0); //构造函数  ~Widget(); //析构函数  public: //公开的成员变量  QLabel \*lb; //标签lb的指针  QPushButton \*bt; //按钮bt  QHBoxLayout \*hb; //横向布局hb  QVBoxLayout \*vb; //纵向布局vb  public slots: //公开槽  void change\_label\_text(); //槽函数chang\_label\_text()：用来改变label的内容  };  #endif // WIDGET\_H  widget.cpp: widget类的实现：源文件  #include "widget.h"  #include <QLabel>  #include <QPushButton>  #include <QHBoxLayout>  #include <QVBoxLayout>  Widget::Widget(QWidget \*parent) //构造函数  : QWidget(parent)  {  this->lb = new QLabel(); //将lb实例化，此时产生一个label  this->bt = new QPushButton("Change"); //将bt实例化，此时产生一个PushButton  lb->setText("Hello world!"); //将lb的内容改为Hello world！  this->hb = new QHBoxLayout; //将hb实例化，此时产生一个横向布局hb  this->vb = new QVBoxLayout; //将vb实例化，此时产生一个纵向布局vb  vb->addWidget(lb); //将lb加入vb中  vb->addWidget(bt); //将bt加入vb中  hb->addLayout(vb); //将vb布局加入hb布局中  setLayout(hb); //将hb布局加入窗体  //连接信号和槽，将bt的clicked信号和当前类的实例连接，触发当前类的change\_label\_text()方法  connect(bt, SIGNAL(clicked(bool)), this, SLOT(change\_label\_text()));  }  Widget::~Widget()  {  }  //当前类的change\_label\_text槽函数的实现  void Widget::change\_label\_text()  {  this->lb->clear();  this->lb->setText("How are you? I am fine.");  }  QTimer的使用：  #include <QTimer>  /\*开启定时器，规定定时器的时间，单位是毫秒\*/  void QTimer::start(int msec);  /\*关闭定时器\*/  void QTimer::stop();  /\*定时器到时的信号\*/  void QTimer::timeout();  图片加载的技术：  QImage \*img = new QImage(filename);  Label->setPixmap(QPixmap::fromImage(\*img));  tar -xvzf time\_pic.tar.gz  注意：图片加载时，一定要是绝对路径，QT对相对路径的支持不是很好，要确保路径是正确的。  一定要自己去转到槽(不能复制粘贴)  QT网络编程（客户端）：  QTcpSocket：  1.工程文件中加入  Qt += network  2.在.h文件中定义一个  QTcpSocket \*sk；  public slots:  void connect\_msg();  3.在.cpp中  构造函数中：  this->sk = new QTcpSocket;  sk->connectToHost("127.0.0.1", 8888);    connect(sk, SIGNAL(connected(bool)), this, SLOT(connect\_msg));  connect(sk, SIGNAL(readyRead()), this, SLOT(recv\_msg()));  void Widget::connect\_msg()  {  qDebug() << "connect success.";  }  4.收数据  sk->readAll();  5.发数据  sk->write()；    QString 和 QByteArray的相互转换：  >QString 转换为 QByteArray  QString msg = ui->LineEdit->text();  QByteArray sendmsg = msg.toLatin1();    >QByteArray 转换为 QString  QByteAarray msg = st->readAll();  QString recvmsg = msg.toStdString.c\_str();    QString recvmsg = msg.data();  收获：  老师今天着重为我们分析的是客户端部分，首先什么是QT,>QT如何建立工程,>如何使用QT建立一个图形化的应用程序,>信号和槽触发机制,>QT下的网络编程。每一部分都讲的十分详细。  主要的难点在于槽函数的调运与编写，如果能正确调用槽函数那对于程序的实现是十分有帮助的，最后弄清楚了槽函数的定义与实质用起来得心应手。 | | |