QT第一天

一、介绍QT

```
1. QT主要用于图形化界面的开发， QT是基于C++编写的一套界面相关的类库，
     进程线程库，网络编程的库，数据库操作的库，文件操作的库.....

2. 学习QT，
     掌握QT中的类库是做什么，如何使用这个类库
     类库实例化对象(构造函数) --> 学习类库中方法(函数)的使用 --> 后台逻辑的实现

3. QT是一个跨平台的GUI图形化界面开发工具

4. QT的使用场合
     汽车仪表盘
     打印机
     医疗器械
     自动化的大型设备

5.  QT的优点
1.跨平台，具有软为完备的图形开发库，你所能想到的的图形的实现基本都有，比window的MFC的库更强大。所以很多之前做桌面开发用MFC的都转了Qt。
2.接口的封装性好，易上手，学习QT框架对学习计算机图形框架有很重要的参考意义。
3.Qt内部基本上有一套自己的简易好用的内存回收机制，对提高C++水平有帮助。
4.有很好的社区环境，市场份额在缓慢上升。
5.轻量级的开发环境，可以做嵌入式开发
```

二、软件安装

```
下载网址： https://download.qt.io/archive/online_installers/4.2/
    Windows : qt-unified-windows-x86-4.2.0-beta-online.exe
    linux : qt-unified-windows-x86-4.2.0-beta-online.run
    MAC : qt-unified-windows-x86-4.2.0-beta-online.dmg
```

三、QT工具介绍

```
1. Assistant  ---> QT类库的帮助手册的工具

2. Designer   ---> 用来设计图形化界面
```

```
    对应的界面文件为***.ui （ui文件中的内容是一种标记性的语言）

3. uic.exe    ---> 将***.ui文件转换为标准的C++的代码 ui_***.h
    C:\Qt\5.15.2\mingw81_64\bin\uic.exe

    在cmd终端下输入以下命令：
    C:\Qt\5.15.2\mingw81_64\bin\uic.exe designer.ui -o ui_designer.h

4. moc.exe    ---> 元对象编辑器工具
    C:\Qt\5.15.2\mingw81_64\bin\moc.exe
    将QT中非标准的信号和槽，转换为标准的C++的代码

5. rcc.exe    ---> 资源管理器
    C:\Qt\5.15.2\mingw81_64\bin\rcc.exe
    将QT资源文件(图片,音频文件,视频文件)，转换为标准的C++代码

6. qmake  ---> 工程管理的工具
    QT工程文件的后缀为***.pro工程文件，
    qmake工具可以根据***.pro文件，生成Makefile文件，
    通过Makefile文件编译C++的代码。

7. QTcreater  --> QT集成开发环境工具(IDE)
    将上边的所有的工具都集成到一起了。
```

四、Assistant帮助文档的使用


```
索引 ----> 查找 ---> 输入要查找的QT类/方法名.
```

```
Contents     ---> 目录
    Properties   ---> 成员变量
    Public Functions   ---> 公有的成员函数
    Reimplemented Public Functions  ---> 重写的公有的成员函数(虚函数)
    Public Slots  --> 公有的槽函数
    Protected Functions  ---> 保护的成员函数
    Reimplemented Protected Functions  ---> 重写的保护的成员函数
    Detailed Description  ---> 详细描述

QPushButton Class  ---> QPushButton
    The QPushButton widget provides a command button. More...  --》类的详细描述

Header:   ---> 头文件
    #include <QPushButton>
qmake:    ---> QPushButton包含在widgets类库中
    QT += widgets
Inherits:  --> 继承自
    QAbstractButton  ---> QPushButton类的基类
Inherited By :  ---> 被继承
    QCommandLinkButton ---> QPushButton类的子类

List of all members, including inherited members
    列出类中的所有的成员，包括继承的成员。

关于QT中类的使用的技巧：
    QT中的类的命名方式：开头为大写Q,后边具有特殊意义的单词组成，首字母大写。
    QT中的类很多都有继承的关系，学习QT中的类的方式时，设置类对象时，调用的可能是相同的方法。

    QPushButton(按钮) --> QAbstractButton --> QWidget --> QObject and QPaintDevice
    QLabel(标签) --> QFrame --> QWidget --> QObject and QPaintDevice
    QLineEdit(行编辑)  --> QWidget --> QObject and QPaintDevice

    QWidget类中的成员函数：
        setDisabled(bool )
        setEnabled(bool)
        setWindowTitle(const QString &)
        resize(int , int )

```

五、创建QT工程

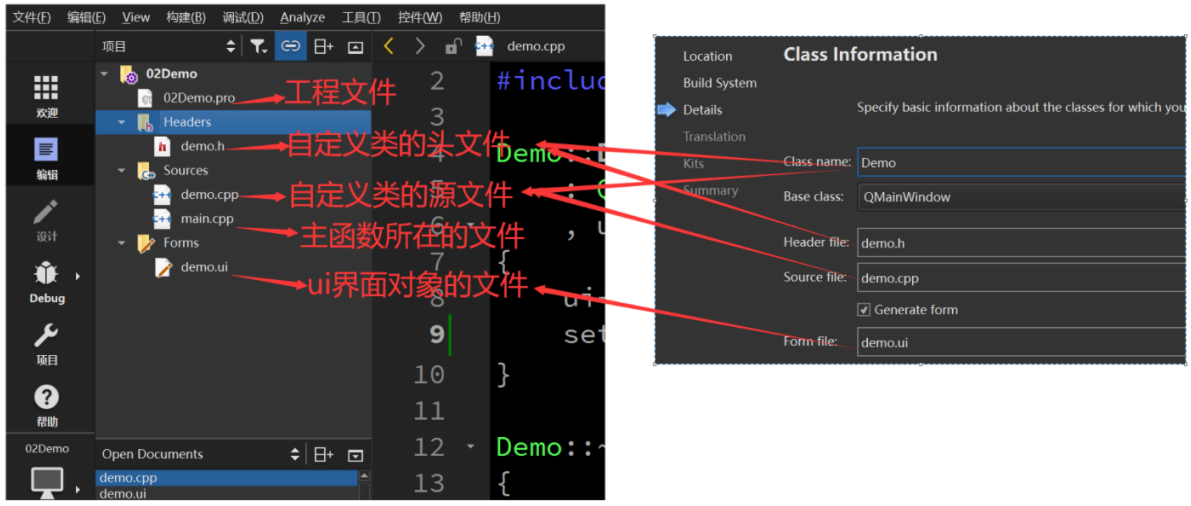
基类：

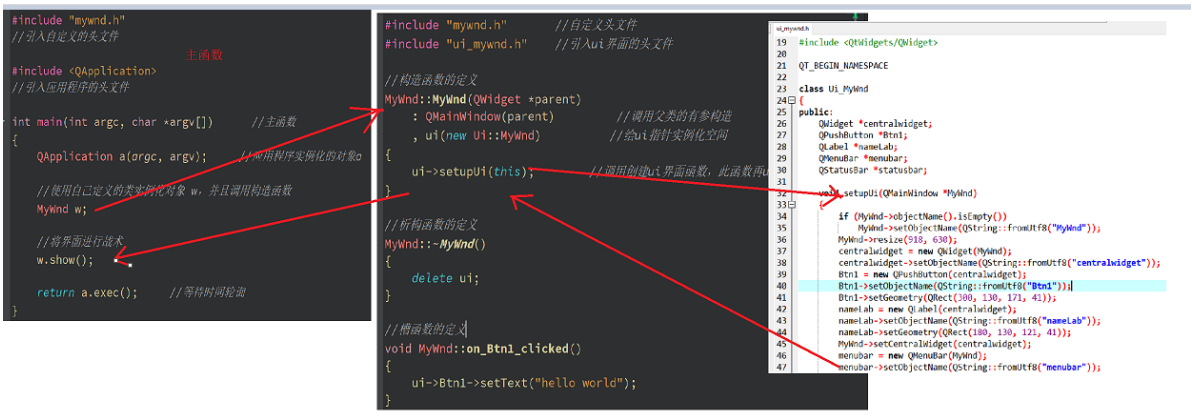
1、QMainWindow：主要用于大屏幕设备的界面开发。
主窗口界面类，主要包含：菜单栏、工具栏、状态栏、中央窗口部件等部分

2、QWidget：主要用于小尺寸屏幕设备界面的开发。
窗口部件类，特点：没有菜单栏、工具栏灯。只有中央窗口部件。

3、QDialog：主要用于弹窗提示。
悬浮于所有界面的最顶层窗口，常见的有消息对话框、文件对话框、输入对

注意：Qt中所有的窗口界面都是由 QMainWindow、QWidget、QDialog基类提供。
因为基类的构造函数中，有窗口界面绘制的功能代码。





六、工程中各个文件的介绍

1> 配置文件

```
QT       += core gui
#使用qt的类库    core核心类库    gui图形化界面类库

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
#版本超过4.0，会自动加上widgets类库

CONFIG += c++11
#支持C++11
```

```
9
10 # The following define makes your compiler emit warnings if
   you use
11 # any Qt feature that has been marked deprecated (the exact
   warnings
12 # depend on your compiler). Please consult the documentation
   of the
13 # deprecated API in order to know how to port your code away
   from it.
14 DEFINES += QT_DEPRECATED_WARNINGS
15
16 # You can also make your code fail to compile if it uses
   deprecated APIs.
17 # In order to do so, uncomment the following line.
18 # You can also select to disable deprecated APIs only up to a
   certain version of Qt.
19 #DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    #
   disables all the APIs deprecated before Qt 6.0.0
20
21 #源文件的配置文件
22 SOURCES += \
23     main.cpp \            #包含主函数
24     widget.cpp            #包含的.cpp文件
25
26 #包含的头文件
27 HEADERS += \
28     widget.h          #头文件
29
30 #配置的ui文件
31 FORMS += \
32     widget.ui
33
34 # Default rules for deployment.
35 qnx: target.path = /tmp/$${TARGET}/bin
36 else: unix:!android: target.path = /opt/$${TARGET}/bin
37 !isEmpty(target.path): INSTALLS += target
38
```

2> 头文件

```
1 #ifndef WIDGET_H              //防止文件重复包含
2 #define WIDGET_H
3
4 #include <QWidget>          //引入基类的头文件，类名即使头文件名
5 #include<QPushButton>
```

```
6
7  QT_BEGIN_NAMESPACE
8  namespace Ui { class Widget; }        //自定义命名空间，声明自定义ui界面类
9  QT_END_NAMESPACE
10
11 class Widget : public QWidget       //自定义的类，公共继承自QWidget
12 {
13     Q_OBJECT             //元对象，处理信号与槽机制使用的
14
15 public:
16     Widget(QWidget *parent = nullptr);            //构造函数声明，参数是父组件的指针
17     ~Widget();                                    //析构函数声明   虚析构函数
18
19 private:
20     Ui::Widget *ui;                  //定义一个ui界面类的指针，以便于使用通过ui文件设计的组件
21
22
23 };
24 #endif // WIDGET_H
25
```

3> 源文件

```
1  #include "widget.h"              //自己工程中的头文件
2  #include "ui_widget.h"           //将ui界面类的头文件引入，该头文件在影子目录中
3
4  Widget::Widget(QWidget *parent)      //构造函数的定义   父组件为QWidget
5      : QWidget(parent)                //调用父类的构造函数
6      , ui(new Ui::Widget)             //给自己的指针成员实例化空间
7  {
8      ui->setupUi(this);              //调用ui文件中的设置ui函数，来展示通过ui界面设计的组件
9
10
11 }
12
13 Widget::~Widget()           //析构函数额声明
14 {
15     delete ui;              //析构自己的指针成员
```

```
}

```

4> 主函数

```
#include "widget.h"         //引入自定义的头文件

#include <QApplication>     //引入应用程序的头文件

#include<iostream>
using namespace std;

int main(int argc, char *argv[])     //主程序
{
    QApplication a(argc, argv);       //实例化应用程序的对象，为了后台轮询使用

    Widget w;           //实例化自定义类的对象，调用无参构造函数

    w.show();           //调用函数，展示界面内容，该函数是父类中已经写好的函数


    int b = a.exec();
    cout<<b<<endl;
    return b;                   //应用程序对象的轮询等待，等待的有：
                                        //1、用户是否触发ui界面的动作
                                        //2、等待相应信号产生
                                        //3、等待相应事件处理
}

```

七、第一个QT界面

```
#include "widget.h"
#include "ui_widget.h"
#include<QDebug>                //信息调试类
#include<QIcon>                //图标头文件
#include<QPalette>          //调色板头文件

Widget::Widget(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::Widget)

```

```
{
    ui->setupUi(this);

    //获取窗口标题
    QString windwName =  this->windowTitle();
    qDebug() << windwName;              //类似于cout
    qDebug("%s\n", "windwName");         //类似于printf
    qDebug()<<QString("%1   %2").arg(520).arg(1314);    //第三种使用方式

    //设置窗口标题
    this->setWindowTitle("我的第一个窗口");

    //设置图标
    //QIcon icon("D:\\22061QT\\day2\\03Login\\icon\\login.png");
    //this->setWindowIcon(icon);
    this->setWindowIcon(QIcon("D:\\22061QT\\day2\\03Login\\icon\\login.png"));

    //设置窗口大小
    this->setFixedSize(200,100);          //设置固定尺寸
    this->setMaximumSize(1024, 768);       //设置 最大尺寸
    this->setMinimumSize(500,400);          //设置最小尺寸
    this->resize(1000,800);              //重新设置大小

    //移动窗口位置
    this->move(50, 50);
    qDebug()<< this->pos();
    qDebug()<< "x: "<<this->x()<<" y:"<<this->y();

    //改变背景色
    //this->setBackgroundRole(QPalette::Dark);
    //this->setAutoFillBackground(true);          //允许改变背景

    //取消框的表头
    //this->setWindowFlags(Qt::FramelessWindowHint);

    //设置窗口透明度
    this->setWindowOpacity(0.7);
    //参数：是实型数据，表示透明的程度
}

Widget::~Widget()
```

```
{
    delete ui;
}
```

八、QDebug类

这是一个信息调试类，使用方法跟c语言的printf和C++中的cout用法差不多

需要引入头文件：

```
#include<QDebug>
```

使用方法

```
    qDebug() << "hello world";             //类似于cout
    qDebug("%s\n", "windwName");          //类似于printf
    qDebug()<<QString("%1   %2").arg(520).arg(1314);    //第三种使用方式
```

九、对象树模型

```
#include <iostream>
#include<list>

using namespace std;

class Object;          //前置声明类

typedef list<Object *> ObjectList;    //类型重定义

class Object
{
public:
    ObjectList children;         //记录子组件的指针

    Object(Object * parent = nullptr)
    {
        if(parent != nullptr)
        {
            //说明该组件有父组件，那么，其父组件就有children链表
            //我们就要将该组件的地址放入父组件的孩子链表中，以便于析构父组件时，将子组件也进行析构
            parent->children.push_back(this);
        }
```

```
    }

    virtual ~Object()
    {
        for(auto it=children.begin(); it!=children.end();
it++)
        {
            delete *it;
        }
    }

    ObjectList &childrenList()
    {
        return this->children;
    }

};

//定义测试类
class A:public Object
{
public:

    A(Object *parent = nullptr) {
        //判断该组件是否设置父组件
        if(parent != nullptr)
        {
            parent->children.push_back(this);

        }

        cout<<"A::构造函数"<<endl;
    }

    ~A()
    {
        cout<<"A::析构函数"<<endl;
    }
};

class B:public Object
{
public:

    B(Object *parent = nullptr) {
```

```
        //判断该组件是否设置父组件
        if(parent != nullptr)
        {
            parent->children.push_back(this);

        }

        cout<<"B::构造函数"<<endl;
    }

    ~B()
    {
        cout<<"B::析构函数"<<endl;
    }
};


int main()
{
    A aa;         //调用a类的构造函数

    B *p = new B(&aa);       //定义一个b类的组件，将a类的对象当做父组件

    return 0;
}

```

作业：

一、将qt工程的每个文件，进行复习，自己将每个给定的代码，注释清楚

二、将对象树模型手动敲一遍，并做到理解