

1 QT写代码的框架

```
1 #include <QApplication> //QT的框架头文件
2
3 int main(int argc, char *argv[])
4 {
5     QApplication a(argc, argv); //QT的框架初始化
6
7
8     return a.exec(); //a.exec()作用是让程序不死 类似于while(1)循环 循环检测事件的产生
9 }
10
```

2 编译时产生的文件

名称	修改日期	类型	大小
01cliked_darw	2020/1/17 14:40	文件夹	
01untitled	2020/5/23 14:32	文件夹	
build-01cliked_darw-Desktop_Qt_5_9_0_MinGw64	2020/1/17 9:21	文件夹	
build-01untitled-Desktop_Qt_5_9_0_MinGw64	2020/5/23 14:32	文件夹	
build-curtbutton-Desktop_Qt_5_9_0_MinGw64	2020/1/17 11:53	文件夹	
build-ShapeWidget-Desktop_Qt_5_9_0_MinGw64	2020/1/17 10:37	文件夹	
build-video-Desktop_Qt_5_9_0_MinGw64	2020/4/28 19:01	文件夹	
curtbutton	2020/1/17 14:40	文件夹	
Image	2020/1/15 11:06	文件夹	
ShapeWidget	2020/1/17 14:40	文件夹	
video	2020/4/28 19:47	文件夹	

debug	2020/5/23 14:32	文件夹	
release	2020/5/23 14:32	文件夹	
.qmake.stash	2020/5/23 14:32	STASH 文件	1 KB
Makefile	2020/5/23 14:32	文件	27 KB
Makefile.Debug	2020/5/23 14:32	DEBUG 文件	37 KB
Makefile.Release	2020/5/23 14:32	RELEASE 文件	37 KB

3 创建空项目

```
1 #include <QApplication> //QT的框架头文件
2 #include <QWidget>
```

```

3 #include <QDebug>
4 int main(int argc, char *argv[])
5 {
6     QApplication a(argc, argv); //QT的框架初始化
7     //QWidget
8     QWidget win;
9     win.resize(400, 300);
10    win.setWindowTitle("hello qt");
11    win.show(); //显示
12    qDebug() << "hello";
13
14    return a.exec(); //a.exec()作用是让程序不死 类似于while(1)循环 循环检测事件的产生
15 }
16

```

QT中的命名规则:

对于类的命名 大驼峰法(单词首字母大写)

类: QApplication

对于函数的命名 小驼峰法(首个单词的首字母小写, 后面的单词首字母大写)

setWindowTitle

4 .pro文件

```

1  在使用Qt向导生成的应用程序.pro文件格式如下:
2
3  QT      += core gui    //包含的模块
4  greaterThan(QT_MAJOR_VERSION, 4): QT += widgets //大于Qt4版本 才包含widget
    模块
5  TARGET = QtFirst    //应用程序名 生成的.exe程序名称
6  TEMPLATE = app      //模板类型      应用程序模板
7  SOURCES += main.cpp\    //源文件
8              mywidget.cpp
9  HEADERS += mywidget.h    //头文件
10
11  .pro就是工程文件(project), 它是qmake自动生成的用于生产makefile的配置文件。.pr
    o文件的写法如下:
12  1  注释
13  从“#”开始, 到这一行结束。

```

```

14 1 模板变量告诉qmake为这个应用程序生成哪种makefile。下面是可供使用的选择： TEMPLATE = app
15 n app -建立一个应用程序的makefile。这是默认值，所以如果模板没有被指定，这个将被使用。
16 n lib - 建立一个库的makefile。
17 n vcapp - 建立一个应用程序的VisualStudio项目文件。
18 n vclib - 建立一个库的VisualStudio项目文件。
19 n subdirs -这是一个特殊的模板，它可以创建一个能够进入特定目录并且为一个项目文件生成makefile并且为它调用make的makefile。
20 1 #指定生成的应用程序名：
21 TARGET = QtDemo
22 1 #工程中包含的头文件
23 HEADERS += include/painter.h
24 1 #工程中包含的.ui设计文件
25 FORMS += forms/painter.ui
26 1 #工程中包含的源文件
27 SOURCES += sources/main.cpp sources
28 1 #工程中包含的资源文件
29 RESOURCES += qrc/painter.qrc
30 1 greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
31 这条语句的含义是，如果QT_MAJOR_VERSION大于4（也就是当前使用的Qt5及更高版本）需要增加widgets模块。如果项目仅需支持Qt5，
32 也可以直接添加“QT += widgets”一句。
33 不过为了保持代码兼容，最好还是按照QtCreator生成的语句编写。
34 1 #配置信息
35 CONFIG用来告诉qmake关于应用程序的配置信息。
36 CONFIG += c++11 //使用c++11的特性
37 在这里使用“+=”，是因为我们添加我们的配置选项到任何一个已经存在中。这样做比使用“=”那样替换已经指定的所有选项更安全。

```

5 qt简单应用程序

```

1 main入口函数中
2 #include "widget.h"
3 #include <QApplication>
4
5 int main(int argc, char *argv[])
6 {
7     QApplication a(argc, argv);
8     Widget w;
9     w.show();
10

```

```
11     return a.exec();
```

```
12 }
```

```
13
```

```
14 解释:
```

```
15 1 Qt系统提供的标准类名声明头文件没有.h后缀
```

```
16 1 Qt一个类对应一个头文件，类名就是头文件名
```

```
17 1 QApplication应用程序类
```

```
18 n 管理图形用户界面应用程序的控制流和主要设置。
```

```
19 n 是Qt的整个后台管理的命脉它包含主事件循环，在其中来自窗口系统和其它资源的所有事件处理和调度。它也处理应用程序的初始化和结束，并且提供对话管理。
```

```
20 n 对于任何一个使用Qt的图形用户界面应用程序，都正好存在一个QApplication 对象，而不论这个应用程序在同一时间内是不是有0、1、2或更多个窗口。
```

```
21 1 a.exec()
```

```
22 程序进入消息循环，等待对用户输入进行响应。这里main()把控制权转交给Qt，Qt完成事件处理工作，当应用程序退出的时候exec()的值就会返回。在exec()中，
```

```
23 Qt接受并处理用户和系统的事件并且把它们传递给适当的窗口部件。
```