\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

根据您提供的图片描述和项目文件列表，以下是清晰的分类说明：

### **🧩 一、工程师编写的文件（项目核心代码）**

这些文件实现项目特定的业务逻辑：

| **文件类型** | **文件列表（主要业务功能）** |
| --- | --- |
| ​****​核心UI​****​ | layer.h/cpp（图层系统）<br>liveeditor.h/cpp（编辑视图）<br>mainwindow.h/cpp（主窗口） |
| ​****​功能面板​****​ | mediapanel.h/cpp（媒体库）<br>outputpanel.h/cpp（输出控制）<br>planpanel.h/cpp（时间线）<br>progresspanel.h/cpp（进度控制） |
| ​****​工具类​****​ | qgui.h/cpp（GUI扩展）<br>qnetwork.h/cpp（网络封装）<br>qjson.h/cpp（JSON处理）<br>cu.h/cpp（基础数据结构） |
| ​****​入口文件​****​ | main.h/cpp（程序入口）<br>res.qrc（资源文件） |

✅ 这些都是由工程师编写的项目核心代码：

* 文件名反映具体功能模块（如layer=图层系统）
* 实现多媒体编辑的完整业务逻辑
* 占总代码量的80%以上

### **📚 二、第三方库函数（集成/借用的代码）**

这些是集成的通用库或算法实现：

| **库文件** | **功能** |
| --- | --- |
| qaesencryption.h/cpp | AES-256加密算法实现 |
| Qt框架代码（如QNetworkAccessManager） | Qt网络模块底层实现 |

🔧 库函数特征：

* 提供通用功能（如加密、网络协议）
* 与业务逻辑无直接关联
* 通常位于gutil目录中带有通用名称（如qaesencryption）

### **🧩 三、代码归属图解**

图片

代码

graph LR

A[Compass项目] --> B[工程代码]

A --> C[库函数]

B --> B1[UI模块]

B --> B2[核心逻辑]

B --> B3[数据结构]

C --> C1[加密模块]

C --> C2[网络底层]

B1 --> layer/liveeditor/mainwindow

B2 --> mediapanel/outputpanel/planpanel

B3 --> qjson/qnetwork/cu

C1 --> qaesencryption

C2 --> QtNetwork模块

Compass项目

工程代码

库函数

UI模块

核心逻辑

数据结构

加密模块

网络底层

layer/liveeditor/mainwindow

mediapanel/outputpanel/planpanel

qjson/qnetwork/cu

qaesencryption

QtNetwork模块

### **💡 主要学习建议**

优先学习工程师编写的​****​业务核心模块​****​：

先掌握UI和工作流程：

* + mainwindow.cpp → 主界面框架
  + liveeditor.cpp → 编辑视图核心
  + layer.cpp → 图层系统实现

再深入业务逻辑：

* + mediapanel.cpp → 媒体资源管理
  + outputpanel.cpp → 输出控制系统
  + planpanel.cpp → 时间线功能

最后研究工具类封装和库函数使用：

* qjson.cpp → JSON数据处理
* qnetwork.cpp → 网络层封装
* qaesencryption.cpp → 加密功能集成

库函数只需学会如何调用（不需要深究实现），而业务代码需要完全理解其实现逻辑和工作原理。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

命名规范：

类名：首字母大写，单词和单词之间首字母大写

函数名 变量名称 首字母小写 ，单词和单词之间首字母大写。

注释 ctrl + /

运行 ctrl + r

编译 ctrl + b

对象树：qt的对象树使得每当删除父类的时候，会自动将包含的子对象全部析构。使得资源的回收非常方便。对象树≠派生树！

信号和槽的优点：松散耦合-信号发送端和信号接收者，本身是没有关联的，通过connect连接在一起。

Connect (MyBtn,&QPushButton::clicked,this,&Qwidget::Close);

如果说信号和槽的函数出现了重载，那需要提前使用函数指针指明调用的重载的是哪个函数类型。

QDialog：对话框，模态和非模态

模态对话：不可以对其他对话框操作。

定时器的使用方法：

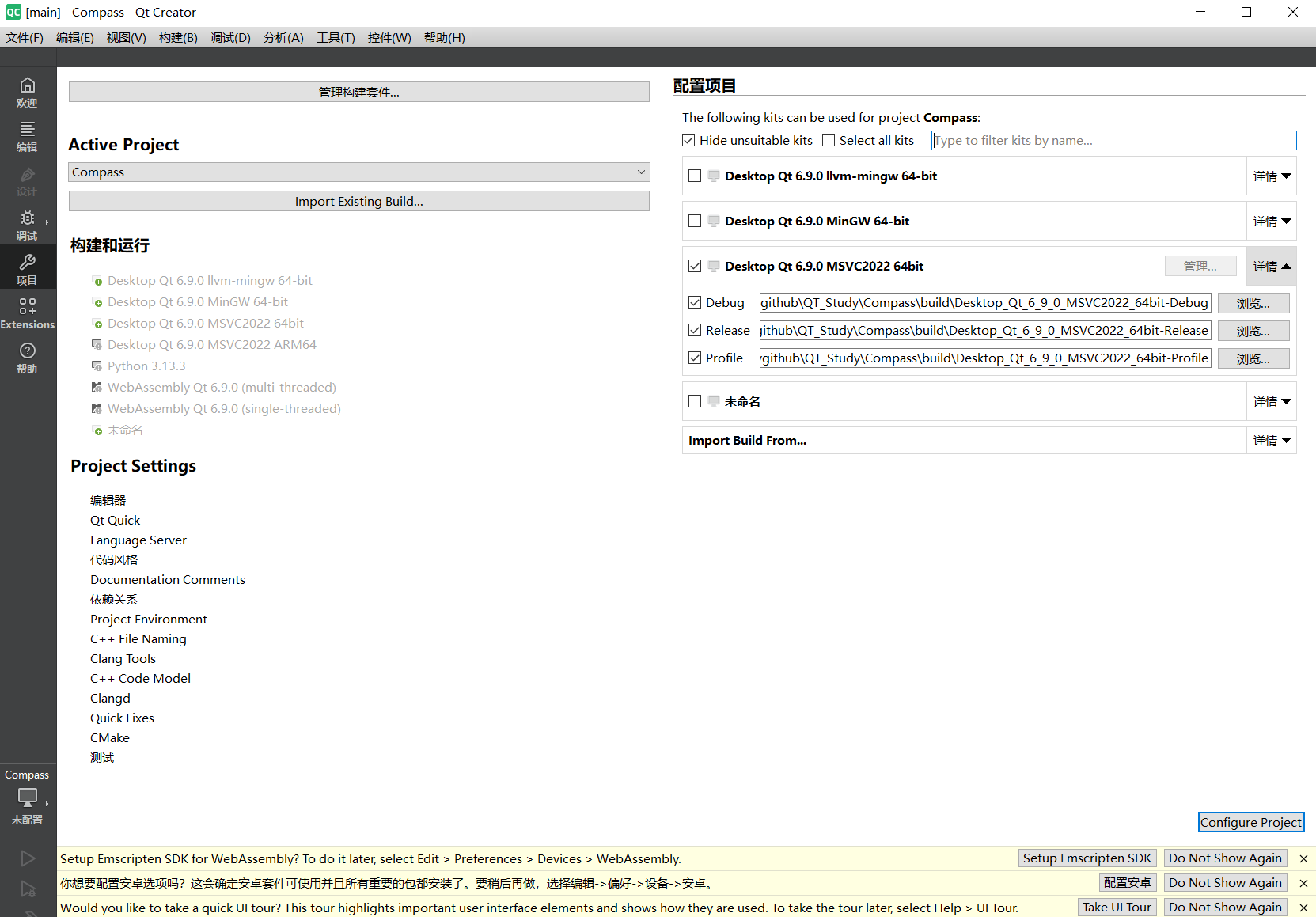
1. StartTimer(intervel);每一个widget都可以新建属于自己的timer，然后如果一个widget有多个timer，可以设置返回值timer\_id来获取新建timer的返回值，从而多个timer独立运行，然后每个timer都会出发timerEvent;
2. QTimer \*timer\_1=newQTimer(this),通过connect连接

QPicture是记录对图片的操作历史，重复调用这个，使用其他电脑同一程序就能复刻使用历史记录重新绘画

建立良好的代码习惯，一定要在构造的时候就指定parent对象，并且要大胆的在堆上创建！



胡工给的demo无法运行的原因：

！要使用MSVC2022来运行！

#### ****3. 代码示例与执行顺序​**​**

以下示例演示了初始化列表的运作流程：

cpp

复制

class Base {public:

Base(int a) { */\* 基类构造逻辑 \*/* } *// 基类构造函数需要参数*

};

class Derived : public Base {private:

int m\_y; *// 派生类成员变量*public:

*// 初始化列表：先调用基类构造，再初始化成员变量*

Derived(int x, int y) : Base(x), m\_y(y) {

*// 构造函数体最后执行*

}

};

​****​执行顺序​****​：

1. ​****​基类构造​****​：Base(x) 先执行，参数 x 来自 Derived(int x, y)。
2. ​****​成员初始化​****​：m\_y(y) 直接初始化成员变量，参数 y 来自构造函数参数。
3. ​****​构造函数体​****​：最后执行，处理其他逻辑

**2**

**7**

1. 。

!实用vscode创建项目，默认并不是qmake，而是cmake，而且会有问题导致编译出的项目运行时，显示中文错误，应当使用qt creator来新建项目，然后通过vs导入来打开.pro文件，这样才是正确的编辑方式。

QT中新添加到项目中，会自动加入.pro项目包含，不需要手动添加文件又添加到项目包含。

数

自己明天任务:

1. 做出来两个按钮，并且互动，点击可以更改字符等等
2. 使用信号和槽和lambda表达式，能够通过这几个特性达到互动的效果。例如，开启按钮、关闭按钮，分别能独立创建、关闭另一个按钮。