SDR C++代码文件处理类的说明：

1. SDR类

作用：SDR界面类，主要负责界面显示任务；

技术点：当前无技术点，信号和槽不算，使用了消息队列中的接口函数：

add\_event（）向队列中添加消息；

1. SDR\_Engine类

作用：作为SDR时间处理线程的开启类，提供了线程开启的入口函数；

技术点：当前无技术点，使用了工厂策略模式构建的类的对象，来处理对应分支的业务，避免将所有业务暴露于客户端；

1. thread\_pool类

作用：线程池类，可以初始化线程池中线程的数量，并使用有限的资源对线程进行处理；

技术点：使用多线程编程，主要包括线程的创建、function<>的使用，bind的使用，mutex的使用，条件变量的使用；

主要接口：

|  |  |
| --- | --- |
| 接口 | 作用 |
| void stop() | 停止线程池的运转 |
| void add\_task(const task\_t&) | 向线程池中添加任务(注意添加任务的方式) |

1. MySingleTon-queue类

作用：构建软件的全局变量，当前构建了各个文件统一使用的消息队列；后期有任何全局变量的需求都可添加到里面；

技术点：使用线程安全的单例模式，使用deque作为消息队列的实际承载体；

主要接口：

|  |  |
| --- | --- |
| 接口 | 作用 |
| void add\_event(const std::string &new\_event); | 向消息队列中添加事件消息 |
| std::string get\_event(); | 向消息队列中取出事件消息 |
| void clear\_event(); | 清除消息队列中的作用消息 |
| bool is\_empty(); | 返回消息队里是否为空 |

1. MySingleTon类（不在项目中）

与上述MySingleTon-queue类的唯一不同的是：在MySingleTon类中的自己实现了循环消息队列来代替deque，目前需要固定消息队列大小，感觉存在不足；

1. MySwitch文件与MyContext文件

作用：对SDR\_Engine中需要处理的不同分支进行封装；

技术点：主要使用了策略与简单工厂结合的设计模式，MySwitch文件中对不同的策略进行实现，每个策略需要继承自抽象类，并实现策略处理函数；

MyContext文件中维护一个对策略进行维护与创建的工厂类；

主要接口：

|  |  |
| --- | --- |
| 接口 | 作用 |
| algorithmInterface() | 继承MySwitch抽象类，实现对应的业务处理分支； |
| hash\_与 hash\_compile\_time | 实现switch中直接对字符串判断的函数 |
| MyContext(MySwitch\*m\_switch)构造函数 | 使用对象直接初始化构造函数 |
| MyContext::MyContext(string type)构造函数 | 使用对应字符初始化构造函数，封装了switch选项，在添加时间分支之后需要在此添加分支 |
| Void DO\_GetResult() | MyContext类提供的不同事件分支处理的函数； |

1. 状态机设计模式构造的类

之前想在设计状态机的时候去使用状态机设计模式，但是后面慢慢发现，SDR\_Engine实际上是根据消息队里进不同处理的分支处理模式；

但是后续的很多V3代码中，就是真的状态机

1. MySingleTon-queue类改进

|  |  |
| --- | --- |
| 接口 | 作用 |
| void add\_event(const std::string &new\_event, const std::string &queue\_name); | 向消息队列中添加事件消息,根据队列名称 |
| std::string get\_event(); | 向消息队列中取出事件消息，根据队列名称 |
| void clear\_event(); | 清除消息队列中的作用消息，根据队列名称 |
| bool is\_empty(); | 返回消息队里是否为空，根据队列名称 |
| bool create\_queue (const std::string & queue\_name) | 根据队列名称，创建队列元素 |
| bool delete\_queue(const std::string & queue\_name) | 根据队列名称，删除队列元素 |
| MySingleTon\_queue get\_queue(const std::string & queue\_name) | 获取队列 |

原版设计思路：

在单例模式中，只有一个list，用来存放一个结构体，结构体为：

struct Queue\_Element

{

string queue\_name;

Queue\_ElementPtr m\_Queue\_ElementPtr;

mutex \*m\_queue\_mutex;

}

其中queue\_name为队列名称为key，m\_Queue\_ElementPtr为指向队列的指针，m\_queue\_mutex为队列配套的锁；

不光是容器中的队列需要配套的加锁，容器本身也需要加锁，在创建队列的时候，需要保证各个线程创建是没有冲突或者重复的

最初想创建一个static的队列指针，在需要哪个队列该指针指向哪个，但是发现在多线程时存在读写冲突，后续直接使用局部变量进行获取队列