第三方库：

ace：底层网络框架，用于网络框架搭建

amqpcpp：MQ接口，用于RabbitMQ消息队列对接

google/protobuf：消息编码，用于消息体的压缩编码

hiredis：redis接口，用于redis数据库的连接和访问

json：json解析封装，用于json消息封装和解析以及配置文件解析

mysql-connector-c：mysql底层C语言官方API接口，提供给mysqlpp底层

mysqlpp：mysql接口面向对象封装，用于mysql数据库连接和访问

openssl：网络加密协议封装，用于数据加密和数字签名

编码风格：

1. 代码使用Google C++ Style Guide编程规范。
2. cpp或cc文件必须有对应的.h或.hpp文件，建议使用.cpp和.h
3. 类成员函数和变量命名不使用大小写混写，统一小写（HTTP等特例除外），单词间使用下划线分隔，或直接全小写，如：string user\_name、string username
4. 成员变量不使用匈牙利命名 属性+类型+名称方法，直接命名，开头或末尾以“\_”标示类成员变量，如：string \_user\_name、string user\_name\_
5. char初始化为\0，int初始化值为0，float为0.0，string不做初始化，char\*用memset初始化为0，指针以NULL或nullptr(c++11)初始化。初始化时不分行，即 int i = 0;不换行初始化int i; i = 0;
6. 底层代码数据类型全部采用DMType.h重新typedef定义的类型，以DM\_开头，业务代码可用可不用
7. 关键结构体或数据类定义必须以构造形式初始化，并且尽可能的重载赋值运算符
8. 类成员变量权限尽可能的使用private，成员函数除接口外同样统一使用private或protect
9. 循环嵌套不超过三层，多余的提取函数，单个函数不超过200行，非特殊场景不使用递归
10. 模板接口函数使用内联函数，并单独使用.inl文件，重复代码块操作使用宏定义函数代替。

设计原则：

1. 上层类尽量从简，只暴露简单接口函数即可，做好权限控制，具体操作或复杂实现，调用等做基类继承
2. 常用功能模块以继承方式封装在底层，如MQ，ACE，MemoryPool，SessionManager，openssl等
3. 基本功能模块单独提取，以组合方式嵌入到功能类，如mysqlpp，hiredis，json等
4. 底层封装和上层应用必须尽可能的彻底解耦，底层或上层各自解耦优先级可降低
5. 所有代码在启动前必须有文档支撑

网络框架依然使用之前的设计：



其中gate、maintain和redis部分优先级较低，后续优化完成，前期暂时不划入。

消息协议有所调整：

消息头一共划分出四个部分，分别占4个字节（32位），一共16个字节。

1. 消息ID，依据时间戳和用户id组合生成，来标示消息的唯一性
2. 消息体长度
3. 消息类型（消息命令码）
4. 消息签名

其中消息签名待框架代码完成后最后实现。

数据库调整：

暂时使用mysql，稳定后，后续优化加入redis，并且将redis和mysql单独封装后再统一到一个接口类中，通过参数来选定操作数据库。Redis只存储少量的热数据和适合NoSQL存储的数据。

代码框架：

优化的新框架在原来基础上会大很多，把更多的通用代码放在了底层，尽可能的让上层跟简单的去使用，并且消除底层和上层之间的耦合，使在写上层代码时，可以做到完全不需要关心和了解底层实现。

底层包含部分：

1. ACE网络处理
2. 连接通道管理
3. MQ消息处理
4. 进程线程处理
5. 消息编码解码
6. 消息派发/转发/回发
7. 消息加密

代码框架：

1. DMServer：集中控制类，继承多进程实现方式，业务代码继承这个类实现新的业务服务，传入配置信息，设置业务需要开启的进程数量，业务线程数量，端口等配置信息。
2. 最外层三个主要功能类，这三个类都在DMServer内，分别是DMAcceptor、DMDispatcher、DMService。分别负责IO上报，消息接收派发处理，以及业务实现。
3. DMAcceptor相对简单，使用Reactor框架，采用epoll模型。Reactor反应器已经由ACE提供。
4. DMDispatcher需要负责接收ACE和MQ的两路消息，剥离消息头部分，校验消息的安全性和完整性，数据可靠且完整后，获取完整的可用消息内容。将消息数据插入到DMService的消息队列。
5. DMService以多线程竞争的方式消费消息队列，具有消息加密解密和编解码功能，以数据类或结构体的形式得到消息数据，随后完成业务代码。



消息依次是从DMAcceptor(网络层)到DMDispatcher(传输层)到DMService(应用层)，其中DMDispatcher与DMService之间存在一个消息队列，用于单线程过度到多线程。

DMServer：继承两层，最底层为MultiProcess，进程处理类，上层继承一个DMServerImpl，为DMServer具体细节实现操作。

DMAcceptor：不做继承，因为ACE实现较多，代码量不大，直接将IO事件通知到DMDispatcher即可。

DMDispatcher：DMServer需要把DMService注册进来，DMDispatcher需要使用DMService的消息队列，DMDispatcher包含多个功能。1.对接MQ。2.接收ACE及MQ两路消息。3.管理ACE连接的消息通道4.对消息进行校验。由于对外屏蔽且功能单一。这个类暂时不做继承，这四个功能就在当前类实现。

DMService：这个类最为复杂，需要完成以下功能。1.多线程消费消息队列。2.消息的加密解密和编码解码。3.系统消息处理。继承五层，最底层用于线程处理，倒数第二层用于消息接收，倒数第三层消息编解码和加密解密处理，倒数第四层为接口封装，部分系统消息根据消息类型进行处理，这层屏蔽系统消息。最上层提供要重载的业务接口。其中倒数第二层和倒数第三层要留接口给最上层用于消息编解码和回发。

功能模块：memorypool(单例)，json，openssl，protomessage/messagefactory，sessionmanager(单例)。

在框架完成后，上层业务使用此框架，只需要继承三个类：1.DMServer继承框架，用于初始化和启动业务。2. DMService业务逻辑，注册到框架中。DMMessageFactory，消息工厂/容器，注册到DMService中，DMService依此来进行消息函数映射和编解码。