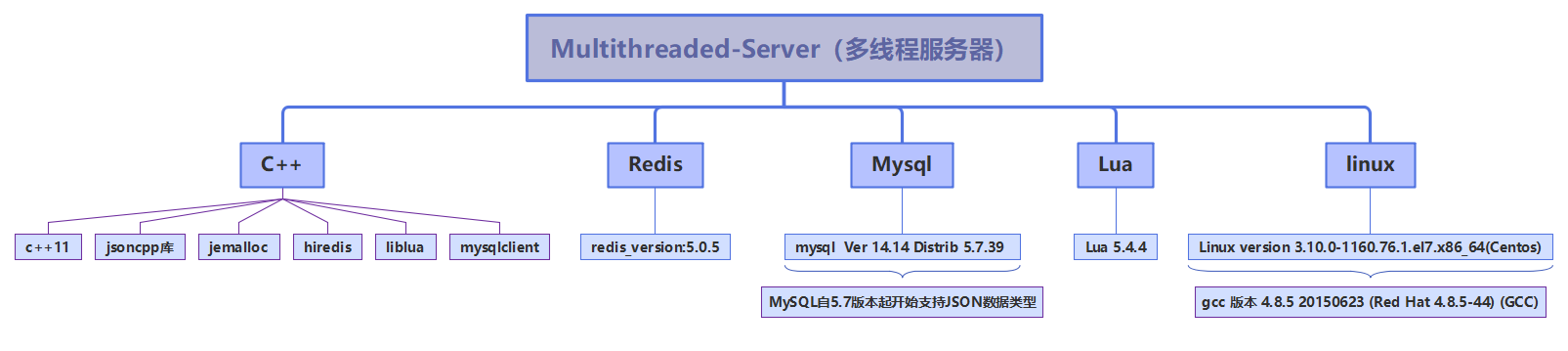
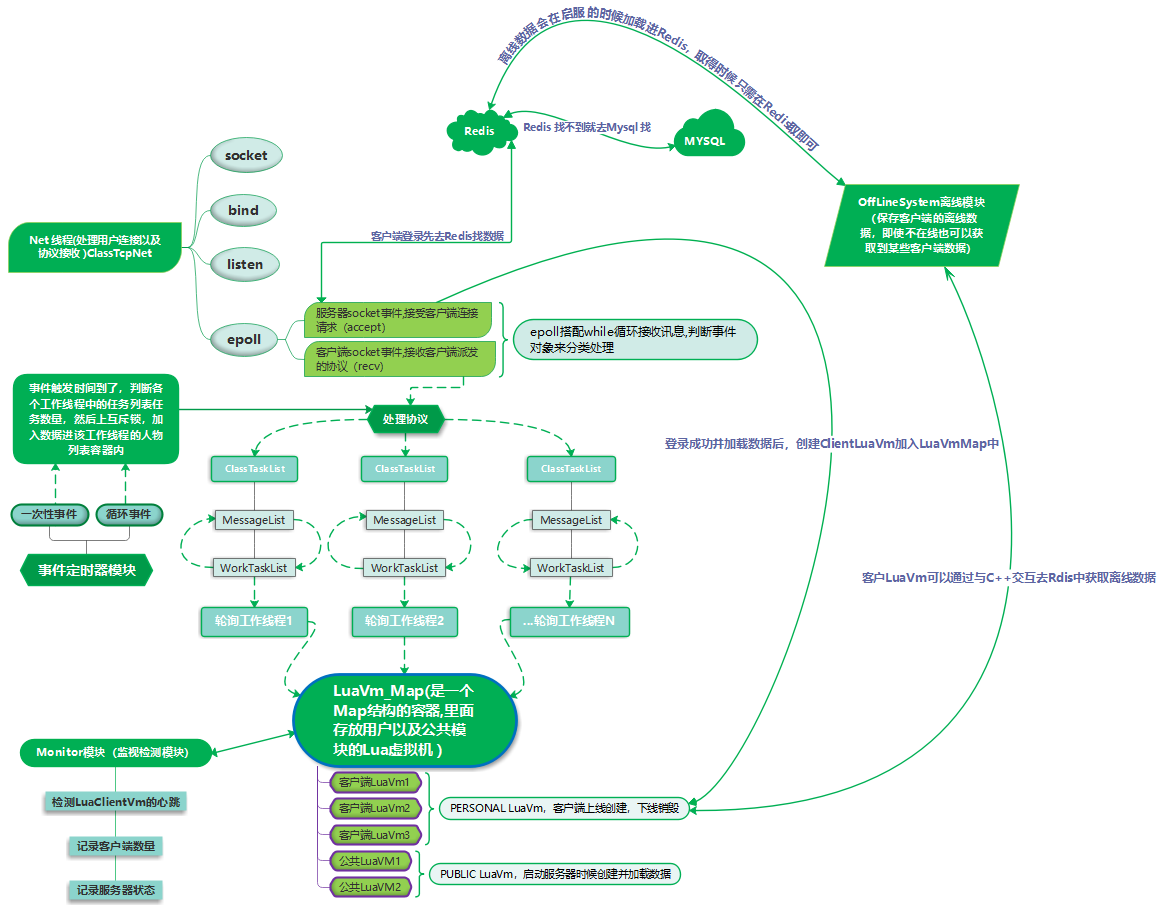
## **Multithreaded-Server**

## 技术栈以及环境：



## **组成结构：**



## **文件概述：**

1. **ClassDB :** 里面封装了Mysql 以及 Redis的类, 并提供了各自的数据库接口

项目使用Mysql来存储持久化数据, 但是由于Mysql存取数据实在硬盘或者磁盘上操, 因此, 存取的速度较慢, 而Redis用于存储使用较为频繁的数据到缓存中, 所以存取速度快, 所以用来存放一些变化频繁的数据。

由于Mysql数据的可持久化存储, 项目把数据在关服或者用户下面又或者通过定时推送来存放在Mysql中, 而其余时间,获取数据可以在Redis 中查找以及存放,若查找不到再从Mysql中加载进Redis中。这样可以提高服务器的效率, 而且把数据保存在Redis也可以不用担心崩服后数据的丢失。

1. **ClassPthreadMgr :** 该文档里定义了工作线程管理类, 以及任务池类

该项目中,我们使用多个工作线程来同时处理其各自任务池的任务列表里面的任务，提高处理业务的效率。

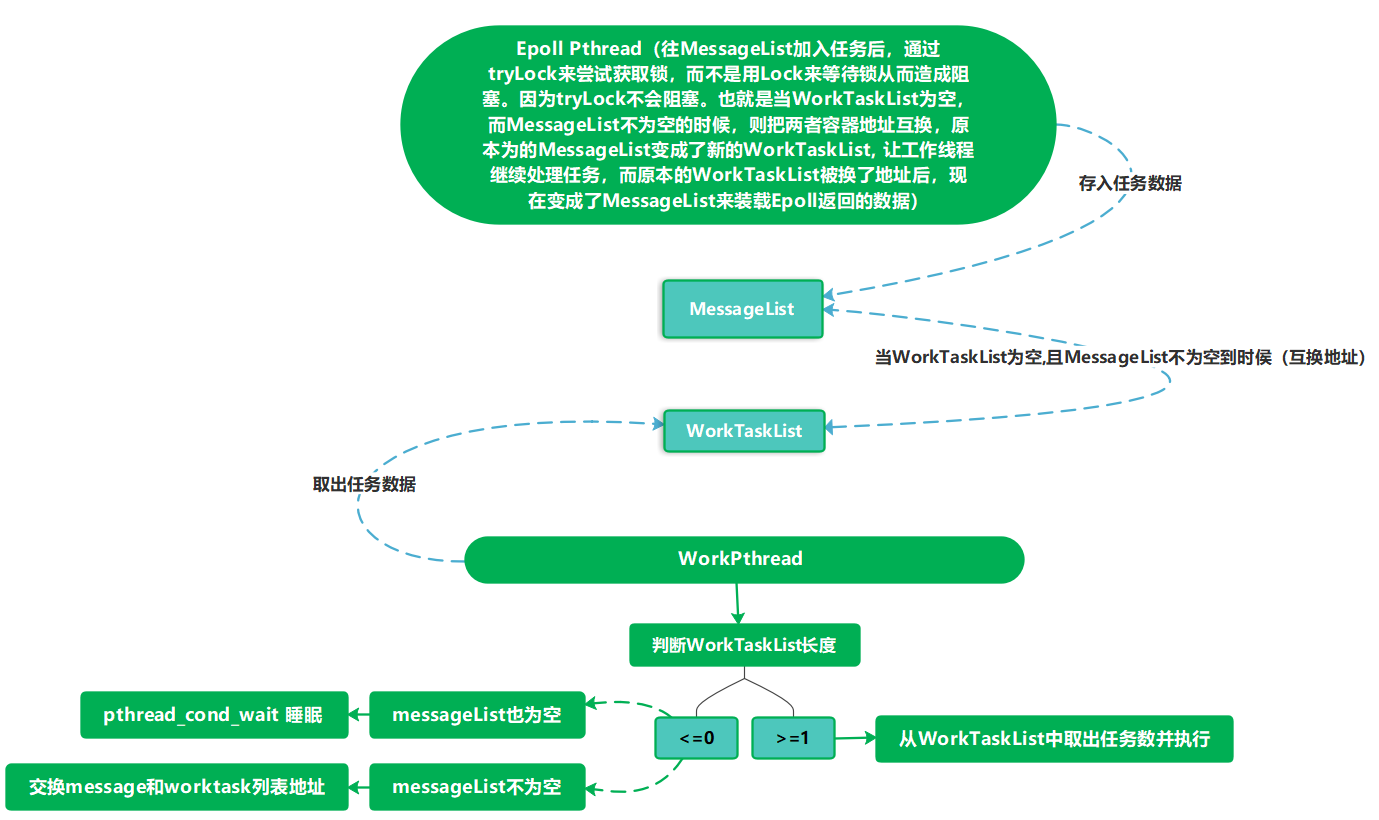
**工作线程和任务池关联思路：**由于是多线程的环境下, 假如用一个任务池存放任务, 则是多个工作线程来争夺一个任务池的资源,并进行删减和处理, 这样会存在数据的不一致性, 而且任务池列表用的结构是std::list,该数据结构非线程安全, 也就是说需要用到锁来确保同一时间只有一个工作线程来获取任务池的处理权。加了锁就意味着效率降低, 这样多线程的优势会大大减少, 所以最后采用了一个工作线程对应一个任务池, 这样一来，每个工作线程都有自己的任务池, 不需要进行加锁来争抢同一个任务池。

**任务池：**任务池是由两个List结构组成的, 一个用来接收Epoll返回的的任务数据, 另一个则是工作线程从中获取数据。这样做的理由是避免任务数列表加锁而降低效率, 因为Epoll线程返回的任务需要加入到任务列表中,而工作线程则需要从任务列表中取出数据, 这样的话存在容器同时存取的问题, 而上面说过了list结构为非线程安全, 所以只有一个任务列表是需要加上互斥锁的, 保证同一时间只有工作线程或者是Epoll线程来操作任务列表。但是这样加锁解锁太过频繁了,会降低服务器性能, 所以,我们可以分两个任务列表来分别**存放任务数据**和**取出任务数据。**

**遇到的问题：**

假如工作线程所取的任务列表为空了（任务处理完了）, 怎么从另一个接收了任务数据的列表中获取任务呢？

如下图：



（1）WorkTaskList为空的时候，需要判断MessageList是否也为空。假若两者为空，则直接睡眠（pthread\_cond\_wait会自动解锁再进行睡眠），降低CPU的使用率，避免线程一致占据资源。

我们把Epoll线程存放数据进入MessageList作为条件变量。条件变量是利用线程间共享的全局变量进行同步的一种机制，主要包括两个动作：一个线程等待"条件变量的条件成立"而挂起；另一个线程使"条件成立"（给出条件成立信号）。为了防止竞争，条件变量的使用总是和一个互斥锁结合在一起。

所以当Epoll放入任务的时候，就会触发pthread\_cond\_signal来唤醒正在睡眠的工作线程，假如多个工作线程都共同作用于一个变量条件,那么唤醒的时候则是多个工作线程之间相互争夺唤醒条件变量所关联的锁。

1. WorkTaskList不为空的时候，则直接交换地址，让空的容器与接收来自Epoll的数据，而装有任务数据的任务列表（来自Epoll线程存放任务的容器），则变成了工作线程所遍历的任务容器了。

总结：

1. 通过两个任务列表，使得解锁和上锁的频率降低，比起使用单个任务列表（加入和取走任务的时候都需要上锁）避免了每次都要操作锁才能进行任务的存取。
2. 当两个容器都为空的时候，工作线程会休眠，降低了CPU占用率，当有新的数据加入MessageList的时候才会唤醒沉睡的工作线程，避免了工作线程做无意义的循环。