# VCS设计方案

## 需求

希望封装一个简单的网络事件库，支持非阻塞和阻塞业务逻辑处理。开源的libevent和libev对于我们的场景来说还是太过于庞大和繁杂，ACE等等网络框架也一样。

## 模式

仿Proactor，linux操作系统没有提供异步网络io，所以无法实现真正的Proactor模式。在仿Proactor的过程中依据简单，易用和高性能的理论做了一下删减，比如不需要不停的注册感兴趣的事件，写操作完成不提供回调等等。

## 理念：简单，易用，高性能

### 简单

基于简单理念做的设计决策有

1. 单线程：没有并发问题，编程简单，逻辑清晰。
2. 只有3个实体：listener, handler, observer

### 易用

基于易用理念做的设计决策有

1. 由事件库负责网络读写操作：业务层只需从buffer里边读数据和告诉事件库我需要写数据即可。
2. 取消通知无用事件：比如只有onData, 没有onWriteCompleted等等事件），比proactor的封装更进一步。（其实linux不支持真正的proactor因为linux没有提供异步系统io
3. 提供工具和功能模块：helper，doctor，worker等等
4. 可插拔设计：doctor，worker功能模块都是基于可以插拔接口设计的。
5. 接口的一致性：所有实体操作统一使用add和remove

### 高性能

基于高性能理念做的设计决策有

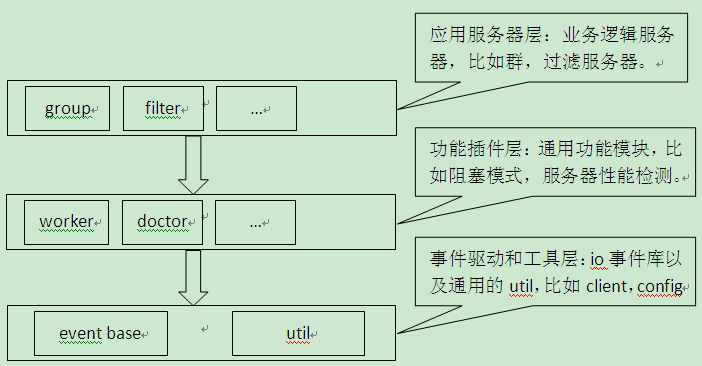
1. 避免数据复制：参数传递，类实现等等
2. 网络事件采取边缘触发：减少网络事件通知

## 设计过程中的纠结点

### 内存管理

这是在做设计时纠结了很久的事情，事件库负责内存管理可以提高一定的易用性，但给事件库带来了一定的复杂性和约束性（比如用于主动连接并要保持长链的handler实体来说是希望一次创建重复实现的），也有一定的不对性与职责混乱（一般来说谁申请的内存谁负责释放）。如果事件库不负责管理内存又会出现自己销毁自己的情况（比如被动连接申请的handler一般是在listener中申请，然后就没有其它合适的地方释放了，只能在handler->onClosed()中进行释放）。最后决定使事件库支持释放内存这个功能，但由业务决定是否需要事件库释放还是由业务来释放，只需在onClosed中返回true则由事件库释放该对象，如果返回false则由业务自己释放。内存分配还是由业务来处理。

## VCS整体框架



## event base框架

event base

listener

handler

observer

## Worker模块

### 需实现角色

#### 基类

WorkerTask

任务基类

WorkerOutcome

工作结果基类

WorkerEncoder

编码器基类，对输出给客户端的WorkerOutcome进行encode成二进制流数据

WorkerDecoder

解码器基类，对客户端的输入的二进制流数据进行decode成WorkerTask

WorkerRouter

路由器基类，选择WorkerTask到WorkerThread 之间的路由

WorkerService

业务服务基类，具体业务服务，承载业务逻辑的地方

WorkerConfig

配置基类， 框架与用户实现的接口类，用于配置WorkerCodec, WorkerService, WorkerRouter

### 内部角色

#### 事件相关

WorkerListener

工作模式下的监听器，绑定客户端fd连接到WorkerClientHandler

WorkerClientHandler

每个客户端对应一个WorkerClientHandler绑定到客户端的fd，负责解析客户端的二进制输入，封装WorkerTask, 分发WorkerTask

WorkerOutcomeHandler

每个线程对应一个WorkerOutcomeHandler绑定到outcome pipe的read fd，负责获取WorkerOutcome，然后对其进行encode，分发到对应的WorkerClientHandler,的output，然后写数据到socket

WorkerObserver

负责监控worker健康程度和退出时清理worker

#### Woker相关

WorkerThread

负责管理WorkerTask的读管道fd，WorkerOutcome的写入fd；

线程创建，线程轮询和管理WorkerService

WorkerManager

统一管理WorkerThread，WorkerTask队列，WorkerOutcome队列，

### worker框架

…

Worker thread

client

client

Main thread

Event base

ClientHandler

…

ClientHandler

OutcomeHandler

Task Router

Worker thread

Task

Pipe

Task

queue

Outcome

Pipe

Outcome

queue

Service

…

…

### worker细节

1. 主线程写阻塞的task pipe效率是非常高的，可以认为基本是没有阻塞和延迟的（否则主线程会阻塞，导致处理性能大大下降）
2. 主线程读 outcome pipe 是非阻塞的，有数据可读则一直读，读到没有为止，主线程的epoll工作于ET模式

## 参考资料

<http://www.artima.com/articles/io_design_patterns.html>

<http://www.artima.com/articles/io_design_patterns2.html>

<http://www.artima.com/articles/io_design_patterns3.html>