Charles的技术博客

自己动手写分布式KV存储引擎（一）： 设计和实现网络框架

2016-09-24 | 分布式

介绍

之前写过一篇博文，描述了本人学习分布式系统的思路(链接)。自己动手写分布式KV存储引擎系列文章的目标是记录基于LevelDB(RockDB)构建一个分布式KV存储引擎实现过程，算是对之前学习思路的实践。初步设想，此系列文章会包含以下主题：

- 如何设计和实现网络框架
- 如何设计和实现RPC库
- 分析LevelDB和RockDB的设计和实现原理
- 如何理解和实现raft/paxos算法
- 如何基于raft/paxos，构建强一致的分布式KV存储引擎
- 如何对分布式KV存储提供事务功能
- 如何对分布式KV存储系统优化性能
- 等等

此系列文章对应的源码放在DSTORE下。

本文为此系列第一篇文章，主要是关于如何设计和实现一个基本的网络框架，全文的组织结构如下：

- 网络框架的要点
- DSTORE网络框架的设计与实现

网络框架的要点

使用TCP还是UDP?

由于TCP相对于UDP来讲，可靠性高很多，保证包的按序达到，这对于高可靠的存储系统来讲是十分必要的，因此，本文的网络框架将基于TCP来实现。

操作系统的选择

由于目前Linux是服务端编程中主流的操作系统平台，因此，本文的网络框架将基于Linux平台，且为X86_64体系架构。

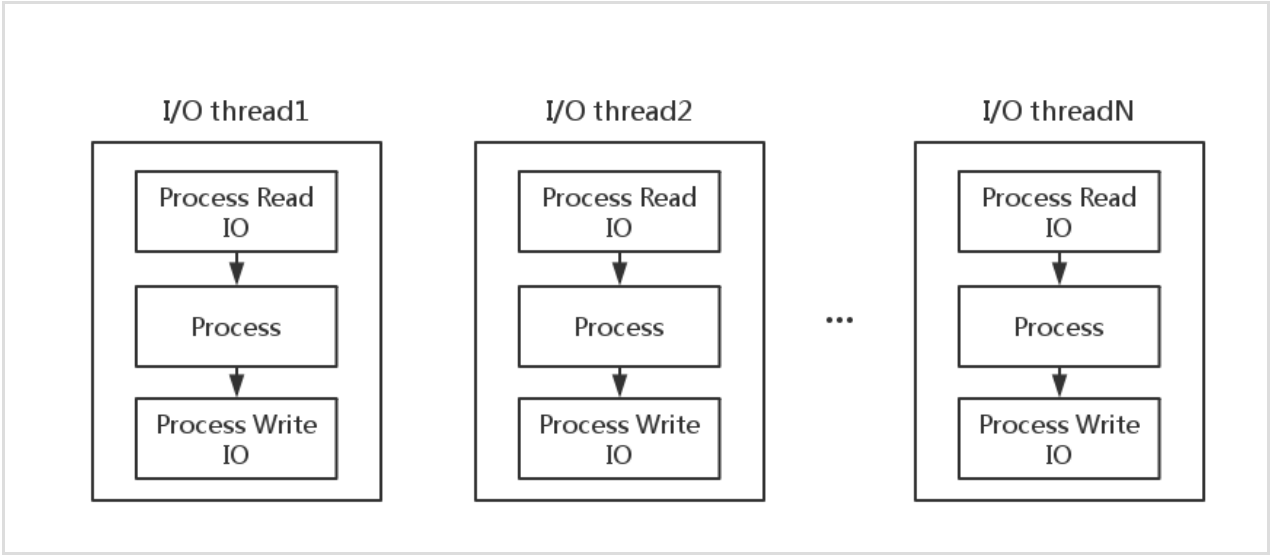
Reactor VS Proactor

一般Reactor模型基于I/O多路复用来实现，Linux平台提供select,epoll等接口，而Proactor模型一般基于异步I/O来实现，目前Linux系统对这块支持不太好，因此，本文的网络框架将基于Reactor来实现。

线程模型

两种常见的线程模型，一是IO线程和工作线程共用相同线程，二是IO线程和工作线程分开。

I/O线程和工作线程共用



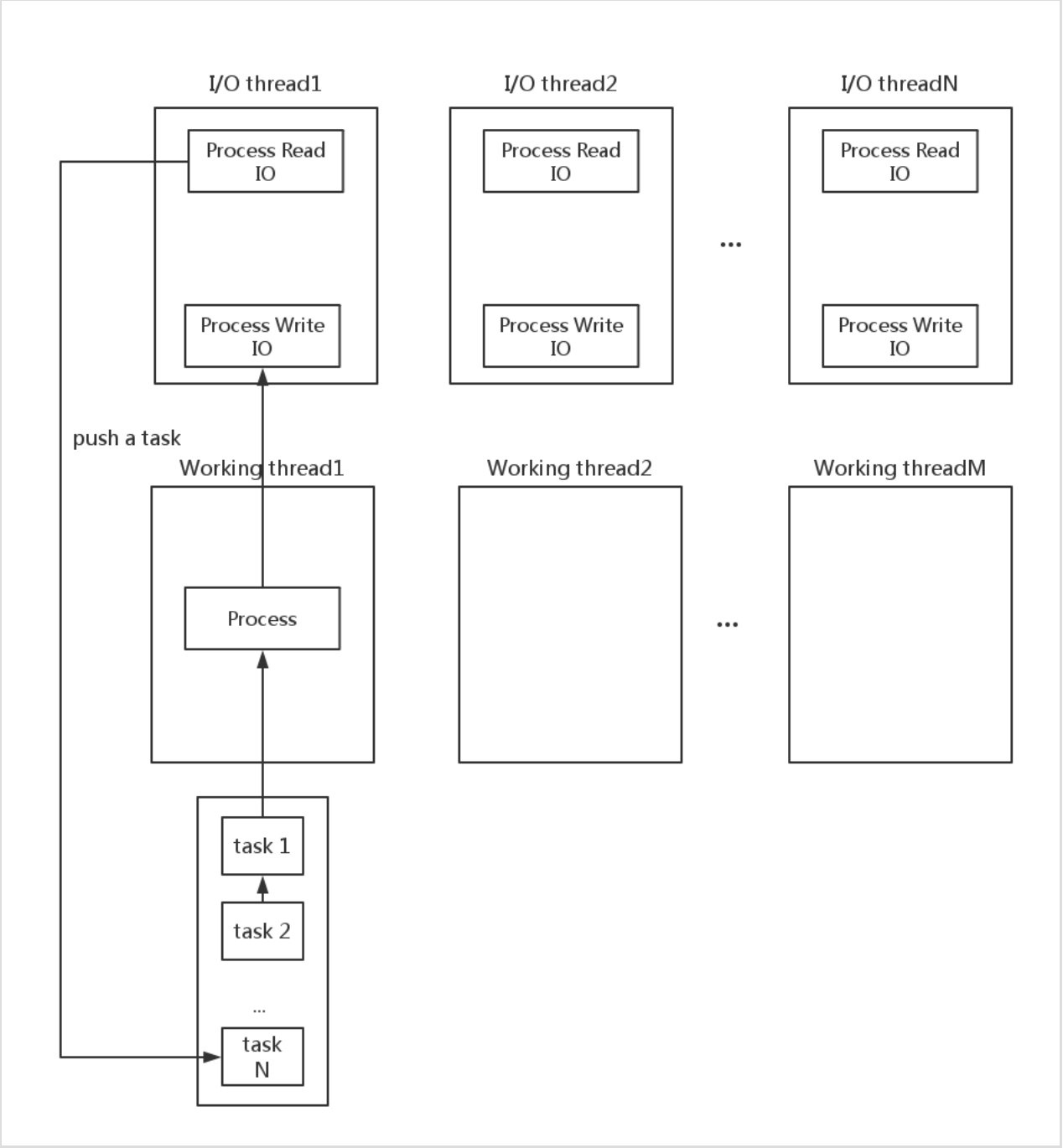
如上图，I/O线程和工作线程共用的线程模型中，实际上是没有专门的工作线程的，I/O线程不仅需要负责处理I/O，还需要真正地处理请求，计算结果。一般典型的处理流程为

- Process Read I/O: 处理读I/O
- Process: 解析请求，计算结果
- Process Write I/O： 处理写I/O，把计算结果返回给客户端

这种线程模型的特点是

- 处理流程相对简单，解析好请求后就能直接在同一线程处理，省去了线程切换的开销，非常适合Process耗费时间较小的请求
- 由于Process过程需要耗费时间，对于大任务，可能时间较长，会影响其他请求的处理

I/O线程和工作线程独立



如上图，在I/O线程和工作线程独立的线程模型中，有专门的工作线程来处理请求，计算结果，I/O线程仅仅需要做读写数据相关的操作。在这种线程模型下，整个流程为

- Process Read I/O:处理读数据，然后解析请求，生成任务，推送到工作线程的队列中，然

后以异步事件方式通知工作线程处理

- Process: 工作线程接收到异步事件后，从其工作队列中拿出任务，依次处理，处理完成后，生成结果，放到I/O线程的队列中，然后以异步事件方式通知I/O线程处理
- Process Write I/O：I/O线程收到通知后，依次处理写数据请求

这种线程模型的特点是

- I/O和计算分开处理，会引入线程切换开销，比较适合Process耗费时间长的任务请求
- 对于小任务请求不适合，大量时间耗费在线程切换开销

对于存储系统，一般计算需求较小，因此采用第一种线程模型。

I/O线程模型

选定好在I/O线程中处理任务之后，又需要确定I/O线程具体是如何分工的，一般有三种方式

- 单线程做accept和I/O
- 单读一个线程accept,其他线程I/O
- 多线程，每个线程accept并且Ｉ/O

主要从两个角度考虑这几个I/O模型的选择

- 连接建立的频繁与否
- I/O的吞吐量高与否

对于第一种模型，比较适合连接建立不频繁的场景，在CPU使用不高的情况下，单线程也可以做到打满网络带宽

对于第二种模型，比较适合连接建立不频繁的场景，可以通过增加I/O线程的数量，来提升I/O的吞吐量

对于第三种模型，比较适合连接建立频繁的场景，可以通过增加线程的数量，来提升连接建立的速度和I/O的吞吐量

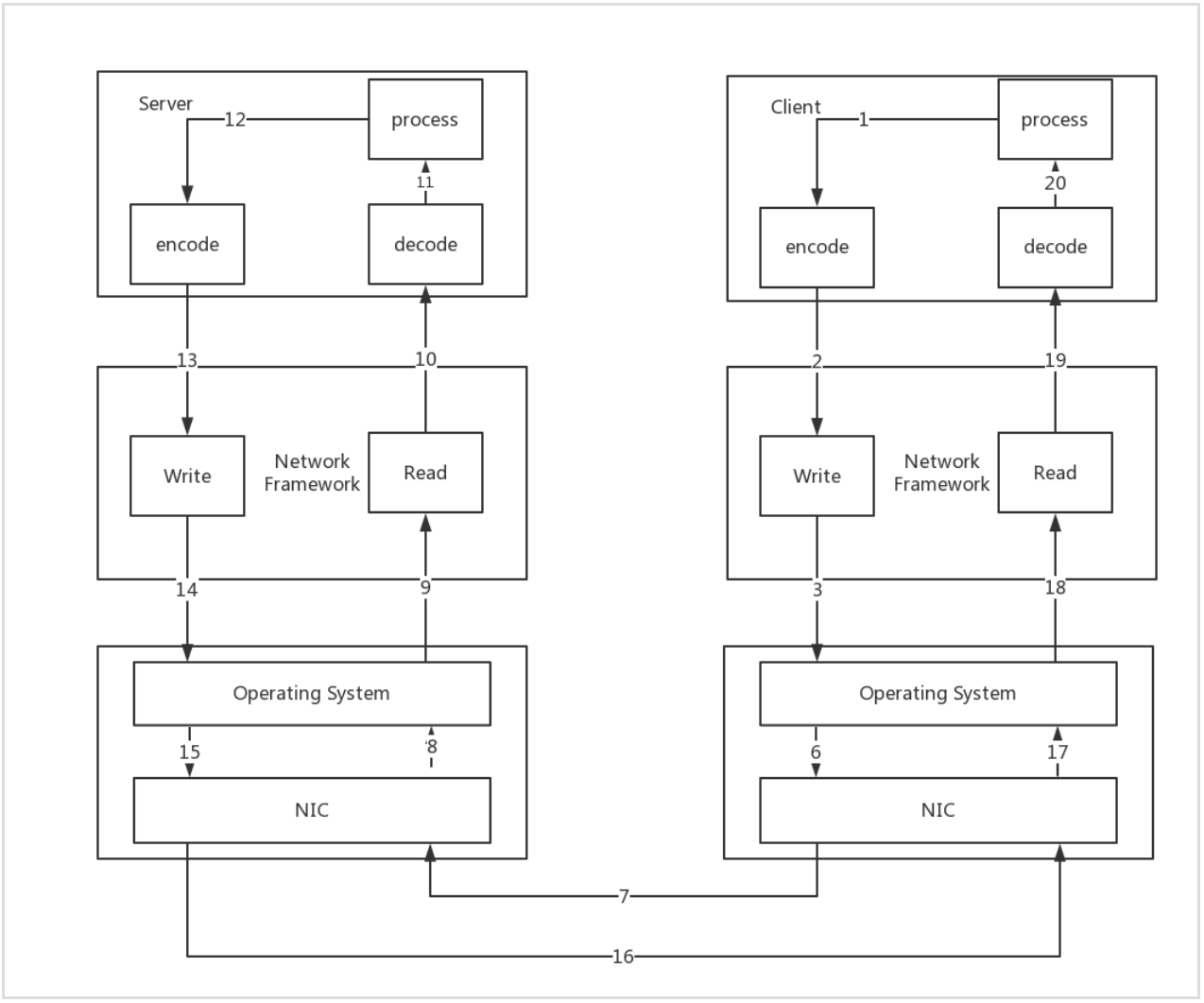
对于存储系统调用者来讲，一般会使用连接池，因此，存储系统一般不会频繁的建立连接；并且一般存储系统对I/O吞吐量需要较高，因此，选择第一种和第二种模型。本文中暂时采用第一种模型，如果在第一种模型不能提供足够的I/O带宽的情况下，考虑采用第二种模型。

DSTORE网络框架设计与实现

网络框架需要处理的事件

在描述DSTORE网络框架设计之前，先分析网络框架需要处理的事件

网络请求处理流程



从上面的处理流程可以看出，对于Client和Server，它们需要关注的事情包括

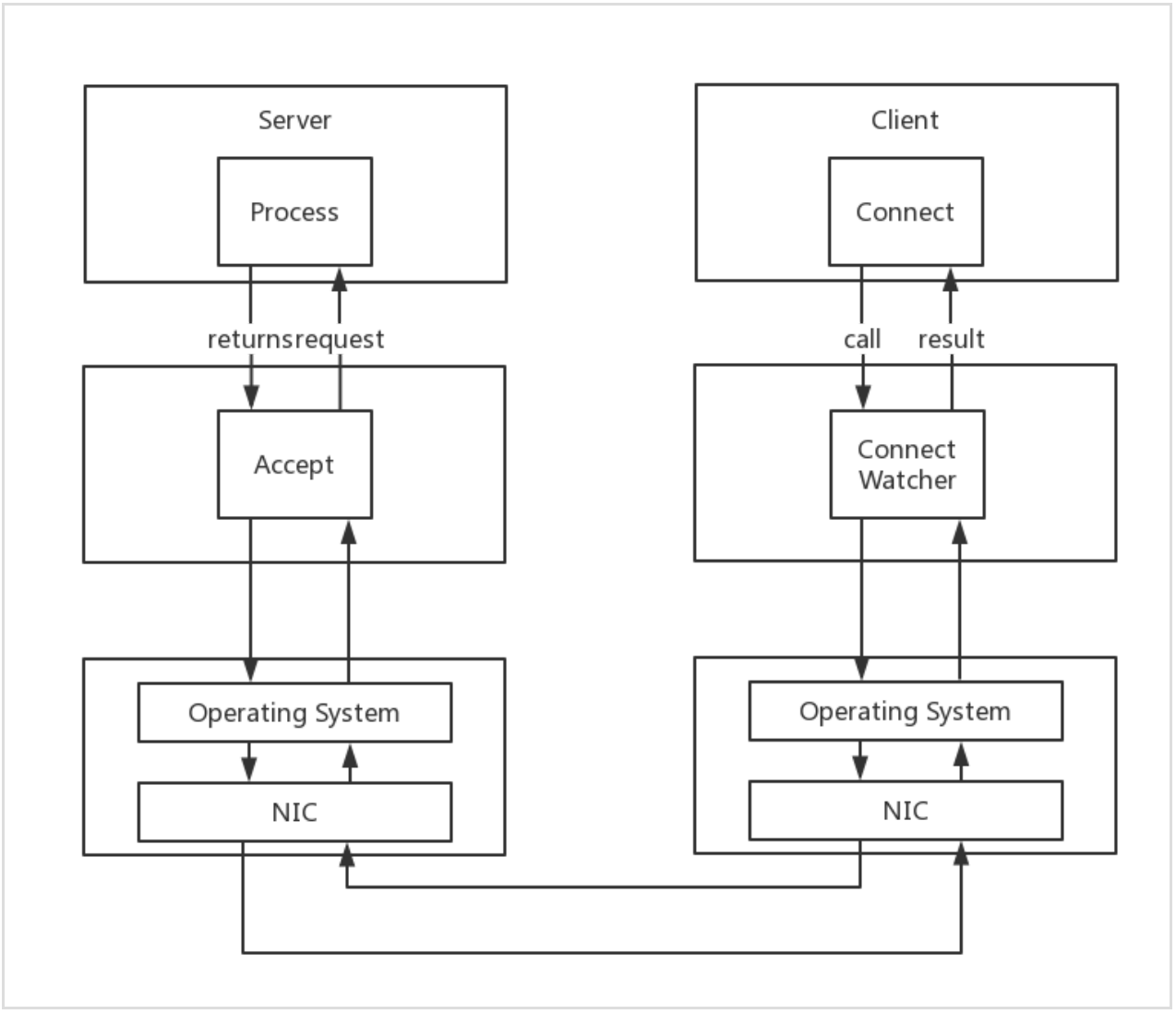
- 消息的解码
- 对消息的处理，生成响应
- 把响应结果根据格式编码

对于网络框架层，需要关注的是

- 读事件及读数据
- 写事件及写数据

网络框架除了需要关注读写事件及读写数据外，还需要处理连接的建立和断开。

网络连接处理流程



网络连接处理流程和网络处理请求流程不太一样，在于Client和Server的处理与网络请求处理的流程不太一致，其流程如下

1. Client发起网络框架提供的API connect来请求建立连接
2. Client端的网络框架记录此事件，并加入监听
3. Client端的操作系统把包发送给通过网卡发送到网络
4. Server端的操作系统读取网卡数据，通知网络框架
5. Server端网络框架调用accept建立连接，并调用Server的建立连接的接口
6. Server端的accept调用会产生一个回包
7. Client端的操作系统收到后，触发网络框架的事件
8. Client端网络框架通知Client连接已建立

其中步骤5中accept返回后，其后半步骤与步骤6是并发的，并没有严格的顺序。

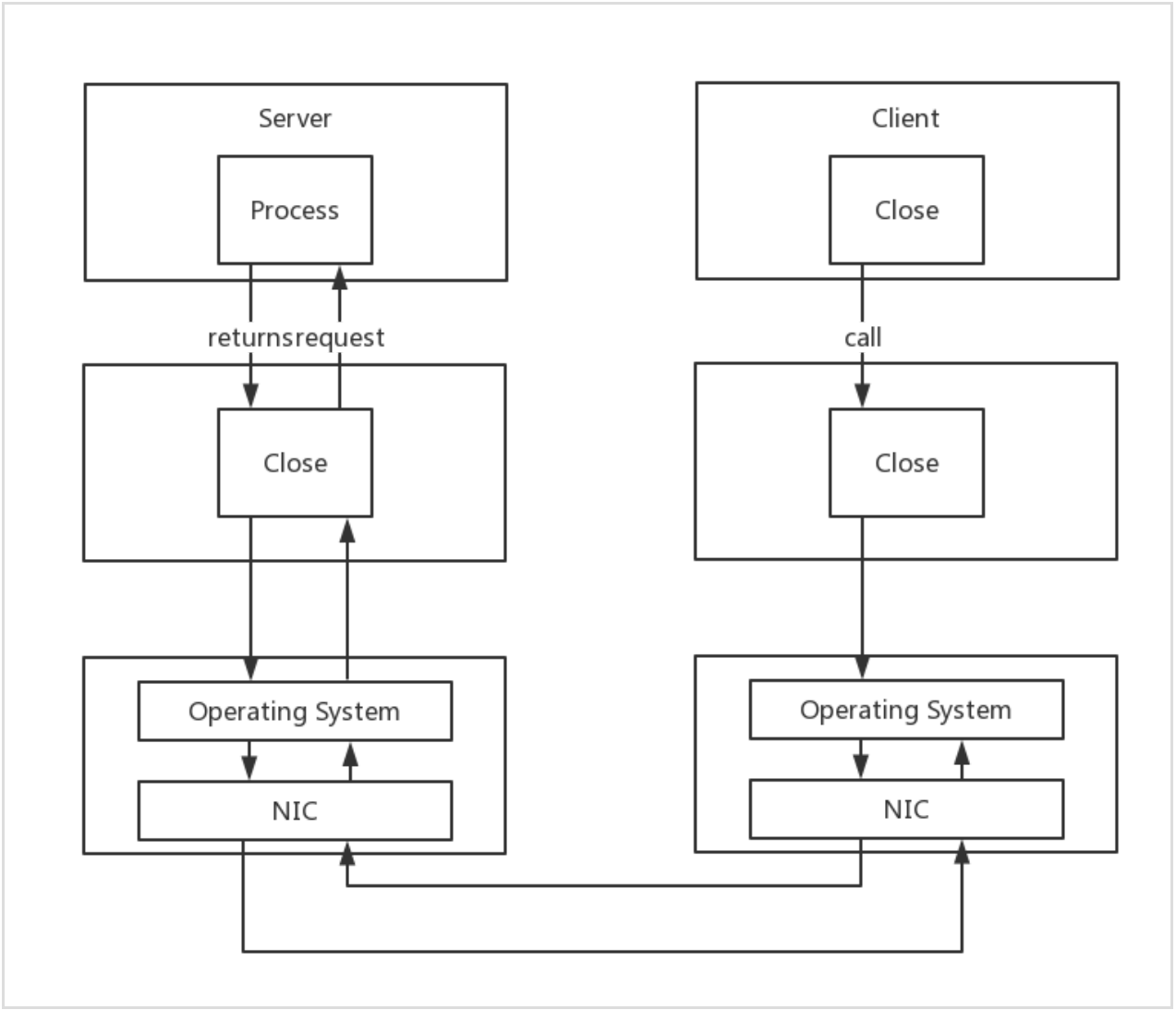
从上述流程可以看出，对于网络连接的建立，Server端和Client端处理调用网络框架的API之外，几乎不需要额外的处理。

而对于网络框架来讲，在Client和Server端的处理流程是不同的，分别是

- 在Client端，网络框架需要调用操作系统提供的connect接口，并且，监听connect完成的事件
- 在Server端，网络框架需要调用操作系统提供的accept接口，并且，需要触发server的新建连接的接口(这个连接维护也可以在网络框架中处理)

网络请求完成后，需要正确地关闭连接，其处理流程如下。

网络连接关闭流程



如上图，对于Client端，需要处理的主要是调用网络框架的close API；对于Server端，则需要处

理其上维护的连接结构体等等。

对于网络框架，需要处理的是

- Client端调用close
- Server端需要监听close，然后触发Server端处理(也可以在网络框架中处理)

网络框架需要处理事件

通过上面的分析，可以总结出网络框架应该处理以下事件

- 处理连接的建立
- 处理连接的关闭
- 处理读事件和读数据
- 处理写事件和写数据

备注：此文写作时，Client端的网络框架尚未实现。

DSTORE网络框架设计

本网络框架的目标是使得Server端和Client端编程时，只需要以下事件

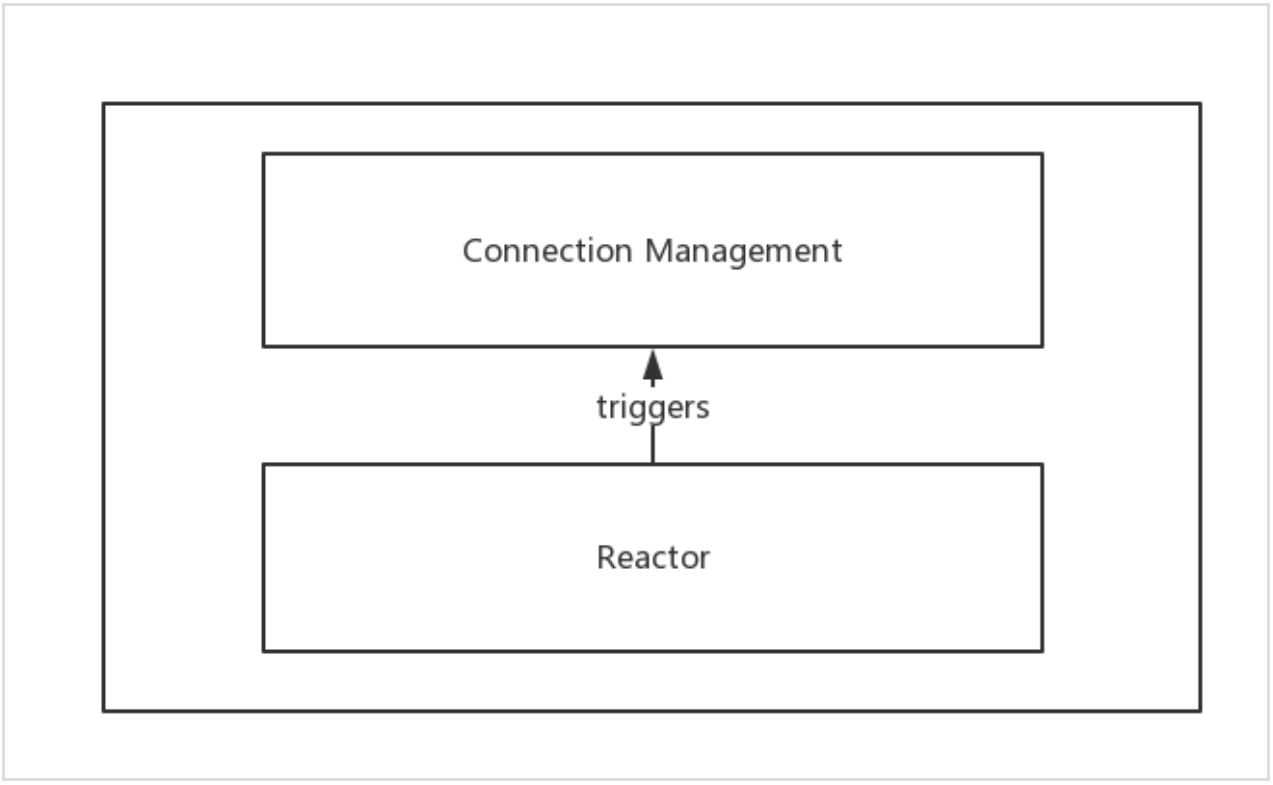
Server端需要关注的事件

- 请求编码
- 请求解码
- 处理

Client端需要关注的事件

- 请求编码
- 请求解码
- 处理
- 主动调用close关闭连接

其他的一律由网络框架部分来处理，网络框架的整体框架如下

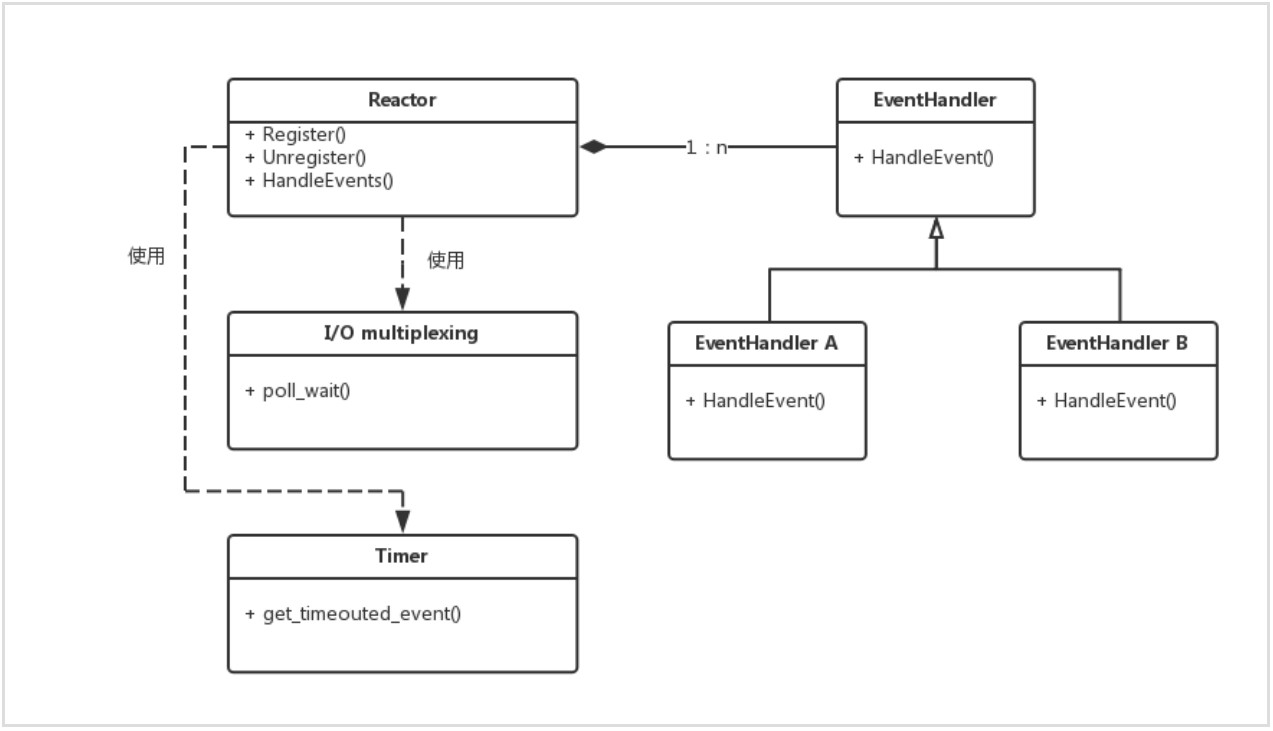


网框框架整体包含两部分：

- Reactor负责监听读写事件
- Connection Management负责根据读写事件，来建立连接，关闭连接，读数据和写数据

Reactor

一个reactor模式如下图：



Reactor中组件包括Reactor，EventHandler，I/O multiplexing和Timer

- EventHandler是事件的接口，一般分为I/O事件、定时器事件等等
- I/O multiplexing即I/O多路复用，linux中一般采用epoll接口
- Timer是管理定时器的类，主要负责注册事件、获取超时事件列表等等，一般由网络框架开发者实现
- Reactor中使用了I/O multiplexing和Timer，有EventHandler注册时，会调用相应的接口。Reactor的HandleEvents中需要先调用I/O multiplexing和Timer的接口，获取已就绪好的事件，最终调用每个EventHandler的HandleEvent接口来处理事件

本文写作时，DSTORE的网络框架还没有实现定时器相关的功能。

Connection Management

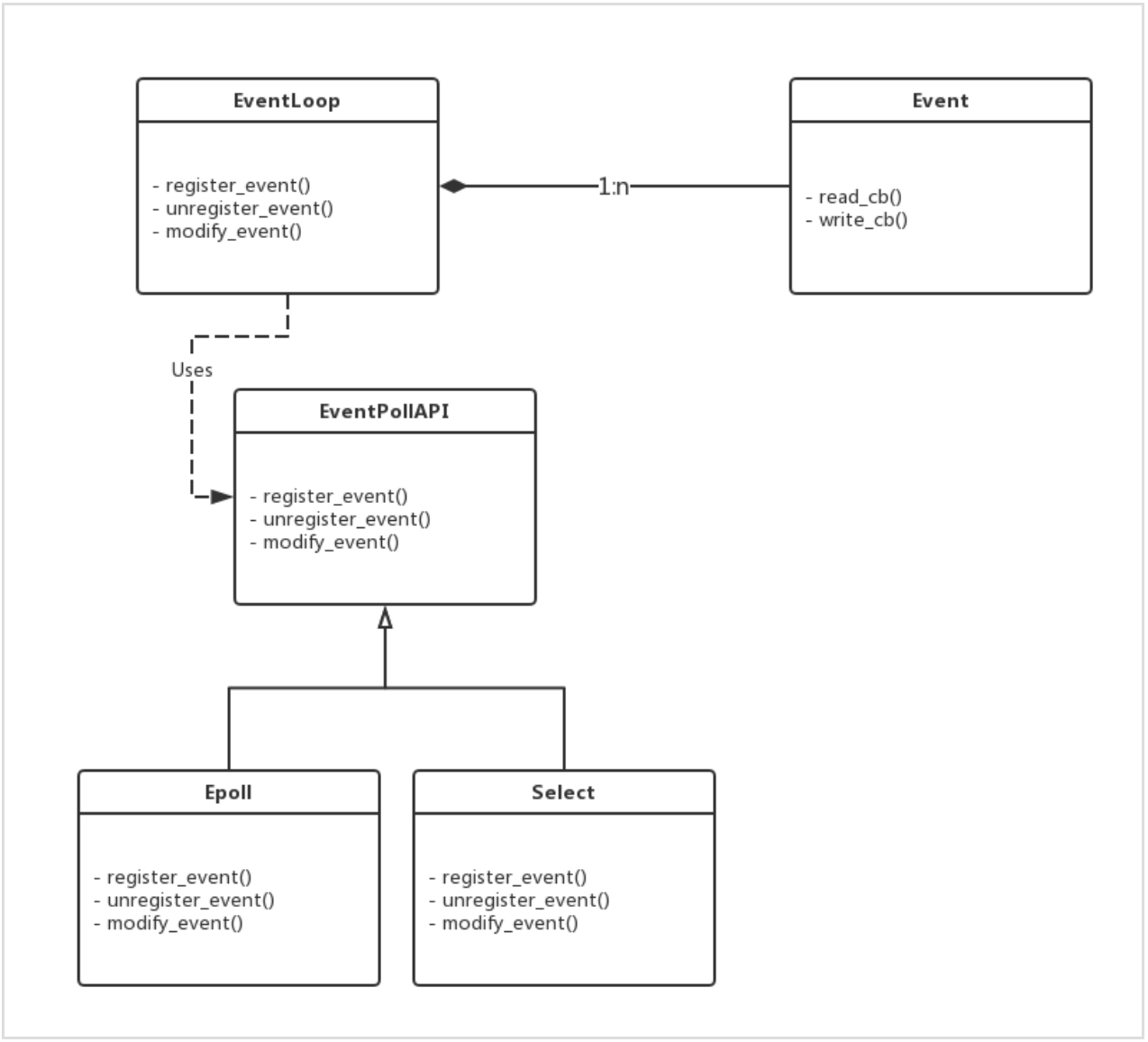
Connection Management主要需要处理如下

- on_read：读事件触发后，读取缓冲区的数据
- on_write：缓冲区空闲后，写入应用所请求要写入的数据
- on_connect：连接建立后，维护连接所必须的结构体和资源
- on_close：连接关闭后，清理连接所必须的结构体和资源

DSTORE网络框架实现

本部分主要描述Reactor和Connection Management部分的实现。

Reactor实现

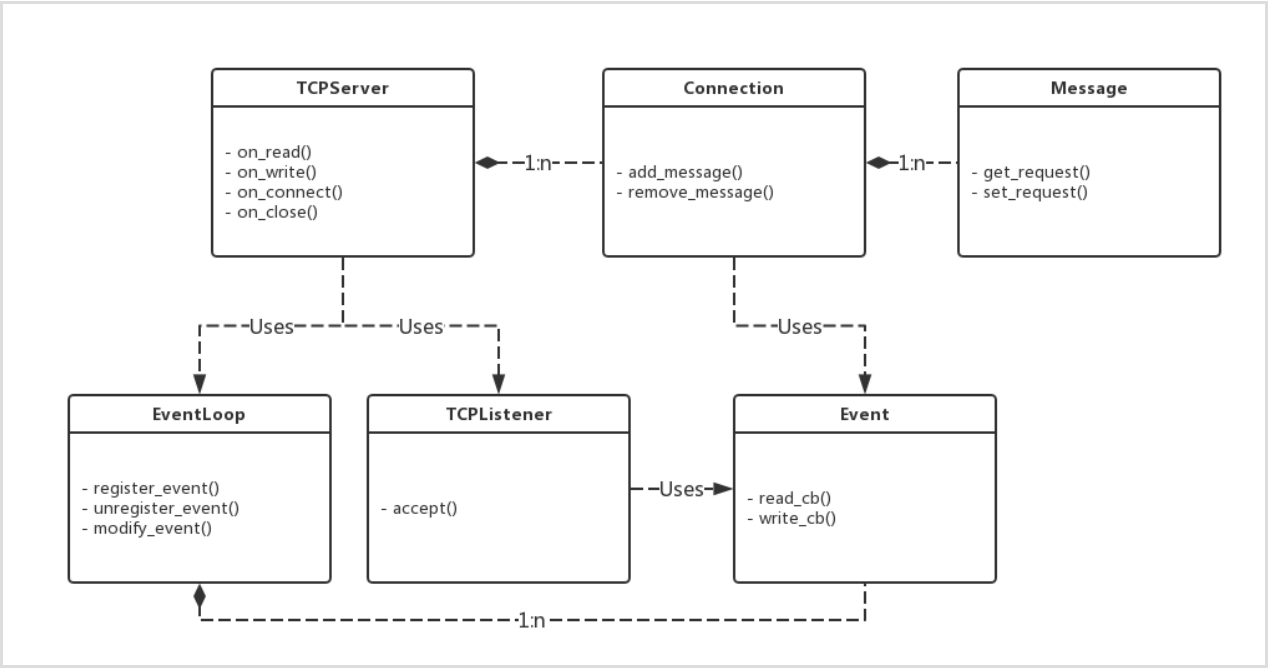


- EventLoop：对应于Reactor，调用epoll或select的接口
- Event：每个文件描述符对应一个事件，read_cb处理读事件,write_cb处理写事件
- EventPollAPI：I/O多路复用的接口，可以有epoll，select，poll和kqueue等多种实现，本文写作时只封装了epoll的接口

源码链接

- EventLoop and Event
- Epoll

Connection Management实现



TCPServer

TCPServer中维护了所有连接的hashmap，用来保存Client端和Server端所有建立的连接情况。

实现了连接管理中的四种功能：

- 读数据
- 写数据
- 连接建立
- 连接关闭

其中读数据和写数据依赖于EventLoop中每个Event的读事件和写事件的触发

源码

TCPListener

TCPListener是用来处理accept相关的事件的，包括服务端socket从创建到listen的全过程，以及accept调用的支持。TCPListener调用accept之后，会触发TCPServer中的on_connect事件。

源码

Connection

Connection代表了Client与Server端的连接，每个连接上可能会收到客户端的多个请求，其使用

链表来维护尚未处理的请求。

源码

Message

Message代表来自Client的一个完整的消息，Server根据消息中的指定的操作，来进行相应的处理。

源码

使用例子

一个简单的使用例子请参照simple_packet_test.cpp

PS:
本博客更新会在第一时间推送到微信公众号，欢迎大家关注。

参考文献

- DSTORE源码
- libev设计与实现
- libeasy实现原理

#C++ #分布式 #网络编程

❮ gfs原理 raft原理（一）：选主与日志复制 ❯

被顶起来的评论

工控资料窝

看起来挺牛逼的样子

2016年9月29日 回复 顶(1) 转发

wongxingjun

谢导牛逼啊！！

2016年9月29日 回复 顶(1) 转发

13 条评论

程序员的难言之隐

你这是要逆天啊!

2016年9月26日 回复 顶 转发

柏拉

你好，我按照您的github上的how to use 去编译时，出现了make错误。我想请教一下您的编译的具体环境是什么？

2016年9月26日 回复 顶 转发

Charles0429

回复 柏拉: ➜ ~ uname -r
4.4.0-38-generic
➜ ~ uname -a
Linux Charles-PC 4.4.0-38-generic #57-Ubuntu SMP Tue Sep 6 15:42:33 UTC 2016 x86_64 x86_64 x86_64 GNU/Linux
➜ ~ gcc --version
gcc (Ubuntu 5.4.0-6ubuntu1~16.04.2) 5.4.0 20160609

2016年9月26日 回复 顶 转发

柏拉

感谢您的回复，我换成Ubuntu的环境后就可以了。

2016年9月26日 回复 顶 转发

Charles0429

回复 柏拉: 之前你用的什么环境呢？

2016年9月26日 回复 顶 转发

Hello_Code

赞

2016年9月28日 回复 顶 转发

wongxingjun

谢导牛逼啊！！ 

2016年9月29日 回复 顶(1) 转发

工控资料窝

看起来挺牛逼的样子

2016年9月29日 回复 顶(1) 转发

柏拉

回复 Charles0429: 抱歉，当时没有看到您的回复，我用的是macbook，gcc是默认的LLVM架构，当时没注意。

2016年10月6日 回复 顶 转发

王彪

https://github.com/fenghui2013/black_hole_c 一个实现reactor模式的网络框架 c+lua 思路类似 互相学习

2016年10月19日 回复 顶 转发

Benedict

勘误：单读一个线程…… 应为 “单独”

2016年10月31日 回复 顶 转发

Benedict

写得真不错

2016年10月31日 回复 顶 转发

Benedict

博主用了多久扫盲相关知识树的呀，求书单～

2016年10月31日 回复 顶 转发