**Ceph网络通信**

1. **Socket技术**

Socket起源于Unix，而Unix/linux基本哲学之一就是“一切皆文件”，都可以用“打开open -> 读写write/read -> 关闭close”模式来操作。Socket也是按照这个思想实现的，而我们工作中主要会用到它的以下接口：

* int Socket(int domain, int type, int protocol)

这个函数创建一个Socket描述符，唯一标识一个Socket，对应于普通文件的打开接口。

* int bind(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen)

该函数把一个具体的地址绑定到相应的Socket描述符上。通常服务器在启动的时候都会绑定一个众所周知的地址（如ip地址+端口号），用于提供服务，客户可以通过它来接连服务器；而客户端就不用指定，有系统自动分配一个端口号和自身的ip地址组合。

* int listen(int sockfd, int backlog)

如果作为一个服务器，在调用Socket()、bind()之后就会调用listen()来监听这个Socket。Socket()函数创建的Socket默认是一个主动类型的，listen函数将Socket变为被动类型的，等待客户的连接请求。

* int connect(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen)

客户端这时调用connect()发出连接请求，服务器端就会接收到这个请求。

* int accept(int sockfd, struct sockaddr \*addr, socklen\_t \*addrlen)

TCP服务器监听到这个请求之后，就会调用accept()函数取接收请求，这样连接就建立好了。之后就可以开始网络I/O操作了，即类同于普通文件的读写I/O操作。

以上这些接口，就可以用于实现客户端和服务器端的Socket连接的建立了。之后就可以通过::send()和::recv()接口从Socket中读取数据了。

Ceph网络通信是基于Socket技术实现的，所以要想搞清楚网络部分，必须先了解Socket的工作机制，这里介绍一些基本的内容，要深入理解Socket请参阅《Linux网络编程》一书和《tcp/ip详解》。

1. **代码结构**

在源代码src/msg 里实现了ceph 的网络通信模块。在msg目录下，定义了网络通信的抽象接口。

msg/Message.cc

msg/Message.h 定义了message

msg/Connection.h 定义了connection

msg/Dispatcher.h 定义了Dispatcher

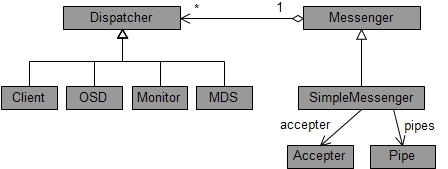
msg/Messenger.cc

msg/Messenger.h 定义了Messenger

msg/msg\_types.cc

msg/msg\_types.h 定义了消息的类型

1. **Ceph网络通信架构**



总体上，Ceph的消息处理框架是发布者订阅者的设计结构。Messenger担当发布者的角色，Dispatcher担当订阅者的角色。Messenger将接收到的消息通知给已注册的Dispatcher，由Dispatcher完成具体的消息处理。

主要包括Accepter、Pipe、 Messenge和Connection这几个实体，这些还要和具体的dispatcher结合起来。

1. **各模块作用**

在服务端，SimpleMessenger通过Accepter实例监听端口，接收来自客户端的连接。Accepter接受客户端的连接后，为该连接创建一个Pipe实例。Pipe实例负责具体消息的接收和发送，一个Pipe实例包含一个读线程和一个写线程。读线程读取到消息后，有三种分发消息的方法：

1. 快速分发，直接在Pipe的读线程中处理掉消息。可快速分发的消息在Dispatcher的ms\_can\_fast\_dispatch中注册。
2. 正常分发，将消息放入DispatchQueue，由单独的线程按照消息的优先级从高到低进行分发处理。需要注意的是，属于同个SimpleMessenger实例的Pipe间使用同个DispatchQueue。
3. 延迟分发，为消息随机设置延迟时间，定时时间到时由单独的线程走快速分发或正常分发的流程分发消息。

4.1 Accepter

单说Socket部分，Ceph将其分拆成了两个部分，Socket连接的维护（创建、绑定、监听、删除）主要在Accepter.cc文件中实现，以及I/O部分主要实现在了Pipe.cc中。用一个公式表示就是：

Accepter ＋ Pipe ＝ Socket

Accepter类继承了Thread类，本身是一个线程，主要用于server端的端口监听，接受连接。

4.2 Pipe

类Pipe 是PipeConnection的具体实现类。其实现了两个端口之间类似管道的通信接口。对于每一个pipe，内部有一个Reader线程 和 一个Writer 线程，分别用来处理有关这个Pipe的消息接收和发送。线程DelayedDelivery用于故障注入测试使用。

Pipe的写线程将消息放入out\_q队列，按照消息的优先级从高到低发送消息。另外，消息(Message)中携带了seq序列号，Pipe使用in\_seq和out\_seq记录它接收到和发送出去的消息的序列号。发送消息时，Pipe用out\_seq设置消息的序列号；接收消息时，通过比较消息的序列号和in\_seq来确定消息是否为旧消息，如果为旧消息则丢弃，否则使用消息的序列号更新in\_seq。

类Pipe的数据结构介绍如下：

SimpleMessenger \*msgr; // msgr的指针

uint64\_t conn\_id; //分配给Pipe 自己唯一的id

char \*recv\_buf; //接收缓存区

int recv\_max\_prefetch; //接收缓冲区一次预期的最大值

int recv\_ofs; //接收的偏移量

int recv\_len; //接收的长度

int sd; //pipe 对应的 socked fd

struct iovec msgvec[IOV\_MAX]; //发送消息的 iovec 结构

int port; //连接端口

int peer\_type; //链接对方的类型

entity\_addr\_t peer\_addr; //对方地址

Messenger::Policy policy; //策略

Mutex pipe\_lock;

int state; //当前链接的状态

atomic\_t state\_closed; // non-zero iff state = STATE\_CLOSED

PipeConnectionRef connection\_state; //PipeConnection 的引用

utime\_t backoff; // backoff time

bool reader\_running, reader\_needs\_join;

bool reader\_dispatching; // reader thread is dispatching without pipe\_lock

bool notify\_on\_dispatch\_done; // something wants a signal when dispatch done

bool writer\_running;

map<int, list<Message\*> > out\_q; // priority queue for outbound msgs

//准备发送的消息 优先队列

DispatchQueue \*in\_q; //接收消息的DispatchQueue

list<Message\*> sent; //要发送的消息

Cond cond;

bool send\_keepalive;

bool send\_keepalive\_ack;

utime\_t keepalive\_ack\_stamp;

bool halt\_delivery; //if a pipe's queue is destroyed, stop adding to it

\_\_u32 connect\_seq, peer\_global\_seq;

uint64\_t out\_seq; //发送消息的序列号

uint64\_t in\_seq, in\_seq\_acked; //接收到消息序号和 应对的序号

43 Messenger

Messenger它是Ceph网络传输的抽象体，包括Accepter、Pipe、Dispatch\_queue。

Ceph通过OSDSession这个类来维护管理client端和server端之间建立的Session连接，并且维护了Pending在Session中的所有Op。OSDSession暴露出来的接口就是一个ConnectionRef结构的send\_message()，而在这个函数里，它实际上又返回过来找到messenger、pipe然后将请求发送出去。

主要的使用流程是：

1）通过Messenger::create\_client\_messenger()或者Messenger::create()创建一个messenger，目前支持SimpleMessenger、AsyncMessenger、Xio这三种，可以在ceph的配置文件中配置。

2）创建一个dispatcher,如果是客户端一般就是创建一个Objecter和Monc，如果是服务端就创建对应的角色实例如OSD、Mon等，并且上一步创建的messenger会被设置为他们的messenger。

3）设置Messenger的policy，用于配置Socket连接的异常处理策略。

4）Dispatcher的初始化，如objecter->init();

5）如果是服务器端（比如OSD），则需要调用messenger的bind()接口，为messenger绑定一个地址和端口，如果是客户端（比如Objecter）则不需要。这个bind操作是通过Accepter的bind()接口完成的，它里面实际上干了三件事情：创建Socket、bind地址和端口、listen地址和端口；

6）通过Messenger的add\_dispatcher\_head()/add\_dispatcher\_tail()将初始化后的dispatcher加入的messenger的dispatcher list中。如果是添加的第一个dispatcher，则会调用Messenger的ready()接口，这个接口会调用Accepter的start()接口。而在客户端和服务端，他们对于start()的处理是不一样的。服务器端会创建一个线程使用poll机制去处理accept的连接，客户端则不会处理。服务器accept的Socket连接会创建相应的Pipe，丢到messenger里面。

7）当pipe建立好之后，客户端就会通过OSDSession中的conn找到messenger的接口，发送数据了。

参考资料：

http://blog.csdn.net/scaleqiao/article/details/51205054

<http://www.cnblogs.com/shanno/p/4014256.html>

<http://blog.csdn.net/zrs19800702/article/details/53005136>

<http://blog.csdn.net/changtao381/article/details/50915328>

http://blog.csdn.net/ywy463726588/article/details/42742355