ZeroMQ源码分析-课件

腾讯课堂 零声教育 Darren

网页版本：https://www.yuque.com/docs/share/b03ef14c-82de-4610-bc4a-96ecddb83652?#
《ZeroMQ源码分析-课件》

消息帧封装

- 消息协议设计
- 如何区分ZMQ_PUB、ZMQ_SUB、ZMQ_REQ、ZMQ_REP、ZMQ_PULL、ZMQ_PUSH、ZMQ_DEALER、ZMQ_ROUTER、
- 消息最大容量

消息水位的问题

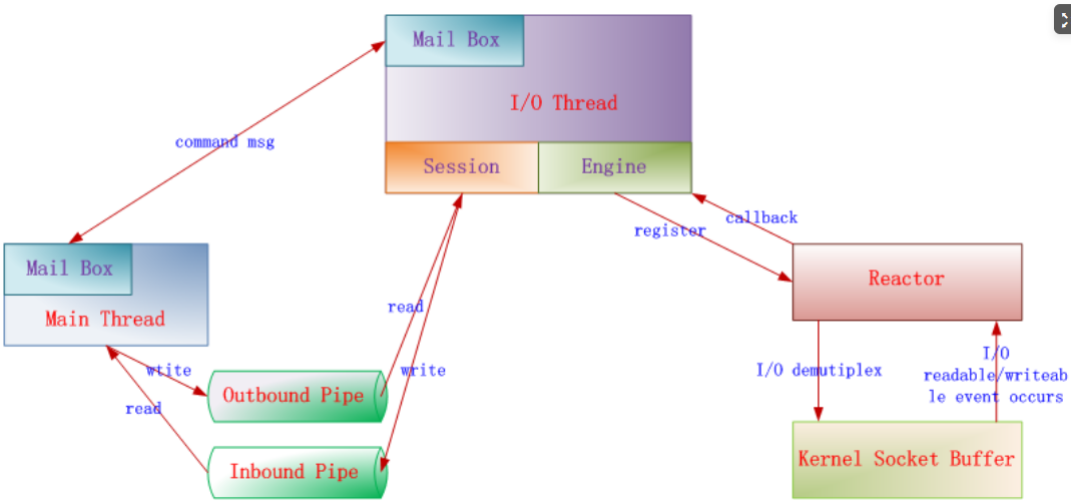
发布订阅模型

请求响应模型

push、pull模型

消息可靠性

1 主线程和IO线程模型



详细参考 《 2-1-预习资料-ZeroMQ源码分析》

2 消息

```
//  Message flags.     enum
    {
        more = 1,    //  Followed by more parts
        command = 2, //  Command frame (see ZMTP spec)
        //  Command types, use only bits 2-5 and compare with ==, not bitwise,
        //  a command can never be of more that one type at the same time
        ping = 4,
        pong = 8,
        subscribe = 12,
        cancel = 16,
        close_cmd = 20,
        credential = 32,
        routing_id = 64,
        shared = 128
    };
```

消息协议设计

简单请求应答

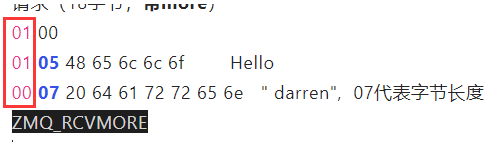
请求（18字节，**带more**）

01 00

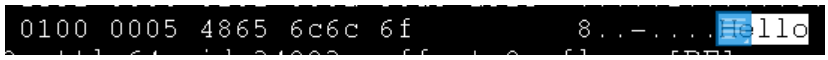
01 **05** 48 65 6c 6c 6f　　Hello

00 **07** 20 64 61 72 72 65 6e　" darren"，07代表字节长度

ZMQ_RCVMORE



只有1个数据帧



01 00

00 05 **48 65 6c 6c 6f**

回应（12字节）

01 00

00 08 57 6f 72 6c 64 20 31 20　"World.1."

复杂的请求应答

REQ->ZMQ_ROUTER

(9字节)

01 00

00 05 48 65 6c 6c 6f　Hello

ZMQ_ROUTER->REQ

（9字节）

01 00

00 05 57 6f 72 6c 64　World

ZMQ_DEALER->REP

16字节

01 05 00 6b 8b 45 *68*
01 00
00 05 48 65 6c 6c 6f Hello

重新连接
16字节
01 05 00 6b 8b 45 **69**
01 00
00 05 48 65 6c 6c 6f Hello

REP->ZMQ_DEALER

01 05 00 6b 8b 45 68
01 00
00 05 57 6f 72 6c 64

发布订阅消息包

2.PUB/SUB模型
wuserver.c PUB
wuclient.c SUB

PUB->SUB回应

length 27
04 19
05 52 4 541 44 59 "READY"
0b 53 6f 63 6b 65 74 2d 54 79 70 65 Socket-Type
00 00
00 03 50 55 42 "PUB"

SUB-PUB设置过滤连接

length 45

04 19
05 52 45 41 44 59 "READY"
0b 53 6f 63 6b 65 74 2d 54 79 70 65 Socket-Type
00 00

00 03 53 55 42 "SUB"
04 10
09 53 55 42 53 43 52 49 42 45 SUBSCRIBE
31 30 30 30 31 20 "10001" 20 为'\0'

消息水位的问题

ZMQ_SNDHWM 发送水位
ZMQ_RCVHWM 接收水位

```
//  High watermark for the outbound pipe.
    int _hwm;
    //  Low watermark for the inbound pipe.
    int _lwm;
```

ZMQ使用高水位标记（HWM）的概念来定义其内部管道的容量。每个连接都有其发送或接受数据的管道和对应的HWM。对于管道和HWM，不同的socket有不同的行为：

- PUB，PUSH只有发送管道
- SUB，PULL，REP，REQ只有接收管道
- DEALER, ROUTER, PAIR两者都有

缺省的值
const int default_hwm = 1000;

可以通过setsocketopt函数来设置HWM的值。
当数据填满管道，达到HWM时，不同的socket也有不同的表现：

- 在 PUB 和 ROUTER 类型套接字上，当在传出管道上达到 SNDHWM 时，将会丢弃消息。DEALER 和 PUSH 类型套接字将会堵塞。
- 在传入管道上，到达 RCVHWM 时，SUB 类型套接字将会丢弃消息，而 PULL 和 DEALER 套接字将拒绝新消息并强制消息在上游等待。

设置范例

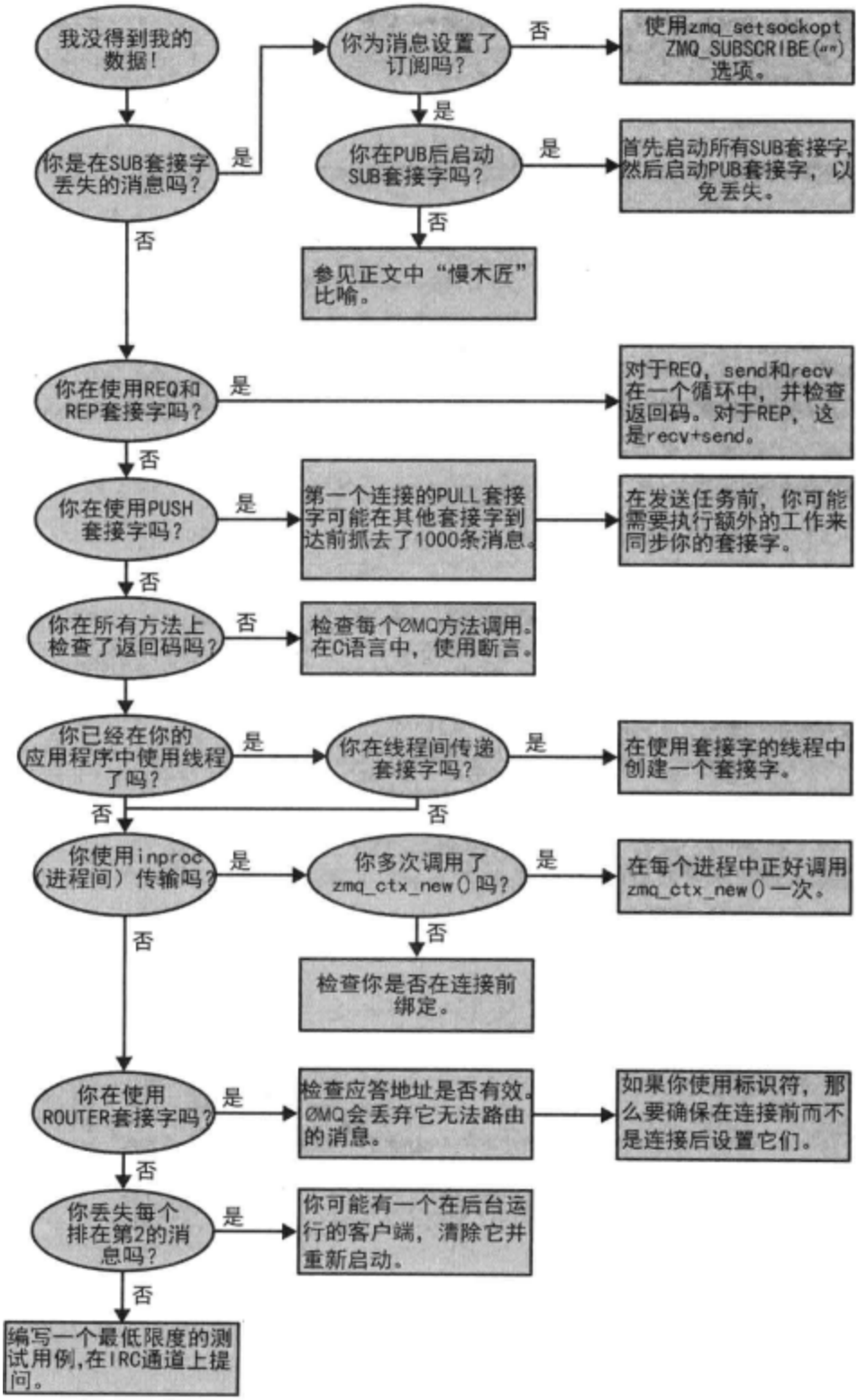
```C++
 void *socket = zmq_socket (context, ZMQ_PUSH);
zmq_connect (requester, "tcp://localhost:5555");
int queue_length = 5000;
zmq_setsockopt(socket, ZMQ_SNDHWM, &queue_length,sizeof(queue_length));
zmq_connect (socket, "tcp://127.0.0.1:5555");
```

水位控制在于pipe_t

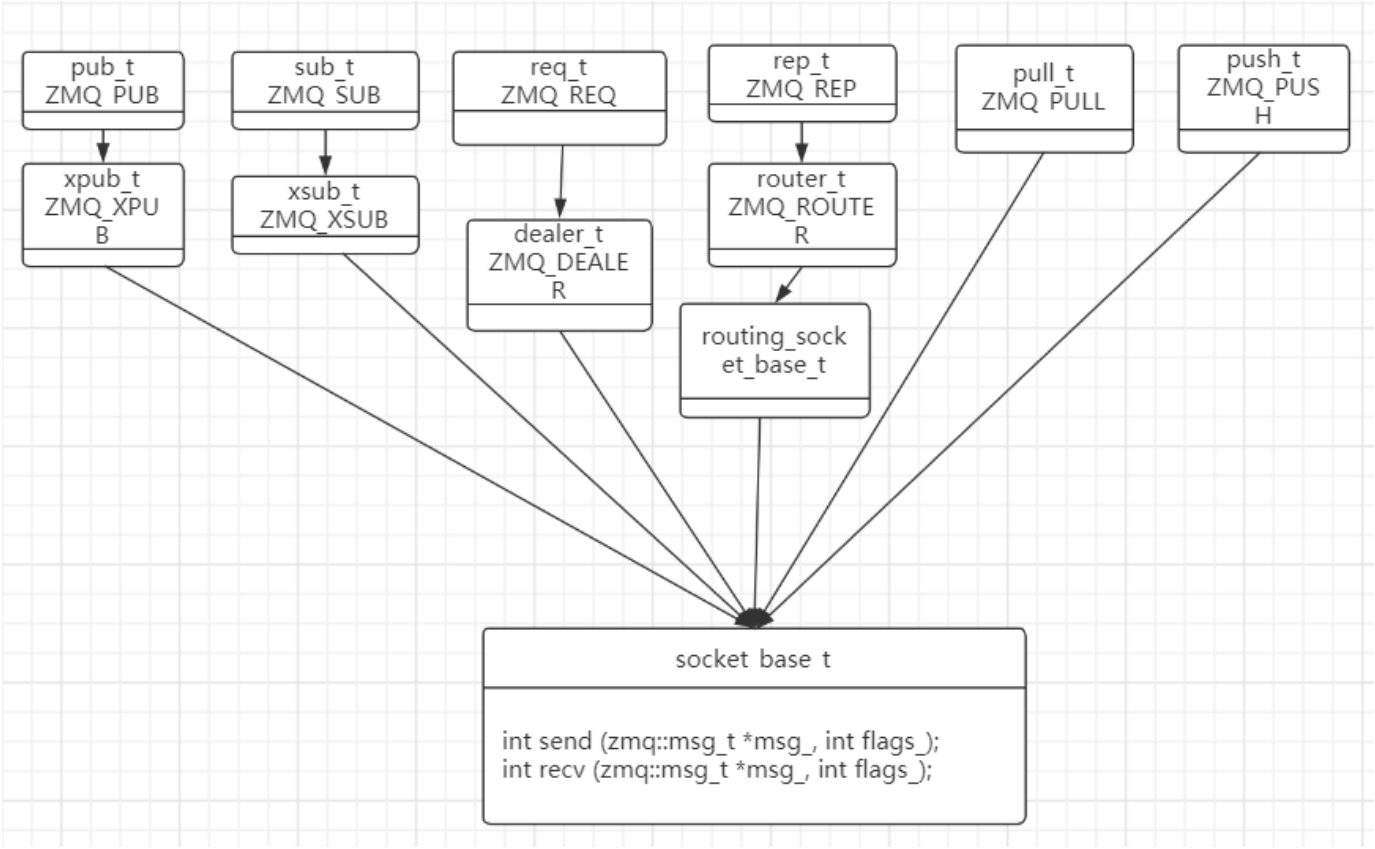
必须为PUB套接字设置阈值，具体数字可以通过最大订阅者数、可供队列使用的最大内存区域、以及消息的平均大小来衡量。举例来说，你预计会有5000个订阅者，有1G的内存可供使用，消息大小在200个字节左右，那么，一个合理的阈值是1,000,000,000 / 200 / 5,000 = 1,000。

zmq::pipe_t::pipe_t (this=0x7ffff0010860, parent_=0x7ffff00010e0, inpipe_=0x7ffff0008660, outpipe_=0x7ffff000c760, inhwm_=1000, outhwm_=1000, conflate_=false

消息可靠性


我没得到我的数据!
你为消息设置了订阅吗?
否
使用zmq_setsockopt ZMQ_SUBSCRIBE("")选项。
是
你是在SUB套接字丢失的消息吗?
是
你在PUB后启动SUB套接字吗?
是
首先启动所有SUB套接字,然后启动PUB套接字,以免丢失。
否
参见正文中“慢木匠”比喻。
否
你在使用REQ和REP套接字吗?
是
对于REQ,send和recv在一个循环中,并检查返回码。对于REP,这是recv+send。
否
你在使用PUSH套接字吗?
是
第一个连接的PULL套接字可能在其他套接字到达前抓去了1000条消息。
在发送任务前,你可能需要执行额外的工作来同步你的套接字。
否
你在所有方法上检查了返回码吗?
否
检查每个0MQ方法调用。在C语言中,使用断言。
你已经在你的应用程序中使用线程了吗?
是
你在线程间传递套接字吗?
是
在使用套接字的线程中创建一个套接字。
否
否
你使用inproc(进程间)传输吗?
是
你多次调用了zmq_ctx_new()吗?
是
在每个进程中正好调用zmq_ctx_new()一次。
否
否
检查你是否在连接前绑定。
你在使用ROUTER套接字吗?
是
检查应答地址是否有效。0MQ会丢弃它无法路由的消息。
如果你使用标识符,那么要确保在连接前而不是连接后设置它们。
否
你丢失每个排在第2的消息吗?
是
你可能有一个在后台运行的客户端,清除它并重新启动。
否
编写一个最低限度的测试用例,在IRC通道上提问。

3 常用模型



ZMQ_REQ/ZMQ_REP请求响应模型

ZMQ_REQ

int zmq::req_t::xsend (msg_t *msg_) 发送数据

- 通过_receiving_reply控制send的发送，当_receiving_reply为true时等待对方应答
- 当消息分帧发送时，只有最后一帧才将_receiving_reply设置为true

int zmq::req_t::xrecv (msg_t *msg_) 接收数据

- 接收完同一个消息的所有帧才将_receiving_reply置为false，运行继续send

ZMQ_REP

int zmq::rep_t::xsend (msg_t *msg_) 发送数据

- 通过_sending_reply控制send的发送，当_sending_reply为false时需要对方发请求后才能send数据
- 当消息分帧时，只有最后一帧才将_sending_reply设置为false，说明send rep完成，等待新的req

int zmq::rep_t::xrecv (msg_t *msg_)接收数据

- _sending_reply如果为true，说明还没有回复消息，此时不能再接收数据

- 接收完同一个消息的所有帧才将_sending_reply置为true，说明转到rep状态，只有send rep后才能再接收数据。

ZMQ_PUB/ZMQ_SUB发布订阅模型

SUB订阅者要先到PUB发布端订阅数据。比如REQ/REP模型复杂。

pub & sub模式的核心在于:

- subscriber向publisher注册filter (topic),
- publisher根据filter(topic)向对应的subscriber发布消息

ZeroMQ使用了两组socket_base_t的派生类。
xsub_t和sub_t用于sub端。**sub_t的xsetsockopt()用于设置filter选项**，它会调用xsub_t的xsend()向pub端发送一个注册请求。当sub端收到发布的消息时，会暂存在xsbub_t的fq_t成员中，以备以后用户调用xrecv()来获取。
xpub_t和pub_t用于pub端。 mtrie_t成员和dist_t成员配合使用，用于过滤和发布消息。mtrie_t是一种字典树，支持prefix match模式。sub端注册filter时，对应的pipe_t同时保存在mtrie_t和dist_t中，在mtrie_t中根据消息头（可以理解为topic）找到期望的目标pipe_t时，会调用mark_as_matching（）在dist_t中标记pipe_t，然后dist_t根据标记发送消息。

ZMQ_SUB

要订阅

ZMQ_PUB

发布

```
int zmq::zmtp_engine_t::process_command_message (msg_t *msg_){
    const uint8_t cmd_name_size =
      *(static_cast<const uint8_t *> (msg_->data ()));
    const size_t ping_name_size = msg_t::ping_cmd_name_size - 1;
    const size_t sub_name_size = msg_t::sub_cmd_name_size - 1;
    const size_t cancel_name_size = msg_t::cancel_cmd_name_size - 1;
    //  Malformed command
    if (unlikely (msg_->size () < cmd_name_size + sizeof (cmd_name_size)))
        return -1;

    const uint8_t *const cmd_name =
      static_cast<const uint8_t *> (msg_->data ()) + 1;     // 获取订阅命令
    if (cmd_name_size == ping_name_size
```

```
        && memcmp (cmd_name, "PING", cmd_name_size) == 0)
        msg_->set_flags (zmq::msg_t::ping);
    if (cmd_name_size == ping_name_size
        && memcmp (cmd_name, "PONG", cmd_name_size) == 0)
        msg_->set_flags (zmq::msg_t::pong);
    if (cmd_name_size == sub_name_size
        && memcmp (cmd_name, "SUBSCRIBE", cmd_name_size) == 0)
        msg_->set_flags (zmq::msg_t::subscribe);
    if (cmd_name_size == cancel_name_size
        && memcmp (cmd_name, "CANCEL", cmd_name_size) == 0)
        msg_->set_flags (zmq::msg_t::cancel);

    if (msg_->is_ping () || msg_->is_pong ())
        return process_heartbeat_message (msg_);

    return 0;
}
```

发布者使用 mtrie_t _subscriptions; 保存对应的订阅以及对应的pipe。

zmq::dist_t::attach绑定对应SUB的pipe

```
#0  zmq::dist_t::attach (this=0x619a88, pipe_=0x7ffff00109d0)    at src/dist.cpp:55
#1  0x00007ffff7b71385 in zmq::xpub_t::xattach_pipe (this=0x619370,
    pipe_=0x7ffff00109d0, subscribe_to_all_=false, locally_initiated_=false)
    at src/xpub.cpp:76
#2  0x00007ffff7b3b3dc in zmq::pub_t::xattach_pipe (this=0x619370,
    pipe_=0x7ffff00109d0, subscribe_to_all_=false, locally_initiated_=false)
    at src/pub.cpp:56
#3  0x00007ffff7b4dbb0 in zmq::socket_base_t::attach_pipe (this=0x619370,
    pipe_=0x7ffff00109d0, subscribe_to_all_=false, locally_initiated_=false)
    at src/socket_base.cpp:417
#4  0x00007ffff7b5230c in zmq::socket_base_t::process_bind (this=0x619370,
    pipe_=0x7ffff00109d0) at src/socket_base.cpp:1536
#5  0x00007ffff7b2895b in zmq::object_t::process_command (this=0x619370,
    cmd_=...) at src/object.cpp:102
#6  0x00007ffff7b52145 in zmq::socket_base_t::process_commands (this=0x619370,
    timeout_=0, throttle_=true) at src/socket_base.cpp:1505
#7  0x00007ffff7b515a5 in zmq::socket_base_t::send (this=0x619370,
```

```
    msg_=0x7fffffffdd40, flags_=0) at src/socket_base.cpp:1244
#8  0x00007ffff7b7a8e4 in s_sendmsg (s_=0x619370, msg_=0x7fffffffdd40,
    flags_=0) at src/zmq.cpp:381
#9  0x00007ffff7b7a9e3 in zmq_send (s_=0x619370, buf_=0x7fffffffde00, len_=16,
    flags_=0) at src/zmq.cpp:409
---Type <return> to continue, or q <return> to quit---
#10 0x00000000004010cb in s_send ()
#11 0x0000000000401755 in main ()
```

generic_mtrie_t<T>::match匹配算法(匹配算法)

```
void generic_mtrie_t<T>::match (prefix_t data_,                                size_t size_,
                                void (*func_) (value_t *pipe_, Arg arg_),
                                Arg arg_)
{
    for (generic_mtrie_t *current = this; current; data_++, size_--) {
        //  Signal the pipes attached to this node.
        if (current->_pipes) {
            for (typename pipes_t::iterator it = current->_pipes->begin (),
                                            end = current->_pipes->end ();
                 it != end; ++it) {
                func_ (*it, arg_);
            }
        }

        //  If we are at the end of the message, there's nothing more to match.
        if (!size_)
            break;

        //  If there are no subnodes in the trie, return.
        if (current->_count == 0)
            break;

        if (current->_count == 1) {
            //  If there's one subnode (optimisation).
            if (data_[0] != current->_min) {
                break;
            }
            current = current->_next.node;
```

```
        } else {
            //  If there are multiple subnodes.
            if (data_[0] < current->_min
                || data_[0] >= current->_min + current->_count) {
                break;
            }
            current = current->_next.table[data_[0] - current->_min];
        }
    }
}
```

zmq::dist_t::match标记为匹配

```
#0  zmq::dist_t::match (this=0x619a88, pipe_=0x7ffff00109d0) at src/dist.cpp:70#1
0x00007ffff7b71f74 in zmq::xpub_t::mark_as_matching (pipe_=0x7ffff00109d0,
    self_=0x619370) at src/xpub.cpp:287
#2  0x00007ffff7b74830 in zmq::generic_mtrie_t<zmq::pipe_t>::match<zmq::xpub_t*>
(this=0x619a58, data_=0x7fffffffdd4e "2649949 24p\223a", size_=10,
    func_=0x7ffff7b71f4a <zmq::xpub_t::mark_as_matching(zmq::pipe_t*, zmq::xpub_t*)>,
arg_=0x619370) at src/generic_mtrie_impl.hpp:557
#3  0x00007ffff7b720ef in zmq::xpub_t::xsend (this=0x619370,
    msg_=0x7fffffffdd40) at src/xpub.cpp:312
#4  0x00007ffff7b51619 in zmq::socket_base_t::send (this=0x619370,
    msg_=0x7fffffffdd40, flags_=0) at src/socket_base.cpp:1259
#5  0x00007ffff7b7a8e4 in s_sendmsg (s_=0x619370, msg_=0x7fffffffdd40,
    flags_=0) at src/zmq.cpp:381
#6  0x00007ffff7b7a9e3 in zmq_send (s_=0x619370, buf_=0x7fffffffde00, len_=16,
    flags_=0) at src/zmq.cpp:409
#7  0x00000000004010cb in s_send ()
#8  0x0000000000401755 in main ()
```

ZMQ_PUSH/ZMQ_PULL

每个收到的消息都有一个*routing_id*，它是一个32位无符号整数。要将消息发送到给定的CLIENT对等方，应用程序必须在消息上设置对等方的 *routing_id*。
如果未指定*routing_id*，或者未引用已连接的客户端对等方，则发送调用将失败。如果客户端对等方的传出缓冲区已满，则发送调用将阻塞。在任何情况下，SERVER套接字都不会丢弃消息。

ZMQ_PULL

zmq::pull_t::xattach_pipe 作为server的时候用

ZMQ_PUSH

zmq::push_t::xattach_pipe作为server的时候用

#0 zmq::push_t::xattach_pipe (this=0x619370, pipe_=0x7ffff000fbc0, subscribe_to_all_=false, locally_initiated_=false) at src/push.cpp:56
#1 0x00007ffff7b4dbb0 in zmq::socket_base_t::attach_pipe (this=0x619370,
    pipe_=0x7ffff000fbc0, subscribe_to_all_=false, locally_initiated_=false)
    at src/socket_base.cpp:417
#2 0x00007ffff7b5230c in zmq::socket_base_t::process_bind (this=0x619370,
    pipe_=0x7ffff000fbc0) at src/socket_base.cpp:1536
#3 0x00007ffff7b2895b in zmq::object_t::process_command (this=0x619370, cmd_=...)
    at src/object.cpp:102
#4 0x00007ffff7b52145 in zmq::socket_base_t::process_commands (this=0x619370, timeout_=-1,
    throttle_=false) at src/socket_base.cpp:1505
#5 0x00007ffff7b517f5 in zmq::socket_base_t::send (this=0x619370, msg_=0x7fffffffe320,
    flags_=0) at src/socket_base.cpp:1294
#6 0x00007ffff7b7a8e4 in s_sendmsg (s_=0x619370, msg_=0x7fffffffe320, flags_=0)
    at src/zmq.cpp:381
#7 0x00007ffff7b7a9e3 in zmq_send (s_=0x619370, buf_=0x7fffffffe3d0, len_=10, flags_=0)
    at src/zmq.cpp:409
#8 0x00000000004010cb in s_send ()
#9 0x00000000004016b0 in main ()

zmq::push_t::xsend

#0 zmq::push_t::xsend (this=0x619370, msg_=0x7fffffffe320) at src/push.cpp:74#1 0x00007ffff7b51831 in zmq::socket_base_t::send (this=0x619370, msg_=0x7fffffffe320,
    flags_=0) at src/socket_base.cpp:1297
#2 0x00007ffff7b7a8e4 in s_sendmsg (s_=0x619370, msg_=0x7fffffffe320, flags_=0)
    at src/zmq.cpp:381
#3 0x00007ffff7b7a9e3 in zmq_send (s_=0x619370, buf_=0x7fffffffe3d0, len_=10, flags_=0)
    at src/zmq.cpp:409
#4 0x00000000004010cb in s_send ()
#5 0x00000000004016b0 in main ()

```
int zmq::push_t::xsend (msg_t *msg_)
{
    return _lb.send (msg_);   // 负载均衡
}
```

最终调用zmq::lb_t::sendpipe，负载均衡只是简单的轮询操作。

```
int zmq::lb_t::sendpipe (msg_t *msg_, pipe_t **pipe_){
    //  Drop the message if required. If we are at the end of the message
    ...

    while (_active > 0) {
        if (_pipes[_current]->write (msg_)) {            // 发送数据到pipe
            if (pipe_)
                *pipe_ = _pipes[_current];
            break;
        }

        // If send fails for multi-part msg rollback other
        // parts sent earlier and return EAGAIN.
        // Application should handle this as suitable
        if (_more) {
            _pipes[_current]->rollback ();
            _dropping = (msg_->flags () & msg_t::more) != 0;
            _more = false;
            errno = EAGAIN;
            return -2;
        }

        _active--;
        if (_current < _active)
            _pipes.swap (_current, _active);
        else
            _current = 0;
    }

   ....

    return 0;
}
```

ZMQ_DEALER和ZMQ_ROUTER

ZMQ_ROUTER回复

zmq::router_t::xsend(msg_t *msg_) 要查找到对应req其pipe是哪个

Thread 1 "rrbroker" hit Breakpoint 4, zmq::router_t::xsend (this=0x619370,
msg_=0x7fffffffe3b0) at src/router.cpp:189
189 {
(gdb) bt
#0 zmq::router_t::xsend (this=0x619370, msg_=0x7fffffffe3b0) at src/router.cpp:189
#1 0x00007ffff7b51619 in zmq::socket_base_t::send (this=0x619370, msg_=0x7fffffffe3b0,
flags_=2) at src/socket_base.cpp:1259
#2 0x00007ffff7b7a8e4 in s_sendmsg (s_=0x619370, msg_=0x7fffffffe3b0, flags_=2)
at src/zmq.cpp:381
#3 0x00007ffff7b7b735 in zmq_msg_send (msg_=0x7fffffffe3b0, s_=0x619370, flags_=2)
at src/zmq.cpp:636
#4 0x0000000000401826 in main ()

zmq::router_t::xrecv (msg_t *msg_)公平队列读取消息

rc = _fq.recvpipe (msg_, &pipe);

#0 zmq::router_t::xrecv (this=0x619370, msg_=0x7fffffffe3b0) at src/router.cpp:291#1
0x00007ffff7b51a34 in zmq::socket_base_t::recv (this=0x619370, msg_=0x7fffffffe3b0,
flags_=0) at src/socket_base.cpp:1347
#2 0x00007ffff7b7aea3 in s_recvmsg (s_=0x619370, msg_=0x7fffffffe3b0, flags_=0)
at src/zmq.cpp:493
#3 0x00007ffff7b7b77e in zmq_msg_recv (msg_=0x7fffffffe3b0, s_=0x619370, flags_=0)
at src/zmq.cpp:644
#4 0x0000000000401766 in main ()

ZMQ_DEALER请求

zmq::dealer_t::xsend

zmq::dealer_t::xrecv

zmq::dealer_t::sendpipe支持负载均衡发送消息

```
int zmq::dealer_t::sendpipe (msg_t *msg_, pipe_t **pipe_){
    return _lb.sendpipe (msg_, pipe_);              // 负载均衡
}
```

```
#0  zmq::dealer_t::sendpipe (this=0x61a110, msg_=0x7fffffffe3b0, pipe_=0x0)    at
src/dealer.cpp:139
#1  0x00007ffff7b0d6e8 in zmq::dealer_t::xsend (this=0x61a110, msg_=0x7fffffffe3b0)
    at src/dealer.cpp:103
#2  0x00007ffff7b51619 in zmq::socket_base_t::send (this=0x61a110, msg_=0x7fffffffe3b0,
    flags_=2) at src/socket_base.cpp:1259
#3  0x00007ffff7b7a8e4 in s_sendmsg (s_=0x61a110, msg_=0x7fffffffe3b0, flags_=2)
    at src/zmq.cpp:381
#4  0x00007ffff7b7b735 in zmq_msg_send (msg_=0x7fffffffe3b0, s_=0x61a110, flags_=2)
    at src/zmq.cpp:636
#5  0x00000000004017a0 in main ()
```

zmq::dealer_t::recvpipe公平读取消息

```
Thread 1 "rrbroker" hit Breakpoint 3, zmq::dealer_t::recvpipe (this=0x61a110,
msg_=0x7fffffffe3b0, pipe_=0x0) at src/dealer.cpp:144
144     return _fq.recvpipe (msg_, pipe_);
(gdb) bt
#0  zmq::dealer_t::recvpipe (this=0x61a110, msg_=0x7fffffffe3b0, pipe_=0x0)
    at src/dealer.cpp:144
#1  0x00007ffff7b0d712 in zmq::dealer_t::xrecv (this=0x61a110, msg_=0x7fffffffe3b0)
    at src/dealer.cpp:108
#2  0x00007ffff7b51a34 in zmq::socket_base_t::recv (this=0x61a110, msg_=0x7fffffffe3b0,
    flags_=0) at src/socket_base.cpp:1347
#3  0x00007ffff7b7aea3 in s_recvmsg (s_=0x61a110, msg_=0x7fffffffe3b0, flags_=0)
    at src/zmq.cpp:493
#4  0x00007ffff7b7b77e in zmq_msg_recv (msg_=0x7fffffffe3b0, s_=0x61a110, flags_=0)
    at src/zmq.cpp:644
#5  0x00000000004017ec in main ()
```