ZeroMQ

Виды сокетов
Простые паттерны применения

Виды сокетов. PAIR

- Двунаправленое эксклюзивное подключение
- Не переподключается в отличие от других сокетов
- Предназначен для внутренней последовательной обработки запросов
- Используется вместе с классом ZThread :
  - Zthread запускает поток и создает два связанных сокета Pair. Один сокет передается запущенному потоку. Другой возвращается

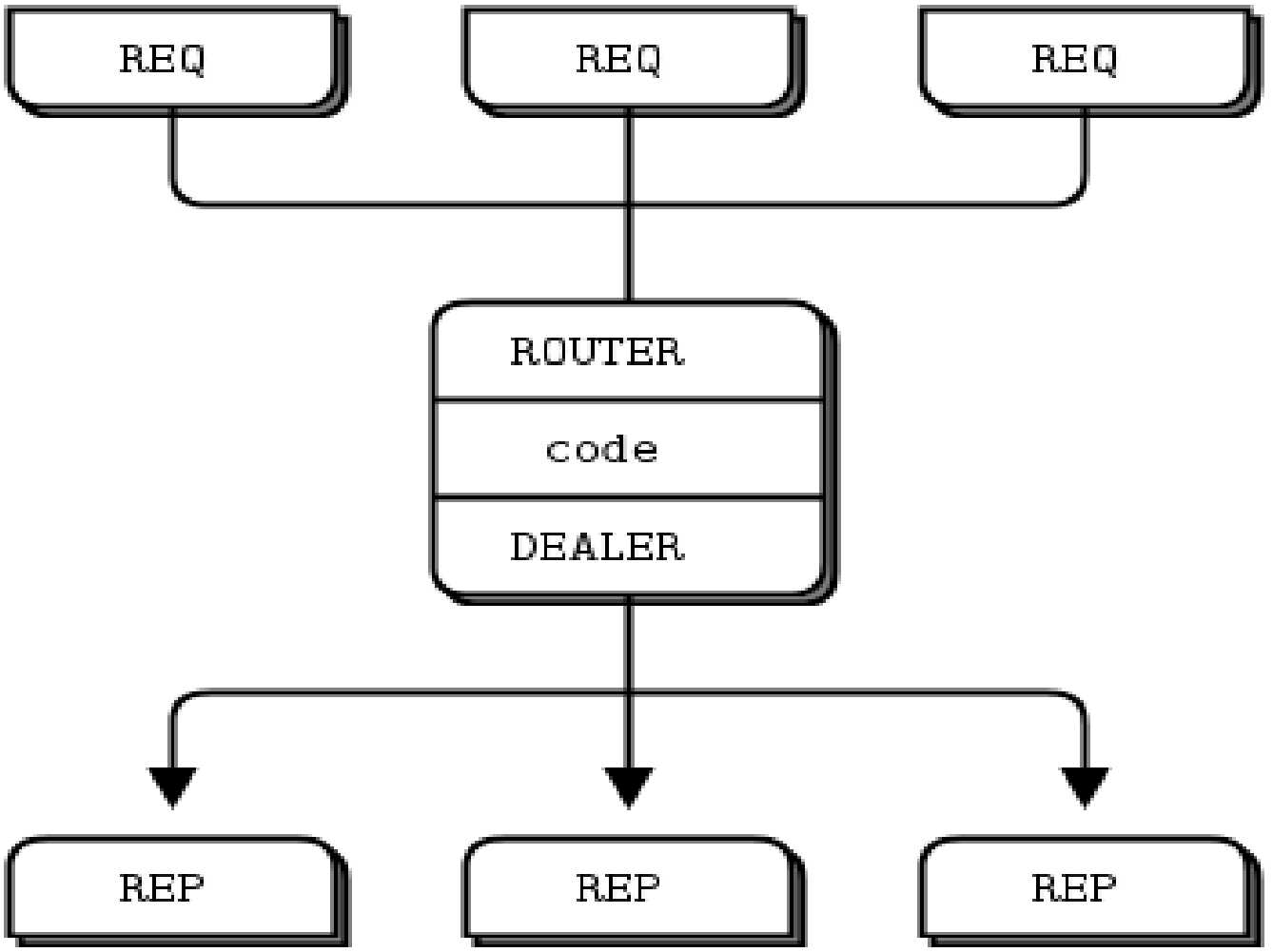
Виды сокетов. ROUTER

- ROUTER предназначен для асинхронного обмена сообщениями с несколькими внешними клиентами.
- К каждому входящему сообщению добавляется фрейм – идентификатор соединения.
- Каждое исходящее сообщение должно иметь первый фрейм – идентификатор соединения
- Каждый обмен данными в сокете ROUTER должен начинаться с сообщения пришедшего снаружи, в противном случае мы никак не можем узнать существующие идентификаторы подключенных клиентов

Виды сокетов. DEALER

- Самый “продвинутый” сокет :
  - Обмен полностью асинхронный
  - Можно как посылать сообщения так и получать
- При отправке сообщение будет отправлено случайному подключенному клиенту
- При получении сообщения – будет взято сообщение из общей очереди

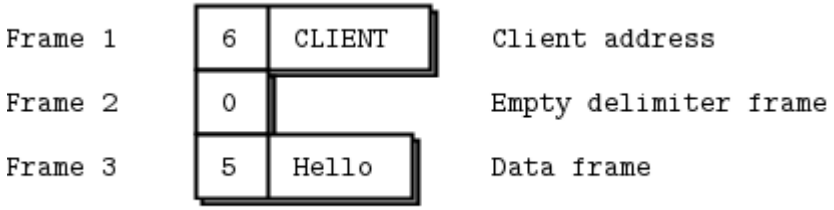
Схема простого прокси REQ-ROUTER:DEALER-REP



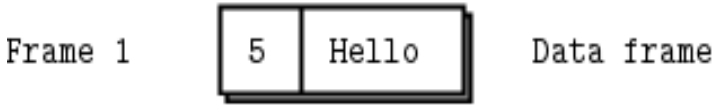
Фреймы сообщений обмена

1.Фрейм отправленный клиентом

2.Фрейм пришедший из ROUTER и отправленный в DEALER

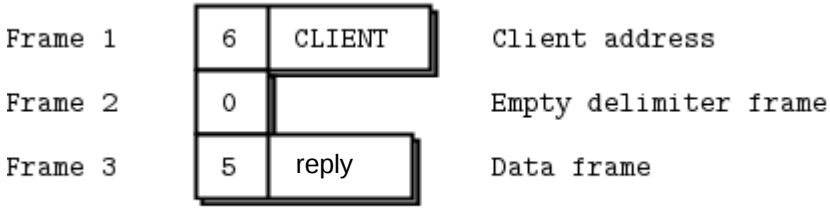


3.Фрейм полученный из REP

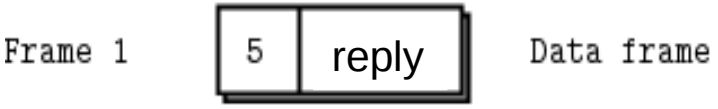


4.Фрейм отправленный в REP

2.Фрейм пришедший из DEALER и отправленный в ROUTER



4.Фрейм полученный из REQ



Пример ROUTER-DEALER Клиент

```
Context context = ZMQ.context(1);
// Socket to talk to server
Socket requester = context.socket(SocketType.REQ);
requester.connect("tcp://localhost:5559");
System.out.println("launch and connect client.");
for (int request_nbr = 0; request_nbr < 10; request_nbr++) {
   requester.send("Hello", 0);
   String reply = requester.recvStr (0);
  System.out.println("Received reply " + request_nbr + " [" + reply + "]");
}
// We never get here but clean up anyhow
requester.close();
context.term();
```

Пример ROUTER-DEALER Worker

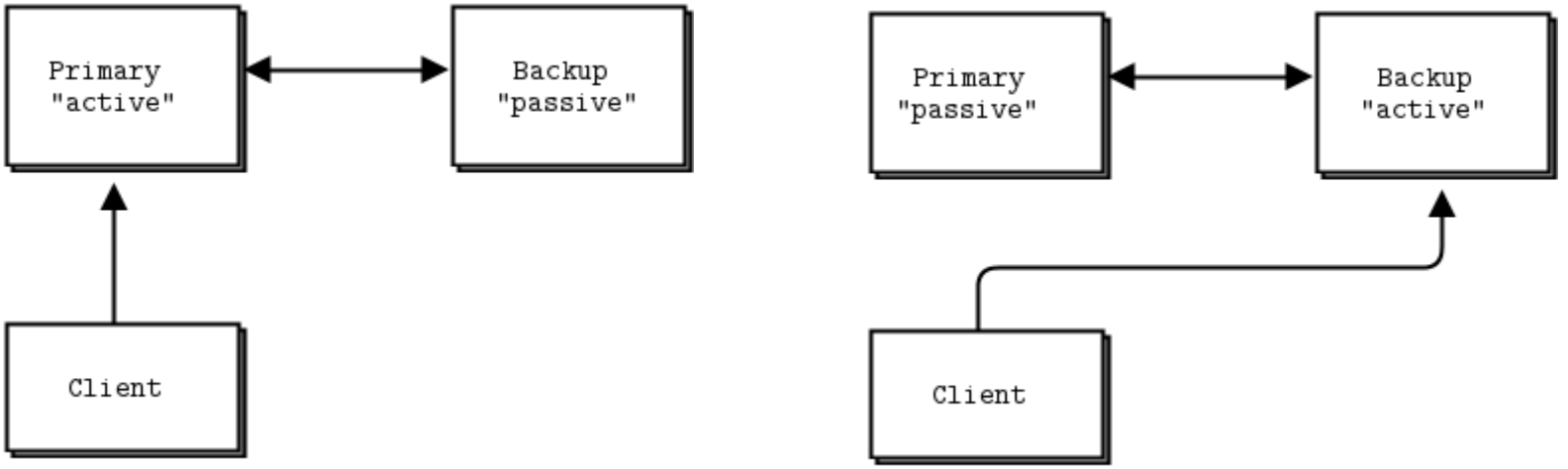
```
Context context = ZMQ.context (1);
// Socket to talk to server
Socket responder = context.socket (SocketType.REP);
responder.connect ("tcp://localhost:5560");
while (!Thread.currentThread ().isInterrupted ()) {
   // Wait for next request from client
  String string = responder.recvStr (0);
  System.out.printf ("Received request: [%s]\n", string);
  // Do some 'work'
  Thread.sleep (1000);
  // Send reply back to client
  responder.send ("World");
}
// We never get here but clean up anyhow
responder.close();
context.term();
```

Пример ROUTER-DEALER Proxy

```
Context context = ZMQ.context(1);
Socket frontend = context.socket(SocketType.ROUTER);
Socket backend = context.socket(SocketType.DEALER);
frontend.bind("tcp://*:5559");
backend.bind("tcp://*:5560");
System.out.println("launch and connect broker.");
// Initialize poll set
Poller items = context.poller (2);
items.register(frontend, Poller.POLLIN);
items.register(backend, Poller.POLLIN);
boolean more = false;
byte[] message;
// Switch messages between sockets
while (!Thread.currentThread().isInterrupted()) {
    // poll and memorize multipart detection
    items.poll();
    if (items.pollin(0)) {
        while (true) {
            message = frontend.recv(0);
            more = frontend.hasReceiveMore();
            backend.send(message, more ? ZMQ.SNDMORE : 0);
            if(!more){
              break;
            }
        }
    }
    if (items.pollin(1)) {
        while (true) {
            message = backend.recv(0);
            more = backend.hasReceiveMore();
            frontend.send(message, more ? ZMQ.SNDMORE : 0);
            if(!more){
                break;
             }
         }
    }
}
```

Binary star

- Данный пример реализует отказоустойчивую архитектуру “звезда”
- Запасной (secondary) сервер запущен параллельно с основным и в случае его падения берет на себя функции основного.
- В случае недоступности сервера клиента переключается на запасной сервер



Архитектура Binary star

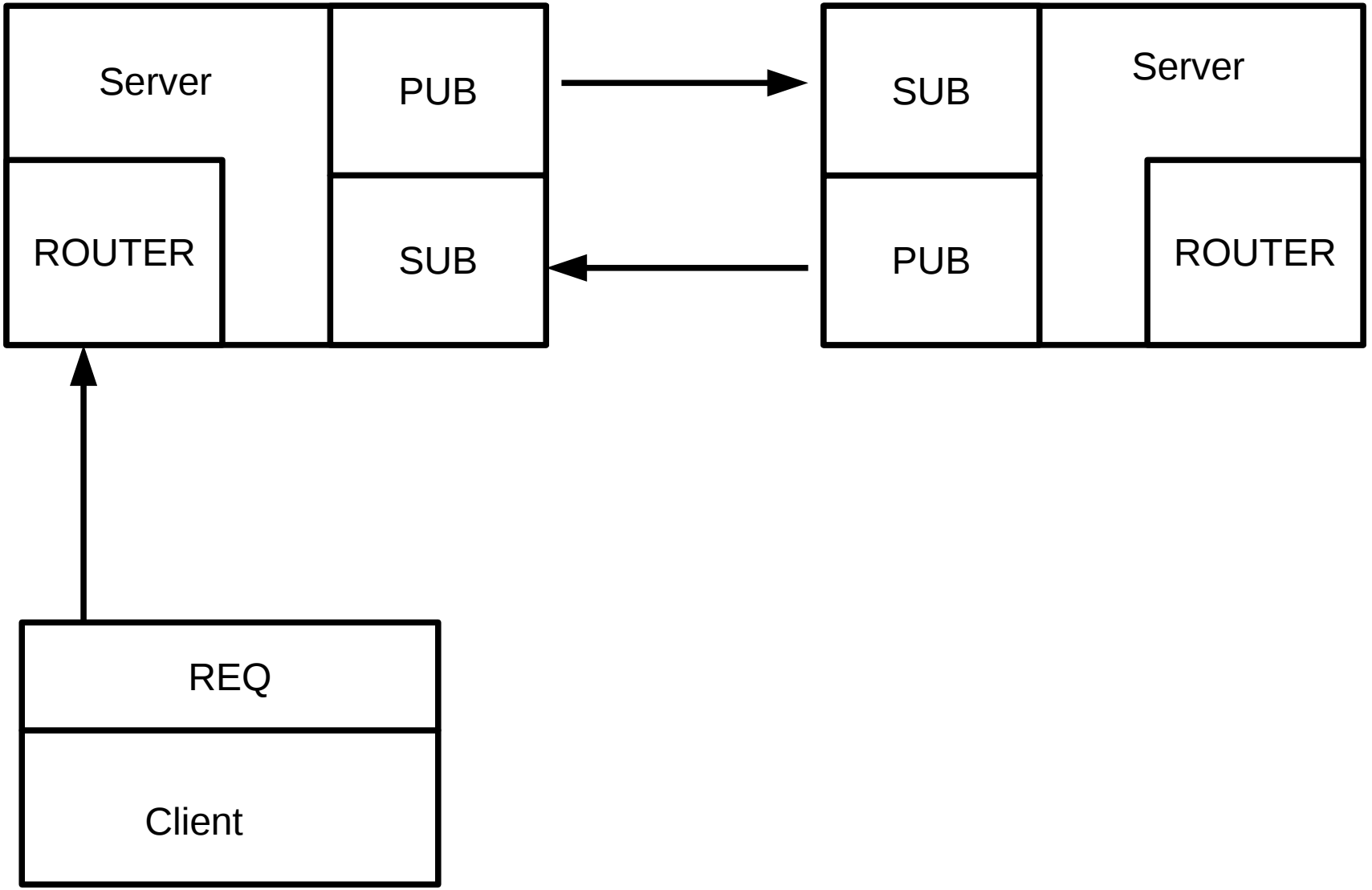
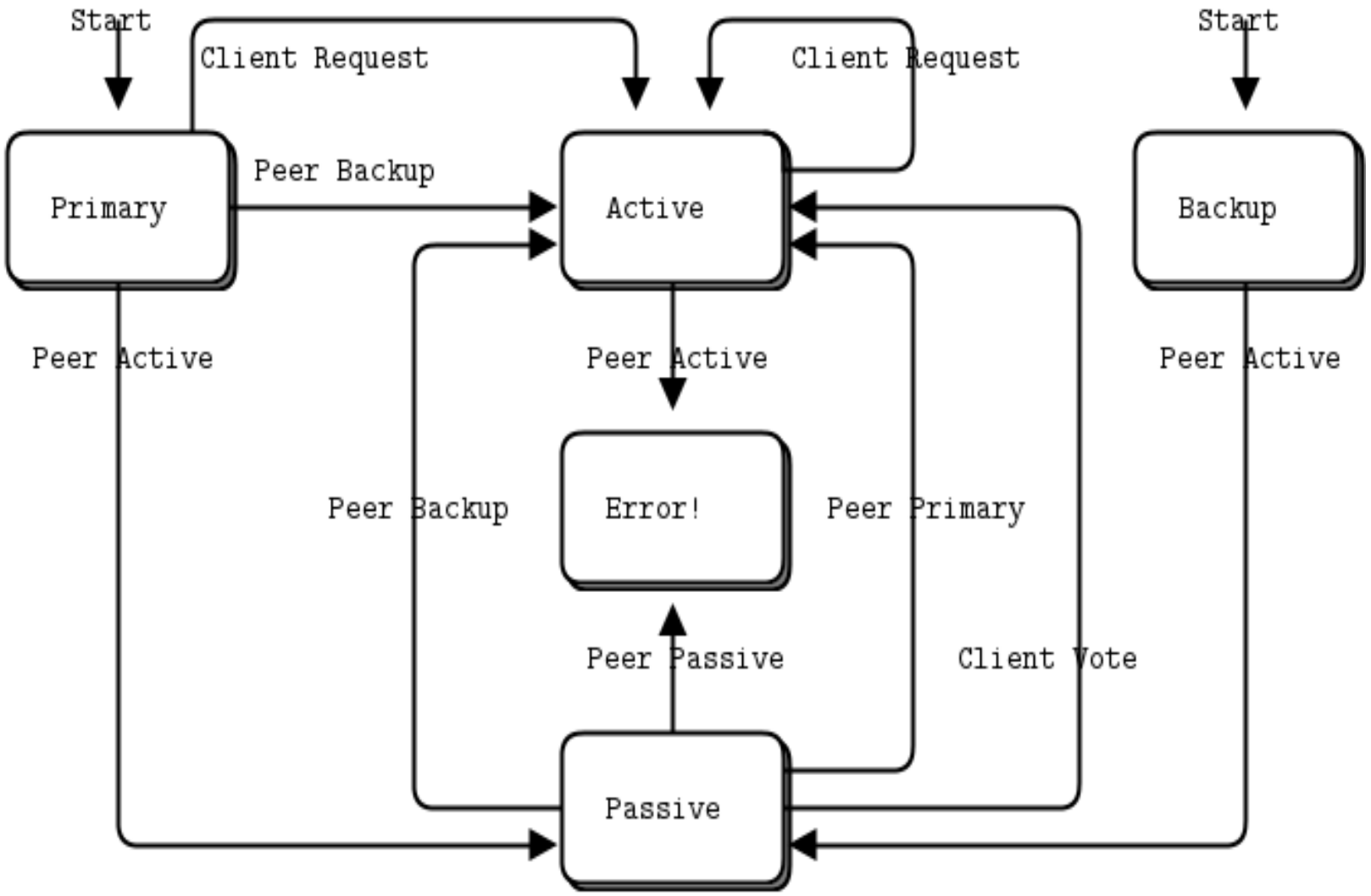


Диаграмма переходов состояний



Binary star.Клиент

```
    private static final long REQUEST_TIMEOUT = 5000l;
    private static final long SETTLE_DELAY = 10000l;
    private static String[] servers;
    private static int currentConnect = 0;
    private static String currentServer = null;


    public static ZMQ.Socket reconnect(ZContext ctx) {
        ZMQ.Socket client = ctx.createSocket(SocketType.REQ);
        currentServer = servers[currentConnect];
        client.connect(currentServer);
        currentConnect = (currentConnect + 1) % 2;
        return client;
    }


    public static void main(String[] args) throws InterruptedException {
        servers = new String[]{args[0], args[1]};
        ZContext ctx = new Zcontext();
        Selector selector = ctx.createSelector();
        ZMQ.Socket socket = reconnect(ctx);
        int count = 0;
```

```
while (!Thread.currentThread().isInterrupted()) {
            String request = String.format("%d", ++count);
            socket.send(request);
            boolean expectReply = true;
            while (expectReply) {
                ZMQ.PollItem items[] = {new ZMQ.PollItem(socket, ZMQ.Poller.POLLIN)};
                int rc = ZMQ.poll(selector, items, 1, REQUEST_TIMEOUT);
                if (rc == -1)  break;
                if (items[0].isReadable()) {
                    String reply = socket.recvStr();
                    int replyValue = Integer.parseInt(reply);
                    if(replyValue ==count) {
                        System.out.println("reply ok!");
                        expectReply= false;
                        Thread.sleep(1000);
                    } else {
                        System.out.println("incorrect reply!"+ replyValue);
                    }
                } else {
                    ctx.destroySocket(socket);
                    socket = reconnect(ctx);
                    Thread.sleep(SETTLE_DELAY);
                    System.out.println("resend request! to " + currentServer);
                    socket.send(request);
                }
            }
        }
```

Binary star. Server. State machine

```
enum State {
    BACKUP,
    PRIMARY,
    ACTIVE,
    PASSIVE,
    ERROR
}
enum Event {
    PEER_BACKUP,
    PEER_PRIMARY,
    PEER_ACTIVE,
    PEER_PASSIVE,
    CLIENT_REQUEST,
    CLIENT_VOTE
}
private static StateMachine<State, Event> stateMachine;
public static boolean fire(Event event) throws Exception {
    System.out.println("fire! " + event);
    if (stateMachine.CanFire(event)) {
        stateMachine.Fire(event);
    }
    return stateMachine.getState().equals(State.ACTIVE);
}
```

```
private static void initState(State initialState) throws
Exception {
    stateMachine = new StateMachine<State,
Event>(initialState);
    stateMachine.Configure(State.PRIMARY)
        .Permit(Event.PEER_BACKUP, State.ACTIVE)
        .Permit(Event.PEER_ACTIVE, State.PASSIVE)
        .Permit(Event.CLIENT_REQUEST, State.ACTIVE);
    stateMachine.Configure(State.BACKUP)
        .Permit(Event.PEER_ACTIVE, State.PASSIVE);
    stateMachine.Configure(State.ACTIVE)
        .Permit(Event.PEER_ACTIVE, State.ERROR);
    stateMachine.Configure(State.PASSIVE)
        .Permit(Event.PEER_BACKUP, State.ACTIVE)
        .Permit(Event.PEER_PRIMARY, State.ACTIVE)
        .Permit(Event.PEER_PASSIVE, State.ERROR)
        .Permit(Event.CLIENT_VOTE, State.ACTIVE);
}
```

Binary star. Server. Основной цикл

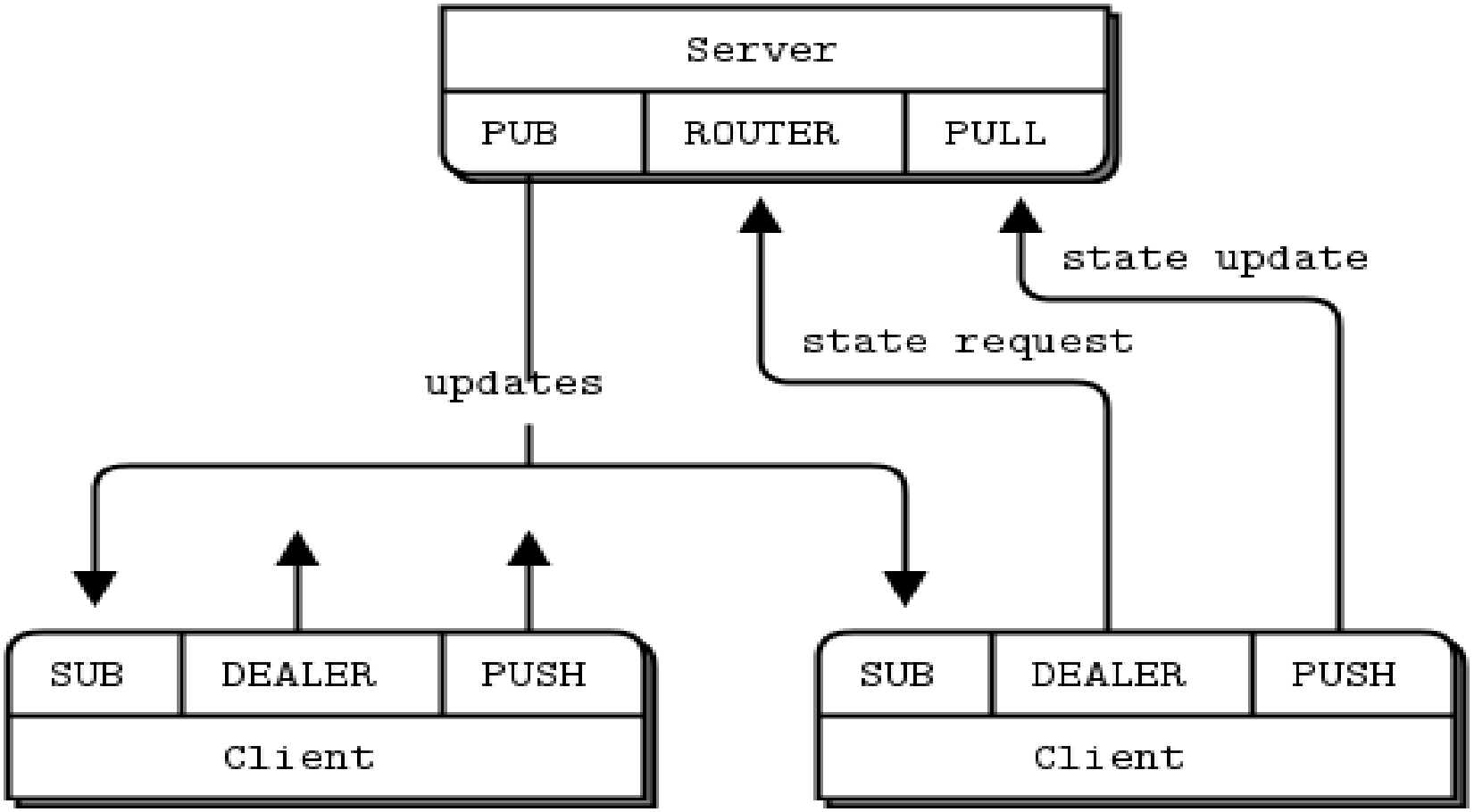
```
public static void main(String[] args) throws Exception {
        initState(State.valueOf(args[0]));
        ZContext ctx = new Zcontext();
        Selector selector = ctx.createSelector();
        ZMQ.Socket statepub = ctx.createSocket(SocketType.PUB);
        ZMQ.Socket statesub = ctx.createSocket(SocketType.SUB);
        ZMQ.Socket frontend =
ctx.createSocket(SocketType.ROUTER);
        statesub.subscribe("".getBytes());
        statepub.bind(args[1]);
        statesub.connect(args[2]);
        frontend.bind(args[3]);
        long sendStateAt = System.currentTimeMillis() + HEARTBEAT;
        long peerExpiry = -1;
        while (!Thread.currentThread().isInterrupted()) {
            ZMQ.PollItem[] items = {
                    new ZMQ.PollItem(statesub, ZMQ.Poller.POLLIN),
                    new ZMQ.PollItem(frontend, ZMQ.Poller.POLLIN)
            };
            long timeLeft = sendStateAt - System.currentTimeMillis();
            int rc = ZMQ.poll(selector,items, timeLeft < 0 ? 0 : timeLeft);
            if (rc == -1) {
                break;
            }
            if (items[0].isReadable()) {
                String message = statesub.recvStr();
                fire(Event.valueOf(PEER + message));
                peerExpiry = System.currentTimeMillis() + 2 * HEARTBEAT;
            }
            if (items[1].isReadable()) {
                fire(Event.CLIENT_REQUEST);
                if (peerExpiry != -1 && System.currentTimeMillis() >=
peerExpiry) {
                    fire(Event.CLIENT_VOTE);
                }
                ZMsg message = ZMsg.recvMsg(frontend);
                if(stateMachine.getState().equals(State.ACTIVE)) {
                    message.send(frontend);
                }
                peerExpiry = System.currentTimeMillis() + 2 * HEARTBEAT;
            }
            if (stateMachine.getState().equals(State.ERROR)) {
                break;
            }
            if (System.currentTimeMillis() >= sendStateAt) {
                statepub.send(stateMachine.getState().toString());
                sendStateAt = System.currentTimeMillis() + HEARTBEAT;
            }

        }
    }
```

Сервер публикации с состянием

- Данный пример реализует следущий сценарий :
- Клиент после подключения запрашивает у сервера публикаций текущие состояние.
- Клиент получает весь текущий массив key:value
- Далее клиент получает все сообщения по подписке на публикацию

Архитектура



Клиент

```
ZContext ctx = new ZContext();
Socket snapshot = ctx.createSocket(SocketType.DEALER);
snapshot.connect("tcp://localhost:5556");
Socket subscriber = ctx.createSocket(SocketType.SUB);
subscriber.connect("tcp://localhost:5557");
subscriber.subscribe("".getBytes());
Socket push = ctx.createSocket(SocketType.PUSH);
push.connect("tcp://localhost:5558");
// get state snapshot
long sequence = 0;
snapshot.send("ICANHAZ?".getBytes(), 0);
while (true) {
    kvsimple kvMsg = kvsimple.recv(snapshot);
    if (kvMsg == null) break; // Interrupted
    sequence = kvMsg.getSequence();
    if ("KTHXBAI".equalsIgnoreCase(kvMsg.getKey())) {
      System.out.println("Received snapshot = " + kvMsg.getSequence());
      break; // done
   }
    System.out.println("receiving " + kvMsg.getSequence());
    clonecli3.kvMap.put(kvMsg.getKey(), kvMsg);
}
Poller poller = ctx.createPoller(1);
poller.register(subscriber);
Random random = new Random();
```

```
• // now apply pending updates, discard out-of-sequence messages
  long alarm = System.currentTimeMillis() + 5000;
  while (true) {
     int rc = poller.poll(Math.max(0, alarm -
System.currentTimeMillis()));
     if (rc == -1) break; // Context has been shut down
     if (poller.pollin(0)) {
       kvsimple kvMsg = kvsimple.recv(subscriber);
       if (kvMsg == null) break; // Interrupted
       if (kvMsg.getSequence() > sequence) {
          sequence = kvMsg.getSequence();
          System.out.println("receiving " + sequence);
          clonecli3.kvMap.put(kvMsg.getKey(), kvMsg);
       }
     }
   if (System.currentTimeMillis() >= alarm) {
     int key = random.nextInt(10000);
     int body = random.nextInt(1000000);
     ByteBuffer b = ByteBuffer.allocate(4);
     b.asIntBuffer().put(body);
     kvsimple kvUpdateMsg = new kvsimple(key + "", 0, b.array());
     kvUpdateMsg.send(push);
     alarm = System.currentTimeMillis() + 1000;
   }
  }
  ctx.destroy();
```

Сервер

```
public void run() {
  ZContext ctx = new ZContext();
  Socket snapshot =
ctx.createSocket(SocketType.ROUTER);
  snapshot.bind("tcp://*:5556");
  Socket publisher = ctx.createSocket(SocketType.PUB);
  publisher.bind("tcp://*:5557");
  Socket collector = ctx.createSocket(SocketType.PULL);
  collector.bind("tcp://*:5558");
  Poller poller = ctx.createPoller(2);
  poller.register(collector, Poller.POLLIN);
  poller.register(snapshot, Poller.POLLIN);
  long sequence = 0;
  while (!Thread.currentThread().isInterrupted()) {
    if (poller.poll(1000) < 0)   break;
   // apply state updates from main thread
   if (poller.pollin(0)) {
      kvsimple kvMsg = kvsimple.recv(collector);
      if (kvMsg == null) break;
      kvMsg.setSequence(++sequence);
     kvMsg.send(publisher);
     clonesrv3.kvMap.put(kvMsg.getKey(), kvMsg);
     System.out.printf("I: publishing update %5d\n",
           sequence);
  }
```

-
```
// execute state snapshot request
if (poller.pollin(1)) {
   byte[] identity = snapshot.recv(0);
    if (identity == null) break; // Interrupted
    String request = snapshot.recvStr();
    if (!request.equals("ICANHAZ?")) {
       System.out.println("E: bad request, aborting");
       break;
    }
    Iterator<Entry<String, kvsimple>> iter = kvMap.entrySet().iterator();
    while (iter.hasNext()) {
       Entry<String, kvsimple> entry = iter.next();
       kvsimple msg = entry.getValue();
       System.out.println("Sending message " +
            entry.getValue().getSequence());
       this.sendMessage(msg, identity, snapshot);
    }
   // now send end message with sequence number
    System.out.println("Sending state snapshot = " + sequence);
    snapshot.send(identity, ZMQ.SNDMORE);
    kvsimple message = new kvsimple("KTHXBAI", sequence, "".getBytes());
    message.send(snapshot);
 }
}
System.out.printf (" Interrupted\n%d messages handled\n", sequence);
ctx.destroy();
}
private void sendMessage(kvsimple msg, byte[] identity, Socket snapshot) {
snapshot.send(identity, ZMQ.SNDMORE);
msg.send(snapshot);
}
```

Класс для хранения key:value

```
public class kvsimple {
private final String key;
private long sequence;
private final byte[] body;
public kvsimple(String key, long sequence, byte[] body) {
  this.key = key; this.sequence = sequence; this.body = body;
}
public String getKey() { return key;}
public long getSequence() { return sequence; }
public void setSequence(long sequence) { this.sequence = sequence;}
public byte[] getBody() {    return body; }
public void send(Socket publisher) {
   publisher.send(key.getBytes(), ZMQ.SNDMORE);
  ByteBuffer bb = ByteBuffer.allocate(8);
  bb.asLongBuffer().put(sequence);
  publisher.send(bb.array(), ZMQ.SNDMORE);
  publisher.send(body, 0);
}
public static kvsimple recv(Socket updates) {
   byte [] data = updates.recv(0);
   if (data == null || !updates.hasReceiveMore()) return null;
  String key = new String(data);
  data = updates.recv(0);
  if (data == null || !updates.hasReceiveMore()) return null;
  Long sequence = ByteBuffer.wrap(data).getLong();
  byte[] body = updates.recv(0);
  if (body == null || updates.hasReceiveMore())  return null;
  return new kvsimple(key, sequence, body);
}
```

```
public String toString() {
return "kvsimple [key=" + key + ", sequence=" + sequence + ",
body=" + Arrays.toString(body) + "]";
}
public int hashCode() {
 final int prime = 31;
 int result = 1;
 result = prime * result + Arrays.hashCode(body);
 result = prime * result + ((key == null) ? 0 : key.hashCode());
 result = prime * result + (int) (sequence ^ (sequence >>> 32));
 return result;
}
public boolean equals(Object obj) {
   if (this == obj) return true;
   if (obj == null) return false;
   if (getClass() != obj.getClass()) return false;
   kvsimple other = (kvsimple) obj;
  if (!Arrays.equals(body, other.body)) return false;
  if (key == null) {
      if (other.key != null) return false;
  } else if (!key.equals(other.key)) return false;
  if (sequence != other.sequence) return false;
  return true;
}
```