

基于SpringCloud+Netty实现的在线网络聊天室

项目地址

简介

后端模块架构

架构设计

技术

重要流程

客户端连接流程

控制台客户端部分代码

chat-room-web-server服务部分源码

注册流程

用户注册请求入栈消息处理器

用户服务实现类

登录流程

用户登录请求入栈消息处理器

用户服务实现类

消息发送流程

聊天请求入栈消息处理器

SessionClusterImpl类(implements Session) 的isLogin方法

NettyController

NettyServiceImpl类的chatRequestMessageSend方法

群聊消息发送流程

集群群聊聊天请求入栈消息处理器

GroupSessionClusterImpl的getMembersAndHost方法

RedisServiceImpl的getMembersAndHost方法

NettyServiceImpl的sendGroupChatMessage方法

群聊创建流程

群聊创建请求入栈消息处理器

GroupSessionClusterImpl的hasGroup方法

GroupSessionClusterImpl的createGroup方法

RedisServiceImpl的createGroup方法

NettyController

NettyServiceImpl的sendGroupCreateMessage方法

ReBalance机制

问题说明

解决方案

关键代码

消息生产者接口

实现类

netty服务端

消息消费者

ReBalanceService接口

ReBalanceServiceImpl

ReBalanceController

ReBalanceService接口

ReBalanceServiceImpl

SessionClusterImpl实现类的reBalance方法

未完成事项和存在的问题

未完成的事项

项目地址

https://github.com/maomao124/netty_chat_room

简介

基于SpringCloud+Netty实现的在线网络聊天室。netty服务是集群部署的，netty共享channel解决方案。

用户方面，有用户登录、用户注册、发送消息、发送群聊消息、创建群聊、加入群聊、查看群聊成员、退出群聊等功能；

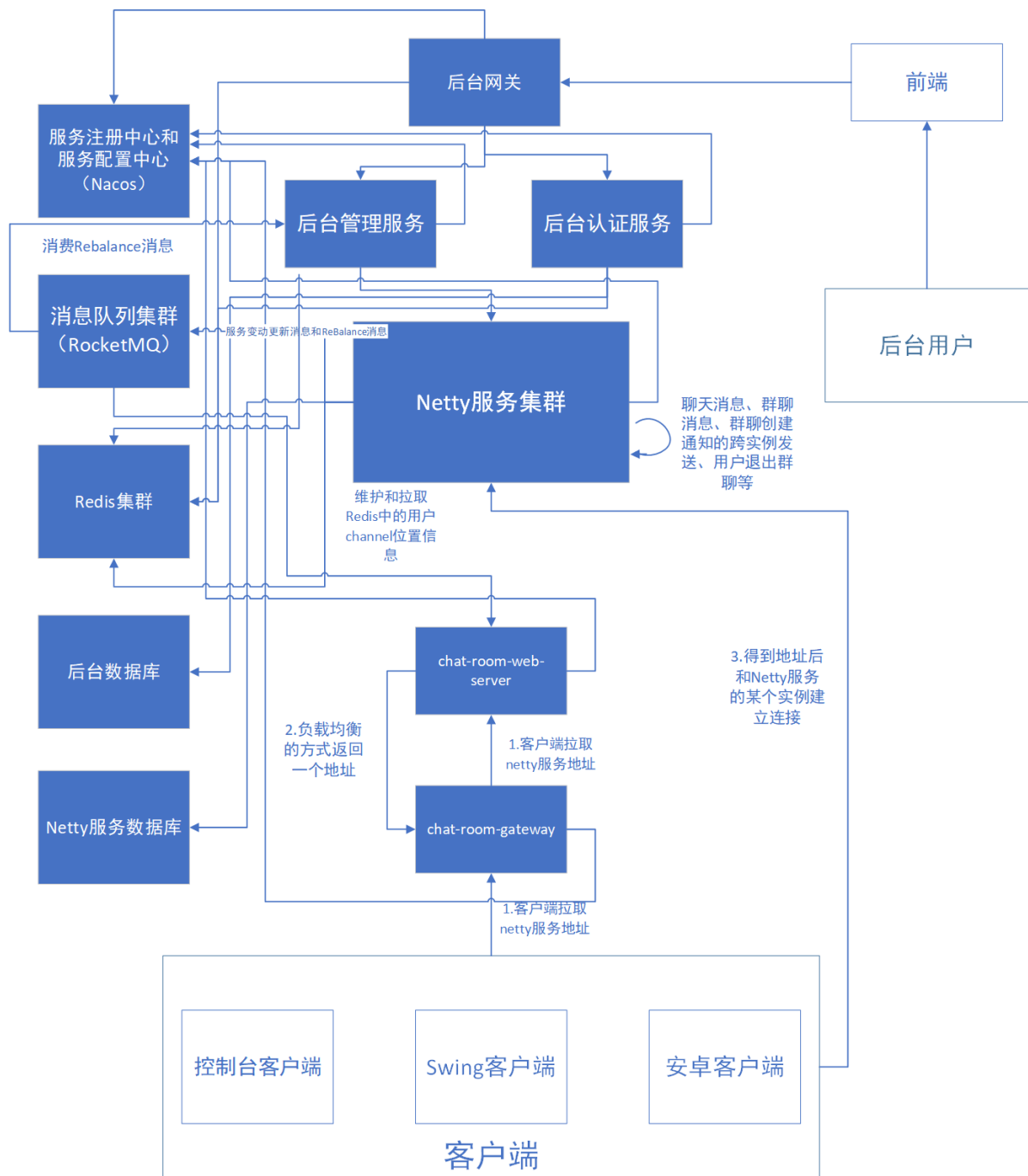
后台方面，有登录次数统计、登录UV统计、注册统计、消息发送统计、群聊消息发送统计、群聊创建统计、Netty服务ReBalance、用户管理、后台用户、资源、角色、菜单、组织、岗位管理、后台用户认证和鉴权等功能。

后端模块架构

1	authority	#聚合工程，用于聚合parent、apps、tools等模块
2	├─ parent	# 父工程，nacos配置及依赖包管理
3	├─ apps	# 应用目录

4	└─ auth	# 权限服务父工程
5	└─ auth-entity	# 权限实体
6	└─ auth-server	# 权限服务
7	└─ gateway	# 后台网关服务
8	└─ chat-room	# 在线聊天室应用
9	└─ chat-room-client-api	# 客户端api模块，放不同类型、不同平台客户端的公共代码的模块
10	└─ chat-room-common	# 在线聊天室公共模块，放客户端和服务端公共代码的模块
11	└─ chat-room-console-client	# 在线聊天室控制台客户端
12	└─ chat-room-console-test-client	# 在线聊天室控制台压力测试客户端(不给用户使用)
13	└─ chat-room-gateway	# 在线聊天室用户http服务网关
14	└─ chat-room-manage	# 聊天室后台管理服务和统计服务
15	└─ chat-room-netty-server	# 在线聊天室netty服务
16	└─ chat-room-server-api	# 服务端api模块，放不同类型、不同平台服务端的公共代码的模块
17	└─ chat-room-swing-client	# 在线聊天室java swing客户端
18	└─ chat-room-web-server	# 在线聊天室web服务，向用户提供http服务
19	└─ tools	# 工具工程
20	└─ tools-common	# 基础组件：基础配置类、函数、常量、统一异常处理、undertow服务器
21	└─ tools-core	# 核心组件：基础实体、返回对象、上下文、异常处理、分布式锁、函数、树
22	└─ tools-databases	# 数据源组件：数据源配置、数据权限、查询条件等
23	└─ tools-dozer	# 对象转换：dozer配置、工具
24	└─ tools-redis-cache	# redis分布式缓存工具类和分布式锁服务，缓存工具类解决著名的3个缓存问题
25	└─ tools-j2cache	# 缓存组件：j2cache、redis缓存
26	└─ tools-jwt	# JWT组件：配置、属性、工具
27	└─ tools-log	# 日志组件：日志实体、事件、拦截器、工具
28	└─ tools-swagger2	# 文档组件：knife4j文档
29	└─ tools-user	# 用户上下文：用户注解、模型和工具，当前登录用户信息注入模块
30	└─ tools-validator	# 表单验证：后台表单规则验证
31	└─ tools-xss	# xss防注入组件

架构设计



技术

- 注册中心和配置中心：Nacos
- 消息队列：RocketMQ
- 关系数据库：MYSQL
- 缓存服务：Redis
- 二级缓存：j2cache
- 负载均衡远程调用：Feign

- 定向调用: RestTemplate
- 网络框架: Netty
- 对象转换: Dozer
- 分布式锁: redisson
- 防XSS攻击: antisamy
- 接口文档: knife4j swagger
- 令牌生成和解析: jjwt
- 网关: zuul
- 验证码生成: captcha
-

重要流程

java代码有48713行，只列举一部分重要的流程

客户端连接流程

1. 客户端请求http服务网关，拉取netty服务实例地址
2. http服务网关负载均衡到chat-room-web-server服务
3. chat-room-web-server服务从Nacos拉取chat-room-netty-server服务实例列表，并通过负载均衡的方式返回其中一个实例（有二级缓存）
4. 客户端接收到网关响应的服务实例
5. 客户端根据响应的实例地址连接该实例

控制台客户端部分代码

```
1 public static void main(String[] args)
2 {
3     RestfulHTTP restfulHTTP = MainApplication.getRestfulHTTP();
4     System.out.println("服务器URL: " + ClientConfig.getServerUrl());
5     System.out.println("正在等待服务器响应...");
6     //todo:超时重试，错误重试，有限重试
7     R<String> r = restfulHTTP.GET(R.class, ClientConfig.getServerUrl(),
8     null, null);
9     if (r.getIsError())
10    {
11        //错误
12        System.out.println("获取netty服务时错误: " + r.getMsg());
13        Toolkit.getDefaultToolkit().beep();
14    }
```

```

13         return;
14     }
15     Server server = r.getData(Server.class);
16     String ip = server.getIp();
17     Integer port = server.getPort();
18     System.out.println(ip + ":" + port);
19
20     NioEventLoopGroup group = new NioEventLoopGroup();
21     LoggingHandler LOGGING_HANDLER = new
LoggingHandler(LogLevel.DEBUG);
22     ClientMessageCodecSharable clientMessageCodecSharable = new
ClientMessageCodecSharable();
23
24     PingResponseMessageHandler pingResponseMessageHandler = new
PingResponseMessageHandler();
25     LoginResponseMessageHandler loginResponseMessageHandler = new
LoginResponseMessageHandler();
26     RegisterResponseMessageHandler registerResponseMessageHandler = new
RegisterResponseMessageHandler();
27     ChatResponseMessageHandler chatResponseMessageHandler = new
ChatResponseMessageHandler();
28     GroupChatResponseMessageHandler groupChatResponseMessageHandler =
new GroupChatResponseMessageHandler();
29     GroupCreateResponseMessageHandler groupCreateResponseMessageHandler
= new GroupCreateResponseMessageHandler();
30     GroupMembersResponseMessageHandler
groupMembersResponseMessageHandler = new
GroupMembersResponseMessageHandler();
31     GroupJoinResponseMessageHandler groupJoinResponseMessageHandler =
new GroupJoinResponseMessageHandler();
32     GroupQuitResponseMessageHandler groupQuitResponseMessageHandler =
new GroupQuitResponseMessageHandler();
33
34     Bootstrap bootstrap = new Bootstrap();
35     ChannelFuture channelFuture = bootstrap.group(group)
36         .channel(NioSocketChannel.class)
37         .handler(new ChannelInitializer<NioSocketChannel>()
38         {
39             @Override
40             protected void initChannel(NioSocketChannel ch) throws
Exception
41             {
42                 ch.pipeline().addLast(LOGGING_HANDLER)
43                     .addLast(new ProtocolFrameDecoder())
44                     .addLast(clientMessageCodecSharable)
45                     .addLast(pingResponseMessageHandler)
46                     .addLast(loginResponseMessageHandler)
47                     .addLast(registerResponseMessageHandler)
48                     .addLast(chatResponseMessageHandler)
49                     .addLast(groupChatResponseMessageHandler)
50                     .addLast(groupCreateResponseMessageHandler)
51
.addLast(groupMembersResponseMessageHandler)
52                     .addLast(groupJoinResponseMessageHandler)
53                     .addLast(groupQuitResponseMessageHandler);

```

```

54         }
55         }).connect(new InetSocketAddress(ip, port));
56
57         channel = channelFuture.channel();
58
59         thread = new LoginAndRegisterThread(channel);
60
61         channelFuture.addListener(new GenericFutureListener<Future<? super
void>>>()
62         {
63             @Override
64             public void operationComplete(Future<? super void> future)
throws Exception
65             {
66                 if (future.isSuccess())
67                 {
68                     System.out.println("客户端启动成功");
69                     thread.start();
70                 }
71                 else
72                 {
73                     String message = future.cause().getMessage();
74                     System.out.println("错误: " + message);
75                     Toolkit.getDefaultToolkit().beep();
76                 }
77             }
78         });
79
80         channel.closeFuture().addListener(new
GenericFutureListener<Future<? super void>>>()
81         {
82             @Override
83             public void operationComplete(Future<? super void> future)
throws Exception
84             {
85                 group.shutdownGracefully();
86             }
87         });
88
89         Runtime.getRuntime().addShutdownHook(new Thread(new Runnable()
90         {
91             @Override
92             public void run()
93             {
94                 try
95                 {
96                     channel.close();
97                 }
98                 catch (Exception ignored)
99                 {
100
101                 }
102                 try
103                 {
104                     group.shutdownGracefully();

```

```

105         }
106         catch (Exception ignored)
107         {
108
109         }
110     }
111 }));
112 }

```

chat-room-web-server服务部分源码

```

1  package mao.chat_room_web_server.service.impl;
2
3  import com.alibaba.fastjson.JSON;
4  import com.alibaba.fastjson.JSONArray;
5  import lombok.SneakyThrows;
6  import lombok.extern.slf4j.Slf4j;
7  import mao.chat_room_common.entity.Server;
8  import mao.chat_room_server_api.constants.CacheConstants;
9  import mao.chat_room_server_api.constants.ServerConstants;
10 import mao.chat_room_server_api.constants.UrlConstants;
11 import mao.chat_room_server_api.utils.ClusterUtils;
12 import mao.chat_room_web_server.service.NettyService;
13 import mao.tools_core.base.R;
14 import mao.tools_core.exception.BizException;
15 import net.oschina.j2cache.CacheChannel;
16 import net.oschina.j2cache.CacheObject;
17 import org.springframework.cloud.client.ServiceInstance;
18 import org.springframework.stereotype.Service;
19 import org.springframework.web.client.RestTemplate;
20
21 import javax.annotation.Resource;
22 import java.nio.file.Path;
23 import java.util.ArrayList;
24 import java.util.List;
25 import java.util.Random;
26 import java.util.concurrent.*;
27 import java.util.concurrent.atomic.AtomicBoolean;
28
29 /**
30  * Project name(项目名称): netty_chat_room
31  * Package(包名): mao.chat_room_web_server.service.impl
32  * Class(类名): NettyServiceImpl
33  * Author(作者): mao
34  * Author QQ: 1296193245
35  * GitHub: https://github.com/maomao124/
36  * Date(创建日期): 2023/4/1
37  * Time(创建时间): 20:49
38  * Version(版本): 1.0
39  * Description(描述): 无
40  */

```



```

41
42 @Slf4j
43 @Service
44 public class NettyServiceImpl implements NettyService
45 {
46
47     @Resource
48     private CacheChannel cacheChannel;
49
50     @Resource
51     private ClusterUtils clusterUtils;
52
53     @Resource
54     private RestTemplate restTemplate;
55
56     private final ThreadPoolExecutor threadPoolExecutor = new
ThreadPoolExecutor(150,
57                     150, 0L, TimeUnit.MILLISECONDS,
58                     new LinkedBlockingQueue<Runnable>(100));
59
60
61     /**
62      * 得到int随机数
63      *
64      * @param min 最小值
65      * @param max 最大值
66      * @return int
67      */
68     public static int getIntRandom(int min, int max)
69     {
70         if (min > max)
71         {
72             min = max;
73         }
74         Random random = new Random();
75         return random.nextInt(max - min + 1) + min;
76     }
77
78     @SneakyThrows
79     @Override
80     public Server getNettyServerAddress()
81     {
82         //得到实例列表
83         CacheObject cacheObject =
cacheChannel.get(CacheConstants.chat_server_key, "1");
84         //判断是否为空
85         if (cacheObject == null || cacheObject.getValue() == null)
86         {
87             //空，需要加载
88             List<ServiceInstance> serviceInstances =
clusterUtils.getServiceInstances(ServerConstants.CHAT_ROOM_NETTY_SERVER);
89             if (serviceInstances == null || serviceInstances.size() == 0)
90             {
91                 throw BizException.wrap("无法获取聊天服务器地址！请稍后在试");
92             }

```

```

93         //不是空
94         int size = serviceInstances.size();
95         CountDownLatch countDownLatch = new CountDownLatch(size);
96         List<Server> list = new ArrayList<>(size);
97         log.debug("加载服务实例");
98         AtomicBoolean isSuccess = new AtomicBoolean(true);
99         for (ServiceInstance serviceInstance : serviceInstances)
100         {
101             threadPoolExecutor.submit(() ->
102             {
103                 try
104                 {
105                     String host = serviceInstance.getHost();
106                     int port = serviceInstance.getPort();
107                     String url = UrlConstants.buildGetPortUrl(
108                         serviceInstance.getHost() + ":" + port);
109                     R r = restTemplate.getForObject(url, R.class);
110                     if (r.getIsError())
111                     {
112                         log.warn(r.getMsg());
113                         isSuccess.set(false);
114                     }
115                     else
116                     {
117                         Integer nettyPort =
118 Integer.valueOf(r.getData().toString());
119                         Server server = new Server()
120                             .setIp(host)
121                             .setPort(nettyPort);
122                         synchronized (list)
123                         {
124                             list.add(server);
125                         }
126                     }
127                     finally
128                     {
129                         countDownLatch.countDown();
130                     }
131                 }
132             });
133         }
134         countDownLatch.await();
135         if (isSuccess.get())
136         {
137             //请求成功
138             //判断是否有数据
139             if (list.size() == 0)
140             {
141                 //无数据
142                 throw BizException.wrap("netty服务集群暂时都不可用，请稍后再
143 试");
144             }
145             String json = JSON.toJSONString(list);
146             log.debug(json);

```

```

146         cacheChannel.set(CacheConstants.chat_server_key, "1",
147         json);
148         return list.get(getIntRandom(0, list.size() - 1));
149     }
150     else
151     {
152         if (list.size() == 0)
153         {
154             throw BizException.wrap("无法获取聊天服务器地址！请稍后在
155             试");
156         }
157         else
158         {
159             return list.get(getIntRandom(0, list.size() - 1));
160         }
161     }
162     else
163     {
164         //不为空
165         String json = cacheObject.getValue().toString();
166         List<Server> list = JSON.parseArray(json, Server.class);
167         return list.get(getIntRandom(0, list.size() - 1));
168     }
169 }
170 @Override
171 public void removeCache()
172 {
173     cacheChannel.clear(CacheConstants.chat_server_key);
174 }
175 }

```

注册流程

1. 客户端向netty服务发起注册消息包，包含用户名和密码
2. 服务端判断用户名是否为空
3. 服务端判断密码是否为空
4. 服务端判断用户名长度是否小于3位
5. 服务端判断密码长度是否小于6位
6. 服务端判断是否为保留字段
7. 服务端查询数据库
8. 服务端判断用户名是否存在，如果存在，用户名已被占用
9. 服务端将数据插入到数据库
10. 服务端返回数据给客户端

用户注册请求入栈消息处理器

```
1 package mao.chat_room_netty_server.handler_cluster;
2
3 import io.netty.channel.ChannelHandler;
4 import lombok.extern.slf4j.Slf4j;
5 import mao.chat_room_netty_server.handler.RegisterRequestMessageHandler;
6 import mao.chat_room_netty_server.service.RedisService;
7 import mao.chat_room_netty_server.service.UserService;
8 import mao.chat_room_netty_server.session.GroupSession;
9 import mao.chat_room_netty_server.session.Session;
10 import org.springframework.stereotype.Service;
11
12 import javax.annotation.Resource;
13
14 /**
15  * Project name(项目名称): netty_chat_room
16  * Package(包名): mao.chat_room_netty_server.handler_cluster
17  * Class(类名): ClusterRegisterRequestMessageHandler
18  * Author(作者): mao
19  * Author QQ: 1296193245
20  * Github: https://github.com/maomao124/
21  * Date(创建日期): 2023/4/8
22  * Time(创建时间): 14:42
23  * Version(版本): 1.0
24  * Description(描述): 集群用户注册请求入栈消息处理器
25  */
26
27 @Slf4j
28 @Service
29 @ChannelHandler.Sharable
30 public class ClusterRegisterRequestMessageHandler extends
    RegisterRequestMessageHandler
31 {
32     @Resource
33     private UserService userService;
34
35     @Resource
36     private Session session;
37
38     @Resource
39     private GroupSession groupSession;
40
41     @Resource
42     private RedisService redisService;
43 }
```

```
1 package mao.chat_room_netty_server.handler;
2
3 import io.netty.channel.ChannelHandler;
4 import io.netty.channel.ChannelHandlerContext;
5 import io.netty.channel.SimpleChannelInboundHandler;
```

```

6  import lombok.extern.slf4j.Slf4j;
7  import mao.chat_room_common.message.RegisterRequestMessage;
8  import mao.chat_room_common.message.RegisterResponseMessage;
9  import mao.chat_room_netty_server.service.RedisService;
10 import mao.chat_room_netty_server.service.UserService;
11 import mao.chat_room_netty_server.session.GroupSession;
12 import mao.chat_room_netty_server.session.Session;
13 import mao.tools_core.exception.BizException;
14 import org.springframework.stereotype.Service;
15
16 import javax.annotation.Resource;
17
18 /**
19  * Project name(项目名称): netty_chat_room
20  * Package(包名): mao.chat_room_netty_server.handler
21  * Class(类名): RegisterRequestMessageHandler
22  * Author(作者): mao
23  * Author QQ: 1296193245
24  * GitHub: https://github.com/maomao124/
25  * Date(创建日期): 2023/3/30
26  * Time(创建时间): 15:13
27  * Version(版本): 1.0
28  * Description(描述): 用户注册请求入栈消息处理器
29  */
30
31 @Slf4j
32 //@Service
33 @ChannelHandler.Sharable
34 public class RegisterRequestMessageHandler extends
SimpleChannelInboundHandler<RegisterRequestMessage>
35 {
36     @Resource
37     private UserService userService;
38
39     @Resource
40     private Session session;
41
42     @Resource
43     private GroupSession groupSession;
44
45     @Resource
46     private RedisService redisService;
47
48
49     @Override
50     protected void channelRead0(ChannelHandlerContext ctx,
51 RegisterRequestMessage
registerRequestMessage) throws Exception
52     {
53         String username = registerRequestMessage.getUsername();
54         String password = registerRequestMessage.getPassword();
55         try
56         {
57             boolean register = userService.register(username, password);
58             if (register)

```

```

59         {
60             //注册成功
61             //响应
62             ctx.writeAndFlush(RegisterResponseMessage.success()
63                 .setReason("注册成功! 请登录"));
64
65             .setSequenceId(registerRequestMessage.getSequenceId()));
66             //统计
67             redisService.registerCount();
68         }
69         else
70         {
71             //注册失败
72             ctx.writeAndFlush(RegisterResponseMessage.fail("注册失败!"));
73
74             .setSequenceId(registerRequestMessage.getSequenceId()));
75         }
76         catch (BizException e)
77         {
78             //错误消息
79             String message = e.getMessage();
80             //注册失败
81             ctx.writeAndFlush(RegisterResponseMessage.fail(message)
82                 .setSequenceId(registerRequestMessage.getSequenceId()));
83         }
84         catch (Exception e)
85         {
86             log.error("服务器错误: ", e);
87             ctx.writeAndFlush(RegisterResponseMessage.fail("服务器错误! 请稍后
88 在试! "));
89             .setSequenceId(registerRequestMessage.getSequenceId()));
90         }
91     }
92 }

```

用户服务实现类

```

1  @Override
2  @Transactional
3  public boolean register(String username, String password)
4  {
5      if (username == null || username.equals(""))
6      {
7          throw new BizException("用户名不能为空");
8      }
9      if (password == null || password.equals(""))
10     {
11         throw new BizException("密码不能为空");
12     }
13     if (username.length() < 3)

```

```

14     {
15         throw new BizException("用户名长度不能小于3位");
16     }
17     if (password.length() < 6)
18     {
19         throw new BizException("密码长度不能小于6位");
20     }
21     //判断是否为保留字段
22     if (username.equals("host"))
23     {
24         throw new BizException("用户名\"host\"为系统保留字段，不能使用");
25     }
26     //查询数据库
27     User user = this.getOne(wraps.<User>1bq().eq(User::getUsername,
28 username));
29     //如果为空，就不存在
30     if (user != null)
31     {
32         //判断用户名是否存在
33         if (user.getUsername().equals(username))
34         {
35             throw new BizException("该用户名\"\" + username + "\"已被占用！ 换一个
36 用户名吧");
37         }
38     }
39     User user1 = new User()
40         .setUsername(username)
41         .setPassword(passwordEncoderService.encoder(password))
42         .setStatus(true)
43         .setRegisterTime(LocalDateTime.now());
44     //插入
45     return this.save(user1);
46 }

```

登录流程

1. 客户端向netty服务发起登录消息包，包含用户名和密码
2. 服务端检查登录状态，禁止在多台设备上同时登录
3. 服务端判断用户名是否为空
4. 服务端判断密码是否为空
5. 服务端判断用户名长度是否小于3位
6. 服务端判断密码长度是否小于6位
7. 服务端查询数据库
8. 服务端判断查询是否为空，如果为空，用户名不存在
9. 服务端判断判断密码错误次数是否大于3次
 1. 判断密码输入间隔是否小于10分钟

2. 如果是，证明10分钟内尝试过，返回密码错误次数过多，请10分钟后再试!
 3. 如果不是，执行下一步
10. 验证密码是否正确
 11. 判断启用状态，如果为非启用状态，证明该账号已被禁用
 12. 更新登录时间
 13. 服务端响应数据给客户端

用户登录请求入栈消息处理器

```
1 package mao.chat_room_netty_server.handler_cluster;
2
3 import io.netty.channel.ChannelHandler;
4 import lombok.extern.slf4j.Slf4j;
5 import mao.chat_room_netty_server.handler.LoginRequestMessageHandler;
6 import mao.chat_room_netty_server.service.RedisService;
7 import mao.chat_room_netty_server.service.UserService;
8 import mao.chat_room_netty_server.session.GroupSession;
9 import mao.chat_room_netty_server.session.Session;
10 import org.springframework.stereotype.Service;
11
12 import javax.annotation.Resource;
13
14 /**
15  * Project name(项目名称): netty_chat_room
16  * Package(包名): mao.chat_room_netty_server.handler_cluster
17  * Class(类名): ClusterLoginRequestMessageHandler
18  * Author(作者): mao
19  * Author QQ: 1296193245
20  * GitHub: https://github.com/maomao124/
21  * Date(创建日期): 2023/4/8
22  * Time(创建时间): 14:39
23  * Version(版本): 1.0
24  * Description(描述): 集群用户登录请求入栈消息处理器
25  */
26
27 @Slf4j
28 @Service
29 @ChannelHandler.Sharable
30 public class ClusterLoginRequestMessageHandler extends
    LoginRequestMessageHandler
31 {
32     @Resource
33     private UserService userService;
34
35     @Resource
36     private Session session;
37
38     @Resource
39     private GroupSession groupSession;
40 }
```



```

41     @Resource
42     private RedisService redisService;
43
44 }

```

```

1  package mao.chat_room_netty_server.handler;
2
3  import io.netty.channel.ChannelHandler;
4  import io.netty.channel.ChannelHandlerContext;
5  import io.netty.channel.SimpleChannelInboundHandler;
6  import lombok.extern.slf4j.Slf4j;
7  import mao.chat_room_common.message.GroupChatResponseMessage;
8  import mao.chat_room_common.message.LoginRequestMessage;
9  import mao.chat_room_common.message.LoginResponseMessage;
10 import mao.chat_room_netty_server.service.RedisService;
11 import mao.chat_room_netty_server.service.UserService;
12 import mao.chat_room_netty_server.session.GroupSession;
13 import mao.chat_room_netty_server.session.Session;
14 import mao.chat_room_server_api.entity.User;
15 import mao.tools_core.exception BizException;
16 import org.springframework.stereotype.Service;
17
18 import javax.annotation.Resource;
19
20 /**
21  * Project name(项目名称): netty_chat_room
22  * Package(包名): mao.chat_room_netty_server.handler
23  * Class(类名): LoginRequestMessageHandler
24  * Author(作者): mao
25  * Author QQ: 1296193245
26  * GitHub: https://github.com/maomao124/
27  * Date(创建日期): 2023/3/30
28  * Time(创建时间): 14:59
29  * Version(版本): 1.0
30  * Description(描述): 用户登录请求入栈消息处理器
31  */
32
33 @Slf4j
34 //@Service
35 @ChannelHandler.Sharable
36 public class LoginRequestMessageHandler extends
SimpleChannelInboundHandler<LoginRequestMessage>
37 {
38
39     @Resource
40     private UserService userService;
41
42     @Resource
43     private Session session;
44
45     @Resource
46     private GroupSession groupSession;
47

```

```

48     @Resource
49     private RedisService redisService;
50
51     @Override
52     protected void channelRead0(ChannelHandlerContext ctx,
LoginRequestMessage loginRequestMessage) throws Exception
53     {
54         String username = loginRequestMessage.getUsername();
55         String password = loginRequestMessage.getPassword();
56
57         //检查登录状态
58         if (session.isLogin(username))
59         {
60             //已登录
61             ctx.writeAndFlush(LoginResponseMessage.fail("禁止在多台设备上同时登
录!"))
62                 .setSequenceId(loginRequestMessage.getSequenceId());
63             return;
64         }
65         try
66         {
67             User user = userService.login(username, password);
68             //登录成功,绑定
69             session.bind(ctx.channel(), username);
70             //响应
71             ctx.writeAndFlush(LoginResponseMessage.success()
72                 .setUsername(username)
73                 .setSequenceId(loginRequestMessage.getSequenceId()));
74             log.debug("用户" + username + "登录成功");
75             //登录统计
76             redisService.loginCount(username);
77         }
78         catch (BizException e)
79         {
80             //得到异常消息
81             String message = e.getMessage();
82             //登录失败
83             ctx.writeAndFlush(LoginResponseMessage.fail(message)
84                 .setSequenceId(loginRequestMessage.getSequenceId()));
85             log.debug("用户" + username + "登录失败");
86         }
87     }
88     catch (Exception e)
89     {
90         log.error("登录过程中服务器错误: ", e);
91         ctx.writeAndFlush(LoginResponseMessage.fail("服务器错误! 请稍后在
试! "))
92             .setSequenceId(loginRequestMessage.getSequenceId());
93     }
94 }
95 }

```

用户服务实现类

```
1  @Resource
2  private PasswordEncoderService passwordEncoderService;
3
4  @Resource
5  private TransactionTemplate transactionTemplate;
6
7  @Resource
8  private PlatformTransactionManager platformTransactionManager;
9
10 @Resource
11 private DozerUtils dozerUtils;
12
13 @Override
14 @Transactional(noRollbackFor = {BizException.class})
15 public User login(String username, String password)
16 {
17     if (username == null || username.equals(""))
18     {
19         throw new BizException("用户名不能为空");
20     }
21     if (password == null || password.equals(""))
22     {
23         throw new BizException("密码不能为空");
24     }
25     if (username.length() < 3)
26     {
27         throw new BizException("用户名长度不能小于3位");
28     }
29     if (password.length() < 6)
30     {
31         throw new BizException("密码长度不能小于6位");
32     }
33     //查询数据库
34     User user = this.getOne(wraps.<User>lbq().eq(User::getUsername,
35 username));
36     if (user == null)
37     {
38         throw new BizException("用户名不存在");
39     }
40     //判断密码错误次数是否大于3次
41     if (user.getPasswordErrorNum() > 3)
42     {
43         //判断密码输入间隔是否小于10分钟
44         LocalDateTime passwordErrorLastTime =
45 user.getPasswordErrorLastTime();
46         LocalDateTime now = LocalDateTime.now();
47         //加10分钟,是否晚于现在时间,如果是,证明10分钟内尝试过
48         if (passwordErrorLastTime.plusMinutes(10).isAfter(now))
49         {
50             //第四次输入可能会跳过
51             throw new BizException("密码错误次数过多, 请10分钟后再试!");
52         }
53     }
54 }
```

```

52
53     //验证密码是否正确
54     boolean verification = passwordEncoderService.verification(password,
user.getPassword());
55     if (!verification)
56     {
57         //密码错误
58         this.update(wraps.<User>lbU()
                    .eq(User::getUsername, username)
59         //密码错误时间
60         .set(User::getPasswordErrorLastTime, LocalDateTime.now())
61         //密码错误次数
62         .set(User::getPasswordErrorNum, user.getPasswordErrorNum() +
1));
63
64         //提交事务
65         throw new BizException("密码错误");
66     }
67     //密码正确,判断启用状态
68     if (!user.getStatus())
69     {
70         //未启用
71         throw new BizException("该账号已被禁用");
72     }
73     //更新登录时间
74     this.update(wraps.<User>lbU()
                    .eq(User::getUsername, username)
75         .set(User::getLastLoginTime, LocalDateTime.now())
76         //将密码错误次数更改成0
77         .set(User::getPasswordErrorNum, 0));
78
79     //密码设空并返回
80     return user.setPassword(null);
81 }

```

消息发送流程

1. 客户端向netty服务发起消息发送请求消息包, 包含from (谁发送的)、to (发送给谁) 和content (消息内容)
2. netty服务端接收到数据包, 处理数据包, 进入聊天请求入栈消息处理器
3. 服务端检查登录状态
4. 服务端判断from是否为空
5. 服务端判断to是否为空
6. 服务端校验身份
7. 服务端判断是否是自己发送给自己
8. 服务端查询对方用户在当前实例上是否存在

9. 如果存在，证明对方用户在本实例上且在线，直接向对方channel写数据，并响应给消息发送者，统计消息发送次数，结束
10. 如果不存在，证明本地不在线或者不存在，需要往下执行
11. 从redis上查询其他实例的信息，得到对方用户在那一台实例上（host），判断用户是否在线
12. 如果其他实例都不在线或者不存在，响应给发送者"对方用户不存在或者不在线"
13. 如果其他实例在线，根据查询到的host发起http请求，让其它实例处理
14. 其它实例发送给接收者，如果没有问题，就会响应给服务调用者成功的状态
15. 服务端根据http请求结果，响应给发送者相应的结果
16. 如果是失败，响应给发送者"服务器错误"的错误消息
17. 如果是成功，响应给消息发送者成功的消息，统计消息发送次数，结束

聊天请求入栈消息处理器

```
1 package mao.chat_room_netty_server.handler_cluster;
2
3 import io.netty.channel.Channel;
4 import io.netty.channel.ChannelHandler;
5 import io.netty.channel.ChannelHandlerContext;
6 import lombok.extern.slf4j.Slf4j;
7 import mao.chat_room_common.message.ChatRequestMessage;
8 import mao.chat_room_common.message.ChatResponseMessage;
9 import mao.chat_room_netty_server.handler.ChatRequestMessageHandler;
10 import mao.chat_room_netty_server.service.RedisService;
11 import mao.chat_room_netty_server.session.Session;
12 import mao.chat_room_server_api.constants.UrlConstants;
13 import mao.tools_core.base.R;
14 import org.springframework.http.ResponseEntity;
15 import org.springframework.stereotype.Service;
16 import org.springframework.web.client.RestTemplate;
17
18 import javax.annotation.Resource;
19
20 /**
21  * Project name(项目名称): netty_chat_room
22  * Package(包名): mao.chat_room_netty_server.handler_cluster
23  * Class(类名): ClusterChatRequestMessageHandler
24  * Author(作者): mao
25  * Author QQ: 1296193245
26  * GitHub: https://github.com/maomao124/
27  * Date(创建日期): 2023/4/1
28  * Time(创建时间): 16:09
29  * Version(版本): 1.0
30  * Description(描述): 聊天请求入栈消息处理器
31  */
32
33 @Slf4j
```

```

34 @Service //这里应该添加Service而不是Component
35 @ChannelHandler.Sharable
36 public class ClusterChatRequestMessageHandler extends
    ChatRequestMessageHandler
37 {
38     @Resource
39     private Session session;
40
41     @Resource
42     private RedisService redisService;
43
44     @Resource
45     private RestTemplate restTemplate;
46
47     /**
48      * 通道读事件触发
49      *
50      * @param ctx          ctx
51      * @param chatRequestMessage 聊天请求消息
52      * @throws Exception 异常
53      */
54     @Override
55     protected void channelRead0(ChannelHandlerContext ctx,
56                                 ChatRequestMessage chatRequestMessage)
57     throws Exception
58     {
59         //检查登录状态
60         if (!session.isLogin(ctx.channel()))
61         {
62             //未登录
63             ctx.writeAndFlush(ChatResponseMessage.fail("请登录")
64                               .setSequenceId(chatRequestMessage.getSequenceId()));
65             return;
66         }
67
68         //谁发的
69         String from = chatRequestMessage.getFrom();
70         //发给谁
71         String to = chatRequestMessage.getTo();
72
73         //判断from是否为空
74         if (from == null || from.equals(""))
75         {
76             ctx.writeAndFlush(ChatResponseMessage.fail("缺失必要参数")
77                               .setSequenceId(chatRequestMessage.getSequenceId()));
78             return;
79         }
80         //判断to是否为空
81         if (to == null || to.equals(""))
82         {
83             ctx.writeAndFlush(ChatResponseMessage.fail("缺失必要参数")
84                               .setSequenceId(chatRequestMessage.getSequenceId()));
85             return;
86         }
87         //校验身份

```

```

87         if (!session.getUsername(ctx.channel()).equals(from))
88         {
89             ctx.writeAndFlush(ChatResponseMessage.fail("身份验证失败! ")
90                 .setSequenceId(chatRequestMessage.getSequenceId()));
91             return;
92         }
93
94         Channel channel = session.getChannel(to);
95         if (to.equals(from))
96         {
97             //自己发送给自己
98             ctx.writeAndFlush(ChatResponseMessage.fail("不能发送给自己")
99                 .setSequenceId(chatRequestMessage.getSequenceId()));
100             return;
101         }
102         if (channel == null)
103         {
104             //为空, 本地不在线或者不存在
105             //查询其他实例
106             String host = redisService.getUserHost(to);
107             if (host == null)
108             {
109                 //其他实例都不在线
110                 ctx.writeAndFlush(ChatResponseMessage.fail("对方用户\" + to
+ "\"不存在或者不在线"));
111
112                 .setSequenceId(chatRequestMessage.getSequenceId()));
113                 return;
114             }
115             //其他实例在线
116             //发起请求
117             //url
118             String url = UrlConstants.buildChatRequestMessageUrl(host);
119             R r = restTemplate.postForObject(url, chatRequestMessage,
R.class);
120             if (r.getIsError())
121             {
122                 //错误
123                 ctx.writeAndFlush(ChatResponseMessage.fail("服务器错误")
124                     .setSequenceId(chatRequestMessage.getSequenceId()));
125             }
126             else
127             {
128                 //写入到自己客户端
129                 ctx.writeAndFlush(ChatResponseMessage
130                     .success(from, null)
131                     .setSequenceId(chatRequestMessage.getSequenceId()));
132                 //聊天统计
133                 redisService.chatCount();
134             }
135         }
136         else

```

```

137         {
138             //在线
139             log.debug(from + "--->" + chatRequestMessage.getTo());
140             //写入到对方客户端
141             channel.writeAndFlush(ChatResponseMessage
142                 .success(from,
143                     chatRequestMessage.getContent())
144                 .setSequenceId(chatRequestMessage.getSequenceId()));
145             //写入到自己客户端
146             ctx.writeAndFlush(ChatResponseMessage
147                 .success(from, null)
148                 .setSequenceId(chatRequestMessage.getSequenceId()));
149             //聊天统计
150             redisService.chatCount();
151         }
152     }
153 }

```

SessionClusterImpl类(implements Session) 的isLogin方法

```

1  @Override
2  public boolean isLogin(String username)
3  {
4      Channel channel = usernameChannelMap.get(username);
5      if (channel != null)
6      {
7          return true;
8      }
9      //在本地未找到
10     boolean hasLogin = redisService.hasLogin(username);
11     if (hasLogin)
12     {
13         log.debug("用户" + username + "在其它服务实例上登录");
14         return true;
15     }
16     log.debug("用户" + username + "未登录");
17     return false;
18 }
19
20 @Override
21 public boolean isLogin(Channel channel)
22 {
23     String username = channelUsernameMap.get(channel);
24     if (username == null)
25     {
26         log.debug("用户" + channel + "未登录");
27         return false;
28     }
29     return true;
30 }

```


NettyController

```
1 package mao.chat_room_netty_server.controller;
2
3 import io.swagger.annotations.Api;
4 import io.swagger.annotations.ApiOperation;
5 import lombok.extern.slf4j.Slf4j;
6 import mao.chat_room_common.message.ChatRequestMessage;
7 import mao.chat_room_common.message.GroupChatResponseMessage;
8 import mao.chat_room_common.message.GroupCreateResponseMessage;
9 import mao.chat_room_netty_server.service.NettyService;
10 import mao.chat_room_netty_server.session.Session;
11 import mao.chat_room_server_api.config.ServerConfig;
12 import mao.tools_core.base.BaseController;
13 import mao.tools_core.base.R;
14 import org.springframework.web.bind.annotation.*;
15
16 import javax.annotation.Resource;
17 import java.util.List;
18 import java.util.Map;
19
20 /**
21  * Project name(项目名称): netty_chat_room
22  * Package(包名): mao.chat_room_netty_server.controller
23  * Class(类名): NettyController
24  * Author(作者): mao
25  * Author QQ: 1296193245
26  * Github: https://github.com/maomao124/
27  * Date(创建日期): 2023/4/1
28  * Time(创建时间): 16:41
29  * Version(版本): 1.0
30  * Description(描述): netty消息接收controller
31  */
32
33 @Slf4j
34 @Api(tags = "netty相关", value = "netty相关")
35 @RestController
36 public class NettyController extends BaseController
37 {
38
39     @Resource
40     private NettyService nettyService;
41
42     @Resource
43     private ServerConfig serverConfig;
44
45     @Resource
46     private Session session;
47
48     /**
49      * 发送聊天消息
50      *
51      * @param chatRequestMessage 聊天请求消息
```

```

52     * @return {@link R}<{@link Boolean}>
53     */
54     @ApiOperation("发送聊天消息")
55     @PostMapping("/send")
56     public R<Boolean> send(@RequestBody ChatRequestMessage
chatRequestMessage)
57     {
58         return nettyService.chatRequestMessageSend(chatRequestMessage);
59     }
60
61     .....
62 }

```

NettyServiceImpl类的chatRequestMessageSend方法

```

1  @Override
2  public R<Boolean> chatRequestMessageSend(ChatRequestMessage
chatRequestMessage)
3  {
4      log.debug("远程发起的聊天发送请求");
5      //发给谁
6      String to = chatRequestMessage.getTo();
7      Channel channel = session.getChannel(to);
8      if (channel == null)
9      {
10         //为空，不在线或者不存在
11         return R.fail("对方用户\" + to + "\"不存在或者不在线");
12     }
13     else
14     {
15         //在线
16         log.debug(chatRequestMessage.getFrom() + "---->" +
chatRequestMessage.getTo());
17         //写入到对方客户端
18         channel.writeAndFlush(ChatResponseMessage
19             .success(chatRequestMessage.getFrom(),
20                 chatRequestMessage.getContent())
21             .setSequenceId(chatRequestMessage.getSequenceId()));
22         //返回成功
23         return R.success();
24     }
25 }

```

群聊消息发送流程

1. 客户端向netty服务发起群聊消息发送请求消息包，包含content、groupName和from
2. netty服务端接收到数据包，处理数据包，进入集群群聊聊天请求入栈消息处理器
3. 服务端检查登录状态
4. 服务端判断from是否为空
5. 服务端校验身份
6. 服务端从redis上得到群聊的成员和成员位置和群聊位置
7. 服务端判断群聊是否存在
8. 如果群聊不存在，响应发送者错误消息"群聊已经不存在"，结束
9. 如果群聊存在，查询群聊的成员和成员位置
10. 判断自己是否在群聊里面
11. 如果自己不在群聊里面，证明未加入群聊，响应给发送者错误消息"请先加入该群聊"
12. 如果自己在群聊里面，需要根据群聊成员的位置分桶，key为host，value一个map，map里面key为用户名，value为GroupChatResponseMessage
13. 遍历群聊成员，如果群聊成员在本实例上，自己发送给此成员，如果不存在，证明在其他实例上，或者不存在，添加到分桶
14. 遍历完成后根据分桶判断是否需要发起http请求
15. 如果不需要发起http请求，群聊聊天发送统计，结束
16. 如果需要发起http请求，遍历分桶，通过http请求发送至需要发送的实例上
17. 由其他实例发送给群聊成员
18. 都调用完成后，群聊聊天发送统计，结束

集群群聊聊天请求入栈消息处理器

```
1 package mao.chat_room_netty_server.handler_cluster;
2
3 import io.netty.channel.Channel;
4 import io.netty.channel.ChannelHandler;
5 import io.netty.channel.ChannelHandlerContext;
6 import io.netty.channel.SimpleChannelInboundHandler;
7 import lombok.extern.slf4j.Slf4j;
8 import mao.chat_room_common.message.ChatResponseMessage;
9 import mao.chat_room_common.message.GroupChatRequestMessage;
10 import mao.chat_room_common.message.GroupChatResponseMessage;
11 import mao.chat_room_netty_server.entity.ClusterGroup;
12 import mao.chat_room_netty_server.handler.GroupChatRequestMessageHandler;
13 import mao.chat_room_netty_server.service.RedisService;
14 import mao.chat_room_netty_server.session.GroupSession;
15 import mao.chat_room_netty_server.session.Session;
16 import mao.chat_room_server_api.constants.UrlConstants;
17 import mao.tools_core.base.R;
18 import org.springframework.stereotype.Service;
19 import org.springframework.web.client.RestTemplate;
20
21 import javax.annotation.Resource;
22 import java.time.LocalDateTime;
```

```

23 import java.util.ArrayList;
24 import java.util.HashMap;
25 import java.util.List;
26 import java.util.Map;
27 import java.util.function.BiConsumer;
28
29 /**
30  * Project name(项目名称): netty_chat_room
31  * Package(包名): mao.chat_room_netty_server.handler_cluster
32  * Class(类名): ClusterGroupChatRequestMessageHandler
33  * Author(作者): mao
34  * Author QQ: 1296193245
35  * GitHub: https://github.com/maomao124/
36  * Date(创建日期): 2023/4/7
37  * Time(创建时间): 18:16
38  * Version(版本): 1.0
39  * Description(描述): 集群群聊聊天请求入栈消息处理器
40  */
41
42 @Slf4j
43 @Service
44 @ChannelHandler.Sharable
45 public class ClusterGroupChatRequestMessageHandler extends
GroupChatRequestMessageHandler
46 {
47
48     @Resource
49     private Session session;
50
51     @Resource
52     private GroupSession groupSession;
53
54     @Resource
55     private RedisService redisService;
56
57     @Resource
58     private RestTemplate restTemplate;
59
60     @Override
61     protected void channelRead0(ChannelHandlerContext ctx,
GroupChatRequestMessage
62 groupChatRequestMessage) throws Exception
63     {
64         //检查登录状态
65         if (!session.isLogin(ctx.channel()))
66         {
67             //未登录
68             ctx.writeAndFlush(GroupChatResponseMessage.fail("请登录"));
69
70             .setSequenceId(groupChatRequestMessage.getSequenceId());
71             return;
72         }
73
74         String groupName = groupChatRequestMessage.getGroupName();
75         String content = groupChatRequestMessage.getContent();

```

```

75     String from = groupChatRequestMessage.getFrom();
76
77     //判断from是否为空
78     if (from == null || from.equals(""))
79     {
80         ctx.writeAndFlush(ChatResponseMessage.fail("缺失必要参数"));
81
82         .setSequenceId(groupChatRequestMessage.getSequenceId());
83         return;
84     }
85
86     //校验身份
87     if (!session.getUsername(ctx.channel()).equals(from))
88     {
89         ctx.writeAndFlush(ChatResponseMessage.fail("身份验证失败!"));
90
91         .setSequenceId(groupChatRequestMessage.getSequenceId());
92         return;
93     }
94
95     //得到群聊的成员和成员位置和群聊位置
96     ClusterGroup clusterGroup =
97     groupSession.getMembersAndHost(groupName);
98     //判断群聊是否存在
99     if (clusterGroup == null)
100     {
101         //不存在
102         ctx.writeAndFlush(GroupChatResponseMessage.fail("群聊已经不存在"));
103
104         .setSequenceId(groupChatRequestMessage.getSequenceId());
105         return;
106     }
107     //群聊存在
108     //得到群聊的成员和成员位置
109     Map<String, String> groupMembersAndHost =
110     clusterGroup.getGroupMembersAndHost();
111     //判断自己是否在里面
112     if (groupMembersAndHost.get(from) == null)
113     {
114         //不在
115         ctx.writeAndFlush(GroupChatResponseMessage.fail("请先加入该群聊"));
116
117         .setSequenceId(groupChatRequestMessage.getSequenceId());
118         return;
119     }
120     //分桶,key为host, value一个map, map里面key为用户名, value为为
121     GroupChatResponseMessage
122     Map<String, Map<String, GroupChatResponseMessage>> map = new
123     HashMap<>();
124     //发给每一位成员的时间要一致
125     LocalDateTime now = LocalDateTime.now();
126     //这里并发很大, 对于服务器而言, 使用异步操作反而会因为线程的上下文切换而影响性能
127     groupMembersAndHost.forEach(new BiConsumer<String, String>()
128     {
129         /**

```

```

122         * 遍历
123         *
124         * @param username 用户名
125         * @param host 用户的位置
126         */
127         @Override
128         public void accept(String username, String host)
129         {
130             //在本实例内取，如果没有取到，证明在其他实例上，或者不存在
131             Channel channel = session.getChannel(username);
132             if (channel != null)
133             {
134                 //在本实例上
135
136                 channel.writeAndFlush(GroupChatResponseMessage.success(from, content,
137                                     groupName)
138
139                 .setSequenceId(groupChatRequestMessage.getSequenceId())
140                                     .setTime(now));
141                 log.debug("用户" + username + "在本实例内，直接发送");
142             }
143             else
144             {
145                 //不在本实例上，往桶里添加
146                 //如果没有，就创建一个空的
147                 Map<String, GroupChatResponseMessage> userMap =
148                     map.computeIfAbsent(host, k -> new HashMap<>
149                     ());
150                 //构建
151                 GroupChatResponseMessage groupChatResponseMessage =
152                     (GroupChatResponseMessage)
153                     GroupChatResponseMessage
154                         .success(from, content, groupName)
155
156                 .setSequenceId(groupChatRequestMessage.getSequenceId())
157                                     .setTime(now);
158                 userMap.put(username, groupChatResponseMessage);
159                 log.debug("用户" + username + "添加到分桶");
160             }
161         }
162     }
163     }
164     log.debug("分桶结果: " + map);
165     //判断是否需要发起http请求
166     if (map.size() != 0)
167     {
168         log.debug("准备发起请求");
169         //这里并发很大，对于服务器而言，使用异步操作反而会因为线程的上下文切换而影
170         响性能
171         map.forEach(new BiConsumer<String, Map<String,
172         GroupChatResponseMessage>>()
173         {
174
175             /**
176              * 遍历分桶
177              *

```

```

169         * @param host                位置
170         * @param groupChatResponseMessageMap 群组聊天响应消息映射
171         */
172         @Override
173         public void accept(String host, Map<String,
GroupChatResponseMessage> groupChatResponseMessageMap)
174         {
175             //远程调用
176             String url =
urlConstants.buildGroupChatRequestMessageUrl(host);
177             log.debug("url:" + url);
178             log.debug("正在发起请求: " + host);
179             R r = restTemplate.postForObject(url,
groupChatResponseMessageMap, R.class);
180             if (r.getIsError())
181             {
182                 //错误
183                 log.warn("发送群聊消息时出现错误:" + r.getMsg());
184             }
185             else
186             {
187                 //正确
188                 log.debug(host + " : 请求完成");
189             }
190         }
191     });
192 }
193 //群聊聊天统计
194 redisService.groupChatCount();
195 }
196 }

```

GroupSessionClusterImpl的getMembersAndHost方法

```

1  @Override
2  public ClusterGroup getMembersAndHost(String name)
3  {
4      Map<Object, Object> membersAndHost =
redisService.getMembersAndHost(name);
5      if (membersAndHost == null || membersAndHost.size() == 0)
6      {
7          return null;
8      }
9      String host = membersAndHost.get("host").toString();
10     ClusterGroup clusterGroup = new ClusterGroup();
11     clusterGroup.setGroupHost(host);
12     membersAndHost.remove("host");
13     Map<String, String> groupMembersAndHost = new HashMap<>
(membersAndHost.size());
14     membersAndHost.forEach((key, value) ->
15     {

```

```

16         String keyString = key.toString();
17         String valueString = membersAndHost.get(key).toString();
18         groupMembersAndHost.put(keyString, valueString);
19     });
20     return clusterGroup.setGroupMembersAndHost(groupMembersAndHost);
21 }

```

RedisServiceImpl的getMembersAndHost方法

```

1  @Override
2  public Map<Object, Object> getMembersAndHost(String name)
3  {
4      String key = RedisConstants.chat_group_key + name;
5      Map<Object, Object> entries =
6      stringRedisTemplate.opsForHash().entries(key);
7      log.debug("获取群聊: " + name + "的所有群成员: " + entries);
8      return entries;
9  }

```

```

1  package mao.chat_room_netty_server.controller;
2
3  import io.swagger.annotations.Api;
4  import io.swagger.annotations.ApiOperation;
5  import lombok.extern.slf4j.Slf4j;
6  import mao.chat_room_common.message.ChatRequestMessage;
7  import mao.chat_room_common.message.GroupChatResponseMessage;
8  import mao.chat_room_common.message.GroupCreateResponseMessage;
9  import mao.chat_room_netty_server.service.NettyService;
10 import mao.chat_room_netty_server.session.Session;
11 import mao.chat_room_server_api.config.ServerConfig;
12 import mao.tools_core.base.BaseController;
13 import mao.tools_core.base.R;
14 import org.springframework.web.bind.annotation.*;
15
16 import javax.annotation.Resource;
17 import java.util.List;
18 import java.util.Map;
19
20 /**
21  * Project name(项目名称): netty_chat_room
22  * Package(包名): mao.chat_room_netty_server.controller
23  * Class(类名): NettyController
24  * Author(作者): mao
25  * Author QQ: 1296193245
26  * Github: https://github.com/maomao124/
27  * Date(创建日期): 2023/4/1
28  * Time(创建时间): 16:41
29  * Version(版本): 1.0

```



```

30  * Description(描述): netty消息接收controller
31  */
32
33  @Slf4j
34  @Api(tags = "netty相关", value = "netty相关")
35  @RestController
36  public class NettyController extends BaseController
37  {
38
39      @Resource
40      private NettyService nettyService;
41
42      @Resource
43      private ServerConfig serverConfig;
44
45      @Resource
46      private Session session;
47
48      .....
49
50      /**
51       * 发送群聊聊天消息
52       *
53       * @param map {@link Map}<{@link String}, {@link
GroupChatResponseMessage}>
54       *           key为用户名, value为GroupChatResponseMessage
55       * @return {@link R}<{@link Boolean}>
56       */
57      @ApiOperation("发送群聊聊天消息")
58      @PostMapping("/sendGroupChatMessage")
59      public R<Boolean> sendGroupChatMessage(@RequestBody Map<String,
GroupChatResponseMessage> map)
60      {
61          return nettyService.sendGroupChatMessage(map);
62      }
63
64      .....
65  }

```

NettyServiceImpl的sendGroupChatMessage方法

```

1  @Override
2  public R<Boolean> sendGroupChatMessage(Map<String, GroupChatResponseMessage>
map)
3  {
4      log.debug("发送群聊消息");
5      map.forEach(new BiConsumer<String, GroupChatResponseMessage>()
6      {
7          /**

```

```

8      * 遍历
9      *
10     * @param username          用户名
11     * @param groupChatResponseMessage 群组聊天响应消息
12     */
13     @Override
14     public void accept(String username, GroupChatResponseMessage
groupChatResponseMessage)
15     {
16         //根据用户名获取channel
17         Channel channel = session.getChannel(username);
18         //判断是否为空
19         if (channel != null)
20         {
21             channel.writeAndFlush(groupChatResponseMessage);
22         }
23         else
24         {
25             //不存在
26             log.info("发送群聊消息时，用户名: " + username + "无法发送");
27         }
28     }
29 });
30 return R.success();
31 }

```

群聊创建流程

1. 客户端向netty服务发起群聊创建请求消息包，包含groupName和members（群聊成员）
2. netty服务端接收到数据包，处理数据包，进入集群群聊创建请求入栈消息处理器
3. 服务端检查登录状态
4. 服务端判断群聊名称是否存在，如果已经存在，响应发送失败的消息，结束
5. 如果群聊不存在，分桶，key为host，value为GroupCreateResponseMessage列表，遍历成员列表
6. 判断某一个群聊成员是否在线（全局在线），如果不在线，什么都不用做，继续遍历
7. 如果群聊成员全局在线，证明群聊成员可能在本实例上在线，也有可能其它实例上在线但是不在此实例上在线，添加至在线列表中，继续执行下一步
8. 判断群聊成员是否在本实例上在线
9. 如果群聊成员在本实例上在线，通知群聊成员您已被拉入群聊的消息
10. 如果群聊成员不在本实例上在线，证明群聊成员在其它实例上，加入到分桶中
11. 遍历完成后，根据分桶判断是否需要发起远程调用，如果不需要发起远程调用，证明所有的在线群聊成员都在此实例上
12. 响应群聊创建者在线成员列表，群聊创建统计，结束

群聊创建请求入栈消息处理器

```
1 package mao.chat_room_netty_server.handler_cluster;
2
3 import io.netty.channel.Channel;
4 import io.netty.channel.ChannelHandler;
5 import io.netty.channel.ChannelHandlerContext;
6 import lombok.extern.slf4j.Slf4j;
7 import mao.chat_room_common.message.GroupCreateRequestMessage;
8 import mao.chat_room_common.message.GroupCreateResponseMessage;
9 import mao.chat_room_netty_server.handler.GroupCreateRequestMessageHandler;
10 import mao.chat_room_netty_server.service.RedisService;
11 import mao.chat_room_netty_server.session.Group;
12 import mao.chat_room_netty_server.session.GroupSession;
13 import mao.chat_room_netty_server.session.Session;
14 import org.springframework.stereotype.Service;
15
16 import javax.annotation.Resource;
17 import java.util.*;
18
19 /**
20  * Project name(项目名称): netty_chat_room
21  * Package(包名): mao.chat_room_netty_server.handler_cluster
22  * Class(类名): ClusterGroupCreateRequestMessageHandler
23  * Author(作者): mao
24  * Author QQ: 1296193245
25  * GitHub: https://github.com/maomao124/
26  * Date(创建日期): 2023/4/3
27  * Time(创建时间): 21:54
28  * Version(版本): 1.0
29  * Description(描述): 群聊创建请求入栈消息处理器
30  */
31
32 @Slf4j
33 @Service
34 @ChannelHandler.Sharable
35 public class ClusterGroupCreateRequestMessageHandler extends
36     GroupCreateRequestMessageHandler
37 {
38     @Resource
39     private GroupSession groupSession;
40
41     @Resource
42     private Session session;
43
44     @Resource
45     private RedisService redisService;
46
47     @Override
48     protected void channelRead0(ChannelHandlerContext ctx,
49         GroupCreateRequestMessage
50         groupCreateRequestMessage) throws Exception
51     {
```

```

50         //检查登录状态
51         if (!session.isLogin(ctx.channel()))
52         {
53             //未登录
54             ctx.writeAndFlush(GroupCreateResponseMessage.fail("请登录"));
55
56             .setSequenceId(groupCreateRequestMessage.getSequenceId()));
57             return;
58         }
59         //组名
60         String groupName = groupCreateRequestMessage.getGroupName();
61         //群成员
62         Set<String> members = groupCreateRequestMessage.getMembers();
63         boolean hasGroup = groupSession.hasGroup(groupName);
64         //判断群聊名称是否存在
65         if (hasGroup)
66         {
67             //已存在
68             ctx.writeAndFlush(GroupCreateResponseMessage.fail("群聊名称\""+
69 groupName + "\"已经存在！换个名字吧"));
70             .setSequenceId(groupCreateRequestMessage.getSequenceId()));
71             }
72             else
73             {
74                 //不存在
75                 //创建群聊
76                 Group group = groupSession.createGroup(groupName, members);
77                 //在线的成员列表
78                 Set<String> members1 = group.getMembers();
79                 ctx.writeAndFlush(GroupCreateResponseMessage.success(members1));
80                 .setSequenceId(groupCreateRequestMessage.getSequenceId()));
81                 //群聊创建统计
82                 redisService.groupCreateCount();
83             }
84         }

```

GroupSessionClusterImpl的hasGroup方法

```

1  @Override
2  public boolean hasGroup(String name)
3  {
4      Group group = groupMap.get(name);
5      if (group != null)
6      {
7          //本地存在
8          return true;
9      }
10     //本地不存在
11     //查询redis
12     return redisService.hasGroup(name);
13 }

```

GroupSessionClusterImpl的createGroup方法

```

1  @Override
2  public Group createGroup(String name, Set<String> members)
3  {
4      log.debug("创建群聊: " + name + ", 成员: " + members);
5      Set<String> members1 = redisService.createGroup(name, members, host);
6      Group group = new Group(name, members1);
7      groupMap.putIfAbsent(name, group);
8      return group;
9  }

```

RedisServiceImpl的createGroup方法

```

1  @SneakyThrows
2  @Override
3  public Set<String> createGroup(String name, Set<String> members, String
    host)
4  {
5      String key = RedisConstants.chat_group_key + name;
6      String key2 = RedisConstants.chat_group_list_key + host;
7      stringRedisTemplate.opsForHash().put(key, "host", host);
8      log.debug("创建组: " + members);
9      //在线成员列表
10     Set<String> members1 = new ConcurrentHashSet<>(members.size());
11     //分桶,key为host, value为GroupCreateResponseMessage列表
12     Map<String, List<GroupCreateResponseMessage>> map = new HashMap<>();
13     CountDownLatch countDownLatch = new CountDownLatch(members.size());
14     for (String username : members)
15     {
16         threadPoolExecutor.submit(new Runnable()
17         {

```

```

18         @Override
19         public void run()
20         {
21             try
22             {
23                 String usernameKey = RedisConstants.chat_user_key +
username;
24                 String host =
stringRedisTemplate.opsForValue().get(usernameKey);
25                 //判断用户是否在线
26                 if (host == null || host.equals(""))
27                 {
28                     //不在线
29                     log.debug("用户" + username + "不在线");
30                 }
31                 else
32                 {
33                     //在线
34                     stringRedisTemplate.opsForHash().put(key, username,
host);
35                     log.debug("用户" + username + "在线, 位于: " + host);
36                     members1.add(username);
37                     //准备通知在线的成员
38                     Channel channel = session.getChannel(username);
39                     //判断该用户是否在本机
40                     if (channel != null)
41                     {
42                         //在本机, 直接通知
43                         //通知
44                         channel.writeAndFlush(new
GroupCreateResponseMessage()
45                             .setSuccess(true)
46                             .setReason("您已被拉入群聊\" + name +
"\!")")
47                             .setSequenceId());
48                     }
49                     else
50                     {
51                         //不在本地, 在其他实例上
52                         map.computeIfAbsent(host, k -> new ArrayList<>
())
53                         .add(new GroupCreateResponseMessage()
54                             .setSuccess(true)
55                             .setReason("您已被拉入群聊\" + name +
"\!")")
56                         .setSequenceId());
57                         //对host加本地进程锁, 相当于锁的map的桶下标的表头
58                         synchronized (host.intern())
59                         {
60                             List<GroupCreateResponseMessage> list =
map.get(host);
61                             //发送推送消息时群成员只有一个, 就是自己
62                             Set<String> usernameSet = new HashSet<>();
63                             usernameSet.add(username);
64                             Message message = new
GroupCreateResponseMessage()
65                                 .setMembers(usernameSet)
66                                 .setSuccess(true)
67                                 .setReason("您已被拉入群聊\" + name +
"\!")")
68                                 .setSequenceId());
69                             list.add(message);
70                             map.put(host, list);
71                         }
72                     }
73                 }
74             }
75             catch (Exception e)
76             {
77                 log.error(e.getMessage());
78             }
79         }
80     }
81 }

```

```

64         .setSequenceId();
65         list.add((GroupCreateResponseMessage)
message);
66     }
67 }
68 }
69 }
70 catch (Exception e)
71 {
72     log.error("错误: ", e);
73 }
74 finally
75 {
76     countdownLatch.countDown();
77 }
78 }
79 });
80 }
81 stringRedisTemplate.opsForSet().add(key2, name);
82 //等待
83 countdownLatch.await();
84 log.debug("在线成员: " + members1);
85 log.debug("分桶结果: " + map);
86 //远程调用其他实例, 通知在线的成员
87 //判断是否需要发起远程调用
88 if (map.size() > 0)
89 {
90     //大于0, 需要发起远程调用
91     log.debug("将发起远程调用");
92     CountdownLatch finalCountDownLatch = new
CountDownLatch(map.size());
93     map.forEach(new BiConsumer<String,
List<GroupCreateResponseMessage>>()
94     {
95         /**
96          * forEach
97          *
98          * @param host 主机地址
99          * @param groupCreateResponseMessages
GroupCreateResponseMessage列表
100          */
101         @Override
102         public void accept(String host,
List<GroupCreateResponseMessage> groupCreateResponseMessages)
103         {
104             threadPoolExecutor.submit(() ->
105             {
106
107                 try
108                 {
109                     log.debug("正在同步的方式推送给" + host);
110                     String url =
UrlConstants.buildGroupCreateRequestMessageUrl(host);
111                     R<? extends Object> r =
restTemplate.postForObject(url, groupCreateResponseMessages, R.class);

```

```

112         if (r.getIsError())
113         {
114             //失败
115             log.warn("推送给" + host + "时出现错误:" +
r.getMsg());
116         }
117         else
118         {
119             //成功
120             log.debug("推送给" + host + "成功");
121         }
122     }
123     catch (Exception e)
124     {
125         log.error("推送给" + host + "时出现错误:", e);
126     }
127     finally
128     {
129         finalCountDownLatch.countDown();
130     }
131     });
132     }
133     });
134
135     //等待
136     finalCountDownLatch.await();
137     log.debug("推送完成");
138 }
139 //返回在线列表
140 return members1;
141 }

```

NettyController

```

1  package mao.chat_room_netty_server.controller;
2
3  import io.swagger.annotations.Api;
4  import io.swagger.annotations.ApiOperation;
5  import lombok.extern.slf4j.Slf4j;
6  import mao.chat_room_common.message.ChatRequestMessage;
7  import mao.chat_room_common.message.GroupChatResponseMessage;
8  import mao.chat_room_common.message.GroupCreateResponseMessage;
9  import mao.chat_room_netty_server.service.NettyService;
10 import mao.chat_room_netty_server.session.Session;
11 import mao.chat_room_server_api.config.ServerConfig;
12 import mao.tools_core.base.BaseController;
13 import mao.tools_core.base.R;
14 import org.springframework.web.bind.annotation.*;
15
16 import javax.annotation.Resource;
17 import java.util.List;

```



```

18 import java.util.Map;
19
20 /**
21  * Project name(项目名称): netty_chat_room
22  * Package(包名): mao.chat_room_netty_server.controller
23  * Class(类名): NettyController
24  * Author(作者): mao
25  * Author QQ: 1296193245
26  * Github: https://github.com/maomao124/
27  * Date(创建日期): 2023/4/1
28  * Time(创建时间): 16:41
29  * Version(版本): 1.0
30  * Description(描述): netty消息接收controller
31  */
32
33 @Slf4j
34 @Api(tags = "netty相关", value = "netty相关")
35 @RestController
36 public class NettyController extends BaseController
37 {
38
39     @Resource
40     private NettyService nettyService;
41
42     @Resource
43     private ServerConfig serverConfig;
44
45     @Resource
46     private Session session;
47
48     /**
49      * 发送聊天消息
50      *
51      * @param chatRequestMessage 聊天请求消息
52      * @return {@link R}<{@link Boolean}>
53      */
54     @ApiOperation("发送聊天消息")
55     @PostMapping("/send")
56     public R<Boolean> send(@RequestBody ChatRequestMessage
57 chatRequestMessage)
58     {
59         return nettyService.chatRequestMessageSend(chatRequestMessage);
60     }
61
62     /**
63      * 得到当前实例的netty的端口号
64      *
65      * @return {@link R}<{@link Integer}>
66      */
67     @ApiOperation("得到当前实例的netty的端口号")
68     @GetMapping("/port")
69     public R<Integer> getPort()
70     {
71         return success(serverConfig.getServerPort());
72     }
73 }

```

```

72
73
74     /**
75      * 发送群聊创建消息
76      *
77      * @param groupCreateResponseMessages 群聊创建响应消息集合
78      * @return {@link R}<{@link Boolean}>
79      */
80     @ApiOperation("发送群聊创建消息")
81     @PostMapping("/sendGroupCreateMessage")
82     public R<Boolean> sendGroupCreateMessage(@RequestBody
83     List<GroupCreateResponseMessage> groupCreateResponseMessages)
84     {
85         return
86         nettyService.sendGroupCreateMessage(groupCreateResponseMessages);
87     }
88
89     /**
90      * 发送群聊聊天消息
91      *
92      * @param map {@link Map}<{@link String}, {@link
93      GroupChatResponseMessage}>
94      *          key为用户名, value为GroupChatResponseMessage
95      * @return {@link R}<{@link Boolean}>
96      */
97     @ApiOperation("发送群聊聊天消息")
98     @PostMapping("/sendGroupChatMessage")
99     public R<Boolean> sendGroupChatMessage(@RequestBody Map<String,
100     GroupChatResponseMessage> map)
101     {
102         return nettyService.sendGroupChatMessage(map);
103     }
104
105     /**
106      * 成员加入本地群聊
107      *
108      * @param name 群聊名字
109      * @param member 群聊成员
110      * @return {@link R}<{@link Boolean}>
111      */
112     @ApiOperation("成员加入本地群聊")
113     @PostMapping("/joinMember")
114     public R<Boolean> joinMember(@RequestParam String name, @RequestParam
115     String member)
116     {
117         return nettyService.joinMember(name, member);
118     }
119
120     /**
121      * 成员退出本地群聊
122      *
123      * @param name 群聊名字
124      * @param member 群聊成员
125      * @return {@link R}<{@link Boolean}>
126      */

```

```

122     @ApiOperation("成员退出本地群聊")
123     @PostMapping("/removeMember")
124     public R<Boolean> removeMember(@RequestParam String name, @RequestParam
String member)
125     {
126         return nettyService.removeMember(name, member);
127     }
128
129     /**
130      * 得到当前实例在线用户数量，不包括未登录但是已经连接上的
131      *
132      * @return {@link R}<{@link Integer}> 此实例在线人数的数量
133      */
134     @ApiOperation("得到当前实例在线用户数量")
135     @GetMapping("/getOnlineUserCount")
136     public R<Integer> getOnlineUserCount()
137     {
138         int size = session.getSize();
139         log.debug("得到当前实例在线用户数量:" + size);
140         return success(size);
141     }
142 }

```

NettyServiceImpl的sendGroupCreateMessage方法

```

1  @Override
2  public R<Boolean> sendGroupCreateMessage(List<GroupCreateResponseMessage>
groupCreateResponseMessages)
3  {
4      for (GroupCreateResponseMessage groupCreateResponseMessage :
groupCreateResponseMessages)
5      {
6          log.debug("发送群聊创建消息");
7          //得到用户名
8          String username =
groupCreateResponseMessage.getMembers().iterator().next();
9          Channel channel = session.getChannel(username);
10         if (channel != null)
11         {
12             channel.writeAndFlush(groupCreateResponseMessage);
13         }
14         else
15         {
16             log.info("发送群聊创建消息时，用户名: " + username + "无法发送");
17         }
18     }
19     return R.success();
20 }

```

ReBalance机制

问题说明

客户端与netty服务是tcp长连接的，假设netty服务有3台实例a、b和c，实例各有客户端连接999、1000和1001，客户端请求连接的时候，是负载均衡的，但是当某一台实例重启后，比如重启c实例，因为客户端的重连机制，c实例的这1001个连接会跑到a和b这两台实例上，现在等c实例重启完成，假设这段时间没有新客户端连接，现在的各个实例的连接数为1500左右、1500左右和0，和预想的1000、1000和1000不符。

解决方案

当netty服务启动时，发送一条ReBalance的延迟MQ消息，消息消费者为管理服务，管理服务接收到消息之后，向netty服务的所有实例发起http请求，得到实例的在线人数根据在线人数通过http请求的方式向netty服务发起请求让连接再次分配，netty服务接收到请求后，随机抽取对应数量的channel，向客户端发送ReBalance数据包，让客户端连接对应的实例上。

关键代码

消息生产者接口

```
1 package mao.chat_room_netty_server.producer;
2
3 /**
4  * Project name(项目名称): netty_chat_room
5  * Package(包名): mao.chat_room_netty_server.producer
6  * Interface(接口名): ServerProducer
7  * Author(作者): mao
8  * Author QQ: 1296193245
9  * GitHub: https://github.com/maomao124/
10 * Date(创建日期): 2023/4/2
11 * Time(创建时间): 13:42
12 * Version(版本): 1.0
13 * Description(描述): 服务相关的消息生产者
14 */
15
16 public interface ServerProducer
17 {
```

```

18     /**
19      * 发送netty服务变动更新消息
20      */
21     void sendNettyServerUpdateMessage();
22
23     /**
24      * 发送重新平衡信息
25      * 当服务重启时，可能会造成负载不均衡的现象，大部分netty channel都跑到了老实例上
26      * 此方法的作用是发送一条消息，重新分配netty的channel，让某些用户断开连接新的实例
27      */
28     void sendReBalanceMessage();
29 }

```

实现类

```

1 package mao.chat_room_netty_server.producer.impl;
2
3 import lombok.extern.slf4j.Slf4j;
4 import mao.chat_room_netty_server.producer.ServerProducer;
5 import mao.chat_room_netty_server.service.RedisService;
6 import mao.chat_room_server_api.constants.RocketMQConstants;
7 import org.apache.rocketmq.common.message.Message;
8 import org.apache.rocketmq.spring.core.RocketMQTemplate;
9 import org.springframework.beans.factory.annotation.Autowired;
10 import org.springframework.beans.factory.annotation.Value;
11 import org.springframework.messaging.MessageHeaders;
12 import org.springframework.stereotype.Component;
13 import org.springframework.stereotype.Service;
14
15 import javax.annotation.Resource;
16 import java.net.InetAddress;
17 import java.net.UnknownHostException;
18 import java.nio.charset.StandardCharsets;
19
20 import org.springframework.messaging.support.MessageBuilder;
21
22 /**
23  * Project name(项目名称): netty_chat_room
24  * Package(包名): mao.chat_room_netty_server.producer.impl
25  * Class(类名): RocketMQServerProducerImpl
26  * Author(作者): mao
27  * Author QQ: 1296193245
28  * GitHub: https://github.com/maomao124/
29  * Date(创建日期): 2023/4/2
30  * Time(创建时间): 13:43
31  * Version(版本): 1.0
32  * Description(描述): rocketMQ 消息生产者
33  */
34
35 @Slf4j
36 @Component

```

```

37 public class RocketMQServerProducerImpl implements ServerProducer
38 {
39
40     @Resource
41     private RocketMQTemplate rocketMQTemplate;
42
43     private final String host;
44
45     @Autowired
46     public RocketMQServerProducerImpl(@Value("${server.port}") String port)
47         throws UnknownHostException
48     {
49         /*
50          * 主机地址
51          */
52         String hostAddress = InetAddress.getLocalHost().getHostAddress();
53         this.host = hostAddress + ":" + port;
54     }
55
56     @Override
57     public void sendNettyServerUpdateMessage()
58     {
59         log.info("发送netty服务变动更新消息");
60
61         rocketMQTemplate.convertAndSend(RocketMQConstants.NETTY_SERVER_UPDATE_MESSA
62             GE_TOPIC, host);
63     }
64
65     @Override
66     public void sendReBalanceMessage()
67     {
68         log.info("发送netty ReBalance消息");
69
70         rocketMQTemplate.syncSend(RocketMQConstants.NETTY_SERVER_RE_BALANCE_TOPIC,
71             MessageBuilder.withPayload(host).build(),
72             5000,
73             4);
74     }
75 }

```

netty服务端

```

1 package mao.chat_room_netty_server;
2
3 import io.netty.bootstrap.ServerBootstrap;
4 import io.netty.channel.Channel;
5 import io.netty.channel.ChannelInitializer;
6 import io.netty.channel.nio.NioEventLoopGroup;
7 import io.netty.channel.socket.nio.NioServerSocketChannel;
8 import io.netty.channel.socket.nio.NioSocketChannel;

```

```

9  import io.netty.handler.logging.LogLevel;
10 import io.netty.handler.logging.LoggingHandler;
11 import io.netty.handler.timeout.IdleStateHandler;
12 import io.netty.util.concurrent.Future;
13 import io.netty.util.concurrent.GenericFutureListener;
14 import lombok.SneakyThrows;
15 import lombok.extern.slf4j.Slf4j;
16 import mao.chat_room_common.protocol.Procoto1FrameDecoder;
17 import mao.chat_room_netty_server.handler.*;
18 import mao.chat_room_netty_server.producer.ServerProducer;
19 import mao.chat_room_netty_server.service.RedisService;
20 import mao.chat_room_server_api.config.ServerConfig;
21 import mao.chat_room_server_api.protocol.ServerMessageCodecSharable;
22 import org.springframework.beans.factory.annotation.Value;
23 import org.springframework.boot.CommandLineRunner;
24 import org.springframework.stereotype.Component;
25
26 import javax.annotation.PostConstruct;
27 import javax.annotation.Resource;
28 import java.net.InetAddress;
29 import java.util.concurrent.locks.LockSupport;
30
31
32 /**
33  * Project name(项目名称): netty_chat_room
34  * Package(包名): mao.chat_room_netty_server
35  * Class(类名): NettyServer
36  * Author(作者): mao
37  * Author QQ: 1296193245
38  * Github: https://github.com/maomao124/
39  * Date(创建日期): 2023/3/28
40  * Time(创建时间): 21:18
41  * Version(版本): 1.0
42  * Description(描述): netty服务器初始化
43  */
44
45 @Slf4j
46 @Component
47 public class NettyServer implements CommandLineRunner
48 {
49
50     @Resource
51     private ServerMessageCodecSharable serverMessageCodecSharable;
52
53     @Resource
54     private ServerConfig serverConfig;
55
56     /**
57      * 协议帧解码器，这里不能共用
58      */
59     @Resource
60     private Procoto1FrameDecoder procoto1FrameDecoder;
61
62     @Resource
63     private ChatRequestMessageHandler chatRequestMessageHandler;

```

```

64
65     @Resource
66     private GroupChatRequestMessageHandler groupChatRequestMessageHandler;
67
68     @Resource
69     private GroupCreateRequestMessageHandler
groupCreateRequestMessageHandler;
70
71     @Resource
72     private GroupJoinRequestMessageHandler groupJoinRequestMessageHandler;
73
74     @Resource
75     private GroupMembersRequestMessageHandler
groupMembersRequestMessageHandler;
76
77     @Resource
78     private GroupQuitRequestMessageHandler groupQuitRequestMessageHandler;
79
80     @Resource
81     private LoginRequestMessageHandler loginRequestMessageHandler;
82
83     @Resource
84     private RegisterRequestMessageHandler registerRequestMessageHandler;
85
86     @Resource
87     private QuitHandler quitHandler;
88
89     @Resource
90     private PingMessageHandler pingMessageHandler;
91
92     @Resource
93     private ServerProducer serverProducer;
94
95     @Resource
96     private RedisService redisService;
97
98     @Value("${server.port}")
99     private String port;
100
101     /**
102      * 运行,禁止长时间阻塞此线程
103      *
104      * @param args 参数
105      * @throws Exception 异常
106      */
107     @Override
108     public void run(String... args) throws Exception
109     {
110         NioEventLoopGroup boss = new NioEventLoopGroup();
111         NioEventLoopGroup worker = new NioEventLoopGroup();
112
113         LoggingHandler LOGGING_HANDLER = new
LoggingHandler(LogLevel.DEBUG);
114
115         try

```



```

116         {
117             ServerBootstrap serverBootstrap = new ServerBootstrap();
118             Channel channel = serverBootstrap.group(boss, worker)
119                 .channel(NioServerSocketChannel.class)
120                 .childHandler(new ChannelInitializer<NioSocketChannel>
121 ()
122         {
123             @Override
124             protected void initChannel(NioSocketChannel ch)
125             throws Exception
126             {
127                 ch.pipeline().addLast(LOGGING_HANDLER)
128                     .addLast(new ProtocolFrameDecoder())
129                     .addLast(new IdleStateHandler(70, 0,
130 0))
131                     .addLast(new ServerDuplexHandler())
132                     .addLast(serverMessageCodecSharable)
133                     .addLast(chatRequestMessageHandler)
134
135 .addLast(groupChatRequestMessageHandler)
136
137 .addLast(groupCreateRequestMessageHandler)
138
139 .addLast(groupJoinRequestMessageHandler)
140
141 .addLast(groupMembersRequestMessageHandler)
142
143 .addLast(groupQuitRequestMessageHandler)
144                     .addLast(loginRequestMessageHandler)
145                     .addLast(registerRequestMessageHandler)
146                     .addLast(pingMessageHandler)
147                     .addLast(quitHandler);
148
149             }
150         }).bind(serverConfig.getServerPort()).sync().channel();
151         log.info("Netty服务器启动成功");
152         serverProducer.sendNettyServerUpdateMessage();
153         serverProducer.sendReBalanceMessage();
154         channel.closeFuture().addListener(new
155 GenericFutureListener<Future<? super Void>>()
156     {
157         /**
158          * 操作完成（这里是关闭）
159          *
160          * @param future netty Future对象
161          * @throws Exception 异常
162          */
163         @Override
164         public void operationComplete(Future<? super Void> future)
165         throws Exception
166         {
167             log.info("正在关闭服务器...");
168             close(boss, worker);
169         }
170     });

```

```

161         Runtime.getRuntime().addShutdownHook(new Thread(new Runnable()
162         {
163             @SneakyThrows
164             @Override
165             public void run()
166             {
167                 log.info("正在关闭服务器...");
168                 serverProducer.sendNettyServerUpdateMessage();
169                 String hostAddress =
170                 InetAddress.getLocalHost().getHostAddress();
171                 String host = hostAddress + ":" + port;
172                 redisService.unbindGroup(host);
173                 close(boss, worker);
174             }
175         }));
176     }
177     catch (Exception e)
178     {
179         log.info("Netty服务器启动失败");
180         throw new RuntimeException(e);
181     }
182 }
183 /**
184  * 关闭
185  *
186  * @param boss    NioEventLoopGroup
187  * @param worker  NioEventLoopGroup
188  */
189 private void close(NioEventLoopGroup boss, NioEventLoopGroup worker)
190 {
191     try
192     {
193         boss.shutdownGracefully();
194     }
195     catch (Exception ignored)
196     {
197     }
198 }
199 try
200 {
201     worker.shutdownGracefully();
202 }
203 catch (Exception ignored)
204 {
205 }
206 }
207 }
208
209 @PostConstruct
210 public void init()
211 {
212     log.info("初始化 NettyServer");
213 }
214

```

消息消费者

位于chat-room-manage服务

```

1  package mao.chat_room_manage.consumer;
2
3  import lombok.Getter;
4  import lombok.extern.slf4j.Slf4j;
5  import mao.chat_room_manage.service.ReBalanceService;
6  import mao.chat_room_server_api.constants.RocketMQConstants;
7  import org.apache.rocketmq.spring.annotation.MessageModel;
8  import org.apache.rocketmq.spring.annotation.RocketMQMessageListener;
9  import org.apache.rocketmq.spring.core.RocketMQListener;
10 import org.springframework.stereotype.Component;
11
12 import javax.annotation.Resource;
13
14 /**
15  * Project name(项目名称): netty_chat_room
16  * Package(包名): mao.chat_room_manage.consumer
17  * Class(类名): ReBalanceConsumer
18  * Author(作者): mao
19  * Author QQ: 1296193245
20  * GitHub: https://github.com/maomao124/
21  * Date(创建日期): 2023/4/13
22  * Time(创建时间): 15:51
23  * Version(版本): 1.0
24  * Description(描述): 消费者
25  */
26
27 @Slf4j
28 @Getter
29 @Component
30 @RocketMQMessageListener(consumerGroup = RocketMQConstants.GROUP,
31                          topic = RocketMQConstants.NETTY_SERVER_RE_BALANCE_TOPIC,
32                          messageModel = MessageModel.CLUSTERING)
33 public class ReBalanceConsumer implements RocketMQListener<String>
34 {
35     @Resource
36     private ReBalanceService reBalanceService;
37
38     @Override
39     public void onMessage(String host)
40     {
41         log.debug("接收到ReBalance 消息");
42         reBalanceService.reBalance(host);
43     }
44 }

```

ReBalanceService接口

```
1 package mao.chat_room_manage.service;
2
3 /**
4  * Project name(项目名称): netty_chat_room
5  * Package(包名): mao.chat_room_manage.service
6  * Interface(接口名): ReBalanceService
7  * Author(作者): mao
8  * Author QQ: 1296193245
9  * GitHub: https://github.com/maomao124/
10 * Date(创建日期): 2023/4/13
11 * Time(创建时间): 15:45
12 * Version(版本): 1.0
13 * Description(描述): 无
14 */
15
16 public interface ReBalanceService
17 {
18     /**
19      * 重新平衡
20      *
21      * @param host 实例的地址
22      */
23     void reBalance(String host);
24 }
```

ReBalanceServiceImpl

```
1 package mao.chat_room_manage.service.impl;
2
3 import lombok.extern.slf4j.Slf4j;
4 import mao.chat_room_manage.entity.Instance;
5 import mao.chat_room_manage.entity.OnlineUserCount;
6 import mao.chat_room_manage.service.NettyService;
7 import mao.chat_room_manage.service.ReBalanceService;
8 import mao.chat_room_server_api.constants.RedisConstants;
9 import mao.chat_room_server_api.constants.UrlConstants;
10 import mao.tools_core.base.R;
11 import mao.tools_redis_cache.entity.LockInfo;
12 import mao.tools_redis_cache.service.RedisLockService;
13 import org.springframework.data.redis.core.StringRedisTemplate;
14 import org.springframework.stereotype.Service;
15 import org.springframework.web.client.RestTemplate;
16
17 import javax.annotation.Resource;
18 import java.util.ArrayList;
```

```

19 import java.util.List;
20
21 /**
22  * Project name(项目名称): netty_chat_room
23  * Package(包名): mao.chat_room_manage.service.impl
24  * Class(类名): ReBalanceServiceImpl
25  * Author(作者): mao
26  * Author QQ: 1296193245
27  * Github: https://github.com/maomao124/
28  * Date(创建日期): 2023/4/13
29  * Time(创建时间): 15:46
30  * Version(版本): 1.0
31  * Description(描述): 无
32  */
33
34 @Slf4j
35 @Service
36 public class ReBalanceServiceImpl implements ReBalanceService
37 {
38
39     @Resource
40     private RedisLockService redisLockService;
41
42     @Resource
43     private StringRedisTemplate stringRedisTemplate;
44
45     @Resource
46     private NettyService nettyService;
47
48     @Resource
49     private RestTemplate restTemplate;
50
51
52     @Override
53     public void reBalance(String host)
54     {
55         //加分布式锁
56         log.debug("尝试获取分布式锁");
57         String lockKey = RedisConstants.re_balance_lock_key;
58         LockInfo lockInfo = null;
59         try
60         {
61             lockInfo = redisLockService.lock(lockKey);
62             log.debug("获取分布式锁成功");
63             //获取当前时间
64             long now = System.currentTimeMillis();
65             String timeKey = RedisConstants.re_balance_time_key;
66             //从redis上获取时间
67             String timeString =
stringRedisTemplate.opsForValue().get(timeKey);
68             //判断是否有这个key
69             if (timeString == null)
70             {
71                 //没有
72                 timeString = "1";

```

```

73     }
74     //转换
75     long time = Long.parseLong(timeString);
76     log.debug("上次reBalance时间: " + time);
77     log.debug("当前时间: " + now);
78     //判断时间差是否小于120秒
79     if (now - time < 120000)
80     {
81         //间隔小于120秒
82         log.debug("不需要reBalance");
83     }
84     else
85     {
86         //间隔大于120秒
87         log.debug("需要reBalance");
88         //得到各实例用户在线人数
89         OnlineUserCount onlineUserCount =
90         nettyService.getOnlineUserCount();
91         //集群数量
92         List<Instance> instanceList =
93         onlineUserCount.getInstanceList();
94         int size = instanceList.size();
95         log.debug("集群数量: " + size);
96         log.debug("总在线人数: " + onlineUserCount.getTotalCount());
97         if (onlineUserCount.getTotalCount() < 150)
98         {
99             log.debug("人数太少, 暂时不需要reBalance");
100             return;
101         }
102         //平均每个实例分配的人数
103         long avgCount = onlineUserCount.getTotalCount() / size;
104         log.debug("平均每个实例分配的人数:" + avgCount);
105
106         log.debug("分配前: " + instanceList);
107
108         List<Instance> lowInstanceList = new ArrayList<>();
109         List<Instance> highInstanceList = new ArrayList<>();
110         List<Instance> resultInstanceList = new ArrayList<>();
111         for (Instance instance : instanceList)
112         {
113             if (instance.getCount() > avgCount)
114             {
115                 highInstanceList.add(instance);
116             }
117             else if (instance.getCount() < avgCount)
118             {
119                 lowInstanceList.add(instance);
120             }
121             else
122             {
123                 resultInstanceList.add(instance);
124             }
125         }
126
127         log.debug("人数较多的实例列表: " + highInstanceList);

```

```

126         log.debug("人数较少的实例列表: " + lowInstanceList);
127
128         while (true)
129         {
130             Instance highInstance = highInstanceList.get(0);
131             Instance lowInstance = lowInstanceList.get(0);
132             if ((highInstance.getCount() - avgCount) > (avgCount -
lowInstance.getCount()))
133             {
134                 log.debug("大于");
135                 long to = avgCount - lowInstance.getCount();
136                 log.debug("分配数量: " + to + " , " + highInstance +
" --> "
137                             + lowInstance);
138
139                 log.debug("low: " + lowInstance.getCount() + "-->"
+ (lowInstance.getCount() + to));
140                 log.debug("high: " + highInstance.getCount() + "--
>" + (highInstance.getCount() - to));
141
142                 lowInstance.setCount(lowInstance.getCount() + to);
143                 highInstance.setCount(highInstance.getCount() -
to);
144
145                 log.debug("发起请求: " + highInstance.getHost() + ",
数量: " + to);
146
147                 String url =
urlConstants.buildReBalanceUrl(highInstance.getHost(),
148                                 lowInstance.getHost(),
149                                 Math.toIntExact(to));
150                 log.debug("url:" + url);
151                 R r = restTemplate.postForObject(url, null,
R.class);
152
153                 if (r.getIsError())
154                 {
155                     log.warn("请求失败: " + r.getMsg());
156                 }
157                 else
158                 {
159                     log.debug("请求成功");
160                     //删除
161                     lowInstanceList.remove(lowInstance);
162                     resultInstanceList.add(lowInstance);
163                 }
164                 else if ((highInstance.getCount() - avgCount) <
(avgCount - lowInstance.getCount()))
165                 {
166                     log.debug("小于");
167                     long to = highInstance.getCount() - avgCount;
168                     log.debug("分配数量: " + to + " , " + highInstance +
" --> "
169                             + lowInstance);
170

```

```

171         log.debug("low: " + lowInstance.getCount() + "-->"
172 + (lowInstance.getCount() + to));
173
174         log.debug("high: " + highInstance.getCount() + "--
175 >" + (highInstance.getCount() - to));
176
177         lowInstance.setCount(lowInstance.getCount() + to);
178         highInstance.setCount(highInstance.getCount() -
179 to);
180
181         log.debug("发起请求: " + highInstance.getHost() + ",
182 数量: " + to);
183
184         String url =
185         UrlConstants.buildReBalanceUrl(highInstance.getHost(),
186         lowInstance.getHost(),
187         Math.toIntExact(to));
188         log.debug("url:" + url);
189         R r = restTemplate.postForObject(url, null,
190 R.class);
191
192         if (r.getIsError())
193         {
194             log.warn("请求失败: " + r.getMsg());
195         }
196         else
197         {
198             log.debug("请求成功");
199         }
200
201         //删除
202         highInstanceList.remove(highInstance);
203         resultInstanceList.add(highInstance);
204     }
205     else
206     {
207         log.debug("等于");
208         long to = highInstance.getCount() - avgCount;
209         log.debug("分配数量: " + to + " , " + highInstance +
210 " --> "
211 + lowInstance);
212
213         log.debug("low: " + lowInstance.getCount() + "-->"
214 + (lowInstance.getCount() + to));
215         log.debug("high: " + highInstance.getCount() + "--
216 >" + (highInstance.getCount() - to));
217
218         lowInstance.setCount(lowInstance.getCount() + to);
219         highInstance.setCount(highInstance.getCount() -
220 to);
221
222         log.debug("发起请求: " + highInstance.getHost() + ",
223 数量: " + to);
224
225         String url =
226         UrlConstants.buildReBalanceUrl(highInstance.getHost(),
227         lowInstance.getHost(),

```



```

214         Math.toIntExact(to));
215         log.debug("url:" + url);
216         R r = restTemplate.postForObject(url, null,
R.class);
217         if (r.getIsError())
218         {
219             log.warn("请求失败: " + r.getMsg());
220         }
221         else
222         {
223             log.debug("请求成功");
224         }
225
226         //删除
227         highInstanceList.remove(highInstance);
228         lowInstanceList.remove(lowInstance);
229         resultInstanceList.add(highInstance);
230         resultInstanceList.add(lowInstance);
231     }
232
233
234         if (highInstanceList.size() == 0 ||
lowInstanceList.size() == 0)
235         {
236             break;
237         }
238
239     }
240     resultInstanceList.addAll(highInstanceList);
241     resultInstanceList.addAll(lowInstanceList);
242
243     log.debug("分配结果: " + resultInstanceList);
244
245     stringRedisTemplate.opsForValue().set(timeKey,
String.valueOf(now));
246     }
247 }
248 finally
249 {
250     log.debug("释放分布式锁");
251     redisLockService.unlock(lockInfo);
252 }
253 }
254 }

```

ReBalanceController

位于netty服务

```
1 package mao.chat_room_netty_server.controller;
2
3 import io.swagger.annotations.Api;
4 import io.swagger.annotations.ApiOperation;
5 import lombok.extern.slf4j.Slf4j;
6 import mao.chat_room_netty_server.service.ReBalanceService;
7 import mao.tools_core.base.BaseController;
8 import mao.tools_core.base.R;
9 import org.springframework.web.bind.annotation.*;
10
11 import javax.annotation.Resource;
12
13 /**
14  * Project name(项目名称): netty_chat_room
15  * Package(包名): mao.chat_room_netty_server.controller
16  * Class(类名): ReBalanceController
17  * Author(作者): mao
18  * Author QQ: 1296193245
19  * GitHub: https://github.com/maomao124/
20  * Date(创建日期): 2023/4/13
21  * Time(创建时间): 15:33
22  * Version(版本): 1.0
23  * Description(描述): 负载均衡相关
24  */
25
26 @Slf4j
27 @Api(value = "reBalance", tags = "reBalance")
28 @RestController
29 @RequestMapping("/reBalance")
30 public class ReBalanceController extends BaseController
31 {
32     @Resource
33     private ReBalanceService reBalanceService;
34
35
36     /**
37      * ReBalance处理
38      * 随机从用户列表中抽 reBalanceNumber 的数量的用户，让他们重新负载均衡到 host 这
39      * 个新实例上
40      * 比如reBalanceNumber为7，host为56.87.28.29:2457，随机抽7个用户让他们重新连
41      * 接到56.87.28.29:2457这个host上
42      *
43      * @param host 实例的地址
44      * @param reBalanceNumber 重新平衡的数量
45      * @return {@link R}<{@link Boolean}>
46      */
47     @PostMapping("/handler")
48     @ApiOperation("ReBalance处理")
49     public R<Boolean> handler(@RequestParam String host,
50                               @RequestParam int reBalanceNumber)
```

```

50         reBalanceService.handler(host, reBalanceNumber);
51         return success();
52     }
53 }

```

ReBalanceService接口

```

1  package mao.chat_room_netty_server.service;
2
3  /**
4   * Project name(项目名称): netty_chat_room
5   * Package(包名): mao.chat_room_netty_server.service
6   * Interface(接口名): ReBalanceService
7   * Author(作者): mao
8   * Author QQ: 1296193245
9   * GitHub: https://github.com/maomao124/
10  * Date(创建日期): 2023/4/13
11  * Time(创建时间): 15:38
12  * Version(版本): 1.0
13  * Description(描述): ReBalance服务
14  */
15
16  public interface ReBalanceService
17  {
18      /**
19       * 处理程序，随机从用户列表中抽 reBalanceNumber 的数量的用户，让他们重新负载均衡到
20       host 这个新实例上
21       * 比如reBalanceNumber为7，host为56.87.28.29:2457，随机抽7个用户让他们重新连
22       接到56.87.28.29:2457这个host上
23       *
24       * @param host 实例的地址
25       * @param reBalanceNumber 重新平衡的数量
26       */
27      void handler(String host, int reBalanceNumber);
28  }

```

ReBalanceServiceImpl

```

1  package mao.chat_room_netty_server.service.impl;
2
3  import io.netty.channel.Channel;
4  import lombok.extern.slf4j.Slf4j;
5  import mao.chat_room_common.message.ReBalanceResponseMessage;
6  import mao.chat_room_netty_server.service.ReBalanceService;
7  import mao.chat_room_netty_server.session.Session;
8  import org.springframework.stereotype.Service;
9

```

```

10 import javax.annotation.Resource;
11 import java.util.List;
12
13 /**
14  * Project name(项目名称): netty_chat_room
15  * Package(包名): mao.chat_room_netty_server.service.impl
16  * Class(类名): ReBalanceServiceImpl
17  * Author(作者): mao
18  * Author QQ: 1296193245
19  * GitHub: https://github.com/maomao124/
20  * Date(创建日期): 2023/4/13
21  * Time(创建时间): 15:43
22  * Version(版本): 1.0
23  * Description(描述): 无
24  */
25
26 @Slf4j
27 @Service
28 public class ReBalanceServiceImpl implements ReBalanceService
29 {
30
31     @Resource
32     private Session session;
33
34     @Override
35     public void handler(String host, int reBalanceNumber)
36     {
37         synchronized (this)
38         {
39             log.debug("触发ReBalance, 数量: " + reBalanceNumber + ", 位置: " +
40 host);
41             List<Channel> channelList = session.reBalance(reBalanceNumber);
42             for (Channel channel : channelList)
43             {
44                 //通知用户重新连接到host上
45
46                 channel.writeAndFlush(ReBalanceResponseMessage.success(host).setSequenceId(
47 ));
48             }
49         }
50     }
51 }

```

SessionClusterImpl实现类的reBalance方法

```

1 @Override
2 public List<Channel> reBalance(int reBalanceNumber)
3 {
4     List<Channel> list = new ArrayList<>();
5     Set<Channel> channels = channelUsernameMap.keySet();
6     if (reBalanceNumber >= channels.size())

```

```
7      {
8          return list;
9      }
10     Iterator<Channel> iterator = channels.iterator();
11     for (int i = 0; i < reBalanceNumber; i++)
12     {
13         list.add(iterator.next());
14     }
15     return list;
16 }
```

未完成事项和存在的问题

未完成的事项

1. 客户端重连机制
2. 客户端HTTP请求超时重试、有限重试机制
3. 客户端接收reBalance消息包处理
4. java Swing客户端的设计与实现
5. 安卓客户端的设计与实现
6. 维护auth-server表数据
7. 管理后台前端

存在的问题

1. netty服务的IP直接暴露给客户端，有被DDos攻击的风险
2. web服务从naocs拉取服务列表时有二级缓存，但是未解决缓存击穿问题，需要加分布式锁来解决此问题
3. naocs拉取服务列表存在限流，对客户端而言，可能会阻塞5秒
4. 用户注册不是通过HTTP请求的方式注册的，用户的用户名消息没有做XSS过滤，管理服务分页查询用户信息时，可能会出现XSS攻击的现象

end

by mao
2022/04/22
