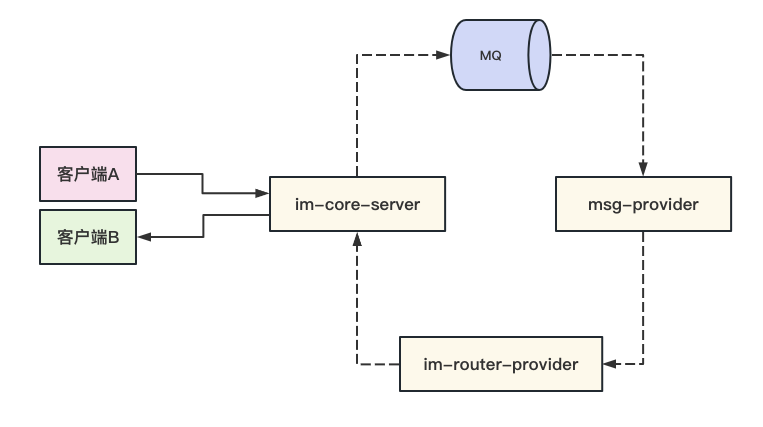
**7-20 ｜ IM系统的Router模块后续完善--用户路由信息绑定**

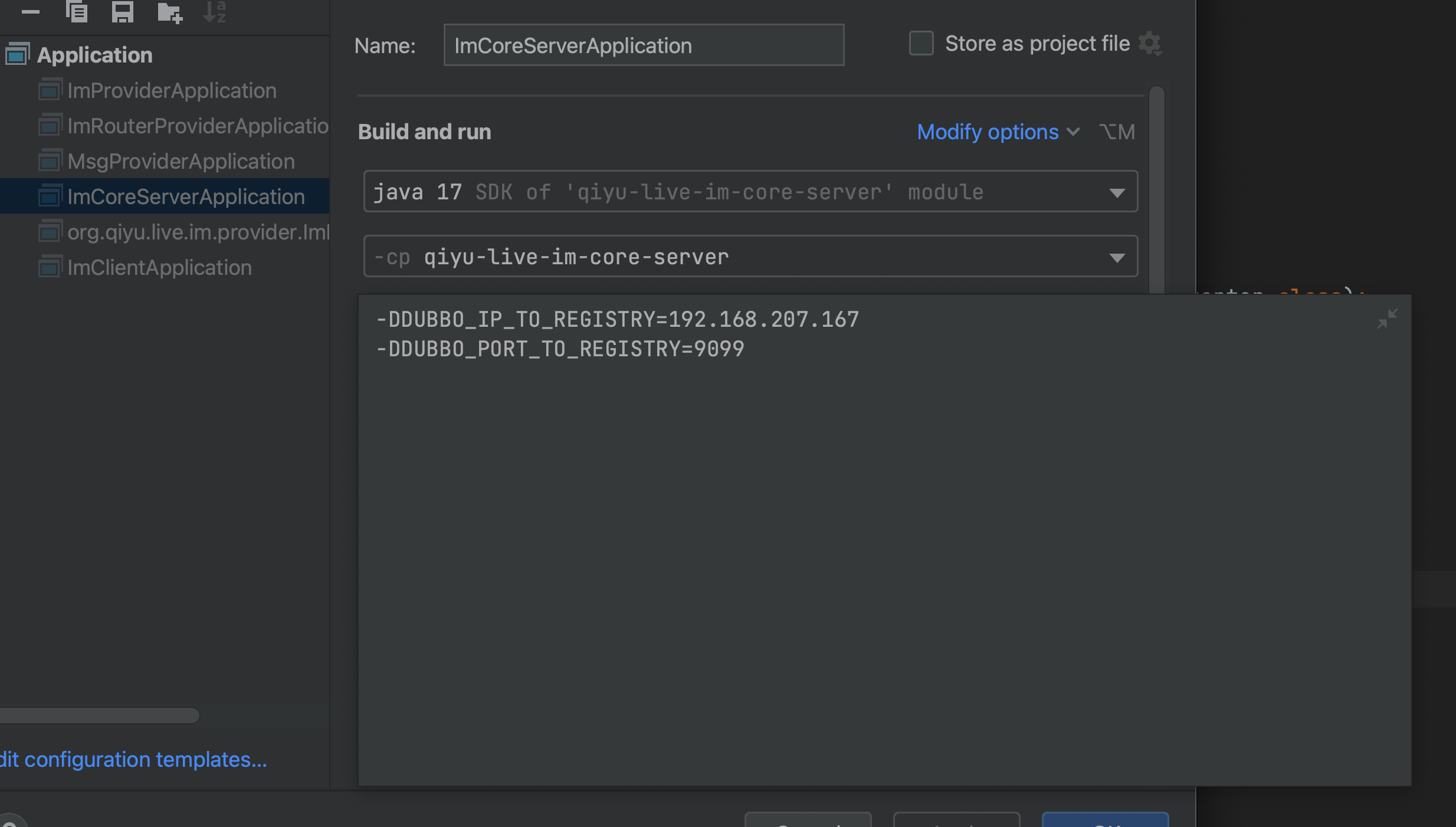
我们之前的服务调用了链路回顾：



其实这里的msg-provider和im-router-provider的关系可以利用mq去解耦，目前我们只是为了方便测试，这里暂时先这么实现，后边再调整为基于MQ的方式去进行交互。

**用户和连接IM服务器的关系记录**

服务启动的时候，将本地的服务ip地址记录下，涉及到改动的类为：NettyImServerStarter，内部需要使用到environment对象去获取我们启动时候设置的dubbo暴露端口和ip地址。如果你是在用IDEA的开发工具启动代码的话，可以加入一个-D的启动参数：



NettyImServerStarter类的完整代码如下：

|  |
| --- |
| Java package org.qiyu.live.im.core.server.starter;  import io.netty.bootstrap.ServerBootstrap; import io.netty.channel.Channel; import io.netty.channel.ChannelFuture; import io.netty.channel.ChannelInitializer; import io.netty.channel.nio.NioEventLoopGroup; import io.netty.channel.socket.nio.NioServerSocketChannel; import jakarta.annotation.Resource; import org.qiyu.live.im.core.server.common.ChannelHandlerContextCache; import org.qiyu.live.im.core.server.common.ImMsgDecoder; import org.qiyu.live.im.core.server.common.ImMsgEncoder; import org.qiyu.live.im.core.server.handler.ImServerCoreHandler; import org.slf4j.Logger; import org.slf4j.LoggerFactory; import org.springframework.beans.factory.InitializingBean; import org.springframework.beans.factory.annotation.Value; import org.springframework.context.annotation.Configuration; import org.springframework.core.env.Environment; import org.springframework.util.StringUtils;  /\*\*  \* @Author idea  \* @Date: Created in 20:35 2023/7/9  \* @Description  \*/ @Configuration public class NettyImServerStarter implements InitializingBean {   private static Logger LOGGER = LoggerFactory.getLogger(NettyImServerStarter.class);   //指定监听的端口  @Value("${qiyu.im.port}")  private int port;  @Resource  private ImServerCoreHandler imServerCoreHandler;  @Resource  private Environment environment;   //基于netty去启动一个java进程，绑定监听的端口  public void startApplication() throws InterruptedException {  //处理accept事件  NioEventLoopGroup bossGroup = new NioEventLoopGroup();  //处理read&write事件  NioEventLoopGroup workerGroup = new NioEventLoopGroup();  ServerBootstrap bootstrap = new ServerBootstrap();  bootstrap.group(bossGroup, workerGroup);  bootstrap.channel(NioServerSocketChannel.class);  //netty初始化相关的handler  bootstrap.childHandler(new ChannelInitializer<>() {  @Override  protected void initChannel(Channel ch) throws Exception {  //打印日志，方便观察  LOGGER.info("初始化连接渠道");  //设计消息体  //增加编解码器  ch.pipeline().addLast(new ImMsgDecoder());  ch.pipeline().addLast(new ImMsgEncoder());  ch.pipeline().addLast(imServerCoreHandler);  }  });  //基于JVM的钩子函数去实现优雅关闭  Runtime.getRuntime().addShutdownHook(new Thread(() -> {  bossGroup.shutdownGracefully();  workerGroup.shutdownGracefully();  }));  //获取im的服务注册ip和暴露端口  String registryIp = environment.getProperty("DUBBO\_IP\_TO\_REGISTR");  String registryPort = environment.getProperty("DUBBO\_PORT\_TO\_REGISTRY");  if (StringUtils.isEmpty(registryPort) || StringUtils.isEmpty(registryIp)) {  throw new IllegalArgumentException("启动参数中的注册端口和注册ip不能为空");  }  ChannelHandlerContextCache.setServerIpAddress(registryIp + ":" + registryPort);  ChannelFuture channelFuture = bootstrap.bind(port).sync();  LOGGER.info("服务启动成功，监听端口为{}", port);  //这里会阻塞掉主线程，实现服务长期开启的效果  channelFuture.channel().closeFuture().sync();  }   @Override  public void afterPropertiesSet() throws Exception {  Thread nettyServerThread = new Thread(new Runnable() {  @Override  public void run() {  try {  startApplication();  } catch (InterruptedException e) {  throw new RuntimeException(e);  }  }  });  nettyServerThread.setName("qiyu-live-im-server");  nettyServerThread.start();  } } |

在用户登录的时候，记录相关绑定的ip地址，涉及改动类为：LoginMsgHandler。其原理是在用户初次登陆我们的im服务的时候，利用redis的记录下用户id正连接的im服务器地址。其涉及到的LoginMsgHandler类代码如下：

|  |
| --- |
| Java package org.qiyu.live.im.core.server.handler.impl;  import com.alibaba.fastjson.JSON; import io.netty.channel.ChannelHandlerContext; import jakarta.annotation.Resource; import org.apache.dubbo.config.annotation.DubboReference; import org.qiyu.live.im.constants.ImConstants; import org.qiyu.live.im.constants.ImMsgCodeEnum; import org.qiyu.live.im.core.server.common.ChannelHandlerContextCache; import org.qiyu.live.im.core.server.common.ImContextUtils; import org.qiyu.live.im.core.server.common.ImMsg; import org.qiyu.live.im.core.server.handler.SimplyHandler; import org.qiyu.live.im.core.server.interfaces.constants.ImCoreServerConstants; import org.qiyu.live.im.dto.ImMsgBody; import org.qiyu.live.im.interfaces.ImTokenRpc; import org.slf4j.Logger; import org.slf4j.LoggerFactory; import org.springframework.data.redis.core.StringRedisTemplate; import org.springframework.stereotype.Component; import org.springframework.util.StringUtils;  import java.util.concurrent.TimeUnit;  /\*\*  \* 登录消息的处理逻辑统一收拢到这个类中  \*  \* @Author idea  \* @Date: Created in 20:40 2023/7/6  \* @Description  \*/ @Component public class LoginMsgHandler implements SimplyHandler {   private static final Logger LOGGER = LoggerFactory.getLogger(LoginMsgHandler.class);   @DubboReference  private ImTokenRpc imTokenRpc;  @Resource  private StringRedisTemplate stringRedisTemplate;   @Override  public void handler(ChannelHandlerContext ctx, ImMsg imMsg) {  //防止重复请求  if (ImContextUtils.getUserId(ctx) != null) {  return;  }  byte[] body = imMsg.getBody();  if (body == null || body.length == 0) {  ctx.close();  LOGGER.error("body error,imMsg is {}", imMsg);  throw new IllegalArgumentException("body error");  }  ImMsgBody imMsgBody = JSON.parseObject(new String(body), ImMsgBody.class);  Long userIdFromMsg = imMsgBody.getUserId();  Integer appId = imMsgBody.getAppId();  String token = imMsgBody.getToken();  if (StringUtils.isEmpty(token) || userIdFromMsg < 10000 || appId < 10000) {  ctx.close();  LOGGER.error("param error,imMsg is {}", imMsg);  throw new IllegalArgumentException("param error");  }  Long userId = imTokenRpc.getUserIdByToken(token);  //token校验成功，而且和传递过来的userId是同一个，则允许建立连接  if (userId != null && userId.equals(userIdFromMsg)) {  //按照userId保存好相关的channel对象信息  ChannelHandlerContextCache.put(userId, ctx);  ImContextUtils.setUserId(ctx, userId);  ImContextUtils.setAppId(ctx, appId);  //将im消息回写给客户端  ImMsgBody respBody = new ImMsgBody();  respBody.setAppId(appId);  respBody.setUserId(userId);  respBody.setData("true");  ImMsg respMsg = ImMsg.build(ImMsgCodeEnum.IM\_LOGIN\_MSG.getCode(), JSON.toJSONString(respBody));  stringRedisTemplate.opsForValue().set(ImCoreServerConstants.IM\_BIND\_IP\_KEY + appId + userId,  ChannelHandlerContextCache.getServerIpAddress(),  ImConstants.DEFAULT\_HEART\_BEAT\_GAP \* 2, TimeUnit.SECONDS);  LOGGER.info("[LoginMsgHandler] login success,userId is {},appId is {}", userId, appId);  ctx.writeAndFlush(respMsg);  return;  }  ctx.close();  LOGGER.error("token check error,imMsg is {}", imMsg);  throw new IllegalArgumentException("token check error");  } } |

在登出handler内部，当客户端断线的时候，将相关绑定的ip地址进行移除，即移除相关的redis记录即可

|  |
| --- |
| Java package org.qiyu.live.im.core.server.handler.impl;  import com.alibaba.fastjson.JSON; import io.netty.channel.ChannelHandlerContext; import jakarta.annotation.Resource; import org.qiyu.live.im.constants.ImMsgCodeEnum; import org.qiyu.live.im.core.server.common.ChannelHandlerContextCache; import org.qiyu.live.im.core.server.common.ImContextUtils; import org.qiyu.live.im.core.server.common.ImMsg; import org.qiyu.live.im.core.server.handler.SimplyHandler; import org.qiyu.live.im.core.server.interfaces.constants.ImCoreServerConstants; import org.qiyu.live.im.dto.ImMsgBody; import org.slf4j.Logger; import org.slf4j.LoggerFactory; import org.springframework.data.redis.core.StringRedisTemplate; import org.springframework.stereotype.Component;  /\*\*  \* 登出消息的处理逻辑统一收拢到这个类中  \*  \* @Author idea  \* @Date: Created in 20:40 2023/7/6  \* @Description  \*/ @Component public class LogoutMsgHandler implements SimplyHandler {   private static final Logger LOGGER = LoggerFactory.getLogger(LogoutMsgHandler.class);   @Resource  private StringRedisTemplate stringRedisTemplate;   @Override  public void handler(ChannelHandlerContext ctx, ImMsg imMsg) {  Long userId = ImContextUtils.getUserId(ctx);  Integer appId = ImContextUtils.getAppId(ctx);  if (userId == null || appId == null) {  LOGGER.error("attr error,imMsg is {}", imMsg);  //有可能是错误的消息包导致，直接放弃连接  ctx.close();  throw new IllegalArgumentException("attr is error");  }  //将im消息回写给客户端  ImMsgBody respBody = new ImMsgBody();  respBody.setAppId(appId);  respBody.setUserId(userId);  respBody.setData("true");  ImMsg respMsg = ImMsg.build(ImMsgCodeEnum.IM\_LOGOUT\_MSG.getCode(), JSON.toJSONString(respBody));  ctx.writeAndFlush(respMsg);  LOGGER.info("[LogoutMsgHandler] logout success,userId is {},appId is {}", userId, appId);  //理想情况下，客户端断线的时候，会发送一个断线消息包  ChannelHandlerContextCache.remove(userId);  stringRedisTemplate.delete(ImCoreServerConstants.IM\_BIND\_IP\_KEY + appId + userId);  ImContextUtils.removeUserId(ctx);  ImContextUtils.removeAppId(ctx);  ctx.close();  } } |

在心跳包请求后台服务器的时候，需要对之前用户id绑定的ip缓存记录进行时间的延长，从而增加缓存中记录的ip的有效性。

|  |
| --- |
| Java package org.qiyu.live.im.core.server.handler.impl;  import com.alibaba.fastjson.JSON; import io.netty.channel.ChannelHandlerContext; import jakarta.annotation.Resource; import org.idea.qiyu.live.framework.redis.starter.key.ImCoreServerProviderCacheKeyBuilder; import org.qiyu.live.im.constants.ImConstants; import org.qiyu.live.im.constants.ImMsgCodeEnum; import org.qiyu.live.im.core.server.common.ImContextUtils; import org.qiyu.live.im.core.server.common.ImMsg; import org.qiyu.live.im.core.server.handler.SimplyHandler; import org.qiyu.live.im.core.server.interfaces.constants.ImCoreServerConstants; import org.qiyu.live.im.dto.ImMsgBody; import org.slf4j.Logger; import org.slf4j.LoggerFactory; import org.springframework.data.redis.core.RedisTemplate; import org.springframework.data.redis.core.StringRedisTemplate; import org.springframework.stereotype.Component;  import java.util.concurrent.TimeUnit;  /\*\*  \* 心跳消息处理器  \*  \* @Author idea  \* @Date: Created in 20:41 2023/7/6  \* @Description  \*/ @Component public class HeartBeatImMsgHandler implements SimplyHandler {   private static final Logger LOGGER = LoggerFactory.getLogger(HeartBeatImMsgHandler.class);   @Resource  private RedisTemplate<String, Object> redisTemplate;  @Resource  private StringRedisTemplate stringRedisTemplate;  @Resource  private ImCoreServerProviderCacheKeyBuilder cacheKeyBuilder;   @Override  public void handler(ChannelHandlerContext ctx, ImMsg imMsg) {  //心跳包基本校验  Long userId = ImContextUtils.getUserId(ctx);  Integer appId = ImContextUtils.getAppId(ctx);  if (userId == null || appId == null) {  LOGGER.error("attr error,imMsg is {}", imMsg);  //有可能是错误的消息包导致，直接放弃连接  ctx.close();  throw new IllegalArgumentException("attr is error");  }  //心跳包record记录，redis存储心跳记录  String redisKey = cacheKeyBuilder.buildImLoginTokenKey(userId, appId);  this.recordOnlineTime(userId, redisKey);  this.removeExpireRecord(redisKey);  redisTemplate.expire(redisKey, 5, TimeUnit.MINUTES);  //延长用户之前保存的ip绑定记录时间  stringRedisTemplate.expire(ImCoreServerConstants.IM\_BIND\_IP\_KEY + appId + userId, ImConstants.DEFAULT\_HEART\_BEAT\_GAP \* 2, TimeUnit.SECONDS);  ImMsgBody msgBody = new ImMsgBody();  msgBody.setUserId(userId);  msgBody.setAppId(appId);  msgBody.setData("true");  ImMsg respMsg = ImMsg.build(ImMsgCodeEnum.IM\_HEARTBEAT\_MSG.getCode(), JSON.toJSONString(msgBody));  LOGGER.info("[HeartBeatImMsgHandler] imMsg is {}", imMsg);  ctx.writeAndFlush(respMsg);  }   /\*\*  \* 清理掉过期不在线的用户留下的心跳记录(在两次心跳包的发送间隔中，如果没有重新更新score值，就会导致被删除)  \*  \* @param redisKey  \*/  private void removeExpireRecord(String redisKey) {  redisTemplate.opsForZSet().removeRangeByScore(redisKey, 0, System.currentTimeMillis() - ImConstants.DEFAULT\_HEART\_BEAT\_GAP \* 1000 \* 2);  }   /\*\*  \* 记录用户最近一次心跳时间到zSet上  \*  \* @param userId  \* @param redisKey  \*/  private void recordOnlineTime(Long userId, String redisKey) {  redisTemplate.opsForZSet().add(redisKey, userId, System.currentTimeMillis());  }   } |

**Router层的指定动态IP进行RPC转发**

前边我们已经对Router层进行了SPI的相关拓展，实现了基于RPC上下文的ip参数，去进行服务提供者路由选择时候的确定。那么下边，我们就需要在router层的服务发送前，从redis中将ip地址取出，并且将其塞入到dubbo的上下文中：（这里有个要注意的点，就是我们的qiyu-live-im-router-provider是需要和qiyu-live-im-core-server服务访问同一个redis集群的）

这里我们需要修改下ImRouterServiceImpl类。

|  |
| --- |
| Java package org.qiyu.live.im.router.provider.service.impl;  import jakarta.annotation.Resource; import org.apache.dubbo.config.annotation.DubboReference; import org.apache.dubbo.rpc.RpcContext; import org.qiyu.live.im.constants.ImConstants; import org.qiyu.live.im.core.server.interfaces.constants.ImCoreServerConstants; import org.qiyu.live.im.core.server.interfaces.rpc.IRouterHandlerRpc; import org.qiyu.live.im.dto.ImMsgBody; import org.qiyu.live.im.router.provider.service.ImRouterService; import org.springframework.data.redis.core.StringRedisTemplate; import org.springframework.stereotype.Service; import org.springframework.util.StringUtils;   /\*\*  \* @Author idea  \* @Date: Created in 10:30 2023/7/12  \* @Description  \*/ @Service public class ImRouterServiceImpl implements ImRouterService {   @DubboReference  private IRouterHandlerRpc routerHandlerRpc;  @Resource  private StringRedisTemplate stringRedisTemplate;   @Override  public boolean sendMsg(ImMsgBody imMsgBody) {  String bindAddress = stringRedisTemplate.opsForValue().get(ImCoreServerConstants.IM\_BIND\_IP\_KEY + imMsgBody.getAppId() + imMsgBody.getUserId());  if (StringUtils.isEmpty(bindAddress)) {  return false;  }  RpcContext.getContext().set("ip", bindAddress);  routerHandlerRpc.sendMsg(imMsgBody);  return true;  } } |

这里要注意，我们调用的类是im-core-server对外暴露的一个rpc接口，该接口的参数，我们做了一些优化，将原先的两个参数变成了一个ImMsgBody对象。

IRouterHandlerRpc接口：

|  |
| --- |
| Java public interface IRouterHandlerRpc {   /\*\*  \* 按照用户id进行消息的发送  \*  \* @param imMsgBody  \*/  void sendMsg(ImMsgBody imMsgBody); } |

RouterHandlerRpcImpl实现类：

|  |
| --- |
| Java @DubboService public class RouterHandlerRpcImpl implements IRouterHandlerRpc {   @Resource  private IRouterHandlerService routerHandlerService;   @Override  public void sendMsg(ImMsgBody imMsgBody) {  routerHandlerService.onReceive(imMsgBody);  } } |

当Router将消息发送到指定的IM服务器之后，我们就需要在im-core-server层中对这部分的逻辑去进行更加深入的一个实现，RouterHandlerServiceImpl类的实现如下：

|  |
| --- |
| Java @Service public class RouterHandlerServiceImpl implements IRouterHandlerService {   @Override  public void onReceive(ImMsgBody imMsgBody) {  //需要进行消息通知的userid  Long userId = imMsgBody.getUserId();  ChannelHandlerContext ctx = ChannelHandlerContextCache.get(userId);  if (ctx != null) {  ImMsg respMsg = ImMsg.build(ImMsgCodeEnum.IM\_BIZ\_MSG.getCode(), JSON.toJSONString(imMsgBody));  ctx.writeAndFlush(respMsg);  }  } } |

**业务服务如何通知到Router层**

接着在我们的msg-provider服务里面，需要修改mq的消费逻辑，当接收到消息之后，应该是交给我们本地的一个方法去处理。

|  |
| --- |
| Java package org.qiyu.live.msg.provider.consumer;  import com.alibaba.fastjson.JSON; import jakarta.annotation.Resource; import org.apache.rocketmq.client.consumer.DefaultMQPushConsumer; import org.apache.rocketmq.client.consumer.listener.ConsumeConcurrentlyStatus; import org.apache.rocketmq.client.consumer.listener.MessageListenerConcurrently; import org.apache.rocketmq.common.consumer.ConsumeFromWhere; import org.apache.rocketmq.common.message.MessageExt; import org.qiyu.live.common.interfaces.topic.ImCoreServerProviderTopicNames; import org.qiyu.live.framework.mq.starter.properties.RocketMQConsumerProperties; import org.qiyu.live.im.dto.ImMsgBody; import org.qiyu.live.msg.provider.consumer.handler.MessageHandler; import org.slf4j.Logger; import org.slf4j.LoggerFactory; import org.springframework.beans.factory.InitializingBean; import org.springframework.stereotype.Component;   /\*\*  \* @Author idea  \* @Date: Created in 15:04 2023/7/11  \* @Description  \*/ @Component public class ImMsgConsumer implements InitializingBean {   private static final Logger LOGGER = LoggerFactory.getLogger(ImMsgConsumer.class);  @Resource  private RocketMQConsumerProperties rocketMQConsumerProperties;  @Resource  private MessageHandler singleMessageHandler;   // 记录每个用户连接的im服务器地址，然后根据im服务器的连接地址去做具体机器的调用  // 基于mq广播思路去做，可能会有消息风暴发生，100台im机器，99%的mq消息都是无效的，  // 加入一个叫路由层的设计，router中转的设计，router就是一个dubbo的rpc层  // A--》B im-core-server -> msg-provider(持久化) -> im-core-server -> 通知到b  @Override  public void afterPropertiesSet() throws Exception {  DefaultMQPushConsumer mqPushConsumer = new DefaultMQPushConsumer();  //老版本中会开启，新版本的mq不需要使用到  mqPushConsumer.setVipChannelEnabled(false);  mqPushConsumer.setNamesrvAddr(rocketMQConsumerProperties.getNameSrv());  mqPushConsumer.setConsumerGroup(rocketMQConsumerProperties.getGroupName() + "\_" + ImMsgConsumer.class.getSimpleName());  //一次从broker中拉取10条消息到本地内存当中进行消费  mqPushConsumer.setConsumeMessageBatchMaxSize(10);  mqPushConsumer.setConsumeFromWhere(ConsumeFromWhere.CONSUME\_FROM\_FIRST\_OFFSET);  //监听im发送过来的业务消息topic  mqPushConsumer.subscribe(ImCoreServerProviderTopicNames.QIYU\_LIVE\_IM\_BIZ\_MSG\_TOPIC, "");  mqPushConsumer.setMessageListener((MessageListenerConcurrently) (msgs, context) -> {  for (MessageExt msg : msgs) {  ImMsgBody imMsgBody = JSON.parseObject(new String(msg.getBody()), ImMsgBody.class);  singleMessageHandler.onMsgReceive(imMsgBody);  }  return ConsumeConcurrentlyStatus.CONSUME\_SUCCESS;  });  mqPushConsumer.start();  LOGGER.info("mq消费者启动成功,namesrv is {}", rocketMQConsumerProperties.getNameSrv());  } } |

对应的handler代码如下，目前我们是不做过多的处理，直接将消息给发送回给router层服务，然后由router将消息传递给消息的接收方。

|  |
| --- |
| Java package org.qiyu.live.msg.provider.consumer.handler;  import org.qiyu.live.im.dto.ImMsgBody;  /\*\*  \* @Author idea  \* @Date: Created in 22:40 2023/7/14  \* @Description  \*/ public interface MessageHandler {   /\*\*  \* 处理im服务投递过来的消息内容  \*  \* @param imMsgBody  \*/  void onMsgReceive(ImMsgBody imMsgBody); } |

MessageHandler接口的实现类如下：

|  |
| --- |
| Java package org.qiyu.live.msg.provider.consumer.handler.impl;  import com.alibaba.fastjson.JSON; import com.alibaba.fastjson.JSONObject; import org.apache.dubbo.config.annotation.DubboReference; import org.qiyu.live.im.constants.AppIdEnum; import org.qiyu.live.im.dto.ImMsgBody; import org.qiyu.live.im.router.interfaces.rpc.ImRouterRpc; import org.qiyu.live.msg.dto.MessageDTO; import org.qiyu.live.msg.enums.ImMsgBizCodeEnum; import org.qiyu.live.msg.provider.consumer.handler.MessageHandler; import org.springframework.stereotype.Component;  /\*\*  \* @Author idea  \* @Date: Created in 22:41 2023/7/14  \* @Description  \*/ @Component public class SingleMessageHandlerImpl implements MessageHandler {   @DubboReference  private ImRouterRpc routerRpc;   @Override  public void onMsgReceive(ImMsgBody imMsgBody) {  int bizCode = imMsgBody.getBizCode();  //直播间的聊天消息  if (ImMsgBizCodeEnum.LIVING\_ROOM\_IM\_CHAT\_MSG\_BIZ.getCode() == bizCode) {  MessageDTO messageDTO = JSON.parseObject(imMsgBody.getData(), MessageDTO.class);  //暂时不做过多的处理  ImMsgBody respMsg = new ImMsgBody();  respMsg.setUserId(messageDTO.getObjectId());  respMsg.setAppId(AppIdEnum.QIYU\_LIVE\_BIZ.getCode());  respMsg.setBizCode(ImMsgBizCodeEnum.LIVING\_ROOM\_IM\_CHAT\_MSG\_BIZ.getCode());  JSONObject jsonObject = new JSONObject();  jsonObject.put("senderId", messageDTO.getUserId());  jsonObject.put("content", messageDTO.getContent());  respMsg.setData(jsonObject.toJSONString());  routerRpc.sendMsg(respMsg);  }  } } |

到此，我们基本上将消息的发送和接收部分逻辑给实现得差不多了，下一节里面，我们将对整个链路的测试