

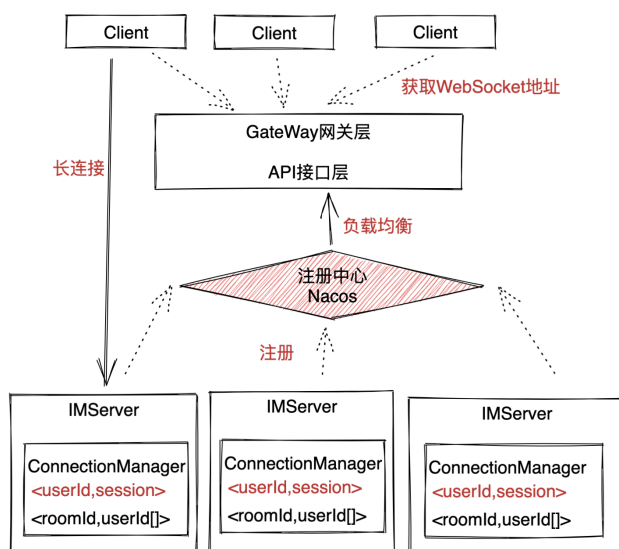
- 一、当前IM服务集群化问题分析
- 二、IM服务集群化设计方案
- 三、RocketMQ的作用
 - 1、RocketMQ的作用
 - 2、RocketMQ客户端功能演示
- 四、实现IM集群化
- 五、IM集群化方案总结与拓展

四、实现IM服务集群化

阶段目标：实现IM服务集群化

一、当前IM服务集群化问题分析

当前我们实现的IM服务，整体功能结构如下图：



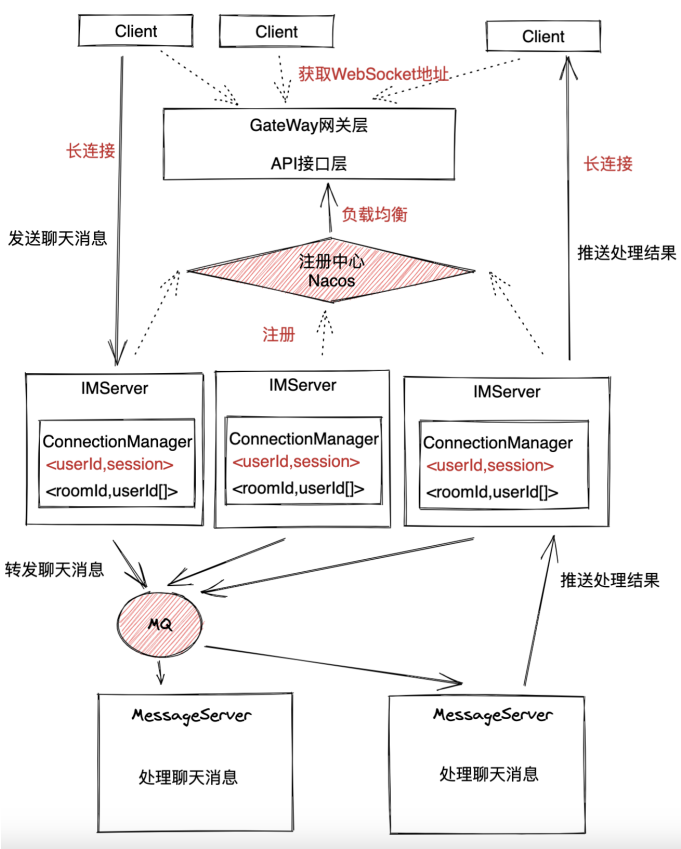
核心问题：虽然IMServer实现了请求接入时的负载均衡功能。但是，每个IMServer的ConnectionManager中保存的CHANNEL_CONTAINER和ROOM_CONTAINER数据还只是保存在每个应用内存当中。因此，如果多个前端的长连接建立在不同的IMServer上，那么就算他们进入了同一个直播间，也是无法实现跨进程的消息推送的。因此我们还需要实现多个IMServer服务实例的集群化设计。

二、IM服务集群化设计方案

通过之前的分析，目前我们的IM服务无法集群化的问题，还是在于多个IMServer的数据无法共享。那么我们就需要设计一种机制，让多个IMServer之间的数据可以打通。

对于ROOM_CONTAINER数据，由于他们都是保存的一些基础数据类型，所以，要打通这些数据，方案就比较简单。直接将这些数据转移到Redis中进行持久化保存，就可以了。

但是对于CHANNEL_CONTAINER数据呢？由于Websocket的Session对象是无法序列化的，也就无法在不同进程之间进行传递。所以也就无法将他们像ROOM_CONTAINER一样直接转移了。这时我们可以换一种思路，既然数据无法打通，那就将消息的处理过程从IMServer中移出来，转移到另外一个第三方的消息处理服务，在消息处理服务中进行统一的处理。然后再将处理结果通知给各个IMServer实例，从而实现消息的统一处理。



核心是对于客户发送过来的聊天消息，通过MQ通知给MessageServer。

而MessageServer处理的结果，还是可以通过我们之前开发的发送房间公告的接口推送回IMServer。

三、RocketMQ的作用

1、RocketMQ的作用

所有MQ的作用，归结为三个关键字：异步、解耦、削峰

- 1、异步：生产者发送消息和消费者消费消息可以不在同一个时间点。
- 2、解耦：生产者发送消息和消费者消费消息的逻辑是完全分开的，不需要相互依赖。
- 3、削峰：MQ中可以把消息暂存起来，慢慢向业务方推送。避免大流量下，业务方负载过大，造成服务崩溃。

从这三个方面思考下，为什么在IMServer后要引入RocketMQ，有什么好处。

2、RocketMQ客户端功能演示

RocketMQ的客户端实现方式非常多，项目中演示与SpringBoot集成的使用方式。

核心pom依赖

```

<!-- 主pom 版本控制 -->
<rocketmq.springboot.version>2.3.0</rocketmq.springboot.version>
<rocketmq.version>5.2.0</rocketmq.version>

<!-- tl-im-server 组件引用-->
<dependency>
    <groupId>org.apache.rocketmq</groupId>
    <artifactId>rocketmq-spring-boot-starter</artifactId>
    <version>${rocketmq.springboot.version}</version>
    <exclusions>
        <exclusion>
            <groupId>org.apache.rocketmq</groupId>
            <artifactId>rocketmq-client</artifactId>
        </exclusion>
        <exclusion>
            <artifactId>rocketmq-acl</artifactId>
            <groupId>org.apache.rocketmq</groupId>
        </exclusion>
    </exclusions>
</dependency>
<dependency>
    <groupId>org.apache.rocketmq</groupId>
    <artifactId>rocketmq-client</artifactId>
    <version>${rocketmq.version}</version>
</dependency>
<dependency>
    <groupId>org.apache.rocketmq</groupId>
    <artifactId>rocketmq-acl</artifactId>
    <version>${rocketmq.version}</version>
</dependency>

```

关键配置：

```

rocketmq:
  name-server: 192.168.65.212:9876
  #必须配置producer
  producer:
    group: tl-live-im-producer
  consumer:
    group: tl-live-im-busi

```

SpringBoot封装RocketMQ生产者方式和Redis基本相似，都是用过注入一个Template对象来实现发消息。

```

@SpringBootTest
@RunWith(SpringRunner.class)
public class RocketMQTest {

    @Resource
    private RocketMQTemplate rocketMQTemplate;

    @Test
    public void sendMQTest(){
        rocketMQTemplate.convertAndSend(IMConstants.MQ_TOPIC_CHAT,"hello world");
        System.out.println("发送成功");
    }
}

```

消费者只需要实现RocketMQListener接口，并添加@RocketMQMessageListener注解

```

@Component
@RocketMQMessageListener(topic = IMConstants.MQ_TOPIC_CHAT,consumerGroup = "tl-live-im-busi")
public class ChatMessageConsumer implements RocketMQListener<String> {
    Logger logger = LoggerFactory.getLogger(ChatMessageConsumer.class);
    @Override
    public void onMessage(String message) {
        logger.info("接收到消息: {}",message);
    }
}

```

RocketMQ的其他核心用法参见官方：

<https://rocketmq.apache.org/zh/docs/featureBehavior/01normalmessage>

四、实现IM集群化

实现步骤：

1、将ConnectionManager中的ROOM_CONTAINER数据转移到Redis中。

直接在ConnectionManager中将ROOM_CONTAINER删除，将所有ROOM_CONTAINER的操作转为对Redis中一个Set结果数据的操作。

2、IMServer引入MQ，推送消息。

引入RocketMQ依赖。

在配置文件中添加rocketmq的配置

3、在MessageHandlerService中对房间聊天消息，改为只往RocketMQ发送消息

MessageHandlerService中对房间聊天消息，只往RocketMQ中发送消息。

3、tl-live-im-busi中添加消费者，接收MQ的消息。

消费者的业务处理逻辑：

1、对消息进行过滤 - 省略

2、对消息进行整合 - 基于Redis简单实现。当同一个直播间的消息缓存超过5条(可能是不同的客户发过来的消息)时，进行一次批量推送。

4、tl-live-im-busi中处理完消息后，将下行消息推送到IMServer

tl-live-im-busi中需要通知集群中所有的IMServer实例，而不是某一个实例。

@DubboReference(cluster = ClusterRules.BROADCAST) 这种方式会依次调用所有Provider。并且只有在所有Provider都成功后才返回true，有任何一个Provider失败都返回false。这种方式会造成阻塞，性能比较低，不是dubbo推荐的方式。但是在现在这个场景还是勉强可以用的。

5、IMServer从Redis中查找房间的所属用户，并与本地CHAT_CONTAINER中的用户ID进行比对。

本地有用户记录，就找到对应的Session，推送消息。

五、IM集群化方案总结与拓展

现在已经实现了一个基础的IM架构了。那么，目前的IM系统还有哪些问题？后续有哪些改进方案呢？

多思考，这或许比写代码更有作用。