链滴

Spring Websocket 在负载均衡下的配置

作者：714593351
原文链接：https://ld246.com/article/1512177012579
来源网站：链滴

上一篇 Spring Websocket 配置 介绍了单机下的服务端 websocket 配置。

这种实现方法，在多台服务器的情况下会出问题。因为我这个业务场景是经常服务器主动下发消息，以会将 WebSocketSession 缓存到内存里，在需要的时候根据用户主键去查找对应的连接发送数据。

那么在多台服务器这种内存缓存就不凑效了，怎么办呢？第一想法是把 WebSocketSession 集中缓，如缓存到 Redis 中。然而 WebSocketSession 不支持序列化，无法存储 redis 中。

既然无法集中缓存，那么，我们在需要发送数据时，分别向各台服务器发送通知：请向用户 A 发消息每台服务器收到通知后，分别遍历自己缓存内的 WebSocketSession 如果有用户 A 的连接，则发消息即可。这不就是 MQ 发布/订阅模式的应用场景嘛？

恰好，我们系统使用了 redis ，而 redis 支持发布/订阅模式，那就开始改造吧。

1. redis 配置

```
<context:annotation-config/>
   <bean class="org.springframework.session.data.redis.config.annotation.web.http.RedisHttSessionConfiguration">
      <!-- 超时时间（秒） -->
      <property name="maxInactiveIntervalInSeconds" value="3600" />
   </bean>

   <bean id="jedisPoolConfig" class="redis.clients.jedis.JedisPoolConfig">
      <property name="maxTotal" value="30"/>
      <property name="maxIdle" value="10"/>
      <property name="minIdle" value="1"/>
      <property name="maxWaitMillis" value="30000"/>
      <property name="testOnBorrow" value="true"/>
      <property name="testOnReturn" value="false"/>
      <property name="testWhileIdle" value="false"/>
   </bean>

   <!--2-->
   <bean id="jedisConnectionFactory" class="org.springframework.data.redis.connection.jedis.JedisConnectionFactory">
      <property name="hostName" value="${redis_server}" />
      <property name="port" value="6379" />
      <property name="password" value="${redis_password}" />
      <property name="timeout" value="3000" />
      <property name="poolConfig" ref="jedisPoolConfig" />
      <property name="usePool" value="true" />
   </bean>

   <bean id="redisTemplate" class="org.springframework.data.redis.core.RedisTemplate">
      <property name="connectionFactory" ref="jedisConnectionFactory"/>
      <property name="defaultSerializer">
         <bean class="org.springframework.data.redis.serializer.StringRedisSerializer"/>
      </property>
   </bean>

   <bean id="websocketTopicMessageListener" class="com.xx.xx.websocket.redisListener.WesocketTopicMessageListener">
```

```
    </bean>

    <bean id="topicContainer" class="org.springframework.data.redis.listener.RedisMessageLis
enerContainer" destroy-method="destroy">
        <property name="connectionFactory" ref="jedisConnectionFactory"/>
        <property name="taskExecutor"><!-- 此处有个奇怪的问题，无法正确使用其他类型的Execu
or -->
            <bean class="org.springframework.scheduling.concurrent.ThreadPoolTaskScheduler"

                <property name="poolSize" value="3"></property>
            </bean>
        </property>
        <property name="messageListeners">
            <map>
                <entry key-ref="websocketTopicMessageListener">
                    <bean class="org.springframework.data.redis.listener.ChannelTopic">
                        <constructor-arg value="websocket:sendMsgTopic"/>
                    </bean>
                </entry>
            </map>
        </property>
    </bean>
```

上面配置中，我们定义了一个 Topic：websocket:sendMsgTopic 并定义了对应的监听器 websocke TopicMessageListener。

监听器的实现如下，实现很简单，收到订阅的消息后通过 userMessagesHandler.sendMessageToU er() 方法向 websocket 连接发送数据。

```
public class WebsocketTopicMessageListener implements MessageListener {

    @Resource
    private RedisTemplate redisTemplate;
    @Resource
    private UserMessagesHandler userMessagesHandler;

    @Override
    @Transactional(readOnly = true)
    public void onMessage(Message message, byte[] pattern) {
        byte[] body = message.getBody();
        byte[] channel = message.getChannel();
        String itemValue = (String) redisTemplate.getValueSerializer().deserialize(body);
        String topic = (String) redisTemplate.getStringSerializer().deserialize(channel);

        Gson gson = new Gson();
        UserNotice userNotice = gson.fromJson(itemValue, UserNotice.class);
        if (null != userNotice) {
            User user = userDAO.get(userNotice.getUserId());
            userMessagesHandler.sendMessageToUser(user.getLoginName(), new TextMessage(it
mValue));
        }
    }
}
```

所以，之前调用 userMessagesHandler.sendMessageToUser() 的地方就可以改为向 Topic：websocket:sendMsgTopic 发布消息了。

```
public void sendNotification(UserNotice userNotice) {
        String channel = "websocket:sendMsgTopic";
        Gson gson = new Gson();
        redisTemplate.convertAndSend(channel, gson.toJson(userNotice));
    }
```

通过以上改造，我们的 websocket 就能支持多服务器负载均衡部署了。