在Spring Boot中使用Spring Data Redis实现基于“发布/订阅”模型的消息队列

2018/10/15 | Spring (https://www.zifangsky.cn/java/spring) | admin (https://www.zifangsky.cn/author/zifangskyw)

一 简介

我们常听说的MQ中间件通常有ActiveMQ、RabbitMQ、Kafka等等。除此之外，Redis也可以用作基于“发布/订阅”模型的消息推送，不过Redis实现的是一种简单的消息队列，不仅在可靠性方面比不上其他专业的消息中间件，而且Redis的消息推送也不支持Topic分组、点对点模型的消息队列。

然而，如果我们已经在项目中使用Redis作数据缓存，同时我们的消息推送数量也不大，对可靠性要求也不是特别高，那么我们就可以使用Redis来实现消息队列了。

注：

- 关于消息队列的基本概念可以参考我之前的这篇文章：https://www.zifangsky.cn/815.html (https://www.zifangsky.cn/815.html)
- 关于Kafka实现消息推送的基本用法可以参考我的这篇文章：https://www.zifangsky.cn/1231.html (https://www.zifangsky.cn/1231.html)

二 Spring Data Redis实现基于“发布/订阅”模型的消息队列

（1）在pom.xml文件中添加相关依赖：

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
```

（2）添加一个消息接收者：

这个消息接收者只是一个简单的POJO，主要包含一个怎么处理接收到的消息的方法。

```java
package cn.zifangsky.model;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.text.MessageFormat;

/**
 * Redis message receiver
 *
 * @author zifangsky
 * @date 2018/10/15
 * @since 1.0.0
 */
public class Receiver {
    private final Logger logger = LoggerFactory.getLogger(getClass());

    public void receiveMessage(User message) {
        logger.info(MessageFormat.format("Received Message: {0}", message));
    }

}
```

需要注意的是，receiveMessage方法中cn.zifangsky.model.User类型的参数“message”表示我们在后面发送到Redis中的消息类型也是cn.zifangsky.model.User类型，二者需要一一对应（或者也使用Object接收）。

（3）Redis相关的JavaConfig配置：

```java
package cn.zifangsky.config;

import cn.zifangsky.model.Receiver;
import com.fasterxml.jackson.annotation.JsonAutoDetect;
import com.fasterxml.jackson.annotation.PropertyAccessor;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.autoconfigure.condition.ConditionalOnClass;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisClusterConfiguration;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.connection.jedis.JedisConnectionFactory;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.data.redis.listener.PatternTopic;
import org.springframework.data.redis.listener.RedisMessageListenerContainer;
import org.springframework.data.redis.listener.adapter.MessageListenerAdapter;
import org.springframework.data.redis.serializer.Jackson2JsonRedisSerializer;
import org.springframework.data.redis.serializer.StringRedisSerializer;
import redis.clients.jedis.JedisCluster;
import redis.clients.jedis.JedisPoolConfig;

import java.util.Arrays;

/**
 * Redis相关配置
 *
```

```
 * @author zifangsky
 * @date 2018/7/30
 * @since 1.0.0
 */
@Configuration
@ConditionalOnClass({JedisCluster.class})
public class RedisConfig {

    @Value("${spring.redis.timeout}")
    private String timeOut;

    @Value("${spring.redis.cluster.nodes}")
    private String nodes;

    @Value("${spring.redis.cluster.max-redirects}")
    private int maxRedirects;

    @Value("${spring.redis.jedis.pool.max-active}")
    private int maxActive;

    @Value("${spring.redis.jedis.pool.max-wait}")
    private int maxWait;

    @Value("${spring.redis.jedis.pool.max-idle}")
    private int maxIdle;

    @Value("${spring.redis.jedis.pool.min-idle}")
    private int minIdle;

    @Bean
    public JedisPoolConfig jedisPoolConfig(){
        JedisPoolConfig config = new JedisPoolConfig();
        config.setMaxTotal(maxActive);
        config.setMaxIdle(maxIdle);
        config.setMinIdle(minIdle);
        config.setMaxWaitMillis(maxWait);

        return config;
    }

    @Bean
    public RedisClusterConfiguration redisClusterConfiguration(){
        RedisClusterConfiguration configuration = new RedisClusterConfiguration(Array
        configuration.setMaxRedirects(maxRedirects);

        return configuration;
    }

    /**
     * JedisConnectionFactory
     */
    @Bean
    public JedisConnectionFactory jedisConnectionFactory(RedisClusterConfiguration co
        return new JedisConnectionFactory(configuration,jedisPoolConfig);
    }

    /**
     * 使用Jackson序列化对象
```

```
86      * @author zifangsky
87      * @date 2018/7/30 16:16
88      * @since 1.0.0
89      * @return org.springframework.data.redis.serializer.Jackson2JsonRedisSerializer<
90      */
91     @Bean
92     public Jackson2JsonRedisSerializer<Object> jackson2JsonRedisSerializer(){
93         Jackson2JsonRedisSerializer<Object> serializer = new Jackson2JsonRedisSeriali
94 
95         ObjectMapper objectMapper = new ObjectMapper();
96         objectMapper.setVisibility(PropertyAccessor.ALL, JsonAutoDetect.Visibility.AN
97         objectMapper.enableDefaultTyping(ObjectMapper.DefaultTyping.NON_FINAL);
98         serializer.setObjectMapper(objectMapper);
99 
100        return serializer;
101    }
102 
103    /**
104     * RedisTemplate
105     */
106    @Bean
107    public RedisTemplate<String, Object> redisTemplate(JedisConnectionFactory factory
108        RedisTemplate<String, Object> redisTemplate = new RedisTemplate<>();
109        redisTemplate.setConnectionFactory(factory);
110 
111        //字符串方式序列化KEY
112        StringRedisSerializer stringRedisSerializer = new StringRedisSerializer();
113        redisTemplate.setKeySerializer(stringRedisSerializer);
114        redisTemplate.setHashKeySerializer(stringRedisSerializer);
115 
116        //JSON方式序列化VALUE
117        redisTemplate.setValueSerializer(jackson2JsonRedisSerializer);
118        redisTemplate.setHashValueSerializer(jackson2JsonRedisSerializer);
119 
120        redisTemplate.afterPropertiesSet();
121 
122        return redisTemplate;
123    }
124 
125    /**
126     * Redis message receiver
127     */
128    @Bean
129    public Receiver redisMessageReceiver(){
130        return new Receiver();
131    }
132 
133    /**
134     * 消息监听器
135     */
136    @Bean
137    MessageListenerAdapter messageListenerAdapter(Receiver receiver, Jackson2JsonRedi
138        //消息接收者以及对应的默认处理方法
139        MessageListenerAdapter messageListenerAdapter = new MessageListenerAdapter(re
140        //消息的反序列化方式
141        messageListenerAdapter.setSerializer(jackson2JsonRedisSerializer);
142 
143        return messageListenerAdapter;
```

```
144     }
145
146     /**
147      * message listener container
148      */
149     @Bean
150     RedisMessageListenerContainer container(RedisConnectionFactory connectionFactory
151             , MessageListenerAdapter messageListenerAdapter){
152         RedisMessageListenerContainer container = new RedisMessageListenerContainer()
153         container.setConnectionFactory(connectionFactory);
154         //添加消息监听器
155         container.addMessageListener(messageListenerAdapter, new PatternTopic("topic-
156
157         return container;
158     }
159
160 }
```

从上面代码可以看出，这里添加了一个消息监听器，监听的Topic是“topic-test”，对应的消息处理就是我们上面定义的cn.zifangsky.model.Receiver类。

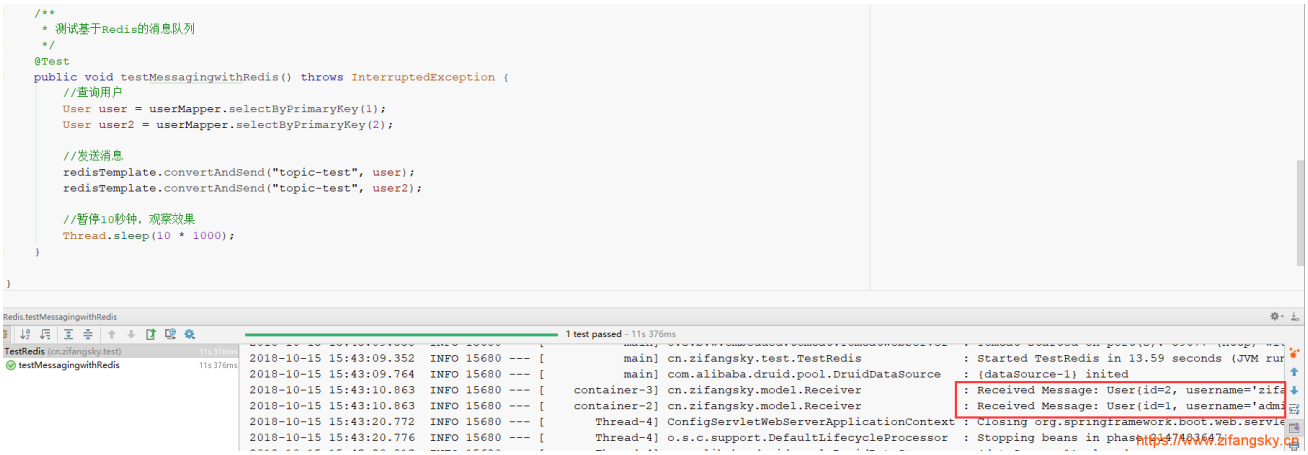
注：上面代码中使用的参数如下：

```
1  spring:
2    #redis
3    redis:
4        cluster:
5          nodes: namenode22:6379,datanode23:6379,datanode24:6379
6          max-redirects: 6
7        timeout: 300000
8        jedis:
9          pool:
10           max-active: 8
11           max-wait: 100000
12           max-idle: 8
13           min-idle: 0
```

（4）测试“发送/接收”消息：

```java
package cn.zifangsky.test;

import cn.zifangsky.mapper.UserMapper;
import cn.zifangsky.model.User;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.test.context.junit4.SpringRunner;

/**
 * 测试Redis的基本操作
 *
 * @author zifangsky
 * @date 2018/7/30
 * @since 1.0.0
 */
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
public class TestRedis {
    @Autowired
    private UserMapper userMapper;

    @Autowired
    private RedisTemplate<String, Object> redisTemplate;

    /**
     * 测试基于Redis的消息队列
     */
    @Test
    public void testMessagingwithRedis() throws InterruptedException {
        //查询用户
        User user = userMapper.selectByPrimaryKey(1);
        User user2 = userMapper.selectByPrimaryKey(2);

        //发送消息
        redisTemplate.convertAndSend("topic-test", user);
        redisTemplate.convertAndSend("topic-test", user2);

        //暂停10秒钟，观察效果
        Thread.sleep(10 * 1000);
    }

}
```

运行测试用例后，效果如下：



参考：

- https://spring.io/guides/gs/messaging-redis/ (https://spring.io/guides/gs/messaging-redis/)

赞 (0)

#Spring Boot (https://www.zifangsky.cn/tag/spring-boot)
#Spring Data Redis (https://www.zifangsky.cn/tag/spring-data-redis)
#消息队列 (https://www.zifangsky.cn/tag/%e6%b6%88%e6%81%af%e9%98%9f%e5%88%97)

上一篇 (https://www.zifangsky.cn/1345.html)
下一篇 (https://www.zifangsky.cn/1355.html)

你可能也喜欢：

- 如何在普通Spring项目中手动实现类似Spring Boot中有条件生成Bean? (https://www.zifangsky.cn/1416.html)
- 基于Spring的项目中Redis存储对象使用Jackson序列化方式 (https://www.zifangsky.cn/1366.html)
- Spring Boot中使用WebSocket总结（三）：使用消息队列实现分布式WebSocket (https://www.zifangsky.cn/1364.html)
- Spring Boot中使用WebSocket总结（二）：向指定用户发送WebSocket消息并处理对方不在线的情况 (https://www.zifangsky.cn/1359.html)
- Spring Boot中使用WebSocket总结（一）：几种实现方式详解 (https://www.zifangsky.cn/1355.html)

发表评论

来都来了，何不留个足迹~

昵称 *

邮箱 *

网址

验证码 *

发表评论

友情链接

iceH's Blog (http://www.secice.cn) 业余草 (http://www.xttblog.com/) 仲威的博客 (https://www.blogme.top)
俄罗斯方块 (https://sale.hacker.bid/) 六阿哥博客 (https://blog.6ag.cn)
太空船博客 (https://www.boatsky.com/) 掘金专栏 (https://juejin.im/user/5819f202d203090055df470e)
朴实的追梦者 (http://www.zmzblog.com) 青木（简书） (https://www.jianshu.com/u/cedd62e70951)