芋道源码 2022-04-03

消息队列（Redis）

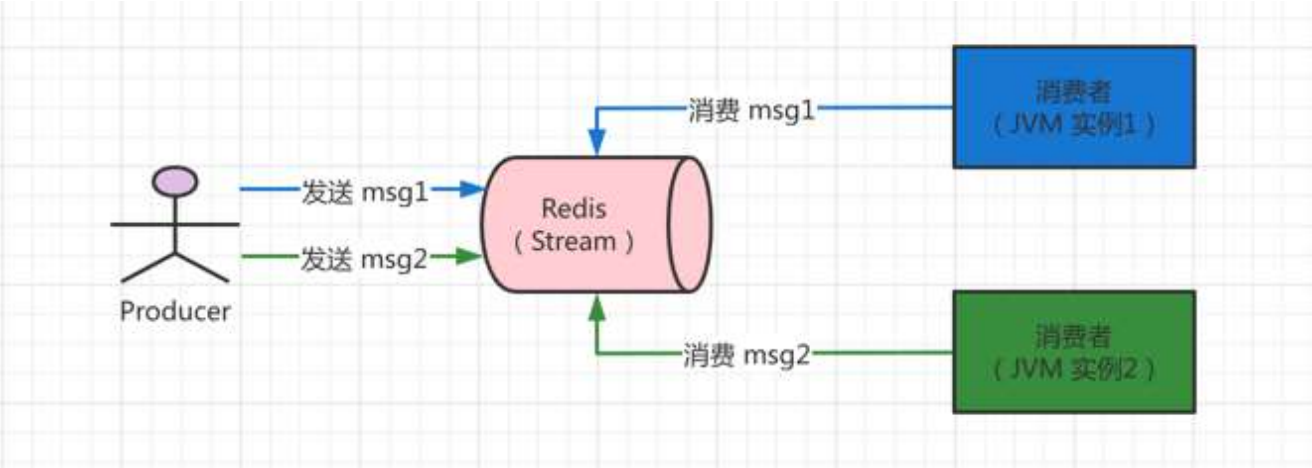
`yudao-spring-boot-starter-mq` 技术组件，基于 Redis 实现分布式消息队列：

- 使用 Stream 特性，提供【集群】消费的能力。
- 使用 Pub/Sub 特性，提供【广播】消费的能力。

> **疑问：什么是【广播】消费？什么是【集群】消费？**
> 参见《阿里云 —— 集群消费和广播消费 》文档

1. 集群消费

集群消费，是指消息发送到 Redis 时，有且只会被一个消费者（应用 JVM 实例）收到，然后消费成功。如下图所示：



> **友情提示：**
> 如果你需要使用到【集群】消费，必须使用 Redis 5.0.0 以上版本，因为 Stream 特性是在该版本之后才引入噢！

1.1 使用场景

集群消费在项目中的使用场景，主要是提供可靠的、可堆积的异步任务的能力。例如说：

- 短信模块，使用它异步发送短信。
- 邮件模块，使用它异步发送邮件。

相比《开发指南 —— 异步任务》来说，Spring Async 在 JVM 实例重启时，会导致未执行完的任务丢失。而集群消费，因为消息是存储在 Redis 中，所以不会存在该问题。

1.2 实现源码

集群消费基于 Redis Stream 实现：

- 实现 AbstractRedisStreamMessage 抽象类，定义【集群】消息。
- 使用 RedisMQTemplate 的 `#send(message)` 方法，发送消息。
- 实现 AbstractRedisStreamMessageListener 接口，消费消息。

最终使用 YudaoRedisMQAutoConfiguration 配置类，扫描所有的 AbstractRedisStreamMessageListener 监听器，初始化对应的消费者。如下图所示：

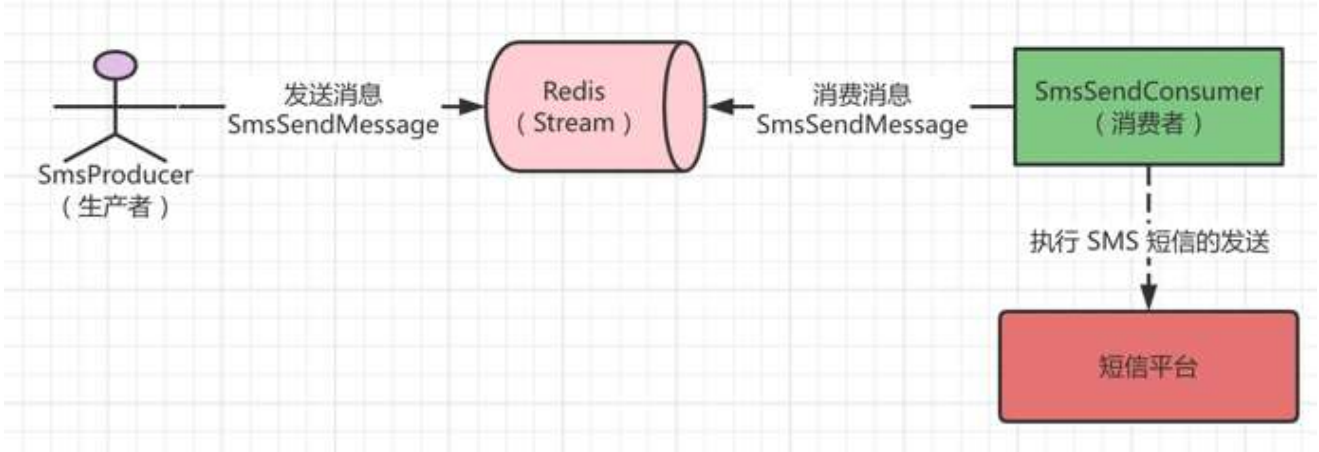
```java
    }

    /** 创建 Redis Stream 集群消费的容器 ...*/
    @Bean(initMethod = "start", destroyMethod = "stop")  启动
    public StreamMessageListenerContainer<String, ObjectRecord<String, String>> redisStreamMessageListenerContainer(
            RedisMQTemplate redisMQTemplate, List<AbstractStreamMessageListener<?>> listeners) {
        RedisTemplate<String, ?> redisTemplate = redisMQTemplate.getRedisTemplate();
        checkRedisVersion(redisTemplate);
        // 第一步，创建 StreamMessageListenerContainer 容器
        // 创建 options 配置
        StreamMessageListenerContainer.StreamMessageListenerContainerOptions<String, ObjectRecord<String, String>> containerOptions =
                StreamMessageListenerContainer.StreamMessageListenerContainerOptions.builder()
                        .batchSize(10) // 一次性最多拉取多少条消息
                        .targetType(String.class) // 目标类型。统一使用 String，通过自己封装的 AbstractStreamMessageListener 去反序列化
                        .build();
        // 创建 container 对象
        StreamMessageListenerContainer<String, ObjectRecord<String, String>> container =
                DefaultStreamMessageListenerContainerX.create(redisMQTemplate.getRedisTemplate().getRequiredConnectionFactory(), containerOptions);

        // 第二步，注册监听器，消费对应的 Stream 主题
        String consumerName = buildConsumerName();
        listeners.parallelStream().forEach(listener -> {
            // 创建 listener 对应的消费者分组
            try {
                redisTemplate.opsForStream().createGroup(listener.getStreamKey(), listener.getGroup());
            } catch (Exception ignore) {}
            // 设置 listener 对应的 redisTemplate
            listener.setRedisMQTemplate(redisMQTemplate);
            // 创建 Consumer 对象
            Consumer consumer = Consumer.from(listener.getGroup(), consumerName);
            // 设置 Consumer 消费进度，以最小消费进度为准
            StreamOffset<String> streamOffset = StreamOffset.create(listener.getStreamKey(), ReadOffset.lastConsumed());
            // 设置 Consumer 监听
            StreamMessageListenerContainer.StreamReadRequestBuilder<String> builder = StreamMessageListenerContainer.StreamReadRequest
                    .builder(streamOffset).consumer(consumer)
                    .autoAcknowledge(false) // 不自动 ack
                    .cancelOnError(throwable -> false); // 默认配置，发生异常就取消消费，显然不符合预期；因此，我们设置为 false
            container.register(builder.build(), listener);  注册
        });
        return container;
    }
```

1.3 实战案例

以【短信发送】举例子，改造使用 Redis 作为消息队列，同时也是讲解集群消费的使用。如下图所示：



1.3.0 引入依赖

在 `yudao-module-system-biz` 模块中，引入 `yudao-spring-boot-starter-mq` 技术组件。如下所示：

```
<dependency>
    <groupId>cn.iocoder.boot</groupId>
    <artifactId>yudao-spring-boot-starter-mq</artifactId>
</dependency>
```

1.3.1 Message 消息

在 `message` 包下，修改 SmsSendMessage 类，短信发送消息。代码如下：

```
@Data
public class SmsSendMessage extends AbstractRedisStreamMessage { // 重点：需要继承

    /**
     * 短信日志编号
     */
    @NotNull(message = "短信日志编号不能为空")
    private Long logId;
    /**
     * 手机号
     */
    @NotNull(message = "手机号不能为空")
    private String mobile;
    /**
     * 短信渠道编号
     */
    @NotNull(message = "短信渠道编号不能为空")
    private Long channelId;
    /**
     * 短信 API 的模板编号
     */
    @NotNull(message = "短信 API 的模板编号不能为空")
    private String apiTemplateId;
    /**
     * 短信模板参数
     */
    private List<KeyValue<String, Object>> templateParams;

}
```

1.3.2 SmsProducer 生产者

在 `producer` 包下，修改 SmsProducer 类，Sms 短信相关消息的生产者。代码如下：

```
@Slf4j
@Component
public class SmsProducer {

    @Resource
    private RedisMQTemplate redisMQTemplate; // 重点：注入 RedisMQTemplate 对象

    /**
     * 发送 {@link SmsSendMessage} 消息
     *
     * @param logId 短信日志编号
     * @param mobile 手机号
     * @param channelId 渠道编号
     * @param apiTemplateId 短信模板编号
     * @param templateParams 短信模板参数
     */
    public void sendSmsSendMessage(Long logId, String mobile,
                                   Long channelId, String apiTemplateId, List<Ke
        SmsSendMessage message = new SmsSendMessage().setLogId(logId).setMobile(
        message.setChannelId(channelId).setApiTemplateId(apiTemplateId).setTempl
        redisMQTemplate.send(message); // 重点：使用 RedisMQTemplate 发送消息
    }

}
```

1.3.3 SmsSendConsumer 消费者

在 `consumer` 包下，修改 SmsSendConsumer 类，SmsSendMessage 的消费者。代码如下：

```
@Component
@Slf4j
public class SmsSendConsumer extends AbstractRedisStreamMessageListener<SmsSendM

    @Resource
    private SmsSendService smsSendService;

    @Override // 重点：实现 onMessage 方法
```

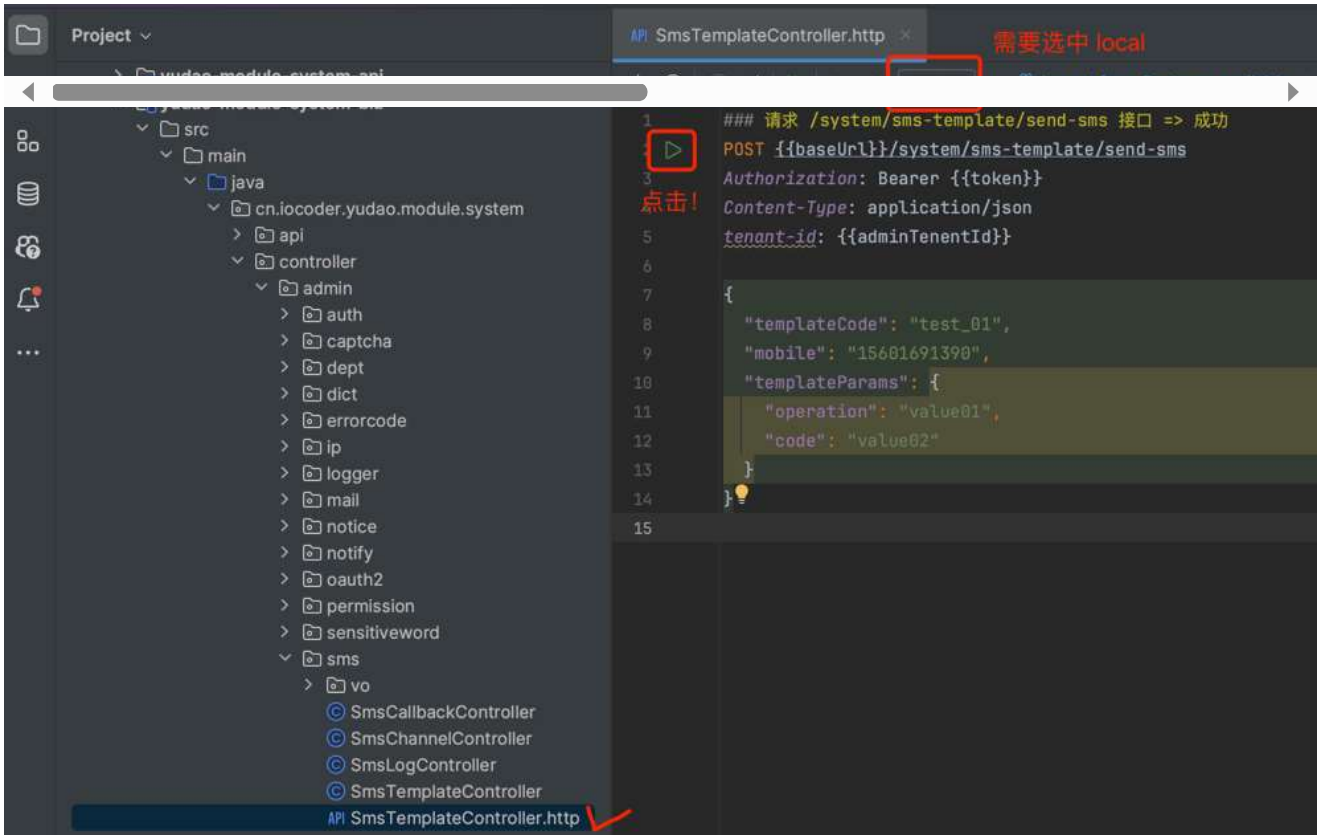
```
        public void onMessage(SmsSendMessage message) {
            log.info("[onMessage][消息内容({})]", message);
            smsSendService.doSendSms(message);
        }

    }
```

1.3.4 简单测试

① Debug 启动后端项目，可以在 SmsProducer 和 SmsSendConsumer 上面打上断点，稍微调试下。

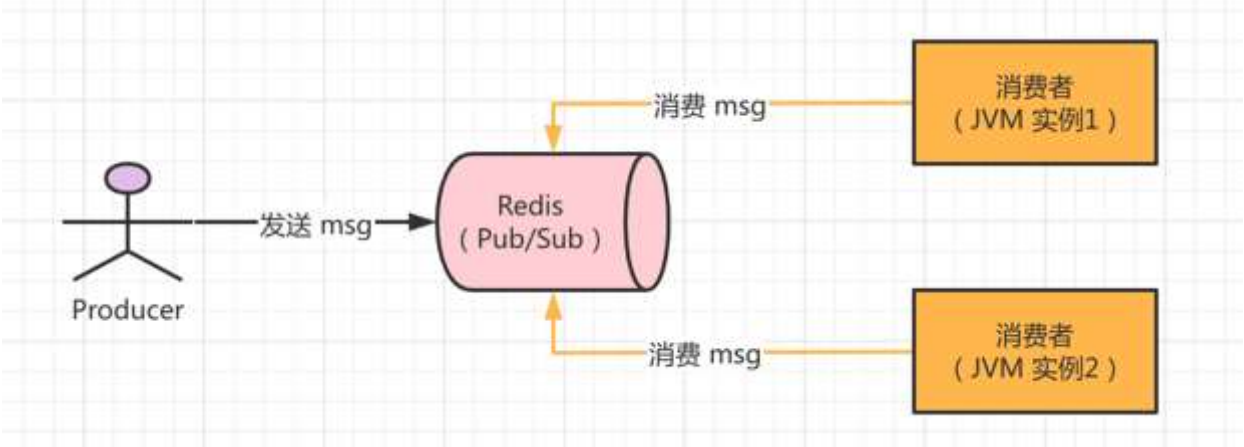
② 打开 `SmsTemplateController.http` 文件，使用 IDEA httpclient 发起请求，发送短信。如下图所示：



如果 IDEA 控制台看到 `[onMessage][消息内容` 日志内容，说明消息的发送和消费成功。

2. 广播消费

广播消费，是指消息发送到 Redis 时，所有消费者（应用 JVM 实例）收到，然后消费成功。如下图所示：



2.1 使用场景

例如说，在应用中，缓存了数据字典等配置表在内存中，可以通过 Redis 广播消费，实现每个应用节点都消费消息，刷新本地内存的缓存。

又例如说，我们基于 WebSocket 实现了 IM 聊天，在我们给用户主动发送消息时，因为我们不知道用户连接的是哪个提供 WebSocket 的应用，所以可以通过 Redis 广播消费。每个应用判断当前用户是否是和自己提供的 WebSocket 服务连接，如果是，则推送消息给用户。

2.2 实现源码

广播消费基于 Redis Pub/Sub 实现：

- 实现 AbstractChannelMessage 抽象类，定义【广播】消息。
- 使用 RedisMQTemplate 的 `#send(message)` 方法，发送消息。
- 实现 AbstractRedisChannelMessageListener 接口，消费消息。

最终使用 YudaoRedisMQAutoConfiguration 配置类，扫描所有的 AbstractRedisChannelMessageListener 监听器，初始化对应的消费者。如下图所示：

YudaoMQAutoConfiguration.java

```
51      // ========== 消费者相关 ==========
52
53      /** 创建 Redis Pub/Sub 广播消费的容器 */
56      @Bean
57      public RedisMessageListenerContainer redisMessageListenerContainer(
58              RedisMQTemplate redisMQTemplate, List<AbstractChannelMessageListener<?>> listeners) {
59          // 创建 RedisMessageListenerContainer 对象
60          RedisMessageListenerContainer container = new RedisMessageListenerContainer();
61          // 设置 RedisConnection 工厂。
62          container.setConnectionFactory(redisMQTemplate.getRedisTemplate().getRequiredConnectionFactory());
63          // 添加监听器
64          listeners.forEach(listener -> {
65              listener.setRedisMQTemplate(redisMQTemplate);
66              container.addMessageListener(listener, new ChannelTopic(listener.getChannel()));
67              log.info("[redisMessageListenerContainer][注册 Channel({}) 对应的监听器({})]",
68                      listener.getChannel(), listener.getClass().getName());
69          });
70          return container;
71      }
```

2.3 实战案例

参见 《开发指南 —— 本地缓存》

← 消息队列（内存）

消息队列（RocketMQ）→