芋道源码 2023-11-02

消息队列（RocketMQ）

RocketMQ-Spring

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

如果你对 RocketMQ 不太了解，可以看看 《芋道 Spring Boot 消息队列 RocketMQ 入门》 文档。

> **如何安装一个 RocketMQ 服务?**
>
> 参考 《芋道 RocketMQ 极简入门 》 文档。

2. 使用示例

以【短信发送】举例子，改造使用 RocketMQ 作为消息队列。

2.0 引入依赖与配置

① 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>
```

② 修改 `yudao-spring-boot-starter-mq` 的 `pom.xml` 文件，引入 `rocketmq-spring-boot-starter` 依赖。如下所示：

```
<!-- 实际只要删除 <optional>true</optional> 部分即可 -->
<dependency>
    <groupId>org.apache.rocketmq</groupId>
    <artifactId>rocketmq-spring-boot-starter</artifactId>
</dependency>
```

记得需要手动在 IDEA 刷新下 Maven 依赖。

③ 修改 `application.xml` 配置文件，添加 RocketMQ 全局配置。如下所示：

```
# rocketmq 配置项，对应 RocketMQProperties 配置类
rocketmq:
  # Producer 配置项
  producer:
    group: ${spring.application.name}_PRODUCER # 生产者分组
```

ps：默认已经添加，无需操作。

④ 修改 `application-local.xml` 配置文件，添加 RocketMQ `name-server` 配置。如下所示：

```
# rocketmq 配置项，对应 RocketMQProperties 配置类
rocketmq:
  name-server: 127.0.0.1:9876 # RocketMQ Namesrv
```

ps：默认已经添加，无需操作。

2.1 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 {

    public static final String TOPIC = "SMS_SEND_TOPIC"; // 重点：需要增加消息对应

    /**
     * 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;

}
```

2.2 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 RocketMQTemplate rocketMQTemplate; // 重点：注入 RocketMQTemplate 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
        rocketMQTemplate.syncSend(SmsSendMessage.TOPIC, message); // 重点：使用 R
    }

}
```

2.3 SmsSendConsumer 消费者

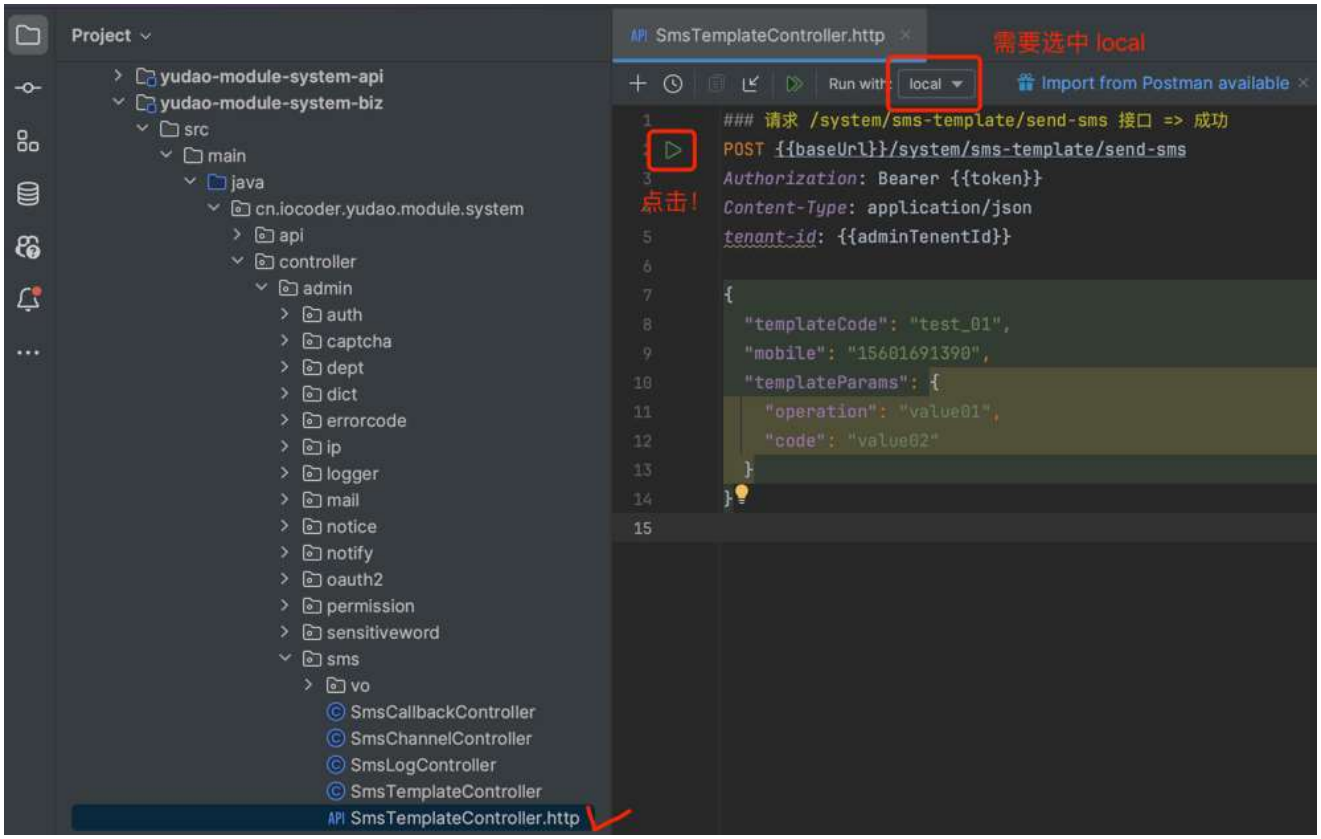
在 `consumer` 包下，修改 SmsSendConsumer 类，SmsSendMessage 的消费者。代码如下：

```
@Component
@RocketMQMessageListener( // 重点：添加 @RocketMQMessageListener 注解，声明消费的 t
        topic = SmsSendMessage.TOPIC,
        consumerGroup = SmsSendMessage.TOPIC + "_CONSUMER"
)
@Slf4j
public class SmsSendConsumer implements RocketMQListener<SmsSendMessage> { // 重

    @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);
    }

}
```

2.4 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。

← 消息队列（Redis） 消息队列（RabbitMQ） →