# 消息队列 | Apache kafka

Kafka 是一种由 Scala 和 java 编写高吞吐量的分布式发布订阅消息系统，它可以处理消费者在网站中的所有动作流数据。kafka 强依赖 zookeeper。

## 代码实现

在进行消息发送与接收之前，我们需要先下载并启动 kafka 服务。

### 消息发送

引入依赖

XML

```xml
<dependency>
    <groupId>org.springframework.kafka</groupId>
    <artifactId>spring-kafka</artifactId>
    <version>2.2.6.RELEASE</version>
</dependency>
```

配置 kafka 地址

YAML

```yaml
spring:
  kafka:
    bootstrap-servers: localhost:9092
```

kafka 主题配置

Java

```java
@Configuration
public class KafkaConfig {

    /**
     * 创建消息主题：消费者监听XX主题XX分区
     * 名为order、8个分区、2个分区副本
     */
    @Bean
    public NewTopic createTopic() {
        return new NewTopic("order",8, (short) 2 );
    }

    /**
     * 修改消息主题
     * 1.修改配置、重启项目就能实现分区数修改
     * 2.修改分区后数据会保留，因此分区数量只能增大不能减小
     * 增加到10个分区
     */
    @Bean
    public NewTopic updateTopic() {
        return new NewTopic("order",10, (short) 2 );
    }

}
```

消息发送

Java

```java
@Service
public class KafKaService {

    @Resource
    private KafkaTemplate<String,Object> kafkaTemplate;

    @Resource
    private KafkaCallback kafkaCallback;

    /**
     * 消息发送：不声明事务,保错时已发送!
     * @param topic 主题
     * @param message 消息
     */
    public void sendMessage(String topic, String message) {
        kafkaTemplate.send(topic,message).addCallback(kafkaCallback);
        throw new RuntimeException("消息发送异常!");
    }

    /**
     * 消息发送：声明事务，报错时不发送
     * @param topic 主题
     * @param message 消息
     */
    public void sendMessageTran(String topic, String message) {
        kafkaTemplate.executeInTransaction(operations ->{
            operations.send(topic,message);
            throw new RuntimeException("消息发送异常!");
        });
    }
}
```

消息发送回调

Java

```java
@Slf4j
@Component
public class KafkaCallback implements ListenableFutureCallback<SendResult<String, Object>> {

    @Autowired
    private KafKaService kafKaService;

    @Autowired
    private MessageRecordService messageRecordService;
    /**
     * 失败回调
     * @param throwable 异常
     */
    @Override
    public void onFailure(Throwable throwable) {
        log.error("消息发送失败!" + throwable.getMessage());

    }


    /**
     * 成功回调
     * @param result 成功信息
     */
    @Override//成功回调
    public void onSuccess(SendResult<String, Object> result) {
        assert result != null;
        log.info("消息发送成功!" + result.getRecordMetadata().topic());

    }
}
```

## 消息接收

引入依赖

XML

```xml
<dependency>
    <groupId>org.springframework.kafka</groupId>
    <artifactId>spring-kafka</artifactId>
```

```
    <version>2.2.6.RELEASE</version>
</dependency>
```

配置kafka地址

YAML

```yaml
spring:
  kafka:
    bootstrap-servers: localhost:9092
```

消息监听

Java

```java
@Component
public class KafKaListener {

    @KafkaListener(id = "consumer",topics = "order")
    public void getMessage(ConsumerRecord<?,?> record){
        System.out.println("接收到消息："+record.value());
    }
}
```

## 整合到项目

数据库中新建一张用于存放消息的数据表，业务进行时会产生消息并存放在数据表中，通过定时任务来定时获取数据库中的消息并通过kafka发送给消费者。详情

包装消息发送服务：

Java

```java
@Service
public class MessageRecordServiceImpl implements MessageRecordService {

    @Resource
    private MessageRecordMapper messageRecordMapper;

    @Autowired
    private KafKaService kafKaService;

    @Override
    public boolean commit(String messageId, boolean isCommit) {
        //不提交、回滚
        if (!isCommit){
            messageRecordMapper.deleteByMessageId(messageId);
            return true;
        }
        //获得消息并发送到MQ
        MessageRecord messageRecord = messageRecordMapper.selectByMessageId(messageId);
        kafKaService.sendMessage(KafkaName.TOPIC,  JSON.toJSONString(messageRecord));
        return true;
    }

    @Override
    public void saveMessageRecord(MessageRecord messageRecord) {
        messageRecordMapper.saveMessageRecord(messageRecord);
    }


    @Override
    public MessageRecord findMessageRecord(String messageId) {
        return messageRecordMapper.selectByMessageId(messageId);
    }

    @Override
    public void updateMessageSendSuccess(String messageId) {
        messageRecordMapper.updateMessageSendSuccess(messageId);
    }

    @Override
    public List<MessageRecord> findAll(int status) {
        return messageRecordMapper.selectAllMessageRecord(status);
    }

    @Override
    public void deleteSuccessMessage() {
        messageRecordMapper.deleteByStatus(1);
```

```
    }


}
```

消息实体：

```java
public interface MessageRecordService {
    /**
     * 保存消息记录
     * @param messageRecord 消息记录
     * @return 受影响行数
     */
    void saveMessageRecord(MessageRecord messageRecord);

    /**
     * 提交消息
     * @param messageId 消息id
     * @param isCommit 是否进行提交，为false回滚
     * @return 操作是否成功
     */
    boolean commit(String messageId, boolean isCommit);

    /**
     * 根据id查询消息
     * @param messageId 消息id
     * @return 消息对象
     */
    MessageRecord findMessageRecord(String messageId);

    /**
     * 设置消息发送状态为成功
     * @param messageId 消息id
     */
    void updateMessageSendSuccess(String messageId);

    /**
     * 查询消息列表
     * @return 消息列表
     */
    List<MessageRecord> findAll(int status);

    /**
     * 删除已发送成功的消息
     */
    void deleteSuccessMessage();
}
```

消息 mapper

```java
@Mapper
public interface MessageRecordMapper {

    /**
     * 删除消息
     * @param messageId 消息ID
     */
    @Delete("DELETE FROM message_record WHERE message_id=#{messageId}")
    void deleteByMessageId(String messageId);

    /**
     * 查询消息
     * @param messageId 消息ID
     */
    @Select("SELECT *FROM message_record WHERE message_id=#{messageId}")
    @Results(id = "messageRecord",value = {
            @Result(id = true,column = "id",property = "id"),
            @Result(column = "business_id",property = "businessId"),
            @Result(column = "business_type",property = "businessType"),
            @Result(column = "message_id",property = "messageId"),
            @Result(column = "retries_number",property = "retriesNumber"),
            @Result(column = "status",property = "status"),
            @Result(column = "create_time",property = "createTime")
    })
    MessageRecord selectByMessageId(String messageId);


```

```java
    /**
     * 设置消息发送成功
     * @param messageId 消息ID
     */
    @Update("UPDATE message_record SET status=1 WHERE message_id=#{messageId}")
    void updateMessageSendSuccess(String messageId);


    /**
     * 查询消息列表
     * @return 消息列表
     */
    @ResultMap("messageRecord")
    @Select("SELECT *FROM message_record WHERE status=#{status}")
    List<MessageRecord> selectAllMessageRecord(int status);

    /**
     * 添加消息到数据库
     * @param messageRecord 消息
     * @return 影响行数
     */
    @Insert("INSERT INTO message_record(business_id,business_type,message_id,retries_number,status,create_time) VALUES(#{businessId},#{
    int saveMessageRecord(MessageRecord messageRecord);

    /**
     * 删除消息
     * @param status 消息状态，1为已发送、0为未发送
     */
    @Delete("DELETE FROM message_record WHERE status=#{status}")
    void deleteByStatus(int status);
}
```

# 服务搭建

启动kafka服务需要先启动zookeeper服务。

## Zookeeper 服务

### 单机部署

```ini
# 下载地址：http://zookeeper.apache.org/releases.html#download
1.下载.tar.gz后缀的文件并解压(windows也下载这个)

2.将conf目录下“zoo_sample.cfg”重命名为“zoo.cfg”，并添加配置：
      # windows下 \\ 或 / 都行
      dataDir=D:\\wen_install\\cloud\\apache-zookeeper-3.6.3
      # zookeeper端口号
      clientPort=2181

3.bin目录在命令行输入 zkServer 即可启动。
```

### Zookeeper 集群

打开zoo.fig文件，加入如下配置（3个主机的信息）：

```ini
# 集群: server.编号=IP:端口:选举端口
server.0=192.168.113.201:9898:9999
server.1=192.168.113.202:9898:9999
server.2=192.168.113.203:9898:9999
```

在每给节点将dataDir指定一个路径

```ini
dataDir=D:\\wen_install\\cloud\\apache-zookeeper-3.6.3\\data
```

每个节点的dataDir目录，创建myid文件（无后缀），并将对应编号写入，如192.168.113.202的内容为：

```ini
1
```

## Kafka 服务

### 安装配置

```SQL
# 下载地址：http://kafka.apache.org/downloads
1.下载.tgz后缀的文件，解压(windows也下载这个)。

2.在主目录创建kafka-logs目录。

3.打开config\server.properties文件，修改log.dirs值为kafka主目录下的kafka-logs目录，并将 \ 换为\ \ 。

4.将server.properties文件的zookeeper.connect改为zookeeper地址：localhost:2181，kafka默认端口为9092。

5.windows启动命令：.\bin\windows\kafka-server-start.bat .\config\server.properties

6.linux启动，命令：bin\kafka-server-start.sh config\server.properties
```

### 手动测试

从主目录打开命令行，然后输入命令：

```Bash
# 创建主题test
.\bin\windows\kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic test

# 查看主题
.\bin\windows\kafka-topics.bat --list --zookeeper localhost:2181

# 创建一个生产者，使用test主题
.\bin\windows\kafka-console-producer.bat --broker-list localhost:9092 --topic test

# 创建一个消费者，监听test主题
.\bin\windows\kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic test --from-beginning
```

注意：生产者创建完成，可以进行键盘输入，此时输入的内容就是发送的消息，可以再开一个消费者窗口接收。（或者退出发送，直接创建消费者就能收到）。

### UI 工具

- kafkaTool： https://www.kafkatool.com/download.html
- Kafka Monitor：https://github.com/Morningstar/kafka-offset-monitor/releases/tag/0.4.6
- Kafka Manager：https://github.com/yahoo/kafka-manager/archive/1.3.3.15.zip
- Kafka Eagle：https://github.com/smartloli/kafka-eagle-bin/archive/v1.4.8.tar.gz