**kafka**

<https://www.imooc.com/video/17859>

卡夫卡原理：<https://www.cnblogs.com/gxyandwmm/p/11432598.html>

linux安装卡夫卡：<https://www.cnblogs.com/toutou/p/linux_install_kafka.html>

zookeeper分布式搭建：<https://blog.csdn.net/HcJsJqJSSM/article/details/85337209>

zookeeper伪分布式：<https://blog.csdn.net/LOVE_LJing/article/details/82314876>

消息队列原理：<https://blog.csdn.net/nsxqf/article/details/80600330>

手动创建topic和代码创建的区别：<https://blog.csdn.net/yuanlong122716/article/details/105160545/>

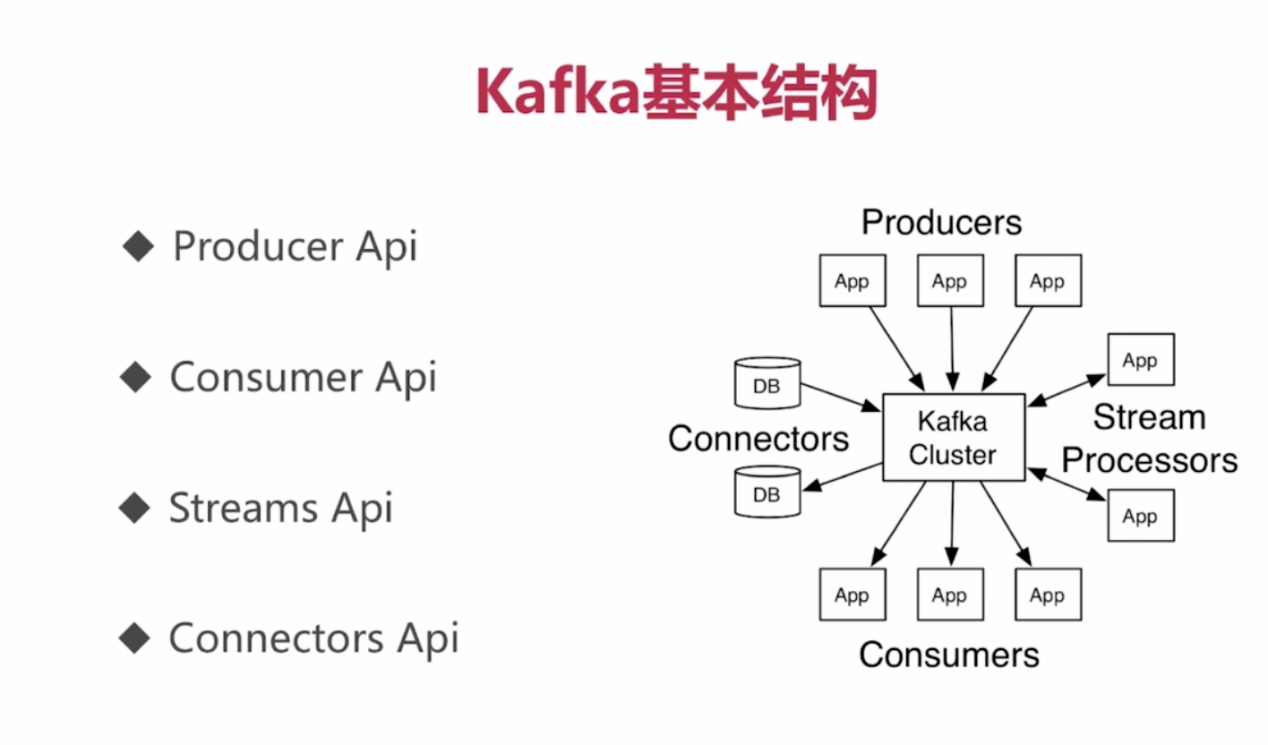
# 基本概念

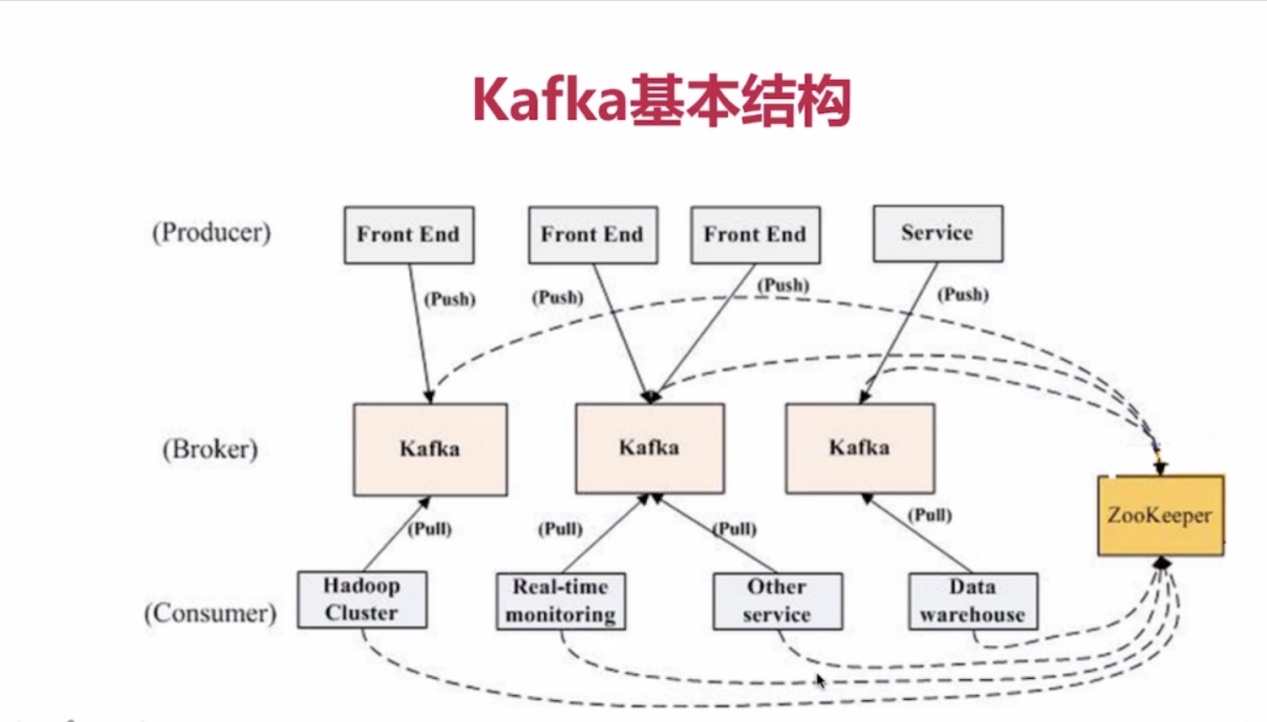






# 基本结构





# kafka特点







# 应用场景





# 安装

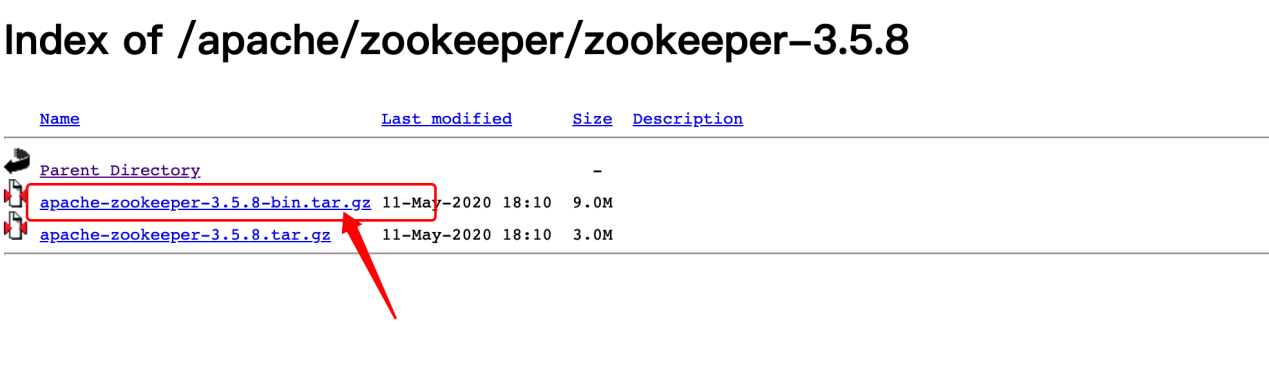


# 单机版zookeeper安装

## 下载tar.gz包

采用镜像下载，快些





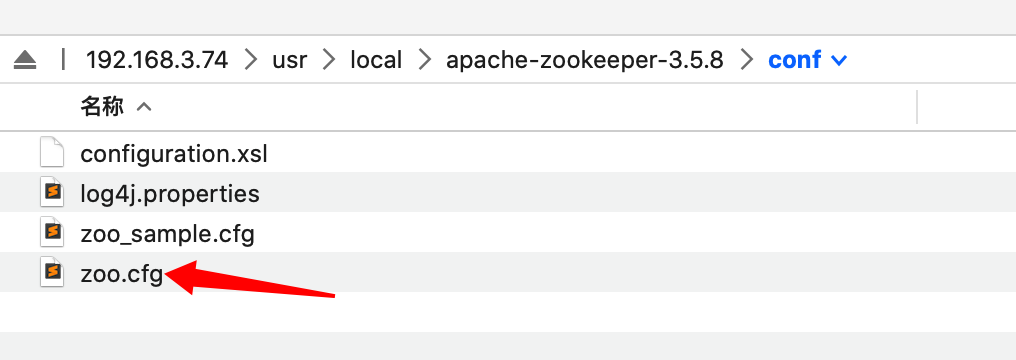
要选择带bin的包，不然运行不了。

## 上传Linux后操作

解压

在文件夹里新建data和log文件夹

到conf文件夹里复制一份配置文件

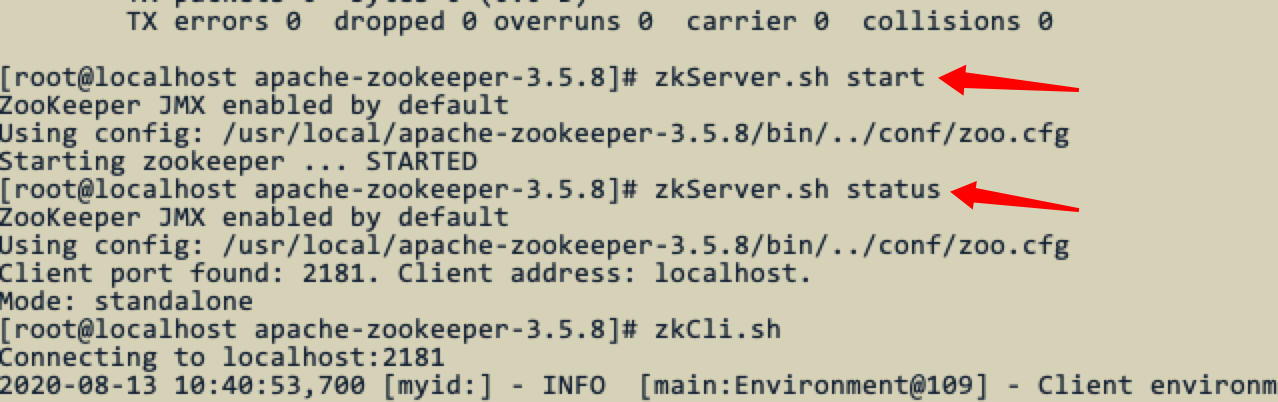


修改里面内容



如图，修改成自己的文件夹

好了，就可以启动了



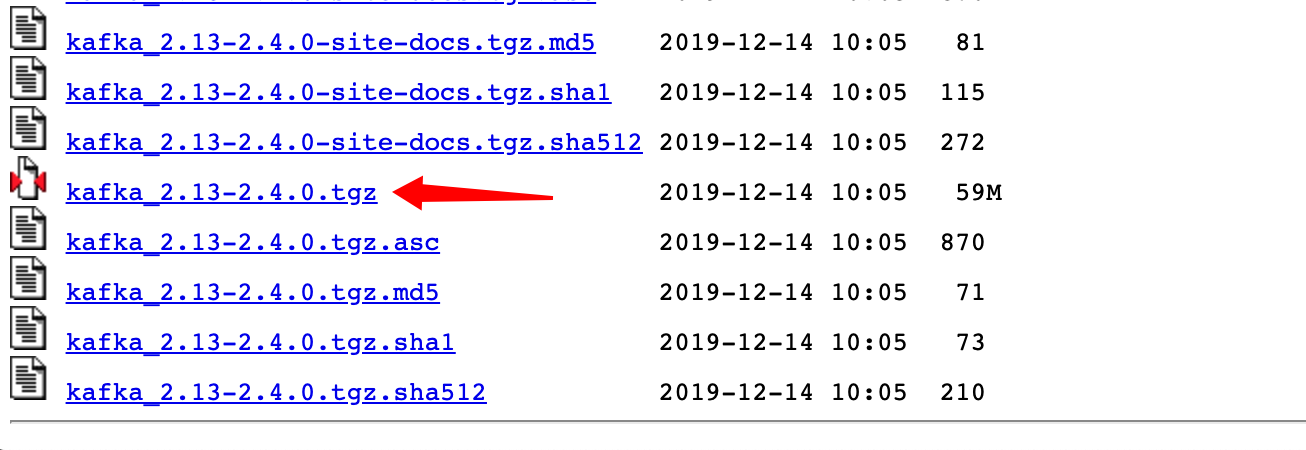
这里是配置了环境变量的，环境变量如下：



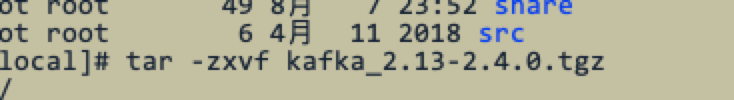
# kafka安装

## 下载

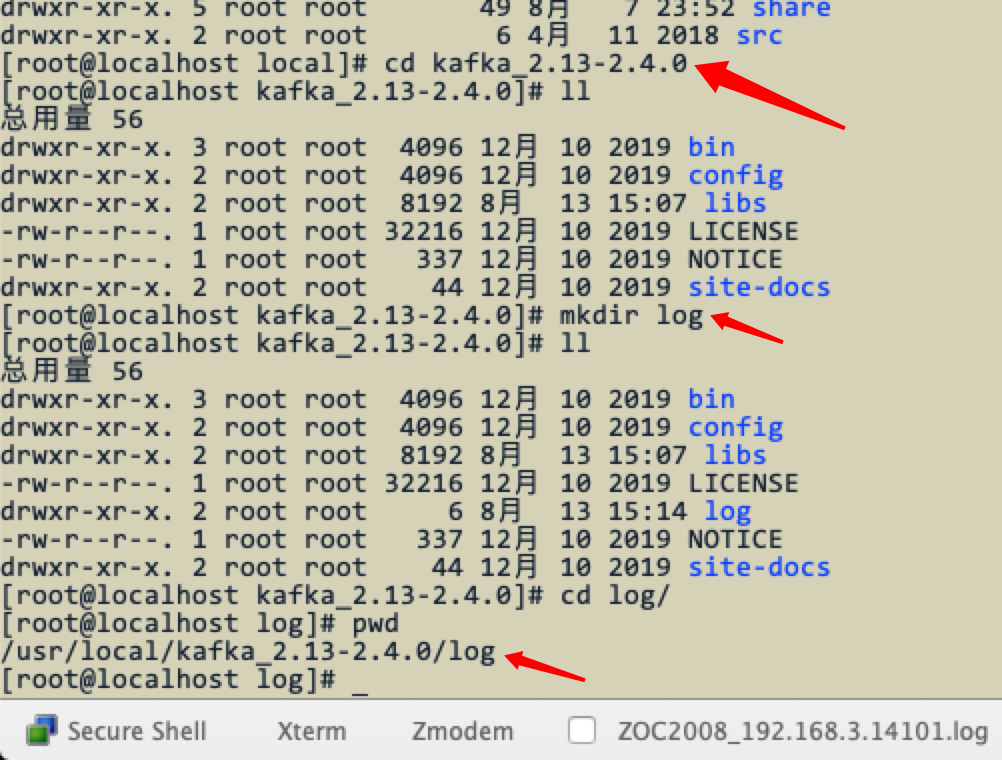




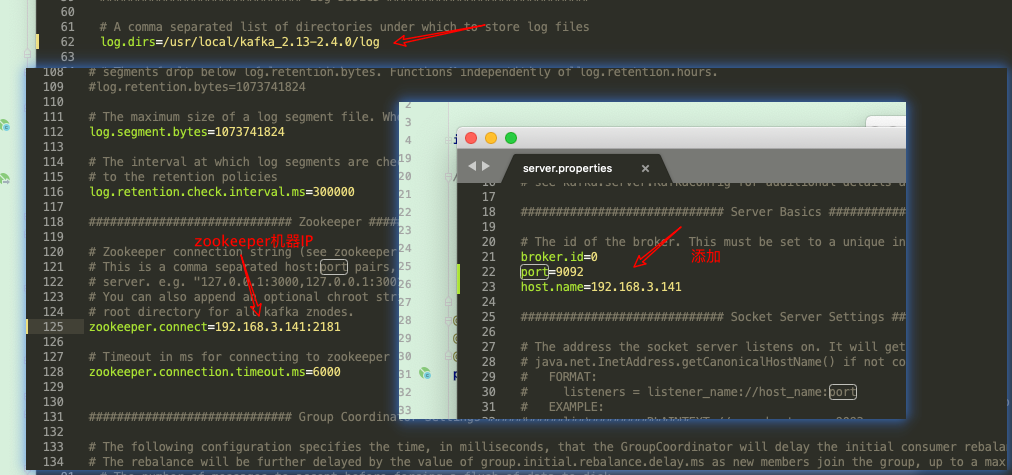
## 上传解压



cd到解压文件夹，建立一个log文件夹



修改配置文件



修改

port和host.name

日志路径

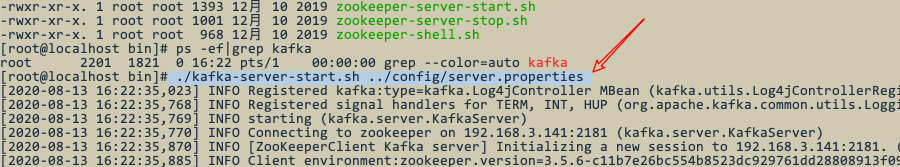
zookeeper连接地址

## 启动

先启动zookeeper。

启动命令：

|  |
| --- |
| ./kafka-server-start.sh ../config/server.properties |



## 测试

## 手动建立topic和代码建立的区别

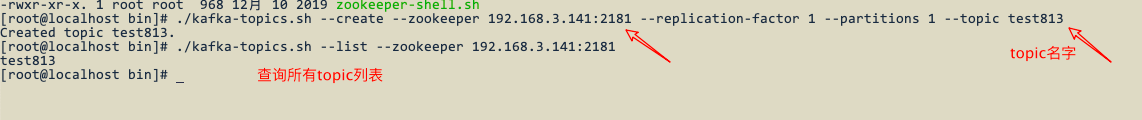


命令：

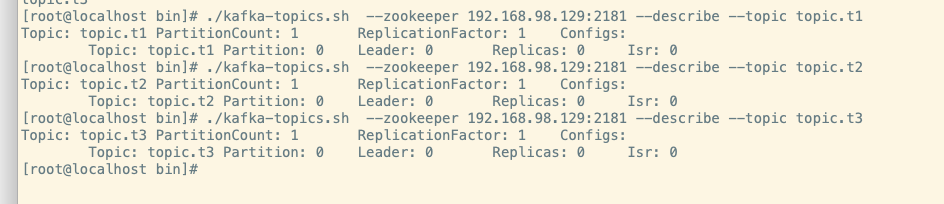


### 新建一个topic

|  |
| --- |
| 命令：在kafka 目录下bin目录下执行：  ./kafka-topics.sh --create --zookeeper 192.168.3.141:2181 --replication-factor 1 --partitions 1 --topic test813 test是topic名字可以随便换哦。  命令：在kafka 目录下bin目录下执行：  ./kafka-topics.sh --list --zookeeper 192.168.3.141:2181 |



### 查看一个topic的分区和副本



|  |
| --- |
| ./kafka-topics.sh --zookeeper 192.168.98.129:2181 --describe --topic topic.t3 |

# springboot集成kafka

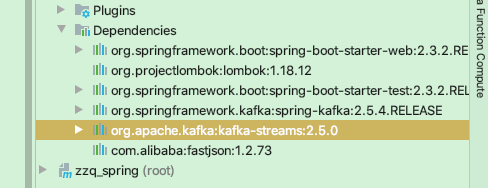
## 启动zookeeper

## 启动kafka

## 新建springboot项目

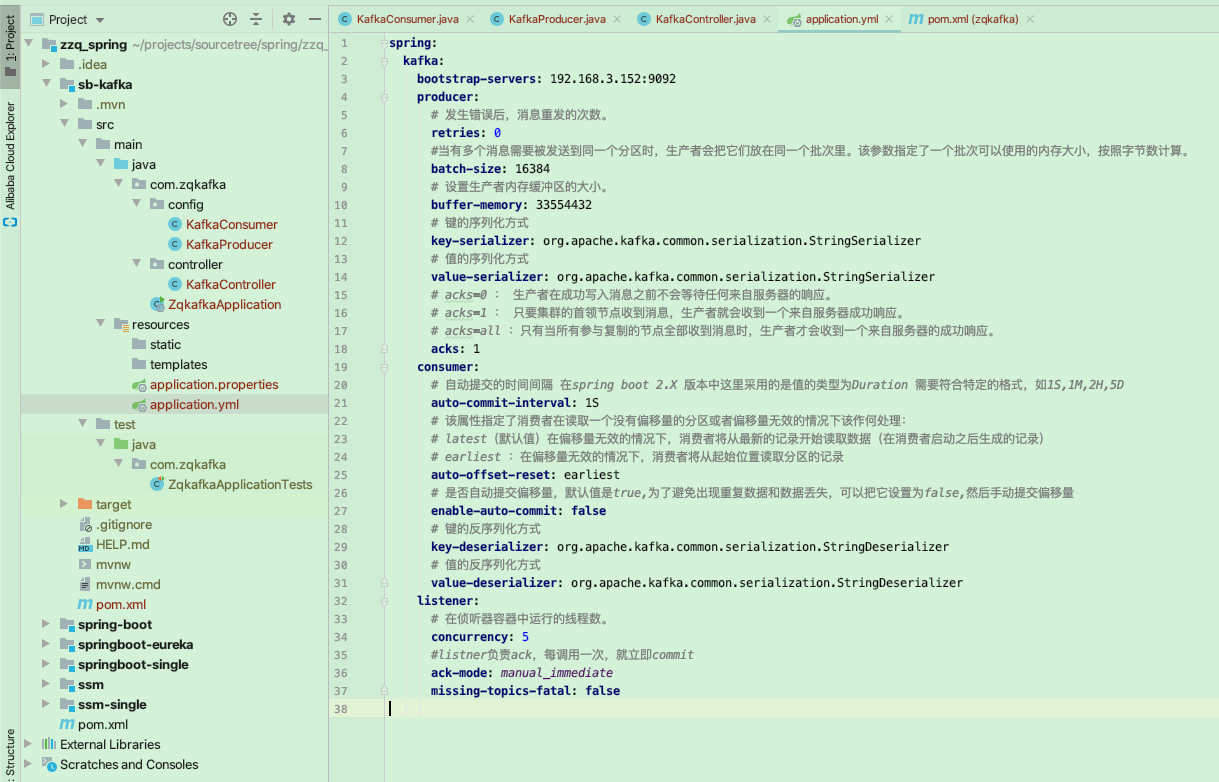
## 添加依赖

|  |
| --- |
| <**parent**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-parent</**artifactId**>  <**version**>2.3.2.RELEASE</**version**>  <**relativePath**/> *<!-- lookup parent from repository -->* </**parent**> |
| <**dependency**>  <**groupId**>org.springframework.kafka</**groupId**>  <**artifactId**>spring-kafka</**artifactId**> </**dependency**> |



## yml配置

|  |
| --- |
| **spring**:  **kafka**:  **bootstrap-servers**: 192.168.3.152:9092  **producer**:  *# 发生错误后，消息重发的次数。* **retries**: 0  *#当有多个消息需要被发送到同一个分区时，生产者会把它们放在同一个批次里。该参数指定了一个批次可以使用的内存大小，按照字节数计算。* **batch-size**: 16384  *# 设置生产者内存缓冲区的大小。* **buffer-memory**: 33554432  *# 键的序列化方式* **key-serializer**: org.apache.kafka.common.serialization.StringSerializer  *# 值的序列化方式* **value-serializer**: org.apache.kafka.common.serialization.StringSerializer  *# acks=0 ： 生产者在成功写入消息之前不会等待任何来自服务器的响应。  # acks=1 ： 只要集群的首领节点收到消息，生产者就会收到一个来自服务器成功响应。  # acks=all ：只有当所有参与复制的节点全部收到消息时，生产者才会收到一个来自服务器的成功响应。* **acks**: 1  **consumer**:  *# 自动提交的时间间隔 在spring boot 2.X 版本中这里采用的是值的类型为Duration 需要符合特定的格式，如1S,1M,2H,5D* **auto-commit-interval**: 1S  *# 该属性指定了消费者在读取一个没有偏移量的分区或者偏移量无效的情况下该作何处理：  # latest（默认值）在偏移量无效的情况下，消费者将从最新的记录开始读取数据（在消费者启动之后生成的记录）  # earliest ：在偏移量无效的情况下，消费者将从起始位置读取分区的记录* **auto-offset-reset**: earliest  *# 是否自动提交偏移量，默认值是true,为了避免出现重复数据和数据丢失，可以把它设置为false,然后手动提交偏移量* **enable-auto-commit**: **false** *# 键的反序列化方式* **key-deserializer**: org.apache.kafka.common.serialization.StringDeserializer  *# 值的反序列化方式* **value-deserializer**: org.apache.kafka.common.serialization.StringDeserializer  **listener**:  *# 在侦听器容器中运行的线程数。* **concurrency**: 5  *#listner负责ack，每调用一次，就立即commit* **ack-mode**: *manual\_immediate* **missing-topics-fatal**: **false** |



## 生产者

|  |
| --- |
| **package** com.zqkafka.config;  **import** com.alibaba.fastjson.JSONObject; **import** lombok.extern.slf4j.Slf4j; **import** org.springframework.beans.factory.annotation.Autowired; **import** org.springframework.kafka.core.KafkaTemplate; **import** org.springframework.kafka.support.SendResult; **import** org.springframework.stereotype.Component; **import** org.springframework.util.concurrent.ListenableFuture; **import** org.springframework.util.concurrent.ListenableFutureCallback;  **import** java.util.UUID;  */\*\*  \** ***@author*** *iszengziqiang@163.com  \** ***@date*** *2020/8/17 18:02  \** ***@Version:*** *1.0  \** ***@desc*** *//****todo*** *\*/* @SuppressWarnings(**"all"**) @Component @Slf4j **public class** KafkaProducer {   @Autowired  **private** KafkaTemplate<String, Object> **kafkaTemplate**;   *//自定义topic* **public static final** String ***TOPIC\_TEST*** = **"topic.t1"**;   *//* **public static final** String ***TOPIC\_GROUP1*** = **"topic.group1"**;   *//* **public static final** String ***TOPIC\_GROUP2*** = **"topic.group2"**;   **public void** send(Object obj) {  String obj2String = JSONObject.*toJSONString*(obj);  ***log***.info(**"准备发送消息为：{}"**, obj2String);  *//发送消息* ListenableFuture<SendResult<String, Object>> future = **kafkaTemplate**.send(***TOPIC\_TEST***, 0, UUID.*randomUUID*().toString(), obj);  future.addCallback(**new** ListenableFutureCallback<SendResult<String, Object>>() {  @Override  **public void** onFailure(Throwable throwable) {  *//发送失败的处理* ***log***.info(***TOPIC\_TEST*** + **" - 生产者 发送消息失败："** + throwable.getMessage());  }   @Override  **public void** onSuccess(SendResult<String, Object> stringObjectSendResult) {  *//成功的处理* ***log***.info(***TOPIC\_TEST*** + **" - 生产者 发送消息成功："** + stringObjectSendResult.toString());   }  });  } } |

备注：如果采用代码创建topic方式，那么partition只有一个，及0号位置，所以在send方法里传0才能保证消费者收到消息。所以最好采用终端，使用客户端方式进行创建topic。

参数中相同的key会放到同一个partition里，保证了同一类key的顺序性。

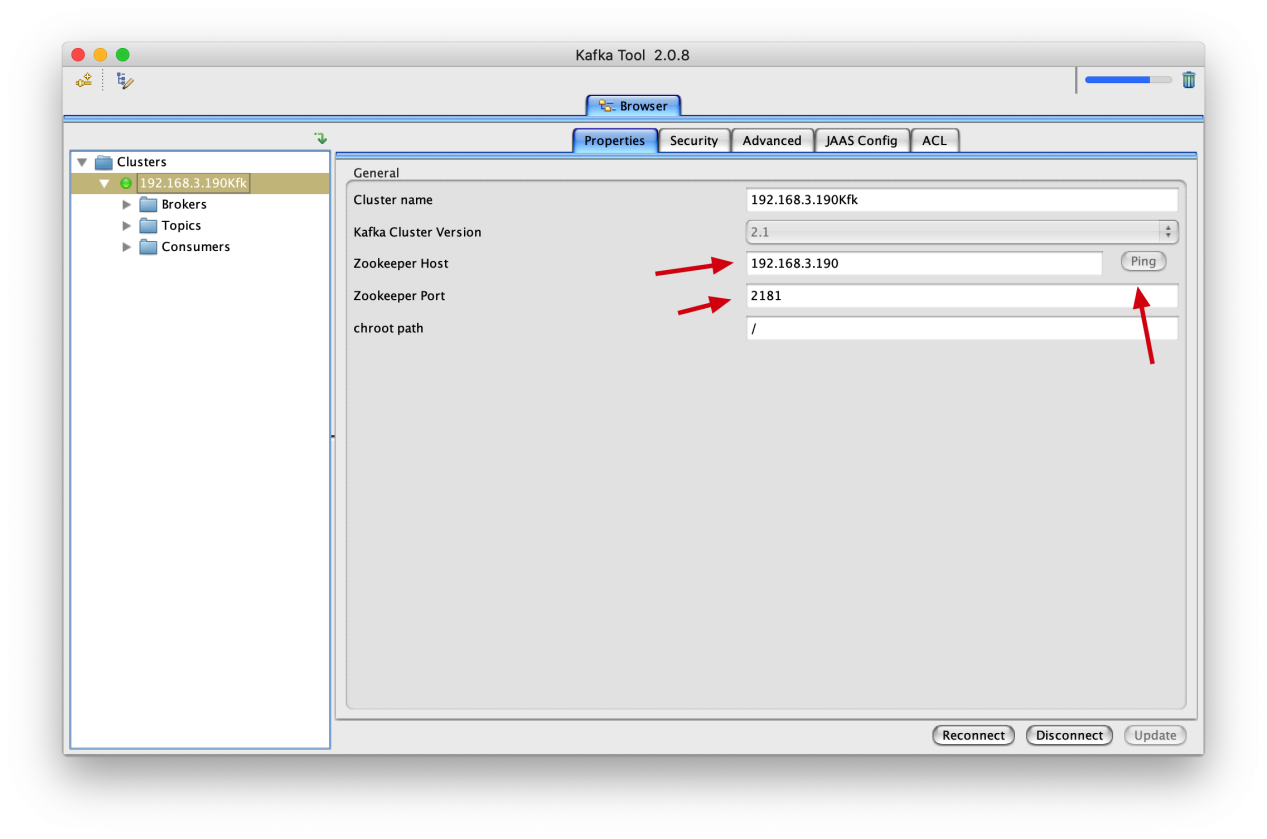
## 消费者

|  |
| --- |
| **package** com.zqkafka.config;  **import** lombok.extern.slf4j.Slf4j; **import** org.apache.kafka.clients.consumer.ConsumerRecord; **import** org.springframework.kafka.annotation.KafkaListener; **import** org.springframework.kafka.support.Acknowledgment; **import** org.springframework.kafka.support.KafkaHeaders; **import** org.springframework.messaging.handler.annotation.Header; **import** org.springframework.stereotype.Component;  **import** java.util.Optional;  */\*\*  \** ***@author*** *iszengziqiang@163.com  \** ***@date*** *2020/8/17 18:01  \** ***@Version:*** *1.0  \** ***@desc*** *//****todo*** *\*/* @Component @Slf4j **public class** KafkaConsumer {   *// @KafkaListener(topics = KafkaProducer.TOPIC\_TEST, groupId = KafkaProducer.TOPIC\_GROUP1) // public void topic\_test(ConsumerRecord<?, ?> record, Acknowledgment ack, @Header(KafkaHeaders.RECEIVED\_TOPIC) String topic) { // System.out.println("1ConsumerRecord:====>" + record.toString()); // Optional message = Optional.ofNullable(record.value()); // if (message.isPresent()) { // Object msg = message.get(); // log.info("topic\_test 消费了： Topic:" + topic + ",Message:" + msg); // ack.acknowledge(); // } // } // // @KafkaListener(topics = KafkaProducer.TOPIC\_TEST, groupId = KafkaProducer.TOPIC\_GROUP2) // public void topic\_test1(ConsumerRecord<?, ?> record, Acknowledgment ack, @Header(KafkaHeaders.RECEIVED\_TOPIC) String topic) { // System.out.println("2ConsumerRecord:====>" + record.toString()); // Optional message = Optional.ofNullable(record.value()); // if (message.isPresent()) { // Object msg = message.get(); // log.info("topic\_test1 消费了： Topic:" + topic + ",Message:" + msg); // ack.acknowledge(); // } // }* @KafkaListener(topics = KafkaProducer.***TOPIC\_TEST***, groupId = KafkaProducer.***TOPIC\_GROUP1***)  **public void** topic\_test1(ConsumerRecord<?, ?> record, Acknowledgment ack, @Header(KafkaHeaders.***RECEIVED\_TOPIC***) String topic) {  System.***out***.println(**"2ConsumerRecord:====>"** + record.toString());  Optional message = Optional.*ofNullable*(record.value());  **if** (message.isPresent()) {  Object msg = message.get();  ***log***.info(**"topic\_test1 消费了： Topic:"** + topic + **",Message:"** + msg);  ack.acknowledge();  }  } } |

备注：消费者消费必须配置groupId，不然会报错。

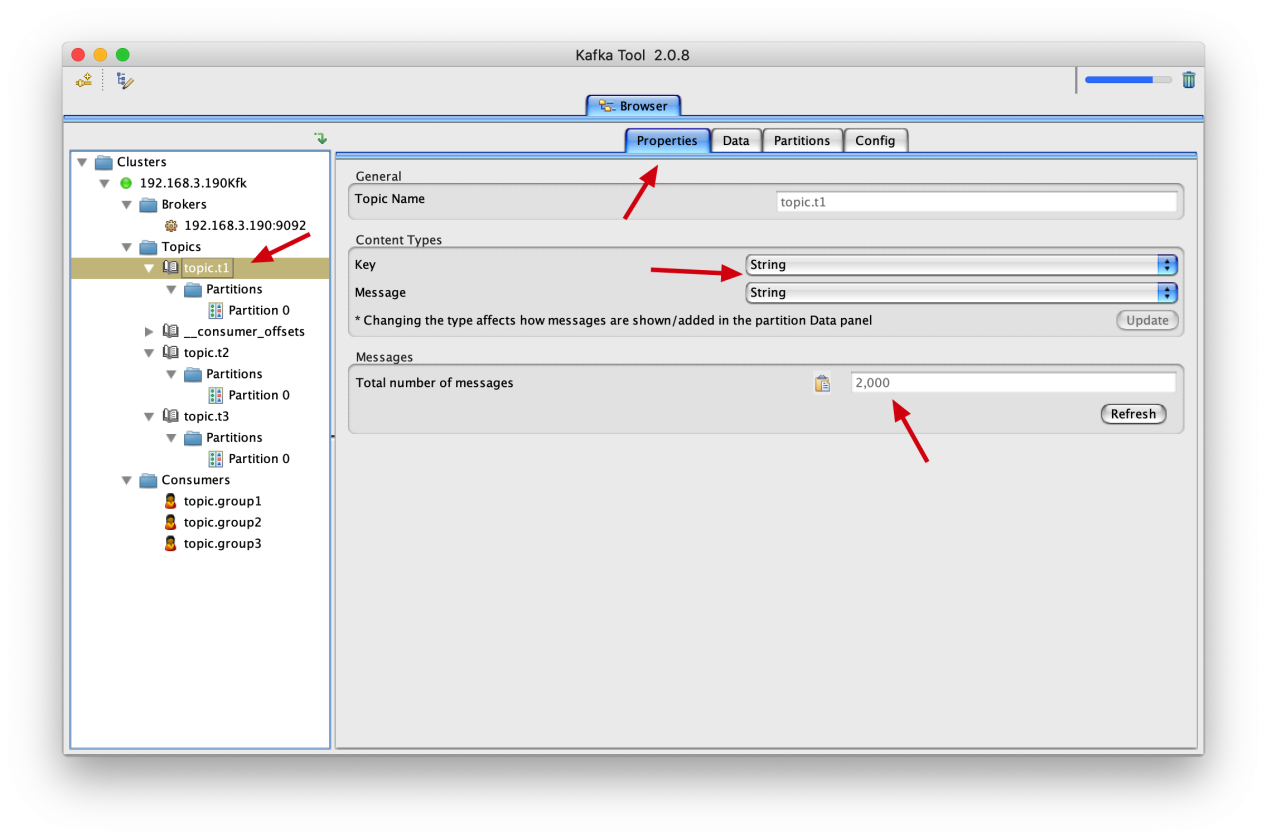
# kafka客户端使用

配置zookeeper地址

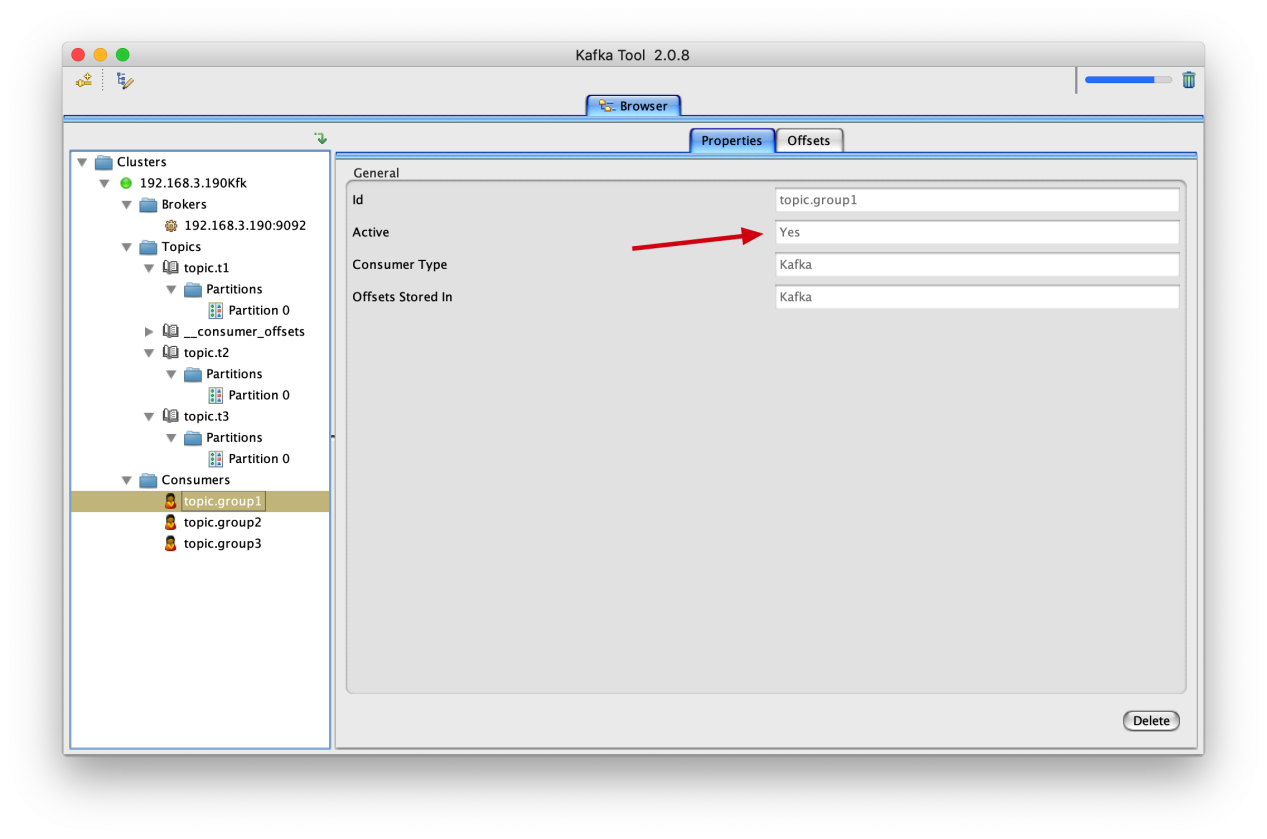


配置kafka地址，集群用逗号隔开





查看消费者存活状态



查看还有多少条消息为消费 -- lag表示未消费数量

