拉勾教育
—互联网人实战大学—

《31 讲带你搞懂 SkyWalking》

徐郡明 资深技术专家

— 拉勾教育出品 —

第33讲：优化 Trace 上报性能 让你的 OAP 集群轻松抗住百万流量

背景

Agent 中

TraceSegmentServiceClient 上报 TraceSegment 数据的方式是 gRPC（客户端流式发送）

使用客户端流式 gRPC

比 HTTP 1.1 的交互方式更快地提供响应

但是在微服务的架构中，依然会面临一些挑战

背景

1. **削峰**

Trace 数据会先写入到 Kafka 中，然后由 OAP 服务进行消费

如果出现了尖峰流量，也会先缓存到 Kafka 集群中，这样 OAP 服务不会被突增流量打垮

待尖峰流量过去之后，OAP 服务会将 Kafka 缓存的数据全部消费掉

2. **扩展性**

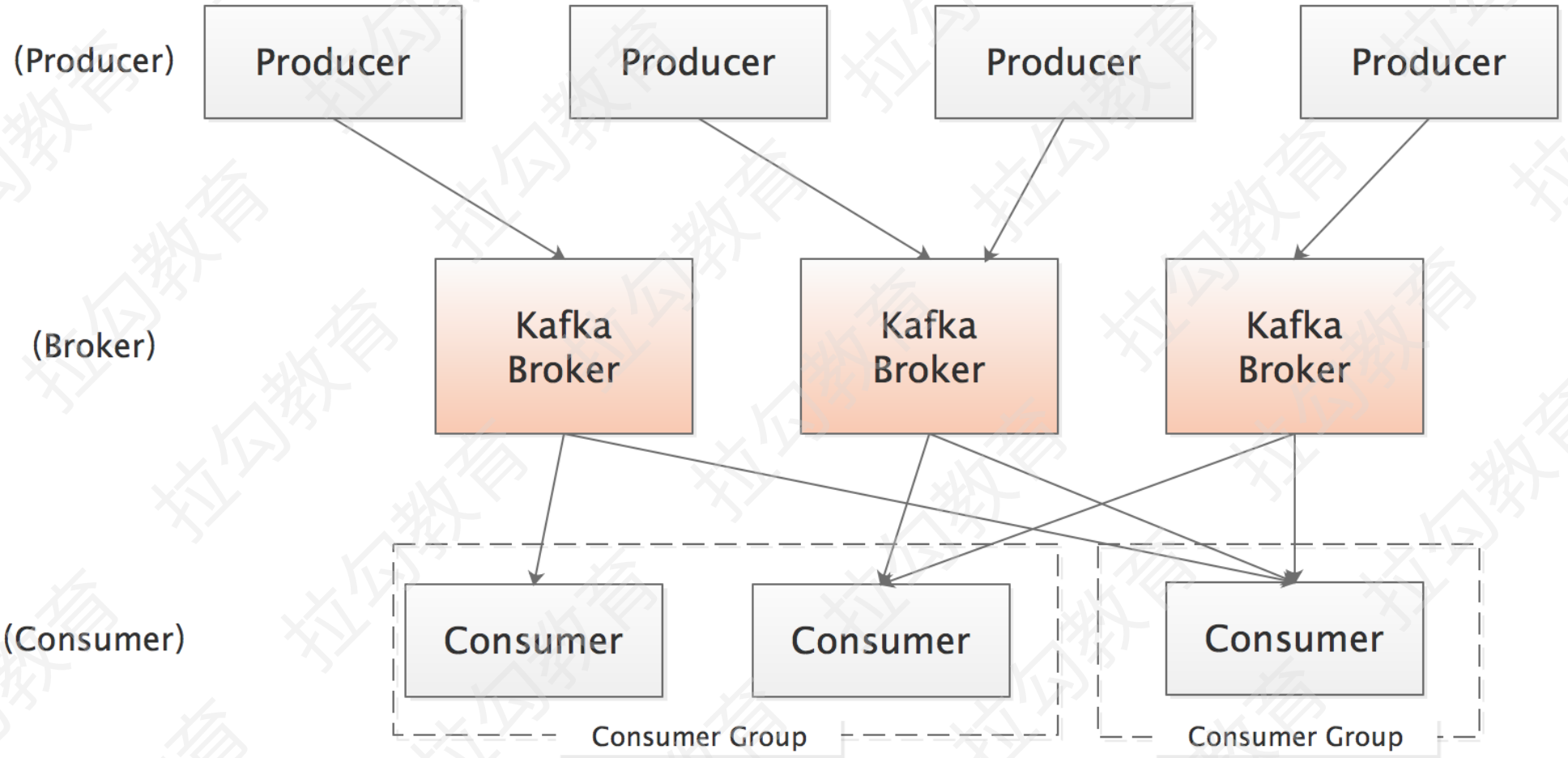
当 Trace 数据或是其他 JVM 监控数据增大到 OAP 集群的处理上限之后，只需要增加新的 OAP 服务即可

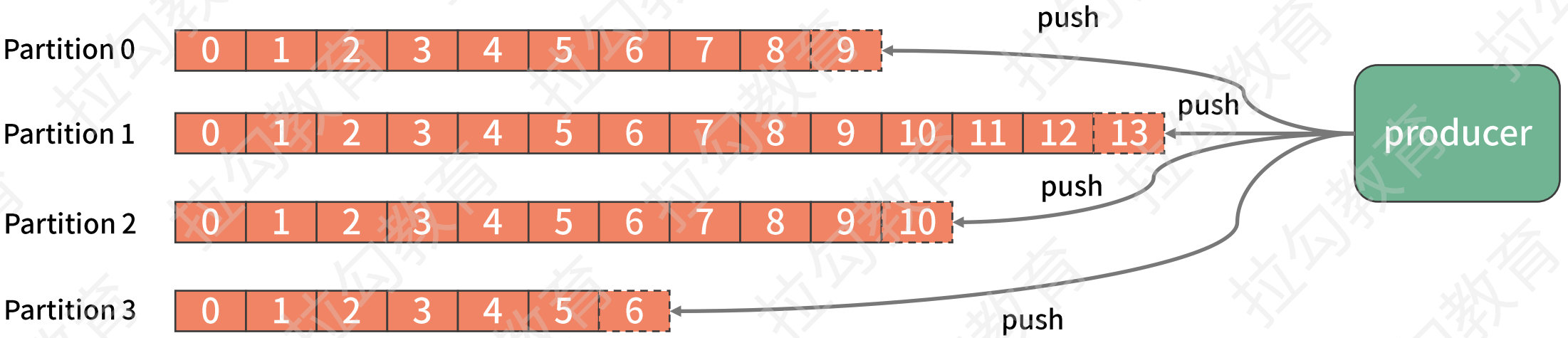
3. **多副本**

Kafka 中的消息会有多个副本

即使 Kafka 集群中的一台机器或是 OAP 集群的一个实例宕机，也不会导致数据丢失

Kafka 基础入门


(Producer)
Producer
Producer
Producer
Producer
(Broker)
Kafka Broker
Kafka Broker
Kafka Broker
(Consumer)
Consumer
Consumer
Consumer
Consumer Group
Consumer Group



保留策略（Retention Policy）& 日志压缩（Log Compaction）

无论消费者是否已经消费了消息，Kafka 都会一直保存这些消息，但并不会像数据库那样长期保存

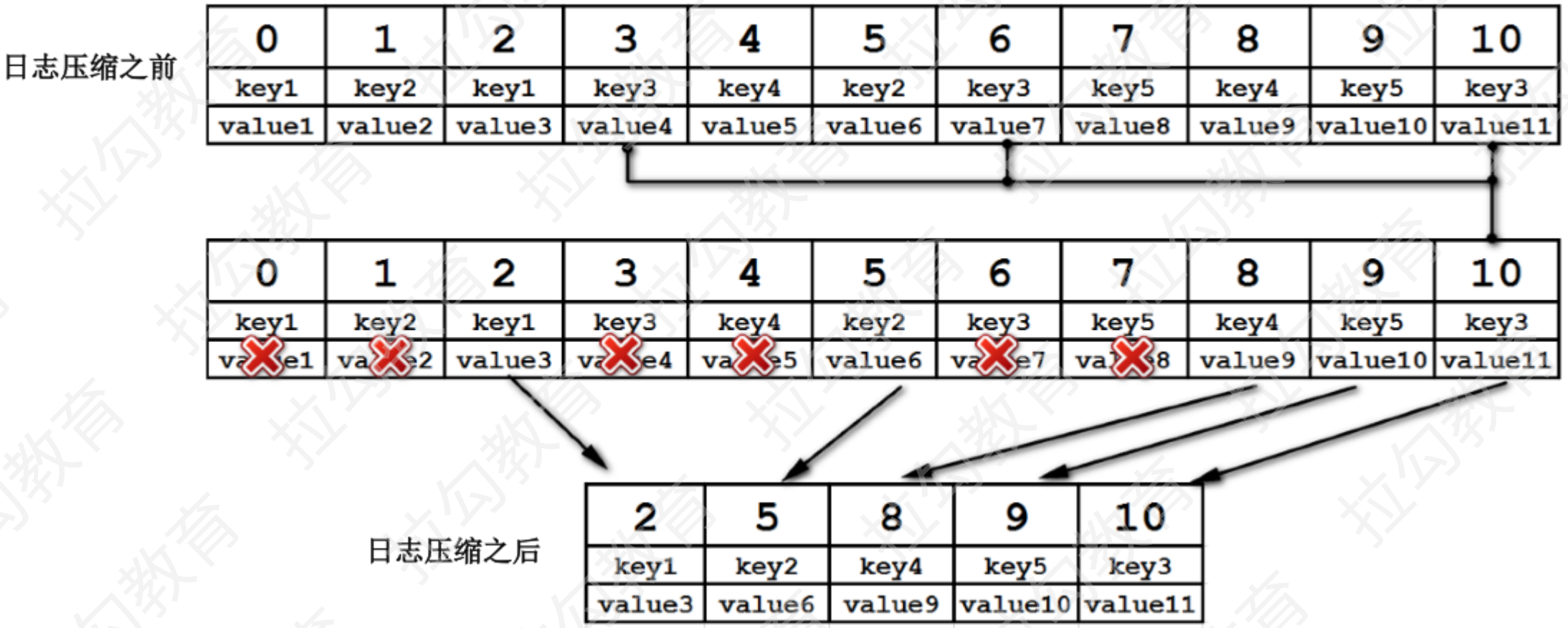
为了避免磁盘被占满，Kafka 会配置相应的“保留策略”（Retention Policy）

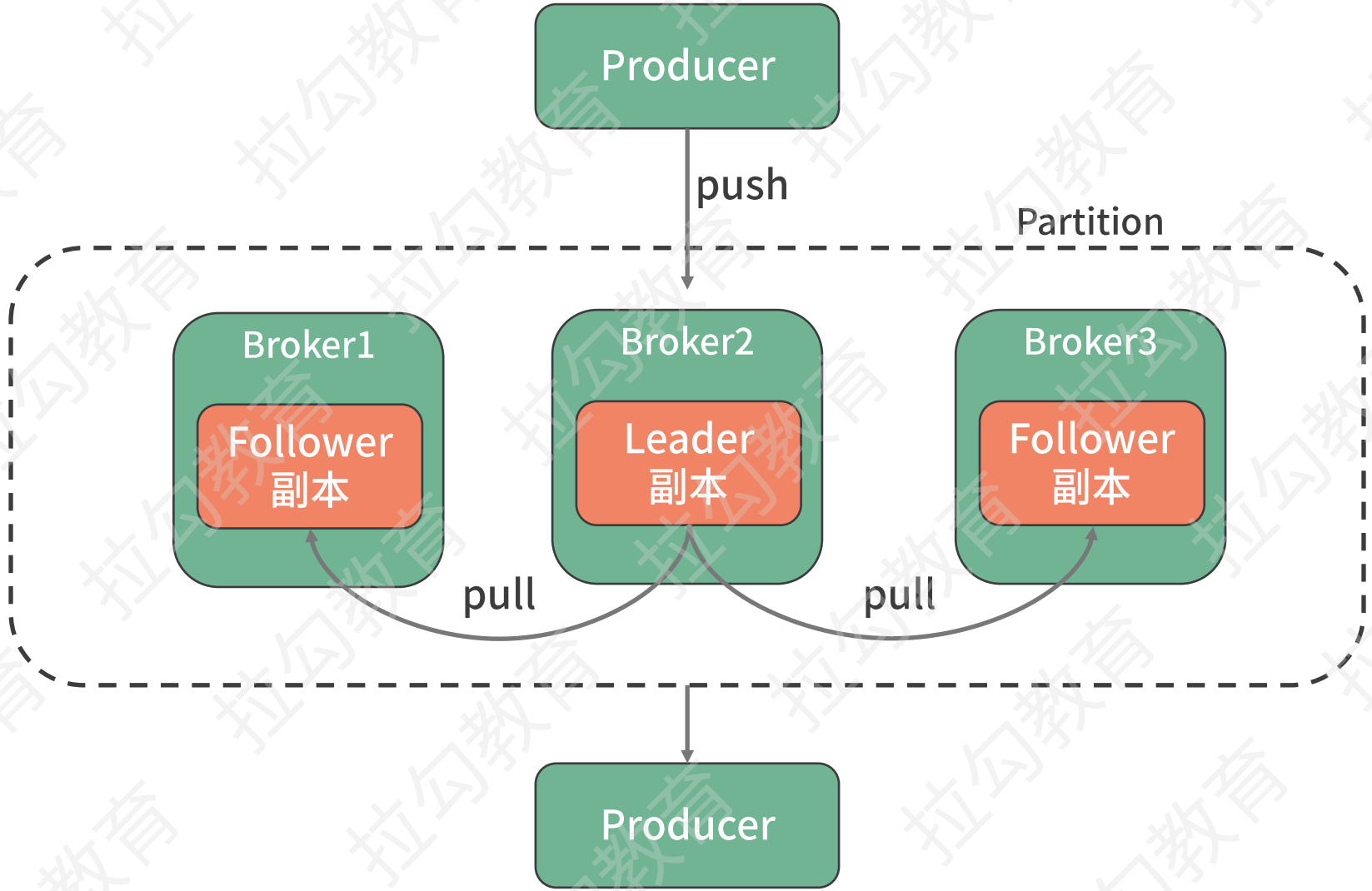
以实现周期性的删除陈旧的消息。

Kafka 基础入门

Kafka 中有两种“保留策略”

1. 根据消息保留的时间，当消息在 Kafka 中保存的时间超过了指定时间，就可以被删除
2. 根据 Topic 存储的数据大小，当 Topic 所占的日志文件大小大于一个阈值，则可以开始删除最旧的消息



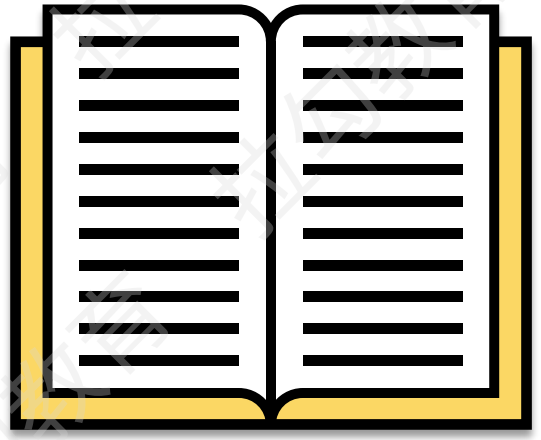


ISR（In-Sync Replica）集合

表示的是目前“可用”（alive）且消息量与 Leader 相差不多的副本集合

其实际含义是ISR集合中的副本必须满足下面两个条件：

1. 副本所在节点必须维持着与ZooKeeper的连接
2. 副本最后一条消息的 offset 与 Leader 副本的最后一条消息的 offset 之间的差值不能超出指定的阈值

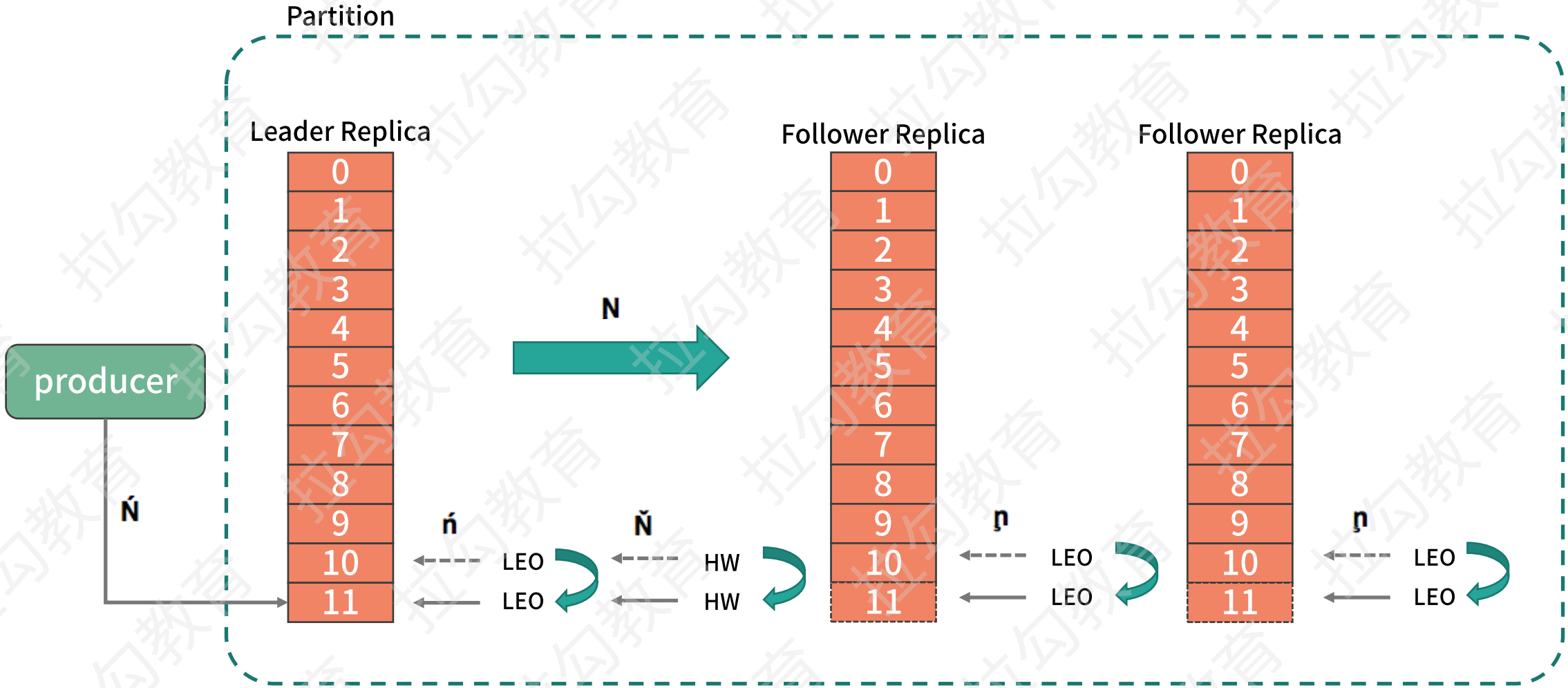


Kafka 基础入门

HW（HighWatermark）标记了一个特殊的 offset

当消费者处理消息的时候，只能拉取到 HW 之前的消息

HW 之后的消息对消费者来说是不可见的，也是由 Leader Replica 管理



常用的方案有同步复制和异步复制：

- 同步复制要求所有能工作的 Follower Replica 都复制完，这条消息才会被认为提交成功
- 异步复制中，Leader Replica 收到生产者推送的消息后，就认为此消息提交成功

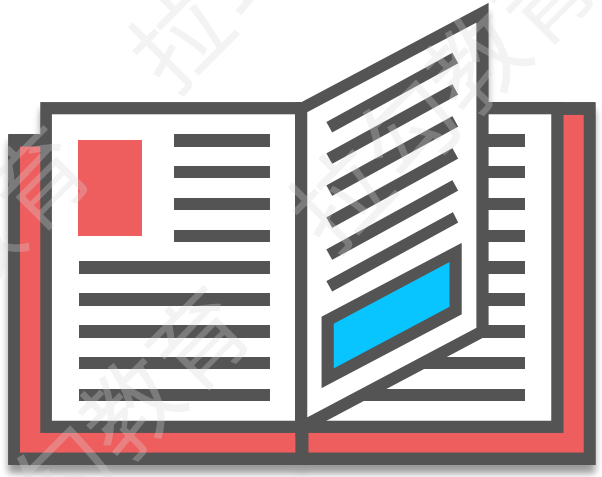
Cluster&Controller

多个 Broker 可以做成一个 Cluster（集群）对外提供服务

每个 Cluster 当中会选举出一个 Broker 来担任 Controller，Controller 是 Kafka 集群的指挥中心

而其他 Broker 则听从 Controller 指挥实现相应的功能

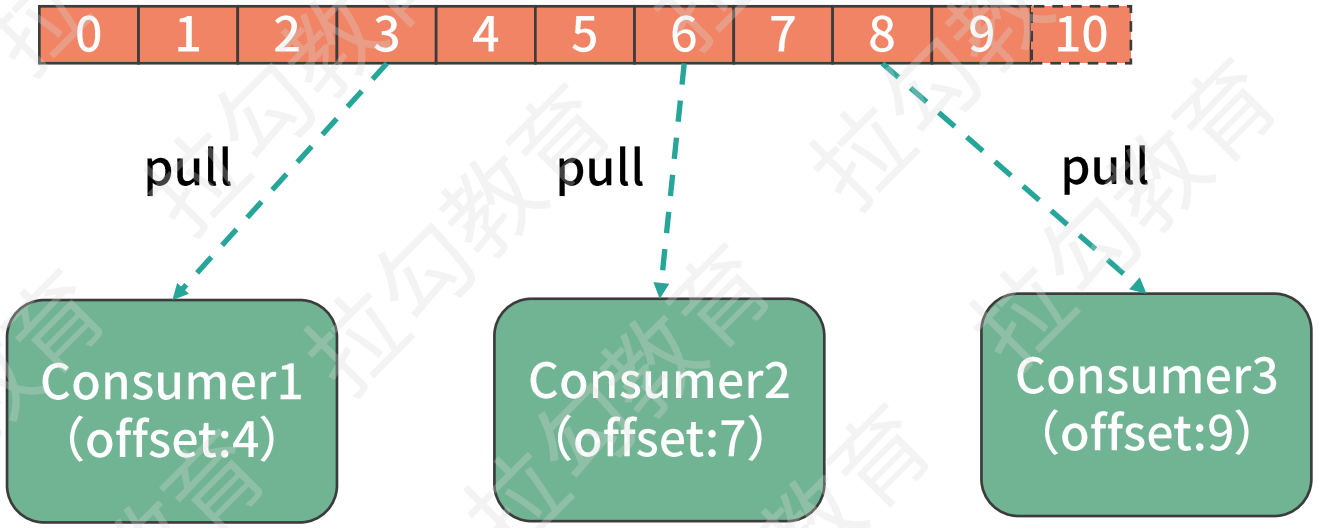
Controller 负责管理分区的状态、管理每个分区的 Replica 状态、监听 Zookeeper 中数据的变化等工作



Consumer

从 Topic 中拉取消息，并对消息进行消费

某个消费者消费到 Partition 的哪个位置（offset）的相关信息，是 Consumer 自己维护的



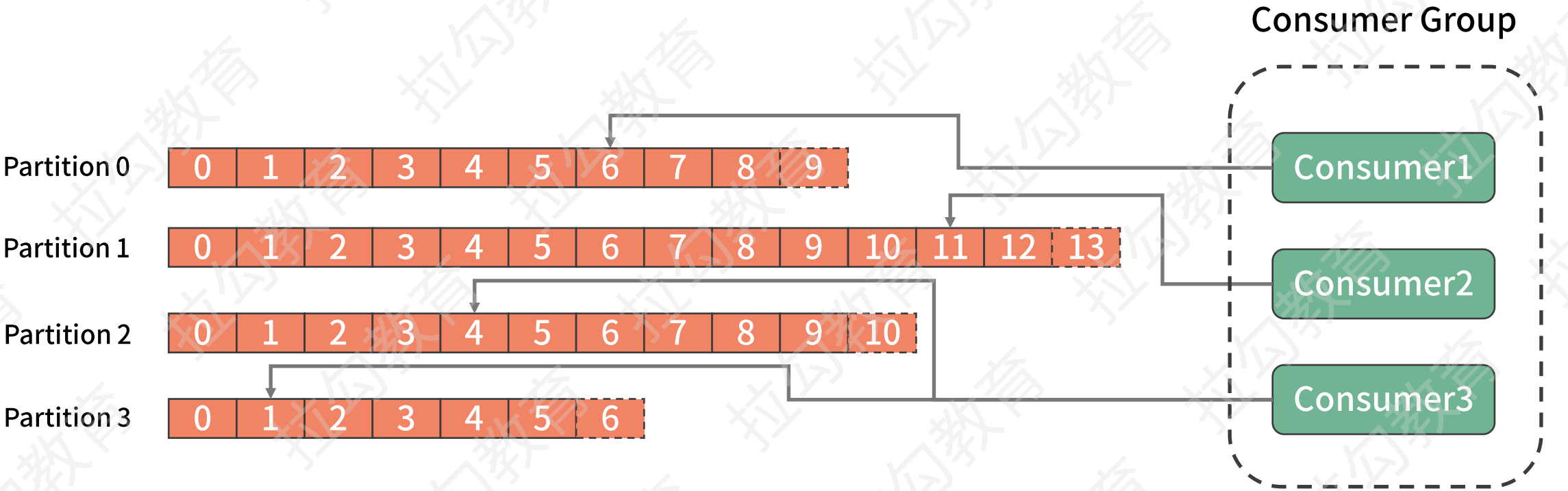
背景

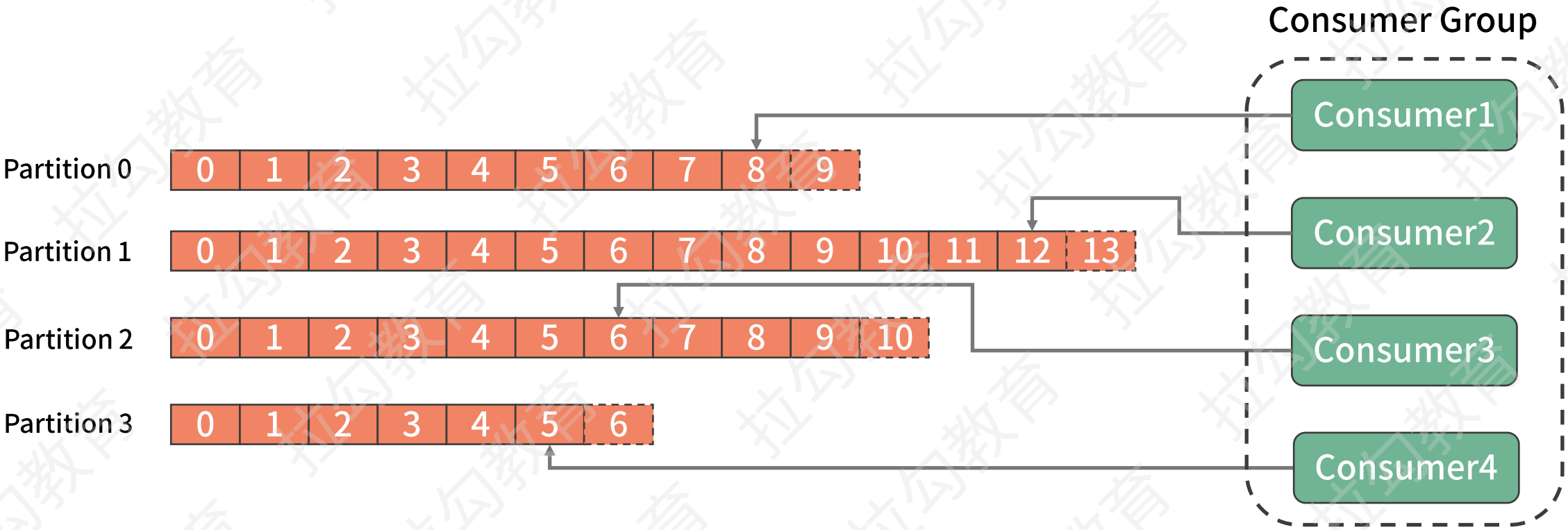
Consumer Group

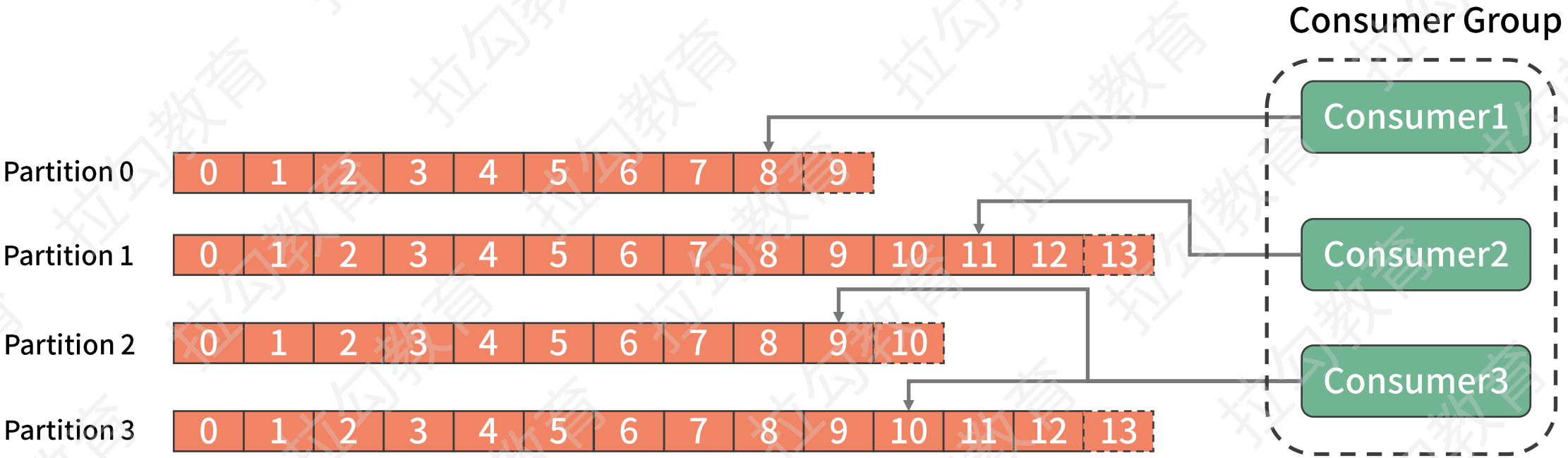
在 Kafka 中，多个 Consumer 可以组成一个 Consumer Group

一个Consumer 只能属于一个 Consumer Group

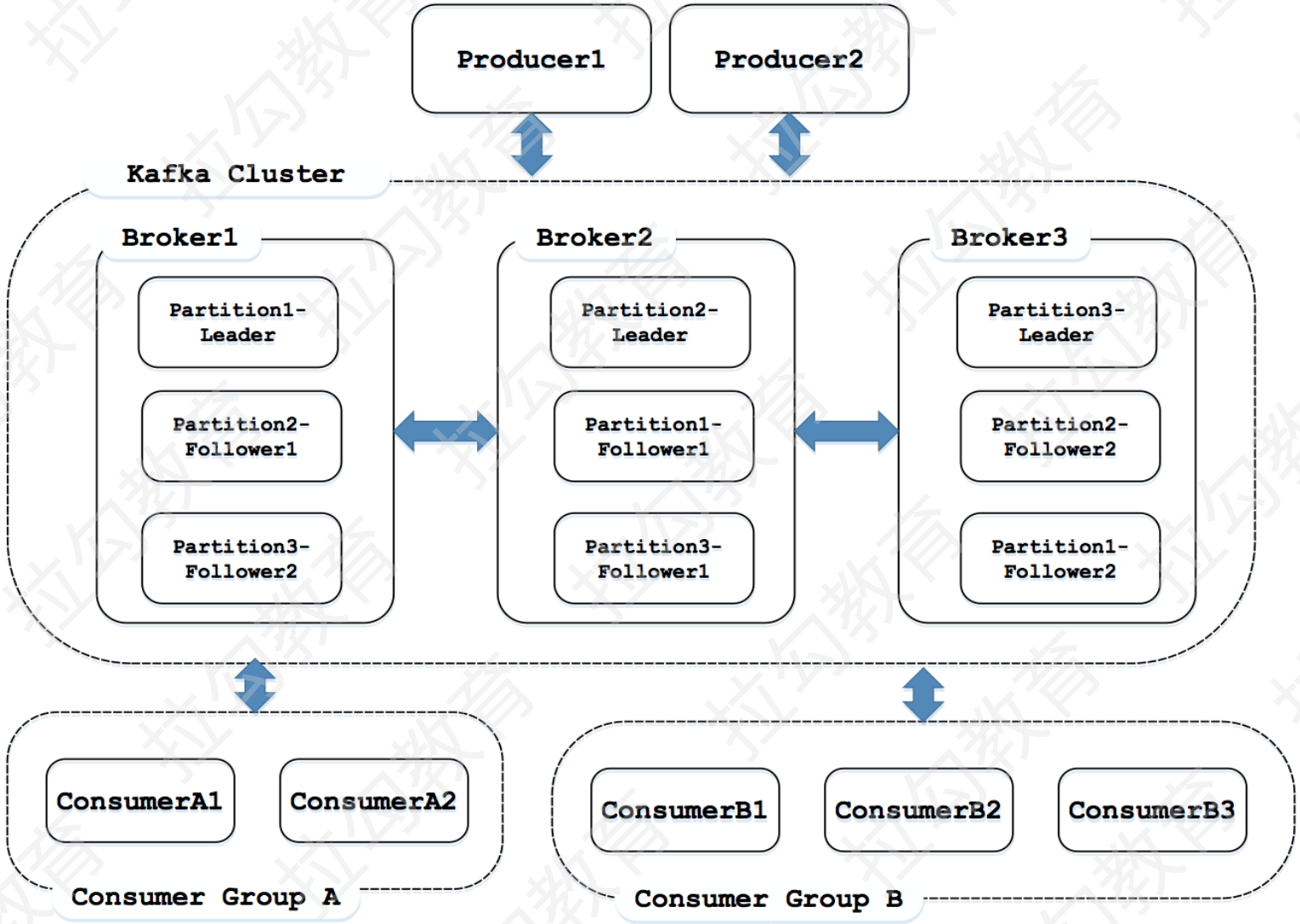
Consumer Group 保证其订阅的 Topic 的每个Partition 只被分配给此 Consumer Group 中的一个消费者处理







Kafka 基础入门


Producer1
Producer2
Kafka Cluster
Broker1
Partition1-Leader
Partition2-Follower1
Partition3-Follower2
Broker2
Partition2-Leader
Partition1-Follower1
Partition3-Follower1
Broker3
Partition3-Leader
Partition2-Follower2
Partition1-Follower2
ConsumerA1
ConsumerA2
Consumer Group A
ConsumerB1
ConsumerB2
ConsumerB3
Consumer Group B

Kafka 环境搭建

拉勾教育
互联网人实战大学

Kafka 是使用 **Scala** 语言编写的

Scala 是一种现代多范式编程语言，集成了面向对象和函数式编程的特性

这里使用 **Scala 2.13** 版本

从官网下载 Scala 安装包并执行命令解压：

```
tar -zxf scala-2.13.1.tgz
```

Kafka 环境搭建

编辑 .bash_profile 文件添加 $SCALA_HONME

```
export SCALA_HOME=/Users/xxx/scala-2.13.1
export PATH=$PATH:$JAVA_HOME:$SCALA_HOME/bin
```

Kafka 环境搭建

编辑完成后，保存并关闭 .bash_profile 文件，执行 source 命令

```
source .bash_profile
```

Kafka 环境搭建

最后执行 scala -version 命令，看到如下输出即安装成功

```
scala -version
Scala code runner version 2.13.1 -- Copyright 2002-2019, LAMP/EPFL and Lightbend, Inc.
```

Download

2.4.0 is the latest release. The current stable version is 2.4.0.

You can verify your download by following these procedures and using these KEYS.

2.4.0

- Released December 16, 2019
- Release Notes
- Source download: kafka-2.4.0-src.tgz (asc, sha512)
- Binary downloads:
  - Scala 2.11 - kafka_2.11-2.4.0.tgz (asc, sha512)
  - Scala 2.12 - kafka_2.12-2.4.0.tgz (asc, sha512)
  - Scala 2.13 - kafka_2.13-2.4.0.tgz (asc, sha512)

We build for multiple versions of Scala. This only matters if you are using Scala and
Otherwise any version should work (2.12 is recommended).

安装 Kafka

执行如下命令解压缩

```
tar -zxf kafka_2.13-2.4.0.tgz
```

安装 Kafka

打开 ./config/server.properties 文件

将其中的 log.dirs 这一项指向上面创建的 logs 目录

```
vim ./config/server.properties

# A comma separated list of directories under which to store log files
log.dirs=/Users/xxx/kafka_2.13-2.4.0/logs
```

安装 Kafka

执行如下命令即可启动 Kafka，启动过程中关注一下日志，不报错即可

```
./bin/kafka-server-start.sh ./config/server.properties
```

验证

创建一个名为“test”的 Topic

```
./bin/kafka-topics.sh --create --zookeeper localhost:2181 \
  --replication-factor 1 --partitions 1 --topic test
#输出下面的一行，即为创建成功
Created topic test.
```

验证

启动命令行 Producer，并输入一条消息“This is a test Message”
以回车结束

```
./bin/kafka-console-producer.sh --broker-list localhost:9092 \
--topic test
>This is a test Message
```

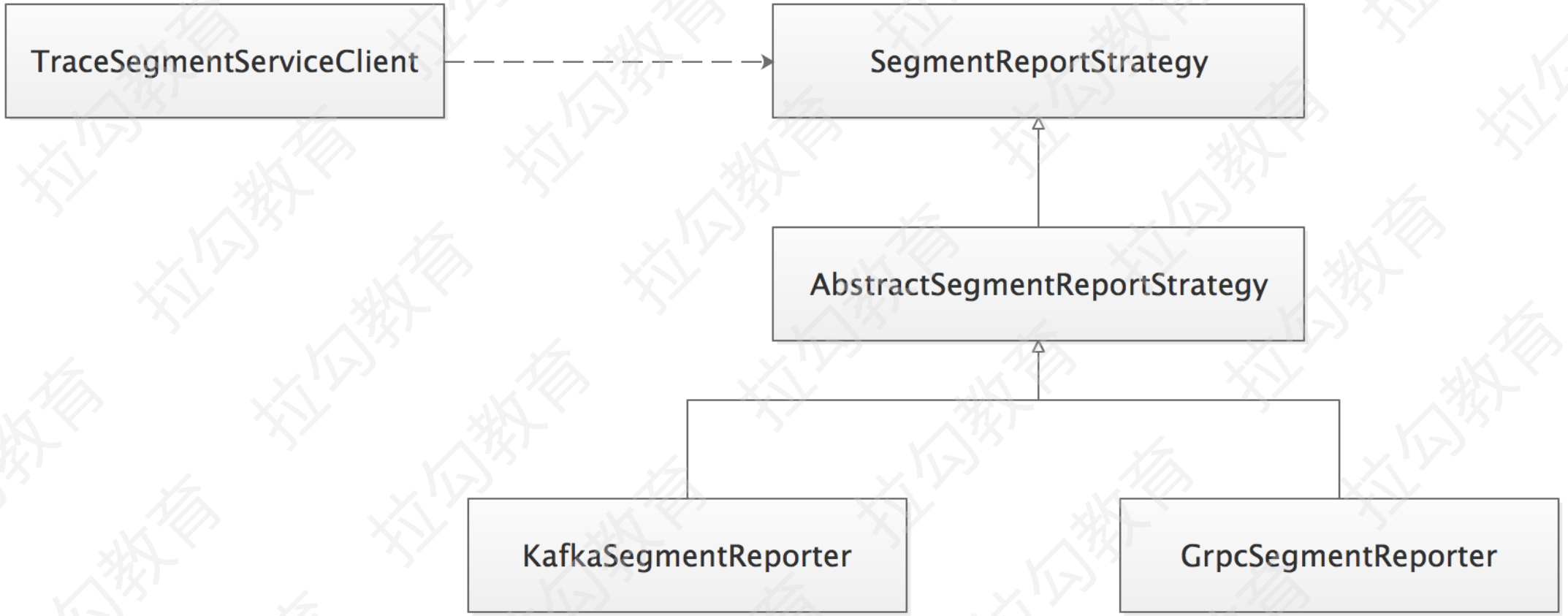
验证

启动命令行 Consumer，可以接收到前面输入的消息

如下所示即表示 Kafka 安装并启动成功

```
./bin/kafka-console-consumer.sh --bootstrap-server localhost:9092
\
--topic test --from-beginning
>This is a test Message
```

Agent 改造



Agent 改造


```
public interface SegmentReportStrategy extends GRPCChannelListener{

    void report(List<TraceSegment> data);

}
```

Agent 改造


```
<dependency>
  <groupId>org.apache.kafka</groupId>
  <artifactId>kafka-clients</artifactId>
  <version>2.4.0</version>
</dependency>
```

Agent 改造


```
public KafkaSegmentReporter(String topic) {
  if (!StringUtil.isEmpty(topic)) {
    this.topic = topic; //默认 topic为 "sw_segment_topic"
  }
  Properties props = new Properties();
  // Kafka服务端的主机名和端口号，关于 Kafka集群的配置可以写到 agent.config
  //配置文件中，然后通过 Config读取，这里为了演示简单，直接硬编码了
  props.put("bootstrap.servers", "localhost:9092");
  // UpstreamSegmentSerializer用来将UpstreamSegment对象序列化成字节数组
  props.put("value.serializer", "org.apache.skywalking.apm.agent
      .core.remote.UpstreamSegmentSerializer");
  producer = new KafkaProducer<>(props); //生产者的核心类
}
```

Agent 改造

```
public void doReport(List<TraceSegment> data) {
  for (TraceSegment segment : data) {
    //将 TraceSegment封装成 UpstreamSegment对象
    UpstreamSegment upstreamSegment = segment.transform();
    //只添加了消息 value，并未指定消息的 key
    ProducerRecord<Object, UpstreamSegment> record =
        new ProducerRecord<>(topic, upstreamSegment);
    //发送消息
    producer.send(record, (recordMetadata, e) -> {
      if (e != null) { //该回调用来监听发送过程中出现的异常
        segmentUplinkedCounter += data.size();
        segmentAbandonedCounter += data.size();
      }
    });
  }
}
```

Agent 改造


```
public void prepare() throws Throwable {

    ServiceManager.INSTANCE.findService(GRPCChannelManager.class)
        .addChannelListener(this);
    if (Config.Report.strategy == Strategy.GRPC) {
        segmentReportStrategy = new GrpcSegmentReporter();
    } else {
        segmentReportStrategy = new
            KafkaSegmentReporter(Config.Report.topic);
    }
}
```

Agent 改造

从 DataCarrier 中消费 TraceSegment 的时候

只需委托给当前 SegmentReportStrategy 对象即可

```
public void consume(List<TraceSegment> data) {
    segmentReportStrategy.report(data);
}
```

Agent 改造

在 demo-webapp、demo-provider 使用的 agent.config 配置文件的末尾添加如下配置，将它们切换为 Kafka 方式上报

```
report.strategy = ${SW_LOGGING_LEVEL:KAFKA}
```

Agent 改造

在 Config 中需要添加相应的 Report 内部类来读取该配置

```
public static class Report{
    public static Strategy strategy = Strategy.GRPC;
}
```

trace-receiver-plugin 改造

为了处理 Kafka 上报方式，我们先要引入 Kafka Client 的依赖

```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>2.4.0</version>
</dependency>
```

trace-receiver-plugin 改造

拉勾教育
互联网人实战大学

```
public TraceSegmentReportServiceConsumer(SegmentParseV2.Producer
segmentProducer, String topic) {
    Properties props = new Properties();
    props.put("bootstrap.servers", "localhost:9092"); // Broker的地址
    props.put("group.id", "sw_trace"); //所属Consumer Group的Id
    props.put("enable.auto.commit", "true");//自动提交offset
    //自动提交offset的时间间隔
    props.put("auto.commit.interval.ms", "1000");
    props.put("session.timeout.ms", "30000");
    // value使用的反序列化器
```

trace-receiver-plugin 改造

拉勾教育
互联网人实战大学

```
props.put("value.deserializer","org.apache.skywalking.oap.server   .receiver.tra
ce.provider.handler.kafka.UpstreamSegmentDeserializer");
    this.consumer = new KafkaConsumer<>(props);
    this.segmentProducer = segmentProducer;
    this.topic = topic;
    //负责消费的线程
    this.consumerExecutor =
        Executors.newSingleThreadScheduledExecutor();
}
```

trace-receiver-plugin 改造

拉勾教育
互联网人实战大学

```
private void consume() {
    consumer.subscribe(Arrays.asList(topic)); // 订阅Topic
    while (true) {
      //从 Kafka集群拉取消息，每次poll()可以拉取多个消息
      ConsumerRecords<String, UpstreamSegment> records =
            consumer.poll(100);
      // 消费信息
      for (ConsumerRecord<String, UpstreamSegment> record:records){

        segmentProducer.send(record.value(), SegmentSource.Agent);
      }
    }
  }
```

trace-receiver-plugin 改造

拉勾教育
互联网人实战大学

```
String reportStrategy = moduleConfig.getReportStrategy();
if(!StringUtil.isEmpty(reportStrategy) &&
    "kafka".equals(reportStrategy.toLowerCase())){
  segmentReportServiceConsumer = new
    TraceSegmentReportServiceConsumer(segmentProducerV2,
      moduleConfig.getKafkaTopic());
  segmentReportServiceConsumer.start();
}
```

trace-receiver-plugin 改造


```
public class TraceServiceModuleConfig extends ModuleConfig {
    ... ... // 省略其他已有字段
    @Setter @Getter private String reportStrategy = "kafka";
    @Setter @Getter private String kafkaTopic = "sw_segment_topic";
}


receiver-trace:
  default:
    #省略已有的配置信息
    reportStrategy: ${SW_REPORT_STRATEGY:kafka}
    kafkaTopic: ${SW_KAFKA_TOPIC:sw_segment_topic}
```

验证

为了验证上述的改造是否成功

将改造后的 Agent 切换成 Kafka 上报模式

打开 trace-receiver-plugin 插件接收 Kafka 上报 Trace 的功能

同时还可以开启一个命令行 Kafka Consumer

从 apm-sdk-plugin 模块中暂时删除 apm-kafka-v1-plugin-6.2.0 模块

验证


```
[@ kafka_2.13-2.4.0]$ ./bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic sw_segment_topic

A+00000

@+n0000ē00- 0ŋ00-:9com.xxx.service.DefaultHelloService.say(java.lang.String)P0000000000ē00- 0ŋ00-*@

A+0000 A*172.17.32.91:20880BAB
/hello/xxxR
/hello/xxx:(com.xxx.service.HelloService.say(String)X`zJ
urlCdubbo://172.17.32.91:20880/com.xxx.service.HelloService.say(String) @


A+00000
A+00000 00- 0ŋ00-:(com.xxx.service.HelloService.say(String)J172.17.32.91:20880PX`zJ
urlCdubbo://172.17.32.91:20880/com.xxx.service.HelloService.say(String)00000- 0ŋ00-:?com.xxx.controller.HelloWorldController.hello(java.lang.String)Pz
hello-trace activationxxxe0000000000000- 0ŋ00-:
/hello/xxxX`z&
urlhttp://localhost:8000/hello/xxxz
```

验证


/hello/xxx
65.43.15803553454550001
起始点 2020-01-30 11:35:45
持续时间 4669 ms
跨度 5
列表
树结构
表格
demo-webapp
demo-provider
0
500
1s
1.5s
2s
2.5s
3s
3.5s
4s
4.5s
/hello/xxx
Http - Tomcat
com.xxx.controller.HelloWorldController....
Unknown
com.xxx.service.HelloService.say(String)
RPCFramework - Dubbo
com.xxx.service.HelloService.say(String)
RPCFramework - Dubbo
com.xxx.service.DefaultHelloService.say(...
Unknown

Next：第34讲《实现线程级别监控，轻松搞定 Thread Dump》

拉勾教育

—互联网人实战大学—

关注拉勾「教育公众号」
获取更多课程信息

L / A / G / O / U