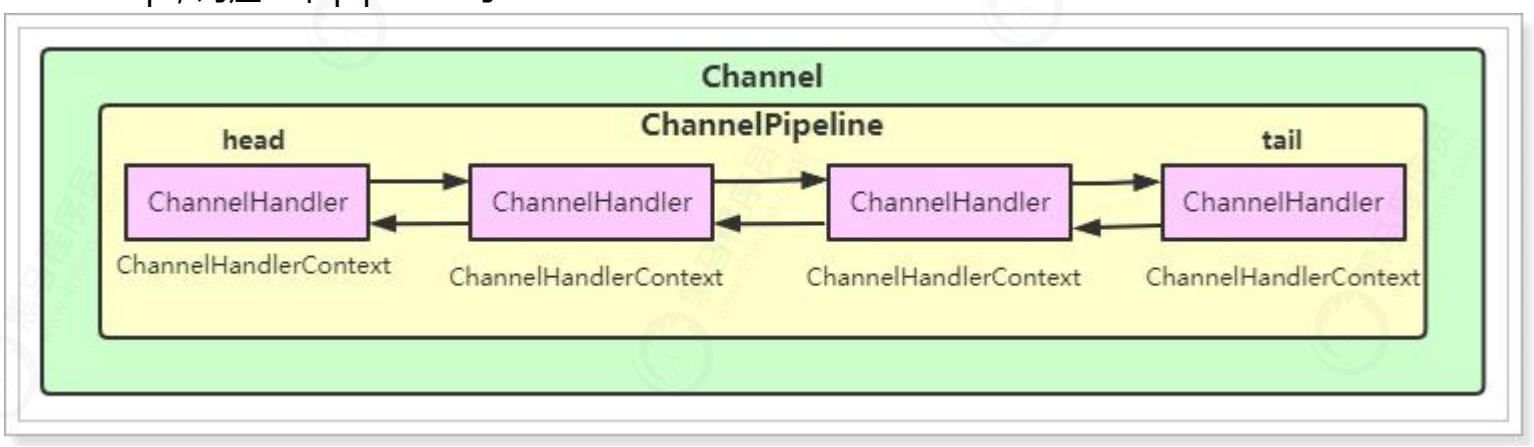
1 期 Java 架构师训练营第 2 阶段课程答疑收集

1、日臻完善-Netty 怎么用更优

- netty 中零拷贝是通过堆外内存实现的，既然不是 jvm 管理的内存那 java 程序是如何访问控制堆外内存的数据
  - 通过底层的 c 语言编写的代码实现直接申请内存空间进行操作
    https://www.cnblogs.com/zhxiansheng/p/11327808.html
- Netty 只是用于 websocket 么
  - 不是的，只要涉及到网络场景都适用，比如：http、rpc、websocket、tcp 等
- 一般用途在哪里
  - 游戏、web、远程会议、即时通讯等
- netty Selector 主从事件什么意思 ，Netty 连接请求是重哪里

进.io.netty.channel.nio.NioEventLoop#processSelectedKey 只是看断点进入有没有业务代码进入的入口

  - 笔误，应该是注册事件
  - 服务端启动后， NioEventLoop 线程会不停地去循环处理网络 IO 事件、普通任务和定时任务。在处理网络 IO 事件时，当轮询到 IO 事件类型为 OP_ACCEPT 时，就表示有新客户端来连接服务端了，也就是检测到了新连接。这个时候，服务端 channel 就会进行新连接的读取。
  - https://blog.csdn.net/qq_34436819/article/details/103728191·
- channel.什么时候注册写事件，写是服务写给客户端，还是客户端写给服务？
  - 写事件一般是针对文件操作的。
- Pipeline（管道）是保证一个大的业务顺序执行吗？
  - 不是
- 一个 EventLoop，对应一个 pipeline 吗？



  -
  - 一个 channel 在一个生命周期内只会注册到一个 EventLoop。
- EventLoop 服务于多个 channel 时，所有 channel 的所有 io 操作（ChannelHandler）都是放到这个 pipeline 里吗？
  - 不是
- 什么时候截断入站处理流水线？什么时候对原消息不做处理？
  - 没有向下传递就是截断了处理流水线
  - 拿到消息后，不做任何处理，直接向下游传递

5.5.3、小结

- 入站处理流程中，如果对原消息不做处理，调用 ctx.fireChannelRead(msg) 把原消息往下传，由流水线最后一棒 TailHandler 完成自动释放。
- 如果截断了入站处理流水线，则可以继承 SimpleChannelInboundHandler ，完成入站ByteBuf 自动释放。
- 出站处理过程中，申请分配到的 ByteBuf，通过 HeadHandler 完成自动释放。
- 入站处理中，如果将原消息转化为新的消息并调用 ctx.fireChannelRead(newMsg)往下传，那必须把原消息release掉;
- 入站处理中，如果已经不再调用 ctx.fireChannelRead(msg) 传递任何消息，也没有继承SimpleChannelInboundHandler 完成自动释放，那更要把原消息release掉;

  -
- 如果不 release 掉，让出站 HeadHandler 完成自动释放 可以吗？
  - 如果是出站，是由 HeadHandler 完成的，如果是入站不可以。
- 解码器不足四个字节，是不是不会往下传递了，后面的 handler 还会执吗？ 还会出站吗？
  - 如果不足 4 个字节，会等待后续的数据，不会向 handler 传递。
  - 不会出站
- 添加的 handler 顺序可以颠倒吗？
  - 自定义的 handler 根据需求设置顺序，内置的 HeadHandler、TaiHandler 不能调整顺序。
- 解码操作，decode 会执行多次？.handler() 添加到 boss 线程执行 ，什么时候需要添加？
  - 每次数据来了都会进行解码操作。
  - serverBootstrap.handler()所设置的 Handler 是在服务启动后就执行，而. childHandler()是在有连接接入之后才执行。

◆ 例如：serverBootstrap.handler(new LoggingHandler(LogLevel.INFO))，在服务启动后就会打印一些 channel 的状态信息

```
九月 08, 2020 4:13:47 下午 io.netty.handler.logging.LoggingHandler channelRegistered
信息: [id: 0x414d0fc6] REGISTERED
九月 08, 2020 4:13:47 下午 io.netty.handler.logging.LoggingHandler bind
信息: [id: 0x414d0fc6] BIND: 0.0.0.0/0.0.0.0:5566
服务器启动成功，端口为：5566
九月 08, 2020 4:13:47 下午 io.netty.handler.logging.LoggingHandler channelActive
信息: [id: 0x414d0fc6, L:/0:0:0:0:0:0:0:0:5566] ACTIVE
九月 08, 2020 4:14:14 下午 io.netty.handler.logging.LoggingHandler channelRead
信息: [id: 0x414d0fc6, L:/0:0:0:0:0:0:0:0:5566] READ: [id: 0x9da6f3cf, L:/127.0.0.1:5566 - R:/127.0.0.1:61235]
九月 08, 2020 4:14:14 下午 io.netty.handler.logging.LoggingHandler channelReadComplete
信息: [id: 0x414d0fc6, L:/0:0:0:0:0:0:0:0:5566] READ COMPLETE
接收到消息：hello netty!
```

2、大促期间网络编程与安全解读

● 分布式环境，跨域页面跳转时，身份验证信息通过哪种方式传递比较好?特别是验证信息很长的情况下

与当前课程关系不大；谈谈自己的经验

1、客户端缓存---cookie

2、服务端缓存---redis

访问的时候携带或者传递 sessionID 就行了，这样就不用担心长度问题了

● nginx 开启 https 后，ssl 初始化时间很长，应该怎么优化

确实是存在这样的问题；原因是 https 相比 http 在进行握手的时候会大量消耗时间

方案

1、在购买证书的时候；安全加密等级不要太高；如果使用更高的加密安全等级；ssl 握手会大量消耗时间

2、新版的 OpenSSL

3、也可以使用第三方的 https 加速负载均衡网关；例如 ksg、宝塔

4、使用硬件加速：比如 SSL 专用加速卡；流用法都是将硬件插入到服务器的 PCI 插槽

● 对于安全怎么操作

以 websocket 举例

1、ws 升级成 wss

2、申请 ssl 证书（收费免费都可以）

3、配置 nginx 的 ssl、或者将证放到工程下面

● 网络通讯对称性的 key 由服务端或者客户端生成，哪种方式更好？

对称加密只有公钥、用于服务端和客户端的加密与解密

优点：性能高

缺点：但是不易于管理公钥

非对称加密采用私钥解密公钥加密；

优点：易于管理公钥

缺点：存在性能问题

所以；在 ssl 中的公钥采用混合加密的方式；也就是对称与非对称的组合

3、基于数据结构和算法的深入应用

1）使用 lua 脚本怎么控制当并发访问时，只让第一个请求去后端，其余请求等有缓存后走缓存

◆ 首先，不可取。因为请求之间并没有互相通信一说。

◆ 举例，请求 1 到来，发现没有缓存，去后端。请求 2 到来，发现还没有缓存，它并不知道前面有没有请求正在写缓存，也不能一直在这里等下去。

◆ 如果硬要这么设计的话。就需要为 lua 配一把分布式锁，可以是服务器文件，可以是 redis。用脚本做双重判断类似于 java 里的单例模式初始化的过程。

◆ 先判断缓存，为空，再判断锁，第一个请求能拿到锁，放进去请求后端

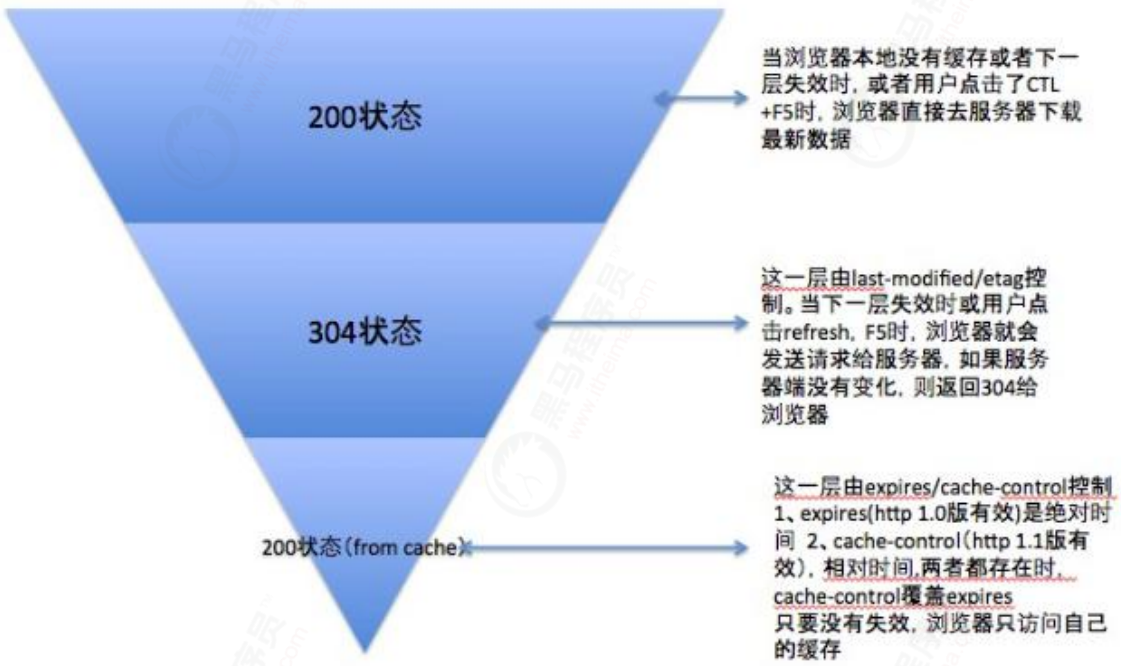
◆ 第二个请求判断缓存为空，发现锁被占，一直循环等待，直到缓存出现再返回

◆ 这个难度太大，也不好维护。所以，不推荐。

2）使用 proxy_cache_purge 清理缓存后，立即请求后端刷新缓存，观察到缓存文件确实更新了，但通过浏览器访问仍然访问到旧数据;如果在清理缓存后过一会再刷新缓存就没有这个问题，这种情况是什么原因？

◆ 请求链路上可能有多级缓存，从前往后顺序一般是，浏览器（用户端）→ nginx 缓存 → app 缓存

◆ 从现象来看，你的缓存在 nginx 已确认刷新过了，而浏览器没有生效。所以浏览器端缓存了旧数据的可能性大。



- 参考地址：https://www.cnblogs.com/kevingrace/p/10459429.html

3）老师可以说说技术选型吗？比如为什么选 redis 不选 mysql

- 这个选型要根据实际场景来。简单来说：mysql，业务数据。Redis，缓存
- Mysql 是典型的关系型数据库，成熟，历史悠久，支持事务保障数据的完整性。Mysql 的数据要写到盘里，落盘后的数据易于保存、备份、相对安全。但是磁盘读写速度慢很多。所以一般用于存放业务数据。如订单、用户、商品、等等。
- Redis 是 nosql，一般用于做缓存。在 web 程序中，也有类似搜索关键词提示，热度排行等地方，使用 redis 的 zset 来完成。Redis 的数据在内存中，所以读取快，天生做缓存的材料。一般放在 mysql 之前存放热度数据，避免读取 mysql（慢）。缺点就是内存相比磁盘昂贵的多，断电内存丢失。Redis 可以支持持久化，但是用它来存储业务数据的企业相对较少。

4）高可用，主从同步怎么处理

- 不同的中间件有不同的处理方式，一般来说，高可用，那就是做冗余。一份靠不住我加一份和你一样的做备胎。
  典型的案例：nginx 对多台 java 服务做负载均衡，proxy_pass 。再比如 keepalive 对数据库集群做浮动 ip。
- 主从同步，一般常见的中间件自身会支持，比如 mysql，比如 redis 的主从，kafka 的副本分区等。这些都可以根据官方要求进行配置一下，就可以自动帮你完成主从的同步。

5）空间复杂度和时间复杂度如何用数学公式计算

- 复杂度，就是根据输入数据的量级，如何衡量你运算得到最终结果所需要的资源的多少。
- 空间那就是方法里所要开辟的临时变量大小
- 时间就是需要执行的次数
- 如果换成数学公式，输入数据的长度为 x（比如数组长度）
- 方法执行中，需要定义临时变量时，最大需要多少长度为 y ，那么 y 和 x 成什么关系？这个表示成数学函数，就是空间复杂度
- 需要执行代码的循环次数最大为多少?和 x 成什么关系？这个函数就是时间复杂度。只不过，我们学的数学函数符号是 f ， 这里是 O

6）一般用途

- 数据结构和算法，不一定会出现在你的业务代码里。但是它背后的理论和思想要掌握。比如时间轮、限流、一致性 hash。这些算法都非常的经典，面试架构可能会问到。负载均衡、调度等算法，springcloud，dubbo，nginx 网关等地方的配置要用到。在课程中像 topk，敏感词过滤，则是实际的业务。在实际开发中可以应用到你的代码中。

7）Nginx 加权轮询算法

- 课题扩展部分：

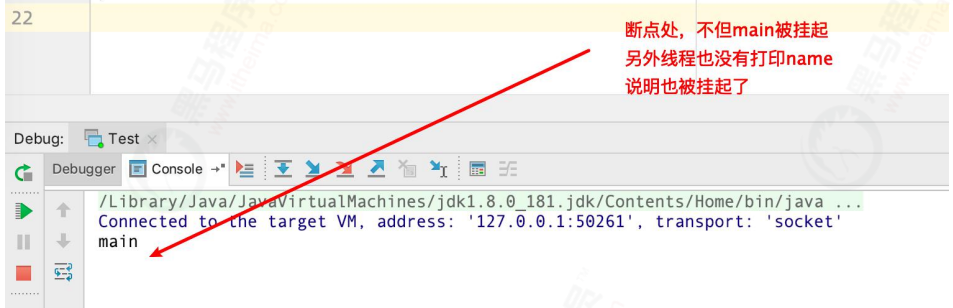
```
upstream bakend{
    server 10.0.0.11:9090 weight=4;    //a
    server 10.0.0.11:8080 weight=2;    //b
    server 10.0.0.11:7070 weight=1;    //c
}
```

次数	响应前	被选中	响应后
1	4，2，1	a	-3，2，1
2	1，4，2	b	1，-3，2
3	5，-1，3	a	-2，-1，3
4	2，1，4	c	2，1，-3
5	6，3，-2	a	-1，3，-2
6	3，5，-1	b	3，-2，-1
7	7，0，0	a	0，0，0

8）多线程 debug 只会刮起当前线程吗

◆ 不是的，会刮起所有线程

```
1   package com.itheima.prize.msg;
2
3   public class Test implements Runnable {
4       @Override
5       public void run() {
6           try {
7               Thread.sleep( millis: 3000);
8           } catch (InterruptedException e) {
9               e.printStackTrace();
10          }
11          System.out.println(Thread.currentThread().getName());
12      }
13
14      public static void main(String[] args) {  args: {}
15          System.out.println("main");
16          new Thread(new Test()).start();
17          new Thread(new Test()).start();
18          new Thread(new Test()).start();
19          System.out.println("end");
20      }
21  }
22
```



4、大促抗住零点洪峰-缓存架构体系

- redis 高并发大数据集中写入会不会出现问题

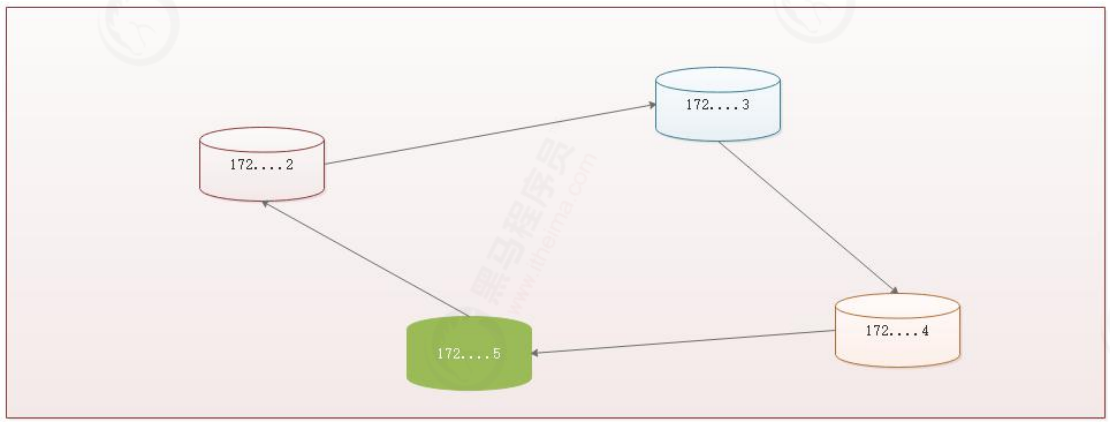
◆ Redis 数据存储在缓存中，高并发场景下，如果是比较大的数据写入，是存在问题的。

1）占用大量内存空间，因为 Redis 数据是存储在内存中，如果数据比较大，会占用大量内存空间，因此不建议存储一些大文件。如果是大文件，建议按照不同维度进行拆分。或者采用 Memcache 效率要高些。

2）大文件存储，消耗时间比较长，会影响 Redis 的性能，并发处理能力会降低。(QPS 降低)

3）持久化操作 IO 消耗大，主从同步数据困难加大。

- redis 集群设置网络有什么用



◆ Docker 容器一共有 4 种网络模式，分别为 bridge 桥接模式、host 模式、container 模式和 none 模式 启动容器时可以使用 –net 参数指定，默认是桥接模式。

- **桥接模式：** 该模式下 Docker Container 不具有一个公有 IP，即和宿主机的 eth0 不处于同一个网段。导致 的结果是宿主机以外的世界不能直接和容器进行通信。容器间网络的传输效率高。

- **host 模式：**可以直接使用 宿主机的 IP 地址与外界进行通信，若宿主机的 eth0 是一个公有 IP，那么容器也拥有这个公 有 IP。直接暴露在公网中了，危险系数增加。

- **container 模式：**新创建的容器和已经存在的一个容器共享一个 Network Namespace，而不是和宿主机共享，新创建的容器不会创建自己的网卡，配置自己的 IP，而是和一个指定的容器共享 IP、端口范围等。能提升共享网络内容器之间数据传输的效率，非共享网络内的容器之间传输数据效率并不能得到提升。

- **none 模式：**不为 Docker Container 任何的网络环境，要想访问容器，必须通过宿主机当前网卡访问。

- 高可用，主从同步怎么处理（Redis）

3 双十一缓存架构设计
- 3.1 缓存架构设计
- 3.2 Redis集群高级应用
  - **3.2.1 Redis版本特性介绍**
  - 3.2.2 Redis集群配置
  - 3.2.3 主从复制
  - 3.2.4 集群扩容收容
    - 3.2.4.1 添加集群主节点
    - 3.2.4.2 哈希槽分配
    - 3.2.4.3 添加集群从节点
    - 3.2.4.4 缩容
- 3.3 Redis Sentinel
  - 3.3.1 哨兵讲解
  - 3.3.2 Sentinel搭建
  - 3.3.3 Sentinel集群讲解

- 老师可以说说技术选型吗？比如为什么选 redis 不选 mysql？

◆ MySQL 特点：保障数据完整性

使用C 和 C++编写，并使用了多种编译器进行测试，保证源代码的可移植性

支持多种操作系统，如 Linux、Windows、AIX、FreeBSD、HP-UXMacOS、NovellNetware、OpenBSD、OS/2 Wrap、Solaris 等

为多种编程语言提供了 API，如 C、C++、Python、Java、Perl、PHP、Eiffel、Ruby 等

支持多线程，充分利用CPU 资源

优化的 SQL 查询算法，有效地提高查询速度

提供多语言支持，常见的编码如 GB2312、BIG5、UTF8

提供 TCP/IP、ODBC 和 JDBC 等多种数据库连接途

提供用于管理、检查、优化数据库操作的管理工具

大型的数据库。可以处理拥有上千万条记录的大型数据库

支持多种存储引擎

MySQL 软件采用了双授权政策，它分为社区版和商业版，由于其体积小、速度快、总体拥有成本低，尤其是开放源码这一特点，一般中小型网站的开发都选择 MySQL 作为网站数据库

MySQL 使用标准的 SQL 数据语言形式

Mysql 是可以定制的，采用了 GPL 协议，你可以修改源码来开发自己的 Mysql 系统

在线 DDL 更改功能

复制全局事务标识

复制无崩溃从机

复制多线程

◆ Redis 特点：提升数据读写能力

内存数据库，速度快，也支持数据的持久化，可以将内存中的数据保存在磁盘中，重

启的时候可以再次加载进行使用。

Redis 不仅仅支持简单的 key-value 类型的数据，同时还提供 list，set，zset，hash 等数据结构的存储。

Redis 支持数据的备份，即 master-slave 模式的数据备份。

支持事务

性能极高 – Redis 能读的速度是 110000 次/s,写的速度是 81000 次/s 。

◆ **全文搜索 Elasticsearch**：海量数据检索

支持海量的、PB 级的大数据搜索
Elasticsearch 是分布式全文搜索引擎
Elasticsearch 海量数据接近实时的搜索。

- 使用 lua 连接 mysql 时，连接配置中 host 如果写 mysql 集群的 fqdn 则报错(报错的地方是 resty.mysql 中的 _send_packet 方法 错误消息是 packet_no 为 nil 不能进行+1 操作)，写 ip 就能正常连接上。由于 ip 经常发生变化(故障转移时 ip 会变化)，必须使用 fqdn 的方式连接，应该怎么解决？

```
if host then
    local port = opts.port or 3306
    if not pool then
        pool = user .. ":" .. database .. ":" .. host .. ":" .. port
    end

    ok, err = sock:connect(host, port, { pool = pool })
```

◆ 协议走的是 Sockt 协议，直接将字符串当做目标 IP 地址了，而且数据库的地址一般变更频率较低，可以直接写 IP。

◆ 如果一定要使用 host 配置，需要用 lua 获取指定的域名映射 ip
local ip, resolved = socket.dns.toip("db_server")
注意：需要安装 socket.lua

- 使用 lua 脚本怎么控制当并发访问时，只让第一个请求去后端，其余请求等有缓存后走缓存 使用 proxy_cache_purge 清理缓存后，立即请求后端刷新缓存，观察到缓存文件确实更新了，但通过浏览器访问仍然访问到旧数据;如果在清理缓存后过一会再刷新缓存就没有这个问题，这种情况是什么原因？

需要添加：add_header **Cache**-Control **no**-store; 防止浏览器缓存。

```
#配置Nginx缓存
location /user111 {
        #禁止浏览器缓存
        add_header Cache-Control no-store;
         #启用缓存openresty_cache
        proxy_cache openresty_cache;
        #针对指定请求缓存
        #proxy_cache_methods GET;
         #设置指定请求会缓存
        proxy_cache_valid 200 304 10s;
         #最少请求1次才会缓存
        proxy_cache_min_uses 3;
        #如果并发请求，只有第1个请求会去服务器获取数据
        #proxy_cache_lock on;
        #唯一的key
        proxy_cache_key $host$uri$is_args$args;
        proxy_pass http://192.168.211.1:18082;
}
```

5、高并发秒杀公平不超卖-消息 MQ 架构体系

- 老师可以说说几种 mq 怎么选型吗？

◆ mq 的选型在课程中已经有所介绍，需要结合好几个维度进行综合考虑：目前项目的规模，公司整体开发人员的技术实力，mq 学习的成本，后期维护的成本。Rabbitmq 目前使用的比较多，Rabbitmq 学习成本比较低，资料比较多，可以快速的上手，并且 Rabbitmq 也提供了良好的管理界面可以方便的对 Rabbitmq 进行维护，因此只要是一般的项目我们都可以去使用 Rabbitmq。RocketMq 学习资料比较少，学习成本比较大，如果公司的技术实力比较强，可以考虑选择。Kafka 主要用于大数据领域进行日志数据的采集，吞吐量最大，常常和 logstash 以及 ElasticSeatch 进行配合使用。

- rocket 处理方式是什么样的

◆ MQ 的相关的产品比较多，课程中把所有的 MQ 的产品都讲解一遍不太现实。MQ 产品的功能都是比较类似，使用场景也是比较类似。通过课程中的讲解我们需要提取出一些思想，在使用 mq 的时候可能存在的问题：

◆ 1、消息的丢失问题

◆ 2、消息的重复消费

- 3、消息的顺序性保障
- 4、消息队列的高可用

关于 RocketMQ 大家可查考一些资料自行学习一下。

- 安装服务的时候，为什么不能直接配置 mysql 而是挂载
- 一般情况下在使用 mysql 容器的时候我们需要做两种目录映射：1、配置文件的目录映射 2、数据目录的映射
  好处 1：后期要对 mysql 的配置文件进行修改的时候，我们不需要进入到 mysql 容器中修改，直接修改宿主机中的配置文件重启 mysql 即可。
- 好处 2：后期如果 mysql 容器因为某些原因启动不起来了，不至于我们的数据丢失，我们可以在启动一个 mysql 容器，做对应的目录映射即可。
- 什么时候可以讲解一下 pulsar 这个未来的中间件
- 这个技术需要做一下市场调研，看看市场占有率以及口碑。后续根据实际的调研情况，看看是否需要加入到课程中。您可以及时关注传智的课程更新，如果有该技术的研发资源，可以向班主任老师索取。

- 集群 rabbitmq 在内存节点宕机之后怎么实现数据不丢失
- 内存节点将所有的队列，交换机，绑定关系、用户、权限和 vhost 的元数据定义都存储在内存中。当该节点出现宕机以后，所有的数据都会丢失。如果想在该节点宕机以后还可以继续使用这些信息，可以配置镜像队列，保障该队列的其他镜像所在的节点的类型为磁盘节点。

- Lazy Queue 我看老师上课的时候演示失败了，后面找到原因了吗？
- 在课堂上以及演示出了正确的结果，报错的原因是因为在 Rabbitmq 服务端已经存在了该队列信息了，而我们在启动服务的时候又需要去声明队列，而本次队列的特性和之前有所不同因此就报错了。可以先删除 Rabbitmq 服务端的队列信息，然后在启动服务即可。Lazy Queue 和持久化消息是最佳的排挡

6、电商终极搜索的深入优化使用

- db 到 es 做数据同步的时候可以使用 canal 替换 logstash 吗？
- 可以。Canal 是阿里巴巴开源的一款数据库日志增量解析组件，工作原理就是监听 binlog；
  发送：Canal----dump---->MySql
  推送：MySql---Binlog---Canal
  过程不一样、目的是一样的。
- 如果业务存在敏感信息，需要加密存储，但又需要支持模糊查询，是否有好的方案?例如手机号等用户信息

Es 搜索不会有这样的场景

上面的场景应该说的关系型数据库，因为敏感信息是不往 es 同步的

常用的加密与查询方案

1、加密后正常查询（非模糊）

客户端和服务端约定加密算法，比如 SHA256，客户端查询的时候还是正常的输入手机号

只不过在应用层加密了，然后在去 DB 查询

2、 加密后模糊查询（冷门）

分段(多列)加密存储：比如手机号 138 1234 5678

核心部分就是 1234 ，常用保密格式 138 **** 5678

思路

138 列 1 加密存储

1234 列 2 加密存储 ；通过字符串距离算法再次将 1234 分段比如 123、或者 234 存储

5678 列 3 加密存储

浏览器输入 123、234、1234 的时候去匹配，这样的话，就可以将数据查询出来

弊端：不能跨段查询

3、或者做的好点就做一个表，专门扩展存储加密和明文的映射。

查询的时候到映射表中查，权限做到最高，对研发不可见即可

4、还有就是 mysql 自带一些机密函数；可以支持双向加密解密 AES_ENCRYPT()与 AES_DECRYPT()

- 老师可以说说 sole.es 技术选型吗？

首先；es、solr 都是基于 Lucene 的；都是当前最流行的搜索应用服务器

如果基于分布式、集群、学习成本、功能、性能、社区活跃度角度去讲，建议选择 es

理由如下

1、易于安装和配置，学习和使用成本较低（ElasticSearch 入门简单，只要有数据库和编程知识，solr 略复杂）；

2、支持单机也支持分布式，内置了分布式组件，降低了学习和使用成本（Solr 通过 Apache ZooKeeper 实现分布式，需要额外的学习成本）；

3、除了搜索之外，ElasticSearch 还支持实时的过滤、分析、统计功能，可以为我们后续的功能扩展提供支持；

4、在创建索引的同时进行搜索，ElasticSearch 比 Solr 更优，而我们的场景需要实时的创建索引（Solr 创建索引的时候会堵塞 IO）；

5、随着数据量增加 ElasticSearch 无明显的性能损失（Solr 会明显变慢）

6、社区活跃的 es 高于 solr

- 优化点？

◆ 集群策略与参数、节点数量、分片策略（主分片、副分片）与数量、内存（避免内存交换、文件系统缓存大小、jvm 内存大小）、磁盘（大小、警戒水位、SSD）、GC 调优、Refresh 间隔设置、指定路由

es 的分词器，语言处理器如何自定义

读分词

写分词

"analyzer": "ik_smart"

- 纠错的语料库和自动不全的预料库是共用一个的吗？2.能自动补全+纠错一起处理吗

就是使用同一个预料库

不能一起处理，是先补全后纠错；原因如下

1. 补全是正式搜索之前的一个前缀匹配查询
2. 补全后点击搜索，如果词项有问题，在进行纠错（同时也查询出来了纠错成功后的数据，可参考京东）

- filter source -> "message" 然后 remove 这个这样的配置有什么意义？2.为什么不能直接配置 mysql 而是挂载 mysql

Message 存储每个文档数据

将 messge 给了 source，为了节省空间，索引 remove

mysql：说的是那个 mysql-conector-java 驱动把？配置 mysql 没用

，因为我们操作的是 logstash，logstash 需要连接 mysql

既然要连接 mysql 就需要给 logstash 挂载这个驱动

- es 作删除同步的 sql 语句 statement 该如何配置，还是在别的地方告知 es 做删除操作

在下游业务方法内，删除关系型数据的地方调用远程 restAPI 删除 es 数据

- 目前讲解的怎么使用，能否深入讲解下原理

Es 从宏观角度来看分两大部分

1、 es 读取

2、 Es 写入

这些在 es 课程基础的时候讲解的，架构师不涉及这块内容，如果有兴趣，我可以整理下资料给大家看看

- 一个 es 如何一边处理业务数据一边处理 elk 中的日志数据呢

肯定是多个索引

1、业务索引

2、日志索引

- es 能否实现对多表的关联查询,不是局限于对单表的查询

没有表这个概念（索引、文档）

GET index1index2/_search

建议在数据写入 ES 时进行数据扁平化处理，通过适当的数据冗余来避免跨索引查询，空间换性能

Mysql：产品表+产品明细

处理后

和并到 ES 一个 index

- 搜索中台化的数据都要以 DSL 的格式传输过来，那不意味着前端需要掌握 kibanna 的语法，现阶段我们都还是以 json 的格式传输数据，现在大企业里面传输数据就是以 DSL 传输的吗？

是一样的，因为 DSL 语言它也是标准的 json 格式

和我们现在传送的 json 格式参数是一模一样的，只是他有语法

即使不使用 dsl 作为参数，你也要学会 DSL 语法，因为你写完一个查询，是不是需要去 kibana 去验证下你

写的查询结果对不对？这个时候也要去查询 dsl 怎么去写，看到执行结果后；然后在验证你的接口数据是不是对

大企业一般做的终搜都是基于 DSL 的，因为，它需要兼容下游业务千变万化的参数

我们以前都会对于 DSL 前台参数 API 做一次封装，下游业务在传递参数的时候都是以填空的形式进行传递

反之，如果你只是做项目，不顾及前台参数变化后台也要跟着变，也可以使用非 DSL 的形式

- 2.logstash 怎么做到类似 cannal 的效果？3.ELK 后面会有介绍如何使用吗？

都可以做全量与增量，过程不一样，但是最终效果是一样的

Logstash 更新 查询和全量在课程中已经讲解

删除见上面的解答

后面应该没有 ELK 的课程

- 为什么不用 ElasticsearchRestTemplate，从源码上看它也是封装的 RestHighLevelClient

  风险很高（升级带来的兼容问题）

  ElasticsearchRestTemplate 是 spring-data-elasticsearch 项目中的一个类

  在新版的 spring-data-elasticsearch 中，ElasticsearchRestTemplate 代替了原来的 ElasticsearchTemplate。

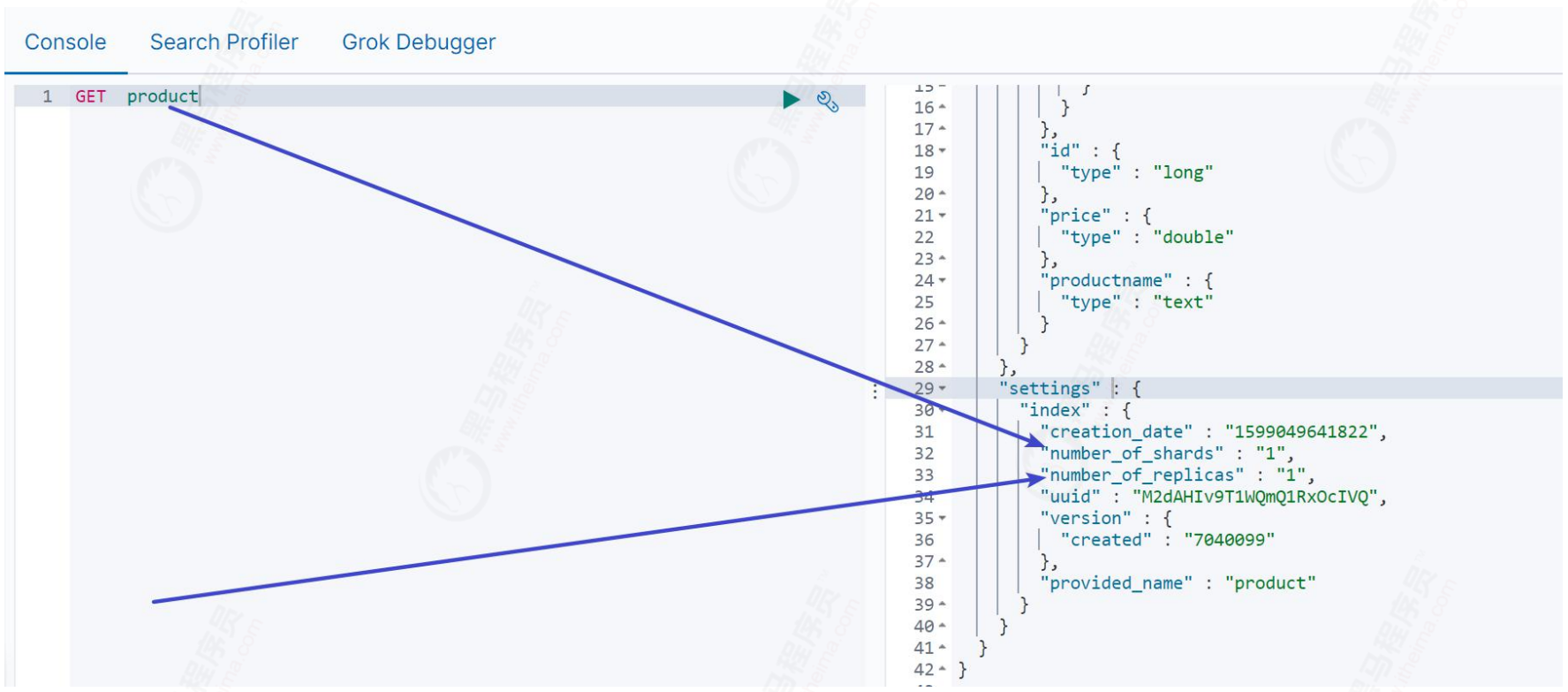
- 原因是 ElasticsearchTemplate 基于 TransportClient，TransportClient 即将在 8.x 以后的版本中移除。ES 集群如何判断某一个 index 在某一个 node 上

  ◆ 确切的说；应该是

ES 集群如何判断某一个 index 对应的分片在某一个 node 上

首先；我们第一次创建索引 index 的时候，需要指定分片信息对吧？如果不自定义主、副分片数量

默认是 1 个主分片和 1 个副分片（此处需要注意 es 版本，7.x 之前默认是 5 个副分片），如下图



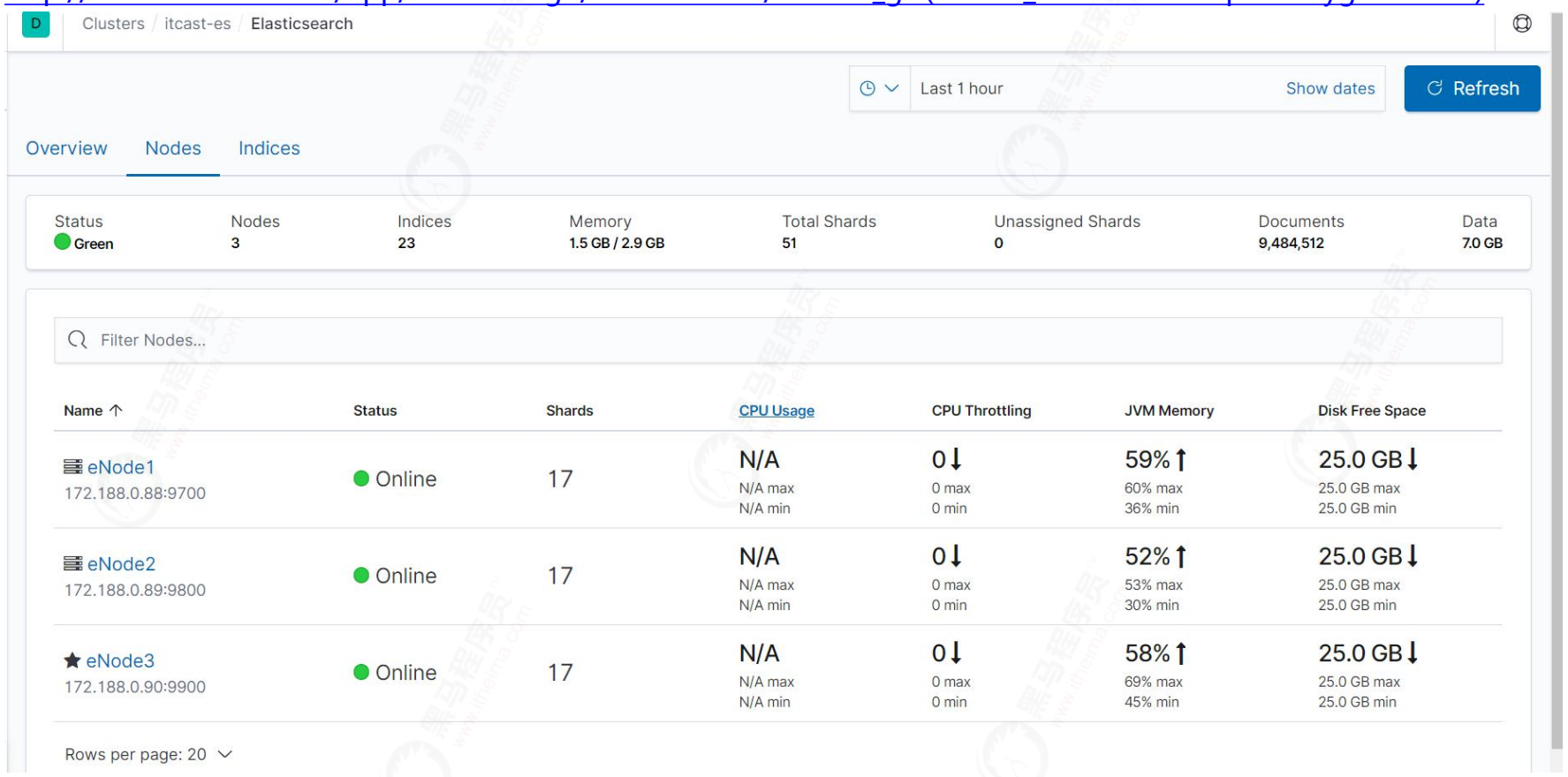
一共有三种查看方式

我们仔细往下看

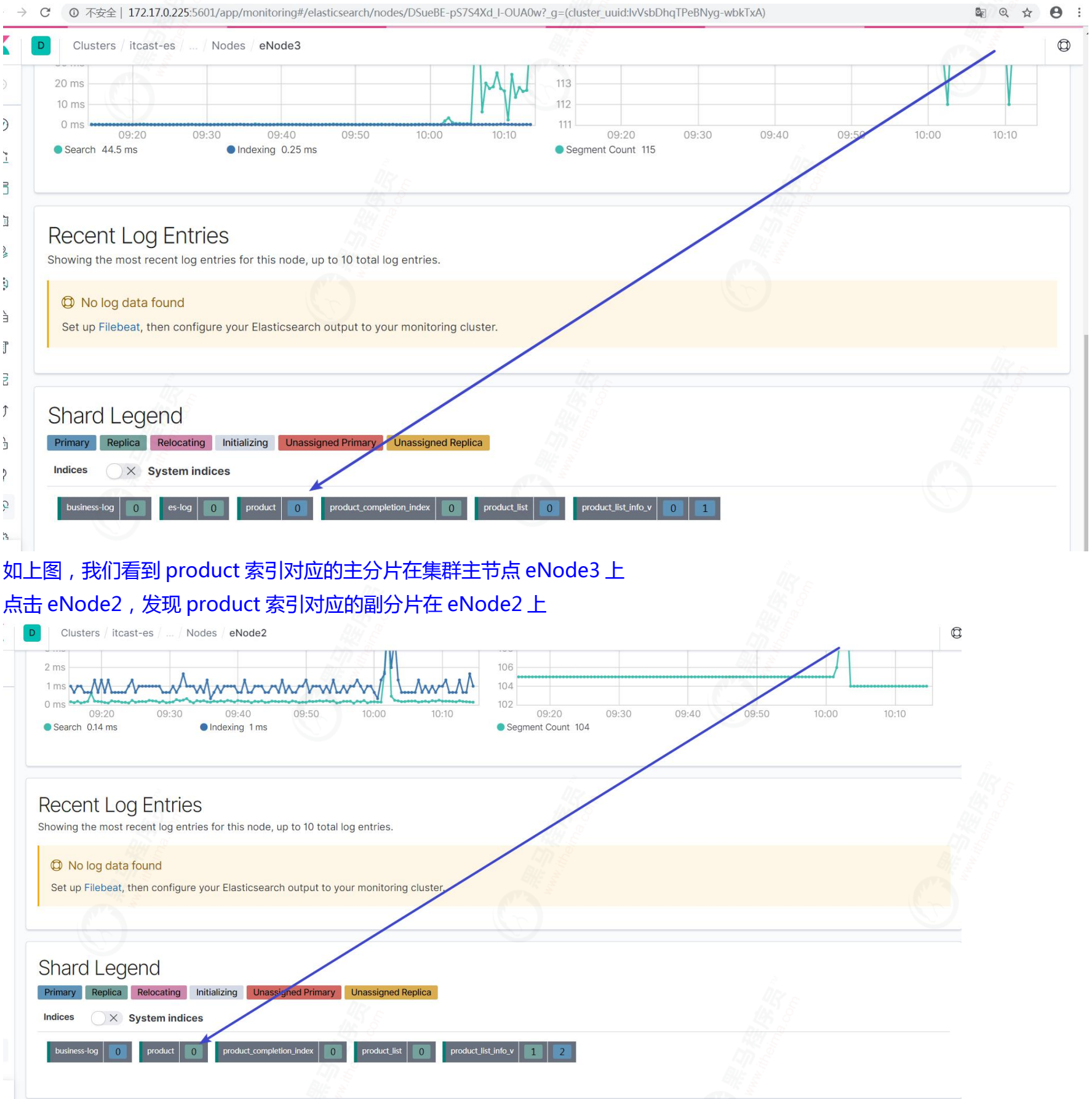
第一种查看方式

那么接下来；我们就看下这个 product 索引对应的主分片和副分片在哪个 node 上；打开 kinana

http://172.17.0.225:5601/app/monitoring#/elasticsearch/nodes?_g=(cluster_uuid:lvVsbDhqTPeBNyg-wbkTxA)



上图是我们 es 的集群节点，主节点是 eNode3；我们点击 eNode3 进入到下面的页面



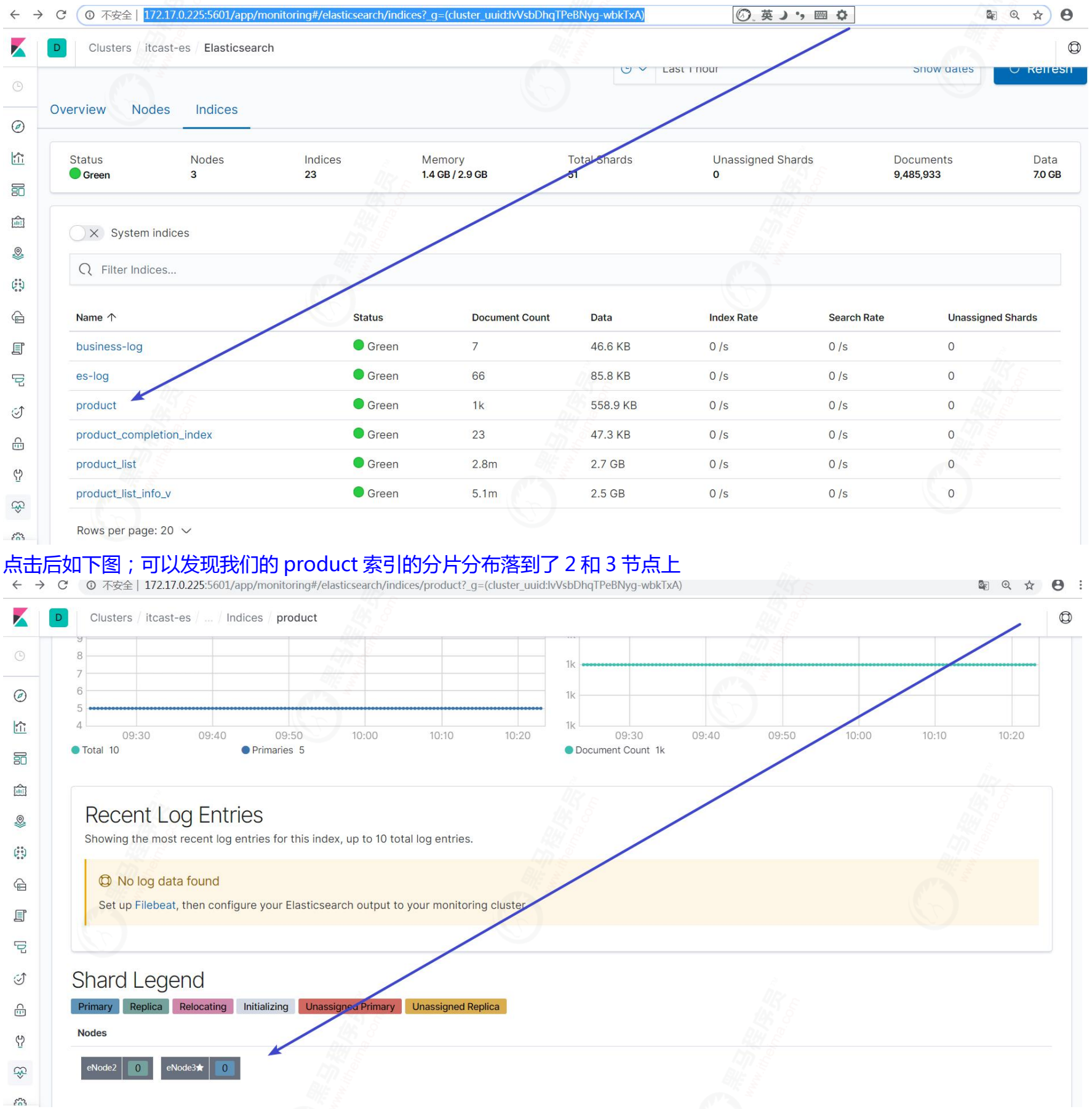
如上图，我们看到 product 索引对应的主分片在集群主节点 eNode3 上

点击 eNode2，发现 product 索引对应的副分片在 eNode2 上

第二种查看方式

打开 kibanahttp://172.17.0.225:5601/app/monitoring#/elasticsearch/indices?_g=(cluster_uuid:lvVsbDhqTPeBNyg-wbkTxA)

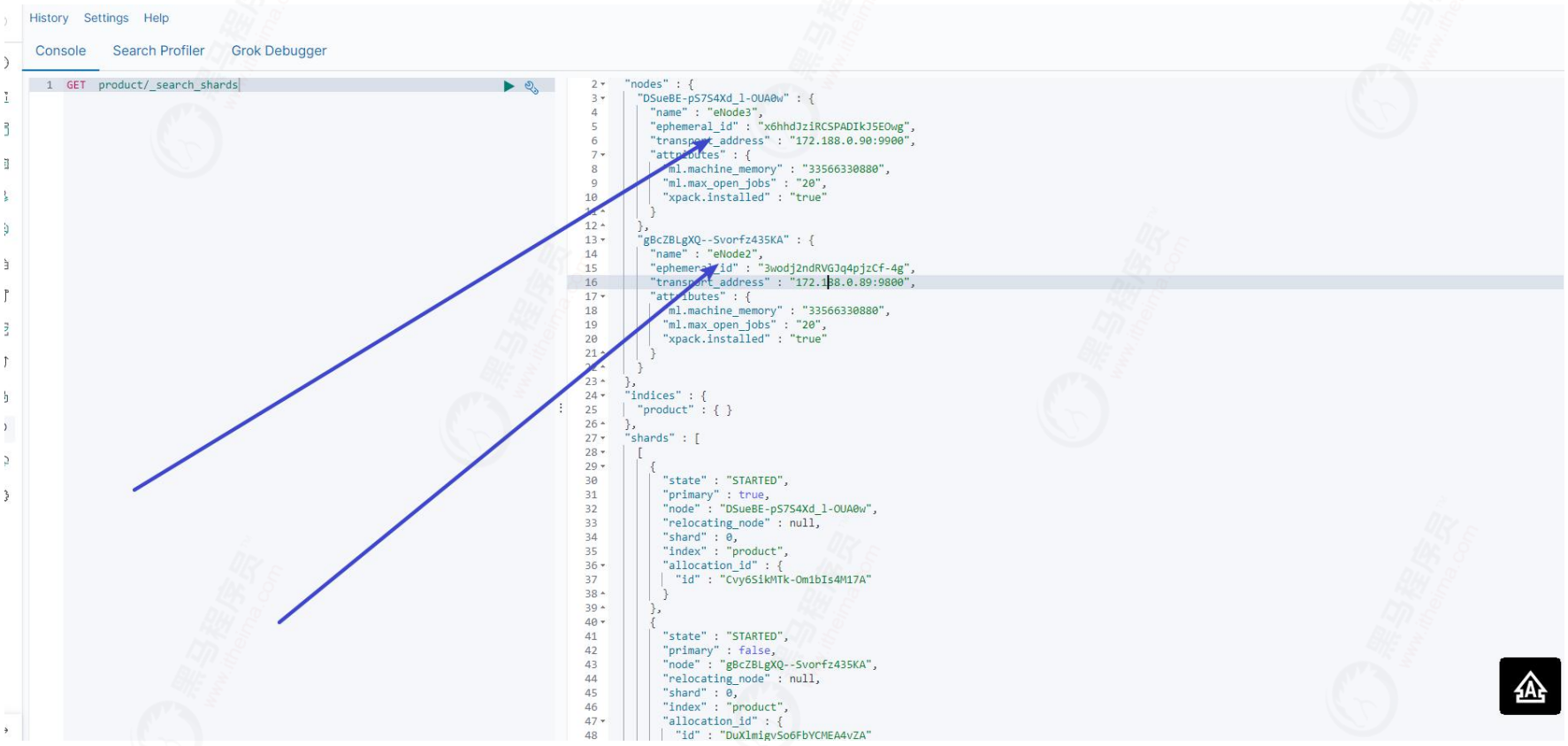
点击



点击后如下图；可以发现我们的 product 索引的分片分布落到了 2 和 3 节点上

第三种查看方式

GET product/_search_shards



我们分析下上面的执行效果

返回的 json 分两大部分

1. nodes 对应节点信息
2. 分片信息

我们详细的看下

Nodes 里面返回了 eNode3、eNode2；也就是我们的索引 product 对应的分片落到了这 2 个节点上面

那么；每个节点下有哪些分片呢

我们看下 shards 的信息

```
  "shards" : [
    [
      {
        "state" : "STARTED",
        "primary" : true,
        "node" : "DSueBE-pS7S4Xd_l-OUA0w",
        "relocating_node" : null,
        "shard" : 0,
        "index" : "product",
        "allocation_id" : {
          "id" : "Cvy6SikMTk-Om1bIs4M17A"
        }
      },
```

可以通过 "node" : "DSueBE-pS7S4Xd_l-OUA0w",

这个 id 搜索下，最后我们发现 product 的主分片 0 在 eNode3 上面

副分片 0 在 eNode2 上面

-

7、多维系统下单点登录的深入讲解

- 如何对 web 端的机器登录进行限制?即只允许某一台或者某几台设备登录，IP 可能会变化，web 浏览器也很难准确取到机器的 Mac 码

```
OAuth2AccessToken oAuth2AccessToken = tokenStore.readAccessToken(token);
tokenStore.removeAccessToken(oAuth2AccessToken);
```

- 公钥+token 就可以解密的话，那这些信息被劫持了怎么办

  https, jwe，rsa 非对称加密

- 浏览器同源策略限制是什么

URL	结果	原因
http://store.company.com/dir2/other.html	同源	只有路径不同
http://store.company.com/dir/inner/another.html	同源	只有路径不同
https://store.company.com/secure.html	失败	协议不同
http://store.company.com:81/dir/etc.html	失败	端口不同 (http:// 默认端口是80)
http://news.company.com/dir/other.html	失败	主机不同

  <script>,<image>等静态资源标签， 不受同源策略影响。如果没有同源限制， 就可以通过<iframe>指向银行网站，然后通过 dom 节点获取账户密码等信息。

- SSO 应用场景：云服务应用是什么 ？

  https://developer.aliyun.com/article/726066

对于一家中型、大型和跨国企业来说，云上资源的管控需要多个岗位的员工共同参与协作，例如开发、运维、财务和审计等角色。同时企业有强合规，强监管的需求。

在云上，为了解决管理隔离，数据泄漏，审计和合规的问题，企业的身份管理员为参与协作的每个员工创建一个RAM账号，并授予相应的权限。员工使用独立的用户名、密码和多因素认证访问云服务的控制台。

看起来现状已经很美好，但是随着员工的入职、离职，且相对应在云上的的账号需要随之调整。**企业规模扩大，使用了多个云平台的服务，需要完整的在多个云平台上管理员工账号，企业和员工的账号管理成本也不断的升高。**

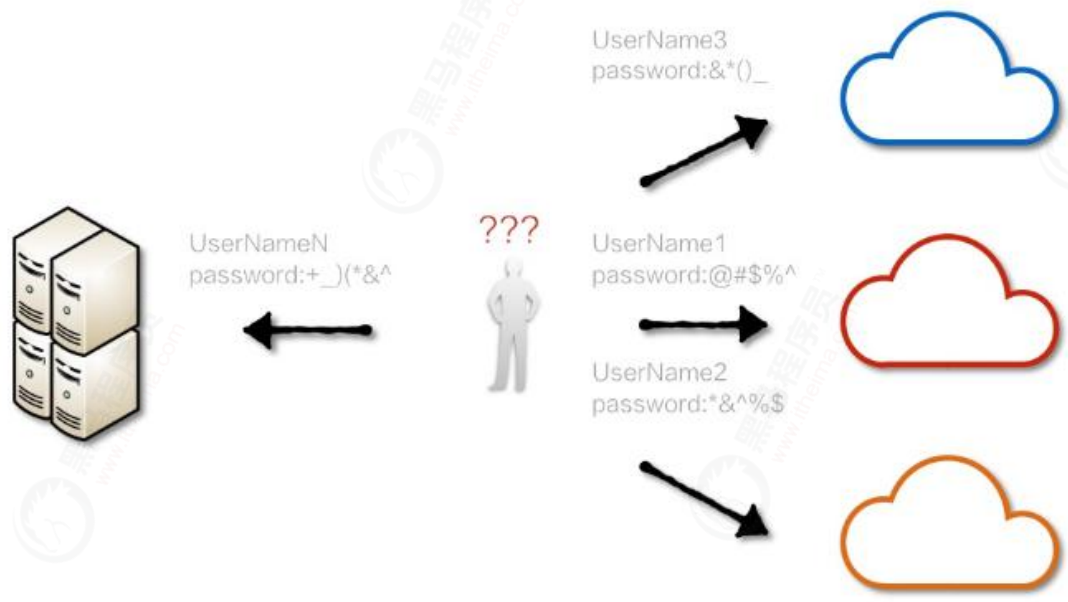


图1 本地和多云管理多套用户身份

- 用途

  做认证与鉴权

- shiro 和 security 哪个比较好

  ◆ Shiro 简洁，灵活易使用

  ◆ security 功能更多，和 Spring 无缝对接， 比如支持 spel 表达式，包含 OAuth2 支持， 也支持 Spring Social 和第三方完成社交登录；

  ◆ security 庞大社区， 丰富文档支持

- 整个都不是很理解

◆路径：设计方案 -> 技术协议 -> 核心 OAuth2 -> OAuth2 流程 -> 实践案例

◆有哪些设计方案-> 涉及哪些技术协议？协议的特征与应用场景？ -> OAuth2 特点，与其他鉴权技术的差异-> 深入 OAuth2 模式，流程，解决方案 -> 如何在项目中集成使用，哪些核心配置。