### 首页：

查询所有帖子(非拉黑，置顶，精华，发帖人id为0，每页多少第几页)+发帖人信息+自定义PageHelper复用

路径中的参数名current与方法中的属性current名字一致，可以自动注入

渲染到html上时，数据可以使用任意实体类的get方法或者属性

Controller出现了Model变量，则方法所有形参自动注入Model渲染到视图。

thymeleaf渲染的html中只要${ }中不是多级调用就不会出现空指针异常，最多没有值

### 注册：

主页->注册界面->Controller

Controller中检验输入信息是否合法，不合法重新渲染到注册界面，由于user自动注入model，所以可以回显用户上一次输入的信息

合法则插入新user，然后往用户邮箱发送激活url，url必须携带userId和激活码。

激活链接点击后修改数据库中用户状态，如果用户已经被激活过了，或者激活码不对，都返回index界面，激活成功后返回登录界面。

### 生成验证码：

springboot3以上，所有response，session，request改为jakarta包下

导入jar包，因为不是starter，所以不自动注入ioc，需要手写Configration配置类，然后@Bean返回一个DefaultProducer类，此类即为生成验证码的主类。@Bean中实例化Properties对象存储配置键值对，然后放入Config对象，最后返回DefaultProducer对象。

Controller中方法：注入DefaultProducer对象之后，使用其生成text方法生成验证码，并存入session域，然后生成image，response设置返回值类型为image/png，然后用resopnse字节流输出(实际ImageIO输出)。

login页面中，验证码src=controller，但是要给刷新验证码按钮绑定js函数，在全局js文件中var上下文路径为”/zzh”，默认省略<http://localhost:8080>，但是不省略自己定义的上下文路径。然后在login页面中js刷新函数中使用id选择器选中验证码，修改其src属性为controller，为了防止html当成静态资源不刷新验证码，需要在src后面附加一个任意变量=Math.random()

### 登录：

跳转到登陆界面，Session中存储验证码

登陆时验证账号密码验证码，合格后生成一个登录凭证LoginTicket，存储到数据库中，状态设置为0，同时存储到持久化Cookie中，response对象发送给浏览器。

以后浏览器的每次请求都会携带凭证。

退出登录时，获取request中的凭证，从数据库中找到对应的凭证，状态置为1.

### 忘记密码：

因为注册时会检测邮箱是否重复，所以邮箱可作为为此处唯一标记。

异步请求发送验证码：

使用ajax发送异步请求到服务器，向前端页面发送的email发送4位随机验证码。并将email+验证码组合形式保存到session中，然后服务器使用阿里提供的JSONObject对象设置属性和值，调用toJSONString方法输出Json格式字符串，发送到前端，前端接收json字符串后解析为json对象，根据状态码属性弹框是否成功。

用户输入验证码新密码之后，再次发送请求到服务器修改密码，服务器检测email和用户输入的验证码与session中存储的是否一致，一致后说明验证码正确。

然后user表中根据email查询user，如果查不到，说明此邮箱未注册任何用户。如果查到了，则修改此用户的密码为新密码。重定向到登陆界面。

当修改密码因为某些用户原因如邮箱二次输入不对，验证码不对，由于还是同一个请求，所以前端使用th:value=”${param.xxxx}”获取request请求服务器时携带的数据渲染到页面，以免用户麻烦输入。

虽然model设置的也是request域，但是model直接${xxx}获取即可，语法不同。

### 拦截器检测登陆状态线程容器存储user：

**注意并不是未登录就跳转到登录界面**

自定义拦截器类实现HandlerInceptor接口，使用@Compoment注入ioc。注入userService。注入本地线程ThreadLocal。

重写拦截器的pre方法拦截请求，取出cookie中的凭证，检测凭证是否有效，有效则查询出凭证具体信息，进而查询出用户信息，存储到线程容器中，这样整个请求处理过程中，也就是本次线程的任意时刻都可以使用user信息。

重写拦截器的渲染之前的方法，需要注意判断ModelAndView不为null时，才将本地线程容器中的user保存进modelAndView，从而可以在请求渲染的htlm上使用user信息。因为有些请求是不需要渲染的，可能@Responsbody返回json，此时ModelAndView对象就是null。

重写请求结束前的方法，直接在本地线程容器中将user移除掉。

然后新建配置类实现WebMvcConfig接口，标明此配置类可以配置web级别的过滤器监听器拦截器等。在配置类中注入自定义的拦截器

然后重写addInterceptor方法，使用形参的register注册对象注册自定义的拦截器，并排除掉静态资源请求不拦截。其他所有请求都需要拦截看是否有凭证且有效，保存user。即不管需不需要，都根据凭是否有效查一次user保存到本地线程容器。

### 账号设置：

头像设置：

Controller中使用MultipartFile类型接收前端的文件，如果多个文件可以使用数组类型，将文件重命名后存入盘中，并从线程容器中获取user，更新user表中headerUrl为<http://localhost:8080/user/header/>重命名文件。

任意html界面上头像的路径为表中loginUser.headerUrl

在Controller中新写一个方法拦截user/header/{文件名}，也就是表中headerUrl，根据url中文件名，new FileInputStream(“盘路径+文件名”)，读取文件，然后buffer缓冲，sign信号标志，response的输出流输出到客户端。

密码设置：

线程容器中得到user

前端发送的旧密码与user.getSalt组合然后MD5加密，与user.getPassword匹配后说明用户输入的原密码正确。

将新密码与salt使用MD5加密后更新user表中。

从cookie中得到通行证，在通行证表中将通行证状态置为1，失效，重定向到登陆界面

### 拦截器检查登录状态防止访问敏感功能并跳回登录界面：

每个请求除了检测是否含有通行证从而决定是否在线程容器中存储user或者html上显示user信息之外。为了防止用户直接在地址栏请求更改密码，更改头像这种需要用户登录后才可以访问的功能。再加一个拦截器。

此拦截器在Config中第一个拦截器之后再注册，这样就会先通过第一个拦截器检查凭证，再使用这个拦截器。

此拦截器专门拦截需要用户登录才能访问的资源，但是一个个添加url也麻烦，所以直接在Controller方法之上自定义注解，拦截器拦截url时，如果是访问Controller中方法(if handle instance of HanderMethod)，那么HanderlMapping则将Controller方法形参handle注入为HandlerMethod类型，强转handle为HandlerMethod，这个就是反射获取的方法，再反射获取自定义方法注解，查看获取的注解是否为null，如果是null，说明要访问的此Controller方法不需要登录，直接return true放行。如果不是null，说明需要登录，则在本地线程容器中获取user，user不为null放行，为null直接使用resopnse重定向到login页面，并且return false。

### 过滤器过滤用户贴中的敏感词：

存储敏感词生成树：

使用树存储敏感词，树根为空，一级存储敏感词第一个字，二级存储第二个字，以此类推。

字符串存储中文时一个字节(char)就是一个汉字。

过滤敏感词：

三个指针：树指针，字符串开始和当前指针。

开始指针指向的字符未在一级中找到字符，说明该字符不是敏感词开头。开始和当前指针到下一个字符，树指针归位root，当前字符添加到结果中。

开始指针指向的字符在一级中找到字符，说明该字符是敏感词开头，如果该字符是敏感词最后字符，则添加\*\*\*到最后结果中。

开始指针指向的字符在一级中找到字符，说明该字符是敏感词开头，如果该字符不是敏感词最后字符，则当前指针指向下一个字符，判断该字符在树的下一级节点中存在情况。

### 发布帖子：

视图层：

主页面判断当前Model中是否有user，有说明用户登录，显示发帖按钮。

获取发帖框中的标题和文本，使用AJAX以Post方式发送异步请求到服务器。

服务器接收参数后在Controller中先判断线程容器中是否有user，没有说明用户是非法在地址栏中访问此url，抛出异常。

正常后new帖子对象接收客户端发送的参数并根据user设置其他参数。

服务器new Json对象存储键值对code:”状态码”，msg:”信息”，map<String,Object>存储的其他内容，调用Json对象的toString方法输出为字符串到前端。

服务层：

service中使用敏感词过滤器过滤帖子对象中标题和文本，并将其内容进行html标签体转义，防止内容中含有html标签显示在网页上时被解析为html语言破坏网页。比如<>转义为alt&,这样网页解释alt&就是单纯显示<

持久层：

mapper插入帖子到数据库

网页：

接收服务器返回字符串的内容到data对象中，重新解析为Json对象，输出Json对象的code，msg，map属性。

### 帖子详情

index界面中已存在map，map存储每个帖子及其作者。

视图层：

index中获取帖子id，发送到Controller，查询到帖子详情，得到userId后分步查询帖子作者。

存储帖子和作者到Model中返回前端。

服务层：

postId查询post，userId查询user。

持久层：

postId查询post，userId查询user。

### 显示评论

数据库：针对帖子的评论和针对评论的回复，针对某个用户回复的回复，都是一张表

前端携带帖子id

服务器端：

①查询出所有针对帖子的评论到List1<Map>,每个map存放评论，发表评论的用户信息。

②新建List2<Map>，每个map中存放List1<Map>中每个评论的回复(包括该评论下回复的回复，因为即使是针对某用户回复的回复，也是针对于该评论的。

),发表回复的用户信息，回复针对于某个用户的该用户的信息。

③将循环产生的多个List2放入List1<Map>的map中。

④渲染到前端

### 添加评论

无论是添加针对帖子的评论还是针对评论的回复还是针对某用户回复的回复，都是添加到通一张表

如果是添加帖子的评论，前端携带帖子id，评论内容

如果是添加评论的回复，前端携带评论id，回复内容

如果是添加回复的回复，前端携带评论id，回复针对的目标用户id，回复内容

### 私信列表

展示此用户与其他用户之间的最新一条会话。

sql：按照会话id分组，聚合函数找出每组的最大id（每个会话最新一条消息），order by 再次每个会话最新一条倒序排序

点某个会话，展示此用户与某用户之间的所有私信。

### 发送私信，设置已读

前端使用ajax异步发送post请求，后端插入数据库。

在每次点进某会话时，将此会话的未读信息设置为已读，需判断未读信息是否是自己发送的，自己发送的不算。

### 统一处理异常

在类上加入@ControllerAdvice（annotation=”Controller.class”）注解表示该类是控制器通知类，作用于所有加入@Controller注解的类

在新方法上加入@HandlerException({Exception.class})表示，当controller中发生异常时，则会在@ControllerAdvice类中调用@HandlerException方法。来处理异常。

@HandlerException方法形参可以自动注入一些常用参数，如使用Exception e记录错误信息，request和response对象返回前端。

方法中使用logger记录错误日志，然后判断请求类型，如果是异步请求，则一般是向前端返回json，使用response得到输出流向前端返回普通字符串即可，前端$.parseJson再转为json对象。如果是同步请求，则response重定向至错误页面。

### 统一记录日志

AOP，非侵入式，作用于方法上的切面（如目标类继承了接口则代理模式，无接口则实例化其子类）

作用如我们可以在任意service类中方法被调用时记录一下是哪些用户在什么时间调用的做一个记录。其中service中的这些方法就是切入点，记录逻辑这些手写的提取出来的方法叫做切面。

在类上加入@Compontent，@Aspect注解，标明这是一个切面类。

在任意方法上使用@Pointcut(“excution(\* 包路径.\*.\*(..))”)

// 第一个\* ：所有返回值类型  
// service后面的第一个\*表示该包下的所有service类  
// service后面的第二个\*表示所有service类中的所有方法  
// (..)表示方法的所有参数类型

在任意方法上使用@Before(“使用了@Pointcut注解的方法名带括号”)，表示这个方法会在切入点方法执行之前被执行。且此方法形参可以自动注入JoinPoint，该对象就是切入点方法。

同理还有@After，@AfterRuturning，@AfterThrowing，@Around，其中异常通知如果执行了，说明切入点方法抛异常，则@AfterRuturning不会被执行。

### Redis整合

Redis事务：Mysql事务实际对数据库修改，查询也事实查询，有问题再使用Undo日志回滚。但是Redis事务中所有命令全部先存命令队列，提交事务之后，所有命令全部执行一遍没有错误，才会对数据库做实际修改。所以在Redis的事务中，如果数据库原本是空的，先增加一条数据再查询数据，那么Redis先表面上增加数据（因为事务没有结束，所以此次增加不会对数据库产生实际影响），然后查询，此时如果输出这个查询，就会显示空值。直到这两条命令全部执行完毕，此时事务结束，事务提交，此后的查询操作才能查到新增加的这条数据。

所以Redis的声明式事务不被使用一般，因为声明式事务（加注解）只能作用在方法上，而一个方法中就可能会有查询操作。

所以改用编程式事务。

pom引入jar包，properties配置使用几号数据库，Redis的ip和端口

新建配置类，自定义RedisTemplate这个Bean替换springboot容器中旧的bean。

并且在方法中设置链接对象，Config类中自定义的@Bean方法形参会被SpringBoot自动注入

，所以可以把连接对象注入到方法中，然后设置进去。设置普通键值对和哈希键值对序列化方式，一般键序列化为String，值序列化为json。

### 点赞

在Reids中以Set形式存储键：like+实体类型+实体id，实体类型即为帖子或者评论等。

值：userId，统计点赞数量时可以统计userId有多少。

如果以后要开发踩的功能，那么可以存储新的set，键：cai+实体类型+实体id。

查看用户是否点过赞，使用redisTemplate.opsForSet.isMember即可。

查看点赞数量，使用redisTemplate.opsForSet.size。

点赞是异步请求，如果用户点赞过了，则在Redis中移除此用户即可。

用户每次点赞都向前端返回两个值，即用户是否已经点赞（0代表未赞，1代表已赞），实体点赞数量。

### 我收到的赞

在Reids中以键值对形式存储，键：like：user：userId，值就是数量

点赞帖子，评论回复时，重构Controller，在Controller中，获取这些实体的发布者用户id作为key，向redis中increment+1，取消点赞则decrement。

前端点赞时也需要重构，携带userId。

### 关注，取消关注

异步请求。

用户关注某实体时，同时向redis存两分数据

①该用户关注的实体--reids中的key为followee:userId:entityType,值为entityId。

②该实体被哪些用户关注—redis中的key为follow:entityType:entityId，值为userid。

关注/取关之后重新刷新profile页面，此时服务器端从redis中新增获取三个数据：

①该用户被多少人关注

②该用户关注了多少人

③登录用户是否关注了该用户，用于按钮的变化。

### 关注/粉丝列表

关注和粉丝列表的逻辑大体一致。

进入用户个人主页，点关注数，发送userId到服务器。

从redis中查询出关注列表（此时查询出的Set为关注的人的id集合），再each从redis查询出其分数（按照此倒序并转为关注时间）和mysql中的user存入map。

controller中遍历map，从redis中查询出登录用户对关注列表中所有人的关注状态。

渲染到关注页面。此页面引入了profile.js，所以关注和已关注按钮可以根据关注状态改变，取关操作后，js中reload刷新当前页面。

分页显示。

### Reids优化其他模块

验证码：放入Session域，保存半小时且浪费服务器内存。事实通常几分钟内过期。

登陆凭证：每个请求都会验证，所以及其频繁，废除sql表，且退出登录不能删除凭证，凭证可以作为记录在其他功能使用，如按照凭证日期，筛选出用户登录天数，给予奖励。

User缓存：查询user时先在redis中查找，没有再去sql找并加入redis。修改user时先更新sql中的user再删除redis中的缓存，更新失败redis中还有缓存。

为什么不更新缓存而是删除缓存：redis虽然是单线程，这个单线程指的是所有指令排队执行罢了。而redis命令与业务逻辑（如数据库命令）结合在一起时就是一个多线程并发操作，比如某账号被AB两个人同时使用，A修改头像更新sql，此时B也修改头像更新sql，B又抢到时间片，B更新缓存，然后A更新缓存，这时sql中是B的数据，但是redis中是A的数据，就出现数据不一致问题，在B那里就会显示A的缓存。

### 消息队列

BlockingQueue为一种阻塞队列，其put（生产）和take（消费）方法相当于内置了检测队列中数据是否为满或者空，满或者空的话就wait，不满不空就notify唤醒对方。

### Kafka

#### （cmd了解）

cmd窗口

d:

cd D:\IDEADevInstall\kafka\_2.13-3.5.0

bin\windows\zookeeper-server-start.bat config\zookeeper.properties（启动zookeeper）

新建cmd窗口

d:

cd D:\IDEADevInstall\kafka\_2.13-3.5.0

bin\windows\kafka-server-start.bat server.properties（启动kafka）

新建cmd窗口

d:

cd D:\IDEADevInstall\kafka\_2.13-3.5.0\bin\windows

kafka-topics.bat --create --bootstrap-server localhost:9092 --replication-factor 1 --partitions 1 --topic test(创建主题，在哪个kafka服务器上，副本多少个，分区多少个，主题名称)

kafka-topic.bat --list --bootstrap-server localhost:9092(查看9092kafka服务器上的所有主题列表)

kafka-console-producer.bat --broker-list localhost:9092 --topic test(list标明后面可以跟多个kafka端口，往kafka集群的某个主题下发送消息)

新建cmd窗口

d:

cd D:\IDEADevInstall\kafka\_2.13-3.5.0\bin\windows

kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic test --from-beginning(从哪个kafka服务器中哪个主题上获取消息)

即生产者可以往多个kafka服务器发送消息，消费者可以监听某kafka服务器的消息。

#### 流程

**主题Topic：**

主题即为存储消息的队列。一个主题可以分为多个partation（分区），所以实际上一个分区就是一个队列，一个主题可以包含一个或多个队列。一台服务器可以有一个或多个主题。

**集群：**

在生产环境中，为了提高吞吐量，一个主题中的多个分区可以分布在不同的机器上（kafka服务器）。

**生产者：**

生产的数据必须指定一个主题，但是发送到主题上的哪个分区可以自行指定或者由kafka轮询或其他算法自动指定。如果分区是在另一台服务器上也可以。

**消费者组：**

同一组内的一个消费者消费一个分区的数据，每个组都会试图消费所有分区的数据，这就是广播。

现有组1和组2：

①如果消费者1和2在组1，默认情况下，分区1和2都有数据，则同一时间内消费者1消费分区1，消费者2消费分区2.

②如果只有消费者1在组1，分区1和2都有数据，则消费者1消费分区1数据之后，空闲，kafka自动均衡使得消费者1再去消费分区2的数据。除非自行指定消费者1只消费分区1的数据，那么它将不再去消费分区2的数据。

③主题现有P1，2，3，4四个分区，分区1和2在Server1服务器上，分区34在S2上。消费者c1和c2在组1，c3，4，5，6在组2。现在假设生产者生产了足够多的数据给了四个分区。则：

组1的c1消费p1和p2，组1的c2消费p3和p4.

组2的c3消费p1，c4消费p2，c5消费p3，c6消费p4

这样即为最佳zookeeper集群划分方式。

**特别的：**

同一组中的c1和c2，可以消费主题1中的p1和主题2中的p2.即对于消费者来说，只能消费一个分区的消息，但是不限制消费哪个主题的。

### 发送系统通知

发送的消息封装到对象Event中（主题，事件触发者，操作的实体类型，实体id，实体创造者id，以及一个map用来存放其他数据）。

生产者EventProducer对象中的发送方法负责发送event对象。

消费者EventConsumer对象中的消费方法负责监听主题中的event对象并将其属性转化到Message中，然后将message插入到数据库中。（map中的所有数据都转为content字段）。

在评论，点赞，关注的Controller中，加入生产者对象，将必要的数据封装到event，发送至某主题。

消费者自动监听消息队列中的event。

在自定义的aspect类中，我们将所有service中的方法都设置成了切入点，并记录调用此service方法的请求，以此来获取用户的ip地址，但是消费者不需要用户调用，而是Spring调用，所以，获取请求如果为空，可直接return不记录日志。

### 显示系统通知

对于登录的用户，从表中查询出三个主题（conversation\_id字段）中from\_id=1，to\_id=登录用户id且status！=2未删除的最新一条message，并将message中的某些字段及content用JSONObject.parseObject转为map，从map中得到一些必须字段如entityType，用于判断触发用户触发的实体是登录用户的帖子还是评论，以便于在前端显示不一样的字体。

通知详情页面，则针对某个主题查询出所有未被删除消息，同样JSONObject将content转map，获取一些必要字段。

并且还要将此页的message的id得到，遍历如果发现有未读的，需要设置status为1已读状态。

### ES

#### 基础

版本与pom文件中保持一致。

config目录下的elasticsearch.yml中修改cluster.name:为集群名。path.data为存放被搜索的数据备份路径，path.logs为日志路径。

配置环境变量。D:\IDEADevInstall\elasticsearch-8.5.1\bin

安装中文分词插件ik，因为英文一个句子中每个单词之间都有空格，中文没有，而是需要热门词汇拆分。且必须将ik解压至D:\IDEADevInstall\elasticsearch-8.5.1\plugins\ik

在ik的config目录下，dic后缀结尾的就是词典，里面就是各种词汇。

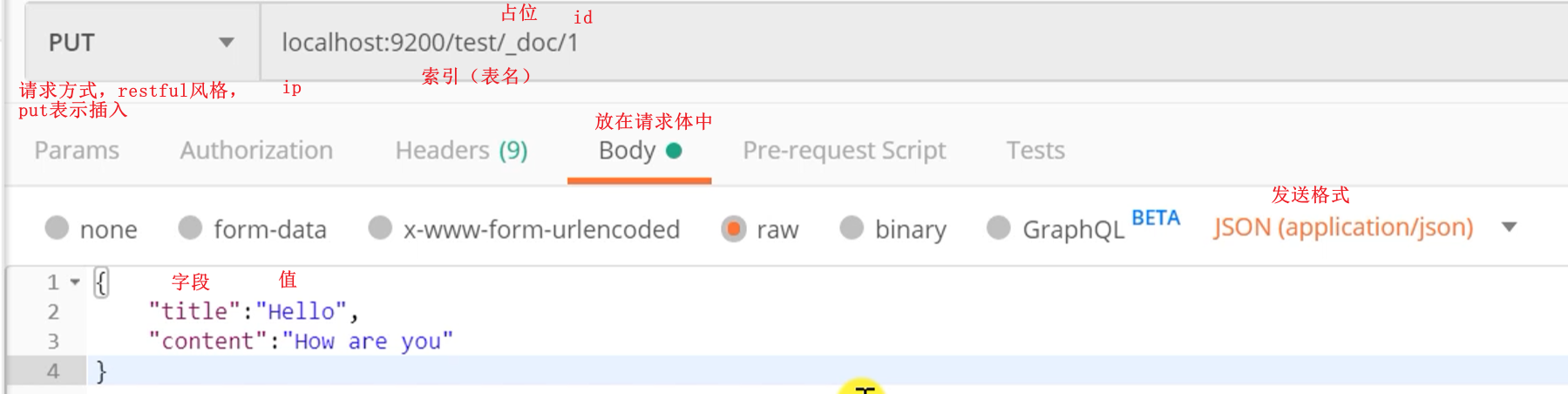
可以新建自己的dic字典（后缀是\_dict），然后在IKAnalyzer.cfg.xml中的entry标签手动添加dic。

bin下的bat启动es。

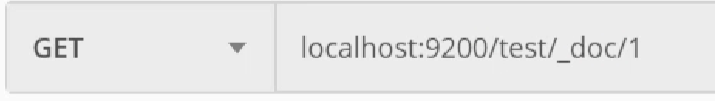
端口http形式访问默认为9200，TCP形式默认为9300

**记得关闭SSL安全认证：即关闭账号密码，**elasticsearch.yml，里面有2个true改成false

**增**



**查**



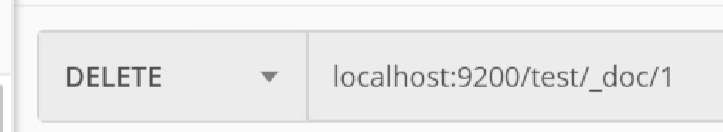
查询全部



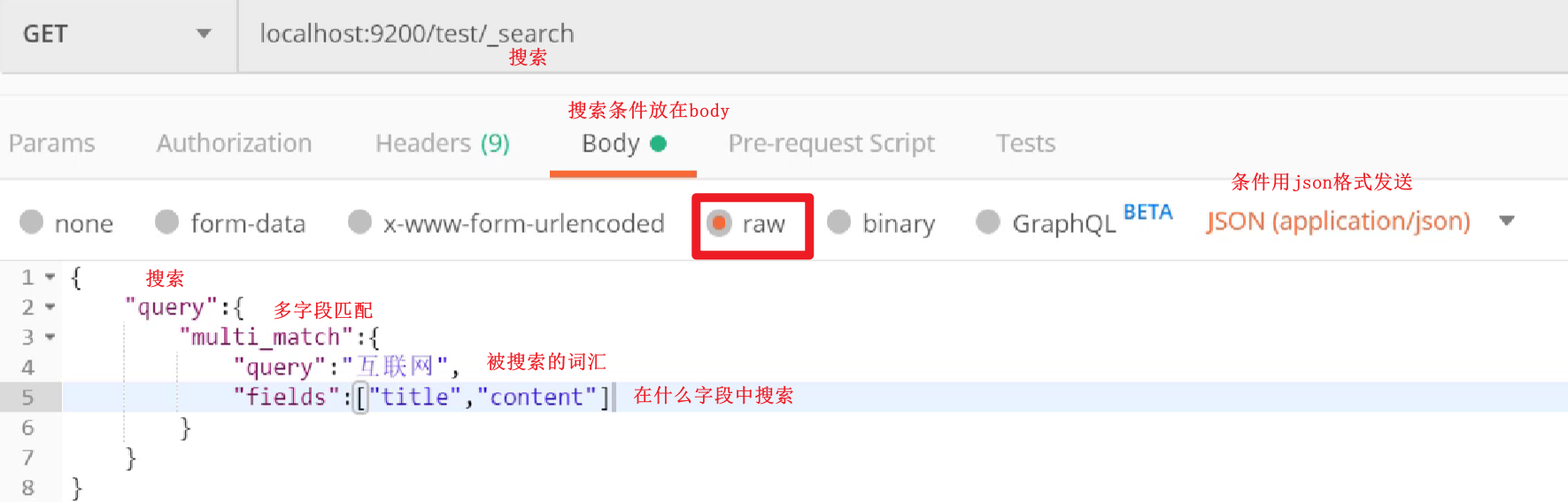
**改**

依旧是put，只将值修改一下即可，覆盖原值（先删后加）。

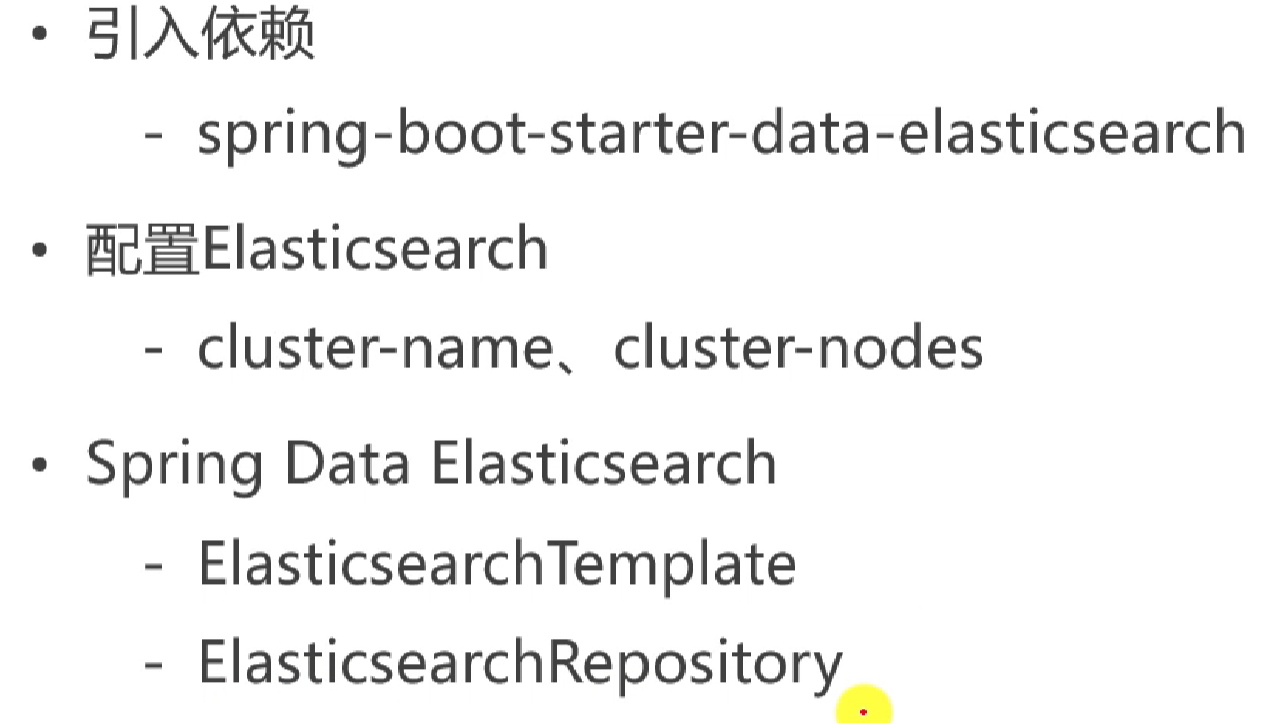
**删**



**搜索**



#### Spring整合ES



解决与redis共享netty问题。

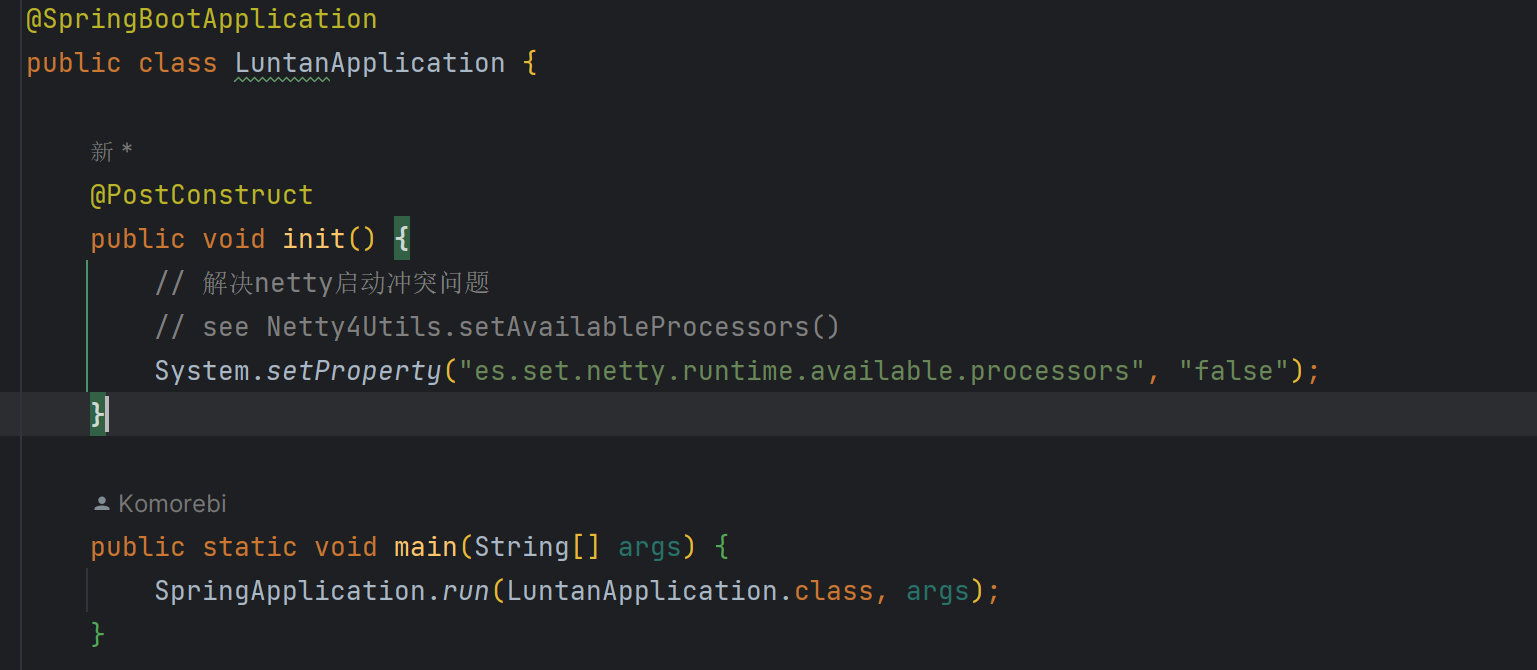
在主启动类luntanApplication（程序入口，也是最早启动的配置类）中，使用@PostConstruct注解表示在主启动类构造器实例化后，就解决此问题。

构造一个方法，在方法中设置System.setProperty("es.set.netty.runtime.available.processors","false")。

### 开发社区搜索功能

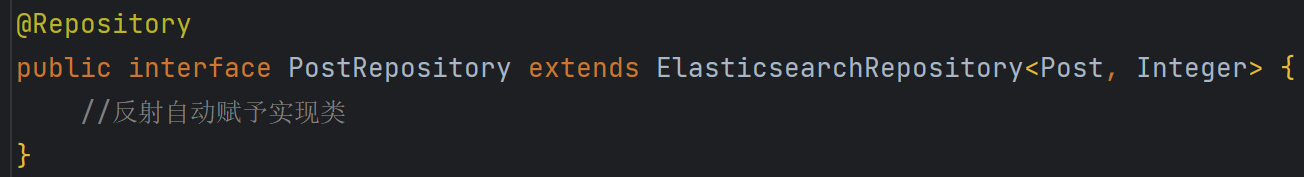
#### 前置准备：

引入pom，properties中配置端口为9200，主启动类中解决redis与es冲突



#### dao层，操作es服务器：

①使用@Repository注解，新建接口postRepository继承于ElasticsearchRepository<Post, Integer>接口



②在SearchService中还需要注入ElasticsearchTemplate。

#### Elasticsearch Service

声明三个方法

①往es服务器中插入/更新一条数据，使用postRepository.save（Post post）

②删除一条数据从es，使用postRepository.deleteById（int postId）

③分页搜索

构造高亮条件对象

构造标准Criteria条件对象（从什么类型的实体的哪些属性中搜索）

构造完整搜索条件，实例化criteriaQueryBuilder对象传入Criteria条件对象，使用withSort方法构造排序条件，使用withPageable方法构造分页条件，使用withHighlightQuery方法传入构造高亮条件对象构造高亮条件。

传入criteriaQueryBuilder实例化criteriaQuery对象。

使用elasticsearchTemplate.search（）传入criteriaQuery对象进行查询，并得到post结果集，命名为result。

使用result. getSearchHits()得到搜索后的post以及高亮字段的集合searchHits。

此时searchHits中的post的被搜索字段并没有高亮，需要使用searchHits.getHighlightField("title/content").get(0)获取第一个高亮字段的标题/content即可，无需获取所有高亮字段。

获取后post.setTitle/setContent将含有高亮字段的高亮标题/content覆盖即可。

将这些post循环装入新的list。

因为分页需要用到page信息，如总条数在result中就有，所以返回new PageImpl<>(postList, PageRequest.of(current, limit), result.getTotalHits())，

#### Post/Comment/SearchController

新增帖子时，需要往es服务器新增帖子。

评论帖子时，因为修改了数据库中帖子表的commentCount字段，所以也需要更新es中的帖子。

这两种情况都可以使用elasticsearchTemplate.save（post）方法操作。

在postController的发布帖子中，将帖子存储到mysql后，使用kafka发送event事件到消息队列，topic定义为publish。

event存储topic，userId（当前用户），entityType（帖子类型==1），entityId（帖子id）

同理commentController中，发布评论操作中会更新mysql中post表的commentCount字段，判断是针对帖子发布的评论后，再发送event事件到消息队列，topic仍然是publish。

event存储topic，userId（当前用户），entityType（帖子类型==1），entityId（帖子id）

消费者eventConsumer中，监听主题publish，将event中的entityId(postId)取出，从数据库中查询出post，elasticsearchTemplate.save（post）更新es中的此帖子。

searchController负责响应前端的搜索get请求，将搜索关键字携带于请求头中，注意pageHelper.setRows（）需要在搜索获取page对象后设置（page.getTotalElements()）。

### SpringSecurity

#### csrf

csrf：主要用于提交表单之类的重要数据的请求防止被劫持

每次请求，过滤器中会给请求addAttribute(\_csrf对象)。

如果html中有表单，则访问该html的resopnse经过过滤器时，会给表单中增加一个隐藏输入框，内容就是\_csrf.token，用户提交表单的请求会携带此token与session中的token核对。

如果html没有表单，或者有表单但是我们不用表单提交方式，说明我们要做异步请求了。那么我们应该手动在请求中添加token信息。我们在将要发送异步请求的页面，加入两个meta标签，一个标签value设置为\_csrf.token，一个设置为\_csrf.headerName（从服务器得到此页面时这两个数据会被渲染填充）然后从此页面发送的异步请求，在ajax中使请求头携带这两个值，以键值对形式存在，此时服务器session中就可以核对数据了。

**Security会拦截所有PATCH，POST，PUT和DELETE方法的请求，我们在项目中的ajax异步请求基本都是post请求，所以我们必须在所有触发异步请求的界面上加上这两个meta标签，并且在ajax中将这两个标签当作键值对设置到请求头中，Securuty才可以正常核对。**

**即要么在form表单中提交隐藏框的token到请求体给Security校对，要么自定义键值对（从\_csrf中获取）放到请求头中给Security校对.**

#### SecurityContextHolder

通过 (SecurityContextRepository)repo.loadContext() 方法从请求Session中获取

SecurityContext（Security 上下文 ，类似 ApplicaitonContext ） 对象

SecurityContextHolder.setContext() 将 SecurityContext 对象放入

SecurityContextHolder进行管理（SecurityContextHolder默认使用ThreadLocal

策略来存储认证信息）

最后通过 SecurityContextHolder.clearContext() 将 SecurityContext 对象 从

SecurityContextHolder中清除

本人发布的详解：对于是否应该clearSecurityContentHolder这件事：（SecurityContext可以看作holder中的一个属性）

如果使用了Security配置类的认证框架，用户在登陆后会自动将用户信息存储到SecurityContext然后将SecurityContext存储到session域。

在过滤其他请求时，先从session域中取出用户信息，保存到SecurityContext线程容器中，这样就可以进行检验是否登录或者鉴权操作。请求结束再次经过过滤器时，会自动将SecurityContext中的用户数据抹除。（SecurityContext中的用户数据只存在一个线程的时间，体会一下，所以SecurityContext其实是一个线程容器）

按照视频讲解在拦截器中将用户信息存储到SecurityContext，并没有将SecurityContext存储到session中，请求结束清除了SecurityContext数据，那么下一次请求到过滤器中对请求进行鉴权时，SecurityContext完全是空的，就会跳转到登录页面。

所以我们在拦截器中一定要将用户信息不光保存到holder中，还要保存到session，这样主页之后的所有请求经过Security的过滤器鉴权时，才能从session中获取到用户信息。保存session的方法在什么时候才能有offerd的解释中提到了，大家可以翻一下。

至于为什么不只进行一次保存session，而是每次请求重复保存，其实也是没办法的事，因为你修改资料的时候就需要更新session中的用户信息。图省事，直接这样就行

#### 退出清理session：

退出后session中有用户信息，如果这时直接地址栏访问需要登录的功能，security会放行，因为session中有用户数据，这就导致未登录情况下直接访问服务器，执行到需要本地线程容器的user的时候就会抛异常。

1. 在SecurityConfig中配置一个LogoutHandler  
    @Bean  
    public SecurityContextLogoutHandler securityContextLogoutHandler() {  
        return new SecurityContextLogoutHandler();  
    }  
2. 在LoginController里注入securityContextLogoutHandler（代码略）  
3. 修改我们的logout功能，调用securityContextLogoutHandler  
    @GetMapping("/logout")  
    public String logout(@CookieValue("ticket") String ticket, HttpServletRequest request, HttpServletResponse response, Authentication authentication) {  
        userService.logout(ticket);  
        // 加入下面这一句  
        securityContextLogoutHandler.logout(request, response, authentication);  
        return "redirect:/login";  
    }

### 置顶加精删除

**SecurityConfig配置类：**

@Bean：

实例化SecurityContextRepository对象用于SecurityContext用户数据的session域存储。

实例化WebSecurityCustomizer对象并忽略静态资源的鉴权。

实例化SecurityContextLogoutHandler对象用于退出登录时清理session域中的用户数据。

实例化SecurityFilterChain对象，增加过滤器链，用于

①配置url权限，且最后勿忘.anyRequest().permitAll()放行所有除去鉴权的url。

②异常处理：session中无用户信息的登陆跳转处理和用户访问权限不足的处理。

③覆盖掉Security自带的退出路径。

**检查凭证的拦截器：**

登录之后，跳转主页过程中，凭证有效，进入拦截器内部逻辑，保存用户信息到本地线程容器，并且保存到SecurityContext线程容器和使用SecurityContextRepository的对象保存到session域中。

请求处理完毕返回到拦截器时，清除本地线程容器和SecurityContext容器。但是session中仍有user信息（user，密码，权限）。

下一次请求如访问私信列表， Security的过滤器生效，因为session中有用户信息，所以正常鉴权。如果用户未登录，鉴权没有用户信息会自动跳转到登录页面，因为登录页面这种我们并没有配置过滤所以不会拦住（未登录的处理config中已手动配置）。如果用户无权限则也有相应处理。

退出登录时，在Controller中使用SecurityContextLogoutHandler清除session中用户信息，否则，此时如果直接访问需要登录的请求，过滤器发现session中有用户数据会放行，而拦截器中发现凭证过期所以不进入内部逻辑，也就不会将user存储到本地线程容器，这就会导致要访问的controller中一旦用到用户信息就会抛出异常。

**导包：**

Security-starter包和thymeleaf-extras-springsecurity6包。

**前端：**

给三个按钮使用ajax发送异步请求。

html中，新增命名空间xmlns:sec=<http://www.thymeleaf.org/extras/spring-security>，在按钮中使用sec:authorize="hasAnyAuthority('moderator')"，表示从Session中获取用户权限，拥有版主权限此按钮才显示。

**数据访问层：**

mapper方法新增更新帖子type，置顶type改为1，加精status改为1，删除status改为2

**视图层：**

如果是置顶加精帖子，那么需要重新查数据库取出post，覆盖掉es中的帖子。

如果是删除帖子则直接从es中删除即可。

controller中接收postId，对数据库的帖子数据修改后，发送event事件到消息队列，消费者负责更新es中数据。

### DV和DAU

DV：网站每日有效访客量，即不重复访客，访客=ip地址

DAU：网站每日用户访客量，也是不重复访客，访客=userId

在新建拦截器中将数据存储到redis中，ip存入hyperloglog，将userid存入bitmap，userId是多少则将bitmap中的第几位设为1。键即为当天的日期，到年月日即可。然后将拦截器配置到config中。

从redis统计访问量时，需要将客户指定的开始日期和结束日期这些天作为key存储到list中。然后

对于UV直接调用redisTemplete.opsForHyperloglog的方法传入新的key和key的集合list，本质上是去重后存入新的键值对中。然后查看新key的值的长度即可。

DAU则必须使用redisTemplete.excute（new RedisCallback）方法，方法内匿名实现写入我们查询逻辑，同样将每天作为key，放入list，然后调用redis针对bimap的或运算方法，依旧是传入新的key和list，获取新key的值的长度。

controller中如果使用return”forward:/xxxx”,则相当于跳转到下一个controller处理，此时仍然是同一请求，model中数据也会传递。

网页上的String类型的日期，想要被Controller中的Date类型形参接收，可以在形参上使用注解@DateTimeFormat(pattern = "yyy-MM-dd")即可。

在SecurityConfig中将/admin/\*\*设置为admin权限

### 分数

使用Spring的Quartz线程池执行定时任务。

引入pom，properties文件中配置。

new xxxJob类实现Job接口，重写excute方法，方法内从redis中取出set集合，集合中是postId，然后将postId弹出，重新计算这些帖子的分数，更新mysql，搜索出post，并更新分数，存入es。

配置类中@Bean JobDetailFactoryBean将xxxJob配置进去，@Bean SimpleTriggerFactoryBean 配置触发器的参数，并将JobDetailFactoryBean配置进去，设置触发时间为2小时.

发布帖子，点赞帖子，评论帖子，加精帖子的controller中，都需要往redis的set集合中存入postId。

访问主页的路径中加入orderMode参数，当为0时表示按照置顶和时间排序，当为1时表示按照置顶，分数，时间排序。

### 生成长图

下载wkhtmltopdf

properties文件中手动配置变量存放生成长图的位置和wk命令所在位置

新建配置类，使用@PostConstruct注解标注一个方法，在方法中我们检测一下生成长图的位置存不存在，不存在则先创建一个。

第一个controller中，我们将随机生成的文件名，文件后缀，前端传输的目标网站生成长图的url，主题设置进event，然后放入消息队列。带着文件名返回一个指向第二个controller的url给前端。

消费者中，将event中元素提取出来，加上配置文件中的手动创建的两个属性，拼接成一个cmd命令，然后执行java控制台命令执行cmd。

用户复制第一个controller返回的url粘贴进地址栏回车后，访问第二个controller，在这里结合各个参数加载文件，然后写出到客户端，注意使用字节流。

### 七牛云

用户修改头像时，将头像上传至七牛云，上传之前需要准备上传凭证，fileName，returnBody等

然后将cdn存储至数据库中用户头像。

用户请求修改页面时，将凭证，fileName，returnBody等准备好渲染到修改页面。然后用户点击上传，异步发送请求到七牛云上传头像，接收到七牛云的returnBody正确后，再次发送异步请求到服务器修改用户头像为七牛云链接。

用户生成分享长图时，在消费者启动cmd命令生成长图到本地时，此时是新线程，所以我们并不知道什么时候生成长图，只能使用spring提供的定时器不断上传长图到七牛云。

在定时器中设置时间间隔为500ms启动一次，新建实现runnbale接口的线程类，run方法中不断检测是否已经生成长图，生成了则上传，上传失败记录失败次数+1，次数超过3次或者超时则使用future.cancle方法取消定时器的执行。这个future对象需要在消费者中使用定时器生成，生成后直接传入线程对象中，因为定时器500ms后才启动，所以启动后线程对象里面已经有了future对象了。

用户请求生成长图功能后返回一个七牛云的路径。

### Caffeine本地缓存优化

由于spring整合不同的缓存工具时，是统一定义过期时间，缓存大小，如果不整合则需要分别定义时间大小。

所以此处不再使用spring的starter整合缓存工具，而是使用缓存工具自己的pom包即可。

适用对象：频繁被访问的，且一段时间之内不会轻易变化的，如热帖排行，一般到了2小时才会自动刷新一次，而缓存设置180秒即可。又如帖子总行数，一般不会有人看到很后面。

目标：

①将查询的时候userId=0，orderMode=1，即热帖排行的帖子缓存起来

②将userId=0的时候的帖子行数缓存起来，即不管是热帖排行还是正常排行的帖子总数用一个值缓存起来。

参数设置：缓存池大小15，过期时间180秒。

由于Service是controller必经之地，所以在postService中使用缓存，在service中的初始化方法中实例化LodingCache对象，设置大小，过期时间，和缓存未命中从二级缓存redis或者直接从数据库取数据的处理方法。

帖子的缓存对象，key是String类型（offset:limit作为每一页的标识），value就是List<Post>（每一页帖子）。

行数的缓存对象，key是Integer类型（userId=0），value也是Integer类型（行数）。

每个缓存对象的大小都是15，过期时间都是180

然后在service中取post（userid==0，orderMode==1）和rows（userId==0），直接从缓存对象中取数据，取不到自动从数据库中取数据，存到缓存然后返回给用户。

Jmeter压测，吞吐量提升大概18倍

### 单元测试

测试类中，最好是将测试对象声明为全局变量。

然后使用@BeforeEach 注解一个方法，此方法内可以对变量进行初始化，这样每个test方法调用前都会获得独立的属于自己的数据。并使用@AfterEach注解标注一个方法对变量进行销毁。

@ BeforeAll @ AfterAll 注解则是在整个测试类启动和结束只调用一次的方法。

### 项目监控

监控单个SpringBoot项目，pom引入actuator的starter。

properties中可以配置开放的监控功能和屏蔽的监控功能。

可以自定义监控点，使用@Component和@EndPoint(**id**=”xxx”)注解类之上。

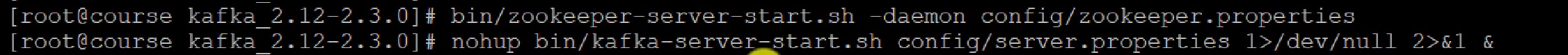
方法上使用@ReadOperation注解标识该方法可以被get请求访问。方法可以返回Json字符串，则方法声明为String类型。

地址栏中直接[localhost:8080/luntan/actuator/**id**](http://localhost:8080/luntan/actuator/database)即可访问我们自定义的端点。

### 项目上线

没啥好说的，云服务器开放所有端口，下载各个程序，改改配置文件，加加环境变量，关关防火墙

启动zookeeper和kafka，后台方式



启动es：



高版本java环境变量的jre引用javahome即可。因为jre不再单独成一个文件夹，被合并了。

nginx作用：分发请求。一所房子两个门，一个8080，一个80，nginx用于将从80进来的人打住，让他从8080走。即反向代理。常用用于隐藏tomcat运行的端口号.