```
author = "gaj"
title = "每日八股"
description = ""
date = 2024-11-9T16:02:00+08:00
draft = true
tags = []
categories = [
   "每日八股"
]
```

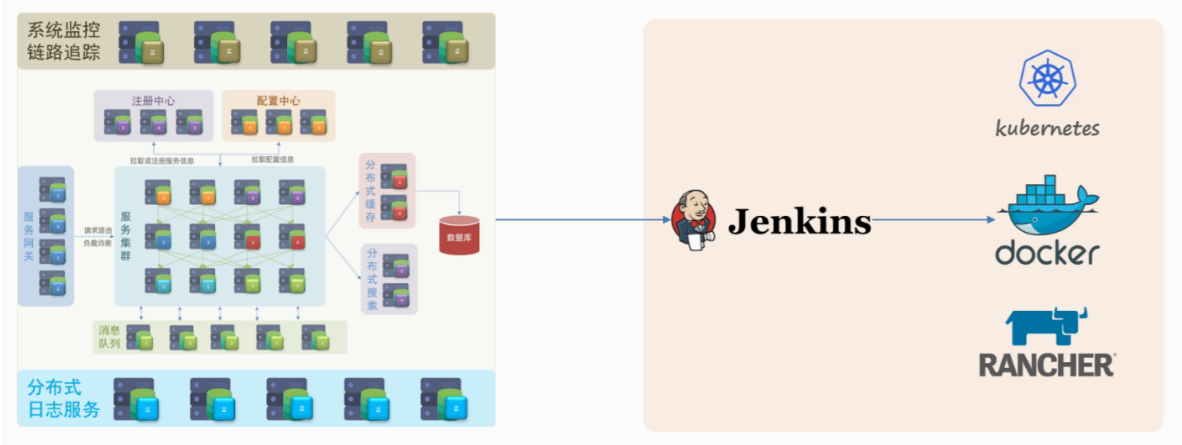
每日八股

> 本文档只记录答题思路，旨在当面试时遇到时能快速切入重点，然后灵活的组织语言回答问题。同时，为了回答的流畅性，应当尽可能的学会怎么读这些英文单词，用英文表达

day1

Q1：微服务你用到了那些中间件

这张图的演变能非常好的描述我们单体结构向微服务转变过程



> 如果可以，请从**单体向微服务架构转变**的思路来回答每个中间件对应的职责（跟面试官说你在公司经历过单体项目向微服务转型的过程，希望以此出发展开回答）

- 单体的耦合度太高，同时无法很好的承载高并发，简单的水平拓展成本过高（也不治本），需要**拆分服务**
- 而服务与服务之间需要远程调用，需要统一的**注册中心和配置中心**（人工维护太繁琐）
- 服务于服务之间的调用，并不必须同步，可以采取引入**消息队列**完成异步调用
- 用户的请求需要统一的服务网关来路由（注意这里路由是个动词），以及负载均衡
- 服务集群的增长，带来了更多的数据，对我们的数据、缓存有更高的挑战，所以需要**分布式缓存、分库分表、分布式搜索**
- 集群规模增大带来，debug不局限于某一台服务，需要**分布式日志服务**和**系统监控链路追踪**
- 引入自动化部署（DevOps）来做到持续开发、持续交付，降低部署维护成本

有了演变过程，哪些中间件做哪些事情，自然就有了答案

> 思考一个问题：假如一个交易平台对于订单需要严格的排序，需要一个定序系统来生产一个全局唯一的序列号，请问这种情况时候还能做水平拓展？如何容灾？设计定序系统 - Java教程 - 廖雪峰的官方网站 (liaoxuefeng.com)（看文末的**如何在定序器崩溃后自动恢复?** 问题，拓展抢锁）

Q2：Nacos注册中心的心跳机制

> 注册中心、心跳检测（ping pong）

每个加入nacos的服务都会**注册自己信息**以及**定时拉取服务列表**（why？for rpc）

注册到nacos的服务以及nacos本身都需要通过主动或者被动的方式维持自己的存活（**主动上报或是主动询问**）

类似场景拓展：Websocket (kookapp.cn)（看连接流程、重连、连接流程示意图即可）

Q3：常见的负载均衡算法

> 轮询、权重、响应时间

Q4：ThreadLocal

> 看成一个**全局** `Map<Thread, Object>`
>
> **线程池复用**会带来污染
>
> **Spring是否也需要**考虑相关问题

- 基本概念

实际上，可以把 `ThreadLocal` 看成一个**全局** `Map<Thread, Object>`：每个线程获取 `ThreadLocal` 变量时，总是**使用** `Thread` **自身作为key**：

因此，`ThreadLocal` 相当于给每个线程都开辟了一个独立的存储空间（上下文语境，学点高级词汇），各个线程的 `ThreadLocal` 关联的实例互不干扰。

最后，特别注意 `ThreadLocal` 一定要在 `finally` 中remove！！！！

- 拓展（问就是看过了SpringMVC源码）

清除非常重要，为什么？**线程池复用**会带来污染

再想一个问题，我们有这样的问题需要使用ThreadLocal，**Spring是否也需要**？结合SpringMVC执行流程

SpringMVC把每一个请求交给一个独立的线程处理，内部同样用到了线程池、ThreadLocal。

DispatcherServlet和RequestContextHolder（注意context这个单词，就是内容或则说上下文的意思）

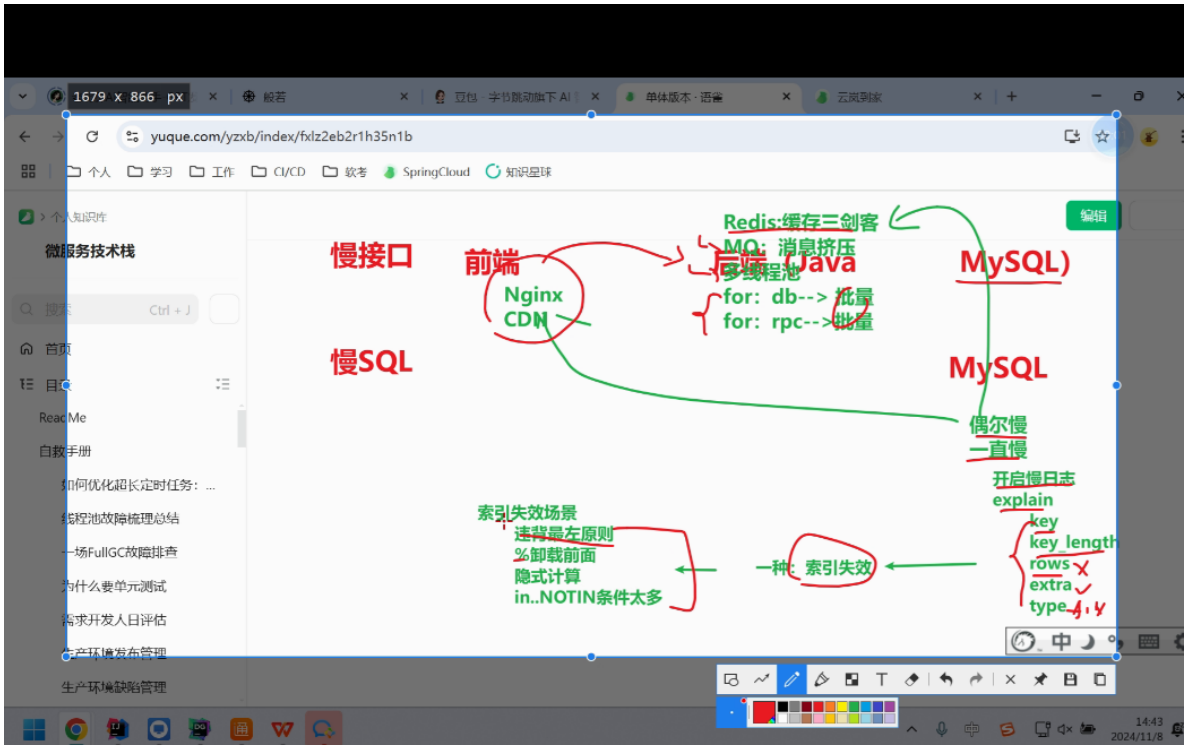
RequestContextHolder会存储请求和响应的相关消息，也就是一个请求的上下文，所以毫无疑问SpringMVC同样有使用到ThreadLocal

Q5：聚集索引和非聚集索引

> 数据和索引在一起存储
>
> 数据和索引不在一起
>
> 主动提及回表查询和覆盖查询、索引的维护有代价

Q6：如何定位慢查询

> 注意慢查询和慢sql
>
> 前端：静态资源丢失
>
> 后端：
>
> - redis缓存
> - MQ积压
>
> 数据库：
>
> - 慢日志
> - explain
>   - key
>   - key_length
>   - rows
>   - extra
>   - type
>
> 主动提及索引失效的场景



Q7：Java的Map

> - 基本
>
>   HashMap的七七八八，你们懂的
> - 拓展
>
>   HashMap/WeakHashMap优化枚举类预加载（弱引用、垃圾回收器）

这里讲讲HashMap/WeakHashMap优化枚举类预加载

首先，我们使用枚举类往往无法避免valueOf()操作，但这个操作是O(n)的，对这个操作非常常见的优化就是通过static静态代码块在类加载的时候把它们加载到一个Map中，也就是缓存，可以优化查找速度到O(1)。

用WeakHashMap的原因，就是为了避免OOM（以及把面试官往JVM引导）