```
author = "gaj"
title = "每日八股"
description = ""
date = 2024-11-09T16:02:00+08:00
draft = true
tags = [
  "每日八股"
]
categories = [
  "每日八股"
]
```

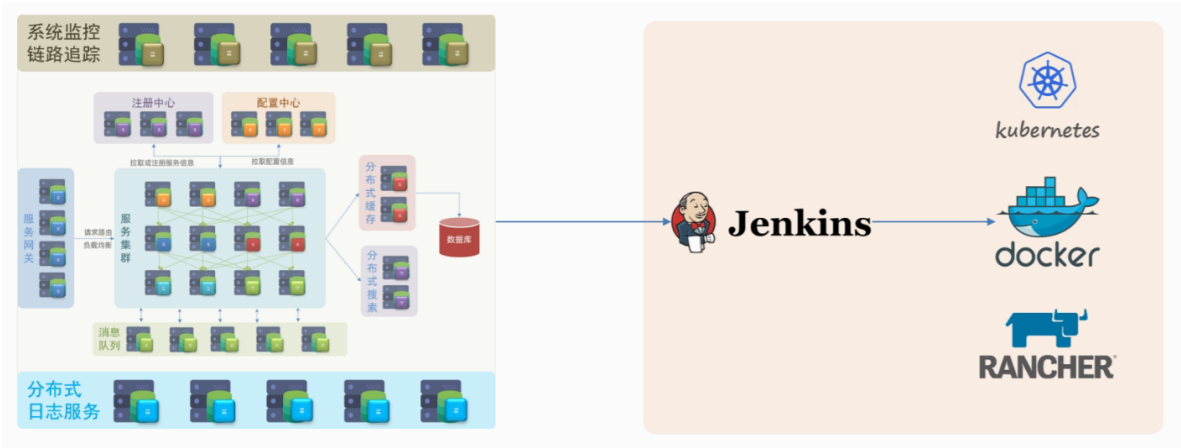
每日八股

> 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）

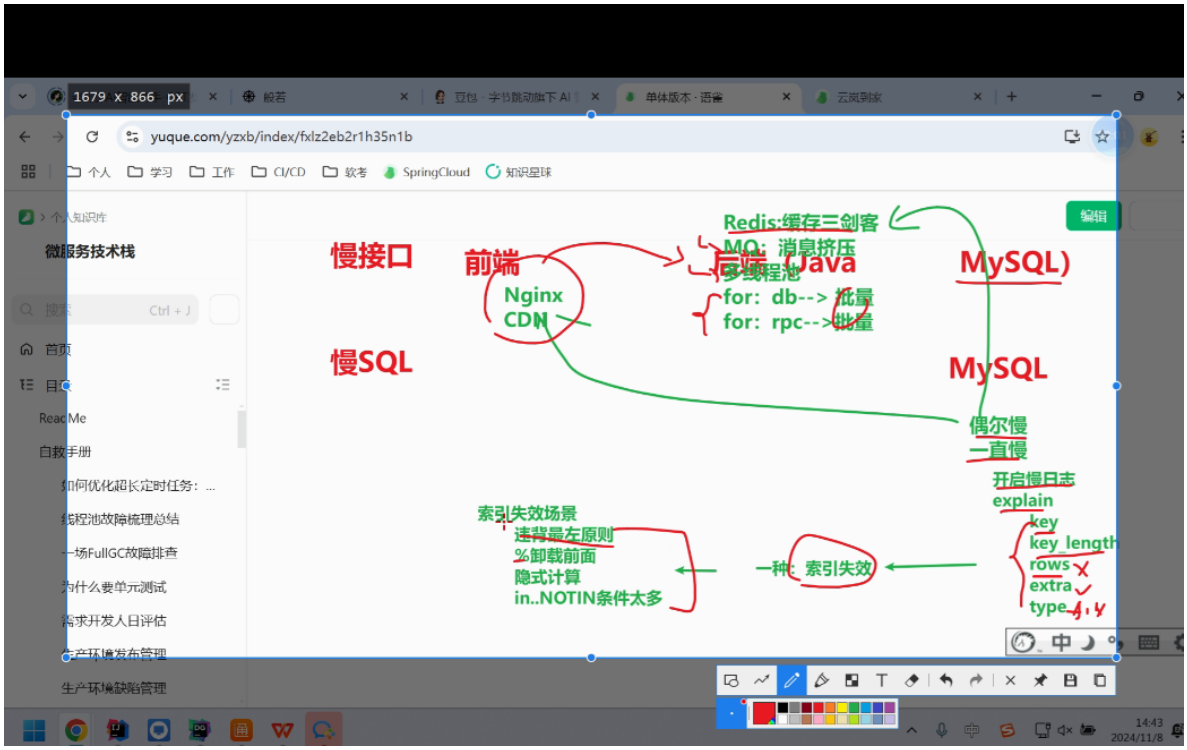
RequestContextHolder会存储请求和响应的相关消息，也就是一个请求的上下文，所以毫无疑问SpringMVC同样有使用到ThreadLocal

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）

day2

> 今天务必要去看一看JVM（看看，挑简单的记，复杂的记大纲），后面的时间可能很难沉下心来去理解，JVM重点关注内存模型和垃圾回收器
>
> 设计模式 - Java教程 - 廖雪峰的官方网站 (liaoxuefeng.com)
>
> 工厂方法设计模式 (refactoringguru.cn)

Q1：GataWay实现原理

> GataWay三大作用

作用

- 路由（网关就是看门老大爷）、负载（分清楚网关负载和服务调用负载的区别）、拦截（注意拦截范围和之前的拦截器更大）

实现原理

- Spring WebFlux、Netty（和传统Servlet有什么区别）
- 结合作用讲请求到网关到具体的服务的流程

Q2：你们项目中的远程调用用的什么技术

> 务必理解当前我们的远程调用就是http请求，那么对于一个http请求我们需要封装交给httpClient发送请求并接收响应
>
> 这种调用往往是同步的，我们还会需要消息队列来达成一些异步调用的操作

openFeign、底层是http客户端、dubbo

Q3：JVM的组成

> 主要组成部分（不会就硬背）

组成部分

- **类加载器**
- 运行时数据区
  - **方法区**

    用于**存储已被加载的类的信息**
  - **堆**

    JVM 管理的最大的一块内存区域，被所有线程共享。**用于存储对象实例和数组**，几乎所有的对象实例和数组都是在堆中**分配内存**的。（和垃圾回收器有关）
  - **程序计数器**

    当前线程所执行的字节码的**行号指示器**（学过计组的话会有类似概念）
  - **虚拟机栈**
- 执行引擎

  两种执行方式
  - **解释执行**：一条一条地读取字节码指令并执行
  - **即时编译**：将热点代码编译成机器码，提高执行效率

示例：在一个简单的 Java 循环语句中，刚开始执行时可能是通过解释执行，当循环执行次数足够多，被判定为热点代码后，执行引擎就会对这个循环部分进行即时编译，之后再执行这个循环时就会以更快的机器码形式执行。

- 本地库接口

  使得Java能调用Native Method Library（java源码中如果见到native往往指的就是c相关的东西）中的函数

> 该文章在字节码的层面分析了异常处理，可以看看，作为了解JVM的一个开头
>
> Java异常处理和最佳实践（含案例分析）-阿里云开发者社区 (aliyun.com)

Q4：类加载机制(AI来的，详情大家自己ai)

> 五个过程、三种类加载器（简称五过三类），不行就背小标题

五个过程

- 加载（loading）

  查找并读取类的字节码文件

  三个加载器

  双亲委派模型（**避免用户自定义的类覆盖Java核心类库的类**）

  - 类加载器收到类加载请求时先找它的父类加载器，再找子加载器
- 验证（verification）

  确保符合 Java 虚拟机规范
- 准备（preparation）

  为类的**静态变量分配内存**并设置初始值
- 解析（resolution）

  将类、方法、字段等的符号引用**转换为直接引用**（便于快速访问）
- 初始化（initialization）

  主要是执行类的**初始化代码**

三种类加载器

- 启动类加载器（**JVM的启动**）

  JVM内置，负责加载Java核心类库
- 拓展类加载器

  负责加载拓展类库
- 应用程序类加载器

  负责加载classpath下的类

Q5：垃圾回收器有哪些，你们项目用的是什么

> G1、CMS，我们用的G1
>
> 1.9开始默认是G1
>
> 先了解GC相关的概念如：FullGC、YoungGC、分代收集、STW

要了解相关配置参数如：

- **-XX:G1HeapRegionSize=n**指定分区大小

- **-XX:G1NewSizePercent**设置年轻代在堆中占比
- **-XX:MaxGCPauseMillis**指定回收时最大暂停时间（G1和CMS不同的点很大程度在于G1是规定时间内尽力完成回收：软实时性）这是**最重要的参数之一**（只记一个那就记这个）
- 。。。

分代收集算法（见3.4.4 分代收集算法，先了解，明天背）

Q6：JVM内存模型（JMM）

JUC（8）JMM-阿里云开发者社区 (aliyun.com)

> 俩内存、三关键字

- 主内存

  线程共享

- 工作内存

  线程独有，存储对共享变量的**副本**（注意是副本）

- 指令重排序导致数据不一致

  使用volatile修饰变量，**确保变量的修改立即对其他线程可见**

day3

> 看设计模式，有时间看看springMVC源码
>
> MVC高级开发 - Java教程 - 廖雪峰的官方网站 (liaoxuefeng.com)
>
> redis面试题选自航哥面试专题

Q1：Docker部署和常用指令

> 部署：dockerfile和dockercompose
>
> 镜像、容器、数据卷、dockerfile、dockercompose

镜像pull、push，docker images，还有docker rmi；还有容器相关的docker run，docker start、docker stop，进入容器的**docker exec -it** 容器名称 bash，查看日志的**docker logs。**

Q2：linux指令

> chmod、cat info.log | grep '关键字'

Q3：都有哪些设计模式，用过哪些，具体场景?

> 经典的面向对象设计模式：三大类型、23种
>
> 还有领域驱动模型（DDD）：先当成思想和沟通语言去理解（通用语言和限界上下文）
>
> DDD基础教程：一文带你读懂DDD分层架构-阿里云开发者社区 (aliyun.com)

创建型模式

- 工厂模式
- 原型
- 单例

结构型模式

- 适配器

- 代理

行为型模式

- 责任链（filter）

  责任链模式是一种把多个处理器组合在一起，依次处理请求的模式

- 命令

  把请求封装成一个命令，然后执行该命令。

- 解释器

  给定一个语言，定义它的**文法**的一种表示，并定义一个解释器，这个解释器使用该表示来**解释语言中的句子。**

- 中介

  目的是把多方会谈变成双方会谈，从而实现多方的松耦合。

- 观察者模式（发布订阅模式）
- 状态
- 模板方法（template）

DDD

- 了解的不多，但可以在开发和沟通的时候带着领域驱动思想去帮助我思考问题，比如
- 通用语言：规定整个项目组（包括开发和产品）当前项目的名词（比如zzyl的PRD的前几页有对护理等级的定义和解释），帮助项目组**提高沟通效率**（但学习成本相应地提高）
- 限界上下文：限界上下文是一个显式的语义和语境上的边界，领域模型便存在于边界之内。
- 聚合、聚合根：
- 充血模型、贫血模型

> 总的来说DDD不是我们现阶段需要重点关注的，但是能够带着它的思想去开发、分析需求绝对是能体现出个人开发经验的

Q4：Redis数据类型

> 结合自己业务说，如：云岚面试第八章抢券业务的Redis数据结构用的什么？具体说说

Q5：Redis三剑客以及如何解决

> 击穿、穿透、雪崩
>
> 不用背八股文！一文搞懂redis缓存击穿、穿透、雪崩！-腾讯云开发者社区-腾讯云 (tencent.com)

- 击穿：（热点数据过期导致大量请求直接打到数据库）
  - **热点数据**即将过期时，提前异步更新缓存
  - 设置热点数据永不过期（谨慎使用）
- 穿透：（查询一个数据库不存在的值导致每次都要查询数据库，恶意请求）
  - 缓存空值
  - 布隆过滤器，底层是bitmap（查询数据时，先通过布隆过滤器判断键是否可能存在，如果不存在就直接返回空值）
- 雪崩：短时间大量的key同时失效
  - key：固定+随机时间
  - 缓存预热
  - redis架构升级

Q6：Redis数据持久化

> RDB和AOF

RDB

对redis生成快照（二进制文件），非常快，但有数据丢失风险

AOF

- 将每个写命令追加到AOF文件末尾，恢复时重新执行这些命令
- 同步策略
    - always
    - everysec（默认这个）
    - no

Q7：redis集群方案

> 主从复制、哨兵模式、分片

主从：降低读写压力

哨兵模式：故障恢复（这里有个哨兵选举流程需要展开）

分片：提高存储上限（hash插槽分配原理）

Q8：springmvc执行流程和常用注解

> 结合d1q4的拓展点

day4

> 看MQ、云岚秒杀、优惠卷