【自我介绍】

**面试官**您好，我叫**胡景峰**。

**本科**就读于 合肥工业大学 计算机学院

**硕士**就读于 东南大学 软件学院（明年6月毕业）

本科和硕士阶段的 **课程成绩排名** 均为前5%

硕士组内是以 **虚拟现实与人机交互** 为大方向

我的科研方向主要是 **渲染+边缘计算** 做一些优化

个人技术栈以**Java**为主，熟悉Spring、MySQL、Redis等

平时有写文档或博客**反思总结**的习惯

是一个**长期主义**者，相信 微小而正确的习惯 带来的 复利效应

【英文介绍】

Hello, my name is Jingfeng Hu.

I **completed undergraduate studies** in Computer Science at Hefei University of Technology.

I’m currently **pursuing Master's degree** in Software Engineering at Southeast University. （and will graduate in 2026.）

I have earned **several honors**, including:

* National Scholarship
* Title of Excellent Graduate

and prizes from competitions like

* China Software Cup
* Mathematical Modeling Contest

I usually use **Java** and I’ve learned about:

* Concurrency Mechanisms
* TCP/IP and HTTP protocols
* MySQL and Redis

I **value teamwork**, **enjoy challenges**, and am committed to **continuous learning and improvement**. I believe my **technical skills and positive attitude** will allow me to **quickly adapt and contribute effectively to our team**.

【项目经历】

吃了么外卖平台

这是一个**网络教程项目**，主要想通过项目实战**理解并应用Spring**。

分页查询

使用 **LIMIT 子句（偏移量、每页记录数）**实现分页查询。

当分页到比如1k页以上时，会出现**深度分页查询**，导致：

1. 数据库需要扫描大量的数据页才能定位到指定页的数据
2. 临时结果集占用过多内存，使用磁盘存储数据，慢

优化方案：

1. **覆盖索引**避免回表
2. WHERE子句过滤数据。使用上一页的最后一条记录的主键ID作为下一页查询的条件。
3. **优化**数据库的缓存大小、应用层Redis**缓存** 、分库分表减少单表数据量

MyBatis分页插件PageHelper简化分页操作

分页排序

在分页查询中添加 ORDER BY 子句可以实现排序

注意：

确保排序字段上有索引，以提高排序性能

避免对排序字段进行函数操作，如 ORDER BY DATE(create\_time)，这会导致索引失效

MyBatis分页插件PageHelper简化分页操作

公共字段自动填充

用于自动填充**创建时间、更新时间**等字段

1. 自定义注解@AutoFill标识需要自动填充的Mapper方法
2. 创建切面类AutoFillAspect获取方法参数并通过反射设置公共字段的值

MyBatis-Plus

1. 在字段上添加@TableField注解，并设置fill属性（INSERT、INSERT\_UPDATE）
2. 实现MetaObjectHandler接口，重写insertFill和updateFill方法
3. 在用户登录成功后将用户ID存入ThreadLocal中，在MetaObjectHandler实现类中获取并设置

订单状态定时处理

1. 支付超时订单的自动取消
2. 派送中订单的自动完成

使用Spring Task实现定时任务

1. 在启动类上添加@EnableScheduling注解，开启定时任务功能。
2. 任务类添加@Component注解，使其被Spring容器管理。
3. 任务方法添加@Scheduled注解，定义任务的执行时间。

Cron 表达式

配置定时任务执行时间的 字符串格式  
秒 分 时 日 月 周 [年]

\*：表示任意值

?：仅用于日期和星期字段

如何处理支付超时的订单？

@Scheduled(cron = "0 \* \* \* \* ?")

1. 定义一个定时任务，每秒执行一次。
2. 在任务中查询状态为“待支付”且下单时间早于当前时间减去15分钟的订单。
3. 将这些订单的状态更新为“已取消”，并记录取消原因和取消时间。

如何处理长时间处于“派送中”状态的订单？

@Scheduled(cron = "0 0 1 \* \* ?")

1. 定义一个定时任务，每天凌晨1点执行。
2. 在任务中查询状态为“派送中”且派送开始时间早于当前时间减去60分钟的订单。
3. 将这些订单的状态更新为“已完成”。

对菜品信息进行缓存、数据一致性问题、Redis

**方案1：先写缓存、再写数据库（×）**

**方案2：先写数据库、再写缓存（×）**

**方案3：先删缓存、再写数据库（×）**

**方案4：先删缓存再写数据库再删缓存（缓存双删）**

写数据库后、二删缓存前，写数据库前的查询刚好回写了旧数据到缓存

**方案5：先写数据库再删缓存**

写数据库后、删缓存前，缓存里的数据不是最新的，此时被查询会不一致

**方案6：先写数据库再通过Binlog异步更新缓存**

理由：高并发场景删除缓存可能并不合适

Spring IoC

依赖注入的3种方式

控制反转IoC是**思想，**依赖注入DI是实现方式

1. **构造器注入（推荐）**：适用于强制依赖的注入
2. **Setter注入**：适用于可选依赖或个人开发的模块的注入
3. **@Autowired字段注入（不推荐）**：因为无法通过构造器或Setter方法修改依赖关系，降低了代码的可维护性和可测试性。

Bean 的作用域

1. **singleton（默认）：** 单例bean
2. **prototype：**适用于短期使用的、有状态、非线程安全的对象

* 唯一标识符（UUID 、验证码、Token）
* 临时数据（表单数据缓存、文件上传临时存储）

1. **request：** 每个 HTTP 请求创建一个实例，如表单数据处理。
2. **session：** 每个会话创建一个实例，如用户信息缓存。
3. **application/global-session：** 在Web应用启动时创建一个实例，如统计数据。
4. **websocket：** 每个 WebSocket 会话创建一个实例，适用于WebSocket 连接状态管理。

Bean 的生命周期

1. 容器通过**反射**根据**配置文件**实例化 Bean
2. **依赖注入**
3. **初始化**
   1. **Aware注入**
   2. **BeanPostProcessor#postProcessBeforeInitialization**（应用：创建动态代理，实现AOP）
   3. Initialization（应用：缓存初始化）
      * + @PostConstruct
        + InitializingBean#afterPropertiesSet
        + initMethod
   4. **BeanPostProcessor#postProcessAfterInitialization**
4. **使用**
5. **销毁**@PreDestroy调用DisposableBean#destroy。

Spring AOP

AOP面向切面编程

通用功能模块化（权限、日志、事务、性能），避免代码重复。

通过代理的方式，先拦截穿插，再调用真正方法。

切面(Aspect) = 切点(Pointcut) + 通知(Advice)

* **切面（Aspect）：**封装了通用功能
* **切点（Pointcut）：**通过切点表达式找到想要织入的方法
* **通知（Advice）：**拦截后要执行的操作

JDK vs CGLIB

**JDK 动态代理**

* SpringFramework的默认实现
* 基于接口实现，通过 Java 的反射机制实现
* 适合实现接口的的类，没有接口的话会有报错

**CGLIB 动态代理**

* SpringBoot2的默认实现
* 基于类继承，通过ASM 字节码生成工具生成继承目标类的子类。
* 适合没有接口的类
* 不能代理 final 类和 final 方法

12306 铁路购票系统

GitHub找的**开源项目二开**，主要想对**高并发**有一个实践和应对。

通过复合分片算法完成订单数据分库分表，支持订单号和用户查询

订单分库分表 选择**订单号**和**用户ID**作为复合分片键。

采用基因算法，将用户ID的哈希值嵌入到订单号中。

采用映射表将用户名/手机/邮箱与用户ID建立映射关系，存储在单独的索引表中。登录时先通过索引表查询到用户ID，再根据用户ID定位到具体的分片，从而避免全表扫描。

分库分表后选择雪花算法作为ID（有序、唯一、性能）

时间戳 + 机器ID + 序列号

封装缓存组件库，避免注册时用户名全局唯一带来的缓存穿透问题

缓存穿透定义：

请求不存在于数据库的数据 造成 数据库负载、缓存内存耗尽、用户体验差

尝试解决方案：

1. 对不存在的 Key 缓存并把值设为Null，设置短暂过期时间（如 60 秒）。  
   **缺点：**尝试但没注册一个不存在的用户名，该值60s内都不可被注册
2. 查询缓存不命中时使用分布式锁来保证只有一个线程访问数据库。  
   **缺点：**其他用户注册请求缓慢或超时
3. 布隆过滤器存已注册用户名，不在布隆过滤器的一定不存在-可用，如果用户名在布隆过滤器中再查询缓存或数据库，。  
   **缺点：**布隆过滤器不能删除元素，注销的用户名无法再次使用。
4. Redis Set存已注册用户名，检查是否在集合内。  
   **缺点：**占用内存。

最终解决方案：

布隆过滤器 + Redis Set缓存

1. 布隆过滤器不存在，说明数据库没有-可用

2. Redis Set缓存存在，说明已注销-可用

3. 查询数据库到底有没有

使用 BinLog 配合 RocketMQ 消息队列完成 MySQL 数据库与 Redis 缓

存之间的数据最终一致性。

**方案1：先写缓存、再写数据库（×）**

**方案2：先写数据库、再写缓存（×）**

**方案3：先删缓存、再写数据库（×）**

**方案4：先删缓存再写数据库再删缓存（缓存双删）**

写数据库后、二删缓存前，写数据库前的查询刚好回写了旧数据到缓存

**方案5：先写数据库再删缓存**

写数据库后、删缓存前，缓存里的数据不是最新的，此时被查询会不一致

**方案6：先写数据库再通过Binlog异步更新缓存**

理由：高并发场景删除缓存可能并不合适

通过 RocketMQ 延时消息实现购票十分钟未支付情况下取消订单功能

**RocketMQ延时消息**：

* 订单生成时，向 RocketMQ 发送一条延时消息，设置延时时间为十分钟。
* 十分钟后，RocketMQ 将消息投递给消费者，检查订单状态。
* 如果订单状态仍为未支付，则释放库存并将订单状态标记为已取消。

订单延时关闭功能技术选型

1. **定时任务（xxl-job、PowerJob、shardingsphere-elasticjob）**
   1. **延时精度**
   2. **高并发场景下，可能导致大量的定时任务同时执行，造成系统负载过大。**
   3. **分库分表问题：订单表按照用户标识和订单号进行了分库分表，那这样的话，和上面说的根据订单创建时间去扫描一批订单进行关闭，自然就行不通。因为根据创建时间查询没有携带分片键，存在读扩散问题。**
2. RabbitMQ
   1. 延时精度：基于消息的 TTL（Time-To-Live）来实现的，因此消息的延时时间并不是完全准确的，可能会有一定的误差。
   2. 高并发问题：消息堆积
   3. 重复消息问题：由于网络原因或其他不可预知的因素，可能会导致消息重复发送到订单队列。如果没有处理好消息的幂等性，可能会导致订单重复关闭的问题，从而造成数据不一致或其他异常情况。
3. Redis 过期监听
   1. 延时精度：Redis 的过期时间是通过定时器实现的
   2. Redis 宕机或重启，那些已经设置了过期时间但还未过期的订单信息将会丢失，导致这部分订单无法正确关闭  
      Redis 5.0 之前是不保证延迟消息持久化的，如果客户端消费过程中宕机或者重启，这个消息不会重复投递。5.0 之后推出了 Stream 功能，有了持久化等比较完善的延迟消息功能。
4. Redisson 的 RDelayedQueue 功能  
   基本上 Redis 过期监听消息存在的问题，RDelayedQueue 也都会有，因为 RDelayedQueue 本质上也是依赖 Redis 实现。
5. RocketMQ  
   时间轮算法  
   在订单生成时，我们将订单关闭消息发送到 RocketMQ，并设置消息的延迟时间为十分钟。RocketMQ 支持设置消息的延迟时间，可以通过设置消息的 delayLevel 来指定延迟级别，每个级别对应一种延迟时间。这样，订单关闭消息将在十分钟后自动被消费者接收到。  
   rocketmq好整合一点，性能更高。rabbitmq是一个大型中间件，光docker部署就上1个G了。而且还封闭，性能比不过rocketmq。

如何避免已支付错误取消

在消费者处理延时消息时进行**状态校验**：消费者接收到延时消息后，首先检查订单的支付状态。如果订单已支付，则直接忽略消息，不执行取消操作。

【荣誉奖项】

“中国软件杯”大学生软件设计大赛国家三等奖

* 基于金蝶云低代码平台这一完全陌生的技术栈，通过阅读平台文档短时间内零基础学习开发完成图书管理系统
* 锻炼了在未知领域中快速学习与落地的能力
* 培养了协调资源、带领团队的组织能力

全国大学生数学建模竞赛省二等奖

* 与队友连续奋战三天三夜，从问题建模、算法实现到论文撰写，每一个环节都精雕细琢。过程中我深刻体会到持续专注推进一件复杂任务所带来的成就感。
* 锤炼了数理逻辑与编程建模能力
* 提升了在高压下保持思维清晰、推进任务执行的能力

【反问环节】

* 能谈一谈您作为一个公司老员工对公司的感受，和其他公司不同之处？
* 我觉得我这次表现不是很好，您可以给一些评价或建议吗？
* 这个岗位为什么还在招人？
* 实习生主要工作？