java后端面试准备

目录来源：力扣Leetbook

较基础的，或者说我已经有准备的这里不冗余描述，只记录我个人觉得有欠缺的地方

基础机制

HashMap

HashMap 主要由数组和链表（在 Java 8 及以后版本中，还可以是红黑树）组成。

- 数组：HashMap 的核心是一个数组，数组中的每个元素称为桶（bucket）。
- 链表：每个桶可以存放一个链表，当多个键的哈希值相同且映射到同一个桶时，这些键值对会以链表的形式存储在这个桶中。
- 红黑树：当某个桶中的链表长度超过阈值（默认是 8）时，链表会转换为红黑树，以提高查找效率。

SPI机制

基于反射实现的，可以直接通过类的全限定名实现对类的调用。实际应用场景下，主要是为了实现接口与方法的分离，也可以在运行时动态发现和加载服务实现。

同样我们也可以实现自定义的SpiLoader来满足更灵活的需求。

数据库相关

MySQL 数据库的四个隔离级别。

- 读未提交（Read Uncommitted）

  > 最低的隔离级别，允许事务读取未被其他事务提交的数据。这意味着一个事务可以看到其他事务尚未提交的修改，可能会导致脏读、不可重复读和幻读问题。

- 读已提交（Read Committed）

  > 允许一个事务只能看到已经提交的其他事务的修改。这可以避免脏读，但可能会出现不可重复读和幻读问题。

- 可重复读（Repeatable Read）

  > MySQL 默认的隔离级别。确保在同一个事务中多次读取同一行数据时，结果保持一致。其他事务对数据的修改在当前事务结束之前是不可见的，可以避免脏读和不可重复读，但仍可能出现

幻读问题。

- 串行化（Serializable）

  最高的隔离级别，通过强制事务串行执行来避免所有并发问题，包括脏读、不可重复读和幻读。它通过对所有的读操作和写操作加锁来实现隔离，保证事务之间的完全隔离性。

什么是mvcc，具体机制是什么

MVCC是一种提高数据库的并发性和性能，同时保证数据的一致性和隔离性的机制；相应的，其实开销也大，实现困难。其实现主要有以下几点。

- 版本控制：每条记录在数据库中都有多个版本，每个版本都有一个唯一的时间戳或事务ID。时间戳或事务ID用于标识数据版本的创建时间或事务。
- 读操作读的是历史版本
- 写操作是创建一个新版本
- 删除操作其实只是将记录标记为忽略读取

数据结构

B+ Tree 与 B-Tree 的结构很像，但是也有自己的特性

- 数值都存储在叶子节点中
- 叶子节点形成链表

与B树相比，树更矮，有更短的查询路径，范围查询的效率更高

红黑树的查询效率为什么高？

红黑树依赖旋转操作，使得树尽量保持一个平衡的状态，树的高度不高，同时，设计上也符合二分查找的思想，查询效率比较高

除了用红黑树，还能用什么方法解决 hash 冲突(链过长的情况)？

不了解，不过可以有个思路，比如能否像二级指针那样，再hash。具体细节不清楚。

hashmap 什么时候会触发扩容？

hashmap有一个默认的初始容量16，有一个负载因子，当使用的容量比例大于这个因子时，会触发扩容，扩容的内容是当前空间的2倍，而且会重新计算当前hash，和位置。扩容的开销较大，我目前没有想到相应的替代方案。

线程

线程与协程

- 线程在核上运行，多个线程可以在多个核上同时运行
- 协程则是让出执行权，主动控制其暂停点 协程一般用在I/O密集的场景，需要高并发但不需要多核处理

容器的线程安全与线程不安全

我只使用过concurrentHashMap类，他的线程安全机制是利用分段锁来避免同时对数据进行读写操作

java Spirng

bean的生命周期

我大致分为三个阶段，创建，实例，销毁。当然，创建和销毁也还可以继续下分。

单例与多例bean的区别

单例的bean是全程交由spring容器管理，多例则是在使用时会创建相应的实例，但spring不会去管理后续的操作，像销毁这些操作则由客户端完成

SSM开发框架

是一个经典的 Java Web 开发框架组合，将 Spring 的依赖注入和事务管理、Spring MVC 的 Web 开发功能、MyBatis 的持久化层能力结合在一起

项目相关

Dubbo 异步调用的工作原理

基于netty的线程池实现，发请请求后不必阻塞等待响应结果。

如何确保单例模式在多线程环境下的安全性

我使用的是在工厂模式用static语句块实现懒加载完成的，相应的我了解的还有静态内部类的方法，大致和我这个差不多，还有饿汉式单例，双重检查锁定等，不过我了解的不多。