链滴

记录一些面试中记忆比较深刻的问题

作者：wuhongxu

原文链接：https://ld246.com/article/1595086679806

来源网站：链滴

前言

最近在广东找工作，天天疲于奔波各种面试，大大小小的公司零零总总，记录一些自己感觉有意思的试/笔试题/机试题。根据各个面试官的提问，做了一部分汇总，所以记录中是所有公司面试的一个超，不代表某一个公司。

题目来自记忆，并不是照抄，只表达含义。

更重要的是，我记录的大多来自记忆，有些可能已经有错了，如有遗漏/错误，请留言更正，帮助我及更多的人纠正，谢谢！

因面试未完成，持续更新~

向大家请教以下面试经验

如何回答以下两个问题，真心求解，一直感觉答不好：

- 介绍一下自己
- 开发过程中遇到过什么问题，怎么解决的？

正文

一些笔试

实现两个大整数相乘，不能使用BigInteger之类的现有api，需适当考虑效率

自己用的竖式做出来，时间复杂度为O(n^2)，其实当第一次看到这个问题时，首先就想到分段计算，但是仔细想了想，仍然是O(n^2)的时间复杂度，优化空间并不大，具体代码很简单，就不赘述了。

在与小伙伴们聊到这个问题时，帮我提供了一种算法：**Karatsuba**(分治算法)。

中心思想是这样子：**将大数分成两段后变成较小的数位，然后做3次乘法，并附带少量的加法操作和位操作。**

首先，有两个大整数a,b。

可以写成：

a = a0 * 10^m + a1;

b = b0 * 10^m + b1;

其中m是最小的正整数，使得a0,a1,b0,b1 < B^m，也就是把a,b切成了两半。接下来推导：

a * b = (a0 * 10^m + a1)(b0 * 10^m + b1)

= a0 * b0 * 10^2m + (a0 * b1 + b0 * a1) * 10^m + a1 * b1

= a0 * b0 * 10^2m + (a0 + a1)(b0 + b1)* 10^m - (a0 * b0 + a1 * b1)*10 ^ m + a1 * b1

在这之后，发现只需要计算三次乘法 a0 * b0、a1 * b1、(a0 + a1) * (b0 + b1)以及六次加法。

时间复杂度为：

T(n) = 3T(n / 2) + 6n = O(n^log3) ~= n ^ 1.585

一些面试

讲述一下hashmap

没看过jdk7的源码，只看过jdk8的，所以我的描述来自jdk8。

jdk提供的键值对容器，内部使用数组加链表结构存储，通过对键进行hash均匀分布在数组中，当出现ash冲突时，进行链表链接，当链表达到一定长度之后，转换为红黑树。

默认大小？负载因子?

默认大小是16，负载因子是0.75

转换成红黑树的时机

链表长度大于等于8时转化为红黑树

线程是否安全，如何在线程安全下使用

线程不安全，可以有以下几种方式：

- Collections.synchronizedMap包装
- 使用HashTable
- 最主流的还是使用ConcurrentHashMap类

说说ConcurrentHashMap如何实现线程安全?

ConcurrentHashMap使用了大量的volatile,final,CAS等无锁技术减少锁竞争。在不得不加锁的情况：例如put，使用了局部加锁而不是全局加锁。

说说synchronized关键字

synchronized是用来保证同一时刻只有一个线程可以执行它修饰的方法和代码块，具有可重入性。

说说单例模式？多线程下怎么使用?

单例模式分为懒汉模式和饿汉模式。

饿汉模式是在类初始化时即初始化，没有线程安全问题。

懒汉模式是延迟加载的，多线程下会有线程安全问题，简单的可以直接使用synchnoized方法修饰获方法或者代码块，为了效率进行优化，可以使用两层检查，也就是一层不加锁的情况下保证在已经初化之后的获取性能，在判断中加入同步代码块再进行一次检查，保证多个线程都在第一次获取时的线安全问题。

synchronized实现原理了解过吗?

简单看过，jvm对于每个对象都会有一个monitor监视器，当使用synchronized包装时，实际上就是与monitor进行交流，底层使用操作系统的内核api，应该是叫MutexLock，它的数量有限，并且因调用它的时候会产生用户空间和内核空间的交互，所以效率较低，jvm为它提供了锁升级过程进行优。（这里忘了说jvm的moniterenter和monitorexit指令，应该可以加点分吧）

后面为自己加了点戏：

锁的升级过程是无锁->偏向锁->轻量级锁->重量级锁。当第一个线程进入到同步代码块时，jvm并不首先为它向内核空间获取锁，而是放置了一个偏向锁标志，然后就直接进入到代码块中，当其他线程产生竞争时，锁就会升级到轻量级锁，此时也不会去内核空间是申请锁，而是采用自旋的方式，使用AS尝试不断的修改标志中的线程id为自己，如果自旋超时，或者还有其他线程过来竞争（也就是竞争激烈的情况下）就会去回退轻量级锁，去内核申请锁，也就是升级为重量级锁。

设计模式分为哪三大类，分别用来干什么?

设计模式分为创建型模式，结构型模式和行为型模式。

创建型模式主要用来创建对象，屏蔽创建细节，将对象的创建和使用分离

结构型模式主要用来组装（组合？）一系列对象，分为类结构性模式使用继承机制组织接口和类，对结构型模式使用聚合的方式来组合对象。

行为型模式主要用来控制运行流程，描述如何写作共同完成某个任务。分为类行为模式和对象行为模，前者使用继承机制在类之间分派行为，后者采用组合或者聚合的形式在对象间分配行为。

看你也有自己的开源项目，举例说说用过哪些模式？如何使用的？举说说jdk中使用哪些模式？描述下某个模式的使用场景

说得有点多，略过。

用过ThreadLocal吗？用来干什么?

用过，ThreadLocal主要用来共享线程私有变量，例如在Spring Security中就使用（默认）ThreadLoal传递当前请求中缓存的权限信息。

ThreadLocal使用需要注意什么吗?

主要应该是注意内存泄漏或者说脏数据的问题，当在使用它时，线程结束时会自动销毁，但是在线程情况下，因为线程一般并未销毁，所以数据依然存在，可能发生内存泄漏。如果线程池中可能会有本并未写数据，但是上次写入了，那么就会产生脏数据。

ThreadLocal的原理了解吗?

简单了解过，ThreadLocal实际上使用的是线程本身私有的一个ThreadLocalMap对象。当它在进行取时，是将ThreadLocal本身做为键去Map中进行查询，这里还有一个延迟加载的初始化操作，如果次查询未查询到，则会调用可继承实现的initialValue方法，我们在使用时也可以重写此方法以赋予初值。当进行赋值时实际上也是延迟在线程中创建map的，默认容量时16。

ThreadLocalMap和HashMap有什么不同?

- ThreadLocalMap是一个完全和HashMap不同的实现，它是一个数组结构，存储的对象是一个Entr，使用ThreadLocal作为键，value作为值，没有HashMap那样的链表延长操作，当出现hash冲突的候，它会向后依次寻找合适的位置进行插入，在获取时也是依次向后获取的。
- 它的负载因子是2/3而不是3/4。
- 它的hash算法也是不一样的，它是每次固定增加一个基准值（不记得是多少）。
- 它的key是弱引用而不是hashMap那样的强引用。

说说对Spring的ApplicationContext的理解

ApplicationContext是Spring的核心IOC容器接口，用于生成和管理Bean。它继承拓展了BeanFactor接口，提供了更多的功能。ApplicationContext下面的具体实现例如 ClassPathXmlApplicationContext等等用来解析不同的配置文件，也可以使用 GenericApplicationContext来组合多个配置来源。

...一天写一点吧，今天就这样....