2025.03.17 虾皮 1.5h【a2】

编程

Lc16

import java.util.\*;

public class Main {

public static void main(String[] args) {

Scanner in = new Scanner(System.in);

String s = in.nextLine();

int res = 1;

int n = s.length();

if (n <= 1) {

System.out.println(n);

return;

}

int left = 0, right = 0;

Map<Character, Integer> mp = new HashMap<>();

while (right < n) {

char rc = s.charAt(right);

int idx = mp.getOrDefault(rc, 0);

left = Math.max(left, idx);

res = Math.max(res, right-left+1);

mp.put(rc, right+1);

right++;

}

System.out.println(res);

return;

}

}

Lc21

import java.util.Scanner;

import java.util.regex.Pattern;

import java.util.regex.Matcher;

public class Main {

public static void main(String[] args) {

Scanner in = new Scanner(System.in);

String ss = in.nextLine();import java.util.\*;

import java.util.Scanner;

import java.util.regex.Pattern;

import java.util.regex.Matcher;

import java.util.stream.Collectors;

public class Main {

public static void main(String[] args) {

Scanner in = new Scanner(System.in);

String ss = in.nextLine();

// System.out.println(ss);

Pattern pattern = Pattern.compile("L1 = \\[(.\*?)\\], L2 = \\[(.\*?)\\]");

Matcher matcher = pattern.matcher(ss);

List<Integer> L1 = new LinkedList<>();

List<Integer> L2 = new LinkedList<>();

if (matcher.find()) {

L1 = Arrays.stream(matcher.group(1).split(","))

.map(Integer::parseInt)

.collect(Collectors.toCollection(LinkedList::new));

L2 = Arrays.stream(matcher.group(2).split(","))

.map(Integer::parseInt)

.collect(Collectors.toCollection(LinkedList::new));

// System.out.println(L1);

// System.out.println(L2);

}

List<Integer> merged = new LinkedList<>();

Iterator<Integer> it1 = L1.iterator();

Iterator<Integer> it2 = L2.iterator();

Integer val1 = it1.hasNext() ? it1.next() : null;

Integer val2 = it2.hasNext() ? it2.next() : null;

while (val1!=null || val2!=null) {

if (val1 == null) {

merged.add(val2);

val2 = it2.hasNext() ? it2.next() : null;

} else if (val2 == null) {

merged.add(val1);

val1 = it1.hasNext() ? it1.next() : null;

} else if (val1 <= val2) {

merged.add(val1);

val1 = it1.hasNext() ? it1.next() : null;

} else {

merged.add(val2);

val2 = it2.hasNext() ? it2.next() : null;

}

}

StringBuilder sb = new StringBuilder();

sb.append("[");

for (Integer num : merged) {

sb.append(num).append(",");

}

sb.deleteCharAt(sb.length()-1);

sb.append("]");

System.out.println(sb.toString());

return;

}

}

// System.out.println(ss);

Pattern pattern = Pattern.compile("s = \"(.\*?)\", k = (\\d+)");

Matcher matcher = pattern.matcher(ss);

StringBuilder sb = new StringBuilder();

int k = 0;

if (matcher.find()) {

sb.append(matcher.group(1));

k = Integer.parseInt(matcher.group(2));

// System.out.println(sb);

// System.out.println(k);

}

char[] ca = sb.toString().toCharArray();

int res = 0;

int vow = 0;

for (int i = 0; i < ca.length; i++) {

if (ca[i] == 'a' || ca[i] == 'e' || ca[i] == 'i' || ca[i] == 'o' || ca[i] == 'u') {

vow++;

}

if (i < k-1) {

continue;

}

res = Math.max(res, vow);

char c = ca[i-k+1];

if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u') {

vow--;

}

}

System.out.println(res);

return;

}

}

Lc1456

import java.util.Scanner;

import java.util.regex.Pattern;

import java.util.regex.Matcher;

public class Main {

public static void main(String[] args) {

Scanner in = new Scanner(System.in);

String ss = in.nextLine();

// System.out.println(ss);

Pattern pattern = Pattern.compile("s = \"(.\*?)\", k = (\\d+)");

Matcher matcher = pattern.matcher(ss);

StringBuilder sb = new StringBuilder();

int k = 0;

if (matcher.find()) {

sb.append(matcher.group(1));

k = Integer.parseInt(matcher.group(2));

// System.out.println(sb);

// System.out.println(k);

}

char[] ca = sb.toString().toCharArray();

int res = 0;

int vow = 0;

for (int i = 0; i < ca.length; i++) {

if (ca[i] == 'a' || ca[i] == 'e' || ca[i] == 'i' || ca[i] == 'o' || ca[i] == 'u') {

vow++;

}

if (i < k-1) {

continue;

}

res = Math.max(res, vow);

char c = ca[i-k+1];

if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u') {

vow--;

}

}

System.out.println(res);

return;

}

}

2025.03.15 美团 1.5h【a1.5】

选择

模板模式、组合模式、装饰器模式、原型模式哪个是行为型

哪种垃圾回收方式按照时间分割任务，垃圾回收和增变者交错

49 38 65 97 76 13 27 49，前三轮排序结果都是38.49.65.97.76.13.27.49，可能是什么排序算法

6个数据链路层帧，物理信道失败率10%，无连接服务，则发送网络层分组成功率

XYZN四个字符任意方式进栈，任意方式出栈得到字符串，由多少种入栈排列得到XYZN（卡塔兰数）

已经知道段号、段首地址、段长度和逻辑地址1262，怎么计算对应物理地址

在create table时候，unique（a,b）语句在a，b字段语句之前之后之中，有什么影响

按照时间分割工作任务是什么回收方式

部分回收、世代垃圾回收、增量回收

编程

1、签到题，字符串处理，if

2、i范围l1r1 j范围l2r2，排列组合，如果i是j的倍数，res++（二维前缀和）

3、最小生成树

2025.03.15 OPPO 2h【a3】

选择

try后面必须跟catch快，finally可以没有，对吗

53842，用冒泡排序，两次完整遍历后是什么样

数据表1有abbc，数据表2有aaabbcd，那么select\*from1 2 where1.x==2。x返回结果是什么

HTTP缓存作用是减少流量还是增加带宽

深度为3的完全二叉树节点最少多少

linux通配符解释? \* [] {}

列出将-r--r--r--权限改为631的所有方法

15，9，7，23，45，8，67，93，3快速排序，以第一个记录为基准，从小到大第一次划分结果.3.9.7.8.15.23.45.93.67还是3.9.7.8.15.45.67.93.23？

范式级别越高，数据库性能越好

53842冒泡

编程

1、签到题

2、MEX

3、有一个数字，位数很多，里面有一些数字被污染遮盖（可以是0-9），如果整个数是3的倍数，有多少种排列组合数？注意（1）避免前导0（2）结果记得取余（3）用dp思维来做

2025.03.19 美的 1.5h【a3】

单选

操作系统动态分区算法回收内存，如果回收区仅仅与空闲区链插入点前一个分区相邻接，那么需要？

java8引用特性，names.stream().map(string.touppercase.foreachsystem.,out.println),为我修改使其语法正确

java定义类可以不写构造函数吗

类里面的private的x属性可以被实例obj.x读取到吗

被中断的进程可以立即获得CPU控制权恢复执行，对还是错

被中断的进程将进入就绪队列等待执行，对还是错

为我介绍ICMP

书写方法头，使用类名作为前缀就可以调用A.function，方法头应该写成什么样

linux哪个指令为脚本取消可执行权限

多选

mysql的四种常见日志哪些需要解析后才能查看

SSL协议客户端和服务器握手阶段交互的信息有？使用的对称算法、不重数、MAC算法、使用的公钥算法，哪些是? SSL握手阶段交互MAC算法了吗？

XSS攻击可分为存储型和反射型？对吗？XSS攻击主要方法是什么？

对特殊字符转义处理可以预防SQL注入吗？

为我枚举java实现多线程的方法？自定义实现runnable缺点是没有返回值吗？

需要构造一个thread实例去启动？

按需分配可以破坏死锁的什么条件？银行家算法是死锁解除算法，正确吗？资源动态分配过程防止系统安全状态，可以避免发生死锁，对吗？在资源动态分配过程中防止系统进入安全状态可以避免死锁吗？

为我判断sql是否正确 update set id = 10 from a where id = 1？为我判断sql是否正确select \* from a limit(7,10)？

编程

最长递增子序列

给你一个nums整数数组，求最长递增子序列长度

循环字符串

一个循环字符串的子字符串s，可以多次连接到自身以获得此循环字符串的最小可能字符串长度是多少？

输入“cabca“

输出3

因为cab长度为3

public class MinRepeatedSubstring {

public static int minRepeatedSubstringLength(String s) {

int n = s.length();

// 尝试每个可能的子串长度 i

for (int i = 1; i <= n; i++) {

// 如果字符串长度可以被 i 整除

if (n % i == 0) {

String sub = s.substring(i, j);

// 构造一个由 sub 重复拼接的字符串

StringBuilder sb = new StringBuilder();

for (int j = 0; j < n / i; j++) {

sb.append(sub);

}

// 如果拼接得到的字符串 sb 包含原字符串 s，返回子串的长度 i

if (sb.toString().contains(s)) {

return i;

}

}

}

return n; // 如果找不到，返回整个字符串长度

}

public static void main(String[] args) {

String s = "cabca";

System.out.println(minRepeatedSubstringLength(s)); // 输出 3

}

}

孤岛

4 5

00000

00\*00

0\*0\*0

00\*00

Bfs

2025.03.27 携程【a 1+0.6+0.3+0.2】

100%

import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

        Scanner in = new Scanner(System.in);

        String timeString = in.nextLine();

        int hours = Integer.parseInt(timeString.substring(0, 2));

        int minutes = Integer.parseInt(timeString.substring(3, 5));

        int res = 0;

        while (true) {

            String curTime = String.format("%02d:%02d", hours, minutes);

            if (isParlindrome(curTime)) {

                System.out.println(res);

                break;

            }

            minutes++;

            res++;

            if (minutes == 60) {

                minutes = 0;

                hours++;

                if (hours == 24) {

                    hours = 0;

                }

            }

        }

        return;

    }

    public static boolean isParlindrome(String time) {

        return time.charAt(0) == time.charAt(4) && time.charAt(1) == time.charAt(3);

    }

}

60%

import java.util.Scanner;

import java.util.\*;

public class Main {

    public static void main(String[] args) {

        Scanner in = new Scanner(System.in);

        int n = in.nextInt();

        int[] a = new int[n];

        for (int i = 0; i < n; i++) {

            a[i] = in.nextInt();

        }

        Stack<Integer> st = new Stack<>();

        int res1 = -1, res2 = -1;

        for (int i = 0; i < a.length; i++) {

            int aa = a[i];

            if (aa > 0) {

                st.push(aa);

            } else {

                int ex = -aa;

                if (!st.isEmpty() && st.peek() == ex) {

                    st.pop();

                } else {

                    res1 = i+1;

                    break;

                }

            }

        }

        res2 = res1+1;

        System.out.println(res1 + " " + res2);

    }

}

33.3%

import java.util.Scanner;

// 注意类名必须为 Main, 不要有任何 package xxx 信息

public class Main {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int T = sc.nextInt();

        while (T-- > 0) {

            int n = sc.nextInt();

            String s = sc.next();

            System.out.println(getWays(s));

        }

        sc.close();

    }

    public static int getWays(String s) {

        int n  = s.length();

        if (n == 0) return 0;

        int[] dp = new int[n + 1];

        dp[0] = 1;

        for (int i = 1; i <= n; i++) {

            if (s.charAt(i - 1) >= '1' && s.charAt(i - 1) <= '9') {

                dp[i] += dp[i - 1];

            }

            if (i > 1) {

                int num = Integer.parseInt(s.substring(i - 2, i));

                if (num >= 10 && num <= 26) {

                    dp[i] += dp[i - 2];

                }

            }

        }

        return dp[n];

    }

}

20%

import java.util.Scanner;

import java.util.\*;

// 注意类名必须为 Main, 不要有任何 package xxx 信息

public class Main {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

        PriorityQueue<int[]> queue = new PriorityQueue<>(

            (a,b) -> a[0]!=b[0] ? Integer.compare(a[0], b[0]) : Integer.compare(a[1], b[1])

        );

        PriorityQueue<Integer> queue2 = new PriorityQueue<>();

        for (int i = 0; i < n; i++) {

            int start = sc.nextInt();

            int end = sc.nextInt();

            queue.offer(new int[]{start, end});

        }

        int res = 0;

        while (!queue.isEmpty()) {

            int[] cur = queue.poll();

            int start = cur[0];

            int end = cur[1];

            while (!queue2.isEmpty() && queue2.peek()<start) {

                queue2.poll();

            }

            res += queue2.size();

            queue2.offer(end);

        }

        System.out.println(res);

        sc.close();

    }

}

2025.03.15 美团AI面试

TCP与UDP的区别及使用场景

**TCP**：

面向连接

可靠

按字节

文件传输

**UDP**：

无连接协议

不保证可靠性

报文

实时通信

Linux查看文件的命令？列出至少三条

cat example.txt用于显示整个文件的内容。

less example.txt逐页显示文件内容，适用于较大的文件。可以向上或向下翻页。  
按 空格键（Space）可以向下翻一页。

按 b 键可以向上翻一页。

head -n 5 example.txt查看 example.txt 文件的前5行

请你解释一下String对象不可变

不可变意味着一旦创建了 String 对象之后，它的值无法被改变。所有修改 String 内容的方法都会返回一个新的 String 对象，而不会改变原始对象的值。

如果 String 是可变的，可能会导致安全性问题，比如字符串在不同线程之间的共享时产生不一致的结果。

Java中的字符串常量池（String Pool）就是一个用于存储唯一字符串的池。每次创建一个字符串时，JVM会先检查常量池中是否已存在该字符串，如果存在就返回该字符串的引用，避免重复创建相同的字符串，从而节省内存。

什么是ACID？

1. 原子性 (Atomicity)：要么全部执行成功，要么全部不执行
2. 一致性 (Consistency)：一致性保证事务开始前和结束后，遵守数据库的所有约束（如外键、唯一性等）
3. 隔离性 (Isolation)：多个事务并发执行时，一个事务的执行不应受到其他事务的干扰
4. 持久性 (Durability)：一旦事务被提交，其对数据库的修改是永久性的，不会丢失，即使系统崩溃或重启后也能恢复。

hashCode和equals的关系？为什么要重写hashcode和equals？

如果两个对象通过 equals 方法被认为是相等的，那么这两个对象的 hashCode 方法必须返回相同的哈希值。

HashMap、HashSet、Hashtable依赖于 hashCode 方法来确定对象的位置，依赖于 equals 方法来判断对象是否相等。

通过什么途径关注最新技术趋势？说一下你最近关注的新技术。

1. 技术博客（**Stack Overflow**、**GitHub**）
2. 学术资源（google scholar、arXiv）
3. 播客（声动早咖啡）
4. 白皮书/行业报告

可解释AI（Explainable AI）：随着AI系统在各行各业的应用越来越广泛，如何让AI模型的决策过程更加透明和可解释，变得至关重要。学者和研究人员正在积极开发新的可解释性算法和框架。

如何快速学习一门新技术？实习或项目中，快速学习的情况如何处理？具体描述一个案例。遇到不会的技术领域问题怎么解决？

1. 紧急程度、学习目标【开发一个基于 Spring Boot 的简单 RESTful API，用于处理用户信息】
2. 文档、教程、社区、书籍
3. 动手实践、渐进式学习【从简单的“Hello World”开始，逐步添加用户管理的功能，如用户的增、删、查、改操作】
4. 请教他人（技术栈、最佳实践）
5. 总结复盘

未来三年职业规划

1. 技能提升与基础建设：精通 Java 或 Python 编程语言，掌握常用框架（如 Spring Boot、Django），参与至少两个项目，增加开发经验。
2. 深入领域与专业化：深入学习架构设计，掌握微服务、容器化技术（Docker、Kubernetes）等，争取担任项目负责人，带领团队完成开发任务。
3. 领导职位与影响力：观察学习技术领导职位，如技术经理或架构师，负责技术路线选择、团队技术指导等，提升公司技术影响力，培养团队成员。

2025.03.20 美团一面

面试官：陶伟 履约平台技术部 base北京

自我介绍

* 胡景峰
* 硕士就读于东南大学软件学院，成绩排名第3  
  本科毕业于合肥工业大学计算机学院，绩点排名第17，均为前5%  
  计算机网络、操作系统、数据库系统等课程成绩都是90+甚至95+成绩
* 技术栈Java，熟悉Java并发、Spring框架、MySQL和Redis
* 长期主义、注重积累

单词1500+天、刷题500+道、读书1000+小时

* 对待问题有求索精神，常写博客整理反思  
  CSDN单篇博客3w+阅读、300+点赞、700+收藏

之前有过实习经历吗？实习时间大概什么时候？能实习几个月？实习地点XX有问题吗？

6月初能够到岗Landing。

至少3个月，后续可以根据具体情况实习更久。

HashMap源码及其扩容机制？

**首先，从HashMap的底层结构、扩容阈值、扩容条件说起：**

底层结构是数组+链表，Java8后引入红黑树，当链表长度≥8且数组长度≥64时，链表会转换为红黑树。

假设初始容量为 16、负载因子为 0.75的话，那么扩容阈值16\*0.75=12，当存入第13个元素时扩容为2倍。

Java7超过扩容阈值 && 发生哈希冲突时扩容。

Java8超过扩容阈值即可扩容。还有一种情况，哈希冲突链表长度≥8且数组长度＜64，优先扩容而不是转为红黑树。

**Rehash是HashMap扩容的关键步骤：**

1）新建一个2倍大小的数组。

2）条件判断e.hash & oldCap，0接在loTail后、1接在hiTail后。

HashMap数组长度是2的次方，且扩容为2倍，因此用位运算代替取模，计算好的hash值 & (数组长度-1) ，这是前置基础。

举例，扩容前数组长度16，扩容后32，“数组长度-1”由“4个1”变为“5个1”，其区别相当于将扩容前长度的16的二进制表示作为Mask看hash值对应位置是否为1。

（追问）简单说一下扩容机制存在的问题？

1. **哈希值分布不均问题**：即使扩容，部分桶中的元素数量仍然较多，影响性能。
2. **Rehash遍历开销问题**：当元素数量庞大时开销过大。
3. **频繁扩容问题**：负载因子过小会频繁扩容，可能导致内存不足。
4. **线程安全问题**：HashMap并非线程安全，Java7多线程扩容可能导致环形链表，进而导致死循环。

（追问）HashMap和ConcurrentHashMap的区别？

Java 7采用分段锁（Segment）分成多个段，每个段有自己的锁，不同的线程可以同时访问不同的段。

Java 8采用 CAS + synchronized ，当插删时，首先尝试CAS，如果失败则使用 synchronized锁住对应的桶，锁粒度更细。

（追问）CAS是什么机制？

**CAS（Compare And Swap）**由 CPU 硬件指令保证原子性的无锁并发​：

比较当前值与预期值，相等则将新值写入，否则不操作。

1. ​使用版本号或时间戳解决**ABA 问题**
2. ​限制自旋次数解决**不断自旋重试消耗 CPU**的问题
3. ​封装为复合操作解决**无法直接支持多个变量的原子操作**的问题

SpringAOP原理

AOP面向切面编程：**切面 = 切点 + 通知**

* **切面（Aspect）：**封装了通用功能，比如权限、日志、事务、性能
* **切点（Pointcut）：**通过切点表达式找到想要织入weaving的方法
* **通知（Advice）：** 拦截到目标对象的连接点之后**要执行的**增强逻辑

（追问）Spring AOP代理具体如何实现？

**JDK动态代理（**SpringFramework**）**

* 基于接口，通过Java的反射机制实现
* 适合实现接口的的类

**CGLIB动态代理（**SpringBoot2**）**

* 基于类继承，通过字节码生成工具生成子类
* 不能代理 final 类和 final 方法

MySQL事务隔离级别？

**数据一致性问题：**

* **脏读：**一个事务读取到另一个事务未提交的数据。如果该未提交事务最终被回滚，那么第一个事务读取的数据就是脏的。
* **不可重复读：**在同一事务中，读取同一数据两次，但由于其他事务的提交，读取的结果不同。
* **幻读：**在同一事务执行相同的查询操作，返回的结果集记录数量由于其他事务的插入而发生变化。

**事务隔离级别：**

* **读未提交：**可能发生脏读、不可重复读、幻读现象
* **读已提交：**解决脏读（即不会读到未提交事务的数据）
* **可重复读（MySQL InnoDB默认）**：解决不可重复读
* **串行化**：解决幻读（即数据记录数量前后一致）

什么情况下需要分库分表？

1. 单表数据量过大
2. 表结构复杂（字段过多或索引过多）
3. 并发访问量过高

（追问）如何分库分表

* 水平 拆分 数据行：相同表结构复制
* 垂直 拆分 字段：根据业务划分

分 2 个库以及对应业务 16 张表：

Ds\_0、ds\_1

分片键选择

订单ID+用户ID的复合分片键

创建订单id的时候把用户id的后六位冗余到订单id的尾部，这样在查询时，既可以传用户id也可以传订单id，统一取后六位作为分片键。

分片键只是用来定位具体的表，避免读请求扩散。当定位到具体的表之后，还是需要根据传入的完整查询条件建立的索引来查找具体的记录。

为了保证订单号生成递增，我们参考雪花算法自定义了一个 DistributedIdGenerator，生成后的分布式 ID 再拼接上用户的后六位。

（追问）实际如何通过用户ID找到订单？

首先判断 SQL 是否包含用户 ID，如果包含直接取用户 ID 后六位

如果对订单中的 SQL 语句不包含用户 ID 那么就要从订单号中获取后六位，也就是用户 ID 后六位

由于按照后六位分片，所以同一用户自己的所有订单都在一个分片里，通过后六位找到这个分片之后，再去通过**完整的用户 ID** 在这个分片里查询同一用户的所有订单，而不会导致读扩散问题（也就是不用再到其他分片里查找）。

虽然还是拿两个字段作为了分片键，但是由于**自定义了分片算法**，所以无论 SQL 中只有用户 ID，还是只有带用户后六位的订单号，都能找到对应的分片。

Q：那既然我创建订单id的时候把用户id的后六位冗余到订单id的尾部，为什么分片键还选择了shardingColumns: user\_id,order\_sn？不能只是order\_sn？

A：确保 **数据的精确分布** 和 **查询的灵活性。**  
user\_id, order\_sn 的复合分片键可以确保分片策略能够根据这两个字段决定数据存储的位置，而不仅仅是 order\_sn。即使你在 order\_sn 的尾部冗余了用户的后六位，分片策略并不会依赖于这种冗余，而是依据 **完整的复合分片键** 来进行分片，以确保在数据分布、查询和性能优化时都能够一致且高效。**user\_id 和 order\_sn 的复合分片键** 可以确保一个用户的所有订单都在同一个分片，避免了跨分片查询的问题。

如何保证订单ID唯一性？

分库分表后为什么选择雪花算法作为ID？

全局唯一：时间戳 + 机器ID + 序列号

有序性：ID按时间递增

高性能：本地生成、速度快，无需依赖数据库或外部服务

如何保障雪花算法在大规模集群下生成不重复？

确保每个节点的机器ID唯一：

集中分配：使用Zookeeper或Redis集中分配和管理机器ID。

配置文件：在部署时为每个节点配置唯一的机器ID。

启动检测：在节点启动时检测机器ID是否冲突。

（追问）分布式雪花算法如何把用户ID拼到订单ID后面？会不会超过Long最大值？

通过 DISTRIBUTED\_ID\_GENERATOR.generateId() 生成雪花算法生成的全局唯一 ID，并将 **userId % 1000000** 作为用户 ID 的后六位拼接到这个订单 ID 后。

雪花算法生成的 ID 是一个 **64 位的长整型（long）**，通常采用以下结构：

* 1 位：符号位（通常为 0）
* 41 位：时间戳（毫秒级）
* 10 位：工作机器 ID
* 12 位：序列号

通过这些字段，雪花算法可以生成唯一且递增的 ID。

**拼接用户 ID 后六位**：

* 代码中 String.valueOf(userId % 1000000) 取的是用户 ID 的 **后六位**。例如，假设 userId = 123456789，那么 userId % 1000000 的结果是 345789（即用户 ID 的后六位）。
* 然后将该六位数字拼接到雪花生成的订单 ID 后面，形成最终的订单 ID。

**是否会超过 Long 最大值？**

* **雪花算法生成的 ID** 本身是一个 **64 位的 long 类型数字**。在 Java 中，long 类型的最大值为 9223372036854775807，即约为 9.2\*10^18。
* 拼接操作是将用户 ID 的后六位（最多是 6 位数字，范围是 0 到 999,999）作为一个字符串加到雪花算法生成的数字后面。拼接后会变成一个字符串类型，而不是 long 类型。

例如，如果雪花生成的订单 ID 是 123456789012345678，而用户 ID 后六位是 345789，那么最终的订单 ID 字符串就是 123456789012345678345789，它实际上并不会超过 long 的最大值，因为拼接后是字符串类型。

**结论：不会超过 long 的最大值**，因为雪花算法生成的是 long 类型的 ID，而拼接的是用户 ID 后六位的字符串。最终生成的是一个字符串，**不涉及数值类型溢出**。

Redis你是怎么用的？

Redis进行车次查询，在前端进行筛选

缓存穿透问题怎么解决？

缓存穿透 ：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

缓存穿透的常见方案

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。

12306解决注册穿透：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

Q：用户频繁申请后注销导致Redis Set。

Redis大Key问题是指在Redis中存在单个Key对应的Value数据量过大。

**A：**

1. 限制1个证件号最多注销5次
2. 对缓存进行分片处理，根据用户名的HashCode进行取模操作，将数据分散存储在多个Set结构中。

Q：现在redis中有一个大key，如何平滑拆分？

**A：**

双写：大的redis key正常写入，同时还要写一个拆分后的。在某一个时机两边数据能够对齐的情况下，从大 Key迁移到新Key上。

（追问）什么是缓存击穿？如何避免？

Redis缓存穿透

穿透不存在：指查询一个不存在的数据，缓存中没有相应的记录，每次请求都会去数据库查询，造成数据库负载激增。

解决：

* 使用布隆过滤器，过滤掉不存在的请求，避免直接访问数据库。
* 对查询结果进行缓存，即使是不存在的数据，也可以缓存一个标识，以减少对数据库的请求。

Redis缓存击穿

击穿热点：指某个热点数据在缓存中过期，导致大量请求同时访问数据库，造成数据库负载激增。

解决：

* 使用互斥锁，确保同一时间只有一个请求可以去数据库查询并更新缓存。
* 热点数据永不过期。

Redis 缓存雪崩

多个雪崩：指多个缓存数据在同一时间过期，导致大量请求同时访问数据库，造成数据库负载激增。

解决：

* 采用随机过期时间策略，避免多个数据同时过期。
* 使用双缓存策略，将数据同时存储在两层缓存中，减少数据库直接请求。
  + 步骤1：读取数据时的逻辑
    - 首先尝试从主缓存中读取数据。
    - 如果主缓存中没有数据，则尝试从备份缓存中读取数据。
    - 如果备份缓存也没有数据，则从数据库中读取数据，并同时更新主缓存和备份缓存。
  + 步骤2：写入数据时的逻辑
    - 当有新的数据写入缓存时，同时更新主缓存和备份缓存。
    - 为主缓存设置一个短的过期时间，为备份缓存设置一个较长的过期时间。
    - 当主缓存失效时，备份缓存可以继续提供数据，减少对数据库的直接访问。

OSI七层模型？

OSI/RM七层模型

① 应用层：在应用与网络间作为接口

② 表示层：对数据编码、加密、压缩、转换

③ 会话层

④ 传输层（通信子网与资源子网的分界层）：提供端到端的可靠透明数据传输

TCP协议

⑤ 网络层（通信子网的最高层）：逻辑地址寻址、路由选择

IP协议

⑥ 数据链路层：提供可靠数据传输链路

⑦ 物理层：屏蔽具体设备的差异，使链路层不必关心具体传输介质。

TCP/IP网络四层模型

应用层：提供网络服务，比如 FTP、DNS等

传输层：提供端到端可靠数据流传输服务

网际层：负责异构网或同构网的进程间通信,

将传输层分组封装为数据报格式进行传送,

每个数据报必须包含目的地址和源地址。

网络接口层：是网络访问层，负责与物理网络的连接。

TCP和UDP的区别？

TCP（Transmission Control Protocol）和UDP（User Datagram Protocol）是两种常用的传输层协议，它们在网络通信中有一些重要的区别：

1. 连接性：

TCP： 提供面向连接的通信。在数据传输之前，必须先建立连接，然后进行可靠的数据传输，最后再释放连接。

UDP： 是面向无连接的通信。每个数据包都是独立的，不需要先建立连接，直接发送数据。因此，UDP通信更为轻量，但不保证可靠性。

2. 可靠性：

TCP： 提供可靠的数据传输，通过序列号、确认和重传机制确保数据的完整性和顺序性。如果数据包丢失，TCP会进行重传。

UDP： 不提供可靠性保证，数据包可能会丢失，也不保证数据包的顺序性。对于某些实时性要求高的应用，可以容忍少量的丢包。

6. 连接和断开：

TCP： 建立连接时有三次握手的过程，断开连接有四次挥手的过程。

UDP： 无连接的特性，不需要建立和断开连接的过程。

5. 适用场景：

TCP： 适用于对数据完整性和顺序性有要求的应用，如文件传输、网页访问等。

UDP： 适用于实时性要求高、可以容忍少量数据丢失的应用，如语音通话、视频直播等。

3. 数据传输方式：

TCP： 数据以流的方式进行传输，被划分为小的数据段，然后通过序列号进行重新组装。数据传输是可靠而有序的。

UDP： 数据以数据包的形式传输，每个数据包都是独立的。UDP更适用于需要快速传输的场景，如实时音视频。

4. 头部开销：

TCP： TCP头部相对较大，包含序列号、确认号、窗口大小等字段，用于保证可靠性和流控制。

UDP： UDP头部较小，只包含源端口、目标端口、长度和校验和等基本信息，开销较小。

总体来说，选择TCP还是UDP取决于具体的应用需求。TCP适用于要求可靠性和有序性的场景，而UDP适用于实时性要求高、可以容忍少量数据丢失的场景。在某些应用中，也可以结合使用TCP和UDP，根据不同的数据传输需求选择合适的协议。

进程和线程的区别？

进程：是资源分配的基本单位，一个进程可以包含多个线程

进程有独立的上下文（代码、数据、文件）等，创建和切换开销大

进程之间是相互独立的，需要进程间通信（IPC）机制，如管道、消息队列、共享内存、套接字等

线程：是CPU调度的基本单位，属于进程

线程组内共享资源，但有独立的PC和堆栈，创建和切换开销较小

线程直接读写内存即可，但需要同步机制以避免数据竞争

（追问）如何创建多个线程？

1. 继承 Thread 类

继承 Thread 类，并重写 run() 方法，该方法包含了线程要执行的任务。之后创建该类的实例，调用 start() 方法启动线程。

2. 实现 Runnable 接口

实现 Runnable 接口的 run() 方法，把任务逻辑写在其中。接着创建 Thread 类的实例，将实现了 Runnable 接口的对象作为参数传入，最后调用 start() 方法启动线程。

3. 实现 Callable 接口并结合 FutureTask

Callable 接口的 call() 方法有返回值，能抛出异常。借助 FutureTask 包装 Callable 对象，再将 FutureTask 对象作为参数创建 Thread 对象，最后调用 start() 方法启动线程。

4. 使用线程池

通过 ExecutorService 创建线程池，提交任务到线程池执行。

讲一讲项目

外卖平台主要是为了在实践中深入理解Spring IoC、AOP思想

12306主要是为了学习缓存、消息队列、分库分表等技术

为什么要分页排序？分页排序如何实现？

分页排序在数据处理和展示中具有重要作用，以下是对其原因及实现方式的介绍。

**原因**

* **提升性能**：当数据量较大时，如果一次性获取所有数据并进行排序，会消耗大量的内存和时间。分页排序可以每次只获取部分数据并进行排序，减少内存占用和提高查询效率。
* **优化用户体验**：在网页或应用程序中展示数据时，将数据分页展示可以避免一次性加载过多数据导致页面加载缓慢。同时，用户可以更方便地浏览和定位到自己需要的数据，提高数据的可读性和可操作性。

**实现方式**

不同的数据库系统实现分页排序的方式略有不同，以下以常见的 MySQL 数据库为例进行说明。

* **使用 LIMIT 子句**：LIMIT 子句用于指定要返回的行数。可以结合 ORDER BY 子句先对数据进行排序，再通过 LIMIT 子句限制返回的结果集数量，实现分页排序。例如，要查询按照 id 升序排序的第一页数据，每页显示 10 条记录，可以使用以下语句：

（追问）从SQL底层原理和深度分页问题回答分页到1k页2k页会面临什么问题？如何优化和解决？

从 SQL 底层原理来看，分页通常是通过 LIMIT 子句（以 MySQL 为例）来实现。当分页到 1k 页、2k 页时，会面临以下问题：

**面临的问题**

* **性能问题**
  + **数据扫描量大**：LIMIT 子句是先从数据表中读取数据，然后根据偏移量和行数进行筛选。随着页数增加，偏移量增大，数据库需要扫描大量的数据页才能定位到指定页的数据。例如，每页 10 条数据，1k 页的偏移量就是 10000，数据库需要先扫描前面 10000 条记录，这会消耗大量的时间和资源。
  + **索引失效风险**：在深度分页时，由于需要扫描大量数据，可能会导致索引无法有效发挥作用。数据库可能会选择全表扫描来获取数据，而不是通过索引快速定位，进一步降低查询性能。
* **内存占用问题**：数据库处理查询时，会在内存中存储临时结果集。深度分页查询需要获取大量数据，可能使临时结果集占用过多内存。若内存不足，数据库可能会使用磁盘临时文件来存储数据，这会导致查询性能大幅下降，因为磁盘 I/O 操作比内存操作慢得多。

**优化和解决方法**

* **优化查询语句**
  + **利用覆盖索引**：确保查询语句中使用的列都在索引中，这样数据库可以直接从索引中获取数据，而无需回表查询，减少数据扫描量。例如，如果查询只需要 id、name 列，且这两列在一个联合索引中，那么可以创建覆盖索引来优化查询。
  + **合理使用索引**：分析查询条件，确保在经常用于过滤和排序的列上创建索引。对于深度分页查询，索引可以帮助快速定位到指定页的数据，减少全表扫描的可能性。
* **采用书签式分页**：可以使用上一页的最后一条记录的某个唯一标识（如主键）作为下一页查询的条件，通过 WHERE 子句过滤数据。例如，假设主键是 id，上一页最后一条记录的 id 是 100，每页显示 10 条记录，那么下一页的查询语句可以是 SELECT \* FROM your\_table\_name WHERE id > 100 ORDER BY id LIMIT 10。这样可以避免使用大的偏移量，提高查询性能。
* **优化数据库配置**：适当调整数据库的缓存大小、查询缓存等配置参数，以提高数据库的性能。例如，增加 InnoDB 存储引擎的缓冲池大小，使更多的数据和索引可以缓存在内存中，减少磁盘 I/O 操作。
* **数据分库分表**：如果数据量非常大，可以考虑将数据按照一定的规则进行分库分表。例如，按照时间范围、业务模块等进行划分。这样可以将数据分散到多个数据库或表中，减少单个表的数据量，提高查询性能。在查询时，可以根据具体的查询条件，只查询相关的数据库或表。
* **应用层缓存**：在应用层使用缓存技术，如 Redis 等，将常用的数据缓存起来。对于分页查询，可以将查询结果缓存起来，当用户再次访问相同页面时，直接从缓存中获取数据，减少数据库查询次数。同时，可以设置缓存的过期时间，以保证数据的及时性。

外卖平台的定时任务是什么任务？（区分12306的延时消息）

定时任务，Chron表达式

AI信息？平时学习AI的工具使用？

腾讯云黑客松

Deepseek

清华大学PPT

推理模型 vs 模型

知识掌握得比人多

需求掌握得比人少

提示词工程、编程思路、自我判断

（追问）大模型幻觉是什么？为什么会造成幻觉？

手撕代码

将数字num逆序输出，注意符号和边界

输入：123

输出：321

输入：-456

输出：-654

输入：214647483638

输出：0

（追问）简单说一下思路

214748364？/10

反问和分享

业务内容Base地点和HC分布？

北京

核心本地商业-业务研发平台

以前的到家到店合并成立的业务研发平台

我们属于到家下面的履约平台技术部

笔试过后的面试部门分配是怎么来的？

每天早上10点，各个部门去抢笔试出来的简历，看机缘

纸质简历会处理吗？

不清楚，猜测应该会录入系统

线上投递了，纸质简历意义不大

实习职责

Mentor带领了解整个履约平台技术部的工作、在前一周介绍组内业务、代码规范、开发流程

小的、非核心项目，尝试写一些技术方案设计、代码、单元测试、上线

一般实习3个月的都会写1-2个需求上线

美团学习资料还是挺多的

面试改进点？哪些好的地方继续保持？

项目清晰度、不熟悉

大模型开发：AI兴趣、全拥抱AI、学习积极度

八股基础还可以