一 java基础

1 集合有哪些?数据结构?初始长度?扩容机制?哪些是线程安全的?hashmap的底层原理?

1 基本概念

1. Java中所有的集合类都位于java.util包下，主要由两个接口派生出来，分别是Collection和Map
    - Collection包含了List和Set两大分支。Map是一个映射接口
    - Set 、Map、List可以看做集合的三大类。
2. 遍历集合的工具有Iterator和Enumeration
3. 操作数组集合的两个工具类: Arrays和Collection
4. Java中的集合主要分为四类：
    - List列表：有序的，可重复的；
    - Queue队列：有序，可重复的；
    - Set集合：不可重复；
    - Map映射：无序，键唯一，值不唯一

2 集合的种类

1. Collection
    - List:
        - ArrayList
        - LinkedList
        - vector 1 Stack
        - ArrayQueue
    - Set:
        - HashSet 1 LinkedHashSet

        - SortedSet 1 TreeSet
        - EnumSet
    - Queue
        - Dqueue 1 ArrayDequeue 2 LinkedList
        - PriorityQueue
2. Map
    - HashMap
        - LinkedHashMap
    - TreeMap

3 数据结构

1. List
    - ArrayList: 结构: 数组
        - transient Object[] elementData; // 默认的长度是10
    - LindedList: 结构: 双向链表 private static class Node { E item;// 实际存入的元素 Node next;// 下一个节点的引用 Node prev;// 前一个节点的引用 Node(Node prev, E element, Node next) { this.item = element; this.next = next; this.prev = prev; } }
    - hashSet: 结构: map集合, 哈希表
        - private transient HashMap<E, Object> map;
    - LinkedHashSet: 结构: map集合, 哈希表+双向链表
    - TreeSet: 红黑树
    - Map: 哈希表
    - hashMap: 哈希表
    - LinkedHashMap: 哈希表+双向链表
    - TreeMap: 红黑树

4 初始长度

1. arrayList初始长度10
2. Vector初始长度10
3. hashSet初始长度为16
4. hashmap初始长度为16
5. linkedList 是一个双向链表，没有初始化大小，也没有扩容的机制，就是一直在前面或者后面新增就好

5 扩容机制

1. arraylist: 扩容为原容量的0.5倍+1, 如10扩容后为16
    - 加载因子为1, 即存储数据超过容量时扩容
2. vector: 扩容为原容量的1倍, 如10扩容后为20
    - 加载因子为1
3. hashSet
    - 加载因子为0.75, 即当元素个数超过长度的0.75倍后扩容
    - 每次扩容为原容量的一倍
4. hashmap
    - 加载因子0.75
    - 扩容为原容量的一倍

6 哪些是线程安全的

1. Collection集合中提供了synchronizedxxx(xxx)方法把对应的xxx集合(collection,list, map, set等)集合转换为线程安全的集合
2. vector, vector, hashTable, properties, ConcurrentHashMap是线程安全的
    - vector和ArrayList类似, 但是比ArrayList多了个同步机制(线程安全)
    - hashTable中的每个方法都加了synchronized
    - ConcurrentHashMap:是一种高效但是线程安全的集合

        - 其实现机制是使用了部分锁, 效率上比hashTable的方法上加synchronized锁的效率要高
    - Stack: 栈，也是线程安全的, 继承于Vector
3. ArrayList、LinkedList、HashSet、TreeSet、HashMap、TreeMap等都是线程不安全的

7 hashmap的底层原理

1. 哈希表: 数组+链表+红黑树
2. key的哈希值/数组长度=索引
    - 索引相同的key产生哈希冲突, 其内容存储到该索引对应的链表中
    - 如果在链表中值也相同的话, 则值覆盖
    - 链表长度超过8, 则改为红黑树

8 HashMap底层数据结构

1. 底层数据w结构
    - JDK1.7及之前: 数组+链表
    - JDK1.8: 数组+链表+红黑树
2. HashMap添加元素分析
    - 根据一个元素的哈希值和数组长度计算下标来准确定位该元素的位置
        - 如为了使元素时分布均匀会使用取模运算: index=hashCode % arr.length 1 实际使用的不是取模运算, 而是与运算 1 与运算(n-1) & hash取代取模运算hash%length, 两种方式记算出来的结果是一致的(n就是length) 2 即: (length-1)&hash = hash%length 2 计算机的运行效率: 加法(减法)>乘法>除法>取模, 取模运算效率最低 1 而与运算(位操作)的效率是远远高于取模运算的
        - index即为要插入的元素的数组索引位置
    - 哈希冲突: 两个不同的元素计算后会得出相同的索引位置, 这时就需要链表和红黑树了

        - 冲突的元素会在该索引处以链表(或红黑树)的形式保存
    - 链表: 当链表的长度过长时, 查询效率较低, 就需要使用红黑树来存储
    - 红黑树
        - 红黑树是一棵接近于平衡的二叉树,其查询时间复杂度为O(logn), 远远比链表的查询效率高
        - 但如果链表长度不到一定的阈值, 直接使用红黑树代替链表也是不行的 1 因为红黑树的自身维护的代价也是比较高的 2 每插入一个元素都可能打破红黑树的平衡性 3 这就需要每时每刻对红黑树再平衡(左旋, 右旋, 重新着色)

3. 数组的长度必须是2的指数次幂

- HashMap中数组的初始长度为16
- 空参的HashMap初始容量是16, 默认加载因子为0.75
- 由于一个元素存储时HashMap底层使用了与运算代替了取模运算
    - 只有当数组的长度为2的指数次幂时, 两种方式计算的结果才一样, 否则计算结果不一样
    - 但最重要的一点, 是要保证定位出来的值是在数组的长度之内的, 不能超出数组长度, 并且减少哈希碰撞, 让每个位都可能被取到
        1. 当数组长度为2的指数次幂时
            - (length-1) & hash 的结果可以取到所有的数组索引值
            - 2的指数次幂-1 的二进制的结果无论是几位二进制, 所有的二进制位都是1 如 16-1 = 15, 二进制为 1111 1111 与hash(任意值)的二进制 与运算后, 结果为 ----, "-"表示0或1都可以 这样结果就可以取到 0-15 之间的所有索引值
        2. 当数组长度不为2的指数次幂时
            - (length-1) & hash 的结果不能取到所有的数组索引值
            - 如: (7-1) & hash 的结果在索引范围 0-6之间 但是 7-1 = 6 的二进制(这里使用4位)为: 0110 0110与hash(任意值)的二进制进行与运算, 所得结果必定为 0--0, 首尾二进制和末尾一定为0 这种情况下0-6索引之间的某些索引是无法取到的, 如 0001(1), 0011(3), 0101(5) 即计算结果无法得

出 1, 3, 5, 的索引值, 数组中这些索引值位置也无法存储元素

4. HashMap的加载因子(负载因子)要设置为0.75
    - 加载因子: 即HashMap的数组空间使用了多少时对数组进行扩容
    - 加载因子如果定的太大, 比如1, 即一直等数组填满才扩容
        - 虽然达到了最大的数组空间利用率, 但会产生大量的哈希碰撞, 同时产生更多的链表
    - 如果设置的过小, 比如0.5
        - 保证了数组空间很充足, 减少了哈希碰撞, 这种情况下查询效率很高, 但消耗了大量空间
5. 链表长度大于等于8时转成了红黑树
    - 链表长度大于等于8时转成红黑树正是遵循泊松分布
        - 泊松分布: 适合描述单位时间内随机事件发生的概率
        - 如果红黑树节点数少于6, 再次转换为链表
    - 根据泊松分布计算得出的概率(HashMap源码中提供) 存储元素个数: 概率
        - 0: 0.60653066
        - 1: 0.30326533
        - 2: 0.07581633
        - 3: 0.01263606
        - 4: 0.00157952
        - 5: 0.00015795
        - 6: 0.00001316
        - 7: 0.00000094
        - 8: 0.00000006
    - 根据上述概率可以看到当链表中的元素个数为8时的概率已经很小了
6. 计算机的运行效率: 加法(减法)>乘法>除法>取模
    - 赋值运算(x=y): 0.8ms
    - 加法运算(x+y): 41.45ms

- 减法运算: 42.95ms
  - 可见, 在计算机内部实现中, 把减法变成加法的求补码过程是较快的
- 乘法运算: 58.25ms
- 除法运算: 1210.2ms
- 取模运算: 1178.15ms
- 位操作比 + , -, *, /快

9 hashTable是线程安全的

1. 默认长度为11, 加载因子0.75, 每次扩容为2倍+1
2. 数据结构也是哈希表: 数组+链表

10 ConcurrentHashMap

1. 数据结构
   - jdk1.7其数据结构是: 一个Segment数组和多个HashEntry组成
   - jdk1.8其数据结构是: 哈希表, 即 Node数组+链表+红黑树
2. 桶上链表长度达到 8 个或者以上，并且数组长度为 64 以下时只会触发扩容而不会将链表转为红黑树
3. 加载因子0.75

2 线程的创建?开启?状态?sleep和wait的区别, 线程池?死锁?如何保证线程安全?

1 线程创建的方式

1. 继承Thread类, 重写run()方法
2. 实现Runnable接口, 重写run()方法
3. 实现Callable接口, 重写call方法
4. 使用线程池

2 启动线程, 对应上面四种方式

1. 创建自定义类的对象, 使用start()方法
    - new 自定义类().start()
2. 创建Thread类对象, 把Runnable接口的实现类作为参数传递过去
    - new Thread(new Runnable).start()
3. new Thread(new FutureTask(new Callable())).start();

3 线程状态

1. NEW: 新建状态
2. Runnable: 可运行状态
3. Blocked: 锁阻塞状态
4. Waitting: 无限等待状态
5. Timed waiting: 计时等待状态
6. Terminated: 被终止状态 状态 创建, 就绪, 运行, 阻塞, 死亡
7. 如何停止一个线程?
    1. 没有直接停止线程的方法, 只能等线程运行完毕
    2. 如果要手动停止, 你可以使用volatile布尔变量来退出run()方法的循环或者是取消任务来中断线程

4 sleep和wait的区别

1. sleep: 让当前线程睡眠, 在同步方法或代码块中不释放锁
    - 该方法可以写在任何位置
2. wait: 让当前线程等待, 同时释放已经拿到的锁
    - 该方法必须写在同步方法或同步块中

5 线程池

1. 线程池的组成
    - 线程池管理器（ThreadPool）：用于创建并管理线程池，包括创建线程池，销毁线程池，添加新任务；
    - 工作线程（PoolWorker）：线程池中线程，在没有任务时处于等待状态，可以循环的执行任务；
    - 任务接口（Task）：每个任务必须实现的接口，以供工作线程调度任务的执行，它主要规定了任务的入口，任务执行完后的收尾工作，任务的执行状态等；
    - 任务队列（taskQueue）：用于存放没有处理的任务。提供一种缓冲机制。
2. 一个服务器完成一项任务所需时间为: T1 创建线程时间，T2 在线程中执行任务的时间，T3 销毁线程时间。
    - 如果：T1 + T3 远大于 T2，则可以采用线程池，以提高服务器性能。
3. 线程池类ThreadPool
    - 默认初始化5个工作线程
    - 我们可以通过execute(List task)方法将自己的任务传递进去
        - 或使用其他重载的方法 execute(Runnable task) execute(Runnable[] task)
        - 自己的任务实现Runnable接口, 重写其中的run()方法
        - 但只是把自己的任务加入了线程池, 什么时候执行由线程池决定
4. 线程池有几种
    - Executors.newFixedThreadPool, 创建固定线程数的线程池，使用的是LinkedBlockingQueue无界队列，线程池中实际线程数永远不会变化
    - Executors.newSingleThreadExecutor (单线程线程池),创建只有一个线程的线程池，使用的是LinkedBlockingQueue无界队列，线程池中实际线程数只有一个
    - Executors.newCachedThreadPool (弹性缓存线程池),创建可供缓存的线程池，该线程池中的线程空闲时间超过60s会自动销

毁，使用的是SynchronousQueue特殊无界队列
    - Executors.newScheduledThreadPool (定时器线程池),创建可供调度使用的线程池（可延时启动，定时启动），使用的是DelayWorkQueue无界延时队列
    - Executors.newWorkStealingPool 拥有多个任务队列的线程池,jdk1.8提供的线程池，底层使用的是ForkJoinPool实现，创建一个拥有多个任务队列的线程池，可以减少连接数，创建当前可用cpu核数的线程来并行执行任务
    - ForkJoinPool
5. 线程池参数 corePoolSize,线程池中的常驻核心线程数 int maximumPoolSize,线程池能够容纳同时执行的最大线程数，此值必须大于等于1 long keepAliveTime,多余的空闲线程的存活时间。TimeUnit unit,keepAliveTime的单位 BlockingQueue workQueue,任务队列，被提交但尚未被执行的任务 ThreadFactory threadFactory,表示生成线程池中工作线程的线程工厂，用于创建线程一般默认即可 RejectedExecutionHandler handler拒绝策略，表示当队列满了并且工作线程大于等于线程池最大线程数（maximumPoolSize）时如何处理

6 死锁

1. 死锁介绍: 死锁就是线程1需要请求的资源被线程2持有, 而线程2想要获取的资源被线程1持有
    - 两个线程都不释放自己的资源, 但又无法获取所需的资源而进入阻塞的一种状态
    - 死锁是指两个或两个以上的进程在执行过程中，由于竞争资源或者进程在运行过程中，请求和释放资源的顺序不当而造成的一种阻塞的现象，若无外力作用，它们都将无法推进下去
2. 死锁产生的条件
    - 有多把锁
    - 有多个线程
    - 有同步代码块嵌套, 即一个线程完成一个任务可能需要拿到多把锁
3. 死锁解决方法

    - 杀死一个或多个进程(最简单的方法)
    - 抢占: 即从一个线程中获取资源, 将其分配给其他线程, 但可能会导致一些问题
    - 回滚: 系统可以定期记录每个进程的状态, 在产生死锁时, 将所有内容回滚到最后一个检查点
        - 然后重新启动, 但以不同的方式分配资源, 以免发生死锁
    - 按照顺序加锁
        - 即如果有两把锁, 必须先获取第一把锁之后才能获取第二把锁
        - 不能直接先获取第二把锁
    - 不要用synchronized无限等待, 可以使用ReentranLock实现超时等待
        - 即拿到第一把锁之后,在一定时间后拿不到第二把锁,就释放已经拿到的锁
4. 活锁: 死锁的变体
    - 例子: 两个人在走廊上面对面相遇, 每个人一边走一边让对方通过，但最终却在没有任何进展的情况下左右摇摆, 因为他们总是以相同的方式移动同时
5. 只有一个锁, 如果这个锁是互斥锁时, 也可能会产生死锁 如果同一个线程先后两次调用lock，在第二次调用时，由于锁已经被占用，该线程会挂起等待别的线程释放锁； 然而锁正是被自己占用着的，该线程又被挂起而没有机会释放锁,。因此就永远处于挂起等待状态了

7 如何保证线程安全?

1. 线程安全在三个方面体现
    - 原子性: 供互斥访问，同一时刻只能有一个线程对数据进行操作，（atomic,synchronized）
    - 可见性: 一个线程对主内存的修改可以及时地被其他线程看到，（synchronized,volatile）
    - 有序性: 一个线程观察其他线程中的指令执行顺序，由于指令重排序，该观察结果一般杂乱无序
2. 解决方法

- 使用原子类
  - 在Java.util.concurrent.atomic包下定义了一些对"变量"操作的"原子类"
  - 原子类又称为乐观锁, cas机制 compare and swap
- 使用 synchronized 关键字
- 使用 Lock 锁 lock()获得锁 unlock()释放锁

3 ==和equals的区别?

1. 介绍: 值类型是存储在内存中的栈中的, 而引用类型的变量在栈中存储的只是一个地址值

而这个引用类型中的具体数据是存储在堆中的

2. ==操作比较的是两个变量的值是否相等对于引用型变量表示的是两个变量在堆中存储的地址是否相同, 即栈中的内容是否相同
3. equals操作表示的两个变量是否是对同一个对象的引用，即堆中的内容是否相同
4. ==比较的是两个对象的地址，而equals比较的是两个对象的内容

4 对反射的理解? 获取class类的方式?如何用反射获取私有属性的Field对象?

1 反射的理解

1. 该机制是指在运行时去获取一个类的变量和方法信息。然后通过获取到的信息来创建对象,调用方法的一种机制。
2. 反射的定义: 计算机科学中的反射（reflection）指计算机程序在运行时（runtime）
   - 可以访问、检测和修改它本身状态或行为的一种能力。
   - 通俗说，反射就是程序在运行的时候能够“观察”并且修改自己的行为，
   - 是程序对自身的反思、自检、修改。英文 reflection 本身就有反思的意思。

    - 而反思一词听起来让程序有了生命力，也说明了反射只有在程序运行时才能起作用，
    - 因为程序代码只有在CPU上“跑”起来才有“生命力”。
3. 正射: 在Java程序中，在没有用到反射相关的类的时候，我们都是在做正射
4. 在写程序的时候, 我们在调用某个方法时, 会自己主动思考该类中是否有这个方法
    - 而反射做的事就是将这种“思考”交给程序自身, 由程序在运行时确定类中是否有我们需要调用的方法, 进而去调用它
5. 反射的作用，即在运行时访问、修改对象的状态和行为

2 获取class类的方式

1. 调用运行时类本身的.class属性: Class<类名> c = 类名.class;
2. 通过运行时类的对象获取: Class c = 对象.getClass();
3. 通过Class的静态方法获取:体现反射的动态性: Class c = Class.forName(全限定类名);
4. 通过类的加载器 ClassLoader classLoader = this.getClass().getClassLoader();// 获取类加载器 Class c = classLoader.loadClass(全限定类名);// 加载类

3 如何使用反射获取类的私有变量的Field对象: 使用暴力反射

1. 先获取类的class对象: c
2. 获取变量的的Field数组: Field[] fields = c.getDeclaredFields();
3. 变量数组, 在调用Field类中的方法之前, 先使用暴力反射方法 setAccessible, 开启私有方法的访问权限
    - field.setAccessible(true);值为 true 则指示反射的对象在使用时应该取消 Java 语言访问检查
    - 然后在调用field中的其他方法

5 常用的设计模式有哪些? 在项目中哪里用到了? 单例中懒汉饿汉的优缺点?

1 常用的设计模式有哪些

1. 模板模式 jdbcTemplate
2. 装饰设计模式
    - Java IO的API是典型的应用。对扩展开放，对修改关闭
3. 单例设计模式
    - 在spring的IOC依赖注入的时候，默认的是采用单例模式
    - 多线程中线程池的设计一般也采用单例设计模式，这是由于线程池要方便对池中的线程进行控制
4. 多例设计模式
5. 动态代理模式
    - spring中的AOP
    - 基于接口的和基于子类(cglib)的动态代理
    - 目标接口实现了接口, 则默认使用jdk动态代理, 即接口的动态代理, 但是可以强制使用cglib动态代理
        - 如果目标对象没有实现接口, 只能用cglib动态代理
6. 工厂设计模式 BeanFactory, ApplicationContext 观察者模式 定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并被自动更新 spring中Observer模式常用的地方是listener的实现。如ApplicationListener。发布者-订阅者模式，接受通知

2 单例中懒汉饿汉的优缺点

1. 懒汉式
    - 懒汉式是典型的时间换空间
        - 每次获取实例都会进行判断，看是否需要创建实例，浪费判断的时间

        - 当然，如果一直没有人使用的话，那就不会创建实例，则节约内存空间
    - 懒汉式的线程安全, 需要自己在获取对象的方法上加上synchronized
2. 饿汉式
    - 饿汉式是典型的空间换时间
        - 当类装载的时候就会创建类实例，不管你用不用，先创建出来, 但是浪费空间
        - 每次调用的时候，就不需要再判断了，节省了运行时间
    - 饿汉式是线程安全的，因为虚拟机保证只会装载一次，在装载类的时候是不会发生并发的

6 jdk1.8的新特性有哪些?

1 Lambda表达式

2 函数式接口

1. Function类型接口
2. Consumer系列
3. Predicate系列
4. Supplier系列

3 方法引用和构造器调用

4 Stream API: 并行流和串行流

5 接口中的默认方法和静态方法

1. 可以通过parallel()与sequential()方法在并行流和串行流之间进行切换
2. jdk1.8并行流使用的是fork/join框架进行并行操作
    - Fork/Join 框架：就是在必要的情况下，将一个大任务，进行拆分(fork)成若干个小任务（拆到不可再拆时），再将一个个的小

任务运算的结果进行 join 汇总。
    - 关键字：递归分合、分而治之。
    - 采用 “工作窃取”模式（work-stealing）:当执行新的任务时它可以将其拆分分成更小的任务执行，并将小任务加到线 程队列中，然后再从一个随机线程的队列中偷一个并把它放在自己的队列中

6 新时间日期API

7 hashMap数据结构的优化, 添加了红黑树

8 Optional

1. Optional仅仅是一个容器，可以存放T类型的值或者null。它提供了一些有用的接口来避免显式的null检查

9 并行数组

1. 最重要的方法是parallelSort()，可以显著加快多核机器上的数组排序

7 session的实现原理? session的生命周期? session如何存储数据?

1 session的实现原理

1. 客户对向服务器端发送请求后，Session 创建在服务器端，返回Sessionid给客户端浏览器保存在本地，
2. 当下次发送请求的时候，在请求头中传递sessionId获取对应的从服务器上获取对应的Sesison

2 session的生命周期

1. 一次会话范围, 即打开浏览器到关闭浏览器
2. 注意:

- 关闭浏览器后session并不是销毁了, session是保存在服务器端的
  - 关闭浏览器只是销毁了客户端保存的sessionid
  - 只要拿到这个id, 那么还可以再次获取到对应的session信息
- 一个浏览器只有一个session

3 session如何存储数据

1. 在服务器端有一个session池，用来存储每个用户提交session中的数据

8 类加载机制? 代码块的执行顺序?

1 类加载机制

1. 虚拟机把描述类的数据从Class文件加载到内存，并对数据进行校验、转换解析和初始化，最终形成可以被虚拟机直接使用的Java类型，这就是虚拟机的类加载机制
2. 类的整个生命周期包括: 加载（Loading）、验证（Verification）、准（Preparation）、解析（Resolution）、初始化（Initialization）、使用（Using）和卸载（Unloading）7个阶段
3. 其中验证、准备、解析3个部分统称为连接（Linking）
4. 加载、验证、准备、初始化和卸载这5个阶段的顺序是确定的, 类的加载过程必须按照这种顺序按部就班地开始
   - 而解析阶段则不一定：它在某些情况下可以在初始化阶段之后再开始，这是为了支持Java语言的运行时绑定（也称为动态绑定或晚期绑定）

2 代码块的执行顺序

1. 静态代码块: 在当前类第一次被使用时, 会执行该代码块
2. 构造代码块: 在执行类的构造方法之前执行
3. 局部代码块: 在方法内部, 执行方法时执行

9 cookie和session的区别?

1. cookie: 客户端技术, 数据保存在客户端, 数据不安全, 存储数据大小4kb
2. session: 服务器端技术, 数据保存在服务器, 数据相对安全, 存储数据大小根据服务器容量确定

10 java中字符串方法有哪些? string stringbuild stringbuffer的区别?

1 java中字符串方法

1. 获取字符串长度: int length()
2. 比较字符串内容
    - boolean equals(Object anObject)
    - boolean equalsIgnoreCase(String anotherString)
3. 根据索引获取字符:char charAt(int index)
4. 截取字符串中的一部分
    - String substring(int beginIndex)
    - String substring(int beginIndex, int endIndex)
5. 大小写转换
    - toUpperCase()
    - toLowerCase()
6. 替换字符串中的内容
    - public String replace(String s, String replacement)
    - public String replaceAll(String regex,String replacement)
7. 拼接字符串:String concat(String str)
8. 判断字符串是否为空:boolean isEmpty()
9. 判断字符串中是否包含给定的字符串:boolean contains(CharSequence s)
10. 判断字符串是否以给定的字符串结尾:boolean endsWith(String suffix)

11. 判断字符串是否以给定的字符串开头:boolean startsWith(String prefix)
12. 切分字符串:String[] split(String regex)
13. 移除首尾空格:trim()
14. 返回字符串对应的字节数组

- byte[] getBytes()
- byte[] getBytes(Charset charset)

15. 把字符串转换为字符数组:char[] toCharArray();
16. 返回指定字符串在此字符串中第一次出现的索引位置: public int indexOf(String str)

2 string stringbuild stringbuffer的区别

1. String的值是不可变的，这就导致每次对String的操作都会生成新的String对象
    - 这样不仅效率低下，而且大量浪费有限的内存空间
    - 例: string s = "123", s = s + "234"
        - s的变化是指针的变化, 首先指针指向"123", 系统会给"123"开辟一个空间
        - s = s + "234"; "234"开辟一个空间, "123234"有开辟一个空间
        - 最终s指向"123234"的空间地址
        - 这样短短的两个字符串，却需要开辟三次内存空间，不得不说这是对内存空间的极大浪费
2. 当对字符串进行修改的时候，需要使用 StringBuffer 和 StringBuilder 类
    - 和 String 类不同的是，StringBuffer 和 StringBuilder 类的对象能够被多次的修改，并且不产生新的未使用对象。
3. StringBuilder 类在 Java 5 中被提出
4. string stringbuild stringbuffer 的区别
    - 区别一:
        - string: 不可变字符序列

        - stringbuild: 可变字符序列, 线程不安全, 执行速度快
        - stringbuffer: 可变字符序列, 线程安全, 执行速度慢
    - 字符修改上的区别
    - 初始化上的区别
        - String可以赋值为null, 另外两个不可以, 会报错
5. 使用方式
    - 如果要操作少量的数据用 String
    - 多线程操作字符串缓冲区下操作大量数据 StringBuffer
    - 单线程操作字符串缓冲区下操作大量数据 StringBuilder

11 jvm调优和垃圾回收机制?

jvm调优知识简介

堆与栈的概念 堆和栈是程序运行的关键：栈是运行时的单位，而堆是存储的单位。 栈解决程序的运行问题，即程序如何执行，或者说如何处理数据；堆解决的是数据存储的问题，即数据怎么放、放在哪儿。 在Java中一个线程就会相应有一个线程栈与之对应，而堆则是所有线程共享的 栈存储的信息都是跟当前线程（或程序）相关信息的，包括：局部变量、程序运行状态、方法返回值等等。 堆只负责存储对象信息。 简单来说：堆中存的是对象，栈中存的是基本数据类型和堆中对象的引用。

jvm调优: 性能调优

1. 性能调优的目的
    - 减少垃圾回收执行的次数, 以及执行的时间
    - 使用较小的内存占用来获得较高的吞吐量或者较低的延迟
2. 性能调优的手段
    - 使用JDK提供的内存查看工具，如JConsole和Java VisualVM
    - 控制堆内存各个部分所占的比例
    - 采用合适的垃圾收集器
3. 假设线程池、连接池、程序代码等都已经做过优化，效果（系统吞吐量、响应性能）仍然不理想，我们就可以考虑JVM调优了

4. 什么情况下需要做jvm调优？
    - heap 内存（老年代）持续上涨达到设置的最大内存值；
    - Full GC 次数频繁；
    - GC 停顿时间过长（超过1秒）；
    - 应用出现OutOfMemory 等内存异常；
    - 应用中有使用本地缓存且占用大量内存空间；
    - 系统吞吐量与响应性能不高或下降
5. JVM调优原则
    - 多数的Java应用不需要在服务器上进行JVM优化；
    - 多数导致GC问题的Java应用，都不是因为我们参数设置错误，而是代码问题；
    - 在应用上线之前，先考虑将机器的JVM参数设置到最优（最适合）；
    - 减少创建对象的数量；
    - 减少使用全局变量和大对象；
    - JVM优化是到最后不得已才采用的手段；
    - 在实际使用中，分析GC情况优化代码比优化JVM参数更好；
6. JVM调优目标
    - GC低停顿
    - GC低频率
    - 低内存占用
    - 高吞吐量
7. JVM调优量化目标（示例）
    - Heap 内存使用率 <= 70%;
    - Old generation内存使用率<= 70%;
    - Full gc 次数0 或 avg pause interval >= 24小时 ;
    - 注意：不同应用，其JVM调优量化目标是不一样的。
8. JVM调优经验
    1. JVM调优经验总结 JVM调优的一般步骤为： 第1步：分析GC日志及dump文件，判断是否需要优化，确定瓶颈问题点； 第2步：确定JVM调优量化目标 第3步：确定JVM调优参数（根据历史JVM参数来调整）； 第4步：调优一台服务器，对比观察调优前后的差异； 第5步：不断的分析和调整，直到找到合适的JVM

参数配置； 第6步：找到最合适的参数，将这些参数应用到所有服务器，并进行后续跟踪。

2. JVM调优重要参数解析 注意：不同应用，其JVM最佳稳定参数配置是不一样的。 配置： -server 重要参数（可调优）解析： -Xms12g：初始化堆内存大小为12GB。 -Xmx12g：堆内存最大值为12GB 。 -Xmn2400m：新生代大小为2400MB，包括Eden区与2个Survivor区。 -XX:SurvivorRatio=1：Eden区与一个Survivor区比值为1:1。 XX:MaxDirectMemorySize=1G：直接内存。报java.lang.OutOfMemoryError: Direct buffer memory 异常可以上调这个值。 -XX:+DisableExplicitGC：禁止运行期显式地调用 System.gc() 来触发fulll GC。 注意: Java RMI的定时GC触发机制可通过配置-Dsun.rmi.dgc.server.gcInterval=86400来控制触发的时间。 -XX:CMSInitiatingOccupancyFraction=60：老年代内存回收阈值，默认值为68。 -XX:ConcGCThreads=4：CMS垃圾回收器并行线程线，推荐值为CPU核心数。 -XX:ParallelGCThreads=8：新生代并行收集器的线程数。 -XX:MaxTenuringThreshold=10：设置垃圾最大年龄。如果设置为0的话，则年轻代对象不经过Survivor区，直接进入年老代。对于年老代比较多的应用，可以提高效率。如果将此值设置为一个较大值，则年轻代对象会在Survivor区进行多次复制，这样可以增加对象再年轻代的存活时间，增加在年轻代即被回收的概论。 -XX:CMSFullGCsBeforeCompaction=4：指定进行多少次fullGC之后，进行tenured区 内存空间压缩。 -XX:CMSMaxAbortablePrecleanTime=500：当abortable-preclean预清理阶段执行达到这个时间时就会结束。

JVM调优工具

1. console: jdk自带，功能简单，但是可以在系统有一定负荷的情况下使用。对垃圾回收算法有很详细的跟踪。详细说明参考这里
2. JProfiler: 商业软件，需要付费。功能强大。详细说明参考这里
3. VisualVM: JDK自带，功能强大，与JProfiler类似。推荐

jvm调优参数

1. -Xmx4g：堆内存最大值为4GB
2. -Xms4g：初始化堆内存大小为4GB
3. -Xmn1200m：设置年轻代大小为1200MB
4. -Xss512k：设置每个线程的堆栈大小
5. -XX:NewRatio=4：设置年轻代（包括Eden和两个Survivor区）与年老代的比值（除去持久代）,设置为4，则年轻代与年老代所占比值为1：4，年轻代占整个堆栈的1/5
6. -XX:SurvivorRatio=8：设置年轻代中Eden区与Survivor区的大小比值。设置为8，则两个Survivor区与一个Eden区的比值为2:8，一个Survivor区占整个年轻代的1/10
7. -XX:PermSize=100m：初始化永久代大小为100MB。
8. -XX:MaxPermSize=256m：设置持久代大小为256MB。
9. -XX:MaxTenuringThreshold=15：设置垃圾最大年龄。如果设置为0的话，则年轻代对象不经过Survivor区，直接进入年老代

- 对于年老代比较多的应用，可以提高效率
    - 如果将此值设置为一个较大值，则年轻代对象会在Survivor区进行多次复制，这样可以增加对象再年轻代的存活时间，增加在年轻代即被回收的概论

垃圾回收机制

0. JVM内存由几个部分组成：堆、方法区、栈、程序计数器、本地方法栈
    - JVM垃圾回收仅仅针对公共内存区域即：堆和方法区进行
1. 分代管理
    - 将堆和方法区按照对象不同年龄进行分代
        - 堆中会频繁创建对象，基于一种分代的思想，按照对象存活时间将堆划分为新生代和旧生代两部分
        - 1 我们不能一次垃圾回收新生代存活的对象就放入旧生代，而是要经过几次GC后还存活的对象，我们才放入旧生代

        - 2 所以我们又把新生代再次划分为Eden区和两个Survivor区
        - 3 让对象创建在Eden区
        - 4 然后在两个Survivor之间反复复制，最后仍然存活的对象才复制到旧生代中。
        - 方法区存放的是常量、加载的字节码文件信息等

          信息相对稳定。因为不会频繁创建对象，所以不需要分代，直接GC即可。
    - 新生代
        - 1.所有新对象创建发生在Eden区，Eden区满后触发新生代上的minor GC，将Eden区和非空闲Survivor区存活对象复制到另一个空闲的Survivor区中。
        - 2.永远保证一个Survivor是空的，新生代minor GC就是在两个Survivor区之间相互复制存活对象，直到Survivor区满为止。
        - 经过大概15次后, 还存活的对象会gc到老年代
    - 旧生代(老年代)
        - Eden区满后触发minor GC将存活对象复制到Survivor区，Survivor区满后触发minor GC将存活对象复制到旧生代。
        - 经过新生代的两个Survivor之间多次复制，仍然存活下来的对象就是年龄相对比较老的，就可以放入到旧生代了，随着时间推移， 如果旧生代也满了，将触发Full GC，针对整个堆（包括新生代、旧生代和持久代）进行垃圾回收。
    - 持久代(1.5之前)(之后就改为元空间了) 持久代如果满，将触发Full GC
    - 建议：
    - 如果新生代太小，会导致频繁GC，而且大对象对直接进入旧生代引发full gc
    - 如果新生代太大，会诱发旧生代full gc，而且新生代的gc耗时会延长
    - 建议新生代占整个堆1/3合适

2. 垃圾回收 要执行gc关键在于两点，一是检测出垃圾对象，二是释放垃圾对象所占用的空间。
    - 检测垃圾对象 检测出垃圾对象一般有两种算法：引用计数法, “可达性分析”算法
        - 1 引用计数法因为无法检测对象之间相互循环引用的问题，基本没有被采用 现在主流的语言的垃圾收集中检测垃圾对象主要还是“可达性分析”方法
        - 2 “可达性分析”算法描述？
            - 通过一系列的名为“GC Root”的对象作为起点，从这些节点向下搜索，
            - 1 搜索所走过的路径称为引用链(Reference Chain)，
            - 2 当一个对象到GC Root没有任何引用链相连时，则该对象不可达，该对象是不可使用的，垃圾收集器将回收其所占的内存。
            - 所以JVM判断对象需要存活的原则是：能够被一个根对象到达的对象。
            - 什么是能够到达呢: 就是对象A中引用了对象B，那么就称A到B可达。
        - GCRoot对象集合？ a.java虚拟机栈(栈帧中的本地变量表)中的引用的对象。 b.方法区中的类静态属性引用的对象。 c.方法区中的常量引用的对象。 d.本地方法栈中JNI本地方法的引用对象。
    - 释放空间 垃圾回收算法: 在检测出垃圾对象之后，需要按照特定的垃圾回收算法进行内存回收 复制(Copying) 标记-清除(Mark-Sweep) 标记-整理(Mark-Compact) 分代(Generational Collection)，借助前面三种算法实现

12 java中锁的种类和基本原理?

1 synchronized同步锁(对象锁)

1. 同一时刻, 一个同步锁只能被一个线程访问
2. 以对象为依据，通过synchronized关键字来进行同步，实现对竞争资源的互斥访问
3. 哪个线程先执行带synchronized关键字的方法或synchronized代码块，哪个线程就有该方法或该代码块所持有的锁，其他线程只能呈现等待状态，
    - 前提是多个线程访问同一个对象
4. synchronized的用途
    - 锁方法
        - synchronized修饰普通方法 :在修饰方法的时候默认是当前对象作为锁的对象
        - Java中每个对象都有一个锁或者称为监视器，当访问某个对象的synchronized方法时，表示将该对象上锁，而不仅仅是为该方法上锁。
        - 这样如果一个对象的synchronized方法被某个线程执行时，其他线程无法访问该对象的任何synchronized方法（但是可以调用其他非synchronized的方法）。直至该synchronized方法执行完 1 但是如果在synchronized方法中调用的wait方法, 则其他线程能方位该对象的其他synchronized方法 1 原因为: 执行wait后会线程会释放自己已经获得的锁 2 但如果使用sleep()方法是, 不能访问其他的synchronized方法, sleep不释放锁
    - 静态的synchronized方法调用情况
        - 使用的锁是synchronized方法所在对象对应的Class对象
        - 当调用一个对象的静态synchronized方法时，它锁定的并不是synchronized方法所在的对象，而是synchronized方法所在对象对应的Class对象。这样，其他线程就不能调用该类的其他静态synchronized方法了，但是可以调用非静态的synchronized方法
    - 锁代码块

        - 使用synchronized创建同步代码块： 在修饰代码块的时候需要一个reference对象作为锁的对象.
        - synchronized同步代码块只是锁定了该代码块，代码块外面的代码还是可以被访问的, 同样的方法也只是普通的非同步方法
    - 修饰类时候: 在修饰类时候默认是当前类的Class对象作为锁的对象
        - 即在使用synchronized修饰代码块时, 用"类名.class"作为锁对象
5. 同步锁又成为悲观锁
6. synchronized关键字不能继承, 即子类中重写的父类的带有synchronized方法后, 子类中的方法不是同步方法, 而是一个普通方法
    - 为什么不能被继承?虽然可以使用synchronized来定义方法，但synchronized并不属于方法定义的一部分
7. 注意:
    - 在定义接口方法时不能使用synchronized关键字
    - 构造方法不能使用synchronized关键字，但可以使用synchronized代码块来进行同步
    - 实现同步是要很大的系统开销作为代价的，甚至可能造成死锁，所以尽量避免无谓的同步控制
8. 总结
    - 作用于方法时，锁住的是对象的实例(this)
    - 当作用于静态方法时，锁住的是Class实例
    - synchronized 作用于一个对象实例时，锁住的是所有以该对象为锁的代码块
    - synchronized 作用于一个class实例时，锁住的是所有以该class为锁的代码块

2 独占锁(可重入的互斥锁)

1. 互斥，即在同一时间点，只能被一个线程持有
2. 可重入，即可以被单个线程多次获取
3. 根据锁的获取机制，它分为“公平锁”和“非公平锁”
    - Java中通过ReentrantLock实现独占锁，默认为非公平锁

3 公平锁

1. 是按照通过CLH等待线程按照先来先得的规则, 线程依次排队, 公平的获取锁, 是独占锁的一种
    - 加锁前检查是否有排队等待的线程，优先排队等待的线程，先来先得
2. java中, ReetrantLock中有一个Sync类型的成员变量sync
    - sync它的实例为FairSync类型的时候，ReetrantLock为公平锁
    - 设置sync为FairSync类型，只需——Lock lock = new ReetrantLock(true)

4 非公平锁

1. 是当线程要获取锁时，它会无视CLH等待队列而直接获取锁
    - 加锁时不考虑排队等待问题，直接尝试获取锁，获取不到自动到队尾等待
2. ReetrantLock默认为非公平锁，或Lock lock = new ReetrantLock(false)。

5 共享锁

1. 能被多个线程同时获取、共享的锁。即多个线程都可以获取该锁，对该锁对象进行处理
2. 典型的就是读锁——ReentrantReadWriteLock.ReadLock
    - 即多个线程都可以读它，而且不影响其他线程对它的读，但是大家都不能修改它

3. CyclicBarrier, CountDownLatch和Semaphore也都是共享锁

6 读写锁

1. 维护了一对相关的锁
    - “读取锁”用于只读操作，它是“共享锁”，能同时被多个线程获取
    - “写入锁”用于写入操作，它是“独占锁”，只能被一个线程锁获取
2. 读写锁为ReadWriteLock 接口定义，其实现类是ReentrantReadWriteLock，包括内部类ReadLock和WriteLock
    - 方法readLock()、writeLock()分别返回读操作的锁和写操作的锁。

7 全局锁

1. 实现全局锁有两种方式
    - 将synchronized关键字用在static方法上
        - synchronized加到static静态方法上是对Class类上锁
        - 而synchronized加到非static方法上是给对对象上锁, Class锁可以对类的所有对象实例起作用
    - 用synchronized对类的Class对象进行上锁
        - synchronized(class)代码块的作用与synchronized static方法的作用一样

13 collection和collections的区别

1 Collection 是一个 集合框架的父接口

1. 它提供了对集合对象进行基本操作的通用接口方法
2. Collection接口在 Java 类库中有很多具体的实现
3. Collection接口的意义是为各种具体的集合提供了最大化的统一操作方式

2 Collections是一个操作集合的工具类, 服务于Java的Collection框架

1. 它包含有各种有关集合操作的, 静态多态方法

2. 提供一系列静态方法实现对各种集合的搜索、排序、线程安全化等操作

14 java如何跳出循环?

1 continue, break, return

1. continue：中止本次循环，继续下次循环。continue以后的循环体中的语句不会继续执行，下次循环继续执行，循环体外面的会执行
2. break：直接结束一个循环，跳出循环体。break以后的循环体中的语句不会继续执行，循环体外面的会执行
    - 注意: 只跳出一层循环
3. return：return的功能是结束一个方法。 一旦在循环体内执行return，将会结束该方法，循环自然也随之结
    - 可以跳出多层循环

2 循环标记使用(标签变量)

```
1. 场景: 多层循环，如果在里层循环，达到某个条件后，结束指定循环
2. 测试代码
      public static void main(String[] args) {
        ee:for (int i = 0; i < 3; i++) {
          for (int j = 0; j <3 ; j++) {
            System.out.println("j==========" + j);
            break ee;//跳出指定标记的循环
          }
          System.out.println("i=" + i);
        }
        System.out.println("跳出循环");
      }
3. 测试代码执行结果: j==========0
```

3 利用try...catch...

1. 给整个for循环加上try...catch...

2. 如果在循环中需要退出, 就抛出一个异常即可: throw new Exception("跳出循环");

4 利用标识变量

1. 即定义一个全局变量, 在for循环中进行判断, 当满足标识变量的值满足某个值时, 就使用break语句退出循环

15 排序有哪些? 原理是什么?

1 冒泡排序

1. 原理: 相邻元素比较, 后面的比前面的小, 就交换位置, 把最大的元素冒泡出来(即排到最后)
2. 时间复杂度：n的平方。原地修改数组
3. 但是性能会比较差

2 快速排序

3 选择排序

1. 原理:
    - 先选择出最小的元素, 记录其索引位置, 然后和第一个索引位置的元素交换位置
    - 然后选择出第2小的元素, 和第二个索引位置的元素交换
    - ...
2. 时间复杂度：n的平方。原地修改数组

4 插入排序

1. 原理
    - 把所有元素分为两组，已排序的和未排序的
    - 找到未排序组中的第一个元素，向已经排序的组中进行插入, 与已排序的数据进行比较, 将元素插入到合适的位置
    - 倒序遍历已经排序的元素,
2. 时间复杂度：n的平方。原地修改数组
3. 简单直观且稳定的排序算法

5 希尔排序(插入排序的改良版)

6 归并排序

16 什么是堆栈?什么是内存溢出?有垃圾回收机制为什么还会出现内存溢出的情况?

1. 堆与栈是两个相对的概念：堆指先进先出（first in first out），栈指先进后出（first in last out）
2. 这里的堆栈就是栈
    1. 堆是一个树状数据结构
    2. 而栈是一个链表, 栈是先进后出的, 有入栈和出栈的操作, 只在栈的一端进行操作
    3. 在java中堆一般用来存储对象和数组，栈一般用来存储方法和基本类型 栈的存取速度比堆快。栈中的数据是可以共享的。
3. 内存溢出 1. 应用系统中存在无法回收的内存 2. 或使用的内存过多 3. 最终使得程序运行要用到的内存大于虚拟机能提供的最大内存
4. 有垃圾回收机制为什么还会出现内存溢出的情况
    1. 垃圾回收机制只能回收没有引用的对象，也就是说只能回收没有“指针”的对象，对于非引用类对象，垃圾回收机制就不能起作用
    2. 比如说，如果打开过多的数据库连接，那么这些不能被垃圾回收机制所处理。由于一般情况下很少有人打开过多的数据库连接，所以很少有人注意到这一点。
    3. 还有一种可能就是死循环也会出现内存泄漏，但是不是所有的死循环都会出现内存泄漏现象。对ArrayList操作的时候才有可能出现，

17 内存模型的理解?

1 java内存模型（Java Memory Model，JMM）

1. 就是一种符合内存模型规范的，
2. 屏蔽了各种硬件和操作系统访问差异的

3. 保证了java程序在各种平台下对内存的访问都能保证效果一致的机制和规范
4. 作用: 为了保证并发编程中可以满足原子性、可见性以及有序性

2 Java内存模型所有的变量都存储在主内存，

1. 每条线程有自己的工作内存，线程的工作内存保存了该线程中用到的变量的主内存副本拷贝
2. 线程对变量所有的操作都必须在工作内存中进行，不能直接读写主内存
3. 不同的线程之间也无法直接访问对方工作内存中的变量，线程间变量的传递均需要自己的工作内存和主存之间进行数据同步

3 JMM是一种规范

1. 目的是解决由于多线程通过共享内存进行通信时，存在
    - 本地内存数据不一致
    - 编译器会对代码指令重排序
    - 处理器会对代码乱序执行等带来的问题
2. 目的就是保证并发编程场景中的原子性、可见性和有序性。

4 Java内存模型的实现

java提供了一系列和并发处理相关的关键字，比如volatile、synchronized、final、concurrent包等 synchronized保证了原子性可见性和有序性 volatile保证了可见性和有序性，但是不能保证原子性。volatile禁止指令重排序 final保证了可见性

5 JVM内存空间分为五部分，分别是：方法区、堆、Java虚拟机栈、本地方法栈、程序计数器

1. 方法区主要用来存放类信息、类的静态变量、常量、运行时常量池等，方法区的大小是可以动态扩展
2. 堆主要存放的是数组、类的实例对象、字符串常量池等
3. Java虚拟机栈是描述JAVA方法运行过程的内存模型，Java虚拟机栈会为每一个即将执行的方法创建一个叫做“栈帧”的区域，该区域用来存储该方法运行时需要的一些信息

    - Java虚拟机栈是运行Java方法的区域
4. 本地方法栈是运行本地方法的内存模型
5. 程序计数器是一个比较小的内存空间，用来记录当前线程正在执行的那一条字节码指令的地址
    - 如果当前线程正在执行的是本地方法，那么此时程序计数器为空
    - 程序计数器有两个作用，
        - 字节码解释器通过改变程序计数器来一次读取指令，从而实现代码的流程控制，比如我们常见的顺序、循环、选择、异常处理等
        - 在多线程的情况下，程序计数器用来记录当前线程执行的位置，当线程切换回来的时候仍然可以知道该线程上次执行到了哪里

18 泛型的理解?

1. 泛型是JDK5中引入的特性,它提供了编译时类型安全监测机制,该机制允许在编译时监测到非法的类型
2. 它的本质是参数化类型,即所操作的数据类型被指定为一个参数
    1. 参数化类型:就是将原来的具体的类型参数化,然后在使用/调用时传入具体的类型
    2. 例:List,ArrayList: 调用时:List list = new ArrayList();
    3. 这种参数类型可以用在类、方法、接口中,分别被称为泛型类、泛型方法、泛型接口
3. 当没有指定泛型时,默认为Object类型
4. 泛型的好处： 1. 类型参数化，通用、可以像方法一样的参数一样传递，非常实用。 2. 安全、编译时检查类型是否正确，降低类型强转报错率。 3. 提高代码重用率。
5. 类型擦除
    1. 泛型信息只存在于代码编译阶段
    2. 在进入 JVM 之前，与泛型相关的信息会被擦除掉，专业术语叫做类型擦除
6. 泛型转译

19 java的基本类型是什么, int占几个字节?byte占几个字节?

1. byte(1), short(2), int(4), long(8), float(4), double(8), boolean(1), char(2)
2. char
    1. char在Java中占用2字节, Java编译器默认使用Unicode编码，因此2字节可以表示所有字符
    2. 如果使用UTF-8编码, 则char存储字符占用的字节数是不定的, 1到3个字节

20 常见的异常有哪些? 异常处理的方式有哪些?

1 常见的异常有哪些

(1)NullPointerException 当应用程序试图访问空对象时，则抛出该异常。

(2)SQLException 提供关于数据库访问错误或其他错误信息的异常。

(3)IndexOutOfBoundsException指示某排序索引（例如对数组、字符串或向量的排序）超出范围时抛出。

(4)NumberFormatException当应用程序试图将字符串转换成一种数值类型，但该字符串不能转换为适当格式时，抛出该异常。

(5)FileNotFoundException当试图打开指定路径名表示的文件失败时，抛出此异常。

(6)IOException当发生某种I/O异常时，抛出此异常。此类是失败或中断的I/O操作生成的异常的通用类。

(7)ClassCastException当试图将对象强制转换为不是实例的子类时，抛出该异常。

(8)ArrayStoreException试图将错误类型的对象存储到一个对象数组时抛出的异常。

(9)IllegalArgumentException 抛出的异常表明向方法传递了一个不合法或不正确的参数。

(10)ArithmeticException当出现异常的运算条件时，抛出此异常。例如，一个整数“除以零”时，抛出此类的一个实例。

(11)NegativeArraySizeException如果应用程序试图创建大小为负的数组，则抛出该异常。

(12)NoSuchMethodException无法找到某一特定方法时，抛出该异常。

(13)SecurityException由安全管理器抛出的异常，指示存在安全侵犯。

(14)UnsupportedOperationException当不支持请求的操作时，抛出该异常。

(15)RuntimeException是那些可能在Java虚拟机正常运行期间抛出的异常的超类

2 异常处理的方式有哪些

1. 方案一(处理异常):使用try...catch...finally语句
2. 方案二(抛出异常):throws 异常类名;
3. 方案三(产生异常):throw 异常对象;

21 枚举的了解?

1.所有地方都能查到的解释：(证明很基本，也很重要)

a.枚举类是一种特殊的类，它和普通的类一样，有自己的变量、方法和构造器。 它的构造器只能使用private访问修饰符，所以无法从外部调用构造器，构造器只能在构造枚举值时被调用 b.一个java源文件中只能有一个public类型的枚举类，而且该原文件的名字也必须和该枚举类的名字一致。 也就是在一个文件中只能有一个public修饰的枚举类。这里不包括内部类哈，指的是公共访问的.java文件的入口。 c.枚举类和class,interface地位是等同的，枚举也能实现接口。 d.枚举类的对象

是有限且固定的，常用于状态、类型 e.枚举类默认集成了java.lang.Enum 类，并实现了java.lang.Seriablizable 和java.lang.Comparable 两个接口 f.所有的枚举值都默认是public static final的，不用重复声明，而且枚举值应该显式的在枚举类第一行列出，否则无法产生实例 非抽象的枚举类不能再派生子类。 g.枚举值列举的时候也可以实现该枚举类实现的接口方法。

2.枚举值常用的方法：

a.String toString() 默认返回的枚举的名称，可自定义进行重写 b.int ordinal() 返回枚举对象的索引值，根据枚举值声明的顺序而定，从0开始； c.String name() 返回枚举值 d.static values() 包含所有的枚举实例的数组，可用来遍历 e.int compareTo(E o) 用于与制定枚举对象比较顺序，只能同类型比较如果该对象位于指定对象之后则返回正整数反之返回负整数，否则返回零，整数值为二者顺序之差。 f.boolean equals() 比较两个枚举实例的引用。

22 final, finally, finalize关键字的区别?volaile关键字的了解?

1. Final是一个修饰符
2. Finally, 是try...catch...finally
3. finalize(), object类中的一个方法, 是GC （ garbage collector垃圾回收 ）运行机制的一部分

   当gc判断一个对象没有其他的引用(不存在该对象的引用), 调用该方法回收对象的资源
4. 任何被volatile修饰过的变量，都不拷贝副本到工作内存，任何修改都及时写在主存。因此对于volaile修饰的变量的修改，所有线程马上就能看到，但是volatile不能保证对变量的修改是有序的
   1. volatile解决共享变量的可见性,有序性(volatile修改的变量禁止代码重排), 不能解决原子性
   2. 解决可见性, 每个线程都有自己的工作空间, java内存模型把所有的变量存储在主内存

1. 单个线程使用时, 会将变量复制一份到自己的工作空间, 执行完成后再同步到主内存
2. 而volatile关键字修饰变量时, 该变量只存在于主内存, 单个线程使用时不会复制变量副本, 而是直接操作主内存的变量

23 在一个list中存放的String类型的字符串, 如何实现把其中所有带"王"的字符串从list中去除?

1 遍历list集合判断后赋给另一个list集合

2 用赋给set集合再返回给list集合

3 解决方法

```
1. 方法一: 普通for倒着遍历list集合, 并删除重复数据
2. 方法二: 普通for正序遍历list集合, 每删除一个元素后, 加入循环变量i的值-1操作
3. 方法三: 迭代器iterator遍历
      Iterator it = list.iterator();
        while(it.hasNext()){
          String x = it.next();
          if(x.equals(“del
            it.remove();// 注意这里使用的是迭代器的remove()方法, 不能使用list的remove()
        }
      }
```

4 注意: 使用增强for遍历集合并删除数据, 会出现ConcurrentModificationException //这是一个并发修改异常报错

调用list.remove()方法导致modCount和expectedModCount的值不一致而报异常

24 String a="123", String b="123", a+=b; 共产生了几个对象

1. 共产生了2个对象: "123", "123123"
2. 这两个对象都是常量池中的对象, 而不是new出来的对象

25 如何序列化和反序列化?序列化的目的?

0. 序列化需要实现Serializable接口
1. 如何序列化和反序列化
    1. ObjectOutputStream代表对象输出流：
        - 它的writeObject(Object obj)方法可对参数指定的obj对象进行序列化，把得到的字节序列写到一个目标输出流中。
    2. ObjectInputStream代表对象输入流：
        - 它的readObject()方法从一个源输入流中读取字节序列，再把它们反序列化为一个对象，并将其返回。
2. 序列化的目的
    1. 当你想把的内存中的对象状态保存到一个文件中或者数据库中时候；
    2. 当你想用套接字在网络上传送对象的时候；
    3. 当你想通过RMI传输对象的时候
3. 序列化的意义
    1. 将对象或者异常等写入文件，通过文件传输信息
    2. 将对象或者异常通过网络进行传输
    3. 把内存中的对象状态保存到一个文件中或者数据库中时

二 数据库

1 sql优化: 即减少sql语句的执行时间

1 说说对SQL语句优化有哪些方法

0. 用EXISTS替代IN、用NOT EXISTS替代NOT IN

select * from table where exists (子查询) 子查询有返回结果, 则exists()返回true, 主查询返回结果 子查询每有返回结果, 则exists()返回false, 主查询不返回结果

1. 避免在索引列上使用计算，因为这样会导致索引失效 避免在索引列上使用 IS NULL 和 IS NOT NULL
2. 对查询进行优化，应尽量避免全表扫描，首先可以考虑在where和order by涉及的列上建立索引
3. 尽量避免在where的子句中对字段进行null值判断，否则引擎就会放弃使用索引，从而进行全表扫描 尽量避免在where子句中对字段进行表达式操作，否则引擎就会放弃使用索引，从而进行全表扫描 尽量避免在where子句中使用!=或<>操作符，否则引擎就会放弃使用索引，从而进行全表扫描
4. 尽量避免在where子句中使用or来连接条件，否则引擎就会放弃使用索引，从而进行全表扫描
5. 如果数值是连续的，能用between就不要用in
6. 任何地方都不要使用 select * from table ，要用具体的字段代替“*”，只取出需要的字段 模糊搜索尽量避免使用前置百分号，否则引擎就会放弃使用索引，从而进行全表扫描 一个表的索引数最好不要超过6个，因为太多的索引会影响到表的更新速度 如果字段只包含数值信息，尽量不要设计为字符类型，否则会降低查询和连接的性能，同时还会增加存储开销 避免频繁创建和删除临时表，减少系统表资源的消耗 尽量使用 varchar/nvarchar 代替 char/nchar ，因为变长字段的存储空间小，这样可以节省存储空间，同时对于查询来说，在一个相对较小的字段内搜索，效率自然会更高 尽量使用表变量来代替临时表
7. 减少表连接，可适当增加冗余字段。
8. 表连接情况下，把表数据量大的放于最前面,减少查询行数

2 如何创建索引?

查看表中已有索引: show index from 表名; 主键列会自动创建索引 创建索引的语法格式: alter table 表名 add index[索引名](列名, ..) 索引名不指定，默认使用字段名 删除索引的语法格式: alter table 表名 drop index 索引名 如果不知道索引名，可以查看创表sql语句: show create

table classes; 联合索引的好处: 减少磁盘空间开销，因为每创建一个索引，其实就是创建了一个索引文件，那么会增加磁盘空间

3 创建索引的原则?

1. 为常作为查询条件的字段建立索引
2. 对于经常存取的列避免建立索引
3. 尽量使用数据量少的索引
4. 尝试建立复合索引来进一步提高系统性能
5. 在SQL语句中经常进行GROUP BY、ORDER BY的字段上建立索引
   数据库建立索引的原则
   - 确定针对该表的操作是大量的查询操作还是大量的增删改操作。
   - 尝试建立索引来帮助特定的查询。检查自己的sql语句，为那些频繁在where子句中出现的字段建立索引。
   - 尝试建立复合索引来进一步提高系统性能。修改复合索引将消耗更长时间，同时，复合索引也占磁盘空间。
   - 对于小型的表，建立索引可能会影响性能
   - 应该避免对具有较少值的字段进行索引。
   - 避免选择大型数据类型的列作为索引。
6. 索引查询是数据库中重要的记录查询方法，要不要进入索引以及在那些字段上建立索引都要和实际数据库系统的查询要求结合来考虑，下面给出实际中的一些通用的原则：
   - 在经常用作过滤器的字段上建立索引；
   - 在SQL语句中经常进行GROUP BY、ORDER BY的字段上建立索引；
   - 在不同值较少的字段上不必要建立索引，如性别字段；
   - 对于经常存取的列避免建立索引；
   - 用于联接的列（主健/外健）上建立索引；
   - 在经常存取的多个列上建立复合索引，但要注意复合索引的建立顺序要按照使用的频度来确定；
   - 缺省情况下建立的是非簇集索引，但在以下情况下最好考虑簇集索引，如：含有有限数目（不是很少）唯一的列；进行大范围的查询；充分的利用索引可以减少表扫描I/0的次数，有效的避免

对整表的搜索。当然合理的索引要建立在对各种查询的分析和预测中，也取决于DBA的所设计的数据库结构。

7. oracle 索引建立的若干原则

   我们首先要考虑的是数据量，数据量级别的不同，要考虑的问题有很大区别。几千条记录建不建索引其实都无所谓了，差个几豪秒也感觉不出来，但数据量一旦要增长到百万， 千万级别，索引的重要性就体现出来了。没有索引一个查询可能要几个小时甚至几天才能出来，对数据库的影响不仅仅是查询速度的降低， 在io 上的花费和对数据库连接资源的占用甚至会拖跨数据库。那么怎样建立索引，建立什么类型的索引呢，应该按照几个方面来考虑。

8. 先要了解业务需求，总结出应用会按照哪几个字段来进行查询。例如一个人员查询系统可能会按照（姓名，身份证件号码，性别，住址，民族等），当然可能是几个字段的组合查询。

9. 确定了第一步后要估计数据的分布。相同值有多少，是不是均匀。如姓名字段就会有大量的重复情况；性别可能只有两个值（1，2 分别代表男，女），民族会有56 个。

10. 确定索引的类型。选择性高的字段建立B- 树索引最好，如果数据量太大，可考虑把索引分区，在并发情况下通常会表现很好。相当于把一棵很大的B树拆开成了多棵小树。只有几个值的字段如性别并且数据分布比较均匀，查询的平均命中率要是非常高就不需要建立索引，否则可以建立位图索引（但会影响并发）。

11. oracle建立索引原则

    - 索引需要平衡query和DML的需要，常用于(子)查询的表应建立索引；
    - 把索引建到不同的表空间中；
    - 用统一的extent大小:
    - 五个block的倍数或者tablespace指定的MINIMUM EXTENT的倍数；
    - 创建索引考虑用NOLOGGING参数，重建索引的时候也一样；
    - 创建索引时INITRANS值应该比相应的table的值高一些；
    - 对常用SQL语句的where条件中的列建立唯一索引或组合索引，组合条件查询中相应的组合索引更有效；

- 对于组合索引，根据列的唯一值概率，安排索引顺序；
- 如果一个列具有很低的数据基数，并且或者可具有空值，不应作为索引列；
- 如果where语句中不得不对查询列采用函数查询，如upper函数，最好建立相应函数索引；
- 对于低基数集的列，并包含OR等逻辑运算，考虑用Bitmap索引，对于从大量行的表中返回大量的行时也可以考虑Bitmap索引；
- 避免在有大量并发DML运算的表中使用Bitmap索引

4 索引的优缺点

1. 优点 第一，通过创建唯一性索引，可以保证数据库表中每一行数据的唯一性。 第二，可以大大加快 数据的检索速度，这也是创建索引的最主要的原因。
   第三，可以加速表和表之间的连接，特别是在实现数据的参考完整性方面特别有意义。
   第四，在使用分组和排序子句进行数据检索时，同样可以显著减少查询中分组和排序的时间 第五，通过使用索引，可以在查询的过程中，使用优化隐藏器，提高系统的性能。
2. 缺点 第一，创建索引和维护索引要耗费时间，这种时间随着数据 量的增加而增加。 第二，索引需要占物理空间，除了数据表占数据空间之外，每一个索引还要占一定的物理空间，如果要建立聚簇索引，那么需要的空间就会更大。 第三，当对表中的数据进行增加、删除和修改的时候，索引也要动态的维护，这样就降低了数据的维护速度。
3. 在哪些列上能创建索引。一般来说，应该在这些列 上创建索引 在经常需要搜索的列上，可以加快搜索的速度； 在作为主键的列上，强制该列的唯一性和组织表中数据的排列结构； 在经常用在连接的列上，这 些列主要是一些外键，可以加快连接的速度； 在经常需要根据范围进行搜索的列上创建索引，因为索引已经排序，其指定的范围是连续的； 在经常需要排序的列上创 建索引，因为索引已经排序，这样查询可以利用索引的排序，加快排序查询时间； 在经常使用在WHERE子句中的列上面创建索引，加快条件的判断速度。

2 sql去重

1 SQL中的三种去重方法

```
1. distinct
2. group by
3. row_number 窗口函数进行去重，在支持窗口函数的 sql（如 Hive SQL、Oracle等等） 中可以使用
```

3 内连接和外连接的区别

内连接（自然连接）：只有两张表相匹配的行才能出现在结果集 inner join, 消除笛卡尔积查询 外连接: outer join

- 左外连接：左边为主表，左边的表显示全部；右边为副表，右边无符号数据时显示null，不符合的不显示；
- 右外连接：右边为主表，右边的表显示全部；左边为副表，左边无符号数据时显示null，不符合的不显示；
- 全外连接：左边为主表，右边为副表，主表和副表全部显示，右边无符号数据时显示null，左边无符号数据时显示null，符合条件的数据会显示在一行；
- 交叉连接：左边为主表，右边为副表，显示的数据为笛卡尔乘积的形式 cross join

4 java中如何使用redis

java中利用jedis连接redis 获取连接: Jedis jedis = new Jedis("localhost", "6376");

redis支持的数据类型及各数据类型的使用场景?

string, 简单的key-value类型，普通的key/ value 存储都可以归为此类

- 存放session key
- 短信的验证码

hash: 即map键值对格式的

- 存放结构化数据，比如用户信息
    - 用户信息比如用户的昵称、年龄、性别、积分等，我们需要先序列化后存储为一个字符串的值
    - 在修改时, 就需要反序列化修改某一项的值，再序列化存储回去。这样不仅增大了开销, 也不适用于一些可能并发操作的场合
    - 而Redis的Hash结构可以使你像在数据库中Update一个属性一样只修改某一项属性值
- 各种列表，比如twitter的关注列表、粉丝列表等，最新消息排行、每篇文章的评论等也可以用Redis的list结构来实现
- 消息队列，可以利用Lists的PUSH操作，将任务存在Lists中，然后工作线程再用POP操作将任务取出执行

set: 是一种无序的集合, 不重复。将重复的元素放入Set会自动去重。

- 某些需要去重的列表
- 可以存储一些集合性的数据

sortedset: 有序集合

- 存放一个有序的并且不重复的集合列表
- 可以做带权重的队列
- 排行榜相关
- 新闻按照用户投票和时间排序

redis如何解决数据过期?

最近我们在Redis集群中发现了一个有趣的问题。在花费大量时间进行调试和测试后，通过更改key过期，我们可以将某些集群中的Redis内存使用量减少25% Redis 提供了两种的方式，用于删除过期的数据 定期删除 Redis 默认 100ms 随即抽取部分设置过期时间的 key，过期了就删除。优点是避免长时间的在扫描过期 key，缺点是有些过期 key 无法被删除 不扫描全部 key 的原因是，当设置了过期时间的 key 太多的情况下，会很耗时间，O(n) 的时间复杂度。 惰性删除 如果查询了某个过期 key，但定期删除没有删除掉，那就将其删除了。key 没过期就正常返回。

5 数据库表的设计

1 数据库表的设计注意事项有哪些?

1. 数据库表命名，将业务和基础表区分，采用驼峰表示法等
2. 数据不要物理删除，应该加一个标志位，以防用户后悔时，能够恢复
3. 排序字段，按照某种类型来排序（sortcode）最好不依赖id排序，这样方便我们查询记录时按照某种方式排序，而不依赖id。
4. 数据是否允许删除和允许编辑，例如管理员不能删除，这样我们在查询数据时就可以根据该字段标示来决定某条记录是否可以编辑。而不用固化到代码中。
5. 增加备注字段，虽然我们考虑了很多用户需要输入信息的需求，但是无论何时我们都不可能考虑全，因此可以定义一个备注字段，允许用户将其它的信息填写在这里。无论表设计的再神奇，那么还是加一个备注字段。
6. 添加时间，有添加时间可以明确知道记录什么时候添加的。
7. 修改时间，可以知道记录什么时候被修改了，一旦数据出现问题，可以根据修改时间来定位问题。比如某人在这个时间做了哪些事。

2 三大范式的了解?

第一范式（原子性） 是最基本的范式。如果数据库表中的所有字段值都是不可分解的原子值 第二范式（完全依赖主键） 需要确保数据库表中每一列都和主键相关，而不能只与主键的某一部分相关（主要针对联合主键而言）。也就是说在一个数据库表中，一个表中只能保存一种数据，不可以把多种数据保存在同一张数据库表中。一张表中出现数据重复就可以将其拆分成两个表 第三范式（直接依赖主键） 需要确保数据表中的每一列数据都和主键直接相关，而不能间接相关。 有时为了满足查询速度，可以有意识的让某些表有些冗余，这是为了提高整个数据库的性能，所以有些时候，不一定要拘泥于达到第三范式或bcn 范式，只要数据库的设计可以提高整个数据库的性能，这就是一个合理的数据库

6 存储过程的了解和使用?

什么是存储过程 SQL语句需要先编译然后执行，而存储过程（Stored Procedure）是一组为了完成特定功能的SQL语句集，经编译后存储在数据库中，用户通过指定存储过程的名字并给定参数（如果该存储过程带有参数）来调用执行它。 存储过程是可编程的函数，在数据库中创建并保存，可以由SQL语句和控制结构组成。 当想要在不同的应用程序或平台上执行相同的函数，或者封装特定功能时，存储过程是非常有用的。 数据库中的存储过程可以看做是对编程中面向对象方法的模拟，它允许控制数据的访问方式。 存储过程的优点： (1).增强SQL语言的功能和灵活性：存储过程可以用控制语句编写，有很强的灵活性，可以完成复杂的判断和较复杂的运算。 (2).标准组件式编程：存储过程被创建后，可以在程序中被多次调用，而不必重新编写该存储过程的SQL语句。而且数据库专业人员可以随时对存储过程进行修改，对应用程序源代码毫无影响。 (3).较快的执行速度：如果某一操作包含大量的Transaction-SQL代码或分别被多次执行，那么存储过程要比批处理的执行速度快很多。因为存储过程是预编译的。在首次运行一个存储过程时查询，优化器对其进行分析优化，并且给出最终被存储在系统表中的执行计划。而批处理的Transaction-SQL语句在每次运行时都要进行编译和优化，速度相对要慢一些。 (4).减少网络流量：针对同一个数据库对象的操作（如查询、修改），如果这一操作所涉及的Transaction-SQL语句被组织进存储过程，那么当在客户计算机上调用该存储过程时，网络中传送的只是该调用语句，从而大大减少网络流量并降低了网络负载。 (5).作为一种安全机制来充分利用：通过对执行某一存储过程的权限进行限制，能够实现对相应的数据的访问权限的限制，避免了非授权用户对数据的访问，保证了数据的安全。

7 数据库的分页

1 数据库如何实现分页?

```
1. MySQL实现分页：使用limit
2. Oralce实现分页：
  Oracle中有个rownum，其含义更加明显，就是第几行的意思，这样我们就可以通过where条件来进行分段查询了。
    select * from t_user where rownum>=2 and rownum<=4
    注意：oracle上面的语句查不到数据，应该套一层，如下
    select * from (select a.*, rownum rn from t_user   a where rownum <= 4) where rn >= 2
  minus 运算符来实现分页，查询语句为：
    select rownum,t.* from T_ACCOUNT t where rownum<=20
    minus
    select rownum,t.* from T_ACCOUNT t where rownum<=10
3. sqlserver:  在分页查询上，SQL Server比较费劲，没有一个专门的分页的语句，靠的是一种巧妙的方法实现分页查
  select * from t_user select * from (select top 2 * from (select top 6 * from t_user order by id asc ) as aaa order by id desc) as bbb order by id asc
```

2 百万级量的数据分页查询如何优化?

```
mysql: select * from news where id>=(select id from news limit 490000,1) limit 10;
limit只有一个参数，默认查询第1页，limit后跟的是每页记录数
```

8 数据库的乐观锁和悲观锁的理解和使用?

1 理解

1. 悲观锁，就是对数据的冲突采取一种悲观的态度，也就是说假设数据肯定会冲突，所以在数据开始读取的时候就把数据锁定住。【数据锁定：数据将暂时不会得到修改】 悲观锁是读取的时候为后面的

更新加锁，之后再来的读操作都会等待。这种是数据库锁 悲观锁是数据库实现，他阻止一切数据库操作

2. 乐观锁，认为数据一般情况下不会造成冲突，所以在数据进行提交更新的时候，才会正式对数据的冲突与否进行检测，如果发现冲突了，则让用户返回错误的信息。让用户决定如何去做 乐观锁是一种思想，具体实现是，表中有一个版本字段，第一次读的时候，获取到这个字段。处理完业务逻辑开始更新的时候， 需要再次查看该字段的值是否和第一次的一样。如果一样更新，反之拒绝 之所以叫乐观，因为这个模式没有从数据库加锁 乐观锁优点程序实现，不会存在死锁等问题。他的适用场景也相对乐观。阻止不了除了程序之外的数据库操作。

2 使用

1. ORM框架中悲观锁乐观锁的应用
2. 悲观锁
    - select... for update
3. 乐观锁
    - 乐观锁使用的是版本号
        - 修改数据之前先查询数据对应的版本号
        - 一个线程修改时先检查数据库中的版本号和自己拿到的版本号是否一致
        - 修改完毕后把版本号加1
    - 乐观锁不能防止其他线程修改数据

9 数据库中日期和字符串的相互转换?

1 Oracle

时间转字符串 to_char(date,format): select to_char(sysdata,'YYYY"年"MM"月"DD"日"') 时间转字符串 from dual 字符串转时间 to_date(str,format): select to_date('2019-10-25 17:15:20','yyyy-MM-dd HH24:mi:ss') 字符串转时间 from dual

2 MySQL

时间转字符串DATE_FORMAT(): select DATE_FORMAT(SYSDATE(),'%Y年%m月%d日') MySQL日期转字符串 from DUAL； 字符串转时间 str_to_date(): select str_to_date('2019-10-25 15:43:28','%Y-%m-%d %H:%i:%s'); H大写是指的是：24小时制；h小写是指的是12小时制；

10 union和unionAll区别?

UNION 并集，表中的所有数据，并且去除重复数据（工作中主要用到的是这个）；

1. UNION 操作符用于合并两个或多个 SELECT 语句的结果
2. 需要满足以下条件：
    - 相同数量的列；
    - 列也必须拥有相似的数据类型；
    - 同时，每条 SELECT 语句中的列的顺序必须相同。
3. union去重, 对结果排序 UNION ALL，表中的数据都罗列出来
4. 不去重, 结果不排序

11 mysql的存储引擎有哪些?

MyISAM存储引擎: 非事务处理存储引擎 innoDB存储引擎: 具备外键支持功能的事务处理引擎 MEMORY存储引擎: 置于内存的表 ARCHIVE存储引擎: 用于数据存档的引擎, 数据被插入后就不能再修改了

12 事务的隔离级别有哪些?

- Read uncommitted (读未提交)：最低级别，任何情况都无法保证。
- Read committed (读已提交)：可避免脏读的发生。
- Repeatable read (可重复读)：可避免脏读、不可重复读的发生。
- Serializable (串行化)：可避免脏读、不可重复读、幻读的发生。

mysql和oracle默认的隔离级别是什么?

- mysql默认的事务处理级别是'REPEATABLE-READ(repeatable read)',也就是可重复读

- oracle数据库支持READ COMMITTED 和 SERIALIZABLE这两种事务隔离级别。
- 默认系统事务隔离级别是READ COMMITTED,也就是读已提交

13 sql如何行转列和列转行?

1 行转列

1. 行转列一般通过CASE WHEN 语句来实现
2. 也可以通过 SQL SERVER 2005 新增的运算符PIVOT来实现。 用传统的方法，比较好理解。层次清晰，而且比较习惯。 但是PIVOT 、UNPIVOT提供的语法比一系列复杂的SELECT...CASE 语句中所指定的语法更简单、更具可读性
3. 主要思路是分组后使用case进行条件判断处理

2 列转行

主要思路也是分组后使用case 也可以用union运算符 select 'a' as 'q'; 这个语句执行后可以把'a'作为列'q'中的值

14 如何查看sql的执行计划?

MySQL使用 explain 关键字来查看SQL的执行计划

15 oracle中的分析函数有哪些?

row_number()可以通过over 根据某字段排序完之后进行组内（如果有partition by）排序。 rank()是排名的函数，该函数组内排序后会进行跳号，分数相同的作为并列。 dense_rank()该函数不会跳号，分数相同为并列第一，下一个是第二

16 数据库中除了聚合函数之外还有哪些常用的函数?

常用函数 * 时间函数 获取当前系统时间select now(); 获取系统的时分秒select curtime(); 获取系统的年月日select curdate(); * 数学函数 向上取舍select ceil(数值); 向下取舍select floor(数值); 获取四位小数 select ceil(rand()*10000); 随机数select rand();//0~1之间的随机数； oracle数据库merge()函数的作用和使用? 通常我们对数据库数据进行插入的时候，会判断数据是否已经存在，如果存在就修改，不存在就插入， 一般我们会写两个sql，一个insert一个update,那么其实oracle中存在merge函数，可以一次性解决这个问题 作用: 能够在一个SQL语句中对一个表同时执行inserts和updates操作. MERGE命令从一个或多个数据源中选择行来updating或inserting到一个或多个表 使用: 比如向数据库插入一条数据 如果ID和name都相同 就更新AGE的数据，否则就插入

17 sql 中 drop, truncate, delete 的区别?

相同点：drop、delete、truncate 都是删除表的内容。 不同点： drop：删除表内容和结构，释放空间，没有备份表之前要慎用； truncate：删除表的内容，表的结构存在，可以释放空间,没有备份表之前要慎用 delete：删除表的内容，表的结构还存在，不释放空间，可以回滚恢复；

drop：drop test 删除表test，并释放空间，将test删除的一干二净 truncate：truncate test 删除表test里的内容，并释放空间，但不删除表的定义，表的结构还在 truncate删除数据不能只删除一行, 只能删除整张表的数据 delete：（1）删除制定数据：删除表test中年龄等于30且国家为US的数据：delete from test where age=30 and country ='US'； （2）删除整个表：仅删除表test内的所有内容，保留表的定义，不释放空间：delete from test 或delete *from test 都删除整张表的话 truncate table 速度更快，占用的日志更少，这是因为 TRUNCATE TABLE 直接释放数据页并且在事务日志中也只记录数据页的释放 而 DELETE 是一行一行地删除，在事务日志中要记录每一条记录的删除 drop > truncate > delete sql语句的不同 drop table table_name; truncate table_name; delete from table_name;

18 mysql如何忽略表名的大小写?

linux下mysql默认是要区分表名大小写的 mysql是否区分大小写设置是由参数lower_case_table_names决定的 其中：找到你mysql的配置文件my.ini（linux下是my.cnf）,打开后找到“[mysqld]”节点，在下面加上一句话： 1.）lower_case_table_names = 0 (默认) 区分大小写（即对大小写不敏感），默认是这种设置。这样设置后，在mysql里创建的表名带不带大写字母都没有影响，都可以正常读出和被引用。 2.）lower_case_table_names = 1 不区分大小写（即对大小写敏感）。这样设置后，表名在硬盘上以小写保存，MySQL将所有表名转换为小写存储和查找表上。该行为也适合数据库名和表的别名。 也就是说，mysql设置为不分区大小写后，创建库或表时，不管创建时使用大写字母，创建成功后，都是强制以小写保存！ MySQL在Linux下数据库名、表名、列名、别名大小写规则是这样的： 1.）数据库名与表名是严格区分大小写的； 2.）表的别名是严格区分大小写的； 3.）列名与列的别名在所有的情况下均是忽略大小写的； 4.）变量名也是严格区分大小写的； 5.）MySQL在Windows下都不区分大小写，但是在Linux下默认是区分大小写的。 6.）如果想在查询时区分字段值的大小写，则字段值需要设置BINARY属性，设置的方法有多种： a）创建时设置：CREATE TABLE T(A VARCHAR(10) BINARY); b）使用alter修改 所以在不同操作系统中为了能使程序和数据库都能正常运行，最好的办法是在设计表的时候都转为小写！！ 修改mysql为不区分大小写设置：mysqladmin -uroot -p shutdown //以安全模式关闭数据库 修改my.cnf //添加下面一行设置 ..... [mysqld]
lower_case_table_names=1 .....

启动mysql

19 having和where的区别?

“Where”是一个约束声明，在查询数据库的结果返回之前对数据库中的查询条件进行约束，即在结果返回之前起作用，且where后面不能使用“聚合函数”； where后面之所以不能使用聚合函数是因为where的执行顺序在聚合函数之前 “Having”是一个过滤声明，所谓过滤是在查询数据库的结果返回之后进行过滤，即在结果返回之后起作用，并且having

后面可以使用“聚合函数”。 having既然是对查出来的结果进行过滤，那么就不能对没有查出来的值使用having 聚合函数是比较where和having的关键

20 游标的作用和使用

1 作用: 用来存储查询结果集的

2 使用: 遍历游标中的数据, 相当于有一个指针指向游标中的第一行数据,每获取一行记录,指针向下移一行

3 语法格式

1. 声明游标
    - 格式 declare cursor_name cursor for select_statement
    - 格式介绍
        - cursor_name:游标名称,存储sql语句执行的结果
        - select_statement:sql语句
2. 打开游标
    - 格式 open cursor_name;
3. 遍历游标中的数据
    - 介绍
        - 相当于有一个指针指向游标中的第一行数据,每获取一行记录,指针向下移一行
    - 格式 fetch cursor_name into var_name [, var_name] ...
    - s格式介绍
        - 一行fetch只能获取游标中的一行记录,并把记录中每一列的值赋值给var_name
        - 一个var_name保存一行记录中一个列的值,若有多个列,需要多个变量
        - 要输出游标中的所有数据,需要使用循环语句
4. 关闭游标
    - 格式 close cursor_name;

21 如何使用数据库中的定时器, 触发器, 定时任务?

定时器 自 MySQL5.1.6起，增加了一个非常有特色的功能–事件调度器(Event Scheduler)，可以用做定时执行某些特定任务（例如：删除记录、对数据进行汇总等等），来取代原先只能由操作系统的计划任务来执行的工作 更值得 一提的是MySQL的事件调度器可以精确到每秒钟执行一个任务，而操作系统的计划任务（如：Linux下的CRON或Windows下的任务计划）只能精 确到每分钟执行一次 可将一些数据库操作定时任务，从程序中挪到数据库中进行操作，可大大提升执行效率 触发器 触发器是mysql5新增的功能,触发器和存储过程一样，都是嵌入到mysql的一段程序。(备注如果after触发器执行失败事务会回滚) 定时任务 mysql怎么让一个存储过程定时执行 查看event是否开启: show variables like '%sche%'; 或者 SHOW VARIABLES LIKE 'event_scheduler';

OFF表示关闭。那么，可以使用 sql去开启事件执行

SET GLOBAL event_scheduler = ON;

到此就可以定时执行执行的过程。

将事件计划开启: set global event_scheduler=1; 关闭事件任务: alter event e_test ON COMPLETION PRESERVE DISABLE; 开户事件任务: alter event e_test ON COMPLETION PRESERVE ENABLE;

22 oracle中如何实现递归查询?

1 oracle中的递归查询采用的语法为 start with 。。。 connect by ..... = prior ....

2 sql语句

```
`SELECT * FROM tree START WITH id = 2 CONNECT BY PRIOR pid = id -- 递归查询父节点
union
ELECT * FROM tree START WITH id = 2 CONNECT BY pid = PRIOR id; -- 递归查询子节点`
```

23 高并发下如何保证修改数据安全?

Mysql中的有两种方法：select...for update或lock in share mode
Select...for update的实现方式：

```
set autocommit =0;//关闭自动提交
begin;//开始事务
select * from order where id=989879 for update;//查询信息
update order set name='names';//修改信息
commit;//提交事务
```

执行select...for update时，一般的SELECT查询则不受影响。 Lock in share mode的是实现方式：

```
set autocommit =0;//关闭自动提交
begin;//开始事务
select * from order where id=989879 lock in share mode;//锁定查询的字段
update order set name='names';//修改信息
commit;//提交事务
```

lock in share mode或select...for update是在事务内起作用的，涉及行锁的概念。 能保证当前session事务锁定的行不被其他session所修改。 前者属于共享锁，允许其他事务添加共享锁，不允许其他事务修改或者加排它锁。 后者属于排它锁，不允许其他事务添加共享锁和排它锁，也不允许修改。 一般情况下推荐使用select...for update。使用lock in share mode时需要注意死锁的问题，就是如果两个session同时对一行加锁，那么将无法执行修改，必须等一个session退出以后才能执行 如果有两个session加锁后同时修改一行，那么将有一个session被引擎强制关闭并重启，这样另一个session的修改就可以顺利

执行 如果只有一个session修改，另一个session不做修改动作，那么这个尝试修改的session将会一直等待锁被释放才能继续执行。 乐观锁: 版本号 悲观锁: for update

24 oracle中如何实现主键自增?

1 oracle中没有自增字段，可通过序列+触发器间接实现

2 实现

```
1. 建立数据表
2. 创建自动增长序列
      create sequence seq_tb_user
      minvalue 1
      nomaxvalue
      start with 1
      increment by 1
      nocycle   --一直累加，不循环
      --nocache;  --不缓存
      cache 10; --缓存10条
3. 创建触发器
      CREATE OR REPLACE TRIGGER tr_tb_user
      BEFORE INSERT ON tb_user FOR EACH ROW WHEN (new.id is null)
      begin
      select seq_tb_user.nextval into:new.id from dual;
      end;
4. 提交
```

25 delete误删数据没有备份怎么恢复?

1 mysql可以通过日志记录进行恢复

1. 将日志记录中的sql语句重新执行一次

2. 找到mysql的data目录下的mysql-bin.00000X文件，类似这种的，应该有很多个，因为配置时候文件名可以配，所以不保证一定是这样的，要是不一样应该也很好找，就是有一组有规律XXXX.00000X文件就是了。
3. 将日志文件导出成sql文件，方法是在命令行窗口下（其实就是cmd窗口）调用mysql的bin目录mysqlbinlog程序
    - mysqlbinlog --start-date="2013-10-01 00:00:00" --stop-date="2013-12-12 12:00:00" E:\mysql-5.5.21-win32\data\mysql-bin.000067 > e:\67.sql

2 可以使用数据库闪回

26 oracle死锁如何处理?

1 一般情况下，只要将产生Oracle死锁的语句提交就可以了，但是在实际的执行过程中。用户可能不知道产生死锁的语句是哪一句。可以将程序关闭并重新启动就可以了

2 二

查找Oracle死锁的进程： kill掉这个Oracle死锁的进程：

3 三

--1 查看数据库中那些用户产生了锁 --2 杀掉ORACLE进程 --3 查找并杀死死锁的进程

三 框架

1 spring特性是什么?

1. IOC控制反转
2. aop面向切面编程

2 ioc和aop的原理是什么?

1. ioc: 把创建对象的权利交给spring, 有spring来管理对象

   使用对象的时候通过依赖注入来注入对象

   autowired, qualifier, resource

2. aop 一是采用动态代理技术，利用截取消息的方式，对该消息进行装饰，以取代原有对象行为的执行 二是采用静态织入的方式，引入特定的语法创建“切面”，从而使得编译器可以在编译期间织入有关“切面”的代码，属于静态代理

3 aop的注解有哪些?

@Aspect： 作用：把当前类声明为切面类 @Before： 作用：把当前方法看成是前置通知 属性： value：用于指定切入点表达式，还可以指定切入点表达式的引用。 讲师：陈飞 @AfterReturning 作用：把当前方法看成是后置通知 属性： value：用于指定切入点表达式，还可以指定切入点表达式的引用。 @AfterThrowing 作用：把当前方法看成是异常通知 属性： value：用于指定切入点表达式，还可以指定切入点表达式的引用。 @After 作用：把当前方法看成是最终通知 属性： value：用于指定切入点表达式，还可以指定切入点表达式的引用。 @Around 作用：把当前方法看成是环绕通知 属性： value：用于指定切入点表达式，还可以指定切入点表达式的引用 @Pointcut 作用：指定切入点表达式 属性： value：指定表达式的内容

2 springboot自动装配的原理是什么?

1. Spring Boot关于自动配置的源码在spring-boot-autoconfigure-x.x.x.x.jar中：
2. @SpringBootApplication是一个复合注解或派生注解，在@SpringBootApplication中有一个注解@EnableAutoConfiguration，翻译成人话就是开启自动配置
3. @EnableAutoConfiguration也是一个派生注解，其中的关键功能由 @Import提供

4. @Import导入的AutoConfigurationImportSelector的selectImports()方法通过SpringFactoriesLoader.loadFactoryNames()扫描所有具有META-INF/spring.factories的jar
    - spring-boot-autoconfigure-x.x.x.x.jar里就有一个这样的spring.factories文件
    - 这个spring.factories文件也是一组一组的key=value的形式
    - 其中一个key是EnableAutoConfiguration类的全类名，而它的value是一个xxxxAutoConfiguration的类名的列表，这些类名以逗号分隔
    - 这个 @EnableAutoConfiguration 注解通过@SpringBootApplication被间接的标记在了Spring Boot的启动类上
    - 在SpringApplication.run(...)的内部就会执行selectImports()方法，找到所有JavaConfig自动配置类的全限定名对应的class，然后将所有自动配置类加载到Spring容器中
5. 自动配置生效
    - 每一个XxxxAutoConfiguration自动配置类都是在某些条件之下才会生效的
    - 这些条件的限制在Spring Boot中以注解的形式体现
    - 即使用一些条件注解类 @ConditionalOnBean：当容器里有指定的bean的条件下。 @ConditionalOnMissingBean：当容器里不存在指定bean的条件下。 @ConditionalOnClass：当类路径下有指定类的条件下。 @ConditionalOnMissingClass：当类路径下不存在指定类的条件下。 @ConditionalOnProperty 指定的属性是否有指定的值，比如@ConditionalOnProperties(prefix=”xxx.xxx”, value=”enable”, matchIfMissing=true)，代表当xxx.xxx为enable时条件的布尔值为true，如果没有设置的情况下也为true。
6. 面试回答可以为: Spring Boot启动的时候会通过@EnableAutoConfiguration注解找到META-INF/spring.factories配置文件中的所有自动配置类， 并对其进行加载，而这些自动配置类都是以AutoConfiguration结尾来命名的， 它实际上就是一个JavaConfig形式的Spring容器配置类， 它能通过以Properties结尾

命名的类中取得在全局配置文件中配置的属性如：server.port， 而XxxxProperties类是通过 @ConfigurationProperties注解与全局配置文件中对应的属性进行绑定的。

7. springboot配置文件中常用配置有哪些? properties, yml, yaml, bootstrap

3 springboot项目如何打包,部署,运行?

1 jar包方式

1. 直接在idea中执行打包命令: mvn package
2. 打包后进入到jar包目录, 然后进入cmd用命令, java -jar jar名名称

2 war包方式

1. pom.xml中配置打包方式为war包
2. 执行打包命令
3. 把打成的war包放到tomcat的webapps目录中, 启动tomcat即可

4 spring是如何控制事务的?

使用aop控制事务, 使用动态代理技术

5 springmvc常用注解及作用?

1 @Controller 定义了一个控制器类

2 @RequestMapping 用来处理请求地址映射的注解

3 @Resource 和 @Autowired 做bean的注入时使用

4 @ModelAttribute 和 @SessionAttributes

1. 该Controller的所有方法在调用前，先执行此 @ModelAttribute方法

   可用于注解和方法参数中，可以把这个 @ModelAttribute特性，应用在BaseController当中，所有的Controller继承BaseController，即可实现在调用Controller时，先执行 @ModelAttribute方法。

2. @SessionAttributes即将值放到session作用域中，写在class上面

3. 传递和保存数据

5 @PathVariable 用于将请求URL中的模板变量映射到功能处理方法的参数上，即取出uri模板中的变量作为参数

6 @requestParam SpringMVC后台控制层获取请求参数

7 @ResponseBody 该注解用于将Controller的方法返回的对象，通过适当的HttpMessageConverter转换为指定格式后，写入到Response对象的body数据区

使用时机：返回的数据不是html标签的页面，而是其他某种格式的数据时（如json、xml等）使用；

8 @Component 相当于通用的注解，当不知道一些类归到哪个层时使用，但是不建议。

9 @Repository 于注解dao层，在daoImpl类上面注解。

10 @Service

11 @GetMapping, @PostMapping, @PutMapping, @DeleteMapping

6 springmvc的工作流程是什么?

1、 用户发送请求至前端控制器DispatcherServlet

2、 DispatcherServlet收到请求调用HandlerMapping处理器映射器

3、 处理器映射器找到具体的处理器(可以根据xml配置、注解进行查找)，生成处理器对象及处理器拦截器(如果有则生成)一并返回给DispatcherServlet

4、 DispatcherServlet调用HandlerAdapter处理器适配器

5、 HandlerAdapter经过适配调用具体的处理器(Controller，也叫后端控制器)

6、 Controller执行完成返回ModelAndView

7、 HandlerAdapter将controller执行结果ModelAndView返回给DispatcherServlet

8、 DispatcherServlet将ModelAndView传给ViewReslover视图解析器

9、 ViewReslover解析后返回具体View

10、DispatcherServlet根据View进行渲染视图（即将模型数据填充至视图中）

11、DispatcherServlet响应用户

7 mybatis的工作原理是什么?

1. mybatis底层还是采用原生jdbc来对数据库进行操作的，
    - 只是通过 SqlSessionFactory，SqlSession Executor,StatementHandler，
    - ParameterHandler,ResultHandler和TypeHandler等几个处理器封装了这些过程
2. 执行器：Executor (update, query, flushStatements, commit, rollback, getTransaction, close, isClosed) 参数处理器：ParameterHandler (getParameterObject, setParameters) 结构处理器 ResultSetHandler (handleResultSets, handleOutputParameters) sql查询处理器：StatementHandler (prepare, parameterize, batch, update, query)
3. Mybatis工作原理： MyBatis应用程序根据XML配置文件创建SqlSessionFactory， SqlSessionFactory在根据配置获取一个SqlSession。(配置来源于两个地方，一处是配置文件，一处是Java代码的注解) SqlSession包含了执行sql所需要的所有方法， 可以通过SqlSession实例直接运行映射的sql语句， 完成对数据的增删改查和事务提交等， 用完之后关闭SqlSession SqlSessionFactory，SqlSession是对jdbc的封装, 底层还是调用的jdbc对数据库进行操作

2 mybatis的缓存的理解?

mybatis提供查询缓存，用于减轻数据压力，提高数据库性能 Mybatis内部存储缓存使用的是一个HashMap对象 key为 hashCode + sqlId + sql 语句 而value值就是从查询出来映射生成的java对象 1. 一级缓存 一级缓存是SqlSession级别的缓存 我们都知道在操作数据库时需要构造 sqlSession对象 而在sqlSession对象中有一个数据结构（HashMap）用于存储缓存数据 比如查询id为1的用户信息, 会先从sqlSession的缓存中查找, 如果有结果, 就不用再查数据库了 如果缓存中没有, 才去查询数据库 不同的sqlSession之间的缓存数据区域（HashMap）是互相不影响的 即一级缓存的不同sqlSession之间数据不共享 一级缓存默认启用, 想要关闭一级缓存可以再select标签上配置 flushCache = “true” 一级缓存存在于Sqlsession 的声明周期中，在同一个SqlSeesion中查询时， Mybatis会把执行的方法和参数通过算法生成缓存的键值，将键值和查询结果存入一个Map对象中， 如果同一个SqlSession中执行的方法和参数完全一致，那么通过算法会生成相同的键值， 当Map缓存对象中已经存在该键时，则会返回缓存中的对象 任何的insert，update，delete操作都会清空一级缓存 是为了保证缓存里面的数据肯定是准确数据避免脏读 2. 二级缓存 二级缓存以namespace为单位进行缓存，二级缓存默认开启，但是必须利用标签才能开启单个namespace二级缓存 相同namespace的mapper查询数据放在同一个区域 多个SqlSession去操作同一个Mapper的sql语句, 多个SqlSession可以共用二级缓存, 二级缓存是可以横跨跨SqlSession的二级缓存有时会出现脏读情况，是因为不同的namespace中相同的查询所致，每当进行update,delete,insert操作时，会以namespace为单位清空缓存 3. 当具有数据连接时，sqlSession会先访问二级缓存(如果二级缓存开启的情况下)，如果二级缓存开启并其中具有相应缓存则返回，如果没有命中或二级缓存没有开启，则在一级缓存中找寻数据，如命中，直接返回，反之查询数据库

8 mybatis中#{} 和{} 的区别?

${}字符串替换

1. 其中间的变量就是直接替换成值的

2. $ 的作用实际上是字符串拼接，所以要特别小心sql注入问题

#{}预编译处理

1. 会根据变量的类型来进行替换
2. #把参数部分用一个占位符？代替, 这样可以有效的防止sql注入
3. 具体执行时，# 都是用到了prepareStement，这样对效率也有一定的提升
4. #方式一般用于传入插入/更新的值或查询/删除的where条件 能同时用#和$的时候最好用#

9 springboot的异常处理?

一 自定义错误页面 只需要在resources/templates/目录下新建一个名为error的视图页面即可 当出现异常时，SpringBoot 会像/error 的 url 发送请求。在 springBoot 中提供了一个 叫 BasicExceptionController 来处理/error 请求，然后跳转到默认显示异常的页面来展示异常信息。 缺点：拦截所有错误，不灵活 二 使用注解 @ExceptionHandler处理 在controller类中添加错误处理的方法 返回ModelandView， 并使用 @ExceptionHandler(value={})注解，value中标注错误类型，当发生错误时会自动跳转到你设置的错误页面中 缺点：只能在一个controller中使用，不能跨controller使用 三 使用注解 @ControllerAdvice处理 定义一个类用来装所有的错误类型，并且使用 @controllerAdvice注解 其中可以定义处理错误的方法, 方法上使用 @ExceptionHandler(value={})注解 缺点：如果有多个错误还是要写多个方法，代码重复严重 四 使用 @Configuration注解处理 新建类，在类中使用 @Configuration注解，在类中定义方法，方法使用 @Bean注解，并且返回值为：SimpleMappingExceptionResolver 将所有的错误都定义到一个方法中，发生错误时，会根据判断自动跳转到指定错误页面 缺点：不能将错误信息传递到页面中 五 自定义 HandlerExceptionResolver 类处理异常 新建类，使用Configuration 注解并且实现HandlerExceptionResolver接口，实现方法 推荐使用第五种方法，既可以将错误定义到一个方法中，又可以传递错误信息到错误页面中

10 spring中如何配置拦截器, 过滤器?

1 拦截器: interceptor(拦截器)

1. 创建拦截器类实现HandlerInterceptor接口
2. 实现接口中的三个方法
    - boolean preHandle(request, response, Object handler){ }: 该方法在控制器方法执行之前执行
    - void postHandle(request, response, handler, modelAndView){ }: 该方法在控制器方法执行之后, 视图对象返回之前执行
    - void afterCompletion(request, response, handler, Exception e){ }: 该方法在流程都执行完毕后执行, 即服务器处理完成一次请求
3. 方法执行顺序
    - 一个拦截器: preHandle -> 控制器方法 -> postHandle -> afterCompletion
    - 两个拦截器: 拦截器1(配置在前, 先执行), 拦截器2
        - preHandle(1) -> preHandle(2) -> 控制器方法 -> postHandle(2) -> postHandle(1) -> afterCompletion(2) -> afterCompletion(1)
4. spring配置文件中需要配置拦截器以及拦截路径

2 过滤器

1. 方法一(spring):
    - 过滤器不是spring的, 而是web的是基于servlet的过滤器
    - 自定义类实现Filter接口
        - 重写方法doFilter()
    - 在web.xml中配置过滤器, 使用 filter 标签和 filter-mapping 标签
2. 方法二(springboot):
    - 自定义类实现Filter接口, 类上添加注解 @WebFilter

- springboot启动类上添加包扫描注解: @ServletComponentScan(basePackages = "自定义filter全限定类名")

11 spring管理的bean是否会存在并发?

1 存在

2 Spring默认的单例模式的bean

1. 因为是单例的，所以会避免不断的创建新的实例从而导致并发量很大时垃圾回收效率低的问题。
2. 也有缺点，就是状态不好管理，也就是说bean里面的全局变量不好管理，因为这样很容易会导致多线程问题。

3 一般来说我们用Spring管理的类一般是各种Service类，无需设计成有状态的bean，而状态最好不要在bean里面保存，因为集群环境下bean管理的状态会有问题。可以考虑使用缓存或者数据库来管理状态。

如果一个对象是单例的, 而这个对象中又有全局变量时, 在多线程中就会产生线程安全问题

12 springsession的原理是什么?

1 session只能在单台服务器上使用, 而在分布式或集群中, 用户访问一台服务器产生的session, 在访问另一台服务器时无法被识别

2 springsession就是在分布式或集群中使用的, 可以代替session

3 原理

1. 方式一: 利用redis来实现session同步，
   - 既当客户端访问服务器端的时候，将服务器端生成的session信息保存在redis中
   - 然后访问的时候通过redis获取session信息
   - springsession使用该方式, 支持数据库，redis等存储方式；
2. 方式二: 使用token 替代 session 即 auth2认证方案，一般用于移动端的开发

13 spring的类加载器是什么?

AppClassLoader: 应用程序类加载器 其他 1. 启动类加载器（Bootstrap ClassLoader）： 这个类加载器负责将存放在<JAVA_HOME>\lib目录中， 或者被-XbootClasspath参数所指定的路径中的， 并且是虚拟机识别的（仅按照文件名识别，如rt.jar， 名字不符合的类库即使放在lib目录中也不会被加载）类库加载到虚拟机内存中。 启动类加载器无法直接被java程序引用， 用户在编写自定义类加载器时，如果需要把加载请求委派给引导类加载器，那直接使用null代替即可。 2. 扩展类加载器（Extension ClassLoader）： 这个加载器由sun.misc.Launcher$ExtClassLoader实现， 它负责加载<JAVA_HOME>\lib\ext目录中，或者被java.ext.dirs系统变量所指定的路径中的类库， 开发者可以直接使用扩展类加载器。 3.、 应用程序类加载器（Application ClassLoader）： 这个类加载器由sun.misc.Launcher$AppClassLoader实现。 由于这个类加载器是ClassLoader中的getSystemClassLoader()方法的返回值， 所以一般也称它为系统类加载器，它负责加载用户类路径（classpath）上指定的类库， 开发者可以直接使用这个类加载器，如果应用程序中没有自定义过自己的类加载器， 一般情况下这个就是程序中的默认类加载器

14 springcloud的组件有哪些? 作用分别是什么?

1 eureka: 治理组件

1. 服务注册中心，特性有失效剔除、服务保护。
2. 还有consul, nacos作用和eureka类似

2 ribbon: 客户端负载均衡组件

1. 客户端负载均衡，特性有区域亲和、重试机制

3 hystrix: 服务容错保护组件

1. 客户端容错保护，特性有服务降级、服务熔断、请求缓存、请求合并、依赖隔离。

4 feign: 声明式服务调用组件

1. 声明式服务调用，本质上就是Ribbon+Hystrix

5 zuul: API网关治理组件

1. API服务网关，功能有路由分发和过滤。
2. 类似的还有gateway

6 config: 分布式配置中心组件

1. 分布式配置中心，支持本地仓库、SVN、Git、Jar包内配置等模式

7 bus: 消息总线组件

1. 消息总线，配合Config仓库修改的一种Stream实现

8 stream: 消息驱动组件

1. 消息驱动，有Sink、Source、Processor三种通道，特性有订阅发布、消费组、消息分区。

9 sleuth: 分布式跟踪组件

1. 分布式服务追踪，需要搞清楚TraceID和SpanID以及抽样，如何与ELK整合。

10 Dashboard，Hystrix仪表盘，

1. 监控集群模式和单点模式，其中集群模式需要收集器Turbine配合。

15 dubbo和springcloud的区别和优缺点?

1 dubbo

1. dubbo由于是二进制的传输，占用带宽会更少
2. dubbo的开发难度较大，原因是dubbo的jar包依赖问题很多大型工程无法解决
3. dubbo的注册中心可以选择zookeeper,redis等多种

2 springcloud

1. springCloud是http协议传输，带宽会比较多，同时使用http协议一般会使用JSON报文，消耗会更大
2. springcloud的接口协议约定比较自由且松散，需要有强有力的行政措施来限制接口无序升级
3. springcloud的注册中心能用eureka, consul, nacos

3 spring cloud整机，dubbo需要自己组装；整机的性能有保证，组装的机子更自由

4 区别

1. dubbo是rpc框架, spirngcloud是微服务框架
2. Dubbo基于RPC；SpringCloud基于HTTP的Rest风格
3. dubbo只是实现了服务治理, 但springcloud中包含了微服务架构下的各个方面的组件, 更加全面

16 gateway的动态路由如何实现?

1 动态路由: uri: lb://GATEWAY-PROVIDER

1. "//"后面的为在服务注册中心中的服务名称

17 鉴权框架的了解和使用?

1. 单体应用下的常用方案
    - 传统的单体应用，一般会写一个固定的认证和鉴权的包，里面包含很多的认证和鉴权的类
    - 当用户请求时可以利用session的方式，把用户存入session并生成一个sessionid，之后返回客户端
    - 客户端可以存在cookie里，从而在后续的请求中顺利通过验证。
    - 常用框架：shiro 、自定义注解、Filter拦截等
2. 微服务下的SSO单点登陆方案
    - 单点登录(Single Sign On),简称为 SSO, 是目前比较流行的企业业务 整合的解决方案之一
    - SSO的定义是在多个应用系统中,用户只需要登录一次就可以访问所有相互信任的应用系统

- 但是针对微服务(服务之间调用)：每个 服务都进行每个用户端的sso动作，那么每个服务里都会做用户的认证和鉴权，可能保存每个用户的信息或者每个用户都会和鉴权服务打交道，这些情况都会带来非常大的网路开销和性能消耗，也有可能会造成数据的不一致，所以不建议用这种方案。

3. 分布式Session与网关结合方案
   - 步骤
     - 用户在网关进行sso登陆 ，进行用户认证，检查用户是否存在和有效
     - 如果用户通过，则将用户信息存储在第三方中间件中，如mysql、redis
     - 后端可以从共享存储拿到用户的数据
   - 很多 场景下，这种方案是推荐的，因为方便扩展，也 可以保证高可用的方案
   - 但是这种方案的缺点是依赖于第三方中间件，且这些部件需要做高可用，并且增加安全的控制，所有对于实现有一定的复杂度
4. 客户端Token与 网关结合方案
   - 实现步骤
     - 客户端持有一个token，通常可用jwt或者其它加密的算法实现自己的一种Token，然后通过token保存用户的信息
     - 发起用户请求并携带token，token传到网关层后，网关 层进行认证和校验
     - 校验通过，携带token到后端服务中
     - 如果涉及到用户的大量信息存放，token就有可能不太合适（或者用中间件来存放）
   - 这种方案也是业界很常用的方案，但是对于 token来说，他的注销有一定的麻烦，需要在网关层进行Token的注销
5. 浏览器Cookie与网关结合方案
   - 实现: 把用户的信息存放在cookie里，然后通过网关来解析cookie，从而 获取用户的相关信息
   - 这种方式在一些老系统做改造时遇到的比较多，适合做为老系统改造时采取的方案，因为很多系统需要继承，这时cookie在别的

系统中也是同样的适用。

6. 网关Token和 服务间鉴权结合
    - 实现步骤
        - 在gateway网关层做认证，通过用户校验后，传递用户信息到header中，后台做服务在收到header后进行解析，解析完后查看是否有调用此服务或者某个url的权限，然后完成鉴权
        - 从服务内部发出的请求，在出去时进行拦截，把用户信息保存在header里，然后传出去,被调用方取到header后进行解析和鉴权

2 SpringSecurity

配置文件中配置相关配置 ublic class UserService implements UserDetailsService { public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException { 用户名密码判断 授予权限 } }
controller方法添加注解 @PreAuthorize("hasAuthority('add')")
@PreAuthorize("hasRole('ROLE_ADMIN')")

3 Shiro

4 SpringSecurity和Shiro都是基于RBAC的

1. RBAC是Role Based Access Control的英文缩写，意思是基于角色访问控制。
2. 在RBAC中，权限与角色相关联，用户通过成为适当角色的成员而得到这些角色的权限
    - 这就极大地简化了权限的管理。
    - 这样管理都是层级相互依赖的，权限赋予给角色，而把角色又赋予用户，
    - 这样的权限设计很清楚，管理起来很方便。
3. spring security和shiro的异同
    1. 相同点

- 认证功能2、授权功能3、加密功能4、会话管理5、缓存支持
- rememberMe功能

2. 不同点

- Spring Security 基于Spring 开发，项目若使用 Spring 作为基础，配合 Spring Security 做权限更加方便，而 Shiro 需要和 Spring 进行整合开发；
- Spring Security 功能比 Shiro 更加丰富些，例如安全维护方面；
- Spring Security 社区资源相对比 Shiro 更加丰富；
- Spring Security对Oauth、OpenID也有支持,Shiro则需要自己手动实现。而且Spring Security的权限细粒度更高 spring security 接口 RequestMatcher 用于匹配路径,对路径做特殊的请求，类似于shiro的抽象类 PathMatchingFilter，但是 RequestMatcher 作用粒度更细
- Shiro 的配置和使用比较简单，Spring Security 上手复杂些；
- Shiro 依赖性低，不需要任何框架和容器，可以独立运行.Spring Security 依赖Spring容器
- shiro 不仅仅可以使用在web中，还支持非web项目它可以工作在任何应用环境中。在集群会话时Shiro最重要的一个好处或许就是它的会话是独立于容器的。

apache shiro的话，简单，易用，功能也强大，spring官网就是用的shiro，可见shiro的强大

18 mybatis同时操作通一条数据该怎么解决并发问题?

1 发现并发问题

1. 可以给表增加一个版本号字段, 新的时候给版本号字段加上 1
2. 更新执行后, 会返回一个更新结果的行数,
3. 如果更新执行返回的数量是 0 表示产生并发修改了, 需要重新获得最新的数据后再进行更新操作

2 Hibernate、JPA 等 ORM 框架或者实现，是使用版本号

1. 判断 UPDATE 后返回的数值, 如果这个值小于 1 时则抛出乐观锁并发修改异常。

3 使用版本号的方式属于乐观锁

19 mybatis中传递参数有哪些方式?

1.第一种方式 匿名参数 顺序传递参数, (param1, param2,...), (arg0, arg1,...)

```
1. mapper: List<Employee> selectByGenderAndAge(Short gender,String age );
2. xml:
      <select id="selectByGenderAndAge" resultMap="BaseResultMap" >
        select * from employee where gender = #{param1} and age = #{param2}
      </select>
  •   或在#{}中使用 arg0, arg1
3. 这种传参方式的缺点是不够灵活，必须严格按照参数顺序来引用
```

2.第二种方式 使用 @Param注解

```
1. mapper: List<Employee> selectByGenderAndAge( @Param("gender") Short gender,@Param("age") String age );
  *  使用 @Param注解显示的告诉mybatis参数的名字，这样在xml中就可以按照参数名去引用了
2. xml:
      <select id="selectByGenderAndAge" resultMap="BaseResultMap" >
        select * from employee where gender = #{gender} and age = #{age}
      </select>
```

3.使用Map传递参数

```
1. mapper: List<Employee> selectByMapParams(Map params);
  *  所有要传递的参数都被封装到map中了
2. xml:
      <select id="selectByMapParams" resultMap="BaseResultMap" parameterType="map">
        select * from employee where gender = #{gender} and age = #{age}
      </select>
  •   #{}中使用的值就是map值的key值
```

4.用过java bean传递多个参数

```
1. mapper: List <Employee> selectByBeans(Employee employee);
  *  数据被封装到一个javaBean中
2. xml:
      <select id="selectByBeans" resultMap="BaseResultMap" parameterType="com.wg.demo.po.Employee">
        select * from employee where gender = #{gender} and age = #{age}
      </select>
  •   #{}中填写的为javabean中的属性的名称
```

5.直接使用JSON传递参数

```
1. mapper: List <Employee>
findByJSONObject(JSONObject params);
  *  controller的方法中接收参数是就是用 @RequestBody
JSONObject params 来接收json数据
2. xml
      <select id="findByJSONObject"
resultMap="BaseResultMap"
parameterType="com.alibaba.fastjson.JSONObject">
        select * from employee where gender = #
{gender} and age = #{age}
      </select>
```

- #{}中填写的为json中的key值

6.传递集合类型参数List、Set、Array, 需要使用标签遍历

```
1. mapper: List <Employee> findByList(List list);
2. xml:
      <select id="findByList"
resultMap="BaseResultMap" >
        SELECT * from employee where age in
          <foreach collection="list" open="("
separator="," close=")" item="age">
            #{age}
          </foreach>
      </select>
```

- 这里foreach表示循环操作，遍历取出集合中的数据

7.参数类型为对象+集合

```
1. 该类参数与java Bean参数形式类似，但是在对象的属性中不仅有基本类型的变量，还有集合类型的变量
  * 集合中存储的是对象数据
2. mapper: List <Employee> findByDepartment(@Param("department")Department department);
3. xml:
      <select id="findByDepartment" resultMap="BaseResultMap" parameterType="com.wg.demo.po.Department">
        SELECT * from employee where dept_id =#{department.id} and age in
        <foreach collection="department.employees" open="(" separator="," close=")" item="employee">
          #{employee.age}
        </foreach>
      </select>
```

20 spring中bean的生命周期? 作用域有哪些?

1 spring中bean的生命周期

1. 单例bean对象
   - 对象创建: 当应用加载, 创建spring容器时, 对象就被创建了
   - 对象存活: 只要容器存在, 对象一直存活
   - 对象销毁: 当应用卸载, 销毁容器时, 对象被销毁
2. 多例bean对象
   - 对象创建: 当使用对象时, 创建新的对象实例
   - 对象存活: 只要对象在使用中, 对象一直存活
   - 对象销毁: 当对象长时间不用时, 被java的垃圾回收器回收

2 作用域有哪些

1. singleton: 单例的(默认值), bean.xml文件加载后立即创建其中的对象

2. prototype: 多例的, 对象在使用时才会被创建
3. request: 作用与web应用的请求范围, web项目中, 将创建的对象存入request域中
4. session: 作用于web应用的会话范围, web项目中, 将创建的对象存入session域中
5. global-session: 作用于集群环境的会话范围(全局会话范围)

21 怎么实现mybatis批量插入?

1 xml配置

1. 方式一: 写一个insert标签, java代码重复多次调用
2. 方式二: insert标签中使用foreach标签遍历要插入的数据, 并拼接到sql语句上

2 注解方式

22 junit如何使用?

1 spring

1. 导入junitjar包:
2. 测试类添加注解
    - @Runwith(SpringJunit4ClassRunner.class)
    - @ContextConfiguration(classes={配置类的路径})
        - value属性指定配置文件路径: value="classpath:文件路径"

2 springboot

1. 坐标: org.springframework.boot spring-boot-starter-test
2. 测试类添加注解
    - @Runwith(SpringRunner.class)
    - @SpringBootTest(classes = 引导类名.class)
        - 如果测试类所在包名和引导类所在包名相同时, 可以不写classea参数

3 junit注解

1. @Test: 测试方法
2. @Before: 在测试方法执行之前执行
3. @After: 在测试方法执行之后执行
4. @Ignore: 忽略方法, 即不对该方法进行测试
5. @BeforeClass: 在测试类执行之前执行
6. @AfterClass: 在测试类执行之后执行

23 spring的工厂模式如何使用?

24 springboot是如何对线程进行封装的?

1 你直接new出来的对象是没法使用的，

1. 就算你能new成功，但是bean里面依赖的其他组件比如Dao，
2. 是没法初始化的，因为你饶过了spring，
3. 默认的spring初始化一个类时，其相关依赖的组件都会被初始化，
4. 但是自己new出来的类，是不具备这种功能的

2 需要通过spring来获取我们自己的线程类

3 springboot对多线程的封装

1. Spring是通过任务执行器(TaskExecutor)来实现多线程和并发编程
2. 使用ThreadPoolTaskExecutor来创建一个基于线城池的TaskExecutor
    - 在使用线程池的大多数情况下都是异步非阻塞的
3. 注解 @EnableAsync: 来开启Springboot对于异步任务的支持
4. 然后在实际执行的方法上配置注解 @Async上声明是异步任务

4 线程池的配置方式

1. 方式一: 创建一个配置类, 并实现AsyncConfigurer接口
    - 类上添加注解

        - @Configuration
        - @EnableAsync // 开启Springboot对于异步任务的支持
    - 类中重写接口中的方法

```
public Executor getAsyncExecutor() {
      ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();
      executor.setCorePoolSize(5);
      executor.setMaxPoolSize(15);
      executor.setQueueCapacity(25);
      executor.initialize();
      return executor;
    }
    public AsyncUncaughtExceptionHandler getAsyncUncaughtExceptionHandler() {
      return null;
    }
```

2. 方式二: 创建一个配置类
    - 类上添加注解
        - @Configuration
        - @EnableAsync // 开启Springboot对于异步任务的支持
    - 类中添加方法: public Executor getAsyncExecutor() { }
        - 方法中内容和上面的相同
        - 方法上添加注解: @Bean
3. 使用上述两种方式之一后, 使用时只需要在对应的方法上添加 @Async 注解即可
    - 如在service的一个方法上添加 @Async 注解, 表明该方法是异步方法
    - 如果Async注解在类上，那表明这个类里面的所有方法都是异步的
    - 在调用该方法时, 框架中会自动把该方法用一个新的线程来执行

25 springboot中starter种类有哪些?如何写一个springboot的starter组件?

1 常用starter

spring-boot-starter-web spring-boot-starter-thymeleaf * 依赖作用:模板配置 * 分析 * 在springboot中使用 @RestController时候不需要模板技术 * 但是如果使用 @Controller不加入模板技术的话, 启动服务器不会报错, 但是无法网页访问项目 * 且如果设置了html网页, 则建议使用 @Controller, 且添加该模板坐标 spring-boot-starter-data-redis spring-boot-starter-data-jpa * 用于操作mysql的数据库用的, 用于表的基本CRUD * 复杂的查询操作则使用mybatis spring-boot-starter-data-elasticsearch * elasticSearch 其实也算是一种nosql数据库 * 如果是用spring-boot-starter-data-elasticsearch的话在增删方面用起来和JPA几乎没什么两样 spring-boot-starter-data-mongodb spring-boot-starter-test spring-boot-starter-jdbc spring-boot-starter-aop spring-boot-starter-security spring-boot-starter-actuator spring-boot-starter-tomcat

2 如何写一个springboot的starter组件

```
1. 创建starter项目，spring应用
2. pom文件添加一个SpringBoot坐标，以及其他相关依赖
3. 创建xxxAutoConfiguration类
  *  加上注解
    *  @Configuration（表明这是一个配置类）
    *  @EnableConfigurationProperties(RedisProperty.class)（从名称上可以看出开启自动配置，后面跟上属性的property）
    *  @ConditionalOnClass(Redisson.class)（当其他项目引用这个组件的时候会根据对应的条件选择注入bean）
4. RedisProperty类(ConfigurationProperties) 就是我们的属性设置
  *  添加 @ConfigurationProperties(prefix="前缀")
5. 配置文件: 在resource文件夹下创建META_INF文件夹，spring.factories文件
  *  org.springframework.boot.autoconfigure.EnableAutoConfiguration=xxxAutoConfiguration全限定类名
```

```
6. 添加配置文件
  *  additional-spring-configuration-metadata.json, 在META_INF文件夹中添加该文件
    *  文件内容
          {
            "properties": [
            {
              "name": "xjp.redisson.host",
              "type": "java.lang.String",
              "description": "redis服务器地址",
              "defaultValue": "localhost"
            },{
              "name": "xjp.redisson.port",
              "type": "java.lang.Integer",
              "description": "redis服务器的端口",
              "defaultValue": 6379
            }
            ]
          }
  *  作用: 为了在yml文件中配置属性时有提示
```

26 设计一个开放接口, 如何保证接口安全性?

1 数据加密

1. 数据在传输过程中是很容易被抓包
    - 如果直接传输比如通过http协议, 那么用户传输的数据可以被任何人获取, 所以必须对数据加密
    - 常见的做法对关键字段加密比如用户密码直接通过md5加密
    - 现在主流的做法是使用https协议
        - 在http和tcp之间添加一层加密层(SSL层)，这一层负责数据的加密和解密；
2. 注意: 加密的部分其实只是在外网
    - 数据进入服务器后就会被解析为正常的数据,

- 而在服务器内部(内网)可能需要经过很多服务跳转,

2 数据加签

1. 数据加签就是由发送者产生一段无法伪造的一段数字串，来保证数据在传输过程中不被篡改
2. 数据加签可以防止内网中数据被篡改

3 时间戳机制

1. 数据是很容易被抓包的，但是经过如上的加密，加签处理，就算拿到数据也不能看到真实的数据
    - 但是有不法者不关心真实的数据，而是直接拿到抓取的数据包进行恶意请求
2. 时间戳机制: 在每次请求中加入当前的时间，服务器端会拿到当前时间和消息中的时间相减
    - 看看是否在一个固定的时间范围内比如5分钟内
    - 这样恶意请求的数据包是无法更改里面时间的，所以5分钟后就视为非法请求了；

4 AppId机制: 相当于用户名和密码的验证机制, 这里使用的是appid+秘钥

5 限流机制

1. 本来就是真实的用户，并且开通了appid，但是出现频繁调用接口的情况
2. 这种情况需要给相关appid限流处理

6 黑名单机制

1. 如果此appid进行过很多非法操作
2. 或者说专门有一个中黑系统，经过分析之后直接将此appid列入黑名单，所有请求直接返回错误码

7 数据合法性校验

1. 这个可以说是每个系统都会有的处理机制，只有在数据是合法的情况下才会进行数据处理；每个系统都有自己的验证规则，当然也可

能有一些常规性的规则，比如身份证长度和组成，电话号码长度和组成等等
2. 各个系统都有自己的处理数据的格式

四 实用技术

1 文件上传和文件下载如何实现?

1 文件上传(spring中)

1. 客户端表单实现
    - 表单格式
        - 表单项, 文件上传input输入框设置属性: type="file"
        - 表单的提交方式是post
        - 表单的enctype属性是多部分表单形式, 即 enctype="multipart/form-data"
2. 服务端代码实现 配置多媒体解析器
    - controller中方法添加参数: MultipareFile uploadFile
    - uploadFile和文件上传输入框的name属性值一致
    - MultipareFile类中的方法
        - 获得上传文件的名称: uploadFile.getOriginalFilename();
            - 保存文件: uploadFile.transferTo(new File("文件路径" + 文件名称))
            - isEmpty(): 判断文件是否为空

2 多文件上传(spring中)

1. 表单上传文件输入框改为多个, name属性值相同
2. controller中方法参数改为MultipareFile[]数组格式
    - 注意: 这里不能使用 @RequestBody 注解

3 文件上传(springboot中)

1. 表单部分和上面的相同

2. controller中方法添加参数: MultipareFile uploadFile
    - 其他方法参看上面的相关代码

4 文件下载

1. controller方法
    - 设置相应头信息
        - response.setContentType("application/force-download");// 设置强制下载不打开 1 浏览器收到服务器的返回数据之后, 会直接打开一个下载窗口, 进行下载
        - response.addHeader("Content-Disposition", "attachment;fileName=" + fileName);// 设置文件名
    - 使用这个输出流: OutputStream os = response.getOutputStream();
        - 将文件内容发送到客户端

2 第三方的工作流有哪些?如何使用的?

1 工作流

1. 工作流是一项分离业务操作和系统流程的技术
2. 组成
    - 实体(Entity)
        - 实体是工作流的主体，是需要随着工作流一起流动的物件(Object)
        - 例如，在一个采购申请批准流程中，实体就是采购申请单；在公文审批流程中，实体就是公文。
    - 参与者(Participant)
        - 参与者是各个处理步骤中的责任人
        - 可能是人，也可能是某个职能部门，还可能是某个自动化的设备
    - 流程定义(Flow Definition)
        - 流程定义是预定义的工作步骤，它规定了实体流动的路线。

        - 它可能是完全定义的，即对每种可能的情况都能完全确定下一个参与者
        - 也可能是不完全定义的，需要参与者根据情况决定下一个参与者；
    - 工作流引擎(Engine)
        - 工作流引擎是驱动实体按流程定义从一个参与者流向下一个参与者的机制
    - 前三个要素是静态的，而第四个要素是动态的，它将前三者结合起来，是工作流的核心组成元素。

2 JBPM

1. jBPM，全称是Java Business Process Management，是一种基于J2EE的轻量级工作流管理系统。
2. 特色
    - jBPM的一个特色是采用了它自己定义的JBoss jBPM Process definition language (jPdl)。jPdl认为一个商务流程可以被看作是一个UML状态图。jPdl就是详细定义了这个状态图的每个部分，如起始、结束状态，状态之间的转换等。
    - BPM的另一个特色是它使用Hibernate来管理它的数据库。Hibernate是目前Java领域最好的一种数据持久层解决方案。通过Hibernate，jBPM将数据的管理职能分离出去，自己专注于商务逻辑的处理。
3. jbpm工作流步骤：
    - 加载（发布）流程定 这个意思是，我们通过jbpm的designer插件，或者是用其他工具，制定出processDefinition， 然后将其加载到应用中的过程。这个加载可以是写入内存中，或者是直接写入数据库等。
    - 启动流程 创建流程实例的过程。具体创建实例的方法有多种，可根据自己的需要自行选择。
    - 处理任务 流程流转的过程中，JBPM引擎会为我们生成任务的实例，我们就需要针对这些任务实例来进行处理，然后结束这些任务实例，并推动流程的流转。

- 记录流程的相关状态 记录流程状态这点包括且不限于以下内容：
  - 流程实例的开启
  - 任务实例的创建
  - 任务实例的开始执行
  - 任务实例的结束
  - 流程实例的结束

3

1 对activemq, rabbitmq kafka的了解和使用?

1. ActiveMQ: 基于JMS, java语言开发
   - 单击吞吐量: 万级(最差)
   - 消息延迟: 毫秒级
2. RabbitMQ: 基于AMQP协议, erlang语言开发, 稳定性好
   - 单击吞吐量: 万级(其次)
   - 消息延迟: 微秒级
3. Kafka: 类似MQ的产品, 分布式消息系统, 高吞吐量
   - 单击吞吐量: 十万级(次之)
   - 消息延迟: 毫秒级
   - 只支持主要的MQ功能, 毕竟是为大数据领域准备的

2 消息发送失败如何处理?

1. rabbitmq中的三种确认模式 confirm确认模式: 消息从producer到exchange后, 交换机会返回一个confirmCallback确认消息 配置文件配置 定义回调方法 return退回模式: 消息从 exchange 向 queue 投递失败则会返回一个 returnCallback 配置文件 回调方法 ack确认模式: 中间件向消费端转发消息, 消费端接受到消息之后的返回确认操作

3 如何防止消息的重复消费?

1. 对于需要保存到数据库的数据，我们可以设置某条数据的某个值，比如订单号之类的，设置一个唯一索引，这样的话即使重复消费也不会生效数据
2. 乐观锁，也就是我们每次插入一条数据或者更新的时候判断某个版本号是不是与预期一样，如果不是，那么就不进行操作
3. 使用redis进行存储，保留我们消费过的数据的每个特征，然后每次操作数据的时候先去redis进行判断，如果存在的话，那么这条数据就是重复消费的，然后我们可以丢弃或者做其他处理。

4 svn和git冲突如何解决?

pull一下，更新程序 同步 查看冲突代码 人工修改 保持本地最新版 提交修改 再pull一下 更新 人工合并 更新到最新版 commit提交 push git代码冲突解决 修改文件后点击提交可以成功, 提交是提交到本地了, 但是pull推送代码时会出现冲突 idea中点击pull推送后会弹出一个Push Rejected窗口 push of current branch master was rejected, 推送当前分支有冲突 然后点击merge(合并) 之后idea会将远程仓库中的代码拉取下来, 同时和有冲突的代码显示在一个文件中, 并用特殊的分隔符标记 还会弹出Files Merged with Conflicts窗口 窗口中有三个选项 Accept Yours: 使用你的代码 Accept Theirs: 使用其他人的代码 Merge...: 合并代码 点击Merge...后会弹出一个 Merge Revisions for 窗口 在这个窗口中可以选择那些代码最终被留下 选择完毕后点击Apply按钮即可 查看历史版本 idea中: VCS -> Git -> Show History 在打开的视图中, 选择对应的版本, 点击版本, 右键单击选择 show diff(查看该版本和当前版本的代码差异) 如果要回滚到之前的版本, 选择版本后, 右键单击选择 copy Revision Number(复制版本号) 然后在VCS菜单, 或项目名右键单击选择 git -> Repository -> ResetHEAD... 之后弹出一个Reset Head窗口(回滚的窗口) Reset Type选择 Hard To Commit中粘贴复制的版本号 点击Reset按钮就可以回滚到以前的版本 快捷键: ctrl shift k: 打开推送窗口 创建分支 项目右键单击, 选择 Git -> Repository -> Branches... 弹出Git Branches窗口, 点击New Branch, 输入分支名称, 点击ok

5 如何测试自己的接口?

1 Swagger(网页ui)

2 postman(软件)

3 knife4j

6 poi导入excel时如果数据溢出如何解决?

可以将excel的xlsx格式转换为易读取的csv格式进行读取

7 linux常用命令有哪些?

ps -ef | grep ssh netstat -nltp: 查看系统中网络进程的端口监听情况可以查看端口号

8 如何调用第三方的接口?

1 第三方API是由第三方（通常是一些公司）提供的API，允许您通过API接口访问和调用其功能，并在您自己的站点上使用它

2 比如我们可以使用阿里云的短信发送API、邮箱推送API、文件存储API来调用相关的功能

3 前端, ajax调用第三方接口会出现跨域问题

4 服务端:

1. 使用工具类调用第三方接口, 或使用spring restTemplate对象调用
2. 使用feign调用
    - 接口就是服务

9 分布式, 集群, 微服务的理解?

1 分布式

1. 分布式：一个业务分拆多个子业务，部署在不同的服务器上
    - 分布式（Distribution）这个术语，指的是各个异构的节点形成的系统，所谓异构，就是结构不同、功能不同的节点。
2. 分布式和集群都是为了解决两个问题：
    - 高吞吐量（throughput）
    - 高可用（availability): 一个服务挂掉, 其他服务还能正常运行

2 集群

1. 集群：同一个业务，部署在多个服务器上
    - 集群的各个节点呢？他们是同构的

3 微服务

1. 微服务和分布式比较相似
    - 微服务的意思也就是将模块拆分成一个独立的服务单元通过接口来实现数据的交互。
    - 微服务与分布式的细微差别是，微服务的应用不一定是分散在多个服务器上，他也可以是同一个服务器
    - 分布式和微服的架构很相似，只是部署的方式不一样而已
2. 微服务是一种架构风格，一个大型复杂软件应用由一个或多个微服务组成
    - 统中的各个微服务可被独立部署，各个微服务之间是松耦合的
    - 每个微服务仅关注于完成一件任务并很好地完成该任务
    - 在所有情况下，每个任务代表着一个小的业务能力
3. 分布式也是属于微服务的

10 分布式事务的逻辑处理?怎么实现的?

1 分布式事务

1. 如数据库分库分表之后，原来在一个单库上的操作可能会跨越多个数据库
2. 系统服务化拆分之后，原来的在一个系统上的操作可能会跨越多个系统
3. 我们平时经常使用到的缓存(如redis、memcache等)也可能涉及分布式事务
    - 缓存和数据库是两个不同的实体，如何保证数据在缓存和数据库间的一致性也是要重点考虑的

2 分布式事务就是指事务要处理的资源分别位于分布式系统中的不同节点之上的事务

这些小操作分布在不同服务器上，分布式事务需要保证这些小操作要么全部成功，要么全部失败

3 处理方法

1. 避免出现分布式事务
2. 事务补偿 可以用来解决缓存和数据库中的数据不一致问题, (例如网络抖动造成短暂的缓存不可用)就会造成缓存和DB的不一致
    - 创建一张事务补偿表transaction_log
    - 在更新数据前，先将要更新的模型数据记录到transaction_log中, 如记录更新数据的id值等
    - 记录成功后，再去更新数据库表中的内容，最后更新缓存中的数据 缓存更新成功后，就可以删除transaction_log表中对应的记录
    - 假设在更新完数据库后, 由于网络抖动等原因导致缓存更新失败, 则transaction_log表中对应的记录就会一直存在，表示这个事务没有完成的一种记录
    - 应用会创建一个定时任务，周期性的扫描transaction_log表中的记录 发现有符合条件的记录，就尝试执行补偿逻辑, 即把数据库中的记录同步到缓存中 补偿任务执行完成后，就可以删除transaction_log表中对应的记录 如果补偿任务执行再次失败，就保留transaction_log表中的记录，等待下个周期再次执行
3. 事务型消息

4. 两阶段提交
    - 第一阶段(投票阶段)
        - 协调者节点向所有参与者节点询问是否可以执行提交操作(vote)，并开始等待各参与者节点的响应
        - 参与者节点执行询问发起为止的所有事务操作，并将Undo信息和Redo信息写入日志。（注意：若成功这里其实每个参与者已经执行了事务操作）
        - 各参与者节点响应协调者节点发起的询问。 1 如果参与者节点的事务操作实际执行成功，则它返回一个”同意”消息； 2 如果参与者节点的事务操作实际执行失败，则它返回一个”中止”消息。
    - 第二阶段(提交执行阶段) 当协调者节点从所有参与者节点获得的相应消息都为”同意”时：
        - 协调者节点向所有参与者节点发出”正式提交(commit)”的请求。
        - 参与者节点正式完成操作，并释放在整个事务期间内占用的资源。
        - 参与者节点向协调者节点发送”完成”消息。
        - 协调者节点受到所有参与者节点反馈的”完成”消息后，完成事务

11 事务的特性?

1. 原子性（Atomicity）：事务中所有操作是不可再分割的原子单位。事务中所有操作要么全部执行成功，要么全部执行失败。
2. 一致性（Consistency）：事务执行后，数据库状态与其它业务规则保持一致。如转账业务，无论事务执行成功与否，参与转账的两个账号余额之和应该是不变的。
3. 隔离性（Isolation）：隔离性是指在并发操作中，不同事务之间应该隔离开来，使每个并发中的事务不会相互干扰
4. 持久性（Durability）：一旦事务提交成功，事务中所有的数据操作都必须被持久化到数据库中，即使提交事务后，数据库马上崩溃，在数据库重启时，也必须能保证通过某种机制恢复数据。

2 分布式如何保证数据一致性?

12 docker的常用命令有哪些?

1 docker images: 查看镜像信息列表 镜像是静态的

2 docker ps -a: 查看运行中的所有容器

3 docker pull [images]:[version]: 从dockerhub拉取指定镜像

4 docker rm [containerID]: 删除容器

5 docker rmi [imageID]: 删除镜像, 注意：删除镜像时，要先将该镜像创建的容器先删除

13 模板引擎的了解和使用?

1. 模板引擎都是为了实现业务逻辑层和表现层的代码分离, 将规定格式的模板代码转换为业务数据的算法实现
2. 简单地说，模板引擎的作用就是取得数据并加以处理，最后显示出数据
3. 模板引擎可以让（网站）程序实现界面与数据分离，业务代码与逻辑代码的分离，这就大大提升了开发效率，良好的设计也使得代码重用变得更加容易。
4. 模板引擎不只是可以让你实现代码分离（业务逻辑代码和用户界面代码），
    - 也可以实现数据分离（动态数据与静态数据），
    - 还可以实现代码单元共享（代码重用），
    - 甚至是多语言、动态页面与静态页面自动均衡（SDE）等等与用户界面可能没有关系的功能
5. 模板就是把公共的部分抽取出来, 减少代码的书写 常用的模板引擎
6. jinja(python)
7. freemarker(java)
8. velocity(java)

14 项目中的定时任务怎么写?

1 Quartz

2 SpringTask定时任务框架

1. 介绍: Springtask内置在spring中, 由spring提供一种定时任务操作
2. spring配置文件配置 <context:component-scan base-package="com.itheima.task"/> <task:annotation-driven/>
3. 自定义一个java类, 定时任务类
    - 类上添加 Component 注解
    - 定义方法(具体的定时任务), 方法名随意
        - 方法使用注解 @Scheduled(cron="* * * * * ?") 2 @Scheduled用来标识定时任务调度执行的操作 1 注解参数为 corn 表达式, 用来指定该任务多长时间执行一次
    - 使用了两个注解
        - Component: 类上
        - Scheduled: 定时任务方法上
4. 以上为在spirng中的使用方式, 如果在springboot中使用, 需要如下方式
    - 引导类上添加 @EnableScheduling 注解: 可以开启相关的定时任务功能
    - 只需要定义一个 Spring Bean , 一个java类, 类上使用 @Component 注解
        - 类中添加方法(定时任务逻辑方法), 并使用 @Scheduled 注解标记该方法即可
        - @Scheduled 注解中一定要声明定时任务的执行策略 cron 、fixedDelay、fixedRate 三选一
        - @Scheduled 提供了四个属性 1 cron 表达式 2 fixedDelay: 它的间隔时间是根据上次的任务结束的时候开始计时的 3 fixedRate: 4 initialDelay 1 initialDelay 初始化延迟时间，也就是第一次延迟执行的时间。这个参数对 cron 属性无效，只能配合 fixedDelay 或 fixedRate 使用。 2 如

@Scheduled(initialDelay=5000,fixedDelay = 1000) 表示第一次延迟 5000 毫秒执行，下一次任务在上一次任务结束后 1000 毫秒后执行。

5. Spring 的定时任务默认是单线程执行

15 如何实现邮箱和短信的发送?

1 邮箱发送

```
1. 首先我们需要准备两个邮箱账号A和B，A作为发送邮箱，B作为测试接收邮箱
2. 进入A邮箱设置，将其开启POP3/SMTP服务，以允许我们通过第三方客户端发送邮件。并且获取授权码（不是登录密码）
3. 使用JavaMail发送邮件非常简单
  *  创建连接对象javax.mail.Session
String from = "xxx@qq.com";// 发件人电子邮箱
String host = "smtp.qq.com"; // 指定发送邮件的主机 smtp.qq.com(QQ)|smtp.163.com(网易)
Properties properties = System.getProperties();// 获取系统属性
properties.setProperty("mail.smtp.host", host);// 设置邮件服务器
properties.setProperty("mail.smtp.auth", "true");// 打开认证
// QQ邮箱需要下面这段代码，163邮箱不需要
MailSSLSocketFactory sf = new MailSSLSocketFactory();
sf.setTrustAllHosts(true);
properties.put("mail.smtp.ssl.enable", "true");
properties.put("mail.smtp.ssl.socketFactory", sf);
*  创建邮件对象 javax.mail.Message
  // 1.获取默认session对象
  Session session = Session.getDefaultInstance(properties, new Authenticator() {
    public PasswordAuthentication getPasswordAuthentication() {
```

```
        return new PasswordAuthentication("xxx@qq.com", "xxx"); // 发件人邮箱账号、授权码
    }
  });
// 2.创建邮件对象
Message message = new MimeMessage(session);
// 2.1设置发件人
message.setFrom(new InternetAddress(from));
// 2.2设置接收人
message.addRecipient(Message.RecipientType.TO, new InternetAddress(email));
// 2.3设置邮件主题
message.setSubject("账号激活");
// 2.4设置邮件内容
String content = "<html><head></head><body><h1>这是一封激活邮件,激活请点击以下链接</h1><h3>
  <a href='http://localhost:8080/RegisterDemo/ActiveServlet?code="
  + code + "'>http://localhost:8080/RegisterDemo/ActiveServlet?code=" + code
  + "</href></h3></body></html>";
message.setContent(content, "text/html;charset=UTF-8");
*  发送邮件
  Transport.send(message);
```

2 短信发送

```
1. 使用阿里云的短信服务
  *  设置短信签名
  *  设置短信模板
  *  设置access keys: 发送短信时需要进行身份认证
2. maven坐标
  *  com.aliyun aliyun-java-sdk-core 3.3.1
  *  com.aliyun aliyun-java-sdk-dysmsapi 1.0.0
```

```
3. 代码工具类
      public static final String VALIDATE_CODE =
"SMS_173344488";//发送短信验证码
      public static final String ORDER_NOTICE =
"SMS_173344486";//体检预约成功通知
      public static void sendShortMessage(String
templateCode,String phoneNumbers,String param)
throws ClientException{
        // 设置超时时间-可自行调整

System.setProperty("sun.net.client.defaultConnectTi
meout", "10000");

System.setProperty("sun.net.client.defaultReadTimeo
ut", "10000");
        // 初始化ascClient需要的几个参数
        final String product = "Dysmsapi";// 短信API
产品名称（短信产品名固定，无需修改）
        final String domain =
"dysmsapi.aliyuncs.com";// 短信API产品域名（接口地址固
定，无需修改）
        // 替换成你的AK
        //AccessKeyID:LTAI4FsezDbizHVvxXzwCefC

//AccessKeySecret:ZHhzRN3e6DaxXOotmgnLVJ9flb69UH
        final String accessKeyId =
"LTAI4FsezDbizHVvxXzwCefC";// 你的accessKeyId,参考本
文档步骤2
        final String accessKeySecret =
"ZHhzRN3e6DaxXOotmgnLVJ9flb69UH";// 你的
accessKeySecret，参考本文档步骤2
        // 初始化ascClient,暂时不支持多region（请勿修
改）
        IClientProfile profile =
DefaultProfile.getProfile("cn-hangzhou",
accessKeyId, accessKeySecret);
        DefaultProfile.addEndpoint("cn-hangzhou",
"cn-hangzhou", product, domain);
```

```
        IAcsClient acsClient = new DefaultAcsClient(profile);
        // 组装请求对象
        SendSmsRequest request = new SendSmsRequest();
        // 使用post提交
        request.setMethod(MethodType.POST);
        // 必填:待发送手机号。支持以逗号分隔的形式进行批量调用，批量上限为1000个手机号码,批量调用相对于单条调用及时性稍有延迟,验证码类型的短信推荐使用单条调用的方式
        request.setPhoneNumbers(phoneNumbers);
        // 必填:短信签名-可在短信控制台中找到
        request.setSignName("黑马");
        // 必填:短信模板-可在短信控制台中找到
        request.setTemplateCode(templateCode);
        // 可选:模板中的变量替换JSON串,如模板内容为"亲爱的${name},您的验证码为${code}"时,此处的值为
        // 友情提示:如果JSON中需要带换行符,请参照标准的JSON协议对换行符的要求,比如短信内容中包含\r\n的情况在JSON中需要表示成\\r\\n,否则会导致JSON在服务端解析失败
        request.setTemplateParam("{\"number\":\""+ param +"\"}");
        // 可选-上行短信扩展码(扩展码字段控制在7位或以下，无特殊需求用户请忽略此字段)
        // request.setSmsUpExtendCode("90997");
        // 可选:outId为提供给业务方扩展字段,最终在短信回执消息中将此值带回给调用者
        // request.setOutId("yourOutId");
        // 请求失败这里会抛ClientException异常
        SendSmsResponse sendSmsResponse = acsClient.getAcsResponse(request);
        if (sendSmsResponse.getCode() != null && sendSmsResponse.getCode().equals("OK")) {
          // 请求成功
          System.out.println("请求成功");
        }
      }
```

16 两个异构项目之间的实时数据互通你会怎么做?

两个项目之间的数据交换可以使用ESB企业服务总线

ESB 是传统中间件技术与XML、Web服务等技术相互结合的产物

17 nginx的了解和使用?

1 Nginx 是一款高性能的 http 服务器/反向代理服务器及电子邮件(IMAP/POP3)代理服务器

1. 官方测试nginx能够支支撑5万并发链接，并且cpu、内存等资源消耗却非常低，运行非常稳定。

2 Nginx是一个http服务可以独立提供http服务

1. c语言开发

3 tomcat和nginx的对比

1. 在低并发的情况下，用户可以直接访问tomcat服务器
2. nginx常用做静态内容服务和代理服务器（不是你FQ那个代理），直面外来请求转发给后面的应用服务（tomcat，django什么的）
3. tomcat更多用来做做一个应用容器，让java web app跑在里面的东西

4 Nginx的应用场景

1. http服务器。Nginx是一个http服务可以独立提供http服务。可以做网页静态服务器。
2. 虚拟主机。可以实现在一台服务器虚拟出多个网站。例如个人网站使用的虚拟主机。
3. 反向代理，负载均衡。当网站的访问量达到一定程度后，单台服务器不能满足用户的请求时，需要用多台服务器集群可以使用nginx做反向代理。并且多台服务器可以平均分担负载，不会因为某台服务器负载高宕机而某台服务器闲置的情况

- 可以解决客户端访问服务器的跨域问题

5 作为中间件

Nginx可以作为各个应用之间的中间件，他是一个开源、高性能、可靠的HTTP中间件、代理服务

18 maven如何解决jar包冲突?

1 Maven默认处理策略

1. 最短路径优先
2. 最先声明优先

2 移除依赖: 用于排除某项依赖的依赖jar包

1. 我们可以借助Maven Helper插件中的Dependency Analyzer分析冲突的jar包，然后在对应标红版本的jar包上面点击execlude，就可以将该jar包排除出去。
2. 手动排除: 或者手动在pom.xml中使用标签去排除冲突的jar包
3. mvn分析包冲突命令: mvn dependency:tree

3 版本锁定原则: 一般用在继承项目的父项目中

1. 在父项目的pom.xml中的dependencyManagement标签中声明依赖并定义好版本号
2. 这样子项目要使用某个依赖时, 直接使用父pom.xml中定义的公共依赖就可以

19 树形结构的表怎么设计的?

1 设计1：邻接表

1. 即给表添加一个字段, 保存其父节点的id值
2. 这样是最常见的设计，能正确的表达菜单的树状结构且没有冗余数据，但在跨层级查询需要递归处理。
3. 优点: 结构简单
4. 缺点: 不使用递归情况下无法查询某节点所有父级，所有子集

2 设计2：路径枚举

1. 在设计1基础上新增一个父部门id集字段，用来存储所有父集，多个以固定分隔符分隔，比如逗号
2. 优点
    - 方便查询所有的子集；
    - 可以因此通过比较字符串dept_parent_ids长度获取当前节点层级；
3. 缺点
    - 新增节点时需要将dept_parent_ids字段值处理好；
    - dept_parent_ids字段的长度很难确定，无论长度设为多大，都存在不能够无限扩展的情况；
    - 节点移动复杂，需要同时变更所有子集中的dept_parent_ids字段值；

3 设计3：闭包表

1. 设计
    - 闭包表是解决分级存储的一个简单而优雅的解决方案，这是一种通过空间换取时间的方式；
    - 需要额外创建了一张TreePaths表它记录了树中所有节点间的关系；
    - 包含两列，祖先列与后代列，即使这两个节点之间不是直接的父子关系；同时增加一行指向节点自己；
2. 优点
    - 非递归查询减少冗余的计算时间
    - 方便非递归查询任意节点所有的父集
    - 方便查询任意节点所有的子集
    - 可以实现无限层级
    - 支持移动节点
3. 缺点
    - 层级太多情况下移动树节点会带来关系表多条操作；
    - 需要单独一张表存储对应关系，在新增与编辑节点时操作相对复杂；

20 如何实现菜单及增删改查的功能权限管理?

实现菜单 可以定义一张树状结构的表, 根据不同的用户所拥有的权限, 可以从表中查询到不同的菜单 增删改查的权限管理 security, shiro框架

21 开发程序需要哪些文档?

1. 可行性分析报告 说明该软件开发项目的实现在技术上、经济上和社会因素上的可行性，评述为了合理地达到开发目标可供选择的各种可能实施方案，说明并论证所选定实施方案的理由。
2. 项目开发计划 为软件项目实施方案制订出具体计划，应该包括各部分工作的负责人员、开发的进度、开发经费的预算、所需的硬件及软件资源等。
3. 软件需求说明书（软件规格说明书） 对所开发软件的功能、性能、用户界面及运行环境等作出详细的说明。它是在用户与开发人员双方对软件需求取得共同理解并达成协议的条件下编写的，也是实施开发工作的基础。该说明书应给出数据逻辑和数据采集的各项要求，为生成和维护系统数据文件做好准备。
4. 概要设计说明书 该说明书是概要实际阶段的工作成果，它应说明功能分配、模块划分、程序的总体结构、输入输出以及接口设计、运行设计、数据结构设计和出错处理设计等，为详细设计提供基础。
5. 详细设计说明书 着重描述每一模块是怎样实现的，包括实现算法、逻辑流程等。
6. 用户操作手册 本手册详细描述软件的功能、性能和用户界面，使用户对如何使用该软件得到具体的了解,为操作人员提供该软件各种运行情况的有关知识，特别是操作方法的具体细节。
7. 测试计划 为做好集成测试和验收测试，需为如何组织测试制订实施计划。划应包括测试的容、进度、条件、人员、测试用例的选取原则、测试结果允许的偏差范围等。
8. 测试分析报告 测试工作完成以后，应提交测试计划执行情况的说明，对测试结果加以分析，并提出测试的结论意见。

9. 开发进度月报 该月报系软件人员按月向管理部门提交的项目进展情况报告，报告应包括进度计划与实际执行情况的比较、阶段成果、遇到的问题和解决的办法以及下个月的打算等。
10. 项目开发总结报告 软件项目开发完成以后，应与项目实施计划对照，总结实际执行的情况，如进度、成果、资源利用、成本和投入的人力，此外，还需对开发工作做出评价，总结出经验和教训。
11. 软件维护手册 主要包括软件系统说明、程序模块说明、操作环境、支持软件的说明、维护过程的说明，便于软件的维护。
12. 软件问题报告 指出软件问题的登记情况，如日期、发现人、状态、问题所属模块等，为软件修改提供准备文档。
13. 软件修改报告 软件产品投入运行以后，发现了需对其进行修正、更改等问题，应将存在的问题、修改的考虑以及修改的影响作出详细的描述，提交审批。

五 前端

1 ajax请求方式, 参数类型, 状态码?

```
  1 jquery ajax常用参数：
    $.ajax({
      url:"", // ajax 请求地址
      type:"GET",//请求方式 ’GET‘或’POST‘，默认为’GET‘。
      dataType:"json",//根据返回数据类型，可以有如下数据可选：xml html script json jsonp text
      data:{}, //发送到服务器的数据，可以直接传对象{code: 456}，如果是get请求会自动拼接到url后面，如：&code=456;
      //请求成功后的回调函数
      success: function(data){
      },
      //请求失败时调用此函数。有以下三个参数：XMLHttpRequest 对象、错误信息、（可选）捕获的异常对象。
      error: function (XMLHttpRequest, textStatus, errorThrown) {
        console.log(textStatus);// 打印错误信息
```

```
        }
    });
  2 jquery ajax发送一个get请求
    $.get("请求url","发送的数据对象","成功回调","返回数据类型");
    $.get/post( url , data , function(参数){} , dataType);
  3 post请求
  4 状态码
    200: 请求响应成功
    302: 需要进行重定向操作
    304: 浏览器缓存
    400: 表示服务器不理解客户端请求的语法
    404: 页面找不到，资源找不到
    405: 表示对应的servlet中没有重写doget或dopost等doxxx的方法，会出现405的信息
    500: 服务器代码错误
    501: 服务器不支持请求的函数
```

2 jquery常用选择器

```
  1 $('标签名')        标签选择器
  2 $('#id名')         id选择器
  3 $('.class名')      类选择器
  4 属性选择器;
    $('A[属性名]') 获得有属性名的A元素‘;
    $('A[属性名=值]') 获得属性名 等于 值的A元素;
    $('A[属性名!=值]') 获得属性名 不等于 值的A元素
  1 基本选择器
    1 标签(元素)选择器
      1 语法: let l = $("html标签名")
      2 获得所有匹配标签名称的元素
    2 id选择器
      1 语法: $("#id属性值")
      2 获得指定的id属性值匹配的元素
    3 类选择器
      1 语法: $(".class的属性值")
      2 获得与指定的class属性值匹配的元素
```

```
  2 层级关系选择器
    1 并集选择器
      1 语法: let l = $(选择器1, 选择器2, ...);
      2 不同选择器之间用逗号隔开
    2 后代选择器
      1 语法: $(父选择器 子选择器 ...);
      2 空格: 表示子孙后代, 不同选择器之间用空格隔开
      3 如果把空格改为用">", 则表示父元素的所有子元素, 孙子元素不要
      4 prev + next: 获取所有跟在prev标签后的第一个同辈的next标签
      5 prev ~ next: 获取prev后边所有与prev标签同辈的next标签
    3 属性选择器
      1 语法: let l = $("标签名称[属性='属性值']");
    4 复合选择器
      1 语法: let l = $("标签名称[属性='属性值'][属性='属性值'][...]...");
      2 可以写多个属性进行筛选标签
  3 过滤选择器
    1 获取列表标签第一个元素
      1 语法: $("li:first")
    2 获取偶数索引元素
      1 语法: $("li:even")
      2 even 代表偶数
      3 odd 代表奇数
    3 获取指定索引 2 的元素
      1 语法: $("li:eq(2)")
      2 eq 代表等于
    4 获取大于索引2的元素
      1 语法: $("li:gt(2)")
      2 gt代表大于
      3 lt 代表小于
    5 注意: 过滤选择器的过滤条件可以有多个
      1 例: $("li:gt(2):lt(5)")
      2 表示所有的 li 标签中的第3个到第5个标签
      3 不包含第2个, 包含第5个
```

3 数据交互

1 前后端是如何做数据交互的? 1 通过表单传递参数 2 通过ajax传递参数（有post和get写法） 2 前端如何传给后端json数据? 1 前端向后端传递json数据时，需要使用JSON.stringify()将json对象转化为json字符串 2 ajax中添加参数: contentType : "application/json", type: "POST"

4 前端和后台都是如何解决跨域问题的?

1 前端 1 被请求页面加上下面的代码，最好content填写域名； 1 2 利用反向代理实现跨域, 反向代理需要用到nginx 实现原理: 原理大体相同，但是处理的端不同，反向代理实在服务器端进行处理。 首先修改hosts文件，将域名指向开发者的电脑本身，把自己伪装成服务端， 再通过nginx对不同的请求进行转发，把静态资源指向开发者本地电脑的资源， 将接口指向实际的服务器 2 第二种方法：在请求控制器加上加上下面的代码； 1 header("Access-Control-Allow-Origin: *");

5 前端如何防止表单重复提交?

```
1  1 第一种：用flag标识，即在表单提交过一次后把标识设置为true，下次提交时，如果标识为true，则不提交
2     1 或者可以把flag设置为时间，进行时间判断，5分钟内或其他时间内只能提交一次
3  2 第二种：在onsubmit事件中设置，在第一次提交后使提交按钮失效或将提交按钮隐藏
4     <form action=”about:blank” method=”post” onsubmit
   =”getElementById(‘submitInput').disabled=true;retur
   n true;” target=”_blank”>
5       <input type=”submit” id=”submitInput”/>
6     </form>
7  3 使用Post/Redirect/Get
8     1 Post/Redirect/Get简称PRG，是一种可以防止表单数据重复提交的一种Web设计模式
9     2 即当用户提交成功之后，执行客户端重定向，跳转到提交成功页面
10 4 使用session设置令牌
```

1 产生页面时，服务器为每次产生的Form分配唯一的随机标识号，
并且在form的一个隐藏字段中设置这个标识号，
同时在当前用户的Session中保存这个标识号。
当提交表单时，服务器比较hidden和session中的标识号是否相同，
相同则继续，处理完后清空Session，否则服务器忽略请求
2 注意：恶意用户可利用这一性质，不断重复访问页面，
以致Session中保存的标识号不断增多，最终严重消耗服务器内存。
可以采用在Session中记录用户发帖的时间，
然后通过一个时间间隔来限制用户连续发帖的数量来解决这一问题

6 VUE的基础知识

1 vue的生命周期?
1 vue生命周期可以分为八个阶段
1 beforeCreate(创建前)：对应的钩子函数为beforeCreate
1 此阶段为实例初始化之后，此时的数据观察和事件机制都未形成，不能获得DOM节点。
2 created(创建后)：对应的钩子函数为created
1 在这个阶段vue实例已经创建，仍然不能获取DOM元素。
3 beforeMount(载入前)：对应的钩子函数是beforeMount
1 在这一阶段，我们虽然依然得不到具体的DOM元素，但vue挂载的根节点已经创建，下面vue对DOM的操作将围绕这个根元素继续进行；beforeMount这个阶段是过渡性的，一般一个项目只能用到一两次。
4 mounted(载入后)：对应的钩子函数是mounted
1 mounted是平时我们使用最多的函数了，一般我们的异步请求都写在这里。在这个阶段，数据和DOM都已被渲染出来
5 beforeUpdate(更新前)：对应的钩子函数是beforeUpdate

```
        1 在这一阶段，vue遵循数据驱动DOM的原则。beforeUpdate函数在数据更新后虽然没立即更新数据，但是DOM中的数据会改变，这是Vue双向数据绑定的作用。
      6 updated（更新后）：对应的钩子函数是updated
        1 在这一阶段DOM会和更改过的内容同步。
      7 beforeDestroy(销毁前)：对应的钩子函数是beforeDestroy
        1 在上一阶段Vue已经成功的通过数据驱动DOM更新，当我们不再需要vue操纵DOM时，就要销毁Vue,也就是清除vue实例与DOM的关联，调用destroy方法可以销毁当前组件。在销毁前，会触发beforeDestroy钩子函数。
      8 destroyed（销毁后）：对应的钩子函数是destroyed
        1 在销毁后，会触发destroyed钩子函数
  2 vue的指令
    1 显示数据(v-text和v-html)
    2 数据双向绑定(v-model)
    3 事件处理(v-on)
    4 循环遍历(v-for)
    5 判断语句(v-if和v-show)
    6 显示数据(v-bind)
  3 vue的双向绑定如何实现?
    1 数据双向绑定(v-model)配合vue中的data
  4 vue如何实现自定义事件?
    使用 $on(eventName, function(参数){事件处理代码})监听事件
    使用 $emit(eventName, [参数1], [参数2]) 触发事件
  5 vue如何实现监听?
    $watch
  6 vue如何实现请求?
    axios: ajax请求
  7 vue的路由跳转方式?
    1 router-link标签
      1. 不带参数
        <router-link :to="{name:'home'}">
        <router-link :to="{path:'/home'}">
//name,path都行，建议用name
        // 注意：router-link中链接如果是'/'开始就是从根路由开始，如果开始不带'/'，则从当前路由开始。
```

```
        2.带参数
          <router-link :to="{name:'home', params:
{id:1}}">
    2 this.$router.push
        跳转到指定url路径，并想history栈中添加一个记录，点
击后退会返回到上一个页面
        1.  不带参数
          this.$router.push('/home')
          this.$router.push({name:'home'})
          this.$router.push({path:'/home'})
        2. query传参
          this.$router.push({name:'home',query:
{id:'1'}})
          this.$router.push({path:'/home',query:
{id:'1'}})
        3. params传参
          this.$router.push({name:'home',params:
{id:'1'}})  // 只能用 name
    3 this.$router.replace
        跳转到指定url路径，但是history栈中不会有记录，点击
返回会跳转到上上个页面 (就是直接替换了当前页面)
    4 this.$router.go(n)
        向前或者向后跳转n个页面，n可为正整数或负整数
```

7 get和post有什么区别?

```
  1 Get是不安全的，因为在传输过程，数据被放在请求的URL中；
Post的所有操作对用户来说都是不可见的。
  2 Get传送的数据量较小，这主要是因为受URL长度限制；Post传
送的数据量较大，一般被默认为不受限制。
  3 Get限制Form表单的数据集的值必须为ASCII字符；而Post支
持整个ISO10646字符集。
  4 Get执行效率却比Post方法好。Get是form提交的默认方法。
```

8 jquery如何获取当前日期?

```
  var myDate = new Date();
  var year=myDate.getFullYear();        //获取当前年
  var month=myDate.getMonth()+1;   //获取当前月
  var date=myDate.getDate();            //获取当前日
  var h=myDate.getHours();              //获取当前小时数(0-23)
  var m=myDate.getMinutes();            //获取当前分钟数(0-59)
  var s=myDate.getSeconds();          //获取当前秒
  var now=year+'-'+getNow(month)+"-"+getNow(date)+" "+getNow(h)+':'+getNow(m)+":"+getNow(s);
```

9 如何给画面中所有的checkbox标签添加一个事件?

```
  var x = document.getElementByTagName("checkbox").
  var i;
  for (i = 0; i < x.length; i++) {
    x[i].onclick
  }
```

10 jquery怎么打开一个模态窗口?

```
  SimpleModal是一个轻量级的jQuery插件，它提供了一个强大的界面模态对话框发展。
  SimpleModal提供2个简单的方法来调用一个模式对话框。
  （1）作为一个串连的jQuery函数，你可以调用一个jQuery元素modal()函数使用该元素的内容将显示一个模式对话框。
    $("#element-id").modal();
  (2)作为一个独立的功能，可以通过一个jQuery对象，一个DOM元素，或者一个普通的字符串（可以包含HTML）创建一个模态对话框。
    $("#element-id").modal({options});
    $.modal("<div><h1>SimpleModal</h1></div>", {options});
```

11 表单验证是怎么实现的?

```
1  用document获取输入框的value可以判断是否为空或者是否为数字
2  其他类型的校验 比如手机号 邮箱之类的可以用正则表达式校验
3  先获取表单中输入框的值，然后可以用正则表达式对值进行验证
```

12 readonly和disable有什么区别?

```
1  readonly并没有对input[type="button"]产生作用，按钮效果仍然在，并没有“不可用”;
2  而disabled直接对input[type="button"]的按钮效果产生作用，导致按钮不可点击。
3  Disable设置为true之后是不可以向后台提交数据的，此时可以选择改用readonly进行禁用，或者在提交数据时取消禁用
4  Disable比readonly的使用范围比广，适用文本框、文本域、下拉框、button按钮、单选框…….而readonly只适用于input（text、passwork、textarea）
5     disabled属性可以作用于所有的表单元素，readonly只对可以输入的表单元素有效
6  表面上可看到的区别就是当这两个词都设置为true时，都为禁用状态
7     当鼠标移上时使用disable的相关控件时鼠标出现禁用样式，并且不可做任何操作
8     而Readonly还可以获取文本框里的焦点
9  readonly不可编辑，但可以选择和复制；值可以传递到后台
10    主要用于表单输入框
11 disabled不能编辑，不能复制，不能选择；值不可以传递到后台
12    可以用于所有的表单元素
```

13 如何画一个表格?如何合并单元格?

```
1  在<table></table>的标签内只要有几行就要写几个<tr></tr>，而<tr></tr>内又包含着<td></td>
2  合并单元格，在td标签中添加属性
3    colspan:跨列合并单元格，横向合并
4    rowspan:跨行合并单元格，纵向合并
```

14 如何获取一个文本框的内容?

```
  var textbox= document.getElementById("textbox");
  var text = textbox.value;
```

15 jquery如何获取form表单的值?

```
  把form表单的值序列化成一个Json对象，如{username:admin，password:123}
  var params = $('#searchForm').serializeObject();
  把form表单的值序列化成一个字符串，如username=admin&password=admin
  var params = $('#searchForm').serialize();
  将字符串转换为Json对象
  JSON.parse(str)
```

16 前端如何延时调用一个函数?

```
  jQuery.delay(speed,queueName)方法
    speed: 可选。规定延迟的速度
      可能的值：毫秒，"slow"，"fast"
      延时时间（duration参数）是以毫秒为单位的，数值越大，动画越慢，不是越快
      字符串 'fast' 和 'slow' 分别代表200和600毫秒的延时
    queueName: 可选。规定队列的名称
      默认是 "fx"，标准效果队列
  一次性定时器: var id = setTimeout(调用方法,毫秒值)
    1 调用方法: 如: "fun()"
  循环定时器: var iid = setInterval(调用方法,毫秒值)
```

17 如何取出树形结构的数据?

如何取出树形结构的数据
  首先先遍历所有的节点数据，生成id 和parent_id的关系，然后遍历id 和parent_id的关系，因此将子节点数据放入children 这个集合当中。
  前端如何显示树形结构数据
  通过js遍历数据对象，拼接成dom字符串，插入到html中

18 js中的this在各种情况下的指向?

  作为对象的方法调用
    当函数作为对象的方法被调用时，this指向该对象
  作为普通函数调用
    当函数不作为对象的属性被调用时，也就是作为普通函数调用，此时的this总是指向全局对象。在浏览器的JavaScript环境中，这个全局对象时window对象
  构造器调用
    当用 new 运算符调用函数时，该函数总 会返回一个对象，通常情况下，构造器里的 this 就指向返回的这个对象

19 如何将数据保存到前端?

  1 虽然cookie可以存储一些数据，但是仍然存储下面一些缺点
    （1）cookie需要在客户端和服务器端之间来回传送，会浪费不必要的资源
    （2）cookie的存储大小有限制，对于每个域，一般只能设置20个cookie，每个cookie大小不能超过4KB
    （3）cookie的安全性，cookie因为保存在客户端中，其中包含的任何数据都可以被他人访问，cookie安全性比较低
  2 web存储机制
    接下来，我们要说一下html5中的存储啦，主要是sessionStorage和localStrorage
      localStorage 和 sessionStorage 属性允许在浏览器中存储 key/value 对的数据
    由于sessionStorage对象是Storage的一个实例，所以存储数据时可以使用setItem()或者直接设置新的属性来存储数据
      当我们要获取某个数据的时候，可以使用getItem来获取数据

```
10          如果你只想将数据保存在当前会话下，可以使用
    sessionStorage，数据将会临时保存，关闭窗口后则会被删除
11          localStorage也是Storage的实例，可以像使用
    sessionStorage一样来使用它
12          localStorage用于长久的保存整个网站的数据（string
    类型存储），保存的数据没有过期时间，直到手动删除，
13          并且localStorage的属性是只读的（不过如果是在浏览器
    的隐私模式下，它是不可读取的）
14          保存数据语法：localStorage.setItem("key",
    "value");
15          读取数据语法：var lastname =
    localStorage.getItem("key");
16          删除数据语法：localStorage.removeItem("key");
```

20 如何实时刷新页面?

```
1   1 页面自动刷新：把如下代码加入<head>区域中
2     <meta http-equiv="refresh" content="5">
3       content="20;url=url地址": 如果没有写url，则默认跳
    转到当前页面
4   2 reload
5     reload 方法，该方法强迫浏览器刷新当前页面。
6     语法：location.reload([bForceGet])
7     参数：bForceGet，可选参数，默认为 false，从客户端缓存
    里取当前页。true，则以 GET 方式，从服务端取最新的页面，相
    当于客户端点击 F5("刷新")
```

21 http协议

什么是http协议无状态协议? 无状态是指协议对于事务处理没有记忆功能。缺少状态意味着，假如后面的处理需要前面的信息，则前面的信息必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要前面信息时，应答就较快 直观地说，就是每个请求都是独立的，与前面的请求和后面的请求都是没有直接联系的是相互隔离的，请求本身包含了相应端为相应这一请求所需的全部信息。 1、协议对于事务处理没有记忆能力【事物处理】【记忆能力】 2、对同一个url请求没有上下文关系【上下文关系】 3、每次的请求都是独立的，它的执行

情况和结果与前面的请求和之后的请求是无直接关系的，它不会受前面的请求应答情况直接影响，也不会直接影响后面的请求应答情况【无直接联系】【受直接影响】 4、服务器中没有保存客户端的状态，客户端必须每次带上自己的状态去请求服务器【状态】 怎么解决http协议无状态协议? 1、通过Cookies保存状态信息 通过Cookies，服务器就可以清楚的知道请求2和请求1来自同一个客户端。 2 通过Session保存状态信息 Session的实现方式： 1、使用Cookie来实现 服务器给每个Session分配一个唯一的JSESSIONID，并通过Cookie发送给客户端。 当客户端发起新的请求的时候，将在Cookie头中携带这个JSESSIONID。这样服务器能够找到这个客户端对应的Session。 2、使用URL回写来实现 URL回写是指服务器在发送给浏览器页面的所有链接中都携带JSESSIONID的参数，这样客户端点击任何一个链接都会把JSESSIONID带会服务器。如果直接在浏览器输入服务端资源的url来请求该资源，那么Session是匹配不到的。 Tomcat 对Session的实现，是一开始同时使用Cookie和URL回写机制，如果发现客户端支持Cookie，就继续使用Cookie，停止使用URL回 写。如果发现Cookie被禁用，就一直使用URL回写。jsp开发处理到Session的时候，对页面中的链接记得使用response.encodeURL() 。 3、通过表单变量保持状态 除了Cookies之外，还可以使用表单变量来保持状态，比如Asp.net就通过一个叫ViewState的Input="hidden"的框来保持状态,比如: 这个原理和Cookies大同小异，只是每次请求和响应所附带的信息变成了表单变量。 4、通过QueryString保持状态 QueryString通过将信息保存在所请求地址的末尾来向服务器传送信息，通常和表单结合使用，一个典型的QueryString比如:www.xxx.com/xxx.aspx?var1=value&var2=value2

六 其他

二叉树 红黑树 hashtable 加锁

1. Hashtable是线程安全的，它的每个方法中都加入了Synchronize方法。

2. HashMap不是线程安全的，在多线程并发的环境下，可能会产生死锁等问题
   - 虽然HashMap不是线程安全的，但是它的效率会比Hashtable要好很多

    - 2 HashMap大部分时间是单线程操作的
    - 3 多线程操作的时候可以使用线程安全的ConcurrentHashMap
    - 4 ConcurrentHashMap虽然也是线程安全的，但是它的效率比Hashtable要高好多倍
    - 5 ConcurrentHashMap使用了分段锁，并不对整个数据进行锁定
    3. 遍历方式

- Hashtable、HashMap都使用了 Iterator。而由于历史原因，Hashtable还使用了Enumeration的方式
- JDK8之前的版本中，Hashtable是没有fast-fail机制的。在JDK8及以后的版本中 ，HashTable也是使用fast-fail的
- HashMap的Iterator是fail-fast迭代器

    4. Hashtable默认的初始大小为11，之后每次扩充，容量变为原来的2n+1
    5. HashMap默认的初始化大小为16。之后每次扩充，容量变为原来的2倍。

集合遍历 远程oracle服务器 索引失效 mysql悲观锁 树形结构的数据, 集合套集合套集合 吞吐量比较大的时候使用kafka, 自带缓存 枚举类似常量类 数据库拼接字符串 1 oracle: concat() 2 mysql: concat()数){}