全网最火Java面试题

原创 IT老哥 IT老哥 5月6日

第一部分 JAVA 基础

第一节 IO/NIO

第二节 反射

第三节 多线程

第四节 集合

第五节 Web

第六节 其他

第七节 关键字

第八节 操作符

第九节 基础类型

第十节 异常

第十一节 JDBC

第十二节 OOP

第二部分 JVM调优及原理

第一节 JVM原理

第二节 JVM调优

第三节 JVM GC原理

第三部分 数据库

第一节 SQL编写

第二节 事物

第三节 锁

第四节 索引分区

第四部分 Ridis/ZK

第五部分 SSM使用及原理

第一节 spring 原理

第二节 Spring 事务

第三节 IOC/AOP

第四节 Spring MVC原理

第五节 Mybatic

第六部分 Linux 常用及Shell编程

第七部分 设计模式及算法

第八部分 项目介绍

第九部分 网络

- **JAVA 基础**

兄弟：
面试过了，记得请老哥吃饭，哈哈
祝你成功！
说下面试流程：

电话面：聊基础
笔试：基础
一轮技术：基础+
二轮技术：基础++
Boss:唠嗑
人力：为啥离职，期望薪资、唠嗑
过了人力面就等offer吧，基本稳了。
这是最全流程，基本都在这个范围了。

简历中把这些内容加上，这样被问题到下面题目的概率比较大。
1.JAVA基础扎实、熟练运用设计模式、理解IO/NIO、反射、多线程编程、了解JVM原理;
2.熟悉Oracle、MySQL、了解非关系型数据库redis;
3.熟悉Spring、Spring MVC、MyBatis框架。

- **IO/NIO**

1. **IO 和 NIO的区别，NIO优点。**

答：

IO	NIO
面向流	面向缓存
阻塞IO	非阻塞IO
无	选择器

面向流与面向缓冲：
面向流意味着每次从流中读取一个或多个字节。直至读取所有字节，它们没有被缓存在任何地方。不能够前后移动流中的数据，如果需要移动，则需要先缓存在一个缓冲区。NIO数据读取到一个它稍后处理的缓冲区，需要时可在缓冲区前后移动，增加了处理过程的灵活性，需要检查数据完整性。
阻塞与非阻塞IO：
IO的各种流是阻塞的，意味着当一个线程调用read()或write()时，该线程被阻塞，直到有一些数据被读取，或数据完全写入，该线程在此期间不能再干任何事。NIO为非阻塞，线程从某通道发送请求读取数据，但是它仅能得到目前可用的数据，如果目前没有数据可用，就什么都不获取，不保持线程阻塞，所以直至数据可以读取之前，线程可以继续做其他事情，非阻塞写也是相同，不必等到完全写入，线程通常将非阻塞IO的空闲时间用在其他通道上执行操作，所以一个单独的线程，可以管理多个输入和输出通道。
选择器：
Java NIO的选择器允许一个单独的线程来监视多个输入通道，你可以注册多个通道使用一个选择器，然后使用一个单独的线程来“选择”通道：这些通道里已经有可以处理的输

入，或者选择已准备写入的通道。这种选择机制，使得一个单独的线程很容易来管理多个通道。
注：JAVA中有两个包：
package java.io;//下面的类都是面向流的。
package java.nio;//都是面向缓冲的，类名多包涵Buffer

Socket 工作在 TCP/IP 协议栈是哪一层?

基础：

TCP/IP	OSI
应用层	应用层 表示层 会话层
传输层	传输层
网络层（又称互联层）	网络层
网络接口层（又称链路层）	数据链路 物理层

答：传输层

2.TCP、UDP 区别及 Java 实现方式?

TCP是面向连接（可靠传输），UDP面向非连接（不可靠）：
TCP建立连接时需要传说的三次握手，服务端与客户端需要确认对方身份而已，建立好连接后，就开始传递消息，直到有一方断开连接位置。就好比两个人打电话，要先通了才能说话。
UDP只是数据报发送，它的优点速度快，并非要向TCP那样麻烦建立，它只负责将信息发出，但是并不确保信息的准确完整性等，就好比发短信，短信是出去了，但是中间是否有问题，是否对方手机能收到就不管了。
在java中想要实现上述两种协议通信，可采用socket建立连接，socket可以理解为码头，其实是套接字，这里简单说下，就好比两个城市运输货物，通过码头走货一样。

3.说几点 IO 的最佳实践?

a）使用有缓冲区的 IO 类，而不要单独读取字节或字符。
b）使用 NIO 和 NIO2
c）在 finally 块中关闭流，或者使用 try-with-resource 语句。
d）使用内存映射文件获取更快的 IO。

4.IO、AIO、NIO是什么?

5.如何创建缓冲区，缓冲区的内存回收问题如何解决？

这块时间充足的话多写写代码帮助理解，时间紧的话可以只了解概念。

- **反射**

1. java反射机制了解么？
2. 什么是 Java 的反射？说说你了解的java反射。
3. 在没有get和set方法的前提下,修改类中的 private a;这个属性的值。

提示：通过java反射实现，一定要手写代码去了解。仔细看下Class对象中的方法如何使用。

这块建议好好看看Jvm类加载的内容，对于java反射会有很深入的理解。

1. 在你的项目中是否使用到java反射，应用场景是什么。

- **多线程**

1. 线程池构造器中的参数你知道哪些，都有什么用？
2. 线程池有几种创建方式，分别有什么特点。
3. 你常用的线程池创建方式是什么？用cache的方式会有什么问题。
4. 如果线程池中的线程数已经超过最大线程数，会发生什么。
5. 为什么要使用线程池？
6. Synchronized用在类上、方法上、静态方法上有什么区别。

synchronized是Java中的关键字，是一种同步锁。它修饰的对象有以下几种：

1. 修饰一个代码块，被修饰的代码块称为同步语句块，其作用的范围是大括号{}括起来的代码，作用的对象是调用这个代码块的对象；
2. 修饰一个方法，被修饰的方法称为同步方法，其作用的范围是整个方法，作用的对象是调用这个方法的对象；
3. 修改一个静态的方法，其作用的范围是整个静态方法，作用的对象是这个类的所有对象；
4. 修改一个类，其作用的范围是synchronized后面括号括起来的部分，作用主的对象是这个类的所有对象。

1. 一个线程访问一个对象的Synchronized方法，其他线程能否同时访问这个类的非Synchronized方法。
2. 一线程访问一个对象的同步代码块，其他线程能否同时访问这个类的Synchronized方法？其他Synchronized同步代码块呢？非Synchronized方法呢？

当两个并发线程访问同一个对象object中的这个synchronized(this)同步代码块时，一个时间内只能有一个线程得到执行。另一个线程必须等待当前线程执行完这个代码块以后才能执行该代码块。

当一个线程访问object的一个synchronized(this)同步代码块时，其他线程对object中所有其它synchronized(this)同步代码块的访问将被阻塞。

然而，当一个线程访问object的一个synchronized(this)同步代码块时，另一个线程仍然可以访问该object中的除synchronized(this)同步代码块以外的部分。
第三个例子同样适用其它同步代码块。也就是说，当一个线程访问object的一个synchronized(this)同步代码块时，它就获得了这个object的对象锁。结果，其它线程对该object对象所有同步代码部分的访问都被暂时阻塞。
以上规则对其它对象锁同样适用。

1. **Synchronized是如何实现同步机制的?**

执行同步代码块后首先要先执行monitorenter指令，退出的时候monitorexit指令。通过分析之后可以看出，使用Synchronized进行同步，其关键就是必须要对对象的监视器monitor进行获取，当线程获取monitor后才能继续往下执行，否则就只能等待。而这个获取的过程是互斥的，即同一时刻只有一个线程能够获取到monitor。上面的demo中在执行完同步代码块之后紧接着再会去执行一个静态同步方法，而这个方法锁的对象依然就这个类对象，那么这个正在执行的线程还需要获取该锁吗？答案是不必的，从上图中就可以看出来，执行静态同步方法的时候就只有一条monitorexit指令，并没有monitorenter获取锁的指令。这就是锁的重入性，即在同一锁程中，线程不需要再次获取同一把锁。Synchronized先天具有重入性。每个对象拥有一个计数器，当线程获取该对象锁后，计数器就会加一，释放锁后就会将计数器减一。

1. Synchronized锁的是对象还是代码?
   对象
1. 除了Synchronized，java还有什么别的方法保证线程安全?
2. Volatile了解吗？在什么情况下使用，是否能够保证线程安全。
3. Volatile是如何保证可见性的。

Volatile实现内存可见性是通过store和load指令完成的；也就是对volatile变量执行写操作时，会在写操作后加入一条store指令，即强迫线程将最新的值刷新到主内存中；而在读操作时，会加入一条load指令，即强迫从主内存中读入变量的值。但volatile不保证volatile变量的原子性

1. 使用100次for循环做i++操作，同时有5个线程调用该循环，最终结果是什么情况？为什么?

一定要写代码测试

1. Int是线程安全的吗？java提供什么解决方案?
2. automicInterger 的实现原理是什么?
3. 什么是cas?
4. 用java模拟10匹马赛跑，你如何设计?

考点是如何让10个线程在某一时刻同时执行。

7.多线程的优点?

答:

优点

1)．耗时的操作使用线程，提高应用程序响应

2)．并行操作时使用线程，如C/S架构的服务器端并发线程响应用户的请求。

3)．多CPU系统中，使用线程提高CPU利用率

4)．改善程序结构。一个既长又复杂的进程可以考虑分为多个线程，成为几个独立或半独立的运行部分，这样的程序会利于理解和修改。

5). 提高应用程序响应。这对图形界面的程序尤其有意义，当一个操作耗时很长时，整个系统都会等待这个操作，此时程序不会响应键盘、鼠标、菜单的操作，而使用多线程技术，将耗时长的操作（time consuming）置于一个新的线程，可以避免这种尴尬的情况。

6). 使多CPU系统更加有效。操作系统会保证当线程数不大于CPU数目时，不同的线程运行于不同的CPU上。

7). 改善程序结构。一个既长又复杂的进程可以考虑分为多个线程，成为几个独立或半独立的运行部分，这样的程序会利于理解和修改。

缺点

1).等候使用共享资源时造成程序的运行速度变慢。这些共享资源主要是独占性的资源 ,如打印机等。

2).对线程进行管理要求额外的 CPU开销。线程的使用会给系统带来上下文切换的额外负担。当这种负担超过一定程度时,多线程的特点主要表现在其缺点上,比如用独立的线程来更新数组内每个元素。

3).线程的死锁。即较长时间的等待或资源竞争以及死锁等多线程症状。

4).对公有变量的同时读或写。当多个线程需要对公有变量进行写操作时,后一个线程往往会修改掉前一个线程存放的数据,从而使前一个线程的参数被修改;另外 ,当公用变量的读写操作是非原子性时,在不同的机器上,中断时间的不确定性,会导致数据在一个线程内的操作产生错误,从而产生莫名其妙的错误,而这种错误是程序员无法预知的。

8.多线程的几种实现方

1).继承Thread类创建线程

2).实现Runnable接口创建线程

3).实现Callable接口通过FutureTask包装器来创建Thread线程

4).使用ExecutorService、Callable、Future实现有返回结果的线程

9.用 Runnable 还是 Thread？为什么?

Runnable和Thread都可以实现run方法，但一个是接口，一个是类，前者可以无限地创建Thread进行run，而后者进行一次run之后就无法再次run。注意：Thread执行了start之后不可以再次执行start！

因此，要实现线程能重复运行，如果采用XXX extends Thread，那么每次运行都必须new一个XXX，这十分损耗资源；如果使用XXX implements Runnable，那每次运行只

需要新开一个new Thread(xxx)即可，节省了很多时空消耗
Thread start()方法被用来启动新创建的线程，而且start()内部调用了run()方法，这和直接调用run()方法的效果不一样。当你调用run()方法的时候，只会是在原来的线程中调用，没有新的线程启动，start()方法才会启动新线程。
处理线程不安全的方法有哪些?

1. .定义为局部变量使用。
2. .加一把线程同步锁：synchronized(lock)
3. .使用ThreadLocal

10.哪些集合类是线程安全的?
线程安全:
•Vector
•HashTable
•StringBuffer
线程不安全:
•ArrayList ：
•LinkedList:
•HashMap:
•HashSet:
•TreeMap:
•TreeSet:
•StringBulider

11.什么是线程局部变量?
ThreadLocal
为每一个线程提供一个变量的副本。每个线程可以单独访问自己的副本，而不与其他线程共享，当前线程结束后GC。
实现原理：ThreadLocal类中有一个map，key为线程对象，value为对应的副本值。

12.ConcurrentHashMap 的线程安全是如何实现的？这样实现有什么优点?

13.CountDownLatch的用法。

构造方法:

```
/**
*       count 锁存器容量
*        理解为要记几个数
*
```

```
*/
   public CountDownLatch(int count) {
         if (count < 0) throw new IllegalArgumentException("count < 0");
         this.sync = new Sync(count);
}
等待方法：
/**
*        该方法未返回前执行for (;;) {} 循环体按照以下逻辑检测
*      timeout 需要等待多久
*       unit timeout时间单位
*       如果超过时间未等到锁存器值为0则返回false.如果等待线程被中断，则抛出异常
*       if (Thread.interrupted())
*           throw new InterruptedException();
*/
public boolean await(long timeout, TimeUnit unit)
public boolean await()//同上，只是没有时间限制。
/**
*        该方法对锁存器减1.
*/
    public void countDown() {
         sync.releaseShared(1);
}
```

14.ConcurrentHashMap的并发度是什么?

15.CyclicBarrier 和 CountDownLatch有什么不同？各自的内部原理和用法是什么?

不同：

1.CyclicBarrier的某个线程运行到屏障点上之后，该线程立即停止运行，直到所有的线程都到达了这个屏障点，所有线程才依次按顺序被唤醒重新运行；CountDownLatch是某个线程运行到某个点上之后，只是给计数器数值减一，该线程扔继续运行；

2.CyclicBarrier唤醒等待线程虽然是唤醒全部，但等待线程是按顺序依次执行的；CountDownLatch是唤醒多个任务，抢占式执行；

3.CyclicBarrier可重用的，因为内部计数器可重置；CountDownLatch不可重用，计数器值为0该CountDownLatch就不可再用。

Semaphore的用法

思想：有许可就执行，也是使用计数的思想。

//初始化

final Semaphore semp = new Semaphore(5); //给出5许可数量

```
//如果获取到许可，则方法返回线程继续执行，否则阻塞，方法内部有for(;;)。
semp .sacquire();
semp .sacquire(int a );//需要a个许可。
//释放一个许可
semp.release();
  //释放a个许可。
semp.release(int a);
```

16.启动一个线程是调用 run() 还是 start() 方法？start() 和 run() 方法有什么区别

启动一个线程调用：start();

区别：

start方法创建一个线程，并启动线程，新线程处于就绪状态。该方法由父线程执行。

run 方法执行具体的业务内容，由start方法创建出的新线程执行，run方法返回，该线程结束。

直接调用run方法和调用普通方法一样，还是在同一个线程中执行，需要等到run方法返回，才可以继续执行。

17.sleep() 方法和对象的 wait() 方法都可以让线程暂停执行，它们有什么区别?

sleep()方法（休眠）是线程类（Thread）的静态方法，调用此方法会让当前线程暂停执行指定的时间，将执行机会（CPU）让给其他线程，但是对象的锁依然保持，因此休眠时间结束后会自动恢复。

wait()是Object类的方法，调用对象的wait()方法导致当前线程放弃对象的锁（线程暂停执行），进入对象的等待池（wait pool），只有调用对象的notify()方法（或notifyAll()方法）时才能唤醒等待池中的线程进入等锁池（lockpool），如果线程重新获得对象的锁就可以进入就绪状态。

18.yield方法有什么作用？sleep() 方法和 yield() 方法有什么区别?

1).sleep()方法暂停当前线程后，会给其他线程执行机会，不会理会其他线程的优先级；但yield()方法只会给优先级相同，或优先级更高的线程执行机会。

2).sleep()方法会将线程转入阻塞状态，直到经过阻塞时间才会转入就绪状态；而yield()方法不会将线程转入阻塞状态，它只是强制当前线程进入就绪状态。因此完全有可能某个线程调用yield()方法暂停之后，立即再次获得处理器资源被执行。

3).sleep()方法声明抛出了InterruptedException异常，所以调用sleep()方法时要么捕捉该异常，要么显式声明抛出该异常；而yield()方法则没有声明抛出任何异常。

4).sleep()方法比yield()方法有更好的可移植性，通常不建议使用yield()方法来控制并发线程的执行。

19.Java 中如何停止一个线程?

1).run方法结束，线程终止，所以可以使用一个boolean类型的标记，判断是否继续执行while循环，控制run方法的返回，达到终止线程的目的。该变量应加volatile 修饰。保证可见性。
2).interrupted 方法，将线程是否被终止的标示修改为true,如果此时调用sleep或wait方法将抛出InterruptedException异常，需要捕获异常并处理才可以终止线程，否则线程标识会被修改为false,而无法终止线程。线程非静态方法isInterrupted和静态方法interrupted可获取到线程是否被终止。
3).stop 方法 不推荐使用 会导致程序执行错误。活干一半不干了。

20.如何在两个线程间共享数据
练习：http://blog.csdn.net/hejingyuan6/article/details/47053409
1).如果每个线程执行的任务相同，可以使用同一个Runable对象，在该对象中定义共享的数据。
2).如果线程执行不同的任务，则有两种方式：
(2.1)将Runable作为内部类，run方法访问外部类中的共享数据。
(2.2)定义一个共享数据类型，包括对数据的操作。实现Runable接口，添加共享数据　　类型属性，并添加构造方法，初始化其中的共享数据类型。

21.如何让正在运行的线程暂停一段时间?
无法让一个线程准确的暂停一段时间，主要原因是线程调度的原因。
1).Thread 类的Sleep方法只能保证线程至少暂停的时间，当时间到达后，线程进入就绪状态，具体执行时间还受线程调度的影响。
2).Object 的wait方法，可以暂停当前执行线程，并释放对象的锁，线程进入对象的等待队列，当调用notify或notifyAll时，线程从对象的等待队列进入线程的锁队列，当线程获得对象的锁后，线程变为就绪状态，才有执行的可能。

22.有哪些不同的线程生命周期?
新建（new Thread）
当创建Thread类的一个实例（对象）时，此线程进入新建状态（未被启动）。
例如：Thread t1=new Thread();
就绪（runnable）
线程已经被启动，正在等待被分配给CPU时间片，也就是说此时线程正在就绪队列中排队等候得到CPU资源。例如：t1.start();
运行（running）
线程获得CPU资源正在执行任务（run()方法），此时除非此线程自动放弃CPU资源或者有优先级更高的线程进入，线程将一直运行到结束。
死亡（dead）

当线程执行完毕或被其它线程杀死，线程就进入死亡状态，这时线程不可能再进入就绪状态等待执行。
自然终止：正常运行run()方法后终止
异常终止：调用stop()方法让一个线程终止运行
堵塞（blocked）
由于某种原因导致正在运行的线程让出CPU并暂停自己的执行，即进入堵塞状态。
正在睡眠：用sleep(long t) 方法可使线程进入睡眠方式。一个睡眠着的线程在指定的时间过去可进入就绪状态。
正在等待：调用对象wait()方法。（调用notify()方法回到就绪状态）
被另一个线程所阻塞：调用suspend()方法。（调用resume()方法恢复）

23.线程状态，BLOCKED 和 WAITING 有什么区别?

24.画一个线程的生命周期状态图

25.如何创建一个Java线程池
Java通过Executors提供四种线程池，分别为：
ExecutorService typeThreadPool
=newCachedThreadPool创建一个可缓存线程池，如果线程池长度超过处理需要，可灵活回收空闲线程，若无可回收，则新建线程。
=newFixedThreadPool 创建一个定长线程池，可控制线程最大并发数，超出的线程会在队列中等待。
=newScheduledThreadPool 创建一个定长线程池，支持定时及周期性任务执行。
=newSingleThreadExecutor 创建一个单线程化的线程池，它只会用唯一的工作线程来执行任务，保证所有任务按照指定顺序(FIFO, LIFO, 优先级)执行。

1. **ThreadPoolExecutor用法。**

构造器：
ThreadPoolExecutor(corePoolSize,maximumPoolSize,keepAliveTime,milliseconds,runnableTaskQueue,handler)
方法：
execute(Runnable runnable)提交任务，但是execute方法没有返回值。
submit 方法来提交任务，它会返回一个future对象，用于查看线程运行状态，获取返回值。

26.提交任务时，线程池队列已满时会发会生什么?
submit()方法将会抛出一个RejectedExecutionException

27.什么是多线程中的上下文切换?

发生在cup执行线程期间，cpu执行下一个任务前会保存上一个任务的任务状态，以便下次切换回这个任务的时候可以加载任务的执行状态，继续执行。从任务保存，到再加载的过程，就是一次上下文切换。

就是说不要让你的程序依赖于线程的优先级）。

28.synchronized 和 ReentrantLock 有什么不同

练习：http://blog.csdn.net/eclipser1987/article/details/7301828

1).synchronized 一直等待，ReentrantLock 可以释放等待。

2).公平锁，多个线程等待同一个锁时，必须按照申请锁的时间顺序获得锁，Synchronized锁非公平锁，ReentrantLock默认的构造函数是创建的非公平锁，可以通过参数true设为公平锁，但公平锁表现的性能不是很好。

3).锁绑定多个条件，一个ReentrantLock对象可以同时绑定对个对象。

ReentrantLock用法:

```
Lock lock=new ReentrantLock();
 lock.lock();//加锁
 try{
//同步区域
 }finally{
  lock.unlock();//释放锁
}
```

必须在finally里执行释放锁，以防止异常导致锁未释放，其他线程阻塞。

29.什么场景下可以使用 volatile 替换 synchronized

Volatile 如何保证可见性：

一个线程对共享变量的修改能够立刻被其他线程看到，这叫做可见性。cup对共享变量的访问是从内存中读取到高速缓存中，使用volatile修饰的变量在汇编层会加上lock，该操作有两个特点，第一是该变量通过线程操作后写入高速缓存后会被立刻写回内存中。第二写入内存后，其他cpu的高速缓存中的变量被设置为无效缓存，其实现原理为是否能够命中缓存，命中的判断是变量在内存中的地址是否发生变化。因此，在使用了volatile修饰共享变量后，每一次写操作都相当于直接操作内存中的变量，而不是高速缓存中的变量，每一次读也是直接读内存中的变量值。

Volatile 只能保证可见性，不能保证原子性。Synchronized保证原子性和可见性。

Volatile 只能在全局变量上使用，Synchronized用于方法、类、块。

30.有T1，T2，T3三个线程，怎么确保它们按顺序执行？怎样保证T2在T1执行完后执行，T3在T2执行完后执行

T3需要执行T2.join(),T2需要执行T1.join().

31.同步块内的线程抛出异常会发生什么

使用synchronized 同步块出现异常，锁还是会被释放，其他等待线程依旧可以执行。

32.Java Concurrency API 中 的 Lock 接口是什么？对比同步它有什么优势

1).可以使锁更公平；
2).可以使线程在等待锁的时候响应中断；搜索
3).可以让线程尝试获取锁，并在无法获取锁的时候立即返回或者等待一段时间；
4).可以在不同的范围，以不同的顺序获取和释放锁。

33.ReadWriteLock是什么?

练习：http://blog.csdn.net/zhuhezan/article/details/6613108
读-读 不互斥
读-写 互斥
写-读 互斥
写-写 互斥
注意：同一线程中，持有读锁后，不能直接调用写锁的lock方法 ，否则会造成死锁。
用法：

```
ReadWriteLock lock = new ReentrantReadWriteLock(false);
//获取读锁
Lock.readLock().lock();
//释放读锁
Lock.readLock().unlock();
//获取写锁
Lock.writeLock().lock();
//释放写锁
Lock.writeLock().unlock();
```

34.**什么是乐观锁（Optimistic Locking）？如何实现乐观锁？如何避免ABA问题**

乐观锁(Optimistic Lock), 顾名思义，就是很乐观，每次去拿数据的时候都认为别人不会修改，所以不会上锁，但是在更新的时候会判断一下在此期间别人有没有去更新这个数据，可以使用版本号等机制。乐观锁适用于多读的应用类型，这样可以提高吞吐量，像数据库如果提供类似于write_condition机制的其实都是提供的乐观锁。
悲观锁(Pessimistic Lock), 顾名思义，就是很悲观，每次去拿数据的时候都认为别人会修改，所以每次在拿数据的时候都会上锁，这样别人想拿这个数据就会block直到它拿到

锁。传统的关系型数据库里边就用到了很多这种锁机制，比如行锁，表锁等，读锁，写锁等，都是在做操作之前先上锁。

35.什么是死锁(Deadlock)？导致线程死锁的原因？如何确保 N 个线程可以访问 N 个资源同时又不导致死锁

多线程产生死锁的四个必要条件：

1). 互斥条件：一个资源每次只能被一个进程使用。

2). 保持和请求条件：一个进程因请求资源而阻塞时，对已获得资源保持不放。

3). 不可剥夺调教：进程已获得资源，在未使用完成前，不能被剥夺。

4). 循环等待条件：若干进程之间形成一种头尾相接的循环等待资源关系。

只要破坏其中任意一个条件，就可以避免死锁，其中最简单的就是破环循环等待条件。按同一顺序访问对象，加载锁，释放锁。

36.死锁与活锁的区别，死锁与饥饿的区别

死锁：互不相让，造成线程等待。

活锁：过于谦让，造成线程都在等待。

饥饿：轮不到，造成线程等待。

37.如何实现分布式锁

1).基于数据库实现分布式锁。

2).基于缓存（redis，memcached，tair）实现分布式锁

3).基于Zookeeper实现分布式锁

38.有哪些无锁数据结构，他们实现的原理是什么

39.读写锁可以用于什么应用场景

大量度操作，少量写操作。

40.在线程中你怎么处理不可捕捉异常

线程的一个回调方法：Thread.UncaughtExceptionHandler 可以在run 方法之外捕获异常。

单词意思：Uncaught 漏网；未捕获

实际项目中如何使用多线程的。你在多线程环境中遇到的常见的问题是什么？你是怎么解决它的

41.假如有一个第三方接口，有很多个线程去调用获取数据，现在规定每秒钟最多有 10 个线程同时调用它，如何做到

使用semaPhore类。

42.如何在 Windows 和 Linux 上查找哪个线程使用的 CPU 时间最长

top -H -p pid

43.如何确保 main() 方法所在的线程是 Java 程序最后结束的线程

在main方法中，使用子线程对象调用其join方法。

- **集合**

1. **常用集合都有哪些?**
   挑自己会的说。

2. arraylist使用什么存储数据，扩容机制是什么?

3.Linkedlist使用什么存储数据?

4.Arraylist和linkedlist的区别，两者的使用场景是什么，为什么这样选择。

5.hashmap为什么是线程不安全的？多线程情况下会有什么问题产生?

这个问题好好研究，搞明白那个死循环是怎么产生的?

HashMap在put的时候，插入的元素超过了容量（由负载因子决定）的范围就会触发扩容操作，就是rehash，这个会重新将原数组的内容重新hash到新的扩容数组中，在多线程的环境下，存在同时其他的元素也在进行put操作，如果hash值相同，可能出现同时在同一数组下用链表表示，造成闭环，导致在get时会出现死循环，所以HashMap是线程不安全的。

6. Hashmap使用的数据存储结构是什么，讲一下put方法，和get方法的内容。

7.Hashmap扩容机制是什么样的？什么情况下会扩容，扩容多少。

Hashmap的扩容需要满足两个条件：当前数据存储的数量（即size()）大小必须大于等于阈值；当前加入的数据是否发生了hash冲突。

因为上面这两个条件，所以存在下面这些情况

（1）、就是hashmap在存值的时候（默认大小为16，负载因子0.75，阈值12），可能达到最后存满16个值的时候，再存入第17个值才会发生扩容现象，因为前16个值，每个值在底层数组中分别占据一个位置，并没有发生hash碰撞。

（2）、当然也有可能存储更多值（超多16个值，最多可以存26个值）都 还 没有扩容。原理：前11个值全部hash碰撞，存到数组的同一个位置（ 这时元素个数小于阈值12，不会扩容），后面所有存入的15个值全部分 散 到数组剩下的15个位置（这时元素个数大于等于阈值，但是每次存入 的元素并没有发生hash碰撞，所以不会扩容），前面11+15=26，所以 在存入第27个值的时候才同时满足上面两个条件，这时候才会发生扩容 现 象。

8. Hashmap默认的大小。
16

9. String,Stringbuffer,Stringbiuder区别，特点，使用场景。

- **Web**

1. **Cookie 和 Session的区别?**
2.get 和 post请求的区别
3.RPC 通信和 RMI 区别

1)方法调用方式不同：
RMI中是通过在客户端的Stub对象作为远程接口进行远程方法的调用。每个远程方法都具有方法签名。如果一个方法在服务器上执行，但是没有相匹配的签名被添加到这个远程接口(stub)上，那么这个新方法就不能被RMI客户方所调用。**RPC中是通过网络服务协议向远程主机发送请求**，请求包含了一个参数集和一个文本值，通常形成“classname.methodname(参数集)”的形式。RPC远程主机就去搜索与之相匹配的类和方法，找到后就执行方法并把结果编码，通过网络协议发回。
2) 适用语言范围不同：
RMI只用于Java；
RPC是网络服务协议，与操作系统和语言无关。
3) 调用结果的返回形式不同：
Java是面向对象的，所以RMI的调用结果可以是对象类型或者基本数据类型；
RMI的结果统一由**外部数据表示（External Data Representation, XDR）语言**表示，这种语言抽象了字节序类和数据类型结构之间的差异。

Java数据类型有哪些，长度多少?

类型	字节	表示范围	包装类
byte(字节型)	1	-128~127	Byte
short(短整型)	2	-32768～32767	Short
int(整型)	4	-2147483648～2147483647	Integer
long(长整型)	8	-9223372036854775808 ～ 9223372036854775807	Long
float(浮点型)	4	-3.4E38～3.4E38	Float
double(双精度型)	8	-1.7E308～1.7E308	Double
char(字符型)	2	从字符型对应的整型数来划分，其表示范围是0～65535	Charater
booealn(布尔型)	1	true或false	Boolean

1.说说你对面向对象的理解。

不是一朝一夕能讲明白的，看你自己的积累了。

1. **类和类之间有什么关系?**
2. **接口可以多继承吗?**

只支持类之间的单继承，但支持接口之间的多继承

- **JVM调优及原理**
- **JVM原理**

1.简单说说你了解的类加载器。是否实现过类加载器

大部分Java程序一般会使用到以下三种系统提供的类加载器：

1)启动类加载器（Bootstrap ClassLoader）：负责加载JAVA_HOME\lib目录中并且能被虚拟机识别的类库到JVM内存中，如果名称不符合的类库即使放在lib目录中也不会被加载。该类加载器无法被Java程序直接引用。

2)扩展类加载器（Extension ClassLoader）：该加载器主要是负责加载JAVA_HOME\lib\，该加载器可以被开发者直接使用。

3)应用程序类加载器（Application ClassLoader）：该类加载器也称为系统类加载器，它负责加载用户类路径（Classpath）上所指定的类库，开发者可以直接使用该类加载

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默认的类加载器。
实现类加载器：继承ClassLoader ，重写findClass方法。

2.什么是Java虚拟机？为什么Java被称作是“平台无关的编程语言”
Java虚拟机：将.java文件，翻译为响应平台可执行的机器码，以供机器执行。
Java是一个精通各国语言的翻译，所以平台无关。

3.说说你知道的几种主要的jvm 参数
可以通过java -help命令查看标准参数。Java -X 查看非标准参数。
1).-server //服务器模式
2).-Xmx2g //JVM最大允许分配的堆内存，按需分配
3).-Xms2g //JVM初始分配的堆内存，一般和Xmx配置成一样以避免每次gc后JVM重新分配内存。
4).-Xmn256m //年轻代内存大小，整个JVM内存=年轻代 + 年老代 + 持久代
5).-XX:PermSize=128m //持久代内存大小
6).-Xss256k //设置每个线程的堆栈大小
7).-XX:+DisableExplicitGC //忽略手动调用GC, System.gc()的调用就会变成一个空调用，完全不触发GC
8).-XX:+UseConcMarkSweepGC //并发标记清除（CMS）收集器
9).-XX:+CMSParallelRemarkEnabled //降低标记停顿
10).-XX:+UseCMSCompactAtFullCollection //在FULL GC的时候对年老代的压缩
11).-XX:LargePageSizeInBytes=128m //内存页的大小
12).-XX:+UseFastAccessorMethods //原始类型的快速优化
13).-XX:+UseCMSInitiatingOccupancyOnly //使用手动定义初始化定义开始CMS收集
14).-XX:CMSInitiatingOccupancyFraction=70 //使用cms作为垃圾回收使用70％后开始CMS收集

4.-XX:+UseCompressedOops 有什么作用
当你将你的应用从 32 位的 JVM 迁移到 64 位的 JVM 时，由于对象的指针从 32 位增加到了 64 位，因此堆内存会突然增加，差不多要翻倍。这也会对 CPU 缓存（容量比内存小很多）的数据产生不利的影响。因为，迁移到 64 位的 JVM 主要动机在于可以指定最大堆大小，通过压缩 OOP 可以节省一定的内存。通过 -XX:+UseCompressedOops 选项，JVM 会使用 32 位的 OOP，而不是 64 位的 OOP。

5.JVM如何加载.class文件?
开启加载的方式：

1).**隐式装载**，程序在运行过程中碰到new等方式生成对象时，隐式调用类加载器加载　　对应的类到JVM中
2).**显示装载**，通过class.forName（）等方法，显示加载需要的类，得到class对象。可以调用class对象的new newInstance()方法创建类的实例对象。
确定加载器：
两种方式都需要调用类加载器。
Java中有三个基本的类加载器，通过双亲委派模型协调使用哪一个加载器所持有的.class字节流加载类。
加载：
1).通过"类全名"来获取定义此类的二进制字节流
2).将字节流所代表的静态存储结构转换为方法区的运行时数据结构
3).在java堆中生成一个代表这个类的java.lang.Class对象，作为方法区这些数据的访问入口
后续还会有：
- - 以下为连接阶段
验证：(格式、元数据、字节码)
准备：为类变量分配内存 分配在方法区
解析：虚拟机常量池内的符号引用替换为直接引用的过程。
- - 连接结束
初始化：执行构造器
使用：
卸载

6.**JVM内存分哪几个区，每个区的作用是什么?**
好好看看：http://blog.csdn.net/lmq321281/article/details/51058200
https://www.cnblogs.com/hanzai/p/6261223.html
1.方法区，所有线程的共享区域，保存已加载的类，常量，静态变量，又称为永久代。这个区域内存耗尽会报java.lang.OutOfMemoryError: PermGen full
2).堆，所有线程的共享区域，分配对象实例，分为新生代（eden较大、survivor from、survivor to）和年老代（新生代中呆了足够长时间后移到这里）。堆中内存耗尽会报java.lang.OutOfMemoryError: Java heap space。
3).本地方法栈，线程私有，调用native方法时用的栈。
4).虚拟机栈，线程私有，保存局部变量、方法调用栈，递归过深会报StackOverflowError。如果虚拟机支持栈扩展，直到申请不到内存报OutOfMemoryError。
5).程序计数器，线程私有，指示下一个字节码，执行native方法时为空，不会报OutOfMemoryError。

7.一个对象从创建到销毁都是怎么在这些部分里存活和转移的

字节码文件加载到方法区 初始化类变量常量。初始化class对象到堆中。调用构造函数初始化实例对象保存在堆中新生代，调用方法构建栈帧，压入虚拟机栈等待执行，一次垃圾回收后，方法未执行，对象从堆中的新生代移动到老年代，方法出栈并无其他引用，则销毁堆中实例对象。

8.解释内存中的栈(stack)、堆(heap)和方法区(method area)的用法?

堆区(heap):

门用来保存对象的实例(new 创建的对象和数组)，实际上也只是保存对象实例的属性值，属性的类型和对象本身的类型标记等，并不保存对象的方法（方法是指令，保存在Stack中）

1).存储的全部是对象，每个对象都包含一个与之对应的class的信息。(class的目的是得到操作指令)

2).jvm只有一个堆区(heap)被所有线程共享，堆中不存放基本类型和对象引用，只存放对象本身.

3).一般由程序员分配释放， 若程序员不释放，程序结束时可能由OS回收 。

栈区(stack):对象实例在Heap 中分配好以后，需要在Stack中保存一个4字节的Heap内存地址，用来定位该对象实例在Heap 中的位置，便于找到该对象实例。

1).每个线程包含一个栈区，栈中只保存基础数据类型的对象和自定义对象的引用(不是对象)，对象都存放在堆区中

2).每个栈中的数据(原始类型和对象引用)都是私有的，其他栈不能访问。

3).栈分为3个部分：基本类型变量区、执行环境上下文、操作指令区(存放操作指令)。

4.由编译器自动分配释放 ，存放函数的参数值，局部变量的值等.

静态区/方法区:

1).方法区又叫静态区，跟堆一样，被所有的线程共享。方法区包含所有的class和static变量。

2).方法区中包含的都是在整个程序中永远唯一的元素，如class文件，static变量。

3).全局变量和静态变量的存储是放在一块的，初始化的全局变量和静态变量在一块区域， 未初始化的全局变量和未初始化的静态变量在相邻的另一块区域。

9.JVM中哪个参数是用来控制线程的栈堆大小?

-Xss128k：设置每个线程的堆栈大小。JDK5.0以后每个线程堆 栈大小为1M，以前每个线程堆栈大小为256K。根据应用的线程所需内存大小进行调整

10.简述重排序，内存屏障，happen-before，主内存，工作内存

重排序:对指令的重新排序

内存屏障：禁止重排序

happen-before：现行发生 对volatile的写操作现行发生于读操作

主内存：堆和方法区 其中的变量对于所有线程是共享的
工作内存：每条线程独有的内存，变量是从主内存中拷贝而来。不同线程之间不能够互相访问工作内存。

11.Java中存在内存泄漏问题吗？请举例说明
1).静态集合类，例如HashMap和Vector。如果这些容器为静态的，由于它们的声明周期与程序一致，那么容器中的对象在程序结束之前将不能被释放，从而造成内存泄漏。
2).各种连接，例如数据库的连接、网络连接以及IO连接等。
3).监听器。在Java语言中，往往会使用到监听器。通常一个应用中会用到多个监听器，但在释放对象的同时往往没有相应的删除监听器，这也可能导致内存泄漏。
4).变量不合理的作用域。一般而言，如果一个变量定义的作用域大于其使用范围，很有可能会造成内存泄漏，另一方面如果没有及时地把对象设置为Null，很有可能会导致内存泄漏的放生

14.jstack，jstat，jmap，jconsole怎么用

jstack：
用于打印出给定的java进程ID或core file或远程调试服务的Java堆栈信息，如果是在64位机器上，需要指定选项"-J-d64"，Windows的jstack使用方式只支持以下的这种方式：jstack [-l] pid 如果java程序崩溃生成core文件，jstack工具可以用来获得core文件的java stack和native stack的信息，从而可以轻松地知道java程序是如何崩溃和在程序何处发生问题。另外，jstack工具还可以附属到正在运行的java程序中，看到当时运行的java程序的java stack和native stack的信息, 如果现在运行的java程序呈现hung的状态。可以用此工具解决。
jstat:
用于监控基于HotSpot的JVM，对其堆的使用情况进行实时的命令行的统计，使用jstat我们可以对指定的JVM做如下监控：
- 类的加载及卸载情况
- 查看新生代、老生代及持久代的容量及使用情况
- 查看新生代、老生代及持久代的垃圾收集情况，包括垃圾回收的次数及垃圾回收所占用时间
- 查看新生代中Eden区及Survior区中容量及分配情况等
jmap:
堆内存使用情况，对象加载情况，对象存活情况，类加载器信息，使用的GC算法，堆配置参数等。
jconsole：

java GUI监视工具，可以以图表化的形式显示各种数据.支持远程监控。

35.什么情况下会发生栈内存溢出

线程请求的栈深度大于虚拟机所允许的最大深度，将抛出StackOverflowError异常

1).是否有递归调用

2).是否有大量循环或死循环

3).全局变量是否过多

4).数组、List、map数据是否过大

5).使用DDMS工具进行查找大概出现栈溢出的位置

36.垃圾回收器都干嘛的，区别。

- **JVM调优**

1.你是如何处理内存泄露或者栈溢出问题的

http://blog.csdn.net/mine_song/article/details/62851790

2.你们线上应用的 JVM 参数有哪些

http://blog.csdn.net/chenleixing/article/details/43230527

3.怎么提升系统的QPS和吞吐量

1).集群+负载均衡

2).增加缓存

3).系统拆分

4).分库分表

5).垂直拆分+水平拆分

6).异步化+MQ

- **JVM GC原理**

1.什么时候会导致垃圾回收

采用分代回收，不同代开始GC的情况不同，但都会因为所在内存区达到容量阈值启动GC。

2.GC是怎么样运行的

由jvm控制，内部有算法机制判断在合适触发GC。

1. **新生代，老年代，永久代是什么?**

新生代：对象在新生代被创建，频繁发生垃圾回收的区域。Minor GC。复制清除算法。Eden和sruvior比例为8:1:1.

老年代：新生代在进行GC过程中存活下的对象，按照每GC增长一岁，默认达到十五岁的 对象会被移动到老年代，XX:MaxTenuringThreshold参数可调整。大对象可以被直接 分配到老年代。标记清除或标记整理算法。

永久代：存放很难被回收的对象。标记清除或标记整理算法。

4.GC 有几种方式？怎么配置

1).**标记清除方式** 扫描所有对象是否有引用 有就打标机 没有就不标机 然后清楚不标记的对象。

2).**复制收集方式** 从GCroot开始递归的寻找所引用的对象，并移动到新开辟的内存空间，寻找完成后，删除剩下的对象。

3).**引用计数** 有引用计数加一，引用失效减一 为0 就是没有引用可以删除 缺点是有环状的情况下，垃圾环无法被清除。

5.什么时候一个对象会被GC？如何判断一个对象是否存活

没有被引用的时候可能会被GC。

引用计数法和可达性分析法。

可作为GC Roots的对象包括下面几种：

•虚拟机栈中引用的对象

•方法区中类静态属性引用的对象

•方法区中常量引用的对象

•本地方法栈中JNI[即一般说的Native]引用的对象

6.System.gc() Runtime.gc()会做什么事情？能保证 GC 执行吗

通知垃圾回收机制进行垃圾回收，但是不能保证GC执行。能提高优先级吧。

7.垃圾回收器可以马上回收内存吗？有什么办法主动通知虚拟机进行垃圾回收？

System.gc()和Runtime.gc().不会马上回收内存。

1. **Minor GC 、Major GC 与 Full GC分别在什么时候发生**

众说纷纭：

版本一：minor GC 负责年轻代回收，minor GC 会触发Mjor GC,所以他俩是一回事，Full GC 全局GC.

版本二：minor GC 年轻代 Full 和 Major 全局GC。

8.GC收集器有哪些？

	作用域	算法	暂停	线程数	开关	模式	其他
serial	新生代	停止复制	是	单	-XX:+UseSerialGC	Serial+Serial Old	Client默认模式
ParNew	新生代	停止复制	是	多	-XX:+UseParNewGC	ParNew+Ser	-XX:ParallelG

						ial Old	CThreads 设置线程数
Parallel Scavenge	新生代	停止复制	是	单	-XX:+UseParallelGC	Parallel Scavenge +Serial Old	Server 默认模式 -XX:GCTimeRatio 设置用户时间的比例默认为 99:1
Serial Old	老年代	标记整理	是	单			
Parallel Old	老年代	标记整理	是	多	-XX:+UseParallelOldGC		
CMS	老年代	标记清除	否	多	XX:+UseConcMarkSweepGC	ParNew+CMS+Serial Old	
G1							

9.Serial 与 Parallel GC之间的不同之处
前者单线程，后者多线程。
CMS 收集器 与 G1 收集器的特点与区别
CMS垃圾回收器的工作过程
吞吐量优先和响应优先的垃圾收集器选择
举个实际的场景，选择一个GC策略

1. **JVM的永久代中会发生垃圾回收吗?**

不会，但是当满的时候会触发full GC.

- **数据库**
- **SQL编写**

1. **写行列置换sql.**
2. **Sql优化最佳实践。**

- **事物**

1. **什么是 ACID?**

Atomicity原子性：
独立的操作单元，要么全做，要么全不做。

Consistency一致性
一次事物之后数据应该从一个一致性状态到达另一个一致性状态。

Isolation隔离性
事务之间不会互相影响。各干各的事情，在事务结束时的结果满足预期结果。

Durability持久性
提交了就一直都在。

1. **mysql都有哪些数据库引擎，有什么优缺点，各自的使用场景。**

MYISAM（默认）
MYISAM是MYSQL的ISAM扩展格式和缺省的数据库引擎。除了提供ISAM里所没有的索引和字段管理的大量功能，MYISAM还使用一种表格锁定的机制，来优化多个并发的读写操作。其代价是你需要经常运行OPTIMIZETABLE命令，来恢复被更新机制所浪费的空间。MYISAM还有一些有用的扩展，例如用来修复数据库文件的MYISAMCHK工具和用来恢复浪费空间的MYISAMPACK工具

MYISAM
优势：ISAM执行读取操作的速度很快，而且不占用大量的内存和存储资源
不足：它不支持事务处理，也不能够容错：如果你的硬盘崩溃了，那么数据文件就无法恢复了。如果你正在把ISAM用在关键任务应用程序里，那就必须经常备份你所有的实时数据，通过其复制特性，MYSQL能够支持这样的备份应用程序。

INNODB和DBD
INNODB和BERKLEYDB（BDB）数据库引擎都是造就MYSQL灵活性的技术的直接产品，这项技术就是MYSQL++API。在使用MYSQL的时候，你所面对的每一个挑战几乎都源于ISAM和MYISAM数据库引擎不支持事务处理也不支持外来键。尽管要比ISAM和MYISAM引擎慢很多，但是INNODB和BDB包括了对事务处理和外来键的支持，这两点

都是前两个引擎所没有的。如前所述，如果你的设计需要这些特性中的一者或者两者，那你就要被迫使用后两个引擎中的一个了。

HEAP

HEAP允许只驻留在内存里的临时表格。驻留在内存里让HEAP要比ISAM和MYISAM都快，但是它所管理的数据是不稳定的，而且如果在关机之前没有进行保存，那么所有的数据都会丢失。在数据行被删除的时候，HEAP也不会浪费大量的空间。HEAP表格在你需要使用SELECT表达式来选择和操控数据的时候非常有用。要记住，在用完表格之后就删除表格。

1. **数据外键优缺点。**

外键约束该表中外键对应的值必须在参照表对应字段中。

优点：会帮助校验数据库完整性，表间关系清晰。

缺点：

数据库需要维护外键的内部管理；

外键等于把数据的一致性事务实现，全部交给数据库服务器完成；

有了外键，当做一些涉及外键字段的增，删，更新操作之后，需要触发相关操作去检查，而不得不消耗资源；

外键还会因为需要请求对其他表内部加锁而容易出现死锁情况；

1. **mysql数据库的隔离级别。**

读未提交

会产生脏读---事物间能读到为提交的数据。

读提交

会产生不可重复读现象---在一个事物内两次读取相同数据得到的结果不相同

可重复读

会产生幻读---可重复读加行锁，只对修改有效，对新增无效，第一次读和第二次读中间新增或删除数据时产生幻读。

串行化

效率低 事物串行操作 无并发。

数据库隔离级别实现原理。

使用的锁：

共享锁(S锁) 读读不互斥，读写互斥，其他事务只能加共享锁，不能加排它锁。

排它锁(X锁) 读写互斥 其他事务不能再加锁。

读未提交

读操作：不加锁 即产生脏读的原因。

写操作：行级共享锁 其他线程只能读 不能写。

读提交
读操作：行级共享锁 只有在读的时候加锁 读完就释放。
写操作：行级排它锁 在写操作的瞬间加锁 不允许其他事务做读写操作。

可重复读
读操作：行排他锁。
写操作：行级排它锁 在写操作的瞬间加锁 不允许其他事务做读写操作。

串行化
读操作：表级共享锁 只有在读的时候加锁 读完就释放。
写操作：表级排它锁 在写操作的瞬间加锁 不允许其他事务做读写操作。
以上为悲观锁的实现方式。

解释下mvcc。
多版本并发控制协议。
以乐观锁的方式实现数据隔离级别的实现。是myssql 数据库innodb引擎采用的方式。
每行数据添加了创建时间和删除时间。保存事物的ID号，新事务的ID号递增。
修改操作会将当前的事务ID号保存到创建时间，删除操作会把事务ID保存到删除时间内。
查询按照需要满足 创建时间<=当前事务ID
删除时间>当前事务ID
就是这个事务未结束前，有其他事务删除数据，这些数据在这个事务的时候还未发生删除，所以也要得到到。
以下为专业解释：

- SELECT时，读取创建版本号<=当前事务版本号，删除版本号为空或>当前事务版本号。
- INSERT时，保存当前事务版本号为行的创建版本号
- DELETE时，保存当前事务版本号为行的删除版本号
- UPDATE时，插入一条新纪录，保存当前事务版本号为行创建版本号，同时保存当前事务版本号到原来删除的行

- **索引分区**

1. **如何创建索引和分区。**
2. **索引和分区的实现原理。**
3. **Mysql索引的种类有哪些。**

唯一索引
主键索引
联合索引

4. 联合索引失效问题。

例如索引是key index （a，b，c）。可以支持（a）（ab）（abc）3种组合进行查找，但不支持（bc）进行查找 ，当最左侧字段是常量引用时，索引就十分有效。

5. 如何防止sql注入

使用预编译，字符串过滤，正则表达式校验

- **Ridis/ZK**

1.Redis数据类型

2.Redis能干什么?

答你知道的，会继续问你实现原理。比如：

实现分布式锁的实现、

如何解决缓存一致性问题。

为什么要使用redis,有什么优势。

- **SSM使用及原理**
- **spring 原理**

1. **什么是控制反转（Inversion of Control）与依赖注入（Dependency Injection）**

控制反转即IoC (Inversion of Control)，它把传统上由程序代码直接操控的对象的调用权交给容器，通过容器来实现对象组件的装配和管理。所谓的“控制反转”概念就是对组件对象控制权的转移，从程序代码本身转移到了外部容器。

IoC是一个很大的概念，可以用不同的方式来实现。其主要实现方式有两种：<1>依赖查找（Dependency Lookup）：**容器**提供回调接口和上下文环境给**组件**。EJB和Apache Avalon都使用这种方式。<2>依赖注入（Dependency Injection）：组件不做定位查询，只提供普通的Java方法让容器去决定依赖关系。后者是时下最流行的IoC类型，其又有接口注入（Interface Injection），设值注入（Setter Injection）和构造子注入（Constructor Injection）三种方式。

2. 哪种依赖注入方式你建议使用，构造器注入，还是 Setter方法注入

两种依赖方式都可以使用，构造器注入和Setter方法注入。最好的解决方案是用构造器参数实现强制依赖，setter方法实现可选依赖。

3. ioc的启动过程，把那个流程说一遍。

第一个阶段是容器的启动阶段，第二个阶段是 Bean 实例化阶段

4.ioc解决什么问题。

5.bean的作用域都有哪些？默认是什么。

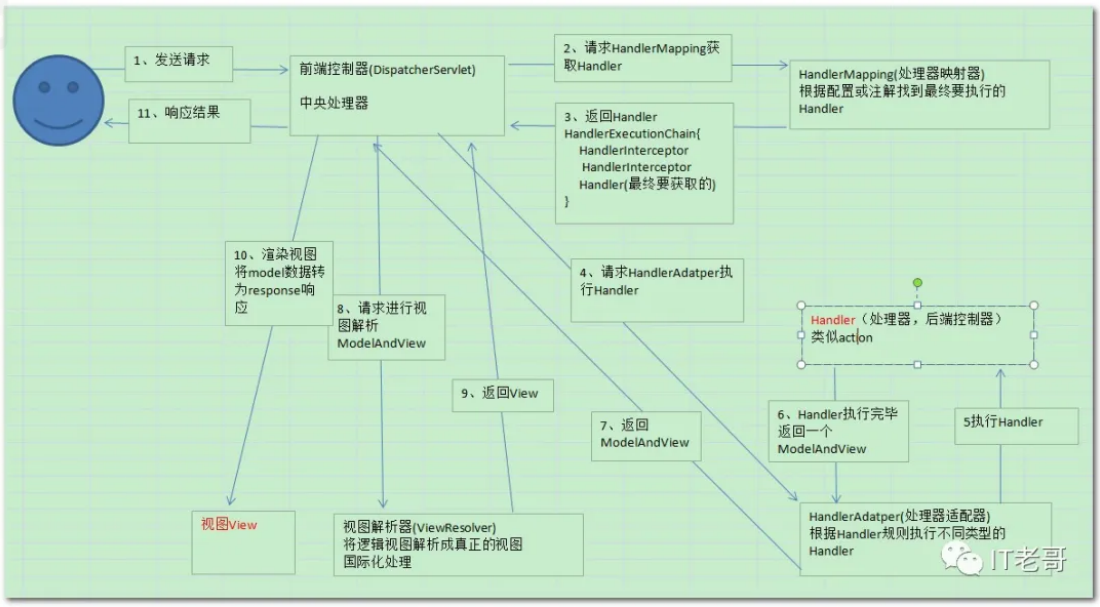
singleton：单例模式，在整个Spring IoC容器中，使用singleton定义的Bean将只有一个实例（默认）
prototype：原型模式，每次通过容器的getBean方法获取prototype定义的Bean时，都将产生一个新的Bean实例
request：对于每次HTTP请求，使用request定义的Bean都将产生一个新实例，即每次HTTP请求将会产生不同的Bean实例。只有在Web应用中使用Spring时，该作用域才有效
session：对于每次HTTP Session，使用session定义的Bean豆浆产生一个新实例。同样只有在Web应用中使用Spring时，该作用域才有效
globalsession：每个全局的HTTP Session，使用session定义的Bean都将产生一个新实例。典型情况下，仅在使用portlet context的时候有效。同样只有在Web应用中使用Spring时，该作用域才有效

6.解释一下什么叫AOP（面向切面编程）

7.你如何理解AOP中的连接点（Joinpoint）、切点（Pointcut）、增强（Advice）、引介（Introduction）、织入（Weaving）、切面（Aspect）这些概念
8.aop的实现方式
Jdk动态代理、cglib动态代理。这俩代理手写使用代码，并且搞明白实现原理，对比区别和优缺点。

- **Spring MVC原理**

1. **查Springmvc父子上下文的相关东西看看，了解下就行**



- **Mybatic**

1.#和$的区别，这块问过的最多的问题就是这个。

#能够防止sql注入。推荐使用。#：用于变量替换，$:实质上是字符串拼接

- **Linux 常用及Shell编程**

  说说你常用的linux命令。

  简单的背下，复杂的就手写练习吧。

  Awk 了解下。

- **设计模式及算法**

  1.JAVA动态代理如何使用。

```
public class HelloServiceProxy implements InvocationHandler{

    private Object target;
    public  Object bind(Object target) {
           this.target = target;
           //取得代理对象
          return Proxy.newProxyInstance(target.getClass().getClassLoader(),

                   target.getClass().getInterfaces(), this);
       }

    @Override
    public Object invoke(Object proxy, Method method, Object[] args) throws Th
         System.err.println("#############我是JDK动态代理################");
           Object result = null;
```

```
            //反射方法前调用
            System.err.println("我准备说hello。");
            //反射执行方法   相当于调用target.sayHelllo;
            result=method.invoke(target, args);
            //反射方法后调用.
            System.err.println("我说过hello了");
            return result;
    }
        public static void main(String[] args) {
            HelloServiceProxy proxy = new HelloServiceProxy();
            HelloService service = new HelloServiceImpl();
            //绑定代理对象。
            service = (HelloService) proxy.bind(service);
            //这里service经过绑定，就会进入invoke方法里面了。
            service.doSomeThing();
        }
}
```

2描述动态代理的几种实现方式，分别说出相应的优缺点
两种：JDK动态代理、cglib第三方动态代理工具。
相比：JDK动态代理生成代理类的速度更快，cglib相对较慢。
JDK动态代理的前提是被代理类需要实现接口，而cglib不需要实现接口。
JDK代理是目标对象和代理对象同时实现一个接口。
Cglib代理是为被代理类生成一个子类，覆盖方法。被代理类不能被生命才final。

3.这块会先问你知道什么设计模式，然后从你说的那些里面挑一个问怎么用。

4.把线程安全的单例模式写会了，然后搞明白其中为什么要用双检锁，为什么使用volaitle。

- **项目介绍**

1. **简历里哪个项目最近做的，或者你最熟悉的，挑一个聊聊。**

会先让你介绍下这个项目干嘛，给谁用的，解决什么问题。

1. **项目中都有到什么你觉得比较牛逼的技术点。或者让你印象深刻的点。**
2. **遇到过什么坑，你怎么处理的。**

写在简历上的项目，面试前要捋捋思路。别稀里糊涂的，面试官不喜欢不了解自己项目的人。

- **网络**

1. **http和tcp 的区别。**
   1.http属于应用层协议，tcp协议属于传输层协议。
   2. http协议建立在tcp协议之上。
   3. keep -Alive