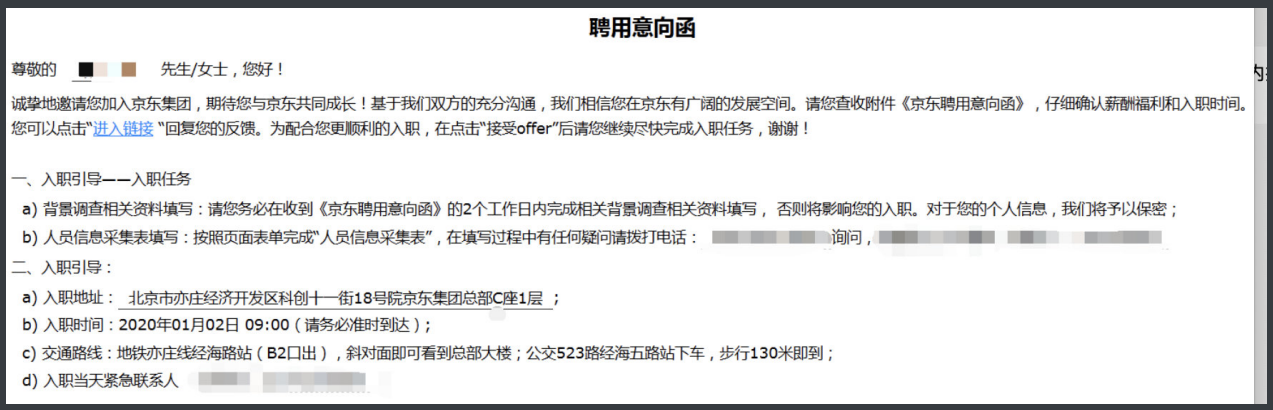
> 你知道的越多，你不知道的越多

> 点赞再看，养成习惯

> 本文 **GitHub** https://github.com/JavaFamily 已收录，有一线大厂面试点思维导图，也整理了很多我的文档，欢迎Star和完善，大家面试可以参照考点复习，希望我们一起有点东西。

前言



聘用意向函

尊敬的 [redacted] 先生/女士，您好！

诚挚地邀请您加入京东集团，期待您与京东共同成长！基于我们双方的充分沟通，我们相信您在京东有广阔的发展空间。请您查收附件《京东聘用意向函》，仔细确认薪酬福利和入职时间。您可以点击"进入链接 "回复您的反馈。为配合您更顺利的入职，在点击"接受offer"后请您继续尽快完成入职任务，谢谢！

一、入职引导——入职任务

a) 背景调查相关资料填写：请您务必在收到《京东聘用意向函》的2个工作日内完成相关背景调查相关资料填写，否则将影响您的入职。对于您的个人信息，我们将予以保密；

b) 人员信息采集表填写：按照页面表单完成"人员信息采集表"，在填写过程中有任何疑问请拨打电话：[redacted]询问，[redacted]

二、入职引导：

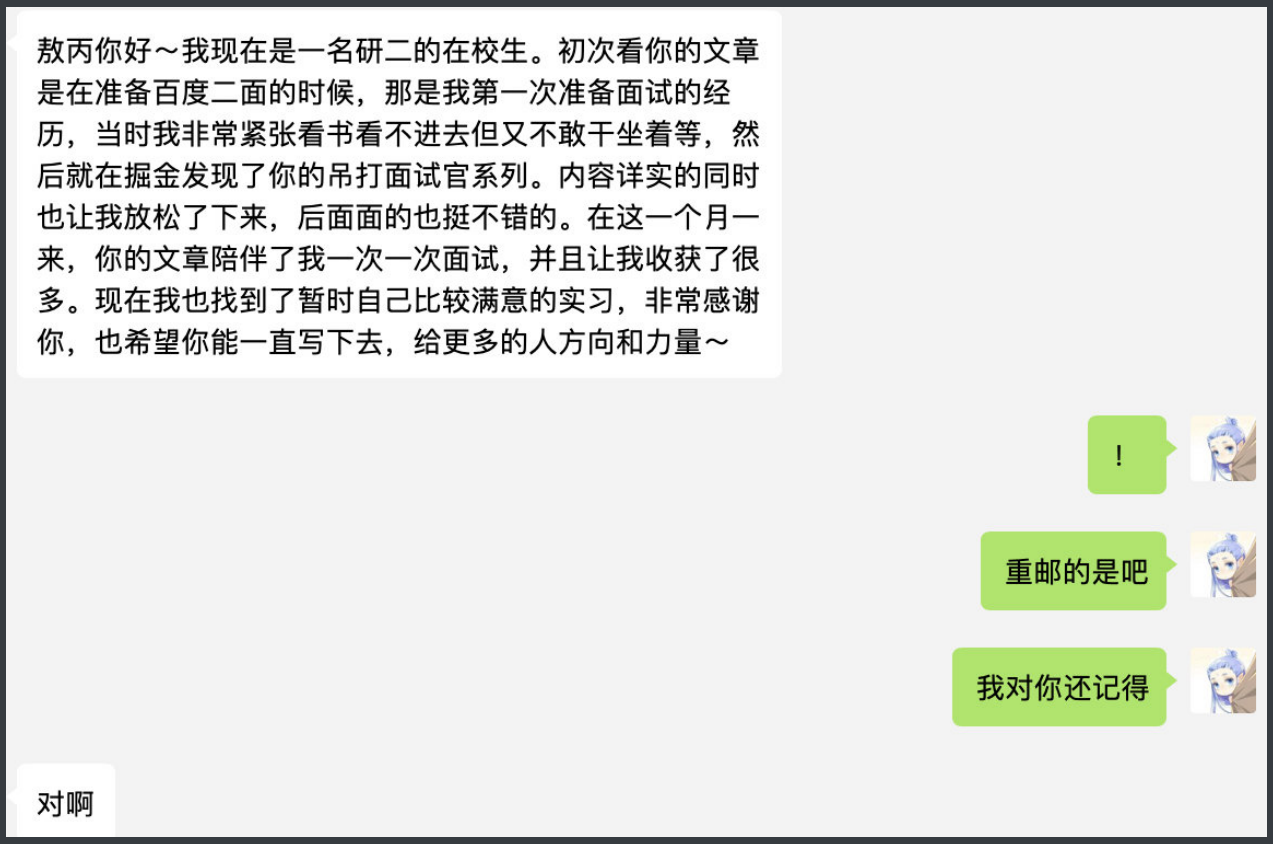
a) 入职地址：北京市亦庄经济开发区科创十一街18号院京东集团总部C座1层；

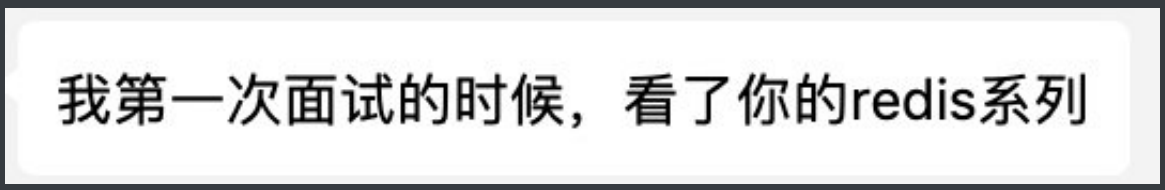
b) 入职时间：2020年01月02日 09:00（请务必准时到达）；

c) 交通路线：地铁亦庄线经海路站（B2口出），斜对面即可看到总部大楼；公交523路经海五路站下车，步行130米即到；

d) 入职当天紧急联系人 [redacted]

还记得我上周说的重庆邮电研二的读者么？





知道他拿了Offer之后我也很开心，我就想把它的面试经历和面试题分享出来，这样让大家也知道，目前实习的面试大概是怎么样子的。

很优秀的是，他每次面试都会自己做总结，我一说他就直接给到了我文档，优秀的仔，我可能知道他拿这么多Offer的原因了吧。

整个文章我本来想贴一下面试过程，然后贴下面试题就完事了，这就可以水一篇了。

但是你们肯定会骂我是渣男，不负责任，题目都给了还不给我答案。

我暖男来的，最后还是给你们写了答案，别爱我，没结果。

正文

这是一个秋高气爽的早上，Angela丙丙一如既往的睡着懒觉，呼、呼、呼的呼噜声忽弱忽强。

叮叮叮，⏰闹钟响了，Angela丙丙如往常般，闭着眼睛摸到了手机并关了，正准备呼呼大睡。

突然一个激灵，Angela丙丙想到了什么，“今天还有百度和京东的面试，卧*差点睡过头”。

一眨眼的功夫，Angela丙丙已经坐在了，京东和百度一面的会议室了。（我整合了两个公司一面的问题，做了去重）

> 门开了，一个大腹便便，穿着格子衬衣的中年男子，拿着一个满是划痕的mac向你走来，看着快秃顶的头发，心想着肯定是尼玛顶级架构师吧！

> 小伙子自我介绍下，顺便把你做过的项目介绍一下。

面试官您好我叫**Angela丙丙**今年23岁，我来自贵州遵义，毕业于蓝翔技工职业学校电子信息工程专业，今年已经工作两年了，之前有过华为的实习经历，主要在nova团队负责核心组件开发，和团队小伙伴攻坚难题。

现在的公司是天猫国际，主要负责活动小组的研发，担任小组的小组长，在公司一年期间，参加过大大小小活动30多场，活动为公司带来9次过千万的GMV的活动，负责的项目主要是使用了微服务的设计思想和分布式的部署方式。

经常用到的中间件有Redis，RocketMq，ElasticSearch，Logstash，Canal，、Dubbo等等，对Dubbo的源码有着深入的研究和见解，并且仿照Dubbo用Netty也写了一个类Rpc框架，现在已经开源到了GitHub，并且有了2万的Star。

并且会用业余时间撰写技术博客，在各大博客论坛都要 一定的粉丝量。

我对电商所有的业务流程都有一些了解，对线上问题处理以及性能调优都有自己的理解，对业务的研发设计流程也十分熟悉。

因为个人原因选择了离职，如果能有幸加入贵公司一起共事，我想我能第一时间上手并产出结果，以上就是我的自我介绍，谢谢面试官。

Tip：以上自我介绍全部内容都是虚构的，只是给大家一个面试自我介绍的Demo，面试题就是Angela小丙的面试题了。

> **Angela丙丙**？简历上看到你做过项目组长，你在做组长之后做了那些工作，说一下你学到了那些东西。

帅气的面试官您好，进入公司之后，我因为项目调动成为了小组的Leader，这并没有我想的爽，因为之前就是负责自己分内的事情就好了，现在需要有项目的大局观，还有项目进度把控，已经组员的代码和方案的Review。

小组同学都非常优秀，我如果要Hold住他们，就得逼迫自己去不断的学习，所以这段时光非常痛苦，但是得到的收获却是实实在在的。

> 有看过HashMap源码吗？

嗯嗯这个我看过的，因为实际开发过程中，Map还是比较常见的面试题。

> JDK1.7中HashMap的put()方法全过程。

> JDK1.8有那些变化。

> JDK1.7当中HashMap中线程不安全问题有那些？原因分别是什么？

> JDK1.8之后如何链地址法，链表长度是多少的时候会转换成红黑树。

> 节点个数是多少的时候，红黑树会退回链表。

> 为什么会选择8作为链表转红黑树的阈值。

根据泊松分布，在负载因子默认为0.75的时候，单个hash槽内元素个数为8的概率小于百万分之一，所以将7作为一个分水岭，等于7的时候不转换，大于等于8的时候才进行转换，小于等于6的时候就化为链表。

> HashMap与HashTable有什么区别？

> 有没有了解过ConcurrentHashMap？

> JDK1.8之后ConcurrentHashMap如何保证线程安全性？(CAS+synchronized)，这里还顺便问了synchronized和可重入锁的区别。

> 与JDK1.7相比有那些优化?

关于HashMap和ConcurrentHashMap的知识点我写过了，就不做重复工作了，去看看我HashMap和ConcurrentHashMap相关的文章就好了(跟我介绍的内容基本一致，以我为准):

- HashMap
- ConcurrentHashMap&HashTable

> 说到synchronized，说些synchronized加载static关键字前和普通方法前的区别?

Synchronized修饰非静态方法，实际上是对调用该方法的对象加锁，俗称“对象锁”

Synchronized修饰静态方法，实际上是对该类对象加锁，俗称“类锁”。

- 对象锁钥匙只能有一把才能互斥，才能保证共享变量的唯一性
- 在静态方法上的锁，和 实例方法上的锁，默认不是同样的，如果同步需要制定两把锁一样。
- 关于同一个类的方法上的锁，来自于调用该方法的对象，如果调用该方法的对象是相同的，那么锁必然相同，否则就不相同。比如 new A().x() 和 new A().x(),对象不同，锁不同，如果A的单利的，就能互斥。
- 静态方法加锁，能和所有其他静态方法加锁的 进行互斥
- 静态方法加锁，和xx.class 锁效果一样，直接属于类的

> 看你熟悉单例，介绍下单例模式并且说下单例懒汉式和饿汉式的区别？（手写）

单例的介绍：

意图：保证一个类仅有一个实例，并提供一个访问它的全局访问点。

主要解决：一个全局使用的类频繁地创建与销毁。

何时使用：当您想控制实例数目，节省系统资源的时候。

如何解决：判断系统是否已经有这个单例，如果有则返回，如果没有则创建。

关键代码：构造函数是私有的。

应用实例：

- 一个班级只有一个班主任。
- Windows 是多进程多线程的，在操作一个文件的时候，就不可避免地出现多个进程或线程同时操作一个文件的现象，所以所有文件的处理必须通过唯一的实例来进行。
- 一些设备管理器常常设计为单例模式，比如一个电脑有两台打印机，在输出的时候就要处理不能两台打印机打印同一个文件。

优点：

- 在内存里只有一个实例，减少了内存的开销，尤其是频繁的创建和销毁实例（比如管理学院首页页面缓存）。
- 避免对资源的多重占用（比如写文件操作）。

缺点：没有接口，不能继承，与单一职责原则冲突，一个类应该只关心内部逻辑，而不关心外面怎么样来实例化。

使用场景：

- 要求生产唯一序列号。
- WEB 中的计数器，不用每次刷新都在数据库里加一次，用单例先缓存起来。
- 创建的一个对象需要消耗的资源过多，比如 I/O 与数据库的连接等。

注意事项：getInstance() 方法中需要使用同步锁 synchronized (Singleton.class) 防止多线程同时进入造成 instance 被多次实例化。

懒汉

```
public class Singleton {
    private static Singleton instance;
    private Singleton (){}

    public static Singleton getInstance() {
    if (instance == null) {
        instance = new Singleton();
    }
    return instance;
    }
}
```

饿汉

```
public class Singleton {
    private static Singleton instance = new Singleton();
    private Singleton (){}
    public static Singleton getInstance() {
    return instance;
    }
}
```

> 懒汉式下如何保证线程安全?

```
public class Singleton {
    private static Singleton instance;
    private Singleton (){}
    public static synchronized Singleton getInstance() {
    if (instance == null) {
        instance = new Singleton();
    }
    return instance;
    }
}
```

> 创建线程安全的单例有那些实现方法?

双检锁/双重校验锁（DCL，即 double-checked locking）

```
public class Singleton {
    private volatile static Singleton singleton;
    private Singleton (){}
    public static Singleton getSingleton() {
    if (singleton == null) {
        synchronized (Singleton.class) {
        if (singleton == null) {
            singleton = new Singleton();
        }
        }
    }
    return singleton;
    }
}
```

登记式/静态内部类

```
public class Singleton {
    private static class SingletonHolder {
    private static final Singleton INSTANCE = new Singleton();
    }
    private Singleton (){}
    public static final Singleton getInstance() {
    return SingletonHolder.INSTANCE;
    }
}
```

枚举

```
public enum Singleton {
    INSTANCE;
    public void whateverMethod() {
    }
}
```

> 说一下JVM的内存模型？(每一个模块都说)

方法区（Method Area） 方法区主要是放一下类似类定义、常量、编译后的代码、静态变量等，在JDK1.7中，HotSpot VM的实现就是将其放在永久代中，这样的好处就是可以直接使用堆中的GC算法来进行管理，但坏处就是经常会出现内存溢出，即PermGen Space异常，所以在JDK1.8中，HotSpot VM取消了永久代，用元空间取而代之，元空间直接使用本地内存，理论上电脑有多少内存它就可以使用多少内存，所以不会再出现PermGen Space异常。

堆（Heap） 几乎所有对象、数组等都是在此分配内存的，在JVM内存中占的比例也是极大的，也是GC垃圾回收的主要阵地，平时我们说的什么新生代、老年代、永久代也是指的这片区域，至于为什么要进行分代后面会解释。

虚拟机栈（Java Stack） 当JVM在执行方法时，会在此区域中创建一个栈帧来存放方法的各种信息，比如返回值，局部变量表和各种对象引用等，方法开始执行前就先创建栈帧入栈，执行完后就出栈。

本地方法栈（Native Method Stack） 和虚拟机栈类似，不过区别是专门提供给Native方法用的。

程序计数器（Program Counter Register） 占用很小的一片区域，我们知道JVM执行代码是一行一行执行字节码，所以需要一个计数器来记录当前执行的行数。

> 熟不熟悉垃圾回收算法？给我介绍一下垃圾回收算法有哪些？

标记清除

标记-清除算法将垃圾回收分为两个阶段：**标记阶段和清除阶段**。

在标记阶段首先通过**根节点(GC Roots)**，标记所有从根节点开始的对象，未被标记的对象就是未被引用的垃圾对象。然后，在清除阶段，清除所有未被标记的对象。

复制算法

从根集合节点进行扫描，标记出所有的存活对象，并将这些存活的对象复制到一块儿新的内存（图中下边的那一块儿内存）上去，之后将原来的那一块儿内存（图中上边的那一块儿内存）全部回收掉

标记整理

复制算法的高效性是建立在存活对象少、垃圾对象多的前提下的。

这种情况在新生代经常发生，但是在老年代更常见的情况是大部分对象都是存活对象。如果依然使用复制算法，由于存活的对象较多，复制的成本也将很高。

分代收集算法

分代收集算法就是目前虚拟机使用的回收算法，它解决了标记整理不适用于老年代的问题，将内存分为各个年代。一般情况下将堆区划分为老年代（Tenured Generation）和新生代（Young Generation），在堆区之外还有一个代就是永久代（Permanet Generation）。

在不同年代使用不同的算法，从而使用最合适的算法，新生代存活率低，可以使用复制算法。而老年代对象存活率搞，没有额外空间对它进行分配担保，所以只能使用标记清除或者标记整理算法。

> 如何判定一个对象是否应该回收。

为了解决循环引用的问题，java中采取了正向可达的方式，主要是通过 `Roots` 对象作为起点进行搜索，搜索走过的路径称为“引用链”，当一个对象到 `Roots` 没有任何的引用链相连时时，证明此对象不可用，当然被判定为不可达的对象不一定就会成为可回收对象。

被判定为不可达的对象要成为可回收对象必须至少经历两次标记过程，如果在这两次标记过程中仍然没有逃脱成为可回收对象的可能性，则基本上就真的成为可回收对象了，能否被回收其实主要还是要看 `finalize()` 方法有没有与引用链上的对象关联，如果在 `finalize()` 方法中有关联则自救成功，改对象不可被回收，反之如果没有关联则成功被二次标记成功，就可以称为要被回收的垃圾了。

> 除了垃圾回收，还有那些工作会造成CPU负载过高(其实这里给出的是一个场景，就是让描述一下除了垃圾回收之外，还有那些工作会让线上CPU占用到百分之90-100，并且给出排查过程。)。

> 说一下CMS垃圾回收器和G1收集器的特点，和收集过程。

CMS收集器是一种以获取最短回收停顿时间为目标的收集器。基于“标记-清除”算法实现，它的运作过程如下：

- 初始标记
- 并发标记
- 重新标记
- 并发清除

  初始标记、从新标记这两个步骤仍然需要“stop the world”，初始标记仅仅只是标记一下GC Roots能直接关联到的对象，熟读很快，并发标记阶段就是进行GC Roots Tracing，而重新标记阶段则是为了修正并发标记期间因用户程序继续运作而导致标记产生表动的那一部分对象的标记记录，这个阶段的停顿时间一般会比初始标记阶段稍长点，但远比并发标记的时间短。 CMS是一款优秀的收集器，主要优点：并发收集、低停顿。

缺点：

- CMS收集器对CPU资源非常敏感。在并发阶段，它虽然不会导致用户线程停顿，但是会因为占用了一部分线程而导致应用程序变慢，总吞吐量会降低。
- CMS收集器无法处理浮动垃圾，可能会出现“Concurrent Mode Failure（并发模式故障）”失败而导致Full GC产生。

浮动垃圾：由于CMS并发清理阶段用户线程还在运行着，伴随着程序运行自然就会有新的垃圾不断产生，这部分垃圾出现的标记过程之后，CMS无法在当次收集中处理掉它们，只好留待下一次GC中再清理。这些垃圾就是“浮动垃圾”。

- CMS是一款“标记--清除”算法实现的收集器，容易出现大量空间碎片。当空间碎片过多，将会给大对象分配带来很大的麻烦，往往会出现老年代还有很大空间剩余，但是无法找到足够大的连续空间来分配当前对象，不得不提前触发一次Full GC。

G1是一款面向服务端应用的垃圾收集器。G1具备如下特点：

- 并行于并发：G1能充分利用CPU、多核环境下的硬件优势，使用多个CPU（CPU或者CPU核心）来缩短stop-The-World停顿时间。部分其他收集器原本需要停顿Java线程执行的GC动作，G1收集器仍然可以通过并发的方式让java程序继续执行。
- 分代收集：虽然G1可以不需要其他收集器配合就能独立管理整个GC堆，但是还是保留了分代的概念。它能够采用不同的方式去处理新创建的对象和已经存活了一段时间，熬过多次GC的旧对象以获取更好的收集效果。
- 空间整合：与CMS的“标记--清理”算法不同，G1从整体来看是基于“标记整理”算法实现的收集器；从局部上来看是基于“复制”算法实现的。
- 可预测的停顿：这是G1相对于CMS的另一个大优势，降低停顿时间是G1和ＣＭＳ共同的关注点，

但G 1 除了追求低停顿外，还能建立可预测的停顿时间模型，能让使用者明确指定在一个长度为M毫秒的时间片段内，

- G1运作步骤：

1、初始标记；2、并发标记；3、最终标记；4、筛选回收

> String a = “abc” ; 和String b = new String(“abc”) ; 是不是一样的？为什么？ 他们对应的内存空间分别是什么?

不一样

常量池 指的是在编译期确定，并被保存在已编译的字节码文件中的一些数据，它包括类、方法、接口等中的常量，存放字符串常量和基本类型常量（public static final）。 栈（stack）

主要保存基本数据类型（或者叫内置类型）（char、byte、short、int、long、float、double、boolean）和对象的引用，数据可以共享，速度仅次于寄存器（register），快于堆。 所以他们在 == 的时候是false

> 说一下JVM创建对象的过程。

常量池中定位类的符号引用

↓

检查符号引用所代表的类是否已被加载，解析和初始化过 →

↓↓

分配内存（类加载完成后，内存需求确定） ← 加载

↓

根据java堆是否规整（GC方法）选择分配方法

↙ ↘

指针碰撞 空闲列表

↓

分配内存的并发保证（指针更新的原子性）

↙ ↘

CAS+失败重试 按照线程划分在不同的空间中进行TLAB -XX:+UseTLAB -XX:-UseTLAB

↓

内存空间初始化为0值，保证对象的实例字段可以不赋初值就可以使用。

↓

设置对象头信息（Object Header）：引用指针，元数据，hash值，GC分代年龄，锁相关

↓

执行对象方法

> 说一下byte a = 127， byte b = 127; a+=b 和a = a+b的区别分别会出现什么问题。

a+=b 会出现负数。

a=a+b 会直接报错。

强制类型提升造成的

> 是否熟悉mysql? 说一下mysql的隔离级别和对应的问题。

Read Uncommitted（读取未提交内容）

在该隔离级别，所有事务都可以看到其他未提交事务的执行结果。本隔离级别很少用于实际应用，因为它的性能也不比其他级别好多少。读取未提交的数据，也被称之为脏读（Dirty Read）。

Read Committed（读取提交内容）

这是大多数数据库系统的默认隔离级别（但不是MySQL默认的）。它满足了隔离的简单定义：一个事务只能看见已经提交事务所做的改变。这种隔离级别 也支持所谓的不可重复读（Nonrepeatable Read），因为同一事务的其他实例在该实例处理其间可能会有新的commit，所以同一select可能返回不同结果。

Repeatable Read（可重读）

这是MySQL的默认事务隔离级别，它确保同一事务的多个实例在并发读取数据时，会看到同样的数据行。不过理论上，这会导致另一个棘手的问题：幻读 （Phantom Read）。简单的说，幻读指当用户读取某一范围的数据行时，另一个事务又在该范围内插入了新行，当用户再读取该范围的数据行时，会发现有新的“幻影” 行。InnoDB和Falcon存储引擎通过多版本并发控制（MVCC，Multiversion Concurrency Control）机制解决了该问题。

Serializable（可串行化）

这是最高的隔离级别，它通过强制事务排序，使之不可能相互冲突，从而解决幻读问题。简言之，它是在每个读的数据行上加上共享锁。在这个级别，可能导致大量的超时现象和锁竞争。这四种隔离级别采取不同的锁类型来实现，若读取的是同一个数据的话，就容易发生问题。例如：

- 脏读(Drity Read)：某个事务已更新一份数据，另一个事务在此时读取了同一份数据，由于某些原因，前一个RollBack了操作，则后一个事务所读取的数据就会是不正确的。
- 不可重复读(Non-repeatable read):在一个事务的两次查询之中数据不一致，这可能是两次查询过程中间插入了一个事务更新的原有的数据。
- 幻读(Phantom Read):在一个事务的两次查询中数据笔数不一致，例如有一个事务查询了几列(Row)数据，而另一个事务却在此时插入了新的几列数据，先前的事务在接下来的查询中，就有几列数据是未查询出来的，如果此时插入和另外一个事务插入的数据，就会报错。

> 什么是MVCC，主要是为了做什么？

MVCC(Mutil-Version Concurrency Control)，就是多版本并发控制。MVCC 是一种并发控制的方法，一般在数据库管理系统中，实现对数据库的并发访问。

在Mysql的InnoDB引擎中就是指在已提交读(READ COMMITTD)和可重复读(REPEATABLE READ)这两种隔离级别下的事务对于SELECT操作会访问版本链中的记录的过程。

这就使得别的事务可以修改这条记录，反正每次修改都会在版本链中记录。SELECT可以去版本链中拿记录，这就实现了读-写，写-读的并发执行，提升了系统的性能。

> 我们的数据库当中如何做的优化？

- 对查询进行优化，要尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索引。
- 应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描
- 应尽量避免在 where 子句中使用 != 或 <> 操作符，否则将引擎放弃使用索引而进行全表扫描。
- 应尽量避免在 where 子句中使用 or 来连接条件，如果一个字段有索引，一个字段没有索引，将导致引擎放弃使用索引而进行全表扫描
- in 和 not in 也要慎用，否则会导致全表扫描
- like模糊全匹配也将导致全表扫描

> 说一下mybaits和hibernate的区别。

共同点：

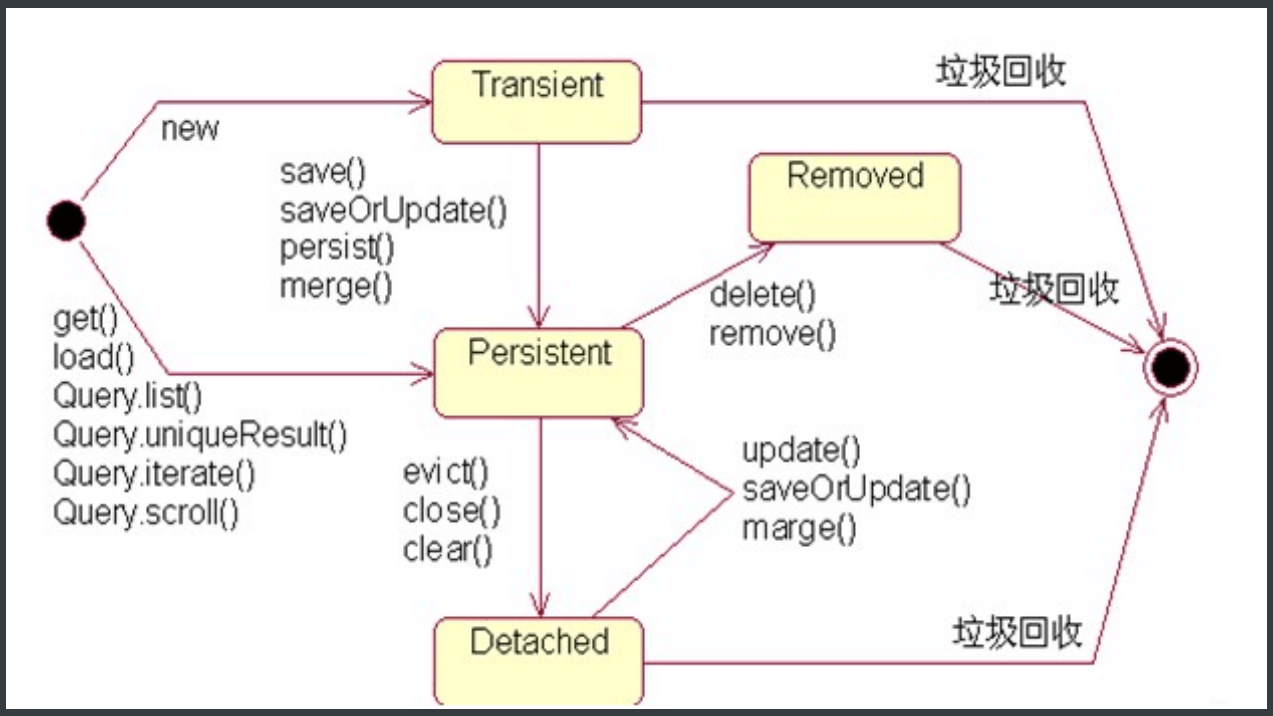
MyBaits和HiBernate都是通过ORM对象关系映射框架，都是持久层数据框架。

不同点：

- HiBernate它重量级的框架，而MyBaits是轻量级的框架。
- HiBernate对JDBC的封装比较深，对开发者些sql的能力不是很高，只需要通过Hql语句操作对象即可完成数据的持久化操作了。
- MyBaits也是对JDBC的封装，但是没有Hibernate那么深，它的sql语句，都再配置里，可以通过重写配置里sql，来实现数据优化，实施起来也比较方便。
- 处理大数据的时候，建议使用MyBaits它优化sql语句更方便。

> Hibernate状态的转换关系。

最新的Hibernate文档中为Hibernate对象定义了四种状态（原来是三种状态，面试的时候基本上问的也是三种状态），分别是：瞬时态（new, or transient）、持久态（managed, or persistent）、游离态（detached）和移除态（removed，以前Hibernate文档中定义的三种状态中没有移除态），如下图所示，就以前的Hibernate文档中移除态被视为是瞬时态。



- 瞬时态：当new一个实体对象后，这个对象处于瞬时态，即这个对象只是一个保存临时数据的内存区域，如果没有变量引用这个对象，则会被JVM的垃圾回收机制回收。

  这个对象所保存的数据与数据库没有任何关系，除非通过Session的save()、saveOrUpdate()、persist()、merge()方法把瞬时态对象与数据库关联，并把数据插入或者更新到数据库，这个对象才转换为持久态对象。

- 持久态：持久态对象的实例在数据库中有对应的记录，并拥有一个持久化标识（ID）。

  对持久态对象进行delete操作后，数据库中对应的记录将被删除，那么持久态对象与数据库记录不再存在对应关系，持久态对象变成移除态（可以视为瞬时态）。

  持久态对象被修改变更后，不会马上同步到数据库，直到数据库事务提交。

- 游离态：当Session进行了close()、clear()、evict()或flush()后，实体对象从持久态变成游离态，对象虽然拥有持久和与数据库对应记录一致的标识值，但是因为对象已经从会话中清除掉，对象不在持久化管理之内，所以处于游离态（也叫脱管态）。

游离态的对象与临时状态对象是十分相似的，只是它还含有持久化标识。

> 说一下Spring的理解，IOC和AOP在项目里是怎么用的。

Spring是一个开源框架，处于MVC模式中的控制层，它能应对需求快速的变化，其主要原因它有一种面向切面编程（AOP）的优势，其次它提升了系统性能，因为通过依赖倒置机制（IOC），系统中用到的对象不是在系统加载时就全部实例化，而是在调用到这个类时才会实例化该类的对象，从而提升了系统性能。

这两个优秀的性能使得Spring受到许多J2EE公司的青睐，如阿里里中使用最多的也是Spring相关技术。

Spring的优点：

- 降低了组件之间的耦合性，实现了软件各层之间的解耦。
- 可以使用容易提供的众多服务，如事务管理，消息服务，日志记录等。
- 容器提供了AOP技术，利用它很容易实现如权限拦截、运行期监控等功能。

Spring中AOP技术是设计模式中的动态代理模式，只需实现jdk提供的动态代理接口InvocationHandler，所有被代理对象的方法都由InvocationHandler接管实际的处理任务。

面向切面编程中还要理解切入点、切面、通知、织入等概念。

> AOP的两种实现方式，并且说一下哪一个效率更高一些，为什么。

两种方式：一种是JDK动态代理，另一种是CGLib的方式。

JDK动态代理具体实现原理：

- 通过实现InvocationHandlet接口创建自己的调用处理器；
- 通过为Proxy类指定ClassLoader对象和一组interface来创建动态代理；
- 通过反射机制获取动态代理类的构造函数，其唯一参数类型就是调用处理器接口类型；
- 通过构造函数创建动态代理类实例，构造时调用处理器对象作为参数参入；

JDK动态代理是面向接口的代理模式，如果被代理目标没有接口那么Spring也无能为力，Spring通过Java的反射机制生产被代理接口的新的匿名实现类，重写了其中AOP的增强方法。

CGLib动态代理：

CGLib是一个强大、高性能的Code生产类库，可以实现运行期动态扩展java类，Spring在运行期间通过CGlib继承要被动态代理的类，重写父类的方法，实现AOP面向切面编程呢。

两者对比：

JDK动态代理是面向接口的。

CGLib动态代理是通过字节码底层继承要代理类来实现（如果被代理类被final关键字所修饰，那么抱歉会失败）。

性能：

关于两者之间的性能的话，JDK动态代理所创建的代理对象，在以前的JDK版本中，性能并不是很高，虽然在高版本中JDK动态代理对象的性能得到了很大的提升，但是他也并不是适用于所有的场景。

主要体现在如下的两个指标中：

- CGLib所创建的动态代理对象在实际运行时候的性能要比JDK动态代理高不少，有研究表明，大概要高10倍；
- 但是CGLib在创建对象的时候所花费的时间却比JDK动态代理要多很多，有研究表明，大概有8倍的差距；
- 因此，对于singleton的代理对象或者具有实例池的代理，因为无需频繁的创建代理对象，所以比较适合采用CGLib动态代理，反正，则比较适用JDK动态代理。

> 说一些Spring的事务传播机制。

传播记住有如下几种：

```
public interface TransactionDefinition {
    int PROPAGATION_REQUIRED = 0;
    int PROPAGATION_SUPPORTS = 1;
    int PROPAGATION_MANDATORY = 2;
    int PROPAGATION_REQUIRES_NEW = 3;
    int PROPAGATION_NOT_SUPPORTED = 4;
    int PROPAGATION_NEVER = 5;
    int PROPAGATION_NESTED = 6;
}
```

- PROPAGATION_REQUIRED：如果当前没有事务，就新建一个事务，如果已经存在一个事务中，加入到这个事务中。这是最常见的选择。
- PROPAGATION_SUPPORTS：支持当前事务，如果当前没有事务，就以非事务方式执行。
- PROPAGATION_MANDATORY：使用当前的事务，如果当前没有事务，就抛出异常。
- PROPAGATION_REQUIRES_NEW：新建事务，如果当前存在事务，把当前事务挂起。
- PROPAGATION_NOT_SUPPORTED：以非事务方式执行操作，如果当前存在事务，就把当前事务挂起。
- PROPAGATION_NEVER：以非事务方式执行，如果当前存在事务，则抛出异常。
- PROPAGATION_NESTED：如果当前存在事务，则在嵌套事务内执行。如果当前没有事务，则执行与PROPAGATION_REQUIRED类似的操作。

常用的主要有Required，RequiresNew，Nested三种。

- Required：简单理解就是事务方法会判断是否存在事务，有事务就用已有的，没有就重新开启一个。
- RequiresNew：简单理解就是开启新事务，若当前已有事务，挂起当前事务。新开启的事务和之前的事务无关，拥有自己的锁和隔离级别，可以独立提交和回滚，内层事务执行期间，外层事务挂起，内层事务执行完成后，外层事务恢复执行。
- Nested：简单理解就是嵌套事务，如果外部事务回滚，则嵌套事务也会回滚！！！外部事务提交的时候，嵌套它才会被提交。嵌套事务回滚不会影响外部事务。子事务是上层事务的嵌套事务，在子事务执行之前会建立savepoint，嵌套事务的回滚会回到这个savepoint，不会造成父事务的回滚。

如果想事务一起执行可以用Required满足大部分场景，如果不想让执行的子事务的结果影响到父事务的提交可以将子事务设置为RequiresNew。

> 你有什么想问我的么？

我们京东\百度平时有技术分享这样的活动么？

> 有的你放心，技术分享我们都会不同的同事轮流进行不同技术栈的分享。

好的，那我没什么问题了，期待能和您一起共事。

总结

所有的题目都是粉丝提供的面试真题，我把所有的题目都查了下答案总结了一下，其实大家可以看出，很多题目都是我平时文章写的内容，我基本上就是跟着面试的思路写的。

而且面试的题目都很基础，是不是觉得京东百度的面试其实也不难，我们好好准备下真心随便进的那种，都是基础，其实平时多留心下，都不需要突击啥的。

点关注，不迷路

好了各位，以上就是这篇文章的全部内容了，我是敖丙，励志做一名让大家都记得住的博主，能看到这里的人呀，都是**人才**。

我后面会每周都更新几篇一线互联网大厂面试和常用技术栈相关的文章，非常感谢**人才**们能看到这里，如果这个文章写得还不错，觉得「敖丙」我**有点东西**的话 **求点赞**👍 **求关注**❤️ **求分享**👥 对暖男我来说真的 **非常有用**！！！

白嫖不好，创作不易，各位的支持和认可，就是我创作的最大动力，我们下篇文章见！

敖丙 | 文 【原创】

如果本篇博客有任何错误，请批评指教，不胜感激！

> 文章每周持续更新，可以微信搜索「 **三太子敖丙** 」第一时间阅读和催更（比博客早一到两篇哟），本文 **GitHub** https://github.com/JavaFamily 已经收录，有一线大厂面试点思维导图，也整理了很多我的文档，欢迎Star和完善，大家面试可以参照考点复习，希望我们一起有点东西。