**Cvte提前批**

**一面（电话）**

1. 自我介绍
2. 介绍你的项目
3. 加密解密了解么？几种算法，讲一下你了解的
4. 多线程了解么？什么是线程安全？
5. 说一个你最熟悉的设计模式
6. 讲一下你项目中用到了哪些设计模式
7. Java的hashmap的原理
8. Hashmap的线程安全性，什么是线程安全的？如何实现线程安全

**二面（视频）**

1. 自我介绍
2. 介绍项目
3. Mysql的数据库引擎，区别特点
4. 设计模式了解？讲一下最熟悉的
5. 写一个单例模式，答主写的是双检查锁单例，问了为什么用Volatile，synchronize移到方法最外面会怎么样？
6. 单例模式在你项目里哪些应用？
7. 数据连接池
8. 对高负载有了解么
9. 你意向的技术方向是哪块？（答主回答的高并发，然后面试官说他是做高负载的）
10. 对高并发有了解么？

**阿里内推**

**一面（电话）**

1. 听说你有博客，博客里大概有什么内容？
2. 项目介绍，最复杂的表
3. Hashmap的原理
4. Hashmap为什么大小是2的幂次

使取模运算变为位运算，提高效率

1. 介绍一下红黑树

节点或红或黑

根节点红，

叶节点红

红节点的子节点是黑色

任意一个节点到其叶节点的所有路径上均有相同数量的黑色节点

1. Arraylist的原理

根据负载因子扩容或缩小

1. 场景题：设计判断论文抄袭的系统
2. 堆排序的原理
3. 抽象工厂和工厂方法模式的区别
4. 工厂模式的思想

为创建对象提供过度接口，以便将创建对象的具体过程屏蔽隔离起来，达到提高灵活性的目的

1. object类你知道的方法

hahsCode toString wait notify notifyAll finalize clone getClass

1. 哪里用到了工厂模式

Hibernate 的sessionFactory

1. Forward和redirect的区别

转发与重定向

**二面（视频）**

1， 自我介绍

2， 项目介绍

3， 项目架构

4， 项目难点

5， Synchronize关键字为什么jdk1.5后效率提高了

引入了读写锁

6， 线程池的使用时的注意事项

线程安全性，最大存活时间

7， Spring中autowire和resourse关键字的区别

Autowire按类型装配，resource按名称装配

8， Hashmap的原理

9， Hashmap的大小为什么指定为2的幂次

10， 讲一下线程状态转移图

11， 消息队列了解么

12， 分布式了解么

1.Integer和int区别

2.在Java中，final修饰的类有什么特点

2.多线程实现方法

3.hashmap原理，链表里面存的是什么？

4.线程池作用

减少创建和销毁线程的次数，每个工作线程可以多次使用

可根据系统情况调整执行的线程数量，防止消耗过多内存

5.垃圾回收

6.Spring Ioc aop概念，ioc注入

Aop：面向切面编程，默认使用动态代理实现

术语：通知：即运行到切点时切面要完成的工作

连接点：即可以织入通知的点

切点：即要织入通知的方法

切面：即切点+通知

**Ioc—Inversion of Control，即“控制反转”，不是什么技术，而是一种设计思想。**在Java开发中，**Ioc意味着将你设计好的对象交给容器控制，而不是传统的在你的对象内部直接控制。**如何理解好Ioc呢？理解好Ioc的关键是要明确“谁控制谁，控制什么，为何是反转（有反转就应该有正转了），哪些方面反转了”，那我们来深入分析一下：

　　●谁控制谁，控制什么：传统Java SE程序设计，我们直接在对象内部通过new进行创建对象，是程序主动去创建依赖对象；而IoC是有专门一个容器来创建这些对象，即由Ioc容器来控制对 象的创建；**谁控制谁？当然是IoC 容器控制了对象；控制什么？那就是主要控制了外部资源获取（不只是对象包括比如文件等）。**

　　●为何是反转，哪些方面反转了：有反转就有正转，传统应用程序是由我们自己在对象中主动控制去直接获取依赖对象，也就是正转；而反转则是由容器来帮忙创建及注入依赖对象；为何是反转？**因为由容器帮我们查找及注入依赖对象，对象只是被动的接受依赖对象，所以是反转；哪些方面反转了？依赖对象的获取被反转了。**

Setter注入，接口注入，构造器注入

7.bean的生命周期

1. 实例化一个Bean，也就是我们通常说的new

2. 按照Spring上下文对实例化的Bean进行配置，也就是IOC注入

3. 如果这个Bean实现了BeanNameAware接口，会调用它实现的setBeanName(String beanId)方法，此处传递的是Spring配置文件中Bean的ID

4. 如果这个Bean实现了BeanFactoryAware接口，会调用它实现的setBeanFactory()，传递的是Spring工厂本身（可以用这个方法获取到其他Bean）

5. 如果这个Bean实现了ApplicationContextAware接口，会调用setApplicationContext(ApplicationContext)方法，传入Spring上下文，该方式同样可以实现步骤4，但比4更好，以为ApplicationContext是BeanFactory的子接口，有更多的实现方法

6. 如果这个Bean关联了BeanPostProcessor接口，将会调用postProcessBeforeInitialization(Object obj, String s)方法，BeanPostProcessor经常被用作是Bean内容的更改，并且由于这个是在Bean初始化结束时调用After方法，也可用于内存或缓存技术

7. 如果这个Bean在Spring配置文件中配置了init-method属性会自动调用其配置的初始化方法

8. 如果这个Bean关联了BeanPostProcessor接口，将会调用postAfterInitialization(Object obj, String s)方法

注意：以上工作完成以后就可以用这个Bean了，那这个Bean是一个single的，所以一般情况下我们调用同一个ID的Bean会是在内容地址相同的实例

9. 当Bean不再需要时，会经过清理阶段，如果Bean实现了DisposableBean接口，会调用其实现的destroy方法

10. 最后，如果这个Bean的Spring配置中配置了destroy-method属性，会自动调用其配置的销毁方法

以上10步骤可以作为面试或者笔试的模板，另外我们这里描述的是应用Spring上下文Bean的生命周期，如果应用Spring的工厂也就是BeanFactory的话去掉第5步就Ok了

8.bean的作用域

* 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
* 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

1. spring中有哪些设计模式，一种或两种

工厂模式，bean工厂

代理模式，aop的动态代理

10.单例模式

11.进程和线程区别

12.死锁？死锁产生的原因？死锁的必要条件？怎么处理死锁？

13.进程间通信方式

15.数据库的事务

16.介绍项目，遇到难题如何处理

17.学的技术

18.计算机网络tcp和udp区别，三次握手，四次挥手

19.排序算法，树的遍历

Mysql部分

Acid特性：

原子性是指事务是一个**不可再分割的工作单元**，事务中的操作要么都发生，要么都不发生。

可采用“**A向B转账**”这个例子来说明解释

在DBMS中，默认情况下**一条SQL就是一个单独事务**，事务是**自动提交**的。只有显式的使用**start transaction**开启一个事务，才能将一个代码块放在事务中执行。

### ****2. 一致性****

一致性是指在**事务开始之前和事务结束以后**，**数据库的完整性约束没有被破坏**。这是说数据库事务不能破坏**关系数据的完整性**以及**业务逻辑上的一致性**。

如A给B转账，不论转账的事务操作是否成功，其两者的存款总额不变（这是业务逻辑的一致性，至于数据库关系约束的完整性就更好理解了）。

保障机制（也从两方面着手）：**数据库层面**会在一个事务执行之前和之后，数据会符合你设置的**约束**（**唯一约束，外键约束,check约束**等)和触发器设置；此外，数据库的内部数据结构（如 B 树索引或双向链表）都必须是正确的。业务的一致性一般由开发人员进行保证，亦可转移至数据库层面。

### ****3. 隔离性****

**多个事务并发访问时，事务之间是隔离的**，一个事务不应该影响其它事务运行效果。

在并发环境中，当**不同的事务同时操纵相同的数据**时，每个事务都有**各自的完整数据空间**。由并发事务所做的修改必须与任何其他并发事务所做的修改隔离。事务查看数据更新时，数据所处的状态要么是另一事务修改它之前的状态，要么是另一事务修改它之后的状态，**事务不会查看到中间状态的数据**。

事务最复杂问题都是由事务隔离性引起的。完全的隔离性是不现实的，完全的隔离性要求数据库同一时间只执行一条事务，这样会严重影响性能。

关于隔离性中的事务隔离等级（事务之间影响），参见相应博文

### ****4. 持久性****

这是最好理解的一个特性：持久性，意味着在事务完成以后，**该事务所对数据库所作的更改便持久的保存在数据库之中，并不会被回滚。**（完成的事务是**系统永久的部分**，对系统的影响是永久性的，该修改即使出现致命的系统故障也将一直保持）

**write ahead logging**：SQL Server中使用了WAL（Write-Ahead Logging）技术来保证事务日志的ACID特性，在数据写入到数据库之前，先写入到日志，再将日志记录变更到存储器中。

隔离级别：

## 1，脏读

　　脏读是指在一个事务处理过程里读取了另一个未提交的事务中的数据。

　　当一个事务正在多次修改某个数据，而在这个事务中这多次的修改都还未提交，这时一个并发的事务来访问该数据，就会造成两个事务得到的数据不一致。例如：用户A向用户B转账100元，对应SQL命令如下

update account set money=money+**100** where name=’B’; (此时A通知B)

update account set money=money - **100** where name=’A’;

　　当只执行第一条SQL时，A通知B查看账户，B发现确实钱已到账（此时即发生了脏读），而之后无论第二条SQL是否执行，只要该事务不提交，则所有操作都将回滚，那么当B以后再次查看账户时就会发现钱其实并没有转。

## 2，不可重复读

　　不可重复读是指在对于数据库中的某个数据，一个事务范围内多次查询却返回了不同的数据值，这是由于在查询间隔，被另一个事务修改并提交了。

　　例如事务T1在读取某一数据，而事务T2立马修改了这个数据并且提交事务给数据库，事务T1再次读取该数据就得到了不同的结果，发送了不可重复读。

　　不可重复读和脏读的区别是，脏读是某一事务读取了另一个事务未提交的脏数据，而不可重复读则是读取了前一事务提交的数据。

　　在某些情况下，不可重复读并不是问题，比如我们多次查询某个数据当然以最后查询得到的结果为主。但在另一些情况下就有可能发生问题，例如对于同一个数据A和B依次查询就可能不同，A和B就可能打起来了……

## 3，虚读(幻读)

　　幻读是事务非独立执行时发生的一种现象。例如事务T1对一个表中所有的行的某个数据项做了从“1”修改为“2”的操作，这时事务T2又对这个表中插入了一行数据项，而这个数据项的数值还是为“1”并且提交给数据库。而操作事务T1的用户如果再查看刚刚修改的数据，会发现还有一行没有修改，其实这行是从事务T2中添加的，就好像产生幻觉一样，这就是发生了幻读。

　　幻读和不可重复读都是读取了另一条已经提交的事务（这点就脏读不同），所不同的是不可重复读查询的都是同一个数据项，而幻读针对的是一批数据整体（比如数据的个数）。

　　现在来看看MySQL数据库为我们提供的四种隔离级别：

　　① Serializable (串行化)：可避免脏读、不可重复读、幻读的发生。

　　② Repeatable read (可重复读)：可避免脏读、不可重复读的发生。

　　③ Read committed (读已提交)：可避免脏读的发生。

　　④ Read uncommitted (读未提交)：最低级别，任何情况都无法保证。

三大范式：

**、1NF(第一范式)**

第一范式是指数据库表中的每一列都是不可分割的基本数据项，同一列中不能有多个值，即实体中的某个属性不能有多个值或者不能有重复的属性。

如果出现重复的属性，就可能需要定义一个新的实体，新的实体由重复的属性构成，新实体与原实体之间为一对多关系。第一范式的模式要求属性值不可再分裂成更小部分，即属性项不能是属性组合或是由一组属性构成。

简而言之，第一范式就是无重复的列。例如，由“职工号”“姓名”“电话号码”组成的表(一个人可能有一部办公电话和一部移动电话)，这时将其规范化为1NF可以将电话号码分为“办公电话”和“移动电话”两个属性，即职工(职工号，姓名，办公电话，移动电话)。

**2、2NF(第二范式)**

第二范式(2NF)是在第一范式(1NF)的基础上建立起来的，即满足第二范式(2NF)必须先满足第一范式(1NF)。第二范式(2NF)要求数据库表中的每个实例或行必须可以被唯一地区分。为实现区分通常需要为表加上一个列，以存储各个实例的唯一标识。

如果关系模型R为第一范式，并且R中的每一个非主属性完全函数依赖于R的某个候选键，则称R为第二范式模式(如果A是关系模式R的候选键的一个属性，则称A是R的主属性，否则称A是R的非主属性)。

例如，在选课关系表(学号，课程号，成绩，学分)，关键字为组合关键字(学号，课程号)，但由于非主属性学分仅依赖于课程号，对关键字(学号，课程号)只是部分依赖，而不是完全依赖，因此此种方式会导致数据冗余以及更新异常等问题，解决办法是将其分为两个关系模式：学生表(学号，课程号，分数)和课程表(课程号，学分)，新关系通过学生表中的外关键字课程号联系，在需要时进行连接。

**3、3NF(第三范式)**

如果关系模型R是第二范式，且每个非主属性都不传递依赖于R的候选键，则称R是第三范式的模式。

以学生表(学号，姓名，课程号，成绩)为例，其中学生姓名无重名，所以该表有两个候选码(学号，课程号)和(姓名，课程号)，故存在函数依赖：学号——>姓名，(学号，课程号)——>成绩，唯一的非主属性成绩对码不存在部分依赖，也不存在传递依赖，所以属性属于第三范式。

乐观锁与悲观锁：

悲观锁，正如其名，它指的是对数据被外界（包括本系统当前的其他事务，以及来自外部系统的事务处理）修改持保守态度(悲观)，因此，在整个数据处理过程中，将数据处于锁定状态。 悲观锁的实现，往往依靠数据库提供的锁机制 （也只有数据库层提供的锁机制才能真正保证数据访问的排他性，否则，即使在本系统中实现了加锁机制，也无法保证外部系统不会修改数据）

在关系数据库管理系统里，乐观并发控制（又名“乐观锁”，Optimistic Concurrency Control，缩写“OCC”）是一种并发控制的方法。它假设多用户并发的事务在处理时不会彼此互相影响，各事务能够在不产生锁的情况下处理各自影响的那部分数据。在提交数据更新之前，每个事务会先检查在该事务读取数据后，有没有其他事务又修改了该数据。如果其他事务有更新的话，正在提交的事务会进行回滚。乐观事务控制最早是由孔祥重（H.T.Kung）教授提出。

Java中锁的状态：

### 成为偏向锁

一个对象刚开始实例化的时候，没有任何线程来访问它的时候。它是可偏向的，意味着，它现在认为只可能有一个线程来访问它，所以当第一个线程来访问它的时候，它会偏向这个线程，此时，对象持有偏向锁。偏向第一个线程，这个线程在修改对象头成为偏向锁的时候使用CAS操作，并将对象头中的ThreadID改成自己的ID，之后再次访问这个对象时，只需要对比ID，不需要再使用CAS在进行操作。

### 成为轻量级锁

一旦有第二个线程访问这个对象，因为偏向锁不会主动释放，所以第二个线程可以看到对象是偏向状态，这时表明在这个对象上已经存在竞争了，检查原来持有该对象锁的线程是否依然存活，如果挂了，则可以将对象变为无锁状态，然后重新偏向新的线程，如果原来的线程依然存活，则马上执行那个线程的操作栈，检查该对象的使用情况，如果仍然需要持有偏向锁，则偏向锁升级为轻量级锁，（**偏向锁就是这个时候升级为轻量级锁的**）。**假如一个对象先被线程1加锁解锁，再被线程2加锁解锁，这过程中没有锁冲突，也一样会发生偏向锁失效，不同的是这回要先退化为无锁的状态，再加轻量锁，如果是线程1持有锁，且2也要争夺偏向锁，则直接到轻量级锁状态**

### 成为重量级锁

轻量级锁认为竞争存在，但是竞争的程度很轻，一般两个线程对于同一个锁的操作都会错开，或者说稍微等待一下（自旋），另一个线程就会释放锁。 但是**当自旋超过一定的次数，或者一个线程在持有锁，一个在自旋，又有第三个来访时，轻量级锁膨胀为重量级锁，重量级锁使除了拥有锁的线程以外的线程都阻塞，防止CPU空转。**

Wait与sleep的区别：

Sleep不会释放锁

Wait的线程必须使用sleep来唤醒

如何停止一个线程：

使用共享变量

自己设置停止方法

等待线程自动终止

调用interrupt方法

自旋锁：不会等待，会一直观测锁的保持者是否已经释放锁

Java有哪几种锁：

## 公平锁/非公平锁

公平锁是指多个线程按照申请锁的顺序来获取锁。  
非公平锁是指多个线程获取锁的顺序并不是按照申请锁的顺序，有可能后申请的线程比先申请的线程优先获取锁。有可能，会造成优先级反转或者饥饿现象。  
对于Java ReentrantLock而言，通过构造函数指定该锁是否是公平锁，默认是非公平锁。非公平锁的优点在于吞吐量比公平锁大。  
对于Synchronized而言，也是一种非公平锁。由于其并不像ReentrantLock是通过AQS的来实现线程调度，所以并没有任何办法使其变成公平锁。

## 可重入锁

可重入锁又名递归锁，是指在同一个线程在外层方法获取锁的时候，在进入内层方法会自动获取锁。说的有点抽象，下面会有一个代码的示例。  
对于Java ReentrantLock而言, 他的名字就可以看出是一个可重入锁，其名字是Re entrant Lock重新进入锁。  
对于Synchronized而言,也是一个可重入锁。可重入锁的一个好处是可一定程度避免死锁。

synchronized void setA() throws Exception{

Thread.sleep(1000);

setB();

}

synchronized void setB() throws Exception{

Thread.sleep(1000);

}

上面的代码就是一个可重入锁的一个特点，如果不是可重入锁的话，setB可能不会被当前线程执行，可能造成死锁。

## 独享锁/共享锁

独享锁是指该锁一次只能被一个线程所持有。  
共享锁是指该锁可被多个线程所持有。

对于Java ReentrantLock而言，其是独享锁。但是对于Lock的另一个实现类ReadWriteLock，其读锁是共享锁，其写锁是独享锁。  
读锁的共享锁可保证并发读是非常高效的，读写，写读 ，写写的过程是互斥的。  
独享锁与共享锁也是通过AQS来实现的，通过实现不同的方法，来实现独享或者共享。  
对于Synchronized而言，当然是独享锁。

## 互斥锁/读写锁

上面讲的独享锁/共享锁就是一种广义的说法，互斥锁/读写锁就是具体的实现。  
互斥锁在Java中的具体实现就是ReentrantLock  
读写锁在Java中的具体实现就是ReadWriteLock

## 乐观锁/悲观锁

乐观锁与悲观锁不是指具体的什么类型的锁，而是指看待并发同步的角度。  
悲观锁认为对于同一个数据的并发操作，一定是会发生修改的，哪怕没有修改，也会认为修改。因此对于同一个数据的并发操作，悲观锁采取加锁的形式。悲观的认为，不加锁的并发操作一定会出问题。  
乐观锁则认为对于同一个数据的并发操作，是不会发生修改的。在更新数据的时候，会采用尝试更新，不断重新的方式更新数据。乐观的认为，不加锁的并发操作是没有事情的。

从上面的描述我们可以看出，悲观锁适合写操作非常多的场景，乐观锁适合读操作非常多的场景，不加锁会带来大量的性能提升。  
悲观锁在Java中的使用，就是利用各种锁。  
乐观锁在Java中的使用，是无锁编程，常常采用的是CAS算法，典型的例子就是原子类，通过CAS自旋实现原子操作的更新。

## 分段锁

分段锁其实是一种锁的设计，并不是具体的一种锁，对于ConcurrentHashMap而言，其并发的实现就是通过分段锁的形式来实现高效的并发操作。  
我们以ConcurrentHashMap来说一下分段锁的含义以及设计思想，ConcurrentHashMap中的分段锁称为Segment，它即类似于HashMap（JDK7与JDK8中HashMap的实现）的结构，即内部拥有一个Entry数组，数组中的每个元素又是一个链表；同时又是一个ReentrantLock（Segment继承了ReentrantLock)。  
当需要put元素的时候，并不是对整个hashmap进行加锁，而是先通过hashcode来知道他要放在那一个分段中，然后对这个分段进行加锁，所以当多线程put的时候，只要不是放在一个分段中，就实现了真正的并行的插入。  
但是，在统计size的时候，可就是获取hashmap全局信息的时候，就需要获取所有的分段锁才能统计。  
分段锁的设计目的是细化锁的粒度，当操作不需要更新整个数组的时候，就仅仅针对数组中的一项进行加锁操作。

## 偏向锁/轻量级锁/重量级锁

这三种锁是指锁的状态，并且是针对Synchronized。在Java 5通过引入锁升级的机制来实现高效Synchronized。这三种锁的状态是通过对象监视器在对象头中的字段来表明的。  
偏向锁是指一段同步代码一直被一个线程所访问，那么该线程会自动获取锁。降低获取锁的代价。  
轻量级锁是指当锁是偏向锁的时候，被另一个线程所访问，偏向锁就会升级为轻量级锁，其他线程会通过自旋的形式尝试获取锁，不会阻塞，提高性能。  
重量级锁是指当锁为轻量级锁的时候，另一个线程虽然是自旋，但自旋不会一直持续下去，当自旋一定次数的时候，还没有获取到锁，就会进入阻塞，该锁膨胀为重量级锁。重量级锁会让其他申请的线程进入阻塞，性能降低。

## 自旋锁

在Java中，自旋锁是指尝试获取锁的线程不会立即阻塞，而是采用循环的方式去尝试获取锁，这样的好处是减少线程上下文切换的消耗，缺点是循环会消耗CPU。  
典型的自旋锁实现的例子，可以参考[自旋锁的实现](http://ifeve.com/java_lock_see1/)

**便利蜂内推**

**一面（电话）**

1. 自我介绍
2. 项目介绍
3. volatile和synchronized

volatile禁止指令重排序，强制每个线程都从主存中读取变量，弱同步机制，如果变量后来的值依赖于原来的值则无法保证线程安全性

1. 来个算法题：一个无序数组，其中一个数字出现的次数大于其他数字之和，求这个数字 （主元素）
2. 答完再来一个：一个数组，有正有负，不改变顺序的情况下，求和最大的最长子序列
3. 项目用到什么数据库？隔离级别？每个隔离级别各做了什么
4. 数据库的索引？mysql不同引擎索引的区别
5. 垃圾回收算法的过程
6. 你了解的垃圾收集器？ Cms收集器的过程
7. 怎样进入老年代？

经历过多次垃圾回收仍然没有被gc

1. 平时用到了什么设计模式？

单例，策略，工厂，创建者，命令

12.讲一下你最熟的两个设计模式

单例，工厂

13.用过什么系统？shell写过脚本吗？

**小米内推**

**一面（电话）**

1. 自我介绍
2. 看你最近博客写的是redis，介绍redis和mysql的区别
3. Redis的应用场景
4. Hashmap的原理
5. Hashmap中jdk1.8之后做了哪些优化

链表换成红黑树，提高了查找效率

1. 垃圾回收的过程
2. Jvm的参数设置
3. 项目中的优化

**金山wps内推**

**一面（电话）**

1. 自我介绍
2. 项目介绍
3. 对Java的面向对象的理解
4. 对java多线程的理解
5. 数据库的索引
6. 数据库的隔离级别
7. 设计模式的理解
8. 讲几个设计模式
9. 对算法有什么了解？答主先回答了动态规划，解释了一下dp的思想

10.快排的思想讲一下

**二面（电话）**

1. 自我介绍
2. 项目介绍
3. Tcp怎么保证可靠传输（中间穿插了好多小问题）
4. Tcp的拥塞控制
5. 让你设计一个即时聊天的系统
6. 支付宝转账，是如何实现，几个小时通知转账成功的（面试官想让回答长连接，答主一直没get到点）
7. 解释一下长连接

**多益网络**

**一面（视频）**

1. 自我介绍
2. 对面向对象的理解
3. 介绍多态
4. Java新建线程有哪几种方式
5. 线程池的作用
6. 看过框架源码么

**拼多多学霸批**

**一面（现场面）**

1. 自我介绍
2. 项目介绍
3. 手撕算法：一棵二叉排序树，给定一个数，找到与给定数差值最小的数
4. 场景题：设计一个系统，解决抢购时所需要的大量的短链接的功能，如何保证高并发，如何设计短链接

**二面（现场面）**

1. 代码量多少
2. 给了一张纸，各种名词，会的写出来
3. 然后给它解释那些会的
4. 设计题：设计一个系统，记录qq用户前一天的登录状态，提供16g内存和2tb的硬盘，要做到查询指定qq号的前一天的登录状态，快速查询O(1)复杂度

**搜狗校招**

**一面（现场）：**

1. 自我介绍
2. 项目介绍
3. 手撕算法：两个排序的数组A和B分别含有m和n个数，找到两个排序数组的中位数，答主用的二分，时间复杂度为O(log (m+n))。结果面试官不满意，让用归并的思想做，时间复杂度其实更高了
4. 介绍网络编程

**涂鸦移动**

**一面（现场）**

1. 自我介绍
2. 项目介绍
3. 数据库的索引原理
4. 索引使用的注意事项
5. 数据库的引擎
6. Java垃圾回收机制
7. Java的finalize，finally，final三个关键字的区别和应用场景
8. String类可以被继承么

手撕算法：假设你是一个专业的窃贼，准备沿着一条街打劫房屋。每个房子都存放着特定金额的钱。你面临的唯一约束条件是：相邻的房子装着相互联系的防盗系统，且 **当相邻的两个房子同一天被打劫时，该系统会自动报警**。

给定一个非负整数列表，表示每个房子中存放的钱， 算一算，如果今晚去打劫，你最多可以得到多少钱 **在不触动报警装置的情况下**。

**二面（电话）**

1. 自我介绍
2. 对游戏的了解
3. 项目介绍
4. 算法题：给一个整数数组，找到两个数使得他们的和等于一个给定的数 *target*。
5. 红黑树
6. Redis的应用

**中国电信it研发中心**

**一面（现场）**

1. 自我介绍
2. 项目介绍
3. 项目里用的什么服务器
4. 自己写一个tomcat服务器，你会怎么写
5. 分布式服务器会出现哪些问题
6. 怎么解决session一致性缓存的问题
7. Redis的优势和特点
8. 一千万用户并发抢购，怎么设计
9. 如果成功的用户有10万，redis存不下怎么处理
10. 你项目中的难点

**二面（现场）**

1. 自我介绍
2. 项目介绍
3. 介绍spring中的熟悉的注解
4. 让你实现autowire注解的功能你会如何实现
5. Redis和mysql的区别
6. Redis的持久化有哪些方式，具体原理

**中兴**

**专业面（现场）**

1. 自我介绍
2. 项目介绍
3. 你了解的设计模式，讲两个
4. Java collection类，集合，讲两个你了解的，说实现原理
5. Java线程池的作用
6. 你觉得你在你实验室处于什么水平

**综合面试（现场）**

说好的综合面试纯聊天呢？

1． 自我介绍

2． 项目介绍

3． 说一下你知道的设计模式

4． 画一个策略模式的uml图

5． Java多线程的理解

6． 内存屏障是什么

7． 数据库索引

8． 项目中的优化

9． 然后开始聊人生

10． 你的缺点，你最不喜欢什么样的人，你的家庭等等

**华为**

**一面（现场）**

1. 自我介绍
2. 项目介绍
3. 项目架构
4. 项目一个完整的执行流程（由于我是搞java的，而面试官是搞c的，所以全程尬聊）
5. 项目优化

**二面（现场）**

1. 自我介绍
2. 项目介绍
3. 怎么管理项目进度
4. 平常的爱好
5. 感觉面试官也不是搞java的，所以又是一阵尬聊

**苏宁内推**

**一面（现场）**

1. 自我介绍
2. 项目介绍
3. 面过哪些公司了
4. 有哪些offer了
5. 聊到多益，于是开始聊最近微博上很火的多益老板
6. 得出结论，我和面试官都觉得多益老板三观有问题，但做游戏就是要偏执的人
7. 你博客主要哪方面的
8. 多线程并发包了解么
9. 讲一下countDownLatch

苏宁聊了20分钟八卦就面完了，一轮技术面

**美团内推**

**一面（电话）**

1. 自我介绍
2. 项目介绍
3. Redis介绍
4. 了解redis源码么
5. 了解redis集群么
6. Hashmap的原理
7. hashmap容量为什么是2的幂次
8. hashset的源码
9. object类你知道的方法
10. hashcode和equals
11. 你重写过hashcode和equals么，要注意什么
12. 假设现在一个学生类，有学号和姓名，我现在hashcode方法重写的时候，只将学号参与计算，会出现什么情况？
13. 往set里面put一个学生对象，然后将这个学生对象的学号改了，再put进去，可以放进set么？并讲出为什么
14. Redis的持久化？有哪些方式，原理是什么？
15. 讲一下稳定的排序算法和不稳定的排序算法
16. 讲一下快速排序的思想

**二面（现场）**

1. 自我介绍
2. 讲一下数据的acid
3. 什么是一致性
4. 什么是隔离性
5. Mysql的隔离级别
6. 每个隔离级别是如何解决
7. Mysql要加上nextkey锁，语句该怎么写
8. Java的内存模型，垃圾回收
9. 线程池的参数
10. 每个参数解释一遍
11. 然后面试官设置了每个参数，给了是个线程，让描述出完整的线程池执行的流程
12. Nio和IO有什么区别
13. Nio和aio的区别
14. Spring的aop怎么实现
15. Spring的aop有哪些实现方式
16. 动态代理的实现方式和区别
17. Linux了解么
18. 怎么查看系统负载
19. Cpu load的参数如果为4，描述一下现在系统处于什么情况
20. Linux，查找磁盘上最大的文件的命令
21. Linux，如何查看系统日志文件
22. 手撕算法：leeetcode原题 22，Generate Parentheses，给定 n 对括号，请写一个函数以将其生成新的括号组合，并返回所有组合结果。

**三面（现场）**

三面没怎么问技术，问了很多技术管理方面的问题

1. 自我介绍
2. 项目介绍
3. 怎么管理项目成员
4. 当意见不一致时，如何沟通并说服开发成员，并举个例子
5. 怎么保证项目的进度
6. 数据库的索引原理
7. 非聚簇索引和聚簇索引
8. 索引的使用注意事项
9. 联合索引
10. 从底层解释最左匹配原则
11. Mysql对联合索引有优化么？会自动调整顺序么？哪个版本开始优化？
12. Redis的应用
13. Redis的持久化的方式和原理
14. 技术选型，一个新技术和一个稳定的旧技术，你会怎么选择，选择的考虑有哪些
15. 说你印象最深的美团点评技术团队的三篇博客
16. 最近在学什么新技术
17. 你是怎么去接触一门新技术的
18. 会看哪些书
19. 怎么选择要看的书

**百度**

**一面（现场）**

1. 自我介绍
2. Java中的多态
3. Object类下的方法
4. Finalize的作用和使用场景
5. Hashcode和equals
6. 为什么要同时重写hashcode和equals
7. 不同时重写会出现哪些问题
8. Hashmap的原理
9. Hashmap如何变线程安全，每种方式的优缺点
10. 垃圾回收机制
11. Jvm的参数你知道的说一下
12. 设计模式了解的说一下啊
13. 手撕一个单例模式
14. 快速排序的思想讲一下
15. 给个数组，模拟快排的过程
16. 手写快排
17. 设计题，一个图书馆管理系统，数据库怎么设计，需求自己定

**二面（现场）**

1. 自我介绍
2. 项目介绍
3. Redis的特点
4. 分布式事务了解么
5. 反爬虫的机制，有哪些方式
6. 手撕算法：反转单链表
7. 手撕算法：实现类似微博子结构的数据结构，输入一系列父子关系，输出一个类似微博评论的父子结构图
8. 手写java多线程
9. 手写java的soeket编程，服务端和客户端
10. 进程间的通信方式
11. 手撕算法： 爬楼梯，写出状态转移方程
12. 智力题：时针分针什么时候重合

**三面（现场）**

由于三面面试官不懂java，我不熟c加加，所以全程尬聊

1. 自我介绍
2. 项目介绍
3. 手撕算法：给定一个数字三角形，找到从顶部到底部的最小路径和。每一步可以移动到下面一行的相邻数字上。
4. 然后继续在这个问题上扩展
5. 求出最短那条的路径
6. 递归求出所有的路径
7. 设计模式讲一下熟悉的
8. 会不会滥用设计模式
9. 多线程条件变量为什么要在while体里
10. 你遇到什么挫折

**腾讯**

**一面（现场）**

1. 自我介绍
2. 项目介绍
3. Hibernate的作用，你的理解
4. 多线程的理解，如何保证线程安全
5. mysql数据库的引擎和区别
6. 场景题：千万用户抢购，如何处理高并发，并且有一个链接，指向前一天抢购成功的用户，如何设计这个系统和数据库
7. 如果后台处理抢购请求的服务器，每次最多承受200的负载，系统该怎么设计
8. 手撕算法：最小公倍数和最大公约数

**二面**

1. 自我介绍
2. 项目介绍
3. 项目里一个完整请求的流程
4. 项目的优化
5. Hibernate和mybatis的区别
6. 为什么用ssh框架
7. Mysql的容灾备份
8. Redis和memcache 的区别
9. 为什么选择redis
10. Java的full gc
11. Full gc会导致什么问题

**招商银行信用卡**

**一面**

1. 自我介绍
2. 分布式事务
3. 设计模式
4. 访问者模式
5. 装饰者模式
6. 有哪些offer
7. 为什么还来我们这

**招银网络科技**

**一面**

1. 自我介绍
2. 写一个两个有序链表合并成一个有序链表
3. 死锁是什么呢
4. 怎么解决死锁
5. http请求流程
6. 为什么负载均衡
7. 怎么实现负载均衡
8. 数据库挂了怎么办？除了热备份还有什么方法
9. 讲讲你对spring的理解，不要把ioc和aop背给我听

**二面**

1. 自我介绍
2. 项目介绍
3. 算法：找出两个数组相等的数，不能用其他数据结构
4. 算法：给定一个数字，一个数组，找出数组中相加等于这两个数的和，不能用数据结构
5. 算法：如何判断一个树是不是另一颗树的子树
6. 如何解决并发访问的错误

**网易**

**一面（现场）**

1. 自我介绍
2. 项目介绍
3. I++操作怎么保证线程安全
4. 场景题：设计一个下单系统，下单成功后可以给用户发优惠券
5. 接上面场景题：服务器挂了，优惠券还没发怎么办
6. 数据库挂了怎么怎么办
7. 怎么保证一致性
8. 分布式事务知道么
9. 介绍分布式事务
10. 你的职业规划

**二面**

1. 自我介绍
2. 项目介绍
3. Nio的原理
4. Channel和buffer
5. directBuffer和buffer的区别
6. nio和aio的区别
7. 锁的实现原理
8. 怎么解决缓存和主存的一致性问题
9. 缓存还没更新到主存，服务器挂了怎么办
10. 数据库挂了怎么办

**Vivo**

**一面**

1. 自我介绍
2. 项目介绍
3. Hibernate的batch有数量限制么
4. Jquery用过么
5. Extjs的优缺点
6. 有没有扩展过extjs
7. 读写锁
8. 什么时候用读锁
9. 什么时候用写锁
10. Cas的原理，使用场景
11. 数据库的瓶颈