1.如何理解oop?

类是面向对象中一个重要的概念。类是具有相同属性和行为特征的对象的抽象，类是对象的概念模型，对象是类的一个实例，

通过类来创建对象，同一类的所有对象具有相同的属性和行为特征。类具有三个基本特征：封装、继承、多态。

**封装**:封装的本义就是隐藏实现细节，提高代码安全性。它有代码实现者来控制要对外暴露什么方法,谁能来调用,是一种基于安全考虑的解决思路

**继承**:主要描述类与类之间的关系,子类能调用父类的非private修饰的成员，同时还可以自己添加一些新的成员，扩充父类，甚至重写父类已有的方法，更其表现符合子类的特征。让子类的表现更独特，更专业。

**多态**:从一定角度来看，封装和继承几乎都是为多态而准备的,父类引用指向子类对象,多态是同一个行为具有多个不同表现形式或形态的能力。多态性是对象多种表现形式的体现,类中多个方法的重载叫多态，父子类中方法的覆盖也叫多态。

**oop之抽象**:用abstract 进行修饰的内容我们称之为抽象,abstract可以使用在类和方法中,抽象类可以没有抽象方法,但是不能new,也就是不能实例化

抽象方法所在的类必须是抽象类

如何理解抽象?

抽象方法的主要作用就是声明要求，具体实现由子类实现，自己不做具体实现,也可以把抽象方法理解成为规则，在继承关系中，所有非抽象子类必须遵守，必须重写

java中==和equals的区别,equals和hashCode的区别?

==是运算符，用于比较两个变量是否相等。

equals，是Objec类的方法，用于比较两个对象是否相等，默认Object类的equals方法是比较两个对象的地址，跟==的结果一样。Object的equals方法如下：

public boolean equals(Object obj) {

return (this == obj);

}

hashCode也是Object类的一个方法。返回一个离散的int型整数。在集合类操作中使用，为了提高查询速度。（HashMap，HashSet等）

java中的数据类型，可分为两类：

1.基本数据类型，也称原始数据类型。byte,short,char,int,long,float,double,boolean

他们之间的比较，应用双等号（==）,比较的是他们的值。

2.复合数据类型(类)

当他们用（==）进行比较的时候，比较的是他们在内存中的存放地址，所以，除非是同一个new出来的对象，他们的比较后的结果为true，否则比较后结果为false。 JAVA当中所有的类都是继承于Object这个基类的，在Object中的基类中定义了一个equals的方法，这个方法的初始行为是比较对象的内存地 址，但在一些类库当中这个方法被覆盖掉了，如String,Integer,Date在这些类当中equals有其自身的实现，而不再是比较类在堆内存中的存放地址了。

对于复合数据类型之间进行equals比较，在没有覆写equals方法的情况下，他们之间的比较还是基于他们在内存中的存放位置的地址值的，因为Object的equals方法也是用双等号（==）进行比较的，所以比较后的结果跟双等号（==）的结果相同。

如果两个对象根据equals()方法比较是相等的，那么调用这两个对象中任意一个对象的hashCode方法都必须产生同样的整数结果。

如果两个对象根据equals()方法比较是不相等的，那么调用这两个对象中任意一个对象的hashCode方法，则可能产生相同的整数结果

从而在集合操作的时候有如下规则：

将对象放入到集合中时，首先判断要放入对象的hashcode值与集合中的任意一个元素的hashcode值是否相等，如果不相等直接将该对象放入集合中。如果hashcode值相等，然后再通过equals方法判断要放入对象与集合中的任意一个对象是否相等，如果equals判断不相等，直接将该元素放入到集合中，否则不放入。

回过来说get的时候，HashMap也先调key.hashCode()算出数组下标，然后看equals如果是true就是找到了，所以就涉及了equals。

JVM,调优,类加载机制,JMM,gc?

java中常说的堆和栈分别是什么数据结构,另外,为什么要分为堆和栈存储数据?

TCP/IP:

Tcp/ip是协议族按照层次由上到下,层层包装

最上面的就是应用层,这里有http,ftp,telnet等我们熟悉的协议

第二层则是传输层,著名的TCP/UDP协议就是这个层次的

第三层是网络层,IP协议就是在这一层,它负责对数据加上IP地址和其他数据,以确定传输的目标,

第四层是数据链路层,这个层次为待传送的数据加入一个以太网协议头,并进行CRC编码,为最后的数据传输做准备

再往下就是硬件层,不再说,因为它和tcp/ip协议的编写者没关系

IP想像成一种高速公路，它允许其它协议在上面行驶并找到到其它电脑的出口。TCP和UDP是高速公路上的“卡车”，它们携带的货物就是像HTTP，文件传输协议FTP这样的协议等。

TCP三次握手,四次挥手?

Object的方法有哪些?

1. finalize()-对象被垃圾回收器回收时调用的:可以引出垃圾回收gc
2. clone()-拷贝一个对象副本出来:可以回引出设计模式:享元模式
3. getClass()-返回对象运行时类:可以引出反射
4. toString()
5. wait()-表示持有对象锁的线程A准备释放对象锁权限，释放cpu资源并进入等待。需要notify()或者notifyAll()才能唤醒,但也可以设置超时时间,超过时间自动唤醒

* notify()-表示持有对象锁的线程A准备释放对象锁权限，通知jvm唤醒某个竞争该对象锁的线程X。线程A synchronized 代码作用域结束后，线程X直接获得对象锁权限，其他竞争线程继续等待(即使线程X同步完毕，释放对象锁，其他竞争线程仍然等待，直至有新的notify ,notifyAll被调用)。
* notifyAll 表示持有对象锁的线程A准备释放对象锁权限，通知jvm唤醒所有竞争该对象锁的线程，线程A synchronized 代码作用域结束后，jvm通过算法将对象锁权限指派给某个线程X，所有被唤醒的线程不再等待。线程X synchronized 代码作用域结束后，之前所有被唤醒的线程都有可能获得该对象锁权限，这个由JVM算法决定。

Spring的IOC和AOP实现原理?

SpringMVC工作原理?

java AIO,BIO,NIO

Spring事务(隔离级别,传播行为)

ArrayList,Vector,LinkedList,HashSet,TreeSet,HashMap,HashTable,ConcurrentHashMap底层实现,相互比较

HashMap底层采用数组+链表的数据存储方式,数组是hashmap的主体, **链表则是主要为了解决哈希冲突而存在的**,为什么要采用这种方式呢,是因为哈希冲突,不管再好的哈希函数也很难保证不会产生哈希冲突,说一下存数据,先根据key的哈希值去找Entry的数组下标所在位置,如果该位置没有数据,直接把该数据放到该位置,如果已经有了数据,在去比较key的equals方法,如果不相等,则将该数据放入到该链表的下一个节点上,取数据也是一样的原理,hashmap的链表越少,性能越好,所以就有一个概念,是以空间换时间,对hashmap进行扩容,减少哈希碰撞,hashmap初始容量是16,负载因子是0.75,所以负载因子越少,就越能减少链表,性能也会越好,但这也有弊端,就很容易出现无效扩容

比较一下hashmap和hashtable->hashmap的key,value可以允许为null值,但key为null只准有一个,但hashtable的key和value都不能为null,hashtable的初始容量是11

从线程安全性来说的话,hashmap不是线程安全的,在多线程环境下,需要手动进行同步处理

Hashtable是线程安全的,它采用的同步机制是使用synchronized锁住整张hash表让线程独占,ConcurrentHashmap也是线程安全的,它J.U.C下的AQS并发组件,底层采用多段数组+链表的数据结构,它的Entry的value变量也是volatile的,所以能够保证读取到最新的值,它采用了锁分离技术,锁住一段数组,不影响其他线程并发访问,ConcurrentHashmap在插入数据之前检测需不需要扩容,能够很好的避免无效扩容

线程安全方面:vector是线程安全的,底层采用synchronized同步,但也不是绝对安全的

ArrayList和LinkedList是线程不安全的

Arrylist和vector底层都是采用动态数组的数据结构,通过索引读取数据,包括在删除数组最后元素或在最后增加元素速度都比较快,但是如果在中间插入或者删除元素,速度较慢,因为要移动部分数据,arraylist扩容是原来的1.5倍,vector是原来的2倍,但是vector可以设置容量增量,arraylist不行

Linkedlist增加删除元素比较快,但是查找速度比较慢,因为要从头开始查找,移动指针

Hashset元素的位置是通过hash值来决定的,所以不具有排序功能,hashset可以放入null,但只能放入一个null,hashset不能放入重复元素,判断重复的方法是调用该对象的equals方法,然后在调用hashcode方法,所以比较对象时,要重写equals和hashCode方法

Treeset可以保证放入的元素是有序的,去重的标准是equals方法和compareTo方法

说说http,https

HTTP是一个应用层协议，由请求和响应构成，是一个标准的客户端服务器模型

工作流程:

1）首先客户机与服务器需要建立连接。只要单击某个超级链接，HTTP的工作开始。

2）建立连接后，客户机发送一个请求给服务器，请求方式的格式为：统一资源标识符（URL）、协议版本号，后边是MIME信息包括请求修饰符、客户机信息和可能的内容。

3）服务器接到请求后，给予相应的响应信息，其格式为一个状态行，包括信息的协议版本号、一个成功或错误的代码，后边是MIME信息包括服务器信息、实体信息和可能的内容。

4）客户端接收服务器所返回的信息通过浏览器显示在用户的显示屏上，然后客户机与服务器断开连接。

如果在以上过程中的某一步出现错误，那么产生错误的信息将返回到客户端，有显示屏输出。对于用户来说，这些过程是由HTTP自己完成的，用户只要用鼠标点击，等待信息显示就可以了。

信号量是什么，怎么使用?volatile关键字是什么？

Java中HashMap的key值要是为类对象则该类需要满足什么条件？

类必须重写equals()和hashCode()方法

并且要保证equals相等的两个对象要保证hashCode返回的hash值也是相等的

DB的四大特性和事务的隔离级别

1. 原子性:原子性是指事务所包含的所有操作要么全部成功,要么全部失败回滚
2. 一致性:一致性是指事务必须是数据库从一个一致性状态变换到另一个一致性状态,也就是说一个事务在执行之前和执行之后必须处于一致性
3. 隔离性:隔离性是指当多个用户并发访问数据库时,比如操作同一张表,数据库为每一个用户开启一个事务,不能被其他事务所干扰,多个并发事务之间要隔离
4. 持久性:持久性是指一个事务一旦提交了,那么这个改变就是永久的,即便在数据库系统出现故障,也不会丢失已经提交的事务

**Mysql数据库提供的四种隔离级别**

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

1）其他事务读未提交数据，出现脏读；  
2）如果一个事务已经开始写数据，则另外一个事务则不允许同时进行写操作，但允许其他事务读此行数据。该隔离级别可以通过“排他写锁”实现。  
3）避免了更新丢失，却可能出现脏读。也就是说事务B读取到了事务A未提交的数据。  
（读未提交：一个事务写数据时，只允许其他事务对这行数据进行读，所以会出现脏读，事务T1读取T2未提交的数据）

Read committed (读已提交)：可避免脏读的发生.

1）允许写事务，所以会出现不可重复读  
2）读取数据的事务允许其他事务继续访问该行数据，但是未提交的写事务将会禁止其他事务访问该行。  
3）该隔离级别避免了脏读，但是却可能出现不可重复读。事务A事先读取了数据，事务B紧接了更新了数据，并提交了事务，而事务A再次读取该数据时，数据已经发生了改变。  
（读已提交：读取数据的事务允许其他事务进行操作，避免了脏读，但是会出现不可重复读，事务T1读取数据，T2紧接着更新数据并提交数据，事务T1再次读取数据的时候，和第一次读的不一样。即虚读）

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

1）禁止写事务；  
2）读取数据的事务将会禁止写事务（但允许读事务），写事务则禁止任何其他事务。  
3）避免了不可重复读取和脏读，但是有时可能出现幻读。这可以通过“共享读锁”和“排他写锁”实现。  
（可重复读：读事务会禁止所有的写事务，但是允许读事务，避免了不可重复读和脏读，但是会出现幻读，即第二次查询数据时会包含第一次查询中未出现的数据）

Serializable (串行化)：可避免脏读、不可重复读、幻读的发生。

1）禁止任何事务，一个一个进行；  
2）提供严格的事务隔离。它要求事务序列化执行，事务只能一个接着一个地执行，但不能并发执行。如果仅仅通过“行级锁”是无法实现事务序列化的，必须通过其他机制保证新插入的数据不会被刚执行查询操作的事务访问到。  
3）序列化是最高的事务隔离级别，同时代价也花费最高，性能很低，一般很少使用，在该级别下，事务顺序执行，不仅可以避免脏读、不可重复读，还避免了幻读。

以上四种隔离级别最高的是Serializable级别，最低的是Read uncommitted级别，当然级别越高，执行效率就越低。像Serializable这样的级别，就是以锁表的方式(类似于Java多线程中的锁)使得其他的线程只能在锁外等待，所以平时选用何种隔离级别应该根据实际情况。在MySQL数据库中默认的隔离级别为Repeatable read (可重复读)。

在MySQL数据库中，支持上面四种隔离级别，默认的为Repeatable read (可重复读)；而在Oracle数据库中，只支持Serializable (串行化)级别和Read committed (读已提交)这两种级别，其中默认的为Read committed级别。

**事务并发调度引起的问题:**

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。
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。
3. 虚读(幻读): 幻读是事务非独立执行时发生的一种现象。

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）。

数据库的锁：行锁，表锁,乐观锁,悲观锁

mysql调优,索引,关键字的使用比较(having,exist,in)

mybatis中#和$使用区别

mq:rabbitMQ,kafka

nosql:redis,memcache

常用的设计模式?

单例模式:具体的几种写法

工厂模式:简单工厂模式,工厂方法模式,抽象工厂模式

工厂模式是将产品实例化的过程封装起来,客户端可以使用,但不知道具体的实现细节,使客户端和产品解耦

简单工厂模式缺点:扩展性比较差,如果想增加一个产品还要

策略模式:

观察者模式:

单例模式几种写法,相互比较

1. 懒汉式:未对静态域进行初始化,单例对象在第一次使用时被创建,不是线程安全的
2. 饿汉式:对静态域进行初始化,单例对象在类被加载时被创建,这种要基于私有构造内部不复杂,并且这个类肯定能被用到的情况下使用,不然会造成资源的浪费,效率较低
3. 懒汉式:静态工厂方法通过加入synchronized关键字进行描述,保证同一时间只有一个线程能访问此方法保证线程安全
4. 懒汉式:double check双重检测机制,但由于指令重排序,并不是线程安全的,如果要保证线程安全,对声明的静态域做volatile描述
5. 枚举模式:是最安全的

springboot优缺点

CAS的实现原理以及问题

websocket原理: