7­­­东方财富的面试

1. 对于面向对象的理解

答：万物皆对象，任何物体都可以归于一类，每一个个体都是一类事物的实例。面向对象编程是将功能通过对象来实现，让对象实现具体的细节。

类：一类事物的抽象，比如人类，而人类中的你我他，则是类的实例，是一个个体。

对象：类的实例 .

面向对象的三大特征：

封装：将对象的属性和行为隐藏起来，而将对象的数学和行为隐藏起来的载体就是类，类通常对客户隐藏起实现的细节，仅对外提供公共方法的访问。将变化隔离，提高了安全性

继承：一个类从现有的类中派生叫做继承，子类可以继承父类的属性和方法，可以定义自己的属性和方法，提高了扩展性，子类取不到private的方法，不能覆盖final修饰的方法，方法重写，子类的访问限定符范围应大于等于父类

多态：同一个操作作用于不同的对象有不同的结果，比如在eclipse中Ctrl+Alt+下，是复制一行，而在VisualCode中则是实现多行同时输入。

在程序运行中才知道要执行哪一个类中的方法

多态的前提，继承，父类引用指向子类，方法的重写，

多态的好处：扩展性，替换性，接口性，灵活，简化

1. 面向对象编程

面向对象的程序是有对象组成的，每个对象包含对用户公开的特定功能部分和隐藏的实现部分。

一个程序员的工作方式应该是：先考虑如何组织数据，再考虑如何操作数据的算法。

其中有两个最重要的概念：

类：一类事物的抽象，是构造对象的模板。

对象：类的实例，是个体。

面向对象的三大特征：

封装：数据隐藏，对对象的使用者隐藏了数据，仅通过公共方法和其他对象进行数据交互。给对象赋予了黑盒的特征，提高了可靠性。

继承：一个类从现有的类中派生叫继承，子类可以复用 父类的属性和方法，还可以定义自己的属性和方法，具有很强的扩展性。需要注意的是private的修饰的属性和方法，子类不能获取到，因为父类将其隐藏了。在重写方法的时候，访问限定符一定要注意，调用子类的构造方法是，会先调用父类的默认无参的构造方法，用final修饰的类不被覆盖。

多态：同一个操作对于不同的对象会有不同的响应，比如在eclipse中Ctrl+Alt+下，是复制一行，而在VisualCode中则是实现多行同时输入。

它的前提是：继承，重写，父类引用指向子类

1. 讲一讲你对线程安全的理解

答：当多线程访问某个对象时，无论线程怎么交替执行，在不增加额外的同步下，得到的结果是正确的，那么我们可以说这个类是线程安全

1. 谈谈对同步关键字synchronized,Volatile,lock的区别和理解

Volatile：保证了线程的可见性和有序性，如果一个共享变量被volatile修饰，那么一个线程修改了这个变量之后，（直接刷新到主内存中）其他线程都会立马得知，但不意味着，被它修饰就是线程安全的。

Synchronized：被Synchronized修饰的代码块，可以防止被多个线程同时执行，必须一个线程执行完后，其他线程才可以执行。保证了同一时刻，只有一个线程可以操作这个同步代码块，保证了线程的原子性，有序性，可见性

Volatile与Synchronized的区别：

1. volatile不会发生阻塞，而Synchronized可能会线程阻塞
2. volatile只能修饰变量，而Synchronized可能修饰方法和代码块
3. volatile不能保证原子性，而Synchronized可以

lock：为了提高同步的效率，因为Synchronized修饰一个代码块的话，一个线程获得了当前代码块的锁，那么其它线程只能等待，就像多线程执行读写操作，读和写可能会发生线程不安全，但同时读操作不会，但Synchronized一个在读的话，其它线程只能等待，效率太低，要完成多个线程只是进行读操作是，线程不会冲突，lock可以办到

可以知道是否成功获得锁，可以让等待锁的线程响应中断，线程竞争激烈的时候，lock优于Synchronized。

Lock如果没有主动去释放锁，会造成死锁，而Synchronized不会

1. 线程的原子性，有序性和可见性

原子性：线程的多个操作是一个整体，要么不执行，要么全部执行

有序性：禁止进行指令从排序，指令重排序可以提高执行效率，但可能会导致多线程并发的正确性

可见性：一个线程一旦修改了共享变量的值，会立马刷新到主内存，其他线程会立刻得知。

1. UDP与TCP的区别

TCP：传输控制协议，面向连接，可靠性高，数据包按顺序准确送达，开销大

UDP：使用者数据元协议，不面向连接，可靠性低，不能保证数据包准确送到，是否接受到

1. 三次握手和四次挥手

三次握手：

四次祸首：

A --请求--> B

A <--确认-- B

A <--请求— B

A –-确认--> B

三次握手的中间两个步骤可以合并成一次，但四次挥手不行，因为四次挥手发送的FIN保温仅仅表示对方不在发送数据了但还是能接收数据，所以要等自己这边发出FIN之后，才能close

三次握手：1.Client向Server发送，连接请求

2.Server向Client发送，确认连接，同时发送你准备好了没

3.Client向Server发送，我准备好了---连接

四次挥手：可以是Server或Client先发送请求

1. Client向Server发送，我数据发送完了
2. Server向Client发送，好的，（但服务端可能没发送完，所以还不能断开连接）
3. Server向Client发送，我数据发送完了
4. Client向Server发送，我也接收完了，断开连接吧----TCP连接断开
5. 为什么TCP连接是三次握手，而关闭是四次挥手？

SYN：同步，建立联机 ACK：应答，确认

FIN：结束

答：因为ACK是确认包，可以和SYN+ACK一起发送，

但关闭连接时，当服务端或客户端收到FIN报文，很可能不会关闭套接字，因为自己可能没有发送完，所以只能回复ACK确认“你的FIN报文收到了”，等到数据发送完毕，在发送FIN报文，所以不能一起发送，客户端接收到FIN报文后，发送确认，断开连接

1. 为什么TCP连接是三次握手，而不是俩次握手？

答：既要确认双方都要做好发送数据的准备，还要允许双方就序列号进行协商，两次握手的话，可能会导致数据丢失和死锁的可能。

1. JVM的内存模型

程序计数器：当前线程执行字节码的指示器，实现代码的流程控制，比如线程切换回来之后，要从之前地方继续执行—线程私有

堆：线程共享的区域，几乎所有的对象都在这里分配内存，也是垃圾收集器的主要场所

方法区：线程共享的区域，用来存放类的信息，常量，静态变量，及时编译器的代码，是堆的一个逻辑部分

虚拟机栈：线程私有的部分，用来存放局部变量表，操作数栈等等，

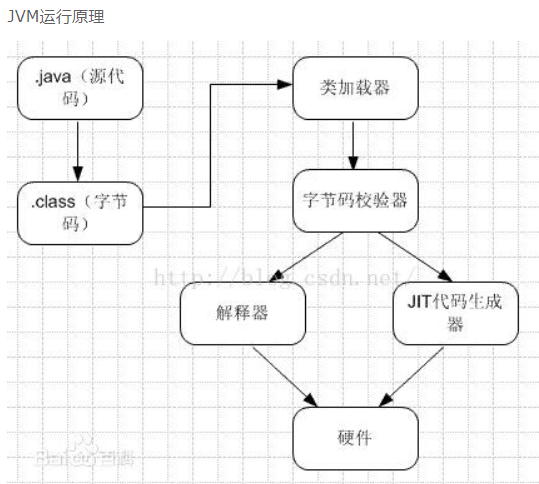
本地方法栈：和虚拟机栈类似，只是虚拟机栈执行的是java的方法，本地方法栈执行的是本地方法、

运行时常量区：存储的是字符串常量和基本类型常量

两个异常：StackOverFlowError表示线程申请的栈超过了实现预定好的栈的最大深度

OutOfMemoryError是指当线程申请栈发现栈满了，内存用光了。

JIT：即时编译器，对于频繁执行的代码，比解释器更加快速



为什么不是所有的对象都在堆内存中分配空间？

随着JIT编译器的发展，在编译期间，如果JIT经过逃逸分析，发现有些对象没有逃逸出方法，那么有可能堆内存分配会被优化成栈内存分配。

1. 反射讲解一下

答：Java的反射机制在程序的运行中，对于一个类，我们可以知道它的属性和方法，对于一个对象，我们可以调用这个类的属性和方法，这就是java的反射

作用：在运行时调用任意一个对象的属性和方法，生成动态代理

动态代理：就是根据对象在内存中加载的Class类创建运行时类对象，从而调用代理类方法和属性

1. 那些框架用到了反射？

答：AOP，Spring，Hiberbate。

1. start()和run()方法的理解?

答：start()是启动线程，run()是运行线程，执行线程里面的方法，

Start()是线程处于就绪状态，一旦得到CPU时间片，线程就会执行run（）方法，run方法结束，线程也就终止了。

1. wait()和sleep()的区别

答：sleep()使线程处于睡眠状态，并让出cpu时间片，但并不会释放锁，到指定时间后会继续执行

Wait()使线程处于等待状态，让出执行时间，只有notify()将其唤醒后，才可以继续竞争锁，只能处于同步代码块中

1. 为什么HashMap线程是线程不安全的？

答：数据丢失和死循环

JDK1.8之前，a.如果多个线程同时调用put()方法添加元素，正好添加元素的hash值是一样的，那么肯定会添加到同一位置，其中一个会被覆盖 b.如果多个线程同时检测到元素个数超过了数组大小\*负载因子，这样的话，多个线程会同时进行扩容，都在重新计算元素位置，但只有一个线程新建的newTable赋给原数组，其他线程都会丢失，如果有的线程刚刚开始扩容（transfer）的话，那么它操作的数组就是已经赋值了的数组，会引起死循环问题

Jdk1.8之后，引进了红黑树的概念，不会发生死循环问题，但会发生数据丢失

1. 那么如何实现HashMap的线程安全呢？

答：a 使用HashTable,将方法加上了同步处理，当一个线程在对表进行操作的的时候，其他线程只能等待，效率很低。

b 使用Collections，返回一个同步的Map，底层封装了HashMap的所有方法，全部加上了同步关键字

c ConcurrentHashMap JDK1.8之前采用的是锁分段技术，将每一个槽进行加锁，在多线程操作不同的槽时，自然不会出现线程安全问题，在同时操作同一个槽时，同一时刻只有一个线程可以获得锁，其它线程只能等待。

Jdk1.8之后，采用的是CAS无锁算法和Synchronized将每个槽的节点锁住，还采用了数组+链表+红黑树。

1. 单例设计模式

答：有时候，有一些对象，我们只需要一个，比如线程池

好处：a对于频繁使用的对象，可以省略创建对象的花销  
 那为什么不适用全局变量确保一个类只有一个实例呢？因为静态变量也可以保证该类只有一个实例，只要程序加载了类的字节码，那么不用创建任何实例，静态变量都会被分配空间，如果全程没用到这个对象，那么资源就浪费了。单例模式的话，只有我们需要使用时才创建对象对象，。而且大量的使用全局变量，会给程序的调试，维护带来很大的负担

1. 工厂模式

答：在基类中创建一个接口，让子类决定实例化那一个类。工厂方法让一个类的实例化延迟到子类中进行。就像饭店一样，客户不会关心饭菜是怎么做的。

为什么要使用工厂模式？

A 解耦：把对象的创建和使用的过程分开

B 降低代码的重复性：如果某个对象的创建很复杂的话，而且很多地方会用到，那么就会有很多重复的代码

C 降低维护成本：创建过程都由工厂统一管理，当需要修改的时候，不需要逐个修改，在工厂中修改即可，降低维护成本。

工厂模式有哪些？

A 简单工厂：将对象的实现细节隐藏的起来，只要传入相应的字段，就可以拿到对象—还可以利用反射—违反了开放-封闭原则，在新加入功能的时候，要在工厂类中修改代码，不好维护

B 一般工厂：让子类决定实例化哪一个类，没有违反开放-封闭原则，新增一个产品的时候，只需要增加一个产品类和工厂类。但生产的产品过用单一

C 抽象工厂，由来一个产品族的概念，生产一整套的产品，这些产品必须是有关系或右依赖的，比如枪，子弹

接口 枪 子弹

接口的实现类 AK、M4 AK子弹和M4子弹

工厂接口 生产枪 生产子弹

工厂 生产AK和AK子弹的工厂 生产M4和m4子弹的工厂

1. 观察者模式

答：在对象之间定义一对多的依赖，当一个对象的状态改变的时候，依赖它的对象都会收到通知并更新

最好的例子就是微信公众号，公众号推送消息，关注了的客户会收到消息，而没关注的则不会收到。

为什么要使用观察者模式？

A 一个对象的数据更新时，需要通知其他对象，而又不希望对象之间紧密耦合，就可以使用观察者模式

B 一个对象的数据更新时，其他对象的数据也都要更新，但又不知道有几个对象。

在哪用到了观察者模式？

我在自己设计一个计算器的过程中用到了观察者模式，

有视图和模型，通过控制器来实现交互

1. 建造者模式

答：一种对象创建模式。用户只需要指定类型和内容就可以生成一个复杂的对象，对象的构建过程隐藏了起来。

为什么要使用建造者模式？

A 将产品本身和产品的创建分离开来，使相同的创建过程，可以产生不同的对象

B 将产品的创建细节分解在不同的方法中，创建过程更加清晰，方便程序进行控制

C 如果要其他的类型的电脑，只需要增加一个具体的建造者，符合开闭原则

缺点：产品差异大或产品内部很复杂，不能使用建造者模式

1. 工厂模式和建造者模式的区别？

答：工厂模式实现的是对产品家族的创建，可以生成不同组合的产品，比如宝马卡车，奥迪卡车，宝马轿车，不关心对像的创建过程。

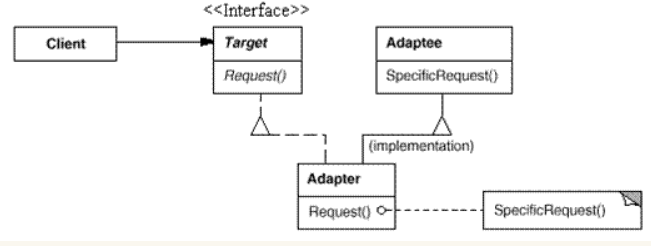
建造者模式：关心产品的创建过程—构建，通过指定的产品的配件样式来，组装零配件来创建产品，比如，生成电脑，cpu我要什么什么样的，显卡我要什么什么样的。

1. 适配器模式

答：挂羊头卖狗肉。，主要是解决接口的不兼容问题，这是我认为的，当我们要使用一个已存在的类，而接口不符合我们的需求，一般来说，不能在接口中添加方法，因为可能一旦添加，其他实现了这个接口的类，都要做出修改，在程序设计中，这是不允许的。所以我们需要添加一个适配器，让其具有个这个功能。建议是非不得已的情况下，不使用适配器。会使内部非常混乱，明明调用的是A接口，在内部却被替换成B接口。

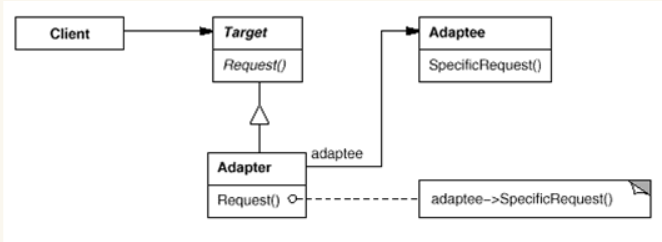
适配器：有新的需求，接口不能实现，我自己适配一个，（接口中的方法我还是要调用）

A 类的适配器模式



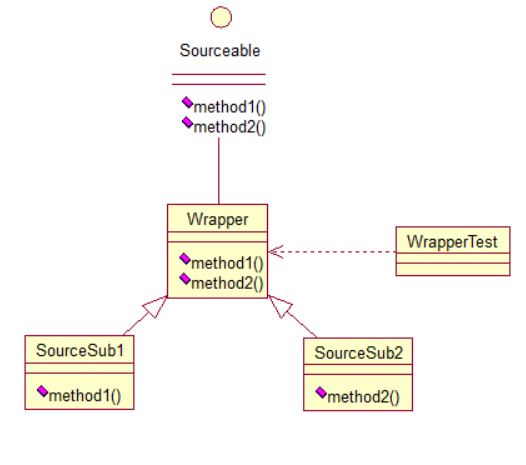
B 对象的适配器模式

创建一个我们我们满足我们需求的类实例对象，在



C 接口的适配器模式

使用一个抽象类，继承接口，改变内部的方法，这个抽象类就是我们需要的



1. 自动装箱和拆箱

答：装箱：基本类型 --> 包装类类型

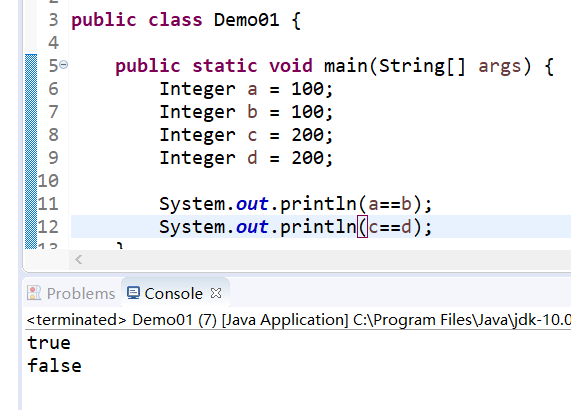
拆箱：包装类类型 –-> 基本类型

以Integer为例：

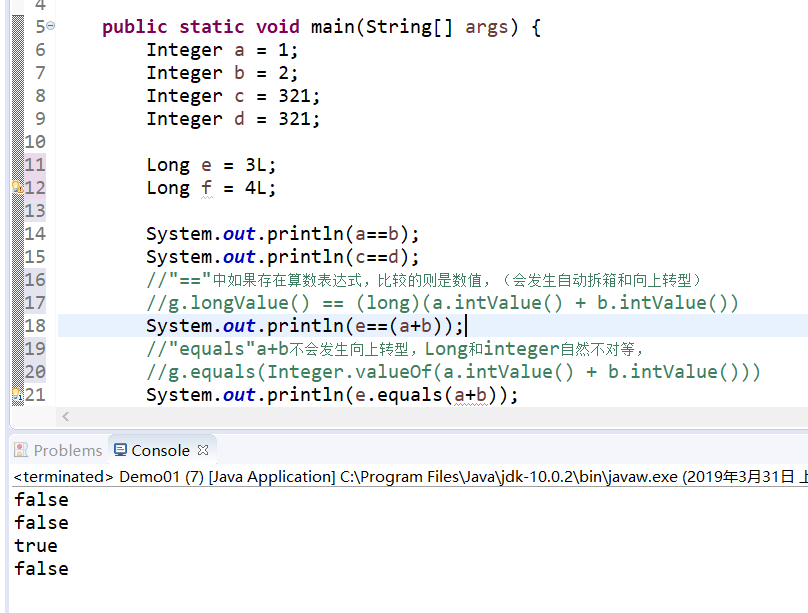
装箱过程：Integer.ValueOf(x)

拆箱过程：.intValue；

在valueOf创建Integer的过程中，如果数值在[-128~127]之间，如果常量池中存在，就返回指向IntegerCache.cache已存在的对象，否则就创建一个对象。



第一个返回true，是因为常量池中存在，因此直接取出已存在的对象，a和b指向的是同一个对象，c和d指向的是不同的对象那个



true

1. String、StringBuilder和StringBuffer的区别？

答：String类中是用final修饰字符数组来保存字符串的，所以是不可变的。StringBuilder和StringBuffer则是继承abstractStringBuilder类，也是用字符数组保存字符串我的，但并未用final修饰，所有是可变的。

线程安全问题：

String类中是用final修饰，是常量，线程安全。

StringBuffer对类中的方法加了同步锁，线程安全。

StringBuilder，线程不安全

性能：

String:每次对String类型继续改变，都会生成一个新的对象。

StringBuffer：对本身进行操作，是可变的

StringBuilder：因为没加同步锁的原因，效率更高；

总结：

操作少量的数据---String

单线程操作大量的数据---StringBuilder

多线程操作大量的数据---StringBuffer

1. HashMap的 put的过程

答：key的hashcode高16位与低16位异或运算得到新的hash，为了让数据（特别是在当前容量不大的情况下）散列更均匀，然后把异或计算出的新hash与此时的hashmap容量-1做&运算，得到插入下标。

1. 问:为什么要做&运算，还有什么方式。

答:二进制运算速度快，还可以取模。之后如果下标位数组无数据则直接插入，如果有数据则链表往下逐个进行hash比较，如果产生hash碰撞再进行==或者equals比较key是否一样，一样则覆盖原数据，否则添加到链表后面。

1. 为什么扩容是以2次幂进行的？

答：a 分布更加均匀 hash&（length-1）

B 不需要重新计算hash值， 将hash值与原容量做&，是1则newindex = index+Oldcap 新的索引= 之前的索引+旧容量，是0，则就是原来的位置。

1. JDk1.8的HashMap并发的情况下，会发生死循环吗？

答：不会，扩容的源代码中取消了transfer的操作，声明了两个指针，维护了两个链表。

1. 讲一下线程池的原理

答：

1. 为什么要用到线程池？

答：a 降低资源消耗：通过重复使用以创建的线程，来减少资源的消耗

B 提高响应的速度：任务不需要等待线程的创建

C 提高线程的可管理性，统一分配，调控

1. 
2. 内部类的作用

答： 1.内部类可以很好的实现隐藏

一般的非内部类，是不允许有 private 与protected权限的，但内部类可以

2．内部类拥有外围类的所有元素的访问权限

3.可是实现多重继承

4.可以避免修改接口而实现同一个类中两种同名方法的调用。

1. N个台阶，一次一步或两步，有多少种走法？

public int findStep(int n) {

if(n==0 || n==1 || n==2) {

return n;

}

return findStep(n-1)+findStep(n-2);

}

1. 对象的初始化顺序

静态成员变量

静态初始化块

成员变量

初始化块

构造方法

1. 类的加载的过程

答：加载 验证 准备 解析 初始化 使用 卸载

1. Java的内存模型

读取 载入 使用 赋值 存储 写入