---

title 面经 categories:

- shengjunjie tags:
-

2021.05.11 阿里JAVA研发工程师一面

面试流程（目前只记得这么多）

1.询问了项目的相关情况，主要包括介绍项目，以及自己目前开发做了哪些工作，你认为哪些比较困难，有什么收获。

2.讲一下静态static?静态方法能不能调用非静态的成员变量？为什么？

- 被static关键字修饰的方法或者变量不需要依赖于对象来进行访问，只要类被加载了，就可以通过类名去进行访问
- 在static方法内部不能调用非静态方法，反过来是可以的。而且可以在没有创建任何对象的前提下，仅仅通过类本身来调用static方法。
- 在静态方法中访问非静态方法/变量的话，那么如果在main方法中有下面一条语句：

```
MyObject.print2();
```

此时对象都没有，str2根本就不存在，所以就会产生矛盾了。同样对于方法也是一样，由于你无法预知在print1方法中是否访问了非静态成员变量，所以也禁止在静态成员方法中访问非静态成员方法。而对于非静态成员方法，它访问静态成员方法/变量显然是毫无限制的。

- 由于其被所有的对象所共享（当且仅当类初次加载时才会被初始化），其常常用于写工具类的工具方法以及单例模式。

```
//单例模式
public class Singleton{
    private staitc volatile Singleton instace=null;
    private Singleton(){

    }
    public static Singleton getInstance(){
        if(instance == null){
            synchronized(Singtон.class){
                if(instance == null){
                    instacce=new Singleton();
                }
            }
        }
        return instace;
    }
}
```

3.讲一下多态？

多态是同一个行为具有多个不同表现形式或形态的能力,就是同一个接口，使用不同的实例而执行不同操作;父类应用指向子类对象(左父右子)，可以使程序有良好的扩展，并可以对所有类的对象进行通用处理。常用于配合设计模式

```
//工厂模式
public interface Shape {
   void draw();
}

public class Rectangle implements Shape {

   @Override
   public void draw() {
      System.out.println("Inside Rectangle::draw() method.");
   }
}

public class Square implements Shape {

   @Override
   public void draw() {
      System.out.println("Inside Square::draw() method.");
   }
}

public class Circle implements Shape {

   @Override
   public void draw() {
      System.out.println("Inside Circle::draw() method.");
   }
}

public class ShapeFactory {

   //使用 getShape 方法获取形状类型的对象
   public Shape getShape(String shapeType){
      if(shapeType == null){
         return null;
      }
      if(shapeType.equalsIgnoreCase("CIRCLE")){
         return new Circle();
      } else if(shapeType.equalsIgnoreCase("RECTANGLE")){
         return new Rectangle();
      } else if(shapeType.equalsIgnoreCase("SQUARE")){
         return new Square();
      }
      return null;
   }
```

```
}

public class FactoryPatternDemo {

   public static void main(String[] args) {
      ShapeFactory shapeFactory = new ShapeFactory();

      //获取 Circle 的对象，并调用它的 draw 方法
      Shape shape1 = shapeFactory.getShape("CIRCLE");

      //调用 Circle 的 draw 方法
      shape1.draw();

      //获取 Rectangle 的对象，并调用它的 draw 方法
      Shape shape2 = shapeFactory.getShape("RECTANGLE");

      //调用 Rectangle 的 draw 方法
      shape2.draw();

      //获取 Square 的对象，并调用它的 draw 方法
      Shape shape3 = shapeFactory.getShape("SQUARE");

      //调用 Square 的 draw 方法
      shape3.draw();
   }
}
```

4.讲一下抽象类与接口的区别？

抽象类

```
abstract class Demo  {

    abstract void method1();

    abstract void method2();

    …

}
```

接口类

```
interface Demo {

    void method1();

    void method2();
```

```
    …
}
```

- 在abstract class方式中，Demo可以有自己的数据成员，也可以有非abstract的成员方法，而在interface方式实现中，Demo只能有静态的不能被修改的数据成员（也就是必须是static final的，不过在interface中一般不定义数据成员），所有的成员方法都是抽象的。从某种意义上说，interface时一种特殊的abstract class。
- 首先，抽象类在Java中代表的是继承关系，一个类只能使用一次继承关系。但是，一个类却可以实现多个接口。其次，在抽象类的定义中，我们可以赋予方法的默认行为。但是在接口的定义中，方法却不能拥有默认行为。不过在jdk1.8中可以使用default关键字实现默认方法。

```
interface InterfaceA {
    default void foo() {
        System.out.println("InterfaceA foo");
    }
}
```

总结:

- 抽象类和接口都不能直接实例化，如果要实例化，抽象类变量必须指向实现所有抽象方法的子类对象，接口变量必须指向实现所有接口方法的类对象。
- 抽象类要被子类继承，接口要被类实现。
- 接口里定义的变量只能是公共的静态的常量，抽象类中的变量是普通变量。
- 抽象类里可以没有抽象方法。
- 接口可以被类多实现（被其他接口多继承），抽象类只能被单继承。
- 接口中没有 this 指针，没有构造函数，不能拥有实例字段（实例变量）或实例方法。
- 抽象类不能在Java 8 的 lambda 表达式中使用。

5.讲一下线程的并发与调度？

- (并行指在同一时间点同时执行)并发是指在同一时间片段同时执行，多线程只能并发执行，实际还是顺执行，只是在同一时间片段，假似同时执行，cpu可以按时间切片执行，单核cpu同一个时刻只支持一个线程执行任务，多线程并发事实上就是多个线程排队申请调用cpu，cpu处理任务速度非常快，所以看上去多个线程任务说并发处理。
- 在运行池中，会有多个处于就绪状态的线程在等待CPU，JAVA虚拟机的一项任务就是负责线程的调度，线程调度是指按照特定机制为多个线程分配CPU的使用权。

1. 分时调度模式: 是指让所有的线程轮流获得cpu的使用权,并且平均分配每个线程占用的cpu的时间片.
2. 抢占式调度模式: JAVA虚拟机采用抢占式调度模式,是指优先让可运行池中优先级高的线程占用CPU,如果可运行池中的线程优先级相同,那就随机选择一个线程,使其占用CPU.处于运行状态的线程会一直运行,直至它不得不放弃CPU. 调整各个线程的优先级 让处于运行状态的线程进入block调用Thread.sleep()方法 让处于运行状态的线程进入Runable调用Thread.yield()方法 让处于运行状态的线程调用另一个线程的join()方法等等 线程切换：不是所有的线程切换都需要进入内核模式 ##### （注意：sleep()方法是Thread类里面的，主要的意义就是让当前线程停止执行，让出cpu给其他的线程，但是不会释放对象锁资源以及监控的状态，当指定的时间到了之后又会自动恢复运行状态。wait()方法是Object类里面的，主要的意义就是

让线程放弃当前的对象的锁，进入等待此对象的等待锁定池，只有针对此对象调动notify方法后本线程才能够进入对象锁定池准备获取对象锁进入运行状态。）

6.讲一下乐观锁与悲观锁的区别？

- 乐观锁：是应用系统层面和数据的业务逻辑层次上的（实际上并没有加锁，只是一种锁思想），利用程序处理并发， 它假定当某一个用户去读取某一个数据的时候，其他的用户不会来访问修改这个数据，但是在最后进行事务的提交的时候会进行数据的检查，以判断在该用户的操作过程中，没有其他用户修改了这个数据。乐观锁的实现大部分都是基于版本控制实现的， 除此之外，还有CAS操作实现
- 悲观锁：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。

7.讲一下Spring中IOC与AOP的原理，spring的实现方式以及Spring的好处？

1. IOC的原理？好处？ 原理：“控制反转”：借助于“第三方”实现具有依赖关系的对象之间的解耦，通过容器，使两个对象之间失去直接的联系，假设当对象A运行到需要对象B的时候，容器会主动创建一个对象B注入到对象A需要的地方。 好处：1.各个类之间只有与容器连接时才具有相关性，所以任何一方出问题都不会影响到另一方的运行，增加了维护性与单元测试性，便于调试程序。2.由于容器与类之间的无关性，在保证接口标准的情况下，可以将一个大中型项目分割为多个子项目，团队成员分工明确，提高产品的开发效率；3.具有可重用性
2. AOP的原理？好处？ 原理：通过动态代理的方式进行实现，主要包括JDK动态代理（代理对象必须必须是接口，核心•InvocationHandler和Proxy）和cglib动态代理（cglib是一个代码生成的类库，可以在运行时动态生成某个类的子类）。 好处：降低模块之间的耦合，2.使系统更容易扩展。3.避免修改业务代码，避免引入重复的代码，有更好的重用性。 使用场景：在日志处理以及事务处理
3. 自动装载机制的原理 （https://blog.csdn.net/weixin_44588495/article/details/106310221）
4. 好处： 1、非侵入式设计 Spring是一种非侵入式（non-invasive）框架，它可以使应用程序代码对框架的依赖最小化。 2、方便解耦、简化开发 Spring就是一个大工厂，可以将所有对象的创建和依赖关系的维护工作都交给Spring容器的管理，大大的降低了组件之间的耦合性。 3、支持AOP Spring提供了对AOP的支持，它允许将一些通用任务，如安全、事物、日志等进行集中式处理，从而提高了程序的复用性。 4、支持声明式事务处理 只需要通过配置就可以完成对事物的管理，而无须手动编程。 5、方便程序的测试 Spring提供了对Junit4的支持，可以通过注解方便的测试Spring程序。 6、方便集成各种优秀框架 Spring不排斥各种优秀的开源框架，其内部提供了对各种优秀框架（如Struts、Hibernate、MyBatis、Quartz等）的直接支持。 7、降低Jave EE API的使用难度。 Spring对Java EE开发中非常难用的一些API（如JDBC、JavaMail等），都提供了封装，使这些API应用难度大大降低。

8.讲一下GC的算法？

- 标记-清除算法：标记无用对象，然后进行清除回收。缺点：效率不高，无法清除垃圾碎片。
- 标记-整理算法：标记无用对象，让所有存活的对象都向一端移动，然后直接清除掉端边界以外的内存。
- 复制算法：按照容量划分二个大小相等的内存区域，当一块用完的时候将活着的对象复制到另一块上，然后再把已使用的内存空间一次清理掉。缺点：内存使用率不高，只有原来的一半。
- 分代算法：根据对象存活周期的不同将内存划分为几块，一般是新生代和老年代，新生代基本采用复制算 法，老年代采用标记整理算法。

9.常见的运行时异常

1，java.lang.NullPointerException

这个异常的解释是 "程序遇上了空指针 "，简单地说就是调用了未经初始化的对象或者是不存在的对象，这个错误经常出现在创建图片，调用数组这些操作中，比如图片未经初始化，或者图片创建时的路径错误等等。

2，java.lang.ClassNotFoundException

异常的解释是"指定的类不存在"，这里主要考虑一下类的名称和路径是否正确即可

3，java.lang.ArrayIndexOutOfBoundsException

这个异常的解释是"数组下标越界"，现在程序中大多都有对数组的操作，因此在调用数组的时候一定要认真检查，看自己调用的下标是不是超出了数组的范围，一般来说，显示（即直接用常数当下标）调用不太容易出这样的错，但隐式（即用变量表示下标）调用就经常出错了.

4，java.lang.NoSuchMethodError

方法不存在错误。当应用试图调用某类的某个方法，而该类的定义中没有该方法的定义时抛出该错误。

5，java.lang.IndexOutOfBoundsException

索引越界异常。当访问某个序列的索引值小于0或大于等于序列大小时，抛出该异常。

6，java.lang.NumberFormatException

数字格式异常。当试图将一个String转换为指定的数字类型，而该字符串确不满足数字类型要求的格式时，抛出该异常。

7，java.sql.SQLException

Sql语句执行异常 8，java.io.IOException

输入输出异常

9，java.lang.IllegalArgumentException

方法参数错误 10java.lang.IllegalAccessException

无访问权限异常

10.runnable与callable的区别

Runnable执行方法是run(),Callable是call() 实现Runnable接口的任务线程无返回值；实现Callable接口的任务线程能返回执行结果 call方法可以抛出异常，run方法若有异常只能在内部消化 注意Callable接口支持返回执行结果，需要调用FutureTask.get()方法实现，此方法会阻塞主线程直到获取结果；当不调用此方法时，主线程不会阻塞！

11.Dao的设计模式

https://www.cnblogs.com/ysyasd/p/11941423.html

Dao设计模式封装了操作具体数据库的细节，对业务层提供操作数据库的接口，因此降低了业务层代码与具体数据库之间的耦合，有利于人员分工，增加了程序的可移植性。 Dao设计模式中主要包含这5个模块：

1、VO类：VO（Value Object）即值对象，每一个值对象对应一张数据库表，便于我们传递数据。

2、Dao接口：Dao接口定义了操作数据库的方法，业务层通过调用这些方法来操作数据库。

3、Dao实现类：操作数据库的方法的具体实现，封装了操作数据库的细节。

4、Dao工厂类：用于代替new操作，进一步降低业务层与数据层之间的耦合。

5、数据库连接类：封装了连接数据库、关闭数据库等常用的操作，减少重复编码。

12.“equal”与“==”

对于基本类型，== 判断两个值是否相等，基本类型没有 equals() 方法。 对于引用类型，== 判断两个变量是否引用同一个对象，而 equals() 判断引用的对象是否等价。

```
Integer x = new Integer(1);
Integer y = new Integer(1);
System.out.println(x.equals(y)); // true
System.out.println(x == y);      // false
```

13.算法题（动态规划类型的）

2021.05.12 美团 后端研发工程师 一面

面试流程（目前只记得这么多）

1.讲解一下自己的项目，根据项目进行提问。

2.为什么java是跨平台的？

java会把文件编译成二进制字节码的class文件，由于java是运行在jvm上的，所以它的代码能在不同的平台的jvm上运行。

3.String的创建方式？

字符串常量池（String Pool）保存着所有字符串字面量（literal strings），这些字面量在编译时期就确定。不仅如此，还可以使用 String 的 intern() 方法在运行过程中将字符串添加到 String Pool 中。 当一个字符串调用 intern() 方法时，如果 String Pool 中已经存在一个字符串和该字符串值相等（使用 equals() 方法 进行确定），那么就会返回 String Pool 中字符串的引用；否则，就会在 String Pool 中添加一个新的字符串，并返回 这个新字符串的引用。 下面示例中，s1 和 s2 采用 new String() 的方式新建了两个不同字符串，而 s3 和 s4 是通过 s1.intern() 方法取得一 个字符串引用。intern() 首先把 s1 引用的字符串放到 String Pool 中，然后返回这个字符串引用。因此 s3 和 s4 引用的是同一个字符串。

```
String s1 = new String("aaa");
String s2 = new String("aaa");
System.out.println(s1 == s2);           // false
String s3 = s1.intern();
String s4 = s1.intern();
System.out.println(s3 == s4);           // true
```

如果是采用 "bbb" 这种字面量的形式创建字符串，会自动地将字符串放入 String Pool 中。

```
String s5 = "bbb";
String s6 = "bbb";
System.out.println(s5 == s6);  // true
```

4.讲一下常用的集合？

(https://blog.csdn.net/qq_40574571/article/details/97612100讲解了HashMap) hashMap:在每个数组元素上都一个链表结构，当数据被Hash后，得到数组下标，把数据放在对应下标元素的链表上。系统将调用key的hashCode()方法得到其hashCode 值（该方法适用于每个Java对象），然后再通过Hash算法的后两步运算（高位运算和取模运算，下文有介绍）来定位该键值对的存储位置，有时两个key会定位到相同的位置，表示发生了Hash碰撞。当然Hash算法计算结果越分散均匀，Hash碰撞的概率就越小，map的存取效率就会越高。在JDK1.8版本中，对数据结构做了进一步的优化，引入了红黑树。而当链表长度太长（默认超过8）时，链表就转换为红黑树，利用红黑树快速增删改查的特点提高HashMap的性能，其中会用到红黑树的插入、删除、查找等算法。
list、map、stack、queue、set

5.面试说了spring是不变的，我这里说没听过，然后给我讲了一下

这里面试官通过string的例子给我大致讲解了一下，其类似于finald的关键字

6.问了怎么部署的？单节点还是多节点？

说了docker，提了一下dockerFile这些，之后又问我怎么保证单节点部署的时候数据不丢失，回答了：通过挂载的方式，面试关建议多节点通过docker部署。

7.算法（力扣82. 删除排序链表中的重复元素 II)

题目：存在一个按升序排列的链表，给你这个链表的头节点 head ，请你删除链表中所有存在数字重复情况的节点，只保留原始链表中 没有重复出现的数字。

返回同样按升序排列的结果链表。

```
public ListNode deleteDuplicates(ListNode head) {
            if(head==null||head.next==null) return head;

        ListNode dummy=new ListNode(-1,head);
        ListNode pre=dummy;

        while(head!=null&&head.next!=null){
            if(head.val==head.next.val){
                ListNode temp=head.next;
                while(temp!=null&&temp.val==head.val){
                    temp=temp.next;
                }
                pre.next=temp;
                head=temp;
```

```
            }else{
                head=head.next;
                pre=pre.next;
            }

        }

        return dummy.next;
    }
```