技术交流请加我微信：nnsouthwind

数据结构和算法

1、判断身份证：要么是15位，要么是18位，最后一位可以为字母，并写程序提出其中的年月日。

> 我们可以用正则表达式来定义复杂的字符串格式，(\d{17}[0-9a-zA-Z]|\d{14}[0-9a-zA-Z])可以用来判断是否为合法的15位或18位身份证号码。
>
> 因为15位和18位的身份证号码都是从7位到第12位为身份证为日期类型。这样我们可以设计出更精确的正则模式，使身份证号的日期合法，这样我们的正则模式可以进一步将日期部分的正则修改为 [12][0-9]{3}[01][0-9][123][0-9] ，当然可以更精确的设置日期。
>
> 在jdk的java.util.Regex包中有实现正则的类,Pattern和Matcher。以下是实现代码：

```
import java.util.regex.Matcher;
import java.util.regex.P attern;
public class RegexTest {
    /**
     * @param args
     */
    public static void main(String[] args) {

        // 测试是否为合法的身份证号码
        String[] strs = { "130681198712092019", "13068119871209201x",
```

```
                "13068119871209201", "123456789012345",
"12345678901234x",
                "1234567890123" };
        Pattern p1 = Pattern.compile("(\\d{17}[0-9a-zA-Z]|\\d{14}[0-
9a-zA-Z])");
        for (int i = 0; i < strs.length; i++) {
            Matcher matcher = p1.matcher(strs[i]);
            System.out.println(strs[i] + ":" + matcher.matches());
        }
        Pattern p2 = Pattern.compile("\\d{6}(\\d{8}).*"); // 用于提取出
生日字符串
        Pattern p3 = Pattern.compile("(\\d{4})(\\d{2})(\\d{2})");// 用
于将生日字符串进行分解为年月日
        for (int i = 0; i < strs.length; i++) {
            Matcher matcher = p2.matcher(strs[i]);
            boolean b = matcher.find();
            if (b) {
                String s = matcher.group(1);
                Matcher matcher2 = p3.matcher(s);
                if (matcher2.find()) {
                    System.out
                            .println("生日为" + matcher2.group(1) +
"年"
                                    + matcher2.group(2) + "月"
                                    + matcher2.group(3) + "日");
                }
            }

        }
    }
}
```

2、说明生活中遇到的二叉树，用Java实现二叉树

这是组合设计模式。

我有很多个(假设10万个)数据要保存起来，以后还需要从保存的这些数据中检索是否存在某个数据，（我想说出二叉树的好处，该怎么说呢？那就是说别人的缺点），假如存在数组中，那么，碰巧要找的数字位于99999那个地方，那查找的速度将很慢，因为要从第1个依次往后取，取出来后进行比较。平衡二叉树（构建平衡二叉树需要先排序，我们这里就不作考虑了）可以很好地解决这个问题，但二叉树的遍历（前序，中序，后序）效率要比数组低很多，代码如下：

```
public class Node {
    public int value;
    public Node left;
    public Node right;
    public void store(int value)
    {
```

```
        if(value<this.value)
        {
            if(left == null)
            {
                left = new Node();
                left.value=value;
            }
            else
            {
                left.store(value);
            }
        }
        else if(value>this.value)
        {
            if(right == null)
            {
                right = new Node();
                right.value=value;
            }
            else
            {
                right.store(value);
            }
        }
    }
    public boolean find(int value)
    {
        System.out.println("happen " + this.value);
        if(value == this.value)
        {
            return true;
        }
        else if(value>this.value)
        {
            if(right == null) return false;
            return right.find(value);
        }else
        {
            if(left == null) return false;
            return left.find(value);
        }
    }
    public  void preList()
    {
        System.out.print(this.value + ",");
        if(left!=null) left.preList();
        if(right!=null) right.preList();
    }
    public void middleList()
```

```
    {
        if(left!=null) left.preList();
        System.out.print(this.value + ",");
        if(right!=null) right.preList();
    }
    public void afterList()
    {
        if(left!=null) left.preList();
        if(right!=null) right.preList();
        System.out.print(this.value + ",");
    }
    public static void main(String [] args)
    {
        int [] data = new int[20];
        for(int i=0;i<data.length;i++)
        {
            data[i] = (int)(Math.random()*100) + 1;
            System.out.print(data[i] + ",");
        }
        System.out.println();
        Node root = new Node();
        root.value = data[0];
        for(int i=1;i<data.length;i++)
        {
            root.store(data[i]);
        }
        root.find(data[19]);
        root.preList();
        System.out.println();
        root.middleList();
        System.out.println();
        root.afterList();
    }
}
```

3、第1个人10，第2个比第1个人大2岁，依次递推，请用递归方式计算出第8个人多大?

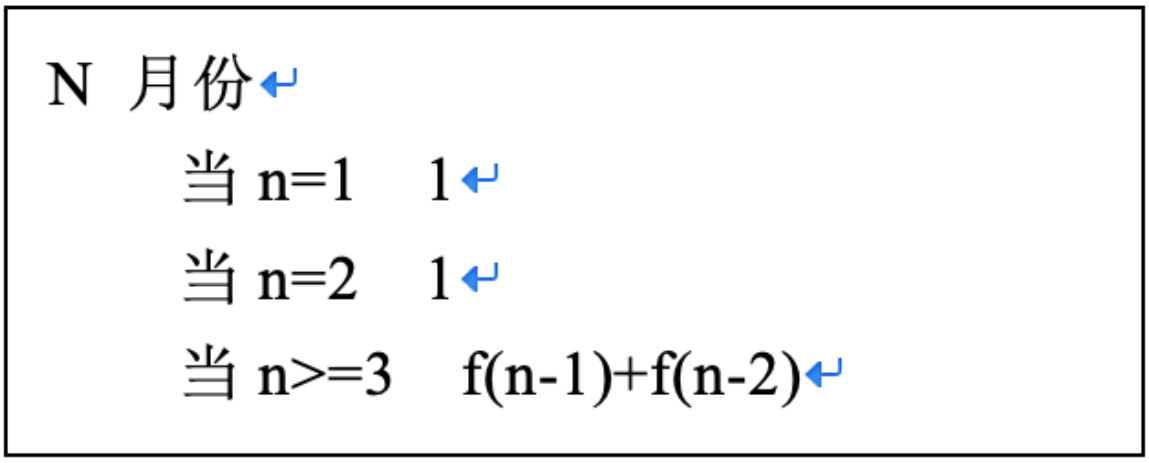
```
import java.util.Date;
public class A1 {
    public static void main(String [] args){
        System.out.println(computeAge(8));
    }

    public static int computeAge(int n){
        if(n==1) return 10;
        return computeAge(n-1) + 2;
```

```
    }
}
public static void toBinary(int n,StringBuffer result){
    if(n/2 != 0)
    toBinary(n/2,result);
    result.append(n%2);
}
```

4、有一对兔子，从出生后第3个月起每个月都生一对兔子，小兔子长到第四个月后每个月又生一对兔子，假如兔子都不死，问第月的兔子总数为多少?

程序分析： 兔子的规律为数列1,1,2,3,5,8,13,21....




```
public class exp1 {
    public static void main(String args[]) {
        int i = 0;
        for (i = 1; i <= 20; i++)
            System.out.println(f(i));
    }

    public static int f(int x) {
        if (x == 1 || x == 2)
            return 1;   else
            return f(x - 1) + f(x - 2);
    }
}

public class exp1 {
     public static void main(String args[]) {
        int i = 0;
        math mymath = new math();
        for (i = 1; i <= 20; i++)
            System.out.println(mymath.f(i));
    }
}
```

```
class math {
    public int f(int x) {
        if (x == 1 || x == 2)
            return 1;
        else
            return f(x - 1) + f(x - 2);
    }
}
```

5、打印出所有的 "水仙花数 "，所谓 "水仙花数 "是指一个三位数，其各位数字立方和等于该数本身。例 如：153是一个 "水仙花数 "，因为153=1的三次方＋5的三次方＋3的三次方。在2000以内的数字。

程序分析：利用for循环控制100-999个数，每个数分解出个位，十位，百位。

```
public class exp2 {
    public static void main(String args[]) {
        int i = 0;
        math mymath = new math();
        for (i = 100; i <= 999; i++)
            if (mymath.shuixianhua(i) == true)
                System.out.println(i);
    }
}

class math {
    public int f(int x) {
        if (x == 1 || x == 2)
            return 1;
        else
            return f(x - 1) + f(x - 2);
    }

    public boolean iszhishu(int x) {
        for (int i = 2; i <= x / 2; i++)    if (x % 2 == 0)
            return false;
            return true;
    }

    public boolean shuixianhua(int x) {
        int i = 0, j = 0, k = 0;
        i = x / 100;   j = (x % 100) / 10;
        k = x % 10;
        if (x == i * i * i + j * j * j + k * k * k)
            return true;
        else    return false;
    }
```

```
}
```

6、输入两个正整数 m和 n，求其最大公约数和最小公倍数。

程序分析：利用辗除法。 最大公约数：

```
public class CommonDivisor {
    public static void main(String args[]) {
        commonDivisor(24, 32);
    }

    static int commonDivisor(int M, int N) {
         if (N < 0 || M < 0) {
             System.out.println("ERROR!");
                return -1;
            }

        if (N == 0) {
            System.out.println("the biggest common divisor is :" + M);
   return M;
            }

        return commonDivisor(N, M % N);

    }
}
```

最小公倍数和最大公约数：

```
import java.util.Scanner;
public class CandC {
    // 下面的方法是求出最大公约数
    public static int gcd(int m, int n) {
        while (true) {
            if ((m = m % n) == 0)     return n;
            if ((n = n % m) == 0)
                return m;
        }
 }

 public static void main(String args[]) throws Exception {
     // 取得输入值
    Scanner chin = new Scanner(System.in);
    int a = chin.nextInt(), b = chin.nextInt();   int a = 23;
    int b = 32;
    int c = gcd(a, b);
    System.out.println("最小公倍数: " + a * b / c + "\n最大公约数: " + c);
```

```
    }
}
```

7、输入一行字符，分别统计出其中英文字母、空格、数字和其它字符的个数。

> 程序分析：利用 while 语句,条件为输入的字符不为 '\n '。

```
import java.util.Scanner;

public class ex7 {
    public static void main(String args[]) {
        System.out.println("请输入字符串：");
        Scanner scan = new Scanner(System.in);
        String str = scan.next();
        String E1 = "[\u4e00-\u9fa5]";
        String E2 = "[a-zA-Z]";
        int countH = 0;
        int countE = 0;
        char[] arrChar = str.toCharArray();
        String[] arrStr = new String[arrChar.length];
        for (int i = 0; i < arrChar.length; i++) {
            arrStr[i] = String.valueOf(arrChar[i]);
        }

        for (String i : arrStr) {
            if (i.matches(E1)) {
                countH++;
            }if (i.matches(E2)) {
                 countE++;
            }
        }
        System.out.println("汉字的个数" + countH);
System.out.println("字母的个数" + countE);
    }
}
```

8、求 s=a+aa+aaa+aaaa+aa...a 的值，其中 a 是一个数字。例如 2+22+222+2222+22222(此时 共有 5 个数相加)，几个数相加有键盘控制。

> 程序分析：关键是计算出每一项的值。

```
import java.io.*;
public class Sumloop {
    public static void main(String[] args) throws IOException {
        int s = 0;
```

```
        String output = "";
        BufferedReader stadin = new BufferedReader(new
InputStreamReader(
    System.in));
        System.out.println("请输入a的值");
        String input = stadin.readLine();
        for (int i = 1; i <= Integer.parseInt(input); i++) {
            output += input;
            int a = Integer.parseInt(output);
            s += a;
        }
        System.out.println(s);

    }
}

import java.io.*;
public class Sumloop {

    public static void main(String[] args) throws IOException {
        int s = 0;
        int n;
        int t = 0;
        BufferedReader stadin = new BufferedReader(new
InputStreamReader(
    System.in));
        String input = stadin.readLine();
        n = Integer.parseInt(input);
        for (int i = 1; i <= n; i++) {
            t = t * 10 + n;
            s = s + t;
            System.out.println(t);
        }
        System.out.println(s);
    }
}
```

9、一个数如果恰好等于它的因子之和，这个数就称为 "完数 "。例如6=1＋2＋3.编程 找出 1000 以内 的所有完数。

```
public class Wanshu {
    public static void main(String[] args) {
        int s;
        for (int i = 1; i <= 1000; i++) {
            s = 0;
            for (int j = 1; j < i; j++)
                if (i % j == 0)
```

```
                s = s + j;
                if (s == i)
            System.out.print(i + " ");
        }
  System.out.println();
    }
}
```

10、有 1、2、3、4 个数字，能组成多少个互不相同且无重复数字的三位数？都是多少?

> 程序分析：可填在百位、十位、个位的数字都是 1、2、3、4。组成所有的排列后再去 掉不满足条件的排列。

```
public class Wanshu {

    public static void main(String[] args) {
        int i = 0;
        int j = 0;
        int k = 0;
        int t = 0;
        for (i = 1; i <= 4; i++)
            for (j = 1; j <= 4; j++)
                for (k = 1; k <= 4; k++)
                    if (i != j && j != k && i != k) {
                            t += 1;
                            System.out.println(i * 100 + j * 10 + k);

                    }

        System.out.println(t);

    }
}
```

11、一个整数，它加上 100 后是一个完全平方数，加上 168 又是一个完全平方数，请问该数是多少?

> 程序分析：在 10 万以内判断，先将该数加上100 后再开方，再将该数加上 268 后再开方，如果开方后的结果满足如下条 件，即是结果。请看具体分析：

```
public class test {

    public static void main(String[] args) {
        long k = 0;
        for (k = 1; k <= 1000001; k++)
            if (Math.floor(Math.sqrt(k + 100)) == Math.sqrt(k + 100)
&& Math.floor(Math.sqrt(k + 168)) == Math.sqrt(k + 168))
                System.out.println(k);
    }
}
```

12、猴子吃桃问题：猴子第一天摘下若干个桃子，当即吃了一半，还不瘾，又多吃了一个 第二天早上又 将剩下的桃子吃掉一半，又多吃了一个。以后每天早上都吃了前一天剩下的一半零一个。到第 10 天早上想再吃时，见只 剩下一个桃子了。求第一天共摘了多少。

> 程序分析：采取逆向思维的方法，从后往前推断。

```
public class Monkey {
    static int total(int day) {
        if (day == 10) {
        return 1;
        } else {
            return (total(day + 1) + 1) * 2;
        }
    }
 public static void main(String[] args) {
  System.out.println(total(1));
 }
}
```

13、有一分数序列：2/1，3/2，5/3，8/5，13/8，21/13...求出这个数列的前 20 项之和。

> 程序分析：请抓住分子与分母的变化规律。

```
public class test20 {
    public static void main(String[] args) {
        float fm = 1f;
        float fz = 1f;    float temp;
        float sum = 0f;
        for (int i = 0; i < 20; i++) {
            temp = fm;
            fm = fz;
            fz = fz + temp;
```

```
            sum += fz / fm;
        }
        System.out.println(sum);
    }
}
```

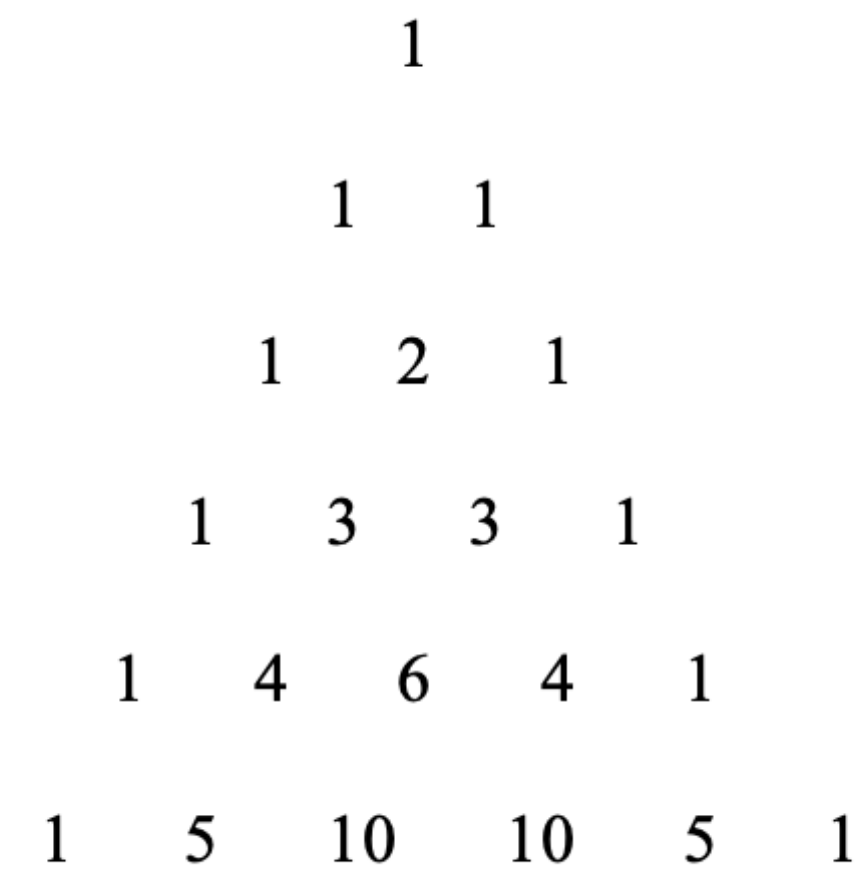
14、有 5 个人坐在一起，问第五个人多少岁？他说比第 4 个人大 2 岁。问第4 个人岁数，他说比第 3 个人 大 2 岁。问第三个人，又说比第 2 人大两岁。问第 2 个人，说比第一个人大两岁。最后问第一个人，他说是10 岁。请问第 五个人多大?

> 程序分析：利用递归的方法，递归分为回推和递推两个阶段。要想知道第五个人岁数，需知道第四人的岁数，依次类推， 推到第一人（10 岁），再往回推。

```
public class Ex23 {
    static int getAge(int n) {
        if (n == 1) {
            return 10;
        }
        return 2 + getAge(n - 1);
    }

    public static void main(String[] args) {
        System.out.println("第五个的年龄为:" + getAge(5));
    }
}
```

15、打印出杨辉三角形




```
public class Ex33 {
    public static void main(String args[]) {   int i, j;
        int a[][];
        a = new int[8][8];
        for (i = 0; i < 8; i++) {
            a[i][i] = 1;    a[i][0] = 1;
        }
        for (i = 2; i < 8; i++) {
            for (j = 1; j <= i - 1; j++) {
                a[i][j] = a[i - 1][j - 1] + a[i - 1][j];
            }
        }
        for (i = 0; i < 8; i++) {
            for (j = 0; j < i; j++) {
                System.out.printf("  " + a[i][j]);
            }
            System.out.println();
        }
    }
}
```

JVM

16、 Java类加载过程?

> java类加载需要经历一下7个过程：
>
> 1. **加载**
>
>   加载是类加载的第一个过程，在这个阶段，将完成一下三件事情：
>
>   - 通过一个类的全限定名获取该类的二进制流。
>   - 将该二进制流中的静态存储结构转化为方法去运行时数据结构。
>   - 在内存中生成该类的Class对象，作为该类的数据访问入口。
> 2. **验证**
>
>   验证的目的是为了确保Class文件的字节流中的信息不回危害到虚拟机.在该阶段主要完成以下四钟验证:
>
>   - 文件格式验证：验证字节流是否符合Class文件的规范，如主次版本号是否在当前虚拟机范围内，常量池中的常量是否有不被支持的类型.
>   - 元数据验证:对字节码描述的信息进行语义分析，如这个类是否有父类，是否集成了不被继承的类等。
>   - 字节码验证：是整个验证过程中最复杂的一个阶段，通过验证数据流和控制流的分析，确定程序语义是否正确，主要针对方法体的验证。如：方法中的类型转换是否正确，跳转指令是否正确等。
>   - 符号引用验证：这个动作在后面的解析过程中发生，主要是为了确保解析动作能正确执行。
> 3. **准备**
>
>   准备阶段是为类的静态变量分配内存并将其初始化为默认值，这些内存都将在方法区中进行分配。准备阶段不分配类中的实例变量的内存，实例变量将会在对象实例化时随着对象一起分配在Java堆中。
>
> public static int value=123;//在准备阶段value初始值为0 。在初始化阶段才会变为123 。
>
> 4. **解析**
>
> 该阶段主要完成符号引用到直接引用的转换动作。解析动作并不一定在初始化动作完成之前，也有可能在初始化之后。
>
> 5. **初始化**
>
> 初始化时类加载的最后一步，前面的类加载过程，除了在加载阶段用户应用程序可以通过自定义类加载器参与之外，其余动作完全由虚拟机主导和控制。到了初始化阶段，才真正开始执行类中定义的Java程序代码。
>
> 6. **使用**
> 7. **卸载**

17、描述一下JVM加载class文件的原理机制?

> Java语言是一种具有动态性的解释型语言，类（class）只有被加载到JVM后才能运行。当运行指定程序时，JVM会将编译生成的.class文件按照需求和一定的规则加载到内存中，并组织成为一个完整的Java应用程序。这个加载过程是由类加载器完成，具体来说，就是由ClassLoader和它的子类来实现的。类加载器本身也是一个类，其实质是把类文件从硬盘读取到内存中。

> 类的加载方式分为隐式加载和显示加载。隐式加载指的是程序在使用new等方式创建对象时，会隐式地调用类的加载器把对应的类加载到JVM中。显示加载指的是通过直接调用class.forName()方法来把所需的类加载到JVM中。
>
> 任何一个工程项目都是由许多类组成的，当程序启动时，只把需要的类加载到JVM中，其他类只有被使用到的时候才会被加载，采用这种方法一方面可以加快加载速度，另一方面可以节约程序运行时对内存的开销。此外，在Java语言中，每个类或接口都对应一个.class文件，这些文件可以被看成是一个个可以被动态加载的单元，因此当只有部分类被修改时，只需要重新编译变化的类即可，而不需要重新编译所有文件，因此加快了编译速度。
>
> 在Java语言中，类的加载是动态的，它并不会一次性将所有类全部加载后再运行，而是保证程序运行的基础类（例如基类）完全加载到JVM中，至于其他类，则在需要的时候才加载。
>
> 类加载的主要步骤：
>
> - 装载。根据查找路径找到相应的class文件，然后导入。
> - 链接。链接又可分为3个小步：
>   - 检查，检查待加载的class文件的正确性。
>   - 准备，给类中的静态变量分配存储空间。
>   - 解析，将符号引用转换为直接引用（这一步可选）
> - 初始化。对静态变量和静态代码块执行初始化工作。

18、 Java内存分配

> **寄存器**：我们无法控制
>
> **静态域**：static定义的静态成员
>
> **常量池**：编译时被确定并保存在.class文件中的（final）常量值和一些文本修饰的符号引用（类和接口的全限定名，字段的名称和描述符，方法和名称和描述符）
>
> **非ram存储**：硬盘等永久存储空间
>
> **堆内存**：new创建的对象和数组，由java虚拟机自动垃圾回收器管理,存取速度慢
>
> **栈内存**：基本类型的变量和对象的引用变量（堆内存空间的访问地址），速度快，可以共享，但是大小与生存期必须确定，缺乏灵活性

19、Java堆的结构是什么样子的？什么是堆中的永久代(Perm Gen space)?

> JVM的堆是运行时数据区，所有类的实例和数组都是在堆上分配内存。它在JVM启动的时 候被创建。对象所占的堆内存是由自动内存管理系统也就是垃圾收集器回收。
>
> 堆内存是由存活和死亡的对象组成的。存活的对象是应用可以访问的，不会被垃圾回收。死亡的对象是应用不可访问尚且还没有被垃圾收集器回收掉的对象。一直到垃圾收集器把这些 对象回收掉之前，他们会一直占据堆内存空间。

20、GC是什么? 为什么要有GC?

> GC是垃圾收集的意思（Gabage Collection）,内存处理是编程人员容易出现问题的地方，忘记或者错误的内存回收会导致程序或系统的不稳定甚至崩溃，Java提供的GC功能可以自动监测对象是否超过作用域从而达到自动回收内存的目的，Java语言没有提供释放已分配内存的显示操作方法。

21、简述Java垃圾回收机制。

> 在java中，程序员是不需要显示的去释放一个对象的内存的，而是由虚拟机自行执行。在JVM中，有一个垃圾回收线程，它是低优先级的，在正常情况下是不会执行的，只有在虚拟机空闲或者当前堆内存不足时，才会触发执行，扫面那些没有被任何引用的对象，并将它们添加到要回收的集合中，进行回收。

22、如何判断一个对象是否存活?(或者GC对象的判定方法)

> 判断一个对象是否存活有两种方法:
>
> 1. 引用计数法
>
>   所谓引用计数法就是给每一个对象设置一个引用计数器，每当有一个地方引用这个对象时，就将计数器加一，引用失效时，计数器就减一。当一个对象的引用计数器为零时，说明此对象没有被引用，也就是“死对象”,将会被垃圾回收.
>
>   引用计数法有一个缺陷就是无法解决循环引用问题，也就是说当对象A引用对象B，对象B又引用者对象A，那么此时A,B对象的引用计数器都不为零，也就造成无法完成垃圾回收，所以主流的虚拟机都没有采用这种算法。
>
> 2. 可达性算法(引用链法)
>
>   该算法的思想是：从一个被称为GC Roots的对象开始向下搜索，如果一个对象到GC Roots没有任何引用链相连时，则说明此对象不可用。
>
> 在java中可以作为GC Roots的对象有以下几种:
>
> - 虚拟机栈中引用的对象
> - 方法区类静态属性引用的对象
> - 方法区常量池引用的对象
> - 本地方法栈JNI引用的对象
>
> 虽然这些算法可以判定一个对象是否能被回收，但是当满足上述条件时，一个对象比不一定会被回收。当一个对象不可达GC Root时，这个对象并不会立马被回收，而是出于一个死缓的阶段，若要被真正的回收需要经历两次标记.
>
> 如果对象在可达性分析中没有与GC Root的引用链，那么此时就会被第一次标记并且进行一次筛选，筛选的条件是是否有必要执行finalize()方法。当对象没有覆盖finalize()方法或者已被虚拟机调用过，那么就认为是没必要的。 如果该对象有必要执行finalize()方法，那么这个对象将会放在一个称为F-Queue的对队列中，虚拟机会触发一个Finalize()线程去执行，此线程是低优先级的，并且虚拟机不会承诺一直等待它运行完，这是因为如果finalize()执行缓慢或者发生了死锁，那么就会造成F-Queue队列一直等待，造成了内存回收系统的崩溃。GC对处于F-Queue中的对象进行第二次被标记，这时，该对象将被移除”即将回收”集合，等待回收。

23、垃圾回收的优点和原理。并考虑2种回收机制。

> Java语言中一个显著的特点就是引入了垃圾回收机制，使c++程序员最头疼的内存管理的问题迎刃而解，它使得Java程序员在编写程序的时候不再需要考虑内存管理。由于有个垃圾回收机制，Java中的对象不再有"作用域"的概念，只有对象的引用才有"作用域"。垃圾回收可以有效的防止内存泄露，有效的使用可以使用的内存。垃圾回收器通常是作为一个单独的低级别的线程运行，不可预知的情况下对内存堆中已经死亡的或者长时间没有使用的对象进行清楚和回收，程序员不能实时的调用垃圾回收器对某个对象或所有对象进行垃圾回收。
>
> 回收机制有分代复制垃圾回收和标记垃圾回收，增量垃圾回收。

24、垃圾回收器的基本原理是什么？垃圾回收器可以马上回收内存吗？有什么办法主动通知虚拟机进行垃圾回收？

> 对于GC来说，当程序员创建对象时，GC就开始监控这个对象的地址、大小以及使用情况。通常，GC采用有向图的方式记录和管理堆(heap)中的所有对象。通过这种方式确定哪些对象是"可达的"，哪些对象是"不可达的"。当GC确定一些对象为"不可达"时，GC就有责任回收这些内存空间。可以。程序员可以手动执行System.gc()，通知GC运行，但是Java语言规范并不保证GC一定会执行。

25、 Java中会存在内存泄漏吗，请简单描述。

> 所谓**内存泄露**就是指一个不再被程序使用的对象或变量一直被占据在内存中。java中有垃圾回收机制，它可以保证一对象不再被引用的时候，即对象变成了孤儿的时候，对象将自动被垃圾回收器从内存中清除掉。由于Java 使用有向图的方式进行垃圾回收管理，可以消除引用循环的问题，例如有两个对象，相互引用，只要它们和根进程不可达的，那么GC也是可以回收它们的，例如下面的代码可以看到这种情况的内存回收：

```
import java.io.IOException;
public class GarbageTest {
    /**
     * @param args
     * @throws IOException
     */
    public static void main(String[] args) throws IOException {
        // TODO Auto-generated method stub
        try {
            gcTest();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        System.out.println("has exited gcTest!");
        System.in.read();
        System.in.read();
```

```
        System.out.println("out begin gc!");
        for(int i=0;i<100;i++)
        {
            System.gc();
            System.in.read();
            System.in.read();
        }
    }
    private static void gcTest() throws IOException {
        System.in.read();
        System.in.read();
        Person p1 = new Person();
        System.in.read();
        System.in.read();
        Person p2 = new Person();
        p1.setMate(p2);
        p2.setMate(p1);
        System.out.println("before exit gctest!");
        System.in.read();
        System.in.read();
        System.gc();
        System.out.println("exit gctest!");
    }
    private static class Person
    {
        byte[] data = new byte[20000000];
        Person mate = null;
        public void setMate(Person other)
        {
            mate = other;
        }
    }
}
```

Java中的内存泄露的情况：长生命周期的对象持有短生命周期对象的引用就很可能发生内存泄露，尽管短生命周期对象已经不再需要，但是因为长生命周期对象持有它的引用而导致不能被回收，这就是java中内存泄露的发生场景，通俗地说，就是程序员可能创建了一个对象，以后一直不再使用这个对象，这个对象却一直被引用，即这个对象无用但是却无法被垃圾回收器回收的，这就是java中可能出现内存泄露的情况，例如，缓存系统，我们加载了一个对象放在缓存中(例如放在一个全局map对象中)，然后一直不再使用它，这个对象一直被缓存引用，但却不再被使用。

检查java中的内存泄露，一定要让程序将各种分支情况都完整执行到程序结束，然后看某个对象是否被使用过，如果没有，则才能判定这个对象属于内存泄露。

如果一个外部类的实例对象的方法返回了一个内部类的实例对象，这个内部类对象被长期引用了，即使那个外部类实例对象不再被使用，但由于内部类持久外部类的实例对象，这个外部类对象将不会被垃圾回收，这也会造成内存泄露。

下面内容来自于网上（主要特点就是清空堆栈中的某个元素，并不是彻底把它从数组中拿掉，而是把存储的总数减少，本人写得可以比这个好，在拿掉某个元素时，顺便也让它从数组中消失，将那个元素所在的位置的值设置为null即可）：

我实在想不到比那个堆栈更经典的例子了,以致于我还要引用别人的例子，下面的例子不是我想到的，是书上看到的，当然如果没有在书上看到，可能过一段时间我自己也想的到，可是那时我说是我自己想到的也没有人相信的。

```
public class Stack {
    private Object[] elements=new Object[10];
    private int size = 0;
    public void push(Object e){
        ensureCapacity();
        elements[size++] = e;
    }

    public Object pop(){
        if( size == 0) throw new EmptyStackException();
        return elements[--size];
    }

    private void ensureCapacity(){
        if(elements.length == size){
            Object[] oldElements = elements;
            elements = new Object[2 * elements.length+1];
            System.arraycopy(oldElements,0, elements, 0, size);
            }
    }
}
```

上面的原理应该很简单，假如堆栈加了10个元素，然后全部弹出来，虽然堆栈是空的，没有我们要的东西，但是这是个对象是无法回收的，这个才符合了内存泄露的两个条件：无用，无法回收。但是就是存在这样的东西也不一定会导致什么样的后果，如果这个堆栈用的比较少，也就浪费了几个K内存而已，反正我们的内存都上G了，哪里会有什么影响，再说这个东西很快就会被回收的，有什么关系。下面看两个例子。

```
public class Bad{
    public static Stack s=Stack();
    static{
        s.push(new Object());
        s.pop(); //这里有一个对象发生内存泄露
        s.push(new Object()); //上面的对象可以被回收了，等于是自愈了
    }
}
```

> 因为是static，就一直存在到程序退出，但是我们也可以看到它有自愈功能，就是说如果你的Stack最多有100个对象，那么最多也就只有100个对象无法被回收其实这个应该很容易理解，Stack内部持有100个引用，最坏的情况就是他们都是无用的，因为我们一旦放新的进取，以前的引用自然消失！
>
> 内存泄露的另外一种情况：当一个对象被存储进HashSet集合中以后，就不能修改这个对象中的那些参与计算哈希值的字段了，否则，对象修改后的哈希值与最初存储进HashSet集合中时的哈希值就不同了，在这种情况下，即使在contains方法使用该对象的当前引用作为的参数去HashSet集合中检索对象，也将返回找不到对象的结果，这也会导致无法从HashSet集合中单独删除当前对象，造成内存泄露。

26、深拷贝和浅拷贝

> 简单来讲就是复制、克隆；
>
> Person p=new Person(“张三”);
>
> 浅拷贝就是对对象中的数据成员进行简单赋值，如果存在动态成员或者指针就会报错。
>
> 深拷贝就是对对象中存在的动态成员或指针重新开辟内存空间。

27、System.gc()和Runtime.gc()会做什么事情?

> 这两个方法用来提示JVM要进行垃圾回收。但是，立即开始还是延迟进行垃圾回收是取决于JVM的。

28、finalize()方法什么时候被调用？析构函数(finalization)的目的是什么?

> 垃圾回收器(garbage colector)决定回收某对象时，就会运行该对象的finalize()方法 但是在Java中很不幸，如果内存总是充足的，那么垃圾回收可能永远不会进行，也就是说filalize() 可能永远不被执行，显然指望它做收尾工作是靠不住的。 那么finalize()究竟是做什么的呢？ 它最主要的用途是回收特殊渠道申请的内存。Java程序有垃圾回收器，所以一般情况下内存问题不用程序员操心。但有一种JNI(Java Native Interface)调用non-Java程序（C或C++）， finalize()的工作就是回收这部分的内存。

29、如果对象的引用被置为null，垃圾收集器是否会立即释放对象占用的内存?

> 不会，在下一个垃圾回收周期中，这个对象将是可被回收的。

30、什么是分布式垃圾回收(DGC)？它是如何工作的?

> DGC叫做分布式垃圾回收。RMI使用DGC来做自动垃圾回收。因为RMI包含了跨虚拟机的远程对象的引用，垃圾回收是很困难的。DGC使用引用计数算法来给远程对象提供自动 内存管理。

31、串行(serial)收集器和吞吐量(throughput)收集器的区别是什么?

> 吞吐量收集器使用并行版本的新生代垃圾收集器，它用于中等规模和大规模数据的应用程序。而串行收集器对大多数的小应用(在现代处理器上需要大概100M左右的内存)就足够了。

32、在Java中，对象什么时候可以被垃圾回收?

> 当对象对当前使用这个对象的应用程序变得不可触及的时候，这个对象就可以被回收了。

33、简述Java内存分配与回收策率以及Minor GC和Major GC

> - 对象优先在堆的Eden区分配。
> - 大对象直接进入老年代.
> - 长期存活的对象将直接进入老年代.
>
> 当Eden区没有足够的空间进行分配时，虚拟机会执行一次Minor GC.Minor Gc通常发生在新生代的Eden区，在这个区的对象生存期短，往往发生Gc的频率较高，回收速度比较快;Full Gc/Major GC 发生在老年代，一般情况下，触发老年代GC的时候不会触发Minor GC,但是通过配置，可以在Full GC之前进行一次Minor GC这样可以加快老年代的回收速度。

34、JVM的永久代中会发生垃圾回收么?

> 垃圾回收不会发生在永久代，如果永久代满了或者是超过了临界值，会触发完全垃圾回收(Full GC)。
>
> (注：Java8中已经移除了永久代，新加了一个叫做元数据区的native内存区)

35、Java中垃圾收集的方法有哪些?

> 标记-清除:这是垃圾收集算法中最基础的，根据名字就可以知道，它的思想就是标记哪些要被回收的对象，然后统一回收。这种方法很简单，但是会有两个主要问题：1.效率不高，标记和清除的效率都很低；2.会产生大量不连续的内存碎片，导致以后程序在分配较大的对象时，由于没有充足的连续内存而提前触发一次GC动作。
>
> 复制算法:为了解决效率问题，复制算法将可用内存按容量划分为相等的两部分，然后每次只使用其中的一块，当一块内存用完时，就将还存活的对象复制到第二块内存上，然后一次性清楚完第一块内存，再将第二块上的对象复制到第一块。但是这种方式，内存的代价太高，每次基本上都要浪费一般的内存。 于是将该算法进行了改进，内存区域不再是按照1：1去划分，而是将内存划分为8:1:1三部分，较大那份内存交Eden区，其余是两块较小的内存区叫Survior区。每次都会优先使用Eden区，若Eden区满，就将对象复制到第二块内存区上，然后清除Eden区，如果此时存活的对象太多，以至于Survivor不够时，会将这些对象通过分配担保机制复制到老年代中。(java堆又分为新生代和老年代)

> 标记-整理:该算法主要是为了解决标记-清除，产生大量内存碎片的问题；当对象存活率较高时，也解决了复制算法的效率问题。它的不同之处就是在清除对象的时候现将可回收对象移动到一端，然后清除掉端边界以外的对象，这样就不会产生内存碎片了。
>
> 分代收集 :现在的虚拟机垃圾收集大多采用这种方式，它根据对象的生存周期，将堆分为新生代和老年代。在新生代中，由于对象生存期短，每次回收都会有大量对象死去，那么这时就采用复制算法。老年代里的对象存活率较高，没有额外的空间进行分配担保；

36、什么是类加载器，类加载器有哪些?

> 实现通过类的权限定名获取该类的二进制字节流的代码块叫做类加载器。
>
> 主要有一下四种类加载器:
>
> - 启动类加载器(Bootstrap ClassLoader)用来加载java核心类库，无法被java程序直接引用。
> - 扩展类加载器(extensions class loader):它用来加载 Java 的扩展库。Java 虚拟机的实现会提供一个扩展库目录。该类加载器在此目录里面查找并加载 Java 类。
> - 系统类加载器（system class loader）：它根据 Java 应用的类路径（CLASSPATH）来加载 Java 类。一般来说，Java 应用的类都是由它来完成加载的。可以通过 ClassLoader.getSystemClassLoader()来获取它。
> - 用户自定义类加载器，通过继承 java.lang.ClassLoader类的方式实现。

37、类加载器双亲委派模型机制?

> 当一个类收到了类加载请求时，不会自己先去加载这个类，而是将其委派给父类，由父类去加载，如果此时父类不能加载，反馈给子类，由子类去完成类的加载。

Java核心基础

37、一个".java"源文件中是否可以包括多个类（不是内部类）？有什么限制?

> 可以有多个类，但只能有一个public的类，并且public的类名必须与文件名相一致。

38、说说&和&&的区别。

> &和&&都可以用作逻辑与的运算符，表示逻辑与（and），当运算符两边的表达式的结果都为true时，整个运算结果才为true，否则，只要有一方为false，则结果为false。
>
> &&还具有短路的功能，即如果第一个表达式为false，则不再计算第二个表达式，例如，对于if(str != null && !str.equals(“”))表达式，当str为null时，后面的表达式不会执行，所以不会出现NullPointerException如果将&&改为&，则会抛出NullPointerException异常。If(x==33 & ++y>0) y会增长，If(x==33 && ++y>0)不会增长。

> &还可以用作位运算符，当&操作符两边的表达式不是boolean类型时，&表示按位与操作，我们通常使用0x0f来与一个整数进行&运算，来获取该整数的最低4个bit位，例如，0x31 & 0x0f的结果为0x01。
>
> 备注：这道题先说两者的共同点，再说出&&和&的特殊之处，并列举一些经典的例子来表明自己理解透彻深入、实际经验丰富。

39、值传递和引用传递

> 值传递是针对基本数据类型而言，传递的是值得副本，对副本的改变不会影响到原变量。
>
> 引用传递就是将一个堆内存空间的使用权交给多个栈内存空间，每一个栈内存空间都可以对堆内存空间进行修改。

40、在Java中如何跳出当前的多重嵌套循环?

> 在Java中，要想跳出多重循环，可以在外面的循环语句前定义一个标号，然后在里层循环体的代码中使用带有标号的break 语句，即可跳出外层循环。例如：

```
for(int i=0;i<10;i++)
{
    for(int j=0;j<10;j++)
    {
        System.out.println("i=" + i + ",j=" + j);
        if(j == 5) break ok;
    }
}
```

> 另外，我个人通常并不使用标号这种方式，而是让外层的循环条件表达式的结果可以受到里层循环体代码的控制，例如，要在二维数组中查找到某个数字。

```
int arr[][] = {{1,2,3},{4,5,6,7},{9}};
boolean found = false;
for(int i=0;i<arr.length && !found;i++)
    {
        for(int j=0;j<arr[i].length;j++)
        {
            System.out.println("i=" + i + ",j=" + j);
            if(arr[i][j]  == 5)
            {
                found = true;
                break;
            }
        }
    }
```

41、访问修饰符public,private,protected,以及不写（默认）时的区别?

修饰符	当前类	同 包	子 类	其他包
public	√	√	√	√
protected	√	√	√	×
Default	√	√	×	×
private	√	×	×	×

42、switch语句能否作用在byte上，能否作用在long上，能否作用在String上?

在switch（expr1）中，expr1只能是一个整数表达式或者枚举常量（更大字体），整数表达式可以是int基本类型或Integer包装类型，由于，byte,short,char都可以隐含转换为int，所以，这些类型以及这些类型的包装类型也是可以的。显然，long和String类型都不符合switch的语法规定，并且不能被隐式转换成int类型，所以，它们不能作用于swtich语句中。

43、char型变量中能不能存贮一个中文汉字?为什么?

char型变量是用来存储Unicode编码的字符的，unicode编码字符集中包含了汉字，所以，char型变量中当然可以存储汉字啦。不过，如果某个特殊的汉字没有被包含在unicode编码字符集中，那么，这个char型变量中就不能存储这个特殊汉字。补充说明：unicode编码占用两个字节，所以，char类型的变量也是占用两个字节。

备注：后面一部分回答虽然不是在正面回答题目，但是，为了展现自己的学识和表现自己对问题理解的透彻深入，可以回答一些相关的知识，做到知无不言，言无不尽。

44、用最有效率的方法算出2乘以8等於几?

2 << 3

因为将一个数左移n位，就相当于乘以了2的n次方，那么，一个数乘以8只要将其左移3位即可，而位运算cpu直接支持的，效率最高，所以，2乘以8等於几的最效率的方法是2 << 3。

45、使用final关键字修饰一个变量时，是引用不能变，还是引用的对象不能变?

使用final关键字修饰一个变量时，是指引用变量不能变，引用变量所指向的对象中的内容还是可以改变的。例如，对于如下语句：

```
final StringBuffer a=new StringBuffer("immutable");
```

执行如下语句将报告编译期错误：

```
a=new StringBuffer("");
```

但是，执行如下语句则可以通过编译：

```
a.append(" broken!");
```

有人在定义方法的参数时，可能想采用如下形式来阻止方法内部修改传进来的参数对象：

```
public void method(final  StringBuffer  param)
{
}
```

实际上，这是办不到的，在该方法内部仍然可以增加如下代码来修改参数对象：

```
param.append("a");
```

46、"=="和equals方法究竟有什么区别?

（单独把一个东西说清楚，然后再说清楚另一个，这样，它们的区别自然就出来了，混在一起说，则很难说清楚）

==操作符专门用来比较两个变量的值是否相等，也就是用于比较变量所对应的内存中所存储的数值是否相同，要比较两个基本类型的数据或两个引用变量是否相等，只能用==操作符。

如果一个变量指向的数据是对象类型的，那么，这时候涉及了两块内存，对象本身占用一块内存（堆内存），变量也占用一块内存，例如Objet obj = new Object();变量obj是一个内存，new Object()是另一个内存，此时，变量obj所对应的内存中存储的数值就是对象占用的那块内存的首地址。对于指向对象类型的变量，如果要比较两个变量是否指向同一个对象，即要看这两个变量所对应的内存中的数值是否相等，这时候就需要用==操作符进行比较。

equals方法是用于比较两个独立对象的内容是否相同，就好比去比较两个人的长相是否相同，它比较的两个对象是独立的。例如，对于下面的代码：

```
String a=new String("foo");

String b=new String("foo");
```

两条new语句创建了两个对象，然后用a,b这两个变量分别指向了其中一个对象，这是两个不同的对象，它们的首地址是不同的，即a和b中存储的数值是不相同的，所以，表达式a==b将返回false，而这两个对象中的内容是相同的，所以，表达式a.equals(b)将返回true。

在实际开发中，我们经常要比较传递进行来的字符串内容是否等，例如，String input = ...;input.equals("quit")，许多人稍不注意就使用==进行比较了，这是错误的，随便从网上找几个项目实战的教学视频看看，里面就有大量这样的错误。记住，字符串的比较基本上都是使用equals方法。

如果一个类没有自己定义equals方法，那么它将继承Object类的equals方法，Object类的equals方法的实现代码如下：

```
boolean equals(Object o){

return this==o;

}
```

这说明，如果一个类没有自己定义equals方法，它默认的equals方法（从Object 类继承的）就是使用==操作符，也是在比较两个变量指向的对象是否是同一对象，这时候使用equals和使用==会得到同样的结果，如果比较的是两个独立的对象则总返回false。如果你编写的类希望能够比较该类创建的两个实例对象的内容是否相同，那么你必须覆盖equals方法，由你自己写代码来决定在什么情况即可认为两个对象的内容是相同的。

47、静态变量和实例变量的区别?

在语法定义上的区别：静态变量前要加static关键字，而实例变量前则不加。

在程序运行时的区别：实例变量属于某个对象的属性，必须创建了实例对象，其中的实例变量才会被分配空间，才能使用这个**实例变量**。静态变量不属于某个实例对象，而是属于类，所以也称为**类变量**，只要程序加载了类的字节码，不用创建任何实例对象，静态变量就会被分配空间，静态变量就可以被使用了。总之，实例变量必须创建对象后才可以通过这个对象来使用，静态变量则可以直接使用类名来引用。

例如，对于下面的程序，无论创建多少个实例对象，永远都只分配了一个staticVar变量，并且每创建一个实例对象，这个staticVar就会加1；但是，每创建一个实例对象，就会分配一个instanceVar，即可能分配多个instanceVar，并且每个instanceVar的值都只自加了1次。

```
public class VariantTest{
    public static int staticVar = 0;
    public int instanceVar = 0;
    public VariantTest()
    {
        staticVar++;
        instanceVar++;
        System.out.println(“staticVar=” + staticVar + ”,instanceVar=”
+ instanceVar);
    }
}
```

备注：这个解答除了说清楚两者的区别外，最后还用一个具体的应用例子来说明两者的差异，体现了自己有很好的解说问题和设计案例的能力，思维敏捷，超过一般程序员，有写作能力！

48、是否可以从一个static方法内部发出对非static方法的调用?

不可以。因为非static方法是要与对象关联在一起的，必须创建一个对象后，才可以在该对象上进行方法调用，而static方法调用时不需要创建对象，可以直接调用。也就是说，当一个static方法被调用时，可能还没有创建任何实例对象，如果从一个static方法中发出对非static方法的调用，那个非static方法是关联到哪个对象上的呢？这个逻辑无法成立，所以，不可以一个static方法内部发出对非static方法的调用。

49、Math.round(11.5)等于多少? Math.round(-11.5)等于多少?

Math类中提供了三个与取整有关的方法：ceil、floor、round，这些方法的作用与它们的英文名称的含义相对应，例如，ceil的英文意义是天花板，该方法就表示向上取整，所以，Math.ceil(11.3)的结果为12,Math.ceil(-11.3)的结果是-11；floor的英文意义是地板，该方法就表示向下取整，所以，Math.floor(11.6)的结果为11,Math.floor(-11.6)的结果是-12；最难掌握的是round方法，它表示“四舍五入”，算法为Math.floor(x+0.5)，即将原来的数字加上0.5后再向下取整，所以，Math.round(11.5)的结果为12，Math.round(-11.5)的结果为-11。

50、Overload和Override的区别。Overload的方法是否可以改变返回值的类型?

Overload是重载的意思，Override是覆盖的意思，也就是重写。

重载Overload表示同一个类中可以有多个名称相同的方法，但这些方法的参数列表各不相同（即参数个数或类型不同）。

重写Override表示子类中的方法可以与父类中的某个方法的名称和参数完全相同，通过子类创建的实例对象调用这个方法时，将调用子类中的定义方法，这相当于把父类中定义的那个完全相同的方法给覆盖了，这也是面向对象编程的多态性的一种表现。子类覆盖父类的方法时，只能比父类抛出更少的异常，或者是抛出父类抛出的异常的子异常，因为子类可以解决父类的一些问题，不能比父类有更多的问题。子类方法的访问权限只能比父类的更大，不能更小。如果父类的方法是private类型，那么，子类则不存在覆盖的限制，相当于子类中增加了一个全新的方法。

至于Overloaded的方法是否可以改变返回值的类型这个问题，要看你倒底想问什么呢？这个题目很模糊。如果几个Overloaded的方法的参数列表不一样，它们的返回者类型当然也可以不一样。但我估计你想问的问题是：如果两个方法的参数列表完全一样，是否可以让它们的返回值不同来实现重载Overload。这是不行的，我们可以用反证法来说明这个问题，因为我们有时候调用一个方法时也可以不定义返回结果变量，即不要关心其返回结果，例如，我们调用map.remove(key)方法时，虽然remove方法有返回值，但是我们通常都不会定义接收返回结果的变量，这时候假设该类中有两个名称和参数列表完全相同的方法，仅仅是返回类型不同，java就无法确定编程者倒底是想调用哪个方法了，因为它无法通过返回结果类型来判断。

override可以翻译为覆盖，从字面就可以知道，它是覆盖了一个方法并且对其重写，以求达到不同的作用。对我们来说最熟悉的覆盖就是对接口方法的实现，在接口中一般只是对方法进行了声明，而我们在实现时，就需要实现接口声明的所有方法。除了这个典型的用法以外，我们在继承中也可能会在子类覆盖父类中的方法。在覆盖要注意以下的几点：

- 覆盖的方法的标志必须要和被覆盖的方法的标志完全匹配，才能达到覆盖的效果；
- 覆盖的方法的返回值必须和被覆盖的方法的返回一致；

- 覆盖的方法所抛出的异常必须和被覆盖方法的所抛出的异常一致，或者是其子类；
- 被覆盖的方法不能为private，否则在其子类中只是新定义了一个方法，并没有对其进行覆盖。

overload对我们来说可能比较熟悉，可以翻译为重载，它是指我们可以定义一些名称相同的方法，通过定义不同的输入参数来区分这些方法，然后再调用时，VM就会根据不同的参数样式，来选择合适的方法执行。在使用重载要注意以下的几点：

- 在使用重载时只能通过不同的参数样式。例如，不同的参数类型，不同的参数个数，不同的参数顺序（当然，同一方法内的几个参数类型必须不一样，例如可以是fun(int,float)，但是不能为fun(int,int)）；
- 不能通过访问权限、返回类型、抛出的异常进行重载；
- 方法的异常类型和数目不会对重载造成影响；
- 对于继承来说，如果某一方法在父类中是访问权限是priavte，那么就不能在子类对其进行重载，如果定义的话，也只是定义了一个新方法，而不会达到重载的效果。

51、ClassLoader如何加载class

JVM里有多个类加载，每个类加载可以负责加载特定位置的类，例如，bootstrap类加载负责加载jre/lib/rt.jar中的类， 我们平时用的jdk中的类都位于rt.jar中。extclassloader负责加载jar/lib/ext/*.jar中的类，appclassloader负责classpath指定的目录或jar中的类。除了bootstrap之外，其他的类加载器本身也都是java类，它们的父类是ClassLoader。

52、序列化接口的id有什么用?

对象经常要通过IO进行传送，让你写程序传递对象，你会怎么做？把对象的状态数据用某种格式写入到硬盘，Person->“zxx,male,28,30000”àPerson，既然大家都要这么干，并且没有个统一的干法，于是，sun公司就提出一种统一的解决方案，它会把对象变成某个格式进行输入和输出，这种格式对程序员来说是透明（transparent）的，但是，我们的某个类要想能被sun的这种方案处理，必须实现Serializable接口。

```
ObjectOutputStream.writeObject(obj);
Object obj = ObjectInputStream.readObject();
```

假设两年前我保存了某个类的一个对象，这两年来，我修改该类，删除了某个属性和增加了另外一个属性，两年后，我又去读取那个保存的对象，或有什么结果？未知！sun的jdk就会蒙了。为此，一个解决办法就是在类中增加版本后，每一次类的属性修改，都应该把版本号升级一下，这样，在读取时，比较存储对象时的版本号与当前类的版本号，如果不一致，则直接报版本号不同的错!

53、hashCode方法的作用?

> hashcode这个方法是用来鉴定2个对象是否相等的。与equals方法功能类似，但是有区别。一般来讲，equals这个方法是给用户调用的，如果你想判断2个对象是否相等，你可以重写equals方法，然后在代码中调用，就可以判断他们是否相等了。简单来讲，equals方法主要是用来判断从表面上看或者从内容上看，2个对象是不是相等。举个例子，有个学生类，属性只有姓名和性别，那么我们可以认为只要姓名和性别相等，那么就说这2个对象是相等的。
>
> hashcode方法一般用户不会去调用，比如在hashmap中，由于key是不可以重复的，他在判断key是不是重复的时候就判断了hashcode这个方法，而且也用到了equals方法。这里不可以重复是说equals和hashcode只要有一个不等就可以了！所以简单来讲，hashcode相当于是一个对象的编码，就好像文件中的md5，他和equals不同就在于他返回的是int型的，比较起来不直观。我们一般在覆盖equals的同时也要覆盖hashcode，让他们的逻辑一致。举个例子，还是刚刚的例子，如果姓名和性别相等就算2个对象相等的话，那么hashcode的方法也要返回姓名的hashcode值加上性别的hashcode值，这样从逻辑上，他们就一致了。

54、构造器Constructor是否可被override?

> 构造器Constructor不能被继承，因此不能重写Override，但可以被重载Overload。

55、接口是否可继承接口? 抽象类是否可实现(implements)接口? 抽象类是否可继承具体类(concrete class)? 抽象类中是否可以有静态的main方法?

> 接口可以继承接口。抽象类可以实现(implements)接口，抽象类可继承具体类。抽象类中可以有静态的main方法。
>
> 备注：只要明白了接口和抽象类的本质和作用，这些问题都很好回答，你想想，如果你是java语言的设计者，你是否会提供这样的支持，如果不提供的话，有什么理由吗？如果你没有道理不提供，那答案就是肯定的了。
>
> 只有记住抽象类与普通类的唯一区别就是不能创建实例对象和允许有abstract方法。

56、Java中实现多态的机制是什么?

> 靠的是父类或接口定义的引用变量可以指向子类或具体实现类的实例对象，而程序调用的方法在运行期才动态绑定，就是引用变量所指向的具体实例对象的方法，也就是内存里正在运行的那个对象的方法，而不是引用变量的类型中定义的方法。

57、abstract class和interface有什么区别?

> 含有abstract修饰符的class即为抽象类，abstract 类不能创建的实例对象。含有abstract方法的类必须定义为abstract class，abstract class类中的方法不必是抽象的。abstract class类中定义抽象方法必须在具体(Concrete)子类中实现，所以，不能有抽象构造方法或抽象静态方法。如果的子类没有实现抽象父类中的所有抽象方法，那么子类也必须定义为abstract类型。
>
> 接口（interface）可以说成是抽象类的一种特例，接口中的所有方法都必须是抽象的。接口中的方法定义默认为public abstract类型，接口中的成员变量类型默认为public static final。

下面比较一下两者的语法区别：

- 抽象类可以有构造方法，接口中不能有构造方法。
- 抽象类中可以有普通成员变量，接口中没有普通成员变量
- 抽象类中可以包含非抽象的普通方法，接口中的所有方法必须都是抽象的，不能有非抽象的普通方法。
- 抽象类中的抽象方法的访问类型可以是public，protected和（默认类型,虽然eclipse下不报错，但应该也不行），但接口中的抽象方法只能是public类型的，并且默认即为public abstract类型。
- 抽象类中可以包含静态方法，接口中不能包含静态方法
- 抽象类和接口中都可以包含静态成员变量，抽象类中的静态成员变量的访问类型可以任意，但接口中定义的变量只能是public static final类型，并且默认即为public static final类型。
- 一个类可以实现多个接口，但只能继承一个抽象类。

下面接着再说说两者在应用上的区别：**

接口更多的是在系统架构设计方法发挥作用，主要用于定义模块之间的通信契约。而抽象类在代码实现方面发挥作用，可以实现代码的重用，例如，模板方法设计模式是抽象类的一个典型应用，假设某个项目的所有Servlet类都要用相同的方式进行权限判断、记录访问日志和处理异常，那么就可以定义一个抽象的基类，让所有的Servlet都继承这个抽象基类，在抽象基类的service方法中完成权限判断、记录访问日志和处理异常的代码，在各个子类中只是完成各自的业务逻辑代码，伪代码如下：

```
public abstract class BaseServlet extends HttpServlet{
    public final void service(HttpServletRequest request,
HttpServletResponse response) throws IOExcetion,ServletException{
        //记录访问日志
        //进行权限判断
        if(具有权限){
            try {
                doService(request,response);
            }catch(Excetpion e){
                //记录异常信息
            }
        }
    }
    protected abstract void doService(HttpServletRequest request,
HttpServletResponse response) throws IOExcetion,ServletException;
    //注意访问权限定义成protected，显得既专业，又严谨，因为它是专门给子类用的
}

public class MyServlet1 extends BaseServlet{
    protected void doService(HttpServletRequest request,
HttpServletResponse response) throws IOExcetion,ServletException{
    //本Servlet只处理的具体业务逻辑代码
    }
}
```

父类方法中间的某段代码不确定，留给子类干，就用模板方法设计模式。

备注：这道题的思路是先从总体解释抽象类和接口的基本概念，然后再比较两者的语法细节，最后再说两者的应用区别。比较两者语法细节区别的条理是：先从一个类中的构造方法、普通成员变量和方法（包括抽象方法），静态变量和方法，继承性等6个方面逐一去比较回答，接着从第三者继承的角度的回答，特别是最后用了一个典型的例子来展现自己深厚的技术功底。

58、abstract的method是否可同时是static,是否可同时是native，是否可同时是synchronized?

abstract的method 不可以是static的，因为抽象的方法是要被子类实现的，而static与子类扯不上关系。

native方法表示该方法要用另外一种依赖平台的编程语言实现的，不存在着被子类实现的问题，所以，它也不能是抽象的，不能与abstract混用。例如，FileOutputSteam类要硬件打交道，底层的实现用的是操作系统相关的api实现，例如，在windows用c语言实现的，所以，查看jdk 的源代码，可以发现FileOutputStream的open方法的定义如下：

```
private native void open(String name) throws FileNotFoundException;
```

如果我们要用java调用别人写的c语言函数，我们是无法直接调用的，我们需要按照java的要求写一个c语言的函数，又我们的这个c语言函数去调用别人的c语言函数。由于我们的c语言函数是按java的要求来写的，我们这个c语言函数就可以与java对接上，java那边的对接方式就是定义出与我们这个c函数相对应的方法，java中对应的方法不需要写具体的代码，但需要在前面声明native。

关于synchronized与abstract合用的问题，我觉得也不行，因为在我几年的学习和开发中，从来没见到过这种情况，并且我觉得synchronized应该是作用在一个具体的方法上才有意义。而且，方法上的synchronized同步所使用的同步锁对象是this，而抽象方法上无法确定this是什么。1.1 什么是内部类？Static Nested Class 和 Inner Class的不同。

59、 什么是内部类？Static Nested Class 和 Inner Class的不同。

内部类就是在一个类的内部定义的类，内部类中不能定义静态成员（静态成员不是对象的特性，只是为了找一个容身之处，所以需要放到一个类中而已，这么一点小事，你还要把它放到类内部的一个类中，过分了啊！提供内部类，不是为让你干这种事情，无聊，不让你干。我想可能是既然静态成员类似c语言的全局变量，而内部类通常是用于创建内部对象用的，所以，把“全局变量”放在内部类中就是毫无意义的事情，既然是毫无意义的事情，就应该被禁止），内部类可以直接访问外部类中的成员变量，内部类可以定义在外部类的方法外面，也可以定义在外部类的方法体中，代码如下：

```
public class Outer{
    int out_x  = 0;
    public void method(){
        Inner1 inner1 = new Inner1();
```

```
        //在方法体内部定义的内部类
        class Inner2
        {
            public void method() {
                out_x = 3;
            }
        }
        Inner2 inner2 = new Inner2();
    }

    //在方法体外面定义的内部类
    public class Inner1 {

    }
}
```

在方法体外面定义的内部类的访问类型可以是public,protecte,默认的，private等4种类型，这就好像类中定义的成员变量有4种访问类型一样，它们决定这个内部类的定义对其他类是否可见；对于这种情况，我们也可以在外面创建内部类的实例对象，创建内部类的实例对象时，一定要先创建外部类的实例对象，然后用这个外部类的实例对象去创建内部类的实例对象，代码如下：

```
Outer outer = new Outer();
Outer.Inner1 inner1 = outer.new Innner1();
```

在方法内部定义的内部类前面不能有访问类型修饰符，就好像方法中定义的局部变量一样，但这种内部类的前面可以使用final或abstract修饰符。这种内部类对其他类是不可见的其他类无法引用这种内部类，但是这种内部类创建的实例对象可以传递给其他类访问。这种内部类必须是先定义，后使用，即内部类的定义代码必须出现在使用该类之前，这与方法中的局部变量必须先定义后使用的道理也是一样的。这种内部类可以访问方法体中的局部变量，但是，该局部变量前必须加final修饰符。

对于这些细节，只要在eclipse写代码试试，根据开发工具提示的各类错误信息就可以马上了解到。

在方法体内部还可以采用如下语法来创建一种匿名内部类，即定义某一接口或类的子类的同时，还创建了该子类的实例对象，无需为该子类定义名称：

```
public class Outer {
    public void start(){
        new Thread(new Runnable() {
            @Override
            public void run() {

            }
        }).start();
    }
}
```

最后，在方法外部定义的内部类前面可以加上static关键字，从而成为Static Nested Class，它不再具有内部类的特性，所有，从狭义上讲，它不是内部类。Static Nested Class与普通类在运行时的行为和功能上没有什么区别，只是在编程引用时的语法上有一些差别，它可以定义成public、protected、默认的、private等多种类型，而普通类只能定义成public和默认的这两种类型。在外面引用Static Nested Class类的名称为“外部类名.内部类名”。在外面不需要创建外部类的实例对象，就可以直接创建Static Nested Class，例如，假设Inner是定义在Outer类中的Static Nested Class，那么可以使用如下语句创建Inner类：

```
Outer.Inner inner = new Outer.Inner();
```

由于static Nested Class不依赖于外部类的实例对象，所以，static Nested Class能访问外部类的非static成员变量。当在外部类中访问Static Nested Class时，可以直接使用Static Nested Class的名字，而不需要加上外部类的名字了，在Static Nested Class中也可以直接引用外部类的static的成员变量，不需要加上外部类的名字。

在静态方法中定义的内部类也是Static Nested Class，这时候不能在类前面加static关键字，静态方法中的Static Nested Class与普通方法中的内部类的应用方式很相似，它除了可以直接访问外部类中的static的成员变量，还可以访问静态方法中的局部变量，但是，该局部变量前必须加final修饰符。

备注：首先根据你的印象说出你对内部类的总体方面的特点：例如，在两个地方可以定义，可以访问外部类的成员变量，不能定义静态成员，这是大的特点。然后再说一些细节方面的知识，例如，几种定义方式的语法区别，静态内部类，以及匿名内部类。

60、String s = "Hello";s = s + " world!";这两行代码执行后，原始的String对象中的内容到底变了没有?

没有。因为String被设计成不可变(immutable)类，所以它的所有对象都是不可变对象。在这段代码中，s原先指向一个String对象，内容是 "Hello"，然后我们对s进行了+操作，那么s所指向的那个对象是否发生了改变呢？答案是没有。这时，s不指向原来那个对象了，而指向了另一个String对象，内容为"Hello world!"，原来那个对象还存在于内存之中，只是s这个引用变量不再指向它了。

通过上面的说明，我们很容易导出另一个结论，如果经常对字符串进行各种各样的修改，或者说，不可预见的修改，那么使用String来代表字符串的话会引起很大的内存开销。因为 String对象建立之后不能再改变，所以对于每一个不同的字符串，都需要一个String对象来表示。这时，应该考虑使用StringBuffer类，它允许修改，而不是每个不同的字符串都要生成一个新的对象。并且，这两种类的对象转换十分容易。

同时，我们还可以知道，如果要使用内容相同的字符串，不必每次都new一个String。例如我们要在构造器中对一个名叫s的String引用变量进行初始化，把它设置为初始值，应当这样做：

```
public class Demo {
    private String s;
    ...
    public Demo {
        s = "Initial Value";
    }
    ...
}
```

后者每次都会调用构造器，生成新对象，性能低下且内存开销大，并且没有意义，因为String对象不可改变，所以对于内容相同的字符串，只要一个String对象来表示就可以了。也就说，多次调用上面的构造器创建多个对象，他们的String类型属性s都指向同一个对象。 上面的结论还基于这样一个事实：对于字符串常量，如果内容相同，Java认为它们代表同一个String对象。而用关键字new调用构造器，总是会创建一个新的对象，无论内容是否相同。

至于为什么要把String类设计成不可变类，是它的用途决定的。其实不只String，很多Java标准类库中的类都是不可变的。在开发一个系统的时候，我们有时候也需要设计不可变类，来传递一组相关的值，这也是面向对象思想的体现。不可变类有一些优点，比如因为它的对象是只读的，所以多线程并发访问也不会有任何问题。当然也有一些缺点，比如每个不同的状态都要一个对象来代表，可能会造成性能上的问题。所以Java标准类库还提供了一个可变版本，即StringBuffer。

61、String s = new String("xyz");创建了几个String Object? 二者之间有什么区别?

两个对象，”xyz”对应一个对象，这个对象放在字符串常量缓冲区，常量”xyz”不管出现多少遍，都是缓冲区中的那一个。New String每写一遍，就创建一个新的对象，它一句那个常量”xyz”对象的内容来创建出一个新String对象。如果以前就用过’xyz’，这句代表就不会创建”xyz”自己了，直接从缓冲区拿。

62、StringBuffer与StringBuilder的区别

StringBuffer和StringBuilder类都表示内容可以被修改的字符串，**StringBuilder是线程不安全的，运行效率高**，如果一个字符串变量是在方法里面定义，这种情况只可能有一个线程访问它，不存在不安全的因素了，则用StringBuilder。如果要在类里面定义成员变量，并且这个类的实例对象会在多线程环境下使用，那么最好用StringBuffer。

63、final,finally, finalize的区别。

- final 用于声明**属性，方法和类**，分别表示属性不可变，方法不可覆盖，类不可继承。 内部类要访问局部变量，局部变量必须定义成final类型。
- finally是**异常处理语句结构**的一部分，表示总是执行。
- finalize是Object类的一个方法，在垃圾收集器执行的时候会调用被回收对象的此方法，可以覆盖此方法提供垃圾收集时的其他资源回收，例如关闭文件等。JVM不保证此方法总被

> 调用。

64、Java中的异常处理机制的简单原理和应用。

> 异常是指java程序运行时（非编译）所发生的非正常情况或错误，与现实生活中的事件很相似，现实生活中的事件可以包含事件发生的时间、地点、人物、情节等信息，可以用一个对象来表示，Java使用面向对象的方式来处理异常，它把程序中发生的每个异常也都分别封装到一个对象来表示的，该对象中包含有异常的信息。
>
> Java对异常进行了分类，不同类型的异常分别用不同的Java类表示，所有异常的根类为java.lang.Throwable，Throwable下面又派生了两个子类：Error和Exception，Error 表示应用程序本身无法克服和恢复的一种严重问题，程序只有死的份了，例如，说内存溢出和线程死锁等系统问题。Exception表示程序还能够克服和恢复的问题，其中又分为系统异常和普通异常，系统异常是软件本身缺陷所导致的问题，也就是软件开发人员考虑不周所导致的问题，软件使用者无法克服和恢复这种问题，但在这种问题下还可以让软件系统继续运行或者让软件死掉，例如，数组脚本越界（ArrayIndexOutOfBoundsException），空指针异常（NullPointerException）、类转换异常（ClassCastException）；普通异常是运行环境的变化或异常所导致的问题，是用户能够克服的问题，例如，网络断线，硬盘空间不够，发生这样的异常后，程序不应该死掉。
>
> java为系统异常和普通异常提供了不同的解决方案，编译器强制普通异常必须try..catch处理或用throws声明继续抛给上层调用方法处理，所以普通异常也称为checked异常，而系统异常可以处理也可以不处理，所以，编译器不强制用try..catch处理或用throws声明，所以系统异常也称为unchecked异常。
>
> 提示答题者：就按照三个级别去思考：虚拟机必须宕机的错误，程序可以死掉也可以不死掉的错误，程序不应该死掉的错误。

多线程/高并发

65、stop()和suspend()方法为何不推荐使用?

> **反对使用stop()，是因为它不安全**。它会解除由线程获取的所有锁定，而且如果对象处于一种不连贯状态，那么其他线程能在那种状态下检查和修改它们。结果很难检查出真正的问题所在。
>
> **suspend()方法容易发生死锁**。调用suspend()的时候，目标线程会停下来，但却仍然持有在这之前获得的锁定。此时，其他任何线程都不能访问锁定的资源，除非被"挂起"的线程恢复运行。对任何线程来说，如果它们想恢复目标线程，同时又试图使用任何一个锁定的资源，就会造成死锁。所以不应该使用suspend()，而应在自己的Thread类中置入一个标志，指出线程应该活动还是挂起。若标志指出线程应该挂起，便用wait()命其进入等待状态。若标志指出线程应当恢复，则用一个notify()重新启动线程。

66、sleep() 和 wait() 有什么区别?

> sleep就是正在执行的线程主动让出cpu，cpu去执行其他线程，在sleep指定的时间过后，cpu才会回到这个线程上继续往下执行，如果当前线程进入了同步锁，sleep方法并不会释放锁，即使当前线程使用sleep方法让出了cpu，但其他被同步锁挡住了的线程也无法得到执行。wait是指在一个已经进入了同步锁的线程内，让自己暂时让出同步锁，以便其他正在等待此锁的线程可以得到同步锁并运行，只有其他线程调用了notify方法（notify并不释放锁，只是告诉调用过wait方法的线程可以去参与获得锁的竞争了，但不是马上得到锁，因为锁还在别人手里，别人还没释放。如果notify方法后面的代码还有很多，需要这些代码执行完后才会释放锁，可以在notfiy方法后增加一个等待和一些代码，看看效果），调用wait方法的线程就会解除wait状态和程序可以再次得到锁后继续向下运行。

67、同步和异步有何异同，在什么情况下分别使用他们?

> 如果数据将在线程间共享。例如正在写的数据以后可能被另一个线程读到，或者正在读的数据可能已经被另一个线程写过了，那么这些数据就是共享数据，必须进行同步存取。
>
> 当应用程序在对象上调用了一个需要花费很长时间来执行的方法，并且不希望让程序等待方法的返回时，就应该使用异步编程，在很多情况下采用异步途径往往更有效率。

68、当一个线程进入一个对象的一个synchronized方法后，其它线程是否可进入此对象的其它方法?

> - 其他方法前是否加了synchronized关键字，如果没加，则能。
> - 如果这个方法内部调用了wait，则可以进入其他synchronized方法。
> - 如果其他个方法都加了synchronized关键字，并且内部没有调用wait，则不能。
> - 如果其他方法是static，它用的同步锁是当前类的字节码，与非静态的方法不能同步，因为非静态的方法用的是this。

69、简述synchronized和java.util.concurrent.locks.Lock的异同?

> 主要相同点：Lock能完成synchronized所实现的所有功能。
>
> 主要不同点：Lock有比synchronized更精确的线程语义和更好的性能。synchronized会自动释放锁，而Lock一定要求程序员手工释放，并且必须在finally从句中释放。Lock还有更强大的功能，例如，它的tryLock方法可以非阻塞方式去拿锁。
>
> 举例说明（对下面的题用lock进行了改写）

```
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

public class ThreadTest {

    /**
     * @param args
     */
```

```
    private int j;
    private Lock lock = new ReentrantLock();
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        ThreadTest tt = new ThreadTest();
        for(int i=0;i<2;i++)
        {
            new Thread(tt.new Adder()).start();
            new Thread(tt.new Subtractor()).start();
        }
    }

    private class Subtractor implements Runnable
    {

        @Override
        public void run() {
            // TODO Auto-generated method stub
            while(true)
            {
                /*synchronized (ThreadTest.this) {
                    System.out.println("j--=" + j--);
                    //这里抛异常了，锁能释放吗?
                }*/
                lock.lock();
                try
                {
                    System.out.println("j--=" + j--);
                }finally
                {
                    lock.unlock();
                }
            }
        }

    }

    private class Adder implements Runnable
    {

        @Override
        public void run() {
            // TODO Auto-generated method stub
            while(true)
            {
                /*synchronized (ThreadTest.this) {
                System.out.println("j++=" + j++);
                }*/
                lock.lock();
```

```
                try
                {
                    System.out.println("j++=" + j++);
                }finally
                {
                    lock.unlock();
                }
            }
        }

    }
}
```

70、概括的解释下线程的几种可用状态

- 新建 new。
- 就绪 放在可运行线程池中，等待被线程调度选中，获取cpu。
- 运行 获得了cpu。
- 阻塞
  - 等待阻塞 执行wait() 。
  - 同步阻塞 获取对象的同步琐时，同步锁被别的线程占用。
  - 其他阻塞 执行了sleep()或join()方法)。
- 死亡。

71、什么是ThreadLocal?

ThreadLocal用于创建线程的本地变量，我们知道一个对象的所有线程会共享它的全局变量，所以这些变量不是线程安全的，我们可以使用同步技术。但是当我们不想使用同步的时候，我们可以选择ThreadLocal变量。

每个线程都会拥有他们自己的Thread变量，它们可以使用get()\set()方法去获取他们的默认值或者在线程内部改变他们的值。ThreadLocal实例通常是希望它们同线程状态关联起来是private static属性。

72、run()和start()区别

run( )：只是调用普通run方法

start( )：启动了线程,由Jvm调用run方法

启动一个线程是调用start()方法，使线程所代表的虚拟处理机处于可运行状态，这意味着它可以由JVM调度并执行。这并不意味着线程就会立即运行。run()方法可以产生必须退出的标志来停止一个线程。

73、请说出你所知道的线程同步的方法

> **wait()**：使一个线程处于等待状态，并且释放所持有的对象的lock。 **sleep()**：使一个正在运行的线程处于睡眠状态，是一个静态方法，调用此方法要捕捉InterruptedException异常。 **notify()**：唤醒一个处于等待状态的线程，注意的是在调用此方法的时候，并不能确切的唤醒某一个等待状态的线程，而是由JVM确定唤醒哪个线程，而且不是按优先级。 **notityAll()**：唤醒所有处入等待状态的线程，注意并不是给所有唤醒线程一个对象的锁，而是让它们竞争。

74、线程调度和线程控制

> **线程调度（优先级）：**
>
> 与线程休眠类似，线程的优先级仍然无法保障线程的执行次序。只不过，优先级高的线程获取CPU资源的概率较大，优先级低的并非没机会执行。 线程的优先级用1-10之间的整数表示，数值越大优先级越高，默认的优先级为5。 在一个线程中开启另外一个新线程，则新开线程称为该线程的子线程，子线程初始优先级与父线程相同。
>
> **线程控制**
>
> - **sleep( )** //线程休眠 **join( )** //线程加入 **yield( )** //线程礼让 **setDaemon( )** //线程守护
>
> **中断线程**
>
> - **stop( ) interrupt( )** ==(首先选用)==

75、什么是线程饿死，什么是活锁?

> 当所有线程阻塞，或者由于需要的资源无效而不能处理，不存在非阻塞线程使资源可用。
> JavaAPI中线程活锁可能发生在以下情形：
>
> - 当所有线程在序中执行Object.wait(0)，参数为0的wait方法。程序将发生活锁直到在相应的对象上有线程调用Object.notify()或者Object.notifyAll()。
> - 当所有线程卡在无限循环中。

76、多线程中的忙循环是什么?

> 忙循环就是程序员用循环让一个线程等待，不像传统方法wait(), sleep() 或 yield() 它们都放弃了CPU控制，而忙循环不会放弃CPU，它就是在运行一个空循环。这么做的目的是为了保留CPU缓存。
>
> 在多核系统中，一个等待线程醒来的时候可能会在另一个内核运行，这样会重建缓存。为了避免重建缓存和减少等待重建的时间就可以使用它了。

77、volatile 变量是什么？volatile 变量和 atomic 变量有什么不同

> volatile则是保证了所修饰的变量的可见。因为volatile只是在保证了同一个变量在多线程中的可见性，所以它更多是用于修饰作为开关状态的变量，即Boolean类型的变量。

> volatile多用于修饰类似开关类型的变量、Atomic多用于类似计数器相关的变量、其它多线程并发操作用synchronized关键字修饰。
>
> **volatile 有两个功用**：
>
> - 这个变量不会在多个线程中存在复本，直接从内存读取。
> - 这个关键字会禁止指令重排序优化。也就是说，在 volatile 变量的赋值操作后面会有一个内存屏障（生成的汇编代码上），读操作不会被重排序到内存屏障之前。

78、volatile 类型变量提供什么保证？能使得一个非原子操作变成原子操作吗

> volatile 提供 happens-before 的保证，确保一个线程的修改能对其他线程是可见的。
>
> 在Java 中除了 long 和 double 之外的所有基本类型的读和赋值，都是原子性操作。而64位的 long 和 double 变量由于会被JVM当作两个分离的32位来进行操作，所以不具有原子性，会产生字撕裂问题。但是当你定义long或double变量时，如果使用 volatile关键字，就会获到（简单的赋值与返回操作的）原子性。

集合框架

79、ArrayList和Vector的区别

> 这两个类都实现了List接口（List接口继承了Collection接口），他们都是有序集合，即存储在这两个集合中的元素的位置都是有顺序的，相当于一种动态的数组，我们以后可以按位置索引号取出某个元素，并且其中的数据是允许重复的，这是HashSet之类的集合的最大不同处，HashSet之类的集合不可以按索引号去检索其中的元素，也不允许有重复的元素（本来题目问的与hashset没有任何关系，但为了说清楚ArrayList与Vector的功能，我们使用对比方式，更有利于说明问题）。接着才说ArrayList与Vector的区别，这主要包括两个方面。
>
> - 同步性：
>
> Vector是线程安全的，也就是说是它的方法之间是线程同步的，而ArrayList是线程序不安全的，它的方法之间是线程不同步的。如果只有一个线程会访问到集合，那最好是使用ArrayList，因为它不考虑线程安全，效率会高些；如果有多个线程会访问到集合，那最好是使用Vector，因为不需要我们自己再去考虑和编写线程安全的代码。
>
> 备注：对于Vector&ArrayList、Hashtable&HashMap，要记住线程安全的问题，记住Vector与Hashtable是旧的，是java一诞生就提供了的，它们是线程安全的，ArrayList与HashMap是java2时才提供的，它们是线程不安全的。所以，我们讲课时先讲老的。
>
> - 数据增长：
>
> ArrayList与Vector都有一个初始的容量大小，当存储进它们里面的元素的个数超过了容量时，就需要增加ArrayList与Vector的存储空间，每次要增加存储空间时，不是只增加一个存储单元，而是增加多个存储单元，每次增加的存储单元的个数在内存空间利用与程序效率之间要取得一定的平衡。Vector默认增长为原来两倍，而ArrayList的增长策略在文档中没有明确规定（从源代码看到的是增长为原来的1.5倍）。ArrayList与Vector都可以设置初始的空间大小，Vector还可以设置增长的空间大小，而ArrayList没有提供设置增长空间的方法。

> 总结：即Vector增长原来的一倍，ArrayList增加原来的0.5倍。

说说ArrayList,Vector, LinkedList的存储性能和特性

> ArrayList和Vector都是使用数组方式存储数据，此数组元素数大于实际存储的数据以便增加和插入元素，它们都允许直接按序号索引元素，但是插入元素要涉及数组元素移动等内存操作，所以索引数据快而插入数据慢，Vector由于使用了synchronized方法（线程安全）。
>
> 通常性能上较ArrayList差，而LinkedList使用双向链表实现存储，按序号索引数据需要进行前向或后向遍历，但是插入数据时只需要记录本项的前后项即可，所以插入速度较快 。
>
> ArrayList在查找时速度快，LinkedList在插入与删除时更具优势。

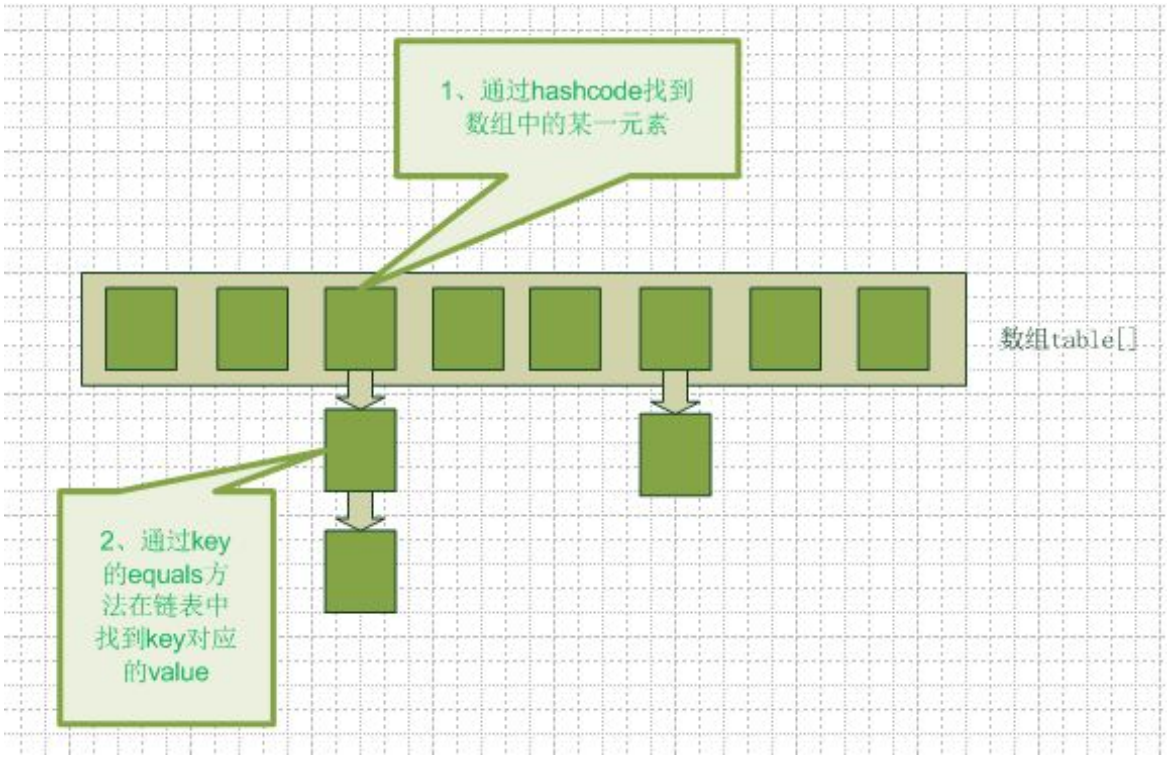
80、HashMap和Hashtable的区别

> HashMap是Hashtable的轻量级实现（非线程安全的实现），他们都完成了Map接口，主要区别在于HashMap允许空（null）键值（key）,由于非线程安全，在只有一个线程访问的情况下，效率要高于Hashtable。
>
> HashMap允许将null作为一个entry的key或者value，而Hashtable不允许。
>
> HashMap把Hashtable的contains方法去掉了，改成containsvalue和containsKey。因为contains方法容易让人引起误解。
>
> Hashtable继承自Dictionary类，而HashMap是Java1.2引进的Map interface的一个实现。
>
> 最大的不同是，Hashtable的方法是Synchronize的，而HashMap不是，在多个线程访问Hashtable时，不需要自己为它的方法实现同步，而HashMap 就必须为之提供外同步。
>
> Hashtable和HashMap采用的hash/rehash算法都大概一样，所以性能不会有很大的差异。
>
> 就HashMap与Hashtable主要从三方面来说。
>
> - 历史原因
>
> Hashtable是基于陈旧的Dictionary类的，HashMap是Java 1.2引进的Map接口的一个实现。
>
> - 同步性
>
> Hashtable是线程安全的，也就是说是同步的，而HashMap是线程序不安全的，不是同步的。
>
> - 值：
>
> 只有HashMap可以让你将空值作为一个表的条目的key或value。

81、快速失败(fail-fast)和安全失败(fail-safe)的区别是什么?

> Iterator的安全失败是基于对底层集合做拷贝，因此，它不受源集合上修改的影响。java.util 包下面的所有的集合类都是快速失败的，而java.util.concurrent包下面的所有的类都是安全失败的。快速失败的迭代器会抛出ConcurrentModificationException异常，而安全失败的迭代器永远不会抛出这样的异常。

82、hashmap的数据结构

> 在java编程语言中，最基本的结构就是两种，一个是数组，另外一个是模拟指针（引用），所有的数据结构都可以用这两个基本结构来构造的，hashmap也不例外。Hashmap实际上是一个数组和链表的结合体（在数据结构中，一般称之为“链表散列“）
>
> 
>

83、HashMap的工作原理是什么?

> Java中的HashMap是以键值对(key-value)的形式存储元素的。HashMap需要一个hash 函数，它使用hashCode()和equals()方法来向集合/从集合添加和检索元素。当调用put() 方法的时候，HashMap会计算key的hash值，然后把键值对存储在集合中合适的索引上。 如果key已经存在了，value会被更新成新值。HashMap的一些重要的特性是它的容量 (capacity)，负载因子 (load factor)和扩容极限(threshold resizing)。

84、Hashmap什么时候进行扩容呢?

> 当hashmap中的元素个数超过数组大小*loadFactor时，就会进行数组扩容，loadFactor的默认值为0.75，也就是说，默认情况下，数组大小为16，那么当hashmap中元素个数超过16*0.75=12的时候，就把数组的大小扩展为2*16=32，即扩大一倍，然后重新计算每个元素在数组中的位置，而这是一个非常消耗性能的操作，所以如果我们已经预知hashmap中元素的个数，那么预设元素的个数能够有效的提高hashmap的性能。比如说，我们有1000个元素new HashMap(1000), 但是理论上来讲new HashMap(1024)更合适，不过上面annegu已经说过，即使

> *是1000，hashmap也自动会将其设置为1024。 但是new HashMap(1024)还不是更合适的，因为0.75*1000 < 1000, 也就是说为了让0.75 * size > 1000, 我们必须这样new HashMap(2048)才最合适，既考虑了&的问题，也避免了resize的问题。

85、List、Map、Set三个接口，存取元素时，各有什么特点?

> 这样的题属于随意发挥题：这样的题比较考水平，两个方面的水平：一是要真正明白这些内容，二是要有较强的总结和表述能力。如果你明白，但表述不清楚，在别人那里则等同于不明白。
>
> 首先，List与Set具有相似性，它们都是单列元素的集合，所以，它们有一个功共同的父接口，叫Collection。Set里面不允许有重复的元素，所谓重复，即不能有两个相等（注意，不是仅仅是相同）的对象 ，即假设Set集合中有了一个A对象，现在我要向Set集合再存入一个B对象，但B对象与A对象equals相等，则B对象存储不进去，所以，Set集合的add方法有一个boolean的返回值，当集合中没有某个元素，此时add方法可成功加入该元素时，则返回true，当集合含有与某个元素equals相等的元素时，此时add方法无法加入该元素，返回结果为false。Set取元素时，没法说取第几个，只能以Iterator接口取得所有的元素，再逐一遍历各个元素。
>
> List表示有先后顺序的集合， 注意，不是那种按年龄、按大小、按价格之类的排序。当我们多次调用add(Obj e)方法时，每次加入的对象就像火车站买票有排队顺序一样，按先来后到的顺序排序。有时候，也可以插队，即调用add(int index,Obj e)方法，就可以指定当前对象在集合中的存放位置。一个对象可以被反复存储进List中，每调用一次add方法，这个对象就被插入进集合中一次，其实，并不是把这个对象本身存储进了集合中，而是在集合中用一个索引变量指向这个对象，当这个对象被add多次时，即相当于集合中有多个索引指向了这个对象，如图x所示。List除了可以以Iterator接口取得所有的元素，再逐一遍历各个元素之外，还可以调用get(index i)来明确说明取第几个。
>
> Map与List和Set不同，它是双列的集合，其中有put方法，定义如下：put(obj key,obj value)，每次存储时，要存储一对key/value，不能存储重复的key，这个重复的规则也是按equals比较相等。取则可以根据key获得相应的value，即get(Object key)返回值为key 所对应的value。另外，也可以获得所有的key的结合，还可以获得所有的value的结合，还可以获得key和value组合成的Map.Entry对象的集合。
>
> List 以特定次序来持有元素，可有重复元素。Set 无法拥有重复元素,内部排序。Map 保存key-value值，value可多值。
>
> HashSet按照hashcode值的某种运算方式进行存储，而不是直接按hashCode值的大小进行存储。例如，"abc" ---> 78，"def" ---> 62，"xyz" ---> 65在hashSet中的存储顺序不是62,65,78，这些问题感谢以前一个叫崔健的学员提出，最后通过查看源代码给他解释清楚，看本次培训学员当中有多少能看懂源码。LinkedHashSet按插入的顺序存储，那被存储对象的hashcode方法还有什么作用呢？学员想想!hashset集合比较两个对象是否相等，首先看hashcode方法是否相等，然后看equals方法是否相等。new 两个Student插入到HashSet中，看HashSet的size，实现hashcode和equals方法后再看size。
>
> 同一个对象可以在Vector中加入多次。往集合里面加元素，相当于集合里用一根绳子连接到了目标对象。往HashSet中却加不了多次的。

86、Set里的元素是不能重复的，那么用什么方法来区分重复与否呢? 是用==还是equals()? 它们有何区别?

> Set里的元素是不能重复的，元素重复与否是使用equals()方法进行判断的。
>
> equals()和==方法决定引用值是否指向同一对象equals()在类中被覆盖，为的是当两个分离的对象的内容和类型相配的话，返回真值。

87、两个对象值相同(x.equals(y) == true)，但却可有不同的hash code，这句话对不对?

> 对。如果对象要保存在HashSet或HashMap中，它们的equals相等，那么，它们的hashcode值就必须相等。
>
> 如果不是要保存在HashSet或HashMap，则与hashcode没有什么关系了，这时候hashcode不等是可以的，例如arrayList存储的对象就不用实现hashcode，当然，我们没有理由不实现，通常都会去实现的。

88、heap和stack有什么区别。

> Java的内存分为两类，一类是**栈内存**，一类是**堆内存**。栈内存是指程序进入一个方法时，会为这个方法单独分配一块私属存储空间，用于存储这个方法内部的局部变量，当这个方法结束时，分配给这个方法的栈会释放，这个栈中的变量也将随之释放。
>
> 堆是与栈作用不同的内存，一般用于存放不放在当前方法栈中的那些数据，例如，使用new创建的对象都放在堆里，所以，它不会随方法的结束而消失。方法中的局部变量使用final修饰后，放在堆中，而不是栈中。

89、Java集合类框架的基本接口有哪些?

> 集合类接口指定了一组叫做元素的对象。集合类接口的每一种具体的实现类都可以选择以它 自己的方式对元素进行保存和排序。有的集合类允许重复的键，有些不允许。
>
> Java集合类提供了一套设计良好的支持对一组对象进行操作的接口和类。Java集合类里面 最基本的接口有：
>
> **Collection**：代表一组对象，每一个对象都是它的子元素。
>
> **Set**：不包含重复元素的Collection。
>
> **List**：有顺序的collection，并且可以包含重复元素。
>
> **Map**：可以把键(key)映射到值(value)的对象，键不能重复。

90、HashSet和TreeSet有什么区别?

> HashSet是由一个hash表来实现的，因此，它的元素是无序的。add()，remove()，contains()

> TreeSet是由一个树形的结构来实现的，它里面的元素是有序的。因此，add()， remove()，contains()方法的时间复杂度是O(logn)。

91、HashSet的底层实现是什么?

> 通过看源码知道HashSet的实现是依赖于HashMap的，HashSet的值都是存储在HashMap中的。在HashSet的构造法中会初始化一个HashMap对象，HashSet不允许值重复，因此，HashSet的值是作为HashMap的key存储在HashMap中的，当存储的值已经存在时返回false。

92、LinkedHashMap的实现原理?

> LinkedHashMap也是基于HashMap实现的，不同的是它定义了一个Entry header，这个header不是放在Table里，它是额外独立出来的。LinkedHashMap通过继承hashMap中的Entry,并添加两个属性Entry before,after,和header结合起来组成一个双向链表，来实现按插入顺序或访问顺序排序。LinkedHashMap定义了排序模式accessOrder，该属性为boolean型变量，对于访问顺序，为true；对于插入顺序，则为false。一般情况下，不必指定排序模式，其迭代顺序即为默认为插入顺序。

93、为什么集合类没有实现Cloneable和Serializable接口?

> 克隆(cloning)或者是序列化(serialization)的语义和含义是跟具体的实现相关的。因此，应该 由集合类的具体实现来决定如何被克隆或者是序列化。

94、什么是迭代器(Iterator)?

> Iterator接口提供了很多对集合元素进行迭代的方法。每一个集合类都包含了可以返回迭代 器实例的迭代方法。迭代器可以在迭代的过程中删除底层集合的元素,但是不可以直接调用集合的remove(Object Obj)删除，可以通过迭代器的remove()方法删除。

95、Iterator和ListIterator的区别是什么?

> 下面列出了他们的区别：
>
> Iterator可用来遍历Set和List集合，但是ListIterator只能用来遍历List。
>
> Iterator对集合只能是前向遍历，ListIterator既可以前向也可以后向。
>
> ListIterator实现了Iterator接口，并包含其他的功能，比如：增加元素，替换元素，获取前一个和后一个元素的索引，等等。

96、数组(Array)和列表(ArrayList)有什么区别？什么时候应该使用Array而不是ArrayList?

> Array可以包含基本类型和对象类型，ArrayList只能包含对象类型。
>
> Array大小是固定的，ArrayList的大小是动态变化的。
>
> ArrayList处理固定大小的基本数据类型的时候，这种方式相对比较慢。

97、Java集合类框架的最佳实践有哪些?

> - 假如元素的大小是固 定的，而且能事先知道，我们就应该用Array而不是ArrayList。
> - 有些集合类允许指定初始容量。因此，如果我们能估计出存储的元素的数目，我们可以设置 初始容量来避免重新计算hash值或者是扩容。
> - 为了类型安全，可读性和健壮性的原因总是要使用泛型。同时，使用泛型还可以避免运行时的ClassCastException。
> - 使用JDK提供的不变类(immutable class)作为Map的键可以避免为我们自己的类实现hashCode()和equals()方法。
> - 编程的时候接口优于实现。
> - 底层的集合实际上是空的情况下，返回长度是0的集合或者是数组，不要返回null。

98、Set里的元素是不能重复的，那么用什么方法来区分重复与否呢? 是用==还是equals()? 它们有何区别

> Set里的元素是不能重复的，那么用iterator()方法来区分重复与否。equals()是判读两个Set是否相等
>
> equals()和==方法决定引用值是否指向同一对象equals()在类中被覆盖，为的是当两个分离的对象的内容和类型相配的话，返回真值

99、Comparable和Comparator接口是干什么的？列出它们的区别。

> Java提供了只包含一个compareTo()方法的Comparable接口。这个方法可以个给两个对象排序。具体来说，它返回负数，0，正数来表明输入对象小于，等于，大于已经存在的对象。
>
> Java提供了包含compare()和equals()两个方法的Comparator接口。compare()方法用来给两个输入参数排序，返回负数，0，正数表明第一个参数是小于，等于，大于第二个参数。equals()方法需要一个对象作为参数，它用来决定输入参数是否和comparator相等。只有当输入参数也是一个comparator并且输入参数和当前comparator的排序结果是相同的时 候，这个方法才返回true。

100、Collection和Collections的区别

> collection是集合类的上级接口,继承与它的接口主要是set和list。
>
> collections类是针对集合类的一个帮助类.它提供一系列的静态方法对各种集合的搜索,排序,线程安全化等操作。

IO流

101、字节流与字符流的区别

要把一片二进制数据数据逐一输出到某个设备中，或者从某个设备中逐一读取一片二进制数据，不管输入输出设备是什么，我们要用统一的方式来完成这些操作，用一种抽象的方式进行描述，这个抽象描述方式起名为IO流，对应的抽象类为OutputStream和InputStream ，不同的实现类就代表不同的输入和输出设备，它们都是针对字节进行操作的。

在应用中，经常要完全是字符的一段文本输出去或读进来，用字节流可以吗？计算机中的一切最终都是二进制的字节形式存在。对于“中国”这些字符，首先要得到其对应的字节，然后将字节写入到输出流。读取时，首先读到的是字节，可是我们要把它显示为字符，我们需要将字节转换成字符。由于这样的需求很广泛，人家专门提供了字符流的包装类。

底层设备永远只接受**字节数据**，有时候要写字符串到底层设备，需要将字符串转成字节再进行写入。字符流是字节流的包装，字符流则是直接接受字符串，它内部将串转成字节，再写入底层设备，这为我们向IO设别写入或读取字符串提供了一点点方便。

字符向字节转换时，要注意编码的问题，因为字符串转成字节数组，

其实是转成该字符的某种编码的字节形式，读取也是反之的道理。

讲解字节流与字符流关系的代码案例：

```
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.InputStreamReader;
import java.io.PrintWriter;

public class IOTest {
     public static void main(String[] args) throws Exception {
           String str = "中国人";
           /*FileOutputStream fos  = new FileOutputStream("1.txt");
           fos.write(str.getBytes("UTF-8"));
           fos.close();*/
           /*FileWriter fw = new FileWriter("1.txt");
           fw.write(str);
           fw.close();*/
           PrintWriter pw = new PrintWriter("1.txt","utf-8");
           pw.write(str);
           pw.close();
           /*FileReader fr = new FileReader("1.txt");
           char[] buf = new char[1024];
           int len = fr.read(buf);
           String myStr = new String(buf,0,len);
```

```
            System.out.println(myStr);*/
            /*FileInputStream fr = new FileInputStream("1.txt");
            byte[] buf = new byte[1024];
            int len = fr.read(buf);
            String myStr = new String(buf,0,len,"UTF-8");
            System.out.println(myStr);*/
            BufferedReader br = new BufferedReader(new
InputStreamReader(new FileInputStream("1.txt"),"UTF-8"));
            String myStr = br.readLine();
            br.close();
            System.out.println(myStr);
        }
}
```

102、什么是Java序列化，如何实现Java序列化？或者请解释Serializable接口的作用。

> 我们有时候将一个Java对象变成字节流的形式传出去或者从一个字节流中恢复成一个Java对象，例如，要将Java对象存储到硬盘或者传送给网络上的其他计算机，这个过程我们可以自己写代码去把一个Java对象变成某个格式的字节流再传输，但是，jre本身就提供了这种支持，我们可以调用OutputStream的writeObject方法来做，如果要让Java 帮我们做，要被传输的对象必须实现serializable接口，这样，javac编译时就会进行特殊处理，编译的类才可以被writeObject方法操作，这就是所谓的序列化。需要被序列化的类必须实现Serializable接口，该接口是一个mini接口，其中没有需要实现的方法，implements Serializable只是为了标注该对象是可被序列化的。
>
> 例如，在web开发中，如果对象被保存在了Session中，tomcat在重启时要把Session对象序列化到硬盘，这个对象就必须实现Serializable接口。如果对象要经过分布式系统进行网络传输或通过rmi等远程调用，这就需要在网络上传输对象，被传输的对象就必须实现Serializable接口。

数据库

103、请简洁描述MySQL中InnoDB支持的四种事务隔离级别名称，以及逐级之间的区别?

> SQL标准定义的四个隔离级别为：
>
> **read uncommited**：读到未提交数据
>
> **read committed**：脏读，不可重复读
>
> **repeatable read**：可重读
>
> **serializable**：串行事物

104、在MySQL中ENUM的用法是什么?

> ENUM是一个字符串对象，用于指定一组预定义的值，并可在创建表时使用。
>
> SQL语法如下：

```
Create table size(name ENUM('Smail,'Medium','Large');
```

105、CHAR和VARCHAR的区别?

> CHAR和VARCHAR类型在存储和检索方面有所不同
>
> CHAR列长度固定为创建表时声明的长度，长度值范围是1到255
>
> 当CHAR值被存储时，它们被用空格填充到特定长度，检索CHAR值时需删除尾随空格。

106、列的字符串类型可以是什么?

> 字符串类型是：
>
> SET
>
> BLOB
>
> ENUM
>
> CHAR
>
> TEXT
>
> VARCHAR

107、MySQL中使用什么存储引擎?

> 存储引擎称为表类型，数据使用各种技术存储在文件中。
>
> 技术涉及：
>
> Storage mechanism
>
> Locking levels
>
> Indexing
>
> Capabilities and functions.

108、TIMESTAMP在UPDATE CURRENT_TIMESTAMP数据类型上做什么?

> 创建表时TIMESTAMP列用Zero更新。只要表中的其他字段发生更改，UPDATE CURRENT_TIMESTAMP修饰符就将时间戳字段更新为当前时间。

109、主键和候选键有什么区别?

> 表格的每一行都由主键唯一标识,一个表只有一个主键。
>
> 主键也是候选键。按照惯例，候选键可以被指定为主键，并且可以用于任何外键引用。

110、MySQL数据库服务器性能分析的方法命令有哪些?

> Show status 一些值得监控的变量值： Bytes_received和Bytes_sent 和服务器之间来往的流量。 Com*服务器正在执行的命令。 Created在查询执行期限间创建的临时表和文件。 Handler_存储引擎操作。 Select*不同类型的联接执行计划。 Sort*几种排序信息。 Show session status like ‘Select’; Show profiles SET profiling=1; Show profiles\G Show profile;

111、LIKE和REGEXP操作有什么区别?

> LIKE和REGEXP运算符用于表示^和%。
>
> ```
> SELECT * FROM <tablename> WHERE * REGEXP "^b";
> SELECT * FROM <tablename> WHERE * LIKE "%b";
> ```

112、BLOB和TEXT有什么区别?

> **BLOB**
>
> BLOB是一个二进制对象，可以容纳可变数量的数据。有四种类型的BLOB
>
> - TINYBLOB
> - BLOB
> - MEDIUMBLOB和
> - LONGBLOB
>
> 它们只能在所能容纳价值的最大长度上有所不同。
>
> **TEXT**
>
> TEXT是一个不区分大小写的BLOB。四种TEXT类型
>
> - TINYTEXT
> - TEXT
> - MEDIUMTEXT和
> - LONGTEXT
>
> 它们对应于四种BLOB类型，并具有相同的最大长度和存储要求。
>
> BLOB和TEXT类型之间的唯一区别在于对BLOB值进行排序和比较时区分大小写，对TEXT值不区分大小写。

113、数据库的三范式?

> 第一范式：数据库表的每一个字段都是不可分割的。
>
> 第二范式：数据库表中的非主属性只依赖于主键。
>
> 第三范式：不存在非主属性对关键字的传递函数依赖关系。

114、MySQL表中允许有多少个TRIGGERS?

> 在MySQL表中允许有六个触发器，如下：
>
> BEFORE INSERT
>
> AFTER INSERT
>
> BEFORE UPDATE
>
> AFTER UPDATE
>
> BEFORE DELETE and
>
> AFTER DELETE

115、什么是通用SQL函数?

> **数学函数**
>
> - Abs（num）求绝对值
> - floor（num）向下取整
> - ceil（num）向上取整
>
> ---
>
> **字符串函数**
>
> - insert (s1,index,length,s2) 替换函数
> - S1表示被替换的字符串
> - s2表示将要替换的字符串
> - Index表示被替换的位置,从1开始
> - Lebgth表示被替换的长度
> - upper（str），ucase（str）将字母改为大写
> - lower（str），lcase（str）将字母改为小写
> - left（str，length）返回str字符串的前length个字符
> - right（str，length）返回str字符串的后length个字符
> - substring（str，index，length）返回str字符串从index位开始长度为length个字符（index从1开始）
> - reverse（str）将str字符串倒序输出

日期函数

- curdate（）、current_date( )获取当前日期
- curtime（）、current_time( )获取当前日期
- now（）获取当前日期和时间
- datediff（d1、d2）d1和d2之间的天数差
- adddate（date，num）返回date日期开始，之后num天的日期
- subdate（date，num）返回date日期开始，之前num天的日期

聚合函数

- Count（字段）根据某个字段统计总记录数（当前数据库保存到多少条数据）
- sum（字段）计算某个字段的数值总和
- avg（字段）计算某个字段的数值的平均值
- Max（字段）、min（字段）求某个字段最大或最小值

116、MySQL中有哪几种锁?

MyISAM支持表锁，InnoDB支持表锁和行锁，默认为行锁

表级锁：开销小，加锁快，不会出现死锁。锁定粒度大，发生锁冲突的概率最高，并发量最低

行级锁：开销大，加锁慢，会出现死锁。锁力度小，发生锁冲突的概率小，并发度最高

117、MySQL数据优化

- **优化数据类型**
  - 避免使用NULL，NULL需要特殊处理,大多数时候应该使用NOT NULL，或者使用一个特殊的值，如0，-1作为默认值。
  - 仅可能使用更小的字段，MySQL从磁盘读取数据后是存储到内存中的，然后使用cpu周期和磁盘I/O读取它，这意味着越小的数据类型占用的空间越小.
- **小心字符集转换**
  - 客户端或应用程序使用的字符集可能和表本身的字符集不一样，这需要MySQL在运行过程中隐含地进行转换，此外，要确定字符集如UTF-8是否支持多字节字符，因此它们需要更多的存储空间。
- **优化count(my_col)和count(*)**
- **优化子查询**
  - 遇到子查询时，MySQL查询优化引擎并不是总是最有效的，这就是为什么经常将子查询转换为连接查询的原因了，优化器已经能够正确处理连接查询了，当然要注意的一点是，确保连接表(第二个表)的连接列是有索引的，在第一个表上MySQL通常会相对于第二个表的查询子集进行一次全表扫描，这是嵌套循环算法的一部分。
- **优化UNION**
  - 在跨多个不同的数据库时使用UNION是一个有趣的优化方法，UNION从两个互不关联的表中返回数据，这就意味着不会出现重复的行，同时也必须对数据进行排序，我

> 们知道排序是非常耗费资源的，特别是对大表的排序。
>
> - UNION ALL可以大大加快速度，如果你已经知道你的数据不会包括重复行，或者你不在乎是否会出现重复的行，在这两种情况下使用UNION ALL更适合。此外，还可以在应用程序逻辑中采用某些方法避免出现重复的行，这样UNION ALL和UNION返回的结果都是一样的，但UNION ALL不会进行排序。

118、MySQL的关键字

> 添加索引：
>
> ```
> alter table tableName add 索引（索引字段）
> ```
>
> 主键：primary key
>
> 唯一：unique
>
> 全局：fulltext
>
> 普通：index
>
> 多列： index index_name
>
> 页级:引擎 BDB。次锁定相邻的一组记录。
>
> 表级:引擎 MyISAM ， 理解为锁住整个表，可以同时读，写不行。 行级:引擎 INNODB ， 单独的一行记录加锁，对指定的记录进行加锁，这样其它进程还是可以对同一个表中的其它记录进行操作。 表级锁速度快，但冲突多，行级冲突少，但速度慢。

119、存储引擎

> 存储引擎说白了就是如何存储数据、如何为存储的数据建立索引和如何更新、查询数据等技术的实现方法。
>
> - MyISAM：这种引擎是mysql最早提供的。这种引擎又可以分为静态MyISAM、动态MyISAM 和压缩MyISAM三种：
>   - 静态MyISAM：如果数据表中的各数据列的长度都是预先固定好的，服务器将自动选择这种表类型。因为数据表中每一条记录所占用的空间都是一样的，所以这种表存取和更新的效率非常高。当数据受损时，恢复工作也比较容易做。
>   - 动态MyISAM：如果数据表中出现varchar、text或BLOB字段时，服务器将自动选择这种表类型。相对于静态MyISAM，这种表存储空间比较小，但由于每条记录的长度不一，所以多次修改数据后，数据表中的数据就可能离散的存储在内存中，进而导致执行效率下降。同时，内存中也可能会出现很多碎片。因此，这种类型的表要经常用optimize table 命令或优化工具来进行碎片整理。
>   - 压缩MyISAM：以上说到的两种类型的表都可以用myisamchk工具压缩。这种类型的表进一步减小了占用的存储，但是这种表压缩之后不能再被修改。另外，因为是压缩数据，所以这种表在读取的时候要先时行解压缩。
>
> 但是，不管是何种MyISAM表，目前它都不支持事务，行级锁和外键约束的功能。

- MyISAM Merge引擎：这种类型是MyISAM类型的一种变种。合并表是将几个相同的MyISAM表合并为一个虚表。常应用于日志和数据仓库。
- InnoDB：InnoDB表类型可以看作是对MyISAM的进一步更新产品，它提供了事务、行级锁机制和外键约束的功能。
- memory(heap)：这种类型的数据表只存在于内存中。它使用散列索引，所以数据的存取速度非常快。因为是存在于内存中，所以这种类型常应用于临时表中。
- archive：这种类型只支持select 和 insert语句，而且不支持索引。
- Desc[ribe] tablename：查看数据表的结构。
- show engines：命令可以显示当前数据库支持的存储引擎情况。

120、数据库备份

必须要在未登录状态下

- 导出整个数据库

```
mysqldump -u 用户名 -p 数据库名 > 导出的文件名
```

- 导出一个表

```
mysqldump -u 用户名 -p 数据库名 表名> 导出的文件名
```

- 导出一个数据库结构

```
mysqldump -u dbuser -p -d --add-drop-table dbname >d:/dbname_db.sql
```

-d 没有数据 --add-drop-table 在每个create语句之前增加一个drop table

121、truncate delete drop的区别：

drop(DDL语句)：是不可逆操作，会将表所占用空间全部释放掉；

truncate(DDL语句)：只针对于删除表的操作，在删除过程中不会激活与表有关的删除触发器并且不会把删除记录放在日志中；当表被truncate后，这个表和索引会恢复到初始大小；

delete(DML语句)：可以删除表也可以删除行，但是删除记录会被计入日志保存，而且表空间大小不会恢复到原来；

执行速度：drop>truncate>delete。

122、Redis是什么？两句话做下概括

是一个完全开源免费的key-value内存数据库 2. 通常被认为是一个数据结构服务器，主要是因为其有着丰富的数据结构 strings、map、 list、sets、 sorted sets。

- Redis使用最佳方式是全部数据in-memory。

- Redis更多场景是作为Memcached的替代者来使用。
- 当需要除key/value之外的更多数据类型支持时，使用Redis更合适。
- 当存储的数据不能被剔除时，使用Redis更合适。

123、Redis（管道，哈希）

- Redis不仅仅支持简单的k/v类型的数据，同时还提供list，set，zset，hash等数据结构的存储。
- Redis支持数据的备份，即master-slave模式的数据备份。
- Redis支持数据的持久化，可以将内存中的数据保持在磁盘中，重启的时候可以再次加载进行使用。

124、Redis实现原理或机制

redis是一个key-value存储系统。和Memcached类似，但是解决了断电后数据完全丢失的情况，而且她支持更多无化的value类型，除了和string外，还支持lists（链表）、sets（集合）和zsets（有序集合）几种数据类型。这些数据类型都支持push/pop、add/remove及取交集并集和差集及更丰富的操作，而且这些操作都是原子性的。

Redis是一种基于客户端-服务端模型以及请求/响应协议的TCP服务。这意味着通常情况下一个请求会遵循以下步骤：

客户端向服务端发送一个查询请求，并监听Socket返回，通常是以阻塞模式，等待服务端响应。服务端处理命令，并将结果返回给客户端。

在服务端未响应时，客户端可以继续向服务端发送请求，并最终一次性读取所有服务端的响应。

Redis管道技术最显著的优势是提高了 redis 服务的性能。

分区是分割数据到多个Redis实例的处理过程，因此每个实例只保存key的一个子集。

通过利用多台计算机内存的和值，允许我们构造更大的数据库。

通过多核和多台计算机，允许我们扩展计算能力；通过多台计算机和网络适配器，允许我们扩展网络带宽。

redis的一些特性在分区方面表现的不是很好：

涉及多个key的操作通常是不被支持的。举例来说，当两个set映射到不同的redis实例上时，你就不能对这两个set执行交集操作。

涉及多个key的redis事务不能使用。

当使用分区时，数据处理较为复杂，比如你需要处理多个rdb/aof文件，并且从多个实例和主机备份持久化文件。

增加或删除容量也比较复杂。redis集群大多数支持在运行时增加、删除节点的透明数据平衡的能力，但是类似于客户端分区、代理等其他系统则不支持这项特性。然而，一种叫做presharding的技术对此是有帮助的。

125、Redis有两种类型分区

> 最简单的分区方式是按范围分区，就是映射一定范围的对象到特定的Redis实例。
>
> 比如，ID从0到10000的用户会保存到实例R0，ID从10001到 20000的用户会保存到R1，以此类推。
>
> 这种方式是可行的，并且在实际中使用，不足就是要有一个区间范围到实例的映射表。这个表要被管理，同时还需要各 种对象的映射表，通常对Redis来说并非是好的方法。
>
> 哈希分区：另外一种分区方法是hash分区。这对任何key都适用，也无需是object_name:这种形式，像下面描述的一样简单：
>
> 用一个hash函数将key转换为一个数字，比如使用crc32 hash函数。对key foobar执行crc32(foobar)会输出类似93024922的整数。
>
> 对这个整数取模，将其转化为0-3之间的数字，就可以将这个整数映射到4个Redis实例中的一个了。93024922 % 4 = 2，就是说key foobar应该被存到R2实例中。注意：取模操作是取除的余数，通常在多种编程语言中用%操作符实现。
>
> 实际上，上面的集群模式还存在两个问题：
>
> 1. 扩容问题：
>
>   因为使用了一致性哈稀进行分片，那么不同的key分布到不同的Redis-Server上，当我们需要扩容时，需要增加机器到分片列表中，这时候会使得同样的key算出来落到跟原来不同的机器上，这样如果要取某一个值，会出现取不到的情况，对于这种情况，Redis的作者提出了一种名为Pre-Sharding的方式：
>
>   Pre-Sharding方法是将每一个台物理机上，运行多个不同断口的Redis实例，假如有三个物理机，每个物理机运行三个Redis实际，那么我们的分片列表中实际有9个Redis实例，当我们需要扩容时，增加一台物理机，步骤如下：
>
>   1. 在新的物理机上运行Redis-Server；
>   2. 该Redis-Server从属于(slaveof)分片列表中的某一Redis-Server（假设叫RedisA）；
>   3. 等主从复制(Replication)完成后，将客户端分片列表中RedisA的IP和端口改为新物理机上Redis-Server的IP和端口；
>   4. 停止RedisA。
>
>   这样相当于将某一Redis-Server转移到了一台新机器上。Prd-Sharding实际上是一种在线扩容的办法，但还是很依赖Redis本身的复制功能的，如果主库快照数据文件过大，这个复制的过程也会很久，同时会给主库带来压力。所以做这个拆分的过程最好选择为业务访问低峰时段进行。
>
> 2. 单点故障问题：
>
>   还是用到Redis主从复制的功能，两台物理主机上分别都运行有Redis-Server，其中一个Redis-Server是另一个的从库，采用双机热备技术，客户端通过虚拟IP访问主库的物理IP，当主库宕机时，切换到从库的物理IP。只是事后修复主库时，应该将之前的从库改为主库（使用命令slaveof no one），主库变为其从库（使命令slaveof IP PORT），这样才能保证修复期间新增数据的一致性。

126、MongoDB

> 非关系型数据库(NoSql),Mongo DB很好的实现了面向对象的思想(OO思想),在Mongo DB中 每一条记录都是一个Document对象。Mongo DB最大的优势在于所有的数据持久操作都无需开发人员手动编写SQL语句,直接调用方法就可以轻松的实现CRUD操作.

127、MongoDB特点

> 高性能、易部署、易使用，存储数据非常方便。主要功能特性有：
>
> 面向集合存储，易存储对象类型的数据。
>
> 模式自由。
>
> 支持动态查询。
>
> 支持完全索引，包含内部对象。
>
> 支持查询。
>
> 支持复制和故障恢复。
>
> 使用高效的二进制数据存储，包括大型对象（如视频等）。
>
> 自动处理碎片，以支持云计算层次的扩展性
>
> 支持Python，PHP，Ruby，Java，C，C#，Javascript，Perl及C++语言的驱动程序，社区中也提供了对Erlang及.NET等平台的驱动程序。
>
> 文件存储格式为BSON（一种JSON的扩展）。
>
> 可通过网络访问。

128、MongoDB功能

> 面向集合的存储：适合存储对象及JSON形式的数据。
>
> 动态查询：Mongo支持丰富的查询表达式。查询指令使用JSON形式的标记，可轻易查询文档中内嵌的对象及数组。
>
> 完整的索引支持：包括文档内嵌对象及数组。Mongo的查询优化器会分析查询表达式，并生成一个高效的查询计划。
>
> 查询监视：Mongo包含一个监视工具用于分析数据库操作的性能。
>
> 复制及自动故障转移：Mongo数据库支持服务器之间的数据复制，支持主-从模式及服务器之间的相互复制。复制的主要目标是提供冗余及自动故障转移。
>
> 高效的传统存储方式：支持二进制数据及大型对象（如照片或图片）
>
> 自动分片以支持云级别的伸缩性：自动分片功能支持水平的数据库集群，可动态添加额外的机器。

129、MongoDB适用场景

网站数据：Mongo非常适合实时的插入，更新与查询，并具备网站实时数据存储所需的复制及高度伸缩性。

缓存：由于性能很高，Mongo也适合作为信息基础设施的缓存层。在系统重启之后，由Mongo搭建的持久化缓存层可以避免下层的数据源 过载。

大尺寸，低价值的数据：使用传统的关系型数据库存储一些数据时可能会比较昂贵，在此之前，很多时候程序员往往会选择传统的文件进行存储。

高伸缩性的场景：Mongo非常适合由数十或数百台服务器组成的数据库。Mongo的路线图中已经包含对MapReduce引擎的内置支持。

用于对象及JSON数据的存储：Mongo的BSON数据格式非常适合文档化格式的存储及查询。

130、Redis、memcache、MongoDB 对比

mongodb和memcached不是一个范畴内的东西。mongodb是文档型的非关系型数据库，其优势在于查询功能比较强大，能存储海量数据。

和memcached更为接近的是redis。它们都是内存型数据库，数据保存在内存中，通过tcp直接存取，优势是速度快，并发高，缺点是数据类型有限，查询功能不强，一般用作缓存。

1. 性能

   redis和memcache差不多，要大于mongodb。

2. 操作的便利性

   memcache数据结构单一。

   redis丰富一些，数据操作方面，redis更好一些，较少的网络IO次数。

   mongodb支持丰富的数据表达，索引，最类似关系型数据库，支持的查询语言非常丰富。

3. 内存空间的大小和数据量的大小

   redis在2.0版本后增加了自己的VM特性，突破物理内存的限制；可以对key value设置过期时间（类似memcache）。

   memcache可以修改最大可用内存,采用LRU算法。

   mongoDB适合大数据量的存储，依赖操作系统VM做内存管理，吃内存也比较厉害，服务不要和别的服务在一起。

4. 可用性（单点问题）

   redis对于单点问题，依赖客户端来实现分布式读写；主从复制时，每次从节点重新连接主节点都要依赖整个快照,无增量复制，因性能和效率问题，所以单点问题比较复杂；不支持自动sharding,需要依赖程序设定一致hash 机制。一种替代方案是，不用redis本身的复制机制，采用自己做主动复制（多份存储），或者改成增量复制的方式（需要自己实现），一致性问题和性能的权衡。

   Memcache本身没有数据冗余机制，也没必要；对于故障预防，采用依赖成熟的hash或者环状的算法，解决单点故障引起的抖动问题。

mongoDB支持master-slave,replicaset（内部采用paxos选举算法，自动故障恢复）,auto sharding机制，对客户端屏蔽了故障转移和切分机制。

5. 可靠性（持久化）

   对于数据持久化和数据恢复，redis支持（快照、AOF）：依赖快照进行持久化，aof增强了可靠性的同时，对性能有所影响。

   memcache不支持，通常用在做缓存,提升性能；

   MongoDB从1.8版本开始采用binlog方式支持持久化的可靠性。

6. 数据一致性（事务支持）

   Memcache 在并发场景下，用cas保证一致性。

   redis事务支持比较弱，只能保证事务中的每个操作连续执行。

   mongoDB不支持事务。

7. 数据分析

   mongoDB内置了数据分析的功能(mapreduce),其他不支持。

8. 应用场景

   redis：数据量较小的更性能操作和运算上。

   memcache：用于在动态系统中减少数据库负载，提升性能;做缓存，提高性能（适合读多写少，对于数据量比较大，可以采用sharding）。

   MongoDB:主要解决海量数据的访问效率问题。

131、Redis有什么用？只有了解了它有哪些特性，我们在用的时候才能扬长避短，为我们所用

1. **速度快**：使用标准C写，所有数据都在内存中完成，读写速度分别达到10万/20万。
2. **持久化**：对数据的更新采用Copy-on-write技术，可以异步地保存到磁盘上，主要有两种策略，一是根据时间，更新次数的快照（save 300 10 ）二是基于语句追加方式(Append-only file，aof) 。
3. **自动操作**：对不同数据类型的操作都是自动的，很安全。
4. 快速的主--从复制，官方提供了一个数据，Slave在21秒即完成了对Amazon网站10G key set的复制。
5. **Sharding技术**： 很容易将数据分布到多个Redis实例中，数据库的扩展是个永恒的话题，在关系型数据库中，主要是以添加硬件、以分区为主要技术形式的纵向扩展解决了很多的应用场景，但随着web2.0、移动互联网、云计算等应用的兴起，这种扩展模式已经不太适合了，所以近年来，像采用主从配置、数据库复制形式的，Sharding这种技术把负载分布到多个特理节点上去的横向扩展方式用处越来越多。

132、这里对Redis数据库做下小结

1. 提高了DB的可扩展性，只需要将新加的数据放到新加的服务器上就可以了
2. 提高了DB的可用性，只影响到需要访问的shard服务器上的数据的用户
3. 提高了DB的可维护性，对系统的升级和配置可以按shard一个个来搞，对服务产生的影响

> 较小
> 4. 小的数据库存的查询压力小，查询更快，性能更好

Java WEB

133、Tomcat的优化经验

> Tomcat作为Web服务器，它的处理性能直接关系到用户体验，下面是几种常见的优化措施:
>
> - 去掉对web.xml的监视，把jsp提前编辑成Servlet。有富余物理内存的情况，加大tomcat使用的jvm的内存。
> - 服务器资源 服务器所能提供CPU、内存、硬盘的性能对处理能力有决定性影响。
> - 对于高并发情况下会有大量的运算，那么CPU的速度会直接影响到处理速度。
> - 内存在大量数据处理的情况下，将会有较大的内存容量需求，可以用-Xmx -Xms -XX:MaxPermSize等参数对内存不同功能块进行划分。我们之前就遇到过内存分配不足，导致虚拟机一直处于full GC，从而导致处理能力严重下降。
> - 硬盘主要问题就是读写性能，当大量文件进行读写时，磁盘极容易成为性能瓶颈。最好的办法还是利用下面提到的缓存。
> - 利用缓存和压缩 对于静态页面最好是能够缓存起来，这样就不必每次从磁盘上读。这里我们采用了Nginx作为缓存服务器，将图片、css、js文件都进行了缓存，有效的减少了后端tomcat的访问。　　另外，为了能加快网络传输速度，开启gzip压缩也是必不可少的。但考虑到tomcat已经需要处理很多东西了，所以把这个压缩的工作就交给前端的Nginx来完成。　　除了文本可以用gzip压缩，其实很多图片也可以用图像处理工具预先进行压缩，找到一个平衡点可以让画质损失很小而文件可以减小很多。曾经我就见过一个图片从300多kb压缩到几十kb，自己几乎看不出来区别。
> - 采用集群 单个服务器性能总是有限的，最好的办法自然是实现横向扩展，那么组建tomcat集群是有效提升性能的手段。我们还是采用了Nginx来作为请求分流的服务器，后端多个tomcat共享session来协同工作。可以参考之前写的《利用nginx+tomcat+memcached组建web服务器负载均衡》。
> - 优化tomcat参数
>
> 这里以tomcat7的参数配置为例，需要修改conf/server.xml文件，主要是优化连接配置，关闭客户端dns查询。
>
> ```
> <Connector port="8080"
>    protocol="org.apache.coyote.http11.Http11NioProtocol"
>    connectionTimeout="20000"
>    redirectPort="8443"
>    maxThreads="500"
>    minSpareThreads="20"
>    acceptCount="100"
>    disableUploadTimeout="true"
>    enableLookups="false"
>    URIEncoding="UTF-8" />
> ```

134、HTTP请求的GET与POST方式的区别

> GET请求，请求的数据会附加在URL之后，以 **?** 分割URL和传输数据，多个参数用 **&** 连接。URL的编码格式采用的是ASCII编码，而不是unicode，即是说所有的非ASCII字符都要编码之后再传输。
>
> POST请求：POST请求会把请求的数据放置在HTTP请求包的包体中。
>
> 因此，GET请求的数据会暴露在地址栏中，而POST请求则不会。
>
> - 传输数据的大小
>
> 在HTTP规范中，没有对URL的长度和传输的数据大小进行限制。但是在实际开发过程中，对于GET，特定的浏览器和服务器对URL的长度有限制。因此，在使用GET请求时，传输数据会受到URL长度的限制。
>
> 对于POST，由于不是URL传值，理论上是不会受限制的，但是实际上各个服务器会规定对POST提交数据大小进行限制，Apache、IIS都有各自的配置。
>
> - 安全性
>
> POST的安全性比GET的高。这里的安全是指真正的安全，而不同于上面GET提到的安全方法中的安全，上面提到的安全仅仅是不修改服务器的数据。比如，在进行登录操作，通过GET请求，用户名和密码都会暴露再URL上，因为登录页面有可能被浏览器缓存以及其他人查看浏览器的历史记录的原因，此时的用户名和密码就很容易被他人拿到了。除此之外，GET请求提交的数据还可能会造成Cross-site request frogery攻击。
>
> - HTTP中的GET，POST，SOAP协议都是在HTTP上运行的。

135、解释一下什么是Servlet

> Servlet是一种服务器端的Java应用程序，具有独立于平台和协议的特性,可以生成动态的Web页面。 它担当客户请求（Web浏览器或其他HTTP客户程序）与服务器响应（HTTP服务器上的数据库或应用程序）的中间层。 Servlet是位于Web服务器内部的服务器端的Java应用程序，与传统的从命令行启动的Java应用程序不同，Servlet由Web服务器进行加载，该Web服务器必须包含支持Servlet的Java虚拟机。

136、说一说Servlet的生命周期

> Servlet有良好的生存期的定义，包括加载和实例化、初始化、处理请求以及服务结束。这个生存期由javax.servlet.Servlet接口的init、service和destroy方法表达。
>
> Servlet被服务器实例化后，容器运行其init方法，请求到达时运行其service方法，service方法自动派遣运行与请求对应的doXxx方法（doGet，doPost）等，当服务器决定将实例销毁的时候调用其destroy方法。
>
> web容器加载Servlet，生命周期开始。通过调用Servlet的init()方法进行Servlet的初始化。通过调用service()方法实现，根据请求的不同调用不同的do***()方法。结束服务，web容器调用Servlet的destroy()方法。

137、Servlet API中forward() 与redirect()的区别?

> 前者仅是容器中控制权的转向，在客户端浏览器地址栏中不会显示出转向后的地址；后者则是完全的跳转，浏览器将会得到跳转的地址，并重新发送请求链接。这样，从浏览器的地址栏中可以看到跳转后的链接地址。所以，前者更加高效，在前者可以满足需要时，尽量使用forward()方法，并且，这样也有助于隐藏实际的链接。在有些情况下，比如，需要跳转到一个其它服务器上的资源，则必须使用sendRedirect()方法。

138、request.getAttribute() 和 request.getParameter() 有何区别?

> getParameter是用来接受用post或get方法传递过来的参数的。
>
> getAttribute必须先setAttribute。
>
> request.getParameter()取得是通过容器的实现来取得通过类似post，get等方式传入的数据，request.setAttribute()和getAttribute()只是在web容器内部流转，仅仅是请求处理阶段。
>
> request.getParameter()方法传递的数据，会从Web客户端传到Web服务器端，代表HTTP请求数据。request.getParameter()方法返回String类型的数据。
>
> request.setAttribute()和getAttribute()方法传递的数据只会存在于Web容器内部。还有一点就是，HttpServletRequest 类有 setAttribute()方法，而没有setParameter() 方法。

139、 JSP有哪些动作?作用分别是什么?

> JSP共有以下6种基本动作：
>
> - jsp:include：在页面被请求的时候引入一个文件。
> - jsp:useBean：寻找或者实例化一个JavaBean。
> - jsp:setProperty：设置JavaBean的属性。
> - jsp:getProperty：输出某个JavaBean的属性。
> - jsp:forward：把请求转到一个新的页面。
> - jsp:plugin：根据浏览器类型为Java插件生成OBJECT或EMBED标记。

140、JSP的常用指令

> 常用的指令有三个：page、include、taglib;
>
> - **page**指令
>
> <%@page language=”java” contentType=”text/html;charset=gb2312” session=”true” buffer=”64kb” autoFlush=”true” isThreadSafe=”true” info=”text” errorPage=”error.jsp” isErrorPage=”true” isELIgnored=” true” pageEncoding=”gb2312” import=”java.sql.*”%>
>
> 说明:isErrorPage(是否能使用Exception对象)，isELIgnored(是否忽略表达式)
>
> - **include**指令

> <%@ include file="filename"%>
>
> - **taglib**指令
>
> <%@ taglib prefix="c"uri="http://....."%>

141、JSP和Servlet有哪些相同点和不同点，他们之间的联系是什么?

> JSP是Servlet技术的扩展，本质上是Servlet的简易方式，更强调应用的外表表达。JSP编译后是"类servlet"。Servlet和JSP最主要的不同点在于，Servlet的应用逻辑是在Java文件中，并且完全从表示层中的HTML里分离开来。而JSP的情况是Java和HTML可以组合成一个扩展名为.jsp的文件。JSP侧重于视图，Servlet主要用于控制逻辑。

142、MVC的各个部分都有那些技术来实现?如何实现?

> MVC是Model－View－Controller的简写。Model 代表的是应用的业务逻辑（通过JavaBean，EJB组件实现）， View 是应用的表示面（由JSP页面产生），Controller 是提供应用的处理过程控制（一般是一个Servlet），通过这种设计模型把应用逻辑，处理过程和显示逻辑分成不同的组件实现。这些组件可以进行交互和重用。

企业级框架

143、谈谈你对Spring的理解

> - Spring实现了**工厂模式的工厂类**（在这里有必要解释清楚什么是工厂模式），这个类名为BeanFactory（实际上是一个接口），在程序中通常BeanFactory的子类ApplicationContext。Spring相当于一个大的工厂类，在其配置文件中通过元素配置用于创建实例对象的类名和实例对象的属性。
> - Spring提供了对**IOC**良好支持，IOC是一种编程思想，是一种架构艺术，利用这种思想可以很好地实现模块之间的解耦。IOC也称为DI（Depency Injection），什么叫依赖注入呢?譬如:
>
> ```
> class Programmer {
>     Computer computer = null;
>     public void code(){
>         //Computer computer = new IBMComputer();
>         //Computer computer = beanfacotry.getComputer();
>         computer.write();
>     }
>     public void setComputer(Computer computer)
>     {
>         this.computer = computer;
>     }
> }
> ```

另外两种方式都由依赖，第一个直接依赖于目标类，第二个把依赖转移到工厂上，第三个彻底与目标和工厂解耦了。在spring的配置文件中配置片段如下：

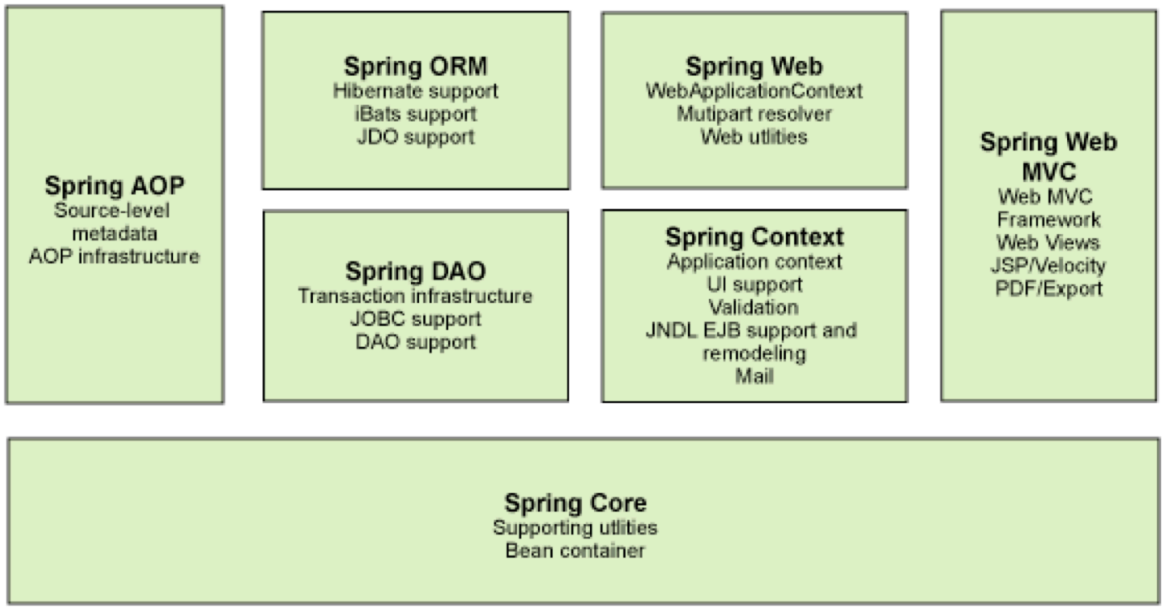
```
<bean id="computer" class="cn.itcast.interview.Computer">
</bean>
<bean id="programmer" class="cn.itcast.interview.Programmer">
    <property name="computer"  ref="computer"></property>
</bean>
```

- Spring提供了对AOP技术的良好封装， AOP称为**面向切面编程**，就是系统中有很多各不相干的类的方法，在这些众多方法中要加入某种系统功能的代码，例如，**加入日志，加入权限判断，加入异常处理**，这种应用称为AOP。实现AOP功能采用的是代理技术，客户端程序不再调用目标，而调用代理类，代理类与目标类对外具有相同的方法声明，有两种方式可以实现相同的方法声明，一是实现相同的接口，二是作为目标的子类在，JDK中采用Proxy类产生动态代理的方式为某个接口生成实现类，如果要为某个类生成子类，则可以用CGLI B。在生成的代理类的方法中加入系统功能和调用目标类的相应方法，系统功能的代理以Advice对象进行提供，显然要创建出代理对象，至少需要目标类和Advice类。spring提供了这种支持，只需要在spring配置文件中配置这两个元素即可实现代理和aop功能，例如:

```
<bean id="proxy" type="org.spring.framework.aop.ProxyBeanFactory">
    <property name="target" ref=""></property>
    <property name="advisor" ref=""></property>
</bean>
```

144、什么是Spring框架？Spring框架有哪些主要模块?

Spring框架是一个为Java应用程序的开发提供了**综合、广泛**的基础性支持的Java平台。Spring帮助开发者解决了开发中基础性的问题，使得开发人员可以专注于应用程序的开发。Spring框架本身亦是按照设计模式精心打造，这使得我们可以在开发环境中安心的集成Spring框架，不必担心Spring是如何在后台进行工作的。



Spring框架至今已集成了20多个模块。这些模块主要被分如下图所示的核心容器、数据访问/集成、Web、AOP（面向切面编程）、工具、消息和测试模块。

145、什么是控制反转(IOC)？什么是依赖注入？

控制反转是应用于软件工程领域中的，在运行时被装配器对象来绑定耦合对象的一种编程技巧，对象之间耦合关系在编译时通常是未知的。在传统的编程方式中，业务逻辑的流程是由应用程序中的早已被设定好关联关系的对象来决定的。在使用控制反转的情况下，业务逻辑的流程是由对象关系图来决定的，该对象关系图由装配器负责实例化，这种实现方式还可以将对象之间的关联关系的定义抽象化。而绑定的过程是通过“依赖注入”实现的。

控制反转是一种以给予应用程序中目标组件更多控制为目的设计范式，并在我们的实际工作中起到了有效的作用。

依赖注入是在编译阶段尚未知所需的功能是来自哪个的类的情况下，将其他对象所依赖的功能对象实例化的模式。这就需要一种机制用来激活相应的组件以提供特定的功能，所以依赖注入是控制反转的基础。否则如果在组件不受框架控制的情况下，框架又怎么知道要创建哪个组件?

在Java中依然注入有以下三种实现方式：

- 构造器注入
- Setter方法注入
- 接口注入

146、BeanFactory和ApplicationContext有什么区别?

BeanFactory 可以理解为含有bean集合的工厂类。BeanFactory 包含了种bean的定义，以便在接收到客户端请求时将对应的bean实例化。

BeanFactory还能在实例化对象的时生成协作类之间的关系。此举将bean自身与bean客户端的配置中解放出来。BeanFactory还包含了bean生命周期的控制，调用客户端的初始化方法（initialization methods）和销毁方法（destruction methods）。

从表面上看，application context如同bean factory一样具有bean定义、bean关联关系的设置，根据请求分发bean的功能。但application context在此基础上还提供了其他的功能。

1.提供了支持国际化的文本消息

2.统一的资源文件读取方式

3.已在监听器中注册的bean的事件

以下是三种较常见的 **ApplicationContext** 实现方式：

1、**ClassPathXmlApplicationContext**：从classpath的XML配置文件中读取上下文，并生成上下文定义。应用程序上下文从程序环境变量中取得。

```
ApplicationContext context = new
ClassPathXmlApplicationContext(“bean.xml”);
```

2、**FileSystemXmlApplicationContext**：由文件系统中的XML配置文件读取上下文。

```
ApplicationContext context = new
FileSystemXmlApplicationContext(“bean.xml”);
```

3、**XmlWebApplicationContext**：由Web应用的XML文件读取上下文。

147、谈谈你对Hibernate的理解

Hibernate是一个ORM框架，是对JDBC的封装。目的就是简化对数据库表访问的操作。 ORM的意思是对象关系的映射，通过实体类与数据表建立映射，就可以通过持久层操作来代替sql语句操作数据库。

hibernate的核心原理就是对象关系映射（ORM），就是通过java的反射机制来实现。 java反射机制：在java运行状态中，对于任何一个类，都能够知道这个类的所有属性和方法，就是对于任意一个对象都能够调用他的任意属性和方法，这种动态获取信息以及动态调用的方法就是功能就是java的反射机制。

以上这些就是hibernate的核心原理。

- 使用Hibernate的基本流程是：配置Configuration对象、产生SessionFactory、创建session对象，启动事务，完成CRUD操作，提交事务，关闭session。
- 使用Hibernate时，先要配置hibernate.cfg.xml文件，其中配置数据库连接信息和方言等，还要为每个实体配置相应的hbm.xml文件，hibernate.cfg.xml文件中需要登记每个hbm.xml文件。
- 在应用Hibernate时，重点要了解Session的缓存原理，级联，延迟加载和hql查询。

148、Hibernate中的update()和saveOrUpdate()的区别，session的load()和get()的区别

saveorupdate()如果传入的对象在数据库中有就做update操作，如果没有就做save操作。

save()在数据库中生成一条记录，如果数据库中有，会报错说有重复的记录。

Hibernate中**get**方法和**load**方法的根本区别:

如果你使用**load**方法，hibernate认为该id对应的对象（数据库记录）在数据库中是一定存在的，所以它可以放心的使用，它可以放心的使用代理来延迟加载该对象。在用到对象中的其他属性数据时才查询数据库，但是万一数据库中不存在该记录，那没办法，只能抛异常ObjectNotFoundException，所说的load方法抛异常是指在使用该对象的数据时，数据库中不存在该数据时抛异常，而不是在创建这个对象时。由于session中的缓存对于hibernate来说是个相当廉价的资源，所以在load时会先查一下session缓存看看该id对应的对象是否存在，不存在则创建代理。所以如果你知道该id在数据库中一定有对应记录存

在就可以使用load方法来实现延迟加载。

对于**get**方法，hibernate会确认一下该id对应的数据是否存在，首先在session缓存中查找，然后在二级缓存中查找，还没有就查数据库，数据库中没有就返回null。

虽然好多书中都这么说："get()永远只返回实体类"，但实际上这是不正确的，get方法如果在session缓存中找到了该id对应的对象，如果刚好该对象前面是被代理过的，如被load方法使用过，或者被其他关联对象延迟加载过，那么返回的还是原先的代理对象，而不是实体类对象，如果该代理对象还没有加载实体数据（就是id以外的其他属性数据），那么它会查询二级缓存或者数据库来加载数据，但是返回的还是代理对象，只不过已经加载了实体数据。

前面已经讲了，get方法首先查询session缓存，没有的话查询二级缓存，最后查询数据库；反而load方法创建时首先查询session缓存，没有就创建代理，实际使用数据时才查询二级缓存和数据库。

总之对于get和load的根本区别，一句话，Hibernate对于load方法认为该数据在数据库中一定存在，可以放心的使用代理来延迟加载，如果在使用过程中发现了问题，就抛异常；而对于get方法，Hibernate一定要获取到真实的数据，否则返回null。

149、Hibernate的inverse属性的作用?

inverse属性，是在维护关联关系的时候起作用的。表示控制权是否转移。(在一的一方起作用)。

inverse , 控制反转。inverse = false 不反转，当前方有控制权；true 控制反转，当前方没有控制权。

维护关联关系中，是否设置inverse属性：

1.保存数据有影响。

如果设置控制反转,即inverse=true, 然后通过部门方维护关联关系。在保存部门的时候，同时保存员工，数据会保存，但关联关系不会维护。即外键字段为NULL。

2.获取数据 无

3.解除关联关系？有影响。

inverse = false， 可以解除关联

> inverse = true， 当前方(部门)没有控制权，不能解除关联关系(不会生成update语句,也不会报错)
>
> 4.删除数据对关联关系的影响？有影响。
>
> inverse=false，有控制权。可以删除。先清空外键引用，再删除数据。
>
> inverse=true，没有控制权。如果删除的记录有被外键引用，会报错，违反主外键引用约束！如果删除的记录没有被引用，可以直接删除。

150、介绍一下Hibernate的二级缓存

> 按照以下思路来回答：（1）首先说清楚什么是缓存，（2）再说有了hibernate的Session就是一级缓存，即有了一级缓存，为什么还要有二级缓存，（3）最后再说如何配置Hibernate的二级缓存。
>
> （1）缓存就是把以前从数据库中查询出来和使用过的对象保存在内存中（一个数据结构中），这个数据结构通常是或类似Hashmap，当以后要使用某个对象时，先查询缓存中是否有这个对象，如果有则使用缓存中的对象，如果没有则去查询数据库，并将查询出来的对象保存在缓存中，以便下次使用。下面是缓存的伪代码：
>
> 引出hibernate的第二级缓存，用下面的伪代码分析了Cache的实现原理：

```
Dao{
    hashmap map = new map();
    User getUser(integer id)  {
        User user = map.get(id)
        if(user == null)
        {
            user = session.get(id);
            map.put(id,user);
        }
        return user;
    }
}
Dao
{
Cache cache = null
    setCache(Cache cache)
    {
        this.cache = cache
    }

    User getUser(int id)
    {
        if(cache!=null)
        {
            User user = cache.get(id);
            if(user ==null)
```

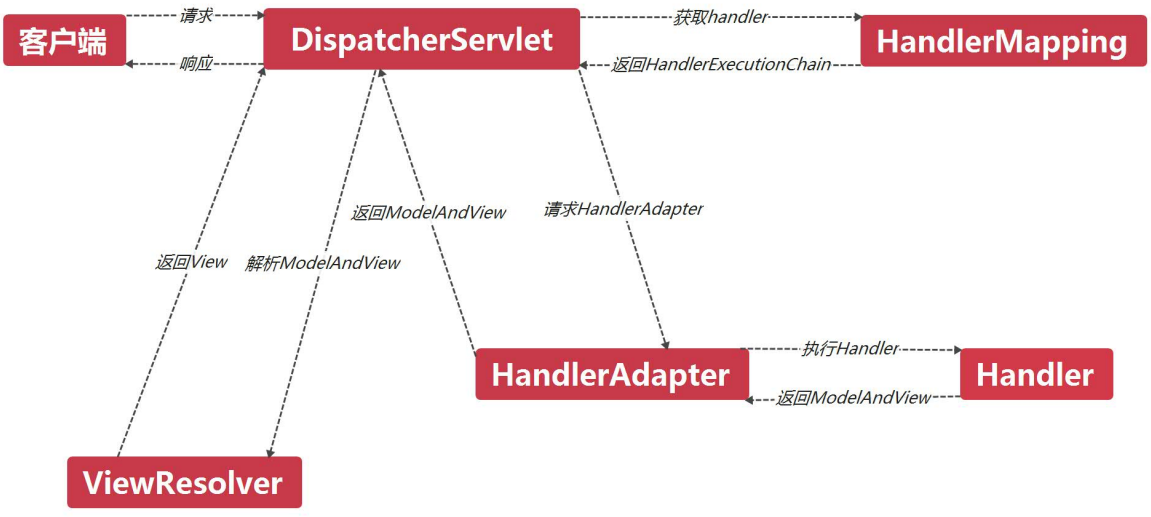
```
            {
                user = session.get(id);
                cache.put(id,user);
            }
            return user;
        }

        return session.get(id);
    }
}
```

（2）Hibernate的Session就是一种缓存，我们通常将之称为Hibernate的一级缓存，当想使用session从数据库中查询出一个对象时，Session也是先从自己内部查看是否存在这个对象，存在则直接返回，不存在才去访问数据库，并将查询的结果保存在自己内部。由于Session代表一次会话过程，一个Session与一个数据库连接相关连，所以Session最好不要长时间保持打开，通常仅用于一个事务当中，在事务结束时就应关闭。并且Session是线程不安全的，被多个线程共享时容易出现问题。通常只有那种全局意义上的缓存才是真正的缓存应用，才有较大的缓存价值，因此，Hibernate的Session这一级缓存的缓存作用并不明显，应用价值不大。Hibernate的二级缓存就是要为Hibernate配置一种全局缓存，让多个线程和多个事务都可以共享这个缓存。我们希望的是一个人使用过，其他人也可以使用，session没有这种效果。

（3）二级**缓存是独立于**Hibernate的软件部件，属于第三方的产品，多个厂商和组织都提供有缓存产品，例如，EHCache和OSCache等等。在Hibernate中使用二级缓存，首先就要在hibernate.cfg.xml配置文件中配置使用哪个厂家的缓存产品，接着需要配置该缓存产品自己的配置文件，最后要配置Hibernate中的哪些实体对象要纳入到二级缓存的管理中。明白了二级缓存原理和有了这个思路后，很容易配置起Hibernate的二级缓存。扩展知识：一个SessionFactory可以关联一个二级缓存，也即一个二级缓存只能负责缓存一个数据库中的数据，当使用Hibernate 的二级缓存后，注意不要有其他的应用或SessionFactory来更改当前数据库中的数据，这样缓存的数据就会与数据库中的实际数据不一致。

151、简单的谈一下Spring MVC的工作流程



> 流程 1、用户发送请求至前端控制器DispatcherServlet。 2、DispatcherServlet收到请求调用HandlerMapping处理器映射器。 3、处理器映射器找到具体的处理器，生成处理器对象及处理器拦截器(如果有则生成)一并返回给DispatcherServlet。 4、DispatcherServlet调用HandlerAdapter处理器适配器。 5、HandlerAdapter经过适配调用具体的处理器(Controller，也叫后端控制器)。 6、Controller执行完成返回ModelAndView。 7、HandlerAdapter将controller执行结果ModelAndView返回给DispatcherServlet。 8、DispatcherServlet将ModelAndView传给ViewReslover视图解析器。 9、ViewReslover解析后返回具体View。 10、DispatcherServlet根据View进行渲染视图（即将模型数据填充至视图中）。 11、DispatcherServlet响应用户。

152、Spring MVC Framework的特点

> - 它是基于组件技术的全部的应用对象,无论控制器和视图,还是业务对象之类的都是java组件，并且和Spring提供的其他基础结构紧密集成。
> - 不依赖于Servlet API(目标虽是如此,但是在实现的时候确实是依赖于Servlet的)。
> - 可以任意使用各种视图技术,而不仅仅局限于JSP。
> - 支持各种请求资源的映射策略。
> - 它应是易于扩展的。

153、什么是MyBatis的接口绑定，有什么好处?

> 接口映射就是在IBatis中任意定义接口,然后把接口里面的方法和SQL语句绑定，我们直接调用接口方法就可以,这样比起原来了SqlSession提供的方法我们可以有更加灵活的选择和设置。

154、接口绑定有几种实现方式,分别是怎么实现的?

> 接口绑定有两种实现方式,一种是通过注解绑定,就是在接口的方法上面加上@Select@Update等注解里面包含Sql语句来绑定,另外一种就是通过xml里面写SQL来绑定，在这种情况下,要指定xml映射文件里面的namespace必须为接口的全路径名。

155、MyBatis实现一对一有几种方式，具体怎么操作的?

> 有联合查询和嵌套查询，联合查询是几个表联合查询，只查询一次，通过在resultMap里面配置association节点配置一对一的类就可以完成；
>
> 嵌套查询是先查一个表,根据这个表里面的结果的外键id,去再另外一个表里面查询数据,也是通过association配置,但另外一个表的查询通过select属性配置。

156、MyBatis实现一对多有几种方式，怎么操作的?

> 有联合查询和嵌套查询,联合查询是几个表联合查询,只查询一次,通过在resultMap里面配置collection节点配置一对多的类就可以完成;

> 嵌套查询是先查一个表,根据这个表里面的结果的外键id,去再另外一个表里面查询数据,也是通过配置collection,但另外一个表的查询通过select节点配置。

157、MyBatis里面的动态Sql是怎么设定的?用什么语法?

> MyBatis里面的动态Sql一般是通过if节点来实现,通过OGNL语法来实现,但是如果要写的完整,必须配合where,trim节点,where节点是判断包含节点有内容就插入where,否则不插入,trim节点是用来判断如果动态语句是以and或or开始,那么会自动把这个and或者or取掉。

158、JDO是什么?

> JDO是Java对象**持久化**的新的规范，为java data object的简称,也是一个用于存取某种数据仓库中的对象的标准化API。JDO提供了透明的对象存储，因此对开发人员来说，存储数据对象完全不需要额外的代码（如JDBC API的使用）。这些繁琐的例行工作已经转移到JDO产品提供商身上，使开发人员解脱出来，从而集中时间和精力在业务逻辑上。另外，JDO很灵活，因为它可以在任何数据底层上运行。JDBC只是面向关系数据库（RDBMS）JDO更通用，提供到任何数据底层的存储功能，比如关系数据库、文件、XML以及对象数据库（ODBMS）等等，使得应用可移植性更强。
>
> 应用程序的开发人员通过访问JDO Instance , 达到访问JDO Instance 所代表的数据对象,包括:ERP，数据库系统等.使数据的存储介质对于应用的开发人员完全透明。
>
> JDO最早是由Sun召集众多的**O/R Mappin**g开发团队集中起来共同提出的，首先是通过会议确定了JDO需要包括的内容，然后正式提出一个Java规范请求（JSR-12），正式开始了JDO规范的制定。

Linux

159、LINUX下线程，GDI类的解释。

> LINUX实现的就是基于核心轻量级进程的"一对一"线程模型，一个线程实体对应一个核心轻量级进程，而线程之间的管理在核外函数库中实现。
>
> GDI类为图像设备编程接口类库。

160、绝对路径用什么符号表示？当前目录、上层目录用什么表示？主目录用什么表示? 切换目录用什么命令?

> 绝对路径： 如/etc/init.d 当前目录和上层目录： ./ ../ 主目录： ~/ 切换目录： cd

161、怎么查看当前进程？怎么执行退出？怎么查看当前路径?

> 查看当前进程： ps 执行退出： exit 查看当前路径： pwd

162、怎么清屏？怎么退出当前命令？怎么执行睡眠？怎么查看当前用户 id？查看指定帮助用什么命令？

> 清屏： clear 退出当前命令： ctrl+c 彻底退出 执行睡眠： ctrl+z 挂起当前进程 fg 恢复后台 查看当前用户 id： ”id“：查看显示目前登陆账户的 uid 和 gid 及所属分组及用户名 查看指定帮助： 如 man adduser 这个很全 而且有例子； adduser --help 这个告诉你一些常用参数； info adduesr；

163、Ls 命令执行什么功能？ 可以带哪些参数，有什么区别？

> ls 执行的功能： 列出指定目录中的目录，以及文件。 哪些参数以及区别： a 所有文件 l 详细信息，包括大小字节数，可读可写可执行的权限等。

164、建立软链接(快捷方式)，以及硬链接的命令。

> 软链接： ln -s slink source 硬链接： ln link source

165、目录创建用什么命令？创建文件用什么命令？复制文件用什么命令？

> 创建目录： mkdir
> 创建文件：典型的如 touch，vi 也可以创建文件，其实只要向一个不存在的文件输出，都会创建文件
> 复制文件： cp 7. 文件权限修改用什么命令？格式是怎么样的？
> 文件权限修改： chmod
> 格式如下：
> $ chmod u+x file 给 file 的属主增加执行权限
> $ chmod 751 file 给 file 的属主分配读、写、执行(7)的权限，给 file 的所在组分配读、执行(5)的权限，给其他用户分配执行(1)的权限
> $ chmod u=rwx,g=rx,o=x file 上例的另一种形式
> $ chmod =r file 为所有用户分配读权限
> $ chmod 444 file 同上例
> $ chmod a-wx,a+r file 同上例
> $ chmod -R u+r directory 递归地给 directory 目录下所有文件和子目录的属主分配读的权限

166、查看文件内容有哪些命令可以使用？

> vi 文件名 #编辑方式查看，可修改 cat 文件名 #显示全部文件内容 more 文件名 #分页显示文件内容 less 文件名 #与 more 相似，更好的是可以往前翻页 tail 文件名 #仅查看尾部，还可以指定行数 head 文件名 #仅查看头部,还可以指定行数

167、Linux 下命令有哪几种可使用的通配符？分别代表什么含义？

> “？”可替代单个字符。·“*”可替代任意多个字符。·方括号“[charset]”可替代 charset 集中的任何单个字符，如[a-z]，[abABC] 15. 用什么命令对一个文件的内容进行统计？(行号、单词数、字节数) wc 命令 - c 统计字节数。 - l 统计行数。 - w 统计字数。

168、Grep 命令有什么用？ 如何忽略大小写？ 如何查找不含该串的行?

> 是一种强大的文本搜索工具，它能使用正则表达式搜索文本，并把匹 配的行打印出来。
>
> ```
> grep [stringSTRING] filename grep [^string] filename
> ```

169、Linux 中进程有哪几种状态？在 ps 显示出来的信息中，分别用什么符号表示的?

> - 不可中断状态：进程处于睡眠状态，但是此刻进程是不可中断的。不可中断， 指进程不响应异步信号。
> - 暂停状态/跟踪状态：向进程发送一个 SIGSTOP 信号，它就会因响应该信号 而进入 TASK_STOPPED 状态;当进程正在被跟踪时，它处于 TASK_TRACED 这个特殊的状态。 “正在被跟踪”指的是进程暂停下来，等待跟踪它的进程对它进行操作。
> - 就绪状态：在 run_queue 队列里的状态。
> - 运行状态：在 run_queue 队列里的状态。
> - 可中断睡眠状态：处于这个状态的进程因为等待某某事件的发生（比如等待 socket 连接、等待信号量），而被挂起。
> - zombie 状态（僵尸）：父亲没有通过 wait 系列的系统调用会顺便将子进程的尸体（task_struct）也释放掉。
> - 退出状态。
>   - D 不可中断 Uninterruptible（usually IO）
>   - R正在运行，或在队列中的进程
>   - S处于休眠状态
>   - T停止或被追踪
>   - Z 僵尸进程
>   - W 进入内存交换（从内核 2.6 开始无效）
>   - X死掉的进程

170、把后台任务调到前台执行使用什么命令?把停下的后台任务在后台执行起来用什么命令?

> fg 例如：

```
# jobs

[1]+ Running /root/bin/rsync.sh &

# fg 1 bg 类似
```

171、查看当前谁在使用该主机用什么命令? 查找自己所在的终端信息用什么命令?

> w 用户名称;用户的机器名称或 tty 号;远程主机地址;用户登录系统的时间;空闲时间（作用不大）;附加到 tty（终端）的进程所用的时间（JCPU 时间）;当前进程所用时间（PCPU时间）;用户当前正在使用的命令. who 用户名、tty 号、时间日期、主机地址 whoami,id -un 命令用于显示登入的用户名 last 命令可用于显示特定用户登录系统的历史记录(last jason):用户名称;tty 设备号;历史登录时间日期;登出时间日期;总工作时间. 查找自己所在终端信息：who am i

172、通过什么命令指定命令提示符?

> \u 显示当前用户账号 \h 显示当前主机名 \W 只显示当前路径最后一个目录 \w 显示当前绝对路径（当前用户目录会以 ~代替） $PWD_{显示当前全路径}$_{显示命令行}'或者'#'符号 #： 下达的第几个命令 \d： 代表日期，格式为 weekday month date，例如："Mon Aug 1" \t： 显示时间为 24 小时格式，如：HH：MM：SS \T： 显示时间为 12 小时格式 \A： 显示时间为 24 小时格式：HH：MM \v： BASH 的版本信息 如export PS1='[\u@\h \w#]$'

173、du 和 df 的定义，以及区别？ du 显示目录或文件的大小

> df 显示每个<文件>所在的文件系统的信息，默认是显示所有文件系统。 （文件系统分配其中的一些磁盘块用来记录它自身的一些数据，如 i 节点，磁盘分布图，间接块，超级块等。 这些数据对大多数用户级的程序来说是不可见的，通常称为 Meta Data。） du 命令是用户级的程序，它不考虑 Meta Data，而 df 命令则查看文件系统的磁盘分配 图并考虑 Meta Data。 df 命令获得真正的文件系统数据，而 du 命令只查看文件系统的部分情况。

174、awk 详解

```
awk '{pattern + action}' {filenames}
 \#cat /etc/passwd |awk -F ':' '{print $1"\t"$7}' //-F 的意思是以':'分隔
root /bin/bash
 daemon /bin/sh 搜索/etc/passwd 有 root 关键字的所有行

\#awk -F: '/root/' /etc/passwd root:x:0:0:root:/root:/bin/bash
```

软件工程与设计模式

175、UML方面

> 标准建模语言UML。用例图，静态图(包括类图、对象图和包图)，行为图，交互图(顺序图、合作图)，实现图。

176、J2EE常用的设计模式？说明工厂模式。

> 总共23种，分为三大类：创建型，结构型，行为型。我只记得其中常用的6、7种，分别是：创建型（工厂、工厂方法、抽象工厂、单例），结构型（包装、适配器，组合，代理），行为（观察者，模版，策略），然后再针对你熟悉的模式谈谈你的理解即可。
>
> Java中的23种设计模式：
>
> Factory（工厂模式）, Builder（建造模式）,Factory Method（工厂方法模式），Prototype（原始模型模式），Singleton（单例模式），Facade（门面模式），Adapter（适配器模式）， Bridge（桥梁模式）， Composite（合成模式），Decorator（装饰模式），Flyweight（享元模式），Proxy（代理模式），
>
> Command（命令模式）， Interpreter（解释器模式），Visitor（访问者模式），
>
> Iterator（迭代子模式），Mediator（调停者模式），Memento（备忘录模式），
>
> Observer（观察者模式），State（状态模式），Strategy（策略模式），
>
> Template Method（模板方法模式）， Chain Of Responsibleity（责任链模式）
>
> 工厂模式：工厂模式是一种经常被使用到的模式，根据工厂模式实现的类可以根据提供的数据生成一组类中某一个类的实例，通常这一组类有一个公共的抽象父类并且实现了相同的方法，但是这些方法针对不同的数据进行了不同的操作。首先需要定义一个基类，该类的子类通过不同的方法实现了基类中的方法。然后需要定义一个工厂类，工厂类可以根据条件生成不同的子类实例。当得到子类的实例后，开发人员可以调用基类中的方法而不必考虑到底返回的是哪一个子类的实例。

177、开发中都用到了那些设计模式?用在什么场合?

> 每个模式都描述了一个在我们的环境中不断出现的问题，然后描述了该问题的解决方案的核心。通过这种方式，你可以无数次地使用那些已有的解决方案，无需在重复相同的工作。主要用到了MVC的设计模式。用来开发JSP/Servlet或者J2EE的相关应用。简单工厂模式等。

Web Service

178、Web Service名词解释。JSWDL开发包的介绍。JAXP、JAXM的解释。SOAP、UDDI,WSDL解释。

> **Web ServiceWeb Service**是基于网络的、分布式的模块化组件，它执行特定的任务，遵守具体的技术规范，这些规范使得Web Service能与其他兼容的组件进行互操作。
>
> JAXP(Java API for XML Parsing) 定义了在Java中使用DOM, SAX, XSLT的通用的接口。这样在你的程序中你只要使用这些通用的接口，当你需要改变具体的实现时候也不需要修改代码。

> **JAXM(Java API for XML Messaging)** 是为SOAP通信提供访问方法和传输机制的API。
>
> **WSDL**是一种 XML 格式，用于将网络服务描述为一组端点，这些端点对包含面向文档信息或面向过程信息的消息进行操作。这种格式首先对操作和消息进行抽象描述，然后将其绑定到具体的网络协议和消息格式上以定义端点。相关的具体端点即组合成为抽象端点（服务）。
>
> **SOAP**即简单对象访问协议(Simple Object Access Protocol)，它是用于交换XML编码信息的轻量级协议。
>
> **UDDI** 的目的是为电子商务建立标准；UDDI是一套基于Web的、分布式的、为Web Service提供的、信息注册中心的实现标准规范，同时也包含一组使企业能将自身提供的Web Service注册，以使别的企业能够发现的访问协议的实现标准。

179、CORBA是什么?用途是什么?

> **CORBA** 标准是公共对象请求代理结构(Common Object Request Broker Architecture)，由对象管理组织 (Object Management Group，缩写为 OMG)标准化。它的组成是接口定义语言(IDL), 语言绑定(binding:也译为联编)和允许应用程序间互操作的协议。 其目的为：用不同的程序设计语言书写在不同的进程中运行，为不同的操作系统开发。

180、JSWDL 开发包的介绍

> **JAXP**:(Java API for XML Parsing)定义了在Java中使用DOM, SAX, XSLT的通用的接口。
>
> **JAXM**:(Java API for XML Messaging) 是为SOAP通信提供访问方法和传输机制的API。
>
> **SOAP**:简单对象访问协议(Simple Object Access Protocol)，它是用于交换XML编码信息的轻量级协议。
>
> **UDDI**:基于Web的、分布式的、为Web Service提供的、信息注册中心的实现标准规范 。
>
> **WSDL**:是一种 XML 格式，用于将网络服务描述为一组端点，这些端点对包含面向文档信息或面向过程信息的消息进行操作。

答题技巧

> 说明，为了节省大家的时间和提高学习效率，一些过时知识点和被笔试概率极低的题目不再被收录和分析。
>
> 回答问题的思路：先正面叙述一些**基本的核心知识**，然后描述一些**特殊的东西**，最后再来一些**锦上添花的东西**。要注意有些不是锦上添花，而是画蛇添足的东西，不要随便写上。把答题像写书一样写。我要回答一个新技术的问题大概思路和步骤是：我们想干什么，怎么干，干的过程中遇到了什么问题，现在用什么方式来解决。其实我们讲课也是这样一个思路。
>
> 例如，讲ajax时，我们希望不改变原来的整个网页，而只是改变网页中的局部内容，例如，用户名校验，级联下拉列表，下拉树状菜单。用传统方式，就是浏览器自己直接向服务器发请求，服务器返回新页面会盖掉老页面，这样就不流畅了。

对本面试宝典中的题目有信心吗？本来有信心的，结果听你讲完后，就没信心了！我非常理解。因为他觉得我的太深，他想记住我的些东西，可是记不住，所以没信心了。我又问：听懂了吗？他说听懂了。你到现在只要把你的理解尽量清晰地、有条理地表达出来，就很棒了。

这套面试题主要目的是帮助那些还没有java软件开发实际工作经验，而正在努力寻找java软件开发工作的朋友在笔试时更好地赢得笔试和面试。由于这套面试题涉及的范围很泛，很广，很杂，大家不可能一天两天就看完和学完这套面试宝典，即使你已经学过了有关的技术，那么至少也需要一个月的时间才能消化和掌握这套面试宝典，所以，大家应该早作准备，从拿到这套面试宝典之日起，就要坚持在每天闲暇之余学习其中几道题目，日积月累，等到出去面试时，一切都水到渠成，面试时就自然会游刃有余了。

答题时，先答是什么，再答有什么作用和要注意什么（这部分最重要，展现自己的心得）

答案的段落分别，层次分明，条理清晰都非常重要，从这些表面的东西也可以看出一个人的习惯、办事风格、条理等。

要讲你做出答案的思路过程，或者说你记住答案的思想都写下来。把答题想着是辩论赛。答题就是给别人讲道理、摆事实。答题不局限于什么格式和形式，就是要将自己的学识展现出来！

别因为人家题目本来就模棱两可，你就心里胆怯和没底气了，不敢回答了。你要大胆地指出对方题目很模糊和你的观点，不要把面试官想得有多高，其实他和你就是差不多的，你想想，如果他把你招进去了，你们以后就是同事了，可不是差不多的吗？

关于就业薪水，如果你是应届生，那不能要高工资，好比大饼的故事，没有文凭还想拿高工资，就去中关村缺什么补什么吧！少数人基础确实很好，在校期间确实又做过一些项目，那仍然是可以要到相对高的工资的。

公司招聘程序员更看重的要用到的编码技术、而不是那些业务不太相关的所谓项目经历：

1.公司想招什么样的;2.公司面试会问什么;3.简历怎么写;4怎样达到简历上的标准（培训中心教项目的目的）

对于一些公司接到了一些项目，想招聘一些初中级的程序员过来帮助写代码，完成这个项目，你更看重的是他的专业技术功底，还是以前做过几个项目的经历呢？我们先排除掉那些编码技术功底好，又正好做过相似项目的情况，实际上，这种鱼和熊掌兼得的情况并不常见。其实公司很清楚，只要招聘进来的人技术真的很明白，那他什么项目都可以做出来，公司招人不是让你去重复做你以前的项目，而是做一个新项目，业务方面，你只要进了项目团队，自然就能掌握。所以，大多数招聘单位在招聘那些编码级别的程序员时也没指望能招聘到做过类似项目的人，也不会刻意去找做过类似项目的人，用人单位也不是想把你招进，然后把你以前做过的项目重做一遍，所以，用人单位更看重招进来的人对要用到的编码技术的功底到底怎样，技术扎实不扎实，项目则只要跟着开发团队走，自然就没问题。除非是一些非常专业的行业，要招聘特别高级的开发人员和系统分析师，招聘单位才特别注重他的项目经验和行业经验，要去找到行业高手，公司才关心项目和与你聊项目的细节，这样的人通常都不是通过常规招聘渠道去招聘进来的，而是通过各种手段挖过来的，这情况不再我今天要讨论的范围中。

技术学得明白不明白，人家几个问题就把你的深浅问出来了，只要问一些具体的技术点，就很容易看出你是真懂还是假懂，很容看出你的技术深度和实力，所以，技术是来不得半点虚假的，必须扎扎实实。

> 由于项目的种类繁多，涉及到现实生活中的各行各业，什么五花八门的业务都有，例如，酒店房间预定管理，公司车辆调度管理，学校课程教室管理，超市进销存管理，知识内容管理，等等……成千上万等等，但是，不管是什么项目，采用的无非都是我们学习的那些目前流行和常用的技术。技术好、经验丰富，则项目做出来的效率高些，程序更稳定和更容易维护些；技术差点，碰碰磕磕最后也能把项目做出来，无非是做的周期长点、返工的次数多点，程序代码写得差些，用的技术笨拙点。如果一个人不是完完全全做过某个项目，他是不太关心该项目的业务的，对其中的一些具体细节更是一窍不知，(如果我招你来做图书管理，你项目经历说你做过汽车调度，那我能问你汽车调度具体怎么回事吗？不会，所以，你很容易蒙混过去的)而一个程序员的整个职业生涯中能实实在在和完完整整做出来的项目没几个，更别说在多个不同行业的项目了，有的程序员更是一辈子都只是在做某一个行业的项目，结果他就成了这个行业的专家(专门干一件事的家伙)。所以，技术面试官通常没正好亲身经历过你简历写的那些项目，他不可能去问你写的那些项目的具体细节，而是只能泛泛地问你这个项目是多少人做的，做了多长时间，开发的过程，你在做项目的过程中有什么心得和收获，用的什么技术等面上的问题，所以，简历上的项目经历可以含有很多水分，很容易作假，技术面试官也无法在项目上甄别你的真伪。
>
> 简历该怎么写：**精通那些技术，有一些什么项目经历**
>
> 教项目是为了巩固和灵活整合运用技术，增强学习的趣味性，熟悉做项目的流程，或得一些专业课程中无法获得的特有项目经验，增强自己面试的信心。讲的项目应该真实可靠才有价值，否则，表面上是项目，实际上还是知识点的整合，对巩固技术点和增强学习的趣味性，但无法获得实际的项目经验。（项目主要是增加你经验的可信度，获得更多面试机会，真正能不能找到工作，找到好工作，主要看你键盘上的功夫了），好的面试官几下就能面出你是否真有工作经验，他们问技术以外的公司的人和事，并且问开始、过程、结果，看你怎么编。
>
> 建议大家尽量开自己的blog，坚持每天写技术blog。在简历上写上自己的blog地址，可以多转载一些技术文章。

人事问题

181、请讲一下这样一个经历：尽管其他人反对，但是你还是坚持自己的观点，并把事情继续做下去。

> 【思路】:从自己积极方面回答,比如家人和老师都希望我报考会计专业,而我对会计就是不感兴趣,毅然选择了计算机专业.我现在毕业,专业技能很强,而且有自己成熟的作品，计算机是我的事业,将继续做下去。

182、你的一位领导脾气比较急，批评下属时常常不留情面，大家的工作情绪经常受到影响。作为职员，你该怎么办?

> 【思路】:首先对领导的批评应该认真接受，不能因为领导严厉的批评而产生逆反心理，以致影响工作； 其次可以私下找机会和领导沟通，向领导反映下属因此产生的意见和情绪，婉转地说明这种情绪可能会影响工作的正常开展，至于是否接受建议、改变方法，由领导自己决定。

183、与上级意见不一是，你将怎么办?

正确回答：首先呢，作为一个员工我的是不会和上级产生争执的，如果真有意见不一致的时候，我想我会服从领导安排的 【思路】： 1、一般可以这样回答“我会给上级以必要的解释和提醒，在这种情况下，我会服从上级的意见。” 2、如果面试你的是总经理，而你所应聘的职位另有一位经理，且这位经理当时不在场，可以这样回答：“对于非原则性问题，我会服从上级的意见，对于涉及公司利益的重大问题，我希望能向更高层领导反映。”

184、你是应届毕业生，缺乏经验，如何能胜任这项工作?

正确回答：作为应届毕业生，在工作经验方面的确会有所欠缺，因此在读书期间我一直利用各种机会在这个行业里做兼职。我也发现，实际工作远比书本知识丰富、复杂。但我有较强的责任心、适应能力和学习能力，而且比较勤奋，所以在兼职中均能圆满完成各项工作，从中获取的经验也令我受益非浅。请贵公司放心，学校所学及兼职的工作经验使我一定能胜任这个职位。 【思路】： 1、 如果招聘单位对应届毕业生的应聘者提出这个问题，说明招聘单位并不真正在乎“经验”，关键看应聘者怎样回答。 2、对这个问题的回答最好要体现出应聘者的诚恳、机智、果敢及敬业。

185、你为什么觉得自己能够在这个职位上取得成就?

正确回答： 从我的经历来看，这是我的职业生涯中最适合我的一份工作。几年来，我一直在研究这个领域并且关注贵公司，一直希望能有这样的面试机会。我拥有必备的技能（简单讲述一个故事来加以说明），我非常适合这一职位，也确实能做好这份工作。

【思路】：这是一个相当宽泛的问题，它给求职者提供了一个机会，可以让求职者表明自己的热情和挑战欲。对这个问题的回答将为面试人在判断求职者是否对这个职位有足够的动力和自信心方面提供关键信息。

186、你希望5年后达到什么成就?

A.做一天和尚敲一天钟，尽人事听天命、顺其自然。

B.依我的机灵及才干，晋升到部门经理是我的中期目标。

C.自己独当一面开公司。

D.“全力以赴”是我的座右铭，希望能随着经验的增加，被赋予更多的职责及挑战。

解答：最理想的回答是D。

187、5年之内你想处于什么位置?

正确回答：我希望有机会在工厂或国内办事处工作。我也希望通过管理一个小团体发展我的管理技能。 【思路】：不要给出具体的时限或工作头衔。谈你喜欢的东西，你天生的技能，实际的问题和在你所选的领域或行业里你希望有什么机会，你希望从那些经验中学点什么。不要谈论你在那些与你所应聘的工作无关的领域或行业里的目标。这是听起来很明显的道理，但是很多求职者会犯这个错误。不经意间你就表现出了对当前的领域或行业缺乏真正的兴趣。不用

> 说，一失言马上就会把你从进一步的考虑中淘汰掉。

188、如果你有无限的时间和经济来源，你会怎样使用它们呢?

> 正确回答：我希望能参加几次不针对金融专家的有关金融管理的行政研讨会。我还希望能让我的部门放长假，把每一个人都派去参加外界的一些活动。最后，我很可能去旅游并考察一下外国竞争者，同时一路享受当地的美食，您呢? 【思路】：虽然娱乐的事情谈起来很有诱惑力，但一定要紧扣工作或与行业相关的事务，或者紧扣与你应聘的这份工作的技能相关的努力上。例如，你正在应聘教书工作，你可能对义务教授成年人读书识字的项目感兴趣。这就证明了你对自己的工作领域的激情，即对教育重要性的一种信仰，即使是作为一种兴趣而毫无报酬都无所谓。

190、假如现在是你在我们公司做首次年度总结，我该告诉你一些什么呢?

> 正确回答：您要感谢我把工作完成得很好，并说明您渴望能够继续看到我工作的好成绩。更重要的是，我希望您能告诉我，您很欣赏我为一些重要项目加班加点工作的行为，还有我富有创造性的思维是如何有助于对存在的问题提出改革方案的。【思路】：-很明显，在回答该问题时，你想给人留下积极的印象。“但愿您能更准时地出现”绝对不是一个好答案。记住，重点谈一两点你个人的优势。

191、为什么你想在这里工作?

> 正确回答：我几年前就错过了你们公司的一次招标，之后我意识到电脑产品变得越来越相近了，且零售价格的竞争愈趋激烈，以致服务成为了一家公司在竞争中脱颖而出的最好的方法。贵公司在所有的竞争者中享有最好的服务记录，而我相信从长远看，它将主宰这个行业。你的准备和调查研究工作应在这里明显表现出来。【思路】：给出一到两个你对该公司感兴趣的原因，并表明什么最激发你的兴趣。什么是你可以叙述来表明你个人对该公司的认识的最有说服力的事情呢? 它的产品还是它的员工? 答案包括公司的信誉、对该工作本身的描述，或者是跻身于该企业的欲望。

192、你对加班的看法。

> 正确回答：首先我想确认下，是何种性质的加班? 如果是我个人的工作量是在规定的时间内没有完成的话，这种情况是不会发生的，我是个注重工作效率的人。其次如果是公司业务量临时增加的话，我会接受加班。【思路】：首先，明确的告诉对方，如果是因为自己在规定的时间内没有完成工作任务的话，需要加班的情况是几乎不可能出现的，“我是个注重工作效率的人”其次，如果是因为公司业务情况或者其他的一些紧急工作的话是可以适应加班的。

193、你能给公司带来什么?

【思路】:一般外企很想知道未来的员工能为企业做什么，求职者应再次重复自己的优势，然后说：“就我的能力，我可以做一个优秀的员工在组织中发挥能力，给组织带来高效率和更多的收益”。外企喜欢求职者就申请的职位表明自己的能力，比如申请营销之类的职位，可以说：“我可以开发大量的新客户，同时，对老客户做更全面周到的服务，开发老客户的新需求和消费。”等等。

194、你认为你在学校属于好学生吗?

【思路】:企业的招聘者很精明，问这个问题可以试探出很多问题：如果求职者学习成绩好，就会说：“是的，我的成绩很好，所有的成绩都很优异。当然，判断一个学生是不是好学生有很多标准，在学校期间我认为成绩是重要的，其他方面包括思想道德、实践经验、团队精神、沟通能力也都是很重要的，我在这些方面也做得很好，应该说我是一个全面发展的学生。”如果求职者成绩不尽理想，便会说：“我认为是不是一个好学生的标准是多元化的，我的学习成绩还可以，在其他方面我的表现也很突出，比如我去很多地方实习过，我很喜欢在快节奏和压力下工作，我在学生会组织过××活动，锻炼了我的团队合作精神和组织能力。” 有经验的招聘者一听就会明白，企业喜欢诚实的求职者。

195、有人说“成功是对人有益的”，也有人说“失败是对人有益的”，你怎么看?

【思路】:成功是对努力的一种回报，一种肯定，能使人们认识到自身的价值，对自身是一种动力，能激发人们继续创新、学习的勇气！当然，成功是对人有益的。 “失败是对人有益的”，俗话说“失败乃成功之母”，它给予人们更多的是经验与坚韧顽强的精神和永不认输的斗志，所以说“失败是对人有益的”。 这类题的应对方法：辩证地看、联系地看，肯定一方但不否定另一方，两者是有机的统一。

196、如果我们单位录用了你，但工作一段时间却发现你根本不适合这个职位你怎么办?

【思路】:一段时间发现工作不适合我，有两种情况：1.如果你确实热爱这个职业，那你就要不断学习，虚心向领导和同事学习业务知识和处事经验，了解这个职业的精神内涵和职业要求，力争减少差距；2.你觉得这个职业可有可无，那还是趁早换个职业，去发现适合你的，你热爱的职业，那样你的发展前途也会大点，对单位和个人都有好处。

197、你最大的长处和弱点分别是什么？这些长处和弱点对你在企业的业绩会有什么样的影响?

【思路】:这个问题的最大陷阱在于，第一个问题实际上是两个问题，而且还要加上一个后续问题。这两个问题的陷阱并不在于你是否能认真地看待自己的长处，也不在于你是否能正确认识自己的弱点。记住，你的回答不仅是向面试人说明你的优势和劣势，也能在总体上表现你的价值观和对自身价值的看法。 长处来说，我相信我最大的优点是我有一个高度理性的头脑，能够从混乱中整理出头绪来。我最大的弱点是，对那些没有秩序感的人，可能缺乏足够的耐心。我相信我的组织才能可以帮助企业更快地实现目标，而且有时候，我处理复杂问题的能力也能影响我的同事。

198、除了工资，还有什么福利最吸引你?

> 【思路】:尽可能诚实，如果你做足了功课，你就知道他们会提供什么，回答尽可能和他们提供的相配。如果你觉得自己该得到更多，也可以多要一点。

199、有人说，善意的谎言是对的，你如何看?

> 【思路】:这个问题不能一概而论的， 它仅仅动机是善意的， 但是造成的后果好不好呢， 如果反而引起更大的伤害， 那么就得不偿失了；其次是对象，如果对象意志毅力很强，能够接受突如其来的打击，并且不喜欢别人骗他哪怕是善意的，那么善意的谎言便毫无意义，有时反而造成误会。但是善意的谎言在更多程度上都是对的可以接受的，它可以最大地减少不必要的痛苦，能够起到积极的作用。

200、领导要你4天完成一件工作，突然要你2天完成，你该怎么办?

> 【思路】:1.首先分析一下提前完成工作的可能性。 2.如果确定完不成的，那么去跟领导详谈，跟他讲道理摆事实，说明没法完成的理由。一定要有充足的理由，才能说服他。 3.如果可以完成，但是需要其他条件的配合的，那么找领导说明情况。请领导给于支持。 4.如果经过自己努力可以完成的，那么就努力完成吧。

201、你认为这些年来同事对你怎么样?

> 【思路】：面试官问这个问题的目的，主要想从你的同事对你的态度和评价上推测你这个人是什么样的，对于你来说回答这个问题要谨慎。比如：同事对我都很热心（从侧面反衬你对同事也很热心）；同事们有棘手的工作我会主动去帮助他们，所以我有事情的时候他们都来帮助我等等。

202、向面试官提出的问题

> 贵公司对这项职务的工作内容和期望目标为和？有没有什么部分是我可以努力的地方?
>
> 贵公司是否有正式或非正式教育训练?
>
> 贵公司的升迁管道如何?
>
> 贵公司的多角化经营，而且在海内外都设有分公司，将来是否有外派、轮调的机会?
>
> 贵公司能超越同业的最大利基点为何?
>
> 贵公司强调的团队合作中，其它的成员素质和特性如何?
>
> 能否为我介绍一下工作环境，或者是否有机会能参观一下贵公司?
>
> 【思路】:在面试结束前，大多数的主考官都会丢问题给求职者，最常见的就是：你有没有什么问题或疑问，想要提出来的？无论求职者是否有提出问题，其实，这个问题背后的真正含意，通常是主考官用来测试你对这份工作有多大的企图心、决心和热情。

因此，如果你害怕发问不妥当，或是不知道该从何问起，甚至回答没有问题时，都很可能会让主考官认为，你想要这份工作的企图心、决心还不够强。

相反的，求职者应该更积极、主动的利用面试最后一关的机会，适时的提出问题，这不但有助于主考官对你的印象能够加深，而且你也能趁此机会进一步了解这家公司的背景、企业文化是否适合你。

最重要的是，如果能够在面试时，提出漂亮的问题，录取的机率将会大大提高。所以，无论如何，前往面试前，先谨记10个可以反问主考官的问题，以便到时候可以提出。

至于薪水待遇、年假天数、年终奖金、福利措施等问题，有些公司的主考官在面试时，会直接向求职者提出。如果对方没有提及，对社会新鲜人来说，在找第一份工作时，比较不适合提出，除非你有对方不得不录取你的条件.另外，也有人在结束前，谦虚的请教主考官：您认为我今天的表现如何？录取的机率有多大？通常，这个问题也会让对方认为，你对这份工作抱有很大的决心和企图心，而你也可以试着从对方的回答中，约略猜测出自己成功的机率有多大，并且作为下一次面试时表现的参考！