Java 面试宝典 2020 版

前言

作为刚毕业的学生或者正在找工作的 JAVA 程序员，当你应聘一份程序设计、软件开发方面的工作的时候，招聘方面总会安排一次笔试或机试来考查你的程序设计。

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！

通过对本宝典的学习，大家应该掌握关键性的技巧，发现和完善试题的最佳解决方案。对于宝典中的具有代表性的实体，需要举一反三地钻研，相信大家无论以后遇见什么样的面试题，都可以应对自如，逢山开路、遇水搭桥。

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，“狭路相逢勇者胜”，相信自己。

本宝典不是万能钥匙，但却肯定是你工作求职的好帮手！

感谢为本宝典负责做出工作的王丽娜、王杨老师，由张晨光老师负责审定，王向南老师负责总核定。

技术问题反馈邮箱：83193980@qq.com

一. Java 基础部分

基础部分的顺序：基本语法，类相关的语法，内部类的语法，继承相关的语法，异常的语法，线程的语法，集合的语法，io 的语法，虚拟机方面的语法。

1、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。

2、Java 有没有 goto?

java 中的保留字，现在没有在 java 中使用。

3、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。

4、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。

5、short s1 = 1; s1 = s1 + 1;有什么错? short s1 = 1; s1 += 1;有什么错?

对于 short s1 = 1; s1 = s1 + 1; 由于 s1+1 运算时会自动提升表达式的类型，所以结果是 int 型，再赋值给 short 类型 s1 时，编译器将报告需要强制转换类型的错误。

对于 short s1 = 1; s1 += 1;由于 += 是 java 语言规定的运算符，java 编译器会对它进行特殊处理，因此可以正确编译。

6、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。

7、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。

8、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");
```

9、"=="和 equals 方法究竟有什么区别?

==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。

10、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！

11、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，一个 static 方法内部发出对非 static 方法的调用。

12、Integer 与 int 的区别

int 是 java 提供的 8 种原始数据类型之一。Java 为每个原始类型提供了封装类，Integer 是 java 为 int 提供的封装类。int 的默认值为 0，而 Integer 的默认值为 null，即 Integer 可以区分出未赋值和值为 0 的区别，int 则无法表达出未赋值的情况，例如，要想表达出没有参加考试和考试成绩为 0 的区别，则只能使用 Integer。在 JSP 开发中，Integer 的默认为 null，所以用 el 表达式在文本框中显示时，值为空白字符串，而 int 默认的默认值为 0，所以用 el 表达式在文本框中显示时，结果为 0，所以，int 不适合作为 web 层的表单数据的类型。

在 Hibernate 中，如果将 OID 定义为 Integer 类型，那么 Hibernate 就可以根据其值是否为 null 而判断一个对象是否是临时的，如果将 OID 定义为了 int 类型，还需要在 hbm 映射文件中设置其 unsaved-value 属性为 0。

另外，Integer 提供了多个与整数相关的操作方法，例如，将一个字符串转换成整数，Integer 中还定义了表示整数的最大值和最小值的常量。

13、Math.round(11.5)等於多少? Math.round(-11.5)等於多少?

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，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，Math.ceil(11.6)的结果为 11,Math.ceil(-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。

14、请说出作用域 public，private，protected，以及不写时的区别

这四个作用域的可见范围如下表所示。

说明：如果在修饰的元素上面没有写任何访问修饰符，则表示 friendly。

作用域	当前类	同一 package	子孙类	其他 package
public	√	√	√	√
protected	√	√	√	×
friendly	√	√	×	×
private	√	×	×	×

备注：只要记住了有 4 种访问权限，4 个访问范围，然后将全选和范围在水平和垂直方向上分别按排从小到大或从大到小的顺序排列，就很容易画出上面的图了。

15、Overload 和 Override 的区别。Overloaded 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：

1、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；

2、覆盖的方法的返回值必须和被覆盖的方法的返回一致；

3、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；

4、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：

1、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)）；

2、不能通过访问权限、返回类型、抛出的异常进行重载；

3、方法的异常类型和数目不会对重载造成影响；

4、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。

16、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。

17、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，抽象类是否可继承具体类。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。

18、写 clone()方法时，通常都有一行代码，是什么？

clone 有缺省行为，super.clone();因为首先要把父类中的成员复制到位，然后才是复制自己的成员。

19、面向对象的特征有哪些方面

面向对象的编程语言有封装、继承 、抽象、多态等 4 个主要的特征。

1 封装：

封装是保证软件部件具有优良的模块性的基础，封装的目标就是要实现软件部件的“高内聚、低耦合”，防止程序相互依赖性而带来的变动影响。在面向对象的编程语言中，对象是封装的最基本单位，面向对象的封装比传统语言的封装更为清晰、更为有力。面向对象的封装就是把描述一个对象的属性和行为的代码封装在一个“模块”中，也就是一个类中，属性用变量定义，行为用方法进行定义，方法可以直接访问同一个对象中的属性。通常情况下，**只要记住让变量和访问这个变量的方法放在一起，将一个类中的成员变量全部定义成私有的，只有这个类自己的方法才可以访问到这些成员变量，这就基本上实现对象的封装，就很容易找出要分配到这个类上的方法了，就基本上算是会面向对象的编程了。把握一个原则：把对同一事物进行操作的方法和相关的方法放在同一个类中，把方法和它操作的数据放在同一个类中。**

例如，人要在黑板上画圆，这一共涉及三个对象：人、黑板、圆，画圆的方法要分配给哪个对象呢？由于画圆需要使用到圆心和半径，圆心和半径显然是圆的属性，如果将它们在类中定义成了私有的成员变量，那么，画圆的方法必须分配给圆，它才能访问到圆心和半径这两个属性，人以后只是调用圆的画圆方法、表示给圆发给消息而已，画圆这个方法不应该分配在人这个对象上，**这就是面向对象的封装性，即将对象封装成一个高度自治和相对封闭的个体，对象状态（属性）由这个对象自己的行为（方法）来读取和改变。**一个更便于理解的例子就是，司机将火车刹住了，刹车的动作是分配给司机，还是分配给火车，显然，应该分配给火车，因为司机自身是不可能有那么大的力气将一个火车给停下来的，只有火车自己才能完成这一动作，火车需要调用内部的离合器和刹车片等多个器件协作才能完成刹车这个动作，司机刹车的过程只是给火车发了一个消息，通知火车要执行刹车动作而已。

抽象：

抽象就是找出一些事物的相似和共性之处，然后将这些事物归为一个类，这个类只考虑这些事物的相似和共性之处，并且会忽略与当前主题和目标无关的那些方面，将注意力集中在与当前目标有关的方面。例如，看到一只蚂蚁和大象，你能够想象出它们的相同之处，那就是抽象。抽象包括行为抽象和状态抽象两个方面。例如，定义一个 Person 类，如下：

```
class Person{
    String name;
    int age;
}
```

继承：

在定义和实现一个类的时候，可以在一个已经存在的类的基础之上来进行，把这个已经存在的类所定义的内容作为自己的内容，并可以加入若干新的内容，或修改原来的方法使之更适合特殊的需要，这就是继承。继承是子类自动共享父类数据和方法的机制，这是类之间的一种关系，提高了软件的可重用性和可扩展性。

多态：

多态是指程序中定义的引用变量所指向的具体类型和通过该引用变量发出的方法调用在编程时并不确定，而是在程序运行期间才确定，即一个引用变量倒底会指向哪个类的实例对象，该引用变量发出的方法调用到底是哪个类中实现的方法，必须在由程序运行期间才能决定。因为在程序运行时才确定具体的类，这样，不用修改源程序代码，就可以让引用变量绑定到各种不同的类实现上，从而导致该引用调用的具体方法随之改变，即不修改程序代码就可以改变程序运行时所绑定的具体代码，让程序可以选择多个运行状态，这就是多态性。多态性增强了软件的灵活性和扩展性。例如，下面代码中的 UserDao 是一个接口，它定义引用变量 userDao 指向的实例对象由 daofactory.getDao()在执行的时候返回，有时候指向的是 UserJdbcDao 这个实现，有时候指向的是 UserHibernateDao 这个实现，这样，不用修改源代码，就可以改变 userDao 指向的具体类实现，从而导致 userDao.insertUser()方法调用的具体代码也随之改变，即有时候调用的是 UserJdbcDao 的 insertUser 方法，有时候调用的是 UserHibernateDao 的 insertUser 方法：

```
UserDao userDao = daofactory.getDao();
userDao.insertUser(user);
```

20、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。

21、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：

1.抽象类可以有构造方法，接口中不能有构造方法。

2.抽象类中可以有普通成员变量，接口中没有普通成员变量

3.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。

4. 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。

5. 抽象类中可以包含静态方法，接口中不能包含静态方法

6. 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。

7. 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，

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。

22、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。

21、String 是最基本的数据类型吗?

基本数据类型包括 byte、int、char、long、float、double、boolean 和 short。

java.lang.String 类是 final 类型的，因此不可以继承这个类、不能修改这个类。为了提高效率

节省空间，我们应该用 StringBuffer 类

22、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。

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。

23、是否可以继承 String 类?

String 类是 final 类故不可以继承。

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

两个或一个，”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。

25、String 和 StringBuffer 的区别

JAVA 平台提供了两个类：String 和 StringBuffer，它们可以储存和操作字符串，即包含多个字符的字符数据。这个 String 类提供了数值不可改变的字符串。而这个 StringBuffer 类提供的字符串进行修改。当你知道字符数据要改变的时候你就可以使用 StringBuffer。典型地，你可以使用 StringBuffers 来动态构造字符数据。另外，String 实现了 equals 方法，new String(“abc”).equals(new String(“abc”) 的结果为 true, 而 StringBuffer 没有实现 equals 方法，所以，new StringBuffer(“abc”).equals(new StringBuffer(“abc”)的结果为 false。

在讲两者区别时，应把循环的次数搞成 10000，然后用 endTime-beginTime 来比较两者执行的

时间差异，最后还要讲讲 StringBuilder 与 StringBuffer 的区别。String 覆盖了 equals 方法和 hashCode 方法，而 StringBuffer 没有覆盖 equals 方法和 hashCode 方法，所以，将 StringBuffer 对象存储进 Java 集合类中时会出现问题。

26、数组有没有 length()这个方法? String 有没有 length()这个方法？

数组没有 length()这个方法，有 length 的属性。String 有有 length()这个方法。

27、下面这条语句一共创建了多少个对象：String s="a"+"b"+"c"+"d";

答：对于如下代码：

```
String s1 = "a";
String s2 = s1 + "b";
String s3 = "a" + "b";
System.out.println(s2 == "ab");
System.out.println(s3 == "ab");
```

第一条语句打印的结果为 false，第二条语句打印的结果为 true，这说明 javac 编译可以对字符串常量直接相加的表达式进行优化，不必要等到运行期去进行加法运算处理，而是在编译时去掉其中的加号，直接将其编译成一个这些常量相连的结果。

题目中的第一行代码被编译器在编译时优化后，相当于直接定义了一个”abcd”的字符串，所以，上面的代码应该只创建了一个 String 对象。写如下两行代码，

```
String s = "a" + "b" + "c" + "d";
System.out.println(s == "abcd");
```

最终打印的结果应该为 true。

28、try {}里有一个 return 语句，那么紧跟在这个 try 后的 finally {}里的 code 会不会被执行，什么时候被执行，在 return 前还是后?

也许你的答案是在 return 之前，但往更细地说，我的答案是在 return 中间执行，请看下面程序代码的运行结果：

```
public  class Test {

    /**
     * @param args add by zxx ,Dec 9, 2008
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.out.println(new Test().test());;
    }

    static int test()
    {
```

```
        int x = 1;
        try
        {
            return x;
        }
        finally
        {
            ++x;
        }
    }

}
```

```
---------执行结果 ---------
1
```

运行结果是 1，为什么呢？主函数调用子函数并得到结果的过程，好比主函数准备一个空罐子，当子函数要返回结果时，先把结果放在罐子里，然后再将程序逻辑返回到主函数。所谓返回，就是子函数说，我不运行了，你主函数继续运行吧，这没什么结果可言，结果是在说这话之前放进罐子里的。

39、下面的程序代码输出的结果是多少？

```
public class    smallT
{
    public static void    main(String args[])
    {
        smallT t    = new    smallT();
        int    b    =    t.get();
        System.out.println(b);
    }

    public int    get()
    {
        try
        {
            return 1 ;
        }
        finally
        {
            return 2 ;
        }
    }
}
```

返回的结果是 2。
我可以通过下面一个例子程序来帮助我解释这个答案，从下面例子的运行结果中可以发现，try 中的 return 语句调用的函数先于 finally 中调用的函数执行，也就是说 return 语句先执行，finally 语句后执行，所以，返回的结果是 2。Return 并不是让函数马上返回，而是 return 语句执行后，将把返回结果放置进函数栈中，此时函数并不是马上返回，它要执行 finally 语句后才真正开始返回。
在讲解答案时可以用下面的程序来帮助分析：

```
public  class Test {

    /**
     * @param args add by zxx ,Dec 9, 2008
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.out.println(new Test().test());;
    }

    int test()
    {
        try
        {
            return buy();
        }
        finally
        {
            return sale();
        }
    }

    int buy()
    {
        System.out.println("买买买");
        return 1;
    }
    int sale()
    {
        System.out.println("卖卖卖");
        return 2;
    }
}
```

----------执行结果----------

买买买

卖卖卖
2
结论：finally 中的代码比 return 和 break 语句后执行

40、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，例如，一段代码……
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。

41、运行时异常与一般异常有何异同？

异常表示程序运行过程中可能出现的非正常状态，运行时异常表示虚拟机的通常操作中可能遇到的异常，是一种常见运行错误。java 编译器要求方法必须声明抛出可能发生的非运行时异常，但是并不要求必须声明抛出未被捕获的运行时异常。

42、error 和 exception 有什么区别?

error 表示恢复不是不可能但很困难的情况下的一种严重问题。比如说内存溢出。不可能指望程序能处理这样的情况。 exception 表示一种设计或实现问题。也就是说，它表示如果程序运行正常，从不会发生的情况。

43、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；

44、请写出你最常见到的 5 个 runtime exception。

这道题主要考你的代码量到底多大，如果你长期写代码的，应该经常都看到过一些系统方面的异常，你不一定真要回答出 5 个具体的系统异常，但你要能够说出什么是系统异常，以及几个系统异常就可以了，当然，这些异常完全用其英文名称来写是最好的，如果实在写不出，那就用中文吧，有总比没有强！

所谓系统异常，就是…..，它们都是 RuntimeException 的子类，在 jdk doc 中查 RuntimeException 类，就可以看到其所有的子类列表，也就是看到了所有的系统异常。我比较有印象的系统异常有：NullPointerException、ArrayIndexOutOfBoundsException、ClassCastException。

46、sleep() 和 wait() 有什么区别?

（网上的答案：sleep 是线程类（Thread）的方法，导致此线程暂停执行指定时间，给执行机会给其他线程，但是监控状态依然保持，到时后会自动恢复。调用 sleep 不会释放对象锁。 wait 是 Object 类的方法，对此对象调用 wait 方法导致本线程放弃对象锁，进入等待此对象的等待锁定池，只有针对此对象发出 notify 方法（或 notifyAll）后本线程才进入对象锁定池准备获得对象锁进入运行状态。）

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。

47、同步和异步有何异同，在什么情况下分别使用他们？举例说明。

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。

48、多线程有几种实现方法?同步有几种实现方法?

多线程有两种实现方法，分别是继承 Thread 类与实现 Runnable 接口
同步的实现方面有两种，分别是 synchronized,wait 与 notify
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。
Allnotity():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。

49、启动一个线程是用 run()还是 start()? .

启动一个线程是调用 start()方法，使线程就绪状态，以后可以被调度为运行状态，一个线程必须关联一些具体的执行代码，run()方法是该线程所关联的执行代码。

50、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?

分几种情况：
1.其他方法前是否加了 synchronized 关键字，如果没加，则能。
2.如果这个方法内部调用了 wait，则可以进入其他 synchronized 方法。
3.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。
4.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。

51、线程的基本概念、线程的基本状态以及状态之间的关系

一个程序中可以有多条执行线索同时执行，一个线程就是程序中的一条执行线索，每个线程上都关联有要执行的代码，即可以有多段程序代码同时运行，每个程序至少都有一个线程，即 main 方法执行的那个线程。如果只是一个 cpu，它怎么能够同时执行多段程序呢？这是从宏观上来看的，cpu 一会执行 a 线索，一会执行 b 线索，切换时间很快，给人的感觉是 a,b 在同时执行，好比大家在同一个办公室上网，只有一条链接到外部网线，其实，这条网线一会为 a 传数据，一会为 b 传数据，由于切换时间很短暂，所以，大家感觉都在同时上网。

状态：就绪，运行，synchronize 阻塞，wait 和 sleep 挂起，结束。wait 必须在 synchronized 内部调用。
调用线程的 start 方法后线程进入就绪状态，线程调度系统将就绪状态的线程转为运行状态，遇到 synchronized 语句时，由运行状态转为阻塞，当 synchronized 获得锁后，由阻塞转为运行，在这种情况可以调用 wait 方法转为挂起状态，当线程关联的代码执行完后，线

程变为结束状态。

52、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）：

53、介绍 Collection 框架的结构

答：随意发挥题，天南海北谁便谈，只要让别觉得你知识渊博，理解透彻即可。

54、Collection 框架中实现比较要实现什么接口

comparable/comparator

55、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：.

（1）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。

（2）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。

56、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。

一.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

二.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

三.值：只有 HashMap 可以让你将空值作为一个表的条目的 key 或 value

57、List 和 Map 区别?

一个是存储单列数据的集合，另一个是存储键和值这样的双列数据的集合，List 中存储的数据是有顺序，并且允许重复；Map 中存储的数据是没有顺序的，其键是不能重复的，它的值是可以有重复的。

58、List, Set, Map 是否继承自 Collection 接口?

List，Set 是，Map 不是

59、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。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。

60、说出 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。

LinkedList 也是线程不安全的，LinkedList 提供了一些方法，使得 LinkedList 可以被当作堆栈和队列来使用。

61、去掉一个 Vector 集合中重复的元素

```
Vector newVector = new Vector();
For (int i=0;i<vector.size();i++)
```

```
{
    Object obj = vector.get(i);
    if(!newVector.contains(obj);
        newVector.add(obj);
}
```

还有一种简单的方式，HashSet set = new HashSet(vector);

62、Collection 和 Collections 的区别。

Collection 是集合类的上级接口，继承与他的接口主要有 Set 和 List.
Collections 是针对集合类的一个帮助类，他提供一系列静态方法实现对各种集合的搜索、排序、线程安全化等操作。

63、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。

64、你所知道的集合类都有哪些？主要方法？

最常用的集合类是 List 和 Map。 List 的具体实现包括 ArrayList 和 Vector，它们是可变大小的列表，比较适合构建、存储和操作任何类型对象的元素列表。 List 适用于按数值索引访问元素的情形。
Map 提供了一个更通用的元素存储方法。 Map 集合类用于存储元素对（称作"键"和"值"），其中每个键映射到一个值。

ArrayList/Vector→List
→Collection
HashSet/TreeSet→Set

Propeties→HashTable
→Map
Treemap/HashMap

我记的不是方法名，而是思想，我知道它们都有增删改查的方法，但这些方法的具体名称，我记得不是很清楚，对于 set，大概的方法是 add,remove, contains；对于 map，大概的方法就是 put,remove，contains 等，因为，我只要在 eclispe 下按点操作符，很自然的这些方法就出来了。我记住的一些思想就是 List 类会有 get(int index)这样的方法，因为它可以按顺序取元素，而 set 类中没有 get(int index)这样的方法。List 和 set 都可以迭代出所有元素，迭代时先要得到一个

iterator 对象，所以，set 和 list 类都有一个 iterator 方法，用于返回那个 iterator 对象。map 可以返回三个集合，一个是返回所有的 key 的集合，另外一个返回的是所有 value 的集合，再一个返回的 key 和 value 组合成的 EntrySet 对象的集合，map 也有 get 方法，参数是 key，返回值是 key 对应的 value。

65、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。

65、TreeSet 里面放对象，如果同时放入了父类和子类的实例对象，那比较时使用的是父类的 compareTo 方法，还是使用的子类的 compareTo 方法，还是抛异常！

（应该是没有针对问题的确切的答案，当前的 add 方法放入的是哪个对象，就调用哪个对象的 compareTo 方法，至于这个 compareTo 方法怎么做，就看当前这个对象的类中是如何编写这个方法的）
实验代码：

```
public class Parent implements Comparable {
    private int age = 0;
    public Parent(int age){
        this.age = age;
    }
    public int compareTo(Object o) {
        // TODO Auto-generated method stub
        System.out.println("method of parent");
        Parent o1 = (Parent)o;
        return age>o1.age?1:age<o1.age?-1:0;
    }

}

public class Child extends Parent {

    public Child(){
        super(3);
    }
    public int compareTo(Object o) {

            // TODO Auto-generated method stub
```

```
            System.out.println("method of child");
//          Child o1 = (Child)o;
            return 1;

    }
}

public class TreeSetTest {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        TreeSet set = new TreeSet();
        set.add(new Parent(3));
        set.add(new Child());
        set.add(new Parent(4));
        System.out.println(set.size());
    }
}
```

66、说出一些常用的类，包，接口，请各举 5 个

要让人家感觉你对java ee开发很熟，所以，不能仅仅只列core java中的那些东西，要多列你在做ssh项目中涉及的那些东西。就写你最近写的那些程序中涉及的那些类。

常用的类：BufferedReader BufferedWriter FileReader FileWirter String Integer java.util.Date，System，Class，List,HashMap

常用的包：java.lang java.io java.util
java.sql ,javax.servlet,org.apache.strtuts.action,org.hibernate
常用的接口：Remote List Map Document
NodeList ,Servlet,HttpServletRequest,HttpServletResponse,Transaction(Hibernate)、
Session(Hibernate),HttpSession

67、java 中有几种类型的流？JDK 为每种类型的流提供了一些抽象类以供继承，请说出他们分别是哪些类？

字节流，字符流。字节流继承于 InputStream OutputStream，字符流继承于 InputStreamReader OutputStreamWriter。在 java.io 包中还有许多其他的流，主要是为了提高性能和使用方便。

68、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。

68、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。

69、描述一下 JVM 加载 class 文件的原理机制?

JVM 中类的装载是由 ClassLoader 和它的子类来实现的,Java ClassLoader 是一个重要的 Java 运行时系统组件。它负责在运行时查找和装入类文件的类。

70、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。

71、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。

72、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。

73、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。

74、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，即对象编程了孤儿的时候，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。

75、能不能自己写个类，也叫 java.lang.String？

可以，但在应用的时候，需要用自己的类加载器去加载，否则，系统的类加载器永远只是去加载jre.jar包中的那个java.lang.String。由于在tomcat的web应用程序中，都是由webapp自己的类加载器先自己加载WEB-INF/classess目录中的类，然后才委托上级的类加载器加载，如果我们

在tomcat的web应用程序中写一个java.lang.String，这时候Servlet程序加载的就是我们自己写的java.lang.String，但是这么干就会出很多潜在的问题，原来所有用了java.lang.String类的都将出现问题。

虽然java提供了endorsed技术，可以覆盖jdk中的某些类，具体做法是….。但是，能够被覆盖的类是有限制范围，反正不包括java.lang这样的包中的类。

二. 算法与编程

1、编写一个程序，将 a.txt 文件中的单词与 b.txt 文件中的单词交替合并到 c.txt 文件中，a.txt 文件中的单词用回车符分隔，b.txt 文件中用回车或空格进行分隔。

```
package cn.aaaedu;

import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;

public class MainClass{
	public static void main(String[] args) throws Exception{
		FileManager a = new FileManager("a.txt",new char[]{'\n'});
		FileManager b = new FileManager("b.txt",new char[]{'\n',' '});
		FileWriter c = new FileWriter("c.txt");
		String aWord = null;
		String bWord = null;
		while((aWord = a.nextWord()) !=null ){
			c.write(aWord + "\n");
			bWord = b.nextWord();
			if(bWord != null)
				c.write(bWord + "\n");
		}
		while((bWord = b.nextWord()) != null){
			c.write(bWord + "\n");
		}
		c.close();
	}
}


class FileManager{
```

```
    String[] words = null;
    int pos = 0;
    public FileManager(String filename,char[] seperators) throws Exception{
        File f = new File(filename);
        FileReader reader = new FileReader(f);
        char[] buf = new char[(int)f.length()];
        int len = reader.read(buf);
        String results = new String(buf,0,len);
        String regex = null;
        if(seperators.length >1 ){
            regex = "" + seperators[0] + "|" + seperators[1];
        }else{
            regex = "" + seperators[0];
        }
        words = results.split(regex);
    }
    public String nextWord(){
        if(pos == words.length)
            return null;
        return words[pos++];
    }
}
```

2、编写一个程序，将 d:\java 目录下的所有.java 文件复制到 d:\jad 目录下，并将原来文件的扩展名从.java 改为.jad。

（大家正在做上面这道题，网上迟到的朋友也请做做这道题，找工作必须能编写这些简单问题的代码！）

答：**listFiles 方法接受一个 FileFilter 对象，这个 FileFilter 对象就是过虑的策略对象，不同的人提供不同的 FileFilter 实现，即提供了不同的过滤策略。**

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.FilenameFilter;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

public class Jad2Java {

    public static void main(String[] args) throws Exception {
        File srcDir = new File("java");
        if(!(srcDir.exists() && srcDir.isDirectory()))
```

```
                throw new Exception("目录不存在");
        File[] files = srcDir.listFiles(
                new FilenameFilter(){

                        public boolean accept(File dir, String name) {
                                return name.endsWith(".java");
                        }
                }
        );

        System.out.println(files.length);
        File destDir = new File("jad");
        if(!destDir.exists()) destDir.mkdir();
        for(File f :files){
                FileInputStream    fis = new FileInputStream(f);
                String destFileName = f.getName().replaceAll("\\.java$", ".jad");
                FileOutputStream fos = new FileOutputStream(new File(destDir,destFileName));
                copy(fis,fos);
                fis.close();
                fos.close();
        }
    }

    private static void copy(InputStream ips,OutputStream ops) throws Exception{
        int len = 0;
        byte[] buf = new byte[1024];
        while((len = ips.read(buf)) != -1){
                ops.write(buf,0,len);
        }
    }
}
```

由本题总结的思想及策略模式的解析：

1.

class jad2java{

1. 得到某个目录下的所有的 java 文件集合

1.1 得到目录 File srcDir = new File("d:\\java");

1.2 得到目录下的所有 java 文件：File[] files = srcDir.listFiles(new MyFileFilter());

1.3 只想得到.java 的文件： class MyFileFilter implememyts FileFilter{

```
        public boolean accept(File pathname){
                return pathname.getName().endsWith(".java")
        }
}
```

2.将每个文件复制到另外一个目录，并改扩展名

2.1 得到目标目录，如果目标目录不存在，则创建之

2.2 根据源文件名得到目标文件名，注意要用正则表达式，注意.的转义。

2.3 根据表示目录的 File 和目标文件名的字符串，得到表示目标文件的 File。

//要在硬盘中准确地创建出一个文件，需要知道文件名和文件的目录。

2.4 将源文件的流拷贝成目标文件流，拷贝方法独立成为一个方法，方法的参数采用抽象流的形式。

//方法接受的参数类型尽量面向父类，越抽象越好，这样适应面更宽广。

}

分析 listFiles 方法内部的策略模式实现原理

```
File[] listFiles(FileFilter filter){
    File[] files = listFiles();
    //Arraylist acceptedFilesList = new ArrayList();
    File[] acceptedFiles = new File[files.length];
    int pos = 0;
    for(File file: files){
        boolean accepted = filter.accept(file);
        if(accepted){
            //acceptedFilesList.add(file);
            acceptedFiles[pos++] = file;
        }
    }
    Arrays.copyOf(acceptedFiles,pos);
    //return (File[])accpetedFilesList.toArray();
}
```

3、编写一个截取字符串的函数，输入为一个字符串和字节数，输出为按字节截取的字符串，但要保证汉字不被截取半个，如“我 ABC”，4，应该截取“我 AB”，输入“我 ABC 汉 DEF”，6，应该输出“我 ABC”，而不是“我 ABC+汉的半个”。

首先要了解中文字符有多种编码及各种编码的特征。

假设 n 为要截取的字节数。

```
public static void main(String[] args) throws Exception{
    String str = "我 a 爱中华 abc 我爱中国 def';
    String str = "我 ABC 汉";
    int num = trimGBK(str.getBytes("GBK"),5);
    System.out.println(str.substring(0,num) );
}

public static int    trimGBK(byte[] buf,int n){
    int num = 0;
    boolean bChineseFirstHalf = false;
```

```
        for(int i=0;i<n;i++)
        {
            if(buf[i]<0 && !bChineseFirstHalf){
                bChineseFirstHalf = true;
            }else{
                num++;
                bChineseFirstHalf = false;
            }
        }
        return num;
    }
```

4、有一个字符串，其中包含中文字符、英文字符和数字字符，请统计和打印出各个字符的个数。

答：哈哈，其实包含中文字符、英文字符、数字字符原来是出题者放的烟雾弹。

```
String content = "自贸区 aadf 的 111 郑 bbb 安的 zz 贸易";
HashMap map = new HashMap();
for(int i=0;i<content.length;i++)
{
    char c = content.charAt(i);
    Integer num = map.get(c);
    if(num == null)
        num = 1;
    else
        num = num + 1;
    map.put(c,num);
}
for(Map.EntrySet entry : map)
{
    system.out.println(entry.getkey() + ":" + entry.getValue());
}
```

估计是当初面试的那个学员表述不清楚，问题很可能是：

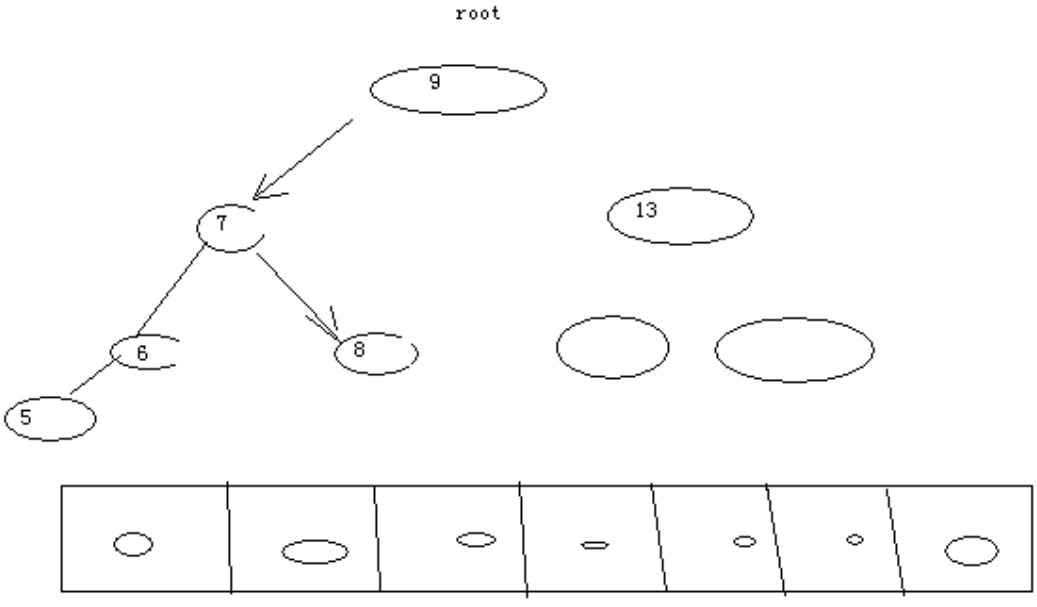
如果一串字符如"aaaabbc少林寺1512"要分别统计英文字符的数量，中文字符的数量，和数字字符的数量，假设字符中没有中文字符、英文字符、数字字符之外的其他特殊字符。

```
int engishCount;
int chineseCount;
int digitCount;
for(int i=0;i<str.length;i++)
{
    char ch = str.charAt(i);
    if(ch>='0' && ch<='9')
    {
        digitCount++
    }
```

```
    else if((ch>='a' && ch<='z') || (ch>='A' && ch<='Z'))
    {
        engishCount++;
    }
    else
    {
        chineseCount++;
    }
}
System.out.println(.............);
```

5、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，原理如下图：



代码如下：

```
package com.huawei.interview;

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();
    }
}
-----------------又一次临场写的代码--------------------------
import java.util.Arrays;
import java.util.Iterator;

public class Node {
    private Node left;
    private Node right;
    private int value;
    //private int num;

    public Node(int value){
        this.value = value;
    }
    public void add(int value){

        if(value > this.value)
        {
            if(right != null)
                right.add(value);
            else
            {
                Node node = new Node(value);
                right = node;
            }
        }
        else{
            if(left != null)
                left.add(value);
            else
            {
                Node node = new Node(value);
                left = node;
            }
        }
    }

    public boolean find(int value){
        if(value == this.value) return true;
        else if(value > this.value){
            if(right == null) return false;
```

```
            else return right.find(value);
        }else{
            if(left == null) return false;
            else return left.find(value);
        }
    }

    public void display(){
        System.out.println(value);
        if(left != null) left.display();
        if(right != null) right.display();
    }

    /*public Iterator iterator(){
    }*/

    public static void main(String[] args){
        int[] values = new int[8];
        for(int i=0;i<8;i++){
            int num = (int)(Math.random() * 15);
            //System.out.println(num);
            //if(Arrays.binarySearch(values, num)<0)
            if(!contains(values,num))
                values[i] = num;
            else
                i--;
        }

        System.out.println(Arrays.toString(values));

        Node root  = new Node(values[0]);
        for(int i=1;i<values.length;i++){
            root.add(values[i]);
        }

        System.out.println(root.find(13));
        root.display();
    }

    public static boolean contains(int [] arr, int value){
        int i = 0;
        for(;i<arr.length;i++){
            if(arr[i] == value) return true;
        }
```

```
        return false;
    }
}
```

6、从类似如下的文本文件中读取出所有的姓名，并打印出重复的姓名和重复的次数，并按重复次数排序：

1,雷军,28
2,马云,35
3,雷军,28
4,丁磊,35
5,雷军,28
6,马云,35
7,钟馗,28
8,荆轲,35

程序代码如下（答题要博得用人单位的喜欢，包名用该公司，面试前就提前查好该公司的网址，如果查不到，现场问也是可以的。还要加上实现思路的注释）：

```
package com.huawei.interview;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.Comparator;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.TreeSet;


public class GetNameTest {
    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        //InputStream ips =
GetNameTest.class.getResourceAsStream("/com/huawei/interview/info.txt
");
        //用上一行注释的代码和下一行的代码都可以，因为info.txt与GetNameTest类在同一包下面，所以，可以用下面的相对路径形式

        Map results = new HashMap();
```

```
        InputStream ips =
GetNameTest.class.getResourceAsStream("info.txt");
        BufferedReader in = new BufferedReader(new
InputStreamReader(ips));
        String line = null;
        try {
            while((line=in.readLine())!=null)
            {
                dealLine(line,results);
            }
            sortResults(results);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    static class User
    {
        public  String name;
        public Integer value;
        public User(String name,Integer value)
        {
            this.name = name;
            this.value = value;
        }

        @Override
        public boolean equals(Object obj) {
            // TODO Auto-generated method stub

            //下面的代码没有执行，说明往treeset中增加数据时，不会使用到equals方
法。
            boolean result = super.equals(obj);
            System.out.println(result);
            return result;
        }
    }

    private static void sortResults(Map results) {
        // TODO Auto-generated method stub
        TreeSet sortedResults = new TreeSet(
                new Comparator(){
                    public int compare(Object o1, Object o2) {
```

```
                    // TODO Auto-generated method stub
                    User user1 = (User)o1;
                    User user2 = (User)o2;
                    /*如果compareTo返回结果0，则认为两个对象相等，新的对象不会增加到集合中去
                     * 所以，不能直接用下面的代码，否则，那些个数相同的其他姓名就打印不出来。
                     * */

                    //return user1.value-user2.value;
                    //return user1.value<user2.value?-1:user1.value==user2.value?0:1;
                    if(user1.value<user2.value)
                    {
                        return -1;
                    }else if(user1.value>user2.value)
                    {
                        return 1;
                    }else
                    {
                        return user1.name.compareTo(user2.name);
                    }
                }

            }
        );
        Iterator iterator = results.keySet().iterator();
        while(iterator.hasNext())
        {
            String name = (String)iterator.next();
            Integer value = (Integer)results.get(name);
            if(value > 1)
            {
                sortedResults.add(new User(name,value));
            }
        }

        printResults(sortedResults);
    }
    private static void printResults(TreeSet sortedResults)
    {
        Iterator iterator  = sortedResults.iterator();
        while(iterator.hasNext())
        {
```

```
            User user = (User)iterator.next();
            System.out.println(user.name + ":" + user.value);
        }
    }
    public static void dealLine(String line,Map map)
    {
        if(!"".equals(line.trim()))
        {
            String [] results = line.split(",");
            if(results.length == 3)
            {
                String name = results[1];
                Integer value = (Integer)map.get(name);
                if(value == null) value = 0;
                map.put(name,value + 1);
            }
        }
    }
}
```

7、写一个 Singleton 出来。

第一种：饱汉模式

```
public class SingleTon {
    private SingleTon(){
        }

    //实例化放在静态代码块里可提高程序的执行效率，但也可能更占用空间
    private final static SingleTon instance = new SingleTon();
    public static SingleTon getInstance(){
        return instance;
    }
}
```

第二种：饥汉模式

```
public class SingleTon {
    private SingleTon(){}

    private static instance = null;//new SingleTon();

    public static synchronized SingleTon getInstance(){
        if(instance == null)
            instance = new SingleTon();
        return instance;
```

```
    }
}
```

第三种：用枚举

```
    public enum SingleTon{
        ONE;

    }
```

第三：更实际的应用（在什么情况用单例）

```
public class SequenceGenerator{
    //下面是该类自身的业务功能代码
    private int count = 0;

    public synchronized int getSequence(){
        ++count;
    }

    //下面是把该类变成单例的代码
    private SequenceGenerator(){}
    private final static instance = new SequenceGenerator();
    public static SingleTon getInstance(){
        return instance;
    }

}
```

第四：

```
    public class MemoryDao
    {
    private HashMap map = new HashMap();

    public void add(Student stu1){
        map.put(SequenceGenerator.getInstance().getSequence(),stu1);
    }

   //把MemoryDao变成单例
  }
```

Singleton 模式主要作用是保证在 Java 应用程序中，一个类 Class 只有一个实例存在。
一般 Singleton 模式通常有几种种形式:
第一种形式: 定义一个类，它的构造函数为 private 的，它有一个 static 的 private 的该类变量，在类初始化时实例话，通过一个 public 的 getInstance 方法获取对它的引用,继而调用其中的方法。

```
public class Singleton {
private Singleton(){}
        //在自己内部定义自己一个实例，是不是很奇怪?
        //注意这是 private  只供内部调用
        private static Singleton instance = new Singleton();
        //这里提供了一个供外部访问本 class 的静态方法，可以直接访问
        public static Singleton getInstance() {
            return instance;
        }
    }
    第二种形式:
public class Singleton {
    private static Singleton instance = null;
    public static synchronized Singleton getInstance() {
    //这个方法比上面有所改进，不用每次都进行生成对象，只是第一次
    //使用时生成实例，提高了效率!
    if (instance==null)
        instance＝new Singleton();
         return instance;
  }
}
```

其他形式:
定义一个类，它的构造函数为 private 的，所有方法为 static 的。
一般认为第一种形式要更加安全些

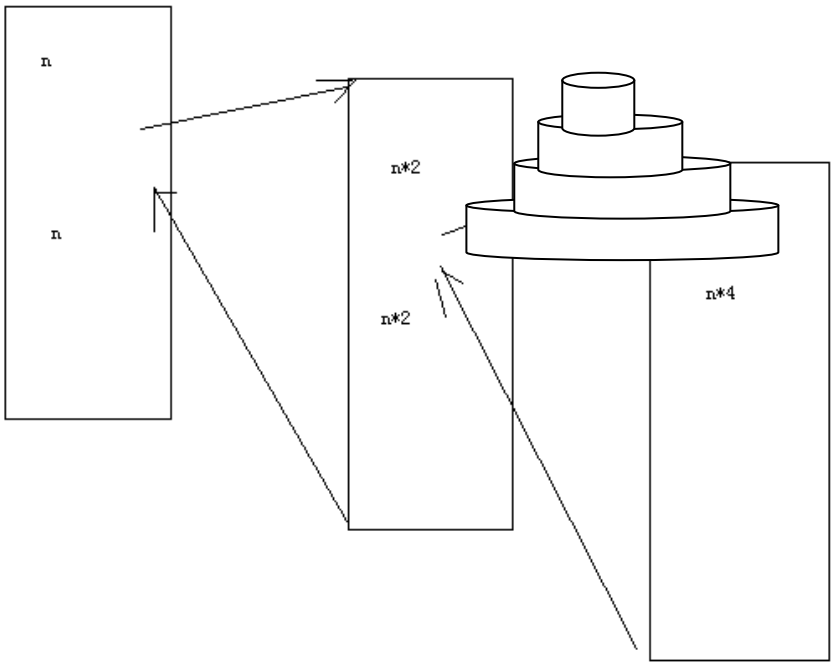
8、递归算法题 1

一个整数，大于 0，不用循环和本地变量，按照 n，2n，4n，8n 的顺序递增，当值大于 5000 时，把值按照指定顺序输出来。
例：n=1237
则输出为：
1237，
2474，
4948，
9896，
9896，
4948，
2474，
1237，

提示：写程序时，先致谢按递增方式的代码，写好递增的以后，再增加考虑递减部分。

```
public static void doubleNum(int n)
{
    System.out.println(n);
    if(n<=5000)
        doubleNum(n*2);
    System.out.println(n);
}
```

Gaibaota(N) = Gaibaota(N-1) + n



9、递归算法题 2

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?

```
package cn.aaaedu;

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);
    }
```

10、排序都有哪几种方法？请列举。用 JAVA 实现一个快速排序。

本人只研究过冒泡排序、选择排序和快速排序，下面是快速排序的代码：

```
public class QuickSort {
/**
* 快速排序
* @param strDate
* @param left
* @param right
*/
public void quickSort(String[] strDate,int left,int right){
String middle,tempDate;
int i,j;
i=left;
j=right;
middle=strDate[(i+j)/2];
do{
while(strDate[i].compareTo(middle)<0&& i<right)
i++; //找出左边比中间值大的数
while(strDate[j].compareTo(middle)>0&& j>left)
j--; //找出右边比中间值小的数
if(i<=j){ //将左边大的数和右边小的数进行替换
tempDate=strDate[i];
strDate[i]=strDate[j];
strDate[j]=tempDate;
i++;
j--;
}
```

```
}while(i<=j); //当两者交错时停止

if(i<right){
quickSort(strDate,i,right);//从
}
if(j>left){
quickSort(strDate,left,j);
}
}
/**
  * @param args
  */
public static void main(String[] args){
      String[] strVoid=new String[]{"11","66","22","0","55","22","0","32"};
      QuickSort sort=new QuickSort();
      sort.quickSort(strVoid,0,strVoid.length-1);
      for(int i=0;i<strVoid.length;i++){
            System.out.println(strVoid[i]+" ");
      }
 }
}
```

11、有数组 a[n]，用 java 代码将数组元素顺序颠倒

```
//用下面的也可以
//for(int i=0,int j=a.length-1;i<j;i++,j--) 是否等效于 for(int i=0;i<a.length/2;i++)
呢?

import java.util.Arrays;

public class SwapDemo{

    public static void main(String[] args){
        int [] a = new int[]{
                        (int)(Math.random() * 1000),
                        (int)(Math.random() * 1000),
                        (int)(Math.random() * 1000),
                        (int)(Math.random() * 1000),

                        (int)(Math.random() * 1000)

        };
```

```
        System.out.println(a);
        System.out.println(Arrays.toString(a));
        swap(a);
        System.out.println(Arrays.toString(a));
    }

    public static void swap(int a[]){
        int len = a.length;
        for(int i=0;i<len/2;i++){
            int tmp = a[i];
            a[i] = a[len-1-i];
            a[len-1-i] = tmp;
        }
    }
}
```

12．金额转换，阿拉伯数字的金额转换成中国传统的形式如：（￥1011）—>（一千零一拾一元整）输出。

去零的代码：

```
return sb.reverse().toString().replaceAll("零[拾佰仟]","零").replaceAll("零+万","万").replaceAll("零+元","元").replaceAll("零+","零");
```

```
public class RenMingBi {

    /**
     * @param args add by zxx ,Nov 29, 2008
     */
    private static final char[] data = new char[]{
            '零','壹','贰','叁','肆','伍','陆','柒','捌','玖'
        };
    private static final char[] units = new char[]{
        '元','拾','佰','仟','万','拾','佰','仟','亿'
    };
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.out.println(
                convert(135689123));
    }

    public static String convert(int money)
    {
```

```
        StringBuffer sbf = new StringBuffer();
        int unit = 0;
        while(money!=0)
        {
            sbf.insert(0,units[unit++]);
            int number = money%10;
            sbf.insert(0, data[number]);
            money /= 10;
        }

        return sbf.toString();
    }
}
```

三. Java web 部分

1、Tomcat 的优化经验

答: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

2、HTTP 请求的 GET 与 POST 方式的区别

答: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。

3、解释一下什么是 servlet;

答: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。

4、说一说 Servlet 的生命周期?

答: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。

5、Servlet 的基本架构

```
public class ServletName extends HttpServlet {
public void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
}
public void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
}
}
```

6、SERVLET API 中 forward() 与 redirect()的区别？

答: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。

7、什么情况下调用 doGet()和 doPost()？

Jsp 页面中的 FORM 标签里的 method 属性为 get 时调用 doGet()，为 post 时调用 doPost()。

8、Request 对象的主要方法：

setAttribute(String name,Object)：设置名字为 name 的 request 的参数值
getAttribute(String name)：返回由 name 指定的属性值
getAttributeNames()：返回 request 对象所有属性的名字集合，结果是一个枚举的实例
getCookies()：返回客户端的所有 Cookie 对象，结果是一个 Cookie 数组
getCharacterEncoding()：返回请求中的字符编码方式
getContentLength()：返回请求的 Body 的长度
getHeader(String name)：获得 HTTP 协议定义的文件头信息
getHeaders(String name)：返回指定名字的 request Header 的所有值，结果是一个枚举的实例
getHeaderNames()：返回所以 request Header 的名字，结果是一个枚举的实例
getInputStream()：返回请求的输入流，用于获得请求中的数据
getMethod()：获得客户端向服务器端传送数据的方法
getParameter(String name)：获得客户端传送给服务器端的有 name 指定的参数值
getParameterNames()：获得客户端传送给服务器端的所有参数的名字，结果是一个枚举的实例

getParametervalues(String name)：获得有 name 指定的参数的所有值
getProtocol()：获取客户端向服务器端传送数据所依据的协议名称
getQueryString()：获得查询字符串
getRequestURI()：获取发出请求字符串的客户端地址
getRemoteAddr()：获取客户端的 IP 地址
getRemoteHost()：获取客户端的名字
getSession([Boolean create])：返回和请求相关 Session
getServerName()：获取服务器的名字
getServletPath()：获取客户端所请求的脚本文件的路径
getServerPort()：获取服务器的端口号
removeAttribute(String name)：删除请求中的一个属性

9、forward 和 redirect 的区别

forward 是服务器请求资源，服务器直接访问目标地址的 URL，把那个 URL 的响应内容读取过来，然后把这些内容再发给浏览器，浏览器根本不知道服务器发送的内容是从哪儿来的，所以它的地址栏中还是原来的地址。

redirect 就是服务端根据逻辑,发送一个状态码,告诉浏览器重新去请求那个地址，一般来说浏览器会用刚才请求的所有参数重新请求，所以 session,request 参数都可以获取。

10. jsp 有哪些内置对象?作用分别是什么？分别有什么方法？

答:JSP 共有以下 9 个内置的对象：
request 用户端请求，此请求会包含来自 GET/POST 请求的参数
response 网页传回用户端的回应
pageContext 网页的属性是在这里管理
session 与请求有关的会话期
application servlet 正在执行的内容
out 用来传送回应的输出
config servlet 的构架部件
page JSP 网页本身
exception 针对错误网页，未捕捉的例外

request 表示 HttpServletRequest 对象。它包含了有关浏览器请求的信息，并且提供了几个用于获取 cookie, header, 和 session 数据的有用的方法。

response 表示 HttpServletResponse 对象，并提供了几个用于设置送回 浏览器的响应的方法（如 cookies,头信息等）

out 对象是 javax.jsp.JspWriter 的一个实例，并提供了几个方法使你能用于向浏览器回送输出结果。

pageContext 表示一个 javax.servlet.jsp.PageContext 对象。它是用于方便存取各种范围的名字空间、servlet 相关的对象的 API，并且包装了通用的 servlet 相关功能的方法。

session 表示一个请求的 javax.servlet.http.HttpSession 对象。Session 可以存贮用户的状态信息

applicaton 表示一个 javax.servle.ServletContext 对象。这有助于查找有关 servlet 引擎和 servlet 环境的信息

config 表示一个 javax.servlet.ServletConfig 对象。该对象用于存取 servlet 实例的初始化参数。
page 表示从该页面产生的一个 servlet 实例

11. jsp 有哪些动作?作用分别是什么?

（这个问题似乎不重要，不明白为何有此题）
答: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

12、两种跳转方式分别是什么?有什么区别?

（下面的回答严重错误，应该是想问 forward 和 sendRedirect 的区别，毕竟出题的人不是专业搞文字艺术的人，可能表达能力并不见得很强，用词不一定精准，加之其自身的技术面也可能存在一些问题，不一定真正将他的意思表达清楚了，严格意思上来讲，一些题目可能根本就无人能答，所以，答题时要掌握主动，只要把自己知道的表达清楚就够了，而不要去推敲原始题目的具体含义是什么，不要一味想着是在答题）
答：有两种，分别为：
<jsp:include page=included.jsp flush=true>
<jsp:forward page= nextpage.jsp/>
前者页面不会转向 include 所指的页面，只是显示该页的结果，主页面还是原来的页面。执行完后还会回来，相当于函数调用。并且可以带参数.后者完全转向新页面，不会再回来。相当于 go to 语句。

13、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。

14、MVC 的各个部分都有那些技术来实现?如何实现?

答: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。

15、我们在 web 应用开发过程中经常遇到输出某种编码的字符，如 iso8859-1 等，如何输出一个某种编码的字符串？

```
Public String translate (String str) {
    String tempStr = "";
    try {
        tempStr = new String(str.getBytes("ISO-8859-1"), "GBK");
        tempStr = tempStr.trim();
    }
    catch (Exception e) {
        System.err.println(e.getMessage());
    }
    return tempStr;
}
```

四. 数据库部分

1、用两种方式根据部门号从高到低，工资从低到高列出每个员工的信息。

employee:
eid,ename,salary,deptid;

```
select * from employee order by deptid desc,salary
```

2、列出各个部门中工资高于本部门的平均工资的员工数和部门号，并按部门号排序

创建表：

mysql> create table employee921(id int primary key auto_increment,name varchar(50),salary bigint,deptid int);

插入实验数据：

mysql> insert into employee921 values(null,'zs',1000,1),(null,'ls',1100,1),(null,'ww',1100,1),(null,'zl',900,1) ,(null,'zl',1000,2), (null,'zl',900,2) ,(null,'zl',1000,2) , (null,'zl',1100,2);

编写 sql 语句：

（）select avg(salary) from employee921 group by deptid;

（）mysql> select employee921.id,employee921.name,employee921.salary,employee921.deptid tid from　employee921 where salary > (select avg(salary) from employee921 where　deptid = tid);

效率低的一个语句，仅供学习参考使用（在 group by 之后不能使用 where，只能使用 having，在 group by 之前可以使用 where，即表示对过滤后的结果分组）：

```
mysql> select employee921.id,employee921.name,employee921.salary,employee921.dep
tid tid from  employee921 where salary > (select avg(salary) from employee921 group by deptid having deptid = tid);
（）select count(*) ,tid
    from (
        select employee921.id,employee921.name,employee921.salary,employee921.deptid tid
        from      employee921
        where salary >
            (select avg(salary) from employee921 where   deptid = tid)
    ) as t
    group by tid ;
```

另外一种方式：关联查询

```
select a.ename,a.salary,a.deptid
 from emp a,
    (select deptd,avg(salary) avgsal from emp group by deptid ) b
 where a.deptid=b.deptid and a.salary>b.avgsal;
```

3、存储过程与触发器必须讲，经常被面试到?

```
create procedure insert_Student (_name varchar(50),_age int ,out _id int)
begin
    insert into student value(null,_name,_age);
    select max(stuId) into _id from student;
end;

call insert_Student('wfz',23,@id);
select @id;

mysql> create trigger update_Student BEFORE update on student FOR EACH ROW
    -> select * from student;
```

触发器不允许返回结果

```
    create trigger update_Student BEFORE update on student FOR EACH ROW
    insert into   student value(null,'zxx',28);
```

mysql 的触发器目前不能对当前表进行操作

```
    create trigger update_Student BEFORE update on student FOR EACH ROW
    delete from articles   where id=8;
```

这个例子不是很好，最好是用删除一个用户时，顺带删除该用户的所有帖子

这里要注意使用 OLD.id

触发器用处还是很多的，比如今日头条、Csdn、抖音，你发一个日志，自动通知好友，其实就是在增加日志时做一个后触发，再向通知表中写入条目。因为触发器效率高。而 UCH 没有

用触发器，效率和数据处理能力都很低。

存储过程的实验步骤：

```
mysql> delimiter |
mysql> create procedure insertArticle_Procedure (pTitle varchar(50),pBid int,out
 pId int)
    -> begin
    -> insert into article1 value(null,pTitle,pBid);
    -> select max(id) into pId from article1;
    -> end;
    -> |
Query OK, 0 rows affected (0.05 sec)

mysql> call insertArticle_Procedure('zhangchen',1,@pid);
    -> |
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;
mysql> select @pid;
+------+
| @pid |
+------+
| 3    |
+------+
1 row in set (0.00 sec)

mysql> select * from article1;
+----+--------------+------+
| id | title        | bid  |
+----+--------------+------+
| 1  | test         | 1    |
| 2  | aaaedu       | 1    |
| 3  | zhangchen    | 1    |
+----+--------------+------+
3 rows in set (0.00 sec)
```

触发器的实验步骤：

```
create table board1(id int primary key auto_increment,name varchar(50),ar
ticleCount int);

create table article1(id int primary key auto_increment,title varchar(50)
,bid int references board1(id));

delimiter |
```

```
create trigger insertArticle_Trigger after insert on article1 for each ro
w begin
    -> update board1 set articleCount=articleCount+1 where id= NEW.bid;
    -> end;
    -> |

delimiter ;

insert into board1 value (null,'test',0);

insert into article1 value(null,'test',1);
```

还有，每插入一个帖子，都希望将版面表中的最后发帖时间，帖子总数字段进行同步更新，用触发器做效率就很高。下次课设计这样一个案例，写触发器时，对于最后发帖时间可能需要用 declare 方式声明一个变量，或者是用 NEW.posttime 来生成。

4、数据库三范式是什么？

第一范式（1NF）：字段具有原子性,不可再分。所有关系型数据库系统都满足第一范式）

数据库表中的字段都是单一属性的，不可再分。例如，姓名字段，其中的姓和名必须作为一个整体，无法区分哪部分是姓，哪部分是名，如果要区分出姓和名，必须设计成两个独立的字段。

第二范式（2NF）：

第二范式（2NF）是在第一范式（1NF）的基础上建立起来的，即满足第二范式（2NF）必须先满足第一范式（1NF）。

要求数据库表中的每个实例或行必须可以被惟一地区分。通常需要为表加上一个列，以存储各个实例的惟一标识。这个惟一属性列被称为主关键字或主键。

第二范式（2NF）要求实体的属性完全依赖于主关键字。所谓完全依赖是指不能存在仅依赖主关键字一部分的属性，如果存在，那么这个属性和主关键字的这一部分应该分离出来形成一个新的实体，新实体与原实体之间是一对多的关系。为实现区分通常需要为表加上一个列，以存储各个实例的惟一标识。简而言之，第二范式就是非主属性非部分依赖于主关键字。

第三范式的要求如下：

满足第三范式（3NF）必须先满足第二范式（2NF）。简而言之，第三范式（3NF）要求一个数据库表中不包含已在其它表中已包含的非主关键字信息。

所以第三范式具有如下特征：

1，每一列只有一个值

2，每一行都能区分。

3，每一个表都不包含其他表已经包含的非主关键字信息。

例如，帖子表中只能出现发帖人的 id，而不能出现发帖人的 id，还同时出现发帖人姓名，否则，只要出现同一发帖人 id 的所有记录，它们中的姓名部分都必须严格保持一致，这就是数据冗余。

5、说出一些数据库优化方面的经验?

用 PreparedStatement 一般来说比 Statement 性能高：一个 sql 发给服务器去执行，涉及步骤：语法检查、语义分析， 编译，缓存
“inert into user values(1,1,1)”-→二进制
“inert into user values(2,2,2)”-→二进制
“inert into user values(?,?,?)”-→二进制

有外键约束会影响插入和删除性能，如果程序能够保证数据的完整性，那在设计数据库时就去掉外键。（比喻：就好比免检产品，就是为了提高效率，充分相信产品的制造商）
（对于 hibernate 来说，就应该有一个变化：empleyee->Deptment 对象，现在设计时就成了 employee→deptid）

看 mysql 帮助文档子查询章节的最后部分，例如，根据扫描的原理，下面的子查询语句要比第二条关联查询的效率高：

```
1.  select e.name,e.salary where e.managerid=(select id from employee where name='zxx');

2.   select e.name,e.salary,m.name,m.salary from employees e,employees m where
  e.managerid = m.id and m.name='zxx';
```

表中允许适当冗余，譬如，主题帖的回复数量和最后回复时间等
将姓名和密码单独从用户表中独立出来。这可以是非常好的一对一的案例哟！

sql 语句全部大写，特别是列名和表名都大写。特别是 sql 命令的缓存功能，更加需要统一大小写，sql 语句→发给 oracle 服务器→语法检查和编译成为内部指令→缓存和执行指令。根据缓存的特点，不要拼凑条件，而是用?和 PreparedStatment

还有索引对查询性能的改进也是值得关注的。

备注：下面是关于性能的讨论举例

4 航班 3 个城市

m*n

```
select * from flight,city where flight.startcityid=city.cityid and city.name='beijing';
```

m + n

```
select * from flight where startcityid = (select cityid from city where cityname='beijing');

select flight.id,'beijing',flight.flightTime from flight where startcityid = (select cityid from city where cityname='beijing')
```

6、union 和 union all 有什么不同?

假设我们有一个表 Student，包括以下字段与数据：

```
drop table student;
create table student
(
id int primary key,
name nvarchar2(50) not null,
score number not null
);
insert into student values(1,'鲁班',78);
insert into student values(2,'Bill',76);
insert into student values(3,'貂蝉',89);
insert into student values(4,'后羿',90);
insert into student values(5,'妲己',73);
insert into student values(6,'东皇太一',61);
insert into student values(7,'诸葛亮',99);
insert into student values(8,'花木兰,56);
insert into student values(9,'程咬金',93);
insert into student values(10,'蔡文姬',90);
commit;
```

Union 和 Union All 的区别。

```
select *
from student
where id < 4
union
select *
from student
where id > 2 and id < 6
```

结果将是

```
1    鲁班    78
2    Bill    76
3    貂蝉    89
4    后羿    90
5    妲己    73
```

如果换成 Union All 连接两个结果集，则返回结果是：

```
1    鲁班    78
2    Bill    76
3    貂蝉    89
3    貂蝉    89
4    后羿    90
5    妲己    73
```

可以看到，Union 和 Union All 的区别之一在于对重复结果的处理。

UNION 在进行表链接后会筛选掉重复的记录，所以在表链接后会对所产生的结果集进行排序运算，删除重复的记录再返回结果。实际大部分应用中是不会产生重复的记录，最常见的是过程表与历史表 UNION。如：

```
select * from gc_dfys
union
select * from ls_jg_dfys
```

这个 SQL 在运行时先取出两个表的结果，再用排序空间进行排序删除重复的记录，最后返回结果集，如果表数据量大的话可能会导致用磁盘进行排序。

而 UNION ALL 只是简单的将两个结果合并后就返回。这样，如果返回的两个结果集中有重复的数据，那么返回的结果集就会包含重复的数据了。

从效率上说，UNION ALL 要比 UNION 快很多，所以，如果可以确认合并的两个结果集中不包含重复的数据的话，那么就使用 UNION ALL，

7.分页语句

取出 sql 表中第 31 到 40 的记录（以自动增长 ID 为主键）

sql server 方案 1：

```
select top 10 * from t where id not in (select top 30 id from t order by id ) orde by id
```

sql server 方案 2：

```
select top 10 * from t where id in (select top 40 id from t order by id) order by id desc
```

mysql 方案：`select * from t order by id limit 30,10`

oracle 方案：`select * from (select rownum r,* from t where r<=40) where r>30`

----------待整理进去的内容----------

```
pageSize=20;
pageNo = 5;
```

1.分页技术 1（直接利用 sql 语句进行分页，效率最高和最推荐的）

```
mysql:sql = "select * from articles limit " + (pageNo-1)*pageSize + "," + pageSize;
oracle: sql = "select * from " +
                                        "(select rownum r,* from " +
                                            "(select * from articles order by postime desc)" +
                                        "where rownum<= " + pageNo*pageSize +") tmp " +
                                    "where r>" + (pageNo-1)*pageSize;
```

注释：第 7 行保证 rownum 的顺序是确定的，因为 oracle 的索引会造成 rownum 返回不同的值

简洋提示：没有 order by 时，rownum 按顺序输出，一旦有了 order by，rownum 不按顺序输出了，这说明 rownum 是排序前的编号。如果对 order by 从句中的字段建立了索引，那么，rownum 也是按顺序输出的，因为这时候生成原始的查询结果集时会参照索引表的顺序来构建。

```
sqlserver:sql = "select top 10 * from id not id(select top " + (pageNo-1)*pageSize + "id from articles)"

DataSource ds = new InitialContext().lookup(jndiurl);
Connection cn = ds.getConnection();
//"select * from user where id=?"    --->binary directive
PreparedStatement pstmt = cn.prepareSatement(sql);
ResultSet rs = pstmt.executeQuery()
while(rs.next())
{
      out.println(rs.getString(1));
}
```

2.不可滚动的游标

```
pageSize=20;
pageNo = 5;
cn = null
stmt = null;
rs = null;
try
{
sqlserver:sql = "select    * from articles";

DataSource ds = new InitialContext().lookup(jndiurl);
Connection cn = ds.getConnection();
//"select * from user where id=?"    --->binary directive
PreparedStatement pstmt = cn.prepareSatement(sql);
ResultSet rs = pstmt.executeQuery()
for(int j=0;j<(pageNo-1)*pageSize;j++)
{
      rs.next();
}

int i=0;

while(rs.next() && i<10)
{
      i++;
      out.println(rs.getString(1));
}
}
cacth(){}
finnaly
{
      if(rs!=null)try{rs.close();}catch(Exception e){}
```

```
    if(stm.....
    if(cn.....
}
```

3.可滚动的游标

```
pageSize=20;
pageNo = 5;
cn = null
stmt = null;
rs = null;
try
{
sqlserver:sql = "select    * from articles";

DataSource ds = new InitialContext().lookup(jndiurl);
Connection cn = ds.getConnection();
//"select * from user where id=?"    --->binary directive
PreparedStatement pstmt = cn.prepareSatement(sql,ResultSet.TYPE_SCROLL_INSENSITIVE,...);
//根据上面这行代码的异常 SQLFeatureNotSupportedException，就可判断驱动是否支持可滚动游标

ResultSet rs = pstmt.executeQuery()
rs.absolute((pageNo-1)*pageSize)
int i=0;
while(rs.next() && i<10)
{
    i++;
    out.println(rs.getString(1));
}
}
cacth(){}
finnaly
{
    if(rs!=null)try{rs.close();}catch(Exception e){}
    if(stm）.....
    if(cn.）.....
}
```

8.用一条 SQL 语句 查询出每门课都大于 80 分的学生姓名

name	kecheng	fenshu
雷军	Hadoop	81
雷军	Spark	75
马云	Hadoop	76

马云　Spark　90
丁磊　Hadoop　81
丁磊　Spark　100
丁磊　Linux　90

准备数据的 sql 代码：
create table score(id int primary key auto_increment,name varchar(20),subject varchar(20),score int);
insert into score values
(null,'雷军','Hadoop',81),
(null,'雷军','Spark',75),
(null,'马云','Hadoop',76),
(null,'马云','Spark',90),
(null,'丁磊','Hadoop',81),
(null,'丁磊','Spark',100),
(null,'丁磊 ','Linux',90);

提示：当百思不得其解时，请理想思维，把小变成大做，把大变成小做，

答案：
A: select distinct name from score where name not in (select distinct name from score where score<=80)

B:select distince name t1 from score where 80< all (select score from score where name=t1);

9.所有部门之间的比赛组合

一个叫 department 的表，里面只有一个字段 name, 一共有 4 条纪录，分别是 a, b, c, d, 对应四个球对，现在四个球对进行比赛，用一条 sql 语句显示所有可能的比赛组合.

答：select a.name, b.name
from team a, team b
where a.name < b.name

10.每个月份的发生额都比 101 科目多的科目

请用 SQL 语句实现：从 TestDB 数据表中查询出所有月份的发生额都比 101 科目相应月份的发生额高的科目。请注意：TestDB 中有很多科目，都有 1－12 月份的发生额。
AccID：科目代码，Occmonth：发生额月份，DebitOccur：发生额。
数据库名：JcyAudit，数据集：Select * from TestDB
准备数据的 sql 代码：
drop table if exists TestDB;
create table TestDB(id int primary key auto_increment,AccID varchar(20), Occmonth date, DebitOccur bigint);

```
insert into TestDB values
(null,'101','2018-1-1',100),
(null,'101','2018-2-1',110),
(null,'101','2018-3-1',120),
(null,'101','2018-4-1',100),
(null,'101','2018-5-1',100),
(null,'101','2018-6-1',100),
(null,'101','2018-7-1',100),
(null,'101','2018-8-1',100);
--复制上面的数据，故意把第一个月份的发生额数字改小一点
insert into TestDB values
(null,'102','2018-1-1',90),
(null,'102','2018-2-1',110),
(null,'102','2018-3-1',120),
(null,'102','2018-4-1',100),
(null,'102','2018-5-1',100),
(null,'102','2018-6-1',100),
(null,'102','2018-7-1',100),
(null,'102','2018-8-1',100);
--复制最上面的数据，故意把所有发生额数字改大一点
insert into TestDB values
(null,'103','2018-1-1',150),
(null,'103','2018-2-1',160),
(null,'103','2018-3-1',180),
(null,'103','2018-4-1',120),
(null,'103','2018-5-1',120),
(null,'103','2018-6-1',120),
(null,'103','2018-7-1',120),
(null,'103','2018-8-1',120);
--复制最上面的数据，故意把所有发生额数字改大一点
insert into TestDB values
(null,'104','2018-1-1',130),
(null,'104','2018-2-1',130),
(null,'104','2018-3-1',140),
(null,'104','2018-4-1',150),
(null,'104','2018-5-1',160),
(null,'104','2018-6-1',170),
(null,'104','2018-7-1',180),
(null,'104','2018-8-1',140);
--复制最上面的数据，故意把第二个月份的发生额数字改小一点
insert into TestDB values
(null,'105','2018-1-1',100),
(null,'105','2018-2-1',80),
(null,'105','2018-3-1',120),
```

```
(null,'105','2018-4-1',100),
(null,'105','2018-5-1',100),
(null,'105','2018-6-1',100),
(null,'105','2018-7-1',100),
(null,'105','2018-8-1',100);
```

答案：

```
select distinct AccID from TestDB
where AccID not in
    (select TestDB.AccIDfrom TestDB,
        (select * from TestDB where AccID='101') as db101
    where TestDB.Occmonth=db101.Occmonth and TestDB.DebitOccur<=db101.DebitOccur
    );
```

11.统计每年每月的信息

```
year    month amount
2011     1         1.1
2011     2         1.2
2011     3         1.3
2011     4         1.4
2012     1         2.1
2012     2         2.2
2012     3         2.3
2012     4         2.4
```

查成这样一个结果

```
year m1    m2    m3    m4
2011.1 1.2 1.3 1.4
2012 2.1 2.2 2.3 2.4
```

提示：这个与工资条非常类似，与学生的科目成绩也很相似。

准备 sql 语句：

```
drop table if exists sales;
create table sales(id int auto_increment primary key,year varchar(10), month
varchar(10), amount float(2,1));
insert into sales values
(null,'2011','1',1.1),
(null,'2011','2',1.2),
(null,'2011','3',1.3),
(null,'2011','4',1.4),
(null,'2012','1',2.1),
(null,'2012','2',2.2),
(null,'2012','3',2.3),
(null,'2012','4',2.4);
```

答案一、

```
select sales.year ,
(select t.amount from sales t where t.month='1' and t.year= sales.year) '1',
(select t.amount from sales t where t.month='1' and t.year= sales.year) '2',
(select t.amount from sales t where t.month='1' and t.year= sales.year) '3',
(select t.amount from sales t where t.month='1' and t.year= sales.year) as '4'
from sales  group by year;
```

12.显示文章标题，发帖人、最后回复时间

表：id,title,postuser,postdate,parentid
准备 sql 语句：

```
drop table if exists articles;
create table articles(id int auto_increment primary key,title varchar(50), postuser varchar(10), postdate datetime,parentid int references articles(id));
insert into articles values
(null,'第一条','雷军','2018-10-10 12:32:32',null),
(null,'第二条','雷军','2018-10-10 12:34:32',null),
(null,'第一条回复 1','马云','2018-10-10 12:35:32',1),
(null,'第二条回复 1','马云','2018-10-10 12:36:32',2),
(null,'第一条回复 2','丁磊','2018-10-10 12:37:32',1),
(null,'第一条回复 3','马云','2018-10-10 12:38:32',1),
(null,'第二条回复 2','马云','2018-10-10 12:39:32',2),
(null,'第一条回复 4','丁磊','2018-10-10 12:39:40',1);
```

答案：

```
select a.title,a.postuser,
    (select max(postdate) from articles where parentid=a.id) reply
from articles a where a.parentid is null;
```

注释：子查询可以用在选择列中，也可用于 where 的比较条件中，还可以用于 from 从句中。

13.删除除了 id 号不同,其他都相同的学生冗余信息

2.学生表 如下:

id 号	学号	姓名	课程编号	课程名称	分数
1	2005001	雷军	0001	Spark	69
2	2005002	马云	0001	Spark	89
3	2005001	雷军	0001	Spark	69

A: delete from tablename where id 号 not in(select min(id 号) from tablename group by 学号,姓名,课程编号,课程名称,分数)
实验：

```
create table student2(id int auto_increment primary key,code varchar(20),name varchar(20));
insert into student2 values(null,'2005001','雷军'),(null,'2005002','马云'),(null,'2005001','雷军');

//如下语句，mysql 报告错误，可能删除依赖后面统计语句，而删除又导致统计语句结果不一致。

delete from student2 where id not in(select min(id) from student2 group by name);
//但是，如下语句没有问题：
select *  from student2 where id not in(select min(id) from student2 group by name);
//于是，我想先把分组的结果做成虚表，然后从虚表中选出结果，最后再将结果作为删除的条件数据。
delete from student2 where id not in(select mid from (select min(id) mid
from student2 group by name) as t);
或者：
delete from student2 where id not in(select min(id) from (select * from s
tudent2) as t group by t.name);
```

14.航空网的几个航班查询题：

表结构如下：

flight{flightID,StartCityID ,endCityID,StartTime}

city{cityID, CityName)

实验环境：

```
create table city(cityID int auto_increment primary key,cityName varchar(20));
create table flight (flightID int auto_increment primary key,
    StartCityID int references city(cityID),
    endCityID   int references city(cityID),
    StartTime timestamp);
//航班本来应该没有日期部分才好，但是下面的题目当中涉及到了日期
insert into city values(null,'北京'),(null,'上海'),(null,'广州');
insert into flight values
    (null,1,2,'9:37:23'),(null,1,3,'9:37:23'),(null,1,2,'10:37:23'),(null,2,3,'10:37:23');
```

1、查询起飞城市是北京的所有航班，按到达城市的名字排序

参与运算的列是我起码能够显示出来的那些列，但最终我不一定把它们显示出来。各个表组合出来的中间结果字段中必须包含所有运算的字段。

```
select   * from flight f,city c
    where f.endcityid = c.cityid and startcityid =
    (select c1.cityid from city c1 where c1.cityname = "北京")
    order by c.cityname asc;
```

```
mysql> select flight.flightid,'北京' startcity, e.cityname from flight,city e wh
ere flight.endcityid=e.cityid and flight.startcityid=(select cityid from city wh
ere cityname='北京');

mysql> select flight.flightid,s.cityname,e.cityname from flight,city s,city e wh
ere flight.startcityid=s.cityid and s.cityname='北京' and flight.endCityId=e.cit
yID order by e.cityName desc;
```

2、查询北京到上海的所有航班纪录（起飞城市，到达城市，起飞时间，航班号）

```
select c1.CityName,c2.CityName,f.StartTime,f.flightID
from city c1,city c2,flight f
where f.StartCityID=c1.cityID
and f.endCityID=c2.cityID
and c1.cityName='北京'
and c2.cityName='上海'
```

3、查询具体某一天（2005-5-8）的北京到上海的的航班次数

```
select count(*) from
(select c1.CityName,c2.CityName,f.StartTime,f.flightID
from city c1,city c2,flight f
where f.StartCityID=c1.cityID
and f.endCityID=c2.cityID
and c1.cityName='北京'
and c2.cityName='上海'
and  查帮助获得的某个日期处理函数(startTime) like '2005-5-8%'
```

mysql 中提取日期部分进行比较的示例代码如下：

```
select * from flight where date_format(starttime,'%Y-%m-%d')='2018-01-02'
```

15.查出比经理薪水还高的员工信息：

```
Drop table if not exists employees;
create table employees(id int primary key auto_increment,name varchar(50)
      ,salary int,managerid int references employees(id));
insert into employees values (null,' lhm',10000,null), (null,' zxx',15000,1
      ),(null,'flx',9000,1),(null,'tg',10000,2),(null,'wzg',10000,3);
```

Wzg 大于 flx,lhm 大于 zxx

解题思路：

根据 sql 语句的查询特点，是逐行进行运算，不可能两行同时参与运算。涉及了员工薪水和经理薪水，所有，一行记录要同时包含两个薪水，所有想到要把这个表自关联组合一下。

首先要组合出一个包含有各个员工及该员工的经理信息的长记录，譬如，左半部分是员工，

右半部分是经理。而迪卡尔积会组合出很多垃圾信息，先去除这些垃圾信息。

```
select e.* from employees e,employees m where e.managerid=m.id and e.sala
ry>m.salary;
```

16、求出小于 45 岁的各个老师所带的大于 12 岁的学生人数

数据库中有 3 个表 teacher 表，student 表，tea_stu 关系表。

teacher 表 teaID name age

student 表 stuID name age

teacher_student 表 teaID stuID

要求用一条 sql 查询出这样的结果

1.显示的字段要有老师 name, age 每个老师所带的学生人数

2 只列出老师 age 为 40 以下，学生 age 为 12 以上的记录

预备知识：

1.sql 语句是对每一条记录依次处理，条件为真则执行动作（select,insert,delete,update）

2.只要是迪卡尔积，就会产生“垃圾”信息，所以，只要迪卡尔积了，我们首先就要想到清除“垃圾”信息

实验准备：

```
drop table if exists tea_stu;
drop table if exists teacher;
drop table if exists student;
  create table teacher(teaID int primary key,name varchar(50),age int);
  create table student(stuID int primary key,name varchar(50),age int);
  create table tea_stu(teaID int references teacher(teaID),stuID int references
student(stuID));
insert into teacher values(1,'zxx',45), (2,'lhm',25) , (3,'wzg',26) , (4,'tg',27);
insert into student values(1,'wy',11), (2,'dh',25) , (3,'ysq',26) , (4,'mxc',27);
insert into tea_stu values(1,1), (1,2), (1,3);
insert into tea_stu values(2,2), (2,3), (2,4);
 insert into tea_stu values(3,3), (3,4), (3,1);
insert into tea_stu values(4,4), (4,1), (4,2) , (4,3);
```

结果：2→3,3→2,4→3

解题思路：（真实面试答题时，也要写出每个分析步骤，如果纸张不够，就找别人要）

1 要会统计分组信息，统计信息放在中间表中：

```
select teaid,count(*) from tea_stu group by teaid;
```

2 接着其实应该是筛除掉小于 12 岁的学生，然后再进行统计，中间表必须与 student 关联才能得到 12 岁以下学生和把该学生记录从中间表中剔除，代码是：

```
select tea_stu.teaid,count(*) total from student,tea_stu
where student.stuid=tea_stu.stuid and student.age>12 group by tea_stu.teaid
```

3.接着把上面的结果做成虚表与 teacher 进行关联，并筛除大于 45 的老师

select teacher.teaid,teacher.name,total from teacher ,***(select tea_stu.tea***id,count(*) total from student,tea_stu where student.stuid=tea_stu.stuid and student.age>12 group by tea_stu.teaid) as tea_stu2 where teacher.teaid=tea_stu2.teaid and teacher.age<45;

17.求出发帖最多的人：

```
select authorid,count(*) total from articles
group by authorid
having total=
    (select max(total2) from (select count(*) total2 from articles group by authorid) as t);
```

```
select t.authorid,max(t.total) from
    （select authorid,count(*) total from articles ）as t
```

这条语句不行，因为 max 只有一列，不能与其他列混淆。

```
select authorid,count(*) total from articles
group by authorid having total=max(total)
```

也不行。

18、一个用户表中有一个积分字段，假如数据库中有 100 多万个用户，若要在每年第一天凌晨将积分清零，你将考虑什么，你将想什么办法解决?

```
alter table drop column score;
alter table add colunm score int;
```

可能会很快，但是需要试验，试验不能拿真实的环境来操刀，并且要注意，
这样的操作时无法回滚的，在我的印象中，只有 inert update delete 等 DML 语句才能回滚，
对于 create table,drop table ,alter table 等 DDL 语句是不能回滚。

解决方案一，update user set score=0;
解决方案二，假设上面的代码要执行好长时间，超出我们的容忍范围，那我就 alter table user drop column score;alter table user add column score int。

下面代码实现每年的那个凌晨时刻进行清零。

```
Runnable runnable =
    new Runnable(){
        public void run(){
            clearDb();
            schedule(this,new Date(new Date().getYear()+1,0,0));
            }
        };
```

```
schedule(runnable,
      new Date(new Date().getYear()+1,0,1));
```

19、一个用户具有多个角色，请查询出该表中具有该用户的所有角色的其他用户。

```
select count(*) as num,tb.id
from
  tb,
  (select role from tb where id=xxx) as t1
where
  tb.role = t1.role and tb.id != t1.id
group by tb.id
having
      num = select count(role) from tb where id=xxx;
```

20. xxx 公司的 sql 面试

Table **EMPLOYEES** Structure:

```
   EMPLOYEE_ID        NUMBER          Primary Key,
   FIRST_NAME         VARCHAR2(25),
   LAST_NAME          VARCHAR2(25),
   Salary number(8,2),
   HiredDate DATE,
   Departmentid number(2)
```

Table **Departments** Structure:

```
   Departmentid number(2)          Primary Key,
   DepartmentName    VARCHAR2(25).
```

（2）基于上述 EMPLOYEES 表写出查询：写出雇用日期在今年的，或者工资在[1000,2000]之间的，或者员工姓名（last_name）以’Obama’打头的所有员工，列出这些员工的全部个人信息。（4 分）

```
select * from employees
where Year(hiredDate) = Year(date())
      or (salary between 1000 and 200)
      or left(last_name,3)='abc';
```

(3) 基于上述 EMPLOYEES 表写出查询：查出部门平均工资大于 1800 元的部门的所有员工，列出这些员工的全部个人信息。（4 分）

```
mysql> select id,name,salary,deptid did from employee1 where (select avg(salary)
  from employee1 where deptid = did) > 1800;
```

(4) 基于上述 EMPLOYEES 表写出查询：查出个人工资高于其所在部门平均工资的员工，列出这些员工的全部个人信息及该员工工资高出部门平均工资百分比。（5 分）

```
select employee1.*,(employee1.salary-t.avgSalary)*100/employee1.salary
from employee1,
      (select deptid,avg(salary) avgSalary from employee1 group by deptid) as t
where employee1.deptid = t.deptid and employee1.salary>t.avgSalary;
```

21、注册 Jdbc 驱动程序的三种方式

22、用 JDBC 如何调用存储过程

代码如下：

```
package com.huawei.interview.lym;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Types;

public class JdbcTest {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Connection cn = null;
        CallableStatement cstmt = null;
        try {
            //这里最好不要这么干，因为驱动名写死在程序中了
            Class.forName("com.mysql.jdbc.Driver");
            //实际项目中，这里应用DataSource数据，如果用框架，
            //这个数据源不需要我们编码创建，我们只需Datasource ds =
context.lookup()
            //cn = ds.getConnection();
            cn =
DriverManager.getConnection("jdbc:mysql:///test","root","root");
            cstmt = cn.prepareCall("{call insert_Student(?,?,?)}");
            cstmt.registerOutParameter(3,Types.INTEGER);
            cstmt.setString(1, "wangwu");
            cstmt.setInt(2, 25);
            cstmt.execute();
            //get第几个，不同的数据库不一样，建议不写
            System.out.println(cstmt.getString(3));
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
```

```
        }
        finally
        {

            /*try{cstmt.close();}catch(Exception e){}
            try{cn.close();}catch(Exception e){}*/
            try {
                if(cstmt != null)
                    cstmt.close();
                if(cn != null)
                    cn.close();
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
```

23、JDBC 中的 PreparedStatement 相比 Statement 的好处

答：一个 sql 命令发给服务器去执行的步骤为：语法检查，语义分析，编译成内部指令，缓存指令，执行指令等过程。
select * from student where id =3----缓存--→xxxxx 二进制命令
select * from student where id =3----直接取-→xxxxx 二进制命令
select * from student where id =4--- -→会怎么干？
如果当初是 select * from student where id =?--- -→又会怎么干？
上面说的是性能提高
可以防止 sql 注入。

24. 写一个用 jdbc 连接并访问 oracle 数据的程序代码

25、Class.forName 的作用?为什么要用?

答：按参数中指定的字符串形式的类名去搜索并加载相应的类，如果该类字节码已经被加载过，则返回代表该字节码的 Class 实例对象，否则，按类加载器的委托机制去搜索和加载该类，如果所有的类加载器都无法加载到该类，则抛出 ClassNotFoundException。加载完这个 Class 字节码后，接着就可以使用 Class 字节码的 newInstance 方法去创建该类的实例对象了。
有时候，我们程序中所有使用的具体类名在设计时（即开发时）无法确定，只有程序运行时才能确定，这时候就需要使用 Class.forName 去动态加载该类，这个类名通常是在配置文件中配置的，例如，spring 的 ioc 中每次依赖注入的具体类就是这样配置的，jdbc 的驱动类名通常也是通过配置文件来配置的，以便在产品交付使用后不用修改源程序就可以更换驱动类名。

26、大数据量下的分页解决方法。

答：最好的办法是利用 sql 语句进行分页，这样每次查询出的结果集中就只包含某页的数据内容。再 sql 语句无法实现分页的情况下，可以考虑对大的结果集通过游标定位方式来获取某页的数据。

sql 语句分页，不同的数据库下的分页方案各不一样，下面是主流的三种数据库的分页 sql：

sql server:

```
String sql =
"select top " + pageSize + " * from students where id not in" +
 "(select top " + pageSize * (pageNumber-1) + " id from students order by id)" + "order by id";
```

mysql:

```
String sql =
"select * from students order by id limit " + pageSize*(pageNumber-1) + "," + pageSize;
```

oracle:

```
String sql =
 "select * from " +
 (select *,rownum rid from (select * from students order by postime desc) where rid<=" + pagesize*pagenumber + ") as t" +
 "where t>" + pageSize*(pageNumber-1);
```

27、用 JDBC 查询学生成绩单，把主要代码写出来（考试概率极大）.

```
Connection cn = null;
PreparedStatement pstmt =null;
Resultset rs = null;
try
{
    Class.forname(driveClassName);
    cn =   DriverManager.getConnection(url,username,password);
    pstmt = cn.prepareStatement("select   score.* from score ,student " +
        "where score.stuId = student.id and student.name = ?");
    pstmt.setString(1,studentName);
    Resultset rs = pstmt.executeQuery();
    while(rs.next())
    {
        system.out.println(rs.getInt("subject")   +   "       " + rs.getFloat("score") );
    }
}catch(Exception e){e.printStackTrace();}
finally
{
    if(rs != null) try{ rs.close() }catch(exception e){}
    if(pstmt != null) try{pstmt.close()}catch(exception e){}
    if(cn != null) try{ cn.close() }catch(exception e){}
```

```
}
```

28、这段代码有什么不足之处?

```
try {
Connection conn = ...;
Statement stmt = ...;

ResultSet rs = stmt.executeQuery("select * from table1");

while(rs.next()) {

}
} catch(Exception ex) {
}
```

答：没有 finally 语句来关闭各个对象，另外，使用 finally 之后，要把变量的定义放在 try 语句块的外面，以便在 try 语句块之外的 finally 块中仍可以访问这些变量。

29、说出数据连接池的工作机制是什么?

J2EE 服务器启动时会建立一定数量的池连接，并一直维持不少于此数目的池连接。客户端程序需要连接时，池驱动程序会返回一个未使用的池连接并将其表记为忙。如果当前没有空闲连接，池驱动程序就新建一定数量的连接，新建连接的数量有配置参数决定。当使用的池连接调用完成后，池驱动程序将此连接表记为空闲，其他调用就可以使用这个连接。实现方式，返回的 Connection 是原始 Connection 的代理，代理 Connection 的 close 方法不是真正关连接，而是把它代理的 Connection 对象还回到连接池中。

30、为什么要用 ORM? 和 JDBC 有何不一样?

orm 是一种思想，就是把 object 转变成数据库中的记录，或者把数据库中的记录转变成 objecdt，我们可以用 jdbc 来实现这种思想，其实，如果我们的项目是严格按照 oop 方式编写的话，我们的 jdbc 程序不管是有意还是无意，就已经在实现 orm 的工作了。

现在有许多 orm 工具，它们底层调用 jdbc 来实现了 orm 工作，我们直接使用这些工具，就省去了直接使用 jdbc 的繁琐细节,提高了开发效率,现在用的较多的 orm 工具是 hibernate。也听说一些其他 orm 工具，如 toplink,ojb 等。

五. XML 部分

1、xml 有哪些解析技术?区别是什么?

答:有 DOM,SAX,STAX 等

DOM:处理大型文件时其性能下降的非常厉害。这个问题是由 DOM 的树结构所造成的，这种结构占用的内存较多，而且 DOM 必须在解析文件之前把整个文档装入内存,适合对 XML 的随机访问 SAX:不现于 DOM,SAX 是事件驱动型的 XML 解析方式。它顺序读取 XML 文件，不需要一次全部装载整个文件。当遇到像文件开头，文档结束，或者标签开头与标签结束时，它会触发一个事件，用户通过在其回调事件中写入处理代码来处理 XML 文件，适合对 XML 的顺序访问 STAX:Streaming API for XML (StAX)

2、你在项目中用到了 xml 技术的哪些方面?如何实现的?

答:用到了数据存贮，信息配置两方面。在做数据交换平台时，将不能数据源的数据组装成 XML 文件，然后将 XML 文件压缩打包加密后通过网络传送给接收者，接收解密与解压缩后再同 XML 文件中还原相关信息进行处理。在做软件配置时，利用 XML 可以很方便的进行，软件的各种配置参数都存贮在 XML 文件中。

3、用 jdom 解析 xml 文件时如何解决中文问题?如何解析?

答:看如下代码,用编码方式加以解决

```
package test;
import java.io.*;
public class DOMTest
{
private String inFile = "c:\\people.xml"
private String outFile = "c:\\people.xml"
public static void main(String args[])
{
new DOMTest();
}
public DOMTest()
{
try
{
javax.xml.parsers.DocumentBuilder builder =
javax.xml.parsers.DocumentBuilderFactory.newInstance().newDocumentBuilder();
org.w3c.dom.Document doc = builder.newDocument();
org.w3c.dom.Element root = doc.createElement("老师");
org.w3c.dom.Element wang = doc.createElement("王");
org.w3c.dom.Element liu = doc.createElement("刘");
wang.appendChild(doc.createTextNode("我是王老师"));
root.appendChild(wang);
doc.appendChild(root);
javax.xml.transform.Transformer transformer =
javax.xml.transform.TransformerFactory.newInstance().newTransformer();
transformer.setOutputProperty(javax.xml.transform.OutputKeys.ENCODING, "gb2312");
transformer.setOutputProperty(javax.xml.transform.OutputKeys.INDENT, "yes");
```

```
transformer.transform(new javax.xml.transform.dom.DOMSource(doc),
new
      javax.xml.transform.stream.StreamResult(outFile));
}
catch (Exception e)
{
      System.out.println (e.getMessage());
}
}
}
```

4、编程用 JAVA 解析 XML 的方式.

答:用 SAX 方式解析 XML，XML 文件如下：

```
<?xml version=1.0 encoding=gb2312?>
<person>
<name>庄周</name>
<college>信息学院</college>
<telephone>6258113</telephone>
<notes>男,1955 年生,博士，95 年调入郑州大学</notes>
</person>
```

事件回调类 SAXHandler.java

```
import java.io.*;
import java.util.Hashtable;
import org.xml.sax.*;
public class SAXHandler extends HandlerBase
{
private Hashtable table = new Hashtable();
private String currentElement = null;
private String currentValue = null;
public void setTable(Hashtable table)
{
this.table = table;
}
public Hashtable getTable()
{
return table;
}
public void startElement(String tag, AttributeList attrs)
throws SAXException
{
currentElement = tag;
}
public void characters(char[] ch, int start, int length)
```

```
throws SAXException
{
    currentValue = new String(ch, start, length);
}
public void endElement(String name) throws SAXException
{
    if (currentElement.equals(name))
    table.put(currentElement, currentValue);
}
}
```

JSP 内容显示源码,SaxXml.jsp:

```
<HTML>
<HEAD>
<TITLE>剖析 XML 文件 people.xml</TITLE>
</HEAD>
<BODY>
<%@ page errorPage=ErrPage.jsp
contentType=text/html;charset=GB2312 %>
<%@ page import=java.io.* %>
<%@ page import=java.util.Hashtable %>
<%@ page import=org.w3c.dom.* %>
<%@ page import=org.xml.sax.* %>
<%@ page import=javax.xml.parsers.SAXParserFactory %>
<%@ page import=javax.xml.parsers.SAXParser %>
<%@ page import=SAXHandler %>
<%
File file = new File(c:\people.xml);
FileReader reader = new FileReader(file);
Parser parser;
SAXParserFactory spf = SAXParserFactory.newInstance();
SAXParser sp = spf.newSAXParser();
SAXHandler handler = new SAXHandler();
sp.parse(new InputSource(reader), handler);
Hashtable hashTable = handler.getTable();
out.println(<TABLE BORDER=2><CAPTION>教师信息表</CAPTION>);
out.println(<TR><TD>姓名</TD> + <TD> +
(String)hashTable.get(new String(name)) + </TD></TR>);
out.println(<TR><TD>学院</TD> + <TD> +
(String)hashTable.get(new String(college))+</TD></TR>);
out.println(<TR><TD>电话</TD> + <TD> +
(String)hashTable.get(new String(telephone)) + </TD></TR>);
out.println(<TR><TD>备注</TD> + <TD> +
(String)hashTable.get(new String(notes)) + </TD></TR>);
out.println(</TABLE>);
```

%>
</BODY>
</HTML>

5、XML 文档定义有几种形式？它们之间有何本质区别？解析 XML 文档有哪几种方式？

a: 两种形式 dtd schema，b: 本质区别:schema 本身是 xml 的，可以被 XML 解析器解析(这也是从 DTD 上发展 schema 的根本目的)，c:有 DOM,SAX,STAX 等

DOM:处理大型文件时其性能下降的非常厉害。这个问题是由 DOM 的树结构所造成的，这种结构占用的内存较多，而且 DOM 必须在解析文件之前把整个文档装入内存,适合对 XML 的随机访问。

SAX:不现于 DOM,SAX 是事件驱动型的 XML 解析方式。它顺序读取 XML 文件，不需要一次全部装载整个文件。当遇到像文件开头，文档结束，或者标签开头与标签结束时，它会触发一个事件，用户通过在其回调事件中写入处理代码来处理 XML 文件，适合对 XML 的顺序访问 STAX:Streaming API for XML (STAX) 。

六. 设计模式

1、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。

2、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。

3、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。

七. J2EE 部分

1、BS 与 CS 的联系与区别。

C/S 是 Client/Server 的缩写。服务器通常采用高性能的 PC、工作站或小型机，并采用大型数据库系统，如 Oracle、Sybase、InFORMix 或 SQL Server。客户端需要安装专用的客户端软件。

B/S 是 Brower/Server 的缩写，客户机上只要安装一个浏览器（Browser），如 Netscape Navigator 或 Internet Explorer，服务器安装 Oracle、Sybase、InFORMix 或 SQL Server 等数据库。在这种结构下，用户界面完全通过 WWW 浏览器实现，一部分事务逻辑在前端实现，但是主要事务逻辑在服务器端实现。浏览器通过Web Server 同数据库进行数据交互。

C/S 与 B/S 区别：

1．硬件环境不同:

C/S 一般建立在专用的网络上，小范围里的网络环境，局域网之间再通过专门服务器提供连接和数据交换服务.

B/S 建立在广域网之上的，不必是专门的网络硬件环境,例与电话上网，租用设备．信息自己管理．有比 C/S 更强的适应范围，一般只要有操作系统和浏览器就行

2．对安全要求不同

C/S 一般面向相对固定的用户群，对信息安全的控制能力很强．一般高度机密的信息系统采用 C/S 结构适宜．可以通过 B/S 发布部分可公开信息.

B/S 建立在广域网之上，对安全的控制能力相对弱，可能面向不可知的用户。

3．对程序架构不同

C/S 程序可以更加注重流程，可以对权限多层次校验，对系统运行速度可以较少考虑.

B/S 对安全以及访问速度的多重的考虑，建立在需要更加优化的基础之上．比 C/S 有更高的要求 B/S 结构的程序架构是发展的趋势，从 MS 的.Net 系列的 BizTalk 2000 Exchange 2000 等，全面支持网络的构件搭建的系统. SUN 和 IBM 推的 JavaBean 构件技术等,使 B/S 更加成熟.

4．软件重用不同

C/S 程序可以不可避免的整体性考虑，构件的重用性不如在 B/S 要求下的构件的重用性好.

B/S 对的多重结构,要求构件相对独立的功能. 能够相对较好的重用.就入买来的餐桌可以再利用,而不是做在墙上的石头桌子

5．系统维护不同

C/S 程序由于整体性, 必须整体考察, 处理出现的问题以及系统升级. 升级难. 可能是再做一个全新的系统

B/S 构件组成,方面构件个别的更换,实现系统的无缝升级. 系统维护开销减到最小.用户从网上自己下载安装就可以实现升级.

6．处理问题不同

C/S 程序可以处理用户面固定, 并且在相同区域, 安全要求高需求, 与操作系统相关. 应该都是相同的系统

B/S 建立在广域网上, 面向不同的用户群, 分散地域, 这是 C/S 无法作到的. 与操作系统平台关系最小.

7．用户接口不同

C/S 多是建立的 Window 平台上,表现方法有限,对程序员普遍要求较高

B/S 建立在浏览器上, 有更加丰富和生动的表现方式与用户交流. 并且大部分难度减低, 减低开发成本.

8．信息流不同

C/S 程序一般是典型的中央集权的机械式处理, 交互性相对低

B/S 信息流向可变化, B-B B-C B-G 等信息、流向的变化, 更像交易中心。

2、应用服务器与 WEB SERVER 的区别？

应用服务器：Weblogic、Tomcat、Jboss　WEB SERVER：IIS、 Apache

3、应用服务器有那些？

BEA WebLogic Server，IBM WebSphere Application Server，Oracle9i Application Server，jBoss，Tomcat

4、J2EE 是什么？

答:Je22 是 Sun 公司提出的多层(multi-diered),分布式(distributed),基于组件(component-base)的企业级应用模型(enterpriese application model).在这样的一个应用系统中，可按照功能划分为不同的组件，这些组件又可在不同计算机上，并且处于相应的层次(tier)中。所属层次包括客户层(clietn tier)组件,web 层和组件,Business 层和组件,企业信息系统(EIS)层。一个另类的回答：j2ee 就是增删改查。

5、J2EE 是技术还是平台还是框架？ 什么是 J2EE

J2EE 本身是一个标准，一个为企业分布式应用的开发提供的标准平台。
J2EE 也是一个框架，包括 JDBC、JNDI、RMI、JMS、EJB、JTA 等技术。

6、请对以下在 J2EE 中常用的名词进行解释(或简单描述)

web 容器：给处于其中的应用程序组件（JSP，SERVLET）提供一个环境，使 JSP,SERVLET 直接更容器中的环境变量接口交互，不必关注其它系统问题。主要有 WEB 服务器来实现。例如：TOMCAT,WEBLOGIC,WEBSPHERE 等。该容器提供的接口严格遵守 J2EE 规范中的 WEB APPLICATION 标准。我们把遵守以上标准的 WEB 服务器就叫做 J2EE 中的 WEB 容器。

EJB 容器：Enterprise java bean 容器。更具有行业领域特色。他提供给运行在其中的组件 EJB 各种管理功能。只要满足 J2EE 规范的 EJB 放入该容器，马上就会被容器进行高效率的管理。并且可以通过现成的接口来获得系统级别的服务。例如邮件服务、事务管理。

JNDI：（Java Naming & Directory Interface）JAVA 命名目录服务。主要提供的功能是：提供一个目录系统，让其它各地的应用程序在其上面留下自己的索引，从而满足快速查找和定位分布式应用程序的功能。

JMS：（Java Message Service）JAVA 消息服务。主要实现各个应用程序之间的通讯。包括点对点和广播。

JTA：（Java Transaction API）JAVA 事务服务。提供各种分布式事务服务。应用程序只需调用其提供的接口即可。

JAF：（Java Action FrameWork）JAVA 安全认证框架。提供一些安全控制方面的框架。让开发者通过各种部署和自定义实现自己的个性安全控制策略。

RMI/IIOP:（Remote Method Invocation /internet 对象请求中介协议）他们主要用于通过远程调用服务。例如，远程有一台计算机上运行一个程序，它提供股票分析服务，我们可以在本地计算机上实现对其直接调用。当然这是要通过一定的规范才能在异构的系统之间进行通信。RMI 是 JAVA 特有的。

八、Mybatis

1. 谈谈 MyBatis

Mybatis 是一个半自动化的 ORM 框架，它对 jdbc 的操作数据库的过程进行封装，使得开发者只需要专注于 SQL 语句本身，而不用去关心注册驱动，创建 connection 等，Mybatis 通过 xml 文件配置或者注解的方式将要执行的各种 statement 配置起来，并通过 java 对象和 statement 中的 sql 进行映射成最终执行的 sql 语句，最后由 Mybatis 框架执行 sql 并将结果映射成 java 对象并返回。每个 MyBatis 应用程序主要都是使用 SqlSessionFactory 实例的，一个 SqlSessionFactory 实例可以通过 SqlSessionFactoryBuilder 获得。SqlSessionFactoryBuilder 可以从一个 xml 配置文件或者一个预定义的配置类的实例获得。

Mybatis 分为三层

（1）API 接口层：提供给外部使用的接口 API

（2）数据处理层：负责具体的 SQL

（3）基础支撑层：负责最基础的功能支撑，如连接管理，事务管理，配置加载和缓存处。理

2. Mybatis 的优点

基于 SQL 语句编程，相当灵活，不会对应用程序或者数据库的现有设计造成任何影响，SQL 写在 XML 里，解除 sql 与程序代码的耦合，便于统一管理；提供 XML 标签，支持编写动态 SQL 语句，并可重用。

与 JDBC 相比，减少了 50%以上的代码量，消除了 JDBC 大量冗余的代码，不需要手动开关连接；

很好的与各种数据库兼容（因为 MyBatis 使用 JDBC 来连接数据库，所以只要 JDBC 支持的数据库 MyBatis 都支持）。

能够与 Spring 很好的集成；

提供映射标签，支持对象与数据库的 ORM 字段关系映射；提供对象关系映射标签，支持对象关系组件维护。

3. Mybatis 的缺点

1. Sql 语句的编写工作量较大，尤其当字段多、关联表多时，对开发人员编写 Sql 语句的功底有一定要求。

2. 对性能的要求很高，或者需求变化较多的项目，如互联网项目，MyBatis 将是不错的选择。

4. 什么是 ORM

对象关系映射（Object Relational Mapping，简称 ORM）是通过使用描述对象和数据库之间映射的元数据，将面向对象语言程序中的对象自动持久化到关系数据库中。常见的 ORM 框架有：Hibernate、TopLink、Castor JDO、Apache OJB、MyBatis 等。

5. 为什么说 Mybatis 是半自动 ORM 映射工具？它与全自动的区别在哪里?

Hibernate 属于全自动 ORM 映射工具，使用 Hibernate 查询关联对象或者关联集合对象时，可以根据对象关系模型直接获取，所以它是全自动的。而 Mybatis 在查询关联对象或关联集合对象时，需要手动编写 sql 来完成，所以，称之为半自动 ORM 映射工具。

6. JDBC 编程有哪些不足之处，MyBatis 是如何解决这些问题的?

1. 数据库链接创建、释放频繁造成系统资源浪费从而影响系统性能，如果使用数据库链接池可解决此问题。解决：在 SqlMapConfig.xml 中配置数据链接池，使用连接池管理数据库链接。
2. Sql 语句写在代码中造成代码不易维护，实际应用 sql 变化的可能较大，sql 变动需要改变

java 代码。解决：将 Sql 语句配置在 XXXXmapper.xml 文件中与 java 代码分离

3. 向 sql 语句传参数麻烦，因为 sql 语句的 where 条件不一定，可能多也可能少，占位符需要和参数一一对应。解决： Mybatis 自动将 java 对象映射至 sql 语句。
4. 对结果集解析麻烦，sql 变化导致解析代码变化，且解析前需要遍历，如果能将数据库记录封装成 pojo 对象解析比较方便。解决：Mybatis 自动将 sql 执行结果映射至 java 对象。

7. Mybatis 的编程步骤是什么样的?

5. 创建 SqlSessionFactory
6. 通过 SqlSessionFactory 创建 SqlSession
7. 通过 sqlsession 执行数据库操作
8. 调用 session.commit()提交事务
9. 调用 session.close()关闭会话

8. Mybatis 中#和$的区别?

1.${}是 Properties 文件中的变量占位符，它可以用于标签属性值和 sql 内部，属于静态文本替换，如：order by #user_id#，如果传入的值是 111,那么解析成 sql 时的值为 order by "111", 如果传入的值是 id，则解析成的 sql 为 order by "id".

2.#{}是 sql 的参数占位符，Mybatis 会将 sql 中的#{}替换为?号，在 sql 执行前会使用 PreparedStatement 的参数设置方法，按序给 sql 的?号占位符设置参数值。比如 ps.setInt(0, parameterValue)，#{item.name}的取值方式为使用反射从参数对象中获取 item 对象的 name 属性值，相当于 param.getItem().getName()。

#方式能够很大程度防止 sql 注入。

$方式无法防止 Sql 注入。

$方式一般用于传入数据库对象，例如传入表名。

一般能用#的就别用$。

9. 使用 MyBatis 的 mapper 接口调用时有哪些要求?

Mapper 接口方法名和 mapper.xml 中定义的每个 sql 的 id 相同

Mapper 接口方法的输入参数类型和 mapper.xml 中定义的每个 sql 的 parameterType 的类型相同

Mapper 接口方法的输出参数类型和 mapper.xml 中定义的每个 sql 的 resultType 的类型相同

4. Mapper.xml 文件中的 namespace 即是 mapper 接口的类路径。

10. Mybatis 中一级缓存与二级缓存?

1. 一级缓存: 基于 PerpetualCache 的 HashMap 本地缓存，其存储作用域为 Session，当 Session flush 或 close 之后，该 Session 中的所有 Cache 就将清空。

2. 二级缓存与一级缓存其机制相同，默认也是采用 PerpetualCache，HashMap 存储，不同

在于其存储作用域为 Mapper(Namespace)，并且可自定义存储源，如 Ehcache。作用域为 namespance 是指对该 namespance 对应的配置文件中所有的 select 操作结果都缓存，这样不同线程之间就可以共用二级缓存。

启动二级缓存：在 mapper 配置文件中：<cache />。二级缓存可以设置返回的缓存对象策略：<cache readOnly="true">。当 readOnly="true"时，表示二级缓存返回给所有调用者同一个缓存对象实例，调用者可以 update 获取的缓存实例，但是这样可能会造成其他调用者出现数据不一致的情况（因为所有调用者调用的是同一个实例）。当 readOnly="false"时，返回给调用者的是二级缓存总缓存对象的拷贝，即不同调用者获取的是缓存对象不同的实例，这样调用者对各自的缓存对象的修改不会影响到其他的调用者，即是安全的，所以默认是 readOnly="false";3. 对于缓存数据更新机制，当某一个作用域(一级缓存 Session/二级缓存 Namespaces)的进行了 C/U/D 操作后，默认该作用域下所有 select 中的缓存将被 clear。

11. MyBatis 在 insert 插入操作时返回主键 ID

数据库为 MySql 时：

<insert id="insert" parameterType="com.test.User" keyProperty="userId" useGeneratedKeys="true" >

“keyProperty”表示返回的 id 要保存到对象的那个属性中，“useGeneratedKeys”表示主键 id 为自增长模式。MySQL 中做以上配置就 OK 了数据库为 Oracle 时：

2. <insert id="insert" parameterType="com.test.User">

<selectKey resultType="INTEGER" order="BEFORE" keyProperty="userId">

SELECT SEQ_USER.NEXTVAL as userId from DUAL

</selectKey>

insert into user (user_id, user_name, modified, state) values(#{userId}, #{userName}, #{modified}, #{state})

</insert>

由于 Oracle 没有自增长一说法，只有序列这种模仿自增的形式，所以不能再使用“useGeneratedKeys”属性。而是使用<selectKey>将 ID 获取并赋值到对象的属性中，insert 插入操作时正常插入 id。

12. Xml 映射文件中，除了常见的 select|insert|updae|delete 标签之外，还有哪些标签?

还有很多其他的标签，<resultMap>、<parameterMap>、<sql>、<include>、<selectKey>，加上动态 sql 的 9 个标签，trim|where|set|foreach|if|choose|when|otherwise|bind 等，其中<sql>为 sql 片段标签，通过<include>标签引入 sql 片段，<selectKey>为不支持自增的主键生成策略标签。

13. 最佳实践中，通常一个 Xml 映射文件，都会写一个 Dao 接口与之对应，请问，这个 Dao 接口的工作原理是什么？Dao 接口里的方法，参数不同时，方法能重载吗?

Dao 接口，就是人们常说的 Mapper 接口，接口的全限名，就是映射文件中的 namespace 的值，接口的方法名，就是映射文件中 MappedStatement 的 id 值，接口方法内的参数，就是传递

给 sql 的参数。Mapper 接口是没有实现类的，当调用接口方法时，接口全限名+方法名拼接字符串作为 key 值，可唯一定位一个 MappedStatement，举例：com.mybatis3.mappers.StudentDao.findStudentById，可以唯一找到 namespace 为 com.mybatis3.mappers.StudentDao 下面 id = findStudentById 的 MappedStatement。在 Mybatis 中，每一个<select>、<insert>、<update>、<delete>标签，都会被解析为一个 MappedStatement 对象。

Dao 接口里的方法，是不能重载的，因为是全限名+方法名的保存和寻找策略。

Dao 接口的工作原理是 JDK 动态代理，Mybatis 运行时会使用 JDK 动态代理为 Dao 接口生成代理 proxy 对象，代理对象 proxy 会拦截接口方法，转而执行 MappedStatement 所代表的 sql，然后将 sql 执行结果返回。

14. 简述 Mybatis 的 Xml 映射文件和 Mybatis 内部数据结构之间的映射关系?

Mybatis 将所有 Xml 配置信息都封装到 All-In-One 重量级对象 Configuration 内部。在 Xml 映射文件中，<parameterMap>标签会被解析为 ParameterMap 对象，其每个子元素会被解析为 ParameterMapping 对象。<resultMap>标签会被解析为 ResultMap 对象，其每个子元素会被解析为 ResultMapping 对象。每一个<select>、<insert>、<update>、<delete>标签均会被解析为 MappedStatement 对象，标签内的 sql 会被解析为 BoundSql 对象。

15. Mybatis 的 Xml 映射文件中，不同的 Xml 映射文件，id 是否可以重复?

不同的 Xml 映射文件，如果配置了 namespace，那么 id 可以重复；如果没有配置 namespace，那么 id 不能重复；毕竟 namespace 不是必须的，只是最佳实践而已。

原因就是 namespace+id 是作为 Map<String, MappedStatement>的 key 使用的，如果没有 namespace，就剩下 id，那么，id 重复会导致数据互相覆盖。有了 namespace，自然 id 就可以重复，namespace 不同，namespace+id 自然也就不同。

16. Mybatis 是如何进行分页的？分页插件的原理是什么?

Mybatis 使用 RowBounds 对象进行分页，它是针对 ResultSet 结果集执行的内存分页，而非物理分页，可以在 sql 内直接书写带有物理分页的参数来完成物理分页功能，也可以使用分页插件来完成物理分页。

分页插件的基本原理是使用 Mybatis 提供的插件接口，实现自定义插件，在插件的拦截方法内拦截待执行的 sql，然后重写 sql，根据 dialect 方言，添加对应的物理分页语句和物理分页参数。

举例：select * from student，拦截 sql 后重写为：select t.* from （select * from student）t limit 0，10

17. 简述 Mybatis 的插件运行原理，以及如何编写一个插件。

Mybatis 仅可以编写针对 ParameterHandler、ResultSetHandler、StatementHandler、Executor 这 4 种接口的插件，Mybatis 使用 JDK 的动态代理，为需要拦截的接口生成代理对象以实现接口方法拦截功能，每当执行这 4 种接口对象的方法时，就会进入拦截方法，具体就是 InvocationHandler 的 invoke()方法，当然，只会拦截那些你指定需要拦截的方法。

实现 Mybatis 的 Interceptor 接口并复写 intercept()方法，然后在给插件编写注解，指定要拦截哪一个接口的哪些方法即可，记住，别忘了在配置文件中配置你编写的插件。

18. Mybatis 是如何将 sql 执行结果封装为目标对象并返回的？都有哪些映射形式？

第一种是使用<resultMap>标签，逐一定义列名和对象属性名之间的映射关系。第二种是使用 sql 列的别名功能，将列别名书写为对象属性名，比如 T_NAME AS NAME，对象属性名一般是 name，小写，但是列名不区分大小写，Mybatis 会忽略列名大小写，智能找到与之对应对象属性名，你甚至可以写成 T_NAME AS NaMe，Mybatis 一样可以正常工作。

有了列名与属性名的映射关系后，Mybatis 通过反射创建对象，同时使用反射给对象的属性逐一赋值并返回，那些找不到映射关系的属性，是无法完成赋值的。

19. Mybatis 动态 sql 是做什么的？都有哪些动态 sql？能简述一下动态 sql 的执行原理不？

Mybatis 动态 sql 可以让我们在 Xml 映射文件内，以标签的形式编写动态 sql，完成逻辑判断和动态拼接 sql 的功能，Mybatis 提供了 9 种动态 sql 标签 trim|where|set|foreach|if|choose|when|otherwise|bind。

其执行原理为，使用 OGNL 从 sql 参数对象中计算表达式的值，根据表达式的值动态拼接 sql，以此来完成动态 sql 的功能。

20. Mybatis 能执行一对一、一对多的关联查询吗？都有哪些实现方式，以及它们之间的区别。

能，Mybatis 不仅可以执行一对一、一对多的关联查询，还可以执行多对一，多对多的关联查询，多对一查询，其实就是一对一查询，只需要把 selectOne()修改为 selectList()即可；多对多查询，其实就是一对多查询，只需要把 selectOne()修改为 selectList()即可。

关联对象查询，有两种实现方式，一种是单独发送一个 sql 去查询关联对象，赋给主对象，然后返回主对象。另一种是使用嵌套查询，嵌套查询的含义为使用 join 查询，一部分列是 A 对象的属性值，另外一部分列是关联对象 B 的属性值，好处是只发一个 sql 查询，就可以把主对象和其关联对象查出来。

21. Mybatis 是否支持延迟加载？如果支持，它的实现原理是什么？

Mybatis 仅支持 association 关联对象和 collection 关联集合对象的延迟加载，association 指的就是一对一，collection 指的就是一对多查询。在 Mybatis 配置文件中，可以配置是否启用延迟加载 lazyLoadingEnabled=true|false。

它的原理是，使用 CGLIB 创建目标对象的代理对象，当调用目标方法时，进入拦截器方法，比如调用 a.getB().getName()，拦截器 invoke()方法发现 a.getB()是 null 值，那么就会单独发送事先保存好的查询关联 B 对象的 sql，把 B 查询上来，然后调用 a.setB(b)，于是 a 的对象 b 属性就有值了，接着完成 a.getB().getName()方法的调用。这就是延迟加载的基本原理。

当然了，不光是 Mybatis，几乎所有的包括 Hibernate，支持延迟加载的原理都是一样的。

22. Mybatis 中如何执行批处理？Mybatis 都有哪些 Executor 执行器？它们之间的区别是什么?

Mybatis 使用 Executor 完成批处理。

Mybatis 有三种基本的 Executor 执行器，SimpleExecutor、ReuseExecutor、BatchExecutor。

SimpleExecutor：每执行一次 update 或 select，就开启一个 Statement 对象，用完立刻关闭 Statement 对象。

ReuseExecutor：执行 update 或 select，以 sql 作为 key 查找 Statement 对象，存在就使用，不存在就创建，用完后，不关闭 Statement 对象，而是放置于 Map<String, Statement>内，供下一次使用。简言之，就是重复使用 Statement 对象。

BatchExecutor：执行 update（没有 select，JDBC 批处理不支持 select），将所有 sql 都添加到批处理中（addBatch()），等待统一执行（executeBatch()），它缓存了多个 Statement 对象，每个 Statement 对象都是 addBatch()完毕后，等待逐一执行 executeBatch()批处理。与 JDBC 批处理相同。

作用范围：Executor 的这些特点，都严格限制在 SqlSession 生命周期范围内。

23. Mybatis 中如何指定使用哪一种 Executor 执行器?

在 Mybatis 配置文件中，可以指定默认的 ExecutorType 执行器类型，也可以手动给 DefaultSqlSessionFactory 的创建 SqlSession 的方法传递 ExecutorType 类型参数。

24. Mybatis 是否可以映射 Enum 枚举类?

Mybatis 可以映射枚举类，不单可以映射枚举类，Mybatis 可以映射任何对象到表的一列上。映射方式为自定义一个 TypeHandler，实现 TypeHandler 的 setParameter()和 getResult()接口方法。TypeHandler 有两个作用，一是完成从 javaType 至 jdbcType 的转换，二是完成 jdbcType 至 javaType 的转换，体现为 setParameter()和 getResult()两个方法，分别代表设置 sql 问号占位符参数和获取列查询结果。

25. Mybatis 映射文件中，如果 A 标签通过 include 引用了 B 标签的内容，请问，B 标签能否定义在 A 标签的后面，还是说必须定义在 A 标签的前面?

虽然 Mybatis 解析 Xml 映射文件是按照顺序解析的，但是，被引用的 B 标签依然可以定义在任何地方，Mybatis 都可以正确识别。原理是，Mybatis 解析 A 标签，发现 A 标签引用了 B 标签，但是 B 标签尚未解析到，尚不存在，此时，Mybatis 会将 A 标签标记为未解析状态，然后继续解析余下的标签，包含 B 标签，待所有标签解析完毕，Mybatis 会重新解析那些被标记为未解析的标签，此时再解析 A 标签时，B 标签已经存在，A 标签也就可以正常解析完成了。

26. Mybatis 框架适用场合

MyBatis 专注于 SQL 本身，是一个足够灵活的 DAO 层解决方案。对性能的要求很高，或者需求变化较多的项目，如互联网项目，MyBatis 将是不错的选择。

九、Hibernate

1. 讲下什么是 ORM?ORM 组件有哪些？

orm 是对象关系映射，是一种程序技术，通过将 java 对象映射到数据库表，通过操作 java 对象，就可以完成对数据表的操作。常用的 orm 组件有 JDBC、Hibernate、mybatis、springDate 等

2. 谈谈你对 Hibernate 的理解。

1.面向对象设计的软件内部运行过程可以理解成就是在不断创建各种新对象、建立对象之间的关系，调用对象的方法来
改变各个对象的状态和对象消亡的过程，不管程序运行的过程和操作怎么样，本质上都是要得到一个结果，程序上一个
时刻和下一个时刻的运行结果的差异就表现在内存中的对象状态发生了变化。
2.为了在关机和内存空间不够的状况下，保持程序的运行状态，需要将内存中的对象状态保存到持久化设备和从持久化
设备中恢复出对象的状态，通常都是保存到关系数据库来保存大量对象信息。从 Java 程序的运行功能上来讲，保存对
象状态的功能相比系统运行的其他功能来说，应该是一个很不起眼的附属功能，java 采用 jdbc 来实现这个功能，这个
不起眼的功能却要编写大量的代码，而做的事情仅仅是保存对象和恢复对象，并且那些大量的 jdbc 代码并没有什么技术含量，基本上是采用一套例行公事的标准代码模板来编写，是一种苦活和重复性的工作。
3.通过数据库保存 java 程序运行时产生的对象和恢复对象，其实就是实现了 java 对象与关系数据库记录的映射关系，称为 ORM（即 Object Relation Mapping），人们可以通过封装 JDBC 代码来实现了这种功能，封装出来的产品称之为 ORM 框架，Hibernate 就是其中的一种流行 ORM 框架。使用 Hibernate 框架，不用写 JDBC 代码，仅仅是调用一个 save 方法，就可以将对象保存到关系数据库中，仅仅是调用一个 get 方法，就可以从数据库中加载出一个对象。

3. 关于 Hibernate 的 orm 思想你了解多少？

ORM 指的是对象关系型映射(Object RelationShip Mapping)，指的就是我们通过创建实体类对象和数据库中的表关系进行一一对应，来实现通过操作实体类对象来更改数据库里边的数据信息。这里边起到关键作用的是通过 Hibernate 的映射文件+Hibernate 的核心配置文件。

4. 简述一下 hibernate 的开发流程

第一步：加载 hibernate 的配置文件，读取配置文件的参数(jdbc 连接参数，数据 库方言，hbm 表与对象关系映射文件)
第二步：创建 SessionFactory 会话工厂(内部有连接池)
第三步：打开 session 获取连接，构造 session 对象(一次会话维持一个数据连接， 也是一级缓存)
第四步：开启事务
第五步：进行操作
第六步：提交事务
第七步：关闭 session(会话)将连接释放第八步：关闭连接池

5. Hibernate 和 JDBC 优缺点对比

相同点：
1）两者都是 java 数据库操作的中间件
2）两者对数据库进行直接操作的对象都是线程不安全的，都需要及时关闭。
3）两者都可对数据库的更新操作进行显式的事务处理。
不同点：
1）JDBC 是 SUN 公司提供一套操作数据库的规范，使用 java 代码操作数据库。Hibernate 是一个基于 jdbc 的主流持久化框架，对 JDBC 访问数据库的代码做了封装。
2）使用的 SQL 语言不同：JDBC 使用的是基于关系型数据库的标准 SQL 语言，Hibernate 使用的是 HQL(Hibernate query language)语言。
3）操作的对象不同：JDBC 操作的是数据，将数据通过 SQL 语句直接传送到数据库中执行，Hibernate 操作的是持久化对象，由底层持久化对象的数据更新到数据库中。
4）数据状态不同：JDBC 操作的数据是“瞬时”的，变量的值无法与数据库中的值保持一致，而 Hibernate 操作的数据是可持久的，即持久化对象的数据属性的值是可以跟数据库中的值保持一致的。

5.Hibernate 和 Mybatis 的区别？

两者相同点：
1）Hibernate 与 MyBatis 都可以是通过 SessionFactoryBuider 由 XML 配置文件生成 SessionFactory，然后由 SessionFactory 生成 Session，最后由 Session 来开启执行事务和 SQL 语句。其中 SessionFactoryBuider，SessionFactory，Session 的生命周期都是差不多的。
2）Hibernate 和 MyBatis 都支持 JDBC 和 JTA 事务处理。
Mybatis 优势：
1）MyBatis 可以进行更为细致的 SQL 优化，可以减少查询字段。
2）MyBatis 容易掌握，而 Hibernate 门槛较高。
Hibernate 优势：
1）Hibernate 的 DAO 层开发比 MyBatis 简单，Mybatis 需要维护 SQL 和结果映射。
2）Hibernate 对对象的维护和缓存要比 MyBatis 好，对增删改查的对象的维护要方便。
3）Hibernate 数据库移植性很好，MyBatis 的数据库移植性不好，不同的数据库需要写不同

SQL。
4）Hibernate 有更好的二级缓存机制，可以使用第三方缓存。MyBatis 本身提供的缓存机制不佳。

6. Hibernate 的查询方式有哪些？

Hibernate 的查询方式常见的主要分为三种： HQL, QBC（命名查询），以及使用原生 SQL 查询（SqlQuery）

7. Hibernate 中有几种检索方式，优缺点？

1、立即检索：立即查询，在执行查询语句时，立即查询所有的数据。get
优点：对应用程序完全透明，不管对象处于持久化状态，还是游离状态，应用程序都可以方便的从一个对象导航到与它关联的对象；
缺点：1.select 语句太多；2.可能会加载应用程序不需要访问的对象白白浪费许多内存空间；
2、延迟检索：延迟查询，在执行查询语句之后，在需要时再查询。
优点：由应用程序决定需要加载哪些对象，可以避免可执行多余的 select 语句，以及避免加载应用程序不需要访问的对象。因此能提高检索性能，并且能节省内存空间；
缺点：应用程序如果希望访问游离状态代理类实例，必须保证他在持久化状态时已经被初始化；
3、迫切左外连接检索
优点：1、对应用程序完全透明，不管对象处于持久化状态，还是游离状态，应用程序都可以方便地冲一个对象导航到与它关联的对象。2、使用了外连接，select 语句数目少；
缺点：1、可能会加载应用程序不需要访问的对象，白白浪费许多内存空间；2、复杂的数据库表连接也会影响检。

8. 说说 HQL 和 QBC，项目中都是怎么用的？

1）HQL 与 qbc 都是面向对象的查询语句，qbc 相对于 hql 更加面向对象，在 qbc 中把查询语句都封装成了方法。
2）qbc：通过调用不同的方法来实现对对象的操作，从而对数据进行操作
3）hql：可以手动编写 sql 语句来进行查询。

9. 说说 hibernate 的三种状态之间如何转换？

hibernate 的三种状态是瞬时状态、持久状态、托管状态
瞬时态(临时态、自由态)：不存在持久化标识 OID，尚未与 Hibernate Session 关联对象， 被认为处于瞬时态，失去引用将被 JVM。
回收持久态：存在持久化标识 OID，与当前 session 有关联，并且相关联的 session 没有关闭 ，并且事务未提交。
脱管态(离线态、游离态)：存在持久化标识 OID，但没有与当前 session 关联，脱管状态 改变 hibernate 不能检测到。
比如有一个 User 实体类和一张 User 表。当 new 了一个 user 对象，但没有开启事务。此时 user 就处于瞬时状态，与数据库的数据没有任何联系，当开启事务后，执行了 session.save()方法后，

session 缓存中存放了该 user 对象，而数据库中也有相应的这条数据，此时就转换为持久状态。当事务提交后，session 被销毁。session 缓存中就没有 user 对象，而数据库表中有相应记录，此时为托管状态。

10. hibernate 的缓存机制。

Hibernate 缓存分为两层：Hibernate 的一级缓存和 Hibernate 二级缓存。
Hibernate 一级缓存（Session 的缓存）：
（1）Session 实现了第一级 Cache，属于事务级数据缓冲。一旦事务结束，缓存随之失效。一个 Session 的生命周期对应一个数据库事务或一个程序事务。
（2）Session-Cache 总是被打开并且不能被关闭的。
（3）Session-Cache 保证一个 Session 中两次请求同一个对象时，取得的对象是同一个 Java 实例，有时它可以避免不必要的数据冲突。a.在对于同一个对象进行循环引用时，不至于产生堆栈溢出。b.当数据库事务结束时，对于同一数据表行，不会产生数据冲突。因为对于数据库中的一行，最多有一个对象来表示它。c.一个事务中可能会有很多个处理单元，在每一个处理单元中做的操作都会立即被其他的数据单元得知。
Hibernate 二级缓存（SessionFactory 的缓存）：
（1）二级缓存是 SessionFactory 范围内的缓存，所有的 Session 共享同一个二级缓存。在二级缓存中保存持久化实例的散装形式的数据。
（2）持久化不同的数据需要不同的 Cache 策略，比如一些因素将影响 Cache 策略的选择：数据的读/写比例、数据表是否能被其他的应用程序所访问等。
（3）设置 Hibernate 二级缓存需要分两步：首先，确认使用什么数据并发策略。然后，配置缓存过期时间并设置 Cache 提供器。

11. 什么是 Hibernate 延迟加载，如何实现延迟加载?

延迟加载机制是为了避免一些无谓的性能开销而提出来的,所谓延迟加载就是当在真正需要数据的时候，才真正执行数据加载操作。在 Hibernate 中提供了对实体对象的延迟加载以及对集合的延迟加载,另外在 Hibernate3 中还提供了对属性的延迟加载。延迟加载的过程:通过代理（Proxy）机制来实现延迟加载。Hibernate 从数据库获取某一个对象数据时、获取某一个对象的集合属性值时，或获取某一个对象所关联的另一个对象时，由于没有使用该对象的数据（除标识符外），Hibernate 并不从数据库加载真正的数据，而只是为该对象创建一个代理对象来代表这个对象，这个对象上的所有属性都为默认值；只有在真正需要使用该对象的数据时才创建这个真正的对象，真正从数据库中加载它的数据。通过 lazy 属性来控制懒加载机制，在映射文件中 set 标签中配置 lazy 属性，一般默认为 true，lazy 属性中有三个值，true 为懒加载 false 为不加载。extra 为及其懒惰(当用户只需要订单数时发送聚合函数去查询)。

12. 如何进行 Hibernate 的优化？

（1）数据库设计调整。（2）HQL 优化。（3）API 的正确使用(如根据不同的业务类型选用不同的集合及查询 API)。 （4）主配置参数(日志，查询缓存，fetch_size, batch_size 等)。 （5）映射文件优化(ID 生成策略，二级缓存，延迟加载，关联优化)。 （6）一级缓存的管理。（7）针对二级缓存，还有许多特有的策略。（8）事务控制策略。

13. 如何搭建一个 Hibernate 的环境

1.先导入 jar 包与配置文件、hibernate 启动 session 的工具类。
2.在配置文件中配置数据库的基本信息与数据库方言
3.进行测试，先创建实体类和数据库中的表。创建映射文件，命名规则是 实体类名.hbm.xml。位置要与实体类同一包下。在映射文件中配置 实体类与数据库表之间的映射关系。在 hibernate.cfg.xml 配置文件中添加映射文件的路径。
4.通过 hibernate 的工具类创建 sessionfactory，通过工厂创建 session 对象，通过 session 开启事务，进行数据操作后，事务提交。

14. Hibernate 中 session 有几种创建方式？都有那些区别？

有两种创建方式：
第一种是：通过 sessionfactory.getcurrentSession()创建 session，它是从当前线程中去找，看有没有 session，如果有则返回 session，如果没有则创建 session。属于单例模式
第二种是：通过 sessionfactory.opensession()创建 session，每次都是新创建一个 session。

15. Hibernate 中怎样实现类之间的关系?(如：一对多、多对多的关系)

一对多：
1.实体类中：一的一方用 set 集合保存多的一方 ，多的一方用对象来保存 1 的一方
2.在一的一方的映射文件中配置 set 标签，set 标签中 name 属性=“存放多的一方的属性”，key 标签中的 column 属性为外键字段。onetomany 标签的 class 属性为多的一方的全类名。
3.在多的一方的映射文件中配置 manytoone 标签，标签中的 name 属性为保存一的一方的属性名。配置 column 标签 标签中的属性 name 为外键
多对多：
1.在实体类中都用 set 集合保存对方
2.在映射文件中配置 set 标签，标签 name 为存放另一方的属性名，标签中 table 属性为中间表名，配置 key 标签，key 标签中 column 属性为当前表的属性，再配置 manytomany 标签。标签中 column 属性为另一方的外键 class 属性为另一方的实体类全类名。

16. 谈谈 Hibernate 中 inverse 的作用？

inverse 常用于一对多，多对多的映射文件中的 set 标签，inverse 属性设置为 true，是讲维护外键权反转到另一方，在一对多中，默认为 1 的一方，在多对多中，双方都维护，不设置权限反转会抛异常。

十、Struts2

1. 什么是 Struts2

Struts2 框架是一个按照 MVC 设计模式设计的 WEB 层框架开源框架，是在 struts 1 和 WebWork 的技术基础上进行了合并的全新的框架，更加灵活，易于使用和扩展。Struts 2 以 WebWork 为核心，采用拦截器的机制来处理用户的请求，这样的设计也使得业务逻辑控制器能够与 ServletAPI 完全脱离开。我们可以把 struts2 理解为一个大大的 servlet，这个 Servlet 名为 ActionServlet 或是 ActionServlet 的子类，我们可以在 web.xml 文件中将符合某种特征的所有请求交给这个 Servlet 处理，这个 Servlet 再参照一个配置文件(通常为 struts.xml)将各个请求分别分配给不同的 action 处理。Struts2 在处理客户端请求时，会先读取 web.xml 配置文件，根据前端控制器将符合条件的请求分给各个不同的 Action 处理。在此之前，ActionServlet 会把数据封装成一个 javaBean。 Struts2 框架提供了许多的拦截器，在封装数据的过程中，我们可以对数据进行一些操 作，例如：数据校验等等。当 Action 执行完后要返回一个结果视图，这个结果视图可以跟据 struts2 的配置文件中配置，选择转发或者重定向。

扩展知识点：struts 的配置文件可以有多个，可以按模块配置各自的配置文件，这样可以防止配置文件的过度膨胀；

优点:

- 实现 MVC 模式，结构清晰，使开发人员仅仅关注业务逻辑的实现。
- 有丰富的 tag 能够用，struts 的标记库(Taglib)，如能灵活动用，则能大大的提高开发效率。
- 页面导航，页面导航将是今后的一个发展方向。其实，这样做，使系统的脉络更加清晰。
- 提供 Exception 处理机制。
- 数据库链接池管理。
- 支持 I18N。

缺点:

- 转到展示层时，需要配置 forward，每次转到展示层。相信大多数都是直接转到 JSP，而是涉及到转向，需要配置 forward。假设有是个展示层的 JSP，需要配置十次 struts，并且还不包括有时候 文件夹、文件变更。需要又一次改动 forward。注意，每次改动配置之后。要求又一次部署整 个项目。而 tomcat 这种 server，还必须又一次启动 server，假设业务变更复杂频繁的系统，这种 操作简单不可想象。
- struts 的 action 必须是 thread-safe 方式，它仅仅同意一个实例去处理全部的请求。所以 action 用到的全部资源都必须统一同步。这个就引起了线程安全的问题。
- 测试不方便，struts 的每一个 action 都同 Web 层耦合在一起。这样它的测试依赖于 Web 容器，单元测试也非常难实现。
- 类型的转换，struts 的 FormBean 把全部的数据都作为 String 类型， 它能够使用工具 Commons-Beanutils 进行类型转化。但它的转化都是在 Class 级别，并且转化的类型是不可配 置的。类型转化时的错误信息返回给用户也是非常困难的。
- 对 Servlet 的依赖性过强，struts 处理 Action 时必须要依赖 ServletRequest 和 ServletResponse。
- 前端表达式语言方面，Struts 集成了 JSTL，所以它主要使用 JSTL 的表达式语言来获取数据。

2. Struts2 执行流程

- 客户端发送请求，请求到达服务端，由 struts 的核心控制器拦截请求。
- 核心控制器调用 action 映射器匹配请求路径和映射路径，判断映射路径是否存在。
- 核心控制器调用 actionProxy 代理器，actionProxy 代理调用配置管理器，配置管理器解析 struts.xml，匹配要 访问的 action，返回结果给 actionProxy 。
- actionProxy 代理调用对应的 action，执行业务逻辑操作，调用之前执行一系列的拦截器(封装请求参 数，数据校验等操作)。
- action 返回 string 字符串，配置管理器确定返回结果，倒着执行一系列的拦截器。
- 返回结果给客户端。

3. 说下 Struts 的设计模式

MVC 模式:
1、web 应用程序启动时就会加载并初始化 ActionServler。
2、用户提交表单时，一个配置好的 ActionForm 对象被创建，并被填入表单相应的数据，ActionServler 根据 Struts-config.xml 文件配置好的设置决定是否需要表单验证，如果需要就调用 ActionForm 的 Validate()验证后选择将请求发送到哪个 Action，如果 Action 不存在，ActionServlet 会先创建这个对象，然后调用 Action 的 execute（）方法.
3、Execute（）从 ActionForm 对象中获取数据，完成业务逻辑，返回一个 ActionForward 对象，ActionServlet 再把客户请求转发给 ActionForward 对象指定的 jsp 组件，ActionForward 对象指定的 jsp 生成动态的网页，返回给客户。

4. 哪个类是 Struts2 中的前端控制器？

org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter 是 Struts2 中的 Front Controller 类，每个请求处理都从这个类开始。早期版本的 Struts2 org.apache.struts2.dispatcher.FilterDispatcher 用作 Front Controller 类。

5. Struts2 中 Action 配置的注意事项有哪些？

1. name 包名称，在 struts2 的配置文件中，包名不能重复，name 并不是真正包名，只是为了管理 Action
2. namespace 和 <action>的 name 属性，决定 Action 的访问路径 (以/开始)
3. extends 继承哪个包，通常开发中继承 struts-default 包 (struts-default 包 在 struts-default.xml 中定义)【可以使用包中默认的拦截器和结果集】

6. Struts2 操作 URI 的默认后缀是什么？我们如何更改它？

Struts2 操作的默认 URI 后缀是.action，在 Struts1 中，默认后缀为.do。我们可以通过在 Struts2 配置文件中定义 struts.action.extension 常量值来更改此后缀：
<constant name="struts.action.extension" value="action，do"></constant>

7.　对我们的操作类使用 Action 接口和 ActionSupport 类有什么区别，您更喜欢哪一个？

我们可以实现 Action 接口来创建我们的动作类。这个接口有一个我们需要实现的方法 execute（）。使用此接口的唯一好处是它包含一些我们可以用于结果页面的常量，这些常量是 SUCCESS，ERROR，NONE，INPUT 和 LOGIN。
ActionSupport 类是 Action 接口的默认实现，它还实现了与 Validation 和 i18n 支持相关的接口。
ActionSupport 类实现 Action，Validateable，ValidationAware，TextProvider 和 LocaleProvider 接口。
我们可以覆盖 ActionSupport 类的 validate（）方法，以在我们的操作类中包含字段级验证登录。
根据需求，我们可以使用任何方法来创建 struts 2 动作类，我最喜欢的是 ActionSupport 类，因为它有助于在动作类中轻松编写验证和 i18n 逻辑。

8.　Struts2 中动作映射中命名空间的用途是什么？

Struts2 命名空间配置允许我们轻松创建模块。我们可以使用命名空间根据其功能分离我们的操作类，例如 admin，user，customer 等。

9.　什么是 struts-default 包，它有什么好处？

struts-default 是一个抽象包，它定义了所有 Struts2 拦截器和常用的拦截器堆栈。建议在配置应用程序包时扩展此程序包，以避免再次配置拦截器。这是为了帮助开发人员消除在我们的应用程序中配置拦截器和结果页面的繁琐任务。

10.　什么是 Struts2 中的拦截器？

拦截器是 Struts2 Framework 的支柱。Struts2 拦截器负责框架完成的大部分处理，例如将请求参数传递给动作类，使 Servlet API 请求，响应，会话可用于 Action 类，验证，i18n 支持等。
ActionInvocation 负责封装 Action 类和拦截器并按顺序触发它们。在 ActionInvocation 中使用的最重要的方法是 invoke（）方法，它跟踪拦截器链并调用下一个拦截器或动作。这是 Java EE 框架中责任链模式的最佳示例之一。

11.　Struts2 中拦截器有哪些好处？

拦截器在实现高度分离关注方面起着至关重要的作用。
Struts2 拦截器是可配置的，我们可以为我们想要的任何动作配置它。
我们可以创建自己的自定义拦截器来执行一些常见任务，例如请求参数记录，身份验证等。这有助于我们在一个位置处理常见任务，从而降低维护成本。
我们可以创建拦截器堆栈以用于不同的操作。

12.　Struts2 拦截器实现了哪种设计模式？

Struts2 拦截器基于拦截滤波器设计模式。拦截器堆栈中拦截器的调用非常类似于责任链设计模式。

13. Struts2 动作和拦截器是否是线程安全的？

Struts2 Action 类是线程安全的，因为对象是为每个处理它的请求实例化的。
Struts2 拦截器是单例类，并且创建了一个新线程来处理请求，因此它不是线程安全的，我们需要仔细实现它们以避免共享数据的任何问题。

14. 哪个拦截器负责将请求参数映射到动作类 Java Bean 属性？

com.opensymphony.xwork2.interceptor.ParametersInterceptor 拦截器负责将请求参数映射到 Action 类的 java bean 属性。此拦截器在 struts-default 包中配置，名称为“params”。此拦截器是 basicStack 和 defaultStack 拦截器堆栈的一部分。

15. 哪个拦截器负责 i18n 支持？

com.opensymphony.xwork2.interceptor.I18nInterceptor 拦截器负责 Struts2 应用程序中的 i18n 支持。此拦截器在 struts-default 包中配置，名称为“i18n”，它是 i18nStack 和 defaultStack 的一部分。

16. execAndWait 拦截器有什么用？

Struts2 为长时间运行的动作类提供了 execAndWait 拦截器。我们可以使用此拦截器将中间响应页面返回给客户端，一旦处理完成，最终响应将返回给客户端。此拦截器在 struts-default 包中定义，实现在 ExecuteAndWaitInterceptor 类中。

17. Struts2 中令牌拦截器的用途是什么？

Web 应用程序的主要问题之一是双表单提交。如果不注意，双重表单提交可能会导致向客户收取双倍金额或两次更新数据库值。我们可以使用令牌拦截器来解决双表格提交问题。这个拦截器是在 struts-default 包中定义的，但它不是任何拦截器堆栈的一部分，所以我们需要在我们的动作类中手动包含它。

18. 我们如何编写自己的拦截器并将其映射为动作？

我们可以实现 com.opensymphony.xwork2.interceptor.Interceptor 接口来创建自己的拦截器。一旦拦截器类准备就绪，我们需要在我们想要使用它的 struts.xml 包中定义它。我们还可以使用自定义拦截器和 defaultStack 拦截器创建拦截器堆栈。之后我们可以为我们想要使用拦截器的动作类配置它。

19. 什么是拦截器的生命周期？

拦截器接口定义了三个方法 - init（），destroy（）和 intercept（）。init 和 destroy 是拦截器的生命周期方法。拦截器是 Singleton 类，Struts2 初始化一个新线程来处理每个请求。创建拦截器实

例时调用 init（）方法，我们可以初始化此方法中的任何资源。应用程序关闭时调用 destroy（）方法，我们可以释放此方法中的任何资源。
intercept（）是每次客户端请求通过拦截器时调用的方法。

20. 简单介绍一下 Struts2 的值栈。

值栈是对应每一个请求对象的数据存储中心。Struts2 的一个很重要的特点就是引入了值栈。之前我们通过缓存或者模型驱动在 action 和页面之间传递数据，数据混乱，并且难以管理，缓存还有时间和数量限制，使用起来非常的困难。值栈的引入解决了这个问题，它可以统一管理页面和 action 之间的数据，供 action、result、interceptor 等使用。我们大多数情况下不需要考虑值栈在哪里，里面有什么，只需要去获取自己需要的数据就可以了，大大的降低了开发人员的工作量和逻辑复杂性。

21. 什么是拦截器堆栈？

拦截器堆栈可帮助我们将多个拦截器组合在一起以供进一步使用。struts-default 包创建了一些最常用的拦截器堆栈--basicStack 和 defaultStack。我们可以在包的开头创建我们自己的拦截器堆栈，然后配置我们的动作类来使用它。

22. 拦截器和过滤器有哪些区别？

- 拦截器是基于 java 的反射机制的，而过滤器是基于函数回调
- 拦截器不依赖与 servlet 容器，而过滤器依赖与 servlet 容器
- 拦截器只能对 action 请求起作用，而过滤器则可以对几乎所有的请求起作用
- 拦截器可以访问 action 上下文、值栈里的对象，而过滤器不能在 action 的生命周期中，拦截器可以多次被调用，而过滤器只能在容器初始化时被调用一 次

23. 在 Struts2 中创建 Action 类有哪些不同的方法？

Struts2 提供了创建动作类的不同方法。通过实现 Action 接口 使用 Struts2 @Action 注释 通过扩展 ActionSupport 类 任何返回 String 的 execute（）方法的普通 java 类都可以配置为 Action 类。

24. 什么是 ValueStack 和 OGNL？

ValueStack 是 Struts2 存储应用程序数据以处理客户端请求的存储区域。数据存储在 ActionContext 使用 ThreadLocal 的对象中，以具有特定于特定请求线程的值。
对象图导航语言（OGNL）是一种功能强大的表达式语言，用于处理存储在 ValueStack 上的数据。正如您在架构图中看到的，拦截器和结果页面都可以使用 OGNL 访问存储在 ValueStack 上的数据。

25. 列举 Struts2 中引入的一些有用的注释？

Struts2 中引入的一些重要注释是：
@Action 创建动作类
@Actions 为多个动作配置单个类
@Namespace 和@Namespaces 用于创建不同的模块
@Result 用于结果页面
@ResultPath 用于配置结果页面位置

26. 提供一些您使用过的重要 Struts2 常量？

1.struts.devMode 在开发模式下运行我们的应用程序。此模式会重新加载属性文件，并提供额外的日志记录和调试功能。它在开发我们的应用程序时非常有用，但我们应该在将代码转移到生产时将其关闭。
2.struts.convention.result.path 用于配置结果页面的位置。默认情况下，Struts2 在{WEBAPP-ROOT} / {Namespace} /中查找结果页面，我们可以使用此常量更改位置。
3.struts.custom.i18n.resources 为 i18n 支持定义全局资源包。
4.struts.action.extension 为 Struts2 应用程序配置 URL 后缀。默认后缀是.action 但有时我们可能想将其更改为.do 或其他内容。我们可以在 struts.xml 文件中配置上面的常量，如下所示。

```
<constant name="struts.devMode" value="true"></constant>
<constant name="struts.action.extension" value="action，do"></constant>
<constant name="struts.custom.i18n.resources" value="global"></constant>
<constant name="struts.convention.result.path" value="/"></constant>
```

27. 我们怎样才能在动作类中获得 Servlet API 请求，响应，HttpSession 等对象？

Struts2 操作类不提供对 Servlet API 组件的直接访问，例如 Request，Response 和 Session。但是，有时我们需要在操作类中进行这些访问，例如检查 HTTP 方法或设置响应中的 cookie。
这就是为什么 Struts2 API 提供了一堆* Aware 接口，我们可以实现这些接口来访问这些对象。Struts2 API 使用依赖注入在操作类中注入 Servlet API 组件。一些重要的 Aware 接口是 SessionAware，ApplicationAware，ServletRequestAware 和 ServletResponseAware。

28. 我们如何在 Struts2 应用程序中集成 log4j？

Struts2 提供了 log4j API 的简单集成以便进行日志记录，我们需要的只是 WEB-INF / classes 目录中的 log4j 配置文件。

29. 什么是不同的 Struts2 标签？我们怎样才能使用它们？

Struts2 提供了许多自定义标记，我们可以在结果页面中使用它们来创建客户端请求的视图。这些标签大致分为三类 - 数据标签，控制标签和 UI 标签。
我们可以通过使用 taglib 指令在 JSP 页面中添加这些标记来使用这些标记。

<%@ taglib uri="/struts-tags" prefix="s" %>
一些重要的 Data 标签是 property，set，push，bean，action，include，i18n 和 text 标签。
控制标签用于处理和导航集合中的数据。一些重要的 Control 标签是 if-elseif-else，iterator，append，merge，sort，subset 和 generator 标签。
Struts2 UI 标签用于生成 HTML 标记语言，将 HTML 表单数据绑定到动作类属性，类型转换，验证和 i18n 支持。一些重要的 UI 标签是 form，textfield，password，textarea，checkbox，select，radio 和 submit 标签。

30. 什么是 Struts2 中的自定义类型转换器？

Struts2 支持 OGNL 表达式语言，它在 Struts 2 中执行两项重要任务 - 数据传输和类型转换。
OGNL 非常灵活，我们可以轻松扩展它以创建我们自己的自定义转换器类。创建和配置自定义类型转换器类非常简单，第一步是修复自定义类的输入格式。第二步是实现转换器类。类型转换器类应该实现 com.opensymphony.xwork2.conversion.TypeConverter 接口。由于在 Web 应用程序中，我们总是以 String 的形式获取请求并以 String 的形式发送响应，Struts 2 API 提供了 TypeConverter 接口的默认实现，即 StrutsTypeConverter。StrutsTypeConverter 包含两个抽象方法 - convertFromString 将 String 转换为 Object，convertToString 将 Object 转换为 String。

31. 结果页面的默认位置是什么？我们如何更改它？

默认情况下，Struts2 在{WEBAPP-ROOT} / {Namespace} /目录中查找结果页面，但有时我们希望将结果页面保存在另一个位置，我们可以在 Struts2 配置文件中提供 struts.convention.result.path 常量值来更改结果页面位置。
另一种方法是在操作类中使用@ResultPath 批注来提供结果页面位置。

32. 我们如何在 Struts2 应用程序中上传文件？

文件上载是 Web 应用程序中的常见任务之一。这就是为什么 Struts2 通过 FileUploadInterceptor 提供对文件上传的内置支持。此拦截器在 struts-default 包中配置，并提供选项以设置文件的最大大小和可以上载到服务器的文件类型。

33. 开发 Struts2 应用程序时要遵循哪些最佳实践？

开发 Struts2 应用程序时的一些最佳实践是：
1.在创建程序包时始终尝试扩展 struts-default 程序包，以避免在配置拦截器时出现代码冗余。
2.对于整个应用程序中的常见任务，例如记录请求参数，请尝试使用拦截器。
3.始终将动作类 java bean 属性保存在单独的 bean 中以便重用代码并实现 ModelDriven 接口。
4.如果您有将在多个操作中使用的自定义拦截器，请为此创建拦截器堆栈，然后使用它。
5.尝试使用基于功能区域的命名空间配置在不同模块中划分应用程序。
6.尝试在结果页面中使用 Struts2 标记进行代码说明，如果需要，可以创建自己的类型转换器。
7.使用开发模式可以加快开发速度，但请确保生产代码不以 dev 模式运行。
8.使用 Struts2 i18n 支持资源包并支持本地化。
9.Struts2 提供了许多可以拥有资源包的地方，但是尝试保留一个全局资源包，一个用于动作类以

避免混淆。struts-default 包配置所有拦截器并创建不同的拦截器堆栈。尝试仅使用所需的内容，例如，如果您没有本地化要求，则可以避免使用 i18n 拦截器。

34. Struts2 的封装方式有哪些？

一、属性封装 1. 在 action 中设置成员变量，变量名与表单中的 name 属性值相同 2. 生成变量的 set 方法实例：获取用户输入的用户名和密码 jsp 页面如下：java 代码如下：
二、模型驱动（常用 1. action 实现 ModeDriven 接口 2. 在 action 里创建实体类对象 3. 实现接口的 getModel 方法并返回所创建的对象示例：获取用户输入的用户名和密码 jsp 页面如下：java 代码如下：需注意的是表单 name 的值应与类的属性名相同。
三、表达式封 1. 在 action 中声明实体类 2. 生成实体类的 set 和 get 方法 3. 在表单输入项的 name 属性值里面写表达式 jsp 页面如下：java 代码如下：

35. Struts2 中的 # 和 % 分别是做什么的？

（1）使用#获取 context 里面数据<s:iterator value = “list” var=”user”><s:property value = “#user.username”></s:iterator>（2）向 request 域放值(获取 context 里面数据，写 ognl 时候，首先添加符号#context 的 key 名称.域对象名称)
（2）在页面中使用 ognl 获取<s:property value = “#request.req”>
（3）%在 struts2 标签中表单标签在 struts2 标签里面使用 ognl 表达式，如果直接在 struts2 表单标签里面使用 ognl 表达式不识别，只有%之后才会识别。<s:textfield name=”username” value=”%{#request.req}”>

36. Struts2 中有哪些常用结果类型？

1）dispatcher ：默认的请求转发的结果类型，Action 转发给 JSP
2) chain ：Action 转发到另一个 Action （同一次请求）
2）redirect ： 重定向，重定向到一个路径信息，路径信息没有限制（不在一个请求中），Action 重定向到 JSP
3）redirectAction ：Action 重定向到另一个 Action
4）stream ：将原始数据作为流传递回浏览器端，该结果类型对下载的内容和图片非常有用。
5）freemarker ：呈现 freemarker 模板。
7）plaintext ：返回普通文本内容。

37. SpringMVC 和 Struts2 的区别？

1、Struts2 是类级别的拦截，一个类对应一个 request 上下文，SpringMVC 是方法级别的拦截，一个方法对应一个 request 上下文，而方法同时又跟一个 url 对应，所以说从架构本身上 SpringMVC 就容易实现 restful url，而 struts2 的架构实现起来要费劲，因为 Struts2 中 Action 的一个方法可以对应一个 url，而其类属性却被所有方法共享，这也就无法用注解或其他方式标识其所属方法了。
2、由上边原因，SpringMVC 的方法之间基本上独立的，独享 request response 数据，请求数据通过参数获取，处理结果通过 ModelMap 交回给框架，方法之间不共享变量，而 Struts2 搞的

就比较乱，虽然方法之间也是独立的，但其所有 Action 变量是共享的，这不会影响程序运行，却给我们编码 读程序时带来麻烦，每次来了请求就创建一个 Action，一个 Action 对象对应一个 request 上下文。

3、由于 Struts2 需要针对每个 request 进行封装，把 request，session 等 servlet 生命周期的变量封装成一个一个 Map，供给每个 Action 使用，并保证线程安全，所以在原则上，是比较耗费内存的。

4、拦截器实现机制上，Struts2 有以自己的 interceptor 机制，SpringMVC 用的是独立的 AOP 方式，这样导致 Struts2 的配置文件量还是比 SpringMVC 大。

5、SpringMVC 的入口是 servlet，而 Struts2 是 filter（这里要指出，filter 和 servlet 是不同的。），这就导致了二者的机制不同，这里就牵涉到 servlet 和 filter 的区别了。

6、SpringMVC 集成了 Ajax，使用非常方便，只需一个注解@ResponseBody 就可以实现，然后直接返回响应文本即可，而 Struts2 拦截器集成了 Ajax，在 Action 中处理时一般必须安装插件或者自己写代码集成进去，使用起来也相对不方便。

7、SpringMVC 验证支持 JSR303，处理起来相对更加灵活方便，而 Struts2 验证比较繁琐，感觉太烦乱。

8、Spring MVC 和 Spring 是无缝的。从这个项目的管理和安全上也比 Struts2 高（当然 Struts2 也可以通过不同的目录结构和相关配置做到 SpringMVC 一样的效果，但是需要 xml 配置的地方不少）。

9、设计思想上，Struts2 更加符合 OOP 的编程思想， SpringMVC 就比较谨慎，在 servlet 上扩展。

10、SpringMVC 开发效率和性能高于 Struts2。

11、SpringMVC 可以认为已经 100%零配置。

十一、Spring

1. Spring

Spring 是个 java 企业级应用的开源开发框架，Spring 主要用来开发 Java 应用，但是有些扩展是针对构建 J2EE 平台的 WEB 应用。Spring 框架目标是简化 Java 企业级应用开发，并通过 POJO 为基础的编程模型促进良好的编程习惯。Spring 可以是使简单的 JavaBean 实现以前只有 EJB 才能实现的功能。

Spring 容器的主要核心是：控制反转（IOC），传统的 java 开发模式中，当需要一个对象时，我们会自己使用 new 或者 getInstance 等直接或者间接调用构造方法创建一个对象。而在 spring 开发模式中，spring 容器使用了工厂模式为我们创建了所需要的对象，不需要我们自己创建了，直接调用 spring 提供的对象就可以了，这是控制反转的思想。依赖注入（DI），spring 使用 javaBean 对象的 set 方法或者带参数的构造方法为我们在创建所需对象时将其属性自动设置所需要的值的过程，就是依赖注入的思想。面向切面编程（AOP），在面向对象编程（oop）思想中，我们将事物纵向抽成一个个的对象。而在面向切面编程中，我们将一个个的对象某些类似的方面横向抽成一个切面，对这个切面进行一些如权限控制、事物管理，记录日志等公用操作处理的过程就是面向切面编程的思想。AOP 底层是动态代理，如果是接口采用 JDK 动态代理，如果是类采用 CGLIB 方式实现动态代理。

2. Spring 好处:

（1）轻量:Spring 是轻量的,基本的版本大约 2MB。
（2）控制反转:Spring 通过控制反转实现了松散耦合,对象们给它们的依赖,而不是创建或查找依赖的对象们。
（3）面向切面编程(AOP):Spring支持面向切面编程,并且把应用业务逻辑和系统服务分开。
（4）容器:Spring 包含并管理应用中对象的声明周期和配置。
（5）MVC 框架:Spring 的 WEB 框架是个精心设计的框架,是 Web 框架的一个很好的替代品
（6）事务管理:Spring 提供一个持续的事务管理接口,可以扩展到上至本地事务下至全局事务(JTA)。
（7）异常处理:Spring 提供方便的 API 把具体技术相关的异常(比如由 JDBC,HibernateorJDO 抛出的)转化为一致的 unchecked 异常。

3. Spring 能帮我们做什么？

a. Spring 能帮我们根据配置文件创建及组装对象之间的依赖关系。Spring 根据配置文件来进行创建及组装对象间依赖关系，只需要改配置文件即可
b. Spring 面向切面编程能帮助我们无耦合的实现日志记录，性能统计，安全控制。Spring 面向切面编程能提供一种更好的方式来完成，一般通过配置方式，而且不需要在现有代码中添加任何额外代码，现有代码专注业务逻辑。
c. Spring 能非常简单的帮我们管理数据库事务。采用 Spring，我们只需获取连接，执行 SQL，其他事物相关的都交给 Spring 来管理了。
d. Spring 还能与第三方数据库访问框架（如 Hibernate、JPA）无缝集成，而且自己也提供了一套 JDBC 访问模板，来方便数据库访问。
e.Spring 还能与第三方 Web（如 Struts、JSF）框架无缝集成，而且自己也提供了一套 SpringMVC 框架，来方便 web 层搭建。f.Spring 能方便的与 JavaEE（如 JavaMail、任务调度）整合，与更多技术整合（比如缓存框架）。

4. Spring 结构

（1）核心容器：包括 Core、Beans、Context、EL 模块。Core 模块：封装了框架依赖的最底层部分，包括资源访问、类型转换及一些常用工具类。Beans 模块：提供了框架的基础部分，包括反转控制和依赖注入。其中 BeanFactory 是容器核心，本质是“工厂设计模式”的实现，而且无需编程实现“单例设计模式”，单例完全由容器控制，而且提倡面向接口编程，而非面向实现编程；所有应用程序对象及对象间关系由框架管理，从而真正把你从程序逻辑中把维护对象之间的依赖关系提取出来，所有这些依赖关系都由 BeanFactory 来维护。Context 模块：以 Core 和 Beans 为基础，集成 Beans 模块功能并添加资源绑定、数据验证、国际化、JavaEE 支持、容器生命周期、事件传播等；核心接口是 ApplicationContext。EL 模块：提供强大的表达式语言支持，支持访问和修改属性值，方法调用，支持访问及修改数组、容器和索引器，命名变量，支持算数和逻辑运算，支持从 Spring 容器获取 Bean，它也支持列表投影、选择和一般的列表聚合等。
（2）AOP、Aspects 模块：AOP 模块：SpringAOP 模块提供了符合 AOPAlliance 规范的面向方面的编程（aspect-orientedprogramming）实现，提供比如日志记录、权限控制、性能统计等通用功能和业务逻辑分离的技术，并且能动态的把这些功能添加到需要的代码中；这样各专其职，降低

业务逻辑和通用功能的耦合。Aspects 模块：提供了对 AspectJ 的集成，AspectJ 提供了比 SpringASP 更强大的功能。数据访问/集成模块：该模块包括了 JDBC、ORM、OXM、JMS 和事务管理。事务模块：该模块用于 Spring 管理事务，只要是 Spring 管理对象都能得到 Spring 管理事务的好处，无需在代码中进行事务控制了，而且支持编程和声明性的事务管理。

（3）JDBC 模块：提供了一个 JBDC 的样例模板，使用这些模板能消除传统冗长的 JDBC 编码还有必须的事务控制，而且能享受到 Spring 管理事务的好处。ORM 模块：提供与流行的“对象-关系”映射框架的无缝集成，包括 Hibernate、JPA、MyBatis 等。而且可以使用 Spring 事务管理，无需额外控制事务。

（4）OXM 模块：提供了一个对 Object/XML 映射实现，将 java 对象映射成 XML 数据，或者将 XML 数据映射成 java 对象，Object/XML 映射实现包括 JAXB、Castor、XMLBeans 和 XStream。

（5）JMS 模块：用于 JMS(JavaMessagingService)，提供一套“消息生产者、消息消费者”模板用于更加简单的使用 JMS，JMS 用于用于在两个应用程序之间，或分布式系统中发送消息，进行异步通信。

（6）Web/Remoting 模块：Web/Remoting 模块包含了 Web、Web-Servlet、Web-Struts、Web-Porlet 模块。

（7）Web 模块：提供了基础的 web 功能。例如多文件上传、集成 IoC 容器、远程过程访问（RMI、Hessian、Burlap）以及 WebService 支持，并提供一个 RestTemplate 类来提供方便的 Restfulservices 访问。

（8）Web-Servlet 模块：提供了一个 SpringMVCWeb 框架实现。SpringMVC 框架提供了基于注解的请求资源注入、更简单的数据绑定、数据验证等及一套非常易用的 JSP 标签，完全无缝与 Spring 其他技术协作。

（9）Web-Struts 模块：提供了与 Struts 无缝集成，Struts1.x 和 Struts2.x 都支持

（10）Test 模块：Spring 支持 Junit 和 TestNG 测试框架，而且还额外提供了一些基于 Spring 的测试功能，比如在测试 Web 框架时，模拟 Http 请求的功能。

5. Spring 核心容器(应用上下文)模块

这是基本的 Spring 模块，提供 Spring 框架的基础功能，BeanFactory 是任何以 Spring 为基础的应用的核心。Spring 框架建立在此模块之上，它使 Spring 成为一个容器。

6. ApplicationContext 通常的实现是什么

FileSystemXmlApplicationContext:此容器从一个 XML文件中加载 beans 的定义 XMLBean 配置文件的全路径名必须提供它的构造函数。ClassPathXmlApplicationContext:此容器也从一个 XML 文件中加载 beans 的定义,这里需要正确设置 classpath 因为这个容器将在 classpath里找 bean 配置。WebXmlApplicationContext:此容器加载一个 XML文件,此文件定义了一个 WEB 应用的所有 bean。

7. 什么是 Springbeans?

Springbeans 是那些形成 Spring 应用的主干的 java 对象。它们被 SpringIOC 容器初始化，装配，和管理。这些 beans 通过容器中配置的元数据创建。比如，以 XML 文件中<bean/>的形式定义。Spring 框架定义的 beans 都是单例 beans。在 beantag 中有个属性”singleton”，如果它被赋为 TRUE，bean 就是单件，否则就是一个 prototypebean。默认是 TRUE，所以所有在 Spring 框架中

的 beans 缺省都是单件。

8. 什么是 Spring 的内部 bean？

当一个 bean 仅被用作另一个 bean 的属性时，它能被声明为一个内部 bean，为了定义 innerbean，在 Spring 的基于 XML 的配置元数据中，可以在<property/>或<constructor-arg/>元素内使用<bean/>元素，内部 bean 通常是匿名的，它们的 Scope 一般是 prototype。

9. 你怎样定义类的作用域?

当定义一个<bean>在 Spring 里，我们还能给这个 bean 声明一个作用域。它可以通过 bean 定义中的 scope 属性来定义。如，当 Spring 要在需要的时候每次生产一个新的 bean 实例，bean 的 scope 属性被指定为 prototype。另一方面，一个 bean 每次使用的时候必须返回同一个实例，这个 bean 的 scope 属性必须设为 singleton。

10. 什么是 bean 的自动装配？

无须在 Spring 配置文件中描述 javaBean 之间的依赖关系（如配置<property>、<constructor-arg>）。IOC 容器会自动建立 javabean 之间的关联关系。

11. 一个 SpringBean 定义包含什么？

一个 SpringBean 的定义包含容器必知的所有配置元数据，包括如何创建一个 bean，它的生命周期详情及它的依赖。

12. 一个 SpringBeans 的定义需要包含什么？

一个 SpringBean 的定义包含容器必知的所有配置元数据，包括如何创建一个 bean，它的生命周期详情及它的依赖。

13. 解释 Spring 支持的几种 bean 的作用域。

Spring 框架支持以下五种 bean 的作用域：
（1）singleton: bean 在每个 Springioc 容器中只有一个实例。
（2）prototype：一个 bean 的定义可以有多个实例。
（3）request：每次 http 请求都会创建一个 bean，该作用域仅在基于 web 的 SpringApplicationContext 情形下有效。
（4）session：在一个 HTTPSession 中，一个 bean 定义对应一个实例。该作用域仅在基于 web 的 SpringApplicationContext 情形下有效。
（5）global-session：在一个全局的 HTTPSession 中，一个 bean 定义对应一个实例。该作用域仅

在基于 web 的 SpringApplicationContext 情形下有效。
缺省的 Springbean 的作用域是 Singleton.

14. 简单介绍一下 Springbean 的生命周期

bean 定义：在配置文件里面用<bean></bean>来进行定义。bean 初始化：有两种方式初始化 1.在配置文件中通过指定 init-method 属性来完成 2. 实现 org.springframwork.beans.factory.InitializingBean 接口 bean 调用：有三种方式可以得到 bean 实例，并进行调用 bean 销毁：销毁有两种方式 1.使用配置文件指定的 destroy-method 属性 2.实现 org.springframwork.bean.factory.DisposeableBean 接口

15. 哪些是重要的 bean 生命周期方法？你能重载它们吗？

1. 有两个重要的 bean 生命周期方法，第一个是 setup，它是在容器加载 bean 的时候被调用。第二个方法是 teardown 它是在容器卸载类的时候被调用。
2. Thebean 标签有两个重要的属性（init-method 和 destroy-method）。用它们你可以自己定制初始化和注销方法。它们也有相应的注解（@PostConstruct 和@PreDestroy）。

16. BeanFactory 常用的实现类有哪些？

Bean 工厂是工厂模式的一个实现，提供了控制反转功能，用来把应用的配置和依赖从正真的应用代码中分离。常用的 BeanFactory 实现有 DefaultListableBeanFactory、XmlBeanFactory、ApplicationContext 等。XMLBeanFactory，最常用的就是 org.springframework.beans.factory.xml.XmlBeanFactory，它根据 XML 文件中的定义加载 beans。该容器从 XML 文件读取配置元数据并用它去创建一个完全配置的系统或应用。

17. BeanFactory 与 AppliacationContext 有什么区别

3. BeanFactory 基础类型的 IOC 容器，提供完成的 IOC 服务支持。如果没有特殊指定，默认采用延迟初始化策略。相对来说，容器启动初期速度较快，所需资源有限。
2.ApplicationContextApplicationContext 是在 BeanFactory 的基础上构建，是相对比较高级的容器实现，除了 BeanFactory 的所有支持外，ApplicationContext 还提供了事件发布、国际化支持等功能。ApplicationContext 管理的对象，在容器启动后默认全部初始化并且绑定完成。

18. Spring 框架中的单例 bean 是线程安全的吗?

Spring 框架中的单例 bean 不是线程安全的。

19. 你怎样定义类的作用域?

当定义一个<bean>在 Spring 里，我们还能给这个 bean 声明一个作用域。它可以通过 bean 定义中的 scope 属性来定义。如，当 Spring 要在需要的时候每次生产一个新的 bean 实例，bean 的 scope

属性被指定为 prototype。另一方面，一个 bean 每次使用的时候必须返回同一个实例，这个 bean 的 scope 属性必须设为 singleton。

20. XMLBeanFactory

最常用的就是 org.springframework.beans.factory.xml.XmlBeanFactory,它根据 XML文件中的定义加载 benas。该容器从 XML文件读取配置元数据并用它去创建一个完全配置的系统或应用。

21. 如何给 Spring 容器提供配置元数据?

这里有三种重要的方法给 Spring 容器提供配置元数据。
XML 配置文件。
基于注解的配置。
基于 java 的配置。

22. Spring 配置文件

Spring 配置文件是个 XML文件，这个文件包含了类信息，描述了如何配置他们，以及如何相互调用。

23. 什么是 SpringIOC 容器？

IOC 控制反转：SpringIOC 负责创建对象，管理对象。通过依赖注入（DI），装配对象，配置对象，并且管理这些对象的整个生命周期。

24. 什么是 Spring 的依赖注入？

IOC 的一个重点是在系统运行中,动态的向某个对象提供它所需要的其他对象。这一点是通过 DI(依赖注入)来实现的。平常的 java 开发中，程序员在某个类中需要依赖其它类的方法，则通常是 new 一个依赖类再调用类实例的方法，这种开发存在的问题是 new 的类实例不好统一管理，spring 提出了依赖注入的思想，即依赖类不由程序员实例化，而是通过 spring 容器帮我们 new 指定实例并且将实例注入到需要该对象的类中。依赖注入的另一种说法是“控制反转”，通俗的理解是：平常我们 new 一个实例，这个实例的控制权是我们程序员，而控制反转是指 new 实例工作不由我们程序员来做而是交给 spring 容器来做。
依赖注入，是 IOC 的一个方面，是个通常的概念，它有多种解释。这概念是说你不用创建对象，而只需要描述它如何被创建。不在代码里直接组装你的组件和服务，但是要在配置文件里描述哪些组件需要哪些服务，之后一个容器（IOC 容器）负责把他们组装起来。那么 DI 是如何实现的呢?Java1.3 之后一个重要特征是反射(reflection),它允许程序在运行的时候动态的生成对象，执行对象的方法、改变对象的属性，spring 就是通过反射来实现注入的。

25. SpringIOC（控制反转）：

SpringIOC（InversionofControl）负责创建对象,管理对象(通过依赖注入(DI))，装配对象，配置对象，并且管理这些对象的整个生命周期。Ioc 是 Spring 所倡导的开发方式,所有的类都会在 spring 容器中登记,告诉 spring 你是个什么东西,你需要什么东西，然后 spring 会在系统运行到适当的时候，把你要的东西主动给你,同时也把你交给其他需要你的东西。所有类的创建,销毁都由 spring 来控制,也就是说控制对象生存周期的不再是引用它的对象，而是 spring。对于某个具体的对象而言，以前是它控制其他对象，现在是所有对象都被 spring 控制，这件控制反转。

26. IOC 的优点是什么？

IOC 或依赖注入把应用的代码量降到最低。它使应用容易测试，单元测试不再需要单例和 JNDI 查找机制。最小的代价和最小的侵入性使松散耦合得以实现。IOC 容器支持加载服务时的饿汉式初始化和懒加载。

27. 有哪些不同类型的 IOC（依赖注入）方式？

Spring 提供了多种依赖注入的方式。
1. Set 注入：Setter 方法注入是容器通过调用无参构造器或无参 static 工厂方法实例化 bean 之后，调用该 bean 的 setter 方法，即实现了基于 setter 的依赖注入。
2. 构造器注入：构造器依赖注入通过容器触发一个类的构造器来实现的，该类有一系列参数，每个参数代表一个对其他类的依赖。
3. 基于注解的注入

28. 解释不同方式的自动装配。

有五种自动装配的方式，可以用来指导 Spring 容器用自动装配方式来进行依赖注入。
1）no：默认的方式是不进行自动装配，通过显式设置 ref 属性来进行装配。
2）byName：通过参数名自动装配，Spring 容器在配置文件中发现 bean 的 autowire 属性被设置成 byname，之后容器试图匹配、装配和该 bean 的属性具有相同名字的 bean。
3）byType：通过参数类型自动装配，Spring 容器在配置文件中发现 bean 的 autowire 属性被设置成 byType，之后容器试图匹配、装配和该 bean 的属性具有相同类型的 bean。如果有多个 bean 符合条件，则抛出错误。
4）constructor：这个方式类似于 byType，但是要提供给构造器参数，如果没有确定的带参数的构造器参数类型，将会抛出异常。
5）autodetect：首先尝试使用 constructor 来自动装配，如果无法工作，则使用 byType 方式。

29. 在 Spring 中如何注入一个 java 集合？

Spring 提供以下几种集合的配置元素：
<list>类型用于注入一列值，允许有相同的值。
<set>类型用于注入一组值，不允许有相同的值。

<map>类型用于注入一组键值对，键和值都可以为任意类型。
<props>类型用于注入一组键值对，键和值都只能为 String 类型。

30. 哪种依赖注入方式你建议使用，构造器注入，还是 Setter 方法注入？

两种依赖方式都可以使用，构造器注入和 Setter 方法注入。最好的解决方案是用构造器参数实现强制依赖，setter 方法实现可选依赖。

31. Spring 中的设计模式

a. 单例模式——spring 中两种代理方式，若目标对象实现了若干接口，spring 使用 jdk 的 java.lang.reflect.Proxy-Java 类代理。若目标兑现没有实现任何接口，spring 使用 CGLIB 库生成目标类的子类。单例模式——在 spring 的配置文件中设置 bean 默认为单例模式。
b. 模板方式模式——用来解决代码重复的问题。比如：RestTemplate、JmsTemplate、JpaTemplate
c. 前端控制器模式——spring 提供了前端控制器 DispatherServlet 来对请求进行分发。
d. 试图帮助（viewhelper）——spring 提供了一系列的 JSP 标签，高效宏来帮助将分散的代码整合在试图中。
e. 依赖注入——贯穿于 BeanFactory/ApplacationContext 接口的核心理念
f. 工厂模式——在工厂模式中，我们在创建对象时不会对客户端暴露创建逻辑，并且是通过使用同一个接口来指向新创建的对象。Spring 中使用 beanFactory 来创建对象的实例。

32. 什么是基于注解的容器配置?

相对于 XML 文件，注解型的配置依赖于通过字节码元数据装配组件，而非尖括号的声明。开发者通过在相应的类，方法或属性上使用注解的方式，直接组件类中进行配置，而不是使用 xml 表述 bean 的装配关系。

33. 怎样开启注解装配?

注解装配在默认情况下是不开启的，为了使用注解装配，我们必须在 Spring 配置文件中配置<context:annotation-config/>元素。

34. Spring 的常用注解

Spring 在 2.5 版本以后开始支持注解的方式来配置依赖注入。可以用注解的方式来代替 xml 中 bean 的描述。注解注入将会被容器在 XML 注入之前被处理，所以后者会覆盖掉前者对于同一个属性的处理结果。注解装配在 spring 中默认是关闭的。所以需要在 spring 的核心配置文件中配置一下才能使用基于注解的装配模式。
配置方式如下：<context:annotation-config/>
常用的注解：@Required:该注解应用于设值方法
@Autowired：该注解应用于有值设值方法、非设值方法、构造方法和变量。
@Qualifier：该注解和@Autowired 搭配使用，用于消除特定 bean 自动装配的歧义。

35. 解释对象/关系映射集成模块

Spring 通过 ORM 模块，支出我们在 JDBC 之上使用一个对象/关系映射(ORM)工具，Spring支持集成主流的 ORM 框架，比如 Hiberate，JDO 和 IBATISSQLMaps。Spring 的事务管理同样支持以上所有 ORM 框架及 JDBC。

36. 简单解释一下 spring 的 AOP

AOP（AspectOrientedProgramming），即面向切面编程，可以说是 OOP（ObjectOrientedProgramming，面向对象编程）的补充和完善。OOP 引入封装、继承、多态等概念来建立一种对象层次结构，用于模拟公共行为的一个集合。不过 OOP 允许开发者定义纵向的关系，但并不适合定义横向的关系，例如日志功能。日志代码往往横向地散布在所有对象层次中，而与它对应的对象的核心功能毫无关系对于其他类型的代码，如安全性、异常处理和透明的持续性也都是如此，这种散布在各处的无关的代码被称为横切（crosscutting），在 OOP 设计中，它导致了大量代码的重复，而不利于各个模块的重用。AOP 技术恰恰相反，它利用一种称为"横切"的技术，剖解开封装的对象内部，并将那些影响了多个类的公共行为封装到一个可重用模块，并将其命名为"Aspect"，即切面。所谓"切面"，简单说就是那些与业务无关，却为业务模块所共同调用的逻辑或责任封装起来，便于减少系统的重复代码，降低模块之间的耦合度，并有利于未来的可操作性和可维护性。使用"横切"技术，AOP 把软件系统分为两个部分：核心关注点和横切关注点。业务处理的主要流程是核心关注点，与之关系不大的部分是横切关注点。横切关注点的一个特点是，他们经常发生在核心关注点的多处，而各处基本相似，比如权限认证、日志、事物。AOP 的作用在于分离系统中的各种关注点，将核心关注点和横切关注点分离开来。AOP 核心就是切面，它将多个类的通用行为封装成可重用的模块，该模块含有一组 API 提供横切功能。比如，一个日志模块可以被称作日志的 AOP 切面。根据需求的不同，一个应用程序可以有若干切面。在 SpringAOP 中，切面通过带有@Aspect 注解的类实现。

37. AOP 底层实现方式？

动态代理，会引入到代理模式的问题。

38. 在 SpringAOP 中，关注点和横切关注的区别是什么？

关注点是应用中一个模块的行为，一个关注点可能会被定义成一个我们想实现的一个功能。横切关注点是一个关注点，此关注点是整个应用都会使用的功能，并影响整个应用，比如日志，安全和数据传输，几乎应用的每个模块都需要的功能。因此这些都属于横切关注点。

39. 什么是目标对象?

被一个或者多个切面所通知的对象。它通常是一个代理对象。也指被通知（advised）对象。

40. 什么是切点？

切入点是一个或一组连接点，通知将在这些位置执行。可以通过表达式或匹配的方式指明切入点。

41. 什么是连接点？

被拦截到的点，因为 Spring 只支持方法类型的连接点，所以在 Spring 中连接点指的就是被拦截到的方法，实际上连接点还可以是字段或者构造器。

42. 什么是织入？什么是织入应用的不同点？

织入是将切面和到其他应用类型或对象连接或创建一个被通知对象的过程。织入可以在编译时，加载时，或运行。

43. 什么是代理?

代理是通知目标对象后创建的对象。从客户端的角度看，代理对象和目标对象是一样的。

44. Spring 的通知是什么？有哪几种类型？

通知是个在方法执行前或执行后要做的动作，实际上是程序执行时要通过 SpringAOP 框架触发的代码段。Spring 切面可以应用五种类型的通知：
1）before：前置通知，在一个方法执行前被调用。
2）after:在方法执行之后调用的通知，无论方法执行是否成功。
3）after-returning:仅当方法成功完成后执行的通知。
4）after-throwing:在方法抛出异常退出时执行的通知。
5）around:在方法执行之前和之后调用的通知。

45. 解释 JDBC 抽象和 DAO 模块。

通过使用JDBC 抽象和 DAO 模块，保证数据库代码的简介，并能避免数据库资源错误关闭导致的问题，它在各种不同的数据库的错误信息之上，提供了一个统一的异常访问层。它还利用 Spring 的 AOP 模块给 Spring 应用中的对象提供事务管理服务。

46. 解释对象/关系映射集成模块。

Spring 通过提供 ORM 模块，支持我们在直接 JDBC 之上使用一个对象/关系映射映射(ORM)工具，Spring 支持集成主流的 ORM 框架，如 Hiberate,JDO 和 iBATISSQLMaps。Spring 的事务管理同样支持以上所有 ORM 框架及 JDBC。

47. Spring 支持的 ORM 框架有哪些？

Spring 支持以下 ORM： Hibernate、iBatis、JPA(JavaPersistenceAPI)、TopLink、JDO(JavaDataObjects)、OJB

48. 请描述一下 Spring 的事务

声明式事务管理的定义：用在 Spring 配置文件中声明式的处理事务来代替代码式的处理事务。这样的好 处是，事务管理不侵入开发的组件，具体来说，业务逻辑对象就不会意识到正在事务管理之中，事实上也应该 如此，因为事务管理是属于系统层面的服务，而不是业务逻辑的一部分，如果想要改变事务管理策划的话，也 只需要在定义文件中重新配置即可，这样维护起来极其方便。

基于 TransactionInterceptor 的声明式事务管理：两个次要的属性： transactionManager，用来 指定一个事务治 理器，并将具体事务相 关的操作请托给它；其 他一个是 Properties 类型的 transactionAttributes 属性，该属性的每一个键值对中，键指定的是方法名，方法名可以行使通配符，而值就是表现呼应方法的所运用的事务属性。

```
<bean id="transactionInterceptor"

    class="org.springframework.transaction.interceptor.TransactionIntercept
or">
    <property name="transactionManager" ref="transactionManager" />
    <property name="transactionAttributes">
        <props>
            <prop key="transfer">PROPAGATION_REQUIRED</prop>
        </props>
    </property>
</bean>
<bean id="bankServiceTarget"
class="footmark.spring.core.tx.declare.origin.BankServiceImpl">
    <property name="bankDao" ref="bankDao" />
</bean>
<bean id="bankService"
class="org.springframework.aop.framework.ProxyFactoryBean">
    <property name="target" ref="bankServiceTarget" />
    <property name="interceptorNames">
        <list>
            <idref bean="transactionInterceptor" />
        </list>
    </property>
</bean>
```

基于 TransactionProxyFactoryBean 的声明式事务管理：设置配置文件与先前比照简化了许多。我们把这类设置配置文件格式称为 Spring 经典的声明式事务治理

```
<bean id="bankServiceTarget"
```

```
            class="footmark.spring.core.tx.declare.classic.BankServiceImpl">
    <property name="bankDao" ref="bankDao" />
</bean>
<bean id="bankService"
    class="org.springframework.transaction.interceptor.TransactionProxyFact
oryBean>
    <property name="target" ref="bankServiceTarget" />
    <property name="transactionManager" ref="transactionManager" />
    <property name="transactionAttributes">
        <props>
            <prop key="transfer">PROPAGATION_REQUIRED</prop>
        </props>
    </property>
</bean>
```

基于 <tx> 命名空间的声明式事务治理：在前两种方法的基础上，Spring 2.x 引入了 <tx> 命名空间，连络行使 <aop> 命名空间，带给开发人员设置配备声明式事务的全新体验。

```
<bean id="bankService"
class="footmark.spring.core.tx.declare.namespace.BankServiceImpl">
    <property name="bankDao" ref="bankDao" />
</bean>
<tx:advice id="bankAdvice" transaction-manager="transactionManager">
    <tx:attributes>
        <tx:method name="transfer" propagation="REQUIRED" />
    </tx:attributes>
</tx:advice>
<aop:config>
    <aop:pointcut id="bankPointcut" expression="execution(* *.transfer(..))"
/>
    <aop:advisor advice-ref="bankAdvice" pointcut-ref="bankPointcut" />
</aop:config>
```

基于 @Transactional 的声明式事务管理：Spring 2.x 还引入了基于 Annotation 的体式格式，具体次要触及@Transactional 标注。@Transactional 可以浸染于接口、接口方法、类和类方法上。算作用于类上时，该类的一切 public 方法将都具有该类型的事务属性。

```
    @Transactional(propagation = Propagation.REQUIRED)
    public boolean transfer(Long fromId， Long toId， double amount) {
        return bankDao.transfer(fromId， toId， amount);
    }
```

编程式事物管理的定义：在代码中显式挪用 beginTransaction()、commit()、rollback()等事务治理相关的方法，

这就是编程式事务管理。Spring 对事物的编程式管理有基于底层 API 的编程式管理和基于 TransactionTemplate 的

编程式事务管理两种方式。

基于底层 API 的编程式管理：凭证 PlatformTransactionManager 、TransactionDefinition 和

TransactionStatus 三个焦点接口，来实现编程式事务管理。

```
public class BankServiceImpl implements BancService {
    private BanckDao bankDao;
    private TransactionDefinition txDefinition;
    private PlatformTransactionManager txManager;

    public boolean transfer(Long fromId,Long toId,double amount) {
        TransactionStatus txStatus = txManager.getTransaction(txDefinition);
        boolean result = false;
        try {
            result = bankDao.transfer(fromId， toId， amount);
            txManager.commit(txStatus);
        } catch (Exception e) {
            result = false;
            txManager.rollback(txStatus);
            System.out.println("Transfer Error!");
        }
        return result;
    }
}
```

基于 TransactionTemplate 的编程式事务管理:为了不损坏代码原有的条理性，避免出现每一个方法中都包括相同的启动事物、提交、回滚事物样板代码的现象，spring 提供了 transactionTemplate 模板来实现编程式事务管理。

```
public class BankServiceImpl implements BankService {
    private BankDao bankDao;
    private TransactionTemplate transactionTemplate;
    public boolean transfer(final Long fromId,final Long toId,final double
amount) {
        return (Boolean) transactionTemplate.execute(new
TransactionCallback(){
            public Object doInTransaction(TransactionStatus status){
                Object result;
                try {
                    result = bankDao.transfer(fromId， toId， amount);
                } catch (Exception e) {
                    status.setRollbackOnly();
                    result = false;
                    System.out.println("Transfer Error!");
                }
                return result;
                }
            });
    }
}
```

编程式事务与声明式事务的区别：
1）编程式事务是自己写事务处理的类，然后调用
2）声明式事务是在配置文件中配置，一般搭配在框架里面使用！

49. Spring 事务隔离级别

Default：使用数据库本身的隔离级别 ORACLE（读已提交）Mysql（可重复读）；
Read_Uncomited（脏读）：读取过期的数据，就是一个事物读到另一个事务未提交的新数据，最低隔离级别，一切皆有可能；
Read_Commited（幻读）：读取临时的数据，就是一个事物在进行修改全表的时候，另一个事务对数据进行了新增，从而第一个事务的执行完后发现还有没有修改的数据，就好像发生了幻觉一样；
RepeaTable_Read（不可重复读）：就是在同一个事务中先后执行两条一样的 select 语句，之间没有执行过 Del 语句但先后结果不一样，这就是不可重复读；；
Serializable：串行化，最高隔离界别，杜绝一切隐患，但效率较低；

50. Spring 怎么设置隔离级别？

用@Transactional 注解声明式事务的事务管理中来设置 isolation 属性的隔离级别
在配置文件中设置事务<tx:method>元素

51. 使用 Spring 通过什么方式访问 Hibernate?

在 Spring 中有两种方式访问 Hibernate：1）控制反转 HibernateTemplate 和 Callback。2）继承 HibernateDAOSupport 提供一个 AOP 拦截器。

52. 解释 SpringJDBC、SpringDAO 和 SpringORM

Spring-DAO 并非 Spring 的一个模块，它实际上是指示你写 DAO 操作、写好 DAO 操作的一些规范。因此，对于访问你的数据它既没有提供接口也没有提供实现更没有提供模板。在写一个 DAO 的时候，你应该使用@Repository 对其进行注解，这样底层技术(JDBC，Hibernate，JPA，等等)的相关异常才能一致性地翻译为相应的 DataAccessException 子类。Spring-JDBC 提供了 Jdbc 模板类，它移除了连接代码以帮你专注于 SQL 查询和相关参数。Spring-JDBC 还提供了一个 JdbcDaoSupport，这样你可以对你的 DAO 进行扩展开发。它主要定义了两个属性：一个 DataSource 和一个 JdbcTemplate，它们都可以用来实现 DAO 方法。JdbcDaoSupport 还提供了一个将 SQL 异常转换为 SpringDataAccessExceptions 的异常翻译器。Spring-ORM 是一个囊括了很多持久层技术(JPA，JDO，Hibernate，iBatis)的总括模块。对于这些技术中的每一个，Spring 都提供了集成类，这样每一种技术都能够在遵循 Spring 的配置原则下进行使用，并平稳地和 Spring 事务管理进行集成。对于每一种技术，配置主要在于将一个 DataSourcebean 注入到某种 SessionFactory 或者 EntityManagerFactory 等 bean 中。纯 JDBC 不需要这样的一个集成类(JdbcTemplate 除外)，因为 JDBC 仅依赖于一个 DataSource。如果你计划使用一种 ORM 技术，比如 JPA 或者 Hibernate，那么你就不需要 Spring-JDBC 模块了，你需要的是这个 Spring-ORM 模块。

53. 在 Spring 框架中如何更有效地使用 JDBC?

使用 SpringJDBC 框架，资源管理和错误处理的代价都会被减轻。所以开发者只需写 statements 和 queries 从数据存取数据，JDBC 也可以在 Spring 框架提供的模板类的帮助下更有效地被使用，这个模板叫 JdbcTemplate。JdbcTemplate 类提供了很多便利的方法解决诸如把数据库数据转变成基本数据类型或对象，执行写好的或可调用的数据库操作语句，提供自定义的数据错误处理。

54. 解释 WEB 模块

Spring 的 WEB 模块式构建在 applicationcontext 模块基础之上，提供一个适合 web 应用的上下文。这个模块也包括支持多种面向 web 的任务，如透明地处理多个文件上传请求和程序级求参数的绑定到你的业务对象。它也有对 JakartaStruts 的支持。

55. 一个 Spring 的应用看起来象什么？

一个定义了一些功能的接口。
这实现包括属性，它的 Setter，getter 方法和函数等。
SpringAOP。
Spring 的 XML 配置文件。
使用以上功能的客户端程序。
也可以自己做更形象的描述。

十二、SpringMVC

1. Spring MVC

是一个基于 MVC 架构的用来简化 web 应用程序开发框架，它是 Spring 的一个模块，无需中间整合层来整合，它和 Struts2 一样都属于表现层的框架，在 web 模型中， MVC 是一种很流行的框架，把 Model，View，Controller 分离，把较为复杂的 web 应用分为逻辑清晰的几部分，简化开发，减少出错，方便组内开发人员之间的配合。

2. SpringMVC 的流程

用户发送请求至前端控制器 DispatcherServlet。 2. DispatcherServlet 收到请求后，调用 HandlerMapping 处理器映射器，请求获取 Handle。 3. 处理器映射器根据请求 url 找到具体的处理器，生成处理器对象及处理器拦截器(如果则生成 并返回给 DispatcherServlet)。 4. 执行处理器(Handler，也叫后端控制器)。 5. Handler 执行完成返回 ModelAndView。 6. HandlerAdapter 将 Handler 执行结果 ModelAndView 返回给 DispatcherServlet。 7. DispatcherServlet 将 ModelAndView 传给 ViewResolver 视图解析器进行解析。8. ViewResolver 解析后返回具体 View。 9. DispatcherServlet 对 View 进行渲染视图(即将模型数据填充至视图中)。 10. DispatcherServlet 响应用户。

3. SpringMVC 的工作原理

a.用户向服务器发送请求，请求被 springMVC 前端控制器 DispatchServlet 捕获；
b.DispatcherServle 对请求 URL 进行解析，得到请求资源标识符（URL），然后根据该 URL 调用 HandlerMapping
将请求映射到处理器 HandlerExcutionChain；
c.DispatchServlet 根据获得 Handler 选择一个合适的 HandlerAdapter 适配器处理；
d.Handler 对数据处理完成以后将返回一个 ModelAndView（）对象给 DisPatchServlet;
e.Handler 返回的 ModelAndView()只是一个逻辑视图并不是一个正式的视图，DispatcherSevlet 通过
ViewResolver 试图解析器将逻辑视图转化为真正的视图 View;
h.DispatcherServle 通过 model 解析出 ModelAndView()中的参数进行解析最终展现出完整的 view 并返回给
客户端;

4. SpringMVC 的优点

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。！它是易于扩展的。

5. SpringMVC 的主要组键

前端控制器 DispatcherServlet，作用：接受请求、响应结果相当于转发器，有了 DispatcherServlet 就减少了其他组件之间的耦合度。
处理器映射器 HandlerMapping，作用：根据请求的 URL 来查找 Handler。
处理器适配器 HandlerAdapter，注意：在编写 Handler 的时候要按照 HandlerAdapter 要求的规则去编写，这样适配器 HandlerAdapter 才可以正确的去执行 Handler。
处理器 Handler(需要程序员开发)。
视图解析器 ViewResolver。
视图 View(需要程序员开发)。

6. SpringMVC 和 Struts2 的区别有哪些？

SpringMVC 的入口是一个 servle 及前端控制器(DispatcherServlet)，而 Struts2 入口是一个 filter 过滤器(StrutsPrepareAndExecuteFilter)。

SpringMVC 是基于方法开发（一个 url 对应一个方法），请求参数传递到方法的形参，可以设计为单例或多例，Struts2 是基于类开发，传递参数通过类的属性，只能设计为多例。

Struts2 采用值栈存储请求和响应的数据，通过 OGNL 存取数据，SpringMVC 通过参数解析器是将 request 请求内容解析，并给方法形参赋值，将数据和视图封装成 ModelAndView 对

象，最后又将 ModelAndView 中的模型数据通过 request 域传输到页面。

7. SpringMVC 如何设定重定向和转发的？

- ✓ 在返回值前面 forward,就可以让结果转发，譬如 forward:user.do?name=method4
- ✓ 在返回值前面 redirect，就可以让返回值重定向，譬如 redirect：http://www.baidu.com

8. SpringMVC 里面拦截器如何写？

有两种写法，一种是实现 HandlerInterceptor 接口，另一种是继承适配器类，接着在接口方法 当中实现处理逻辑，然后在 SpringMVC 的配置文件中配置拦截器即可:

```
<!--配置 SpringMVC 的拦截器-->
<mvc:interceptors>
<!--配置一个拦截器的 Bean 就可以了，默认是对所有请求都拦截-->
<bean id=”myInterceptor” class=”com.et.action.myHandlerInterceptor”> </bean>
<!--只针对部分请求拦截-->
<mvc:interceptor>
 <mvc:mapping path=”/modelMap.do”/>
 <bean class=”com.et.action.MyHandlerInterceptorAdapter”/>
</mvc:interceptor>
 </mvc:interceptors>
```

9. SpringMVC 的异常处理

可以将异常抛给 Spring 框架，由 Spring 的 AOP 来处理，我们只需要配置简单的异常处理器在异常处理器中添加视图页面即可。

10. SpringMVC 的核心入口类是什么？Struts1,Struts2 的分别是什么？

SpringMVC 的是 DispatcherServlet ， Struts1 的是 ActionServlet ， Struts2 的是 StrutsPrepareAndExecuteFilter。

11. SpringMVC 的控制器是不是单例模式，如果是，有什么问题，如何解决。

SpringMVC 的控制器是单例模式，所以在多线程访问的时候有线程安全问题，不要用同步， 会影响性能的，解决方案是在控制器里面不能写字段。

12. SpringMVC 的控制器的注解一般用那个，有没有别的注解可以替代？

一般用@Controller,表示表现层，不能用别的注解替代。

13. SpringMVC 的@RequestMapping 注解用在类上面有什么作用?

是一个用来处理请求地址映射的注解，可以用于类或方法上，表示类中的所有响应请求的方 法都是以该路径作为父路径。

14. SpringMVC 如何把某个请求映射到特定的方法上面?

直接在方法上面加上注解@RequestMapping,并且在这个注解里面写上要拦截的路径。

15. SpringMVC 如果想在拦截的方法里面得到从前台传入的参数，如何得到？

直接在方法中声明这个对象，SpringMvc 就自动会把属性赋值到这个对象里面。

16. SpringMVC 中的函数的返回值是什么？

返回值可以有很多类型，有 String，ModelAndview，Model

17. SpringMVC 用什么对象从后台向前台传递数据的？

通过 ModelMap 对象，可以在这个对象里面用 put 方法，把对象加到里面，前台就可以通过 el 表达式拿到。

18. SpringMVC 中有个类把视图和数据合并在一起，叫什么？

ModelAndview。

19. SpringMVC 中怎么把 ModelMap 里面的数据放入 Session 里面?

可以在类上面加上@SessionAttributes 注解，里面包含的字符串就是要放入 Session 里面的 key。
当一个方法向 AJAX 返回特殊对象，譬如 Object，List 等，需要做什么处理？

20. SpringMVC 如何在方法里面得到 Request 或者 Session？

直接在方法的形参中声明 Request，SpringMvc 就自动把 request 对象传入。获取 Session，也 是同样的方法，但是需要在方法中获取 request 中的 Session，例如：Session session=request.getSession();即可，获取 Response 也是需要在方法的形参中声明 Response。

21. SpringMVC 常用注解都有哪些？

@requestMapping 用于请求 url 映射。
@RequestBody 注解实现接收 http 请求的 json 数据，将 json 数据转换为 java 对象。

@ResponseBody 注解实现将 controller 方法返回对象转化为 json 响应给客户。

22. 如何开启注解处理器和适配器？

我们在项目中一般会在 springmvc.xml 中通过开启<mvc：annotation-driven>来实现注解处理器和适配器的开启。

23. SpringMvc 怎么和 AJAX 相互调用的？

通过 Jackson 框架就可以把 Java 里面的对象直接转化成 Js 可以识别的 Json 对象。具体步骤如下 ：
（1）加入 Jackson.jar
（2）在配置文件中配置 json 的映射
（3）在接受 Ajax 方法里面可以直接返回 Object,List 等,但方法前面要加上@ResponseBody 注解。

24. 如何解决 POST 请求中文乱码问题，GET 的又如何处理呢？

（1）解决 post 请求乱码问题：
在 web.xml 中加入：

```
<filter>
  <filter-name>CharacterEncodingFilter</filter-name>
  <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
  <init-param>
    <param-name>encoding</param-name>
    <param-value>utf-8</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>CharacterEncodingFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

（2）get 请求中文参数出现乱码解决方法有两个：
①修改 tomcat 配置文件添加编码与工程编码一致，如下：

```
<ConnectorURIEncoding="utf-8" connectionTimeout="20000" port="8080" protocol="HTTP/1.1" redi
rectPort="8443"/>
```

②另外一种方法对参数进行重新编码：

```
String userName = new String(request.getParamter("userName").getBytes("ISO8859-1"),"utf-8")
```

ISO8859-1 是 tomcat 默认编码，需要将 tomcat 编码后的内容按 utf-8 编码。

25. 如果在拦截请求中,我想拦截 get 方式提交的方法,怎么配置？

答：可以在@RequestMapping 注解里面加上 method=RequestMethod.GET。

26. 如果前台有很多个参数传入,并且这些参数都是一个对象的,那么怎么样快速得到这个对象？

答：直接在方法中声明这个对象,SpringMvc 就自动会把属性赋值到这个对象里面。

27. 当一个方法向 AJAX 返回特殊对象,譬如 Object,List 等,需要做什么处理？

答：要加上@ResponseBody 注解。

十三、Springboot

1. SpringBoot

Spring Boot 是 Spring 开源组织下的子项目，是 Spring 组件一站式解决方案，它的产生简化了框架的使用，所谓简化，是指简化了 使用 Spring 的难度，简省了繁重的配置，提供了各种启动器，开发者能快速上手，所以 SpringBoot 是一个服务于框架的框架，服务范围是简化配置文件。
Spring Boot 优点，如：
（1）独立运行（2）简化配置（3）自动配置（4）无代码生成和 XML 配置（5）应用监控（6）上手容易

2. SpringBoot 工程的使用特点

一个简单的 SpringBoot 工程是不需要在 pom.xml 手动添加什么配置的，如果与其他技术则需要在 pom.xml 中添加依赖，由程序自动加载依赖 jar 等配置文件。我们之前在利用 SS 或者 SSH 开发的时候，在 resources 中储存各种对应框架的配置文件，而现在我们只需要一个配置文件即可，配置内容大体有服务器端口号、数据库连接的地址、用户名、密码，虽然简单但在一定问题上而言，这也是极不安全的，将所有配置，放在一个文件里，是很危险的，但对于一般项目而言并不会有太大影响。在 SpringBoot 创 建 时 会 自 动 创 建 BootdemolApplication 启动类，代表着本工程项目合服务器的启动加载，在 SpringBoot 中是内含服务器，所以不用手动配置 Tomcat，但注意端口号冲突问题。

3. SpringBoot 2.x 有什么新特性？与 1.x 有什么区别？

1）配置变更。
2）JDK 版本升级。
3）第三方类库升级。
4）响应式 Spring 编程支持。
5）HTTP/2 支持。
6）配置属性绑定。
7）更多改进加强。

4. SpringBoot 默认启动方式是什么？还有什么启动方式？

1. 运行带有 mian 方法类。
类上需要加@SpringBootApplication 注解，main 方法中使用 SpringApplication.run(类名.class，args);自动加载 application.properties 文件。
2. 通过命令行 java -jar 的方式。
java -jar jar_path --param
jar_path：指将项目打包为 jar 包之后的存储路径。
--param：为需要在命令行指定的参数。例如： java -jar emample.jar --server.port=8081 该命令通过启动行指定了项目启动后绑定的端口号，因为该命令行参数将会覆盖 application.properties 中的端口配置。
3. 通过 spring-boot-plugin 的方式。如果需要正常使用该 maven 插件，需要我们在 maven 项目中增 spring-boot-maven-plugin 插件配置。

```
<plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
```

准备工作做好之后，我们进入项目的根目录，执行 mvn spring-boot：run。 通过该命令启动项目，可以为其制定执行参数，例如： 如果有多个参数需要指定，以逗号进行分割。具体用法通过 Drun.arguments 来定： mvn spring-boot：run -Drun.arguments=”--server.port=8080”

5. SpringBoot 的核心配置文件有几个？它们的区别是什么？

SpringBoot 的核心配置文件是 application 和 boostrap 配置文件。Application 配置文件这个容易理解，主要用于 Spring boot 项目的自动化配置。
Bootstrap 配置文件有以下几个应用场景。
（1）使用 Spring Cloud Config 配置中心时，这时需要在 bootstrap 配置文件中添加连接到配置中心的配置属性来加载外部配置中心的配置信息；
（2）一些固定的不能被覆盖的属性；
（3）一些加密/解密的场景；

6. Bootstrap 和 application 的区别？

Bootstrap>application>其他。

7. SpringBoot 的配置文件有哪几种格式？它们有什么区别？

1．properties 和 yml，它们的区别主要是书写格式不同。
2．Properties： app.user.name= javastack

```
Yml：
    app：
      user：name： javastack
```

3．yml 格式不支持@PropertySource 注解导入配置。

4．properties 和 yml 中包含相同属性时，properties 文件优先级高于 yml 文件。

8. 什么是 YAML？

YAML 是一种人类可读的数据序列化语言。它通常用于配置文件。与属性文件相比，如果我们想要在配置文件中添加复杂的属性，YAML 文件就更加结构化，而且更少混淆。可以看出 YAML 具有分层配置数据。

9. 如何在自定义端口上运行 Spring Boot 应用程序？

为了在自定义端口上运行 Spring Boot 应用程序，您可以在 application.properties 中指定端口。
server.port = 8090

10. SpringBoot 的核心注解是哪个？它主要由哪几个注解组成的？

启动类上面注解是@SpringBootApplication，它也是 SpringBoot 的核心注解，主要包含 了以下 3 个注解： 包 括 @ComponentScan ， @SpringBootConfiguration，@EnableAutoConfiguration。
@EnableAutoConfiguration 的作用启动自动的配置， @EnableAutoConfiguration 注解就是 SpringBoot 根据你添加的 jar 包来配置你项目的默认配置，比如根据 spring-boot-starter-web， 来判断你项目是否添加了 webmvc 和 tomcat，就会自动帮你配置 web 项目中所需要的默 配置。
@ComponentScan 扫描当前包及其子包下被 @Component ，@Controller ，@Service ， @Repository 注解标记的类并纳入 spring 容器中进行管理。
@SpringBootConfiguration 继承自@Configuration，二者功能也一直，标注当前类是配置类， 并会将当前类内声明的一个或多个以@Bean 注解标记的方法的实例纳入到 spring 容器中 并且实例名就是方法名。

11. SpringBoot 有哪几种读取配置的方式？

SpringBoot 可以通过@PropertySource，@Value，@Environment，@ConfigurationProperties 来绑定变量。

12. 开启 SpringBoot 特性有哪几种方式？

1. 继承 spring-boot-starter-parent 项目。
2. 导入 spring-boot-dependencies 项目依赖。

13. SpringBoot 需要独立的容器运行吗？

可以不需要，内置了 Tomcat/Jetty 等容器。

14. 运行 SpringBoot 有哪几种方式？

打包用命令或者放到容器中运行。
用 Maven/Gradle 插件运行。
直接执行 main 方法运行。

15. SpringBoot 自动配置原理是什么？

注解@EnableAutoConfiguraction，@Configuration，@ConditionalOnClass 就是自动配置 的核心，首先它得是一个配置文件，其次根据类路径下是否有这个类取自动配置。

16. 你如何理解 SpringBoot 中的 Starters？

Starters 可以理解为启动器，它包含了一系列可以集成到应用里面的依赖包，可以一站 式集成 Spring 及其他技术，而不需要到处找示例代码和依赖包。如想使用 Spring JPA 访问 数据库，只要加入 spring-boot-starter-data-jpa 启动器依赖就能使用了。

17. 如何在 SpringBoot 启动的时候运行一些特定的代码？

可以实现接口 ApplicationRunner 或者 CommandLineRunner，这两个接口实现方式一样， 它们都只提供了一个 run 方法，。

18. SpringBoot 支持哪些日志框架？推荐和默认的日志框架是哪个？

SpringBoot 支持 Java Util Logging，Log4j2，Logback 作为日志框架，如果你使用 Starters 启动器，SpringBoot 将使用 Logback 作为默认框架。

19. SpringBoot 实现热部署有哪几种方式？

主要有两种方式： Spring Loaded Spring-boot-devtools

20. 如何重新加载 Spring Boot 上的更改，而无需重新启动服务器？

这可以使用 DEV 工具来实现。通过这种依赖关系，可以节省任何更改，嵌入式 tomcat 将重新启动。Spring Boot 有一个开发工具（DevTools）模块，它有助于提高开发人员的生产力。Java 开发人员面临的一个主要挑战是将文件更改自动部署到服务器并自动重启服务器。开发人员可以重新加载 Spring Boot 上的更改，而无需重新启动服务器。这将消除每次手动部署更改的需要。Spring Boot 在发布它的第一个版本时没有这个功能。这是开发人员最需要的功能。DevTools 模块完全满足开发人员的需求。该模块将在生产环境中被禁用。它还提供 H2 数据库控制台以更好地测试应用程序。

21. 你如何理解 SpringBoot 配置加载顺序？

在 SpringBoot 里面，可以使用以下几种方式来加载配置。
Properties 文件。
Yaml 文件。
系统环境变量。
命令行参数。

22. SpringBoot 项目 jar 包打成 war 包需要什么？

1. 去掉 pom.xml 的内置 tomcat
2. 在 pom.xml 中配置启动类，使用 spring-boot-maven-plugin 插件。
3. 修改打包方式为<packaging>war</packaging>方式。
4. 修改启动类，继承 SpringBootServletInitializer 类，然后重写里面的 configure 方法，设定为启动类。
5. 打包测试，通过命令 mvn clean package 打包。

23. SpringBoot 怎么定义不同环境配置？

在 SpringBoot 中多环境配置文件名需要满足 application-{profile}.properties 的格式，其中{profile}对应环境标识，比如： application-dev.properties：开发环境。
application-test.properties：测试环境。
application-prod.properties：生产环境。
至于那些具体的配置文件会被加载，需要在 application.properties 文件中通过 spring.profiles.active 属性来设置，其中对应{profile0 值}。如：spring.profiles.active=test

24. springboot 中常用的 starter 的组件有哪些.

spring-boot-starter-parent //boot 项目继承的父项目模块.
spring-boot-starter-web //boot 项目集成 web 开发模块.
spring-boot-starter-tomcat //boot 项目集成 tomcat 内嵌服务器.
spring-boot-starter-test //boot 项目集成测试模块.
mybatis-spring-boot-starter //boot 项目集成 mybatis 框架.
spring-boot-starter-jdbc //boot 项目底层集成 jdbc 实现数据库操作支持.
其他诸多组件，可到 maven 中搜索，或第三方 starter 组件到 github 上查询

25. Spring Boot 中的监视器是什么？

Spring boot actuator 是 spring 启动框架中的重要功能之一。Spring boot 监视器可帮助您访问生产环境中正在运行的应用程序的当前状态。有几个指标必须在生产环境中进行检查和监控。即使一些外部应用程序可能正在使用这些服务来向相关人员触发警报消息。监视器模块公开了一组可直接作为 HTTP URL 访问的 REST 端点来检查状态。

26. 如何在 Spring Boot 中禁用 Actuator 端点安全性？

默认情况下，所有敏感的 HTTP 端点都是安全的，只有具有 ACTUATOR 角色的用户才能访问它们。安全性是使用标准的 HttpServletRequest.isUserInRole 方法实施的。 我们可以使用 management.security.enabled = false 来禁用安全性。只有在执行机构端点在防火墙后访问时，才建议禁用安全性。

27. 如何实现 Spring Boot 应用程序的安全性？

为了实现 Spring Boot 的安全性，我们使用 spring-boot-starter-security 依赖项，并且必须添加安全配置。它只需要很少的代码。配置类将必须扩展 WebSecurityConfigurerAdapter 并覆盖其方法。

28. 如何集成 Spring Boot 和 ActiveMQ？

对于集成 Spring Boot 和 ActiveMQ，我们使用 spring-boot-starter-activemq 依赖关系。 它只需要很少的配置，并且不需要样板代码。

29. 如何使用 Spring Boot 实现分页和排序？

使用 Spring Boot 实现分页非常简单。使用 Spring Data-JPA 可以实现将可分页的 org.springframework.data.domain.Pageable 传递给存储库方法。

30. 什么是 Swagger？你用 Spring Boot 实现了它吗？

Swagger 广泛用于可视化 API，使用 Swagger UI 为前端开发人员提供在线沙箱。Swagger 是用于生成 RESTful Web 服务的可视化表示的工具，规范和完整框架实现。它使文档能够以与服务器相同的速度更新。当通过 Swagger 正确定义时，消费者可以使用最少量的实现逻辑来理解远程服务并与其进行交互。因此，Swagger 消除了调用服务时的猜测。

31. springboot 与 spring 的区别.

java 在集成 spring 等框架需要作出大量的配置，开发效率低，繁琐.所以官方提出 springboot 的核心思想：习惯优于配置.可以快速创建开发基于 spring 框架的项目.或者支持可以不用或很少的 spring 配置即可。

32. springboot 项目需要兼容老项目（spring 框架），该如何实现.

集成老项目 spring 框架的容器配置文件即可：spring-boot 一般提倡零配置，但是如果需要配置，也可增加：@ImportResource({"classpath：spring1.xml" ， "classpath：spring2.xml"})注意：resources/spring1.xml 位置.

十四、SpringCloud

1. SpringCloud

SpringCloud 是一个微服务框架，相比 Dubbo 等 RPC 框架，SpringCloud 提供的全套的分布式系统解决方案。SpringCloud 对微服务基础框架 Netflix 的多个开源组件进行了封装， 同时又实现了和云端平台以及和 SpringBoot 开发框架的集成。SpringCloud 为微服务架构开 发涉及的配置管理，服务管理，熔断机制，智能路由，微代理，控制总线，一次性 token， 全局一致性锁，leader 选举，分布式 session，集群状态管理等操作提供了一种简单的开发 式。SpringCloud 为开发者提供了快速构建分布式系统的工具，开发者可以快速的启动服务 器或构建应用，同时能够快速和云平台资源进行对接。

2. 什么是微服务？

微服务是一种架构风格，一个大型复杂软件应用由一个或多个微服务组成。系统中的各个微服务可被独立部署，各个微服务之间是松耦合的。每个微服务仅关注于完成一件任务并很好地完成该任务。在所有情况下，每个任务代表着一个小的业务能力。
可以在“自己的程序”中运行，并通过“轻量级设备与 HTTP 型 API 进行沟通”。关键在于该服务可以在自己的程序中运行。通过这一点我们就可以将服务公开与微服务架构（在现有系统中分布一个 API）区分开来。在服务公开中，许多服务都可以被内部独立进程所限制。如果其中任何一个服务需要增加某种功能，那么就必须缩小进程范围。在微服务架构中，只需要在特定的某种服务中增加所需功能，而不影响整体进程。

3. 使用 Spring Cloud 有什么优势？

Spring Cloud 来源于 Spring，质量、稳定性、持续性都可以得到保证。
Spirng Cloud 天然支持 Spring Boot，更加便于业务落地。
Spring Cloud 发展非常的快，从 2016 年开始接触的时候相关组件版本为 1.x，到现在将要发布 2.x 系列。
Spring Cloud 是 Java 领域最适合做微服务的框架。
相比于其它框架，Spring Cloud 对微服务周边环境的支持力度最大。
对于中小企业来讲，使用门槛较低。
Spring Cloud 是微服务架构的最佳落地方案。

4. SpringCloud 如何实现服务的注册和发现

服务在发布时指定对应的服务名(服务名包括了 IP 地址和端口)将服务注册到注册中心 (eureka 或者 zookeeper)。 这一过程是 SpringCloud 自动实现只需要在 main 方法添加 EnableDisscoveryClient 同一 服务修改端口就可以启动多个实例。
调用方法：传递服务名称通过注册中心获取所有的可用实例，通过负载均衡策略调用 (ribbon 和 feign)对应的服务。

5. Ribbon 和 Feign 的区别

Ribbon 添加 Maven 依赖 spring-starter-ribbon 使用@RibbonClient(value=”服务名称”)使 RestTemplate 调用远程服务对应的方法。
Feign 添加 Maven 依赖 spring-starter-feign 服务提供方提供对外接口，调用方接口在接 口上使用@FeignClient(“指定服务名”)。
Ribbon 和 Feign 都是用于调用其他服务的，不过方法不同。
1. 启动类使用的注解不同， Ribbon 用的是 @RibbonClient ， Feign 用的 @EnableFeignClients。
2. 服务的指定位置不同，Ribbon 是在@RibbonClient 注解上声明，Feign 则是在定义 抽象方法的接口中使用@FeignClient 声明。
3. 调用方法不同，Ribbon 需要自己构建 HTTP 请求，模拟 HTTP 请求然后使用 RestTemplate 发送给其他服务，步骤相当繁琐。Feign 则是在 Ribbon 的基础上进行了一次 进，采用接口的方式，将需要调用的其他服务的方法定义成抽象方法即可。不需要自己构建 HTTP 请求。不过抽象方法的注解、方法签名要和提供服务的方法完全一致。

6. Spring Cloud 的特性

分布式/版本化配置。
服务注册和发现。
路由。
服务和服务之间的调用。
负载均衡。
断路器。
分布式消息传递。

7. 什么是 Spring Cloud Eureka?

Spring Cloud Eureka 是基于 Spring Cloud Netflix 微服务套件的一部分，它基于 Eureak 做了二次封装，主要负责完成微服务构架中的服务治理功能。Spring Cloud 为 Eureka 增加了 Spring Boot 风格的自动化配置，我们只需通过简单引入依赖和注解配置就能让 Spring Bootg 构建的微服务应用轻松地与 Eureka 服务治理体系进行整合。服务治理是微服务构架中最为核心和基础的模块，它主要用来实现各个微服务实例的自动化注册与发现。

8. 什么是负载均衡？

负载均衡分为服务端负载均衡和客户端负载均衡。
服务端负载均衡：当浏览器向后台发出请求的时候，会首先向反向代理服务器发送请求，反向代理服务器会根据客户端部署的 ip：port 映射表以及负载均衡策略，来决定向哪台服务器发送请求，一般会使用到 nginx 反向代理技术。
客户端负载均衡：当浏览器向后台发出请求的时候，客户端会向服务注册器(例如：Eureka Server)，拉取注册到服务器的可用服务信息，然后根据负载均衡策略，直接命中哪台服务器发送请求。这整个过程都是在客户端完成的，并不需要反向代理服务器的参与。

服务端负载均衡：分为两种，一种是硬件负载均衡，还有一种是软件负载均衡。
我们主要讲客户端负载均衡，Spring cloud Ribbon 是一个基于 Http 和 TCP 的客服端负载均衡工具，它是基于 Netflix Ribbon 实现。Ribbon 不需要独立部署，但它几乎存在于每个微服务的基础设施中。Ribbon 可以通过在客户端中配置 ribbonServerList 来设置服务端列表去轮询访问以达到均衡负载的作用。
当 Ribbon 与 Eureka 联合使用时，ribbonServerList 会被 DiscoveryEnabledNIWSServerList 重写，扩展成从 Eureka 注册中心中获取服务实例列表。同时它也会用 NIWSDiscoveryPing 来取代 IPing，它将职责委托给 Eureka 来确定服务端是否已经启动。Spring Cloud Feign 默认集成了 Ribbon，并和 Eureka 结合，默认实现了负载均衡的效果,也是客户端使用。

9. 什么是服务容错保护？什么是 Spring Cloud Hystrix?

Spring Cloud Hystrix 是服务容错保护，也是服务熔断器。Hystrix 是 Spring Cloud 提供的一种带有熔断机制的框架，由于在微服务系统中同一个操作会由多个不同的微服务来共同完成，所以微服务与微服务之间会由很多相互的调用，由于在分布式环境中经常会出现某个微服务节点故障的情况，所以会由调用失败发生，而熔断器的作用就是当出现远程调用失败的时候提供一种机制来保证程序的正常运行而不会卡死在某一次调用，类似 Java 程序中的 try-catch 结构，而只有当异常发生的时候才会进入 catch 的代码块。

10. 什么是声明式服务调用？

Spring Cloud Feign 是声明式服务调用。Feign 是一个声明式的 Web Service 客户端，它的目的就是让 Web Service 调用更加简单。Feign 提供了 HTTP 请求的模板，通过编写简单的接口和插入注解，就可以定义好 HTTP 请求的参数、格式、地址等信息。而 Feign 则会完全代理 HTTP 请求，我们只需要像调用方法一样调用它就可以完成服务请求及相关处理。

11. 什么是 api 服务网关？

API 网关是一个服务器，是系统的唯一入口。从面向对象设计的角度看，它与外观模式类似。API 网关封装了系统内部架构，为每个客户端提供一个定制的 API。它可能还具有其它职责，如身份验证、监控、负载均衡、缓存、请求分片与管理、静态响应处理。
API 网关方式的核心要点是，所有的客户端和消费端都通过统一的网关接入微服务，在网关层处理所有的非业务功能。通常，网关也是提供 REST/HTTP 的访问 API。服务端通过 API-GW 注册和管理服务。Spring Cloud Zuul 是 API 网关，Zuul 是 Netflix 开源的微服务网关，他可以和 Eureka,Ribbon,Hystrix 等组件配合使用，Filter 是 Zuul 的核心，用来实现对外服务的控制。Filter 的生命周期有 4 个，分是“PRE”、“ROUTING”、“POST”、“ERROR”。

12. 什么是 Spring Cloud Config?

配置管理工具包，让你可以把配置放到远程服务器，集中化管理集群配置，目前支持本地存储、Git 以及 svn。 如果微服务架构中没有使用统一配置中心时，所存在的问题：
配置文件分散在各个项目里，不方便维护
配置内容安全与权限，实际开发中，开发人员是不知道线上环境的配置的

更新配置后，项目需要重启
在分布式系统中，由于服务数量巨多，为了方便服务配置文件统一管理，实时更新，所以需要分布式配置中心组件。市面上开源的配置中心有很多，BAT 每家都出过，360 的 QConf、淘宝的 diamond、百度的 disconf 都是解决这类问题。国外也有很多开源的配置中心 Apache 的 Apache Commons Configuration、owner、cfg4j 等等。在 Spring Cloud 中，有分布式配置中心组件 spring cloud config ，它支持配置服务放在配置服务的内存中（即本地），也支持放在远程 Git 仓库中。在 spring cloud config 组件中，分两个角色，一是 config server，二是 config client。一个配置中心提供的核心功能:

- 提供服务端和客户端支持
- 集中管理各环境的配置文件
- 配置文件修改之后，可以快速的生效
- 可以进行版本管理
- 支持大的并发查询
- 支持各种语言

13. 什么是 Spring Cloud Bus?

在微服务架构的系统中，我们通常会使用轻量级的消息代理来构建一个共用的消息主题让系统中所有微服务实例都连接上来，由于该主题中产生的消息会被所有实例监听和消费，所以我们称它为消息总线。Spring Cloud Bus 就像一个分布式执行器，用于扩展的 Spring Boot 应用程序，但也可以用作应用程序之间的通信通道。Spring Cloud Bus 支持 RabbitMQ 和 Kafka。

14. 什么是 Spring Cloud Stream?

Spring Cloud Stream 是构建消息驱动的微服务应用程序的框架。Spring Cloud Stream 基于 Spring Boot 建立独立的生产级 Spring 应用程序，并使用 Spring Integration 提供与消息代理的连接。它提供了来自几家供应商的中间件的意见配置，介绍了持久发布订阅语义，消费者组和分区的概念。

15. Spring Cloud Stream 与 Spring Cloud Bus 区别？

Spring Cloud Stream 通过对消息中间件进行抽象封装，提供一个统一的接口供我们发送和监听消息，而 Bus 则是在 Stream 基础之上再次进行抽象封装，使得我们可以在不用理解消息发送、监听等概念的基础上使用消息来完成业务逻辑的处理。Spring Cloud Stream 中，异步调用能让各个服务充分解耦而且也更加灵活。而 Spring Cloud Bus 就是借助消息驱动来实现将消息（事件）广播到各个服务中，然后服务对这些消息进行消费。

16. 什么是 Spring Cloud Security？

Spring Cloud Security 提供了一组原语，用于构建安全的应用程序和服务，而且操作简便。可以在外部（或集中）进行大量配置的声明性模型有助于实现大型协作的远程组件系统，通常具有中央身份管理服务。它也非常易于在 Cloud Foundry 等服务平台中使用。在 Spring Boot 和 Spring Security OAuth2 的基础上，可以快速创建实现常见模式的系统，如单点登录，令牌中继和令牌交换。有以下功能：

从 Zuul 代理中的前端到后端服务中继 SSO 令牌
资源服务器之间的中继令牌
使 Feign 客户端表现得像 OAuth2RestTemplate（获取令牌等）的拦截器
在 Zuul 代理中配置下游身份验证

17. SpringBoot 和 SpringCloud

SpringBoot 是 Spring 推出用于解决传统框架配置文件冗余,装配组件繁杂的基于 Maven 的解决方案,旨在快速搭建单个微服务
而 SpringCloud 专注于解决各个微服务之间的协调与配置,服务之间的通信,熔断,负载均衡等
技术维度并相同,并且 SpringCloud 是依赖于 SpringBoot 的,而 SpringBoot 并不是依赖与 SpringCloud,甚至还可以和 Dubbo 进行优秀的整合开发。
总结:
SpringBoot 专注于快速方便的开发单个个体的微服务
SpringCloud 是关注全局的微服务协调整理治理框架,整合并管理各个微服务,为各个微服务之间提供,配置管理,服务发现,断路器,路由,事件总线等集成服务
SpringBoot 不依赖于 SpringCloud,SpringCloud 依赖于 SpringBoot,属于依赖关系
SpringBoot 专注于快速,方便的开发单个的微服务个体,SpringCloud 关注全局的服务治理框架

18. SpringCloud 断路器的作用

当一个服务调用另一个服务由于网络原因或者自身原因出现问题时，调用者就会等待 被调用者的响应，当更多的服务请求到这些资源时。导致更多的请求等待，这样就会发生 锁效应(雪崩效应)，断路器就是解决这一问题。
断路器状态：
完全打开状态：一定时间内达到一定次数无法调用，并且多次检测没有恢复的迹象， 断路器完全打开，那么下次请求就不会请求到该服务。
半开状态：段时间内，有恢复迹象断路器会将部分请求发给该服务，当能正常调用时 断路器关闭。
关闭状态：当服务一直处于正常状态，能正常调用断路器关闭。

19. 什么是服务熔断?什么是服务降级

在复杂的分布式系统中,微服务之间的相互调用,有可能出现各种各样的原因导致服务的阻塞,在高并发场景下,服务的阻塞意味着线程的阻塞,导致当前线程不可用,服务器的线程全部阻塞,导致服务器崩溃,由于服务之间的调用关系是同步的,会对整个微服务系统造成服务雪崩
为了解决某个微服务的调用响应时间过长或者不可用进而占用越来越多的系统资源引起雪崩效应就需要进行服务熔断和服务降级处理。

20. 微服务的优缺点分别是什么?

优点：
每一个服务足够内聚,代码容易理解

开发效率提高,一个服务只做一件事
微服务能够被小团队单独开发
微服务是松耦合的,是有功能意义的服务
可以用不同的语言开发,面向接口编程
易于与第三方集成
微服务只是业务逻辑的代码,不会和 HTML,CSS 或者其他界面组合
开发中,两种开发模式
前后端分离
全栈工程师
可以灵活搭配,连接公共库/连接独立库
缺点
分布式系统的负责性
多服务运维难度,随着服务的增加,运维的压力也在增大
系统部署依赖
服务间通信成本
数据一致性
系统集成测试
性能监控

21. 服务注册和发现是什么意思？Spring Cloud 如何实现？

当我们开始一个项目时，我们通常在属性文件中进行所有的配置。随着越来越多的服务开发和部署，添加和修改这些属性变得更加复杂。有些服务可能会下降，而某些位置可能会发生变化。手动更改属性可能会产生问题。 Eureka 服务注册和发现可以在这种情况下提供帮助。由于所有服务都在 Eureka 服务器上注册并通过调用 Eureka 服务器完成查找，因此无需处理服务地点的任何更改和处理。

22. Spring Cloud 核心组件，在微服务架构中，分别扮演的角色：

Eureka：各个服务启动时，Eureka Client 都会将服务注册到 Eureka Server，并且 Eureka Client 还可以反过来从 Eureka Server 拉取注册表，从而知道其他服务在哪里
Ribbon：服务间发起请求的时候，基于 Ribbon 做负载均衡，从一个服务的多台机器中选择一台
Feign：基于 Feign 的动态代理机制，根据注解和选择的机器，拼接请求 URL 地址，发起请求
Hystrix：发起请求是通过 Hystrix 的线程池来走的，不同的服务走不同的线程池，实现了不同服务调用的隔离，避免了服务雪崩的问题
Zuul：如果前端、移动端要调用后端系统，统一从 Zuul 网关进入，由 Zuul 网关转发请求给对应的服务

23. Eureka 和 ZooKeeper 都可以提供服务注册与发现的功能,请说说两个的区别？

1、ZooKeeper 保证的是 CP,Eureka 保证的是 AP
ZooKeeper 在选举期间注册服务瘫痪,虽然服务最终会恢复,但是选举期间不可用的
Eureka 各个节点是平等关系,只要有一台 Eureka 就可以保证服务可用,而查询到的数据并不是最

新的
自我保护机制会导致
　Eureka 不再从注册列表移除因长时间没收到心跳而应该过期的服务
　Eureka 仍然能够接受新服务的注册和查询请求,但是不会被同步到其他节点(高可用)
　当网络稳定时,当前实例新的注册信息会被同步到其他节点中(最终一致性)
Eureka 可以很好的应对因网络故障导致部分节点失去联系的情况,而不会像 ZooKeeper 一样使得整个注册系统瘫痪
2、ZooKeeper 有 Leader 和 Follower 角色,Eureka 各个节点平等
3、ZooKeeper 采用过半数存活原则,Eureka 采用自我保护机制解决分区问题
4、Eureka 本质上是一个工程,而 ZooKeeper 只是一个进程

24. **你所知道的微服务技术栈有哪些?请列举一二**

多种技术的集合体

我们在讨论一个分布式的微服务架构的话,需要哪些维度

维度(SpringCloud)

服务开发

SpringBoot

Spring

SpringMVC

服务配置与管理

Netfilx 公司的 Archaiusm,阿里的 Diamond

服务注册与发现

Eureka,ZooKeeper

服务调用

Rest,RPC,gRPC

服务熔断器

Hystrix

服务负载均衡

Ribbon,Nginx

服务接口调用

Feign

消息队列

Kafka,RabbitMq,ActiveMq

服务配置中心管理

SpringCloudConfing

服务路由(API 网关)

Zuul

事件消息总线

SpringCloud Bus

十五、SpringSecurity

1. Spring security 的简介

SpringSecurity 一个能够为基于 Spring 的企业应用系统提供声明式的安全訪问控制解决方式的安全框架（简单说是对访问权限进行控制嘛），应用的安全性包括用户认证（Authentication）和用户授权（Authorization）两个部分。用户认证指的是验证某个用户是否为系统中的合法主体，也就是说用户能否访问该系统。用户认证一般要求用户提供用户名和密码。系统通过校验用户名和密码来完成认证过程。用户授权指的是验证某个用户是否有权限执行某个操作。在一个系统中，不同用户所具有的权限是不同的。比如对一个文件来说，有的用户只能进行读取，而有的用户可以进行修改。一般来说，系统会为不同的用户分配不同的角色，而每个角色则对应一系列的权限。spring security 的主要核心功能为认证和授权，所有的架构也是基于这两个核心功能去实现的。

2. 框架原理

对 web 项目得资源安全性得保护，最好得方法是使用 Filter,对方法进行保护，最好得方式使用 AOP。SpringSecurity 对目进行认证和用户得授权时，基于 Sevrvlet 过滤器和 Spring AOP，通过各种各样得拦截器来实现的权限控制，提供了安全性解决方案，可以在 web 项目请求和方法调用过程中处理身份认证和授权，从而实现对项目得安全管理。

3. 核心功能

主要核心核心功能：认证，授权。
认证：指验证某个用户是否是系统中得合法用户，用户是否可以访问该系统，一般要求用户提供用户名和密码进行登录认证。
授权：指验证某个用户是否有权限执行某个操作，在一个系统中，不同用户所拥有得权限是不同得，系统会根据不同得角色分配不同得功能。

4. 框架的核心组件

SecurityContextHolder：提供对 SecurityContext 的访问
SecurityContext,：持有 Authentication 对象和其他可能需要的信息
AuthenticationManager 其中可以包含多个 AuthenticationProvider
ProviderManager 对象为 AuthenticationManager 接口的实现类
AuthenticationProvider 主要用来进行认证操作的类 调用其中的 authenticate()方法去进行认证操作
Authentication：Spring Security 方式的认证主体

GrantedAuthority：对认证主题的应用层面的授权，含当前用户的权限信息，通常使用角色表示
UserDetails：构建 Authentication 对象必须的信息，可以自定义，可能需要访问 DB 得到
UserDetailsService：通过 username 构建 UserDetails 对象，通过 loadUserByUsername 根据 userName 获取 UserDetail 对象 （可以在这里基于自身业务进行自定义的实现　如通过数据库，xml,缓存获取等）

5.　spring security 实现方式

1. 配置文件实现，只需要在配置文件中指定拦截的 url 所需要权限、配置 userDetailsService 指定用户名、密码、对应权限，就可以实现。
2. 实现 UserDetailsService，loadUserByUsername(String userName)方法，根据 userName 来实现自己的业务逻辑返回 UserDetails 的实现类，需要自定义 User 类实现 UserDetails，比较重要的方法是 getAuthorities()，用来返回该用户所拥有的权限。
3. 通过自定义 filter 重写 spring security 拦截器，实现动态过滤用户权限。
4. 通过自定义 filter 重写 spring security 拦截器，实现自定义参数来检验用户，并且过滤权限。

6.　spring security 控制权限的几种方法

在 Spring Security3 的使用中，有 4 种方法：

1．全部利用配置文件，将用户、权限、资源(url)硬编码在 xml 文件中；

2．用户和权限用数据库存储，而资源(url)和权限的对应采用硬编码配置。

3．细分角色和权限，并将用户、角色、权限和资源均采用数据库存储，并且自定义过滤器，代替原有的 FilterSecurityInterceptor 过滤器，并分别实现 AccessDecisionManager、InvocationSecurityMetadataSourceService 和 UserDetailsService，并在配置文件中进行相应配置。

4．修改 spring security 的源代码，主要是修改 InvocationSecurityMetadataSourceService 和 UserDetailsService 两个类。前者是将配置文件或数据库中存储的资源(url)提取出来加工成为 url 和权限列表的 Map 供 Security 使用，后者提取用户名和权限组成一个完整的(UserDetails)User 对象，该对象可以提供用户的详细信息供 AuthentationManager 进行认证与授权使用。

十六、Shiro

1.　简单介绍一下 Shiro 框架

Apache Shiro 是 Java 的一个安全框架。使用 shiro 可以非常容易的开发出足够好的应用，其不仅可以用在 JavaSE 环境，也可以用在 JavaEE 环境。Shiro 可以帮助我们完成：认证、授权、加密、会话管理、与 Web 集成、缓存等。

2.　Shiro 的优点

（1）简单的身份认证，支持多种数据源；
（2）对角色的简单的授权，支持细粒度的授权(方法级)；
（3）支持一级缓存，以提升应用程序的性能；

（4）内置的基于 POJO 企业会话管理，适用于 Web 以及非 Web 的环境；
（5）非常简单的加密 API；
（6）不跟任何的框架或者容器捆绑，可以独立运行；

3. 简述 Shiro 的核心组件

Shiro 架构 3 个核心组件：
（1）Subject：正与系统进行交互的人，或某一个第三方服务。所有 Subject 实例都被绑定到（且这是必须的）一个 SecurityManager 上。
（2）SecurityManager：Shiro 架构的心脏，用来协调内部各安全组件，管理内部组件实例，并通过它来提供安全管理的各种服务。当 Shiro 与一个 Subject 进行交互时，实质上是幕后的 SecurityManager 处理所有繁重的 Subject 安全操作。
（3）Realms：本质上是一个特定安全的 DAO。当配置 Shiro 时，必须指定至少一个 Realm 用来进行身份验证和/或授权。Shiro 提供了多种可用的 Realms 来获取安全相关的数据。如关系数据库(JDBC), INI 及属性文件等。可以定义自己 Realm 实现来代表自定义的数据源。

4. shiro 有哪些组件？

Authentication：身份认证/登录，验证用户是不是拥有相应的身份；
Authorization：授权，即权限验证，验证某个已认证的用户是否拥有某个权限；即判断用户是否能做事情，常见的如：验证某个用户是否拥有某个角色。或者细粒度的验证某个用户对某个资源是否具有某个权限；
Session Manager：会话管理，即用户登录后就是一次会话，在没有退出之前，它的所有信息都在会话中；会话可以是普通 JavaSE 环境的，也可以是如 Web 环境的；
Cryptography：加密，保护数据的安全性，如密码加密存储到数据库，而不是明文存储；
Web Support：Web 支持，可以非常容易的集成到 Web 环境；
Caching：缓存，比如用户登录后，其用户信息、拥有的角色/权限不必每次去查，这样可以提高效率；
Concurrency：shiro 支持多线程应用的并发验证，即如在一个线程中开启另一个线程，能把权限自动传播过去；
Testing：提供测试支持；
Run As：允许一个用户假装为另一个用户（如果他们允许）的身份进行访问；
Remember Me：记住我，这个是非常常见的功能，即一次登录后，下次再来的话不用登录了。
记住一点，Shiro 不会去维护用户、维护权限；这些需要我们自己去设计/提供；然后通过相应的接口注入给 Shiro 即可。

5. Shiro 运行原理

1、Application Code:应用程序代码，就是我们自己的编码，如果在程序中需要进 行权限控制，

需要调用 Subject 的 API。

2、Subject:主体，代表的了当前用户。所有的 Subject 都绑定到 SecurityManager， 与 Subject 的

所有交互都会委托给 SecurityManager,可以将 Subject 当成一个 门面，而真正执行者是 SecurityManager 。

3、SecurityManage:安全管理器，所有与安全有关的操作都会与 SecurityManager 交互，并且它管理所有的 Subject 。

4、Realm:域 shiro 是从 Realm 来获取安全数据（用户，角色，权限）。就是说 SecurityManager 要验证用户身份， 那么它需要从 Realm 获取相应的用户进行比较以确定用户 身份是否合法；也需要从 Realm 得到用户相应的角色/权限进行验证用户是否 能进行操作； 可以把 Realm 看成 DataSource，即安全数据源 。

6. Shiro 认证过程

①应用程序代码调用 Subject。login 方法，传递创建好的包含终端用户的 Principals(身份)和 Credentials(凭证)的 AuthenticationToken 实例

②Subject 实例：通常为 DelegatingSubject(或子类)委托应用程序的 SecurityManager 通过调用 securityManager。login(token) 开始真正的验证。

③ SubjectManager 接收 token，调用内部的 Authenticator 实例调用 authenticator。authenticate(token)。Authenticator 通常是一个 ModularRealmAuthenticator 实例, 支持在身份验证中协调一个或多个 Realm 实例

④如果应用程序中配置了一个以上的 Realm, ModularRealmAuthenticator 实例将利用配置好的 AuthenticationStrategy 来启动 Multi-Realm 认证尝试。在 Realms 被身份验证调用之前, 期间和以后,AuthenticationStrategy 被调用使其能够对每个 Realm 的结果作出反应。

⑤每个配置的 Realm 用来帮助看它是否支持提交的 AuthenticationToken。 如果支持, 那么支持 Realm 的 getAuthenticationInfo 方法将会伴随着提交的 token 被调用。getAuthenticationInfo 方法有效地代表一个特定 Realm 的单一的身份验证尝试。

7. Authentication 和 Authorization

在 shiro 的用户权限认证过程中其通过两个方法来实现：

1、Authentication：是验证用户身份的过程。

2、Authorization：是授权访问控制，用于对用户进行的操作进行人证授权，证明该用户是否允许进行当前操作，如访问某个链接，某个资源文件等。

8. Shiro 工作流程

也就是说对于我们而言，最简单的一个 Shiro 应用：

1、应用代码通过 Subject 来进行认证和授权，而 Subject 又委托给 SecurityManager；

2、我们需要给 Shiro 的 SecurityManager 注入 Realm，从而让 SecurityManager 能得到合法的用户及其权限进行判断。

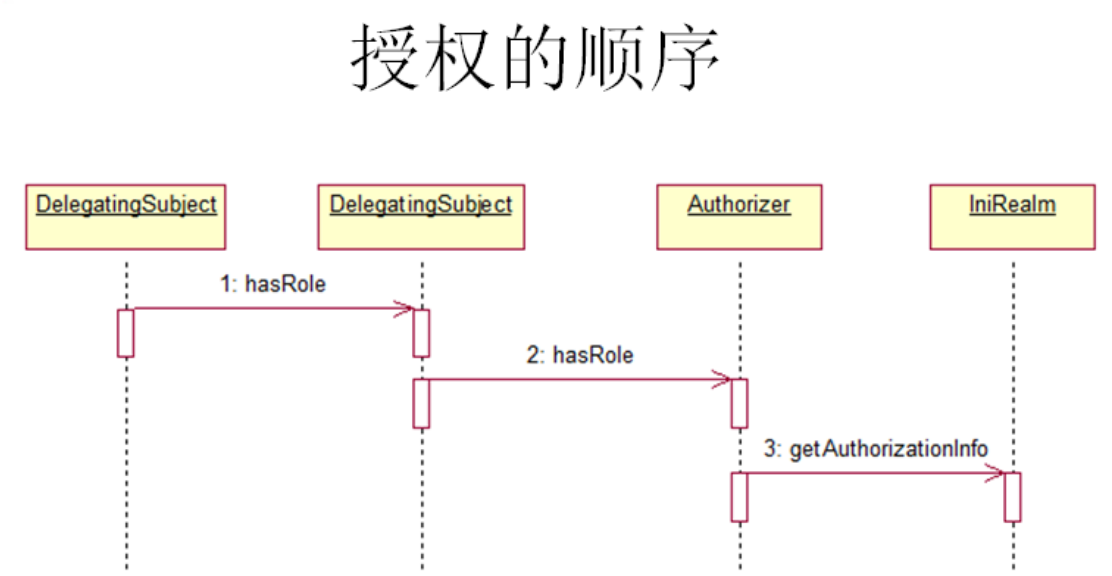
9. Shiro 授权过程

①应用程序或框架代码调用任何 Subject 的 hasRole*, checkRole*, isPermitted*,或者 checkPermission*方法的变体, 传递任何所需的权限

②Subject 的实例一通常是 DelegatingSubject(或子类)，调用 securityManager 的对应的方法。

③SecurityManager 调用 org。apache。shiro。authz。Authorizer 接口的对应方法。默认情况下，authorizer 实例是一个 ModularRealmAuthorizer 实例，它支持协调任何授权操作过程中的一个或多个 Realm 实例

④每个配置好的 Realm 被检查是否实现了相同的 Authorizer 接口。 如果是，Realm 各自的 hasRole*, checkRole*,isPermitted*，或 checkPermission* 方法将被调用。



10. Shiro 如何自实现认证

Shiro 的认证过程由 Realm 执行,SecurityManager 会调用 org.apache.shiro.realm.Realm 的 getAuthenticationInfo(AuthenticationToken token) 方法。 实际开发中，通常提供 org.apache.shiro.realm.AuthenticatingRealm 的实现类，并在该实现类中提供 doGetAuthenticationInfo(AuthenticationToken token)方法的具体实现。

11. shiro 权限认证的三种方式

1.使用 URL 实现权限控制：spring 配置 shirofiter 配置 url 验证规则
2.使用注解实现权限控制：@RequiresPermissions({“ceshi.test”})@RequiresRoles(“adminRole”)
3.使用 shiro 标签进行权限控制

12. 如何实现自实现授权

实际开发中，通常提供 org。apache。shiro。realm。AuthorizingRealm 的实现类，并提供 doGetAuthorizationInfo(PrincipalCollection principals) 方法的具体实现。

13. 如何配置在 Spring 中配置使用 Shiro

①在 web。xml 中配置 Shiro 的 Filter
②在 Spring 的配置文件中配置 Shiro:
配置自定义 Realm：实现自定义认证和授权
配置 Shiro 实体类使用的缓存策略
配置 SecurityManager
配置保证 Shiro 内部 Bean 声明周期都得到执行的 Lifecycle Bean 后置处理器
配置 AOP 式方法级权限检查
配置 Shiro Filter

14. 比较 SpringSecurity 和 Shiro

（1）相比 Spring Security, Shiro 在保持强大功能的同时，使用简单性和灵活性
（2）SpringSecurity：即使是一个一个简单的请求，最少得经过它的 8 个 Filter
（3）SpringSecurity 必须在 Spring 的环境下使用
（4）初学 Spring Security，曲线还是较大，需要深入学习其源码和框架，配置起来也较费力。

十七、Redis

1. Redis 的特点？

Redis 是由意大利人 Salvatore Sanfilippo（网名：antirez）开发的一款内存高速缓存数据库。Redis 全称为：Remote Dictionary Server（远程数据服务），该软件使用 C 语言编写，典型的 NoSQL 数据库服务器，Redis 是一个 key-value 存储系统，它支持丰富的数据类型，如：string、list、set、zset(sorted set)、hash。Redis 本质上是一个 Key-Value 类型的内存数据库，很像 memcached，整个 数据库统统加载在内存当中进行操作，定期通过异步操作把数据库数据 flush 到硬盘 上进行保存。因为是纯内存操作，Redis 的性能非常出色，每秒可以处理超过 10 万次读写操作，是已知性能最快的 Key-Value DB。Redis 的出色之处不仅仅是性能，Redis 最大的魅力是支持保存多种数据结构，此外单 个 value 的最大限制是 1GB，不像 memcached 只能保存 1MB 的数据，另外 Redis 也可以对存入的 Key-Value 设置 expire 时间。Redis 的主要缺点是数据库容量受到物理内存的限制，不能用作海量数据的高性能读写，因此 Redis 适合的场景主要局限在较小数据量的高性能操作和运算上。

2. 为什么 redis 需要把所有数据放到内存中？

Redis 为了达到最快的读写速度将数据都读到内存中，并通过异步的方式将数据写入磁盘。所以 redis 具有快速和数据持久化的特征。如果不将数据放在内存中，磁盘 I/O 速度为严重影响 redis 的性能。在内存越来越便宜的今天，redis 将会越来越受欢迎。如果设置了最大使用的内存，则数据已有记录数达到内存限值后不能继续插入新值。

3. Redis 常见的性能问题都有哪些？如何解决？

（1）、Master 写内存快照，save 命令调度 rdbSave 函数，会阻塞主线程的工作，当快照比较大时对性能影响是非常大的，会间断性暂停服务，所以 Master 最好不要写内存快照。（2）、Master AOF 持久化，如果不重写 AOF 文件，这个持久化方式对性能的影响是最小的，但是 AOF 文件会不断增大，AOF 文件过大会影响 Master 重启的恢复速度。Master 最好不要做任何持久化工作，包括内存快照和 AOF 日志文件，特别是不要启用内存快照做持久化,如果数据比较关键，某个 Slave 开启 AOF 备份数据，策略为每秒同步一次。（3）、Master 调用 BGREWRITEAOF 重写 AOF 文件，AOF 在重写的时候会占大量的 CPU 和内存资源，导致服务 load 过高，出现短暂服务暂停现象。（4）、Redis 主从复制的性能问题，为了主从复制的速度和连接的稳定性，Slave 和 Master 最好在同一个局域网内

4. Redis 最适合的场景有哪些？

（1）、会话缓存（Session Cache） （2）、全页缓存（FPC） （3）、队列（4）、排行榜/计数器（5）、发布/订阅

5. Memcache 与 Redis 的区别都有哪些？

（1）、存储方式不同，Memcache 是把数据全部存在内存中，数据不能超过内存的大小，断电后数据库会挂掉。Redis 有部分存在硬盘上，这样能保证数据的持久性。
（2）、数据支持的类型不同 memcahe 对数据类型支持相对简单，redis 有复杂的数据类型。
（3）、使用底层模型不同 它们之间底层实现方式 以及与客户端之间通信的应用协议不一样。Redis 直接自己构建了 VM 机制 ，因为一般的系统调用系统函数的话，会浪费一定的时间去移动和请求。
（4）、支持的 value 大小不一样 redis 最大可以达到 1GB，而 memcache 只有 1MB。

6. Redis 用过 RedisNX 吗？Redis 有哪几种数据结构？

反正我是不知道 redisnx 是什么，度娘也不清楚，如果面试中问道自己没有接触过或者没有听过的技术可以直接大胆的告诉他，没有接触过，或者没有听过。Redis 的数据结构有五种，分别是：String——字符串 String 数据结构是简单的 key-value 类型，value 不仅可以是 String，也可以是数字（当数字类型用 Long 可以表示的时候 encoding 就是整型，其他都存储在 sdshdr 当做字符串）。Hash——字典在 Memcached 中，我们经常将一些结构化的信息打包成 hashmap，在客户端序列化后存储为一个字符串的值（一般是 JSON 格式），比如用户的昵称、年龄、性别、积分等。List——列表 List 说白了就是链表（redis 使用双端链表实现的 List），相信学过数据结构知识的人都应该能理解其结构。Set——集合 Set 就是一个集合，集合的概念就是一堆不重复值的组合。利用 Redis 提供的 Set 数据结构，可以存储一些集合性的数据。Sorted Set——有序集合和 Sets 相比，Sorted Sets 是将 Set 中的元素增加了一个权重参数 score，使得集合中的元素能够按 score 进行有序排列，1. 带有权重的元素，比如一个游戏的用户得分排行榜 2.比较复杂的数据结构，一般用到的场景不算太多

7. Redis 的优缺点

优点：a) 性能极高 – Redis 能支持超过 100K+ 每秒的读写频率。b) 丰富的数据类型 – Redis 支持二进制案例的 Strings, Lists, Hashes, Sets 及 Ordered Sets 数据类型操作。c) 原子 – Redis 的所有操作都是原子性的，同时 Redis 还支持对几个操作全并后的原子性执行。attention 原子性定义：例如，A 想要从自己的帐户中转 1000 块钱到 B 的帐户里。那个从 A 开始转帐，到转帐结束的这一个过程，称之为一个事务。如果在 A 的帐户已经减去了 1000 块钱的时候，忽然发生了意外，比如停电什么的，导致转帐事务意外终止了，而此时 B 的帐户里还没有增加 1000 块钱。那么，我们称这个操作失败了，要进行回滚。回滚就是回到事务开始之前的状态，也就是回到 A 的帐户还没减 1000 块的状态，B 的帐户的原来的状态。此时 A 的帐户仍然有 3000 块，B 的帐户仍然有 2000 块。我们把这种要么一起成功（A 帐户成功减少 1000，同时 B 帐户成功增加 1000），要么一起失败（A 帐户回到原来状态，B 帐户也回到原来状态）的操作叫原子性操作。如果把一个事务可看作是一个程序,它要么完整的被执行,要么完全不执行，这种特性就叫原子性。•d）丰富的特性 – Redis 还支持 publish/subscribe, 通知, key 过期等等特性。缺点：a）. 由于是内存数据库，所以，单台机器，存储的数据量，跟机器本身的内存大小。虽然 redis 本身有 key 过期策略，但是还是需要提前预估和节约内存。如果内存增长过快，需要定期删除数据。b). 如果进行完整重同步，由于需要生成 rdb 文件，并进行传输，会占用主机的 CPU，并会消耗现网的带宽。不过 redis2.8 版本，已经有部分重同步的功能，但是还是有可能有完整重同步的。比如，新上线的备机。c). 修改配置文件，进行重启，将硬盘中的数据加载进内存，时间比较久。在这个过程中，redis 不能提供服务。

8. Redis 的持久化

RDB 持久化：该机制可以在指定的时间间隔内生成数据集的时间点快照（point-in-time snapshot）。AOF 持久化：记录服务器执行的所有写操作命令，并在服务器启动时，通过重新执行这些命令来还原数据集。AOF 文件中的命令全部以 Redis 协议的格式来保存，新命令会被追加到文件的末尾。 Redis 还可以在后台对 AOF 文件进行重写（rewrite），使得 AOF 文件的体积不会超出保存数据集状态所需的实际大小无持久化：让数据只在服务器运行时存在。同时应用 AOF 和 RDB：当 Redis 重启时， 它会优先使用 AOF 文件来还原数据集， 因为 AOF 文件保存的数据集通常比 RDB 文件所保存的数据集更完整。RDB 的优缺点：优点：RDB 是一个非常紧凑（compact）的文件，它保存了 Redis 在某个时间点上的数据集。 这种文件非常适合用于进行备份： 比如说，你可以在最近的 24 小时内，每小时备份一次 RDB 文件，并且在每个月的每一天，也备份一个 RDB 文件。 这样的话，即使遇上问题，也可以随时将数据集还原到不同的版本。RDB 非常适用于灾难恢复（disaster recovery）：它只有一个文件，并且内容都非常紧凑，可以（在加密后）将它传送到别的数据中心，或者亚马逊 S3 中。RDB 可以最大化 Redis 的性能：父进程在保存 RDB 文件时唯一要做的就是 fork 出一个子进程，然后这个子进程就会处理接下来的所有保存工作，父进程无须执行任何磁盘 I/O 操作。RDB 在恢复大数据集时的速度比 AOF 的恢复速度要快。缺点：如果你需要尽量避免在服务器故障时丢失数据，那么 RDB 不适合你。 虽然 Redis 允许你设置不同的保存点（save point）来控制保存 RDB 文件的频率，但是， 因为 RDB 文件需要保存整个数据集的状态， 所以它并不是一个轻松的操作。 因此你可能会至少 5 分钟才保存一次 RDB 文件。 在这种情况下， 一旦发生故障停机， 你就可能会丢失好几分钟的数据。每次保存 RDB 的时候，Redis 都要 fork() 出一个子进程，并由子进程来进行实际的持久化工作。 在数据集比较庞大时， fork() 可能会非常耗时，造成服务器在某

某毫秒内停止处理客户端；如果数据集非常巨大，并且 CPU 时间非常紧张的话，那么这种停止时间甚至可能会长达整整一秒。

AOF 的优缺点。优点：1、使用 AOF 持久化会让 Redis 变得非常耐久（much more durable）：你可以设置不同的 fsync 策略，比如无 fsync ，每秒钟一次 fsync ，或者每次执行写入命令时 fsync 。 AOF 的默认策略为每秒钟 fsync 一次，在这种配置下，Redis 仍然可以保持良好的性能，并且就算发生故障停机，也最多只会丢失一秒钟的数据（ fsync 会在后台线程执行，所以主线程可以继续努力地处理命令请求）。AOF 文件是一个只进行追加操作的日志文件（append only log）， 因此对 AOF 文件的写入不需要进行 seek ， 即使日志因为某些原因而包含了未写入完整的命令（比如写入时磁盘已满，写入中途停机，等等）， redis-check-aof 工具也可以轻易地修复这种问题。2、Redis 可以在 AOF 文件体积变得过大时，自动地在后台对 AOF 进行重写： 重写后的新 AOF 文件包含了恢复当前数据集所需的最小命令集合。 整个重写操作是绝对安全的，因为 Redis 在创建新 AOF 文件的过程中，会继续将命令追加到现有的 AOF 文件里面，即使重写过程中发生停机，现有的 AOF 文件也不会丢失。 而一旦新 AOF 文件创建完毕，Redis 就会从旧 AOF 文件切换到新 AOF 文件，并开始对新 AOF 文件进行追加操作。缺点：对于相同的数据集来说，AOF 文件的体积通常要大于 RDB 文件的体积。根据所使用的 fsync 策略，AOF 的速度可能会慢于 RDB 。 在一般情况下， 每秒 fsync 的性能依然非常高， 而关闭 fsync 可以让 AOF 的速度和 RDB 一样快， 即使在高负荷之下也是如此。 不过在处理巨大的写入载入时，RDB 可以提供更有保证的最大延迟时间（latency）。AOF 在过去曾经发生过这样的 bug ： 因为个别命令的原因，导致 AOF 文件在重新载入时，无法将数据集恢复成保存时的原样。 （举个例子，阻塞命令 BRPOPLPUSH 就曾经引起过这样的 bug 。） 测试套件里为这种情况添加了测试： 它们会自动生成随机的、复杂的数据集， 并通过重新载入这些数据来确保一切正常。 虽然这种 bug 在 AOF 文件中并不常见， 但是对比来说， RDB 几乎是不可能出现这种 bug 的。

9. 什么是 Redis?

答：Redis 全称为：Remote Dictionary Server（远程数据服务），是一个基于内存的高性能 key-value 数据库。

10. Redis 的数据类型？

答：Redis 支持五种数据类型：string（字符串），hash（哈希），list（列表），set（集合）及 zset(sorted set：有序集合)。
我们实际项目中比较常用的是 string，hash 如果你是 Redis 中高级用户，还需要加上下面几种数据结构 HyperLogLog、Geo、Pub/Sub。
如果你说还玩过 Redis Module，像 BloomFilter，RedisSearch，Redis-ML，面试官得眼睛就开始发亮了。

11、使用 Redis 有哪些好处？

(1) 速度快，因为数据存在内存中，类似于 HashMap，HashMap 的优势就是查找和操作的时间复杂度都是 O(1)
(2) 支持丰富数据类型，支持 string，list，set，Zset，hash 等

(3) 支持事务，操作都是原子性，所谓的原子性就是对数据的更改要么全部执行，要么全部不执行
(4) 丰富的特性：可用于缓存，消息，按 key 设置过期时间，过期后将会自动删除

12、Redis 相比 Memcached 有哪些优势？

(1) Memcached 所有的值均是简单的字符串，redis 作为其替代者，支持更为丰富的数据类型
(2) Redis 的速度比 Memcached 快很多
(3) Redis 可以持久化其数据

13、Memcache 与 Redis 的区别都有哪些？

(1)、存储方式 Memecache 把数据全部存在内存之中，断电后会挂掉，数据不能超过内存大小。Redis 有部份存在硬盘上，这样能保证数据的持久性。
(2)、数据支持类型 Memcache 对数据类型支持相对简单。 Redis 有复杂的数据类型。
(3)、使用底层模型不同 它们之间底层实现方式 以及与客户端之间通信的应用协议不一样。Redis 直接自己构建了 VM 机制 ，因为一般的系统调用系统函数的话，会浪费一定的时间去移动和请求。

14、Redis 是单进程单线程的？

答：Redis 是单进程单线程的，redis 利用队列技术将并发访问变为串行访问，消除了传统数据库串行控制的开销。

15、一个字符串类型的值能存储最大容量是多少？

答：512M

16、Redis 的持久化机制是什么？各自的优缺点？

Redis 提供两种持久化机制 RDB 和 AOF 机制:
1）RDB(Redis DataBase)持久化方式： 是指用数据集快照的方式(半持久化模式)记录 redis 数据库的所有键值对,在某个时间点将数据写入一个临时文件，持久化结束后，用这个临时文件替换上次持久化的文件，达到数据恢复。
优点：
1.只有一个文件 dump.rdb，方便持久化。
2.容灾性好，一个文件可以保存到安全的磁盘。
3.性能最大化，fork 子进程来完成写操作，让主进程继续处理命令，所以是 IO 最大化。(使用单独子进程来进行持久化，主进程不会进行任何 IO 操作，保证了 redis 的高性能) 4.相对于数据集大时，比 AOF 的启动效率更高。
缺点：
1.数据安全性低。(RDB 是间隔一段时间进行持久化，如果持久化之间 redis 发生故障，会发生数

据丢失。所以这种方式更适合数据要求不严谨的时候)

2）AOF(Append-only file)持久化方式： 是指所有的命令行记录以 redis 命令请求协议的格式(完全持久化存储)保存为 aof 文件。

优点：

1.数据安全，aof 持久化可以配置 appendfsync 属性，有 always，每进行一次命令操作就记录到 aof 文件中一次。

2.通过 append 模式写文件，即使中途服务器宕机，可以通过 redis-check-aof 工具解决数据一致性问题。

3.AOF 机制的 rewrite 模式。(AOF 文件没被 rewrite 之前（文件过大时会对命令进行合并重写），可以删除其中的某些命令（比如误操作的 flushall）)

缺点：

1.AOF 文件比 RDB 文件大，且恢复速度慢。

2.数据集大的时候，比 rdb 启动效率低。

17、Redis 常见性能问题和解决方案：

(1) Master 最好不要写内存快照，如果 Master 写内存快照，save 命令调度 rdbSave 函数，会阻塞主线程的工作，当快照比较大时对性能影响是非常大的，会间断性暂停服务。

(2) 如果数据比较重要，某个 Slave 开启 AOF 备份数据，策略设置为每秒同步一次

(3) 为了主从复制的速度和连接的稳定性，Master 和 Slave 最好在同一个局域网内

(4) 尽量避免在压力很大的主库上增加从库

(5) 主从复制不要用图状结构，用单向链表结构更为稳定，即：Master <- Slave1 <- Slave2 <- Slave3...这样的结构方便解决单点故障问题，实现 Slave 对 Master 的替换。如果 Master 挂了，可以立刻启用 Slave1 做 Master，其他不变。

18、redis 过期键的删除策略？

(1)、定时删除:在设置键的过期时间的同时，创建一个定时器(timer). 让定时器在键的过期时间来临时，立即执行对键的删除操作。

(2)、惰性删除:放任键过期不管，但是每次从键空间中获取键时，都检查取得的键是否过期，如果过期的话，就删除该键;如果没有过期，就返回该键。

(3)、定期删除:每隔一段时间程序就对数据库进行一次检查，删除里面的过期键。至于要删除多少过期键，以及要检查多少个数据库，则由算法决定。

19、Redis 的回收策略（淘汰策略）？

volatile-lru：从已设置过期时间的数据集（server.db[i].expires）中挑选最近最少使用的数据淘汰

volatile-ttl：从已设置过期时间的数据集（server.db[i].expires）中挑选将要过期的数据淘汰

volatile-random：从已设置过期时间的数据集（server.db[i].expires）中任意选择数据淘汰

allkeys-lru：从数据集（server.db[i].dict）中挑选最近最少使用的数据淘汰

allkeys-random：从数据集（server.db[i].dict）中任意选择数据淘汰

no-enviction（驱逐）：禁止驱逐数据

注意这里的 6 种机制，volatile 和 allkeys 规定了是对已设置过期时间的数据集淘汰数据还是从全

部数据集淘汰数据，后面的 lru、ttl 以及 random 是三种不同的淘汰策略，再加上一种 no-enviction 永不回收的策略。
使用策略规则：
1、如果数据呈现幂律分布，也就是一部分数据访问频率高，一部分数据访问频率低，则使用 allkeys-lru
2、如果数据呈现平等分布，也就是所有的数据访问频率都相同，则使用 allkeys-random

20、为什么 redis 需要把所有数据放到内存中？

答：Redis 为了达到最快的读写速度将数据都读到内存中，并通过异步的方式将数据写入磁盘。所以 redis 具有快速和数据持久化的特征。如果不将数据放在内存中，磁盘 I/O 速度为严重影响 redis 的性能。在内存越来越便宜的今天，redis 将会越来越受欢迎。如果设置了最大使用的内存，则数据已有记录数达到内存限值后不能继续插入新值。

21、Redis 的同步机制了解么？

答：Redis 可以使用主从同步，从从同步。第一次同步时，主节点做一次 bgsave，并同时将后续修改操作记录到内存 buffer，待完成后将 rdb 文件全量同步到复制节点，复制节点接受完成后将 rdb 镜像加载到内存。加载完成后，再通知主节点将期间修改的操作记录同步到复制节点进行重放就完成了同步过程。

22、Pipeline 有什么好处，为什么要用 pipeline？

答：可以将多次 IO 往返的时间缩减为一次，前提是 pipeline 执行的指令之间没有因果相关性。使用 redis-benchmark 进行压测的时候可以发现影响 redis 的 QPS 峰值的一个重要因素是 pipeline 批次指令的数目。

23、是否使用过 Redis 集群，集群的原理是什么？

(1)、Redis Sentinal 着眼于高可用，在 master 宕机时会自动将 slave 提升为 master，继续提供服务。
(2)、Redis Cluster 着眼于扩展性，在单个 redis 内存不足时，使用 Cluster 进行分片存储。

24、Redis 集群方案什么情况下会导致整个集群不可用？

答：有 A，B，C 三个节点的集群,在没有复制模型的情况下,如果节点 B 失败了，那么整个集群就会以为缺少 5501-11000 这个范围的槽而不可用。

25、Redis 支持的 Java 客户端都有哪些？官方推荐用哪个？

答：Redisson、Jedis、lettuce 等等，官方推荐使用 Redisson。

26、Jedis 与 Redisson 对比有什么优缺点？

答：Jedis 是 Redis 的 Java 实现的客户端，其 API 提供了比较全面的 Redis 命令的支持；Redisson 实现了分布式和可扩展的 Java 数据结构，和 Jedis 相比，功能较为简单，不支持字符串操作，不支持排序、事务、管道、分区等 Redis 特性。Redisson 的宗旨是促进使用者对 Redis 的关注分离，从而让使用者能够将精力更集中地放在处理业务逻辑上。

27、Redis 如何设置密码及验证密码？

设置密码：config set requirepass 123456
授权密码：auth 123456

28、说说 Redis 哈希槽的概念？

答：Redis 集群没有使用一致性 hash,而是引入了哈希槽的概念，Redis 集群有 16384 个哈希槽，每个 key 通过 CRC16 校验后对 16384 取模来决定放置哪个槽，集群的每个节点负责一部分 hash 槽。

29、Redis 集群的主从复制模型是怎样的？

答：为了使在部分节点失败或者大部分节点无法通信的情况下集群仍然可用，所以集群使用了主从复制模型,每个节点都会有 N-1 个复制品.

30、Redis 集群会有写操作丢失吗？为什么？

答：Redis 并不能保证数据的强一致性，这意味这在实际中集群在特定的条件下可能会丢失写操作。

31、Redis 集群之间是如何复制的？

答：异步复制

32、Redis 集群最大节点个数是多少？

答：16384 个。

33、Redis 集群如何选择数据库？

答：Redis 集群目前无法做数据库选择，默认在 0 数据库。

34、怎么测试 Redis 的连通性？

答：使用 ping 命令。

35、怎么理解 Redis 事务？

1）事务是一个单独的隔离操作：事务中的所有命令都会序列化、按顺序地执行。事务在执行的过程中，不会被其他客户端发送来的命令请求所打断。
2）事务是一个原子操作：事务中的命令要么全部被执行，要么全部都不执行。
28、Redis 事务相关的命令有哪几个？
答：MULTI、EXEC、DISCARD、WATCH

36、Redis key 的过期时间和永久有效分别怎么设置？

答：EXPIRE 和 PERSIST 命令。

37、Redis 如何做内存优化？

答：尽可能使用散列表（hashes），散列表（是说散列表里面存储的数少）使用的内存非常小，所以你应该尽可能的将你的数据模型抽象到一个散列表里面。比如你的 web 系统中有一个用户对象，不要为这个用户的名称，姓氏，邮箱，密码设置单独的 key,而是应该把这个用户的所有信息存储到一张散列表里面.

38、Redis 回收进程如何工作的？

答:一个客户端运行了新的命令,添加了新的数据。Redi 检查内存使用情况,如果大于 maxmemory 的限制, 则根据设定好的策略进行回收。一个新的命令被执行，等等。所以我们不断地穿越内存限制的边界，通过不断达到边界然后不断地回收回到边界以下。如果一个命令的结果导致大量内存被使用（例如很大的集合的交集保存到一个新的键），不用多久内存限制就会被这个内存使用量超越。

39、都有哪些办法可以降低 Redis 的内存使用情况呢？

答：如果你使用的是 32 位的 Redis 实例，可以好好利用 Hash,list,sorted set,set 等集合类型数据，因为通常情况下很多小的 Key-Value 可以用更紧凑的方式存放到一起。

40、Redis 的内存用完了会发生什么？

答：如果达到设置的上限，Redis 的写命令会返回错误信息（但是读命令还可以正常返回。）或者你可以将 Redis 当缓存来使用配置淘汰机制，当 Redis 达到内存上限时会冲刷掉旧的内容。

41、一个 Redis 实例最多能存放多少的 keys？List、Set、Sorted Set 他们最多能存放多少元素？

答：理论上 Redis 可以处理多达 232 的 keys，并且在实际中进行了测试，每个实例至少存放了 2 亿 5 千万的 keys。我们正在测试一些较大的值。任何 list、set、和 sorted set 都可以放 232 个元素。换句话说，Redis 的存储极限是系统中的可用内存值。

42、MySQL 里有 2000w 数据，redis 中只存 20w 的数据，如何保证 redis 中的数据都是热点数据？

答：Redis 内存数据集大小上升到一定大小的时候，就会施行数据淘汰策略。
相关知识：Redis 提供 6 种数据淘汰策略：
voltile-lru：从已设置过期时间的数据集（server.db[i].expires）中挑选最近最少使用的数据淘汰
volatile-ttl：从已设置过期时间的数据集（server.db[i].expires）中挑选将要过期的数据淘汰
volatile-random：从已设置过期时间的数据集（server.db[i].expires）中任意选择数据淘汰
allkeys-lru：从数据集（server.db[i].dict）中挑选最近最少使用的数据淘汰
allkeys-random：从数据集（server.db[i].dict）中任意选择数据淘汰
no-enviction（驱逐）：禁止驱逐数据

43、Redis 最适合的场景？

（1）、会话缓存（Session Cache）
最常用的一种使用 Redis 的情景是会话缓存（session cache）。用 Redis 缓存会话比其他存储（如 Memcached）的优势在于：Redis 提供持久化。当维护一个不是严格要求一致性的缓存时，如果用户的购物车信息全部丢失，大部分人都会不高兴的，现在，他们还会这样吗？ 幸运的是，随着 Redis 这些年的改进，很容易找到怎么恰当的使用 Redis 来缓存会话的文档。甚至广为人知的商业平台 Magento 也提供 Redis 的插件。
（2）、全页缓存（FPC）
除基本的会话 token 之外，Redis 还提供很简便的 FPC 平台。回到一致性问题，即使重启了 Redis 实例，因为有磁盘的持久化，用户也不会看到页面加载速度的下降，这是一个极大改进，类似 PHP 本地 FPC。 再次以 Magento 为例，Magento 提供一个插件来使用 Redis 作为全页缓存后端。此外，对 WordPress 的用户来说，Pantheon 有一个非常好的插件 wp-redis，这个插件能帮助你以最快速度加载你曾浏览过的页面。
（3）、队列
Reids 在内存存储引擎领域的一大优点是提供 list 和 set 操作，这使得 Redis 能作为一个很好的消息队列平台来使用。Redis 作为队列使用的操作，就类似于本地程序语言（如 Python）对 list 的 push/pop 操作。 如果你快速的在 Google 中搜索“Redis queues”，你马上就能找到大量的开源项目，这些项目的目的就是利用 Redis 创建非常好的后端工具，以满足各种队列需求。例如，Celery 有一个后台就是使用 Redis 作为 broker，你可以从这里去查看。
（4），排行榜/计数器
Redis 在内存中对数字进行递增或递减的操作实现的非常好。集合（Set）和有序集合（Sorted Set）也使得我们在执行这些操作的时候变的非常简单，Redis 只是正好提供了这两种数据结构。所以，我们要从排序集合中获取到排名最靠前的 10 个用户－我们称之为“user_scores”，我们只需要像

下面一样执行即可： 当然，这是假定你是根据你用户的分数做递增的排序。如果你想返回用户及用户的分数，你需要这样执行： ZRANGE user_scores 0 10 WITHSCORES Agora Games 就是一个很好的例子，用 Ruby 实现的，它的排行榜就是使用 Redis 来存储数据的，你可以在这里看到。

（5）、发布/订阅

最后（但肯定不是最不重要的）是 Redis 的发布/订阅功能。发布/订阅的使用场景确实非常多。我已看见人们在社交网络连接中使用，还可作为基于发布/订阅的脚本触发器，甚至用 Redis 的发布/订阅功能来建立聊天系统！

44、假如 Redis 里面有 1 亿个 key，其中有 10w 个 key 是以某个固定的已知的前缀开头的，如果将它们全部找出来？

答：使用 keys 指令可以扫出指定模式的 key 列表。

对方接着追问：如果这个 redis 正在给线上的业务提供服务，那使用 keys 指令会有什么问题？

这个时候你要回答 redis 关键的一个特性：redis 的单线程的。keys 指令会导致线程阻塞一段时间，线上服务会停顿，直到指令执行完毕，服务才能恢复。这个时候可以使用 scan 指令，scan 指令可以无阻塞的提取出指定模式的 key 列表，但是会有一定的重复概率，在客户端做一次去重就可以了，但是整体所花费的时间会比直接用 keys 指令长。

45、如果有大量的 key 需要设置同一时间过期，一般需要注意什么？

答：如果大量的 key 过期时间设置的过于集中，到过期的那个时间点，redis 可能会出现短暂的卡顿现象。一般需要在时间上加一个随机值，使得过期时间分散一些。

46、使用过 Redis 做异步队列么，你是怎么用的？

答：一般使用 list 结构作为队列，rpush 生产消息，lpop 消费消息。当 lpop 没有消息的时候，要适当 sleep 一会再重试。

如果对方追问可不可以不用 sleep 呢？

list 还有个指令叫 blpop，在没有消息的时候，它会阻塞住直到消息到来。如果对方追问能不能生产一次消费多次呢？使用 pub/sub 主题订阅者模式，可以实现 1:N 的消息队列。

如果对方追问 pub/sub 有什么缺点？

在消费者下线的情况下，生产的消息会丢失，得使用专业的消息队列如 RabbitMQ 等。

如果对方追问 redis 如何实现延时队列？

使用 sortedset，拿时间戳作为 score，消息内容作为 key 调用 zadd 来生产消息，消费者用 zrangebyscore 指令获取 N 秒之前的数据轮询进行处理。

47、设置缓存值的过期时间？

常用的方式：expire key time（以秒为单位）

字符串独有方式：setex（String key，int seconds，String value）

如果没有设置时间，那缓存就是永不过期；