蓝桥面试宝典 2019 版

蓝桥教育

教学部

目录

前言

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

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

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

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

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

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，I/O 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？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、在 Java 中如何跳出当前的多重嵌套循环？

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

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

```
}
```

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

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

5、switch 语句能否作用在 byte 上；作用在 long 上；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。

6、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。

7、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。

8、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。

9、请设计一个一百亿的计算器。

首先要明白这道题目的考查点是什么，一是大家首先要对计算机原理的底层细节要清楚，要知道加减法的位运算原理和知道计算机中的算术运算会发生越界的情况，二是要具备一定的面向对象的设计思想。

首先，计算机中用固定数量的几个字节来存储的数值，所以计算机中能够表示的数值是有一定的范围的，为了便于讲解和理解，我们先以 byte 类型的整数为例，它用 1 个字节进行存储，表示的最大数值范围为-128 到+127。-1 在内存中对应的二进制数据为 11111111，如果两个-1 相加，不考虑 Java 运算时的类型提升，运算后会产生进位，二进制结果为 1,11111110，由于进位后超过了 byte 类型的存储空间，所以进位部分被舍弃，即最终的结果为 11111110，也就是-2，这正好利用溢位的方式实现了负数的运算。-128 在内存中对应的二进制数据为 10000000，如果两个-128 相加，不考虑 Java 运算时的类型提升，运算后会产生进位，二进制结果为 1,00000000，由于进位后超过了 byte 类型的存储空间，所以进位部分被舍弃，即最终的结果为 00000000，也就是 0，这样的结果显然不是我们期望的，这说明计算机中的算术运算是会发生越界情况的，两个数值的运算结果不能超过计算机中的该类型的数值范围。由于 Java 中涉及表达式运算时的类型自动提升，我们无法用 byte 类型来做演示这种问题和现象的实验，大家可以用下面一个使用整数做实验的例子程序体验一下：

```
int a = Integer.MAX_VALUE;
int b = Integer.MAX_VALUE;
int sum = a + b;
System.out.println(“a=”+a+”,b=”+b+”,sum=”+sum);
```

先不考虑 long 类型，由于 int 的正数范围为 2 的 31 次方，表示的最大数值约等于 2*1000*1000*1000，也就是 20 亿的大小，所以，要实现一个一百亿的计算器，我们得自己设计一个类可以用于表示很大的整数，并且提供了与另外一个整数进行加减乘除的功能，大概功能如下：

（）这个类内部有两个成员变量，一个表示符号，另一个用字节数组表示数值的二进制数

（）有一个构造方法，把一个包含有多位数值的字符串转换到内部的符号和字节数组中

（）提供加减乘除的功能

```
public class BigInteger{
int sign;
byte[] val;
public Biginteger(String val){
sign = ;
```

```
val = ;
}
public BigInteger add(BigInteger other){

}
public BigInteger subtract(BigInteger other){

}
public BigInteger multiply(BigInteger other){

}
public BigInteger divide(BigInteger other){

}

}
```

备注：要想写出这个类的完整代码，是非常复杂的，如果有兴趣的话，可以参看 jdk 中自带的 Java.math.BigInteger 类的源码。面试的人也知道谁都不可能在短时间内写出这个类的完整代码的，他要的是你是否有这方面的概念和意识，他最重要的还是考查你的能力，所以，你不要因为自己无法写出完整的最终结果就放弃答这道题，你要做的就是你比别人写得多，证明你比别人强，你有这方面的思想意识就可以了，毕竟别人可能连题目的意思都看不懂，什么都没写，你要敢于答这道题，即使只答了一部分，那也与那些什么都不懂的人区别出来，拉开了距离，算是矮子中的高个，机会当然就属于你了。另外，答案中的框架代码也很重要，体现了一些面向对象设计的功底，特别是其中的方法命名很专业，用的英文单词很精准，这也是能力、经验、专业性、英语水平等多个方面的体现，会给人留下很好的印象，在编程能力和其他方面条件差不多的情况下，英语好除了可以使你获得更多机会外，薪水可以高出一千元。

10、使用 final 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");
```

11、"=="和 equals 方法究竟有什么区别？

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

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

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

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

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

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

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

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

```
boolean equals(Object o){
    return this==o;
}
```

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

12、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！

13、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。

14、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。

15、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。

16、下面的代码有什么不妥之处？

```
1. if(username.equals(“zxx”){}
2.int    x = 1;
return x==1？ true:false;
```

17、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	同类	同包	子孙类	其他 package
public	√	√	√	√
protected	√	√	√	×
default	√	√	×	×
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。

18、Overload 和 Override 的区别。Overloaded 的方法是否可以改变返回值类型？

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。

19、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。

20、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。

21、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。

22、面向对象的特征有哪些方面？

计算机软件系统是现实生活中的业务在计算机中的映射，而现实生活中的业务其实就是一个个对象协作的过程。面向对象编程就是按现实业务一样的方式将程序代码按一个个对象进行组织和编写，让计算机系统能够识别和理解用对象方式组织和编写的程序代码，这样就可以把现实生活中的业务对象映射到计算机系统中。

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。

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;
}
```

人本来是很复杂的事物，有很多方面，但因为当前系统只需要了解人的姓名和年龄，所以上面定义的类中只包含姓名和年龄这两个属性，这就是一种抽像，使用抽象可以避免考虑一些与目标无关的细节。我对抽象的理解就是不要用显微镜去看一个事物的所有方面，这样涉及的内容就太多了，而是要善于划分问题的边界，当前系统需要什么，就只考虑什么。

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);

比喻：人吃饭，你看到的是左手，还是右手？

23、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。

24、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，伪代码如下：

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

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

}
```

父类方法中间的某段代码不确定，留给子类干，就用模板方法设计模式。

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

25、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。

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

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

```
public class Outer{
int out_x  = 0;
public void method(){
Inner1 inner1 = new Inner1();
public class Inner2    //在方法体内部定义的内部类{
public method(){
out_x = 3;
}
}
Inner2 inner2 = new Inner2();
```

```
}

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

}
```

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

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

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

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

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

```
public class Outer
{
public void start()
{
new Thread(
                new Runable(){
public void run(){};
                    }
        ).start();
}
}
```

最后，在方法外部定义的内部类前面可以加上 static 关键字，从而成为 Static Nested Class，它不再具有内部类的特性，所有，从狭义上讲，它不是内部类。Static Nested Class 与普通类在运行时的行为和功能上没有什么区别，只是在编程引用时的语法上有一些差别，它可以定义成 public、protected、默认的、private 等多种类型，而普通类只能定义成 public 和默认的这两种类型。在外面引用 Static Nested Class 类的名称为“外部类名.内部类名”。在外面不需要

创建外部类的实例对象，就可以直接创建 Static Nested Class，例如，假设 Inner 是定义在 Outer 类中的 Static Nested Class，那么可以使用如下语句创建 Inner 类：

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

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

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

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

27、内部类可以引用它的包含类的成员吗？有没有什么限制？

完全可以。如果不是静态内部类，那没有什么限制！

如果你把静态嵌套类当作内部类的一种特例，那在这种情况下不可以访问外部类的普通成员变量，而只能访问外部类中的静态成员，例如，下面的代码：

```
class Outer
{
static int x;
static class Inner
{
void test()
{
syso(x);
}
}
}
```

答题时，也要能察言观色，揣摩提问者的心思，显然人家希望你说的是静态内部类不能访问外部类的成员，但你一上来就顶牛，这不好，要先顺着人家，让人家满意，然后再说特殊情况，让人家吃惊。

28、Anonymous Inner Class（匿名内部类）是否可以 extends(继承)其它类，是否可以 implements(实现)interface(接口)？

可以继承其他类或实现其他接口。不仅是可以，而是必须!

29、super.getClass()方法调用？

下面程序的输出结果是多少？

```
import Java.util.Date;
public    class Test extends Date{
public static void main(String[] args) {
new Test().test();
}

public void test(){
System.out.println(super.getClass().getName());
}
}
```

很奇怪，结果是 Test

这属于脑筋急转弯的题目，在一个 qq 群有个网友正好问过这个问题，我觉得挺有趣，就研究了一下，没想到今天还被你面到了，哈哈。

在 test 方法中，直接调用 getClass().getName()方法，返回的是 Test 类名

由于 getClass()在 Object 类中定义成了 final，子类不能覆盖该方法，所以，在

test 方法中调用 getClass().getName()方法，其实就是在调用从父类继承的 getClass()方法，等效于调用 super.getClass().getName()方法，所以，super.getClass().getName()方法返回的也应该是 Test。

如果想得到父类的名称，应该用如下代码：

```
getClass().getSuperClass().getName();
```

30、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

31、在执行 String s = ″Hello″;s = s + ″ world!″;这两行代码后，原始的 String 对象中的内容到底变了没有？

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

通过上面的说明，我们很容易导出另一个结论，如果经常对字符串进行各种各样的修改，或者说，不可预见的修改，那么使用 String 来代表字符串的话会引起很大的内存开销。因为 String 对象建立之后不能再改变，所以对于每一

个不同的字符串，都需要一个 String 对象来表示。这时，应该考虑使用 StringBuffer 类，它允许修改，而不是每个不同的字符串都要生成一个新的对象。并且，这两种类的对象转换十分容易。
同时，我们还可以知道，如果要使用内容相同的字符串，不必每次都 new 一个 String。例如我们要在构造器中对一个名叫 s 的 String 引用变量进行初始化，把它设置为初始值，应当这样做：

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

而非

```
s = new String("Initial Value");
```

后者每次都会调用构造器，生成新对象，性能低下且内存开销大，并且没有意义，因为 String 对象不可改变，所以对于内容相同的字符串，只要一个 String 对象来表示就可以了。也就说，多次调用上面的构造器创建多个对象，他们的 String 类型属性 s 都指向同一个对象。

上面的结论还基于这样一个事实：对于字符串常量，如果内容相同，Java 认为它们代表同一个 String 对象。而用关键字 new 调用构造器，总是会创建一个新的对象，无论内容是否相同。

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

32、是否可以继承 String 类?

String 类是 final 类故不可以继承。

33、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。

34、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。

接着要举一个具体的例子来说明，我们要把 1 到 100 的所有数字拼起来，组成一个串。

```
StringBuffer sbf = new StringBuffer();
for(int i=0;i<100;i++)
{
sbf.append(i);
}
```

上面的代码效率很高，因为只创建了一个 StringBuffer 对象，而下面的代码效率很低，因为创建了 101 个对象。

```
String str = new String();
for(int i=0;i<100;i++)
{
str = str + i;
}
```

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。

35、如何把一段逗号分割的字符串转换成一个数组？

如果不查 JDK API，我很难写出来！我可以说说我的思路：

用正则表达式，代码大概为：String [] result = orgStr.split(“,”);

用 StingTokenizer ,代码为：

```
StringTokenizer    tokener = StringTokenizer(orgStr,”,”);
String [] result = new String[tokener .countTokens()];
Int i=0;
while(tokener.hasNext(){result[i++]=toker.nextToken();}
```

36、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。

37、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。

38、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 {
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 {

public static void main(String[] args) {
System.out.println(new Test().test());;
}

int test()
{
try
{
return func1();
}
finally
{
return func2();
}
}

int func1()
{
System.out.println("func1");
return 1;
}
int func2()
{
System.out.println("func2");
return 2;
}
}
```

```
----------执行结果----------

func1
func2
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。

45、Java 语言如何进行异常处理，关键字：throws,throw,try,catch,finally 分别代表什么意义？在 try 块中可以抛出异常吗？

Java 通过面向对象的方法进行异常处理，把各种不同的异常进行分类，并提供了良好的接口。

在 Java 中，每个异常都是一个对象，它是 Throwable 类或其子类的实例。当一个方法出现异常后便抛出一个异常对象，该对象中包含有异常信息，调用这个对象的方法可以捕获到这个异常并可以对其进行处理。

Java 的异常处理是通过 5 个关键词来实现的：try、catch、throw、throws 和 finally。一般情况下是用 try 来执行一段程序，如果系统会抛出（throw）一个异常对象，可以通过它的类型来捕获（catch）它，或通过总是执行代码块（finally）来处理；try 用来指定一块预防所有异常的程序；

catch 子句紧跟在 try 块后面，用来指定你想要捕获的异常的类型；

throw 语句用来明确地抛出一个异常；

throws 用来声明一个方法可能抛出的各种异常（当然声明异常时允许无病呻吟）；

finally 为确保一段代码不管发生什么异常状况都要被执行；

try 块中能抛出异常

46、Java 中有几种方法可以实现一个线程？用什么关键字修饰同步方法？ stop()和 suspend()方法为何不推荐使用？

Java5 以前，有如下两种：

第一种：

new Thread(){}.start();这表示调用 Thread 子类对象的 run 方法，new Thread(){}表示一个 Thread 的匿名子类的实例对象，子类加上 run 方法后的代码如下：

```
new Thread(){
public void run(){
}
}.start();
```

第二种：

new Thread(new Runnable(){}).start();这表示调用 Thread 对象接受的 Runnable 对象的 run 方法，new Runnable(){}表示一个 Runnable 的匿名子类的实例对象,runnable 的子类加上 run 方法后的代码如下：

```
new Thread(new Runnable(){
public void run(){
}
}
).start();
```

从 Java5 开始，还有如下一些线程池创建多线程的方式：

```
ExecutorService pool = Executors.newFixedThreadPool(3)
for(int i=0;i<10;i++)
{
  pool.execute(new Runable(){public void run(){}});
}
Executors.newCachedThreadPool().execute(new Runable(){public void run(){}});
Executors.newSingleThreadExecutor().execute(new Runable(){public void run(){}});
```

有两种实现方法，分别使用 new Thread()和 new Thread(runnable)形式，第一种直接调用 thread 的 run 方法，所以，我们往往使用 Thread 子类，即 new SubThread()。第二种调用 runnable 的 run 方法。

有两种实现方法，分别是继承 Thread 类与实现 Runnable 接口

用 synchronized 关键字修饰同步方法

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

wait()命其进入等待状态。若标志指出线程应当恢复，则用一个 notify()重新启动线程。

47、sleep() 和 wait() 有什么区别？

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。对于 wait 的讲解一定要配合例子代码来说明，才显得自己真明白。

```
package com.huawei.interview;
public class MultiThread {
public static void main(String[] args) {
new Thread(new Thread1()).start();
try {
Thread.sleep(10);
} catch (InterruptedException e) {
e.printStackTrace();
}
new Thread(new Thread2()).start();
}


private static class Thread1 implements Runnable
{

@Override
public void run() {
```

//由于这里的 Thread1 和下面的 Thread2 内部 run 方法要用同一对象作为监视器，我们这里不能用 this，因为在 Thread2 里面的 this 和这个 Thread1 的 this 不是同一个对象。我们用 MultiThread.class 这个字节码对象，当前虚拟机里引用这个变量时，指向的都是同一个对象。

```
synchronized (MultiThread.class) {

System.out.println("enter thread1...");

System.out.println("thread1 is waiting");
try {
  //释放锁有两种方式，第一种方式是程序自然离开监视器的范围，也就是离开了synchronized关键字管辖的代码范围，另一种方式就是在synchronized关键字管辖的代码内部调用监视器对象的wait方法。这里，使用wait方法释放锁。
MultiThread.class.wait();
} catch (InterruptedException e) {
e.printStackTrace();
}
System.out.println("thread1 is going on...");
System.out.println("thread1 is being over!");
}
}

}

private static class Thread2 implements Runnable
{

@Override
public void run() {
synchronized (MultiThread.class) {
System.out.println("enter thread2...");
System.out.println("thread2 notify other thread can release wait status..");
//由于 notify 方法并不释放锁， 即使 thread2 调用下面的 sleep 方法休息了 10 毫秒，但 thread1 仍然不会执行，因为 thread2 没有释放锁，所以 Thread1 无法得不到锁。
  MultiThread.class.notify();
System.out.println("thread2 is sleeping ten millisecond...");
try {
Thread.sleep(10);
} catch (InterruptedException e) {
e.printStackTrace();
}
System.out.println("thread2 is going on...");
System.out.println("thread2 is being over!");
}
}
}
```

```
}
```

48、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。

49、下面两个方法同步吗？

```
class Test
  {
  synchronized static void sayHello3()
  {

  }
  synchronized void getX(){}
  }
```

50、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。

51、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。

52、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。

53、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。

54、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

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）：

```
package com.huawei.interview;
import Java.util.concurrent.locks.Lock;
import Java.util.concurrent.locks.ReentrantLock;
public class ThreadTest {
private int j;
private Lock lock = new ReentrantLock();
```

```
public static void main(String[] args) {
ThreadTest tt = new ThreadTest();
for(int i=0;i<2;i++)
{
new Thread(tt.new Adder()).start();
new Thread(tt.new Subtractor()).start();
}
}
private class Subtractor implements Runnable
{
@Override
public void run() {
while(true)
{
/*synchronized (ThreadTest.this) {
System.out.println("j--=" + j--);
//这里抛异常了，锁能释放吗？
}*/
lock.lock();
try
{
System.out.println("j--=" + j--);
}finally
{
lock.unlock();
}
}
}

}
private class Adder implements Runnable
{
@Override
public void run() {
while(true)
{
/*synchronized (ThreadTest.this) {
System.out.println("j++=" + j++);
}*/
lock.lock();
try
{
System.out.println("j++=" + j++);
```

```
}finally
{
lock.unlock();
}
}
}

}
}
```

55、设计 4 个线程，其中两个线程每次对 j 增加 1，另外两个线程对 j 每次减少 1。

以下程序使用内部类实现线程，对 j 增减的时候没有考虑顺序问题。

```
public class ThreadTest1
{
private int j;
public static void main(String args[]){
    ThreadTest1 tt=new ThreadTest1();
    Inc inc=tt.new Inc();
    Dec dec=tt.new Dec();
    for(int i=0;i<2;i++){
        Thread t=new Thread(inc);
        t.start();
    t=new Thread(dec);
        t.start();
        }
    }
private synchronized void inc(){
    j++;
    System.out.println(Thread.currentThread().getName()+"-inc:"+j);
    }
private synchronized void dec(){
    j--;
    System.out.println(Thread.currentThread().getName()+"-dec:"+j);
    }
class Inc implements Runnable{
    public void run(){
        for(int i=0;i<100;i++){
        inc();
        }
    }
}
```

```
class Dec implements Runnable{
    public void run(){
        for(int i=0;i<100;i++){
        dec();
        }
    }
}
}

----------随手再写的一个----------
class A
{
JManger j =new JManager();
main()
{
new A().call();
}

void call
{
for(int i=0;i<2;i++)
{
new Thread(
new Runnable(){ public void run(){while(true){j.accumulate()}}}
).start();
new Thread(new Runnable(){ public void run(){while(true){j.sub()}}}).start();
}
}
    }

    class JManager
    {
    private j = 0;

    public synchronized void subtract()
    {
    j--
    }
    public synchronized void accumulate()
    {
    j++;
    }
    }
```

56、根据所学线程知识，按要求写出代码。

子线程循环 10 次，接着主线程循环 100，接着又回到子线程循环 10 次，接着再回到主线程又循环 100，如此循环 50 次，请写出程序。

代码如下：

```
public class ThreadTest {

/**
 * @param args
 */
public static void main(String[] args) {
// TODO Auto-generated method stub
new ThreadTest().init();

}

public void init()
{
final Business business = new Business();
new Thread(
new Runnable()
{

public void run() {
for(int i=0;i<50;i++)
{
business.SubThread(i);
}
}

}

).start();

for(int i=0;i<50;i++)
{
business.MainThread(i);
}
}

private class Business
{
```

```
boolean bShouldSub = true;//这里相当于定义了控制该谁执行的一个信号灯
public synchronized void MainThread(int i)
{
if(bShouldSub)
try {
this.wait();
} catch (InterruptedException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

for(int j=0;j<5;j++)
{
System.out.println(Thread.currentThread().getName() + ":i=" + i +",j=" + j);
}
bShouldSub = true;
this.notify();

}
public synchronized void SubThread(int i)
{
if(!bShouldSub)
try {
this.wait();
} catch (InterruptedException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

for(int j=0;j<10;j++)
{
System.out.println(Thread.currentThread().getName() + ":i=" + i +",j=" + j);
}
bShouldSub = false;
this.notify();
}
}
}
```

备注：不可能一上来就写出上面的完整代码，最初写出来的代码如下，问题在于两个线程的代码要参照同一个变量，即这两个线程的代码要共享数据，所以，把这两个线程的执行代码搬到同一个类中去：

```
package com.huawei.interview.lym;
public class ThreadTest {
```

```
private static boolean bShouldMain = false;
public static void main(String[] args) {
// TODO Auto-generated method stub
/*new Thread(){
public void run()
{
for(int i=0;i<50;i++)
{
for(int j=0;j<10;j++)
{
System.out.println("i=" + i + ",j=" + j);
}
}
}

}.start();*/


//final String str = new String("");

new Thread(
new Runnable()
{
public void run()
{
for(int i=0;i<50;i++)
{
synchronized (ThreadTest.class) {
if(bShouldMain)
{
try {
ThreadTest.class.wait();}
catch (InterruptedException e) {
e.printStackTrace();
}
}
for(int j=0;j<10;j++)
{
System.out.println(
Thread.currentThread().getName() +
"i=" + i + ",j=" + j);
}
bShouldMain = true;
```

```
ThreadTest.class.notify();
}
}
}
}
).start();

for(int i=0;i<50;i++)
{
synchronized (ThreadTest.class) {
if(!bShouldMain)
{
try {
ThreadTest.class.wait();}
catch (InterruptedException e) {
e.printStackTrace();
}
}
for(int j=0;j<5;j++)
{
System.out.println(
Thread.currentThread().getName() +
"i=" + i + ",j=" + j);
}
bShouldMain = false;
ThreadTest.class.notify();
}
}
}

}
```

下面使用 jdk5 中的并发库来实现的：

```
import Java.util.concurrent.Executors;
import Java.util.concurrent.ExecutorService;
import Java.util.concurrent.locks.Lock;
import Java.util.concurrent.locks.ReentrantLock;
import Java.util.concurrent.locks.Condition;

public class ThreadTest
{
private static Lock lock = new ReentrantLock();
private static Condition subThreadCondition = lock.newCondition();
private static boolean bBhouldSubThread = false;
```

```
public static void main(String [] args)
{
ExecutorService threadPool = Executors.newFixedThreadPool(3);
threadPool.execute(new Runnable(){
public void run()
{
for(int i=0;i<50;i++)
{
lock.lock();
try
{
if(!bBhouldSubThread)
subThreadCondition.await();
for(int j=0;j<10;j++)
{
System.out.println(Thread.currentThread().getName() + ",j=" + j);
}
bBhouldSubThread = false;
subThreadCondition.signal();
}catch(Exception e)
{
}
finally
{
lock.unlock();
}
}
}

});
threadPool.shutdown();
for(int i=0;i<50;i++)
{
lock.lock();
try
{
if(bBhouldSubThread)
subThreadCondition.await();
for(int j=0;j<10;j++)
{
System.out.println(Thread.currentThread().getName() + ",j=" + j);
}
bBhouldSubThread = true;
```

```
subThreadCondition.signal();
}catch(Exception e)
{
}
finally
{
lock.unlock();
}
}
}
}
```

57、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。

58、Collection 框架中实现比较要实现什么接口？

comparable/comparator

59、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。

60、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

61、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。

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

List，Set 是，Map 不是

63、List、Map、Set 三个接口，存取元素时，各有什么特点？

这样的题属于随意发挥题：这样的题比较考水平，两个方面的水平：一是要真正明白这些内容，二是要有较强的总结和表述能力。如果你明白，但表述不清楚，在别人那里则等同于不明白。

首先，List 与 Set 具有相似性，它们都是单列元素的集合，所以，它们有一个功共同的父接口，叫 Collection。Set 里面不允许有重复的元素，所谓重复，即不能有两个相等（注意，不是仅仅是相同）的对象 ，即假设 Set 集合中有了一个 A 对象，现在我要向 Set 集合再存入一个 B 对象，但 B 对象与 A 对象 equals 相等，则 B 对象存储不进去，所以，Set 集合的 add 方法有一个 boolean 的返回值，当集合中没有某个元素，此时 add 方法可成功加入该元素时，则返回 true，当集合含有与某个元素 equals 相等的元素时，此时 add 方法无法加入该元素，返回结果为 false。Set 取元素时，没法说取第几个，只能以 Iterator 接口取得所有的元素，再逐一遍历各个元素。

List 表示有先后顺序的集合， 注意，不是那种按年龄、按大小、按价格之类的排序。当我们多次调用 add(Obj e)方法时，每次加入的对象就像火车站买票有排队顺序一样，按先来后到的顺序排序。有时候，也可以插队，即调用 add(int index,Obj e)方法，就可以指定当前对象在集合中的存放位置。一个对象可以被反复存储进 List 中，每调用一次 add 方法，这个对象就被插入进集合中一次，其实，并不是把这个对象本身存储进了集合中，而是在集合中用一个索引变量指向这个对象，当这个对象被 add 多次时，即相当于集合中有多个索引指向了这个对象，如图 x 所示。List 除了可以以 Iterator 接口取得所有的元素，再逐一遍历各个元素之外，还可以调用 get(index i)来明确说明取第几个。

Map 与 List 和 Set 不同，它是双列的集合，其中有 put 方法，定义如下：put(obj key,obj value)，每次存储时，要存储一对 key/value，不能存储重复的 key，这个重复的规则也是按 equals 比较相等。取则可以根据 key 获得相应的 value，即 get(Object key)返回值为 key 所对应的 value。另外，也可以获得所有的 key 的结合，还可以获得所有的 value 的结合，还可以获得 key 和 value 组合成的 Map.Entry 对象的集合。

List 以特定次序来持有元素，可有重复元素。Set 无法拥有重复元素,内部排序。Map 保存 key-value 值，value 可多值。

HashSet 按照 hashcode 值的某种运算方式进行存储，而不是直接按 hashCode 值的大小进行存储。例如，"abc" ---> 78，"def" ---> 62，"xyz" ---> 65 在 hashSet 中的存储顺序不是 62,65,78，这些问题感谢以前一个叫崔健的学员提出，最后通过查看源代码给他解释清楚，看本次培训学员当中有多少能看懂源码。LinkedHashSet 按插入的顺序存储，那被存储对象的 hashcode 方法还有什么作用呢？学员想想!hashset 集合比较两个对象是否相等，首先看 hashcode 方法是否相等，然后看 equals 方法是否相等。new 两个 Student 插入到 HashSet 中，看 HashSet 的 size，实现 hashcode 和 equals 方法后再看 size。

同一个对象可以在 Vector 中加入多次。往集合里面加元素，相当于集合里用一根绳子连接到了目标对象。往 HashSet 中却加不了多次的。

64、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。

65、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);

66、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。

67、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？

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。

68、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。

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

这句话是对的。

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。

70、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！

本题目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 {
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());
}
}
```

71、说出一些常用的类，包，接口，请各举 5 个。

要让人家感觉你对JavaEE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

72、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 Reader 和 Writer。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。

73、字节流与字符流的区别？

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

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

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

字符向字节转换时，要注意编码的问题，因为字符串转成字节数组，

其实是转成该字符的某种编码的字节形式，读取也是反之的道理。

讲解字节流与字符流关系的代码案例：

```
import Java.io.BufferedReader;
import Java.io.FileInputStream;
import Java.io.FileOutputStream;
import Java.io.FileReader;
import Java.io.FileWriter;
import Java.io.InputStreamReader;
import Java.io.PrintWriter;
public class IOTest {
public static void main(String[] args) throws Exception {
String str = "中国人";
/*FileOutputStream fos    = new FileOutputStream("1.txt");
fos.write(str.getBytes("UTF-8"));
fos.close();*/
/*FileWriter fw = new FileWriter("1.txt");
fw.write(str);
fw.close();*/
PrintWriter pw = new PrintWriter("1.txt","utf-8");
pw.write(str);
pw.close();

/*FileReader fr = new FileReader("1.txt");
char[] buf = new char[1024];
int len = fr.read(buf);
String myStr = new String(buf,0,len);
```

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

}
```

74、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。

75、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。

76、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。

77、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。

78、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。

79、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。

80、什么时候用 assert。

assertion (断言) 在软件开发中是一种常用的调试方式，很多开发语言中都支持这种机制。在实现中，assertion 就是在程序中的一条语句，它对一个 boolean 表达式进行检查，一个正确程序必须保证这个 boolean 表达式的值为 true；如果该值为 false，说明程序已经处于不正确的状态下，assert 将给出警告或退出。一般来说，assertion 用于保证程序最基本、关键的正确性。assertion 检查通常在开发和测试时开启。为了提高性能，在软件发布后，assertion 检查通常是关闭的。

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

```
public class AssertTest {

/**
 * @param args
 */
public static void main(String[] args) {
// TODO Auto-generated method stub
int i = 0;
for(i=0;i<5;i++)
{
System.out.println(i);
}
//假设程序不小心多了一句--i;
--i;
assert i==5;
}

}
```

81、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：

```
package com.huawei.interview;
import Java.io.IOException;
public class GarbageTest {
public static void main(String[] args) throws IOException {
// TODO Auto-generated method stub
try {
gcTest();
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
System.out.println("has exited gcTest!");
System.in.read();
System.in.read();
System.out.println("out begin gc!");
for(int i=0;i<100;i++)
{
```

```
System.gc();
System.in.read();
System.in.read();
}
}

private static void gcTest() throws IOException {
System.in.read();
System.in.read();
Person p1 = new Person();
System.in.read();
System.in.read();
Person p2 = new Person();
p1.setMate(p2);
p2.setMate(p1);
System.out.println("before exit gctest!");
System.in.read();
System.in.read();
System.gc();
System.out.println("exit gctest!");
}

private static class Person
{
byte[] data = new byte[20000000];
Person mate = null;
public void setMate(Person other)
{
mate = other;
}
}
}
```

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

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

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

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

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

```
 public class Stack {
private Object[] elements=new Object[10];
private int size = 0;
public void push(Object e){
ensureCapacity();
elements[size++] = e;
}
public Object pop(){
if( size == 0)
 throw new EmptyStackException();
return elements[--size];
}
private void ensureCapacity(){
if(elements.length == size){
Object[] oldElements = elements;
elements = new Object[2 * elements.length+1];
System.arraycopy(oldElements,0, elements, 0, size);
}
}
}
```

上面的原理应该很简单，假如堆栈加了 10 个元素，然后全部弹出来，虽然堆栈是空的，没有我们要的东西，但是这 10 个对象是无法回收的，这个才符合了内存泄露的两个条件：无用，无法回收。

但是就是存在这样的东西也不一定会导致什么样的后果，如果这个堆栈用的比较少，也就浪费了几个 K 内存而已，反正我们的内存都上 G 了，哪里会有什么影响，再说这个东西很快就会被回收的，有什么关系。下面看两个例子。

例子 1

```
public class Bad{
public static Stack s=Stack();
static{
```

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

因为是 static，就一直存在到程序退出，但是我们也可以看到它有自愈功能，就是说如果你的 Stack 最多有 100 个对象，那么最多也就只有 100 个对象无法被回收其实这个应该很容易理解，Stack 内部持有 100 个引用，最坏的情况就是他们都是无用的，因为我们一旦放新的进取，以前的引用自然消失！

内存泄露的另外一种情况：当一个对象被存储进 HashSet 集合中以后，就不能修改这个对象中的那些参与计算哈希值的字段了，否则，对象修改后的哈希值与最初存储进 HashSet 集合中时的哈希值就不同了，在这种情况下，即使在 contains 方法使用该对象的当前引用作为的参数去 HashSet 集合中检索对象，也将返回找不到对象的结果，这也会导致无法从 HashSet 集合中单独删除当前对象，造成内存泄露。

82、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。

例如，运行下面的程序：

```
package Java.lang;
public class String {
public static void main(String[] args) {
// TODO Auto-generated method stub
System.out.println("string");
}
}
```

报告的错误如下：

```
Java.lang.NoSuchMethodError: main
Exception in thread "main"
```

这是因为加载了jre自带的Java.lang.String，而该类中没有main方法。

83、Java 代码查错

1.

```
abstract class Name {
   private String name;
   public abstract boolean isStupidName(String name) {}
}
```

大侠们，这有何错误？

答案：错。abstract method 必须以分号结尾，且不带花括号。

2.

```
public class Something {
   void doSomething () {
      private String s = "";
      int l = s.length();
   }
}
```

有错吗？

答案：错。局部变量前不能放置任何访问修饰符（private，public，和 protected)。final 可以用来修饰局部变量(final 如同 abstract 和 strictfp，都是非访问修饰符，strictfp 只能修饰 class 和 method 而非 variable)。

3.

```
abstract class Something {
   private abstract String doSomething ();
}
```

这好像没什么错吧？

答案：错。abstract 的 methods 不能以 private 修饰。abstract 的 methods 就是让子类 implement(实现)具体细节的，怎么可以用 private 把 abstract method 封锁起来呢？（同理，abstract method 前不能加 final)。

4.

```
public class Something {
   public int addOne(final int x) {
      return ++x;
   }
}
```

这个比较明显。

答案：错。int x 被修饰成 final，意味着 x 不能在 addOne method 中被修改。

5.

```
public class Something {
   public static void main(String[] args) {
      Other o = new Other();
      new Something().addOne(o);
   }
   public void addOne(final Other o) {
```

```
      o.i++;
    }
}
class Other {
   public int i;
}
```

和上面的很相似，都是关于 final 的问题，这有错吗？
答案: 正确。在 addOne method 中，参数 o 被修饰成 final。如果在 addOne method 里我们修改了 o 的 reference
(比如: o = new Other();)，那么如同上例这题也是错的。但这里修改的是 o 的 member vairable
(成员变量)，而 o 的 reference 并没有改变。
6.

```
class Something {
    int i;
    public void doSomething() {
       System.out.println("i = " + i);
    }
}
```

有什么错呢？ 看不出来啊。
答案: 正确。输出的是"i = 0"。int i 属於 instant variable (实例变量，或叫成员变量)。instant variable 有 default value。int 的 default value 是 0。
7.

```
class Something {
    final int i;
    public void doSomething() {
       System.out.println("i = " + i);
    }
}
```

和上面一题只有一个地方不同，就是多了一个 final。这难道就错了吗？
答案: 错。final int i 是个 final 的 instant variable (实例变量，或叫成员变量)。final 的 instant variable 没有 default value，必须在 constructor (构造器)结束之前被赋予一个明确的值。可以修改为"final int i = 0;"。
8.

```
public class Something {
     public static void main(String[] args) {
       Something s = new Something();
       System.out.println("s.doSomething() returns " + doSomething());
    }
    public String doSomething() {
       return "Do something ...";
    }
}
```

看上去很完美。
答案: 错。看上去在 main 里 call doSomething 没有什么问题，毕竟两个 methods 都在同一个 class 里。但仔细看，main 是 static 的。static method 不能直接 call non-static methods。可改成"System.out.println("s.doSomething() returns " + s.doSomething());"。同理，static method 不能访问 non-static instant variable。
9.
此处，Something 类的文件名叫 OtherThing.Java

```
class Something {
    private static void main(String[] something_to_do) {
       System.out.println("Do something ...");
    }
}
```

这个好像很明显。
答案: 正确。从来没有人说过 Java 的 Class 名字必须和其文件名相同。但 public class 的名字必须和文件名相同。
10.

```
interface   A{
   int x = 0;
}
class B{
   int x =1;
}
class C extends B implements A {
   public void pX(){
     System.out.println(x);
   }
   public static void main(String[] args) {
     new C().pX();
   }
}
```

答案：错误。在编译时会发生错误(错误描述不同的 JVM 有不同的信息，意思就是未明确的 x 调用，两个 x 都匹配（就象在同时 import Java.util 和 Java.sql 两个包时直接声明 Date 一样）。对于父类的变量,可以用 super.x 来明确，而接口的属性默认隐含为 public static final.所以可以通过 A.x 来明确。
11.

```
interface Playable {
    void play();
}
interface Bounceable {
    void play();
}
interface Rollable extends Playable, Bounceable {
    Ball ball = new Ball("PingPang");
```

```
}
class Ball implements Rollable {
    private String name;
    public String getName() {
        return name;
    }
    public Ball(String name) {
        this.name = name;
    }
   public void play() {
        ball = new Ball("Football");
        System.out.println(ball.getName());
    }
}
```

这个错误不容易发现。

答案：错。"interface Rollable extends Playable, Bounceable"没有问题。interface 可继承多个 interfaces，所以这里没错。问题出在 interface Rollable 里的"Ball ball = new Ball("PingPang");"。任何在 interface 里声明的 interface variable (接口变量，也可称成员变量)，默认为 public static final。也就是说"Ball ball = new Ball("PingPang");"实际上是"public static final Ball ball = new Ball("PingPang");"。在 Ball 类的 Play()方法中，"ball = new Ball("Football");"改变了 ball 的 reference，而这里的 ball 来自 Rollable interface，Rollable interface 里的 ball 是 public static final 的，final 的 object 是不能被改变 reference 的。因此编译器将在"ball = new Ball("Football");"这里显示有错。

84、JavaEE 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。

85、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。

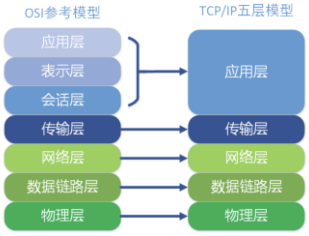
86、计算机网络的分层模型有 TCP/IP 模型和 OSI 模型,请描述一下这两种模型的区别？

OSI（Open System Interconnect），即开放式系统互联。 一般都叫 OSI 参考模型，是 ISO（国际标准化组织）组织在 1985 年研究的网络互连模型。ISO 为了更好的使网络应用更为普及，推出了 OSI 参考模型。其含义就是推荐所有公司使用这个规范来控制网络。这样所有公司都有相同的规范，就能互联了。

OSI 七层模型的划分

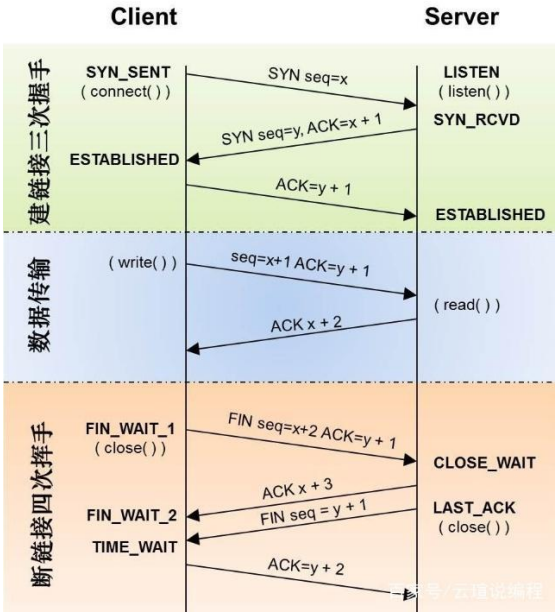
OSI 定义了网络互连的七层框架（物理层、数据链路层、网络层、传输层、会话层、表示
层、应用层），即 ISO 开放互连系统参考模型。每一层实现各自的功能和协议，并完成与相邻层的接口通信。OSI 的服务定义详细说明了各层所提供的服务。某一层的服务就是该层及其下各层的一种能力，它通过接口提供给更高一层。各层所提供的服务与这些服务是怎么实现的无关。

TCP/IP 模型只定义了五层,OSI 的七层只是理论上的标准，而 TCP/IP 的五层模型则成了事实上的标准，因为各大操作系统内核都只是做了 TCP/IP 的实现。其五层与 OSI 的七层的对应关系如下图:



87、描述 TCP 通信的三次握手过程和四次挥手过程。

TCP通信过程包括三个步骤：建立TCP连接通道，传输数据，断开TCP连接通道。如图1所示，给出了TCP通信过程的示意图。



上图主要包括三部分：建立连接、传输数据、断开连接。

建立TCP连接很简单，通过三次握手便可建立连接。建立好连接后，开始传输数据。TCP数据传输牵涉到的概念很多：超时重传、快速重传、流量控制、拥塞控制等等。断开连接的过程也很简单，通过四次握手完成断开连接的过程。三次握手建立连接：

第一次握手：客户端发送syn包(seq=x)到服务器，并进入SYN_SEND状态，等待服务器确认;第二次握手：服务器收到syn包，必须确认客户的SYN(ack=x+1)，同时自己也发送一个SYN包(seq=y)，即SYN+ACK包，此时服务器进入SYN_RECV状态;第三次握手：客户端收到服务器的SYN+ACK包，向服务器发送确认包ACK(ack=y+1)，此包发送完毕，客户端和服务器进入ESTABLISHED状态，完成三次握手。

握手过程中传送的包里不包含数据，三次握手完毕后，客户端与服务器才正式开始传送数据。理想状态下，TCP连接一旦建立，在通信双方中的任何一方主动关闭连接之前，TCP 连接都将被一直保持下去。

四次握手断开连接：

第一次挥手：主动关闭方发送一个FIN，用来关闭主动方到被动关闭方的数据传送，也就是主动关闭方告诉被动关闭方：我已经不会再给你发数据了(当 然，

在fin包之前发送出去的数据，如果没有收到对应的ack确认报文，主动关闭方依然会重发这些数据)，但此时主动关闭方还可以接受数据。

第二次挥手：被动关闭方收到FIN包后，发送一个ACK给对方，确认序号为收到序号+1(与SYN相同，一个FIN占用一个序号)。

第三次挥手：被动关闭方发送一个FIN，用来关闭被动关闭方到主动关闭方的数据传送，也就是告诉主动关闭方，我的数据也发送完了，不会再给你发数据了。

第四次挥手：主动关闭方收到FIN后，发送一个ACK给被动关闭方，确认序号为收到序号+1，至此，完成四次挥手。

88、TCP 和 UDP 有什么区别？

TCP协议---传输控制协议

UDP协议---用户数据报协议

TCP协议

1）面向连接的可靠的传输控制协议,连接的建立需要三次握手，连接的释放需要进行四次握手才能保证连接的建立，数据的同步传输。

2）面向字节流，会把从上层传输下来的数据当作是无结构的字节流。

3）一对一的通信。

4）TCP在IP协议的基础之上添加了序号机制，确认机制，超时重传机制，数据校验，从而保证传输的可靠性，同时保证不出现丢失或者是乱序。

UDP协议

1）无连接的数据包服务，一方向另一方发送数据不需要建立连接。相当于发短信，别人是否收到，短信信息是否丢失都不能知道。

2）面向报文的，从上层接收的数据如果报文不大于传输限制，则直接加上首部传输，如果报文过大，则进行IP分片后，再分别加入首部进行传输。

3）UDP协议可以一对一通信，同时可以一对多通信。

4）UDP仅仅是尽最大的努力进行交付，只是做比较初级的检查，比如端头检查，差错检测，往往在传输过程中会出现分组丢失、乱序、重复传输等问题。

应用层协议	应用	传输层协议
SMTP	电子邮件	TCP
TELNET	远程终端接入	
HTTP	万维网	
FTP	文件传输	
DNS	名字转换	UDP
TFTP	文件传输	
RIP	路由选择协议	
BOOTP，DHCP	IP 地址配置	
SNMP	网络管理	
NFS	远程文件服务器	
专用协议	IP 电话	

最后总结一句，虽然 UDP 是不可靠的，但是在一些应用场景下对可靠传输的要求不高的情况下，UDP 协议具有更好的实时性，工作效率要比 TCP 高。同时，UDP 的段结构要比 TCP 的段结构简单，能降低网络开销。

89、例举常见的二层设备和三层设备，并说出他们工作原理的区别？

交换机是二层设备,路由器是三层设备。

二层设备是工作数据链路层的设备。二层交换机可以识别数据包中的 MAC 地址信息，根据 MAC 地址进行转发，并将这些 MAC 地址与对应的端口记录在自己内部的一个地址表中。具体的工作流程如下：

（1） 当交换机从某个端口收到一个数据包，它先读取包头中的源 MAC 地址，这样它就知道源 MAC 地址的机器是连在哪个端口上的；

（2） 再去读取包头中的目的 MAC 地址，并在地址表中查找相应的端口；

（3） 如表中有与这目的 MAC 地址对应的端口，把数据包直接复制到这端口上；

（4） 如表中找不到相应的端口则把数据包广播到所有端口上，当目的机器对源机器回应时，交换机又可以学习一目的 MAC 地址与哪个端口对应，在下次传送数据时就不再需要对所有端口进行广播了。

不断的循环这个过程，对于全网的 MAC 地址信息都可以学习到，二层交换机就是这样建立和维护它自己的地址表。

三层设备是工作在网络层的设备。路由器是最常用的三层设备，利用不同网络的 ID 号（即 IP 地址）来确定数据转发的地址。IP 地址是在软件中实现的，描述的是设备所在的网络，有时这些第三层的地址也称为协议地址或者网络地址。

90、解释下列名词：IP 地址，网关地址，子网掩码。

(1) IP 地址

IP 地址基本概念

IP 地址在网络层将不同的物理网络地址统一到了全球唯一的 IP 地址上（屏蔽物理网络差异），是唯一标识互联网上计算机的逻辑地址（相当于手机号码，可以通过唯一的手机号码找到手机），所以 IP 地址也被称为互联网地址（可见其重要性）。

IP 地址格式

我们目前常用的 IPv4 中规定，IP 地址长度为 32 位二进制，在表示时，一般将 32 位地址拆分为 4 个 8 位二进制，再转为 4 个十进制数表示，每个数字之间用点隔开，如 127.0.0.1（localhost），这种描述方式被称为“点-数表示法”。

IP 地址层次：分为网络号和主机号两个层次。网络号表示主机所属网络，主机号表示主机本身。网络号与主机号的位数与 IP 地址分类有关。

IP 地址分配

IP 地址分配的基本原则是：要为同一网络（子网、网段）内不同主机分配相同的网络号，不同的主机号。

IP 地址常见分类

	第一字节	第二字节	第三字节	第四字节	备注
A类地址	网络号	主机号			用于有大量主机网络
B类地址	网络号		主机号		中等规模网络
C类地址	网络号			主机号	小规模网络
D类地址	未划分网络号与主机号				特殊网络
E类地址	未划分网络号与主机号				特殊网络

类别	高位字节（类别标识）	网络号范围（重点记忆）	默认掩码
A	0	1.0.0.1到126.0.0.0	255.0.0.0
B	10	128.0.0.0到191.255.255.255	255.255.0.0
C	110	192.0.0.0到223.255.255.255	255.255.255.0
D	1110	224.0.0.0到239.255.255.255	
E	11110	未分配	

A 类 IP 地址

一个 A 类 IP 地址由 1 字节的网络地址和 3 字节主机地址组成，网络地址的最高位必须是“0”， 地址范围从 1.0.0.0 到 126.0.0.0。可用的 A 类网络有 126 个，每个网络能容纳 1 亿多个主机。

B 类 IP 地址

一个 B 类 IP 地址由 2 个字节的网络地址和 2 个字节的主机地址组成，网络地址的最高位必须是“10”，地址范围从 128.0.0.0 到 191.255.255.255。可用的 B 类网络有 16382 个，每个网络能容纳 6 万多个主机 。

C 类 IP 地址

一个 C 类 IP 地址由 3 字节的网络地址和 1 字节的主机地址组成，网络地址的最高位必须是“110”。范围从 192.0.0.0 到 223.255.255.255。C 类网络可达 209 万余个，每个网络能容纳 254 个主机。

D 类地址用于多点广播（Multicast）。

D 类 IP 地址第一个字节以“1110”开始，它是一个专门保留的地址。它并不指向特定的网络，目前这一类地址被用在多点广播（Multicast）中。多点广播地址用来一次寻址一组计算机，它标识共享同一协议的一组计算机。

E 类 IP 地址

以“llll0”开始，为将来使用保留。

特殊的 IP 地址

受限的广播地址：32 位全“1”的 IP 地址，只能做目的地址，用于向本网内部所有主机发送数据包（路由器拒绝向外网发送广播，隔离数据包在本网内）。

直接广播地址：网络号全“1”，只能作目的地址，用于向目标网内所有主机发送数据包（路由器接受向外网转发数据包，意为广播在全网有效），如 C 类 IP 所有主机 255.255.255.主机号、B 类 IP 所有主机 255.255.主机号。

本网络本主机地址：32 位全零的 IP 地址对应于当前主机，只能做源地址。

本网特定主机地址：网络号全零的 IP 地址表示本网络内的特定主机（路由器拒绝目的地址为本网特定主机的数据包，意为只能向本网内特定主机发送数据包），只能做目的地址。

环回地址就是网络 ID 为 127 的 IP 地址，用于一台主机的客户端与服务器端通过 TCP\IP 进行通信或者本机进程间通信，一般在自己的主机上进行软件开发测试时候会用到的 localhost(127.0.0.1)就是环回地址。

在 IP 地址 3 种主要类型里，各保留了 3 个区域作为私有地址，其地址范围如下：

A 类地址：10.0.0.0～10.255.255.255

B 类地址：172.16.0.0～172.31.255.255

C 类地址：192.168.0.0～192.168.255.255

(2)、子网掩码

子网掩码 又叫网络掩码、地址掩码

上面我们说到 IP 地址分为网络号与主机号，但是路由如何区分网络号与主机号呢？就需要通过子网掩码。子网掩码必须与 IP 地址结合使用，A、B、C 类的子网掩码分别为 255.0.0.0，255.255.0.0 与 255.255.255.0（网络号字节为 255，主机号字节为 0）。

也就是说给你一个 IP 地址，那么怎么知道它的网络号和主机号各是多少位呢？

如果不指定，就不知道哪些位是网络号、哪些是主机号，这就需要通过子网掩码来实现

子网掩码的重要作用：就是将某个 IP 地址划分成网络地址和主机地址两部分。

子网掩码的位数就是网络的位数。A 类网络的网络位数是 8 位，子网掩码就是 255.0.0.0，B 类网络的网络位数是 16 位，子网掩码是 255.255.0.0，C 类是 24 位，255.255.255.0。

以 C 类 IP 地址 192.168.1.2 为例使用子网掩码划分网络号与主机号。

当然，在计算主机号时，可不用取反子网掩码二进制，直接令其与 IP 地址异或即可。

例 1 不同子网下的主机能否直接通信（是否在同一网络/段下）

假设两个 IP 地址分别是 172.20.0.18 和 172.20.1.16，子网掩码都是 255.255.255.0。

我们可以知道两者的网络标识分别是 172.20.0 和 172.20.1，无法直接通信，也就无法 PING 通。要想能相互通信，需要将子网掩码改成 255.255.0.0

例 2 如何理解 172.20.1.0/26

上文中的 26 代表主机 ID 的掩码地址长度，从前往后有 26 位，即子网掩码的地址是 255.255.255.192。

子网掩码还可以用来将网络划分为更小的子网，将 IP 的两极结构扩充成三级结构，节约地址空间，减轻路由器负担。

子网掩码的划分

如果要将一个网络划分为多个子网，如何确定子网掩码？步骤如下：

第一步：将要划分的子网数目转换为 2 的 m 次方。如果不是恰好是 2 的多少次方，则按照取大原则。

第二步：将上一步确定的幂 m 按照高序占用主机地址前 m 位，再转化为十进制。如 m 为 3，表示主机位中有 3 位被划分为网络标识号占用，因网络标识号都为 1，故如是 C 类地址，主机号对应的字节变为 11100000，转化为十进制后为 224，故子网掩码为 255.255.255.224，如果是 B 类网络，则子网掩码为 255.255.224.0。

(3)、网关

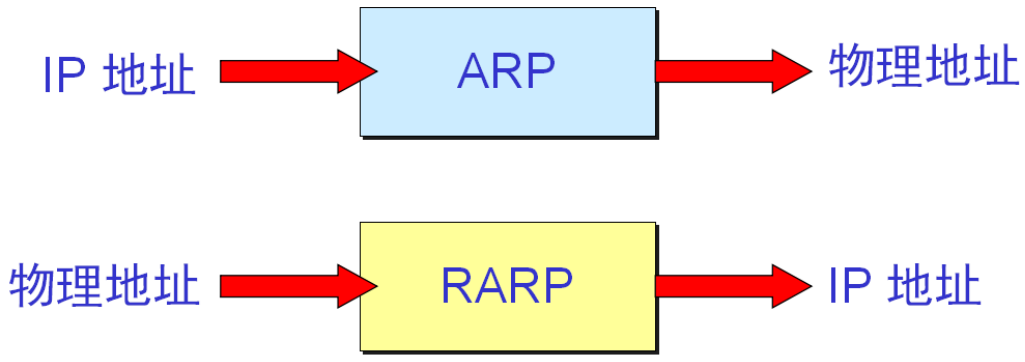
网关（Gateway）又称网间连接器，就是一个网络连接到另一个网络的“关口”。

网关实质上是一个网络通向其他网络的 IP 地址。比如有网络 A 和网络 B，网络 A 的 IP 地址范围为“192.168.1.1~192. 168.1.254”，子网掩码为 255.255.255.0；网络 B 的 IP 地址范围为“192.168.2.1~192.168.2.254”，子网掩码为 255.255.255.0。在没有路由器的情况下，两个网络之间是不能进行 TCP/IP 通信的，即使是两个网络连接在同一台交换机(或集线器)上，TCP/IP 协议也会根据子网掩码(255.255.255.0)判定两个网络中的主机处在不同的网络里。而要实现这两个网络之间的通信，则必须通过网关。如果网络 A 中的主机发现数据包的目的主机不在本地网络中，就把数据包转发给它自己的网关，再由网关转发给网络 B 的网关，网络 B 的网关再转发给网络 B 的某个主机(如附图所示)。网络 B 向网络 A 转发数据包的过程。

所以说，只有设置好网关的 IP 地址，TCP/IP 协议才能实现不同网络之间的相互通信。那么这个 IP 地址是哪台机器的 IP 地址呢？网关的 IP 地址是具有路由功能的设备的 IP 地址，具有路由功能的设备有路由器、启用了路由协议的服务器(实质上相当于一台路由器)、代理服务器(也相当于一台路由器)。

91、什么是 ARP 和 RARP？

(1)、ARP 和 RARP 的基本关系：



(2)、地址解析协议 ARP 的实现过程：

不管网络层使用的是什么协议，在实际网络的链路上传送数据帧时，最终还是必须使用硬件地址。

, 每一个主机都设有一个 ARP 高速缓存(ARP cache)，里面有所在的局域网上的各主机和路由器的 IP 地址到硬件地址的映射表。

ƒ当主机 A 欲向本局域网上的某个主机 B 发送 IP 数据报时，就先在其 ARP 高速缓存中查看有无主机 B 的 IP 地址。如有，就可查出其对应的硬件地址，再将此硬件地址写入 MAC 帧，然后通过局域网将该 MAC 帧发往此硬件地址。

(3)、ARP 高速缓存的作用

为了减少网络上的通信量，主机 A 在发送其 ARP 请求分组时，就将自己的 IP 地址到硬件地址的映射写入 ARP 请求分组。

当主机 B 收到 A 的 ARP 请求分组时，就将主机 A 的这一地址映射写入主机 B 自己的 ARP 高速缓存中。这对主机 B 以后向 A 发送数据报时就更方便了。

(4)、ARP 协议注意到的问题：

ARP 是解决同一个局域网上的主机或路由器的 IP 地址和硬件地址的映射问题。

如果所要找的主机和源主机不在同一个局域网上，那么就要通过 ARP 找到一个位于本局域网上的某个路由器的硬件地址，然后把分组发送给这个路由器，让这个路由器把分组转发给下一个网络。剩下的工作就由下一个网络来做。从 IP 地址到硬件地址的解析是自动进行的，主机的用户对这种地址解析过程是不知道的。 只要主机或路由器要和本网络上的另一个已知 IP 地址的主机或路由器进行通信，ARP 协议就会自动地将该 IP 地址解析为链路层所需要的硬件地址。

(5)、使用 ARP 的四种典型情况

a, 发送方是主机，要把 IP 数据报发送到本网络上的另一个主机。这时用 ARP 找到目的主机的硬件地址。

b, 发送方是主机，要把 IP 数据报发送到另一个网络上的一个主机。这时用 ARP 找到本网络上的一个路由器的硬件地址。剩下的工作由这个路由器来完成。

c, 发送方是路由器，要把 IP 数据报转发到本网络上的一个主机。这时用 ARP 找到目的主机的硬件地址。

d, 发送方是路由器，要把 IP 数据报转发到另一个网络上的一个主机。这时用 ARP 找到本网络上的一个路由器的硬件地址。剩下的工作由这个路由器来完成。

(6)、ARP 协议工作流程

step1: 首先，每台主机都会在自己的 ARP 缓冲区(ARP Cache)中建立一个 ARP 列表，以表示 IP 地址和 MAC 地址的对应关系。

step2: 当源主机需要将一个数据包要发送到目的主机时，会首先检查自己 ARP 列表中是否存在该 IP 地址对应的 MAC 地址，如果有，就直接将数据包发送到这个 MAC 地址；如果没有，就向本地网段发起一个 ARP 请求的广播包，查询此目的主机对应的 MAC 地址。此 ARP 请求数据包里包括源主机的 IP 地址、硬件地址、以及目的主机的 IP 地址。

step3: 网络中所有的主机收到这个 ARP 请求后，会检查数据包中的目的 IP 是否和自己的 IP 地址一致。如果不相同就忽略此数据包；如果相同，该主机首先将发送端的 MAC 地址和 IP 地址添加到自己的 ARP 列表中，如果 ARP 表中已经存在该 IP 的信息，则将其覆盖，然后给源主机发送一个 ARP 响应数据包，告诉对方自己是它需要查找的 MAC 地址；

step4: 源主机收到这个 ARP 响应数据包后，将得到的目的主机的 IP 地址和 MAC 地址添加到自己的 ARP 列表中，并利用此信息开始数据的传输。如果源主机一直没有收到 ARP 响应数据包，表示 ARP 查询失败。

(7)、逆地址解析协议 RARP ：

逆地址解析协议 RARP 使只知道自己硬件地址的主机能够知道其 IP 地址。

RARP 工作原理：

a, 网络上的每台设备都会有一个独一无二的硬件地址，通常是由设备厂商分配的 MAC 地址。PC1 从网卡上读取 MAC 地址，然后在网络上发送一个 RARP 请求的广播数据包，请求 RARP 服务器回复该 PC 的 IP 地址。

b, RARP 服务器收到了 RARP 请求数据包，为其分配 IP 地址，并将 RARP 回应发送给 PC1。

c, PC1 收到 RARP 回应后，就使用得到的 IP 地址进行通讯。

ARP 和 RARP 请求是广播方式，应答都是单播方式

二. 数据库部分

（一）数据库基础和 SQL 语法

1、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。

2、关系数据库与文件数据库的区别在那里？关系数据库一般适用那些方面？

关系数据库系统文件系统的区别在于：

首先，关系型数据库的整体数据是结构化的，采用关系数据模型来描述，这是它与文件系统的根本区别。（数据模型包括：数据结构，数据操作以及完整性约束条件）

其次，关系数据库系统的共享性高，冗余低可以面向整个系统，而文件系统则具有应用范围的局限性，不易扩展。

第三，关系数据库系统采用两级映射机制保证了数据的高独立性，从而使得程序的编写和数据都存在很高的独立性。这方面是文件系统无法达到的，它只能针对于某一个具体的应用。（两级映射：保证逻辑独立性的外模式/模式映射和保证物理独立性的内模式/模式映射。外模式：用户模式，是数据库用户的局部数据的逻辑结构特征的描述。模式：数据库全体数据的逻辑结构特征的描述。内模式：也就是数据最终的物理存储结构的描述。）

第四，就是关系型数据库系统由统一的 DBMS 进行管理，从而为数据提供了如安全性保护，并发控制，完整性检查和数据库恢复服务。

3、什么是事务？有哪些特性？

事务是作为一个逻辑单元执行的一系列操作，一个逻辑工作单元必须有四个属性，称为 ACID(原子性、一致性、隔离性和持久性)属性，只有这样才能成为一个事务:

原子性：事务必须是原子工作单元;对于其数据修改，要么全都执行，要么全都不执行。

一致性：事务在完成时，必须使所有的数据都保持一致状态。在相关数据库中，所有规则都必须应用于事务的修改，以保持所有数据的完整性。事务结束时，所有的内部数据结构(如 B 树索引或双向链表)都必须是正确的。

隔离性：由并发事务所作的修改必须与任何其它并发事务所作的修改隔离。事务查看数据时数据所处的状态，要么是另一并发事务修改它之前的状态，要么是另一事务修改它之后的状态，事务不会查看中间状态的数据。这称为可串行性，因为它能够重新装载起始数据，并且重播一系列事务，以使数据结束时的状态与原始事务执行的状态相同。

持久性：事务完成之后，它对于系统的影响是永久性的。该修改即使出现系统故障也将一直保持。

4、什么是事务一致性？选择熟悉的数据库实现一个事务处理,如信用卡提款.

事务的一致性：是事务原子性的体现，事务所对应的数据库操作要么成功要么失败没有第三种情况。事务不管是提交成功与否都不能影响数据库数据的一致性状态。

事务：用户定义的一个数据库操作序列，这些操作要么全部成功完成要么全部不做，是一个不可分割的整体。定义事务的 SQL 语句有：BEGIN TRANSACTION，COMMIT，ROLLBACK。

事务的原子性：就是事务所包含的数据库操作要么都做,要么都不做.

事务的隔离性：事务对数据的操作不能够受到其他事务的影响。

事务的持续性：也就是说事务对数据的影响是永久的。

对信用卡提款这一事务而言就是要保证’提取到现金’和’卡帐号余额’的修改要同时成功或失败.

```
BEGIN TRANSACTION
    读取 A 的帐户余额 BALANCE
    BALANCE=BALANCE-AMOUNT 转帐金额
    IF（BALANCE<0）
        THEN ROLLBACK
    ELSE BEGIN
        将 A 的新余额写回
        读取 B 的帐户余额 BALANCEB
        BALANCEB=BALANCEB+AMOUNT 转帐金额
        将 B 的新余额写回
        COMMIT
    END IF
END
```

5、触发器的概念,存储过程的概念。

触发器： 是存储在数据库中的过程，当表被修改（增、删、改）时它隐式地被激发。

存储过程：是数据库语言 SQL 的集合，同样也存储在数据库中，但是他是由其他应用程序来启动运行或者也可以直接运行。

6、触发器的作用和使用规范，触发器里是否可以有 commit 为什么？

触发器是可以由事件来启动运行的，存在于数据库服务器中的一个过程。他的作用：可以实现一般的约束无法完成的复杂约束，从而实现更为复杂的完整性要求。使用触发器并不存在严格的限定，只要用户想在无人工参与的情况

下完成一般的定义约束不可以完成的约束，来保证数据库完整性，那么就可以使用触发器。

由于触发器主要是用来保证数据库的完整性的，所以要创建一个触发器，首先要明确该触发器应该属于那一种（DML，INSTEAD OF，SYSTEM）因为他们各有个的用途；其次就是要确定触发器被触发以后所设计到的数据。 触发器中不可以使用 COMMIT。

7、实现索引的方式？索引的原理？索引的代价？索引的类型？

实现索引的方式有两种：针对一张表的某些字段创建具体的索引,如对oracle: create index 索引名称 on 表名(字段名)；在创建表时为字段建立主键约束或者唯一约束，系统将自动为其建立索引。

索引的原理：根据建立索引的字段建立索引表，存放字段值以及对应记录的物理地址，从而在搜索的时候根据字段值搜索索引表的到物理地址直接访问记录。

引入索引虽然提高了查询速度,但本身占用一定的系统存储容量和系统处理时间,需要根据实际情况进行具体的分析.

（oracle）索引的类型有：B 树索引，位图索引，函数索引等。

8、view 的概念，何时应用？

view 是对表级数据的多角度的透视,适用于对查询安全性、灵活性有一定要求的环境.

9、使用存储过程访问数据库比直接用 SQL 语句访问有哪些优点？

存储过程是预编译过的，执行时无须编译，执行速度更快；存储过程封装了一批 SQL 语句，便于维护数据的完整性与一致性；可以实现代码的复用。

10、数据库中回滚的概念？回滚段有什么作用。

回滚就是在事务提交之前将数据库数据恢复到事务修改之前数据库数据状态。

回滚段就是为回滚提供依据，记录的是事务操作数据库之前的数据或者对应于以前操作的操作，这个内容要根据以前的操作而定。比如说以前事务操作如果是 UPDATE 那么回滚段则存储 UPDATE 以前的数据，如果事务是 DELETE 操作那么存储的则是与之相对应的 INSERT 操作语句，相反如果事务操作是 INSERT 那么记录相应的则是 DELETE 操作了。

11、truncate 和 delete 的区别？

1. truncate 在各种表上无论是大的还是小的都非常快。如果有 rollback 命令 delete 将被撤销，而 truncate 则不会被撤销。

2. truncate 是一个 DDL 语言而 delete 是 DML 语句，向其他所有的 DDL 语言一样，他将被隐式提交，不能对 truncate 使用 rollback 命令。

3. truncate 将重新设置高水平线和所有的索引。在对整个表和索引进行完全浏览时，经过 truncate 操作后的表比 delete 操作后的表要快得多。

4. truncate 不能触发触发器，delete 会触发触发器。

5. 不能授予任何人清空他人的表的权限。

6. 当表被清空后表和表的索引讲重新设置成初始大小，而 delete 则不能。

7. 不能清空父表。

12、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')

13、union 和 union all 有什么不同？

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。

14、数据库分页语句？

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);
```

15、谈谈索引的用法及原理？

索引是若干数据行的关键字的列表，查询数据时，通过索引中的关键字可以快速定位到要访问的记录所在的数据块，从而大大减少读取数据块的 I/O 次数，因此可以显著提高性能。

16、使用索引查询一定能提高查询的性能吗？为什么？

通常，通过索引查询数据比全表扫描要快.但是我们也必须注意到它的代价.索引需要空间来存储,也需要定期维护，每当有记录在表中增减或索引列被修改时,索引本身也会被修改. 这意味着每条记录的 INSERT,DELETE,UPDATE 将为此多付出 4,5 次的磁盘 I/O. 因为索引需要额外的存储空间和处理,那些不必要的索引反而会使查询反应时间变慢.使用索引查询不一定能提高查询性能,索引范围查询(INDEX RANGE SCAN)适用于两种情况:

基于一个范围的检索,一般查询返回结果集小于表中记录数的 30%宜采用;

基于非唯一性索引的检索，索引就是为了提高查询性能而存在的,如果在查询中索引没有提高性能,只能说是用错了索引,或者讲是场合不同

17、绑定变量是什么？绑定变量有什么优缺点？

绑定变量是指在 SQL 语句中使用变量，改变变量的值来改变 SQL 语句的执行结果。

优点：使用绑定变量，可以减少 SQL 语句的解析，能减少数据库引擎消耗在 SQL 语句解析上的资源。提高了编程效率和可靠性。减少访问数据库的次数，就能实际上减少 ORACLE 的工作量。

缺点：经常需要使用动态 SQL 的写法，由于参数的不同，可能 SQL 的执行效率不同;

绑定变量是相对文本变量来讲的,所谓文本变量是指在 SQL 直接书写查询条件， 这样的 SQL 在不同条件下需要反复解析,绑定变量是指使用变量来代替直接书写条件，查询 bind value 在运行时传递，然后绑定执行。

优点是减少硬解析,降低 CPU 的争用,节省 shared_pool

缺点是不能使用 histogram,sql 优化比较困难

18、日志的作用是什么？

日志文件（Log File）记录所有对数据库数据的修改，主要是保护数据库以防止故障,以及恢复数据时使用。其特点如下：

1. 每一个数据库至少包含两个日志文件组。每个日志文件组至少包含两个日志文件成员。
2. 日志文件组以循环方式进行写操作。
3. 每一个日志文件成员对应一个物理文件。
4. 记录数据库事务,最大限度地保证数据的一致性与安全性

重做日志文件：含对数据库所做的更改记录，这样万一出现故障可以启用数据恢复,一个数据库至少需要两个重做日志文件

归档日志文件：是重做日志文件的脱机副本，这些副本可能对于从介质失败中进行恢复很必要。

19、简要介绍 SQL 语言的特点？

SQL（Structured Query Language,即结构化查询语言，又简称 SQL 语言）在关系型数据库中的地位就犹如英语在世界上的地位一样，它是数据库系统的通用语言，，用户可以利用它几乎同样的语句在不同的数据库系统上执行同样的操作。比如“select * from 数据表名”代表要从某个数据表中取出全部数据，在 Oracle、SQL Server、Foxpro 等关系型数据库中都可以使用这条语句。SQL 已经被 ANSI（美国国家标准化组织）确定为数据库系统的工业标准。

关系型数据库的主要功能都是通过 SQL 语言来实现的。一般来说，SQL 语言按照功能可以分为 4 大类: 数据查询语言(DQL)、数据定义语言（DLL）、数据操作语言（DML）和数据控制语言（DCL）。数据查询语言 DQL 主要用来查询数据，数据定义语言 DDL 主要用来建立、删除和修改数据对象，数据操纵语言 DML 主要完成数据操作的命令如插入删除修改数据等操作，数据控制语言 DCL 主要用来控制对数据库的访问，服务器的关闭、启动等。

SQL 语言集 DQL、DLL、DML、DCL 于一体，可以实现数据库生命同期的全部活动。数据库中的数据可以用 SQL 语言来进行读取、更新、增加和删除记录。SQL 结构比较简单，其命令总数不超过 30 个，其中常用命令包括：CREATE TABLE、ALTER TABLE、DROP TABLE、INSERT、UPDATE、SELECT、DELETE。SQL 语句对大小写不敏感，但其关键词常用大写来表示。

SQL 语言简单易学、风格统一，利用简单的几个英语单词的组合就可以完成所有的功能，几乎可以不加修改地嵌入到如 VB、PB 这样的前端开发平台上，利用前端工具的计算能力和 SQL 的数据库操纵能力，可以快速建立数据库应用程序。

在 SQL 语言中访问数据表是通过“用户名.数据表”的形式来进行的。比如在 Oracle 数据库服务器安装过程中，默认建立有 scott 用户，该用户对 dept 数据表和 emp 数据表有数据查询的权限，因此访问数据表的语句为 select * from scott.emp。当然，如果用户是用 scott 用户本身登录的，则访问数据表的语句可以简化为 select * from emp，实质上是一样的。

20、简要介绍数据库表之间的连接类型及其特点？

连接类型可分为三种：内连接、外连接和交叉连接。

内连接(INNER JOIN)使用比较运算符进行表间某(些)列数据的比较操作，并列出这些表中与连接条件相匹配的数据行。根据所使用的比较方式不同，内连接又分为等值连接、自然连接和不等连接三种。

外连接分为左外连接(LEFT OUTER JOIN 或 LEFT JOIN)、右外连接(RIGHT OUTER JOIN 或 RIGHT JOIN)和全外连接(FULL OUTER JOIN 或 FULL JOIN)三种。与内连接不同的是，外连接不只列出与连接条件相匹配的行，而是列出左表(左外连接时)、右表(右外连接时)或两个表(全外连接时)中所有符合搜索条件的数据行。

交叉连接(CROSS JOIN)没有 WHERE 子句，它返回连接表中所有数据行的笛卡尔积，其结果集合中的数据行数等于第一个表中符合查询条件的数据行数乘以第二个表中符合查询条件的数据行数。

（二）Oracle 数据库

1、介绍一下 Oracle 的体系结构。

逻辑体系结构：块，区，段，表空间
物理体系结构：表空间，三大文件
软件体系结构：SGA,后台进程.

2、简述 Oracle 中 SGA 的组成部分。

系统全局区包括：共享池、重做日志缓存区、数据高速缓存区，大池，JAVA 池。

3、简述 Oracle 的启动和关闭各有多少步骤？

启动：启动实例、装载数据库数据、打开数据库。 关闭：关闭数据库、卸载数据库数据、关闭实例。

4、Oracle 数据库都有哪些类型的文件？

数据文件，控制文件，日志文件，参数文件

5、Oracle 中，你所创建的表空间信息放在哪里？

存放在数据字典中，数据字典内容对应于系统表空间 SYSTEM 表空间。

6、Oracle 的基本数据类型有哪些？

Char()存储定长字符，定义的时候可以不为他指定长度但是如若往里插入值则会出错；varchar2()存储变长字符定义的时候必须指定长度，date 存储时间日期；Number()数字类型，包括整型，浮点型等；clob()大容量字符串；blob()大二进制对象

7、Oracle 数据库的几种物理文件？

1）数据文件
2）控制文件
3）日志文件

8、控制文件都含有哪些信息？

控制文件存放有实例信息（实例名称创建时间等），数据文件和日志文件信息，还有系统运行时记录的系统变更码(SCN)，检查点信息和归档的当前状

态信息等。数据库在加载数据库的时候首先要读取控制文件获得和数据库有关的物理结构信息之后才能够正确加载数据文件和日志文件并打开数据库。

9、表空间如何扩展？并用语句写出？

两种扩展方式：

1. 增加数据文件 alter tablespace tablespace_name add datafile ‘’ xxMB
2. 扩展数据文件大小 alter database datafile ‘’ resize newMB

10、表空间区管理方式？哪种方式现在是推荐使用的？

1. 字典管理方式 extent management dictionary;默认方式
2. 本地管理方式 extent management local[autoallocate/uniform xxmb]<推荐>;

11、分区表的应用？

1. 一个分区表有一个或多个分区，每个分区通过使用范围分区、散列分区、或组合分区分区的行
2. 分区表中的每一个分区为一个段，可各自位于不同的表空间中
3. 对于同时能够使用几个进程进行查询或操作的大型表分区非常有用

12、存储过程的应用，如何既有输入又有输出？

```
Create procedure pro_name (xxxx in/out type; yyyy in/out/inout type; )
is/as zzzz type;
begin
    sqlpro;
    exception exceptionxxxxx;
    commit;
end;
```

13、Oracle 中的异常有哪几类？

Oracle 中有三种类型的异常。预定义的异常 非预定义的异常 用户定义的异常 第二种非预定义的异常是与特定的 oracle 错误关联。

14、Oracle 的优化策略？（对 MySQL 同样适用）

优化的策略一般包括: 内存优化、操作系统优化、数据存储的优化、网络优化等方法

具体到不同的数据库涉及到要调整不同的数据库配置文件、不同的操作系统参数、网络参数等等, 不同的数据库不同。

15、简单描述 tablespace / segment / extent / block 之间的关系

tablespace: 一个数据库划分为一个或多个逻辑单位，该逻辑单位成为表空间;每一个表空间可能包含一个或多个 Segment;

Segments: Segment 指在 tablespace 中为特定逻辑存储结构分配的空间。每一个段是由一个或多个 extent 组成。包括数据段、索引段、回滚段和临时段。

Extents: 一个 extent 由一系列连续的 Oracle blocks 组成.ORACLE 为通过 extent 来给 segment 分配空间。

Data Blocks：Oracle 数据库最小的 I/O 存储单位，一个 data block 对应一个或多个分配给 data file 的操作系统块。

table 创建时,默认创建了一个 data segment,每个 data segment 含有 min extents 指定的 extents 数,每个 extent 据据表空间的存储参数分配一定数量的 blocks

16、回滚段的作用是什么？

回滚段用于保存数据修改前的映象，这些信息用于生成读一致性数据库信息、在数据库恢复和 Rollback 时使用。一个事务只能使用一个回滚段。

事务回滚：当事务修改表中数据的时候，该数据修改前的值（即前影像）会存放在回滚段中，当用户回滚事务（ROLLBACK）时，ORACLE 将会利用回滚段中的数据前影像来将修改的数据恢复到原来的值。

事务恢复：当事务正在处理的时候，例程失败，回滚段的信息保存在 undo 表空间中，ORACLE 将在下次打开数据库时利用回滚来恢复未提交的数据。

读一致性：当一个会话正在修改数据时，其他的会话将看不到该会话未提交的修改。 当一个语句正在执行时，该语句将看不到从该语句开始执行后的未提交的修改（语句级读一致性） 当 ORACLE 执行 SELECT 语句时，ORACLE 依照当前的系统改变号（SYSTEM CHANGE NUMBER-SCN） 来保证任何前于当前 SCN 的未提交的改变不被该语句处理。可以想象：当一个长时间的查询正在执行时， 若其他会话改变了该查询要查询的某个数据块，ORACLE 将利用回滚段的数据前影像来构造一个读一致性视图

17、SGA 主要有那些部分，主要作用是什么？

系统全局区（SGA）:是 ORACLE 为实例分配的一组共享缓冲存储区，用于存放数据库数据和控制信息，以实现对数据库数据的管理和操作。 SGA 主要包括:

1. 共享池(shared pool) ：用来存储最近执行的 SQL 语句和最近使用的数据字典的数据。

2. 数据缓冲区 (database buffer cache)：用来存储最近从数据文件中读写过的数据。

3. 重作日志缓冲区（redo log buffer）：用来记录服务或后台进程对数据库的操作。 另外在 SGA 中还有两个可选的内存结构：

4. Java pool: 用来存储 Java 代码。

5. Large pool: 用来存储不与 SQL 直接相关的大型内存结构。备份、恢复使用。

18、Oracle 系统进程主要有哪些，作用是什么？

数据写进程(DBWR)：负责将更改的数据从数据库缓冲区高速缓存写入数据文件

日志写进程(LGWR)：将重做日志缓冲区中的更改写入在线重做日志文件

系统监控 (SMON): 检查数据库的一致性如有必要还会在数据库打开时启动数据库的恢复

进程监控 (PMON): 负责在一个 Oracle 进程失败时清理资源

检查点进程(CKPT)：负责在每当缓冲区高速缓存中的更改永久地记录在数据库中时,更新控制文件和数据文件中的数据库状态信息。

归档进程 (ARCH)：在每次日志切换时把已满的日志组进行备份或归档

恢复进程 (RECO): 保证分布式事务的一致性,在分布式事务中,要么同时 commit,要么同时 rollback

19、备份如何分类？

逻辑备份：exp/imp 指定表的逻辑备份

物理备份：

热备份:alter tablespace begin/end backup;

冷备份:脱机备份(database shutdown)

RMAN 备份

full backup/incremental backup(累积/差异)

一、物理备份:

物理备份是最主要的备份方式。用于保证数据库在最小的数据库丢失或没有数据丢失的情况下得到恢复。

（1）冷物理

冷物理备份提供了最简单和最直接的方法保护数据库因物理损坏丢失。建议在以下几种情况中使用。

对一个已经存在大最数据量的数据库，在晚间数据库可以关闭，此时应用冷物理备份。 对需对数据库服务器进行升级，（如更换硬盘），此时需要备份数据库信息，并在新的硬盘中恢复这些数据信息，建议采用冷物理备份。

（2）热物理

主要是指备份过程在数据库打开并且用户可以使用的情况下进行。需要执行热物理备份的情况有：

由于数据库性质要求不间断工作，因而此时只能采用热物理备份。 由于备份的要求的时间过长，而数据库只能短时间关闭时。

二、逻辑备份 (EXP/IMP)

逻辑备份用于实现数据库对象的恢复。但不是基于时间点可完全恢复的备份策略。只能作为联机备份和脱机备份的一种补充。

（1）完全逻辑备份

完全逻辑备份是将整个数据库导出到一个数据库的格式文件中，该文件可以在不同的数据库版本、操作系统和硬件平台之间进行移植。

（2）指定表的逻辑备份

通过备份工具，可以将指定的数据库表备份出来，这可以避免完全逻辑备份所带来的时间和财力上的浪费。

20、数据库死锁的预防与解除

死锁发生的条件：

1、资源不能共享，需要只能由一个进程或者线程使用
2、请求且保持，已经锁定的资源自己保持着不释放
3、不剥夺，自给申请到的资源不能被别人剥夺
4、循环等待

想预防死锁，把上面四个条件破坏一个就可以了。

防止死锁的途径就是避免满足死锁条件的情况发生，为此用户需要遵循以下原则。

（1）尽量避免并发地执行涉及到修改数据的语句。

（2）要求每个事务一次就将所有要使用的数据全部加锁，否则就不予执行。

（3）预先规定一个封锁顺序，所有的事务都必须按这个顺序对数据执行封锁。如不同的过程在事务内部对对象的更新执行顺序应尽量保持一致。

（4）每个事务的执行时间不可太长，在业务允许的情况下可以考虑将事务分割成为几个小事务来执行。【比如说把复杂的多表查询分散成多次单表查询】

（5）数据存储空间离散法。数据存储空间离散法是指采取各种手段，将逻辑上在一个表中的数据分散到若干离散的空间上去，以便改善对表的访问性能。主要通过将大表按行或列分解为若干小表，或者按不同的用户群分解两种方法实现。这种方法类似分散“数据热点”，但是确实，如果数据不是太经常被访问，那么死锁就不会太经常发生。

（6）还是类似（1）的，比如有一个修改上百条记录的 update 语句，我们可以修改成每 10 条一个 update 语句，或者干脆就每条记录一个 update 语句。

（7）将经常更新的数据库和查询数据库分开

21、Oracle 中的控制文件什么时候读取？

Oracle 服务器启动时，先启动实例然后再读取数据库的各个文件当然也包括控制文件。也就是说在数据库服务器启动的第二步时读取。

22、Oracle 索引分为哪几类，说出唯一索引和位图索引的概念。

Oracle 索引有 B 树索引，位图索引，函数索引，簇索引等。

唯一索引也是 B 树索引的一种，它要求被索引的字段值不可以重复。在创建的时候使用 B 树算法创建。

位图索引并不是采用像唯一索引那样存储（索引字段值，记录 ROWID）来创建索引段的，而是为每一个唯一的字段值创建一个位图，位图中使用位元来对应一个记录的 ROWID。位元到 ROWID 是通过映射的到的。

23、描述 tablespace 和 datafile 之间的关系？

一个表空间可包含一个或多个数据文件。表空间利用增加或扩展数据文件扩大表空间，表空间的大小为组成该表空间的数据文件大小的和。一个 datafile

只能属于一个表空间; 一个 tablespace 可以有一个或多个 datafile,每个 datafile 只能在一个 tablespace 内, table 中的数据,通过 hash 算法分布在 tablespace 中的各个 datafile 中,tablespace 是逻辑上的概念,datafile 则在物理上储存了数据库的种种对象

24、解释什么是 Oracle Database 11g 系统中关键网格技术？

Oracle 11g 中“g”是 grid 的缩写，表示网格，网格计算是一种技术，能对同源不同计算类型的分布式网络进行无缝地、大规模地扩展，允许不同厂商的计算机共同工作来提供无尽的共享计算机资源。比如，有两台 PC 机，分别来自不同厂商，内存均为 516M，现系统运行要求 1G 的内存，若 PC 机单独执行肯定不能满足要求，可以通过网格计算和集群技术将两台 PC 机组合在一起，以满足系统运行需求。

25、解释 Oracle Database 11g 的基本文件目录的含义？

在 Oracle Database 11g 中,Oracle 的目录结构是由 Oracle_Base 及其子目录 Oracle_Home、admin、flash_recovery_area 和 oradata 目录构成的。为方便讨论，用 Oracle_Base 代表 Oracle 目录树的根，用 Oracle_Home 表示根目录下的主目录。

（1） Oracle_Base 目录

Oracle_Base 代表 Oracle 目录树的根。如果使用 Oracle Universal Installer 进行安装,则 Oracle_Base 是指 system_drive:\oracle\product\10.2.0。

（2）Oracle_Home 目录

Oracle_Home 主目录位于 system_drive:\Oracle_Base 之下，它包含与 Oracle 软件运行有关的子目录和网络文件以及选定的组件等；若在主机上第一次且只安装了 Oracle 数据库,没有其他 Oracle 产品，则使用默认的主目录\db_1；如果在同一台主机的同一个根目录下安装多个产品或安装了第 2 次，则 Oracle_Home 主目录会以 db_n 的形式出现,即 db_2、db_3 等。这也是为什么在 Oracle_Base 目录可以有多个 Oracle_Home 目录的缘故。

Oracle_Home 目录中包括的主要子目录有:

·\BIN——主要包含用于数据库管理的各种命令等。

·\css——与 Oracle Cluster Synchronization 服务有关的文件。

·\dbs——存放数据库服务器端的参数文件 Spfile。

·\demo——存放数据库实例模式的脚本等。

· \install——用于存储 ORACLE 安装后的端口号,iSQL*Plus 以及 Enterprise Manager Database Control 启动并登录的方式等。

·\network\admin——有关监听器 listener.ora 和 sqlnet.ora 以及 tnsnames.ora 等。

·\sysman\config——用于与 Oracle Enterprise Management 有关的端口管理等。

（3）admin 目录

数据库管理文件均存储在 oracle_base\admin\db_name 目录下。各个子目录的主要含义如下:

·\bdump——后台进程跟踪文件。
·\cdump——信息转储文件（core dump）。
·\create——数据库创建文件。
·\exp——数据库导出文件。
·\pfile——初始化参数文件。
·\udump——用户 SQL 追踪文件。

（4）Oradata 目录

数据库文件存储在 Oracle_Base\oradata\db_name 目录下,该目录主要存储数据库的控制文件、数据文件、重做日志文件。其中*.dbf 文件对应数据库中每个表空间；.ctl 文件为控制文件；.log 文件对应重做日志文件组及其成员。

（5）flash_recovery_area 目录

flash_recovery_area 目录存储并管理与备份和恢复有关的文件。它包含系统中每个数据库的子目录。该目录可用于存储与恢复有关的文件,如控制文件、联机重做日志副本、归档日志、闪回日志以及 Oracle 数据库恢复管理器(RMAN)备份等。

26、解释$ORACLE_HOME 和$ORACLE_BASE 的区别？

$ORACLE_BASE 下是 admin 和 product；$ORACLE_HOME 下则是 ORACLE 的命令、连接库、安装助手、listener 等。$ORACLE_HOME 比 $ORACLE_BASE 目录要更深一些，ORACLE_HOME＝ $ORACLE_BASE/product/version。$ORACLE_BASE 是 oracle 的根目录，$ORACLE_HOME 是 oracle 产品的目录。如果装了 2 个版本的 oracle，那么 $ORACLE_BASE 可以是一个，但$ORACLE_HOME 是 2 个。

27、请完成如下 DQL 练习题。

设有关系 EMP（ENO，ENAME，SALARY，DNO），其中各属性的含义依次为职工号、姓名、工资和所在部门号，以及关系 DEPT（DNO，DNAME，MANAGER），其中各属性含义依次为部门号、部门名称、部门经理的职工号。

试用 SQL 语句完成以下查询：

（1）列出各部门中工资不低于 600 元的职工的平均工资。

（2）写出“查询 001 号职工所在部门名称”的查询语句。

（3）请用 SQL 语句将“销售部”的那些工资数额低于 600 的职工的工资上调 10%。

答:

（1） SELECT AVG(SALARY) FROM EMP WHERE SALARY>=600

（2）SELECT B. DNAME FROM EMP A, DEPT B WHERE A. DNO=B. DNO AND A. ENO=’001’

（3）UPDATE A SET A. SALARY=A. SALARY*1.1
FROM EMP A, DEPT B WHERE A. DNO=B. DNO AND B. DNAME =’ 销售部’ AND A. SALARY<600

（三）JDBC 技术

1、注册 Jdbc 驱动程序的三方式？

方法一:

System.setProperty("jdbc.drivers","com.MySQL.jdbc.Driver");

通过系统的属性设置注册驱动,如果要注册多个驱动，则 System.setProperty("jdbc.drivers","com.MySQL.jdbc.Driver:com.oracle.jdbc.Driver");

这种驱动注册方式很少使用。

方法二

Class.forName("com.MySQL.jdbc.Driver");

这种方式不会对具体的驱动类产生依赖（即不用 import 驱动类）。

Class 类的 forName 方法中对参数指定的类进行了装载操作。在这里将 com.MySQL.jdbc.Driver 类装载到 jvm。众所周知，类装载时，将执行被装载类的静态代码块，而 Driver 类有一个静态代码块如下：

```
static{
   try{
      Java.sql.DriverManager.registerDriver(newDriver());
    }catch(SQLExceptione){
        thrownew RuntimeException("can't register driver!");
    }
}
```

所以在装载过程中即完成了 driver 的注册。这也是使用最多的一种注册驱动方式。

方法三

DriverManager.registerDriver(new com.MySQL.jdbc.Driver());

会造成 DriverManager 中产生两个一样的驱动，并会对具体的驱动类产生依赖。

具体来说就是：

a，装载 Driver 类时注册一次驱动,执行此代码时，又注册一次。

b，由于实例化了 com.MySQL.jdbc.Driver.class，导致必须 import 该类，从而具体驱动产生了依赖。不方便扩展代码。

2、用 JDBC 如何调用存储过程？

代码如下：

```
import Java.sql.CallableStatement;
import Java.sql.Connection;
import Java.sql.DriverManager;
import Java.sql.SQLException;
import Java.sql.Types;
```

```
public class JdbcTest {
public static void main(String[] args) {
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();
}
}
}
```

3、JDBC 中的 PreparedStatement 相比 Statement 的好处？

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。

4、Class.forName 的作用，为什么要用？

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。

5、用 JDBC 查询学生成绩单，把主要代码写出来。

```
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){}
}
```

6、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) {
}
```

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。

7、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。

8、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。

（四）MySQL 数据库

1、数据库应用系统设计？

1.规划
2.需求分析
3.概念模型设计
4.逻辑设计
5.物理设计
6.程序编制及调试
7.运行及维护。

2、列出 MySQL 下常用的 SQL 命令？

创建库:
CREATE DATEBASE database_name
查看数据库
SHOW DATABASE
选择数据库
USE database_name
删除数据库
DORP DATABASE database_name
查看支持的引擎
SHOW ENGINES;
查看默认支持的存储引擎
SHOW VARIABLES LIKE ‘storage_engine%’
创建表
CRATE TABLE table_name(
Field_name data_type,
Field_name data_type,
…
Field_name data_type
)
查看表定义
DESC[RIBE] table_name
删除表
DROP TABLE table_name
修改表
ALTER TABLE old_table_name RENAME [TO] new_table_name
增加字段
ALTER TABLE table_name ADD field_name data_type
查看表详详细定义
SHOW CREATE TABLE table_name

表的第一个位置增加字段

ALTER TABLE table_name ADD field_name data_type FIRST

表的指定位置之后增加字段

ALTER TABLE table_name ADD field_name data_type AFTER field_name

删除字段

ALTER TABLE table_name DORP field_name

修改字段

ALTER TABLE table_name MODIFY field_name data_type

修改字段名字

ALTER TABLE table_name CHANGE old_field_name new_field_name old_data_type

同时修改字段的名字和属性

ALTER TABLE table_name CHANGE old_field_name new_field_name new_data_type

修改字段的顺序

ALTER TABLE table_name MODIFY field_name_1 data_type [FIRST]|[AFTER field_name_2]

字段非空约束

Field_name data_type NOT NULL

字段默认值

Field_name data_type DEFAULT default_value

设置唯一约束

1、Field_name data_type UNIQUE

2、CONSTRAINT constraint_name UNIQUE(field_name)

主键约束

Field_name data_type PRIMARY KEY

多字段主键

CONSTRAINT constraint_name PRIMARY KEY (field_name_1, field_name_2, …)

字段值自动增加

Field_name data_type AUTO_INCREMENT

设置外键约束

CONSTRAINT constraint_name FOREIGN KEY(filed_name)
REFERENCES other_table_name(other_field_name)

创建表时创建普通索引

[表示可先项]
|表示选择
table_name(
column_name
INDEX|KEY [index_name](field_name [(index_length)] [ASC|DESC])
)

在已经存在的表上创建索引

1、CREATE INDEX index_name ON table_name (
field_name [(index_length)] [ASC|DESC])
2、ALTER TABLE table_name ADD INDEX|KEY
index_name(field_name [(index_length)] [ASC|DESC])

创建表时创建唯一索引

table_name(
column_name
UNIQUE INDEX|KEY [index_name](
field_name [(index_length)] [ASC|DESC])
)

在已经存在的表上创建唯一索引

1、CREATE UNIQUE INDEX index_name ON table_name (
field_name [(index_length)] [ASC|DESC])
2、ALTER TABLE table_name ADD UNIQUE INDEX|KEY
index_name(field_name [(index_length)] [ASC|DESC])

创建表时创建全文索引

table_name(
column_name
FULLTEXT INDEX|KEY [index_name](
field_name [(index_length)] [ASC|DESC])
)

在已经存在的表上创建全文索引

1、CREATE FULLTEXT INDEX index_name ON table_name (
field_name [(index_length)] [ASC|DESC])
2、ALTER TABLE table_name ADD FULLTEXT INDEX|KEY
index_name(field_name [(index_length)] [ASC|DESC])

创建多列索引

和上面的方法类似
index_name(field_name_1 [(index_length)] [ASC|DESC],
…,
field_name_n [(index_length)] [ASC|DESC])

删除索引

DROP INDEX index_name ON table_name

创建视图

CREATE VIEW view_name AS select_query

查看视图信息

SHOW TABLE STATUS [FROM database_name] [LIKE ‘pattern’]

查看视图设计信息

DESC[RIBE] view_name

删除视图

DROP VIEW view_name[, view_name]

修改视图

1、CREATE OR REPLACE VIEW view_name AS select_query

2、ALTER VIEW view_name AS select_query

创建触发器

CREATE TIRGGER trigger_name

BEFORE|AFTER DELETE|INSERT|UPDATE

ON table_name FOR EACH ROW

Triggle_statement

Triggle_statement:触发器被触发要执行的语句（增、删、改、查等等）

查看触发器

SHOW TRIGGERS

删除触发器

DROP TRIGGER trigger_name

插入数据

INSERT INTO table_name (field_1, field_2, …) VALUES (value_1, value_2, vaule_3, …)

查看 MySQL 表结构的命令，如下:

desc 表名;

show columns from 表名;

describe 表名;

show create table 表名;

use information_schema

select * from columns where table_name=’表名’;

数据库分页查询

select * from userdetail where userid limit 0,20

3、MySQL 数据库引擎种类？

MySQL-5.5.5 开始,InnoDB 作为默认存储引擎，之前是 MyISAM，更早是 ISAM 你能用的数据库引擎取决于 MySQL 在安装的时候是如何被编译的。要添加一个新的引擎，就必须重新编译 MYSQL。在缺省情况下，MYSQL 支持三个引擎：ISAM、MYISAM 和 HEAP。另外两种类型 INNODB 和 BERKLEYDB（BDB），也常常可以使用。

ISAM 是一个定义明确且历经时间考验的数据表格管理方法，它在设计之时就考虑到数据库被查询的次数要远大于更新的次数。因此，ISAM 执行读取操作的速度很快，而且不占用大量的内存和存储资源。ISAM 的两个主要不足之处在于，它不支持事务处理，也不能够容错：如果你的硬盘崩溃了，那么数据文件就无法恢复了。如果你正在把 ISAM 用在关键任务应用程序里，那就必须经常备份你所有的实时数据，通过其复制特性，MYSQL 能够支持这样的备份应用程序。

MYISAM 是 MYSQL 的 ISAM 扩展格式和缺省的数据库引擎(5.5 之前)。除了提供 ISAM 里所没有的索引和字段管理的大 量功能，MYISAM 还使用一种表格锁定的机制，来优化多个并发的读写操作。其代价是你需要经常运行 OPTIMIZE TABLE 命令，来恢复被更新机制所浪费的空间。MYISAM 还有一些有用的扩展，例如用来修复数据库文件的 MYISAMCHK 工具和用来恢复浪费空间的 MYISAMPACK 工具。

MYISAM 强调了快速读取操作，这可能就是为什么 MYSQL 受到了 WEB 开发如此青睐的主要原因：在 WEB 开发中你所进行的大量数据操作都是读取操作。所以，大多数虚拟主机提供商和 INTERNET 平台提供商只允许使用 MYISAM 格式。

HEAP 允许只驻留在内存里的临时表格。驻留在内存使得 HEAP 比 ISAM 和 MYISAM 的速度都快，但是它所管理的数据是不稳定的，而且如果在关机之前没有进行保存，那么所有的数据都会丢失。在数据行被删除的时候，HEAP 也不会浪费大量的空间，HEAP 表格在你需要使用 SELECT 表达式来选择和操控数据的时候非常有用。要记住，用完表格后要删除表格。

INNODB 和 BERKLEYDB（BDB）数据库引擎都是造就 MYSQL 灵活性的技术的直接产品，这项技术就是 MySQL++ API。在使用 MySQL 的时候，你所面对的每一个挑战几乎都源于 ISAM 和 MYIASM 数据库引擎不支持事务处理也不支持外来键。尽管要比 ISAM 和 MYISAM 引擎慢很多，但是 INNODB 和 BDB 包括了对事务处理和外来键的支持，这两点都是前两个引擎所没有的。如前所述，如果你的设计需要这些特性中的一者或者两者，那你就要被迫使用后两个引擎中的一个了。

4、MySQL 锁类型？

根据锁的类型分，可以分为共享锁，排他锁，意向共享锁和意向排他锁。

根据锁的粒度分，又可以分为行锁，表锁。

对于 MySQL 而言，事务机制更多是靠底层的存储引擎来实现，因此，MySQL 层面只有表锁，而支持事务的 innodb 存 储引擎则实现了行锁(记录锁（在行相应的索引记录上的锁）)，gap 锁（是在索引记录间歇上的锁），next-key 锁（是记录锁和在此索引记录之前的 gap 上的锁的结合）。MySQL 的记录锁实质是索引记录的锁，因为 innodb 是索引组织表；gap 锁是索引记录间隙的锁，这种锁只在 RR 隔离级别下有效；next-key 锁是记录锁加上记录之前 gap 锁的组合。MySQL 通过 gap 锁和 next-key 锁实现 RR 隔离级别。

说明：对于更新操作(读不上锁)，只有走索引才可能上行锁；否则会对聚簇索引的每一行上写锁，实际等同于对表上写锁。

若多个物理记录对应同一个索引，若同时访问，也会出现锁冲突；

当表有多个索引时，不同事务可以用不同的索引锁住不同的行，另外 innodb 会同时用行锁对数据记录(聚簇索引)加 锁。

MVCC(多版本并发控制)并发控制机制下，任何操作都不会阻塞读操作，读操作也不会阻塞任何操作，只因为读不上锁。

共享锁：由读表操作加上的锁，加锁后其他用户只能获取该表或行的共享锁，不能获取排它锁，也就是说只能读不能写

排它锁：由写表操作加上的锁，加锁后其他用户不能获取该表或行的任何锁，典型是 MySQL 事务中的更新操作

意向共享锁（IS）：事务打算给数据行加行共享锁，事务在给一个数据行加共享锁前必须先取得该表的 IS 锁。

意向排他锁（IX）：事务打算给数据行加行排他锁，事务在给一个数据行加排他锁前必须先取得该表的 IX 锁。

5、MySQL 支持事务吗？

在缺省模式下，MySQL 是 autocommit 模式的，所有的数据库更新操作都会即时提交，所以在缺省情况下，MySQL 是不支持事务的。 但是如果你的 MySQL 表类型是使用 InnoDB Tables 或 BDB tables 的话，你的 MYSQL 就可以使用事务处理,使用 SET AUTOCOMMIT=0 就可以使 MYSQL 允许在非 autocommit 模式， 在非 autocommit 模式下，你必须使用 COMMIT 来提交你的更改，或者用 ROLLBACK 来回滚你的更改。

示例如下：

```
START TRANSACTION;
    SELECT @A:=SUM(salary) FROM table1 WHERE type=1;
    UPDATE table2 SET summmary=@A WHERE type=1;
COMMIT;
```

6、MySQL 相比于其他数据库有哪些特点？

1、可以处理拥有上千万条记录的大型数据
2、支持常见的 SQL 语句规范
3、可移植行高，安装简单小巧
4、良好的运行效率，有丰富信息的网络支持
5、调试、管理，优化简单（相对其他大型数据库）

7、如何解决 MySQL 数据库中文乱码问题？

1、在数据库安装的时候指定字符集
2、如果在安完了以后可以更改以配置文件
3、建立数据库时候：指定字符集类型
4、建表的时候也指定字符集

8、如何提高 MySQL 的安全性？

1.如果 MYSQL 客户端和服务器端的连接需要跨越并通过不可信任的网络，那么需要使用 ssh 隧道来加密该连接的通信。

2.使用 set password 语句来修改用户的密码，先“MySQL -u root”登陆数据库系统，然后 MySQL> update MySQL.user set password=password(‘newpwd’)，最后执行 flush privileges 就可以了。

3.MySQL 需要提防的攻击有，防偷听、篡改、回放、拒绝服务等，不涉及可用性和容错方面。对所有的连接、查询、其他操作使用基于 acl 即访问控制列表的安全措施来完成。也有一些对 ssl 连接的支持。

4.设置除了 root 用户外的其他任何用户不允许访问 MySQL 主数据库中的 user 表; 加密后存放在 user 表中的加密 后的用户密码一旦泄露，其他人可以随意用该用户名/密码相应的数据库;

5.使用 grant 和 revoke 语句来进行用户访问控制的工作;

6.不要使用明文密码，而是使用 md5()和 sha1()等单向的哈系函数来设置密码;

7.不要选用字典中的字来做密码;

8.采用防火墙可以去掉 50%的外部危险，让数据库系统躲在防火墙后面工作，或放置在 dmz 区域中;

9.从因特网上用 nmap 来扫描 3306 端口，也可用 telnet server_host 3306 的方法测试，不允许从非信任网络中访问数据库服务器的 3306 号 tcp 端口，需要在防火墙或路由器上做设定;

10.为了防止被恶意传入非法参数，例如 where id=234，别人却输入 where id=234 or 1=1 导致全部显示，所以在 web 的表单中使用”或”“来用字符串，在动态 url 中加入%22 代表双引号、%23 代表井号、%27 代表单引号;传递未检查过的值给 MySQL 数据库是非常危险的;

11.在传递数据给 MySQL 时检查一下大小;

12.应用程序需要连接到数据库应该使用一般的用户帐号，开放少数必要的权限给该用户; pagedevidepagedevide

13.在各编程接口(c c++ php perl Java jdbc 等)中使用特定‘逃脱字符’函数;在因特网上使用 MySQL 数据 库时一定少用传输明文的数据，而用 ssl 和 ssh 的加密方式数据来传输;

14.学会使用 tcpdump 和 strings 工具来查看传输数据的安全性，例如 tcpdump -l -i eth0 -w -src or

dst port 3306 strings。以普通用户来启动 MySQL 数据库服务;

15.不使用到表的联结符号，选用的参数 –skip-symbolic-links;

16.确信在 MySQL 目录中只有启动数据库服务的用户才可以对文件有读和写的权限;

17.不许将 process 或 super 权限付给非管理用户，该 MySQLadmin processlist 可以列举出当前执行的查询 文本;super 权限可用于切断客户端连接、改变服务器运行参数状态、控制拷贝复制数据库的服务器;

18.file 权限不付给管理员以外的用户，防止出现 load data ‘/etc/passwd’到表中再用 select 显示出来的问题;

19.如果不相信 dns 服务公司的服务，可以在主机名称允许表中只设置 ip 数字地址;

20.使用 max_user_connections 变量来使 MySQLd 服务进程，对一个指定帐户限定连接数;

21.grant 语句也支持资源控制选项;

22.–local-infile=0 或 1 若是 0 则客户端程序就无法使用 local load data 了，赋权的一个例子 grant insert(user) on MySQL.user to ‘user_name’@’host_name’;若使用–skip-grant-tables 系统将对任何用户的访问不做任何访问控制，但可以用 MySQLadmin flush-privileges 或 MySQLadmin reload 来开启访问控制;默认情况是 show databases 语句对所有用户开放，可以用–skip-show-databases 来关闭掉。

23.碰到 error 1045(28000) access denied for user ‘root’@’localhost’(using password:no)错误时，你需要重新设置密码，具体方法是:先用–skip-grant-tables 参数启动 MySQLd，然后执行 MySQL -u root MySQL,MySQL>update user set password=password(’newpassword’) where user=’root’;MySQL>flush privileges;，最后重新启动 MySQL 就可以了。

9、MySQL 取得当前时间的函数是？格式化日期的函数是？

取得当前时间用 now() 就行。在数据库中格式化时间用 DATE_FORMAT(date, format)。

根据格式串 format 格式化日期或日期时间值 date，返回结果串。

10、你如何确定 MySQL 是否处于运行状态？

Debian 上运行命令 service MySQL status，在 RedHat 上运行命令 service MySQLd status。然后看看输出即可。

11、为什么要创建索引？

对于查询占主要的应用来说，索引显得尤为重要。很多时候性能问题很简单的就是因为我们忘了添加索引而造成的，或者说没有添加更为有效的索引导致。如果不加索引的话，那么查找任何哪怕只是一条特定的数据都会进行一次全表扫描，如果一张表的数据量很大而符合条件的结果又很少，那么不加索引会引起致命的性能下降。但是也不是什么情 况都非得建索引不可，比如性别可能就只有两个值，建索引不仅没什么优势，还会影响到更新速度，这被称为过度索引。

12、什么是复合索引？

比如有一条语句是这样的：select * from users where area=’beijing’ and age=22;

如果我们是在 area 和 age 上分别创建单个索引的话，由于 MySQL 查询每次只能使用一个索引，所以虽然这样已经相对不做索引时全表扫描提高了很多效率，但是如果在 area、age 两列上创建复合索引的话将带来更高的效率。如果我们创建了(area, age, salary)的复合索引，那么其实相当于创建了(area,age,salary)、 (area,age)、(area)三个索引，这被称为最佳左前缀特性。因此我们在创建复合索引时应该将最常用作限制条件的列放在最左边，依次递减。

13、为什么索引不要包含有 NULL 值的列？

只要列中包含有 NULL 值都将不会被包含在索引中，复合索引中只要有一列含有 NULL 值，那么这一列对于此复合索引就是无效的。所以我们在数据库设计时不要让字段的默认值为 NULL。

14、如何使用短索引，好处是什么？

对串列进行索引，如果可能应该指定一个前缀长度。例如，如果有一个 CHAR(255)的列，如果在前 10 个或 20 个字符内，多数值是惟一的，那么就不要对整个列进行索引。短索引不仅可以提高查询速度而且可以节省磁盘空间和 I/O 操作。

15、排序的索引问题？

MySQL 查询只使用一个索引，因此如果 where 子句中已经使用了索引的话，那么 order by 中的列是不会使用索引的。因此数据库默认排序可以符合要求的情况下不要使用排序操作；尽量不要包含多个列的排序，如果需要最好给这些列创建复合索引。

16、like 语句操作？

一般情况下不鼓励使用 like 操作，如果非使用不可，如何使用也是一个问题。like “%aaa%” 不会使用索引而 like “aaa%”可以使用索引。

17、MySQL 数据库设计数据类型选择需要注意哪些地方？

VARCHAR 和 CHAR 类型，varchar 是变长的，需要额外的 1-2 个字节存储，能节约空间，可能会对性能有帮助。但由于是变长，可能发生碎片，如更新数据；

使用 ENUM（MySQL 的枚举类）代替字符串类型，数据实际存储为整型。

字符串类型

要尽可能地避免使用字符串来做标识符，因为它们占用了很多空间并且通常比整数类型要慢。特别注意不要在 MYISAM 表上使用字符串标识符。MYISAM 默认情况下为字符串使用了压缩索引（Packed Index），这使查找更为缓慢。据测试，使用了压缩索引的 MYISAM 表性能要慢 6 倍。

还要特别注意完全‘随机’的字符串，例如由 MD5（）、SHA1（）、UUID（）产生的。它们产生的每一个新值都会被任意地保存在很大的空间范围内，这会减慢 INSERT 及一些 SELECT 查询。1）它们会减慢 INSERT 查询，因为插入的值会被随机地放入索引中。这会导致分页、随机磁盘访问及聚集存储引擎上的聚集索引碎片。2)它们会减慢 SELECT 查询，因为逻辑上相邻的行会分布在磁盘和内存中的各个地方。3）随机值导致缓存对所有类型的查询性能都很差，因为它们会使缓存赖以工作的访问局部性失效。如果整个数据集都变得同样“热”的时候，那么把特定部分的数据缓存到内存中就没有任何的优势了。并且如果工作集不能被装入内存中，缓存就会进行很多刷写的工作，并且会导致很多缓存未命中。

如果保存 UUID 值，就应该移除其中的短横线，更好的办法是使用 UHEX（）把 UUID 值转化为 16 字节的数字，并把它保存在 BINARY（16）列中。

18、MySQL 几种备份方式？（重点）

1、逻辑备份：使用 MySQL 自带的 MySQLdump 工具进行备份。备份成 sql 文件形式。

优点：最大好处是能够与正在运行的 MySQL 自动协同工作，在运行期间可以确保备份是当时的点，它会自动将对应操作的表锁定，不允许其他用户修改(只能访问)。可能会阻止修改操作。sql 文件通用方便移植。

缺点：备份的速度比较慢。如果是数据量很多的时候。就很耗时间。如果数据库服务器处在提供给用户服务状态，在这段长时间操作过程中，意味着要锁定表(一般是读锁定，只能读不能写入数据)。那么服务就会影响的。

2、物理备份：直接拷贝 MySQL 的数据目录。

直接拷贝只适用于 myisam 类型的表。这种类型的表是与机器独立的。但实际情况是，你设计数据库的时候不可能全部使用 myisam 类型表。你也不可能因为 myisam 类型表与机器独立，方便移植，于是就选择这种表，这并不是选择它的理由。

缺点：你不能去操作正在运行的 MySQL 服务器(在拷贝的过程中有用户通过应用程序访问更新数据，这样就无法备份当时的数据)可能无法移植到其他机器上去。

3、双机热备份。

MySQL 数据库没有增量备份的机制。当数据量太大的时候备份是一个很大的问题。还好 MySQL 数据库提供了一种主从备份的机制(也就是双机热备)

优点：适合数据量大的时候。现在明白了。大的互联网公司对于 MySQL 数据备份，都是采用热机备份。搭建多台数据库服务器，进行主从复制。

19、想知道一个查询用到了哪个 index,如何查看？

explain 显示了 MySQL 如何使用索引来处理 select 语句以及连接表。可以帮助选择更好的索引和写出更优化的查询语句。使用方法，在 select 语句前加上 explain 就可以了。所以使用 explain 可以查看。

20、数据库不能停机，请问如何备份？ 如何进行全备份和增量备份？

可以使用逻辑备份和双机热备份。

完全备份：完整备份一般一段时间进行一次，且在网站访问量最小的时候，这样常借助批处理文件定时备份。主要是写一个批处理文件在里面写上处理程序的绝对路径然后把要处理的东西写在后面，即完全备份数据库。

增量备份：对 ddl 和 dml 语句进行二进制备份。且 5.0 无法增量备份，5.1 后可以。如果要实现增量备份需要在 my.ini 文件中配置备份路径即可，重启 MySQL 服务器，增量备份就启动了。

21、如何给 MySQL 添加索引？

普通索引 添加 INDEX

ALTER TABLE ‘table_name’ ADD INDEX index_name (‘column’);

主键索引 添加 PRIMARY KEY

ALTER TABLE ‘table_name’ ADD PRIMARY KEY (‘column’);

唯一索引 添加 UNIQUE

ALTER TABLE ‘table_name’ ADD UNIQUE (‘column’);

全文索引 添加 FULLTEXT

ALTER TABLE ‘table_name’ ADD FULLTEXT (‘column’);

多列索引

ALTER TABLE ‘table_name’ ADD INDEX index_name (‘column1’, ‘column2’, ‘column3’)

22、什么情况下使用索引？

表的主关键字

自动建立唯一索引

如 zl_yhjbqk（用户基本情况）中的 hbs_bh（户标识编号）

表的字段唯一约束

ORACLE 利用索引来保证数据的完整性

如 lc_hj（流程环节）中的 lc_bh+hj_sx（流程编号+环节顺序）

直接条件查询的字段

在 SQL 中用于条件约束的字段

如 zl_yhjbqk（用户基本情况）中的 qc_bh（区册编号）

select * from zl_yhjbqk where qc_bh=’7001’

查询中与其它表关联的字段

字段常常建立了外键关系

如 zl_ydcf（用电成份）中的 jldb_bh（计量点表编号）

select * from zl_ydcf a,zl_yhdb b where a.jldb_bh=b.jldb_bh and b.jldb_bh=’540100214511’

查询中排序的字段

排序的字段如果通过索引去访问那将大大提高排序速度

select * from zl_yhjbqk order by qc_bh（建立 qc_bh 索引）

select * from zl_yhjbqk where qc_bh=’7001’ order by cb_sx（建立 qc_bh+cb_sx 索引，注：只是一个索引，其中包括 qc_bh 和 cb_sx 字段）

查询中统计或分组统计的字段

select max(hbs_bh) from zl_yhjbqk

select qc_bh,count(*) from zl_yhjbqk group by qc_bh

23、什么情况下应不建或少建索引？

1) 表记录太少

如果一个表只有 5 条记录，采用索引去访问记录的话，那首先需访问索引表，再通过索引表访问数据表，一般索引表与数据表不在同一个数据块，这种情况下 ORACLE 至少要往返读取数据块两次。而不用索引的情况下 ORACLE 会将所有的数据一次读出，处理速度显然会比用索引快。

如表 zl_sybm（使用部门）一般只有几条记录，除了主关键字外对任何一个字段建索引都不会产生性能优化，实际上如果对这个表进行了统计分析后 ORACLE 也不会用你建的索引，而是自动执行全表访问。如：select * from zl_sybm where sydw_bh=’5401’（对 sydw_bh 建立索引不会产生性能优化）

2) 经常插入、删除、修改的表

对一些经常处理的业务表应在查询允许的情况下尽量减少索引，如 zl_yhbm，gc_dfss，gc_dfys，gc_fpdy 等业务表。

3) 数据重复且分布平均的表字段

假如一个表有 10 万行记录，有一个字段 A 只有 T 和 F 两种值，且每个值的分布概率大约为 50%，那么对这种表 A 字段建索引一般不会提高数据库的查询速度。

4) 经常和主字段一块查询但主字段索引值比较多的表字段

如 gc_dfss（电费实收）表经常按收费序号、户标识编号、抄表日期、电费发生年月、操作 标志来具体查询某一笔收款的情况，如果将所有的字段都建在一个索引里那将会增加数据的修改、插入、删除时间，从实际上分析一笔收款如果按收费序号索引就已 经将记录减少到只有几条，如果再按后面的几个字段索引查询将对性能不产生太大的影 响。

24、千万级 MySQL 数据库建立索引的事项及提高性能的手段？

1.对查询进行优化，应尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索引。

2.应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描，如：select id from t where num is null 可以在 num 上设置默认值 0，确保表中 num 列没有 null 值，然后这样查询：select id from t where num=0

3.应尽量避免在 where 子句中使用!=或<>操作符，否则引擎将放弃使用索引而进行全表扫描。

4.应尽量避免在 where 子句中使用 or 来连接条件，否则将导致引擎放弃使用索引而进行全表扫描，如：select id from t where num=10 or num=20 可以这样查询：select id from t where num=10 union all select id from t where num=20

5.in 和 not in 也要慎用，否则会导致全表扫描，如：select id from t where num in(1,2,3) 对于连续的数值，能用 between 就不要用 in 了：select id from t where num between 1 and 3

6.避免使用通配符。下面的查询也将导致全表扫描：select id from t where name like ‘李%’若要提高效率，可以考虑全文检索。

7.如果在 where 子句中使用参数，也会导致全表扫描。因为 SQL 只有在运行时才会解析局部变量，但优化程序不能将访问计划的选择推迟到运行时；它必须在编译时进行选择。然而，如果在编译时建立访问计划，变量的值还是未知的，因而无法作为索引选择的输入项。如下面语句将进行全表扫描：select id from t where num=@num 可以改为强制查询使用索引：select id from t with(index(索引名)) where num=@num

8.应尽量避免在 where 子句中对字段进行表达式操作，这将导致引擎放弃使用索引而进行全表扫描。如：select id from t where num/2=100 应改为:select id from t where num=100*2

9.应尽量避免在 where 子句中对字段进行函数操作，这将导致引擎放弃使用索引而进行全表扫描。如：select id from t where substring(name,1,3)=’abc’ ，name 以 abc 开头的 id 应改为:select id from t where name like ‘abc%’

10.不要在 where 子句中的“=”左边进行函数、算术运算或其他表达式运算，否则系统将可能无法正确使用索引。

11.在使用索引字段作为条件时，如果该索引是复合索引，那么必须使用到该索引中的第一个字段作为条件时才能保证系统使用该索引，否则该索引将不会被使用，并且应尽可能的让字段顺序与索引顺序相一致。

12.不要写一些没有意义的查询，如需要生成一个空表结构：select col1,col2 into #t from t where 1=0 这类代码不会返回任何结果集，但是会消耗系统资源的，应改成这样：create table #t(…)

13.很多时候用 exists 代替 in 是一个好的选择：select num from a where num in(select num from b)用下面的语句替换：select num from a where exists(select 1 from b where num=a.num)

14.并不是所有索引对查询都有效，SQL 是根据表中数据来进行查询优化的，当索引列有大量数据重复时，SQL 查询可能不会去利用索引，如一表中有字段 sex，male、female 几乎各一半，那么即使在 sex 上建了索引也对查询效率起不了作用。

15.索引并不是越多越好，索引固然可以提高相应的 select 的效率，但同时也降低了 insert 及 update 的 效率，因为 insert 或 update 时有可能会重建索引，所以怎样建索引需要慎重考虑，视具体情况而定。一个表的索引数最好不要超过 6 个，若太多则应考虑一些不常使用到的列上建的索引是否有 必要。

16.应尽可能的避免更新 clustered 索引数据列，因为 clustered 索引数据列的顺序就是表记录的物理存储 顺序，一旦该列值改变将导致整个表记录的顺序的调整，会耗费相当大的资源。若应用系统需要频繁更新 clustered 索引数据列，那么需要考虑是否应将该索引建为 clustered 索引。

17.尽量使用数字型字段，若只含数值信息的字段尽量不要设计为字符型，这会降低查询和连接的性能，并会增加存储开销。这是因为引擎在处理查询和连接时会逐个比较字符串中每一个字符，而对于数字型而言只需要比较一次就够了。

18.尽可能的使用 varchar/nvarchar 代替 char/nchar ，因为首先变长字段存储空间小，可以节省存储空间，其次对于查询来说，在一个相对较小的字段内搜索效率显然要高些。

19.任何地方都不要使用 select * from t ，用具体的字段列表代替“*”，不要返回用不到的任何字段。

20.尽量使用表变量来代替临时表。如果表变量包含大量数据，请注意索引非常有限（只有主键索引）。

21.避免频繁创建和删除临时表，以减少系统表资源的消耗。

22.临时表并不是不可使用，适当地使用它们可以使某些例程更有效，例如，当需要重复引用大型表或常用表中的某个数据集时。但是，对于一次性事件，最好使用导出表。

23.在新建临时表时，如果一次性插入数据量很大，那么可以使用 select into 代替 create table，避免造成大量 log ，以提高速度；如果数据量不大，为了缓和系统表的资源，应先 create table，然后 insert。

24.如果使用到了临时表，在存储过程的最后务必将所有的临时表显式删除，先 truncate table ，然后 drop table ，这样可以避免系统表的较长时间锁定。

25.尽量避免使用游标，因为游标的效率较差，如果游标操作的数据超过 1 万行，那么就应该考虑改写。

26.使用基于游标的方法或临时表方法之前，应先寻找基于集的解决方案来解决问题，基于集的方法通常更有效。

27.与临时表一样，游标并不是不可使用。对小型数据集使用 FAST_FORWARD 游标通常要优于其他逐行处理方法，尤其是在必须引用几个表才能获得所需的数据时。在结果集中包括“合计”的例程通常要比使用游标执行的速度快。如果开发时间允许，基于游标的方法和基于集的方法都可以尝试一下，看哪一种方法的效果更好。

28.在所有的存储过程和触发器的开始处设置 SET NOCOUNT ON ，在结束时设置 SET NOCOUNT OFF。无需在执行存储过程和触发器的每个语句后向客户端发送 DONE_IN_PROC 消息。

29.尽量避免大事务操作，提高系统并发能力。

30.尽量避免向客户端返回大数据量，若数据量过大，应该考虑相应需求是否合理。

25、MySQL 在建立索引优化时需要注意哪些问题？

1、创建索引

对于查询占主要的应用来说，索引显得尤为重要。很多时候性能问题很简单的就是因为我们忘了添加索引而造成的，或者说没有添加更为有效的索引导致。如果不加索引的话，那么查找任何哪怕只是一条特定的数据都会进行一次全表扫描，如果一张表的数据量很大而符合条件的结果又很少，那么不加索引会引起致命的性能下降。但是也不是什么情况都非得建索引不可，比如性别可能就只有两个值，建索引不仅没什么优势，还会影响到更新速度，这被称为过度索引。

2、复合索引

比如有一条语句是这样的：select * from users where area=’beijing’ and age=22;

如果我们是在 area 和 age 上分别创建单个索引的话，由于 MySQL 查询每次只能使用一个索引，所以虽然这样已经相对不做索引时全表扫描提高了很多效率，但是如果在 area、age 两列上创建复合索引的话将带来更高的效率。如果我们创建了(area, age,salary)的复合索引，那么其实相当于创建了(area,age,salary)、(area,age)、(area)三个索引，这被称为最佳左前缀特性。因此我们在创建复合索引时应该将最常用作限制条件的列放在最左边，依次递减。

3、索引不会包含有 NULL 值的列

只要列中包含有 NULL 值都将不会被包含在索引中，复合索引中只要有一列含有 NULL 值，那么这一列对于此复合索引就是无效的。所以我们在数据库设计时不要让字段的默认值为 NULL。

4、使用短索引

对串列进行索引，如果可能应该指定一个前缀长度。例如，如果有一个 CHAR(255)的 列，如果在前 10 个或 20 个字符内，多数值是惟一的，那么就不要对整个列进行索引。短索引不仅可以提高查询速度而且可以节省磁盘空间和 I/O 操作。

5、排序的索引问题

MySQL 查询只使用一个索引，因此如果 where 子句中已经使用了索引的话，那么 order by 中的列是不会使用索引的。因此数据库默认排序可以符合要求的情况下不要使用排序操作；尽量不要包含多个列的排序，如果需要最好给这些列创建复合索引。

6、like 语句操作

一般情况下不鼓励使用 like 操作，如果非使用不可，如何使用也是一个问题。like “%aaa%” 不会使用索引而 like “aaa%”可以使用索引。

7、不要在列上进行运算

select * from users where YEAR(adddate)

8、不使用 NOT IN 和操作

NOT IN 和操作都不会使用索引将进行全表扫描。NOT IN 可以 NOT EXISTS 代替，id != 3 则可使用 id>3 or id < 3

26、数据库性能下降，想找到哪些 SQL 耗时较长该如何操作？my.cnf 里如何配置？

1、show processlist;

2、select * from information_schema.processlist ;

3、可以在[MySQLd]中添加如下：

log =/var/log/MySQL.log

如果需要监控慢查询可以添加如下内容：

log-slow-queries = /var/log/slowquery.log

long_query_time = 1

27、什么是聚集索引？

术语“聚集”指实际的数据行和相关的键值都保存在一起。每个表只能有一个聚集索引。但是，覆盖索引可以模拟多个聚集索引。存储引擎负责实现索引，因此不是所有的存储索引都支持聚集索引。当前，SolidDB 和 InnoDB 是唯一支持聚集索引的存储引擎。

优点：

可以把相关数据保存在一起。这样从磁盘上提取几个页面的数据就能把某个用户的数据全部抓取出来。如果没有使用聚集，读取每个数据都会访问磁盘。

数据访问快。聚集索引把索引和数据都保存到了同一棵 B-TREE 中，因此从聚集索引中取得数据通常比在非聚集索引进行查找要快。

缺点：

聚集能最大限度地提升 I/O 密集负载的性能。如果数据能装入内存，那么其顺序也就无所谓了。这样聚集就没有什么用处。

插入速度严重依赖于插入顺序。更新聚集索引列是昂贵的，因为强制 InnoDB 把每个更新的行移到新的位置。

建立在聚集索引上的表在插入新行，或者在行的主键被更新，该行必须被移动的时候会进行分页。

聚集表可会比全表扫描慢，尤其在表存储得比较稀疏或因为分页而没有顺序存储的时候。

第二（非聚集）索引可能会比预想的大，因为它们的叶子节点包含了被引用行的主键列。第二索引访问需要两次索引查找，而不是一次。 InnoDB 的第二索引叶子节点包含了主键值作为指向行的“指针”，而不是“行指针”。 这种策略减少了在移动行或数据分页的时候索引的维护工作。使用行的主键值作为指针使得索引变得更大，但是这意味着 InnoDB 可以移动行，而无须更新指针。

28、MySQL 索引类型？

索引类型: B-TREE 索引，哈希索引

B-TREE 索引(默认的索引类型)加速了数据访问，因为存储引擎不会扫描整个表得到需要的数据。相反，它从根节点开始。根节点保存了指向子节点的指针，并且存储引擎会根据指针寻找数据。它通过查找节点页中的值找到正确的指针，节点页包含子节点的指针，并且存储引擎会根据指针寻找数据。它通过查找节点页中的值找到正确的指针，节点页包含子节点中值的上界和下界。最后，存储引擎可能无法找到需要的数据，也可能成功地找到包含数据的叶子页面。

例：B-TREE 索引 对于以下类型查询有用。匹配全名、匹配最左前缀、匹配列前缀、匹配范围值、精确匹配一部分并且匹配某个范围中的另一部分；

B-TREE 索引的局限：如果查找没有从索引列的最左边开始，它就没什么用处。不能跳过索引中的列，存储引擎不能优先访问任何在第一个范围条件右边的列。例：如果查询是 where last_name=’Smith’ AND first_name LIKE ‘J%’ AND dob=’1976-12-23’;访问就只能使用索引的头两列，因为 LIKE 是范围条件。

哈希索引建立在哈希表的基础上，它只对使用了索引中的每一列的精确查找有用。对于每一行，存储引擎计算出了被索引列的哈希码，它是一个较小的值，并且有可能和其他行的哈希码不同。它把哈希码保存在索引中，并且保存了一个指向哈希表中每一行的指针。

因为索引只包含了哈希码和行指针，而不是值自身，MYSQL 不能使用索引中的值来避免读取行。

MYSQL 不能使用哈希索引进行排序，因为它们不会按序保存行。

哈希索引不支持部分键匹配，因为它们是由被索引的全部值计算出来的。也就是说，如果在（A，B）两列上有索引，并且 WHERE 子句中只使用了 A，那么索引就不会起作用。

哈希索引只支持使用了= IN（）和<=>的相等比较。它们不能加快范围查询。例如 WHERE price > 100;

访问哈希索引中的数据非常快，除非碰撞率很高。当发生碰撞的时候，存储引擎必须访问链表中的每一个行指针，然后逐行进行数据比较，以确定正确的数据。如果有很多碰撞，一些索引维护操作就有可能会变慢。

29、FULLTEXT 全文索引

即为全文索引，目前只有 MyISAM 引擎支持。其可以在 CREATE TABLE ，ALTER TABLE ，CREATE INDEX 使用，不过目前只有 CHAR、VARCHAR ，TEXT 列上可以创建全文索引。值得一提的是，在数据量较大时候，现将数据放入一个没有全局索引的表中，然后再用 CREATE INDEX 创建 FULLTEXT 索引，要比先为一张表建立 FULLTEXT 然后再将数据写入的速度快很多。FULLTEXT 索引也是按照分词原理建立索引的。西文中，大部分为字母文字，分词可以很方便的按照空格进行分割。中文不能按照这种方式进行分词。MySQL 的中文分词插件 MySQLcft，有了它，就可以对中文进行分词。

30、介绍一下如何优化 MySQL

一、在编译时优化 MySQL

如果你从源代码分发安装 MySQL，要注意，编译过程对以后的目标程序性能有重要的影响，不同的编译方式可能得到类似的目标文件，但性能可能相差很大，因此，在编译安装 MySQL 适应仔细根据你的应用类型选择最可能好的编译选项。这种定制的 MySQL 可以为你的应用提供最佳性能。

技巧：选用较好的编译器和较好的编译器选项，这样应用可提高性能 10-30%。（MySQL 文档如是说）

1.1、使用 pgcc（Pentium GCC)编译器->（使用合适的编译器）

该编译器（http://www.goof.com/pcg/）针对运行在奔腾处理器系统上的程序进行优化，用 pgcc 编译 MySQL 源代码，总体性能可提高 10%。当然如果你的服务器不是用奔腾处理器，就不必用它了，因为它是专为奔腾系统设计的。

1.2、仅使用你想使用的字符集编译 MySQL

MySQL 目前提供多达 24 种不同的字符集，为全球用户以他们自己的语言插入或查看表中的数据。却省情况下，MySQL 安装所有者这些字符集，热然而，最好的选择是指选择一种你需要的。如，禁止除 Latin1 字符集以外的所有其它字符集：

1.3、将 MySQLd 编译成静态执行文件

将 MySQLd 编译成静态执行文件而无需共享库也能获得更好的性能。通过在配置时指定下列选项，可静态编译 MySQLd。

1.4、配置样本

二、调整服务器

三、表类型（MySQL 中表的类型）

很多 MySQL 用户可能很惊讶，MySQL 确实为用户提供 5 种不同的表类型，称为 DBD、HEAP、ISAM、MERGE 和 MyIASM。DBD 归为事务安全类，而其他为非事务安全类。

3.1、事务安全

DBD

Berkeley DB(DBD)表是支持事务处理的表，它提供 MySQL 用户期待已久的功能-事务控制。事务控制在任何数据库系统中都是一个极有价值的功能，因为它们确保一组命令能成功地执行。

3.2、非事务安全

HEAP

HEAP 表是 MySQL 中存取数据最快的表。这是因为他们使用存储在动态内存中的一个哈希索引。另一个要点是如果 MySQL 或服务器崩溃，数据将丢失。

ISAM

ISAM 表是早期 MySQL 版本的缺省表类型，直到 MyIASM 开发出来。建议不要再使用它。

MERGE

MERGE 是一个有趣的新类型，在 3.23.25 之后出现。一个 MERGE 表实际上是一个相同 MyISAM 表的集合，合并成一个表，主要是为了效率原因。这样可以提高速度、搜索效率、修复效率并节省磁盘空间。

MyIASM

这是 MySQL 的缺省表类型(5.5.5 之前)。它基于 IASM 代码，但有很多有用的扩展。MyIASM 比较好的原因：

MyIASM 表小于 IASM 表，所以使用较少资源。

MyIASM 表在不同的平台上二进制层可移植。

更大的键码尺寸，更大的键码上限。

3.3、指定表类型

四、优化工具

MySQL 服务器本身提供了几条内置命令用于帮助优化。

4.1、SHOW

SHOW 还能做更多的事情。它可以显示关于日志文件、特定数据库、表、索引、进程和权限表中有价值的信息。

4.2、EXPLAIN

当你面对 SELECT 语句时，EXPLAIN 解释 SELECT 命令如何被处理。这不仅对决定是否应该增加一个索引，而且对决定一个复杂的 Join 如何被 MySQL 处理都是有帮助的。

4.3、OPTIMIZE

OPTIMIZE 语句允许你恢复空间和合并数据文件碎片，对包含变长行的表进行了大量更新和删除后，这样做特别重要。OPTIMIZE 目前只工作于 MyIASM 和 BDB 表。

31、MySQL 你都修改了那些配置文件来进行优化，具体修改内容？

innodb_buffer_pool_size:这是你安装完 InnoDB 后第一个应该设置的选项。缓冲池是数据和索引缓存的地方：这个值越大越好，这能保证你在大多数的读取操作时使用的是内存而不是硬盘。典型的值是 5-6GB(8GB 内存)，20-25GB(32GB 内存)，100-120GB(128GB 内存)。

innodb_log_file_size：这是 redo 日志的大小。redo 日志被用于确保写操作快速而可靠并且在崩溃时恢复。一直到 MySQL 5.1，它都难于调整，因为一方面你想让它更大来提高性能，另一方面你想让它更小来使得崩溃后更快恢复。幸运的是从 MySQL 5.5 之后，崩溃恢复的性能的到了很大提升，这样你就可以同时拥有较高的写入性能和崩溃恢复性能了。一直到 MySQL 5.5，redo 日志的总尺寸被限定在 4GB(默认可以有 2 个 log 文件)。这在 MySQL 5.6 里被提高。

一开始就把 innodb_log_file_size 设置成 512M(这样有 1GB 的 redo 日志)会使你有充裕的写操作空间。如果你知道你的应用程序需要频繁的写入数据并且你使用的时 MySQL 5.6，你可以一开始就把它这是成 4G。max_connections:如果你经常看到‘Too many connections’错误，是因为 max_connections 的值太低了。这非常常见因为应用程序没有正确的关闭数据库连接，你需要比默认的 151 连接数更大的值。max_connection 值被设高了(例如 1000 或更高)之后一个主要缺陷是当服务器运行 1000 个或更高的活动事务时会变的没有响应。在应用程序里使用连接池或者在 MySQL 里使用进程池有助于解决这一问题。

InnoDB 配置

从 MySQL 5.5 版本开始，InnoDB 就是默认的存储引擎并且它比任何其他存储引擎的使用都要多得多。那也是为什么它需要小心配置的原因。

innodb_file_per_table：这项设置告知 InnoDB 是否需要将所有表的数据和索引存放在共享表空间里 （innodb_file_per_table = OFF）或者为每张表的数据单独放在一个.ibd 文件（innodb_file_per_table = ON）。每张表一个文件允许你在 drop、truncate 或者 rebuild 表时回收磁盘空间。这对于一些高级特性也是有必要的，比如数据压缩。但是它不会带来任何性能收益。你不想让每张表一个文件的主要场景是：有非常多的表（比如 10k+）。

MySQL 5.6 中，这个属性默认值是 ON，因此大部分情况下你什么都不需要做。对于之前的版本你必需在加载数据之前将这个属性设置为 ON，因为它只对新创建的表有影响。

innodb_flush_log_at_trx_commit：默认值为 1，表示 InnoDB 完全支持 ACID 特性。当你的主要关注点是数据安全的时候这个值是最合适的，比如在一个主节点上。但是对于磁盘（读写）速度较慢的系统，它会带来很巨大的开销，因为每次将改变 flush 到 redo 日志都需要额外的 fsyncs。将它的值设置为 2 会导致不太可靠（reliable）因为提交的事务仅仅每秒才 flush 一次到 redo 日志，但对于一些场景是可以接受的，比如对于主节点的备份节点这个值是可以接受的。如果值为 0 速度就更快了，但在系统崩溃时可能丢失一些数据：只适用于备份节点。

innodb_flush_method: 这项配置决定了数据和日志写入硬盘的方式。一般来说，如果你有硬件 RAID 控制器，并且其独立缓存采用 write-back 机制，并有着电池断电保护，那么应该设置配置为 O_DIRECT；否则，大多数情况下应将其设为 fdatasync（默认值）。sysbench 是一个可以帮助你决定这个选项的好工具。

innodb_log_buffer_size: 这项配置决定了为尚未执行的事务分配的缓存。其默认值（1MB）一般来说已经够用了，但是如果你的事务中包含有二进制大对象或者大文本字段的话，这点缓存很快就会被填满并触发额外的 I/O 操作。看看 Innodb_log_waits 状态变量，如果它不是 0，增加 innodb_log_buffer_size。

其他设置

query_cache_size: query cache（查询缓存）是一个众所周知的瓶颈，甚至在并发并不多的时候也是如此。 最佳选项是将其从一开始就停用，设置 query_cache_size = 0（现在 MySQL 5.6 的默认值）并利用其他方法加速查询：优化索引、增加拷贝分散负载或者启用额外的缓存（比如 memcache 或

redis）。如果你已经为你的应用启用了 query cache 并且还没有发现任何问题，query cache 可能对你有用。这是如果你想停用它，那就得小心了。

log_bin：如果你想让数据库服务器充当主节点的备份节点，那么开启二进制日志是必须的。如果这么做了之后，还别忘了设置 server_id 为一个唯一的值。就算只有一个服务器，如果你想做基于时间点的数据恢复，这（开启二进制日志）也是很有用的：从你最近的备份中恢复（全量备份），并应用二进制日志中的修改（增量备份）。二进制日志一旦创建就将永久保存。所以如果你不想让磁盘空间耗尽，你可以用 PURGE BINARY LOGS 来清除旧文件，或者设置 expire_logs_days 来指定过多少天日志将被自动清除。

记录二进制日志不是没有开销的，所以如果你在一个非主节点的复制节点上不需要它的话，那么建议关闭这个选项。

skip_name_resolve：当客户端连接数据库服务器时，服务器会进行主机名解析，并且当 DNS 很慢时，建立连接也会很慢。因此建议在启动服务器时关闭 skip_name_resolve 选项而不进行 DNS 查找。唯一的局限是之后 GRANT 语句中只能使用 IP 地址了，因此在添加这项设置到一个已有系统中必须格外小心。

32、MySQL 调优？（重点）

I 硬件配置优化

Ø CPU 选择：多核的 CPU，主频高的 CPU

Ø 内存：更大的内存

Ø 磁盘选择：更快的转速、RAID、阵列卡，

Ø 网络环境选择：尽量部署在局域网、SCI、光缆、千兆网、双网线提供冗余、0.0.0.0 多端口绑定监听

II 操作系统级优化

Ø 使用 64 位的操作系统，更好的使用大内存。

Ø 设置 noatime,nodiratime

Ø 优化内核参数

Ø 加大文件描述符限制

Ø 文件系统选择 xfs

III MySQL 设计优化

III.1 存储引擎的选择

Ø Myisam：数据库并发不大，读多写少,而且都能很好的用到索引，sql 语句比较简单的应用，TB 数据仓库

Ø Innodb：并发访问大，写操作比较多，有外键、事务等需求的应用，系统内存较大。

III.2 命名规则

Ø 多数开发语言命名规则：比如 MyAdress

Ø 多数开源思想命名规则：my_address

Ø 避免随便命名

III.3 字段类型选择

字段类型的选择的一般原则：

Ø 根据需求选择合适的字段类型，在满足需求的情况下字段类型尽可能小。

Ø 只分配满足需求的最小字符数，不要太慷慨。 原因：更小的字段类型更小的字符数占用更少的内存，占用更少的磁盘空间，占用更少的磁盘 IO，以及占用更少的带宽。

对于 varchar 和 char 的选择要根据引擎和具体情况的不同来选择，主要依据如下原则:

1.如果列数据项的大小一致或者相差不大，则使用 char。

2.如果列数据项的大小差异相当大，则使用 varchar。

3.对于 MyISAM 表，尽量使用 Char，对于那些经常需要修改而容易形成碎片的 myisam 和 isam 数据表就更是如此，它的缺点就是占用磁盘空间。

4.对于 InnoDB 表，因为它的数据行内部存储格式对固定长度的数据行和可变长度的数据行不加区分（所有数据行共用一个表头部分，这个标头部分存放着指向各有关数据列的指针），所以使用 char 类型不见得会比使用 varchar 类型好。事实上，因为 char 类型通常要比 varchar 类型占用更多的空间，所以从减少空间占用量和减少磁盘 i/o 的角度，使用 varchar 类型反而更有利。

5.表中只要存在一个 varchar 类型的字段，那么所有的 char 字段都会自动变成 varchar 类型，因此建议定长和变长的数据分开。

III.4 编码选择

单字节 latin1

多字节 utf8(汉字占 3 个字节，英文字母占用一个字节)如果含有中文字符的话最好都统一采用 utf8 类型，避免乱码的情况发生。

III.5 主键选择原则

注：这里说的主键设计主要是针对 INNODB 引擎

1.能唯一的表示行。

2.显式的定义一个数值类型自增字段的主键，这个字段可以仅用于做主键，不做其他用途。

3.MySQL 主键应该是单列的，以便提高连接和筛选操作的效率。

4.主键字段类型尽可能小，能用 SMALLINT 就不用 INT，能用 INT 就不用 BIGINT。

5.尽量保证不对主键字段进行更新修改，防止主键字段发生变化，引发数据存储碎片，降低 IO 性能。

6.MySQL 主键不应包含动态变化的数据，如时间戳、创建时间列、修改时间列等。

7.MySQL 主键应当有计算机自动生成。

8.主键字段放在数据表的第一顺序。

推荐采用数值类型做主键并采用 auto_increment 属性让其自动增长。

III.6 其他需要注意的地方

Ø NULL OR NOT NULL

尽可能设置每个字段为 NOT NULL，除非有特殊的需求，原因如下：

1.使用含有 NULL 列做索引的话会占用更多的磁盘空间，因为索引 NULL 列需要而外的空间来保存。

2.进行比较的时候，程序会更复杂。

3.含有 NULL 的列比较特殊，SQL 难优化，如果是一个组合索引，那么这个 NULL 类型的字段会极大影响整个索引的效率。

Ø 索引

索引的优点：极大地加速了查询，减少扫描和锁定的数据行数。

索引的缺点：占用磁盘空间，减慢了数据更新速度，增加了磁盘 IO。

添加索引有如下原则：

1 选择唯一性索引。

2.为经常需要排序、分组和联合操作的字段建立索引。

3.为常作为查询条件的字段建立索引。

4.限制索引的数据，索引不是越多越好。

5.尽量使用数据量少的索引，对于大字段可以考虑前缀索引。

6.删除不再使用或者很少使用的索引。

7.结合核心 SQL 优先考虑覆盖索引。

8.忌用字符串做主键。

Ø 反范式设计

适当的使用冗余的反范式设计，以空间换时间有的时候会很高效。

IV MySQL 软件优化

Ø 开启 MySQL 复制，实现读写分离、负载均衡，将读的负载分摊到多个从服务器上，提高服务器的处理能力。

Ø 使用推荐的 GA 版本，提升性能

Ø 利用分区新功能进行大数据的数据拆分

V MySQL 配置优化

注意：全局参数一经设置，随服务器启动预占用资源。

Ø key_buffer_size 参数

MySQL 索引缓冲，如果是采用 myisam 的话要重点设置这个参数，根据（key_reads/key_read_requests）判断

Ø innodb_buffer_pool_size 参数

INNODB 数据、索引、日志缓冲最重要的引擎参数，根据（hit riatos 和 FILE I/O）判断

Ø wait_time_out 参数

线程连接的超时时间，尽量不要设置很大，推荐 10s

Ø max_connections 参数

服务器允许的最大连接数，尽量不要设置太大，因为设置太大的话容易导致内存溢出

Ø thread_concurrency 参数

线程并发利用数量，(cpu+disk)*2,根据(os 中显示的请求队列和 tickets)判断

Ø sort_buffer_size 参数

获得更快的 - ORDER BY,GROUP BY,SELECT DISTINCT,UNION DISTINCT

Ø read_rnd_buffer_size 参数

当根据键进行分类操作时获得更快的 - ORDER BY

Ø join_buffer_size 参数

join 连接使用全表扫描连接的缓冲大小，根据 select_full_join 判断

Ø read_buffer_size 参数

全表扫描时为查询预留的缓冲大小，根据 select_scan 判断

Ø tmp_table_size 参数

临时内存表的设置，如果超过设置就会转化成磁盘表，根据参数(created_tmp_disk_tables)判断

Ø innodb_log_file_size 参数(默认 5M)

记录 INNODB 引擎的 redo log 文件，设置较大的值意味着较长的恢复时间。

Ø innodb_flush_method 参数(默认 fdatasync)

Linux 系统可以使用 O_DIRECT 处理数据文件，避免 OS 级别的 cache，O_DIRECT 模式提高数据文件和日志文件的 IO 提交性能

Ø innodb_flush_log_at_trx_commit(默认 1)

1.0 表示每秒进行一次 log 写入 cache，并 flush log 到磁盘。

2.1 表示在每次事务提交后执行 log 写入 cache，并 flush log 到磁盘。

3.2 表示在每次事务提交后，执行 log 数据写入到 cache，每秒执行一次 flush log 到磁盘。

VI MySQL 语句级优化

1.性能查的读语句，在 innodb 中统计行数,建议另外弄一张统计表，采用 myisam，定期做统计.一般的对统计的数据不会要求太精准的情况下适用。

2.尽量不要在数据库中做运算。

3.避免负向查询和%前缀模糊查询。

4.不在索引列做运算或者使用函数。

5.不要在生产环境程序中使用 select * from 的形式查询数据。只查询需要使用的列。

6.查询尽可能使用 limit 减少返回的行数，减少数据传输时间和带宽浪费。

7.where 子句尽可能对查询列使用函数，因为对查询列使用函数用不到索引。

8.避免隐式类型转换，例如字符型一定要用’’，数字型一定不要使用’’。

9.所有的 SQL 关键词用大写，养成良好的习惯，避免 SQL 语句重复编译造成系统资源的浪费。

10.联表查询的时候，记得把小结果集放在前面，遵循小结果集驱动大结果集的原则。

11.开启慢查询，定期用 explain 优化慢查询中的 SQL 语句。

33、MySQL 是怎么备份的？（重点）

一、备份的目的

做灾难恢复：对损坏的数据进行恢复和还原

需求改变：因需求改变而需要把数据还原到改变以前

测试：测试新功能是否可用

二、备份需要考虑的问题

可以容忍丢失多长时间的数据；

恢复数据要在多长时间内完；

恢复的时候是否需要持续提供服务；

恢复的对象，是整个库，多个表，还是单个库，单个表。

三、备份的类型

1、根据是否需要数据库离线

冷备（cold backup）：需要关 MySQL 服务，读写请求均不允许状态下进行；

温备（warm backup）： 服务在线，但仅支持读请求，不允许写请求；

热备（hot backup）：备份的同时，业务不受影响。

注：

a、这种类型的备份，取决于业务的需求，而不是备份工具

b、MyISAM 不支持热备，InnoDB 支持热备，但是需要专门的工具

2、根据要备份的数据集合的范围

完全备份：full backup，备份全部字符集。

增量备份: incremental backup 上次完全备份或增量备份以来改变了的数据，不能单独使用，要借助完全备份，备份的频率取决于数据的更新频率。

差异备份：differential backup 上次完全备份以来改变了的数据。

建议的恢复策略：

完全+增量+二进制日志

完全+差异+二进制日志

3、根据备份数据或文件

物理备份：直接备份数据文件

优点：备份和恢复操作都比较简单，能够跨 MySQL 的版本，恢复速度快，属于文件系统级别的

建议：不要假设备份一定可用，要测试 MySQL>check tables；检测表是否可用

逻辑备份: 备份表中的数据和代码

优点：恢复简单、备份的结果为 ASCII 文件，可以编辑与存储引擎无关可以通过网络备份和恢复

缺点：备份或恢复都需要 MySQL 服务器进程参与备份结果占据更多的空间，浮点数可能会丢失精度 还原之后，缩影需要重建

四：备份的对象

1、 数据；

2、配置文件；

3、代码：存储过程、存储函数、触发器

4、os 相关的配置文件

5、复制相关的配置

6、二进制日志

五、备份和恢复的实现

1、利用 select into outfile 实现数据的备份与还原。

2、利用 MySQLdump 工具对数据进行备份和还原

3、利用 lvm 快照实现几乎热备的数据备份与恢复

4、基于 Xtrabackup 做备份恢复。

优势：

1、快速可靠的进行完全备份

2、在备份的过程中不会影响到事务

3、支持数据流、网络传输、压缩，所以它可以有效的节约磁盘资源和网络带宽。

4、可以自动备份校验数据的可用性。

34、MySQL 怎么登入？创建数据库 bbb？创建用户密码并授权？

一、创建用户:

CREATE USER 用于创建新的 MySQL 账户。要使用 CREATE USER，您必须拥有 MySQL 数据库的全局 CREATE USER 权限，或拥有 INSERT 权限。对于每个账户，CREATE USER 会在没有权限的 MySQL.user 表中创建一个新记录。如果账户已经存在，则出现错误。

使用自选的 IDENTIFIED BY 子句，可以为账户给定一个密码。user 值和密码的给定方法和 GRANT 语句一 样。特别是，要在纯文本中指定密码，需忽略 PASSWORD 关键词。要把 密码指定为由 PASSWORD()函数返回的混编值，需包含关键字 PASSWORD

The create user command:MySQL> CREATE USER yy IDENTIFIED BY ‘123’;

面建立的用户可以在任何地方登陆。

MySQL> CREATE USER yy@localhost IDENTIFIED BY ‘123’;

二、授权:

命令:GRANT privileges ON databasename.tablename TO ‘username’@’host’

说明: privileges - 用户的操作权限,如 SELECT , INSERT , UPDATE 等.如果要授予所的权限则使

用 ALL.;databasename - 数据库名,tablename-表名,如果要授予该用户对所有数据库和表的相应操

作权限则可用表示, 如.*.

GRANT SELECT, INSERT ON test.user TO ‘pig’@’%’;

GRANT ALL ON . TO ‘pig’@’%’;

注意:用以上命令授权的用户不能给其它用户授权,如果想让该用户可以授权,用以下命令:

GRANT privileges ON databasename.tablename TO ‘username’@’host’ WITH GRANT OPTION;

刷新系统权限表

flush privileges;

三、设置与更改用户密码

命令:SET PASSWORD FOR ‘username’@’host’ = PASSWORD(‘newpassword’);如果是当前登陆用户用 SET PASSWORD = PASSWORD(“newpassword”);

例子:SET PASSWORD FOR ‘pig’@’%’ = PASSWORD(“123456”);

或：update MySQL.user set password=password(‘新密码’) where User=”phplamp” and Host=”localhost”；

四、撤销用户权限

命令: REVOKE privilege ON databasename.tablename FROM ‘username’@’host’；

说明: privilege, databasename, tablename - 同授权部分.

例子: REVOKE SELECT ON . FROM ‘pig’@’%’；

注意: 假如你在给用户’pig’@’%’授权的时候是这样的(或类似的):GRANT SELECT ON test.user TO ‘pig’@’%’, 则在使用 REVOKE SELECT ON . FROM ‘pig’@’%’;命令并不能撤销该用户对 test 数据库中 user 表的 SELECT 操作.相反,如果授权使用的是 GRANT SELECT ON . TO ‘pig’@’%’;则 REVOKE SELECT ON test.user FROM ‘pig’@’%’;命令也不能撤销该用户对 test 数据库中 user 表的 Select 权限.具体信息可以用命令 SHOW GRANTS FOR ‘pig’@’%’; 查看.

五、删除用户

命令: DROP USER ‘username’@’host’；

或：DELETE FROM user WHERE User=”phplamp” and Host=”localhost”；

//删除用户的数据库

MySQL>drop database phplampDB;

35、MySQL 数据库同步怎样实现？

1、安装配置，两台服务器，分别安装好 MySQL。采用单向同步的方式，就是 Master 的数据是主的数据，然后 slave 主动去 Master 哪儿同步数据回来。两台服务器的配置一样，把关键的配置文件拷贝一下，两台服务器做相同的拷贝配置文件操作。

2、配置 Master 服务器，要考虑我们需要同步那个数据库，使用那个用户同步，我们这里为了简单起见，就使用 root 用户进行同步，并且只需要同步数据库 abc。

3、配置 Slave 服务器，我们的 slave 服务器主要是主动去 Master 服务器同步数据回来。

4、测试安装，首先查看一下 slave 的主机日志：检查是否连接正常， 在 Master 查看信息，查看 Master 状态：查看 Master 下 MySQL 进程信息：在 slave 上查看信息：查看 slave 状态：查看 slave 下 MySQL 进程信息：再在 Master 的 abc 库里建立表结构并且插入数据，然后检查 slave 有没有同步这些数据，就能够检查出是否设置成功。

36、查询 MySQL 数据库中用户，密码，权限的命令？

查看 MYSQL 数据库中所有用户

SELECT DISTINCT CONCAT(‘User: ”’,user,”’@”’,host,”’;’) AS query FROM MySQL.user;

查看数据库中具体某个用户的权限

how grants for ‘cactiuser’@’%’;

37、数据库死锁概念？

多数情况下，可以认为如果一个资源被锁定，它总会在以后某个时间被释放。而死锁发生在当多个进程访问同一数据库时，其中每个进程拥有的锁都是其他进程所需的，由此造成每个进程都无法继续下去。简单的说，进程 A 等待进程 B 释放他的资源，B 又等待 A 释放他的资源，这样就互相等待就形成死锁。

虽然进程在运行过程中，可能发生死锁，但死锁的发生也必须具备一定的条件，死锁的发生必须具备以下四个必要条件。

1）互斥条件：指进程对所分配到的资源进行排它性使用，即在一段时间内某资源只由一个进程占用。如果此时还有其它进程请求资源，则请求者只能等待，直至占有资源的进程用毕释放。

2）请求和保持条件：指进程已经保持至少一个资源，但又提出了新的资源请求，而该资源已被其它进程占有，此时请求进程阻塞，但又对自己已获得的其它资源保持不放。

3）不剥夺条件：指进程已获得的资源，在未使用完之前，不能被剥夺，只能在使用完时由自己释放。

4）环路等待条件：指在发生死锁时，必然存在一个进程——资源的环形链，即进程集合{P0，P1，P2，•••，Pn}中的 P0 正在等待一个 P1 占用的资源；P1 正在等待 P2 占用的资源，……，Pn 正在等待已被 P0 占用的资源。

下列方法有助于最大限度地降低死锁：

（1）按同一顺序访问对象。

（2）避免事务中的用户交互。

（3）保持事务简短并在一个批处理中。

（4）使用低隔离级别。

（5）使用绑定连接。

38、union 和 union all 的区别以及使用？

Union 因为要进行重复值扫描，所以效率低。如果合并没有刻意要删除重复行，那么就使用 Union All 两个要联合的 SQL 语句 字段个数必须一样，而且字段类型要“相容”（一致）；

union 和 union all 的区别是,union 会自动压缩多个结果集合中的重复结果，而 union all 则将所有的结果全部显示出来，不管是不是重复。

Union：对两个结果集进行并集操作，不包括重复行，同时进行默认规则的排序；

Union All：对两个结果集进行并集操作，包括重复行，不进行排序；

Intersect：对两个结果集进行交集操作，不包括重复行，同时进行默认规则的排序；

Minus：对两个结果集进行差操作，不包括重复行，同时进行默认规则的排序。

可以在最后一个结果集中指定 Order by 子句改变排序方式。

39、MySQL 的备份命令是什么？

MySQLdump -hhostname -uusername -ppassword databasename > backupfile.sql

备份 MySQL 数据库为带删除表的格式

备份 MySQL 数据库为带删除表的格式，能够让该备份覆盖已有数据库而不需要手动删除原有数据库。

MySQLdump - – add-drop-table -uusername -ppassword databasename > backupfile.sql

直接将 MySQL 数据库压缩备份

MySQLdump -hhostname -uusername -ppassword databasename | gzip > backupfile.sql.gz

备份 MySQL 数据库某个(些)表

MySQLdump -hhostname -uusername -ppassword databasename specific_table1 specific_table2 > backupfile.sql

同时备份多个 MySQL 数据库

MySQLdump -hhostname -uusername -ppassword – databases databasename1 databasename2 databasename3 > multibackupfile.sql

仅仅备份数据库结构

MySQLdump – no-data – databases databasename1 databasename2 databasename3 > structurebackupfile.sql

备份服务器上所有数据库

MySQLdump – all-databases > allbackupfile.sql

还原 MySQL 数据库的命令

MySQL -hhostname -uusername -ppassword databasename < backupfile.sql

还原压缩的 MySQL 数据库

unzip < backupfile.sql.gz | MySQL -uusername -ppassword databasename

将数据库转移到新服务器

MySQLdump -uusername -ppassword databasename | MySQL – host=... -C databasename

40、在 MySQL 服务器运行缓慢的情况下输入什么命令能缓解服务器压力？

第一步 检查系统的状态

通过操作系统的一些工具检查系统的状态，比如 CPU、内存、交换、磁盘的利用率，根据经验或与系统正常时的状态相比对，有时系统表面上看起来看空闲，这也可能不是一个正常的状态，因为 cpu 可能正等待 IO 的完成。除此之外，还应观注那些占用系统资源(cpu、内存)的进程。

1.1 使用 sar 来检查操作系统是否存在 IO 问题

1.2 使用 vmstat 监控内存 cpu 资源

1.3 磁盘 IO 问题，处理方式：做 raid10 提高性能

1.4 网络问题，telnet 一下 MySQL 对外开放的端口，如果不通的话，看看防火墙是否正确设置了。另外，看看 MySQL 是不是开启了 skip-networking 的选项，如果开启请关闭。

第二步 检查 MySQL 参数

2.1 max_connect_errors

2.2 connect_timeout

2.3 skip-name-resolve

2.4 slave-net-timeout=seconds

2.5 master-connect-retry

第三步 检查 MySQL 相关状态值

3.1 关注连接数

3.2 关注下系统锁情况

3.3 关注慢查询（slow query）日志

41、怎么导出表结构？

1.导出整个数据库

MySQLdump -u 用户名 -p 密码 数据库名 > 导出的文件名

C:\Users\jack> MySQLdump -uroot -pMySQL sva_rec > e:\sva_rec.sql

2.导出一个表，包括表结构和数据

MySQLdump -u 用户名 -p 密码 数据库名 表名> 导出的文件名

C:\Users\jack> MySQLdump -uroot -pMySQL sva_rec date_rec_drv> e:\date_rec_drv.sql

3.导出一个数据库结构

C:\Users\jack> MySQLdump -uroot -pMySQL -d sva_rec > e:\sva_rec.sql

4.导出一个表，只有表结构

MySQLdump -u 用户名 -p 密码 -d 数据库名 表名> 导出的文件名

C:\Users\jack> MySQLdump -uroot -pMySQL -d sva_rec date_rec_drv> e:\date_rec_drv.sql

5.导入数据库

常用 source 命令

进入 MySQL 数据库控制台，

如 MySQL -u root -p

MySQL>use 数据库

然后使用 source 命令，后面参数为脚本文件(如这里用到的.sql)

MySQL>source d:wcnc_db.sql

42、正常登入 MySQL 后使用什么命令查看其进程是否正常？

输入 show processlist;

如果有 SUPER 权限，则可以看到全部的线程，否则，只能看到自己发起的线程（这是指，当前对应的 MySQL 帐户运行的线程）。

43、MySQL 远程连接命令？

一、MySQL 连接本地数据库，用户名为“root”，密码“123”（注意：“-p”和“123” 之间不能有空格）

C:>MySQL -h localhost -u root -p123

二、MySQL 连接远程数据库（192.168.0.201），端口“3306”，用户名为“root”，密码“123”

C:>MySQL -h 192.168.0.201 -P 3306 -u root -p123

44、MySQL 主从用什么方式传输日志？

MySQL 复制基于主服务器在二进制日志中跟踪所有对数据库的更改(更新、删除等等)。每个从服务器从主服务器接收主服务器已经记录到其二进制日志的保存的更新，以便从服务器可以对其数据拷贝执行相同的更新。

45、数据库的备份方式？

1、完全备份，这是大多数人常用的方式，它可以备份整个数据库，包含用户表、系统表、索引、视图和存储过程等所有数据库对象。但它需要花费更多的时间和空间，所以，一般推荐一周做一次完全备份。

2、事务日志备份，事务日志是一个单独的文件，它记录数据库的改变，备份的时候只需要复制自上次备份以来对数据库所做的改变，所以只需要很少的时间。为了使数据库具有鲁棒性，推荐每小时甚至更频繁的备份事务日志。

3、差异备份也叫增量备份。它是只备份数据库一部分的另一种方法，它不使用事务日志，相反，它使用整个数据库的一种新映象。它比最初的完全备份小，因为它只包含自上次完全备份以来所改变的数据库。它的优点是存储和恢复速度快。推荐每天做一次差异备份。

4、文件备份，数据库可以由硬盘上的许多文件构成。如果这个数据库非常大，并且一个晚上也不能将它备份完，那么可以使用文件备份每晚备份数据库的一部分。由于一般情况下数据库不会大到必须使用多个文件存储，所以这种备份不是很常用。

46、查看 MySQL 数据库是否支持 innodb？

查看 MySQL 的存储引擎：show plugins;

如何在 MySQL 某个表中随机抽取 10 条记录

1.通过 MYSQL 内置的函数来操作，具体 SQL 代码如下：

SELECT * FROM tablename ORDER BY RAND() LIMIT 10

2.不要将大量的工作给数据库去做，这样会导致数据库在某一集中并发时间内锁死并阻塞。建议通过 PHP 随机生成一下 1-X(总行数)之间的数字，然后将这 10 个随机数字作为查询条件，具体语句如：

SELECT * FROM tablename where ID in (2,8,4,11,12,9,3,1,33)

可能你还要进行重复排除，并且需要在程序中将 10 个值串联并连接进入 SQL 语句中。

47、写出 MySQL 怎么修改密码？

方法一: (适用于管理员或者有全局权限的用户重设其它用户的密码)

进入命令行模式

MySQL -u root -p

MySQL>use MySQL;

MySQL> UPDATE user SET password=PASSWORD(“new password”) WHERE user=’username’;

MySQL> FLUSH PRIVILEGES;

MySQL> quit;

方法二:

MySQL -u root -p

MySQL>use MySQL;

MySQL> SET PASSWORD FOR username=PASSWORD(‘new password’);

MySQL> QUIT

方法三:

MySQLadmin -u root “old password” “new password”

注：new password 请输入你想要设置的密码。

48、MySQL 怎么修复损坏的表？

有两种方法，一种方法使用 MySQL 的 check table 和 repair table 的 sql 语句，另一种方法是使用 MySQL 提供的多个 myisamchk, isamchk 数据检测恢复工具。

49、简单叙述一下 MySQL 的优化？（重点）

1.数据库的设计：尽量把数据库设计的更小的占磁盘空间.

1) 尽可能使用更小的整数类型.(mediumint 就比 int 更合适).

2) 尽可能的定义字段为 not null,除非这个字段需要 null.

3) 如果没有用到变长字段的话比如 varchar,那就采用固定大小的纪录格式比如 char.

4) 表的主索引应该尽可能的短.这样的话每条纪录都有名字标志且更高效.

5) 只创建确实需要的索引。索引有利于检索记录，但是不利于快速保存记录。如果总是要在表的组合字段上做搜索，那么就在这些字段上创建索引。索引的第一部分必须是最常使用的字段.如果总是需要用到很多字段，首先就应该多复制这些字段，使索引更好的压缩。

6) 所有数据都得在保存到数据库前进行处理。

7) 所有字段都得有默认值。

8) 在某些情况下,把一个频繁扫描的表分成两个速度会快好多。在对动态格式表扫描以取得相关记录时，它可能使用更小的静态格式表的情况下更是如此。

2.系统的用途

1) 尽量使用长连接.

2) explain 复杂的 SQL 语句。

3) 如果两个关联表要做比较话，做比较的字段必须类型和长度都一致.

4) LIMIT 语句尽量要跟 order by 或者 distinct.这样可以避免做一次 full table scan.

5) 如果想要清空表的所有纪录,建议用 truncate table tablename 而不是 delete from tablename.

6) 能使用 STORE PROCEDURE 或者 USER FUNCTION 的时候.

7) 在一条 insert 语句中采用多重纪录插入格式.而且使用 load data infile 来导入大量数据，这比单纯的 insert 快好多.

8) 经常 OPTIMIZE TABLE 来整理碎片.

9) 还有就是 date 类型的数据如果频繁要做比较的话尽量保存在 unsigned int 类型比较快。

3.系统的瓶颈

1) 磁盘搜索。并行搜索,把数据分开存放到多个磁盘中，这样能加快搜索时间.

2) 磁盘读写(IO)。可以从多个媒介中并行的读取数据。

3) CPU 周期。数据存放在主内存中.这样就得增加 CPU 的个数来处理这些数据。

4) 内存带宽。当 CPU 要将更多的数据存放到 CPU 的缓存中来的话,内存的带宽就成了瓶颈.

50、如何确定有哪些存储引擎可用？

MySQL> show engines; 显示了可用的数据库引擎的全部名单以及在当前的数据库服务器中是否支持这些引擎。

51、MySQL 数据库设计数据类型选择需要注意哪些地方？（重点）

VARCHAR 和 CHAR 类型，varchar 是变长的，需要额外的 1-2 个字节存储，能节约空间，可能会对性能有帮助。但由于是变长，可能发生碎片，如更新数据；

使用 ENUM 代替字符串类型，数据实际存储为整型。

字符串类型

要尽可能地避免使用字符串来做标识符，因为它们占用了很多空间并且通常比整数类型要慢。特别注意不要在 MYISAM 表上使用字符串标识符。MYISAM 默认情况下为字符串使用了压缩索引（Packed Index），这使查找更为缓慢。据测试，使用了压缩索引的 MYISAM 表性能要慢 6 倍。

还要特别注意完全‘随机’的字符串，例如由 MD5（）、SHA1（）、UUID（）产生的。它们产生的每一个新值都会被任意地保存在很大的空间范围内，这会减慢 INSERT 及一些 SELECT 查询。

1）它们会减慢 INSERT 查询，因为插入的值会被随机地放入索引中。这会导致分页、随机磁盘访问及聚集存储引擎上的聚集索引碎片。

2)它们会减慢 SELECT 查询，因为逻辑上相邻的行会分布在磁盘和内存中的各个地方。

3）随机值导致缓存对所有类型的查询性能都很差，因为它们会使缓存赖以工作的访问局部性失效。如果整个数据集都变得同样“热”的时候，那么把特定部分的数据缓存到内存中就没有任何的优势了。并且如果工作集不能被装入内存中，缓存就会进行很多刷写的工作，并且会导致很多缓存未命中。

如果保存 UUID 值，就应该移除其中的短横线，更好的办法是使用 UHEX（）把 UUID 值转化为 16 字节的数字，并把它保存在 BINARY（16）列中。

52、innodb 的事务与日志的实现方式？

（1）有多少种日志

错误日志：记录出错信息，也记录一些警告信息或者正确的信息

慢查询日志：设置一个阈值，将运行时间超过该值的所有 SQL 语句都记录到慢查询的日志文件中。

二进制日志：记录对数据库执行更改的所有操作

查询日志：记录所有对数据库请求的信息，不论这些请求是否得到了正确的执行。

（2）日志的存放形式

（3）事务是如何通过日志来实现的，说得越深入越好。

在 Innodb 存储引擎中，事务日志是通过 redo 和 innodb 的存储引擎日志缓冲（Innodb log buffer）来实现 的，当开始一个事务的时候，会记录该事务的 lsn(log sequence number)号；当事务执行时，会往 InnoDB 存 储引擎的日志的日志缓存里面插入事务日志；当事务提交时，必须将存储引擎的日志缓冲写入磁盘（通过 innodb_flush_log_at_trx_commit 来控制），也就是写数据前，需要先写日志。这种方式称为“预写日志方 式”， innodb 通过此方式来保证事务的完整性。也就意味着磁盘上存储的数据页和内存缓冲池上面的页是不同步 的，是先写入 redo log，然后写入 data file，因此是一种异步的方式。

隔离性： 通过锁实现

原子性、一致性和持久性是通过 redo 和 undo 来完成的。

53、数据库有几种数据保护方式？

实现数据库安全性控制的常用方法和技术有：用户标识和鉴别；存取控制；视图机制；审计；数据加密；

54、如何查看连接 MySQL 的当前用户？

show full processlist，在 user 字段中查看有哪些用。

三．前端技术（HTML&CSS&JavaScript）

1、判断第二个日期比第一个日期大。

如何用脚本判断用户输入的的字符串是下面的时间格式 2004-11-21 必须要保证用户的输入是此格式，并且是时间，比如说月份不大于 12 等等，另外我需要用户输入两个，并且后一个要比前一个晚，只允许用 JAVASCRIPT，请详细帮助作答，，

//这里可用正则表达式判断提前判断一下格式，然后按下提取各时间字段内容

```
<script type="text/JavaScript">
    window.onload = function()
```

```
{
//这么写是为了实现 js 代码与 html 代码的分离，当我修改 js 时，不能影响 html 代码。
document.getElementById("frm1").onsubmit =
function(){
var d1 = this.d1.value;
var d2 = this.d2.value;
if(!verifyDate (d1)) {alert("第一个日期格式不对");return false;}
if(!verifyDate (d2)) {alert("第二个日期格式不对");return false;}
if(!compareDate(d1,d2)) {alert("第二个日期比第一日期小");return false;}
};
}

function compareDate(d1,d2)
{
var arrayD1 = d1.split("-");
var date1 = new Date(arrayD1[0],arrayD1[1],arrayD1[2]);
var arrayD2 = d2.split("-");
var date2 = new Date(arrayD2[0],arrayD2[1],arrayD2[2]);
if(date1 > date2) return false;
return true;
}

function verifyDate(d)
{
var datePattern = /^\d{4}-(0？[1-9]|1[0-2])-(0？[1-9]|[1-2]\d|3[0-1])$/;
return datePattern.test(d);
}
</script>
<form id="frm1" action="xxx.html">
<input type="text" name="d1" />
<input type="text" name="d2" />
<input type="submit"/>
</form>
```

2、如何使用 CSS 实现 table 颜色改变？

用 table 显示 n 条记录，每 3 行换一次颜色，即 1，2，3 用红色字体，4，5，6 用绿色字体，7，8，9 用红颜色字体。

```
<body>
<table id="tbl">
<tr><td>1</td></tr>
<tr><td>2</td></tr>
<tr><td>3</td></tr>
```

```
<tr><td>4</td></tr>
<tr><td>5</td></tr>
<tr><td>6</td></tr>
<tr><td>7</td></tr>
<tr><td>8</td></tr>
<tr><td>9</td></tr>
<tr><td>10</td></tr>
</table>
</body>
<script type="text/JavaScript">
window.onload=function()
{
var tbl = document.getElementById("tbl");
rows = tbl.getElementsByTagName("tr");
for(i=0;i<rows.length;i++)
{
var j = parseInt(i/3);
if(j%2==0) rows[i].style.backgroundColor="#f00";
else    rows[i].style.backgroundColor="#0f0";
}
}
</script>
```

3、如何使用 JS 进行表单验证？请写出代码。

要求：HTML 的 form 提交之前验证文本框的内容为数字， 否则的话提示并终止提交？。

```
<form onsubmit=’return chkForm(this)’>
<input type="text" name="d1"/>
<input type="submit"/>
</form>
<script type=”text/JavaScript” />
function chkForm(this)
{
var value = thist.d1.value;
var len = value.length;
for(var i=0;i<len;i++)
{
if(value.charAt(i)>"9" || value.charAt(i)<"0")
{
alert("含有非数字字符");
return false;
}
}
```

```
return true;
}
</script>
```

4、请写出用于校验 HTML 文本框中输入的内容全部为数字的 JavaScript 代码。

```
    <input type="text" id="d1" onblur=" chkNumber (this)"/>
<script type=”text/JavaScript” />
function chkNumber(eleText)
{
var value = eleText.value;
var len = value.length;
for(var i=0;i<len;i++)
{
if(value.charAt(i)>"9" || value.charAt(i)<"0")
{
alert("含有非数字字符");
eleText.focus();
break;
}
}
}
</script>
```

除了写完代码，还应该在网页上写出实验步骤和在代码中加入实现思路，让面试官一看就明白你的意图和检查你的结果。

5、说说你用过那些 AJAX 技术和框架，说说它们的区别？

1. jQueryjQuery 是一个轻量级的 JavaScript 库，兼容 CSS3，还兼容各种浏览器。jQuery 使用户能更方便地处理 HTML documents、events、实现动画效果，并且方便地为网站提供 AJAX 交互。

2. MooToolsMooTools 是一个简洁、模块化、面向对象的 JavaScript 库。它能够帮助你更快、更简单地编写可扩展和兼容性强的 JavaScript 代码。Mootools 跟 prototypejs 相类似，语法几乎一样。但它提供的功能要比 prototypejs 多，而且更强大。比如增加了动画特效、拖放操作等等。

3. PrototypePrototype 是 Sam Stephenson 写的一个非常优雅的 JavaScript 基础类库，对 JavaScript 做了大量的扩展，旨在简化动态 Web 应用程序的开发。Prototype 很好的支持 AJAX，国内外有多个基于此类库实现的效果库，也做得很棒。

6、CSS3 有哪些常用选择器？列出 5 个以上。

1.通配符选择器（所有浏览器支持）

通用选择器用*来表示，用来选择所有元素，，也可以选择某个元素下的所有元素。

2.类选择器 （所有浏览器都支持类选择器）

类选择器根据类名来选择，前面以”.”来标志，是以一独立于文档元素的方式来指定样式，使用类选择器之前需要在 html 元素上定义类名，换句话说需要保证类名在 html 标记中存在。

3.元素选择器（所有浏览器支持）

元素选择器（标签名选择器），是 css3 选择器中最常见而且最基本的选择器。元素选择器其实就是文档的元素，如 html,body,p,div 等等下下面例子中选择了 span 元素，并设置了字体颜色为红色。

4.ID 选择器（所有浏览器都支持）

ID 选择器和上面说的类选择器是很相似的，在使用 ID 选择器之前也需要先在 html 文档中加注 ID 名称，这样在样式选择器中才能找到相对应的元素，不同的是 ID 选择器是一个页面中唯一的值，我们在类使用时是在相对应的类名前加上一个“.”号（.className）而 id 选择器是在名称前使用"#"如(#demo)。

ID 选择器有几个地方需要特别注意：

第一：一个文档中一个 id 选择器只充许使用一次，因为 id 在页面中是唯一的；

第二，id 选择器不能像类选择器一样多个合并使用，一个元素只能命名一个 id 名；

第三，可以在不同的文档中使用相同的 id 名，比如说在“test.html”中给 h1 定了“＃important”,也可给“test1.html”中定义 p 的 id 为"#important"，但前提是不管在 test.html 还是 test1.html 中只充许有一个 id 叫"#important"的存在。

5.群组选择器（所有浏览器都支持）

当几个元素样式属性一样时，可以共同调用一个声明，元素之间用逗号分隔。群组选择器是将具有相同样式的元素分组在一起，每个选择器之间使用逗号“，”隔开，这个逗号告诉浏览器，规则中包含多个不同的选择器，如果不有这个逗号，那么所表达的意就完全不同了，省去逗号就成了我们前面所说的后代选择器，这一点大家在使用中千万要小心加小心。

7、简述事件冒泡？

DOM事件流（event flow ）存在三个阶段：事件捕获阶段、处于目标阶段、事件冒泡阶段。

事件捕获（event capturing）：通俗的理解就是，当鼠标点击或者触发dom事件时，浏览器会从根节点开始由外到内进行事件传播，即点击了子元素，如果父元素通过事件捕获方式注册了对应的事件的话，会先触发父元素绑定的事件。

事件冒泡（dubbed bubbling）：与事件捕获恰恰相反，事件冒泡顺序是由内到外进行事件传播，直到根节点。

无论是事件捕获还是事件冒泡，它们都有一个共同的行为，就是事件传播，它就像一跟引线，只有通过引线才能将绑在引线上的鞭炮（事件监听器）引爆，试想一下，如果引线不导火了，那鞭炮就只有一响了！！！

dom标准事件流的触发的先后顺序为：先捕获再冒泡，即当触发dom事件时，会先进行事件捕获，捕获到事件源之后通过事件传播进行事件冒泡。不同的浏览器对此有着不同的实现，IE10及以下不支持捕获型事件，所以就少了一个事件捕获阶段，IE11、Chrome 、Firefox、Safari等浏览器则同时存在。
说到事件冒泡与捕获就不得不提一下两个用于事件绑定的方法 addEventListener、attachEvent。当然还有其它的事件绑定的方式这里不做介绍。

addEventListener(event, listener, useCapture)

参数定义：event---（事件名称，如click，不带on），listener---事件监听函数，useCapture---是否采用事件捕获进行事件捕捉，默认为false，即采用事件冒泡方式

addEventListener在 IE11、Chrome 、Firefox、Safari等浏览器都得到支持。

attachEvent(event,listener)

参数定义：event---（事件名称，如onclick，带on），listener---事件监听函数。attachEvent主要用于IE浏览器，并且仅在IE10及以下才支持，IE11已经废了这个方法了（微软还是挺识趣的，慢慢向标准靠拢）。

8、如何居中一个浮动元素？

方式 1:设置容器的浮动方式为相对定位，然后确定容器的宽高比如宽 500 高 300 的层，然后设置层的外边距。

```
<!DOCTYPEHTML>
<html>
<head>
<styletype="text/css">
        div{
            width: 500px;
            height: 300px;
            margin: -150px 0 0 -250px;
            position: absolute;
            left: 50%;
            top: 50%;
            background-color: yellow;
        }
</style>
</head>
<body>
```

```
<div>
    元素居中
</div>
</body>
</html>
```

方式 2：需要 position:absolute;绝对定位。而层的定位点，使用外补丁 margin 负值的方法。负值的大小为层自身宽度高度除以二。

```
<styletype="text/css">
<!--
div {
position:absolute;
top:50%;
left:50%;
margin:-150px 0 0 -200px;
width:400px;
height:300px;
border:1px solid #008800;
}
-->
</style>
<div>让层垂直居中于浏览器窗口</div>
```

9、什么情况下会碰到跨域问题？有哪些解决方法？

跨域问题是这是浏览器为了安全实施的同源策略导致的，同源策略限制了来自不同源的document、脚本，同源的意思就是两个URL的域名、协议、端口要完全相同。 script标签jsonp跨域、nginx反向代理、node.js中间件代理跨域、后端在头部信息设置安全域名、后端在服务器上设置cors。

10、如何判断一个变量是对象还是数组？

判断数组和对象分别都有好几种方法，其中用prototype.toString.call()兼容性最好。 function isObjArr(variable){ if (Object.prototype.toString.call(value) === "[object Array]") { console.log('value是数组'); }else if(Object.prototype.toString.call(value)==='[object Object]'){//这个方法兼容性好一点 console.log('value是对象'); }else{ console.log('value不是数组也不是对象') } }

11、HTML 中 title 属性和 alt 属性的区别？

这个问题被问了一次，当时我只记得，alt属性是用于img标签的，当图片失效的时候会出现alt属性里面的内容，title用来标记页面的title，当时面试官问我还有没有其他的区别。我。。。 然后我就找了一篇文章来看，涨了点姿势

1. //1.当图片不输出信息的时候，会显示alt信息鼠标放上去没有信息，当图片正常读取，不会出现alt信息

2. // 2.当图片不输出信息的时候，会显示alt信息 鼠标放上去会出现title信息 //当图片正常输出的时候，不会出现alt信息，鼠标放上去会出现title信息

另外还有一些关于title属性的知识： title属性可以用在除了base，basefont，head，html，meta，param，script和title之外的所有标签 title属性的功能是提示。额外的说明信息和非本质的信息请使用title属性。title属性值可以比alt属性值设置的更长 title属性有一个很好的用途，即为链接添加描述性文字，特别是当连接本身并不是十分清楚的表达了链接的目的。

12、标准盒子模型与 IE 怪异盒子模型。

这个问题主要会出现在笔试题上面，比如：
<div style="width:100px;height="100px;border:10px;padding:10px;">

这个盒子在w3c标准盒子模型和IE的怪异盒子模型下面它的宽度分别是多少？ 标准盒子模型：总宽度=content100px+border 10px2+padding 10px2 //140px 怪异盒子模型: 总宽度=content60px+ border 10px2+padding 10px2 //100px ps： box-sizing： content-box || border-box；//css3 box-sizing设置为border-box将使用怪异盒子模型 当怪异盒子的宽度小于border+padding的宽度的时候，content width将变为0，盒子的宽度会被border和padding的总宽度撑开

13、你知道哪些 http 状态码？

1xx：1开头的是信息状态码
2xx：2开头的是请求成功
3xx：3开头的是重定向
4xx：4开头的是客户端错误
5xx：5开头的是服务器错误

14、垂直居中有哪些方法？

单行文本的话可以使用height和line-height设置同一高度。
position+margin：设置父元素:position: relative;，子元素height: 100px; position:absolute;top: 50%; margin: -50px 0 0 0;（定高） position+transform：设置父元素position:relative,子元素：position: absolute;top: 50%;transform: translate(0, -50%);（不定高） 百搭flex布局(ie10+)，设置父元素 display:flex;align-items: center;（不定高）

15、JS 事件流？

一个完整的JS事件流是从window开始，最后回到window的一个过程。事件流被分为三个阶段(1~ 5)捕获过程、(5~ 6)事件触发过程、(6~ 10)冒泡过程。

16、有了解作用域吗？怎么预防作用域污染？

其实网上有很多作用域的文章，参差不齐，个人觉得的话，作用域无非就是js当做对于function函数声明会提升到其他声明语句前执行，以及对于某个 {} 里面的作用域如果找不到某个属性，就会在该 {}上下文当中查找属性，如果还找不到的话，进而类似。作用域污染，无非就是闭包了吧，个人理解。

其实闭包也就是指有权访问另一个函数作用域的函数而已。常用的创建闭包的方法就是在函数内部创建另一个函数。

function a(){ var a; // ... return function(){ // 这里可以引用a函数里面的作用域，也就是可以使用a // 而且a函数作用域无法使用这里的值。 }}

17、什么是原型链？

每一个对象都会在内部链接到另一个对象(该对象的原型对象)，该对象有一个原型 prototype，当访问对象的属性或是方法的时候，不仅仅会在原对象上查找，还会顺着原型链在原型对象的原型链上查找，直到查到 null(所有原型链的顶层)为止。原型是 JavaScript 实现继承的基础，new 关键字做的主要的事情就是将实例对象的__proto__属性指向原型对象的 prototype。

18、图片懒加载与预加载？

图片懒加载的原理就是暂时不设置图片的 src 属性，而是将图片的 url 隐藏起来，比如先写在 data-src 里面，等某些事件触发的时候(比如滚动到底部，点击加载图片)再将图片真实的 url 放进 src 属性里面，从而实现图片的延迟加载

图片预加载，是指在一些需要展示大量图片的网站，实现图片的提前加载。从而提升用户体验。常用的方式有两种，一种是隐藏在 css 的 background 的 url 属性里面，一种是通过 JavaScript 的 Image 对象设置实例对象的 src 属性实现图片的预加载。相关代码如下：

CSS 预加载图片方式：

```
#preload-01 { background: url(http://domain.tld/image-01.png) no-repeat -9999px -9999px; }
#preload-02 { background: url(http://domain.tld/image-02.png) no-repeat -9999px -9999px; }
    #preload-03 { background: url(http://domain.tld/image-03.png) no-repeat -9999px -9999px; }
//JavaScript 预加载图片的方式：
function preloadImg(url) {
    var img = new Image();
    img.src = url;
    if(img.complete) {
        //接下来可以使用图片了
        //do something here
    } else {
        img.onload = function() {
```

```
                //接下来可以使用图片了
                //do something here
            };
        }
        }
```

19、JavaScript 中如何检测一个变量是一个 String 类型？请写出函数实现。

typeof(obj) === "string"
typeof obj === "string"
obj.constructor === String
请用 js 去除字符串空格。
方法一：使用 replace 正则匹配的方法
去除所有空格: str = str.replace(/\s*/g,"");
去除两头空格: str = str.replace(/^\s*|\s*$/g,"");
去除左空格： str = str.replace(/^\s*/, “ ”);
去除右空格： str = str.replace(/(\s*$)/g, "");
str 为要去除空格的字符串，实例如下：

```
var str = " 23 23 ";
var str2 = str.replace(/\s*/g,"");
console.log(str2); // 2323
```

方法二：使用 str.trim()方法
str.trim()局限性：无法去除中间的空格，实例如下：

```
var str = "    xiao   ming    ";
var str2 = str.trim();
console.log(str2);    //xiao   ming
```

同理，str.trimLeft()，str.trimRight()分别用于去除字符串左右空格。
方法三：使用 jQuery,$.trim(str)方法
$.trim(str)局限性：无法去除中间的空格，实例如下：

```
var str = "    xiao   ming    ";
var str2 = $.trim(str)
console.log(str2);    //   xiao   ming
```

20、你如何获取浏览器 URL 中查询字符串中的参数？

测试地址为：http://www.runoob.com/jQuery/misc-trim.html？channelid=12333&name=xiaoming&age=23
实例如下：

```
function showWindowHref(){
    var sHref = window.location.href;
    var args = sHref.split('？');
    if(args[0] == sHref){
        return "";
```

```
    }
    var arr = args[1].split('&');
    var obj = {};
    for(var i = 0;i< arr.length;i++){
        var arg = arr[i].split('=');
        obj[arg[0]] = arg[1];
    }
    return obj;
}
var href = showWindowHref(); // obj
console.log(href['name']); // xiaoming
```

js 字符串操作函数

我这里只是列举了常用的字符串函数，具体使用方法，请参考网址。

concat() - 将两个或多个字符的文本组合起来，返回一个新的字符串。

indexOf() - 返回字符串中一个子串第一处出现的索引。如果没有匹配项，返回 -1 。

charAt() - 返回指定位置的字符。

lastIndexOf() - 返回字符串中一个子串最后一处出现的索引，如果没有匹配项，返回 -1 。

match() - 检查一个字符串是否匹配一个正则表达式。

substr() 函数 -- 返回从 string 的 startPos 位置，长度为 length 的字符串

substring() - 返回字符串的一个子串。传入参数是起始位置和结束位置。

slice() - 提取字符串的一部分，并返回一个新字符串。

replace() - 用来查找匹配一个正则表达式的字符串，然后使用新字符串代替匹配的字符串。

search() - 执行一个正则表达式匹配查找。如果查找成功，返回字符串中匹配的索引值。否则返回 -1 。

split() - 通过将字符串划分成子串，将一个字符串做成一个字符串数组。

length - 返回字符串的长度，所谓字符串的长度是指其包含的字符的个数。

toLowerCase() - 将整个字符串转成小写字母。

toUpperCase() - 将整个字符串转成大写字母。

21、怎样添加、移除、移动、复制、创建和查找节点？

1）创建新节点

createDocumentFragment() //创建一个 DOM 片段

createElement() //创建一个具体的元素

createTextNode() //创建一个文本节点

2）添加、移除、替换、插入

appendChild() //添加

removeChild() //移除

replaceChild() //替换

insertBefore() //插入

3）查找

getElementsByTagName() //通过标签名称
getElementsByName() //通过元素的 Name 属性的值
getElementById() //通过元素 Id，唯一性

22、写出三个使用 this 的典型应用？

（1）、在 html 元素事件属性中使用，如：

```
<input type=”button” onclick=”showInfo(this);” value=”点击一下”/>
```

（2）、构造函数

```
function Animal(name, color) {
    this.name = name;
    this.color = color;
}
```

（3）、input 点击，获取值

```
<input type="button" id="text" value="点击一下" />
<script type="text/JavaScript">
    var btn = document.getElementById("text");
    btn.onclick = function() {
        alert(this.value);      //此处的 this 是按钮元素
    }
</script>
```

(4)、apply()/call()求数组最值

```
var  numbers = [5, 458 , 120 , -215 ];
var  maxInNumbers = Math.max.apply(this, numbers);
console.log(maxInNumbers);  // 458
var maxInNumbers = Math.max.call(this,5, 458 , 120 , -215);
console.log(maxInNumbers);  // 458
```

23、比较 typeof 与 instanceof？

相同点：JavaScript 中 typeof 和 instanceof 常用来判断一个变量是否为空，或者是什么类型的。

typeof 的定义和用法：返回值是一个字符串，用来说明变量的数据类型。
细节：

(1)、typeof 一般只能返回如下几个结果：number,boolean,string,function,object,undefined。

(2)、typeof 来获取一个变量是否存在，如 if(typeof a!="undefined"){alert("ok")}，而不要去使用 if(a) 因为如果 a 不存在（未声明）则会出错。

(3)、对于 Array,Null 等特殊对象使用 typeof 一律返回 object，这正是 typeof 的局限性。

Instanceof 定义和用法：instanceof 用于判断一个变量是否属于某个对象的实例。

实例演示：

```
a instanceof b？ alert("true"):alert("false"); //a 是 b 的实例？ 真:假
var a = new Array();
alert(a instanceof Array);    // true
alert(a instanceof Object)   // true
```

如上，会返回 true，同时 alert(a instanceof Object) 也会返回 true;这是因为 Array 是 object 的子类。

```
function test(){};
var a = new test();
alert(a instanceof test)      // true
```

细节：

(1)、如下，得到的结果为‘N’,这里的 instanceof 测试的 object 是指 js 语法中的 object，不是指 dom 模型对象。

```
if (window instanceof Object){ alert('Y')} else {    alert('N');}    // 'N'
```

24、如何理解闭包？

1、定义和用法：当一个函数的返回值是另外一个函数，而返回的那个函数如果调用了其父函数内部的其它变量，如果返回的这个函数在外部被执行，就产生了闭包。

2、表现形式：使函数外部能够调用函数内部定义的变量。

3、实例如下：

(1)、根据作用域链的规则，底层作用域没有声明的变量，会向上一级找，找到就返回，没找到就一直找，直到 window 的变量，没有就返回 undefined。这里明显 count 是函数内部的 flag2 的那个 count 。

```
var count=10;     //全局作用域 标记为 flag1
function add(){
      var count=0;       //函数全局作用域 标记为 flag2
      return function(){
            count+=1;     //函数的内部作用域
            alert(count);
      }
}
var s = add()
s();//输出 1
s();//输出 2
```

4、变量的作用域

要理解闭包，首先必须理解 JavaScript 特殊的变量作用域。变量的作用域分类：全局变量和局部变量。

特点：

1）函数内部可以读取函数外部的全局变量；在函数外部无法读取函数内的局部变量。

2）函数内部声明变量的时候，一定要使用 var 命令。如果不用的话，你实际上声明了一个全局变量！

5、使用闭包的注意点

1）滥用闭包，会造成内存泄漏：由于闭包会使得函数中的变量都被保存在内存中，内存消耗很大，所以不能滥用闭包，否则会造成网页的性能问题，在 IE 中可能导致内存泄露。解决方法是，在退出函数之前，将不使用的局部变量全部删除。

2）会改变父函数内部变量的值。所以，如果你把父函数当作对象（object）使用，把闭包当作它的公用方法（Public Method），把内部变量当作它的私有属性（private value），这时一定要小心，不要随便改变父函数内部变量的值。

25、什么是跨域？跨域请求资源的方法有哪些？

1、什么是跨域？

由于浏览器同源策略，凡是发送请求 url 的协议、域名、端口三者之间任意一与当前页面地址不同即为跨域。存在跨域的情况：

1) 网络协议不同，如 http 协议访问 https 协议。
2) 端口不同，如 80 端口访问 8080 端口。
3) 域名不同，如 qianduanblog.com 访问 baidu.com。
4) 子域名不同，如 abc.qianduanblog.com 访问 def.qianduanblog.com。域名和域名对应 ip,如 www.a.com 访问 20.205.28.90.

2、跨域请求资源的方法：

(1)、porxy 代理

定义和用法：proxy 代理用于将请求发送给后台服务器，通过服务器来发送请求，然后将请求的结果传递给前端。

实现方法：通过 nginx 代理；

注意点：1、如果你代理的是 https 协议的请求，那么你的 proxy 首先需要信任该证书（尤其是自定义证书）或者忽略证书检查，否则你的请求无法成功。

(2)、CORS 【Cross-Origin Resource Sharing】

定义和用法：是现代浏览器支持跨域资源请求的一种最常用的方式。

使用方法：一般需要后端人员在处理请求数据的时候，添加允许跨域的相关操作。如下：

```
res.writeHead(200, {
    "Content-Type": "text/html; charset=UTF-8",
    "Access-Control-Allow-Origin":'http://localhost',
    'Access-Control-Allow-Methods': 'GET, POST, OPTIONS',
    'Access-Control-Allow-Headers': 'X-Requested-With, Content-Type'
});
```

(3)、jsonp

定义和用法：通过动态插入一个 script 标签。浏览器对 script 的资源引用没有同源限制，同时资源加载到页面后会立即执行（没有阻塞的情况下）。

特点：通过情况下，通过动态创建 script 来读取他域的动态资源，获取的数据一般为 json 格式。

实例如下：

```
<script>
```

```
    function testjsonp(data) {
        console.log(data.name); //  获取返回的结果
    }
</script>
<script>
    var _script = document.createElement('script');
    _script.type = "text/JavaScript";
    _script.src = "http://localhost:8888/jsonp？callback=testjsonp";
    document.head.appendChild(_script);
</script>
```

缺点：

1、这种方式无法发送 post 请求（这里）

2、另外要确定 jsonp 的请求是否失败并不容易，大多数框架的实现都是结合超时时间来判定。

26、谈谈垃圾回收机制方式及内存管理。

回收机制方式

1、定义和用法：垃圾回收机制(GC:Garbage Collection),执行环境负责管理代码执行过程中使用的内存。

2、原理：垃圾收集器会定期（周期性）找出那些不在继续使用的变量，然后释放其内存。但是这个过程不是实时的，因为其开销比较大，所以垃圾回收器会按照固定的时间间隔周期性的执行。

3、实例如下：

```
function fn1() {
    var obj = {name: 'hanzichi', age: 10};
}
function fn2() {
    var obj = {name:'hanzichi', age: 10};
   return obj;
}
var a = fn1();
var b = fn2();
```

fn1 中定义的 obj 为局部变量，而当调用结束后，出了 fn1 的环境，那么该块内存会被 js 引擎中的垃圾回收器自动释放；在 fn2 被调用的过程中，返回的对象被全局变量 b 所指向，所以该块内存并不会被释放。

4、垃圾回收策略：标记清除(较为常用)和引用计数。

标记清除：

定义和用法：当变量进入环境时，将变量标记"进入环境"，当变量离开环境时，标记为："离开环境"。某一个时刻，垃圾回收器会过滤掉环境中的变量，以及被环境变量引用的变量，剩下的就是被视为准备回收的变量。

到目前为止，IE、Firefox、Opera、Chrome、Safari 的 js 实现使用的都是标记清除的垃圾回收策略或类似的策略，只不过垃圾收集的时间间隔互不相同。

引用计数：

定义和用法：引用计数是跟踪记录每个值被引用的次数。

基本原理：就是变量的引用次数，被引用一次则加 1，当这个引用计数为 0 时，被视为准备回收的对象。

27、什么时候触发垃圾回收？

垃圾回收器周期性运行，如果分配的内存非常多，那么回收工作也会很艰巨，确定垃圾回收时间间隔就变成了一个值得思考的问题。

IE6 的垃圾回收是根据内存分配量运行的，当环境中的变量，对象，字符串达到一定数量时触发垃圾回收。垃圾回收器一直处于工作状态，严重影响浏览器性能。

IE7 中，垃圾回收器会根据内存分配量与程序占用内存的比例进行动态调整，开始回收工作。

28、开发过程中遇到的内存泄露情况，如何解决的？

1、定义和用法：

内存泄露是指一块被分配的内存既不能使用，又不能回收，直到浏览器进程结束。C#和 Java 等语言采用了自动垃圾回收方法管理内存，几乎不会发生内存泄露。我们知道，浏览器中也是采用自动垃圾回收方法管理内存，但由于浏览器垃圾回收方法有 bug，会产生内存泄露。

2、内存泄露的几种情况:

(1)、当页面中元素被移除或替换时，若元素绑定的事件仍没被移除，在 IE 中不会作出恰当处理，此时要先手工移除事件，不然会存在内存泄露。

实例如下:

```
<div id="myDiv">
    <input type="button" value="Click me" id="myBtn">
</div>
<script type="text/JavaScript">
    var btn = document.getElementById("myBtn");
    btn.onclick = function(){
        document.getElementById("myDiv").innerHTML = "Processing...";
    }
</script>
```

解决方法如下：

```
<div id="myDiv">
    <input type="button" value="Click me" id="myBtn">
</div>
<script type="text/JavaScript">
    var btn = document.getElementById("myBtn");
    btn.onclick = function(){
    btn.onclick = null;
        document.getElementById("myDiv").innerHTML = "Processing...";
    }
```

```
</script>
```

(2)、由于是函数内定义函数，并且内部函数--事件回调的引用外暴了，形成了闭包。闭包可以维持函数内局部变量，使其得不到释放。

实例如下：

```
function bindEvent(){
    var obj=document.createElement("XXX");
    obj.onclick=function(){
        //Even if it's a empty function
    }
}
```

解决方法如下：

```
function bindEvent(){
    var obj=document.createElement("XXX");
    obj.onclick=function(){
        //Even if it's a empty function
    }
    obj=null;
}
```

29、 JavaScript 面向对象中继承实现？

面向对象的基本特征有：封闭、继承、多态。

在 JavaScript 中实现继承的方法：

1. 原型链（prototype chaining）
2. call()/apply()
3. 混合方式(prototype 和 call()/apply()结合)
4. 对象冒充

继承的方法如下：

1、prototype 原型链方式：

```
function teacher(name){
    this.name = name;
}
teacher.prototype.sayName = function(){
    console.log("name is "+this.name);
}
var teacher1 = new teacher("xiaoming");
teacher1.sayName();
function student(name){
    this.name = name;
}
student.prototype = new teacher()
var student1 = new student("xiaolan");
student1.sayName();
//    name is xiaoming
```

```
//    name is xiaolan
```

2、call()/apply()方法

```
function teacher(name,age){
    this.name = name;
    this.age = age;
    this.sayhi = function(){
    alert('name:'+name+", age:"+age);
    }
}
function student(){
    var args = arguments;
    teacher.call(this,args[0],args[1]);
    // teacher.apply(this,arguments);
}
var teacher1 = new teacher('xiaoming',23);
teacher1.sayhi();
var student1 = new student('xiaolan',12);
student1.sayhi();
// alert: name:xiaoming, age:23
// alert: name:xiaolan, age:12
```

3、混合方法【prototype,call/apply】

```
function teacher(name,age){
    this.name = name;
    this.age = age;
}
teacher.prototype.sayName = function(){
  console.log('name:'+this.name);
}
teacher.prototype.sayAge = function(){
  console.log('age:'+this.age);
}

function student(){
    var args = arguments;
    teacher.call(this,args[0],args[1]);
}
student.prototype = new teacher();
var student1 = new student('xiaolin',23);
student1.sayName();
student1.sayAge();
// name:xiaolin
// age:23
```

4、对象冒充

```
function Person(name,age){
    this.name = name;
    this.age = age;
    this.show = function(){
    console.log(this.name+", "+this.age);
    }
}
function Student(name,age){
    this.student = Person; //将 Person 类的构造函数赋值给 this.student
    this.student(name,age); //js 中实际上是通过对象冒充来实现继承的
    delete this.student; //移除对 Person 的引用
}
var s = new Student("小明",17);
s.show();
var p = new Person("小花",18);
p.show();
// 小明, 17
// 小花, 18
```

30、判断一个字符串中出现次数最多的字符，统计这个次数。

```
var str = 'asdfssaaasasasasaa';
var json = {};
for (var i = 0; i < str.length; i++) {
    if(!json[str.charAt(i)]){
        json[str.charAt(i)] = 1;
    }else{
        json[str.charAt(i)]++;
    }
};
var iMax = 0;
var iIndex = '';
for(var i in json){
    if(json[i]>iMax){
        iMax = json[i];
        iIndex = i;
    }
}
console.log('出现次数最多的是:'+iIndex+'出现'+iMax+'次');
```

结果如下：出现次数最多的是:a 出现 9 次

31、Array 相关的属性和方法。

Array 对象属性

constructor 返回对创建此对象的数组函数的引用。
length 设置或返回数组中元素的数目。
prototype 使您有能力向对象添加属性和方法。
Array 对象方法
concat() 连接两个或更多的数组，并返回结果。
join() 把数组的所有元素放入一个字符串。元素通过指定的分隔符进行分隔。
pop() 删除并返回数组的最后一个元素。
shift() 删除并返回数组的第一个元素
push() 向数组的末尾添加一个或更多元素，并返回新的长度。
unshift() 向数组的开头添加一个或更多元素，并返回新的长度。
reverse() 颠倒数组中元素的顺序。
slice() 从某个已有的数组返回选定的元素
sort() 对数组的元素进行排序
splice() 删除元素，并向数组添加新元素。
toSource() 返回该对象的源代码。
toString() 把数组转换为字符串，并返回结果。
toLocaleString() 把数组转换为本地数组，并返回结果。
valueOf() 返回数组对象的原始值

32、编写一个方法去掉一个数组的重复元素。

方法一：

```
var arr = [0,2,3,4,4,0,2];
var obj = {};
var tmp = [];
for(var i = 0 ;i< arr.length;i++){
    if( !obj[arr[i]] ){
        obj[arr[i]] = 1;
        tmp.push(arr[i]);
    }
}
console.log(tmp);
```

结果如下： [0, 2, 3, 4]

方法二：

```
var arr = [2,3,4,4,5,2,3,6],
    arr2 = [];
for(var i = 0;i< arr.length;i++){
     if(arr2.indexOf(arr[i]) < 0){
          arr2.push(arr[i]);
     }
}
console.log(arr2);
```

结果为：[2, 3, 4, 5, 6]

方法三：

```
    var arr = [2,3,4,4,5,2,3,6];
    var arr2 = arr.filter(function(element,index,self){
    return self.indexOf(element) === index;
});
console.log(arr2);
```

结果为：[2, 3, 4, 5, 6]

33、jQuery 库中的 $() 是什么？

$() 函数是 jQuery() 函数的别称。$() 函数用于将任何对象包裹成 jQuery 对象，接着你就被允许调用定义在 jQuery 对象上的多个不同方法。你可以将一个选择器字符串传入 $() 函数，它会返回一个包含所有匹配的 DOM 元素数组的 jQuery 对象。

34、如何找到所有 HTML select 标签的选中项？

$('[name=selectname] :selected')

35、$(this) 和 this 关键字在 jQuery 中有何不同？

$(this) 返回一个 jQuery 对象，你可以对它调用多个 jQuery 方法，比如用 text() 获取文本，用 val() 获取值等等。

而 this 代表当前元素，它是 JavaScript 关键词中的一个，表示上下文中的当前 DOM 元素。你不能对它调用 jQuery 方法，直到它被 $() 函数包裹，例如 $(this)。

36、jQuery 怎么移除标签 onclick 属性？

获得 a 标签的 onclick 属性: $("a").attr("onclick")
删除 onclick 属性：$("a").removeAttr("onclick");
设置 onclick 属性：$("a").attr("onclick","test();");

37、jQuery 中 addClass,removeClass,toggleClass 的使用。

$(selector).addClass(class)：为每个匹配的元素添加指定的类名
$(selector).removeClass(class)：从所有匹配的元素中删除全部或者指定的类，删除 class 中某个值；
$(selector).toggleClass(class)：如果存在（不存在）就删除（添加）一个类
$(selector).removeAttr(class);删除 class 这个属性；

38、jQuery 有几种选择器？

(1)、基本选择器：#id，class,element,*;
(2)、层次选择器：parent > child，prev + next ， prev ~ siblings
(3)、基本过滤器选择器：:first，:last ， :not ， :even ， :odd ， :eq ， :gt ， :lt
(4)、内容过滤器选择器： :contains ， :empty ， :has ， :parent
(5)、可见性过滤器选择器：:hidden ， :visible

(6)、属性过滤器选择器：[attribute] ，[attribute=value] ，[attribute!=value] ，[attribute^=value] ，[attribute$=value] ，[attribute*=value]
(7)、子元素过滤器选择器：:nth-child ， :first-child ， :last-child ， :only-child
(8)、表单选择器： :input ， :text ， :password ， :radio ， :checkbox ， :submit 等;
(9)、表单过滤器选择器：:enabled ， :disabled ， :checked ， :selected

39、jQuery 中的 Delegate()函数有什么作用？

delegate()会在以下两个情况下使用到：
1、如果你有一个父元素，需要给其下的子元素添加事件，这时你可以使用 delegate()了，代码如下：

```
$("ul").delegate("li", "click", function(){ $(this).hide(); });
```

2、 当元素在当前页面中不可用时，可以使用 delegate()

40、$(document).ready()方法和 window.onload 有什么区别？

(1)、window.onload 方法是在网页中所有的元素(包括元素的所有关联文件)完全加载到浏览器后才执行的。
(2)、$(document).ready() 方法可以在 DOM 载入就绪时就对其进行操纵，并调用执行绑定的函数。

41、jQuery 中$.get()提交和$.post()提交有区别吗？

相同点：都是异步请求的方式来获取服务端的数据；
异同点：
1、请求方式不同：$.get() 方法使用 GET 方法来进行异步请求的。$.post() 方法使用 POST 方法来进行异步请求的。
2、参数传递方式不同：get 请求会将参数跟在 URL 后进行传递，而 POST 请求则是作为 HTTP 消息的实体内容发送给 Web 服务器的，这种传递是对用户不可见的。
3、数据传输大小不同：get 方式传输的数据大小不能超过 2KB 而 POST 要大的多
4、 安全问题： GET 方式请求的数据会被浏览器缓存起来，因此有安全问题。

42、写出一个简单的$.AJAX()的请求方式？

```
$.AJAX({
    url:'http://www.baidu.com',
    type:'POST',
    data:data,
    cache:true,
    headers:{},
    beforeSend：function(){},
    success:function(){},
```

```
    error:function(){},
    complete:function(){}
});
```

43、什么是盒子模型？

在网页中，一个元素占有空间的大小由几个部分构成，其中包括元素的内容（content），元素的内边距（padding），元素的边框（border），元素的外边距（margin）四个部分。这四个部分占有的空间中，有的部分可以显示相应的内容，而有的部分只用来分隔相邻的区域或区域。4 个部分一起构成了 css 中元素的盒模型。

44、行内元素有哪些？块级元素有哪些？ 空(void)元素有那些？

行内元素：a、b、span、img、input、strong、select、label、em、button、textarea
块级元素：div、ul、li、dl、dt、dd、p、h1-h6、blockquote
空元素：即系没有内容的 HTML 元素，例如：br、meta、hr、link、input、img

45、CSS 实现垂直水平居中。

一道经典的问题，实现方法有很多种，以下是其中一种实现：
HTML 结构：

```
<div class="wrapper">
        <div class="content"></div>
</div>
```

CSS：

```
.wrapper {
    position: relative;
    width: 500px;
    height: 500px;
    border: 1px solid red;
  }
.content{
    position: absolute;
    width: 200px;
    height: 200px;
    /*top、bottom、left 和 right  均设置为 0*/
    top: 0;
    bottom: 0;
    left: 0;
    right: 0;
    /*margin 设置为 auto*/
    margin:auto;
    border: 1px solid green;
```

}

46、简述一下 src 与 href 的区别？

href 是指向网络资源所在位置，建立和当前元素（锚点）或当前文档（链接）之间的链接，用于超链接。

src 是指向外部资源的位置，指向的内容将会嵌入到文档中当前标签所在位置；在请求 src 资源时会将其指向的资源下载并应用到文档内，例如 js 脚本，img 图片和 frame 等元素。

当浏览器解析到该元素时，会暂停其他资源的下载和处理，直到将该资源加载、编译、执行完毕，图片和框架等元素也如此，类似于将所指向资源嵌入当前标签内。这也是为什么将 js 脚本放在底部而不是头部。

47、简述同步和异步的区别？

同步是阻塞模式，异步是非阻塞模式。

同步就是指一个进程在执行某个请求的时候，若该请求需要一段时间才能返回信息，那么这个进程将会一直等待下去，直到收到返回信息才继续执行下去；

异步是指进程不需要一直等下去，而是继续执行下面的操作，不管其他进程的状态。当有消息返回时系统会通知进程进行处理，这样可以提高执行的效率。

48、px 和 em 的区别？

相同点：px 和 em 都是长度单位；
异同点：px 的值是固定的，指定是多少就是多少，计算比较容易。em 得值不是固定的，并且 em 会继承父级元素的字体大小。
浏览器的默认字体高都是 16px。所以未经调整的浏览器都符合: 1em=16px。那么 12px=0.75em, 10px=0.625em。

49、浏览器的内核分别是什么？

IE: trident 内核
Firefox：gecko 内核
Safari：webkit 内核
Opera：以前是 presto 内核，Opera 现已改用 Google Chrome 的 Blink 内核
Chrome：Blink(基于 webkit，Google 与 Opera Software 共同开发)

50、什么叫优雅降级和渐进增强？

渐进增强 progressive enhancement：
针对低版本浏览器进行构建页面，保证最基本的功能，然后再针对高级浏览器进行效果、交互等改进和追加功能达到更好的用户体验。
优雅降级 graceful degradation：
一开始就构建完整的功能，然后再针对低版本浏览器进行兼容。

区别：

a. 优雅降级是从复杂的现状开始，并试图减少用户体验的供给

b. 渐进增强则是从一个非常基础的，能够起作用的版本开始，并不断扩充，以适应未来环境的需要

c. 降级（功能衰减）意味着往回看；而渐进增强则意味着朝前看，同时保证其根基处于安全地带

51、sessionStorage 、localStorage 和 cookie 之间的区别？

共同点：用于浏览器端存储的缓存数据

不同点：

(1)、存储内容是否发送到服务器端：当设置了 Cookie 后，数据会发送到服务器端，造成一定的宽带浪费；web storage,会将数据保存到本地，不会造成宽带浪费；

(2)、数据存储大小不同：Cookie 数据不能超过 4K,适用于会话标识；web storage 数据存储可以达到 5M;

(3)、数据存储的有效期限不同：cookie 只在设置了 Cookid 过期时间之前一直有效，即使关闭窗口或者浏览器；sessionStorage,仅在关闭浏览器之前有效；localStorage,数据存储永久有效；

(4)、作用域不同：cookie 和 localStorage 是在同源同窗口中都是共享的；sessionStorage 不在不同的浏览器窗口中共享，即使是同一个页面；

52、Web Storage 与 Cookie 相比存在的优势？

(1)、存储空间更大：IE8 下每个独立的存储空间为 10M，其他浏览器实现略有不同，但都比 Cookie 要大很多。

(2)、存储内容不会发送到服务器：当设置了 Cookie 后，Cookie 的内容会随着请求一并发送的服务器，这对于本地存储的数据是一种带宽浪费。而 Web Storage 中的数据则仅仅是存在本地，不会与服务器发生任何交互。

(3)、更多丰富易用的接口：Web Storage 提供了一套更为丰富的接口，如 setItem,getItem,removeItem,clear 等,使得数据操作更为简便。cookie 需要自己封装。

(4)、独立的存储空间：每个域（包括子域）有独立的存储空间，各个存储空间是完全独立的，因此不会造成数据混乱。

53、AJAX 的优缺点及工作原理？

定义和用法:

AJAX = Asynchronous JavaScript and XML（异步的 JavaScript 和 XML）。

AJAX 是一种用于创建快速动态网页的技术。AJAX 是一种在无需重新加载整个网页的情况下，能够更新部分网页的技术。

传统的网页（不使用 AJAX）如果需要更新内容，必须重载整个网页页面。

优点：

1.减轻服务器的负担,按需取数据,最大程度的减少冗余请求

2.局部刷新页面,减少用户心理和实际的等待时间,带来更好的用户体验

3.基于 xml 标准化,并被广泛支持,不需安装插件等,进一步促进页面和数据的分离

缺点：

1.AJAX 大量的使用了 JavaScript 和 AJAX 引擎,这些取决于浏览器的支持.在编写的时候考虑对浏览器的兼容性.

2.AJAX 只是局部刷新,所以页面的后退按钮是没有用的.

3.对流媒体还有移动设备的支持不是太好等

AJAX 的工作原理：

1.创建 AJAX 对象（XMLHttpRequest/ActiveXObject(Microsoft.XMLHttp)）

2.判断数据传输方式(GET/POST)

3.打开链接 open()

4.发送 send()

5.当 AJAX 对象完成第四步（onreadystatechange）数据接收完成，判断 http 响应状态（status）200-300 之间或者 304（缓存）执行回调函数

54、请指出 document load 和 document ready 的区别？

共同点：这两种事件都代表的是页面文档加载时触发。

异同点：

ready 事件的触发，表示文档结构已经加载完成（不包含图片等非文字媒体文件）。

onload 事件的触发，表示页面包含图片等文件在内的所有元素都加载完成。

55、写一个 function，清除字符串前后的空格。（兼容所有浏览器）

```
function trim(str) {
    if (str && typeof str === "string") {
        return str.replace(/(^\s*)|(\s*)$/g,""); //去除前后空白符
    }
}
```

56、如何使用正则表达式验证邮箱格式。

```
var reg = /^(\w)+(\.\w+)*@(\w)+((\.\w{2,3}){1,3})$/;
var email = "example@qq.com";
console.log(reg.test(email));    // true
```

57、规避 JavaScript 多人开发函数重名问题。

命名空间

封闭空间

js 模块化 mvc（数据层、表现层、控制层）

seajs

变量转换成对象的属性

对象化

58、请说出三种减低页面加载时间的方法。

压缩 css、js 文件
合并 js、css 文件，减少 http 请求
外部 js、css 文件放在最底下
减少 dom 操作，尽可能用变量替代不必要的 dom 操作

59、你所了解到的 Web 攻击技术。

（1）XSS（Cross-Site Scripting，跨站脚本攻击）：指通过存在安全漏洞的 Web 网站注册用户的浏览器内运行非法的 HTML 标签或者 JavaScript 进行的一种攻击。

（2）SQL 注入攻击

（3）CSRF（Cross-Site Request Forgeries，跨站点请求伪造）：指攻击者通过设置好的陷阱，强制对已完成的认证用户进行非预期的个人信息或设定信息等某些状态更新。

60、web 前端开发，如何提高页面性能优化？

a) 内容方面：
1.减少 HTTP 请求 (Make Fewer HTTP Requests)
2.减少 DOM 元素数量 (Reduce the Number of DOM Elements)
3.使得 AJAX 可缓存 (Make AJAX Cacheable)

b) 针对 CSS：
1.把 CSS 放到代码页上端 (Put Stylesheets at the Top)
2.从页面中剥离 JavaScript 与 CSS (Make JavaScript and CSS External)
3.精简 JavaScript 与 CSS (Minify JavaScript and CSS)
4.避免 CSS 表达式 (Avoid CSS Expressions)

c) 针对 JavaScript ：
1. 脚本放到 HTML 代码页底部 (Put Scripts at the Bottom)
2. 从页面中剥离 JavaScript 与 CSS (Make JavaScript and CSS External)
3. 精简 JavaScript 与 CSS (Minify JavaScript and CSS)
4. 移除重复脚本 (Remove Duplicate Scripts)

d) 面向图片(Image)：
1.优化图片
2 不要在 HTML 中使用缩放图片
3 使用恰当的图片格式
4 使用 CSS Sprites 技巧对图片优化
5、前端开发中，如何优化图像？图像格式的区别？

e) 优化图像：
1、不用图片，尽量用 css3 代替。 比如说要实现修饰效果，如半透明、边框、圆角、阴影、渐变等，在当前主流浏览器中都可以用 CSS 达成。

2、 使用矢量图 SVG 替代位图。对于绝大多数图案、图标等，矢量图更小，且可缩放而无需生成多套图。现在主流浏览器都支持 SVG 了，所以可放心使用！

3.、使用恰当的图片格式。我们常见的图片格式有 JPEG、GIF、PNG。

基本上，内容图片多为照片之类的，适用于 JPEG。

而修饰图片通常更适合用无损压缩的 PNG。

GIF 基本上除了 GIF 动画外不要使用。且动画的话，也更建议用 video 元素和视频格式，或用 SVG 动画取代。

4、按照 HTTP 协议设置合理的缓存。

5、使用字体图标 webfont、CSS Sprites 等。

6、用 CSS 或 JavaScript 实现预加载。

7、WebP 图片格式能给前端带来的优化。WebP 支持无损、有损压缩，动态、静态图片，压缩比率优于 GIF、JPEG、JPEG2000、PG 等格式，非常适合用于网络等图片传输。

61、图像格式的区别？

矢量图：图标字体，如 font-awesome；svg

位图：gif,jpg(jpeg),png

区别：

1、gif:是是一种无损，8 位图片格式。具有支持动画，索引透明，压缩等特性。适用于做色彩简单(色调少)的图片，如 logo,各种小图标 icons 等。

2、JPEG 格式是一种大小与质量相平衡的压缩图片格式。适用于允许轻微失真的色彩丰富的照片，不适合做色彩简单(色调少)的图片，如 logo,各种小图标 icons 等。

3、png:PNG 可以细分为三种格式:PNG8，PNG24，PNG32。后面的数字代表这种 PNG 格式最多可以索引和存储的颜色值。关于透明：PNG8 支持索引透明和 alpha 透明;PNG24 不支持透明;而 PNG32 在 24 位的 PNG 基础上增加了 8 位（256 阶）的 alpha 通道透明;

优缺点：

1、能在保证最不失真的情况下尽可能压缩图像文件的大小。

2、对于需要高保真的较复杂的图像，PNG 虽然能无损压缩，但图片文件较大，不适合应用在 Web 页面上。

62、浏览器是如何渲染页面的？

渲染的流程如下：

1. 解析 HTML 文件

创建 DOM 树。自上而下，遇到任何样式（link、style）与脚本（script）都会阻塞（外部样式不阻塞后续外部脚本的加载）。

2.解析 CSS。

优先级：浏览器默认设置<用户设置<外部样式<内联样式<HTML 中的 style 样式；

3.将 CSS 与 DOM 合并，构建渲染树（Render Tree）

4.布局和绘制，重绘（repaint）和重排（reflow）

63、jQuery 的事件委托方法 bind 、live、delegate、on 之间有什么区别？

(1)、bind 【jQuery 1.3 之前】

定义和用法：主要用于给选择到的元素上绑定特定事件类型的监听函数；

语法：bind(type,[data],function(eventObject))；

特点：

(1)、适用于页面元素静态绑定。只能给调用它的时候已经存在的元素绑定事件，不能给未来新增的元素绑定事件。

(2)、当页面加载完的时候，你才可以进行 bind()，所以可能产生效率问题。

实例如下：$("#members li a").bind("click", function(e) {});

(2)、live 【jQuery 1.3 之后】

定义和用法：主要用于给选择到的元素上绑定特定事件类型的监听函数；

语法：live(type, [data], fn);

特点：

(1)、live 方法并没有将监听器绑定到自己(this)身上，而是绑定到了 this.context 上了。

(2)、live 正是利用了事件委托机制来完成事件的监听处理，把节点的处理委托给了 document，新添加的元素不必再绑定一次监听器。

(3)、使用 live（）方法但却只能放在直接选择的元素后面，不能在层级比较深，连缀的 DOM 遍历方法后面使用，即$(“ul” ").live...可以，但 $("body").find("ul").live...不行；

实例如下：$(document).on("click", "#members li a", function(e) {});

(3)、delegate 【jQuery 1.4.2 中引入】

定义和用法：将监听事件绑定在就近的父级元素上

语法：delegate(selector,type,[data],fn)

特点：

(1)、选择就近的父级元素，因为事件可以更快的冒泡上去，能够在第一时间进行处理。

(2)、更精确的小范围使用事件代理，性能优于.live()。可以用在动态添加的元素上。

实例如下：

$("#info_table").delegate("td","click",function(){/*显示更多信息*/});

$("table").find("#info").delegate("td","click",function(){/*显示更多信息*/});

(4)、on 【1.7 版本整合了之前的三种方式的新事件绑定机制】

定义和用法：将监听事件绑定到指定元素上。

语法：on(type,[selector],[data],fn)

实例如下：$("#info_table").on("click","td",function(){/*显示更多信息*/});参数的位置写法与 delegate 不一样。

说明：on 方法是当前 JQuery 推荐使用的事件绑定方法，附加只运行一次就删除函数的方法是 one()。

总结：.bind(), .live(), .delegate(),.on()
分别对应的相反事为：.unbind(),.die(), .undelegate(),.off()

四. 后端技术（Java Web）

1、Tomcat 的优化经验。

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？

GET 在浏览器回退时是无害的，而 POST 会再次提交请求。
GET 产生的 URL 地址可以被 Bookmark，而 POST 不可以。
GET 请求会被浏览器主动 cache，而 POST 不会，除非手动设置。
GET 请求只能进行 url 编码，而 POST 支持多种编码方式。
GET 请求参数会被完整保留在浏览器历史记录里，而 POST 中的参数不会被保留。
GET 请求在 URL 中传送的参数是有长度限制的，而 POST 么有。
对参数的数据类型，GET 只接受 ASCII 字符，而 POST 没有限制。
GET 比 POST 更不安全，因为参数直接暴露在 URL 上，所以不能用来传递敏感信息。
GET 参数通过 URL 传递，POST 放在 Request body 中。

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、ServletAPI 中 forward() 与 redirect()的区别？

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

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

getParameter 得到的都是 String 类型的。或者是 http://a.jsp？id=123 中的 123，或者是某个表单提交过去的数据。
getAttribute 则可以是对象。
getParameter()是获取 POST/GET 传递的参数值；
getAttribute()是获取对象容器中的数据值；
getParameter：用于客户端重定向时，即点击了链接或提交按扭时传值用，即用于在用表单或 url 重定向传值时接收数据用。
getAttribute：用于服务器端重定向时，即在 sevlet 中使用了 forward 函数，
或 struts 中使用了 mapping.findForward。 getAttribute 只能收到程序
用 setAttribute 传过来的值。
getParameter()是获取 POST/GET 传递的参数值；
getAttribute()是获取 SESSION 的值；
另外，可以用 setAttribute,getAttribute 发送接收对象.而 getParameter 显然只能传字符串。
setAttribute 是应用服务器把这个对象放在该页面所对应的一块内存中去，当你的页面服务器
重定向到另一个页面时，应用服务器会把这块内存拷贝另一个页面所对应的内存中。这样

getAttribute 就能取得你所设下的值，当然这种方法可以传对象。 session 也一样，只是对象
在内存中的生命周期不一样而已。 getParameter 只是应用服务器在分析你送上来的 request 页面的文本时，取得你设在表单或 url 重定向时的值。
getParameter 返回的是 String, 用于读取提交的表单中的值;
getAttribute 返回的是 Object，需进行转换,可用 setAttribute 设置成任意对象，使用很灵活，可随时用;

11、JSP 有哪些内置对象？作用分别是什么？ 分别有什么方法？

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

12、 JSP 有哪些动作？作用分别是什么？

JSP 共有以下 6 种基本动作
jsp:include：在页面被请求的时候引入一个文件。
jsp:useBean：寻找或者实例化一个 JavaBean。
jsp:setProperty：设置 JavaBean 的属性。
jsp:getProperty：输出某个 JavaBean 的属性。

jsp:forward：把请求转到一个新的页面。
jsp:plugin：根据浏览器类型为 Java 插件生成 OBJECT 或 EMBED 标记

13、JSP 的常用指令有哪些？

isErrorPage(是否能使用 Exception 对象)，isELIgnored(是否忽略表达式)

14、JSP 中动态 INCLUDE 与静态 INCLUDE 的区别？

动态 INCLUDE 用 jsp:include 动作实现
<jsp:include page=included.jsp flush=true />它总是会检查所含文件中的变化，适合用于包含动态页面，并且可以带参数
静态 INCLUDE 用 include 伪码实现,定不会检查所含文件的变化，适用于包含静态页面 <%@ include file=included.htm %>

15、Servlet 是线程安全的吗？

Servlet 默认是单例模式，在 web 容器中只创建一个实例，所以多个线程同时访问 Servlet 的时候，Servlet 是线程不安全的。

16、页面间对象传递的方法？

request，session，application，cookie 等

17、JSP 和 Servlet 有哪些相同点和不同点？

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。

18、MVC 的各个部分都有那些技术来实现？如何实现？

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

19、如何输出 ISO-8859-1 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;
}
```

20、Tomcat 有几种部署方式？

1）直接把 Web 项目放在 webapps 下，Tomcat 会自动将其部署

2）在 server.xml 文件上配置<Context>节点，设置相关的属性即可

3）通过 Catalina 来进行配置:进入到 conf\Catalina\localhost 文件下，创建一个 xml 文件，该文件的名字就是站点的名字。编写 XML 的方式来进行设置。

21、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)

讲解这些区别是不需要特别去比较，就像说传智播客与其他培训机构的区别时，我们只需说清楚传智播客有什么特点和优点就行了，这就已经间接回答了彼此的区别。

22、你在项目中用到了 xml 技术的哪些方面？如何实现的？

答:用到了数据存储，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。

23、用 jdom 解析 xml 文件时如何解决中文问题？如何解析？

```
try {
        //1、获取 DOM 解析器工厂，以便产生解析器；2、获取 DOM 解析器，以便解析 DOM
        DocumentBuilder builder = DocumentBuilderFactory
                .newInstance().newDocumentBuilder();
        Document doc = builder.newDocument();
        //创建元素
        Element root = doc.createElement("person");
```

```
            Element a = doc.createElement("name");
            Element b = doc.createElement("age");
            //向指定元素节点中增加子元素节点或增加元素到子节点
            a.appendChild(doc.createTextNode("小白"));
            b.appendChild(doc.createTextNode("18 岁"));
            root.appendChild(a);
            root.appendChild(b);
            doc.appendChild(root);
            //设置后要把 DOM 写回 XML 文件
            Transformer transformer = TransformerFactory
                    .newInstance().newTransformer();
            //编码指定 Transformer 应该使用的首选字符编码，将字符序列作为字节序列进行编码
            transformer.setOutputProperty(OutputKeys.ENCODING, "gb2312");
            //缩进指定 Transformer 是否可以添加额外的空白，同时输出
            transformer.setOutputProperty(OutputKeys.INDENT, "yes");
            transformer.transform(new DOMSource(doc), new
StreamResult(outFile));
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
```

24、XML 文档定义有几种形式？它们之间有何本质区别？

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)。

25、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。

26、应用服务器与 WEB SERVER 的区别？

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

27、应用服务器有那些？

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

28、J2EE 是什么？

JavaEE 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。

29、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。

30、请对以下在 JavaEE 中常用的名词进行解释。

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。

31、四种会话跟踪技术

会话作用域 Servlets、JSP 页面描述：

page 否是代表与一个页面相关的对象和属性。一个页面由一个编译好的 Java Servlet 类（可以带有任何的 include 指令，但是没有 include 动作）表示。这既包括 Servlet 又包括被编译成 Servlet 的 JSP 页面

request 是是代表与 Web 客户机发出的一个请求相关的对象和属性。一个请求可能跨越多个页面，涉及多个 Web 组件（由于 forward 指令和 include 动作的关系）

session 是是代表与用于某个 Web 客户机的一个用户体验相关的对象和属性。一个 Web 会话可以也经常会跨越多个客户机请求

application 是是代表与整个 Web 应用程序相关的对象和属性。这实质上是跨越整个 Web 应用程序，包括多个页面、请求和会话的一个全局作用域

五. 主流的框架

（一）Spring 框架

1、什么是 Spring 框架？Spring 框架有哪些主要模块？

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

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

2、使用 Spring 框架能带来哪些好处？

Dependency Injection(DI) 方法使得构造器和 JavaBean properties 文件中的依赖关系一目了然。

与 EJB 容器相比较，IoC 容器更加趋向于轻量级。这样一来 IoC 容器在有限的内存和 CPU 资源的情况下进行应用程序的开发和发布就变得十分有利。

Spring 并没有闭门造车，Spring 利用了已有的技术比如 ORM 框架、logging 框架、J2EE、Quartz 和 JDK Timer，以及其他视图技术。

Spring 框架是按照模块的形式来组织的。由包和类的编号就可以看出其所属的模块，开发者仅仅需要选用他们需要的模块即可。

要测试一项用 Spring 开发的应用程序十分简单，因为测试相关的环境代码都已经囊括在框架中了。更加简单的是，利用 JavaBean 形式的 POJO 类，可以很方便的利用依赖注入来写入测试数据。

Spring 的 Web 框架亦是一个精心设计的 Web MVC 框架，为开发者们在 web 框架的选择上提供了一个除了主流框架比如 Struts、过度设计的、不流行 web 框架的以外的有力选项。
Spring 提供了一个便捷的事务管理接口，适用于小型的本地事物处理（比如在单 DB 的环境下）和复杂的共同事物处理（比如利用 JTA 的复杂 DB 环境）。

3、什么是控制反转(IOC)？什么是依赖注入？

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

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

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

在 Java 中依然注入有以下三种实现方式：

1) 构造器注入
2) Setter 方法注入
3) 接口注入

4、请解释下 Spring 框架中的 IoC?

Spring 中的 org.Springframework.beans 包和 org.Springframework.context 包构成了 Spring 框架 IoC 容器的基础。

BeanFactory 接口提供了一个先进的配置机制，使得任何类型的对象的配置成为可能。ApplicationContex 接口对 BeanFactory（是一个子接口）进行了扩展，在 BeanFactory 的基础上添加了其他功能，比如与 Spring 的 AOP 更容易集成，也提供了处理 message resource 的机制（用于国际化）、事件传播以及应用层的特别配置，比如针对 Web 应用的 WebApplicationContext。
org.Springframework.beans.factory.BeanFactory 是 Spring IoC 容器的具体实现，用来包装和管理前面提到的各种 bean。BeanFactory 接口是 Spring IoC 容器的核心接口。

5、BeanFactory 和 ApplicationContext 有什么区别？

BeanFactory 可以理解为含有 bean 集合的工厂类。BeanFactory 包含了种 bean 的定义，以便在接收到客户端请求时将对应的 bean 实例化。
BeanFactory 还能在实例化对象的时生成协作类之间的关系。此举将 bean 自身与 bean 客户端的配置中解放出来。BeanFactory 还包含了 bean 生命周期的控

制，调用客户端的初始化方法（initialization methods）和销毁方法（destruction methods）。

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

1) 提供了支持国际化的文本消息
2) 统一的资源文件读取方式
3) 已在监听器中注册的 bean 的事件

6、Spring 有几种配置方式？

1) 基于 XML 的配置
2) 基于注解的配置
3) 基于 Java 的配置

7、解释 Spring 支持的几种 bean 的作用域。

Spring 容器中的 bean 可以分为 5 个范围：

（1）singleton：默认，每个容器中只有一个 bean 的实例，单例的模式由 BeanFactory 自身来维护。

（2）prototype：为每一个 bean 请求提供一个实例。

（3）request：为每一个网络请求创建一个实例，在请求完成以后，bean 会失效并被垃圾回收器回收。

（4）session：与 request 范围类似，确保每个 session 中有一个 bean 的实例，在 session 过期后，bean 会随之失效。

（5）global-session：全局作用域，global-session 和 Portlet 应用相关。当你的应用部署在 Portlet 容器中工作时，它包含很多 portlet。如果你想要声明让所有的 portlet 共用全局的存储变量的话，那么这全局变量需要存储在 global-session 中。全局作用域与 Servlet 中的 session 作用域效果相同。

8、Spring 框架中的单例 Beans 是线程安全的么？

Spring 框架并没有对单例 bean 进行任何多线程的封装处理。关于单例 bean 的线程安全和并发问题需要开发者自行去搞定。但实际上，大部分的 Spring bean 并没有可变的状态(比如 Serview 类和 DAO 类)，所以在某种程度上说 Spring 的单例 bean 是线程安全的。如果你的 bean 有多种状态的话（比如 View Model 对象），就需要自行保证线程安全。最浅显的解决办法就是将多态 bean 的作用域由“singleton”变更为“prototype”。

9、Spring 如何处理线程并发问题？

在一般情况下，只有无状态的 Bean 才可以在多线程环境下共享，在 Spring 中，绝大部分 Bean 都可以声明为 singleton 作用域，因为 Spring 对一些 Bean 中非线程安全状态采用 ThreadLocal 进行处理，解决线程安全问题。

ThreadLocal 和线程同步机制都是为了解决多线程中相同变量的访问冲突问题。同步机制采用了“时间换空间”的方式，仅提供一份变量，不同的线程在

访问前需要获取锁，没获得锁的线程则需要排队。而 ThreadLocal 采用了“空间换时间”的方式。

ThreadLocal 会为每一个线程提供一个独立的变量副本，从而隔离了多个线程对数据的访问冲突。因为每一个线程都拥有自己的变量副本，从而也就没有必要对该变量进行同步了。ThreadLocal 提供了线程安全的共享对象，在编写多线程代码时，可以把不安全的变量封装进 ThreadLocal。

10、Spring 的自动装配。

在 Spring 中，对象无需自己查找或创建与其关联的其他对象，由容器负责把需要相互协作的对象引用赋予各个对象，使用 autowire 来配置自动装载模式。

在 Spring 框架 xml 配置中共有 5 种自动装配：

（1）no：默认的方式是不进行自动装配的，通过手工设置 ref 属性来进行装配 bean。

（2）byName：通过 bean 的名称进行自动装配，如果一个 bean 的 property 与另一 bean 的 name 相同，就进行自动装配。

（3）byType：通过参数的数据类型进行自动装配。

（4）constructor：利用构造函数进行装配，并且构造函数的参数通过 byType 进行装配。

（5）autodetect：自动探测，如果有构造方法，通过 construct 的方式自动装配，否则使用 byType 的方式自动装配。

基于注解的方式：

使用@Autowired 注解来自动装配指定的 bean。在使用@Autowired 注解之前需要在 Spring 配置文件进行配置，<context:annotation-config />。在启动 Spring IoC 时，容器自动装载了一个 AutowiredAnnotationBeanPostProcessor 后置处理器，当容器扫描到@Autowied、@Resource 或@Inject 时，就会在 IoC 容器自动查找需要的 bean，并装配给该对象的属性。在使用@Autowired 时，首先在容器中查询对应类型的 bean：

如果查询结果刚好为一个，就将该 bean 装配给@Autowired 指定的数据；

如果查询的结果不止一个，那么@Autowired 会根据名称来查找；

如果上述查找的结果为空，那么会抛出异常。解决方法时，使用 required=false。

@Autowired 可用于：构造函数、成员变量、Setter 方法

注：@Autowired 和@Resource 之间的区别

(1) @Autowired 默认是按照类型装配注入的，默认情况下它要求依赖对象必须存在（可以设置它 required 属性为 false）。

(2) @Resource 默认是按照名称来装配注入的，只有当找不到与名称匹配的 bean 才会按照类型来装配注入。

11、Spring 框架中都用到了哪些设计模式？

（1）工厂模式：BeanFactory 就是简单工厂模式的体现，用来创建对象的实例；

（2）单例模式：Bean 默认为单例模式。
（3）代理模式：Spring 的 AOP 功能用到了 JDK 的动态代理和 CGLIB 字节码生成技术；
（4）模板方法：用来解决代码重复的问题。比如. RestTemplate, JmsTemplate, JpaTemplate。
（5）观察者模式：定义对象键一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都会得到通知被制动更新，如 Spring 中 listener 的实现--ApplicationListener。

12、Spring 事务的实现方式和实现原理？

Spring 事务的本质其实就是数据库对事务的支持，没有数据库的事务支持，Spring 是无法提供事务功能的。真正的数据库层的事务提交和回滚是通过 binlog 或者 redo log 实现的。
（1）Spring 事务的种类：
Spring 支持编程式事务管理和声明式事务管理两种方式：

a) 编程式事务管理使用 TransactionTemplate。
b) 声明式事务管理建立在 AOP 之上的。其本质是通过 AOP 功能，对方法前后进行拦截，将事务处理的功能编织到拦截的方法中，也就是在目标方法开始之前加入一个事务，在执行完目标方法之后根据执行情况提交或者回滚事务。

声明式事务最大的优点就是不需要在业务逻辑代码中掺杂事务管理的代码，只需在配置文件中做相关的事务规则声明或通过@Transactional 注解的方式，便可以将事务规则应用到业务逻辑中。
声明式事务管理要优于编程式事务管理，这正是 Spring 倡导的非侵入式的开发方式，使业务代码不受污染，只要加上注解就可以获得完全的事务支持。唯一不足地方是，最细粒度只能作用到方法级别，无法做到像编程式事务那样可以作用到代码块级别。
（2）Spring 的事务传播行为：
Spring 事务的传播行为说的是，当多个事务同时存在的时候，Spring 如何处理这些事务的行为。
① PROPAGATION_REQUIRED：如果当前没有事务，就创建一个新事务，如果当前存在事务，就加入该事务，该设置是最常用的设置。
② PROPAGATION_SUPPORTS：支持当前事务，如果当前存在事务，就加入该事务，如果当前不存在事务，就以非事务执行。‘
③ PROPAGATION_MANDATORY：支持当前事务，如果当前存在事务，就加入该事务，如果当前不存在事务，就抛出异常。
④ PROPAGATION_REQUIRES_NEW：创建新事务，无论当前存不存在事务，都创建新事务。
⑤ PROPAGATION_NOT_SUPPORTED：以非事务方式执行操作，如果当前存在事务，就把当前事务挂起。
⑥ PROPAGATION_NEVER：以非事务方式执行，如果当前存在事务，则抛出异常。

⑦ PROPAGATION_NESTED：如果当前存在事务，则在嵌套事务内执行。如果当前没有事务，则按 REQUIRED 属性执行。
（3）Spring 中的隔离级别：
① ISOLATION_DEFAULT：这是个 PlatfromTransactionManager 默认的隔离级别，使用数据库默认的事务隔离级别。
② ISOLATION_READ_UNCOMMITTED：读未提交，允许另外一个事务可以看到这个事务未提交的数据。
③ ISOLATION_READ_COMMITTED：读已提交，保证一个事务修改的数据提交后才能被另一事务读取，而且能看到该事务对已有记录的更新。
④ ISOLATION_REPEATABLE_READ：可重复读，保证一个事务修改的数据提交后才能被另一事务读取，但是不能看到该事务对已有记录的更新。
⑤ ISOLATION_SERIALIZABLE：一个事务在执行的过程中完全看不到其他事务对数据库所做的更新。

13、Spring 框架中有哪些不同类型的事件？

Spring 提供了以下 5 种标准的事件：
（1）上下文更新事件（ContextRefreshedEvent）：在调用 ConfigurableApplicationContext 接口中的 refresh()方法时被触发。
（2）上下文开始事件（ContextStartedEvent）：当容器调用 ConfigurableApplicationContext 的 Start()方法开始/重新开始容器时触发该事件。
（3）上下文停止事件（ContextStoppedEvent）：当容器调用 ConfigurableApplicationContext 的 Stop()方法停止容器时触发该事件。
（4）上下文关闭事件（ContextClosedEvent）：当 ApplicationContext 被关闭时触发该事件。容器被关闭时，其管理的所有单例 Bean 都被销毁。
（5）请求处理事件（RequestHandledEvent）：在 Web 应用中，当一个 http 请求（request）结束触发该事件。
如果一个 bean 实现了 ApplicationListener 接口，当一个 ApplicationEvent 被发布以后，bean 会自动被通知。

14、注解的原理？

注解本质是一个继承了 Annotation 的特殊接口，其具体实现类是 Java 运行时生成的动态代理类。我们通过反射获取注解时，返回的是 Java 运行时生成的动态代理对象。通过代理对象调用自定义注解的方法，会最终调用 AnnotationInvocationHandler 的 invoke 方法。该方法会从 memberValues 这个 Map 中索引出对应的值。而 memberValues 的来源是 Java 常量池。

15、Spring 中的 jdbc 与传统的 jdbc 有什么区别？

Spring 的 JDBC 是在原生态 JDBC 上面的一层简单的封装，提供了一些一用的接口，节省代码，不管连接(Connection)，不管事务、不管异常、不管关闭(con.close() ps.close)。只需要实现 Spring 提供的回调类。传统的 JDBC 执行过程如下：
a、创建连接

b、创建语句（SQL）
c、执行语句
d、返回结果集（设置到 JavaBean 中等处理）
e、关闭连接释放资源

Spring JDBC 只需要实现 b 和 d 两个步骤，其他的都由 Spring 替你完成，而 b 和 d 都是通过实现指定的接口，然后将实现类传递给 Spring 就 OK 了。

16、什么是依赖注入？

依赖注入(DependencyInjection)和控制反转(Inversionof Control)是同一个概念。具体含义是:当某个角色(可能是一个 Java 实例，调用者)需要另一个角色(另一个 Java 实例，被调用者)的协助时，在传统的程序设计过程中，通常由调用者来创建被调用者的实例。但在 Spring 里，创建被调用者的工作不再由调用者来完成，因此称为控制反转;创建被调用者 实例的工作通常由 Spring 容器来完成，然后注入调用者，因此也称为依赖注入。

17、什么是面向切面编程？

针对业务处理过程中的切面进行提取，它所面对的是处理过程中的某个步骤或阶段，以获得逻辑过程中各部分之间低耦合性的隔离效果。

18、为了降低 Java 开发的复杂性， Spring 采取了以下 4 种关键策略？

1) 基于 POJO 的轻量级和最小侵入性编程，
2) 通过依赖注入和面向接口实现松耦合，
3) 基于切面和惯例进行声明式编程，
4) 通过切面和模板减少样板式代码。

19、依赖注入有哪些方式？

接口注入（interface injection） 接口注入指的就是在接口中定义要注入的信息，并通过接口完成注入。

Set 注入（setterinjection）Set 注入指的就是在接受注入的类中定义一个 Set 方法，并在参数中定义需要注入的元素。

构造注入（constructor injection）构造注入指的就是在接受注入的类中定义一个构造方法，并在参数中定义需要注入的元素

20、Spring 应用上下文的有哪些？

ClassPathXmlApplicationContext——从类路径下的 XML 配置文件中加载上下文定义，把应用上下文定义文件当作类资源。

FileSystemXmlApplicationcontext 该取文件系统统下 XML 配置文件并加载上下文定义。

XmlWebApplicationContext——读取 Web 应用下的 XML 配置文件并装载上下文定义。

使用 FileSystemXmlapplicationcontext 和使用 ClassPathXmlApplicationContext 的区别在于:

FileSystemXmlapplicationcontext 在指定的文件系统路径下查找文件，

ClassPathXmlApplicationContext 是在所有的类路径（包括 JAR 文件）下查找文件。

21、Bean 的生命周期？

下面是 Spring3 的 Bean 生命周期。

1) Spring 对 Bean 进行实例化.

2) Spring 将值和 Bean 的引用注入进 Bean 对应的属性中。

3) 如果 Bean 实现了 BeanNameAware 接口， Spring 将 bean 的 ID 传递给 setBeanName() 接口方法.

4) 如果 Bean 实现了 BeanFactoryAware 接口， Spring 将调用 setBeanFactory()接口方法，将 BeanFactory 容器实例传入.

5) 如果 Bean 实现了 ApplicationcontextAware 接口 Spring 将调用 setApplicationContext() 接口方法，将应用上下文的引用传入。

6) 如果 Bean 实现了 BeanPostProcessor 接口 Spring 将调用它们的 postProcessBeforeInitialization 接口方法。

7) 如果 Bean 实现了 InitializingBean 接口，Spring 将调用它们的 afterPropertiesSet()接口方法. 类似地，如果 Bean 使用 init-method 声明了初始化方法，该方法也会被调用。

8) 如果 Bean 实现了 BeanPostProcessor 接口， Spring 将调用它们的 postPoressAfterInitilization 方法.

9) 此时此刻 Bean 已经准备就绪. 可以被应用程序使用了. 它们将一直驻留在应用上下文中. 直到该应用上下文补销毁。

10) 如果 Bean 实现了 DisposableBean 接口，Spring 将调用它的 destroy()接口方法。同样，如果 Bean 使用 destroy-method 声明了销毁方法，方法也会被调用。

22、Spring 如何减少 XML 的配置数量？

自动装配（autowiring）有助于减少甚至消除<property>元素和<constructor-arg>元素，让 Spring 自动识别如何装配 Bean 的依赖关系。

自动检测（autudiscovery）比自动装配更进一步，让 Spring 能够自动识哪些类需要被配置成 Spring Bean，从而减少对 Bean 元素的使用。

23、四种类型的自动装配？

byName——把与 Bean 的属性具有相同名字（或者 ID）的其它 Bean 自动装配到对应属性中。如果没有跟属性的名字相匹配的 Bean，则该属性不进行装配。

byType——把与 Bean 的属性具有相同类型的其它 Bean 自动装配到对应属性中。如果没有跟属性的类型相匹配的 Bean，则该属性不进行装配。

constructor——把与 Bean 构造器参数具有相同类型的其他 Bean 自动装配到 Bean 构造器的对应参数中。

autodetect——首先尝试使用 constructor 进行自动装配，如果失败，再尝试使用 byType 进行自动装配。

24、四种类型自动装配的约束和缺点？

byName——约定：为属性自动装配 ID 与该属性的名字相同的 Bean。缺点：需要假设想 Bean 的名字与其他 Bean 的属性的名字一样。

byType——缺点：如果 Spring 寻找到多个 Bean，并且类型都匹配，Spring 会抛出异常。解决方法：可以为自动装配标识一个首选 Bean 或者可以取消某个 Bean 自动装配的候选资格。

constructor——发现多个 Bean 匹配某个构造器的参数时，Spring 不会尝试猜哪一个 Bean 更适合自动装配。此外如果一个类有多个构造器，它们都满足自动装配条件时，Spring 也不会尝试哪一个构造器更适合使用。

25、使用 Spring 注解装配？

<context:annotation-config>元素告诉 Spring 使用基于注解的自动装配。Spring 支待几种不同的用于自动装配的注解

Spring 自带的@Autowired 注解。(@Autowired(required=false)说明不一定要装配，null 值也可以接受)@Autowired 可以装配构造器，方法和属性。

JSR-330 的@Inject 注解。可以装配构造器，方法和属性。@Inject 注解所标注的依赖关系必须存在，如果不存在就会抛出异常。

JSR-250 的@Resource 注解。

26、自动检测 Bean？

为了配置自动检测，需要使用<context:component-scan>元素。默认情况下，<context:component-scan>使用构造型注所标注类。这些特殊的注解如下

@Component——通用构造型注解，标识该类为 Spring 组件

@Controller——标识将该类定义为 SpringMVC controller

@Respository——标识将该类定义为数据仓库

@Service——标识将该类定义为服务

27、请解释 AOP 理论中相关名词？

关注点(concern)，切面(aspect)，连接点(joinpoint)，切入点(pointcut)，增强(advice)或者通知，织入(Weaving)，目标对象(target object)，代理对象(proxy object)。

a) 关注点(concern)

1) 核心关注点：关注系统的业务逻辑 --> OOP

2) 横切关注点：关注系统级服务,比如事务、安全、日志 --> AOP

b) 切面(aspect):

把散落在系统各处与横切关注点相关的重复代码抽取出来归整到一处形成一个模块,我们称为切面.

c) 连接点(joinpoint):
程序运行过程中的某一点.比如方法调用、属性访问、异常抛出.
d) 切入点(pointcut):
一组连接点，注意: 如果要有选择性地拦截目标对象中的方法的话需要定义切入点
e) 增强(advice)或者通知:
在不修改原有代码的前提下,为某一个对象增加新的功能 (如:事务服务、日志服务),在 Spring 中增强是通过拦截器实现的.
f) 织入(Weaving):
将切面应用到目标对象来创建新的代理对象的过程.
g) 目标对象(target object):
需要被增强功能的对象称之为目标对象,也被称为被增强或被代理对象。
h) 代理对象(proxy object)
为目标对象增加新功能从而产生的一个新的对象称为代理对象.负责调用拦截器和目标对象的方法.

28、Spring 切面可以应用五种类型的通知？

1) Before——在方法调用之前调用通知。
2) After——在方法完成之后调用通知，无论方法执行是否成功。
3) After-returning—— 在生方法成功执行之后调用通知。
4) After-throwing——在方法抛出异常后调用通知。
5) Around——通知包裹了被通知的方法，在被通知的方法调用之前和调用之后执行自定义的行为。

29、织入的时机有哪些？

1) 编译期——切面在目标类编译时被织入。
2) 类加载期——切面在目标类加载到 JVM 时被织入，这种方式需要特殊的类加载器，可以在目标类被引入应用之前增强该目标类的字节。
3) 运行期——切面在应用运行的某个时刻被织入，一般情况下，在织入切面时，AOP 容器会为目标对象动态地创建一个代表对象。

30、Spring 提供的四种各具特色的 AOP 支持？

a)基于代理的经典 AOP。
b)@AspectJ 注解驱动切面。
c)纯 POJO 切面
d)注入式 AspectJ 切面（适合 Spring 各版本）。

31、Spring 提供了在 Spring 上下中配置数据源 Bean 的多种方式？

a)通过 JDBC 驱动程序定义数据源。
b)通过 JNDI 查找数据源。
c)连接池的数据源。

32、Spring 为 JDBC 提供的模板类？

a), JdbcTemplate：最基本的 Spring JDBC 模板，这个模板支持最装简单的 JDBC 数据库访问功能及简单的索引参数查询。

b), NamedParameterJdbcTemplate：使用该模板执行查询时，可以将查询值以命名参数的形式绑定到 SQL 中，而不是使用简单的索引参数。

c), SimpleJdbcTemplate：该模板类利用 Java5 的一些特性，如自动装箱、泛型及可变参数来简化 JDBC 模板的使用。

33、Spring JDBC 数据源？

Spring 提供了两种 JDBC 数据源对象。
DriverManagerDataSource ：在每个连接请求时都会返回一个新建的连接。没有进数据池化管理。
SingleConnectionDataSource：在每个连接请求时都会返回同一个连接。

34、Spring 为 ORM 框架提供的一些服务？

Spring 声明式事物的集成支持。
透明的异常处理
线程安全的、轻量级的模板类。
DAO 支持类。
资源管理。

35、JPA 两种类型的实体管理器？

应用程序管理类型：当应用程序向实体管理器工厂直接请求实体管理器时，工厂会创建一个实体管理器，在这种模式下，程序要负责打开或关闭实体管理器并在事物中对其进行控制。这种方式的实体管理器适用于不运行在 JavaEE 容器中的独立应用程序。

容器管理类型：实体管理器由 JavaEE 创建和管理。应用程序不与实体管理器工厂打交道，实体管理器直接通过注入或者 JNDI 来获取。容器负责配置实体管理器工厂。这种类型的实体管理器最适合应用于 JavaEE 容器。

36、Spring 事务的支持？

Spring 提供了编码式事务管理和声明式事务管理。

37、声明式事务管理的事务属性有五种？

传播行为：传播行为定义了客户端与彼调用方法之间的事务边界。
隔离级别：隔离级别定义了一个事务可能受其他并发事务影响的程度。
只读：表明事务是否是只读的。
事务超时：指定事务运行的最长时间，超时就回滚。
回滚规则：定义了哪些异常会导致事务回滚而哪些不会。

38、脏读、不可重复读和幻读什么意思？

脏读：一个事务读取了另一个事务改写但是未提交的数据时。如果改写在稍后被回滚了，那么第一个事务获取的数据就是无效的。

不可重复读：一个事物执行相同的查询两次或者两次以上，但是每次都得到不同的数据。通常是因为另一个并发事务在两次查询期间更新了数据。

幻读：一个事务读取了几行数据，接着另一个并发事务插入了一些数据。在随后的查询中，第一个事务发现多了一些原本不存在的事务。

39、Spring 注解说明？

采用扫描的方式 配置 Spring XML 书写方式
<context:component-scanbase-package="这里是包名"/>

@Service 用于标注业务层组建
@Controller 用于标注控制层组件
@Component 泛指组件,用于不明确的类
@Repository 用户标注数据库访问组件
@PostConstruct 创建对象时执行的方法
@PreDestroy 对象摧毁时执行的方法
@Autowired 直接访问私有属性，直接给私有属性输入值。
@Resource 根据类型注入

40、Spring 的优点有什么？

1)Spring 是分层的架构，你可以选择使用你需要的层而不用管不需要的部分
2)Spring 是 POJO（Plain Ordinary Java Object），POJO 编程使得可持续构建和可测试能力提高
3)依赖注入和 IoC 使得 JDBC 操作简单化
4)Spring 是开源的免费的
5)Spring 使得对象管理集中化和简单化

41、Spring 的事务管理？

Spring 提供了几个关于事务处理的类：
TransactionDefinition //事务属性定义
TranscationStatus //代表了当前的事务，可以提交，回滚。
PlatformTransactionManager 这个是 Spring 提供的用于管理事务的基础接口，其下有一个实现的抽象类 AbstractPlatformTransactionManager,我们使用的事务管理类例如 DataSou rceTransactionManager 等都是这个类的子类。

42、AOP 的作用？

AOP:aspect oriented programming 被称为面向方面编程，面向方面编程主要关注的是公共部分的处理，把共同部分的逻辑封装成一个方面组件，该组件根据自己的需要可以切入到某一批目标对象方法中，从而达到方面组件和目标对象之间的解耦,即核心业务与横切业务的分离作用!

（二）SpringMVC 框架

1、什么是 SpringMVC？简单介绍下你对 SpringMVC 的理解？

Spring MVC 是一个基于 Java 的实现了 MVC 设计模式的请求驱动类型的轻量级 Web 框架，通过把 Model，View，Controller 分离，将 web 层进行职责解耦，把复杂的 web 应用分成逻辑清晰的几部分，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？

（1）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(如果有则生成)一并返回给 DispatcherServlet；
（4）DispatcherServlet 调用 HandlerAdapter 处理器适配器；
（5）HandlerAdapter 经过适配调用 具体处理器(Handler，也叫后端控制器)；
（6）Handler 执行完成返回 ModelAndView；
（7）HandlerAdapter 将 Handler 执行结果 ModelAndView 返回给 DispatcherServlet；
（8）DispatcherServlet 将 ModelAndView 传给 ViewResolver 视图解析器进行解析；
（9）ViewResolver 解析后返回具体 View；
（10）DispatcherServlet 对 View 进行渲染视图（即将模型数据填充至视图中）
（11）DispatcherServlet 响应用户。

3、SpringMVC 的优点有哪些？

（1）可以支持各种视图技术,而不仅仅局限于 JSP；
（2）与 Spring 框架集成（如 IoC 容器、AOP 等）；
（3）清晰的角色分配：前端控制器(dispatcherServlet)，请求到处理器映射（handlerMapping)，处理器适配器（HandlerAdapter)，视图解析器（ViewResolver）。
（4） 支持各种请求资源的映射策略。

4、SpringMVC 的主要组件？

（1）前端控制器 DispatcherServlet（不需要程序员开发）
作用：接收请求、响应结果，相当于转发器，有了 DispatcherServlet 就减少了其它组件之间的耦合度。
（2）处理器映射器 HandlerMapping（不需要程序员开发）
作用：根据请求的 URL 来查找 Handler
（3）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。
（4）处理器 Handler（需要程序员开发）
（5）视图解析器 ViewResolver（不需要程序员开发）
作用：进行视图的解析，根据视图逻辑名解析成真正的视图（view）
（6）视图 View（需要程序员开发 jsp）
View 是一个接口， 它的实现类支持不同的视图类型（jsp，freemarker，pdf 等等）

5、SpringMVC 和 struts2 的区别有哪些？

（1）SpringMVC 的入口是一个 Servlet 即前端控制器（DispatchServlet），而 struts2 入口是一个 filter 过虑器（StrutsPrepareAndExecuteFilter）。
（2）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(建议单例)，struts2 是基于类开发，传递参数是通过类的属性，只能设计为多例。
（3）Struts 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。JSP 视图解析器默认使用 JSTL。

6、SpringMVC 怎么样设定重定向和转发的？

（1）转发：在返回值前面加"forward:"，譬如"forward:user.do？name=method4"
（2）重定向：在返回值前面加"redirect:"，譬如"redirect:http://www.baidu.com"

7、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。

8、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 中配置一个 CharacterEncodingFilter 过滤器，设置成 utf-8；

```
<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" redirectPort="844
3"/>
```

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。

9、SpringMVC 的异常处理 ？

答：可以将异常抛给 Spring 框架，由 Spring 框架来处理；我们只需要配置简单的异常处理器，在异常处理器中添视图页面即可。

10、SpringMVC 的控制器是不是单例模式,如果是,有什么问题,怎么解决？

答：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。

11、 SpringMVC 常用的注解有哪些？

@RequestMapping：用于处理请求 url 映射的注解，可用于类或方法上。用于类上，则表示类中的所有响应请求的方法都是以该地址作为父路径。

@RequestBody：注解实现接收 http 请求的 json 数据，将 json 转换为 Java 对象。

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

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

答：一般用@Conntroller 注解,表示是表现层,不能用别的注解代替。

13、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。

14、怎样在方法里面得到 Request,或者 Session?

答：直接在方法的形参中声明 request, SpringMVC 就自动把 request 对象传入。

15、如果想在拦截的方法里面得到从前台传入的参数,怎么得到?

答：直接在形参里面声明这个参数就可以,但必须名字和传过来的参数一样。

16、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。

17、SpringMVC 中函数的返回值是什么?

答：返回值可以有很多类型,有 String, ModelAndView。ModelAndView 类把视图和数据都合并的一起的，但一般用 String 比较好。

18、SpringMVC 用什么对象从后台向前台传递数据的?

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

19、怎么样把 ModelMap 里面的数据放入 Session 里面?

答：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。

20、SpringMVC 里面拦截器是怎么写的?

有两种写法,一种是实现 HandlerInterceptor 接口，另外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.zwp.action.MyHandlerInterceptor"></bean>
    <!-- 只针对部分请求拦截 -->
    <mvc:interceptor>
        <mvc:mapping path="/modelMap.do" />
        <bean class="com.zwp.action.MyHandlerInterceptorAdapter" />
    </mvc:interceptor>
</mvc:interceptors>
```

（三）MyBatis 框架

1、#{}和${}的区别是什么？

答：${}是 Properties 文件中的变量占位符，它可以用于标签属性值和 sql 内部，属于静态文本替换，比如${driver}会被静态替换为 com.MySQL.jdbc.Driver。#{}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()。

2、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？

答：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。

3、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？

答：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。

4、Mybatis 是如何进行分页的？分页插件的原理是什么？

答：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

5、简述 Mybatis 的插件运行原理，以及如何编写一个插件。

答：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。

6、Mybatis 执行批量插入，能返回数据库主键列表吗？

答：能，JDBC 都能，Mybatis 当然也能。

7、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？

答：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。

8、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？

答：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。

9、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。

答：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。

那么问题来了，join 查询出来 100 条记录，如何确定主对象是 5 个，而不是 100 个？其去重复的原理是<resultMap>标签内的<id>子标签，指定了唯一确定一条记录的 id 列，Mybatis 根据<id>列值来完成 100 条记录的去重复功能，<id>可以有多个，代表了联合主键的语意。

同样主对象的关联对象，也是根据这个原理去重复的，尽管一般情况下，只有主对象会有重复记录，关联对象一般不会重复。

举例：下面 join 查询出来 6 条记录，一、二列是 Teacher 对象列，第三列为 Student 对象列，Mybatis 去重复处理后，结果为 1 个老师 6 个学生，而不是 6 个老师 6 个学生。

t_id	t_name	s_id
1	teacher	38
1	teacher	39
1	teacher	40
1	teacher	41
1	teacher	42
1	teacher	43

10、Mybatis 是否支持延迟加载？如果支持，它的实现原理是什么？

答：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。

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

答：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。

12、Mybatis 中如何执行批处理？

答：使用 BatchExecutor 完成批处理。

13、Mybatis 都有哪些 Executor 执行器？它们之间的区别是什么？

答：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。

14、Mybatis 中如何指定使用哪一种 Executor 执行器？

答：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。

15、Mybatis 是否可以映射 Enum 枚举类？

答：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。

16、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？

答：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。

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

答：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。

18、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？

答：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。

面试题看似都很简单，但是想要能正确回答上来，必定是研究过源码且深入的人，而不是仅会使用的人或者用的很熟的人，以上所有面试题及其答案所涉及的内容，在我的 Mybatis 系列博客中都有详细讲解和原理分析。

（四）Struts 框架

1、描述 Struts2 的工作原理。

答:客户端发送请求--》请求经过一系列过滤器--》FilterDispatcher 通过 ActionMapper 来决定这个 Request 需要调用哪个 Action --》FilterDispatcher 把请求的处理交给 ActionProxy--》通过 ConfigurationManager 询问 Struts 配置文件（Struts.xml），找到需要调用的 Action 类--》ActionProxy 创建一个 ActionInvocation 的实例 --》调用 Action--》执行完毕，返回结果

2、Struts2 中的拦截器有什么用？列举框架提供的拦截器名称？（至少 3 种，可用中文名）

答：
1）拦截器是 struts2 核心组成部分，它提供了一种机制,使得开发者可以定义一个特定的功能模块,这个模块会在 Action 执行之前或者之后执行,也可以在 Action 执行之前阻止 Action 执行。
2）常用的拦截器有:
chain:在不同请求之间将请求参数在不同名字件转换，请求内容不变
fileUpload:提供文件上传。
i18n：记录用户选择的区域环境
logger：输出 Action 的名字
params：将请求中的参数设置到 Action 中去。

3、Struts2 有哪些优点？

答：
1）在软件设计上 Struts2 的应用可以不依赖于 Servlet API 和 struts API。Struts2 的这种设计属于无侵入式设计；
2）拦截器，实现如参数拦截注入等功能；
3）类型转换器，可以把特殊的请求参数转换成需要的类型；
4）多种表现层技术，如：JSP、freeMarker、Velocity 等；
5）Struts2 的输入校验可以对指定某个方法进行校验；
6）提供了全局范围、包范围和 Action 范围的国际化资源文件管理实现
7) 实现 MVC 模式,结构清晰,使开发者只关注业务逻辑的实现。有丰富的 tag 可以用,大大提高了开发效率。(简要)

4、什么是 OGNL，有什么用途？如何访问存放在 session 中叫 user 的对象的 username 属性

答：OGNL 是 Object-Graph Navigation Language 的缩写，也叫对象导航语言。它是 Struts 的一种功能强大的表达式语言列如：访问 session 中的 user 对象的 username 属性:注意的是：使用前需要在页面头部导入 taglib prefix="s" uri="/struts-tags"

5、在 Struts2 中如何实现转发和重定向？

答：在 struts.xml 中配置 type="redirect"(重定向)；type="redirectAction"(转发)

6、Struts2 中的 type 类型有哪些？至少写 4 种。

答：chain,redirect,redirectAction,json,dispatcher

7、Struts2 默认能解决 get 和 post 提交方式的乱码问题吗？

答：不能。struts.i18n.encoding=UTF-8 属性值只能解析 POST 提交下的乱码问题。

8、说下 Struts 的设计模式。

答：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。

9、拦截器和过滤器的区别。

答：
1）拦截器是基于 Java 反射机制的，而过滤器是基于函数回调的。
2）过滤器依赖于 Servlet 容器，而拦截器不依赖于 Servlet 容器。
3）拦截器只能对 Action 请求起作用，而过滤器则可以对几乎所有请求起作用。
4）拦截器可以访问 Action 上下文、值栈里的对象，而过滤器不能。
5）在 Action 的生命周期中，拦截器可以多次调用，而过滤器只能在容器初始化时被调用一次。

10、请你写出 Struts2 中至少 5 个的默认拦截器？

答：
fileUpload 提供文件上传功能
i18n 记录用户选择的 locale
cookies 使用配置的 name,value 来是指 cookies
checkbox 添加了 checkbox 自动处理代码，将没有选中的 checkbox 的内容设定为 false，而 html 默认情况下不提交没有选中的 checkbox。
chain 让前一个 Action 的属性可以被后一个 Action 访问，现在和 chain 类型的 result（）结合使用。
alias 在不同请求之间将请求参数在不同名字件转换，请求内容不变

11、ActionContext、ServletContext、pageContext 的区别？

答：
1）ActionContext 是当前的 Action 的上下文环境，通过 ActionContext 可以获取到 request、session、ServletContext 等与 Action 有关的对象的引用；

2）ServletContext 是域对象，一个 web 应用中只有一个 ServletContext，生命周期伴随整个 web 应用；
3）pageContext 是 JSP 中的最重要的一个内置对象，可以通过 pageContext 获取其他域对象的应用，同时它是一个域对象，作用范围只针对当前页面，当前页面结束时，pageContext 销毁， 生命周期是 JSP 四个域对象中最小的。

12、拦截器的生命周期与工作过程？

答：
1）每个拦截器都是实现了 Interceptor 接口的 Java 类；
2）init(): 该方法将在拦截器被创建后立即被调用，它在拦截器的生命周期内只被调用一次. 可以在该方法中对相关资源进行必要的初始化；
3）intercept(ActionInvocation invocation): 每拦截一个动作请求，该方法就会被调用一次；
4）destroy: 该方法将在拦截器被销毁之前被调用，它在拦截器的生命周期内也只被调用一次；
5） struts2 中有内置了 18 个拦截器。

13、用自己的话简要阐述 Struts2 的执行流程。

答： Struts 2 框架本身大致可以分为 3 个部分：核心控制器 FilterDispatcher、业务控制器 Action 和用户实现的企业业务逻辑组件。 1）核心控制器 FilterDispatcher 是 Struts 2 框架的基础，包含了框架内部的控制流程和处理机制。
2）业务控制器 Action 和业务逻辑组件是需要用户来自己实现的。用户在开发 Action 和业务逻辑组件的同时，还需要编写相关的配置文件，供核心控制器 FilterDispatcher 来使用。
Struts 2 的工作流程相对于 Struts 1 要简单，与 WebWork 框架基本相同，所以说 Struts 2 是 WebWork 的升级版本。
基本简要流程如下：
1）客户端浏览器发出 HTTP 请求。
2）根据 web.xml 配置，该请求被 FilterDispatcher 接收。
3）根据 struts.xml 配置，找到需要调用的 Action 类和方法， 并通过 IoC 方式，将值注入给 Aciton。
4）Action 调用业务逻辑组件处理业务逻辑，这一步包含表单验证。
5）Action 执行完毕，根据 struts.xml 中的配置找到对应的返回结果 result，并跳转到相应页面。
6） 返回 HTTP 响应到客户端浏览器。

14、谈谈 Struts 中的 Action Servlet。

ActionServlet 类是 Struts 框架的内置核心控制器组件，它继承了 Javax.Servlet.http.HttpServlet 类。Struts 的启动通常从加载 ActionServlet 开始。Web 容器会在首次启动或 Struts 应用的第一个请求到达时加载 ActionServlet。

一般情况下都配置 web 容器比如 tomcat 启动的时候加载 ActionServlet 类，使用<load-on-startup>1</load-on-startup>标签配置启动加载。它有如下几个功能:

1) 读取配置文件 Struts-config.xml;
2) 截取客户端 http 请求，分发到相应的 Action;
3) 从请求中获取数据填充 FormBean（如果需要）。

（五）Hibernate 框架

1、Hibernate 的三种状态之间如何转换

当对象由瞬时状态(Transient)一 save()时，就变成了持久化状态；

当我们在 Session 里存储对象的时候，实际是在 Session 的 Map 里存了一份， 也就是它的缓存里放了一份，然后，又到数据库里存了一份，在缓存里这一份叫持久对象(Persistent)。 Session 一 Close()了,它的缓存也都关闭了,整个 Session 也就失效了,这个时候，这个对象变成了游离状态(Detached)，但数据库中还是存在的。

当游离状态(Detached)update()时，又变为了持久状态(Persistent)。

当持久状态(Persistent)delete()时，又变为了瞬时状态(Transient), 此时，数据库中没有与之对应的记录。

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

update 是更新一个对象，针对的是已经存在的对象

saveOrupdate 是根据实体判断，如果没有的话就保存实体，如果有实体的话才修改（对象的存在与否都不会有人格影响）

重点：最主要的区别就在于是否进行延迟加载 load 不会立即访问数据库，当试图加载的数据不存在的时候，load 方法返回未初始化的代理对象，而 get 方法会直接访问数据库，当试图加载的数据不存在的时候，直接返回 null

(1) 如果未能发现符合条件的记录，get 方法返回 null，而 load 方法会抛出一个 ObjectNotFoundException;

(2) load 方法可返回实体的代理类实例，而 get 方法永远直接返回实体类;

(3) load 方法可以充分利用内部缓存和二级缓存中的现有数据，而 get 方法则仅仅在内部缓存中进行数据查找，如没有发现对应数据，将越过二级缓存，直接调用 SQL 完成数据读取。

3、比较 Hibernate 的三种检索策略优缺点

1) 立即检索;

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： select 语句太多；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：

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： 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。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索性能；

4、Hibernate 都支持哪些缓存策略

Read-only：这种策略适用于那些频繁读取却不会更新的数据，这是目前为止最简单和最有效的缓存策略

Read/write:这种策略适用于需要被更新的数据，比 read-only 更耗费资源，在非 JTA 环境下，每个事务需要在 session.close 和 session.disconnect()被调用

Nonstrict read/write：这种策略不保障两个同时进行的事务会修改同一块数据，这种策略适用于那些经常读取但是极少更新的数据

Transactional：这种策略是完全事务化得缓存策略，可以用在 JTA 环境下

5、Hibernate 工作原理及为什么要用？

工作原理:
step1.读取并解析配置文件
step2.读取并解析映射信息，创建 SessionFactory
step3.打开 Sesssion
step4.创建事务 Transation
step5.持久化操作
step6.提交事务
step7.关闭 Session
step8.关闭 SesstionFactory
为什么要用：
1) 对 JDBC 访问数据库的代码做了封装，大大简化了数据访问层繁琐的重复性代码。
2) Hibernate 是一个基于 JDBC 的主流持久化框架，是一个优秀的 ORM 实现。他很大程度的简化 DAO 层的编码工作
3) hibernate 使用 Java 反射机制，而不是字节码增强程序来实现透明性。
4) hibernate 的性能非常好，因为它是个轻量级框架。映射的灵活性很出色。它支持各种关系数据库，从一对一到多对多的各种复杂关系。

6、Hibernate 的查询方式

SQL、Criteria、objectcomposition

HQL：属性查询 参数查询、命名参数查询、 关联查询、 分页查询、 统计函数

7、如何优化 Hibernate？

a) 使用双向一对多关联，不使用单向一对多
b) 灵活使用单向一对多关联
c) 不用一对一，用多对一取代
d) 配置对象缓存，不使用集合缓存
e) 一对多集合使用 Bag,多对多集合使用 Set
f) 继承类使用显式多态
g) 表字段要少，表关联不要怕多，有二级缓存撑腰

8、谈谈 Hibernate 中 inverse 的作用

inverse 属性默认是 false,就是说关系的两端都来维护关系。 比如 Student 和 Teacher 是多对多关系，用一个中间表 TeacherStudent 维护。

如果 Student 这边 inverse=”true”，那么关系由另一端 Teacher 维护，就是说当插入 Student 时，不会操作 TeacherStudent 表（中间表）。只有 Teacher 插入或删除时才会触发对中间表的操作。所以两边都 inverse=”true”是不对的，会导致任何操作都不触发对中间表的影响；当两边都 inverse=”false”或默认时，会导致在中间表中插入两次关系。

9、什么是 SessionFactory,她是线程安全么？

SessionFactory 是 Hibrenate 单例数据存储和线程安全的，以至于可以多线程同时访问。一个 SessionFactory 在启动的时候只能建立一次。SessionFactory 应该包装各种单例以至于它能很简单的在一个应用代码中储存.

10、介绍一下 Hibernate 的二级缓存

（1）缓存就是把以前从数据库中查询出来和使用过的对象保存在内存中（一个数据结构中），这个数据结构通常是或类似 Hashmap，当以后要使用某个对象时，先查询缓存中是否有这个对象，如果有则使用缓存中的对象，如果没有则去查询数据库，并将查询出来的对象保存在缓存中，以便下次使用。下面是缓存的伪代码：

引出 hibernate 的第二级缓存，用下面的伪代码分析了 Cache 的实现原理

```
Dao
{
hashmap map = new map();
User getUser(integer id)
{
User user = map.get(id)
if(user == null)
{
user = session.get(id);
map.put(id,user);
```

```
}
return user;
}
}

Dao
{
Cache cache = null
setCache(Cache cache)
{
this.cache = cache
}

User getUser(int id)
{
if(cache!=null)
{
User user = cache.get(id);
if(user ==null)
{
user = session.get(id);
cache.put(id,user);
}
return user;
}

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

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

（3）二级缓存是独立于 Hibernate 的软件部件，属于第三方的产品，多个厂商和组织都提供有缓存产品，例如，EHCache 和 OSCache 等等。在 Hibernate 中使用二级缓存，首先就要在 hibernate.cfg.xml 配置文件中配置使用哪个厂家的

缓存产品，接着需要配置该缓存产品自己的配置文件，最后要配置 Hibernate 中的哪些实体对象要纳入到二级缓存的管理中。明白了二级缓存原理和有了这个思路后，很容易配置起 Hibernate 的二级缓存。扩展知识：一个 SessionFactory 可以关联一个二级缓存，也即一个二级缓存只能负责缓存一个数据库中的数据，当使用 Hibernate 的二级缓存后，注意不要有其他的应用或 SessionFactory 来更改当前数据库中的数据，这样缓存的数据就会与数据库中的实际数据不一致。

11、Spring+Hibernate 中委托方案怎么配置？

解决方案一，按照 Object[]数据取出数据，然后自己组 bean
解决方案二，对每个表的 bean 写构造函数，比如表一要查出 field1,field2 两个字段，那么有一个构造函数就是 Bean(type1 filed1,type2 field2) ，然后在 hql 里面就可以直接生成这个 bean 了。

（六）框架整合

1、谈谈你对 Struts 的理解。

1. struts 是一个按 MVC 模式设计的 Web 层框架，其实它就是一个大大的 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/WEB-INF/struts-config.xml）将各个请求分别分配给不同的 action 去处理。

一个扩展知识点：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；

2. ActionServlet 把请求交给 action 去处理之前，会将请求参数封装成一个 formbean 对象（就是一个 Java 类，这个类中的每个属性对应一个请求参数），封装成一个什么样的 formbean 对象呢？看配置文件。

3.要说明的是， ActionServlet 把 formbean 对象传递给 action 的 execute 方法之前，可能会调用 formbean 的 validate 方法进行校验，只有校验通过后才将这个 formbean 对象传递给 action 的 execute 方法，否则，它将返回一个错误页面，这个错误页面由 input 属性指定，（看配置文件）作者为什么将这里命名为 input 属性，而不是 error 属性，我们后面结合实际的运行效果进行分析。

4.action 执行完后要返回显示的结果视图，这个结果视图是用一个 ActionForward 对象来表示的，actionforward 对象通过 struts-config.xml 配置文件中的配置关联到某个 jsp 页面，因为程序中使用的是在 struts-config.xml 配置文件为 jsp 页面设置的逻辑名，这样可以实现 action 程序代码与返回的 jsp 页面名称的解耦。

你对 struts 可能还有自己的应用方面的经验，那也要一并说出来。

2、谈谈你对 Hibernate 的理解。

答:

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。

4.使用 Hibernate 的基本流程是：配置 Configuration 对象、产生 SessionFactory、创建 session 对象，启动事务，完成 CRUD 操作，提交事务，关闭 session。

5.使用 Hibernate 时，先要配置 hibernate.cfg.xml 文件，其中配置数据库连接信息和方言等，还要为每个实体配置相应的 hbm.xml 文件，hibernate.cfg.xml 文件中需要登记每个 hbm.xml 文件。

6.在应用 Hibernate 时，重点要了解 Session 的缓存原理，级联，延迟加载和 hql 查询。

3、你对 Spring 的理解。

1.Spring 实现了工厂模式的工厂类（在这里有必要解释清楚什么是工厂模式），这个类名为 BeanFactory（实际上是一个接口），在程序中通常 BeanFactory 的子类 ApplicationContext。Spring 相当于一个大的工厂类，在其配置文件中通过<bean>元素配置用于创建实例对象的类名和实例对象的属性。

2. Spring 提供了对 IOC 良好支持，IOC 是一种编程思想，是一种架构艺术，利用这种思想可以很好地实现模块之间的解耦。IOC 也称为 DI（Depency Injection），什么叫依赖注入呢？

譬如，Class Programmer

```
{
Computer computer = null;
public void code()
{
//Computer computer = new IBMComputer();
//Computer computer = beanfacotry.getComputer();
```

```
computer.write();
}
public void setComputer(Computer computer)
{
this.computer = computer;
}
}
```

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

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

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

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

4、Struts 优缺点 。

优点：

1) 实现 MVC 模式，结构清晰,使开发者只关注业务逻辑的实现.
2) 有丰富的 tag 可以用 ,Struts 的标记库(Taglib)，如能灵活动用，则能大大提高开发效率
3) 页面导航使系统的脉络更加清晰。通过一个配置文件，即可把握整个系统各部分之间的联系，这对于后期的维护有着莫大的好处。尤其是当另一批开发者接手这个项目时，这种优势体现得更加明显。
4) 提供 Exception 处理机制 .
5) 数据库链接池管理
6) 支持 I18N。

缺点

1) 转到展示层时，需要配置 forward，如果有十个展示层的 jsp，需要配置十次 struts，而且还不包括有时候目录、文件变更，需要重新修改 forward，注意，每次修改配置之后，要求重新部署整个项目，而 tomcate 这样的服务器，还必须重新启动服务器
2) Struts 的 Action 必需是 thread－safe 方式，它仅仅允许一个实例去处理所有的请求。所以 action 用到的所有的资源都必需统一同步，这个就引起了线程安全的问题。
3) 测试不方便. Struts 的每个 Action 都同 Web 层耦合在一起，这样它的测试依赖于 Web 容器，单元测试也很难实现。不过有一个 Junit 的扩展工具 Struts TestCase 可以实现它的单元测试。
4) 类型的转换. Struts 的 FormBean 把所有的数据都作为 String 类型，它可以使用工具 Commons-Beanutils 进行类型转化。但它的转化都是在 Class 级别，而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。
5) 对 Servlet 的依赖性过强. Struts 处理 Action 时必需要依赖 ServletRequest 和 ServletResponse，所有它摆脱不了 Servlet 容器。
6) 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。可是 JSTL 的表达式语言在 Collection 和索引属性方面处理显得很弱。
7) 对 Action 执行的控制困难. Struts 创建一个 Action，如果想控制它的执行顺序将会非常困难。甚至你要重新去写 Servlet 来实现你的这个功能需求。
8) 对 Action 执行前和后的处理. Struts 处理 Action 的时候是基于 class 的 hierarchies，很难在 action 处理前和后进行操作。
9) 对事件支持不够. 在 struts 中，实际是一个表单 Form 对应一个 Action 类(或 DispatchAction)，换一句话说：在 Struts 中实际是一个表单只能对应一个事件，struts 这种事件方式称为 application event，application event 和 component event 相比是一种粗粒度的事件

5、Struts 的应用(如 Struts 架构)

Struts 是采用 Java Servlet/JavaServer Pages 技术，开发 Web 应用程序的开放源码的 framework。 采用 Struts 能开发出基于 MVC(Model-View-Controller)设计模式的应用构架。 Struts 有如下的主要功能： 一.包含一个 controller Servlet，能将用户的请求发送到相应的 Action 对象。 二.JSP 自由 tag 库，并且在 controller Servlet 中提供关联支持，帮助开发员创建交互式表单应用。 三.提供了一系列实用对象：XML 处理、通过 Java reflection APIs 自动处理 JavaBeans 属性、国际化的提示和消息。

6、说说 Struts1 与 Struts2 的区别。

二者都是 MVC 的 WEB 框架。

1) struts1 的老牌框架，应用很广泛，有很好的群众基础，使用它开发风险很小，成本更低！struts2 虽然基于这个框架，但是应用群众并多，相对不成

熟，未知的风险和变化很多，开发人员相对不好招，使用它开发项目的风险系数更大，用人成本更高！

2) struts2 毕竟是站在前辈的基础设计出来，它会改善和完善 struts1 中的一些缺陷，struts1 中一些悬而未决问题在 struts2 得到了解决。
3) struts1 的前端控制器是一个 Servlet，名称为 ActionServlet，struts2 的前端控制器是一个 filter，在 struts2.0 中叫 FilterDispatcher，在 struts2.1 中叫 StrutsPrepareAndExecuteFilter。
4) struts1 的 action 需要继承 Action 类，struts2 的 action 可以不继承任何类；struts1 对同一个路径的所有请求共享一个 Action 实例，struts2 对同一个路径的每个请求分别使用一个独立 Action 实例对象，所有对于 struts2 的 Action 不用考虑线程安全问题。
5) 在 struts1 中使用 formbean 封装请求参数，在 struts2 中直接使用 action 的属性来封装请求参数。
6) struts1 中的多个业务方法放在一个 Action 中时（即继承 DispatchAction 时），要么都校验，要么都不校验；对于 struts2，可以指定只对某个方法进行校验，当一个 Action 继承了 ActionSupport 且在这个类中只编写了 validateXxx()方法，那么则只对 Xxx()方法进行校验。
7) 与 Struts1 不同，Struts2 对用户的每一次请求都会创建一个 Action，所以 Struts2 中的 Action 是线程安全的。
8) 给我印象最深刻的是：struts 配置文件中的 redirect 视图的 url 不能接受参数，而 struts2 配置文件中的 redirect 视图可以接受参数。

7、简述 Hibernate 和 JDBC 的区别和优缺点？ 如何书写一个 one to many 配置文件.

JDBC 与 Hibernate 在性能上相比，JDBC 灵活性有优势。而 Hibernate 在易学性，易用性上有些优势。当用到很多复杂的多表联查和复杂的数据库操作时，JDBC 有优势。

相同点：

1) 两者都是 JAVA 的数据库操作中间件。
2) 两者对于数据库进行直接操作的对象都不是线程安全的，都需要及时关闭。
3) 两者都可以对数据库的更新操作进行显式的事务处理。

不同点：

1) 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
2) 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。
3) 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。

JDBC 与 Hibernate 读取性能

1) JDBC 仍然是最快的访问方式，不论是 Create 还是 Read 操作，都是 JDBC 快。
2) Hibernate 使用 uuid.hex 构造主键，性能稍微有点损失，但是不大。
3) Create 操作，JDBC 在使用批处理的方式下速度比 Hibernate 快，使用批处理方式耗用 JVM 内存比不使用批处理方式要多得多。
4) 读取数据，Hibernate 的 Iterator 速度非常缓慢，因为他是每次 next 的时候才去数据库取数据，这一点从观察任务管理器的 Java 进程占用内存的变化也可以看得很清楚，内存是几十 K 几十 K 的增加。
5) 读取数据，Hibernate 的 List 速度很快，因为他是一次性把数据取完，这一点从观察任务管理器的 Java 进程占用内存的变化也可以看得很清楚，内存几乎是 10M 的 10M 的增加。
6) JDBC 读取数据的方式和 Hibernate 的 List 方式是一样的（这跟 JDBC 驱动有很大关系，不同的 JDBC 驱动，结果会很不一样），这从观察 Java 进程内存变化可以判断出来，由于 JDBC 不需要像 Hibernate 那样构造一堆 Cat 对象实例，所以占用 JVM 内存要比 Hibernate 的 List 方式大概少一半左右。
7) Hibernate 的 Iterator 方式并非一无是处，它适合于从大的结果集中选取少量的数据，即不需要占用很多内存，又可以迅速得到结果。另外 Iterator 适合于使用 JCS 缓冲。

8、MyBatis 与 Hibernate 有什么不同？

相同点：屏蔽 jdbc API 的底层访问细节，使用我们不用与 jdbc API 打交道，就可以访问数据。

jdbc API 编程流程固定，还将 sql 语句与 Java 代码混杂在了一起，经常需要拼凑 sql 语句，细节很繁琐。

ibatis 的好处：屏蔽 jdbc API 的底层访问细节；将 sql 语句与 Java 代码进行分离;提供了将结果集自动封装称为实体对象和对象的集合的功能，queryForList 返回对象集合，用 queryForObject 返回单个对象；提供了自动将实体对象的属性传递给 sql 语句的参数。

Hibernate 是一个全自动的 orm 映射工具，它可以自动生成 sql 语句,ibatis 需要我们自己在 xml 配置文件中写 sql 语句，hibernate 要比 ibatis 功能负责和强大很多。因为 hibernate 自动生成 sql 语句，我们无法控制该语句，我们就无法去写特定的高效率的 sql。对于一些不太复杂的 sql 查询，hibernate 可以很好帮我们完成，但是，对于特别复杂的查询，hibernate 就很难适应了，这时候用 ibatis 就是不错的选择，因为 ibatis 还是由我们自己写 sql 语句。

9、Jdo 是什么？

JDO 是 Java 对象持久化的新的规范，为 Java data object 的简称,也是一个用于存取某种数据仓库中的对象的标准化 API。JDO 提供了透明的对象存储，因此对开发人员来说，存储数据对象完全不需要额外的代码（如 JDBC API 的使用）。这些繁琐的例行工作已经转移到 JDO 产品提供商身上，使开发人员解脱出来，从而集中时间和精力在业务逻辑上。另外，JDO 很灵活，因为它可以在

任何数据底层上运行。JDBC 只是面向关系数据库（RDBMS）JDO 更通用，提供到任何数据底层的存储功能，比如关系数据库、文件、XML 以及对象数据库（ODBMS）等等，使得应用可移植性更强。

六.常用互联网技术

（一）Dubbo

1、Dubbo 是什么？

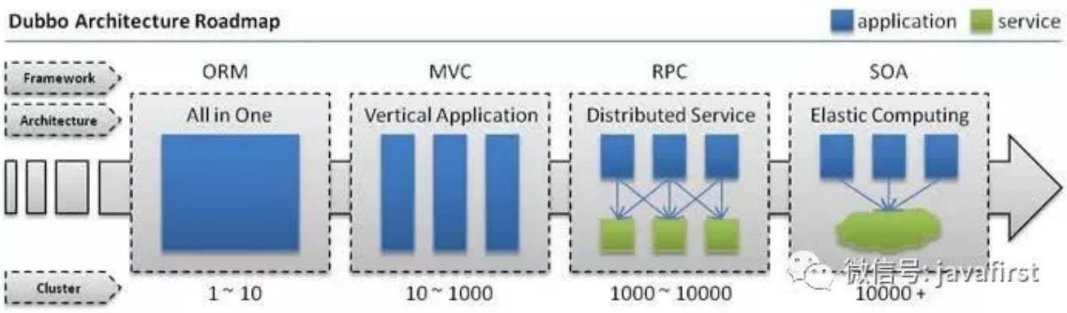
Dubbo 是阿里巴巴开源的基于 Java 的高性能 RPC 分布式服务框架，现已成为 Apache 基金会孵化项目。

2、为什么要用 Dubbo？

因为是阿里开源项目，国内很多互联网公司都在用，已经经过很多线上考验。内部使用了 Netty、Zookeeper，保证了高性能高可用性。

使用 Dubbo 可以将核心业务抽取出来，作为独立的服务，逐渐形成稳定的服务中心，可用于提高业务复用灵活扩展，使前端应用能更快速的响应多变的市场需求。

下面这张图可以很清楚的诠释，最重要的一点是，分布式架构可以承受更大规模的并发流量。



3、Dubbo 和 Spring Cloud 有什么区别？

两个没关联，如果硬要说区别，有以下几点。

1）通信方式不同

Dubbo 使用的是 RPC 通信，而 Spring Cloud 使用的是 HTTP RESTFul 方式。

2）组成部分不同

组件	Dubbo	Spring Cloud
服务注册中心	Zookeeper	Spring Cloud Netflix Eureka
服务监控	Dubbo-monitor	Spring Boot Admin
断路器	不完善	Spring Cloud Netflix Hystrix
服务网关	无	Spring Cloud Netflix Gateway
分布式配置	无	Spring Cloud Config
服务跟踪	无	Spring Cloud Sleuth
消息总线	无	Spring Cloud Bus
数据流	无	Spring Cloud Stream
批量任务	无	Spring Cloud Task
...	...	... 微信号: javafirst

4、Dubbo 都支持什么协议，推荐用哪种？

dubbo://（推荐）
rmi://
hessian://
http://
WebService://
thrift://
memcached://
redis://
rest://

5、Dubbo 需要 Web 容器吗？

不需要，如果硬要用 Web 容器，只会增加复杂性，也浪费资源。

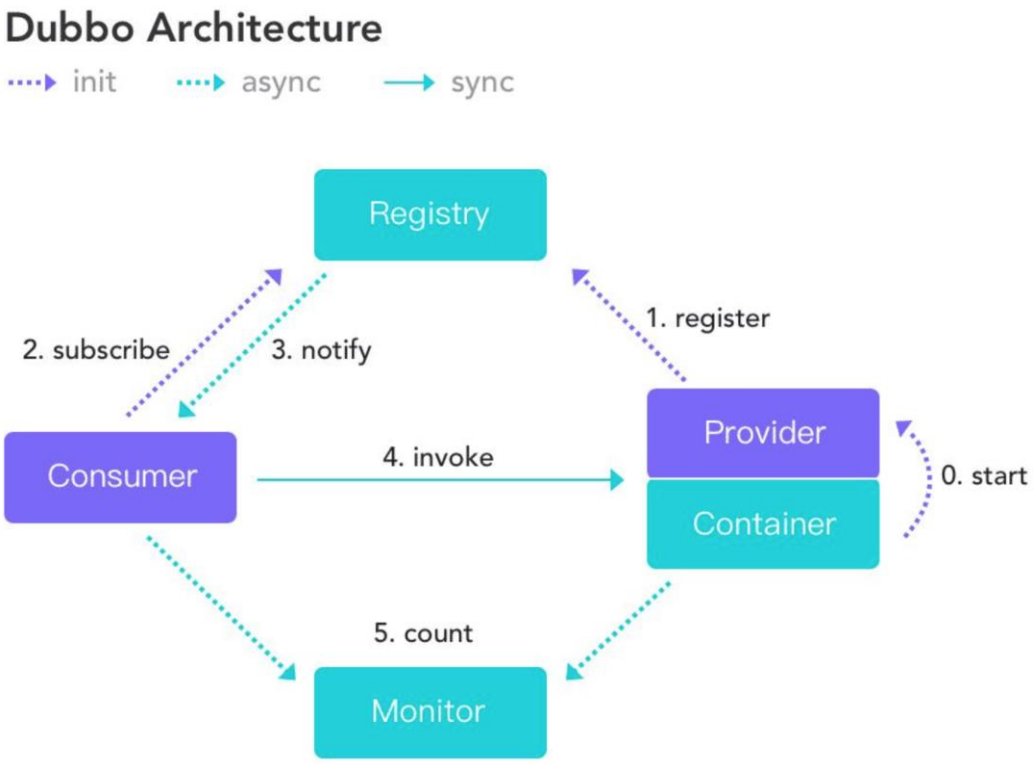
6、Dubbo 内置了哪几种服务容器？

Spring Container
Jetty Container
Log4j Container
Dubbo 的服务容器只是一个简单的 Main 方法，并加载一个简单的 Spring 容器，用于暴露服务。

7、Dubbo 里面有哪几种节点角色？

节点	角色说明
Provider	暴露服务的服务提供方
Consumer	调用远程服务的服务消费方
Registry	服务注册与发现的注册中心
Monitor	统计服务的调用次数和调用时间的监控中心
Container	服务运行容器

8、画一画服务注册与发现的流程图。



9、Dubbo 默认使用什么注册中心，还有别的选择吗？

推荐使用 Zookeeper 作为注册中心，还有 Redis、Multicast、Simple 注册中心，但不推荐。

10、Dubbo 有哪几种配置方式？

1）Spring 配置方式

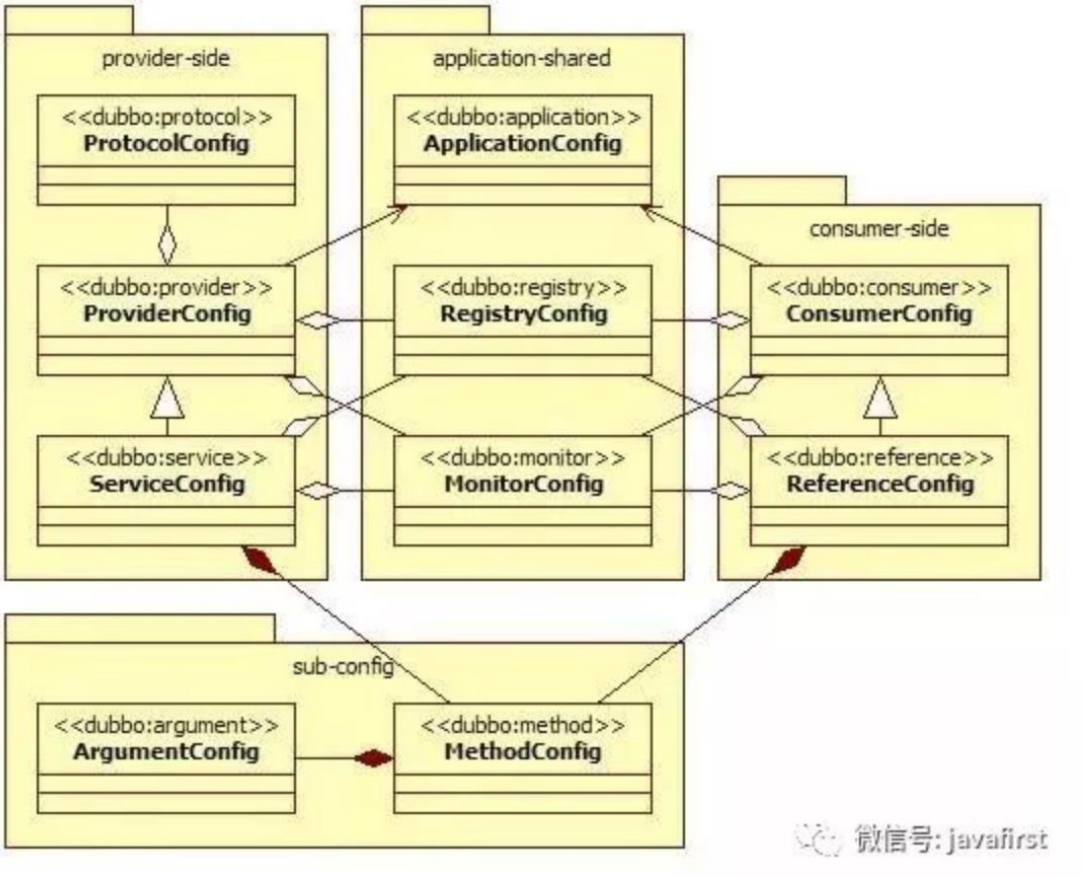
2）Java API 配置方式

11、Dubbo 核心的配置有哪些？

我曾经面试就遇到过面试官让你写这些配置，我也是蒙逼。

配置	配置说明
dubbo:service	服务配置
dubbo:reference	引用配置
dubbo:protocol	协议配置
dubbo:application	应用配置
dubbo:module	模块配置
dubbo:registry	注册中心配置
dubbo:monitor	监控中心配置
dubbo:provider	提供方配置
dubbo:consumer	消费方配置
dubbo:method	方法配置
dubbo:argument	参数配置

配置之间的关系见下图。



12、在 Provider 上可以配置的 Consumer 端的属性有哪些？

1）timeout：方法调用超时
2）retries：失败重试次数，默认重试 2 次
3）loadbalance：负载均衡算法，默认随机
4）actives 消费者端，最大并发调用限制

13、Dubbo 启动时如果依赖的服务不可用会怎样？

Dubbo 缺省会在启动时检查依赖的服务是否可用，不可用时会抛出异常，阻止 Spring 初始化完成，默认 check="true"，可以通过 check="false" 关闭检查。

14、Dubbo 推荐使用什么序列化框架，你知道的还有哪些？

推荐使用 Hessian 序列化，还有 Duddo、FastJson、Java 自带序列化。

15、Dubbo 默认使用的是什么通信框架，还有别的选择吗？

Dubbo 默认使用 Netty 框架，也是推荐的选择，另外内容还集成有 Mina、Grizzly。

16、Dubbo 有哪几种集群容错方案，默认是哪种？

集群容错方案	说明
Failover Cluster	失败自动切换，自动重试其它服务器（默认）
Failfast Cluster	快速失败，立即报错，只发起一次调用
Failsafe Cluster	失败安全，出现异常时，直接忽略
Failback Cluster	失败自动恢复，记录失败请求，定时重发
Forking Cluster	并行调用多个服务器，只要一个成功即返回
Broadcast Cluster	广播逐个调用所有提供者，任意一个报错则报错

17、Dubbo 有哪几种负载均衡策略，默认是哪种？

负载均衡策略	说明
Random LoadBalance	随机，按权重设置随机概率（默认）
RoundRobin LoadBalance	轮询，按公约后的权重设置轮询比率
LeastActive LoadBalance	最少活跃调用数，相同活跃数的随机
ConsistentHash LoadBalance	一致性 Hash，相同参数的请求总是发到同一提供者

18、注册了多个同一样的服务，如果测试指定的某一个服务呢？

可以配置环境点对点直连，绕过注册中心，将以服务接口为单位，忽略注册中心的提供者列表。

19、Dubbo 支持服务多协议吗？

Dubbo 允许配置多协议，在不同服务上支持不同协议或者同一服务上同时支持多种协议。

20、当一个服务接口有多种实现时怎么做？

当一个接口有多种实现时，可以用 group 属性来分组，服务提供方和消费方都指定同一个 group 即可。

21、服务上线怎么兼容旧版本？

可以用版本号（version）过渡，多个不同版本的服务注册到注册中心，版本号不同的服务相互间不引用。这个和服务分组的概念有一点类似。

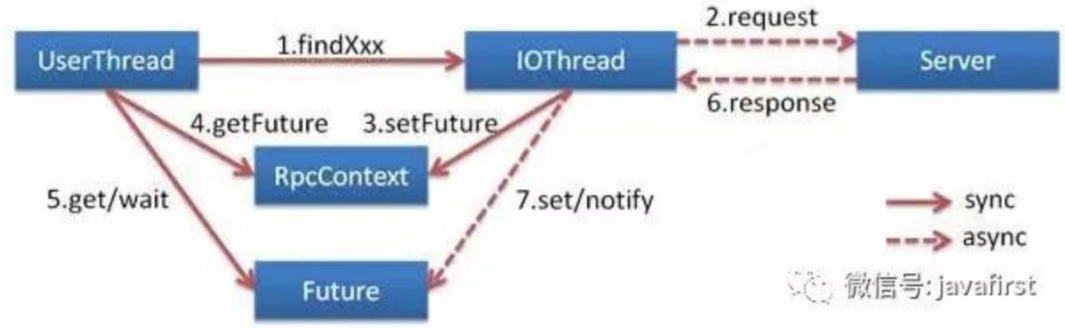
22、Dubbo 可以对结果进行缓存吗？

可以，Dubbo 提供了声明式缓存，用于加速热门数据的访问速度，以减少用户加缓存的工作量。

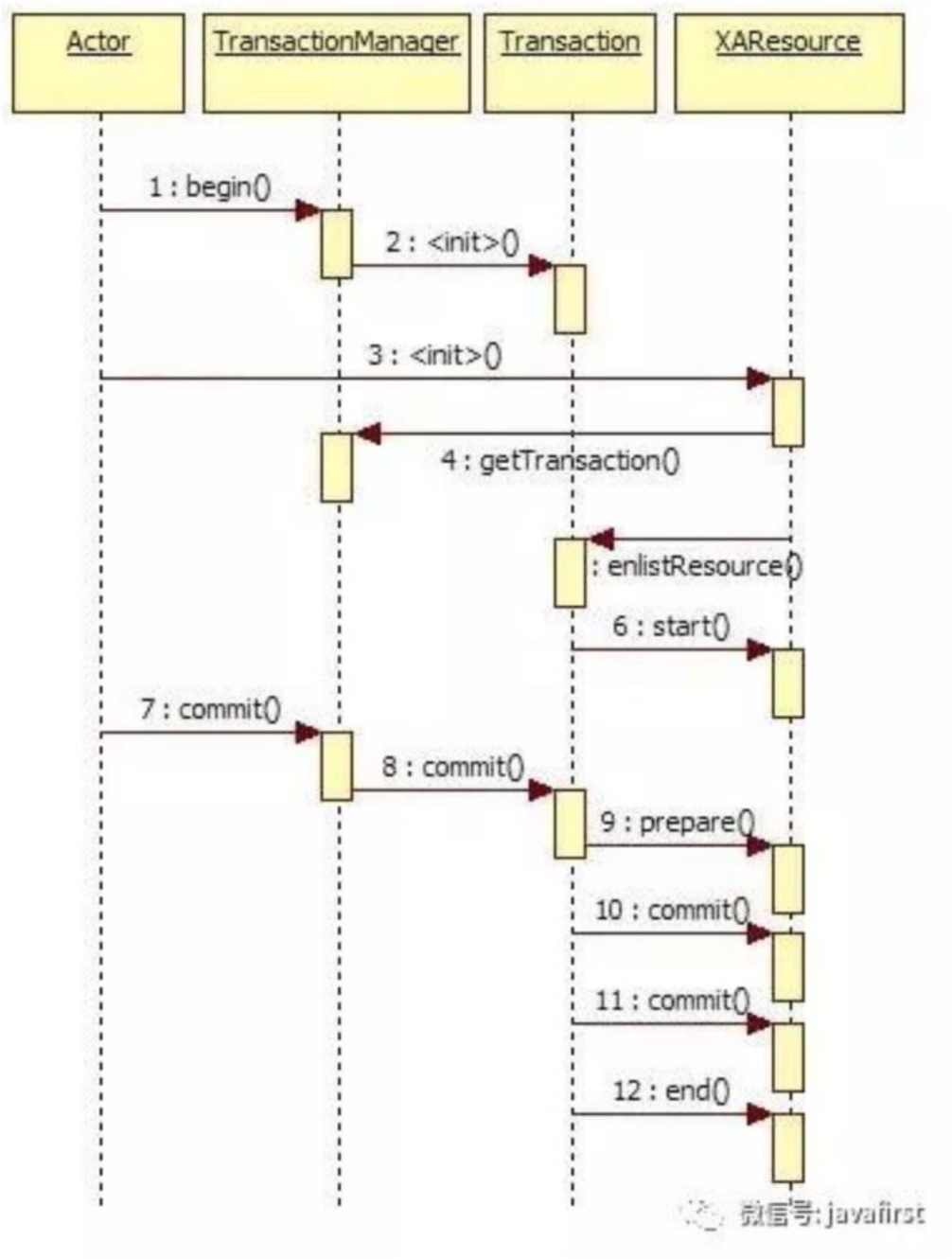
23、Dubbo 服务之间的调用是阻塞的吗？

默认是同步等待结果阻塞的，支持异步调用。

Dubbo 是基于 NIO 的非阻塞实现并行调用，客户端不需要启动多线程即可完成并行调用多个远程服务，相对多线程开销较小，异步调用会返回一个 Future 对象。
异步调用流程图如下。



24、Dubbo 支持分布式事务吗？



25、Dubbo telnet 命令能做什么？

dubbo 通过 telnet 命令来进行服务治理，具体使用看这篇文章《dubbo 服务调试管理实用命令》。
telnet localhost 8090

26、Dubbo 支持服务降级吗？

Dubbo 2.2.0 以上版本支持。

27、Dubbo 如何优雅停机？

Dubbo 是通过 JDK 的 ShutdownHook 来完成优雅停机的，所以如果使用 kill -9 PID 等强制关闭指令，是不会执行优雅停机的，只有通过 kill PID 时，才会执行。

28、服务提供者能实现失效踢出是什么原理？

服务失效踢出基于 Zookeeper 的临时节点原理。

29、如何解决服务调用链过长的问题？

Dubbo 可以使用 Pinpoint 和 Apache Skywalking(Incubator) 实现分布式服务追踪，当然还有其他很多方案。

30、服务读写推荐的容错策略是怎样的？

读操作建议使用 Failover 失败自动切换，默认重试两次其他服务器。
写操作建议使用 Failfast 快速失败，发一次调用失败就立即报错。

31、Dubbo 必须依赖的包有哪些？

Dubbo 必须依赖 JDK，其他为可选。

32、Dubbo 的管理控制台能做什么？

管理控制台主要包含：路由规则，动态配置，服务降级，访问控制，权重调整，负载均衡，等管理功能。

33、说说 Dubbo 服务暴露的过程。

Dubbo 会在 Spring 实例化完 bean 之后，在刷新容器最后一步发布 ContextRefreshEvent 事件的时候，通知实现了 ApplicationListener 的 ServiceBean 类进行回调 onApplicationEvent 事件方法，Dubbo 会在这个方法中调用 ServiceBean 父类 ServiceConfig 的 export 方法，而该方法真正实现了服务的（异步或者非异步）发布。

34、Dubbo 停止维护了吗？

2014 年开始停止维护过几年，17 年开始重新维护，并进入了 Apache 项目。

35、Dubbo 和 Dubbox 有什么区别？

Dubbox 是继 Dubbo 停止维护后，当当网基于 Dubbo 做的一个扩展项目，如加了服务可 Restful 调用，更新了开源组件等。

36、你还了解别的分布式框架吗？

别的还有 Spring cloud、Facebook 的 Thrift、Twitter 的 Finagle 等。

37、Dubbo 能集成 Spring Boot 吗？

可以的，项目地址如下。
https://github.com/apache/incubator-dubbo-Spring-boot-project

38、在使用过程中都遇到了些什么问题？

Dubbo 的设计目的是为了满足高并发小数据量的 rpc 调用，在大数据量下的性能表现并不好，建议使用 rmi 或 http 协议。

（二）Redis 部分

1、什么是 Redis？简述它的优缺点？

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，因此 Redis 可以用来实现很多有用的功能。比方说用他的 List 来做 FIFO 双向链表，实现一个轻量级的高性 能消息队列服务，用他的 Set 可以做高性能的 tag 系统等等。

另外 Redis 也可以对存入的 Key-Value 设置 expire 时间，因此也可以被当作一 个功能加强版的 memcached 来用。 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、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

3、Redis 支持哪几种数据类型？

String、List、Set、Sorted Set、hashes

4、Redis 主要消耗什么物理资源？

内存。

5、Redis 的全称是什么？

Remote Dictionary Server。

6、Redis 有哪几种数据淘汰策略？

noeviction:返回错误当内存限制达到并且客户端尝试执行会让更多内存被使用的命令（大部分的写入指令，但 DEL 和几个例外）

allkeys-lru: 尝试回收最少使用的键（LRU），使得新添加的数据有空间存放。volatile-lru: 尝试回收最少使用的键（LRU），但仅限于在过期集合的键,使得新添加的数据有空间存放。

allkeys-random: 回收随机的键使得新添加的数据有空间存放。

volatile-random: 回收随机的键使得新添加的数据有空间存放，但仅限于在过期集合的键。

volatile-ttl: 回收在过期集合的键，并且优先回收存活时间（TTL）较短的键,使得新添加的数据有空间存放。

7、Redis 官方为什么不提供 Windows 版本？

因为目前 Linux 版本已经相当稳定，而且用户量很大，无需开发 windows 版本，反而会带来兼容性等问题。

8、一个字符串类型的值能存储最大容量是多少？

512M

9、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。

10、Redis 集群方案应该怎么做？都有哪些方案？

1.codis。目前用的最多的集群方案，基本和 twemproxy 一致的效果，但它支持在 节点数量改变情况下，旧节点数据可恢复到新 hash 节点。

2.redis cluster3.0 自带的集群，特点在于他的分布式算法不是一致性 hash，而是 hash 槽的概念，以及自身支持节点设置从节点。具体看官方文档介绍。

3.在业务代码层实现，起几个毫无关联的 redis 实例，在代码层，对 key 进行 hash 计算，然后去对应的 redis 实例操作数据。 这种方式对 hash 层代码要求比较高，考虑部分包括，节点失效后的替代算法方案，数据震荡后的自动脚本恢复，实例的监控，等等。

11、Redis 集群方案什么情况下会导致整个集群不可用？

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。

12、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？

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

13、Redis 有哪些适合的场景？

（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 WITHSCORESAgora 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！

14、Redis 支持的 Java 客户端都有哪些？

官方推荐用哪个？Redisson、Jedis、lettuce 等等，官方推荐使用 Redisson。

15、Redis 和 Redisson 有什么关系？

Redisson 是一个高级的分布式协调 Redis 客服端，能帮助用户在分布式环境中轻松实现一些 Java 的对象 (Bloom filter, BitSet, Set, SetMultimap, ScoredSortedSet, SortedSet, Map, ConcurrentMap, List, ListMultimap, Queue, BlockingQueue, Deque, BlockingDeque, Semaphore, Lock, ReadWriteLock, AtomicLong, CountDownLatch, Publish / Subscribe, HyperLogLog)。

16、Jedis 与 Redisson 对比有什么优缺点？

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。

17、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

18、说说 Redis 哈希槽的概念？

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。

19、Redis 集群的主从复制模型是怎样的？

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.

20、Redis 集群会有写操作丢失吗？为什么？

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。

21、Redis 集群之间是如何复制的？

异步复制

22、Redis 集群最大节点个数是多少？

16384 个。

23、Redis 集群如何选择数据库？

Redis 集群目前无法做数据库选择，默认在 0 数据库。

24、怎么测试 Redis 的连通性？

ping

25、Redis 中的管道有什么用？

一次请求/响应服务器能实现处理新的请求即使旧的请求还未被响应。这样就可以将多个命令发送到服务器，而不用等待回复，最后在一个步骤中读取该答复。这就是管道（pipelining），是一种几十年来广泛使用的技术。例如许多 POP3 协议已经实现支持这个功能，大大加快了从服务器下载新邮件的过程。

26、怎么理解 Redis 事务？

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。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。

27、Redis 事务相关的命令有哪几个？

MULTI、EXEC、DISCARD、WATCH

28、Redis key 的过期时间和永久有效分别怎么设置？

EXPIRE 和 PERSIST 命令。

29、Redis 如何做内存优化？

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。

30、Redis 回收进程如何工作的？

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。

（三）Nginx 部分

1、请解释一下什么是 Nginx？

答：Nginx 是一个 web 服务器和反向代理服务器，用于 HTTP、HTTPS、SMTP、POP3 和 IMAP 协议。

2、请列举 Nginx 的一些特性？

答：Nginx 服务器的特性包括：
1）反向代理/L7 负载均衡器
2）嵌入式 Perl 解释器
3）动态二进制升级
4）可用于重新编写 URL，具有非常好的 PCRE 支持

3、Nginx 和 Apache 的区别？

1）轻量级，同样起 web 服务，比 apache 占用更少的内存及资源
2）抗并发，Nginx 处理请求是异步非阻塞的，而 apache 则是阻塞型的，在高并发下 Nginx 能保持低资源低消耗高性能
3）高度模块化的设计，编写模块相对简单
4）最核心的区别在于 apache 是同步多进程模型，一个连接对应一个进程；Nginx 是异步的，多个连接（万级别）可以对应一个进程

4、Nginx 是如何实现高并发的

答：一个主进程，多个工作进程，每个工作进程可以处理多个请求，每进来一个 request，会有一个 worker 进程去处理。但不是全程的处理，处理到可能发生阻塞的地方，比如向上游（后端）服务器转发 request，并等待请求返回。那么，这个处理的 worker 继续处理其他请求，而一旦上游服务器返回了，就会触发这个事件，worker 才会来接手，这个 request 才会接着往下走。由于 web server 的工作性质决定了每个 request 的大部份生命都是在网络传输中，实际上花费在 server 机器上的时间片不多。这是几个进程就解决高并发的秘密所在。即@skoo 所说的 webserver 刚好属于网络 io 密集型应用，不算是计算密集型。

5、请解释 Nginx 如何处理 HTTP 请求。

答：Nginx 使用反应器模式。主事件循环等待操作系统发出准备事件的信号，这样数据就可以从套接字读取，在该实例中读取到缓冲区并进行处理。单个线程可以提供数万个并发连接。

6、在 Nginx 中，如何使用未定义的服务器名称来阻止处理请求？

答：只需将请求删除的服务器就可以定义为：

```
Server {
   listen 80; server_name “ “ ;
```

```
    return 444;
  }
```

这里，服务器名被保留为一个空字符串，它将在没有“主机”头字段的情况下匹配请求，而一个特殊的 Nginx 的非标准代码 444 被返回，从而终止连接。

7、 使用“反向代理服务器”的优点是什么？

答：反向代理服务器可以隐藏源服务器的存在和特征。它充当互联网云和 web 服务器之间的中间层。这对于安全方面来说是很好的，特别是当您使用 web 托管服务时。

8、请列举 Nginx 服务器的最佳用途。

答：Nginx 服务器的最佳用法是在网络上部署动态 HTTP 内容，使用 SCGI、WSGI 应用程序服务器、用于脚本的 FastCGI 处理程序。它还可以作为负载均衡器。

9、请解释 Nginx 服务器上的 Master 和 Worker 进程分别是什么？

Master 进程：读取及评估配置和维持
Worker 进程：处理请求

10、请解释你如何通过不同于 80 的端口开启 Nginx？

答：为了通过一个不同的端口开启 Nginx，你必须进入/etc/Nginx/sites-enabled/，如果这是默认文件，那么你必须打开名为“default”的文件。编辑文件，并放置在你想要的端口：

```
Like server {
    listen 81;
}
```

11、请解释是否有可能将 Nginx 的错误替换为 502 错误、503？

答： 502 =错误网关 503 =服务器超载 有可能，但是您可以确保 fastcgi_intercept_errors 被设置为 ON，并使用错误页面指令。

Location / { fastcgi_pass 127.0.01:9001; fastcgi_intercept_errors on; error_page 502 =503/error_page.html; #… }

12、在 Nginx 中，解释如何在 URL 中保留双斜线？

答：要在 URL 中保留双斜线，就必须使用 merge_slashes_off;
语法:merge_slashes [on/off]
默认值: merge_slashes on
环境: http，server

13、请解释 ngx_http_upstream_module 的作用是什么？

答：ngx_http_upstream_module 用于定义可通过 fastcgi 传递、proxy 传递、uwsgi 传递、memcached 传递和 scgi 传递指令来引用的服务器组。

14、请解释什么是 C10K 问题？

答：C10K 问题是指无法同时处理大量客户端(10,000)的网络套接字。

15、请陈述 stub_status 和 sub_filter 指令的作用是什么？

1）Stub_status 指令：该指令用于了解 Nginx 当前状态的当前状态，如当前的活动连接，接受和处理当前读/写/等待连接的总数

2）Sub_filter 指令：它用于搜索和替换响应中的内容，并快速修复陈旧的数据

16、解释 Nginx 是否支持将请求压缩到上游？

答：您可以使用 Nginx 模块 gunzip 将请求压缩到上游。gunzip 模块是一个过滤器，它可以对不支持“gzip”编码方法的客户机或服务器使用“内容编码:gzip”来解压缩响应。

17、解释如何在 Nginx 中获得当前的时间？

答：要获得 Nginx 的当前时间，必须使用 SSI 模块、$date_gmt 和 $date_local 的变量。Proxy_set_header THE-TIME $date_gmt;

18、用 Nginx 服务器解释-s 的目的是什么？

答：用于运行 Nginx -s 参数的可执行文件。

19、解释如何在 Nginx 服务器上添加模块？

答：在编译过程中，必须选择 Nginx 模块，因为 Nginx 不支持模块的运行时间选择。

20、负载均衡策略

负载均衡的策略可以大致分为两大类：内置策略 和扩展策略内置策略：一般会直接编译进 Nginx 内核，常用的有、轮询、ip hash、最少连接扩展策略：fair、url hash 等

（四）Lucene 和 Solr 和 Elasticsearch 部分

Lucene 和 Solr 和 Elasticsearch 的定义

Lucene

Lucene 是 apache 下的一个子项目，是一个开放源代码的全文检索引擎工具包，但它不是一个完整的全文检索引擎，而是一个全文检索引擎的架构，提供了完整的查询引擎和索引引擎，部分文本分析引擎。官网地址：https://lucene.apache.org/

Solr

Solr 是一个高性能，采用 Java5 开发，基于 Lucene 的全文搜索服务器。同时对其进行了扩展，提供了比 Lucene 更为丰富的查询语言，同时实现了可配置、可扩展并对查询性能进行了优化，并且提供了一个完善的功能管理界面，是一款非常优秀的全文搜索引擎。官网地址：http://lucene.apache.org/Solr/

Elasticsearch

Elasticsearch 跟 Solr 一样，也是一个基于 Lucene 的搜索服务器，它提供了一个分布式多用户能力的全文搜索引擎，基于 RESTful web 接口。官网地址：https://www.elastic.co/products/elasticsearch

1、Elasticsearch 的优缺点：

优点：

1.Elasticsearch 是分布式的。不需要其他组件，分发是实时的，被叫做”Push replication”。

2.Elasticsearch 完全支持 Apache Lucene 的接近实时的搜索。

3.处理多租户（multitenancy）不需要特殊配置，而 Solr 则需要更多的高级设置。

4.Elasticsearch 采用 Gateway 的概念，使得完备份更加简单。

5.各节点组成对等的网络结构，某些节点出现故障时会自动分配其他节点代替其进行工作。

缺点：

1.只有一名开发者（当前 Elasticsearch GitHub 组织已经不只如此，已经有了相当活跃的维护者）

2.还不够自动（不适合当前新的 Index Warmup API）

2、Solr 的优缺点：

优点

1.Solr 有一个更大、更成熟的用户、开发和贡献者社区。

2.支持添加多种格式的索引，如：HTML、PDF、微软 Office 系列软件格式以及 JSON、XML、CSV 等纯文本格式。

3.Solr 比较成熟、稳定。

4.不考虑建索引的同时进行搜索，速度更快。

缺点

1.建立索引时，搜索效率下降，实时索引搜索效率不高。

3、Elasticsearch 与 Solr 的比较：

1.二者安装都很简单；

2.Solr 利用 Zookeeper 进行分布式管理，而 Elasticsearch 自身带有分布式协调管理功能;

3.Solr 支持更多格式的数据，而 Elasticsearch 仅支持 json 文件格式；

4.Solr 官方提供的功能更多，而 Elasticsearch 本身更注重于核心功能，高级功能多有第三方插件提供；

5.Solr 在传统的搜索应用中表现好于 Elasticsearch，但在处理实时搜索应用时效率明显低于 Elasticsearch。

6.Solr 是传统搜索应用的有力解决方案，但 Elasticsearch 更适用于新兴的实时搜索应用。

使用案例：

1.维基百科使用 Elasticsearch 来进行全文搜做并高亮显示关键词，以及提供 search-as-you-type、did-you-mean 等搜索建议功能。

2.英国卫报使用 Elasticsearch 来处理访客日志，以便能将公众对不同文章的反应实时地反馈给各位编辑。

3.StackOverflow 将全文搜索与地理位置和相关信息进行结合，以提供 more-like-this 相关问题的展现。

4.GitHub 使用 Elasticsearch 来检索超过 1300 亿行代码。

5.每天，Goldman Sachs 使用它来处理 5TB 数据的索引，还有很多投行使用它来分析股票市场的变动。

4、Solr 如何实现搜索的？

倒排索引，先抽取文档中词，并建立词与文档 id 的映射关系，然后查询的时候会根据词去查询文档 id，并查询出文档

5、Solr 过滤器

Solr 的过滤器对接收到的标记流（TokenStream ）做额外的处理。过滤查询，在查询时设置

6、Solr 原理

Solr 是基于 Lucene 开发的全文检索服务器，而 Lucene 就是一套实现了全文检索的 API，其本质就是一个全文检索的过程。全文检索就是把原始文档根据一定的规则拆分成若干个关键词，然后根据关键词创建索引，当查询时先查询索引找到对应的关键词，并根据关键词找到对应的文档，也就是查询结果，最终把查询结果展示给用户的过程

7、Solr 怎么设置搜索结果排名靠前

设置文档中域的 boost 值，值越高相关性越高，排名就靠前。

8、IK 分词器原理

本质上是词典分词，在内存中初始化一个词典，然后在分词过程中逐个读取字符，和字典中的字符相匹配，把文档中的所有词语拆分出来的过程

9、Solr 的索引查询为什么比数据库要快

Solr 使用的是 Lucene API 实现的全文检索。全文检索本质上是查询的索引。而数据库中并不是所有的字段都建立的索引，更何况如果使用 like 查询时很大的可能是不使用索引，所以使用 Solr 查询时要比查数据库快

10、Solr 索引库个别数据索引丢失怎么办

首先 Solr 是不会丢失个别数据的。如果索引库中缺少数据，那就向索引库中添加

11、Lucene 索引优化

直接使用 Lucene 实现全文检索已经是过时的方案，推荐使用 Solr。Solr 已经提供了完整的全文检索解决方案

12、多张表的数据导入 Solr(解决 id 冲突)

在 schema.xml 中添加 uuid，然后 Solrconfig 那边修改 update 的部分，改为使用 uuid 生成

13、Solr 如何分词，新增词和禁用词如何解决

schema.xml 文件中配置一个 IK 分词器，然后域指定分词器为 IK

新增词添加到词典配置文件中 ext.dic，禁用词添加到禁用词典配置文件中 stopword.dic，然后在 schema.xml 文件中配置禁用词典：<filter class="Solr.StopFilterFactory" ignore="true" words="/禁止词文件目录"/>

14、Solr 多条件组合查询

创建多个查询对象，指定他们的组合关系，Occur.MUST（必须满足 and），Occur.SHOULD（应该满足 or），Occur.MUST_NOT（必须不满足 not）

15、Elasticsearch 了解多少，说说你们公司 es 的集群架构，索引数据大小，分片有多少，以及一些调优手段。Elasticsearch 的倒排索引是什么。

ElasticSearch（简称 ES）是一个分布式、Restful 的搜索及分析服务器，设计用于分布式计算；能够达到实时搜索，稳定，可靠，快速。和 Apache Solr 一样，它也是基于 Lucence 的索引服务器，而 ElasticSearch 对比 Solr 的优点在于：

轻量级：安装启动方便，下载文件之后一条命令就可以启动。

Schema free：可以向服务器提交任意结构的 JSON 对象，Solr 中使用 schema.xml 指定了索引结构。

多索引文件支持：使用不同的 index 参数就能创建另一个索引文件，Solr 中需要另行配置。

分布式：Solr Cloud 的配置比较复杂。

倒排索引是实现“单词-文档矩阵”的一种具体存储形式，通过倒排索引，可以根据单词快速获取包含这个单词的文档列表。倒排索引主要由两个部分组成：“单词词典”和“倒排文件”。

16、Elasticsearch 索引数据多了怎么办，如何调优，部署。

使用 bulk API
初次索引的时候，把 replica 设置为 0
增大 threadpool.index.queue_size
增大 indices.memory.index_buffer_size
增大 index.translog.flush_threshold_ops
增大 index.translog.sync_interval
增大 index.engine.robin.refresh_interval

17、Lucence 内部结构是什么

索引(Index)： 在 Lucene 中一个索引是放在一个文件夹中的。 如上图，同一文件夹中的所有的文件构成一个 Lucene 索引。

段(Segment)： 一个索引可以包含多个段，段与段之间是独立的，添加新文档可以生成新的段，不同的段可以合并。

segments.gen 和 segments_X 是段的元数据文件，也即它们保存了段的属性信息。

文档(Document)： 文档是我们建索引的基本单位，不同的文档是保存在不同的段中的，一个段可以包含多篇文档。

新添加的文档是单独保存在一个新生成的段中，随着段的合并，不同的文档合并到同一个段中。

域(Field)：

一篇文档包含不同类型的信息，可以分开索引，比如标题，时间，正文，作者等，都可以保存在不同的域里。 不同域的索引方式可以不同，在真正解析域的存储的时候，我们会详细解读。

词(Term)：

词是索引的最小单位，是经过词法分析和语言处理后的字符串。

18、Solr 和 Lucene 的区别

Solr 和 Lucene 的本质区别有以下三点：搜索服务器，企业级和管理。Lucene 本质上是搜索库，不是独立的应用程序，而 Solr 是。Lucene 专注于搜索底层的建设，而 Solr 专注于企业应用。Lucene 不负责支撑搜索服务所必须的管理，而 Solr 负责。所以说，一句话概括 Solr: Solr 是 Lucene 面向企业搜索应用的扩展

Lucene: 是一个索引与搜索类库，而不是完整的程序。

Solr：是一个高性能，采用 Java5 开发，基于 Lucene 的一个独立的企业级搜索应用服务器，它对外提供类似于 Web-service 的 API 接口。

19、Solr 实现全文检索

索引流程：客户端---》Solr 服务器(发送 post 请求,xml 文档包含 filed，Solr 实现对索引的维护)

搜索流程：客户端---》Solr 服务器(发送 get 请求，服务器返回一个 xml 文档)

20、SolrCore 如何安装

Solrhome 是 Solr 服务运行的主目录，Solrhome 包含多个 SolrCore,一个 SolrCore 目录里面 Solr 实例运行时的配置文件和数据文件

安装很简单就是将 example 下面的 Solr 目录，拷贝到自己创建的 Solrhome 文件夹。

21、SolrCore 配置

Solrhome--->collection1--->conf 当中 Solrconfig.xml 文件当中配置 lib 标签，datadir 标签

requestHandler 标签：Solr.install.dir : Solr 的安装目录，将文件 contrib,dist 进行 copy 到相应的目录下，同时 Solrcore 的安装目录发生所以对应修改相应的配置。

datadir 标签:索引文件的目录

equestHandler 标签：请求处理器。/update 添加，修改，删除 ; /select 搜索

（五）WebService 部分

1、WebService 名词解释。JSWDL 开发包的介绍。JAXP、JAXM 的解释。SOAP、UDDI,WSDL 解释。

Web ServiceWeb Service 是基于网络的、分布式的模块化组件，它执行特定的任务，遵守具体的技术规范，这些规范使得 Web Service 能与其他兼容的组件进行互操作。

JAXP(Java API for XML Parsing) 定义了在 Java 中使用 DOM, SAX, XSLT 的通用的接口。这样在你的程序中你只要使用这些通用的接口，当你需要改变具体的实现时候也不需要修改代码。

JAXM(Java API for XML Messaging) 是为 SOAP 通信提供访问方法和传输机制的 API。

WSDL 是一种 XML 格式，用于将网络服务描述为一组端点，这些端点对包含面向文档信息或面向过程信息的消息进行操作。这种格式首先对操作和消息进行抽象描述，然后将其绑定到具体的网络协议和消息格式上以定义端点。相关的具体端点即组合成为抽象端点（服务）。

SOAP 即简单对象访问协议(Simple Object Access Protocol)，它是用于交换 XML 编码信息的轻量级协议。

UDDI 的目的是为电子商务建立标准；UDDI 是一套基于 Web 的、分布式的、为 Web Service 提供的、信息注册中心的实现标准规范，同时也包含一组使企业能将自身提供的 Web Service 注册，以使别的企业能够发现的访问协议的实现标准。

2、CORBA 是什么？用途是什么？

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

3、什么是 WebService（用你的话描述 WebService）？在什么时候用 WebService（WebService 能给我们解决什么样的问题）？

一句话概括：WebService 是一种跨编程语言和跨操作系统平台的远程调用技术。

所谓跨编程语言和跨操作平台，就是说服务端程序采用 Java 编写，客户端程序则可以采用其他编程语言编写，反之亦然！跨操作系统平台则是指服务端程序和客户端程序可以在不同的操作系统上运行。

所谓远程调用，就是一台计算机 a 上的一个程序可以调用到另外一台计算机 b 上的一个对象的方法。譬如从天气预报系统中获取某个城市的天气数据在自己系统中进行展示；从证券交易系统中获取某只股票的交易信息在自己的系统中进行展示；又譬如一个商城系统中能够展示快递的跟踪信息，而这些信息就是通过 WebService 从具体的快递公司的系统中获取的数据。

其实可以从多个角度来理解 WebService，从表面上看，WebService 就是一个应用程序向外界暴露出一个能通过 Web 进行调用的 API，也就是说能用编程的方法通过 Web 来调用这个应用程序。我们把调用这个 WebService 的应用程序叫做客户端，而把提供这个 WebService 的应用程序叫做服务端。从深层次看，WebService 是建立可互操作的分布式应用程序的新平台，是一个平台，是一套标准。它定义了应用程序如何在 Web 上实现互操作性，你可以用任何你喜欢的语言，在任何你喜欢的平台上写 Web service ，只要我们可以通过 Web service 标准对这些服务进行查询和访问。

4、WSDL 是什么，有什么作用？

WSDL 是 web service definition language 的缩写，即 web service 的定义（描述）语言。

怎样向别人介绍你的 web service 有什么功能，以及每个函数调用时的参数呢？你可能会自己写一套文档，你甚至可能会口头上告诉需要使用你的 web service 的人。这些非正式的方法至少都有一个严重的问题：当程序员坐到电脑前，想要使用你的 web service 的时候，他们的工具（如 Visual Studio）无法给他们提供任何帮助，因为这些工具根本就不了解你的 web service。解决方法是：用机器能阅读的方式提供一个正式的描述文档。web service 描述语言

(WSDL)就是这样一个基于 XML 的语言，用于描述 web service 及其函数、参数和返回值。因为是基于 XML 的，所以 WSDL 既是机器可阅读的，又是人可阅读的，这将是一个很大的好处。一些最新的开发工具既能根据你的 web service 生成 WSDL 文档，又能导入 WSDL 文档，生成调用相应 web service 的代码。

WebService 服务发布之后，通过浏览器访问发布的+？wsdl 即可获得 wsdl 文档。

5、WSDL 文档主要有那几部分组成，分别有什么作用？

一个 WSDL 文档的根元素是 definitions 元素，WSDL 文档包含 7 个重要的元素：types, import, message, portType, operations, binding 和 service 元素。

1、 definitions 元素中一般包括若干个 XML 命名空间；

2、 Types 元素用作一个容器，定义了自定义的特殊数据类型，在声明消息部分（有效负载）的时候，messages 定义使用了 types 元素中定义的数据类型与元素；

3、 Import 元素可以让当前的文档使用其他 WSDL 文档中指定命名空间中的定义；

4、 Message 元素描述了 Web 服务的有效负载。相当于函数调用中的参数和返回值；

5、 PortType 元素定义了 Web 服务的抽象接口，它可以由一个或者多个 operation 元素，每个 operation 元素定义了一个 RPC 样式或者文档样式的 Web 服务方法；

6、 Operation 元素要用一个或者多个 messages 消息来定义它的输入、输出以及错误；

7、 Binding 元素将一个抽象的 portType 映射到一组具体的协议（SOAP 或者 HTTP）、消息传递样式（RPC 或者 document）以及编码样式（literal 或者 SOAP encoding）；

8、 Service 元素包含一个或者多个 Port 元素每一个 Port 元素对应一个不同的 Web 服务，port 将一个 URL 赋予一个特定的 binding，通过 location 实现。可以使两个或者多个 port 元素将不同的 URL 赋给相同的 binding。

6、SOAP 是什么？

SOAP 是 simple object access protocal 的缩写，即简单对象访问协议。 是基于 XML 和 HTTP 的一种通信协议。是 WebService 所使用的一种传输协议，WebService 之所以能够做到跨语言和跨平台，主要是因为 XML 和 HTTP 都是独立于语言和平台的。Soap 的消息分为请求消息和响应消息，一条 SOAP 消息就是一个普通的 XML 文档，包含下列元素：

1、 必需的 Envelope 元素，可把此 XML 文档标识为一条 SOAP 消息

2、 可选的 Header 元素，包含头部信息

3、 必需的 Body 元素，包含所有的调用和响应信息

4、 可选的 Fault 元素，提供有关在处理此消息所发生错误的信息

7、怎么理解 UDDI？

UDDI 是 Universal Description Discovery and Integration 的缩写，即统一描述、发现和整合规范。用来注册和查找服务，把 web services 收集和存储起来，这样当别人访问这些信息的时候就从 UDDI 中查找，看有没有这个信息存在。

8、WebService 的 SEI 指什么？

WebService EndPoint Interface（WebService 终端[Server 端]接口），就是 WebService 服务器端用来处理请求的接口

9、说说你知道的 WebService 框架，他们都有什么特点？

WebService 常用框架有 JWS、Axis2、XFire 以及 CXF。下面分别介绍一个这几种 Web Service 框架的基本概念

1、JWS 是 Java 语言对 WebService 服务的一种实现，用来开发和发布服务。而从服务本身的角度来看 JWS 服务是没有语言界限的。但是 Java 语言为 Java 开发者提供便捷发布和调用 WebService 服务的一种途径。

2、Axis2 是 Apache 下的一个重量级 WebService 框架，准确说它是一个 Web Services / SOAP / WSDL 的引擎，是 WebService 框架的集大成者，它能不但能制作和发布 WebService，而且可以生成 Java 和其他语言版 WebService 客户端和服务端代码。这是它的优势所在。但是，这也不可避免的导致了 Axis2 的复杂性，使用过的开发者都知道，它所依赖的包数量和大小都是很惊人的，打包部署发布都比较麻烦，不能很好的与现有应用整合为一体。但是如果你要开发 Java 之外别的语言客户端，Axis2 提供的丰富工具将是你不二的选择。

3、XFire 是一个高性能的 WebService 框架，在 Java6 之前，它的知名度甚至超过了 Apache 的 Axis2，XFire 的优点是开发方便，与现有的 Web 整合很好，可以融为一体，并且开发也很方便。但是对 Java 之外的语言，没有提供相关的代码工具。XFire 后来被 Apache 收购了，原因是它太优秀了，收购后，随着 Java6 JWS 的兴起，开源的 WebService 引擎已经不再被看好，渐渐的都败落了。

4、CXF 是 Apache 旗下一个重磅的 SOA 简易框架，它实现了 ESB（企业服务总线）。CXF 来自于 XFire 项目，经过改造后形成的，就像目前的 Struts2 来自 WebWork 一样。可以看出 XFire 的命运会和 WebWork 的命运一样，最终会淡出人们的视线。CXF 不但是一个优秀的 Web Services / SOAP / WSDL 引擎，也是一个不错的 ESB 总线，为 SOA 的实施提供了一种选择方案，当然他不是最好的，它仅仅实现了 SOA 架构的一部分。

注：对于 Axis2 与 CXF 之间的关系，一个是 Axis2 出现的时间较早，而 CXF 的追赶速度快。

如何抉择：

1、如果应用程序需要多语言的支持，Axis2 应当是首选了；

2、如果应用程序是遵循 Spring 哲学路线的话，Apache CXF 是一种更好的选择，特别对嵌入式的 Web Services 来说；

3、如果应用程序没有新的特性需要的话，就仍是用原来项目所用的框架，比如 Axis1，XFire，Celtrix 或 BEA 等等厂家自己的 Web Services 实现，就别劳民伤财了。

10、你的系统中是否有使用到 WebService 开发，具体是怎么实现的？

如果你觉得自己掌握的不够好，对自己不够自信的可以回答为“我的系统中没有使用到 WebService 的开发，但是我掌握 WebService 开发的概念和流程”，然后可以给他讲讲相关的概念，也就是上面的这些问题的回答，这样可以绕过这个问题，因为并不是所有的系统都会涉及到 WebService 的开发。

另一种回答即是先给他介绍一种 WebService 开发框架，比如 CXF，然后告诉他你做的是服务端开发还是客户端开发，如果你说你做的事服务端开发，那么你就告诉他怎么定义的 WebService，使用了哪些注解，怎么跟 Spring 进行的整合，怎么发布的服务等等；如果你告诉他你做的事客户端的开发，那么你可以告诉他你怎么生成的本地代码，然后又怎么通过本地代码去调用 WebService 服务。

11、如何选择采用哪种 WebService？SOAP WS 还是 REST？

一般而言，基于 REST 的 Web service 的优势在于其简单、性能不错、可扩展性好，并且也支持多种数据格式。而 SOAP 则适用于安全性和事务处理可靠性方面要求比较高的服务中。

对于这个问题的答案，更多的考虑依据是设计者对功能性和非功能性需求的要求。通过回答下列问题可以帮助你做出选择：

所提供的服务会暴露数据或者业务逻辑吗？（如果会暴露数据的话可以选择 REST 方式，如果会暴露业务逻辑的话可以选择 SOAP WS）。客户或者服务提供商需要一个正式的契约（contract）吗？（SOAP 可以通过 WSDL（Web Service Description Language）提供一个正式契约）

1. 需要支持多种数据格式吗？
2. 需要进行 AJAX 调用吗？（REST 可以采用 XMLHttpRequest 来发送 AJAX 调用）
3. 同步调用还是异步调用？
4. 有状态调用还是无状态调用？（REST 适合无状态 CRUD 操作）
5. 对于安全性的要求？（SOAP WS 对于安全性的支持更好些）
6. 对于事务处理的要求？（SOAP WS 这方面更有优势）
7. 有带宽限制吗？（SOAP 消息比较冗长）
8. 哪种方式更适合开发者呢呢？（REST 更好实现，也更好测试和维护）

12、有什么可以用来测试 WebService 的工具吗？

测试 SOAP WS 可以使用 SoapUI。

测试 RESTFul service 可以采用 Firefox 的“poster”插件。

七. 算法与编程

1、编写一个程序，将多个文件按要求合并成一个。

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.lanqiao;
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，实现文件拷贝和重命名。

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。

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，实现字符串截取。

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 我爱传智 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、编写一个程序，统计字符串中字符个数。

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。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 = “中国 aadf 的 111 萨 bbb 菲的 zz 萨菲”;
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 中国 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。

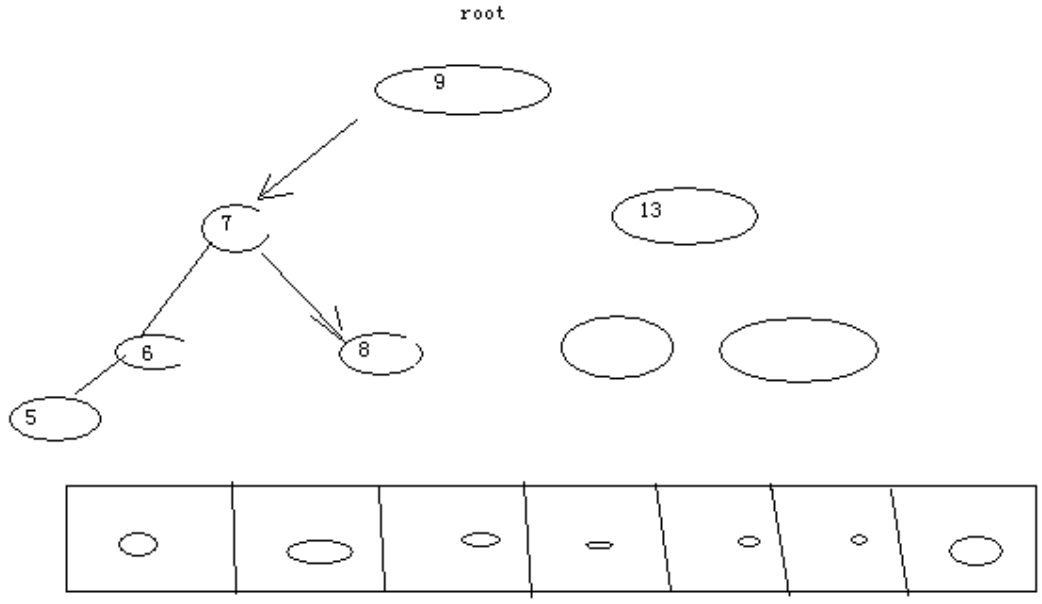
```
int engishCount;
int chineseCount;
int digitCount;
for(int i=0;i<str.length;i++)
```

```
{
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、编写一个程序，实现文本操作和数据统计。

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,张三,28
2,李四,35
3,张三,28
4,王五,35
5,张三,28
6,李四,35
7,赵六,28
8,田七,35

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;
}
}
```

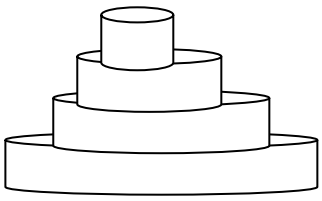
其他形式:
定义一个类，它的构造函数为 private 的，所有方法为 static 的。
一般认为第一种形式要更加安全些

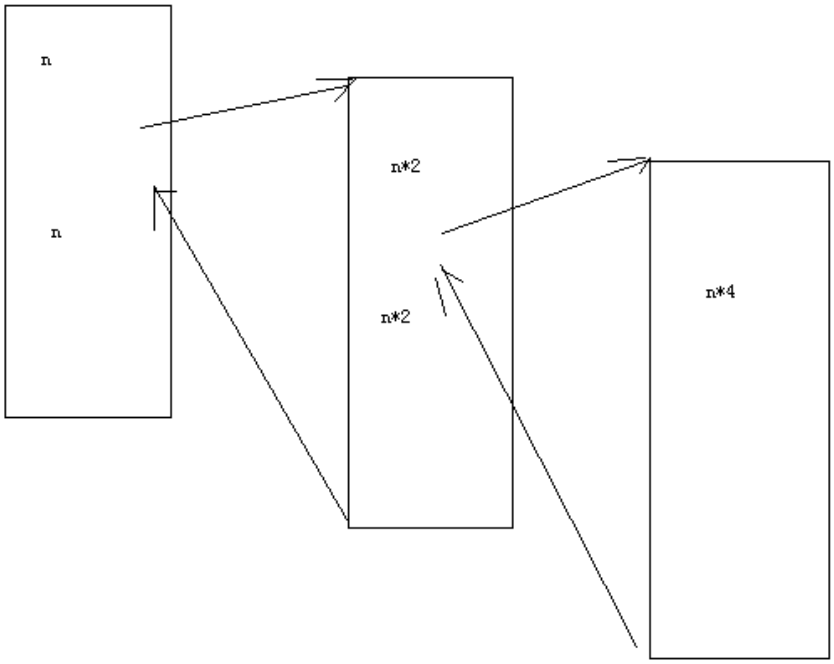
8、递归算法题 1

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.itcast;
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);
}
}
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 {
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) {
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();
}
}
```

八. 大数据

（一）Linux

1、在 Linux 系统中以什么方式访问硬件设备的？

以文件的方式访问设备　比如访问光驱对应的文件是 /dev/cdrom

2、文件属性的解释，说出 ls -l 命令显示的文件每个属性的含义？

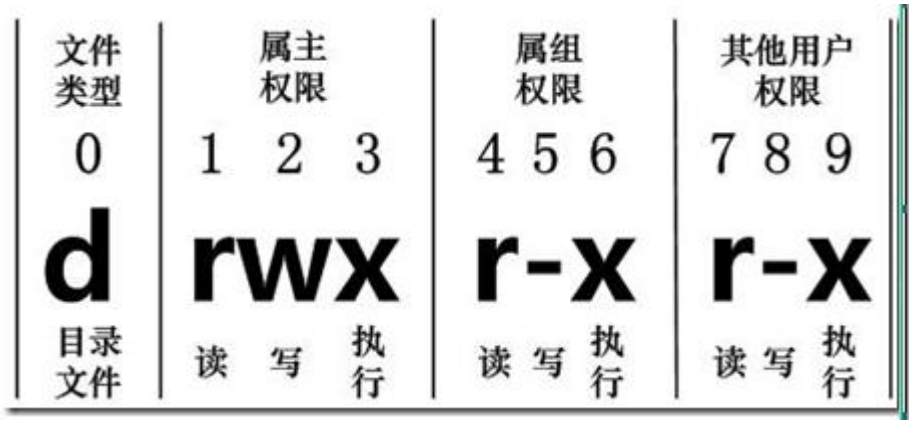
```
[root@www /]# ls -l
total 64
dr-xr-xr-x     2 root root 4096 Dec 14    2012 bin
dr-xr-xr-x     4 root root 4096 Apr 19    2012 boot
```

1) 实例中，bin 文件的第一个属性用"d"表示。"d"在 Linux 中代表该文件是一个目录文件。

2) 在 Linux 中第一个字符代表这个文件是目录、文件或链接文件等等。
- 当为[d]则是目录
- 当为[-]则是文件；
- 若是[l]则表示为链接文档(link file)；
- 若是[b]则表示为装置文件里面的可供储存的接口设备(可随机存取装置)；
- 若是[c]则表示为装置文件里面的串行端口设备，例如键盘、鼠标(一次性读取装置)。

3) 接下来的字符中，以三个为一组，且均为『rwx』 的三个参数的组合。其中，[r]代表可读(read)、[w]代表可写(write)、[x]代表可执行(execute)。要注意的是，这三个权限的位置不会改变，如果没有权限，就会出现减号[-]而已。

每个文件的属性由左边第一部分的 10 个字符来确定（如下图）。



从左至右用 0-9 这些数字来表示。第 0 位确定文件类型，第 1-3 位确定属主（该文件的所有者）拥有该文件的权限。

第 4-6 位确定属组（所有者的同组用户）拥有该文件的权限，第 7-9 位确定其他用户拥有该文件的权限。

其中，第 1、4、7 位表示读权限，如果用"r"字符表示，则有读权限，如果用"-"字符表示，则没有读权限；

第 2、5、8 位表示写权限，如果用"w"字符表示，则有写权限，如果用"-"字符表示没有写权限；第 3、6、9 位表示可执行权限，如果用"x"字符表示，则有执行权限，如果用"-"字符表示，则没有执行权限。

3、如何创建一个用户组？

案例 1：groupadd –g 101 groupname1　　##101 是组的 id
案例 2：groupadd　groupname2　　##直接创建用户组

4、如何删除一个用户组？

groupdel 用户组名

5、如何查看所有用户组的信息？

cat /etc/group

6、如何创建用户？

第一种方式：useradd　username
第二种方式: useradd -s /bin/sh -g group1 -G group2,group3 group1_user1
此命令新建了一个用户 gem，该用户的登录 Shell 是/bin/sh，他属于 group 用户组，同时又属于 adm 和 root 用户组，其中 group 用户组是其主组。
这里可能新建组：groupadd group　及　groupadd adm

7、如何删除用户？

userdel　username　（不会删除用户主目录）
userdel –r username　（会删除用户主目录）

8、如何修改账号所属组？

usermod　-g　ggroup2　g1_user1
把用户 g1_user1 改到 group2　组

9、如何查看所有用户信息？

cat　/etc/passwd

10、如何进行口令管理？

普通用户：

```
$ passwd
Old password:******
New password:*******
Re-enter new password:*******
```

超级用户：

```
passwd sam
New password:*******
Re-enter new password:*******
```

11、如何给普通用户 sudo 权限？

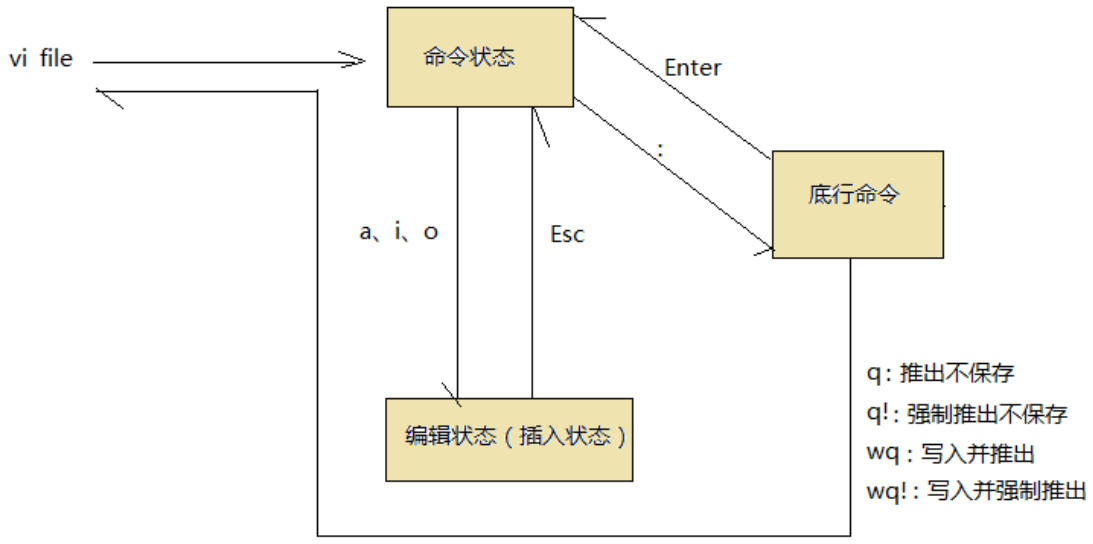
第一步：vi /etc/sudoers
第二步：如下图红线部分给 hadoop 这个用户赋 sudo 权限

```
## Syntax:
##
##      user    MACHINE=COMMANDS
##
## The COMMANDS section may have other options ac
##
## Allow root to run any commands anywhere
root    ALL=(ALL)       ALL
hadoop  ALL=(ALL)       ALL

## Allows members of the 'sys' group to run netwo
## service management apps and more.
# %sys ALL = NETWORKING, SOFTWARE, SERVICES, STOR
ATE, DRIVERS

## Allows people in group wheel to run all commar
# %wheel        ALL=(ALL)       ALL

## Same thing without a password
# %wheel        ALL=(ALL)       NOPASSWD: ALL
```

12、如何修改对这个文件的访问权限？

chmod u+x,g+w f01　　//为文件 f01 设置自己可以执行，组员可以写入的权限
chmod u=rwx,g=rw,o=r f01
chmod 764 f01

13、请用画图的方式描述 vi 编辑器的各种状态转换方式



14、vi 一般模式下的常用命令有哪些？

【h(或向左方向键)】 光标左移一个字符
【j(或向下方向键)】 光标下移一个字符
【k(或向上方向键)】 光标上移一个字符
【l(或向右方向键)】 光标右移一个字符
【[Ctrl] + f】 屏幕向下移动一页（相当于 Page Down 键）
【[Ctrl] + b】 屏幕向上移动一页（相当于 Page Up 键）
【[0]或[Home]】 光标移动到当前行的最前面
【[$]或[End]】 光标移动到当前行的末尾
【G】 光标移动到文件的最后一行（第一个字符处）
【nG】 n 为数字（下同），移动到当前文件中第 n 行
【gg】 移动到文件的第一行，相当于"1G"
【n[Enter]】 光标向下移动 n 行
【/word】 在文件中查找内容为 word 的字符串（向下查找）
【？word】 在文件中查找内容为 word 的字符串（向上查找）
【[n]】 表示重复查找动作，即查找下一个
【[N]】 反向查找下一个
【:n1,n2s/word1/word2/g】 n1、n2 为数字，在第 n1 行到第 n2 行之间查找 word1 字符串，并将其替换成 word2
【:1,s/word1/word2/g】从第一行（第 n 行同理）到最后一行查找 word1 注册，并将其替换成 word2【:1,s/word1/word2/g】从第一行（第 n 行同理）到最后一行查找 word1 注册，并将其替换成 word2【:1,s/word1/word2/gc】 功能同上，只不过每次替换时都会让用户确认
【x,X】 x 为向后删除一个字符，相当于[Delete]，X 为向前删除一个字符，相当于[Backspace]
【dd】 删除光标所在的一整行
【ndd】 删除光标所在的向下 n 行
【yy】 复制光标所在的那一行
【nyy】 复制光标所在的向下 n 行
【p,P】 p 为将已经复制的数据在光标下一行粘贴；P 为将已经复制的数据在光标上一行粘贴
【u】 撤消上一个操作
【[Ctrl] + r】 多次撤消
【.】 这是小数点键，重复上一个操作

15、请列出 vi 编辑器 6 种进入插入模式的命令？

【i】从目前光标所在处插入
【I】 从目前光标
【a】 从当前光标所在的下一个字符处开始插入
【A】 从光标所在行的最后一个字符处开始插入
【o】 英文小写字母 o，在目前光标所在行的下一行处插入新的一行并开始插入

【O】 英文大写字母 O，在目前光标所在行的上一行处插入新的一行并开始插入

16、什么是 Linux Shell？Linux Shell 种类有哪些？

Shell 脚本（shell script），是一种为 shell 编写的脚本程序。

业界所说的 shell 通常都是指 shell 脚本，但读者朋友要知道，shell 和 shell script 是两个不同的概念。由于习惯的原因，简洁起见，本文出现的 "shell 编程" 都是指 shell 脚本编程，不是指开发 shell 自身。

Linux 的 Shell 种类众多，常见的有：

Bourne Shell（/usr/bin/sh 或/bin/sh）

Bourne Again Shell（/bin/bash）

C Shell（/usr/bin/csh）

K Shell（/usr/bin/ksh）

Shell for Root（/sbin/sh）

17、运行 Shell 脚本有几种方法？具体怎么运行？

1、作为可执行程序

将上面的代码保存为 test.sh，并 cd 到相应目录：

```
chmod +x ./test.sh  #使脚本具有执行权限
./test.sh  #执行脚本
```

注意，一定要写成 ./test.sh，而不是 test.sh，运行其它二进制的程序也一样，直接写 test.sh，linux 系统会去 PATH 里寻找有没有叫 test.sh 的，而只有 /bin, /sbin, /usr/bin，/usr/sbin 等在 PATH 里，你的当前目录通常不在 PATH 里，所以写成 test.sh 是会找不到命令的，要用 ./test.sh 告诉系统说，就在当前目录找。

2、作为解释器参数

这种运行方式是，直接运行解释器，其参数就是 shell 脚本的文件名，如：

```
/bin/sh test.sh
/bin/php test.php
```

这种方式运行的脚本，不需要在第一行指定解释器信息，写了也没用。

18、如何定义 Shell 变量和只读变量？

定义变量 ： name="qinjx"

定义只读变量：

使用 readonly 命令可以将变量定义为只读变量，只读变量的值不能被改变。

下面的例子尝试更改只读变量，结果报错：

```
#!/bin/bash
myUrl="http://www.w3cschool.cc"
readonly myUrl
myUrl="http://www.runoob.com"
```

19、Linux Shell 如何定义数组？

在 Shell 中，用括号来表示数组，数组元素用"空格"符号分割开。定义数组的一般形式为：

数组名=(值 1 值 2 ... 值 n)

例如：

```
array_name=(value0 value1 value2 value3)
```

或者

```
array_name=(
value0
value1
value2
value3
)
```

还可以单独定义数组的各个分量：

```
array_name[0]=value0
array_name[1]=value1
array_name[n]=valuen
```

20、用 Linux expect 语言编写批量登录并创建文件夹？

```
#!/bin/bash
passwd='123456'
int=1
while(( $int<=5 ))
do
/usr/local/bin/expect <<-EOF
set time 10
spawn ssh mini02
expect {
"*yes/no" { send "yes\r"; exp_continue }
"*password:" { send "$passwd\r" }
}
expect "*#"
send "mkdir /root/test${int}\r"
expect "*#"
send "exit\r"
interact
expect eof
EOF
        echo $int
        let "int++"
done
```

21、如何挂载 iso 文件？

mkdir /mnt/cdrom
mount /dev/cdrom /mnt/cdrom

22、如何配置本地 yum 源？

vi /etc/yum.repos.d/CentOS-Media.repo

```
[c6-media]
name=CentOS-$releasever - Media
baseurl=file:///mnt/cdrom
gpgcheck=0
enabled=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-6
```

23、如何在 Linux 系统安装 JDK 说出安装过程？

1）下载 jdk-1.8.tar.gz
2）上传 jdk-1.8.tar.gz 到某一个节点
3）创建一个安装目录 mkdir /root/apps
4）把 jdk 解压到此目录 tar –zxvf jdk-1.8.tar.gz –C /root/apps
5）打开 profile 文件 vi /etc/profile
6）配置 JAVA HOME 和 PATH 指定 jdk 对应目录
7）Source /etc/profile 使之生效
8）Java –version 查看安装 jdk 版本 找不到命令则安装失败

（二）Hadoop

1、什么是 Hadoop？

Hadoop 是一个开源软件框架，用于存储大量数据，并发处理/查询在具有多个商用硬件（即低成本硬件）节点的集群上的那些数据。总之，Hadoop 包括以下内容：

HDFS（Hadoop Distributed File System，Hadoop 分布式文件系统）：HDFS 允许你以一种分布式和冗余的方式存储大量数据。例如，1 GB（即 1024 MB）文本文件可以拆分为 16 * 128MB 文件，并存储在 Hadoop 集群中的 8 个不同节点上。每个分裂可以复制 3 次，以实现容错，以便如果 1 个节点故障的话，也有备份。HDFS 适用于顺序的“一次写入、多次读取”的类型访问。

MapReduce：一个计算框架。它以分布式和并行的方式处理大量的数据。当你对所有年龄> 18 的用户在上述 1 GB 文件上执行查询时，将会有“8 个映射”函数并行运行，以在其 128 MB 拆分文件中提取年龄> 18 的用户，然后“reduce”函数将运行以将所有单独的输出组合成单个最终结果。

YARN（Yet Another Resource Nagotiator，又一资源定位器）：用于作业调度和集群资源管理的框架。

Hadoop 生态系统，拥有 15 多种框架和工具，如 Sqoop，Flume，Kafka，Pig，Hive，Spark，Impala 等，以便将数据摄入 HDFS，在 HDFS 中转移数据（即变换，丰富，聚合等），并查询来自 HDFS 的数据用于商业智能和分析。某些工具（如 Pig 和 Hive）是 MapReduce 上的抽象层，而 Spark 和 Impala 等其他工具则是来自 MapReduce 的改进架构/设计，用于显著提高的延迟以支持近实时（即 NRT）和实时处理。

2、为什么组织从传统的数据仓库工具转移到基于 Hadoop 生态系统的智能数据中心？

Hadoop 组织正在从以下几个方面提高自己的能力：

1. 现有数据基础设施：
   - 主要使用存储在高端和昂贵硬件中的“structured data，结构化数据”
   - 主要处理为 ETL 批处理作业，用于将数据提取到 RDBMS 和数据仓库系统中进行数据挖掘，分析和报告，以进行关键业务决策。
   - 主要处理以千兆字节到兆字节为单位的数据量
2. 基于 Hadoop 的更智能的数据基础设施，其中
   - 结构化（例如 RDBMS），非结构化（例如 images，PDF，docs ）和半结构化（例如 logs，XMLs）的数据可以以可扩展和容错的方式存储在较便宜的商品机器中。
   - 可以通过批处理作业和近实时（即，NRT，200 毫秒至 2 秒）流（例如 Flume 和 Kafka）来摄取数据。
   - 数据可以使用诸如 Spark 和 Impala 之类的工具以低延迟（即低于 100 毫秒）的能力查询。
   - 可以存储以兆兆字节到千兆字节为单位的较大数据量。

这使得组织能够使用更强大的工具来做出更好的业务决策，这些更强大的工具用于获取数据，转移存储的数据（例如聚合，丰富，变换等），以及使用低延迟的报告功能和商业智能。

3、基于 Hadoop 的数据中心的好处是什么？

随着数据量和复杂性的增加，提高了整体 SLA（即服务水平协议）。例如，“Shared Nothing”架构，并行处理，内存密集型处理框架，如 Spark 和 Impala，以及 YARN 容量调度程序中的资源抢占。

缩放数据仓库可能会很昂贵。添加额外的高端硬件容量以及获取数据仓库工具的许可证可能会显著增加成本。基于 Hadoop 的解决方案不仅在商品硬件节点和开源工具方面更便宜，而且还可以通过将数据转换卸载到 Hadoop 工具（如 Spark 和 Impala）来补足数据仓库解决方案，从而更高效地并行处理大数据。这也将释放数据仓库资源。

探索新的渠道和线索。Hadoop 可以为数据科学家提供探索性的沙盒，以从社交媒体，日志文件，电子邮件等地方发现潜在的有价值的数据，这些数据通常在数据仓库中不可得。

更好的灵活性。通常业务需求的改变，也需要对架构和报告进行更改。基于 Hadoop 的解决方案不仅可以灵活地处理不断发展的模式，还可以处理来自不同来源，如社交媒体，应用程序日志文件，image，PDF 和文档文件的半结构化和非结构化数据。

4、运行 Hadoop 集群需要哪些守护进程？

DataNode，NameNode，TaskTracker 和 JobTracker 都是运行 Hadoop 集群需要的守护进程。

5、Hadoop 支持哪些操作系统部署？

Hadoop 的主要操作系统是 Linux。 但是，通过使用一些额外的软件，也可以在 Windows 平台上部署，但这种方式不被推荐。

6、Hadoop 常见输入格式是什么？

三种广泛使用的输入格式是：
•文本输入：Hadoop 中的默认输入格式。
•Key 值：用于纯文本文件
•序列：用于依次读取文件

7、RDBMS 和 Hadoop 的主要区别是什么？

RDBMS 用于事务性系统存储和处理数据，而 Hadoop 可以用来存储大量数据。

8、namenode 的重要性是什么？

namenonde 的作用在 Hadoop 中非常重要。它是 Hadoop 的大脑，主要负责管理系统上的分配块，还为客户提出请求时的数据提供特定地址。

9、简要描述如何安装配置 Apache 的一个开源 Hadoop，只描述即可，无需列出具体步骤，列出具体步骤更好。

1）使用 root 账户登录
2）修改 IP
3）修改 host 主机名
4）配置 SSH 免密码登录
5）关闭防火墙
6）安装 JDK
7）解压 Hadoop 安装包
8）配置 Hadoop 的核心文件 hadoop-env.sh，core-site.xml , mapred-site.xml ， hdfs-site.xml
9）配置 Hadoop 环境变量
10）格式化 Hadoop namenode-format
11）启动节点 start-all.sh

10、列出 Hadoop 集群中 Hadoop 都分别需要启动 哪些进程，他们的作用分别都是什么？

namenode：负责管理 hdfs 中文件块的元数据，响应客户端请求，管理 datanode 上文件 block 的均衡，维持副本数量
Secondname:主要负责做 checkpoint 操作；也可以做冷备，对一定范围内数据做快照性备份。
Datanode:存储数据块，负责客户端对数据块的 io 请求
Jobtracker :管理任务，并将任务分配给 tasktracker。
Tasktracker: 执行 JobTracker 分配的任务。

11、项目所需要的大数据工作的一些 Hadoop 的工具

Hive，HBase, Ambari …

12、列出的 Hadoop 1 和 Hadoop 2 之间的差异。

在 Hadoop 的 1.x 中，“Namenode”有单点问题。在 Hadoop 的 2.x 中，我们有主动和被动“Namenodes”。如果主动“的 Namenode”失败，则被动“的 Namenode”负责。正因为如此，高可用性可以 Hadoop 中 2.x 中来实现

此外，在 Hadoop 的 2.X，YARN 提供了一个中央资源管理器。通过 YARN,你现在可以在 Hadoop 中运行多个应用程序，共享公共资源。 MR2 是一种特殊类型的运行于 YARN MapReduce 框架之上的分布式应用。其他工具也可以通过 YARN 执行数据处。

13、什么是主动和被动的“Namenodes”？

在 Hadoop 的 2.x 中，我们有两个 Namenodes-主动“Namenode” 被动“Namenode”。主动“Namenode”是在集群中运行的 Namenode。被动“的 Namenode”是一个备用“的 Namenode”，里面有主动“的 Namenode”的数据。当主动“Namenode”失败，则被动“Namenode”集群中替换主动“Namenode”。因此，集群是从来不会没有“Namenode”，所以它永远不会失败。

14、当两个客户端尝试访问对 HDFS 相同的文件，会发生什么？

HDFS 只支持独占写入。
当第一个客户端连接“Namenode”打开文件进行写入时，“Namenode”授予租约的客户端创建这个文件。当第二个客户端试图打开同一个文件写入时，“Namenode”会注意到该文件的租约已经授予给另一个客户端，并拒绝第二个客户端打开请求

15、为什么我们有时会得到一个“文件只能被复制到 0 节点，而不是 1”的错误？

这是因为“的 Namenode”没有任何可用的 DataNodes。

16、怎样才能在 HDFS 关闭“安全模式”？

您可以使用命令：Hadoop dfsadmin –safemode

17、如何在 HDFS 定义“block”？Hadoop1 和 2 中 Hadoop 块大小是多少？是否可以改变？

“块”是可被读取或写入的数据的最小量。 HDFS 中的文件被分解成块大小的块，它们被存储作为独立的单元。
Hadoop 的 1 默认块大小：64 MB
Hadoop 的 2 默认块大小：128 MB
是，块可以被配置。该 dfs.block.size 参数可在 HDFS-site.xml 文件被用来设置一个块的大小。

18、为什么 Hadoop 适用于大型数据集的应用程序，而不是具有大量的小文件的应用程序？

相较于在多个分布数据量小的文件 ，HDFS 更适合在一个文件中具有大量的数据集。这是因为“Namenode”是非常昂贵的，高性能的系统中，它是不慎重的占据“Namenode”通过了为多个小文件生成的元数据的不必要量的空间。因此，当在一个单独文件中的大量的数据，“Namenode”将占据更少的空间。因此，为获得最佳的性能，HDFS 支持大数据集，而不是多个小文件。

19、什么是传统的关系型数据库和 Hadoop 之间的基本区别？

传统的 RDBMS 是用于交易系统报告和存档数据，而 Hadoop 是存储和处理的分布式文件系统的海量数据的方法。当你想寻求大数据的一个记录 RDBMS 将是有益的。因此，当你在一次存储很大的文件并随后进行分析，Hadoop 将是有益的。

20、为什么在 HDFS，“读”是并行的，但“写”不是？

使用的 MapReduce 程序，该文件可以通过分割成块被读取。不过，写入时 MapReduce 并行 不能适用。

21、如果您在尝试访问 HDFS 或者其相应的文件得到一个“连接被拒绝 Java 异常’的错误会发生什么？

这可能意味着“Namenode”不工作了。“Namenode”可能是在“安全模式”或“Namenode”的 IP 地址可能改变了。

22、什么是“HDFS 块”和“输入分片”之间的区别？

“HDFS 块”是数据的基本物理单元，而“输入分割”是数据的逻辑划分。

23、什么是“全分布式”模式的特点是什么？

“全分布式”模式，在生产环境中使用，在那里我们有形成 Hadoop 集群的机器'N'号。 Hadoop 守护进程在计算机集群上运行。有一些“的 Namenode”运行一台主机，另一台主机上“Datanode”运行，然后有一些“TaskTracker /NodeManager”运行的机器。我们有独立的主机和从机在这种分布的。

24:Hadoop 可以运行在三种模式。

在其中的 Hadoop 可以运行三种模式为：

1.独立（本地）模式
2.伪分布式模式
3.完全分布式模式

25、“zookeeper”在 Hadoop 集群中的作用？

“zookeeper”的目的是集群管理。 “zookeeper”将帮助你实现的 Hadoop 节点之间的协调。 也有助于：

- 管理跨节点配置
- 实现可靠的消息传递
- 实现冗余服务
- 同步流程执行

26、什么是“MapReduce”？

它是一个框架或用于通过使用分布式编程的计算机的集群处理大型数据集的编程模型。

27、Hbase 的特点是什么？

(1) Hbase 一个分布式的基于列式存储的数据库,基于 Hadoop 的 hdfs 存储，zookeeper 进行管理。
(2) Hbase 适合存储半结构化或非结构化数据，对于数据结构字段不够确定或者杂乱无章很难按一个概念去抽取的数据。
(3) Hbase 为 null 的记录不会被存储.
(4)基于的表包含 rowkey，时间戳，和列族。新写入数据时，时间戳更新，同时可以查询到以前的版本.
(5) hbase 是主从架构。hmaster 作为主节点，hregionserver 作为从节点。

28、Hbase 和 Hive 有什么区别？

Hive 和 Hbase 是两种基于 Hadoop 的不同技术--Hive 是一种类 SQL 的引擎，并且运行 MapReduce 任务，Hbase 是一种在 Hadoop 之上的 NoSQL 的 Key/vale 数据库。当然，这两种工具是可以同时使用的。就像用 Google 来搜索，用 FaceBook 进行社交一样，Hive 可以用来进行统计查询，HBase 可以用来进行实时查询，数据也可以从 Hive 写到 Hbase，设置再从 Hbase 写回 Hive。

29、描述 Hbase 的 rowKey 的设计原则。

Rowkey 长度原则：

Rowkey 是一个二进制码流，Rowkey 的长度被很多开发者建议说设计在 10~100 个字节，不过建议是越短越好，不要超过 16 个字节。原因如下：

（1）数据的持久化文件 HFile 中是按照 KeyValue 存储的，如果 Rowkey 过长比如 100 个
字节，1000 万列数据光 Rowkey 就要占用 100*1000 万=10 亿个字节，将近 1G 数据，这会极
大影响 HFile 的存储效率；

（2）MemStore 将缓存部分数据到内存，如果 Rowkey 字段过长内存的有效利用率会降
低，系统将无法缓存更多的数据，这会降低检索效率。因此 Rowkey 的字节长度越短越好。

（3）目前操作系统是都是 64 位系统，内存 8 字节对齐。控制在 16 个字节，8 字节的整数倍利用操作系统的最佳特性。

Rowkey 散列原则：

如果 Rowkey 是按时间戳的方式递增，不要将时间放在二进制码的前面，建议将 Rowkey 的高位作为散列字段，由程序循环生成，低位放时间字段，这样将提高数据均衡分布在每个 Regionserver 实现负载均衡的几率。如果没有散列字段，首字段直接是时间信息将产生所有。

新数据都在一个 RegionServer 上堆积的热点现象，这样在做数据检索的时候负载将会集中在个别 RegionServer，降低查询效率。

Rowkey 唯一原则，必须在设计上保证其唯一性。

30、描述 Hbase 中 scan 和 get 的功能以及实现的异同。

HBase 的查询实现只提供两种方式：

1、按指定 RowKey 获取唯一一条记录，get 方法（org.apache.hadoop.hbase.client.Get）

Get 的方法处理分两种 ：设置了 ClosestRowBefore 和没有设置的 rowlock .主要是用来保证行的事务性，即每个 get 是以一个 row 来标记的.一个 row 中可以有很多 family 和 column.

2、按指定的条件获取一批记录，scan 方法(org.apache.Hadoop.hbase.client.Scan)实现条件查询功能使用的就是 scan 方式.

1)scan 可以通过 setCaching 与 setBatch 方法提高速度(以空间换时间)；

2)scan 可以通过 setStartRow 与 setEndRow 来限定范围([start，end)start 是闭区间，
end 是开区间)。范围越小，性能越高。

3)、scan 可以通过 setFilter 方法添加过滤器，这也是分页、多条件查询的基础。

31、请详细描述 Hbase 中一个 Cell 的结构。

HBase 中通过 row 和 columns 确定的为一个存贮单元称为 cell。
Cell：由{row key, column(=<family> + <label>), version}唯一确定的单元。cell 中的数
据是没有类型的，全部是字节码形式存贮。

32、编写用 Hbase shell 创建一张表语句。

create "emp","enames","jobs" ##emp 表名 enamees 和 jobs 是列族

33、删除表语句需要几步怎么写语句？

第一步：Disanble 表名
第二步：Drop 表名

34、写出添加数据的 Hbase shell 语法。

Put “表名”,”行号”，“列族名:列名”,”列值”

35、请编写扫描 emp 表的 enames 的 name1 这个列。

scan 'emp', {COLUMNS=>'enames:name1'}

36、如何查看 Hbase 表机构。

Desc 表名

37、Hbase shell 如何删除一个单元格。

语法：Delete “表名”，“行号”，“列族：列名”
案例：

```
hbase(main):066:0> delete "emp","1","enames:name1"
0 row(s) in 0.0940 seconds

hbase(main):067:0> scan "emp"
ROW                           COLUMN+CELL
 1                            column=enames:name2, timestamp=154
 1                            column=jobs:job1, timestamp=154109
 1                            column=jobs:job2, timestamp=154109
 1                            column=jobs:job3, timestamp=154109
1 row(s) in 0.0610 seconds
```

38、说说 Hive 数据倾斜的问题。

Hive 数据倾斜原因：
1. key 分布不均匀
2. 业务数据本身的特性
3. SQL 语句造成数据倾斜

解决方法：
hive 设置 hive.map.aggr=true 和 hive.groupby.skewindata=true

有数据倾斜的时候进行负载均衡，当选项设定为 true,生成的查询计划会有两个 MR Job。第一个 MR Job 中，Map 的输出结果集合会随机分布到 Reduce 中，每个 Reduce 做部分聚合操作，并输出结果，这样处理的结果是相同 Group By Key 有可能被分发到不同的 Reduce 中，从而达到负载均衡的目的；第二个 MR Job 在根据预处理的数据结果按照 Group By Key 分布到 Reduce 中(这个过程可以保证相同的 Group By Key 被分布到同一个 Reduce 中)，最后完成最终的聚合操作。

SQL 语句调整:

选用 join key 分布最均匀的表作为驱动表。做好列裁剪和 filter 操作，以达到两表 join 的时候，数据量相对变小的效果。

大小表 Join： 使用 map join 让小的维度表（1000 条以下的记录条数）先进内存。在 Map 端完成 Reduce。

大表 Join 大表：把空值的 Key 变成一个字符串加上一个随机数，把倾斜的数据分到不同的 reduce 上，由于 null 值关联不上，处理后并不影响最终的结果。

count distinct 大量相同特殊值：count distinct 时，将值为空的情况单独处理，如果是计算 count distinct，可以不用处理，直接过滤，在做后结果中加 1。如果还有其他计算，需要进行 group by，可以先将值为空的记录单独处理，再和其他计算结果进行 union.

39、Hive 中的排序关键字有哪些？

sort by ， order by ， cluster by ， distribute by

sort by ： 不是全局排序，其在数据进入 reducer 前完成排序

order by ： 会对输入做全局排序，因此只有一个 reducer(多个 reducer 无法保证全局有序).只有一个 reducer,会导致当输入规模较大时，需要较长的计算时间。

cluster by ： 当 distribute by 和 sort by 的字段相同时，等同于 cluster by.可以看做特殊的 distribute + sort

distribute by ： 按照指定的字段对数据进行划分输出到不同的 reduce

40、Hive 内部表和外部表区别？

创建表时：创建内部表时，会将数据移动到数据仓库指向的路径；若创建外部表，仅记录数据所在的路径， 不对数据的位置做任何改变。

删除表时：在删除表的时候，内部表的元数据和数据会被一起删除， 而外部表只删除元数据，不删除数据。这样外部表相对来说更加安全些，数据组织也更加灵活，方便共享源数据。

41、Hive 分区和分桶的区别？

分区：

是指按照数据表的某列或某些列分为多个区，区从形式上可以理解为文件夹，比如我们要收集某个大型网站的日志数据，一个网站每天的日志数据存在同一张表上，由于每天会生成大量的日志，导致数据表的内容巨大，在查询时进行全表扫描耗费的资源非常多。

那其实这个情况下，我们可以按照日期对数据表进行分区，不同日期的数据存放在不同的分区，在查询时只要指定分区字段的值就可以直接从该分区查找

分桶：

分桶是相对分区进行更细粒度的划分。

分桶将整个数据内容安装某列属性值得 hash 值进行区分，如要按照 name 属性分为 3 个桶，就是对 name 属性值的 hash 值对 3 取摸，按照取模结果对数据分桶。

如取模结果为 0 的数据记录存放到一个文件，取模为 1 的数据存放到一个文件，取模为 2 的数据存放到一个文件

42、Hive 语句

有三张表 course(课程表)、students(学生表)、sc(成绩表)如下图：

表 Course

```
1,数据库
2,数学
3,信息系统
4,操作系统
5,数据结构
6,数据处理
```

表 students

```
95001,李勇,男,20,CS
95002,刘晨,女,19,IS
95003,王敏,女,22,MA
95004,张立,男,19,IS
95005,刘刚,男,18,MA
95006,孙庆,男,23,CS
95007,易思玲,女,19,M
95008,李娜,女,18,CS
95009,梦圆圆,女,18,M
95010,孔小涛,男,19,C
95011,包小柏,男,18,M
95012,孙花,女,20,CS
95013,冯伟,男,21,CS
95014,王小丽,女,19,C
95015,王君,男,18,MA
95016,钱国,男,21,MA
95017,王风娟,女,18,I
95018,王一,女,19,IS
95019,邢小丽,女,19,I
95020,赵钱,男,21,IS
95021,周二,男,17,MA
95022,郑明,男,20,MA
```

表 sc

```
95001,1,81
95001,2,85
95001,3,88
95001,4,70
95002,2,90
95002,3,80
95002,4,71
95002,5,60
95003,1,82
95003,3,90
95003,5,106
95004,1,80
95004,2,92
95004,4,91
95004,5,70
95005,1,70
95005,2,92
95005,3,99
95005,6,87
95006,1,72
95006,2,62
95006,3,106
95006,4,59
```

题目要求和答案：

1）查询全体学生的学号与姓名

```
select Sno, Sname from student;
```

2）查询选修了课程的学生姓名

```
select Sname from student s where s.Sdept is not null;
```

----hive 的 group by 和集合函数

3）查询学生的总人数

```
select count(*) from student;
```

4）计算 1 号课程的学生平均成绩

```
select avg(grade) from sc where cno=1;
```

5）查询各科成绩平均分

```
select cname,avggrade from
  course c,
  (select    cno,avg(grade) avggrade from sc group by cno) s
  where c.cno = s.cno;
```

6）查询选修 1 号课程的学生最高分数

```
select max(grade) from sc where cno=1;
```

7）求各个课程号及相应的选课人数

```
select cno,count(cno) from sc group by cno;
```

8）查询选修了 3 门以上的课程的学生学号

```
  select sno  学生编号,count(sno)  人数  from sc group by sno having  人数>3;
```

9）查询学生信息，结果按学号全局有序
select * from student order by sno;
10）查询学生信息，结果区分性别按年龄有序
select * from student order by sex,sage;
11）查询每个学生及其选修课程的情况
hive> select student.*,sc.* from student join sc on (student.Sno =sc.Sno);
12）查询学生的得分情况。
hive>select student.Sname,course.Cname,sc.Grade from student join sc on student.Sno=sc.Sno join course on sc.cno=course.cno;
13）查询选修 2 号课程且成绩在 90 分以上的所有学生。
----LEFT，RIGHT 和 FULL OUTER JOIN
14）查询所有学生的信息，如果在成绩表中有成绩，则输出成绩表中的课程号
----LEFT SEMI JOIN Hive 当前没有实现 IN/EXISTS 子查询，可以用 LEFT SEMI JOIN 重写子查询语句。
重写以下子查询

```
SELECT a.key, a.value
FROM a
WHERE a.key in
 (SELECT b.key
  FROM B)
```

15）查询与“刘晨”在同一个系学习的学生
hive> select s1.Sname from student s1 left semi join student s2 on s1.Sdept=s2.Sdept and s2.Sname='刘晨';

九. 阿里面试总结：如何介绍自己的项目经验

序言

在面试时，经过寒暄后，一般面试官会让你介绍项目经验。常见的问法是：“说下你最近的（或最拿得出手的）一个项目”。

可能不少程序员对此没准备，说起来磕磕巴巴，甚至有人说出项目经验从时间段或技术等方面和简历上的不匹配，这样就会造成如下的后果：
第一印象就不好了，至少会感觉该候选人表述能力不强。

一般来说，面试官会根据程序员介绍的项目背景来提问题。假设面试时会问 10 个问题，那么至少有 5 个问题会根据程序员所介绍的项目背景来问，程序员如果没说好，那么就没法很好地引导后继问题了，就相当于把提问权完全交给面试官了。

面试时 7 份靠能力，3 份靠技能。而刚开始时的介绍项目又是技能中的重中之重，所以下面将从“介绍”和“引导”两大层面告诉大家如何准备面试时的项目介绍。

1. 在面试前准备项目描述，别害怕，因为面试官什么都不知道

面试官是人，不是神，拿到你的简历的时候，是没法核实你的项目细节的（有些公司会到录用后，用背景调查的方式来核实）。

更何况，你做的项目是以月为单位算的，而面试官最多用 30 分钟来从你的简历上了解你的项目经验，所以你对项目的熟悉程度要远远超过面试官，所以你一点也不用紧张。如果你的工作经验比面试官还丰富的话，甚至还可以控制整个面试流程。

	你	面试官
对你以前的项目和技能	很了解	只能听你说，只能根据你说的内容做出判断
在面试过程中的职责	在很短的时间内防守成功即可	如果找不出漏洞，就只能算你以前做过
准备时间	面试前你有充足的时间准备	一般在面试前用30分钟阅读你的简历
沟通过程	你可以出错，但别出关键性的错误	不会太为难你，除非你太差
技巧	你有足够的技巧，也可以从网上找到足够多的面试题	其实就问些通用的有规律的问

既然面试官无法了解你的底细，那么他们怎么来验证你的项目经验和技术？下面总结了一些常用的提问方式。

提问方式	目的
让你描述工作经验和项目（极有可能是最近的），看看你说的是否和简历上一致	看你是否真的做过这些项目
看你简历上项目里用到的技术，比如框架、数据库，然后针对这些技术提些基本问题	还是验证你是否做过项目，同时看你是否了解这些技术，为进一步提问做准备
针对某个项目，不断深入地问一些技术上的问题，或者从不同侧面问一些技术实现，看你前后回答里面是否有矛盾	深入核实你的项目细节
针对某技术，问些项目里一定会遇到的问题，比如候选人说做过数据库，那么就会问索引方面的问题	通过这类问题，核实候选人是否真的有过项目经验（或者还仅是学习经验）

2. 准备项目的各种细节，一旦被问倒了，就说明你没做过

一般来说，在面试前，大家应当准备项目描述的说辞，自信些，因为这部分你说了算，流利些，因为你经过充分准备后，可以知道你要说些什么。而且这些是你实际的项目经验，那么一旦让面试官感觉你都说不上来，那么可信度就很低了。

不少人是拘泥于“项目里做了什么业务，以及代码实现的细节”，这就相当于把后继提问权直接交给面试官。下表列出了一些不好的回答方式。
在避免上述不好的回答的同时，大家可以按下表所给出的要素准备项目介绍。如果可以，也请大家准备一下用英语描述。其实刚毕业的学生，或者工作经验较少的人，英语能力都差不多，但你说了，这就是质的进步。

要素	样式
控制在1分钟里面，讲出项目基本情况，比如项目名称，背景，给哪个客户做，完成了基本的事情，做了多久，项目规模多大，用到哪些技术，数据库用什么，然后酌情简单说一下模块。重点突出背景，技术，数据库和其他和技术有关的信息。	我在XX公司做了XX外汇保证金交易平台，客户是XX银行，主要完成了挂盘，实盘成交，保证金杠杆成交等功能，数据库是Oracle，前台用到JS等技术，后台用到Java的SSH，几个人做了X个月。不需要详细描述各功能模块，不需要说太多和业务有关但和技术无关的。如果面试官感兴趣，等他问。
要主动说出你做了哪些事情，这部分的描述一定需要和你的技术背景一致。	我做了外汇实盘交易系统，挂单成交系统，XXX模块，做了X个月
描述你在项目里的角色	我主要是做了开发，但在开发前，我在项目经理的带领下参与了业务调研，数据库设计等工作，后期我参与了测试和部署工作。
可以描述用到的技术细节，特别是你用到的技术细节，这部分尤其要注意，你说出口的，一定要知道，因为面试官后面就根据这个问的。 你如果做了5个模块，宁可只说你能熟练说上口的2个。	用到了Java里面的集合，JDBC，..等技术，用到了Spring MVC等框架，用技术连接数据库。
这部分你风险自己承担，如果可以，不露声色说出一些热门的要素，比如Linux，大数据，大访问压力等。但一旦你说了，面试官就会直接问细节。	这个系统里，部署在Linux上，每天要处理的数据量是XX，要求是在4小时，1G内存是的情况下处理完5千万条数据。平均访客是[illegible]@卖小女孩的火柴

面试前，一定要准备，一定要有自信，但也要避免如下的一些情况：

要避免的情况	正确的做法	原因
回答很简单。问什么答什么，往往就用一句话回答	把你知道的都说出来，重点突出你知道的思想，框架	问：你SSH用过吗? 答：用过。 问：在什么项目里用到? 答：一个保险项目 问：你做了哪方面的事情? 答：开发 我直接不问了
说得太流利	适当停顿，边思考边说	让面试官感觉你在背准备的东西，这样后面问题就很难
项目介绍时什么都说，	就说些刚才让准备的一些，而且要有逻辑地说	会让面试官感觉你思路太乱
别太多介绍技术细节，就说你熟悉的技术	技术面点到为止，等面试官来问	你说到的所有技术要点，都可能会被深问。面试官一般会有自己的面试节奏，如果你在介绍时就太多说技术细节，很有可能被打断，从而没法说出你准备好的亮点。 头条 @买小女孩的火柴

3. 一定要主动，面试官没有义务挖掘你的亮点

作为面试者，应当主动说出自己的亮点和优势，而不是等着问，但请注意，说的时候要有技巧，找机会说，通常是找一些开放性的问题说。

比如：在这个项目里用到了什么技术？你除了说一些基本的技术，比如Spring MVC，Hibernate，还有数据库方面的常规技术时，还得说，用到了 Java 内存管理，这样能减少对虚拟机内存的压力，或者说用到了大数据处理技术等。

面试的时候，如果程序员回答问题很简单，有一说一，不会扩展，或者用非常吝啬的语句来回答问题，有些面试官一般会给机会让他们深入讲述（但不能保证每个面试官都会深入提问），如果回答再简洁，那么也会很吝啬地给出好的评语。

记住：面试官不是你的亲戚，面试官很忙，能挖掘出你的亮点的面试官很少，而说出你的亮点是你的义务。

4. 准备些加分点，在介绍时有意提到，但别说全

在做项目介绍的时候，你可以穿插说出一些你的亮点，但请记得，不论在介绍项目还是在回答问题，你当前的职责不是说明亮点而是介绍项目，一旦你详细说，可能会让面试官感觉你跑题了。所以这时你可以一笔带过，比如你可以说，“我们的项目对数据要求比较大，忙的时候平均每小时要处理几十万条数据”，这样就可以把面试官引入“大数据”的方向。

你在面试前可以根据职位的需求，准备好这种“一笔带过”的话。比如这个职位的需求点是 Spring MVC 框架，大数据高并发，要有数据库调优经验，那么介绍以往项目时，你就最好突出这些方面你的实际技能。

面试过程中，面试官一旦听到有亮点，就会等到他说好当前问题后，顺口去问，一般技术面试最多办半小时，你把时间用在回答准备好的问题点上的时候，被问其他问题的时间就会少了。

本文档给出是的方法，不是教条，大家可以按本文给出的方向结合自己的项目背景做准备，而不是死记硬背本文给出的一些说辞。