尚学堂 Java 面试题大全及其答案

1. 下面程序的运行结果是（B）(选择一项）

```
String str1="hello";
String str2=new String("hello");
System.out.println(str1==str2);
```

A	true
B.	false
C.	hello
D.	he

分析：str1 没有使用 new 关键字，在堆中没有开辟空间，其值”hello”在常量池中，str2 使用 new 关键字创建了一个对象，在堆中开辟了空间，” ==”比较的是对象的引用，即内存地址，所以 str1 与 str2 两个对象的内存地址是不相同的

2. 下列说法正确的有（C）(选择一项）

A	class中的construtor不可省略
B.	construtor 与 class 同名，但方法不能与 class 同外
C.	construtor在一个对象被new时执行

D.	一个 class 只能定义一个 construtor
分析：A：如果 class 中的 construtor 省略不写，系统会默认提供一个无参构造 B：方法名可以与类名同名，只是不符合命名规范 D：一个 class 中可以定义 N 多个 construtor，这些 construtor 构成构造方法的重载	

3. 下面程序的运行结果（B）(选择一项)

```
public static void main(String[] args) {
        Thread t=new Thread(){
            public void run(){
                pong();
            }
        };
        t.run();
        System.out.println("ping");
    }
    static void pong(){
        System.out.println("pong");
}
```

A	pingpong
B.	pongping

C.	pingpong和pongping都有可能
D.	都不输出
分析：启动线程需要调用 start()方法，而 t.run()方法，则是使用对象名.方法名()并没有启动线程，所以程序从上到下依次执行，先执行 run()中的静态方法 pong()输出 pong 然执行打印输出 ping	

4. 下列说法正确的有（C）(选择一项）

A.	LinkedList继承自List
B.	AbstractSet 继承自 Set
C.	HashSet继承自AbstractSet
D.	TreeMap 继承自 HashMap
分析：A：LinkedList 实现 List 接口 B：AbstractSet 实现 Set 接口 D：TreeMap 继承 AbstractMap	

5. 下面哪个流类属于面向字符的输入流（D）选择一项）

A	BufferedWriter
B.	FileInputStream
C.	ObjectInputStream
D.	InputStreamReader
分析：A：字符输出的缓冲流 B：字节输入流	

C：对象输入流

6. Java 中接口的修饰符可以为（D）(选择一项)

A	private
B.	protected
C.	final
D.	abstract
分析：接口中的访问权限修饰符只可以是 public 或 default 接口中的所有的方法必须要实现类实现，所以不能使用 final 接口中所有的方法默认都是 abstract 的，所以接口可以使用 abstract 修饰，但通常 abstract 可以省略不写	

7. 下列哪种异常是检查型异常，需要在编写程序时声明（C）

A	NullPointerException
B.	ClassCastException
C.	FileNotFoundException
D.	IndexOutOfBoundsException
分析：NullPointerException 空指针异常 ClassCastException 类型转换异常 IndexOutOfBoundsException 索引超出边界的异常 以上这些异常都是程序在运行时发生的异常，所以不需要在编写程序时声明	

8. 关于 Java 编译，下面哪一个正确（B）(选择一项)

A	Java程序经编译后产生machine code
B.	Java 程序经编译后会生产 byte code
C.	Java程序经编译后会产生DLL
D.	以上都不正确
分析： Java 是解释型语言，编译出来的是字节码； 因此 A 不正确，C 是C/C++语言编译动态链接库的文件为.DLL； 正确答案为 B	

9. 给定以下代码，程序将输出 （B）(选择一项)

```
class A {
        public A(){
            System.out.println("A");
        }
    }
    class B extends A{
        public B(){
            System.out.println("B");
        }
        public static void main(String[] args) {
            B b=new B();
        }
    }
```

A	不能通过编译
B.	通过编译，输出 AB
C.	通过编译，输出B
D.	通过编译，输出 A
分析：在继承关系下，创建子类对象，先执行父类的构造方法，再执行子类的构造方法。	

10. 下面哪些是 Thread 类的方法（AD）(选择两项）

A.	start()
B.	run()
C.	exit()
D.	getPriority()
分析： run()方法是 Runnable 接口中的方法 exit()是 System 类中的方法	

11. 下列属于关系型数据库的是（AB）(选择两项）

A.	Oracle
B.	MySql
C.	IMS
D.	MongoDB
分析： IMS 是 IP Mulitimedia Subsystem 的缩写，是 IP 多媒体系统 MongoDB 分布式文档存储数据库	

12. 下列关于关键字的使用说法错误的是（D）(选择一项)

A.	abstract不能与final并列修饰同一个类
B.	abstract 类中可以有 private 的成员
C.	abstract方法必须在abstract类中
D.	static 方法能处理非 static 的属性
分析：因为 static 得方法在装载 class 得时候首先完成，比 构造方法早，此时非 static 得属性和方法还没有完成初始化所以不能调用。	

13. Java 语言中，String 类中的 indexOf()方法返回值的类型是（C）

A	int16
B.	int32
C.	int
D.	long
分析：	

14. 给定以下代码，程序的运行结果是 （ B）(选择一项)

```
public class Example {
    String str=new String("good");
    char [] ch={'a','b','c'};
    public static void main(String[] args) {
        Example ex=new Example();
        ex.change(ex.str, ex.ch);
```

```
            System.out.print(ex.str+"and");
            System.out.print(ex.ch);
        }
        public void change(String    str,char ch[]){
            str="test ok";
            ch[0]='g';
        }
    }
```

A	goodandabc
B.	goodandgbc
C.	test okandabc
D.	test okandgbc

分析：在方法调用时，在 change 方法中对 str 的值进行修改，是将 str 指向了常量江池中的”test ok”，而主方法中的 ex.str 仍然指向的是常量池中的”good”。字符型数组在方法调用时，将主方法中 ex.ch 的引用传递给 change 方法中的 ch，指向是堆中的同一堆空间，所以修改 ch[0]的时候,ex.ch 可以看到相同的修改后的结果。

15. 要从文件”file.dat”文件中读出第 10 个字节到变量 c 中，下列哪个正确（A）(选择一项)

A.	FileInputStream in=**new** FileInputStream("file.dat"); in.skip(9); **int** c=in.read();

B.	FileInputStream in=**new** FileInputStream("file.dat"); in.skip(10); **int** c=in.read();
C.	FileInputStream in=**new** FileInputStream("file.dat"); **int** c=in.read();
D.	RandomAccessFile in=**new** RandomAccessFile("file.dat"); in.skip(7); **int** c=in.readByte();
分析： skip(long n)该方法中的 n 指的是要跳过的字节数	

16. 下列哪些语句关于内存回收的说法是正确的（B）(选择一项)

A.	程序员必须创建一个线程来释放内存
B.	内存回收程序员负责释放无用内存
C.	内存回收程序允许程序员直接释放内存
D.	内存回收程序可以在指定的时间释放内存对象
分析： A. 程序员不需要创建线程来释放内存. C. 也不允许程序员直接释放内存. D. 不一定在什么时刻执行垃圾回收.	

17. 执行下列代码后，哪个结论是正确的（BD）(选择两项)

String[] s=**new** String[10];	
A.	s[10]为””
B.	s[9]为 null

C.	s[0]为未定义
D.	s.length 为 10
分析： 引用数据类型的默认值均为 null s.length 数组的长度	

18. 选出合理的标识符（AC）(选择两项）

A.	_sysl_111
B.	2 mail
C.	$change
D.	class
分析： 标识符的命令规范，可以包含字母、数字、下划线、$，不能以数字开头，不能是 Java 关键字	

19. 下列哪个方法可用于创建一个可运行的类（A）(选择两项）

A.	public class X implements Runnable{public void run() {……}}
B.	public class X extends Thread{public void run() {……}}
C.	public class X extends Thread{public int run() {……}}
D.	public class X implements Runnable{protected void run() {……}}
分析： 继承 Thread 和实现 Runable 借口	

20. 下列说法正确的是（BCD）(选择多项）

A.	java.lang.Cloneable是类
B.	java.langRunnable 是接口
C.	Double对象在java.lang包中

D.	Double a=1.0 是正确的 java 语句
分析：java.lang.Cloneable 是接口	

21. 定义一个类名为”MyClass.java”的类，并且该类可被一个工程中的所有类访问，那么该类的正确声明为（CD）(选择两项)

A.	private class MyClass extends Object
B.	class MyClass extends Object
C.	public class MyClass
D.	public class MyClass extends Object
分析： A 类的访问权限只能是 public 或 default B 使用默认访问权限的类，只能在本包中访问	

22. 面向对象的特征有哪些方面？请用生活中的例子来描述。

答: 面向对象的三大特征：封装、继承、多态

23. 说明类 java.lang.ThreadLocal 的作用和原理。列举在哪些程序中见过 ThreadLocal 的使用？

作用：

要编写一个多线程安全(Thread-safe)的程序是困难的 ,为了让线程共享资源，必须小心地对共享资源进行同步，同步带来一定的效能延迟，而另一方面，在处理同步的时候，又要注意对象的锁定与释放，避免产生死结，种种因素都使得编写多线程程序变得困难。

尝试从另一个角度来思考多线程共享资源的问题，既然共享资源这么困难，

那么就干脆不要共享，何不为每个线程创造一个资源的复本。将每一个线程存取数据的行为加以隔离，实现的方法就是给予每个线程一个特定空间来保管该线程所独享的资源。

ThreadLocal 的原理

ThreadLocal 是如何做到为每一个线程维护变量的副本的呢？其实实现的思路很简单，在 ThreadLocal 类中有一个 Map，用于存储每一个线程的变量的副本。

24. 说明内存泄漏和内存溢出的区别和联系，结合项目经验描述 Java 程序中如何检测？如何解决？

答：

内存溢出 out of memory，是指程序在申请内存时，没有足够的内存空间供其使用，出现 out of memory；比如申请了一个 integer,但给它存了 long 才能存下的数，那就是内存溢出。

内存泄露 memory leak，是指程序在申请内存后，无法释放已申请的内存空间，一次内存泄露危害可以忽略，但内存泄露堆积后果很严重，无论多少内存,迟早会被占光。

memory leak 会最终会导致 out of memory！

25. 什么是 Java 的序列化，如何实现 Java 的序列化？列举在哪些程序中见过 Java 序列化？

答：Java 中的序列化机制能够将一个实例对象（只序列化对象的属性值，而不会去序列化什么所谓的方法。）的状态信息写入到一个字节流中使其可以通过 socket 进行传输、或者持久化到存储数据库或文件系统中；然后在需

要的时候通过字节流中的信息来重构一个相同的对象。一般而言，要使得一个类可以序列化，只需简单实现 java.io.Serializable 接口即可。

26. String 和 StringBuffer、StringBuilder 的区别是什么？

答：

1、String 类是不可变类，即一旦一个 String 对象被创建后，包含在这个对象中的字符序列是不可改变的，直至这个对象销毁。

2、 StringBuffer 类则代表一个字符序列可变的字符串，可以通过 append、insert、reverse、setChartAt、setLength 等方法改变其内容。一旦生成了最终的字符串，调用 toString 方法将其转变为 String

3、JDK1.5 新增了一个 StringBuilder 类，与 StringBuffer 相似，构造方法和方法基本相同。不同是 StringBuffer 是线程安全的，而 StringBuilder 是线程不安全的，所以性能略高。通常情况下，创建一个内容可变的字符串，应该优先考虑使用 StringBuilder

27. 不通过构造函数也能创建对象吗？

答：Java 创建对象的几种方式（重要）:

1、 用 new 语句创建对象，这是最常见的创建对象的方法。

2、运用反射手段,调用 java.lang.Class 或者 java.lang.reflect.Constructor 类的 newInstance()实例方法。

3、 调用对象的 clone()方法。

4、运用反序列化手段，调用 java.io.ObjectInputStream 对象的 readObject()方法。

(1)和(2)都会明确的显式的调用构造函数 ；(3)是在内存上对已有对象的影

印，所以不会调用构造函数 ；(4)是从文件中还原类的对象，也不会调用构造函数。

28. 对象在虚拟机的大小中可不可以用 size of 取出。

答：可以

29. 乐观锁与悲观锁

答：

悲观锁(Pessimistic Lock)，顾名思义，就是很悲观，每次去拿数据的时候都认为别人会修改，所以每次在拿数据的时候都会上锁，这样别人想拿这个数据就会 block 直到它拿到锁。传统的关系型数据库里边就用到了很多这种锁机制，比如行锁，表锁等，读锁，写锁等，都是在做操作之前先上锁。

乐观锁(Optimistic Lock)，顾名思义，就是很乐观，每次去拿数据的时候都认为别人不会修改，所以不会上锁，但是在更新的时候会判断一下在此期间别人有没有去更新这个数据，可以使用版本号等机制。乐观锁适用于多读的应用类型，这样可以提高吞吐量，像数据库如果提供类似于 write_condition 机制的其实都是提供的乐观锁。

两种锁各有优缺点，不可认为一种好于另一种，像乐观锁适用于写比较少的情况下，即冲突真的很少发生的时候，这样可以省去了锁的开销，加大了系统的整个吞吐量。但如果经常产生冲突，上层应用会不断的进行 retry，这样反倒是降低了性能，所以这种情况下用悲观锁就比较合适。

30. 描述 Java 权限修饰符(public、private、protected、默认)的不同

答：

访问控制	public	protected	默认	private
同一类中成员	是	是	是	是
同一包中其它类	是	是	是	
不同包中的子类	是	是		
不同包中对非子类	是			

类的访问权限只有两种

public 公共的 可被同一项目中所有的类访问。 (必须与文件名同名)

default 默认的 可被同一个包中的类访问。

成员（成员变量或成员方法）访问权限共有四种：

public 公共的 可以被项目中所有的类访问。(项目可见性)

protected 受保护的 可以被这个类本身访问；同一个包中的所有其他的类访问；被它的子类（同一个包以及不同包中的子类）访问。（子类可见性）

default 默认的 被这个类本身访问；被同一个包中的类访问。（包可见性）

private 私有的 只能被这个类本身访问。（类可见性）

31. 在 Java 中怎么实现多线程?描述线程状态的变化过程。

答：当多个线程访问同一个数据时，容易出现线程安全问题，需要某种方式来确保资源在某一时刻只被一个线程使用。需要让线程同步，保证数据安全

线程同步的实现方案：**同步代码块和同步方法，均需要使用 synchronized 关键字**

同步代码块：public void makeWithdrawal(int amt) {

synchronized (acct) { }

}

同步方法：public synchronized void makeWithdrawal(int amt) { }

线程同步的好处：解决了线程安全问题

线程同步的缺点：性能下降，可能会带来死锁

32. 简述 Java 内存管理机制 ,以及垃圾回收的原理和使用过 Java 调优工具

内存管理的职责为分配内存，回收内存。 没有自动内存管理的语言/平台容易发生错误。

典型的问题包括悬挂指针问题，一个指针引用了一个已经被回收的内存地址，导致程序的运行完全不可知。

另一个典型问题为内存泄露，内存已经分配，但是已经没有了指向该内存的指针，导致内存泄露。 程序员要花费大量时间在调试该类问题上。

33. 描述 JVM 加载 class 文件的原理机制

JVM 中类的装载是由类加载器（ClassLoader）和它的子类来实现的，Java 中的类加载器是一个重要的 Java 运行时系统组件，它负责在运行时查

找和装入类文件中的类。

由于 Java 的跨平台性 ,经过编译的 Java 源程序并不是一个可执行程序，而是一个或多个类文件。当 Java 程序需要使用某个类时，JVM 会确保这个类已经被加载、连接（验证、准备和解析）和初始化。类的加载是指把类的.class 文件中的数据读入到内存中，通常是创建一个字节数组读入.class 文件，然后产生与所加载类对应的 Class 对象。加载完成后，Class 对象还不完整，所以此时的类还不可用。当类被加载后就进入连接阶段，这一阶段包括验证、准备（为静态变量分配内存并设置默认的初始值）和解析（将符号引用替换为直接引用）三个步骤。最后 JVM 对类进行初始化，包括：1)如果类存在直接的父类并且这个类还没有被初始化，那么就先初始化父类；2)如果类中存在初始化语句，就依次执行这些初始化语句。

类的加载是由类加载器完成的，类加载器包括：根加载器（BootStrap）、扩展加载器（Extension）、系统加载器（System）和用户自定义类加载器（java.lang.ClassLoader 的子类）。 从 Java 2（JDK 1.2）开始，类加载过程采取了父亲委托机制（PDM）。PDM 更好的保证了 Java 平台的安全性，在该机制中，JVM 自带的 Bootstrap 是根加载器，其他的加载器都有且仅有一个父类加载器。类的加载首先请求父类加载器加载，父类加载器无能为力时才由其子类加载器自行加载。JVM 不会向 Java 程序提供对 Bootstrap 的引用。下面是关于几个类加载器的说明：

Bootstrap：一般用本地代码实现 ,负责加载 JVM 基础核心类库(rt.jar)；

Extension：从 java.ext.dirs 系统属性所指定的目录中加载类库，它的父加载器是 Bootstrap；

System：又叫应用类加载器，其父类是 Extension。它是应用最广泛的类加载器。它从环境变量 classpath 或者系统属性 java.class.path 所指定的目录中记载类，是用户自定义加载器的默认父加载器。

34. 请列出 Java 常见的开源数据连接池，并对参数做出简单的说明

答：在 Java 中开源的常用的数据库连接池有以下几种 ：

（1）DBCP

DBCP 是一个依赖 Jakarta commons-pool 对象池机制的数据库连接池.DBCP 可以直接的在应用程序中使用，Tomcat 的数据源使用的就是 DBCP。

（2）c3p0

c3p0 是一个开放源代码的 JDBC 连接池，它在 lib 目录中与 Hibernate 一起发布,包括了实现 jdbc3 和 jdbc2 扩展规范说明的 Connection 和 Statement 池的 DataSources 对象。

（3）Druid

阿里出品，淘宝和支付宝专用数据库连接池，但它不仅仅是一个数据库连接池，它还包含一个 ProxyDriver，一系列内置的 JDBC 组件库，一个 SQL Parser。支持所有 JDBC 兼容的数据库，包括 Oracle、MySql、Derby、Postgresql、SQL Server、H2 等等。

35. 请写出常用的 linux 指令不低于 10 个 ,请写出 linux tomcat 启动。

答：linux 指令

arch 显示机器的处理器架构(1)

uname -m 显示机器的处理器架构(2)

shutdown -h now 关闭系统(1)

shutdown -r now 重启(1)

cd /home 进入 '/ home' 目录'

cd .. 返回上一级目录

cd ../.. 返回上两级目录

mkdir dir1 创建一个叫做 'dir1' 的目录'

mkdir dir1 dir2 同时创建两个目录

find / -name file1 从 '/' 开始进入根文件系统搜索文件和目录

find / -user user1 搜索属于用户 'user1' 的文件和目录

linuxtomcat 启动

进入 tomcat 下的 bin 目录执行 ./catalina.sh start 直接启动即可，然后使用 tail -f /usr/local/tomcat6/logs/catalina.out 查看 tomcat 启动日志。

36. 请写出您熟悉的几种设计模式，并做简单介绍。

答：

工厂设计模式：程序在接口和子类之间加入了一个过渡端，通过此过渡端可以动态取得实现了共同接口的子类实例化对象。

代理设计模式：指由一个代理主题来操作真实主题，真实主题执行具体的业

务操作，而代理主题负责其他相关业务的处理。比如生活中的通过代理访问网络，客户通过网络代理连接网络（具体业务），由代理服务器完成用户权限和访问限制等与上网相关的其他操作（相关业务）

适配器模式：如果一个类要实现一个具有很多抽象方法的接口，但是本身只需要实现接口中的部分方法便可以达成目的，所以此时就需要一个中间的过渡类，但此过渡类又不希望直接使用，所以将此类定义为抽象类最为合适，再让以后的子类直接继承该抽象类便可选择性的覆写所需要的方法，而此抽象类便是适配器类。

37. 匿名内部类可不可以继承或实现接口。为什么？

答：匿名内部类是没有名字的内部类,不能继承其它类,但一个内部类可以作为一个接口,由另一个内部类实现.

1、由于匿名内部类没有名字，所以它没有构造函数。因为没有构造函数，所以它必须完全借用父类的构造函数来实例化，换言之：匿名内部类完全把创建对象的任务交给了父类去完成。

2、在匿名内部类里创建新的方法没有太大意义，但它可以通过覆盖父类的方法达到神奇效果，如上例所示。这是多态性的体现。

3、因为匿名内部类没有名字，所以无法进行向下的强制类型转换，持有对一个匿名内部类对象引用的变量类型一定是它的直接或间接父类类型。

38. Java 的 HashMap 和 Hashtable 有什么区别 HashSet 和 HashMap 有什么区别？使用这些结构保存的数需要重载的方法是哪些？

答：

HashMap 与 Hashtable 实现原理相同，功能相同，底层都是哈希表结构，查询速度快，在很多情况下可以互用

两者的主要区别如下

1、Hashtable 是早期 JDK 提供的接口，HashMap 是新版 JDK 提供的接口

2、Hashtable 继承 Dictionary 类，HashMap 实现 Map 接口

3、Hashtable 线程安全，HashMap 线程非安全

4、Hashtable 不允许 null 值，HashMap 允许 null 值

HashSet 与 HashMap 的区别

1、HashSet 底层是采用 HashMap 实现的。HashSet 的实现比较简单，HashSet 的绝大部分方法都是通过调用 HashMap 的方法来实现的，因此 HashSet 和 HashMap 两个集合在实现本质上是相同的。

2、HashMap 的 key 就是放进 HashSet 中对象，value 是 Object 类型的。

3、当调用 HashSet 的 add 方法时，实际上是向 HashMap 中增加了一行(key-value 对)，该行的 key 就是向 HashSet 增加的那个对象，该行的 value 就是一个 Object 类型的常量

39. 写出你用过的设计模式，并至少写出 2 种模式的类图或关键代码。

工厂设计模式：

思路说明：由一个工厂类根据传入的参数（一般是字符串参数），动态决定应该创建哪一个产品子类（这些产品子类继承自同一个父类或接口）的实例，并以父类形式返回

优点：客户端不负责对象的创建，而是由专门的工厂类完成；客户端只负责对象的调用，实现了创建和调用的分离，降低了客户端代码的难度；

缺点：如果增加和减少产品子类，需要修改简单工厂类，违背了开闭原则；如果产品子类过多，会导致工厂类非常的庞大，违反了高内聚原则，不利于后期维护。

```
public class SimpleFactory {
    public static Product createProduct(String pname){
        Product product=null;
        if("p1".equals(pname)){
            product = new Product1();
        }else if("p2".equals(pname)){
            product = new Product2();
        }else if("pn".equals(pname)){
            product = new ProductN();
        }
        return product;
    }
}
```

单例模式

```
/**
```

```
 * 饿汉式的单例模式
 * 在类加载的时候创建单例实例，而不是等到第一次请求实例的时候的时候创建
 * 1、私有 的无参数构造方法Singleton()，避免外部创建实例
 * 2、私有静态属性instance
 * 3、公有静态方法getInstance()
 */
public class Singleton {
    private static Singleton instance = new Singleton();
    private Singleton(){   }
    public static Singleton getInstance(){
        return instance;
    }
}
/**
 * 懒汉式的单例模式
 *在类加载的时候不创建单例实例，只有在第一次请求实例的时候的时候创建
*/
public class Singleton {
    private static Singleton instance;
    private Singleton(){   }
    /**
```

```
     * 多线程情况的单例模式，避免创建多个对象
    */
    public static Singleton getInstance(){
        if(instance ==null){//避免每次加锁，只有第一次没有创建对象时才加
锁
            synchronized(Singleton.class){//加锁，只允许一个线程进入
                if(instance==null){ //只创建一次对象
                    instance = new Singleton();
                }
            }
        }
        return instance;
    }
}
```

40. 请写出多线程代码使用 Thread 或者 Runnable ,并说出两种的区别。

方式 1：继承 Java.lang.Thread 类，并覆盖 run() 方法。优势：编写简单；劣势：无法继承其它父类

```
public class ThreadDemo1 {
    public static void main(String args[]) {
```

```
        MyThread1 t = new MyThread1();
        t.start();
        while (true) {
            System.out.println("兔子领先了，别骄傲");
        }
    }
}
class MyThread1 extends Thread {
    public void run() {
        while (true) {
            System.out.println("乌龟领先了，加油");
        }
    }
}
```

方式 2：实现 Java.lang.Runnable 接口，并实现 run()方法。优势：可继承其它类，多线程可共享同一个 Thread 对象；劣势：编程方式稍微复杂，如需访问当前线程，需调用 Thread.currentThread()方法

```
public class ThreadDemo2 {
    public static void main(String args[]) {
        MyThread2 mt = new MyThread2();
        Thread t = new Thread(mt);
```

```
            t.start();
            while (true) {
                System.out.println("兔子领先了，加油");
            }
        }
}
class MyThread2 implements Runnable {
    public void run() {
            while (true) {
                System.out.println("乌龟超过了，再接再厉");
            }
        }
}
```

41. 写一排序算法，输入 10 个数字，以逗号分开，可根据参数选择升序或者降序排序，须注明是何种排序算法。

```
package cn.bjsxt.demo;

import java.util.Scanner;

public class SortDemo {
    /**
```

```
     * 给定的字符串使用，号分隔
     * @param strNumber
     * @return
     */
    public static String [] split(String strNumber){
        String [] strSplit=strNumber.split(",");
        return strSplit;
    }
    /**
     * 将String类型的数组转换成int类型的数组
     * @param strSplit
     * @return
     */
    public static int [] getInt(String [] strSplit){
        int arr[]=new int[strSplit.length];
        for (int i = 0; i < strSplit.length; i++) {
            arr[i]=Integer.parseInt(strSplit[i]);
        }
        return arr;
    }
    /**
     * 冒泡排序
```

```
     * @param arr
     */
    public static void sort(int [] arr){
        for (int i = 0; i < arr.length-1; i++) {
            for (int j = 0; j < arr.length-1-i; j++) {
                if (arr[j]>arr[j+1]) {
                    change(arr,j,j+1);
                }
            }
        }
    }
    /**
     * 两数交换的方法
     * @param arr 数组
     * @param x 数组中元素的下标
     * @param y 数组中元素的下标
     */
    public static void change(int [] arr,int x,int y){
        int temp=arr[x];
        arr[x]=arr[y];
        arr[y]=temp;
    }
```

```
    /**
     * 测试类
     * @param args
     */
    public static void main(String[] args) {
        Scanner input=new Scanner(System.in);
        System.out.println("请输入一个数字串，每个数字以逗号分隔");
        String str=input.next();

        //调用方法
        String [] s=split(str);//使用逗号分隔
        int [] arr=getInt(s);//调有获得整型数组的方法
        sort(arr);//调用排序的方法
        for (int i : arr) {
            System.out.print(i+"\t");
        }
    }
}
```

42. 判断字符串是否是这样的组成的，第一个字母，后面可以是字母、数字、下划线、总长度为 5-20。

```
package cn.bjsxt.demo;
```

```
import java.util.Scanner;


public class StringDemo {
    public static void main(String[] args) {
        Scanner input=new Scanner(System.in);
        System.out.println("请输入一个字符串，第一个字符必须是字母：");
        String str=input.next();
        if (str.length()<5||str.length()>20) {
            System.out.println("对不起，字符串的长度必须在5-20之间!");
        }else{
            char []ch=str.toCharArray();
            if (Character.isLetter(ch[0])) {//判断第一个字符是否是字母
                for (int i = 1; i < ch.length; i++) {
                    if (!Character.isLetterOrDigit(ch[i])&&ch[i]!='_') {
                        System.out.println("字符串不符合要求");
                        break;
                    }
                }
            }
        }
    }
```

```
}
```

43. 在 Java 中，为什么基本类型不能做为 HashMap 的键值，而只能是引用类型，把引用类型做为 HashMap 的健值，需要注意哪些地方。

答:

在 Java 中是使用泛型来约束 HashMap 中的 key 和 value 的类型的，即 HashMap<K, V>；而泛型在 Java 的规定中必须是对象 Object 类型的，也就是说 HashMap<K, V>可以理解为 HashMap<Object, Object>，很显然基本数据类型不是 Object 类型的，因此不能作为键值，只能是引用类型。虽然我们在 HashMap 中可以这样添加数据：“map.put(1, “Java”)；”，但实际上是将其中的 key 值 1 进行了自动装箱操作，变为了 Integer 类型。

(2) 引用数据类型分为两类：系统提供的引用数据类型（如包装类、String 等）以及自定义引用数据类型。系统提供的引用数据类型中已经重写了 HashCode()和 equals()两个方法，所以能够保证 Map 中 key 值的唯一性；但是自定义的引用数据类型需要自己重写 HashCode()和 equals()这两个方法，以保证 Map 中 key 值的唯一性。

44. 在多线程编程里，wait 方法的调用方式是怎样的？

答:

wait 方法是线程通信的方法之一，必须用在 synchronized 方法或者 synchronized 代码块中，否则会抛出异常，这就涉及到一个“锁”的概念，

而 wait 方法必须使用上锁的对象来调用，从而持有该对象的锁进入线程等待状态，直到使用该上锁的对象调用 notify 或者 notifyAll 方法来唤醒之前进入等待的线程，以释放持有的锁。

45. Java 出现 OutOf MemoryError（OOM 错误)的原因有哪些？出现 OOM 错误后，怎么解决？

答:

OutOf MemoryError 这种错误可以细分为多种不同的错误，每种错误都有自身的原因和解决办法，如下所示：

java.lang.OutOfMemoryError: Java heap space

错误原因：此 OOM 是由于 JVM 中 heap 的最大值不满足需要。

解决方法：1）调高 heap 的最大值，即-Xmx 的值调大。2）如果你的程序存在内存泄漏，一味的增加 heap 空间也只是推迟该错误出现的时间而已，所以要检查程序是否存在内存泄漏。

java.lang.OutOfMemoryError: GC overhead limit exceeded

错误原因：此 OOM 是由于 JVM 在 GC 时，对象过多，导致内存溢出，建议调整 GC 的策略，在一定比例下开始 GC 而不要使用默认的策略，或者将新代和老代设置合适的大小，需要进行微调存活率。

解决方法：改变 GC 策略，在老代 80%时就是开始 GC，并且将 -XX:SurvivorRatio （ -XX:SurvivorRatio=8 ） 和 -XX:NewRatio （-XX:NewRatio=4）设置的更合理。

java.lang.OutOfMemoryError: Java perm space

错误原因：此 OOM 是由于 JVM 中 perm 的最大值不满足需要。

解决方法：调高 heap 的最大值，即-XX:MaxPermSize 的值调大。

另外，注意一点，Perm 一般是在 JVM 启动时加载类进来，如果是 JVM 运行较长一段时间而不是刚启动后溢出的话，很有可能是由于运行时有类被动态加载进来，此时建议用 CMS 策略中的类卸载配置。如：-XX:+UseConcMarkSweepGC -XX:+CMSClassUnloadingEnabled。

java.lang.OutOfMemoryError: unable to create new native thread

错误原因：当 JVM 向 OS 请求创建一个新线程时，而 OS 却由于内存不足无法创建新的 native 线程。

解决方法：如果 JVM 内存调的过大或者可利用率小于 20%，可以建议将 heap 及 perm 的最大值下调，并将线程栈调小，即-Xss 调小，如：-Xss128k。

java.lang.OutOfMemoryError: Requested array size exceeds VM limit

错误原因：此类信息表明应用程序（或者被应用程序调用的 APIs）试图分配一个大于堆大小的数组。例如，如果应用程序 new 一个数组对象，大小为 512M，但是最大堆大小为 256M，因此 OutOfMemoryError 会抛出，因为数组的大小超过虚拟机的限制。

解决方法：1) 首先检查 heap 的-Xmx 是不是设置的过小。2) 如果 heap 的-Xmx 已经足够大，那么请检查应用程序是不是存在 bug，例如：应用程序可能在计算数组的大小时，存在算法错误，导致数组的 size 很大，从而导致巨大的数组被分配。

java.lang.OutOfMemoryError: request <size> bytes for <reason>. Out of swap space

错误原因：抛出这类错误，是由于从 native 堆中分配内存失败，并且堆内

存可能接近耗尽。这类错误可能跟应用程序没有关系，例如下面两种原因也会导致错误的发生：1) 操作系统配置了较小的交换区。2)系统的另外一个进程正在消耗所有的内存。

解决办法：1) 检查 os 的 swap 是不是没有设置或者设置的过小。2) 检查是否有其他进程在消耗大量的内存，从而导致当前的 JVM 内存不够分配。

注意：虽然有时<reason>部分显示导致 OOM 的原因，但大多数情况下，<reason>显示的是提示分配失败的源模块的名称，所以有必要查看日志文件，如 crash 时的 hs 文件。

46. 简述 Java 中如何实现多态

答:

实现多态有三个前提条件：

1、 继承的存在；(继承是多态的基础，没有继承就没有多态)。

2、子类重写父类的方法。(多态下会调用子类重写后的方法)。

3、父类引用变量指向子类对象。(涉及子类到父类的类型转换)。

最后使用父类的引用变量调用子类重写的方法即可实现多态。

47. 简述 Tomcat，Apache，JBoss 的区别

Apache：是 C 语言实现的，专门用来提供 HTTP 服务。支持静态页（HTML），不支持动态请求如：CGI、Servlet/JSP、PHP、ASP 等。

Tomcat :是 Java 开发的一个符合 JavaEE 的 Servlet 规范的 JSP 服务器(Servlet 容器)，是 Apache 的扩展。用于解析 jsp，servlet 的 Servlet 容器，是高效，轻量级的容器，但是不支持 EJB，只能用于 java 应用。

Jboss：应用服务器，运行 EJB 的 J2EE 应用服务器，遵循 J2EE 规范，能够提供更多平

台的支持和更多集成功能，如数据库连接，JCA 等；其对 servlet 的支持是通过集成其他 servlet 容器来实现的，如 tomcat 和 jetty。

48. 已排好序的数组 A，一般来说可用二分查找可以很快找到，现有一特殊数组 A，它是循环递增的，如 a[]={17, 19 ,20, 25, 1, 4, 7, 9}，在这样的数组中找一元素，看看是否存在。请写出你的算法，必要时可写伪代码，并分析其空间，时间复杂度。

思路说明：循环递增数组有这么一个性质：以数组中间元素将循环递增数组划分为两部分，则一部分为一个严格递增数组，而另一部分为一个更小的循环递增数组。当中间元素大于首元素时，前半部分为严格递增数组，后半部分为循环递增数组；当中间元素小于首元素时，前半部分为循环递增数组；后半部分为严格递增数组。

记要检索的元素为 e ,数组的首元素为 a[low] ,中间元素为 a[mid] ,末尾元素为 a[high]。

则当 e 等于 a[mid] 时，直接返回 mid 的值即可；当 e 不等于 a[mid] 时:

1) a[mid] > a[low]，即数组前半部分为严格递增数组，后半部分为循环递增数组时，若 key 小于 a[mid]并且不小于 a[low]时，则 key 落在数组前半部分；否则，key 落在数组后半部分。

2) a[mid] < a[high]，即数组前半部分为循环递增数组，后半部分为严格递增数组时，若 key 大于 a[mid]并且不大于 a[high]时，则 key 落在数组后半部分；否则，key 落在数组前半部分。

这种方式的时间复杂度为：O(log(n))，空间复杂度为 O(1)。

```
public class TestBinarySearch {

    public static void main(String[] args) {

        // 定义数组
```

```
        int[] a = { 17, 19, 20, 21, 25, 1, 4, 7 };
        // 调用改进后的二分查找法求索引
        int pos = search(a, 7);
        System.out.println("要查找的元素的索引为：" + pos);
    }

    /** 改进后的二分查找法：e 为要查找的元素 */
    public static int search(int[] a, int e) {
        int low = 0;
        int high = a.length - 1;
        int mid = 0;
        int pos = -1; // 返回-1，表示查找失败
        // 如果 low < high，说明循环查找结束，直接返回-1;否则循环查找
        while (low <= high) {
            // mid 为中间值索引
            mid = (low + high) / 2;
            // 如果中间值刚好是 e，则查找成功，终止查找，e 的索引为 mid
            if (a[mid] == e) {
                pos = mid;
                break;
            }
            // 如果 a[low] <= a[mid]，说明原数组的前半部分是严格递增的，后半部
```

```
分是一个更小的循环递增数组
                if (a[low] <= a[mid]) {
                    // 如果要查找的元素 e 小于 a[mid]并且不小于 a[low]时，则说明 e 落
在数组前半部分
                    if (a[low] <= e && e < a[mid]) {
                        high = mid - 1;
                    } else {// 否则的话，需要在数组的后半部分继续查找
                        low = mid + 1;
                    }
                } else {// 否则，后半部分是严格递增的，前半部分是一个更小的循环递增
数组
                    // 如果要查找的元素 e 大于 a[mid]并且不大于 a[high]时，则说明 e
落在数组后半部分
                    if (a[mid] < e && e <= a[high]) {
                        low = mid + 1;
                    } else {// 否则的话，需要在数组的前半部分继续查找
                        high = mid - 1;
                    }
                }
            }
            return pos;
        }
}
```

49. 请编写一个完整的程序，实现如下功能：从键盘输入数字 n，程序自动计算 n!并输出。（注 1：n！=1*2*3...*n，注 2：请使用递归实现）

思路说明：因为 n! = (n-1)! * n，所以要求 n!首先要求出(n-1)!，而(n-1)! = (n-1-1)! * (n-1)，以此类推，直到 n = 1 为止。

```
import java.util.Scanner;
public class TestFactorial {
    public static void main(String[] args) {
        System.out.print("请输入一个整数：");
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        System.out.println(n + "的阶乘是：" + factorial(n));
    }
    /**求阶乘的方法*/
    public static int factorial(int n) {
        if(n == 1){
            return 1;
        }
        return factorial(n - 1) * n;
    }
}
```

50. 新建一个流对象，下面那个选项的代码是错误的？（B）

A.	new BufferedWriter(new FileWriter(“a.txt”));
B.	new BufferedReader (new FileInputStream(“a.dat”));
C.	new GZIPOutputStream(new FileOutputStream(“a.zip”));
D.	new ObjectInputStream(new FileInputStream(“a.dat”));
分析： BufferedReader 类的参数只能是 Reader 类型的 ,不能是 InputStream 类型。	

51. 以下对继承的描述锚误的是（A)

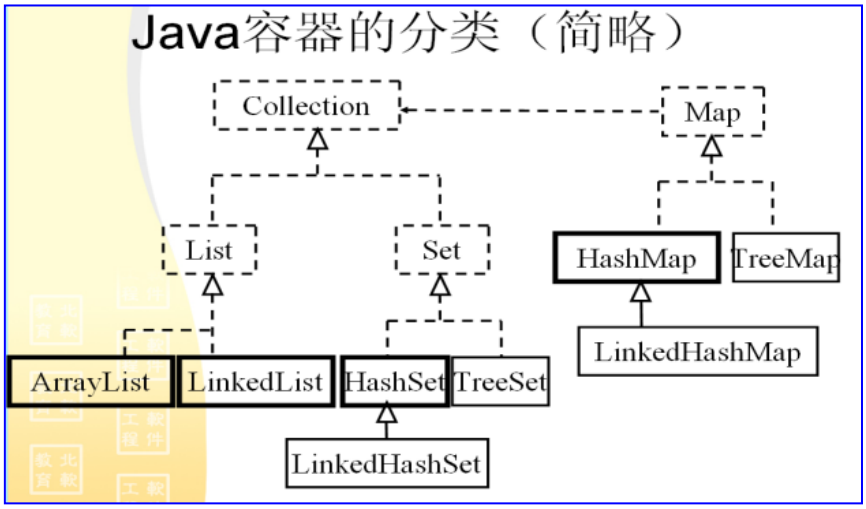
A.	Java 中的继承允许一个子类继承多个父类
B.	父类更具有通用性，子类更具体
C.	Java 中的继承存在着传递性
D.	当实例化子类时会递归调用父类中的构造方法
分析： Java 是单继承的，一个类只能继承一个父类。	

52. 列出 Java 中的集合类层次结构？

答:

Java 中集合主要分为两种：Collection 和 Map。Collection 是 List 和 Set 接口的父接口；ArrayList 和 LinkedList 是 List 的实现类；HashSet 和 TreeSet 是 Set 的实现类；LinkedHashSet 是 HashSet 的子类。HashMap 和 TreeMap 是 Map 的实现类；LinkedHashMap 是 HashMap 的子类。

图中：虚线框中为接口，实线框中为类。



53. Java 中 Math.random（）/Math.random（）值为？

答:

如果除数与被除数均不为 0.0 的话，则取值范围为[0，+∞]。+∞在 Java 中显示的结果为 Infinity。

如果除数与被除数均为 0.0 的话，则运行结果为 NaN（Not a Number 的简写），计算错误。

54. Java 中，如果 Manager 是 Employee 的子类，那么 Pair<Manager>是 Pair<Employee>的子类吗？

答:

不是，两者没有任何关联。

55. 请用递归的方法计算斐波那契数列的同项 F（n），已知 F0=0,F1=1,F(n)=F(n-1)+F(n-2)(n>=2,n∈N*).

思路说明：斐波那契数列的排列是：0，1，1，2，3，5，8，13，21，34，55，89，144……，特别指出的是 0 不是第一项而是第 0 项；因为 F(n)=F(n-1)+F(n-2)，所以要求 F(n)首先要求出 F(n-1)和 F(n-2)，而

F(n-1)=F(n-1-1)+F(n-1-2)，以此类推，直到,F(2)=F(1)+F(0)为止，已知 F(1) = 1，F(0) = 0。

```
import java.util.Scanner;
public class TestFibo {
    public static void main(String[] args) {
        System.out.print("请输要求斐波那契数列的第几项：");
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        System.out.println("斐波那契数列的第"+ n + "是：" + fibo(n));
    }
    public static int fibo(int n) {
        if(n == 0){
            return 0;
        } else if(n == 1){
            return 1;
        }
        return fibo(n -1) + fibo(n - 2);
    }
}
```

56. 现在有整数数组{11,66,22,0,55,32} ,请任意选择一种排序算法，用 Java 程序实现

冒泡思路说明：

(1) 最开始将数组看做一个无序数列(个数是数组的长度)与一个有序数列(0个)的组合；

(2) 每一趟比较完后，找到了无序数列的最大值，将其放到有序数列中(有序数列个数+1)；

(3) N 个数，比较 N-1 趟；

(4) 每一趟挨个进行比较：从数组的第一个元素开始，到无序数列的最后一个为止；

(5) 如果前边一个大于后边一个，那么交换位置；

(6) 每趟比较的次数与趟数有关；

(7) 根据每趟比较是否发生了交换判断数据是否已经有序，从而进行优化。

```
public class TestSort {
    public static void main(String[] args) {
        int[] arr = {11, 66, 22, 0, 55, 32};
        // 调用排序方法
        sort(arr);
        // 输出排除后的数组
        for (int num : arr) {
            System.out.print(num + "\t");
        }
```

```
    }

    public static void sort(int[] arr) {
        // 定义标记
        boolean flag = false;
        int temp;
        // 排序
        // 外层循环控制的是比较的趟数
        for (int i = 0; i < arr.length - 1; i++) {
            // 每一趟比较之前初始化，否则会保留上一堂比较的结果
            flag = false;
            // 内层循环控制的是每趟比较的次数
            for (int j = 0; j < arr.length - 1 - i; j++) {
                // 挨个进行比较：从数组的第一个元素开始，到无序数列的
最后一个
                if (arr[j] > arr[j + 1]) {
                    // 交换
                    temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                    //如果发生交换，改变 flag 的值
                    flag = true;
```

```
                }
            }
            if (!flag) {
                break;
            }
        }
    }
}
```

57. 请根据注释，编码实现下面类的方法

```
// 这个类用于存取一组权限，每个权限用非负整数表示的.这组枳限存储在
// righiString 属性中。如果权限 N 权限存在，rightString 第 N 个字符为“1”，
否则， 为空格。
class RightStore {
    public String righString = "";

    // 如果传入的权限 right 存在，该方法返回 true.否期，为 false.,
    // right 为传入的权限的整数值.
    public boolean getRight(int right) {

        return true;
    }
```

```
    // 该方法存储或消除传入的权限.如果 value 为 true,存储传入的权限,
    // 否则清除该权限.
    // right 为传入的权限的整数值.
    public void setRight(int right, boolean value) {
    }
}
```

思路说明：我们首先要读懂这道题的意思：righString 这个字符串是用来存储一系列权限的，并且权限的取值只有两种：有和没有；在 righString 中使用字符‘1’表示有权限，字符空格‘ ’表示没有权限。举个例子：如果 righString 的长度为 3，第一位表示对订单系统是否有权限，第二位表示对人员管理系统是否有权限，第三位表示对库存系统是否有权限。而方法中的 int right 参数则表示的是字符串的第几位。

上边这些搞明白之后，方法的编写就简单多了。

```
public class RightStore {
    public String righString = "";

    public boolean getRight(int right) {
        //先求得第 right 个字符
        char ch = righString.charAt(right - 1);
        //如果 ch 为'1'，返回 true，否则返回 false
        return ch == '1';
```

```
    }


    public void setRight(int right, boolean value) {
        //如果 value 为 true,存储传入的权限，否则消除权限（改为空格）
        righString.replace(righString.charAt(right - 1), value ? '1' : ' ');
    }
}
```

58. 接口和抽象类的区别

答:

相同点：

(1) 抽象类和接口均包含抽象方法，类必须实现所有的抽象方法，否则是抽象类

(2) 抽象类和接口都不能实例化，他们位于继承树的顶端，用来被其他类继承和实现

59. 两者的区别主要体现在两方面：语法方面和设计理念方面

(1) 语法方面的区别是比较低层次的，非本质的，主要表现在：

1、 接口中只能定义全局静态常量，不能定义变量。抽象类中可以定义常量和变量。

2、 接口中所有的方法都是全局抽象方法。抽象类中可以有 0 个、1 个或多个，甚至全部都是抽象方法。

3、 抽象类中可以有构造方法，但不能用来实例化，而在子类实例化是执行，

完成属于抽象类的初始化操作。接口中不能定义构造方法。

4、 一个类只能有一个直接父类（可以是抽象类），但可以充实实现多个接口。一个类使用 extends 来继承抽象类，使用 implements 来实现接口。

(2) 二者的主要区别还在设计理念上，其决定了某些情况下到底使用抽象类还是接口。

1、 抽象类体现了一种继承关系，目的是复用代码，抽象类中定义了各个子类的相同代码，可以认为父类是一个实现了部分功能的“中间产品”，而子类是“最终产品”。父类和子类之间必须存在“is-a”的关系，即父类和子类在概念本质上应该是相同的。

2、 接口并不要求实现类和接口在概念本质上一致的，仅仅是实现了接口定义的约定或者能力而已。接口定义了“做什么”，而实现类负责完成“怎么做”，体现了功能（规范）和实现分离的原则。接口和实现之间可以认为是一种“has-a 的关系”。

60. 同步代码块和同步方法有什么区别

答:

相同点：

同步方法就是在方法前加关键字 synchronized，然后被同步的方法一次只能有一个线程进入，其他线程等待。而同步代码块则是在方法内部使用大括号使得一个代码块得到同步。同步代码块会有一个同步的“目标”，使得同步块更加灵活一些（同步代码块可以通过“目标”决定需要锁定的对象）。一般情况下，如果此“目标”为 this，那么同步方法和同步代码块没有太大的区别。

区别：

同步方法直接在方法上加 synchronized 实现加锁，同步代码块则在方法内部加锁。很明显，同步方法锁的范围比较大，而同步代码块范围要小点。一般同步的范围越大，性能就越差。所以一般需要加锁进行同步的时候，范围越小越好，这样性能更好。

61. 静态内部类和内部类有什么区别

答:

静态内部类不需要有指向外部类的引用。但非静态内部类需要持有对外部类的引用。

静态内部类可以有静态成员(方法，属性)，而非静态内部类则不能有静态成员(方法，属性)。

非静态内部类能够访问外部类的静态和非静态成员。静态内部类不能访问外部类的非静态成员，只能访问外部类的静态成员。

实例化方式不同：

1) 静态内部类：不依赖于外部类的实例，直接实例化内部类对象

2) 非静态内部类：通过外部类的对象实例生成内部类对象

62. 反射的概念与作用

答:

反射的概念：

反射是指一类应用，它们能够自描述和自控制。也就是说，这类应用通过采用某种机制来实现对自己行为的描述（ self-representation ）和监测

（examination），并能根据自身行为的状态和结果，调整或修改应用所描述行为的状态和相关的语义。

反射机制是 Java 动态性的重要体现。我们可以通过反射机制在运行时加载编译期完全未知的类。通过反射机制，我们可以在运行时加载需要的类，从而动态的改变了程序结构，使我们的程序更加灵活、更加开放。

反射的作用：

通过反射可以使程序代码访问装载到 JVM 中的类的内部信息

1) 获取已装载类的属性信息

2) 获取已装载类的方法

3) 获取已装载类的构造方法信息

63. 提供 Java 存取数据库能力的包是（A）

A	java.sql
B.	java.awt
C.	java.lang
D.	java.swing
分析： java.awt 和 javax.swing 两个包是图形用户界面编程所需要的包； java.lang 包则提供了 Java 编程中用到的基础类。	

64. 下列运算符合法的是（AD）(多选）

A	&&
B.	<>
C.	if

D.	=

分析：

&&是逻辑运算符中的短路与；

<>表示不等于，但是 Java 中不能这么使用，应该是!=；

if 不是运算符；

=是赋值运算符。

65. 执行如下程序代码，c 的值打印出来是（C）

```
public class Test1 {
        public static void main(String[] args) {
            int a = 0;
            int c = 0;
            do{
                --c;
                a = a - 1;
            } while (a > 0);
            System.out.println(c);
        }
}
```

A	0
B.	1
C.	-1

D.	死循环
分析： do-while 循环的特点是先执行后判断，所以代码先执行--c 操作，得到 c 为-1，之后执行 a=a-1 的操作，得到 a 为-1，然后判断 a 是否大于 0，判断条件不成立，退出循环，输出 c 为-1。	

66. 下列哪一种叙述是正确的（D）

A	abstract 修饰符可修饰字段，方法和类
B.	抽象方法的 body 部分必须用一对大括号{}包住
C.	声明抽象方法，大括号可有可无
D.	声明抽象方法不可写出大括号
分析： abstract 只能修饰方法和类，不能修饰字段； 抽象方法不能有方法体，即没有{}； 同 B。	

67. 下列语句正确的是（A）

A	形式参数可被视为 local Variable
B.	形式参数可被所有的字段修饰符修饰
C.	形式参数为方法被调用时，真正被传递的参数
D.	形式参数不可以是对象
分析： local Variable 为局部变量 ,形参和局部变量一样都只有在方法内才会发生作用，也只能在方法中使用，不会在方法外可见；	

对于形式参数只能用 final 修饰符，其它任何修饰符都会引起编译器错误； 真正被传递的参数是实参； 形式参数可是基本数据类型也可以是引用类型（对象）。

68. 下列哪种说法是正确的（D）

A	实例方法可直接调用超类的实例方法
B.	实例方法可直接调用超类的类方法
C.	实例方法可直接调用其他类的实例方法
D.	实例方法可直接调用本类的类方法
分析： 实例方法不可直接调用超类的私有实例方法； 实例方法不可直接调用超类的私有的类方法； 要看访问权限。	

69. Java 程序的种类有（BCD）(多选)

A	类 (Class)
B.	Applet
C.	Application
D.	Servlet
分析： 是 Java 中的类，不是程序； 内嵌于 Web 文件中，由浏览器来观看的 Applet； 可独立运行的 Application； 服务器端的 Servlet。	

70. 下列说法正确的有（BCD）(多选)

A	环境变量可在编译 source code 时指定
B.	在编译程序时，所指定的环境变置不包括 class path
C.	javac 一次可同时编译数个 Java 源文件
D.	javac.exe 能指定编译结果要置于哪个目录（directory)
分析： 环境变量一般都是先配置好再编译源文件。	

71. 下列标识符不合法的有（ACD）(多选)

A	new
B.	$Usdollars
C.	1234
D.	car.taxi
分析： new 是 Java 的关键字； C. 数字不能开头； D. 不能有“.”。	

72. 下列说法错误的有（BCD）(多选)

A	数组是一种对象
B.	数组属于一种原生类
C.	int number[]=(31,23,33,43,35,63)
D.	数组的大小可以任意改变
分析：	

B. Java 中的原生类（即基本数据类型）有 8 种，但不包括数组；

C. 语法错误，应该“{·}”，而不是“(·)”；

D. 数组的长度一旦确定就不能修改。

73. 不能用来修饰 interface 的有（ACD）(多选)

A	private
B.	public
C.	protected
D.	static

分析：

能够修饰 interface 的只有 public、abstract 以及默认的三种修饰符。

74. 下列正确的有（ACD）(多选)

A	call by value 不会改变实际参数的数值
B.	call by reference 能改变实际参数的参考地址
C.	call by reference 不能改变实际参数的参考地址
D.	call by reference 能改变实际参数的内容

分析：

Java 中参数的传递有两种，一种是按值传递（call by value：传递的是具体的值，如基础数据类型），另一种是按引用传递（call by reference：传递的是对象的引用，即对象的存储地址）。前者不能改变实参的数值，后者虽然不能改变实参的参考地址，但可以通过该地址访问地址中的内容从而实现内容的改变。

75. 下列说法错误的有（ACD）(多选)

A	在类方法中可用 this 来调用本类的类办法
B.	在类方法中调用本类的类方法时可以直接调用
C.	在类方法中只能调用本类中的类方法
D.	在类方法中绝对不能调用实例方法
分析： 类方法是在类加载时被加载到方法区存储的，此时还没有创建对象，所以不能使用 this 或者 super 关键字； C. 在类方法中还可以调用其他类的类方法； D. 在类方法可以通过创建对象来调用实例方法。	

76. 下列说法错误的有（ABC）(多选)

A	Java 面向对象语言容许单独的过栈与函数存在
B.	Java 面向对象语言容许单独的方法存在
C.	Java 语言中的方法属于类中的成员（member）
D.	Java 语言中的方法必定隶属于某一类（对象），调用方法与过程或函数相同
分析： B. Java 不允许单独的方法，过程或函数存在，需要隶属于某一类中； C. 静态方法属于类的成员，非静态方法属于对象的成员。	

77. 下列说法错误的有（BCD）(多选)

A	能被 java.exe 成功运行的 java class 文件必须有 main()方法
B.	J2SDK 就是 Java API

C.	Appletviewer.exe 可利用 jar 选项运行.jar 文件
D.	能被 Appletviewer 成功运行的 java class 文件必须有 main()方法
分析： B. J2SDK 是 sun 公司编程工具，API 是指的应用程序编程接口； C. Appletviewer.exe 就是用来解释执行 java applet 应用程序的，一种执行 HTML 文件上的 Java 小程序类的 Java 浏览器； D. 能被 Appletviewer 成功运行的 java class 文件可以没有 main（）方法。	

78. Java 线程的几种状态

答:

线程是一个动态执行的过程，它有一个从产生到死亡的过程，共五种状态：

新建（new Thread）

当创建 Thread 类的一个实例（对象）时，此线程进入新建状态（未被启动）。

例如：Thread t1=new Thread();

就绪（runnable）

线程已经被启动，正在等待被分配给 CPU 时间片，也就是说此时线程正在就绪队列中排队等候得到 CPU 资源。例如：t1.start();

运行（running）

线程获得 CPU 资源正在执行任务（run()方法），此时除非此线程自动放弃 CPU 资源或者有优先级更高的线程进入，线程将一直运行到结束。

死亡（dead）

当线程执行完毕或被其它线程杀死，线程就进入死亡状态，这时线程不可能

再进入就绪状态等待执行。

自然终止：正常运行 run()方法后终止

异常终止：调用 stop()方法让一个线程终止运行

堵塞（blocked）

由于某种原因导致正在运行的线程让出 CPU 并暂停自己的执行，即进入堵塞状态。

正在睡眠：用 sleep(long t) 方法可使线程进入睡眠方式。一个睡眠着的线程在指定的时间过去可进入就绪状态。

正在等待：调用 wait()方法。（调用 motify()方法回到就绪状态）

被另一个线程所阻塞：调用 suspend()方法。（调用 resume()方法恢复）

79. List，Set，Map 各有什么特点

答:

List 接口存储一组不唯一，有序（插入顺序）的对象。

Set 接口存储一组唯一，无序的对象。

Map 接口存储一组键值对象，提供 key 到 value 的映射。key 无序，唯一。value 不要求有序，允许重复。（如果只使用 key 存储，而不使用 value，那就是 Set）。

80. 常见的运行时异常

答:

ClassCastException(类转换异常)

IndexOutOfBoundsException(下标越界异常)

NullPointerException(空指针异常)

ArrayStoreException(数据存储异常，操作数组时类型不一致)

BufferOverflowException(IO 操作时出现的缓冲区上溢异常)

InputMismatchException(输入类型不匹配异常)

ArithmeticException(算术异常)

81. 实现 String 类的 replaceAll 方法

思路说明：replaceAll 方法的本质是使用正则表达式进行匹配，最终调用的其实是 Matcher 对象的 replaceAll 方法。

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;


public class TestStringReplaceAll {
    public static void main(String[] args) {
        String str = "a1s2d3f4h5j6k7";
        // 将字符串中的数字全部替换为 0
        System.out.println(replaceAll(str, "\\d", "0"));
    }


    /**
     * @param str:源字符串
     * @param regex:正则表达式
     * @param newStr:替换后的子字符串
     * @return 返回替换成功后的字符串
```

```
     */
    public static String replaceAll(String str, String regex, String newStr)
{
        Pattern pattern = Pattern.compile(regex);
        Matcher mathcer = pattern.matcher(str);
        String reslut = mathcer.replaceAll(newStr);
        return reslut;
    }
}
```

82. 二分法查询（递归实现）

思路说明：假设在一个已经排好序的有序序列(N 个元素，升序排列)，首先让序列中的中间的元素与需要查找的关键字进行比较，如果相等，则查找成功，否则利用中间位置将序列分成两个子序列，如果待查找的关键字小于中间的元素，则在前一个子序列中同样的方法进一步查找，如果待查找的关键字大于中间的元素，则在后一个子序列中同样的方法进一步查找，重复以上过程一直到查找结束！

```
import java.util.Scanner;
public class TestBinarySearchRecursion {
    public static void main(String[] args) {
        int[] a = { 1, 3, 5, 7, 9, 11, 13 };
        System.out.print("请输入要查找的元素：");
```

```
        int e = new Scanner(System.in).nextInt();
        int index = binarySearch(a, 0, a.length - 1, e);
        System.out.println(index != -1 ? "元素索引为" + index : "没有该元
素");
    }

    private static int binarySearch(int[] a, int low, int high, int e) {
        int mid = 0;
        if (low <= high) {
            mid = (low + high) / 2;
            if (a[mid] == e) {
                return mid;
            } else if (a[mid] > e) {
                return binarySearch(a, low, mid - 1, e);
            } else {
                return binarySearch(a, mid + 1, high, e);
            }
        }
        return -1;
    }
}
```

83. 编写一段 Java 程序，把一句英语中的每个单词中的字母次序倒转，单词次序保持不变，例入输入为“There is a dog.”,输出结果应该是“erehT si a god.”要求不使用 Java 的库函数，例如 String 类的 split，reverse 方法。

函数形如：

```
public static String reverseWords(String input) {
    String str = "";


    return str;
}
```

思路说明：将字符串转化成字符数组，然后根据数组中空格的位置判断每个单词所占的索引范围，根据得到的索引将数组中的每个单词逆序后拼接到新的字符串中。

```
public class TestStringReverse{
    public static void main(String[] args) {
        String input = "There is a dog";
        System.out.println("逆转后的字符串为：" + reverseWords(input));
    }
    public static String reverseWords(String input) {
        String str = "";
```

```
        //将字符串转化成字符数组
        char[] arr = input.toCharArray();
        //index 用来记录每个单词的起始索引
        int index = 0;
        //遍历字符数组，将空格前边的单词挨个拼接到 str 中
        for (int i = 0; i < arr.length; i++) {
            if(arr[i] == ' '){
                //根据空格的位置将空格前边一个单词密续追加到 str 中
                for(int j = i - 1; j >= index; j--){
                    str += arr[j];
                }
                //单词拼接完成后，拼接一个空格
                str += ' ';
                //让 index 指向下一个单词的起始位置
                index = i + 1;
            }
        }
        //将最后一个单词拼接上
        for(int i = arr.length - 1; i >= index; i--){
            str += arr[i];
        }
        return str;
```

```
    }
}
```

84. 说说 JVM 原理？内存泄漏与溢出的区别？何时产生内存泄漏？

答:

JVM 原理：

JVM 是 Java Virtual Machine（Java 虚拟机）的缩写，它是整个 java 实现跨平台的最核心的部分，所有的 Java 程序会首先被编译为.class 的类文件，这种类文件可以在虚拟机上执行，也就是说 class 并不直接与机器的操作系统相对应，而是经过虚拟机间接与操作系统交互，由虚拟机将程序解释给本地系统执行。JVM 是 Java 平台的基础，和实际的机器一样，它也有自己的指令集，并且在运行时操作不同的内存区域。JVM 通过抽象操作系统和 CPU 结构，提供了一种与平台无关的代码执行方法，即与特殊的实现方法、主机硬件、主机操作系统无关。JVM 的主要工作是解释自己的指令集（即字节码）到 CPU 的指令集或对应的系统调用，保护用户免被恶意程序骚扰。JVM 对上层的 Java 源文件是不关心的，它关注的只是由源文件生成的类文件（.class 文件）。

内存泄漏与溢出的区别：

1) 内存泄漏是指分配出去的内存无法回收了。

2) 内存溢出是指程序要求的内存，超出了系统所能分配的范围，从而发生

溢出。比如用 byte 类型的变量存储 10000 这个数据，就属于内存溢出。

3) 内存溢出是提供的内存不够；内存泄漏是无法再提供内存资源。

何时产生内存泄漏：

1) 静态集合类：在使用 Set、Vector、HashMap 等集合类的时候需要特别注意，有可能会发生内存泄漏。当这些集合被定义成静态的时候，由于它们的生命周期跟应用程序一样长，这时候，就有可能会发生内存泄漏。

2) 监听器：在 Java 中，我们经常会使用到监听器，如对某个控件添加单击监听器 addOnClickListener()，但往往释放对象的时候会忘记删除监听器，这就有可能造成内存泄漏。好的方法就是，在释放对象的时候，应该记住释放所有监听器，这就能避免了因为监听器而导致的内存泄漏。

3) 各种连接：Java 中的连接包括数据库连接、网络连接和 io 连接，如果没有显式调用其 close()方法，是不会自动关闭的，这些连接就不能被 GC 回收而导致内存泄漏。一般情况下，在 try 代码块里创建连接，在 finally 里释放连接，就能够避免此类内存泄漏。

4) 外部模块的引用：调用外部模块的时候，也应该注意防止内存泄漏。如模块 A 调用了外部模块 B 的一个方法 ,如 :public void register(Object o)。这个方法有可能就使得 A 模块持有传入对象的引用 ,这时候需要查看 B 模块是否提供了去除引用的方法，如 unregister()。这种情况容易忽略，而且发生了内存泄漏的话，比较难察觉，应该在编写代码过程中就应该注意此类问题。

5) 单例模式：使用单例模式的时候也有可能导致内存泄漏。因为单例对象

初始化后将在 JVM 的整个生命周期内存在，如果它持有一个外部对象（生命周期比较短）的引用，那么这个外部对象就不能被回收，而导致内存泄漏。如果这个外部对象还持有其它对象的引用，那么内存泄漏会更严重，因此需要特别注意此类情况。这种情况就需要考虑下单例模式的设计会不会有问题，应该怎样保证不会产生内存泄漏问题。

85. 下面关于 Java.lang.Exception 类的说法正确的是（A）

A	继承自 Throwable
B.	不支持 Serializable
C.	继承自 AbstractSet
D.	继承自 FiteInputStream
分析： Throwable 是 Exception 和 Error 的父类，Exception 虽然没有实现 Serializable 接口，但是其父类 Throwable 已经实现了该接口，因此 Exception 也支持 Serializable。	

86. 下列说法正确的是 （C）

A	LinkedList 继承自 List
B.	AbstractSet 继承自 Set
C.	HashSet 继承自 AbstractSet
D.	WeakHashMap 继承自 HashMap
分析： LinkedList 继承自 AbstractSequentialList； AbstractSet 继承自 AbstractCollection；	

HashSet 确实是继承自 AbstractSet； WeakHashMap 继承自 AbstractMap。

87. 请问 0.6332 的数据类型是（B）

A	float
B.	double
C.	Float
D.	Double
分析： 小数默认是双精度浮点型即 double 类型的。	

88. 下面哪个流是面向字符的输入流(D)

A	BufferedWriter
B.	FileInputStream
C.	ObjectInputStream
D.	InputStreamReader
分析： 以 InputStream（输入流）/OutputStream（输出流）为后缀的是字节流； 以 Reader（输入流）/Writer（输出流）为后缀的是字符流。	

89. Java 接口的修饰符可以为（D）

A	private
B.	protected
C.	final
D.	abstract

分析： 能够修饰 interface 的只有 public、abstract 以及默认的三种修饰符。

90. 不通过构造函数也能创建对象么（A）

A	是
B.	否
分析： Java 创建对象的几种方式： (1) 用 new 语句创建对象，这是最常见的创建对象的方法。 (2) 运用反射手段，调用 java.lang.Class 或者 java.lang.reflect.Constructor 类的 newInstance()实例方法。 (3) 调用对象的 clone()方法。 (4) 运用反序列化手段，调用 java.io.ObjectInputStream 对象的 readObject()方法。 (1)和(2)都会明确的显式的调用构造函数；(3)是在内存上对已有对象的影印，所以不会调用构造函数；(4)是从文件中还原类的对象，也不会调用构造函数。	

91. ArrayList list=new ArrayList(20);中的 list 扩充几次（A）

A	0
B.	1
C.	2
D.	3
分析：已经指定了长度，所以不扩容	

92. 在 Java 多线程中，请用下面哪种方式不会使线程进入阻塞状态（D）

A	sleep()
B.	suspend()
C.	wait()
D.	yield()
分析： yield 会是线程进入就绪状态	

93. Java 类库中，将信息写入内存的类是（B）

A	Java.io.FileOutputStream
B.	java.ByteArrayOutputStream
C.	java.io.BufferedOutputStream
D.	java,.io.DataOutputStream

94. 当使用 RMI 技术实现远程方法调用时，能为远程对象生成 Sub 和 Skeleton 命令的是（A）

A	Mic
B.	mid
C.	mitegistry
D.	policytool

95. List、Set、Map 哪个继承自 Collection 接口，一下说法正确的是（C）

A	List Map

B.	Set Map
C.	List Set
D.	List Map Set
分析：Map 接口继承了 java.lang.Object 类,但没有实现任何接口.	

96. 在“=”后填写适当的内容：

String []a=new String[10];

则：a[0]~a[9]=null;

a.length=10;

如果是 int[]a=new int[10];

则： a[0]~a[9]= (0)

a.length= (10)

97. GC 线程是否为守护线程？

GC 线程是守护线程。线程分为守护线程和非守护线程（即用户线程）。只要当前 JVM 实例中尚存在任何一个非守护线程没有结束，守护线程就全部工作；只有当最后一个非守护线程结束时，守护线程随着 JVM 一同结束工作。

98. volatile 关键字是否能保证线程安全？

答:

不能.

99. 存在使 i+1<i 的数么?

答:

存在, int 的最大值，加 1 后变为负数.

100. 接口可否继承接口？抽象类是否可实现接口？抽象类是否可继承实体类？

答:

接口可以继承接口，抽象类可以实现接口，抽象类可以继承实体类

101. 是否可以继承 String 类?

答:

不可以，因为 String 是抽象类

102. int 与 Integer 有什么区别？

答:

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。

103. 可序列化对象为什么要定义 serialversionUID 值?

答:

SerialVersionUid，简言之，其目的是序列化对象版本控制，有关各版本反序列化时是否兼容。如果在新版本中这个值修改了，新版本就不兼容旧

版本，反序列化时会抛出 InvalidClassException 异常。如果修改较小，比如仅仅是增加了一个属性，我们希望向下兼容，老版本的数据都能保留，那就不用修改；如果我们删除了一个属性，或者更改了类的继承关系，必然不兼容旧数据，这时就应该手动更新版本号，即 SerialVersionUid。

104. 写一个 Java 正则，能过滤出 html 中的 titl形式中的链接地址和标题.

答:

<a\b[^>]+\bhref="([^"]*)"[^>]*>([\s\S]*?)

分组 1 和分组 2 即为 href 和 value

105. 列出除 Singleton 外的常用的 3 种设计模式，并简单描述

答:

工厂模式：工厂模式是 Java 中最常用的设计模式之一。这种类型的设计模式属于创建型模式，它提供了一种创建对象的最佳方式。在工厂模式中，我们在创建对象时不会对客户端暴露创建逻辑，并且是通过使用一个共同的接口来指向新创建的对象。

适配器模式：适配器模式是作为两个不兼容的接口之间的桥梁。这种类型的设计模式属于结构型模式，它结合了两个独立接口的功能。这种模式涉及到一个单一的类，该类负责加入独立的或不兼容的接口功能。

模板模式：在模板模式中，一个抽象类公开定义了执行它的方法的方式/模板。它的子类可以按需要重写方法实现，但调用将以抽象类中定义的方式进行。

106. 十进制数 72 转换成八进制数是多少？

答: 110

107. WEB 应用中如果有.class 和.jar 类型的文件一般分别应该放在什么位置？

答:

.class 文件放在 WEB-INF/classes 文件下，.jar 文件放在 WEB-INF/lib 文件夹下

108. 元素中有一个输入框（ <input type=”text” name=”username”id=”username”value=”” />,请用 JavaScript 语言写一行代码，取得这个输入框中的值。

答:

document.getElementById(“username”).value;

109. 描述一下 JVM 加载 class 文件的原理机制?

答:

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

由于 Java 的跨平台性 经过编译的 Java 源程序并不是一个可执行程序，而是一个或多个类文件。当 Java 程序需要使用某个类时，JVM 会确保这个类已经被加载、连接（验证、准备和解析）和初始化。类的加载是指把类的.class 文件中的数据读入到内存中，通常是创建一个字节数组读入.class 文件，然后产生与所加载类对应的 Class 对象。加载完成后，Class 对象还

不完整，所以此时的类还不可用。当类被加载后就进入连接阶段，这一阶段包括验证、准备（为静态变量分配内存并设置默认的初始值）和解析（将符号引用替换为直接引用）三个步骤。最后 JVM 对类进行初始化，包括：1)如果类存在直接的父类并且这个类还没有被初始化，那么就先初始化父类；2)如果类中存在初始化语句，就依次执行这些初始化语句。

类的加载是由类加载器完成的，类加载器包括：根加载器（BootStrap）、扩展加载器（Extension）、系统加载器（System）和用户自定义类加载器（java.lang.ClassLoader 的子类）。从 Java 2（JDK 1.2）开始，类加载过程采取了父亲委托机制（PDM）。PDM 更好的保证了 Java 平台的安全性，在该机制中，JVM 自带的 Bootstrap 是根加载器，其他的加载器都有且仅有一个父类加载器。类的加载首先请求父类加载器加载，父类加载器无能为力时才由其子类加载器自行加载。JVM 不会向 Java 程序提供对 Bootstrap 的引用。下面是关于几个类加载器的说明：

Bootstrap：一般用本地代码实现，负责加载 JVM 基础核心类库（rt.jar）；

Extension：从 java.ext.dirs 系统属性所指定的目录中加载类库，它的父加载器是 Bootstrap；

System：又叫应用类加载器，其父类是 Extension。它是应用最广泛的类加载器。它从环境变量 classpath 或者系统属性 java.class.path 所指定的目录中记载类，是用户自定义加载器的默认父加载器。

110. Java 程序中创建新的类对象，使用关键字 new，回收无用的类对象使用关键字 free 正确么？

答:

Java 程序中创建新的类对象，使用关键字 new 是正确的; 回收无用的类对象使用关键字 free 是错误的.

111. Class 类的 getDeclaredFields()方法与 getFields()的区别？

答:

getDeclaredFields(): 可以获取所有本类自己声明的方法，不能获取继承的方法

getFields(): 只能获取所有 public 声 43 明的方法，包括继承的方法

112. 在 switch 和 if-else 语句之间进行选取，当控制选择的条件不仅仅依赖于一个 x 时，应该使用 switch 结构；正确么？

答:

不正确

113. 描述 final、finally、finalize 的区别。

答:

final 修饰符（关键字）如果一个类被声明为 final，意味着它不能再派生出新的子类，不能作为父类被继承。将变量或方法声明为 final，可以保证它们在使用中不被改变。被声明为 final 的变量必须在声明时给定初值，而在以后的引用中只能读取，不可修改。被声明为 final 的方法也同样只能使用，不能重载。

finally 在异常处理时提供 finally 块来执行任何清除操作。如果有 finally 的话，则不管是否发生异常，finally 语句都会被执行。

finalize 方法名。Java 技术允许使用 finalize() 方法在垃圾收集器将对象从内存中清除出去之前做必要清理工作。finalize() 方法是在垃圾收集器删除对象之前被调用的。它是在 Object 类中定义的，因此所有的类都继承了它。子类覆盖 finalize() 方法以整理系统资源或者执行其他清理工作。

114. 写一个 Singleton 例子

```
package com.bjsxt;


class Singleton{
    private Singleton(){}
    private static volatile Singleton singleton ;
    public static Singleton getInstance(){
        if(singleton==null)
            synchronized(Singleton.class){
                if(singleton==null)
                    singleton = new Singleton();
            }
        return singleton;
    }
}
```

115. 请写出常用的 Java 多线程启动方式，Executors 线程池有几种常用类型？

(1) 继承 Thread 类

```
public class java_thread extends Thread{
    public static void main(String args[]) {
        new java_thread().run();
        System.out.println("main thread run ");
    }
    public synchronized   void run() {
        System.out.println("sub thread run ");
    }
}
```

(2) 实现 Runnable 接口

```
public class java_thread implements Runnable{
    public static void main(String args[]) {
        new Thread(new java_thread()).start();
        System.out.println("main thread run ");
    }
    public void run() {
```

```
            System.out.println("sub thread run ");
        }
}
```

116. 手写 9x9 乘法表，冒泡排序

9x9 乘法表:

```
class Demo {
    public static void main(String[] args) {
        for(int x = 0;x <= 9; x++) {
            for(int y = 1;y <= x; y++) {
                System.out.print(y+"*"+x+"="+x*y+"\t");
            }
            System.out.println();
        }
    }
}
```

冒泡排序:

```
public class BubbleSort{
    public static void main(String[] args){
        int score[] = {67, 69, 75, 87, 89, 90, 99, 100};
```

```
            for (int i = 0; i < score.length -1; i++){//最多做 n-1 趟排序
                for(int j = 0 ;j < score.length - i - 1; j++){//对当前无序区间
score[0.....length-i-1]进行排序(j 的范围很关键，这个范围是在逐步缩小的)
                    if(score[j] < score[j + 1]){ //把小的值交换到后面
                        int temp = score[j];
                        score[j] = score[j + 1];
                        score[j + 1] = temp;
                    }
                }
                System.out.print("第" + (i + 1) + "次排序结果：");
                for(int a = 0; a < score.length; a++){
                    System.out.print(score[a] + "\t");
                }
                System.out.println("");
            }
            System.out.print("最终排序结果：");
            for(int a = 0; a < score.length; a++){
                System.out.print(score[a] + "\t");
            }
        }
}
```

117. 合并两个有序的链表

```
public class Solution {

public ListNode mergeTwoLists(ListNode l1, ListNode l2) {

if (l1 == null || l2 == null) {

            return l1 != null ? l1 : l2;

        }

        ListNode head = l1.val < l2.val ? l1 : l2;

        ListNode other = l1.val >= l2.val ? l1 : l2;

        ListNode prevHead = head;

        ListNode prevOther = other;

        while (prevHead != null) {

            ListNode next = prevHead.next;

            if (next != null && next.val > prevOther.val) {

                prevHead.next = prevOther;

                prevOther = next;

            }

            if(prevHead.next==null){

                prevHead.next=prevOther;

                break;

            }

            prevHead=prevHead.next;
```

```
        }
        return head;
}
}
```

118. 用递归方式实现链表的转置。

```
/**
Definition for singly-linked list.
public class ListNode {
int val;
ListNode next;
ListNode(int x) { val = x; }
* }
*/
public class Solution {
public ListNode reverseList(ListNode head) {
if(head==null||head.next ==null)
            return head;
        ListNode prev = reverseList(head.next);
        head.next.next = head;
        head.next = null;
        return prev;
```

```
    }
}
```

119. 题目： 给定一个整数数组，找到是否该数组包含任何重复数字。你的函数应该返回 true 只要有任何数字 在该数组中重复出现，否则返回 false。

```
public class Solution {
    public boolean containsDuplicate(int[] nums) {
        Set<Integer> numSet = new HashSet<Integer>();
        for(int i=0;i<nums.length;i++){·
            if(numSet.contains(nums[i]))
                return true;
            else
                numSet.add(nums[i]);
        }
        return false;
    }
}
```

120. 给定一个数组 nums， 写一个函数来移动所有 0 元素到数组末尾，同时维持数组中非 0 元素的相对顺序不变。要求不能申请额外的内存空间，并且最小化操作次数。

```
public void moveZeroes(int[] nums) {
        int size = nums.length;
        int startIndex = 0;
// 0 元素开始的位置
        int endIndex = 0;
// 0 元素结束的位置
        int currentNum;
        int i= 0;
        // 第一步：找到第一个 0 元素开始的位置
        // 并将第一个 0 元素的游标赋值给 startIndex&endIndex
        while(i < size){
            currentNum = nums[i];
            if (currentNum == 0) {
                startIndex = i;
                endIndex = i;
                break;
            }
            ++i;
        }
```

```
        // 如果当前数组中没有找到 0 元素，则推出
        if (nums[endIndex] != 0)
            return;

        // 将当前 i 的值加 1；直接从刚才 0 元素位置的后一位置开始循环
        ++i;
        while (i < size) {
            currentNum = nums[i];
            if (currentNum == 0){//如果当前元素等于 0，则将 i 值赋值给
endIndex
                endIndex = i;
            } else {
                // 如果不为 0
                //则将当前元素赋值给 nums[startIndex]
                // 并将当前位置的元素赋值为 0
                // startIndex 和 endIndex 都加 1；
                nums[startIndex] = currentNum;
                nums[i] = 0;
                ++startIndex;
                ++endIndex;
            }
            ++i;
```

```
        }
    }
```

121. 给定两个字符串 s 和 t， 写一个函数来决定是否 t 是 s 的重组词。你可以假设字符串只包含小写字母。

```
public class Solution {
    public boolean isAnagram(String s, String t) {
        if(s.length()!=t.length())
            return false;
        int bit[] = new int[26];
        for(int i=0;i<s.length();i++){
            bit[s.charAt(i)-'a']++;
        }

        for(int i=0;i<s.length();i++){
            if(--bit[t.charAt(i)-'a']<0)
                return false;
        }
        return true;
    }
```

```
}
```

122. 给定一个不包含相同元素的整数集合，nums，返回所有可能的子集集合。解答中集合不能包含重复的子集。

```
public class Solution {
    public List<List<Integer>> subsets (int[] nums) {
        List<List<Integer>> res = new ArrayList<ArrayList<Integer>>();
        List<Integer> item = new ArrayList<Integer>();
        if(nums.length == 0 || nums == null)
            return res;
        Arrays.sort(nums); //排序
        dfs(nums, 0, item, res);   //递归调用
        res.add(new ArrayList<Integer>());   //最后加上一个空集
        return res;
    }
    public static void dfs(int[] nums, int start, List<Integer> item,
List<List<Integer>> res){
        for(int i = start; i < nums.length; i ++){
            item.add(nums[i]);
            //item 是以整数为元素的动态数组，而 res 是以数组为元素的数
组，在这一步，当 item 增加完元素后，item 所有元素构成一个完整的子串，再
```

```
由 res 纳入
                res.add(new ArrayList<Integer>(item));
                dfs(nums, i + 1, item, res);
                item.remove(item.size() - 1);
            }
        }
}
```

123. 给定一颗二叉树，返回节点值得先序遍历，请使用迭代（非递归）方式实现。

```
public class Solution {
public List<Integer> preorderTraversal(TreeNode root) {
List<Integer> result = new ArrayList<Integer>();
            if(root == null)
                return result;
            Stack<TreeNode> stack = new Stack<TreeNode>();
            stack.push(root);
            while(!stack.isEmpty()) {
                TreeNode node = stack.pop();
                result.add(node.val);
                if(node.right != null)
```

```
                stack.push(node.right);
            if(node.left != null)
                stack.push(node.left);
        }
        return result;
    }
}
```

124. 验证一棵树是否为有效的二叉搜索树 BST

```
public class Solution {
    private static int lastVisit = Integer.MIN_VALUE;
public boolean isValidBST(TreeNode root) {
    if(root == null) return true;
        boolean judgeLeft = isValidBST(root.left); // 先判断左子树

        if(root.data >= lastVisit && judgeLeft) { // 当前节点比上次访问的
数值要大
            lastVisit = root.data;
        } else {
            return false;
        }
        boolean judgeRight = isValidBST(root.right); // 后判断右子树
```

```
        return judgeRight;
    }
}
```

125. 从一个链表中删除节点

题目： 写一个函数用于在一个单向链表中删除一个节点（非非尾节点），前提是仅仅能够访问要删除的那个节点。

比如给定链表 1 -> 3 -> 5 -> 7 -> 9 -> 16，给定你值为 3 的那个节点， 调用用你的函数后，链表变为

1 -> 5 -> 7 -> 9 -> 16。

```
/**
Definition for singly-linked list.
public class ListNode {
int val;
ListNode next;
ListNode(int x) { val = x; }
* }
*/
public class Solution {
public void deleteNode(ListNode node) {
if(node==null||node.next==null) {
            System.out.println("节点不存在或者是尾节点");
```

```
        }else{
            node.val=node.next.val;
            node.next=node.next.next;
        }
 }

}
```

126. 二叉搜索树 BST 中第 Kth 小的元素 题目：给定一个 BST，写一个函数 kthSmallest 来找到第 kth 小的元素

```
/**

Definition for a binary tree node.

public class TreeNode {

int val;

TreeNode left;

TreeNode right;

TreeNode(int x) { val = x; }

* }

*/

public class Solution2    {

    public int kthSmallest(TreeNode root, int k) {

        Stack<TreeNode> store = new Stack<TreeNode>();

        if (root == null) {

            return -1;

        }
```

```
        store.push(root);
        while (root.left != null) {
            store.push(root.left);
            root = root.left;
        }
        while (!store.empty()) {
            TreeNode cur = store.pop();
            k--;
            if (k == 0) {
                return cur.val;
            }
            if (cur.right != null) {
                root = cur.right;// let cur.right be the current node
                store.push(root);
                while (root.left != null) {
                    store.push(root.left);
                    root = root.left;
                }
            }
        }
        return -1;
    }
```

```
}
```

127. 题目：给定含有 n 个整数的数组 S，S 中是否存在三个元素 a,b,c 使得 a + b + c = 0？找到所有这样的三元 组，并且结果集中不包含重复的三元组。

比如，

S = [-1, 0, 1, 2, -1, -4],,

结果集为: [

[-1, 0, 1],

[-1, -1, 2]

]

```
/**
 * 给定一个 n 个元素的数组，是否存在 a，b，c 三个元素，使用得 a+b+c=0，找出所有符合这个条件的三元组
 * 注意： - 三元组中的元素必须是非递减的 - 结果不能包含重复元素
 */
public class Solution {

    public static void main(String[] args) {
        int[] S = {-1, 0, 1, 2, -1, -4,-3,-4,4,3};
        new Solution().get3Sum(S);
    }
```

```
public Set<String> get3Sum(int[] S){

    if(S.length<3 || S==null){
        return null;
    }

    //接收拼接的字符串
    StringBuffer sb = new StringBuffer();
    for(int i=0; i<S.length; i++){
        for(int j=0; j<S.length; j++){
            for(int z=0; z<S.length; z++){
                //筛选出不是递减的一组元素
                if(S[i]<=S[j] && S[j]<=S[z]){
                    int sum = S[i] + S[j] + S[z];
                    if(sum==0){
                        String str = "("+S[i]+","+S[j]+","+S[z]+")";
                        sb.append(str+";");
                    }
                }
            }
        }
```

```
        }

        String s = sb.toString();
        s = s.substring(0, sb.length()-1);
        String[] arr = s.split(";");

        Set<String> set = new HashSet<String>();
        //将所筛选出来的元素放入 Set 集合中，去重
        for (int k = 0; k < arr.length; k++) {
            set.add(arr[k]);
        }
        System.out.println(set);
        return set;
    }

    public List<List<Integer>> threeSum(int[] nums) {
        List<List<Integer>> result = new LinkedList<>();

        if (nums != null && nums.length > 2) {
            // 先对数组进行排序
            Arrays.sort(nums);
            // i 表示假设取第 i 个数作为结果
```

```
            for (int i = 0; i < nums.length - 2; ) {
                // 第二个数可能的起始位置
                int j = i + 1;
                // 第三个数可能是结束位置
                int k = nums.length - 1;
                while (j < k) {
                    // 如果找到满足条件的解
                    if (nums[j] + nums[k] == -nums[i]) {
                        // 将结果添加到结果含集中
                        List<Integer> list = new ArrayList<>(3);
                        list.add(nums[i]);
                        list.add(nums[j]);
                        list.add(nums[k]);
                        result.add(list);
                        // 移动到下一个位置，找下一组解
                        k--;
                        j++;

                        // 从左向右找第一个与之前处理的数不同的数的下
标
                        while (j < k && nums[j] == nums[j - 1]) {
                            j++;
```

```
                }
                // 从右向左找第一个与之前处理的数不同的数的下
标
                while (j < k && nums[k] == nums[k + 1]) {
                    k--;
                }
            }
            // 和大于 0
            else if (nums[j] + nums[k] > -nums[i]) {
                k--;
                // 从右向左找第一个与之前处理的数不同的数的下
标
                while (j < k && nums[k] == nums[k + 1]) {
                    k--;
                }
            }
            // 和小于 0
            else {
                j++;
                // 从左向右找第一个与之前处理的数不同的数的下
标
                while (j < k && nums[j] == nums[j - 1]) {
```

```
                        j++;
                    }
                }
            }
            // 指向下一个要处理的数
            i++;
            // 从左向右找第一个与之前处理的数不同的数的下标
            while (i < nums.length - 2 && nums[i] == nums[i - 1]) {
                i++;
            }
        }
    }

    return result;
  }
}
```

128. 子集问题

题目： 给定一个不包含相同元素的整数集合，nums，返回所有可能的子集集合。解答中集合不能包含重 复的子集。

比如，

nums = [1, 2, 3], 一一种解答为：

[

[3],

[1],

[2],

[1,2,3],

[1,3],

[2,3],

[1,2], []

]

```
/**
 * 不重复集合求子集
解答采用的是深度优先遍历 ,先取原数组一个元素 ,再构造包括这个元素的两个，三个……n 个元素的集合。dfs 中的 start 就指向这个元素的，它在不断地后移 ( i+1)。
 * @param S: A set of numbers.
 * @return: A list of lists. All valid subsets.
 */
public class Solution1 {
    public static void main(String[] args) {
        int[] first = new int[]{1, 2, 3};
```

```
        ArrayList<ArrayList<Integer>> res = subsets(first);
        for(int i = 0; i < res.size(); i ++){
            System.out.println(res.get(i));
        }
    }
    public static ArrayList<ArrayList<Integer>> subsets(int[] nums) {
        ArrayList<ArrayList<Integer>> res = new
ArrayList<ArrayList<Integer>>();
        ArrayList<Integer> item = new ArrayList<Integer>();
        if(nums.length == 0 || nums == null)
            return res;
        Arrays.sort(nums); //排序
        dfs(nums, 0, item, res);  //递归调用
        res.add(new ArrayList<Integer>());  //最后加上一个空集
        return res;
    }
    public static void dfs(int[] nums, int start, ArrayList<Integer>item,
ArrayList<ArrayList<Integer>>res){
        for(int i = start; i < nums.length; i ++){
            item.add(nums[i]);
            //item 是以整数为元素的动态数组，而 res 是以数组为元素的数
组，在这一步，当 item 增加完元素后，item 所有元素构成一个完整的子串，再
```

```
由 res 纳入
                res.add(new ArrayList<Integer>(item));
                dfs(nums, i + 1, item, res);
                item.remove(item.size() - 1);
            }
        }
}
```

129. 迭代方法实现二叉树的先序遍历：题目： 给定一颗二叉树，返回节点值得先序遍历，请使用迭代（非递归）方式实现。

比如， 给定二叉树{1,#,2,3}, 返回 [1,2,3]

```
/**
Definition for a binary tree node.
public class TreeNode {
int val;
TreeNode left;
TreeNode right;
TreeNode(int x) { val = x; }
* }
*/
public class Solution {
```

```
    List<Integer> result = new ArrayList<Integer>();


    /**
     * 迭代实现，维护一个栈，因为入栈顺序按照根右左进行入栈，因此首先将根出栈，然后出栈左子节点，
     * 最后出栈右子节点。
     * @param root
     * @return
     */
    public List<Integer> preorderTraversal(TreeNode root) {
        if(root == null)
            return result;
        Stack<TreeNode> stack = new Stack<TreeNode>();
        stack.push(root);
        while(!stack.isEmpty()) {
            TreeNode node = stack.pop();
            result.add(node.val);
            if(node.right != null)
                stack.push(node.right);
            if(node.left != null)
                stack.push(node.left);
        }
```

```
        return result;
    }
}
```

130. 验证二叉搜索树 BST：题目： 验证一棵树是否为有效的二叉搜索树 BST 比如 ,二叉树[2, 1, 3] ,返回 true 二叉树[1, 2, 3]，返回 false

```
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
TreeNode(int x) { val = x; }
}
public class BSTChecker {
    private static int lastVisit = Integer.MIN_VALUE;

    public static boolean isBST(TreeNode root) {
        if(root == null) return true;

        boolean judgeLeft = isBST(root.left); // 先判断左子树

        if(root.val >= lastVisit && judgeLeft) { // 当前节点比上次访问的数
```

```
值要大
                lastVisit = root.val;
            } else {
                return false;
            }

            boolean judgeRight = isBST(root.right); // 后判断右子树

            return judgeRight;
        }
}
```

131. 编辑距离题目： 给定两个单词 word1 和 word2，找到最小的操作步骤使得 word1 转换成 word2，每次操作算作一 步。你可以对单词进行以下三种操作：1）插入一个字符 2）删除一个字符 3）替换一个字符

有些绕，理解不是很透彻。

参考地址：http://www.cnblogs.com/masterlibin/p/5785092.html

132. String s=new String(“abc”);创建了几个 String 对象。

两个或一个，” abc”对应一个对象，这个对象放在字符串常量缓冲区，常量” abc”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” abc”对象的内容来创建出一个新

String 对象。如果以前就用过’abc’ ，这句代表就不会创建”abc”自己了，直接从缓冲区拿。

133. 买卖股票问题：题目： 你有一个数组，第 i 个元素表示第 i 天某个股票的价格，设计一个算法找到最大的利润，并且你只能最多完成两次交易。

参考地址：没有完全理解的

http://www.mamicode.com/info-detail-1087177.html
http://blog.csdn.net/cumt_cx/article/details/48015735

```
/**
 * 解题思路：
 * 比如给定一组数组，[1，2，3，6，9，3，10]
 * 最多可以 2 次去获取最大的利益，可以用 2 分的思想，分成 2 部分，
 * 从 0 元素开始遍历分别求出左右 2 边的最大利益，求出的左右 2 边最大的利益即为解
 */
class Solution {


    public static int maxProfit(int[] prices) {
        // write your code here
        if(null==prices||0==prices.length) return 0;
         int sumProfit = 0;
         for(int i=1;i<prices.length;i++){
```

```
                int tmpsum = maxProfit(prices, 0, i)
                        + maxProfit(prices, i+1, prices.length-1);
                sumProfit = Math.max(sumProfit, tmpsum);
            }
            return sumProfit;
        }
        public static int maxProfit(int[] prices,int s,int e){
            if(e<=s) return 0;
            int min = prices[s];
            int maxProfit = 0;
            for(int i=s+1;i<=e;i++){
                maxProfit = Math.max(maxProfit, prices[i]-min);
                min = Math.min(min, prices[i]);
            }
            return maxProfit;
        }
        public static void main(String[] args) {
            int arr [] =   {4,4,6,1,1,4,2,5};
            System.out.println(maxProfit(arr));
        }
}
```

134. 描述&和&&的区别。

&和&&的联系(共同点)：

&和&&都可以用作逻辑与运算符，但是要看使用时的具体条件来决定。

操作数 1&操作数 2，操作数 1&&操作数 2，

表达式 1&表达式 2，表达式 1&&表达式 2，

情况 1：当上述的操作数是 boolean 类型变量时，&和&&都可以用作逻辑与运算符。

情况 2：当上述的表达式结果是 boolean 类型变量时，&和&&都可以用作逻辑与运算符。

表示逻辑与(and)，当运算符两边的表达式的结果或操作数都为 true 时，整个运算结果才为 true，否则，只要有一方为 false，结果都为 false。

&和&&的区别(不同点)：

(1)、&逻辑运算符称为逻辑与运算符，&&逻辑运算符称为短路与运算符，也可叫逻辑与运算符。

对于&：无论任何情况，&两边的操作数或表达式都会参与计算。

对于&&：当&&左边的操作数为 false 或左边表达式结果为 false 时，&&右边的操作数或表达式将不参与计算，此时最终结果都为 false。

综上所述，如果逻辑与运算的第一个操作数是 false 或第一个表达式的结果为 false 时，对于第二个操作数或表达式是否进行运算，对最终的结果没有影响，结果肯定是 false。推介平时多使用&&，因为它效率更高些。

(2)、&还可以用作位运算符。当&两边操作数或两边表达式的结果不是 boolean 类型时，&用于按位与运算符的操作。

135. 使用 final 关键字修饰符一个变量时，是引用不能变，还是引用的对象不能变？

final 修饰基本类型变量，其值不能改变。

但是 final 修饰引用类型变量，栈内存中的引用不能改变，但是所指向的堆内存中的对象的属性值仍旧可以改变。例如

```
class Test {
    public static void main(String[] args) {
        final Dog dog = new Dog("欧欧");
        dog.name = "美美";//正确
        dog = new Dog("亚亚");//错误
    }
}
```

136. 简单描述一下 Servlet 与 JSP 的的相同点和区别点。

区别：

JSP 是在 HTML 代码里写 JAVA 代码,框架是 HTML;而 Servlet 是在 JAVA 代码中写 HTML 代码，本身是个 JAVA 类。

JSP 使人们把显示和逻辑分隔成为可能，这意味着两者的开发可并行进行；而 Servlet 并没有把两者分开。

Servlet 独立地处理静态表示逻辑与动态业务逻辑.这样,任何文件的变动都

需要对此服务程序重新编译;JSP 允许用特殊标签直接嵌入到 HTML 页面,HTML 内容与 JAVA 内容也可放在单独文件中,HTML 内容的任何变动会自动编译装入到服务程序.

Servlet 需要在 web.xml 中配置，而 JSP 无需配置。

目前 JSP 主要用在视图层，负责显示，而 Servlet 主要用在控制层，负责调度

联系：

都是 Sun 公司推出的动态网页技术。

先有 Servlet，针对 Servlet 缺点推出 JSP。JSP 是 Servlet 的一种特殊形式，每个 JSP 页面就是一个 Servlet 实例——JSP 页面由系统翻译成 Servlet，Servlet 再负责响应用户请求。

137. 请简单描述下几个您熟悉 JavaScript 库，它们有哪些作用和特点？

JavaScript 高级程序设计（特别是对浏览器差异的复杂处理），通常很困难也很耗时。为了应对这些调整，许多的 JavaScript 库应运而生。这些 JavaScript 库常被称为 JavaScript 框架。

jQuery:

Ext JS - 可定制的 widget，用于构建富因特网应用程序（rich Internet applications）。

Prototype

MooTools。

YUI - Yahoo! User Interface Framework，涵盖大量函数的大型库，从简

单的 JavaScript 功能到完整的 internet widget。

138. 简单描述下几个 HTML，CSS，Javascript 在 Web 前端开发中分别起什么作用？

1、什么是 HTML（超文本标记语言 Hyper Text Markup Language），HTML 是用来描述网页的一种语言。

2、CSS(层叠样式表 Cascading Style Sheets),样式定义如何显示 HTML 元素，语法为：selector {property：value} (选择符 {属性：值})

3、JavaScript 是一种脚本语言，其源代码在发往客户端运行之前不需经过编译，而是将文本格式的字符代码发送给浏览器由浏览器解释运行

对于一个网页，HTML 定义网页的结构，CSS 描述网页的样子，JavaScript 设置一个很经典的例子是说 HTML 就像 一个人的骨骼、器官，而 CSS 就是人的皮肤，有了这两样也就构成了一个植物人了，加上 javascript 这个植物人就可以对外界刺激做出反应，可以思 考、运动、可以给自己整容化妆（改变 CSS）等等，成为一个活生生的人。

如果说 HTML 是肉身、CSS 就是皮相、Javascript 就是灵魂。没有 Javascript,HTML+CSS 是植物人，没有 Javascript、CSS 是个毁容的植物人。

如果说 HTML 是建筑师，CSS 就是干装修的，Javascript 是魔术师。

139. 输出结果？

```
String str1=“hello”;

Sring str2=“he”+new String(“llo”);

Sysem.out.println(str1==str2));
```

```
Sysem.out.println(str.equal(str2));
```

```
false

true
```

140. 下列程序的输出结果是什么？

```
import java.util.*;

public class Test 6{

    public static void main(String[] args) {

        for (int i = 0; i < 10; i++) {

            Integer k=new Integer(i);

            System.out.println(k+" Hello world");

        }

    }

}
```

```
0 Hello world

1 Hello world

2 Hello world

3 Hello world

4 Hello world

5 Hello world

6 Hello world
```

7 Hello world

8 Hello world

9 Hello world

141. 写一个完整函数，实现拷贝数组

```
PUBLIC CLASS TEST {
    PUBLIC STATIC VOID MAIN(STRING[] ARGS) {
        INT [] ARR1 = {10,20,30,40,50};
        INT [] ARR2 = COPYARRAY(ARR1);
        SYSTEM.OUT.PRINTLN(ARRAYS.TOSTRING(ARR2));
    }

    PUBLIC STATIC INT [] COPYARRAY(INT [] ARR){
        INT [] ARR2 = NEW INT[ARR.LENGTH];
        FOR(INT I=0;I<ARR.LENGTH;I++){
            ARR2[I] = ARR[I];
        }
        RETURN ARR2;
    }
}
```

142. 请写出一段代码，能够完成将字符串写入文件

```
public class Test2 {
```

```
    public static void main(String[] args) {
    String str ="bjsxt";
    writeFile(str);
}
    public static void writeFile(String str){
    File file = new File("c:/test.txt");
    PrintStream ps = null;
    try {
        OutputStream fos = new FileOutputStream(file);
        ps = new PrintStream(fos);
        ps.print(str);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }finally{
        ps.close();
    }
 }
}
```

143. 请解释以下常用正则含义：\d,\D,\s,.,*,?,|,[0-9]{6},\d+

\d: 匹配一个数字字符。等价于[0-9]

\D: 匹配一个非数字字符。等价于[^0-9]

\s: 匹配任何空白字符 ,包括空格、制表符、换页符等等。等价于 [\f\n\r\t\v]

. ：匹配除换行符 \n 之外的任何单字符。要匹配 . ，请使用 \. 。

*：匹配前面的子表达式零次或多次。要匹配 * 字符，请使用 *。

+：匹配前面的子表达式一次或多次。要匹配 + 字符，请使用 \+。

|:将两个匹配条件进行逻辑“或”（Or）运算

[0-9]{6}:匹配连续 6 个 0-9 之间的数字

\d+：匹配至少一个 0-9 之间的数字

144. 关于 sleep()和 wait()，以下描述错误的一项是（D）

A.	sleep 是线程类（Thread）的方法，wait 是 Object 类的方法
B.	Sleep 不释放对象锁，wait 放弃对象锁
C.	Sleep 暂停线程、但监控状态任然保持，结束后会自动恢复
D.	Wait 后进入等待锁定池，只针对此对象发出 notify 方法后获取对象锁进入运行状态。
分析： 针对此对象的 notify 方法后获取对象锁并进入就绪状态 ,而不是运行状态。 另外针对此对象的 notifyAll 方法后也可能获取对象锁并进入就绪状态 ,而不是运行状态	

145. 已知表达式 int m[] = {0,1,2,3,4,5,6}; 下面那个表达式的值与数组的长度相等（B）

A.	m.length()
B.	m.length
C.	m.length()+1

D.	m.length+1
分析：数组的长度是.length	

146. 下面那些声明是合法的？（AD）

A.	long l = 4990
B.	int i = 4L
C.	float f = 1.1
D.	double d = 34.4
分析：B int 属于整数型应该是 int=4 C 应该是 float f=1.1f	

147. 以下选项中选择正确的 java 表达式（CD）

A.	int k=new String（“aa”）
B.	String str = String（“bb”）
C.	char c=74;
D.	long j=8888;
分析：A 需要轻质类型转换　B　String str =new String（“bb”）	

148. 下列代码的输出结果是

```
System.out.println(""+("12"=="12"&&"12".equals("12")));

（“12”==”12”&&”12”.equals（“12”））

“12”==”12”&&”12”.equals（“12”）
```

true

false

149. 以下哪些运算符是含有短路运算机制的？请选择：(BD)

A.	&
B.	&&
C.	|
D.	||
分析：A C 是逻辑与计算	

150. 下面哪个函数是 public void example(){....}的重载函数？（AD）

A.	private void example（int m）{...}
B.	public int example（ ）{...}
C.	public void example2（ ）{...}
D.	public int example（int m.float f）{...}
分析：BC 定义的是新函数	

151. 在调用方法时，若要使方法改变实参的值(即调用方法完成后，参数的内容可能发现改变)，可以（B）

A.	用基本数据类型作为参数
B.	用对象作为参数
C.	A 和 B 都对
D.	A 和 B 都不对
分析：	

152. 给定某 java 程序片段，该程序运行后，j 的输出结果为（B）

```
int i=1;
```

```
Int   j=i++ ;
If ( ( j>++j ) &&(i++==j) ) {j+=i:}
System.out.println(j);
```

A.	1
B.	2
C.	3
D.	4
分析： i++先引用后。++i 先增加后引用	

153. 在 java 中，无论测试条件是什么，下列（B）循环将至少执行一次。

A.	for
B.	do...while
C.	while
D.	while...do
分析： ACD 都不一定进行循环	

154. [编程]任给 n 个整数和一个整数 x。请计算 n 个整数中有多少对整数之和等于 x。

```
public class Test8 {
    public static void main(String[] args) {
        //输入 n 个整数和一个整数
        Scanner input = new Scanner(System.in);
```

```
        System.out.println("请输入 n 个整数，数量任意,以逗号分隔");
        String str = input.next();
        System.out.println("请输入一个整数：");
        int x = input.nextInt();
        //将 n 个整数的字符串转换为数组
        String arr1[] = str.split(",");
        int [] arr2 = new int[arr1.length];
        for(int i=0;i<arr1.length;i++){
            arr2[i] = Integer.parseInt(arr1[i]);
        }
        System.out.println(Arrays.toString(arr2));
        //判断并输出 n 个整数中有几对的和等于 x
        int count = 0;
        for(int i=0;i<arr2.length-1;i++){
            for(int j = i+1;j<arr2.length;j++){
                if(arr2[i]+arr2[j]==x){
                    count++;
                }
            }
        }
        System.out.println(count);
    }
```

```
}
```

155. [编程]请说明快速排序算法的设计思想和时间复杂度，并用高级语言写出对整数数组进行一趟快排的函数实现。

快速排序由 C. A. R. Hoare 在 1962 年提出。它的基本思想是：通过一趟排序将要排序的数据分割成独立的两部分，其中一部分的所有数据都比另外一部分的所有数据都要小，然后再按此方法对这两部分数据分别进行快速排序，整个排序过程可以递归进行，以此达到整个数据变成有序序列。

设要排序的数组是 A[0]……A[N-1]，首先任意选取一个数据（通常选用数组的第一个数）作为关键数据，然后将所有比它小的数都放到它前面，所有比它大的数都放到它后面，这个过程称为一趟快速排序。值得注意的是，快速排序不是一种稳定的排序算法，也就是说，多个相同的值的相对位置也许会在算法结束时产生变动。

一趟快速排序的算法是：

1、设置两个变量 i、j，排序开始的时候：i=0，j=N-1；

2、以第一个数组元素作为关键数据，赋值给 key，即 key=A[0]；

3、从 j 开始向前搜索，即由后开始向前搜索(j--)，找到第一个小于 key 的值 A[j]，将 A[j]和 A[i]互换；

4、从 i 开始向后搜索，即由前开始向后搜索(i++)，找到第一个大于 key 的 A[i]，将 A[i]和 A[j]互换；

5、重复第 3、4 步，直到 i=j； (3,4 步中，没找到符合条件的值，即 3 中 A[j]不小于 key,4 中 A[i]不大于 key 的时候改变 j、i 的值，使得 j=j-1，i=i+1，直至找到为止。找

到符合条件的值，进行交换的时候 i， j 指针位置不变。另外，i==j 这一过程一定正好是 i+或 j-完成的时候，此时令循环结束）。

```
public class Quick {
    public static void main(String[] args) {
        int arr [] = {90,60,70,50,40,80,20,100,10};
        sort(arr,0,arr.length-1);
        System.out.println(Arrays.toString(arr));
    }
    public static void sort(int arr[], int low, int high) {
        //设置两个变量 l、h，排序开始的时候：l=0，h=N-1
        int l = low;
        int h = high;
        //以第一个数组元素作为关键数据，赋值给 key，即 key=A[0]
        int key = arr[low];
        //重复操作，直到 i=j
        while (l < h) {
            //从 h 开始向前搜索，即由后开始向前搜索(h--)，找到第一个小于 key 的值 arr[h]，将 arr[h]和 arr[l]互换
            while (l < h && arr[h] >= key)
                h--;
            if (l < h) {
                int temp = arr[h];
```

```
                arr[h] = arr[l];
                arr[l] = temp;
                l++;
            }
            //从 l 开始向后搜索，即由前开始向后搜索(l++)，找到第一个大于 key 的 arr[l]，将 arr[l]和 arr[h]互换；
            while (l < h && arr[l] <= key)
                l++;

            if (l < h) {
                int temp = arr[h];
                arr[h] = arr[l];
                arr[l] = temp;
                h--;
            }
        }
        //对前一部分进行快速排序
        if (l > low)
            sort(arr, low, l - 1);
        //对前一部分进行快速排序
        if (h < high)
            sort(arr, l + 1, high);
```

```
    }
}
```

156. 打印结果：2

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

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

```
}
```

157. 指出下列程序的运行结果

```
int i=9;
        switch (i) {
        default:
            System.out.println("default");
        case 0:
            System.out.println("zero");
            break;
        case 1:
            System.out.println("one");
            break;
        case 2:
            System.out.println("two");
            break;
```

```
default
zero
```

158. 数组有没有 length()这个方法？String 呢？

数组没有 length()方法，有 length 属性

String 有 length()方法

159. 解释继承、重载、覆盖。

继承（英语：inheritance）是面向对象软件技术当中的一个概念。如果一个类别 A“继承自”另一个类别 B，就把这个 A 称为“B 的子类别”，而把 B 称为“A 的父类别”也可以称“B 是 A 的超类”。继承可以使得子类别具有父类别的各种属性和方法，而不需要再次编写相同的代码。在令子类别继承父类别的同时，可以重新定义某些属性，并重写某些方法，即覆盖父类别的原有属性和方法，使其获得与父类别不同的功能。另外，为子类别追加新的属性和方法也是常见的做法。 一般静态的面向对象编程语言，继承属于静态的，意即在子类别的行为在编译期就已经决定，无法在执行期扩充。

那么如何使用继承呢？用 extends 关键字来继承父类。

如上面 A 类与 B 类，当写继承语句时， class A 类 extends B 类{ } 其中 A 类是子类，B 类是父类。

	英文	位置不同	作用不同
重载	overload	同一个类中	在一个类里面为一种行为提供多种实现方式并提高可读性
重写	override	子类和父类间	父类方法无法满足子类的要求，子类通过方法重写满足要求

	修饰符	返回值	方法名	参数	抛出异常
重载	无关	无关	相同	不同	无关
重写	大于等于	小于等于	相同	相同	小于等于

160. 进程和线程的区别是什么？

进程是具有一定独立功能的程序关于某个数据集合上的一次运行活动,进程是系统进行资源分配和调度的一个独立单位.

线程是进程的一个实体,是 CPU 调度和分派的基本单位,它是比进程更小的能独立运行的基本单位.线程自己基本上不拥有系统资源,只拥有一点在运行中必不可少的资源(如程序计数器,一组寄存器和栈),但是它可与同属一个进程的其他的线程共享进程所拥有的全部资源.

区别	进程	线程
根本区别	作为资源分配的单位	调度和执行的单位
开销	每个进程都有独立的代码和数据空间(进程上下文),进程间的切换会有较大的开销。	线程可以看成时轻量级的进程，同一类线程共享代码和数据空间，每个线程有独立的运行栈和程序计数器(PC)，线程切换的开销小。
所处环境	在操作系统中能同时运行多个任务(程序)	在同一应用程序中有多个顺序流同时执行
分配内存	系统在运行的时候会为每个进程分配不同的内存区域	除了 CPU 之外，不会为线程分配内存（线程所使用的资源是它所属的进程的资源），线程组只能共

		享资源
包含关系	没有线程的进程是可以被看作单线程的，如果一个进程内拥有多个线程，则执行过程不是一条线的，而是多条线（线程）共同完成的。	线程是进程的一部分，所以线程有的时候被称为是轻权进程或者轻量级进程。

161. 什么是编译型语言，什么是解释型语言？java 可以归类到那种？

计算机不能直接理解高级语言，只能理解和运行机器语言，所以必须要把高级语言翻译成机器语言，计算机才能运行高级语言所编写的程序。翻译的方式有两种，一个是编译，一个是解释。

用编译型语言写的程序执行之前，需要一个专门的编译过程，通过编译系统把高级语言翻译成机器语言，把源高级程序编译成为机器语言文件，比如 windows 下的 exe 文件。以后就可以直接运行而不需要编译了，因为翻译只做了一次，运行时不需要翻译，所以一般而言，编译型语言的程序执行效率高。

解释型语言在运行的时候才翻译，比如 VB 语言，在执行的时候，专门有一个解释器能够将 VB 语言翻译成机器语言，每个语句都是执行时才翻译。这样解释型语言每执行一次就要翻译一次，效率比较低。

编译型与解释型，两者各有利弊。前者由于程序执行速度快，同等条

件下对系统要求较低，因此像开发操作系统、大型应用程序、数据库系统等时都采用它，像 C/C++、Pascal/Object Pascal（Delphi）等都是编译语言，而一些网页脚本、服务器脚本及辅助开发接口这样的对速度要求不高、对不同系统平台间的兼容性有一定要求的程序则通常使用解释性语言，如 JavaScript、VBScript、Perl、Python、Ruby、MATLAB 等等。

JAVA 语言是一种编译型-解释型语言，同时具备编译特性和解释特性（其实，确切的说 java 就是解释型语言，其所谓的编译过程只是将.java 文件编程成平台无关的字节码.class 文件，并不是向 C 一样编译成可执行的机器语言，在此请读者注意 Java 中所谓的“编译”和传统的“编译”的区别）。作为编译型语言，JAVA 程序要被统一编译成字节码文件——文件后缀是 class。此种文件在 java 中又称为类文件。java 类文件不能再计算机上直接执行，它需要被 java 虚拟机翻译成本地的机器码后才能执行，而 java 虚拟机的翻译过程则是解释性的。java 字节码文件首先被加载到计算机内存中，然后读出一条指令，翻译一条指令，执行一条指令，该过程被称为 java 语言的解释执行，是由 java 虚拟机完成的。

162. 简述操作符（&，|）与操作符（&&，||）的区别

&和&&的联系(共同点)：

&和&&都可以用作逻辑与运算符，但是要看使用时的具体条件来决定。

操作数 1&操作数 2，操作数 1&&操作数 2，
表达式 1&表达式 2，表达式 1&&表达式 2，

情况 1：当上述的操作数是 boolean 类型变量时，&和&&都可以用作逻辑与运算符。

情况 2：当上述的表达式结果是 boolean 类型变量时，&和&&都可以用作逻辑与运算符。

表示逻辑与(and)，当运算符两边的表达式的结果或操作数都为 true 时，整个运算结果才为 true，否则，只要有一方为 false，结果都为 false。

&和&&的区别(不同点)：

(1)、&逻辑运算符称为逻辑与运算符，&&逻辑运算符称为短路与运算符，也可叫逻辑与运算符。

对于&：无论任何情况，&两边的操作数或表达式都会参与计算。

对于&&：当&&左边的操作数为 false 或左边表达式结果为 false 时，&&右边的操作数或表达式将不参与计算，此时最终结果都为 false。

综上所述，如果逻辑与运算的第一个操作数是 false 或第一个表达式的结果为 false 时，对于第二个操作数或表达式是否进行运算，对最终的结果没有影响，结果肯定是 false。推介平时多使用&&，因为它效率更高些。

(2)、&还可以用作位运算符。当&两边操作数或两边表达式的结果不是 boolean 类型时，&用于按位与运算符的操作。

|和||的区别和联系与&和&&的区别和联系类似

163. try{}里面有一个 return 语句，那么紧跟在这个 try 后的 finally，里面的语句在异常出现后，都会执行么？为什么？

在异常处理时提供 finally 块来执行任何清除操作。

如果有 finally 的话，则不管是否发生异常，finally 语句都会被执行，包括

遇到 return 语句。

finally 中语句不执行的唯一情况中执行了 System.exit(0)语句。

164. 对于一段形如：1，-1~3,1~15×3 的输入

输入会依照以下规则：

1、所有输入为整数、

2、“,”为分隔符

3、“~”表示一个区间，比如“-1~3”表示-1 到 3 总共 5 个整数，同时“~”前的数小于“~”后的数：

4、“x”表示步长，“x3”指每 3 个整数一个，比如“1~15×3”表示 1，4,7,10,13；

根据以上得到的结果进行打印，打印的规则为：

1、所有得到的整数按从小到大排列，以“,”分隔，不计重复；

2、每行最多显示 3 个整数；

3、如果两个整数是连续的，可以放在同一行，否则自动换行。

例如对于输入“1，-1~3,1~15×3”的输出结果为：

-1,0,1，

2,3,4，

7,

10，

13

```
public class Test {
public static void main(String[] args) {
    Map<Integer, Integer> map = new TreeMap<Integer,
```

```
Integer>();
        String str = "5~20x3,1,-1~3,1~15x3";
        String[] s = str.split(",");
        for (int i = 0; i < s.length; i++) {
            if (s[i].contains("~")) {
                String ss[] = s[i].split("~");
                int first = Integer.parseInt(ss[0]);
                if (s[i].contains("x")) {
                    String sss[] = ss[1].split("x");
                    int end = Integer.parseInt(sss[0]);
                    int l = Integer.parseInt(sss[1]);
                    for (int j = first; j < end;) {
                        map.put(j, j);
                        j += l;
                    }
                } else {
                    int end = Integer.parseInt(ss[ss.length - 1]);
                    for (int j = first; j <= end; j++) {
                        map.put(j, j);
                    }
                }
            } else {
```

```
            int j = Integer.parseInt(s[i]);
            map.put(j, j);
        }
    }
    List<Integer> list = new ArrayList<Integer>();
    Set<Integer> set = map.keySet();
    Iterator<Integer> ite = set.iterator();
    while (ite.hasNext()) {
        int key = ite.next();
        int value = map.get(key);
        list.add(value);
        System.out.println("v :" + value);
    }
    System.out.println("==================");
    for (int i = 0; i < list.size();) {
        int value = list.get(i);
        List<Integer> co = new ArrayList<Integer>();
        co.add(value + 1);
        co.add(value + 2);
        if (list.containsAll(co)) {
            System.out.println(value + "," + (value + 1) + ","
                    + (value + 2));
```

```
                i += 3;
            } else {
                System.out.println(value);
                i++;
            }
        }
    }
}
```

165. 有一段 java 应用程序，它的主类名是 al，那么保存它的源文件可以是？(A)

A.	al.java
B.	al.class
C.	al
D.	都对
分析：.class 是 java 的解析文件	

166. Java 类可以作为（c）

A	类型定义机制
B.	数据封装机制
C.	类型定义机制和数据封装机制

D.	上述都不对
分析：	

167. 在调用方法时，若要使方法改变实参的值，可以？（B）

A	用基本数据类型作为参数
B.	用对象作为参数
C.	A和B都对
D.	A 和 B 都不对
分析：基本数据类型不能改变实参的值	

168. Java 语言具有许多优点和特点，哪个反映了 java 程序并行机制的(BC)

A	安全性
B.	多线性
C.	跨平台
D.	可移植
分析：	

169. 下关于构造函数的描述错误是(A)

A	构造函数的返回类型只能是void型
B.	构造函数是类的一种特殊函数，它的方法名必须与类名相同
C.	构造函数的主要作用是完成对类的对象的初始化工作
D.	一般在创建新对象时，系统会自动调用构造函数
分析：构造函数的名字与类的名字相同，并且不能指定返回类型。	

170. 若需要定义一个类域或类方法，应使用哪种修饰符？（A）

A	static
B.	package
C.	private
D.	public
分析：A	

171. 储蓄所有多个储户，储户在多个储户所存取款，储蓄所与储户之间是(D)

A	一对一的联系
B.	多对一的联系
C.	一对多的联系
D.	多对多的联系
分析：D	

视图是一个“虚表”，视图的构造基于(A)

A	基本表或视图
B.	**视图**
C.	数据字典
D.	基本表
分析：A	

172. 设有关系 R(A,B,C,D)及其上的函数相关性集合 F={B→A,BC→D},那么关系 R 最高是(A)

A	第一范式的
B.	第二范式的
C.	第三范式的
D.	BCNF 范式的
分析：A	

173. 已知如下代码：执行结果是什么(A)

```
package com.bjsxt;

public class Test {

    public static void main(String[] args) {

        String s1 = new String("Hello");

        String s2 = new String("Hello");

        System.out.print(s1 == s2);

        String s3 = "Hello";

        String s4 = "Hello";

        System.out.print(s3 == s4);

        s1 = s3;

        s2 = s4;

        System.out.print(s1 == s2);

    }

}
```

A	false true true
B.	true false true
C.	true true false
D.	true true false
分析：A	

174. 下面代码执行后的输出是什么(A)

```
package com.bjsxt;

public class Test {

    public static void main(String[] args) {

        outer: for (int i = 0; i < 3; i++)

        inner: for (int j = 0; j < 2; j++) {

            if (j == 1)

                continue outer;

            System.out.println(j + " and " + i);

        }

    }

}
```

A	0 and 0 0 and 1 0 and 2

B.	1 and 0 1 and 1 1 and 2
C.	2 and 0 2 and1 2 and 2
分析：A	

175. 给出如下代码，如何使成员变量 m 被函数 fun()直接访问(C)

```
package com.bjsxt;
public class Test {
    private int m;

    public static void fun() {
        // some code...
    }
}
```

A	将 private int m 改为 protected int m
B.	将 private int m 改为 public int m
C.	将 private int m 改为 static int m
D.	将 private int m 改为 int m
分析：C	

176. 下面哪几个函数是 public void example（ ）{….}的重载函数(ABD)

A	public void example（int m ）{…}
B.	public int example（int m ）{…}
C.	public void example2（ ）{…}
D.	public int example（int m，float f ）{…}
分析：ABD	

177. 当 DOM 加载完成后要执行的函数，下面哪个是正确的(C)

A	JQuery（expression, [context]）
B.	JQuery（html, [ownerDocument]）
C.	JQuery（callback）
分析：C	

178. 下面哪个是正确的（BD）

A	String temp[ ] = new String{ “a”,” b”,” c” };
B.	String temp[ ] = { “a”,” b”,” c” };
C.	String temp= { “a”,” b”,” c” };
D.	String[ ] temp = { “a”,” b”,” c” };
分析：BD	

179. 关于 java.lang.String 类，以下描述正确的一项是（A）

A	String类是final类故不可继承
B.	String 类 final 类故可以继承
C.	String类不是final类故不可继承

D.	String;类不是 final 类故可以继承
分析：A	

180. 以下结构中，插入性能最高的是（B）

A	ArrayList
B.	Linkedlist
C.	tor
D.	Collection
分析：B	

181. 以下结构中，哪个具有同步功能（B）

A	HashMap
B.	ConcurrentHashMap
C.	WeakHashMap
D.	TreeMap
分析：B	

182. 以下结构中，哪个最适合当作 stack 使用（C）

A	LinkedHashMap
B.	LinkedHashSet
C.	LinkedList
分析：C	

183. 以下锁机机制中，不能保证线程安全的是（C）

A	Lock

B.	Synchronized
C.	Volatile
分析：C	

184. 下面哪个流类属于面向字符的输入流（D）

A	BufferedWriter
B.	FileInputStream
C.	ObjectInputStream
D.	InputStreamReader
分析：D	

不通过构造函数也能创建对象吗(A)

A	是
B.	否
分析：A	

185. 下面哪些是对称加密算法(A)

A	DES
B.	MD5
C.	DSA
D.	RSA
分析：A	

186. Map 的实现类中，哪些是有序的，哪些是无序的，有序的是如何保证其有序性，你觉得哪个有序性性能更高，你有没有更好或者更高效的实现方式？

答：

1.Map 的实现类有 HashMap,LinkedHashMap,TreeMap

2.HashMap 是有无序的，LinkedHashMap 和 TreeMap 都是有序的（LinkedHashMap 记录了添加数据的顺序；TreeMap 默认是自然升序）

3. LinkedHashMap 底层存储结构是哈希表+链表，链表记录了添加数据的顺序

4. TreeMap 底层存储结构是二叉树，二叉树的中序遍历保证了数据的有序性

5. LinkedHashMap 有序性能比较高 ,因为底层数据存储结构采用的哈希表

187. 创建 n 多个线程，如何保证这些线程同时启动？看清，是“同时”。

答：

用一个 for 循环创建线程对象并调用 start 方法启动线程。

188. Java 的类加载器都有哪些，每个类加载器都有加载那些类，什么是双亲委派模型，是做什么的？

答：

类加载器按照层次，从顶层到底层，分为以下三种：

（1）启动类加载器（Bootstrap ClassLoader）

这个类加载器负责将存放在 JAVA_HOME/lib 下的，或者被 -Xbootclasspath 参数所指定的路径中的，并且是虚拟机识别的类库加载到虚拟机内存中。启动类加载器无法被 Java 程序直接引用。

（2）扩展类加载器（Extension ClassLoader）

这个加载器负责加载 JAVA_HOME/lib/ext 目录中的，或者被 java.ext.dirs 系统变量所指定的路径中的所有类库，开发者可以直接使用扩展类加载器

（3）应用程序类加载器（Application ClassLoader）

这个加载器是 ClassLoader 中 getSystemClassLoader()方法的返回值，所以一般也称它为系统类加载器。它负责加载用户类路径（Classpath）上所指定的类库，可直接使用这个加载器，如果应用程序没有自定义自己的类加载器，一般情况下这个就是程序中默认的类加载器

双亲委派模型：

双亲委派模型要求除了顶层的启动类加载器外，其他的类加载器都应当有自己的父类加载器。这里类加载器之间的父子关系一般不会以继承关系来实现，而是都使用组合关系来复用父加载器的代码

工作过程：

如果一个类加载器收到了类加载的请求，它首先不会自己去尝试加载这个类，而是把这个请求委派给父类加载器去完成，每一个层次的类加载器都是如此，因此所有的加载请求最终都应该传递到顶层的启动类加载器中，只有当父类加载器反馈自己无法完成这个请求（它的搜索范围中没有找到所需的类）时，子加载器才会尝试自己去加载。

好处：

Java 类随着它的类加载器一起具备了一种带有优先级的层次关系。例如类 Object，它放在 rt.jar 中，无论哪一个类加载器要加载这个类，最终都是委派给启动类加载器进行加载，因此 Object 类在程序的各种类加载器环境中都是同一个类，判断两个类是否相同是通过 classloader.class 这种方式进行的，所以哪怕是同一个 class 文件如果被两个 classloader 加载，那么他们也是不同的类。

加载过程如图：

189. Unsupported major.minor version 52 是什么异常，怎么造成的，如何解决？

答：

问题的根本原因是工程中某个 jar 包的版本（jar 包编译时的所用的 jdk 版本）高于工程 build path 中 jdk 的版本，这个是不兼容的! 编程中遇到此异常 Unsupported major.minor version 52.0（根据版本号，这里可以为其他数值，52 是 1.8jdk jar 包与 1.8 以下低版本 jdk 不匹配），在将 build path 中 jdk 的版本调整与 jar 包匹配后，解决异常。

190. 垃圾回收器（GC）的基本原理是什么？垃圾回收器可以马上回收内存吗？如何通知虚拟机进行垃圾回收？

答：

1、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。

2、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。

3. System.gc();或者 Runtime.getRuntime().gc();

191. 请问以下代码执行会打印出什么？

父类：

```
package com.bjsxt;
public class FatherClass {
    public FatherClass() {
        System.out.println("FatherClassCreate");
    }
}
```

子类：

```
package com.bjsxt;


import com.bjsxt.FatherClass;
public class ChildClass extends FatherClass {
    public ChildClass() {
        System.out.println("ChildClass Create");
    }
    public static void main(String[] args) {
        FatherClass fc = new FatherClass();
        ChildClass cc = new ChildClass();
    }
}
```

执行：C：\>java com.bjsxt.ChildClass

输出结果：？

答：

FatherClassCreate

FatherClassCreate

ChildClass Create

192. 下面的代码在绝大部分时间内都运行得很正常，请问什么情况下会出现问题？根源在哪里？

```
package com.bjsxt;

import java.util.LinkedList;

public class Stack {

    LinkedList list = new LinkedList();

    public synchronized void push(Object x) {

        synchronized (list) {

            list.addLast(x);

            notify();

        }

    }

    public   synchronized Object pop() throws   Exception{

        synchronized(list){

            if(list.size()<=0){

                wait();

            }

            return list.removeLast( );

        }

    }

}
```

答：

将 if(list.size() <= 0)

改成：

while(list.size() <= 0)

193. 同步和异步有何异同，在什么情况下分别使用它们？

答：

1.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。

2.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。

3.举个例子: 打电话是同步 发消息是异步

194. Action 是单实例还是多实例，为什么？

答：

struts2 中 action 是多例的，即一个 session 产生一个 action

背景:

1) Struts2 会对每一个请求,产生一个 Action 的实例来处理.

2) Spring 的 Ioc 容器管理的 bean 默认是单实例的.

首先从数据安全性的问题上考虑，我们的 Action 应该保证是多例的，这样才不会出现数据问题。但是如果有的 action 比如只有 admin 才能操作，或者某些 action，全站公用一个来提高性能，这样的话，就可以使用单例模式。不过幸好，Spring 的 bean 可以针对每一个设置它的 scope，所以，上面的

问题就不是问题了。如果用单例，就在 spring 的 action bean 配置的时候设置 scope=” prototype”

如果是单例的话,若出现两个用户都修改一个对象的属性值,则会因为用户修改时间不同,两个用户访问得到的属性不一样,操作得出的结果不一样.

举个例子:有一块布长度 300cm,能做一件上衣(用掉 100cm)和一件裤子(用掉 200cm);甲和乙同时访问得到的长度都是 300cm,

甲想做上衣和裤子,他先截取 100cm 去做上衣,等上衣做完再去做裤子,而乙这时正好也拿 100cm 去做上衣,那好,等甲做完上衣再做裤子的时候发现剩下的布(100cm)已经不够做裤子了.....这就是影响系统的性能,解决的办法就是给甲和乙一人一块 300cm 的布,就不会出现布被别人偷用的事情,也是就单实例和多实例的区别

如果设置成单例 ,那么多个线程会共享一个 ActionContext 和 ValueStack，这样并发访问的时候就会出现问题了

struts 2 的 Action 是多实例的并非单例，也就是每次请求产生一个 Action 的对象。原因是：struts 2 的 Action 中包含数据，例如你在页面填写的数据就会包含在 Action 的成员变量里面。如果 Action 是单实例的话，这些数据在多线程的环境下就会相互影响，例如造成别人填写的数据被你看到了。所以 Struts2 的 Action 是多例模式的。

问题出现了，可以让 Struts2 的 action 变成单例模式么？

Struts2 中，可以使用注解开发,在 Action 上@Scope(“prototype”）指定为多例 ， 默认为 singleton()单例)

基本上 action 的 scope 需要是 prototype，就是每次请求都建立新的线程

不写的话，默认是 singleton 了

195. 举例说明 JAVA 中如何解析 xml，不同方式有和优缺点？

答：

1. DOM（Document Object Model)

DOM 是用与平台和语言无关的方式表示 XML 文档的官方 W3C 标准。DOM 是以层次结构组织的节点或信息片断的集合。这个层次结构允许开发人员在树中寻找特定信息。分析该结构通常需要加载整个文档和构造层次结构，然后才能做任何工作。由于它是基于信息层次的，因而 DOM 被认为是基于树或基于对象的。

【优点】

①允许应用程序对数据和结构做出更改。

②访问是双向的，可以在任何时候在树中上下导航，获取和操作任意部分的数据。

【缺点】

①通常需要加载整个 XML 文档来构造层次结构，消耗资源大。

2. SAX（Simple API for XML)

SAX 处理的优点非常类似于流媒体的优点。分析能够立即开始，而不是等待所有的数据被处理。而且，由于应用程序只是在读取数据时检查数据，因此不需要将数据存储在内存中。这对于大型文档来说是个巨大的优点。事实上，应用程序甚至不必解析整个文档；它可以在某个条件得到满足时停止解析。一般来说，SAX 还比它的替代者 DOM 快许多。

选择 DOM 还是选择 SAX？ 对于需要自己编写代码来处理 XML 文档的开

发人员来说， 选择 DOM 还是 SAX 解析模型是一个非常重要的设计决策。DOM 采用建立树形结构的方式访问 XML 文档 ,而 SAX 采用的是事件模型。DOM 解析器把 XML 文档转化为一个包含其内容的树，并可以对树进行遍历。用 DOM 解析模型的优点是编程容易 ,开发人员只需要调用建树的指令，然后利用 navigation APIs 访问所需的树节点来完成任务。可以很容易的添加和修改树中的元素。然而由于使用 DOM 解析器的时候需要处理整个 XML 文档，所以对性能和内存的要求比较高，尤其是遇到很大的 XML 文件的时候。由于它的遍历能力，DOM 解析器常用于 XML 文档需要频繁的改变的服务中。

SAX 解析器采用了基于事件的模型，它在解析 XML 文档的时候可以触发一系列的事件，当发现给定的 tag 的时候，它可以激活一个回调方法，告诉该方法制定的标签已经找到。SAX 对内存的要求通常会比较低，因为它让开发人员自己来决定所要处理的 tag.特别是当开发人员只需要处理文档中所包含的部分数据时，SAX 这种扩展能力得到了更好的体现。但用 SAX 解析器的时候编码工作会比较困难，而且很难同时访问同一个文档中的多处不同数据。

【优势】

①不需要等待所有数据都被处理，分析就能立即开始。

②只在读取数据时检查数据，不需要保存在内存中。

③可以在某个条件得到满足时停止解析，不必解析整个文档。

④效率和性能较高，能解析大于系统内存的文档。

【缺点】

①需要应用程序自己负责 TAG 的处理逻辑（例如维护父/子关系等），文档越复杂程序就越复杂。

②单向导航，无法定位文档层次，很难同时访问同一文档的不同部分数据，不支持 XPath。

3. JDOM(Java-based Document Object Model)

JDOM 的目的是成为 Java 特定文档模型，它简化与 XML 的交互并且比使用 DOM 实现更快。由于是第一个 Java 特定模型，JDOM 一直得到大力推广和促进。正在考虑通过“Java 规范请求 JSR-102”将它最终用作“Java 标准扩展”。从 2000 年初就已经开始了 JDOM 开发。

JDOM 与 DOM 主要有两方面不同。首先，JDOM 仅使用具体类而不使用接口。这在某些方面简化了 API，但是也限制了灵活性。第二，API 大量使用了 Collections 类，简化了那些已经熟悉这些类的 Java 开发者的使用。

JDOM 文档声明其目的是“使用 20%（或更少）的精力解决 80%（或更多）Java/XML 问题”（根据学习曲线假定为 20%）。JDOM 对于大多数 Java/XML 应用程序来说当然是有用的，并且大多数开发者发现 API 比 DOM 容易理解得多。JDOM 还包括对程序行为的相当广泛检查以防止用户做任何在 XML 中无意义的事。然而，它仍需要您充分理解 XML 以便做一些超出基本的工作（或者甚至理解某些情况下的错误）。这也许是比学习 DOM 或 JDOM 接口都更有意义的工作。

JDOM 自身不包含解析器。它通常使用 SAX2 解析器来解析和验证输入 XML 文档（尽管它还可以将以前构造的 DOM 表示作为输入）。它包含一些转换器以将 JDOM 表示输出成 SAX2 事件流、DOM 模型或 XML 文本文档。

JDOM 是在 Apache 许可证变体下发布的开放源码。

【优点】

①使用具体类而不是接口，简化了 DOM 的 API。

②大量使用了 Java 集合类，方便了 Java 开发人员。

【缺点】

①没有较好的灵活性。

②性能较差。

4. DOM4J(Document Object Model for Java)

虽然 DOM4J 代表了完全独立的开发结果，但最初，它是 JDOM 的一种智能分支。它合并了许多超出基本 XML 文档表示的功能，包括集成的 XPath 支持、XML Schema 支持以及用于大文档或流化文档的基于事件的处理。它还提供了构建文档表示的选项，它通过 DOM4J API 和标准 DOM 接口具有并行访问功能。从 2000 下半年开始，它就一直处于开发之中。

为支持所有这些功能，DOM4J 使用接口和抽象基本类方法。DOM4J 大量使用了 API 中的 Collections 类，但是在许多情况下，它还提供一些替代方法以允许更好的性能或更直接的编码方法。直接好处是，虽然 DOM4J 付出了更复杂的 API 的代价，但是它提供了比 JDOM 大得多的灵活性。

在添加灵活性、XPath 集成和对大文档处理的目标时，DOM4J 的目标与 JDOM 是一样的：针对 Java 开发者的易用性和直观操作。它还致力于成为比 JDOM 更完整的解决方案，实现在本质上处理所有 Java/XML 问题的目标。在完成该目标时，它比 JDOM 更少强调防止不正确的应用程序行为。

DOM4J 是一个非常非常优秀的 Java XML API，具有性能优异、功能强大

和极端易用使用的特点，同时它也是一个开放源代码的软件。如今你可以看到越来越多的 Java 软件都在使用 DOM4J 来读写 XML，特别值得一提的是连 Sun 的 JAXM 也在用 DOM4J.

【优点】

①大量使用了 Java 集合类，方便 Java 开发人员，同时提供一些提高性能的替代方法。

②支持 XPath。

③有很好的性能。

【缺点】

①大量使用了接口，API 较为复杂。

二、比较

1. DOM4J 性能最好，连 Sun 的 JAXM 也在用 DOM4J。目前许多开源项目中大量采用 DOM4J ,例如大名鼎鼎的 Hibernate 也用 DOM4J 来读取 XML 配置文件。如果不考虑可移植性，那就采用 DOM4J.

2. JDOM 和 DOM 在性能测试时表现不佳，在测试 10M 文档时内存溢出，但可移植。在小文档情况下还值得考虑使用 DOM 和 JDOM.虽然 JDOM 的开发者已经说明他们期望在正式发行版前专注性能问题，但是从性能观点来看，它确实没有值得推荐之处。另外，DOM 仍是一个非常好的选择。DOM 实现广泛应用于多种编程语言。它还是许多其它与 XML 相关的标准的基础，因为它正式获得 W3C 推荐（与基于非标准的 Java 模型相对），所以在某些类型的项目中可能也需要它（如在 JavaScript 中使用 DOM）。

3. SAX 表现较好，这要依赖于它特定的解析方式 - 事件驱动。一个 SAX 检

测即将到来的 XML 流，但并没有载入到内存（当然当 XML 流被读入时，会有部分文档暂时隐藏在内存中）。

我的看法：如果 XML 文档较大且不考虑移植性问题建议采用 DOM4J；如果 XML 文档较小则建议采用 JDOM；如果需要及时处理而不需要保存数据则考虑 SAX。但无论如何，还是那句话：适合自己的才是最好的，如果时间允许，建议大家讲这四种方法都尝试一遍然后选择一种适合自己的即可。

196. char 型变量中能不能存储一个中文汉字？

答：

1.java 采用 unicode 编码，2 个字节（16 位）来表示一个字符， 无论是汉字还是数字，字母，或其他语言都可以存储。

2.char 在 java 中是 2 个字节，所以可以存储中文

197. Java 线程中，sleep()和 wait()区别

答：

sleep 是线程类(Thread)的方法；作用是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，准备获得对象锁进行运行状态。

198. 如果有两个类 A、B（注意不是接口），你想同时使用这两个类的功能，那么你会如何编写这个 C 类呢？

答：

因为类 A、B 不是接口，所以是不可以直接实现的，但可以将 A、B 类定义成父子类，那么 C 类就能实现 A、B 类的功能了。假如 A 为 B 的父类，B 为 C 的父类，此时 C 就能使用 A、B 的功能。

199. 一个类的构造方法是否可以被重载（overloading），是否可以被子类重写（overrding）？

答：

构造方法可以被重载，但是构造方法不能被重写，子类也不能继承到父类的构造方法

200. TreeMap 和 TreeSet 在排序时如何比较元素？Collections 工具类中的 sort（ ）方法如何比较元素？

答：

TreeSet 要求存放的对象所属的类必须实现 Comparable 接口 ,该接口提供了比较元素的 compareTo()方法，当插入元素时会 回调该方法比较元素的大小。TreeMap 要求存放的键值对映射的键必须实现 Comparable 接口从而根据键对元素进行排序。Collections 工具类的 sort 方法有两种重载的形式，第一种要求传入的待排序容器中存放的对象比较实现 Comparable 接口以实现元素的比较 ;第二种不强制性的要求容器中的元素必须可比较，但是要求传入第二个参数，参数是 Comparator 接口的子类型（需要重写 compare 方法实现元素的比较），相当于一个临时定义的排序规则，其实就是是通过接口注入比较元素大小的算法，也是对回调模式的应用。

201. Java 中 byte 表示的数值范围是什么？

答：

范围是-128 至 127

202. 如何将日期类型格式化为：2013-02-18 10:53:10？

```
public class TestDateFormat2 {

    public static void main(String[] args) throws Exception {

        //第一步：将字符串（2013-02-18 10:53:10）转换成日期Date

        DateFormat  sdf=new SimpleDateFormat("yyyy-MM-dd
hh:mm:ss");

        String sdate="2013-02-18 10:53:10";

        Date date=sdf.parse(sdate);

        System.out.println(date);


        //第二步：将日期Date转换成字符串String

        DateFormat  sdf2=new SimpleDateFormat("yyyy-MM-dd
hh:mm:ss");

        String sdate2=sdf2.format(date);

        System.out.println(sdate2);

    }

}
```

203. List 里面如何剔除相同的对象？

```
public class Test {

    public static void main(String[] args) {

        List<String> li1 = new ArrayList<String>();

        li1.add("8");

        li1.add("8");

        li1.add("9");

        li1.add("9");

        li1.add("0");

        System.out.println(li1);

        //方法:将List中数据取出来来存到Set中

        HashSet<String> set = new HashSet<String>();

        for(int i=0;i<li1.size();i++){

            set.add(li1.get(i));

        }

        System.out.println(set);

    }

}
```

204. 有两个字符串：目标串 S=“s1s2.....sn”，模式串 T="t1t2.....tm"。若存在 T 的每个字符一次和 S 中的一个连续字符序列相等，则匹配成功，返回 T 中第一个字符在 S 中的位置。否则匹配不成功，返回 0。写出你的算法，要求线性时间复杂度

答：

字符串匹配操作定义：

目标串 S="S0S1S2...Sn-1"，模式串 T=“T0T1T2...Tm-1”

对合法位置 0<= i <= n-m （i 称为位移）依次将目标串的字串 S[i ... i+m-1] 和模式串 T[0 ... m-1] 进行比较，若：

1、S[i ... i+m-1] = T[0 ... m-1] ，则从位置 i 开始匹配成功，称模式串 T 在目标串 S 中出现。

2、S[i ... i+m-1] != T[0 ... m-1] ，则从位置 i 开始匹配失败。

字符串匹配算法 —— Brute-Force 算法

字符串匹配过程中，对于位移 i （i 在目标串中的某个位置），当第一次 Sk != Tj 时，i 向后移动 1 位 ， 及 i = i+1，此时 k 退回到 i+1 位置；模式串要退回到第一个字符。该算法时间复杂度 O（M*N），但是实际情况中时间复杂度接近于 O（M + N），以下为 Brute-Force 算法的 Java 实现版本：

```
public static int bruteForce(String target, String pattern, int pos) {
        if (target == null || pattern == null) {
            return -1;
        }
```

```
        int k = pos - 1, j = 0, tLen = target.length(), pLen =
pattern.length();
        while (k < tLen && j < pLen) {
            if (target.charAt(k) == pattern.charAt(j)) {
                j++;
                k++;
            } else {
                k = k - j + 1;
                j = 0;
            }
        }
        if (j == pLen) {
            return k - j + 1;
        }
        return -1;
    }
```

205. 写一个方法，实现字符串的反转，如：输入 abc，输出 cba

```
public class Test {
    public static void main(String[] args) {
        String result=reverse("abc");
        System.out.println(result);
    }
```

```
    public static String reverse(String str){
        StringBuilder result=new StringBuilder("");
        char[] chArra=str.toCharArray();
        for(int i=chArra.length-1;i>=0;i--){
            char ch=chArra[i];
            result.append(ch);
        }
        return result.toString();
    }
}
```

206. 字符串如何转换为 int 类型

```
public class Test {
    public static void main(String[] args) {
        //方式一
        int num=Integer.parseInt("123");
        //方式二
        int num2=Integer.valueOf("123");
        System.out.println(num+"  "+num2);
    }
}
```

207. 如何生成一个 0-100 的随机整数？

```
public class Test {
```

```
    public static void main(String[] args) {
        int num=(int)(Math.random()*101);
        System.out.println(num);
    }
}
```

208. 不通过构造函数也能创建对象吗(A)

A.	是
B.	否

分析：答案：A

Java 创建对象的几种方式（重要）：

(1) 用 new 语句创建对象，这是最常见的创建对象的方法。

(2) 运用反射手段，调用 java.lang.Class 或者 java.lang.reflect.Constructor 类的 newInstance()实例方法。

(3) 调用对象的 clone()方法。

(4) 运用反序列化手段，调用 java.io.ObjectInputStream 对象的 readObject()方法。

(1)和(2)都会明确的显式的调用构造函数 ；(3)是在内存上对已有对象的影印，所以不会调用构造函数 ；(4)是从文件中还原类的对象，也不会调用构造函数。

209. 下面哪些是对称加密算法(A)

A.	DES

B.	MD5
C.	DSA
D.	RSA

分析：答案：A

解析：常用的对称加密算法有：DES、3DES、RC2、RC4、AES

常用的非对称加密算法有：RSA、DSA、ECC

使用单向散列函数的加密算法：MD5、SHA

210. 下面的代码段，当输入为 2 的时候返回值是（C）

```
publicstaticint get Value(int i){
    int result=0;
    switch(i){
    case 1:
        result=result +i
    case 2:
        result=result+i*2
    case 3:
        result=result+i*3
    }
    return result;
}
```

A	0
B.	2
C.	4
D.	10
分析：result = 0 + 2 * 2;	

211. 以下哪个是服务（C）

A.	kill
B.	tar
C.	rsyne
D.	lsof
分析：答案:c A:kill 命令,常用于杀死进程; B:tar 命令,tar 命令是 Unix/Linux 系统中备份文件的可靠方法 ,几乎可以工作于任何环境中，它的使用权限是所有用户 C:类 unix 系统下的数据镜像备份工具 D:在终端下输入 lsof 即可显示系统打开的文件 ,因为 lsof 需要访问核心内存和各种文件，所以必须以 root 用户的身份运行它才能够充分地发挥其功能	

212. 以下可以实现负载均衡的是（C）

A.	nagios
B.	Jenkins
C.	nginx
D.	docker
分析：答案: C Nginx 是一款轻量级的 Web 服务器/反向代理服务器及电子邮件(IMAP/POP3)代理服务器，并在一个 BSD-like 协议下发行。其特点是占有内存少，并发能力强，事实上 nginx 的并发能力确实在同类型的网页服务器中表现较好，中国大陆使用 nginx 网站用户有：百度、京东、新浪、网易、腾讯、淘宝等	

213. 在最佳情况下，以下哪个时间复杂度最高（D）

A.	直接插入排序
B.	直接选择排序
C.	冒泡排序
D.	归并排序

分析：答案: D

排序方法	最坏时间复杂度	最好时间复杂度	平均时间复杂度
直接插入	O(n2)	O(n)	O(n2)
简单选择	O(n2)	O(n2)	O(n2)
冒泡排序	O(n2)	O(n)	O(n2)

快速排序	O(n2)	O(nlog2n)	O(nlog2n)
堆排序	O(nlog2n)	O(nlog2n)	O(nlog2n)
归并排序	O(nlog2n)	O(nlog2n)	O(nlog2n)

214. 以下 Java 代码段会产生几个对象

```
publicvoid test(){
    String a="a";
    String b="b";
    String c="c";
    c=a+""+b+""+c;
    System.out.print(c);
}
```

分析：答案: 一个对象，因为编译期进行了优化，3 个字符串常量直接折叠为一个

215. Math.round（-7.2）的运行结果是。

分析：答案: -11

小数点后第一位=5

正数：Math.round(11.5)=12

负数：Math.round(-11.5)=-11

小数点后第一位<5

正数：Math.round(11.46)=11

负数：Math.round(-11.46)=-11

小数点后第一位>5

正数：Math.round(11.68)=12

负数：Math.round(-11.68)=-12

根据上面例子的运行结果，我们还可以按照如下方式总结，或许更加容易记忆：

参数的小数点后第一位<5，运算结果为参数整数部分。

参数的小数点后第一位>5，运算结果为参数整数部分绝对值+1，符号（即正负）不变。

参数的小数点后第一位=5，正数运算结果为整数部分+1，负数运算结果为整数部分。

终结：大于五全部加，等于五正数加，小于五全不加。

216. 请编写一段 Java 程序将两个有序数组合并成一个有序数组

```
import java.util.Arrays;


publicclass Demo1 {


    publicstaticvoid main(String[] args) {
```

```
        int[] a = { 1, 2, 3, 4, 5, 7, 8, 9, 10 };
        int[] b = { 3, 5, 7, 9, 10 };
        int[] target = newint[a.length + b.length];

        for (inti = 0; i<a.length; i++)
            target[i] = a[i];
        for (intj = 0; j<b.length; j++)
            target[a.length + j] = b[j];

        Arrays.sort(target);
        for (inti = 0; i<target.length; i++)
            System.out.println(target[i]);

    }
}
```

217. 十进制数 278 的对应十六进制数

分析：十进制数 278 的对应十六进制数是 116

218. 线程的状态

分析：线程从创建、运行到结束总是处于下面五个状态之一：新建状态、就绪状态、运行状态、阻塞状态及死亡状态。

219. 常见的 RuntimeException

分析：NullPointerException：一般都是在 null 对象上调用方法了。

NumberFormatException：继承 IllegalArgumentException，字符串转换为数字时。

比如 int i= Integer.parseInt("ab3");

ArrayIndexOutOfBoundsException:数组越界

比如 int[] a=new int[3]; int b=a[3];

StringIndexOutOfBoundsException：字符串越界

比如 String s="hello"; char c=s.chatAt(6);

ClassCastException:类型转换错误

比如 Object obj=new Object(); String s=(String)obj;

220. Java.util.Map 的实现类有

分析：Java 中的 java.util.Map 的实现类

1、HashMap

2、Hashtable

3、LinkedHashMap

4、TreeMap

221. Java 中 int.long 占用的字节数分别是

分析：1："字节" 是 byte，"位" 是 bit ；

2： 1 byte = 8 bit ；

char 在 Java 中是 2 个字节。java 采用 unicode，2 个字节（16 位）来表示一个字符。

short 2 个字节

int 4 个字节

long 8 个字节

222. System.out.println(‘a’ +1);的结果是

分析：'a'是 char 型，1 是 int 行，int 与 char 相加，char 会被强转为 int 行，char 的 ASCII 码对应的值是 97，所以加一起打印 98

223. 编写 java，将“I follow Bill Gate.Tom Gate.John Gate”中的“Gate”全部替换为“Gates”

```
public classDemo1 {
    publicstaticvoid main(String[] args) {
        String s="I follow Bill Gate.Tom Gate.John Gate";
        System.out.println(s);
        s=s.replaceAll("Gate","Gates");
        System.out.println(s);
    }
}
```

224. 下列语句那一个正确（B）

A	java 程序经编译后会产生 machine code
B.	java 程序经编译后会产生 byte code

C.	java 程序经编译后会产生 DLL
D.	以上都不正确
分析：:B　java 程序编译后会生成字节码文件,就是.class 文件	

225. 下列说法正确的有（C）

A	class 中的 constructor 不可省略
B.	constructor 必须与 class 同名，但方法不能与 class 同名
C.	constructor 在一个对象被 new 时执行
D.	一个 class 只能定义一个 constructor
分析：C	

226. 下列运算符合法的是（B）

A	&&
B.	<>
C.	if
D.	=
分析：B &&　与运算	

227. 执行如下程序代码(C)

```
a=0；c=0；
 do{
 ——c；
 a=a-1；
```

} while (a > 0) ; 后，c的值是（ ）	
A	0
B.	1
C.	-1
D.	死循环
分析：C　do{...}while(...);语句至少执行一次	

228. 下列哪一种叙述是正确的（D）

A	abstract 修饰符可修饰字段、方法和类
B.	抽象方法的 body 部分必须用一对大括号 { } 包住
C.	声明抽象方法，大括号可有可无
D.	声明抽象方法不可写出大括号
分析：D abstract 不能修饰字段。既然是抽象方法，当然是没有实现的方法，根本就没有 body 部分。	

229. 下列语句正确的是（A）

A	形式参数可被视为 local variable
B.	形式参数可被字段修饰符修饰
C.	形式参数为方法被调用时，真正被传递的参数
D.	形式参数不可以是对象

分析：答案 A：

A：形式参数可被视为 local variable。形参和局部变量一样都不能离开方法。都只有在方法内才会发生作用，也只有在方法中使用，不会在方法外可见。

B：对于形式参数只能用 final 修饰符，其它任何修饰符都会引起编译器错误。但是用这个修饰符也有一定的限制，就是在方法中不能对参数做任何修改。 不过一般情况下，一个方法的形参不用 final 修饰。只有在特殊情况下，那就是：方法内部类。 一个方法内的内部类如果使用了这个方法的参数或者局部变量的话，这个参数或局部变量应该是 final。

C：形参的值在调用时根据调用者更改，实参则用自身的值更改形参的值（指针、引用皆在此列），也就是说真正被传递的是实参。

D：方法的参数列表指定要传递给方法什么样的信息，采用的都是对象的形式。因此 ,在参数列表中必须指定每个所传递对象的类型及名字。想 JAVA 中任何传递对象的场合一样，这里传递的实际上也是引用，并且引用的类型必须正确。--《Thinking in JAVA》

230. 一个数组，元素为从 0 到 m 的整数，判断其中是否有重复元素，使用 java 语言编写一个方法

```
publicstaticboolean demo(int[] arr){
        for (inti = 0; i<arr.length; i++) {
            for (intj = i + 1; j<arr.length; j++) {
```

```
                if (arr[i] == arr[j]) {
                    returntrue;
                }
            }
        }
        returnfalse;
    }
```

231. 写一个方法，实现字符串的反转，如：输入 abc，输出 cba

```
publicstatic String reverse(String s) {
        intlength = s.length();
        StringBuffer result = new StringBuffer(length);
        for (inti = length - 1; i>= 0; i--)
            result.append(s.charAt(i));
        returnresult.toString();
    }
```

232. 成员变量用 static 修饰和不用 static 修饰有什么区别？

i. 1，两个变量的生命周期不同。

成员变量随着对象的创建而存在，随着对象的被回收而释放。

静态变量随着类的加载而存在，随着类的消失而消失。

2，调用方式不同。

成员变量只能被对象调用。

静态变量可以被对象调用，还可以被类名调用。

对象调用：p.country

类名调用 ：Person.country

3，别名不同。

成员变量也称为实例变量。

静态变量称为类变量。

4，数据存储位置不同。

成员变量数据存储在堆内存的对象中，所以也叫对象的特有数据.

静态变量数据存储在**方法区**(共享数据区)的静态区，所以也叫对象的共享数据.

233. 如果变量用 final 修饰，则怎样？如果方法 final 修饰，则怎样？

1、用 final 修饰的类不能被扩展，也就是说不可能有子类；

2、用 final 修饰的方法不能被替换或隐藏：

①使用 final 修饰的实例方法在其所属类的子类中不能被替换（overridden）;

②使用 final 修饰的静态方法在其所属类的子类中不能被重定义（redefined）而隐藏 （hidden）;

3、用 final 修饰的变量最多只能赋值一次，在赋值方式上不同类型的变量或稍有不同：

①静态变量必须明确赋值一次（不能只使用类型缺省值）;作为类成员

的静态变量， 赋值可以在其声明

中通过初始化表达式完成，也可以在静态初始化块中进行；作为接口成员的静态变量， 赋值只能在其

声明中通过初始化表达式完成；

②实例变量同样必须明确赋值一次（不能只使用类型缺省值）；赋值可以在其声明中通 过初始化表达式

完成，也可以在实例初始化块或构造器中进行；

③方法参数变量在方法被调用时创建，同时被初始化为对应实参值，终止于方法体（body）结束，在此

期间其值不能改变；

④构造器参数变量在构造器被调用（通过实例创建表达式或显示的构造器调用）时创建， 同时被初始化

为对应实参值，终止于构造器体结束，在此期间其值不能改变；

⑤异常处理器参数变量在有异常被 try 语句的 catch 子句捕捉到时创建，同时被初始化 为实际的异常对象

，终止于 catch 语句块结束，在此期间其值不能改变；

⑥局部变量在其值被访问之前必须被明确赋值；

234. 下面所述步骤中，是创建进程做必须的步骤是（BC）

A	由调度程序为进程分配 CPU
B.	建立一个进程控制块
C.	为进程分配内存
D.	为进程分配文件描述符

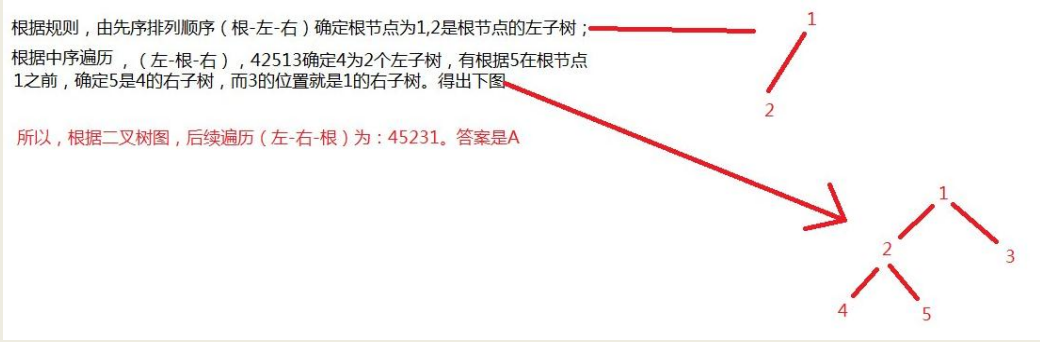
分析：BC

235. 下列叙述中正确的是（D）

A	循环队列有队头和队尾两个指针，因此，循环队列是非线性结构
B.	在循环队列中 ,只需要队头指针就能反映队列中元素的动态变化情况
C.	在循环队列中 ,只需要队尾指针就能反映队列中元素的动态变化情况
D.	在循环队列中元素的个数是由队头指针和队尾指针共同决定的
分析：正确答案：D 循环队列中元素的个数是由队首指针和队尾指针共同决定的，元素的动态变化也是通过队首指针和队尾指针来反映的，当队首等于队尾时，队列为空。	

236. 某二叉树的先序遍历是 12453，中序遍历是 42513，那么其后序遍历是(A)

A	45231
B.	42351
C.	12345
D.	54321



237. 无锁化编程有哪些常见方法？

A	针对计数器，可以使用原子加
B.	只有一个生产者和一个消费者 ,那么就可以做到免锁访问环形缓冲区（Ring Buffer）
C.	RCU（Read-Copy-Update），新旧副本切换机制，对于旧副本可以采用延迟释放的做法
D.	CAS（Compare-and-Swap），如无锁栈，无锁队列等待

分析：答案：D。

A 这方法虽然不太好，但是常见

B ProducerConsumerQueue就是这个，到处都是

C linux kernel里面大量使用

D 本质上其实就是乐观锁，操作起来很困难。。单生产者多消费者或者多生产者单消费者的情况下比较常见，也不容易遇到 ABA 问题。

238. 在 N 个乱序数字中查找第 k 大的数字，时间复杂度可以减小至(B)

A	O(N*logN)
B.	O(N)
C.	O(1)

D.	O(2)
分析：答案：B 实现思路：利用 hash 保存数组中元素 Si 出现的次数，利用计数排序的思想，线性从大到小扫描过程中，前面有 k-1 个数则为第 k 大数，平均情况下时间复杂度 O(n)	

239. 有一台带一个千兆网卡的服务器 A，会把接收到的消息转发给另外两台带一个千兆网卡的服务器 B 和 C，B 和 C 上面的一个服务进程处理一条 10K 字节的消息需要 2 毫秒。如果在 B 和 C 上面各跑 80 个服务进程，在不考虑 CPU 负载和进程切换、内存占用、传输损耗和交互损耗的情况下，B 和 C 服务器每秒一共大约可以处理_____条 10K 字节的消息

A	12500
B.	60000
C.	70000
D.	80000
分析：答案：A 1000Mbps=1000000Kbps=1000000/8 KBps = 125000KBps，也就是每秒 125000KB，最多发送 12500 条。	

240. 以下指令集架构属于复杂指令集架构的是

A.	ARM
B.	MIPS

C.	SPARC
D.	以上皆不是
分析：	

241. 在二进制数据中，小数点向右移一位，则数据

A.	除以 10
B.	除以 2
C.	乘以 2
D.	乘以 10
分析：可以看个例子 101.1 对应的十进制为 2^2*1 + 2^1*0 + 2^0*1 + 2^-1*1 = 5.5 小数点右移一位 1011 对应的十进制为 2^3*1 + 2^2*0 + 2^1*1 + 2^0*1 = 11 所以是扩大到原来的2倍	

242. 设一颗二叉树中有 3 个叶子节点，有八个度为 1 的节点，则该二叉树中总的节点数为

A	12
B.	13
C.	14
D.	15
分析：选 b 子叶节点是度为零的节点,而二叉树的性质可知,度是 0 的节点	

比度是 2 的节点数多 1 个,所以度是 2 的节点为 2 个,所以共有 3+8+2=13

243. 在一个元素个数为 N 的数组里，找到升序排在 N/5 位置的元素的最优算法时间复杂度是

A	O(n)
B.	O(n log n)
C.	O(n (log n)2)
D.	O(n 3/2)
分析：	

244. 下列叙述中正确的是

A	循环队列有队头和队尾两个指针，因此，循环队列是非线性结构
B.	在循环队列中 ,只需要队头指针就能反映队列中元素的动态变化情况
C.	在循环队列中 ,只需要队尾指针就能反映队列中元素的动态变化情况
D.	在循环队列中元素的个数是由队头指针和队尾指针共同决定的
分析：循环队列中元素的个数是由队首指针和队尾指针共同决定的，元素的动态变化也是通过队首指针和队尾指针来反映的，当队首等于队尾时，队列为空。	

245. 面向对象的特征有哪些方面?

答：面向对象的特征主要有以下几个方面：

1、抽象：抽象是将一类对象的共同特征总结出来构造类的过程，包括数据抽象和行为抽象两方面。抽象只关注对象有哪些属性和行为，并不关注这些行为的细节是什么。

2、继承：继承是从已有类得到继承信息创建新类的过程。提供继承信息的类被称为父类（超类、基类）；得到继承信息的类被称为子类（派生类）。继承让变化中的软件系统有了一定的延续性，同时继承也是封装程序中可变因素的重要手段（如果不能理解请阅读阎宏博士的《Java 与模式》或《设计模式精解》中关于桥梁模式的部分）。

3、封装：通常认为封装是把数据和操作数据的方法绑定起来，对数据的访问只能通过已定义的接口。面向对象的本质就是将现实世界描绘成一系列完全自治、封闭的对象。我们在类中编写的方法就是对实现细节的一种封装；我们编写一个类就是对数据和数据操作的封装。可以说，封装就是隐藏一切可隐藏的东西，只向外界提供最简单的编程接口（可以想想普通洗衣机和全自动洗衣机的差别，明显全自动洗衣机封装更好因此操作起来更简单；我们现在使用的智能手机也是封装得足够好的，因为几个按键就搞定了所有的事情）。

4、多态性：多态性是指允许不同子类型的对象对同一消息作出不同的响应。简单的说就是用同样的对象引用调用同样的方法但是做了不同的事情。多态性分为编译时的多态性和运行时的多态性。如果将对象的方法视为对象向外界提供的服务，那么运行时的多态性可以解释为：当 A 系统访问 B 系统提供的服务时，B 系统有多种提供服务的方式，但一切对 A 系统来说都是透明的（就像电动剃须刀是 A 系统，它的供电系统是 B 系统，B 系统可以使用电池

供电或者用交流电，甚至还有可能是太阳能，A 系统只会通过 B 类对象调用供电的方法，但并不知道供电系统的底层实现是什么，究竟通过何种方式获得了动力）。方法重载（overload）实现的是编译时的多态性（也称为前绑定），而方法重写（override）实现的是运行时的多态性（也称为后绑定）。运行时的多态是面向对象最精髓的东西，要实现多态需要做两件事：1. 方法重写（子类继承父类并重写父类中已有的或抽象的方法）；2. 对象造型（用父类型引用引用子类型对象，这样同样的引用调用同样的方法就会根据子类对象的不同而表现出不同的行为）。

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

答：区别如下：

作用域	当前类	同包	子类	其他
public	√	√	√	√
protected	√	√	√	×
default	√	√	×	×
private	√	×	×	×

类的成员不写访问修饰时默认为 default。默认对于同一个包中的其他类相当于公开（public），对于不是同一个包中的其他类相当于私有（private）。受保护（protected）对子类相当于公开，对不是同一包中的没有父子关系的类相当于私有。

247. String 是最基本的数据类型吗?

答 :不是。Java 中的基本数据类型只有 8 个 :byte、short、int、long、float、double、char、boolean；除了基本类型（primitive type）和枚举类型（enumeration type），剩下的都是引用类型（reference type）。

248. float f=3.4;是否正确?

答:不正确。3.4 是双精度数，将双精度型（double）赋值给浮点型（float）属于下转型（down-casting，也称为窄化）会造成精度损失，因此需要强制类型转换 float f =(float)3.4; 或者写成 **float f =3.4F**;。

249. short s1 = 1; s1 = s1 + 1;有错吗?short s1 = 1; s1 += 1;有错吗?

答：对于 short s1 = 1; s1 = s1 + 1;由于 1 是 int 类型，因此 s1+1 运算结果也是 int 型，需要强制转换类型才能赋值给 short 型。而 short s1 = 1; s1 += 1;可以正确编译，因为 s1+= 1;相当于 s1 = (short)(s1 + 1);其中有隐含的强制类型转换。

250. Java 有没有 goto?

答 :goto 是 Java 中的保留字，在目前版本的 Java 中没有使用。(根据 James Gosling（Java 之父）编写的《The Java Programming Language》一书的附录中给出了一个 Java 关键字列表，其中有 goto 和 const，但是这两个是目前无法使用的关键字，因此有些地方将其称之为保留字，其实保留字这个词应该有更广泛的意义，因为熟悉 C 语言的程序员都知道，在系统类库中使用过的有特殊意义的单词或单词的组合都被视为保留字）

251. int 和 Integer 有什么区别?

答：Java 是一个近乎纯洁的面向对象编程语言，但是为了编程的方便还是引入不是对象的基本数据类型，但是为了能够将这些基本数据类型当成对象操作，Java 为每一个基本数据类型都引入了对应的包装类型(wrapper class)，int 的包装类就是 Integer，从 JDK 1.5 开始引入了自动装箱/拆箱机制，使得二者可以相互转换。

Java 为每个原始类型提供了包装类型：

原始类型: boolean，char，byte，short，int，long，float，double

包装类型：Boolean，Character，Byte，Short，Integer，Long，Float，Double

```
package com.bjsxt;


public class AutoUnboxingTest {


    public static void main(String[] args) {
        Integer a = new Integer(3);
        Integer b = 3;                  // 将 3 自动装箱成 Integer 类型
        int c = 3;
        System.out.println(a == b); // false 两个引用没有引用同一对象
            System.out.println(a == c); // true a 自动拆箱成 int 类型再和 c
比较
        }
```

```
}
```

补充：最近还遇到一个面试题，也是和自动装箱和拆箱相关的，代码如下所示：

```
public class Test03 {


    public static void main(String[] args) {
        Integer f1 = 100, f2 = 100, f3 = 150, f4 = 150;
        System.out.println(f1 == f2);
        System.out.println(f3 == f4);
    }
}
```

如果不明就里很容易认为两个输出要么都是 true 要么都是 false。首先需要注意的是 f1、f2、f3、f4 四个变量都是 Integer 对象，所以下面的==运算比较的不是值而是引用。装箱的本质是什么呢？当我们给一个 Integer 对象赋一个 int 值的时候，会调用 Integer 类的静态方法 valueOf，如果看看 valueOf 的源代码就知道发生了什么。

```
public static Integer valueOf(int i) {
        if (i >= IntegerCache.low && i <= IntegerCache.high)
            return IntegerCache.cache[i + (-IntegerCache.low)];
        return new Integer(i);
```

```
    }
```

IntegerCache 是 Integer 的内部类，其代码如下所示：

```
/* Cache to support the object identity semantics of autoboxing for values between
     * -128 and 127 (inclusive) as required by JLS.
     *
     * The cache is initialized on first usage.  The size of the cache
     * may be controlled by the {@code -XX:AutoBoxCacheMax=<size>} option.
     * During VM initialization, java.lang.Integer.IntegerCache.high property
     * may be set and saved in the private system properties in the
     * sun.misc.VM class.
     */

    private static class IntegerCache {
        static final int low = -128;
        static final int high;
        static final Integer cache[];
```

```
        static {
            // high value may be configured by property
            int h = 127;
            String integerCacheHighPropValue =

sun.misc.VM.getSavedProperty("java.lang.Integer.IntegerCache.high");
            if (integerCacheHighPropValue != null) {
                try {
                    int i = parseInt(integerCacheHighPropValue);
                    i = Math.max(i, 127);
                    // Maximum array size is Integer.MAX_VALUE
                    h = Math.min(i, Integer.MAX_VALUE - (-low) -1);
                } catch( NumberFormatException nfe) {
                    // If the property cannot be parsed into an int,
ignore it.
                }
            }
            high = h;

            cache = new Integer[(high - low) + 1];
            int j = low;
            for(int k = 0; k < cache.length; k++)
```

```
                cache[k] = new Integer(j++);


            // range [-128, 127] must be interned (JLS7 5.1.7)
            assert IntegerCache.high >= 127;
        }

        private IntegerCache() {}
}
```

简单的说，如果字面量的值在-128 到 127 之间，那么不会 new 新的 Integer 对象，而是直接引用常量池中的 Integer 对象，所以上面的面试题中 f1==f2 的结果是 true，而 f3==f4 的结果是 false。越是貌似简单的面试题其中的玄机就越多，需要面试者有相当深厚的功力。

252. &和&&的区别？

答：&运算符有两种用法：(1)按位与；(2)逻辑与。&&运算符是短路与运算。逻辑与跟短路与的差别是非常巨大的，虽然二者都要求运算符左右两端的布尔值都是 true 整个表达式的值才是 true。&&之所以称为短路运算是因为，如果&&左边的表达式的值是 false，右边的表达式会被直接短路掉，不会进行运算。很多时候我们可能都需要用&&而不是&，例如在验证用户登录时判定用户名不是 null 而且不是空字符串，应当写为：username != null &&!username.equals(“”)，二者的顺序不能交换，更不能用&运算符，因为第一个条件如果不成立，根本不能进行字符串的 equals 比较，否则会

产生 NullPointerException 异常。注意：逻辑或运算符（|）和短路或运算符（||）的差别也是如此。

补充：如果你熟悉 **JavaScript**，那你可能更能感受到短路运算的强大，想成为 JavaScript 的高手就先从玩转短路运算开始吧。

253. 解释内存中的栈（stack）、堆(heap)和静态存储区的用法。

答：通常我们定义一个基本数据类型的变量，一个对象的引用，还有就是函数调用的现场保存都使用内存中的栈空间；而通过 new 关键字和构造器创建的对象放在堆空间；程序中的字面量（literal）如直接书写的 100、“hello”和常量都是放在静态存储区中。栈空间操作最快但是也很小，通常大量的对象都是放在堆空间，整个内存包括硬盘上的虚拟内存都可以被当成堆空间来使用。

String str = new String(“hello”);

上面的语句中 str 放在栈上，用 new 创建出来的字符串对象放在堆上，而“hello”这个字面量放在静态存储区。

补充：较新版本的 Java 中使用了一项叫“逃逸分析“的技术，可以将一些局部对象放在栈上以提升对象的操作性能。

254. Math.round(11.5) 等于多少？Math.round(-11.5)等于多少？

答 :Math.round(11.5)的返回值是 12 ,Math.round(-11.5)的返回值是-11。四舍五入的原理是在参数上加 0.5 然后进行下取整。

255. swtich 是否能作用在 byte 上，是否能作用在 long 上，是否能作用在 String 上?

答：早期的 JDK 中，switch（expr）中，expr 可以是 byte、short、char、int。从 1.5 版开始，Java 中引入了枚举类型（enum），expr 也可以是枚举，从 JDK 1.7 版开始，还可以是字符串（String）。长整型（long）是不可以的。

256. 用最有效率的方法计算 2 乘以 8?

答： 2 << 3（左移 3 位相当于乘以 2 的 3 次方，右移 3 位相当于除以 2 的 3 次方）。

补充 :我们为编写的类重写 hashCode 方法时 ,可能会看到如下所示的代码，其实我们不太理解为什么要使用这样的乘法运算来产生哈希码（散列码），而且为什么这个数是个素数，为什么通常选择 31 这个数？前两个问题的答案你可以自己百度一下 ,选择 31 是因为可以用移位和减法运算来代替乘法，从而得到更好的性能。说到这里你可能已经想到了 :31 * num <==> (num << 5) - num，左移 5 位相当于乘以 2 的 5 次方（32）再减去自身就相当于乘以 31。现在的 VM 都能自动完成这个优化。

```
package com.bjsxt;


public class PhoneNumber {
    private int areaCode;
    private String prefix;
    private String lineNumber;
```

```
    @Override
    public int hashCode() {
            final int prime = 31;
            int result = 1;
            result = prime * result + areaCode;
            result = prime * result
                    + ((lineNumber == null) ? 0 :
lineNumber.hashCode());
            result = prime * result + ((prefix == null) ? 0 :
prefix.hashCode());
            return result;
        }

        @Override
        public boolean equals(Object obj) {
            if (this == obj)
                return true;
            if (obj == null)
                return false;
            if (getClass() != obj.getClass())
                return false;
```

```
            PhoneNumber other = (PhoneNumber) obj;
            if (areaCode != other.areaCode)
                return false;
            if (lineNumber == null) {
                if (other.lineNumber != null)
                    return false;
            } else if (!lineNumber.equals(other.lineNumber))
                return false;
            if (prefix == null) {
                if (other.prefix != null)
                    return false;
            } else if (!prefix.equals(other.prefix))
                return false;
            return true;
        }


    }
```

257. 数组有没有 length()方法?String 有没有 length()方法？

答：数组没有 length()方法，有 length 的属性。String 有 length()方法。JavaScript 中，获得字符串的长度是通过 length 属性得到的，这一点容易和 Java 混淆。

258. 在 Java 中，如何跳出当前的多重嵌套循环？

答：在最外层循环前加一个标记如 A，然后用 break A;可以跳出多重循环。（Java 中支持带标签的 break 和 continue 语句，作用有点类似于 C 和 C++中的 goto 语句，但是就像要避免使用 goto 一样，应该避免使用带标签的 break 和 continue，因为它不会让你的程序变得更优雅，很多时候甚至有相反的作用，所以这种语法其实不知道更好）

259. 构造器（constructor）是否可被重写（override）？

答：构造器不能被继承，因此不能被重写，但可以被重载。

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

答：不对，如果两个对象 x 和 y 满足 x.equals(y) == true，它们的哈希码（hash code）应当相同。Java 对于 eqauls 方法和 hashCode 方法是这样规定的：(1)如果两个对象相同（equals 方法返回 true），那么它们的 hashCode 值一定要相同；(2)如果两个对象的 hashCode 相同，它们并不一定相同。当然，你未必要按照要求去做，但是如果你违背了上述原则就会发现在使用容器时，相同的对象可以出现在 Set 集合中，同时增加新元素的效率会大大下降（对于使用哈希存储的系统，如果哈希码频繁的冲突将会造成存取性能急剧下降）。

补充：关于 equals 和 hashCode 方法，很多 Java 程序都知道，但很多人也就是仅仅知道而已，在 Joshua Bloch 的大作《Effective Java》（很多软件公司，《Effective Java》、《Java 编程思想》以及《重构：改善既有代码质量》是 Java 程序员必看书籍，如果你还没看过，那就赶紧去亚马逊买一本吧）中是这样介绍 equals 方法的：首先 equals 方法必须满足自反性（x.equals(x)必须返回 true）、对称性（x.equals(y)返回 true 时，y.equals(x)也必须返回 true）、传递性（x.equals(y)和 y.equals(z)都返回 true 时，x.equals(z)也必须返回 true）和一致性（当 x 和 y 引用的对象信息没有被修改时，多次调用 x.equals(y)应该得到同样的返回值），而且对于任何非 null 值的引用 x，x.equals(null)必须返回 false。实现高质量的 equals 方法的诀窍包括：1. 使用==操作符检查“参数是否为这个对象的引用”；2. 使用 instanceof 操作符检查“参数是否为正确的类型”；3. 对于类中的关键属性，检查参数传入对象的属性是否与之相匹配；4. 编写完 equals 方法后，问自己它是否满足对称性、传递性、一致性；5. 重写 equals 时总是要重写 hashCode；6. 不要将 equals 方法参数中的 Object 对象替换为其他的类型，在重写时不要忘掉@Override 注解。

261. 是否可以继承 String 类?

答：String 类是 final 类，不可以被继承。

补充：继承 String 本身就是一个错误的行为，对 String 类型最好的重用方式是关联（HAS-A）而不是继承（IS-A）。

262. 当一个对象被当作参数传递到一个方法后，此方法可改变这个对象的属性，并可返回变化后的结果，那么这里到底是值传递还是引用传递?

答：是值传递。Java 编程语言只有值传递参数。当一个对象实例作为一个参数被传递到方法中时，参数的值就是对该对象的引用。对象的属性可以在被调用过程中被改变，但对象的引用是永远不会改变的。C++和 C#中可以通过传引用或传输出参数来改变传入的参数的值。

补充：Java 中没有传引用实在是非常的不方便，这一点在 Java 8 中仍然没有得到改进，正是如此在 Java 编写的代码中才会出现大量的 Wrapper 类（将需要通过方法调用修改的引用置于一个 Wrapper 类中，再将 Wrapper 对象传入方法），这样的做法只会让代码变得臃肿，尤其是让从 C 和 C++转型为 Java 程序员的开发者无法容忍。

263. String 和 StringBuilder、StringBuffer 的区别?

答：Java 平台提供了两种类型的字符串：String 和 StringBuffer / StringBuilder，

相同点：

它们都可以储存和操作字符串，同时三者都使用 final 修饰，都属于终结类不能派生子类，操作的相关方法也类似例如获取字符串长度等；

不同点：

其中 String 是只读字符串，也就意味着 String 引用的字符串内容是不能被改变的，而 StringBuffer 和 StringBuilder 类表示的字符串对象可以直接进行修改，在修改的同时地址值不会发生改变。StringBuilder 是 JDK 1.5 中

引入的，它和 StringBuffer 的方法完全相同，区别在于它是在单线程环境下使用的，因为它的所有方面都没有被 synchronized 修饰，因此它的效率也比 StringBuffer 略高。在此重点说明一下，String、StringBuffer、StringBuilder 三者类型不一样，无法使用 equals()方法比较其字符串内容是否一样！

补充 1：有一个面试题问：有没有哪种情况用+做字符串连接比调用 StringBuffer / StringBuilder 对象的 append 方法性能更好？如果连接后得到的字符串在静态存储区中是早已存在的，那么用+做字符串连接是优于 StringBuffer / StringBuilder 的 append 方法的。

补充 2：下面也是一个面试题，问程序的输出，看看自己能不能说出正确答案。

```
package com.bjsxt;

    public class smallT {

        public static void main(String[] args) {

                String a = "Programming";

                String b = new String("Programming");

                String c = "Program" + "ming";

                   System.out.println(a == b);

                   System.out.println(a == c);

                    System.out.println(a.equals(b));

                    System.out.println(a.equals(c));

                    System.out.println(a.intern() == b.intern());
```

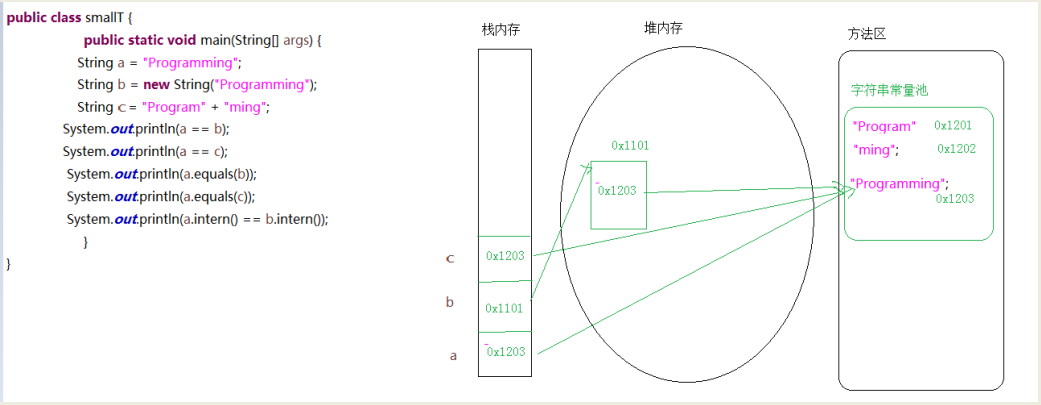
```
        }
    }
```

解析：

String 类存在 intern()方法，含义如下：返回字符串对象的规范化表示形式。

它遵循以下规则：对于任意两个字符串 s 和 t，当且仅当 s.equals(t) 为 true 时，s.intern() == t.intern() 才为 true。

字符串比较分为两种形式，一种使用比较运算符”==”比较，他们比较的是各自的字符串在内存当中的地址值是否相同；一种是使用 equals()方法进行比较，比较的是两个字符串的内容是否相同！



结果如下：

```
a == b-->false

a == c-->true

a.equals(b)-->true

a.equals(c)-->true

a.intern() == b.intern()-->true
```

264. 重载（Overload）和重写（Override）的区别。重载的方法能否根据返回类型进行区分?

答：Java 的三大特征之一，多态机制，包括方法的多态和对象的多态；方法的重载和重写都是实现多态的方式，区别在于前者实现的是编译时的多态性，而后者实现的是运行时的多态性。重载（overload）发生在同一个类中，相同的方法，如果有不同的参数列表（参数类型不同、参数个数不同或者二者都不同）则视为重载；重写（override）发生在子类与父类之间也就是继承机制当中，当父类的方法不能满足子类的要求，此时子类重写父类的方法；要求：方法名、形参列表相同；返回值类型和异常类型，子类小于等于父类；访问权限，子类大于等于父类，切记父类的私有方法以及被 final 修饰的方法不能被子类重写；重载对返回类型没有特殊的要求。

补充：华为的面试题中曾经问过这样一个问题：为什么不能根据返回类型来区分重载，说出你的答案吧！

答：方法的重载，即使返回值类型不同，也不能改变实现功能相同或类似这一既定事实；同时方法的重载只是要求两同三不同，即在同一个类中，相同的方法名称，参数列表当中的参数类型、个数、顺序不同；跟权限修饰符和返回值类无关

265. 描述一下 JVM 加载 class 文件的原理机制?

答：JVM 中类的装载是由类加载器（ClassLoader） 和它的子类来实现的，Java 中的类加载器是一个重要的 Java 运行时系统组件，它负责在运行时查找和装入类文件中的类。

补充：

1.由于 Java 的跨平台性，经过编译的 Java 源程序并不是一个可执行程序，而是一个或多个类文件。当 Java 程序需要使用某个类时，JVM 会确保这个类已经被加载、连接(验证、准备和解析)和初始化。类的加载是指把类的.class 文件中的数据读入到内存中，通常是创建一个字节数组读入.class 文件，然后产生与所加载类对应的 Class 对象。加载完成后，Class 对象还不完整，所以此时的类还不可用。当类被加载后就进入连接阶段，这一阶段包括验证、准备(为静态变量分配内存并设置默认的初始值)和解析(将符号引用替换为直接引用)三个步骤。最后 JVM 对类进行初始化，包括：1 如果类存在直接的父类并且这个类还没有被初始化，那么就先初始化父类；2 如果类中存在初始化语句，就依次执行这些初始化语句。

2.类的加载是由类加载器完成的，类加载器包括：根加载器（BootStrap）、扩展加载器（Extension）、系统加载器（System）和用户自定义类加载器（java.lang.ClassLoader 的子类）。从 JDK 1.2 开始，类加载过程采取了父亲委托机制(PDM)。PDM 更好的保证了 Java 平台的安全性，在该机制中，JVM 自带的 Bootstrap 是根加载器，其他的加载器都有且仅有一个父类加载器。类的加载首先请求父类加载器加载，父类加载器无能为力时才由其子类加载器自行加载。JVM 不会向 Java 程序提供对 Bootstrap 的引用。下面是关于几个类加载器的说明：

a)Bootstrap：一般用本地代码实现，负责加载 JVM 基础核心类库（rt.jar）；

b)Extension：从 java.ext.dirs 系统属性所指定的目录中加载类库，它的父加载器是 Bootstrap；

c)System：又叫应用类加载器，其父类是 Extension。它是应用最广泛的类加载器。它从环境变量 classpath 或者系统属性 java.class.path 所指定的目录中记载类，是用户自定义加载器的默认父加载器。

char 型变量中能不能存贮一个中文汉字?为什么?

答：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。

补充：使用 Unicode 意味着字符在 JVM 内部和外部有不同的表现形式，在 JVM 内部都是 Unicode，当这个字符被从 JVM 内部转移到外部时（例如存入文件系统中），需要进行编码转换。所以 Java 中有字节流和字符流，以及在字符流和字节流之间进行转换的转换流，如 InputStreamReader 和 OutputStreamReader，这两个类是字节流和字符流之间的适配器类，承担了编码转换的任务；对于 C 程序员来说，要完成这样的编码转换恐怕要依赖于 union（联合体/共用体）共享内存的特征来实现了。

抽象类（abstract class）和接口（interface）有什么异同?

答：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 classBaseServlet 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 extendsBaseServlet
{
protected voiddoService(HttpServletRequest request,
HttpServletResponse response) throwsIOExcetion,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。

266. 静态嵌套类(Static Nested Class)和内部类（Inner Class）的不同？

答：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 = newOuter.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(不能直接访问 ,需要创建外部类实例才能访问非静态变量)。**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。

Static Nested Class 是被声明为静态（static）的内部类，它可以不依赖于外部类实例被实例化。而通常的内部类需要在外部类实例化后才能实例化，其语法看起来挺诡异的，如下所示。

```
package com.bjsxt;
/**
 * 扑克类（一副扑克）
 * @author sxt
 *
 */
    public class Poker {
        private static String[] suites = {"黑桃", "红桃", "草花", "方块"};
        private static int[] faces = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13};
            private Card[] cards;
        /**
          * 构造器
         */
        public Poker() {
            cards = new Card[52];
            for(int i = 0; i < suites.length; i++) {
```

```
                for(int j = 0; j < faces.length; j++) {
                        cards[i * 13 + j] = new Card(suites[i], faces[j]);
                    }
                }
            }

            /**
             * 洗牌 （随机乱序）
             */
            public void shuffle() {
                for(int i = 0, len = cards.length; i < len; i++) {
                        int index = (int) (Math.random() * len);
                        Card temp = cards[index];
                        cards[index] = cards[i];
                        cards[i] = temp;
                    }
                }
                /**
                 * 发牌
                 * @param index 发牌的位置
                 */
                public Card deal(int index) {
```

```
        return cards[index];
    }

    /**
     * 卡片类（一张扑克）
     * [内部类]
     * @author sxt
     */
public class Card {
  private String suite;    // 花色
    private int face;         // 点数
  public Card(String suite, int face) {
        this.suite = suite;
        this.face = face;
   }
        @Override
        public String toString() {
            String faceStr = "";
            switch(face) {
                case 1: faceStr = "A"; break;
            case 11: faceStr = "J"; break;
            case 12: faceStr = "Q"; break;
```

```
                case 13: faceStr = "K"; break;
                default: faceStr = String.valueOf(face);
                }
                return suite + faceStr;
            }
        }
    }
```

测试类：

```
package com.bjsxt;


class PokerTest {
    public static void main(String[] args) {
        Poker poker = new Poker();
        poker.shuffle();                    // 洗牌
        Poker.Card c1 = poker.deal(0);   // 发第一张牌
            // 对于非静态内部类Card
        // 只有通过其外部类Poker对象才能创建Card对象
        Poker.Card c2 = poker.new Card("红心", 1);      // 自己创建一
张牌
```

```
            System.out.println(c1);        // 洗牌后的第一张
            System.out.println(c2);        // 打印: 红心A
        }
    }
```

267. Java 中会存在内存泄漏吗，请简单描述。

答：理论上 Java 因为有垃圾回收机制（GC）不会存在内存泄露问题（这也是 Java 被广泛使用于服务器端编程的一个重要原因）；然而在实际开发中，可能会存在无用但可达的对象，这些对象不能被 GC 回收也会发生内存泄露。一个例子就是 hibernate 的 Session（一级缓存）中的对象属于持久态，垃圾回收器是不会回收这些对象的，然而这些对象中可能存在无用的垃圾对象。下面的例子也展示了 Java 中发生内存泄露的情况：

```
package com.bjsxt;
import java.util.Arrays;
import java.util.EmptyStackException;

public class MyStack<T> {
    private T[] elements;
    private int size = 0;
    private static final int INIT_CAPACITY = 16;
        public MyStack() {
```

```
        elements = (T[]) new Object[INIT_CAPACITY];
    }
    public void push(T elem) {
        ensureCapacity();
        elements[size++] = elem;
    }
    public T pop() {
        if(size == 0)
            throw new EmptyStackException();
        return elements[--size];
    }
    private void ensureCapacity() {
        if(elements.length == size) {
            elements = Arrays.copyOf(elements, 2 * size + 1);
        }
    }
}
```

上面的代码实现了一个栈（先进后出（FILO））结构，乍看之下似乎没有什么明显的问题，它甚至可以通过你编写的各种单元测试。然而其中的 pop 方法却存在内存泄露的问题，当我们用 pop 方法弹出栈中的对象时，该对象不会被当作垃圾回收，即使使用栈的程序不再引用这些对象，因为栈内部

维护着对这些对象的过期引用（obsolete reference）。在支持垃圾回收的语言中，内存泄露是很隐蔽的，这种内存泄露其实就是无意识的对象保持。如果一个对象引用被无意识的保留起来了，那么垃圾回收器不会处理这个对象，也不会处理该对象引用的其他对象，即使这样的对象只有少数几个，也可能会导致很多的对象被排除在垃圾回收之外，从而对性能造成重大影响，极端情况下会引发 Disk Paging（物理内存与硬盘的虚拟内存交换数据），甚至造成 OutOfMemoryError。

268. 抽象的（abstract）方法是否可同时是静态的（static）,是否可同时是本地方法（native）,是否可同时被 synchronized 修饰?

答：都不能。抽象方法需要子类重写，而静态的方法是无法被重写的，因此二者是矛盾的。本地方法是由本地代码（如 C 代码）实现的方法，而抽象方法是没有实现的，也是矛盾的。synchronized 和方法的实现细节有关，抽象方法不涉及实现细节，因此也是相互矛盾的。

269. 静态变量和实例变量的区别？

答：静态变量是被 static 修饰符修饰的变量，也称为类变量，它属于类，不属于类的任何一个对象，一个类不管创建多少个对象，静态变量在内存中有且仅有一个拷贝；实例变量必须依存于某一实例，需要先创建对象然后通过对象才能访问到它，静态变量可以实现让多个对象共享内存。两者的相同点：都有默认值而且在类的任何地方都可以调用。在 Java 开发中，上下文类和工具类中通常会有大量的静态成员。

270. 是否可以从一个静态（static）方法内部发出对非静态（non-static）方法的调用？

答：不可以，静态方法只能访问静态成员，因为非静态方法的调用要先创建对象，因此在调用静态方法时可能对象并没有被初始化。

271. 如何实现对象克隆？

答：有两种方式：

1.实现 Cloneable 接口并重写 Object 类中的 clone()方法；

```
package com.bjsxt;

import java.io.ByteArrayInputStream;

import java.io.ByteArrayOutputStream;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;


public class MyUtil {

        private MyUtil() {

            throw new AssertionError();

    }

        public static <T> T clone(T obj) throws Exception {
```

```
            ByteArrayOutputStream bout = new ByteArrayOutputStream();
            ObjectOutputStream oos = new ObjectOutputStream(bout);
            oos.writeObject(obj);
            ByteArrayInputStream bin = new
ByteArrayInputStream(bout.toByteArray());
            ObjectInputStream ois = new ObjectInputStream(bin);
            return (T) ois.readObject();
            // 说明：调用ByteArrayInputStream或ByteArrayOutputStream对
象的close方法没有任何意义
            // 这两个基于内存的流只要垃圾回收器清理对象就能够释放资源
        }
    }
```

2.实现 Serializable 接口，通过对象的序列化和反序列化实现克隆，可以实现真正的深度克隆，代码如下。

下面是测试代码：

```
package com.bjsxt;
import java.io.Serializable;
/**
 * 人类
 * @author sxt
*/
```

```
class Person implements Serializable {

    private static final long serialVersionUID = -9102017020286042305L;

    private String name;      // 姓名

    private int age;          // 年龄

    private Car car;          // 座驾


    public Person(String name, int age, Car car) {

        this.name = name;

        this.age = age;

        this.car = car;

    }

    public String getName() {

        return name;

    }

    public void setName(String name) {

        this.name = name;

    }

    public int getAge() {

        return age;

    }

    public void setAge(int age) {

        this.age = age;
```

```
    }
    public Car getCar() {
        return car;
    }
    public void setCar(Car car) {
        this.car = car;
    }
    @Override
    public String toString() {
     return "Person [name=" + name + ", age=" + age + ", car=" + car + "]";
    }
  }
  /**
 * 小汽车类
 * @author sxt
*/
class Car implements Serializable {
    private static final long serialVersionUID = -5713945027627603702L;
    private String brand;          // 品牌
    private int maxSpeed;          // 最高时速

    public Car(String brand, int maxSpeed) {
```

```
        this.brand = brand;
        this.maxSpeed = maxSpeed;
    }
    public String getBrand() {
        return brand;
    }
    public void setBrand(String brand) {
        this.brand = brand;
    }
    public int getMaxSpeed() {
        return maxSpeed;
    }

    public void setMaxSpeed(int maxSpeed) {
        this.maxSpeed = maxSpeed;
    }

    @Override
    public String toString() {
        return "Car [brand=" + brand + ", maxSpeed=" + maxSpeed + "]";
    }
}
```

```
class CloneTest {
    public static void main(String[] args) {
        try {
            Person p1 = new Person("Hao LUO", 33, new Car("Benz", 300));
            Person p2 = MyUtil.clone(p1);    // 深度克隆
            p2.getCar().setBrand("BYD");
                // 修改克隆的Person对象p2关联的汽车对象的品牌属性
                // 原来的Person对象p1关联的汽车不会受到任何影响
                // 因为在克隆Person对象时其关联的汽车对象也被克隆了
                System.out.println(p1);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
```

注意：基于序列化和反序列化实现的克隆不仅仅是深度克隆，更重要的是通过泛型限定，可以检查出要克隆的对象是否支持序列化，这项检查是编译器完成的，不是在运行时抛出异常，这种是方案明显优于使用 Object 类的 clone 方法克隆对象。

272. GC 是什么？为什么要有 GC？

答：GC 是垃圾收集的意思，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。Java 程序员不用担心内存管理，因为垃圾收集器会自动进行管理。要请求垃圾收集，可以调用下面的方法之一：System.gc() 或 Runtime.getRuntime().gc() ，但 JVM 可以屏蔽掉显示的垃圾回收调用。

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优先级的线程运行，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除和回收，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。在 Java 诞生初期，垃圾回收是 Java 最大的亮点之一，因为服务器端的编程需要有效的防止内存泄露问题，然而时过境迁，如今 Java 的垃圾回收机制已经成为被诟病的东西。移动智能终端用户通常觉得 **iOS** 的系统比 **Android** 系统有更好的用户体验，其中一个深层次的原因就在于 Android 系统中垃圾回收的不可预知性。

补充：垃圾回收机制有很多种，包括：分代复制垃圾回收、标记垃圾回收、增量垃圾回收等方式。标准的 Java 进程既有栈又有堆。栈保存了原始型局部变量，堆保存了要创建的对象。Java 平台对堆内存回收和再利用的基本算法被称为标记和清除，但是 Java 对其进行了改进，采用“分代式垃圾收集”。这种方法会跟 Java 对象的生命周期将堆内存划分为不同的区域，在垃圾收

集过程中，可能会将对象移动到不同区域：

- 伊甸园（Eden）：这是对象最初诞生的区域，并且对大多数对象来说，这里是它们唯一存在过的区域。
- 幸存者乐园（Survivor）：从伊甸园幸存下来的对象会被挪到这里。
- 终身颐养园（Tenured）：这是足够老的幸存对象的归宿。年轻代收集（Minor-GC）过程是不会触及这个地方的。当年轻代收集不能把对象放进终身颐养园时，就会触发一次完全收集（Major-GC），这里可能还会牵扯到压缩，以便为大对象腾出足够的空间。

与垃圾回收相关的 JVM 参数：

- -Xms / -Xmx --- 堆的初始大小 / 堆的最大大小
- -Xmn --- 堆中年轻代的大小
- -XX:-DisableExplicitGC --- 让 System.gc()不产生任何作用
- -XX:+PrintGCDetail --- 打印 GC 的细节
- -XX:+PrintGCDateStamps --- 打印 GC 操作的时间戳

273. String s=new String(“xyz”);创建了几个字符串对象？

答：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。

274. 接口是否可继承（extends）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）？

答：接口可以继承接口。抽象类可以实现(implements)接口，抽象类可以继承具体类。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。

275. 一个“.java”源文件中是否可以包含多个类（不是内部类）？有什么限制？

答：可以，但一个源文件中最多只能有一个公开类（public class）而且文件名必须和公开类的类名完全保持一致。

276. Anonymous Inner Class(匿名内部类)是否可以继承其它类？是否可以实现接口？

答：可以继承其他类或实现其他接口，在 Swing 编程中常用此方式来实现事件监听和回调。 但是有一点需要注意，它只能继承一个类或一个接口。

277. 内部类可以引用它的包含类（外部类）的成员吗？有没有什么限制？

答：一个内部类对象可以访问创建它的外部类对象的成员，包括私有成员。

如果要访问外部类的局部变量，此时局部变量必须使用 final 修饰，否则无法访问。

278. Java 中的 final 关键字有哪些用法？

答：(1)修饰类：表示该类不能被继承；(2)修饰方法：表示方法不能被重写但是允许重载；(3)修饰变量：表示变量只能一次赋值以后值不能被修改（常量）；(4)修饰对象：对象的引用地址不能变，但是对象的初始化值可以变。

279. 指出下面程序的运行结果:

```
package com.bjsxt;
class A{
    static{
        System.out.print("1");
    }
  public A(){
        System.out.print("2");
    }
   }
   class B extends A{
     static{
            System.out.print("a");
        }
        public B(){
            System.out.print("b");
```

```
        }
    }

    public class Hello{
       public static void main(String[] args){
            A ab = new B();
            ab = new B();
        }
}
```

答：执行结果：1a2b2b。创建对象时构造器的调用顺序是：先初始化静态成员，然后调用父类构造器，再初始化非静态成员，最后调用自身构造器。

考点：静态代码块优先级 > 构造方法的优先级

如果再加一个普通代码块，优先顺序如下：

静态代码块>普通代码块>构造方法

280. 数据类型之间的转换:

1)如何将字符串转换为基本数据类型？

2)如何将基本数据类型转换为字符串？

答：

1)调用基本数据类型对应的包装类中的方法 parseXXX(String)或 valueOf(String)即可返回相应基本类型；

2)一种方法是将基本数据类型与空字符串（“ ”）连接（+）即可获得其所

对应的字符串；另一种方法是调用 String 类中的 valueOf(...)方法返回相应字符串

281. 如何实现字符串的反转及替换？

答：方法很多，可以自己写实现也可以使用 String 或 StringBuffer / StringBuilder 中的方法。有一道很常见的面试题是用递归实现字符串反转，代码如下所示：

```
package com.bjsxt;

        public class A{

          public static String reverse(String originStr) {

          if(originStr == null || originStr.length() <= 1)

                return originStr;

             return reverse(originStr.substring(1)) + originStr.charAt(0);

           }

}
```

282. 怎样将 GB2312 编码的字符串转换为 ISO-8859-1 编码的字符串？

答：代码如下所示:

String s1 = "你好";

String s2 = newString(s1.getBytes("GB2312"), "ISO-8859-1");

在 String 类的构造方法当中，存在一个字符集设置的方法，具体如下：

```
public String(byte[] bytes,
              String charsetName)
       throws UnsupportedEncodingException
```

通过使用指定的 charset 解码指定的 byte 数组，构造一个新的 String。新 String 的长度是字符集的函数，因此可能不等于 byte 数组的长度。

当给定 byte 在给定字符集中无效的情况下，此构造方法的行为没有指定。如果需要对解码过程进行更多控制，则应该使用 CharsetDecoder 类。

参数：

bytes - 要解码为字符的 byte

charsetName - 受支持的 charset 的名称

抛出：

UnsupportedEncodingException - 如果指定字符集不受支持

从以下版本开始：

JDK1.1

283. Java 中的日期和时间：

1)如何取得年月日、小时分钟秒？

2)如何取得从 1970 年 1 月 1 日 0 时 0 分 0 秒到现在的毫秒数？

3)如何取得某月的最后一天？

4)如何格式化日期？

答：操作方法如下所示：

1)创建 java.util.Calendar 实例 ,调用其 get()方法传入不同的参数即可获得参数所对应的值

2)以下方法均可获得该毫秒数:

```
Calendar.getInstance().getTimeInMillis();

System.currentTimeMillis();
```

3)示例代码如下:

```
Calendar time = Calendar.getInstance();

time.getActualMaximum(Calendar.DAY_OF_MONTH);
```

4)利用 java.text.DataFormat 的子类（如 SimpleDateFormat 类）中的

format(Date)方法可将日期格式化。

284. 打印昨天的当前时刻。

答：

```
package com.bjsxt;

import java.util.Calendar;

public class YesterdayCurrent {

	public static void main(String[] args){

		Calendar cal = Calendar.getInstance();

		cal.add(Calendar.DATE, -1);

		System.out.println(cal.getTime());

	}

}
```

285. 比较一下 Java 和 JavaSciprt

答 :JavaScript 与 Java 是两个公司开发的不同的两个产品。Java 是原 Sun 公司推出的面向对象的程序设计语言，特别适合于互联网应用程序开发；而 JavaScript 是 Netscape 公司的产品，为了扩展 Netscape 浏览器的功能而开发的一种可以嵌入 Web 页面中运行的基于对象和事件驱动的解释性语言，它的前身是 LiveScript；而 Java 的前身是 Oak 语言。

下面对两种语言间的异同作如下比较：

1）基于对象和面向对象：Java 是一种真正的面向对象的语言，即使是开发

简单的程序，必须设计对象；JavaScript 是种脚本语言，它可以用来制作与网络无关的，与用户交互作用的复杂软件。它是一种基于对象（Object-Based）和事件驱动（Event-Driven）的编程语言。因而它本身提供了非常丰富的内部对象供设计人员使用；

2）解释和编译：Java 的源代码在执行之前，必须经过编译；JavaScript 是一种解释性编程语言，其源代码不需经过编译，由浏览器解释执行；

3）强类型变量和类型弱变量：Java 采用强类型变量检查，即所有变量在编译之前必须作声明；JavaScript 中变量声明，采用其弱类型。即变量在使用前不需作声明，而是解释器在运行时检查其数据类型；

4）代码格式不一样。

补充：上面列出的四点是原来所谓的标准答案中给出的。其实 Java 和 JavaScript 最重要的区别是一个是静态语言，一个是动态语言。目前的编程语言的发展趋势是函数式语言和动态语言。在 Java 中类（class）是一等公民，而 JavaScript 中函数（function）是一等公民。对于这种问题，在面试时还是用自己的语言回答会更加靠谱。

286. 什么时候用 assert？

答：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。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。在实现中，断言是一个包含布尔表达式的语句，在执行这个语句时假定该表达式为 true；如果表达式计算为 false，那么系统会报告一个 AssertionError。

断言用于调试目的：

assert(a > 0); // throws an AssertionError if a <= 0

断言可以有两种形式：

assert Expression1;

assert Expression1 : Expression2 ;

Expression1 应该总是产生一个布尔值。

Expression2 可以是得出一个值的任意表达式 ;这个值用于生成显示更多调试信息的字符串消息。

断言在默认情况下是禁用的，要在编译时启用断言，需使用 source 1.4 标记：

javac -source 1.4 Test.java

要在运行时启用断言，可使用-enableassertions 或者-ea 标记。

要在运行时选择禁用断言，可使用-da 或者-disableassertions 标记。

要在系统类中启用断言，可使用-esa 或者-dsa 标记。还可以在包的基础上启用或者禁用断言。可以在预计正常情况下不会到达的任何位置上放置断言。断言可以用于验证传递给私有方法的参数。不过，断言不应该用于验证传递给公有方法的参数，因为不管是否启用了断言，公有方法都必须检查其参数。不过，既可以在公有方法中，也可以在非公有方法中利用断言测试后置条件。另外，断言不应该以任何方式改变程序的状态。

287. Error 和 Exception 有什么区别?

答：Error 表示系统级的错误和程序不必处理的异常，是恢复不是不可能但很困难的情况下的一种严重问题；比如内存溢出，不可能指望程序能处理这

样的情况；Exception 表示需要捕捉或者需要程序进行处理的异常，是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。补充：2005 年摩托罗拉的面试中曾经问过这么一个问题“If a process reports a stack overflow run-time error, what’s the most possible cause?”，给了四个选项 a. lack of memory; b. write on an invalid memory space; c. recursive function calling; d. array index out of boundary. Java 程序在运行时也可能会遭遇 StackOverflowError，这是一个错误无法恢复，只能重新修改代码了，这个面试题的答案是 c。如果写了不能迅速收敛的递归，则很有可能引发栈溢出的错误，如下所示：

```
package com.bjsxt;
public class StackOverflowErrorTest {
        public static void main(String[] args) {
            main(null);
        }
}
```

因此，用递归编写程序时一定要牢记两点：1. 递归公式；2. 收敛条件（什么时候就不再递归而是回溯了）。

（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。）

288. 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?

答：会执行，在方法返回调用者前执行。Java 允许在 finally 中改变返回值的做法是不好的 ,因为如果存在 finally 代码块 ,try 中的 return 语句不会立马返回调用者，而是记录下返回值待 finally 代码块执行完毕之后再向调用者返回其值，然后如果在 finally 中修改了返回值，这会对程序造成很大的困扰，C#中就从语法上规定不能做这样的事。

(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 classTest {


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


    static int test()
    {
```

```
        int x = 1;
        try
        {
            returnx;
        }
        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。

)

289. Java 语言如何进行异常处理 ,关键字 :throws、throw、try、catch、finally 分别如何使用？

答：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，如果出现异常，系统会抛出（throw）一个异常，这时候你可以通过它的类型来捕捉（catch）它，或最后（finally）由缺省处理器来处理；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，用来指定你想要捕捉的“异常”的类型；throw 语句用来明确地抛出一个“异常”；throws 用来标明一个成员函数可能抛出的各种“异常”；finally 为确保一段代码不管发生什么“异常”都被执行一段代码；可以在一个成员函数调用的外面写一个 try 语句 ,在这个成员函数内部写另一个 try 语句保护其他代码。每当遇到一个 try 语句，“异常”的框架就放到栈上面，直到所有的 try 语句都完成。如果下一级的 try 语句没有对某种“异常”进行处理，栈就会展开，直到遇到有处理这种“异常”的 try 语句。

290. 运行时异常与受检异常有何异同？

答：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，只要程序设计得没有问题通常就不会发生。受检异常跟程序运行的上下文环境有关，即使程序

设计无误，仍然可能因使用的问题而引发。Java 编译器要求方法必须声明抛出可能发生的受检异常，但是并不要求必须声明抛出未被捕获的运行时异常。异常和继承一样，是面向对象程序设计中经常被滥用的东西，神作《Effective Java》中对异常的使用给出了以下指导原则：

- 不要将异常处理用于正常的控制流（设计良好的 API 不应该强迫它的调用者为了正常的控制流而使用异常）
- 对可以恢复的情况使用受检异常，对编程错误使用运行时异常
- 避免不必要的使用受检异常（可以通过一些状态检测手段来避免异常发生）
- 优先使用标准的异常
- 每个方法抛出的异常都要有文档
- 保持异常的原子性
- 不要在 catch 中忽略掉捕获到的异常

(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。)

291. 列出一些你常见的运行时异常？

答：

ArithmeticException（算术异常）

ClassCastException （类转换异常）

IllegalArgumentException （非法参数异常）

IndexOutOfBoundsException （下表越界异常）

NullPointerException （空指针异常）

SecurityException （安全异常）

NumberFormatException(数字格式化异常)

292. final, finally, finalize 的区别?

答：final：修饰符（关键字）有三种用法：如果一个类被声明为 final，意味着它不能再派生出新的子类，即不能被继承，因此它和 abstract 是反义词。将变量声明为 final，可以保证它们在使用中不被改变，被声明为 final 的变量必须在声明时给定初值，而在以后的引用中只能读取不可修改。被声明为 final 的方法也同样只能使用，不能在子类中被重写。finally：通常放在 try...catch 的后面构造总是执行代码块，这就意味着程序无论正常执行还是发生异常，这里的代码只要 JVM 不关闭都能执行，可以将释放外部资源的代码写在 finally 块中。finalize：Object 类中定义的方法，Java 中允许使用 finalize() 方法在垃圾收集器将对象从内存中清除出去之前做必要的清理工作。这个方法是由垃圾收集器在销毁对象时调用的，通过重写 finalize() 方法可以整理系统资源或者执行其他清理工作。

(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

)

293. 类 ExampleA 继承 Exception，类 ExampleB 继承 ExampleA

有如下代码片断：

```
try{
    throw new ExampleB("b")
    }catch（ExampleA e）{
    System.out.println("ExampleA");
    }catch（Exception e）{
    System.out.println("Exception");
    }
}
```

请问执行此段代码的输出是什么？

答：输出：ExampleA。（根据里氏代换原则[能使用父类型的地方一定能使用子类型]，抓取 ExampleA 类型异常的 catch 块能够抓住 try 块中抛出的 ExampleB 类型的异常）

补充：比此题略复杂的一道面试题如下所示（此题的出处是《Java 编程思想》），说出你的答案吧！

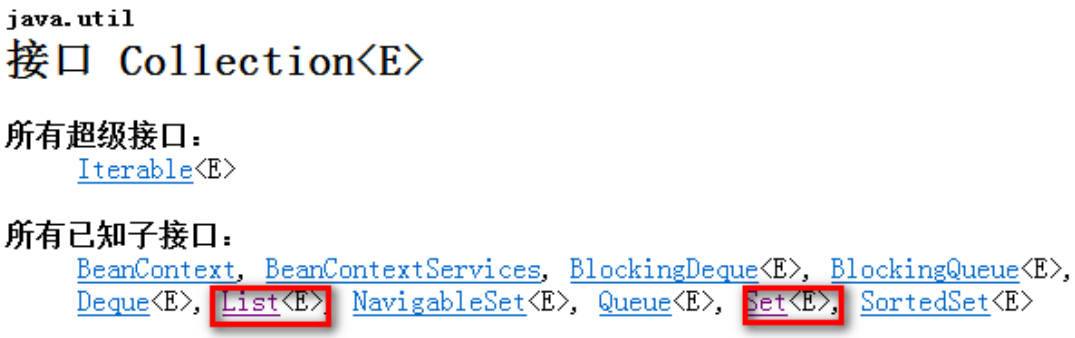
```
package com.bjsxt;
```

```
class Annoyance extends Exception {}

class Sneeze extends Annoyance {}

 class Human {

  public static void main(String[] args)

          throws Exception {

          try {

               try {

                        throw new Sneeze();

                    }

                    catch ( Annoyance a ) {

                         System.out.println("Caught Annoyance");

                         throw a;

                    }

               }

               catch ( Sneeze s ) {

                    System.out.println("Caught Sneeze");

                    return ;

               }

               finally {

                    System.out.println("Hello World!");

               }

         }
```

```
}
```

294. List、Set、Map 是否继承自 Collection 接口？

答：List、Set 的父接口是 Collection，Map 不是其子接口，而是与 Collection 接口是平行关系，互不包含。



Map 是键值对映射容器，与 List 和 Set 有明显的区别，而 Set 存储的零散的元素且不允许有重复元素（数学中的集合也是如此），List 是线性结构的容器，适用于按数值索引访问元素的情形。

295. 说出 ArrayList、Vector、LinkedList 的存储性能和特性？

答：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（将内存中零散的内存单元通过附加的引用关联起来，形成一个可以按序号索引的线性结构，这种

链式存储方式与数组的连续存储方式相比，其实对内存的利用率更高），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。Vector 属于遗留容器（早期的 JDK 中使用的容器，除此之外 Hashtable、Dictionary、BitSet、Stack、Properties 都是遗留容器）,现在已经不推荐使用 ,但是由于 ArrayList 和 LinkedListed 都是非线程安全的，如果需要多个线程操作同一个容器，那么可以通过工具类 Collections 中的 synchronizedList 方法将其转换成线程安全的容器后再使用（这其实是装潢模式最好的例子，将已有对象传入另一个类的构造器中创建新的对象来增加新功能）。

补充：遗留容器中的 Properties 类和 Stack 类在设计上有严重的问题，Properties 是一个键和值都是字符串的特殊的键值对映射 ,在设计上应该是关联一个 Hashtable 并将其两个泛型参数设置为 String 类型 ,但是 Java API 中的 Properties 直接继承了 Hashtable，这很明显是对继承的滥用。这里复用代码的方式应该是 HAS-A 关系而不是 IS-A 关系 ,另一方面容器都属于工具类，继承工具类本身就是一个错误的做法，使用工具类最好的方式是 HAS-A 关系（关联）或 USE-A 关系（依赖）。同理，Stack 类继承 Vector 也是不正确的。

296. Collection 和 Collections 的区别？

答：Collection 是一个接口，它是 Set、List 等容器的父接口；

Collections 是个一个工具类，提供了一系列的静态方法来辅助容器操作，这些方法包括对容器的搜索、排序、线程安全化等等。

延伸：throw 与 throws 之间的区别

相同点： 都属于处理异常的方式

不同点：

位置不同：

throw 是在语句内

throws 是在语句上

内容不同：

throw 后面跟的是具体异常类对象

throws 后面跟的是具体异常类

Array 与 Arrays 之间的区别

相同点：都可以访问 Java 数组对其进行操作。

不同点：所在的包不同 Array-----java.lang.reflect 包

Arrays---java.util 包

是否允许有子类--- Array 因为使用 final 修饰不允许

--- Arrays 可以允许有子类

处理数组的方法

---- Array 类提供了动态创建和访问 Java 数组的方法。

---Arrays 包含用来操作数组（比如排序和搜索）的各种方法。此类还包含一个允许将数组作为列表来查看的静态工厂。

297. List、Map、Set 三个接口，存取元素时，各有什么特点？

答：List 以特定索引来存取元素，可有重复元素。

Set 不能存放重复元素（用对象的 equals()方法来区分元素是否重复）。

Map 保存键值对（key-value pair）映射，映射关系可以是一对一或多对一。Set 和 Map 容器都有基于哈希存储和排序树（红黑树）的两种实现版本，基于哈希存储的版本理论存取时间复杂度为 O(1)，而基于排序树版本的实现在插入或删除元素时会按照元素或元素的键（key）构成排序树从而达到排序和去重的效果。

298. TreeMap 和 TreeSet 在排序时如何比较元素？Collections 工具类中的 sort()方法如何比较元素？

答：TreeSet 要求存放的对象所属的类必须实现 Comparable 接口，该接口提供了比较元素的 compareTo()方法，当插入元素时会回调该方法比较元素的大小。

TreeMap 要求存放的键值对映射的键必须实现 Comparable 接口从而根据键对元素进行排序。

Collections 工具类的 sort 方法有两种重载的形式，第一种要求传入的待排序容器中存放的对象比较实现 Comparable 接口以实现元素的比较；第二种不强制性的要求容器中的元素必须可比较，但是要求传入第二个参数，参数是 Comparator 接口的子类型（需要重写 compare 方法实现元素的比较），相当于一个临时定义的排序规则，其实就是是通过接口注入比较元素大小的**算法**，也是对回调模式的应用。

例子 1：

Student.java

```
package com.bjsxt;
```

```
public class Student implements Comparable<Student> {
    private String name;            // 姓名
    private int age;                // 年龄
    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }
    @Override
    public String toString() {
        return "Student [name=" + name + ", age=" + age + "]";
    }
    @Override
    public int compareTo(Student o) {
        return this.age - o.age; // 比较年龄(年龄的升序)
    }
}
```

Test01.java

```
package com.bjsxt;
import java.util.Set;
import java.util.TreeSet;
```

```
class Test01 {
   public static void main(String[] args) {
            Set<Student> set = new TreeSet<>();        // Java 7的钻石语法
(构造器后面的尖括号中不需要写类型)
                set.add(new Student("Hao LUO", 33));
                set.add(new Student("XJ WANG", 32));
                set.add(new Student("Bruce LEE", 60));
                set.add(new Student("Bob YANG", 22));
                   for(Student stu : set) {
                      System.out.println(stu);
                }
     //         输出结果:
     //         Student [name=Bob YANG, age=22]
     //         Student [name=XJ WANG, age=32]
     //         Student [name=Hao LUO, age=33]
     //         Student [name=Bruce LEE, age=60]
          }
}
```

例子 2：

Student.java

```
package com.bjsxt;


public class Student {
    private String name;      // 姓名
    private int age;          // 年龄
    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }
    /**
     * 获取学生姓名
     */
    public String getName() {
        return name;
    }
    /**
     * 获取学生年龄
     */
    public int getAge() {
        return age;
    }
    @Override
```

```
    public String toString() {
        return "Student [name=" + name + ", age=" + age + "]";
    }
}
```

Test02.java

```
package com.bjsxt;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

class Test02 {
    public static void main(String[] args) {
        List<Student> list = new ArrayList<>();        // Java 7的钻石语法(构造器后面的尖括号中不需要写类型)
        list.add(new Student("Hao LUO", 33));
        list.add(new Student("XJ WANG", 32));
        list.add(new Student("Bruce LEE", 60));
        list.add(new Student("Bob YANG", 22));

        // 通过sort方法的第二个参数传入一个Comparator接口对象
```

```
            // 相当于是传入一个比较对象大小的算法到sort方法中
            // 由于Java中没有函数指针、仿函数、委托这样的概念
            // 因此要将一个算法传入一个方法中唯一的选择就是通过接口回
调
            Collections.sort(list, new Comparator<Student> () {
                @Override
                public int compare(Student o1, Student o2) {
                    return o1.getName().compareTo(o2.getName());
// 比较学生姓名
                }
            });

            for(Student stu : list) {
                System.out.println(stu);
            }
    //        输出结果:
    //        Student [name=Bob YANG, age=22]
    //        Student [name=Bruce LEE, age=60]
    //        Student [name=Hao LUO, age=33]
    //        Student [name=XJ WANG, age=32]
        }
}
```

299. sleep()和 wait()有什么区别?

答：sleep()方法是线程类（Thread）的静态方法，导致此线程暂停执行指定时间，将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（线程回到就绪（ready）状态），因为调用 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(线程暂停执行)，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就绪状态。

补充：这里似乎漏掉了一个作为先决条件的问题，就是**什么是进程，什么是线程**？为什么需要多线程编程？答案如下所示：

进程是具有一定独立功能的程序关于某个数据集合上的一次运行活动，是操作系统进行资源分配和调度的一个独立单位 ;线程是进程的一个实体 ,是 CPU 调度和分派的基本单位，是比进程更小的能独立运行的基本单位。线程的划分尺度小于进程，这使得多线程程序的并发性高；进程在执行时通常拥有独立的内存单元，而线程之间可以共享内存。使用多线程的编程通常能够带来更好的性能和用户体验，但是多线程的程序对于其他程序是不友好的，因为它占用了更多的 CPU 资源。

300. sleep()和 yield()有什么区别?

答：

① sleep()方法给其他线程运行机会时不考虑线程的优先级，因此会给低优先级的线程以运行的机会；yield()方法只会给相同优先级或更高优先级的线

程以运行的机会；

② 线程执行 sleep()方法后转入阻塞（blocked）状态，而执行 yield()方法后转入就绪（ready）状态；

③ sleep()方法声明抛出 InterruptedException，而 yield()方法没有声明任何异常；

④ sleep()方法比 yield()方法（跟操作系统相关）具有更好的可移植性。

301. 当一个线程进入一个对象的 synchronized 方法 A 之后，其它线程是否可进入此对象的 synchronized 方法？

答：不能。其它线程只能访问该对象的非同步方法，同步方法则不能进入。

只有等待当前线程执行完毕释放锁资源之后，其他线程才有可能进行执行该同步方法！

延伸 对象锁分为三种：共享资源、this、当前类的字节码文件对象

302. 请说出与线程同步相关的方法。

答：

1. wait():使一个线程处于等待（阻塞）状态，并且释放所持有的对象的锁；
2. 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；
3. notify():唤醒一个处于等待状态的线程，当然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，而且与优先级无关；
4. notityAll():唤醒所有处入等待状态的线程，注意并不是给所有唤醒线程一个对象的锁，而是让它们竞争；

5. JDK 1.5 通过 Lock 接口提供了显式(explicit)的锁机制，增强了灵活性以及对线程的协调。Lock 接口中定义了加锁（lock()）和解锁(unlock())的方法，同时还提供了 newCondition()方法来产生用于线程之间通信的 Condition 对象；
6. JDK 1.5 还提供了信号量(semaphore)机制，信号量可以用来限制对某个共享资源进行访问的线程的数量。在对资源进行访问之前，线程必须得到信号量的许可（调用 Semaphore 对象的 acquire()方法）；在完成对资源的访问后，线程必须向信号量归还许可（调用 Semaphore 对象的 release()方法）。

下面的例子演示了 100 个线程同时向一个银行账户中存入 1 元钱，在没有使用同步机制和使用同步机制情况下的执行情况。

银行账户类：

```
package com.bjsxt;
/**
 * 银行账户
 * @author sxt
 *
 */
public class Account {
    private double balance;       // 账户余额
        /**
         * 存款
```

```
        * @param money  存入金额
        */
       public void deposit(double money) {
           double newBalance = balance + money;
           try {
               Thread.sleep(10);    // 模拟此业务需要一段处理时间
           }
           catch(InterruptedException ex) {
               ex.printStackTrace();
           }
           balance = newBalance;
       }
       /**
        * 获得账户余额
        */
       public double getBalance() {
           return balance;
       }
}
```

存钱线程类：

```
package com.bjsxt;
```

```
/**
 * 存钱线程
 * @author sxt李端阳
 *
 */
public class AddMoneyThread implements Runnable {
    private Account account;      // 存入账户
        private double money;          // 存入金额

        public AddMoneyThread(Account account, double money) {
            this.account = account;
            this.money = money;
        }

        @Override
        public void run() {
            account.deposit(money);
        }
    }
```

测试类：

```
package com.bjsxt;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;


 public class Test01 {

    public static void main(String[] args) {

          Account account = new Account();

              ExecutorService service =

Executors.newFixedThreadPool(100);

              for(int i = 1; i <= 100; i++) {

                   service.execute(new AddMoneyThread(account, 1));

              }

              service.shutdown();

              while(!service.isTerminated()) {}

              System.out.println("账户余额: " + account.getBalance());

         }

    }
```

在没有同步的情况下，执行结果通常是显示账户余额在 10 元以下，出现这种状况的原因是，当一个线程 A 试图存入 1 元的时候，另外一个线程 B 也能够进入存款的方法中，线程 B 读取到的账户余额仍然是线程 A 存入 1 元钱之前的账户余额，因此也是在原来的余额 0 上面做了加 1 元的操作，同理

线程 C 也会做类似的事情，所以最后 100 个线程执行结束时，本来期望账户余额为 100 元，但实际得到的通常在 10 元以下。解决这个问题的办法就是同步，当一个线程对银行账户存钱时，需要将此账户锁定，待其操作完成后才允许其他的线程进行操作，代码有如下几种调整方案：

1. 在银行账户的存款（deposit）方法上同步（synchronized）关键字

```
package com.bjsxt;
/**
 * 银行账户
 * @author SXT李端阳
*/
public class Account {
    private double balance;        // 账户余额
    /**
     * 存款
     * @param money 存入金额
     */
    public synchronized void deposit(double money) {
        double newBalance = balance + money;
        try {
            Thread.sleep(10);    // 模拟此业务需要一段处理时间
        }
        catch(InterruptedException ex) {
```

```
                ex.printStackTrace();
            }
            balance = newBalance;
        }

        /**
         * 获得账户余额
         */
        public double getBalance() {
            return balance;
        }
}
```

2. 在线程调用存款方法时对银行账户进行同步

```
package com.bjsxt;
/**
 * 存钱线程
 * @author SXT
 *
 */
public class AddMoneyThread implements Runnable {
```

```
    private Account account;      // 存入账户
        private double money;          // 存入金额


        public AddMoneyThread(Account account, double money) {
            this.account = account;
            this.money = money;
        }
        @Override
        public void run() {
            synchronized (account) {
                account.deposit(money);
            }
        }
    }
```

3. 通过 JDK 1.5 显示的锁机制，为每个银行账户创建一个锁对象，在存款操作进行加锁和解锁的操作

```
package com.bjsxt;

import java.util.concurrent.locks.Lock;

import java.util.concurrent.locks.ReentrantLock;


/**
```

```
* 银行账户
*
* @author SXT李端阳
    *
    */
   public class Account {
       private Lock accountLock = new ReentrantLock();
       private double balance; // 账户余额
      /**
        * 存款
        *
        * @param money
        *                存入金额
        */
       public void deposit(double money) {
           accountLock.lock();
           try {
               double newBalance = balance + money;
               try {
                   Thread.sleep(10); // 模拟此业务需要一段处理时间
               }
               catch (InterruptedException ex) {
```

```
                    ex.printStackTrace();
                }
                balance = newBalance;
            }
            finally {
                accountLock.unlock();
            }
        }
        /**
         * 获得账户余额
         */
        public double getBalance() {
            return balance;
        }
}
```

按照上述三种方式对代码进行修改后，重写执行测试代码 Test01，将看到最终的账户余额为 100 元。

303. 编写多线程程序有几种实现方式？

答：Java 5 以前实现多线程有两种实现方法：一种是继承 Thread 类；另一种是实现 Runnable 接口。两种方式都要通过重写 run()方法来定义线程的

行为，推荐使用后者，因为 Java 中的继承是单继承，一个类有一个父类，如果继承了 Thread 类就无法再继承其他类了，同时也可以实现资源共享，显然使用 Runnable 接口更为灵活。

补充：Java 5 以后创建线程还有第三种方式：实现 Callable 接口，该接口中的 call 方法可以在线程执行结束时产生一个返回值，代码如下所示：

```
package com.bjsxt;

import java.util.ArrayList;

import java.util.List;

import java.util.concurrent.Callable;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.Future;


  class MyTask implements Callable<Integer> {

        private int upperBounds;


        public MyTask(int upperBounds) {

            this.upperBounds = upperBounds;

        }


        @Override

        public Integer call() throws Exception {
```

```
            int sum = 0;
            for(int i = 1; i <= upperBounds; i++) {
                sum += i;
            }
            return sum;
        }

    }

    public class Test {
        public static void main(String[] args) throws Exception {
            List<Future<Integer>> list = new ArrayList<>();
            ExecutorService service = Executors.newFixedThreadPool(10);
            for(int i = 0; i < 10; i++) {
            list.add(service.submit(new MyTask((int) (Math.random() *
100))));
            }
            int sum = 0;
            for(Future<Integer> future : list) {
                while(!future.isDone()) ;
                sum += future.get();
            }
```

```
            System.out.println(sum);
        }
    }
```

304. synchronized 关键字的用法？

答：synchronized 关键字可以将对象或者方法标记为同步，以实现对对象和方法的互斥访问，可以用 synchronized(对象) { … }定义同步代码块，或者在声明方法时将 synchronized 作为方法的修饰符。在第 60 题的例子中已经展示了 synchronized 关键字的用法。

305. 举例说明同步和异步

答：如果系统中存在临界资源（资源数量少于竞争资源的线程数量的资源），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，那么这些数据就必须进行同步存取（**数据库**操作中的悲观锁就是最好的例子）。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。事实上，所谓的同步就是指阻塞式操作，而异步就是非阻塞式操作。

306. 启动一个线程是用 run()还是 start()方法?

答：启动一个线程是调用 start()方法，使线程所代表的虚拟处理机处于可运行状态，这意味着它可以由 JVM 调度并执行，这并不意味着线程就会立即运行。run()方法是线程启动后要进行回调（callback）的方法。

API 解释如下：

void	start() 使该线程开始执行；Java 虚拟机调用该线程的 run 方法。

307. 什么是线程池（thread pool）？

答：在面向对象编程中，创建和销毁对象是很费时间的，因为创建一个对象要获取内存资源或者其它更多资源。在 Java 中更是如此，虚拟机将试图跟踪每一个对象，以便能够在对象销毁后进行垃圾回收。所以提高服务程序效率的一个手段就是尽可能减少创建和销毁对象的次数，特别是一些很耗资源的对象创建和销毁，这就是"池化资源"技术产生的原因。线程池顾名思义就是事先创建若干个可执行的线程放入一个池（容器）中，需要的时候从池中获取线程不用自行创建，使用完毕不需要销毁线程而是放回池中，从而减少创建和销毁线程对象的开销。

Java 5+中的 Executor 接口定义一个执行线程的工具。它的子类型即线程池接口是 ExecutorService。要配置一个线程池是比较复杂的，尤其是对于线程池的原理不是很清楚的情况下，因此在工具类 Executors 面提供了一些静态工厂方法，生成一些常用的线程池，如下所示：

- newSingleThreadExecutor：创建一个单线程的线程池。这个线程池只有一个线程在工作，也就是相当于单线程串行执行所有任务。如果这个唯一的线程因为异常结束，那么会有一个新的线程来替代它。此线程池保证所有任务的执行顺序按照任务的提交顺序执行。
- newFixedThreadPool：创建固定大小的线程池。每次提交一个任务就创建一个线程，直到线程达到线程池的最大大小。线程池的大小一旦达到

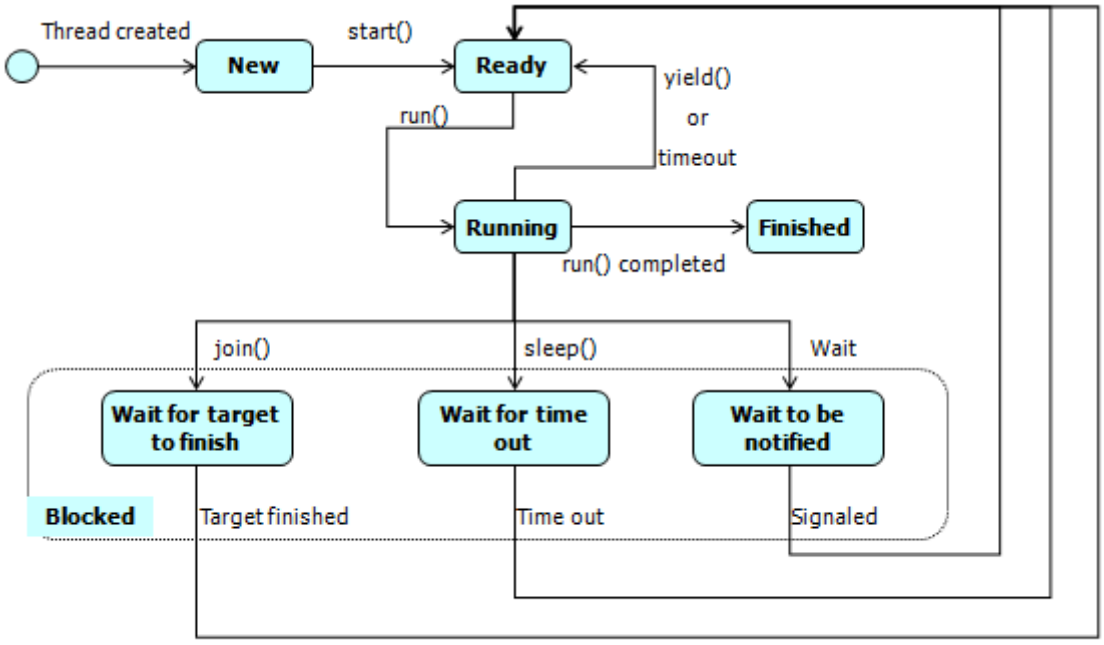
最大值就会保持不变，如果某个线程因为执行异常而结束，那么线程池会补充一个新线程。

- newCachedThreadPool：创建一个可缓存的线程池。如果线程池的大小超过了处理任务所需要的线程，那么就会回收部分空闲（60 秒不执行任务）的线程，当任务数增加时，此线程池又可以智能的添加新线程来处理任务。此线程池不会对线程池大小做限制，线程池大小完全依赖于操作系统（或者说 JVM）能够创建的最大线程大小。
- newScheduledThreadPool：创建一个大小无限的线程池。此线程池支持定时以及周期性执行任务的需求。
- newSingleThreadExecutor：创建一个单线程的线程池。此线程池支持定时以及周期性执行任务的需求。

有通过 Executors 工具类创建线程池并使用线程池执行线程的代码。如果希望在服务器上使用线程池，强烈建议使用 newFixedThreadPool 方法来创建线程池，这样能获得更好的性能。

308. 线程的基本状态以及状态之间的关系？

答：



线程的生命周期图

除去起始（new）状态和结束（finished）状态，线程有三种状态，分别是：就绪（ready）、运行（running）和阻塞（blocked）。其中就绪状态代表线程具备了运行的所有条件，只等待 CPU 调度（万事俱备，只欠东风）；处于运行状态的线程可能因为 CPU 调度（时间片用完了）的原因回到就绪状态，也有可能因为调用了线程的 yield 方法回到就绪状态，此时线程不会释放它占有的资源的锁，坐等 CPU 以继续执行；运行状态的线程可能因为 I/O 中断、线程休眠、调用了对象的 wait 方法而进入阻塞状态（有的地方也称之为等待状态）；而进入阻塞状态的线程会因为休眠结束、调用了对象的 notify 方法或 notifyAll 方法或其他线程执行结束而进入就绪状态。注意：调用 wait 方法会让线程进入等待池中等待被唤醒，notify 方法或 notifyAll 方法会让等待锁中的线程从等待池进入等锁池，在没有得到对象的锁之前，线程仍然无法获得 CPU 的调度和执行。

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

答：Lock 是 Java 5 以后引入的新的 API，和关键字 synchronized 相比主要相同点：Lock 能完成 synchronized 所实现的所有功能；主要不同点：Lock 有比 synchronized 更精确的线程语义和更好的性能。synchronized 会自动释放锁，而 Lock 一定要求程序员手工释放，并且必须在 finally 块中释放（这是释放外部资源的最好的地方）。

310. Java 中如何实现序列化，有什么意义？

答：序列化就是一种用来处理对象流的机制，所谓对象流也就是将对象的内容进行流化。可以对流化后的对象进行读写操作，也可将流化后的对象传输于网络之间。序列化是为了解决对象流读写操作时可能引发的问题（如果不进行序列化可能会存在数据乱序的问题）。

要实现序列化，需要让一个类实现 Serializable 接口，该接口是一个标识性接口，标注该类对象是可被序列化的，然后使用一个输出流来构造一个对象输出流并通过 writeObject(Object obj)方法就可以将实现对象写出(即保存其状态)；如果需要反序列化则可以用一个输入流建立对象输入流，然后通过 readObject 方法从流中读取对象。序列化除了能够实现对象的持久化之外，还能够用于对象的深度克隆（参见 Java 面试题集 1-29 题）

311. Java 中有几种类型的流？

答：两种流分别是字节流，字符流。

字节流继承于 InputStream、OutputStream，字符流继承于 Reader、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。

补充：关于 Java 的 IO 需要注意的有两点：一是两种对称性（输入和输出的对称性，字节和字符的对称性）；二是两种设计模式（适配器模式和装潢模式）。另外 Java 中的流不同于 C#的是它只有一个维度一个方向。

补充：下面用 IO 和 NIO 两种方式实现文件拷贝，这个题目在面试的时候是经常被问到的。

```
package com.bjsxt;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.InputStream;

import java.io.OutputStream;

import java.nio.ByteBuffer;

import java.nio.channels.FileChannel;


public class MyUtil {

        private MyUtil() {

                throw new AssertionError();

          }


          public static void fileCopy(String source, String target)

throws IOException {
```

```
            try (InputStream in = new FileInputStream(source)) {
                try (OutputStream out = new
FileOutputStream(target)) {
                    byte[] buffer = new byte[4096];
                    int bytesToRead;
                    while((bytesToRead = in.read(buffer)) != -1) {
                        out.write(buffer, 0, bytesToRead);
                    }
                }
            }
        }

        public static void fileCopyNIO(String source, String target)
throws IOException {
            try (FileInputStream in = new FileInputStream(source)) {
                try (FileOutputStream out = new
FileOutputStream(target)) {
                    FileChannel inChannel = in.getChannel();
                    FileChannel outChannel = out.getChannel();
                    ByteBuffer buffer = ByteBuffer.allocate(4096);
                    while(inChannel.read(buffer) != -1) {
                        buffer.flip();
```

```
                    outChannel.write(buffer);
                    buffer.clear();
                }
            }
        }
    }
}
```

注意：上面用到 Java 7 的 TWR，使用 TWR 后可以不用在 finally 中释放外部资源 ，从而让代码更加优雅。

312. 写一个方法，输入一个文件名和一个字符串，统计这个字符串在这个文件中出现的次数。

答：代码如下：

```
package com.bjsxt;
import java.io.BufferedReader;
import java.io.FileReader;

public class Account {
    // 工具类中的方法都是静态方式访问的因此将构造器私有不允许创建对象(绝对好习惯)
    private Account() {
        throw new AssertionError();
```

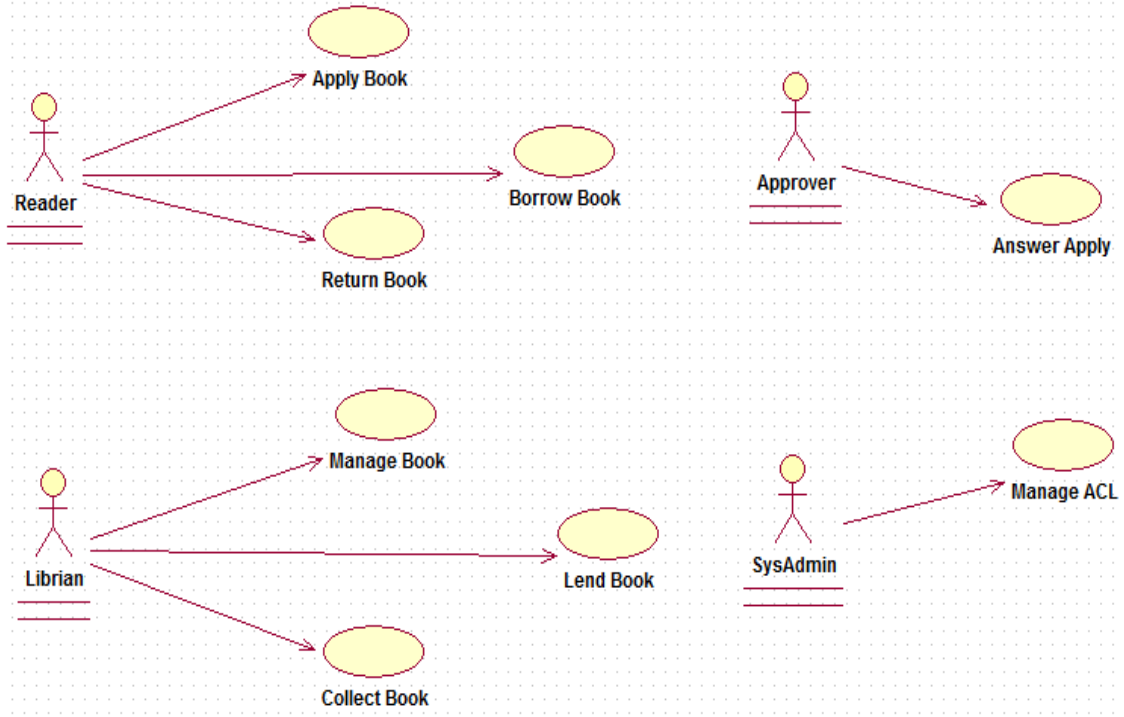
```
        }
        /**
         * 统计给定文件中给定字符串的出现次数
         * @param filename  文件名
         * @param word  字符串
         * @return  字符串在文件中出现的次数
         */
        public static int countWordInFile(String filename, String word) {
            int counter = 0;
            try (FileReader fr = new FileReader(filename)) {
                try (BufferedReader br = new BufferedReader(fr)) {
                    String line = null;
                    while ((line = br.readLine()) != null) {
                        int index = -1;
                        while (line.length() >= word.length() && (index
= line.indexOf(word)) >= 0) {
                            counter++;
                            line = line.substring(index + word.length());
                        }
                    }
                }
            } catch (Exception ex) {
```

```
                ex.printStackTrace();
            }
            return counter;
        }
}
```

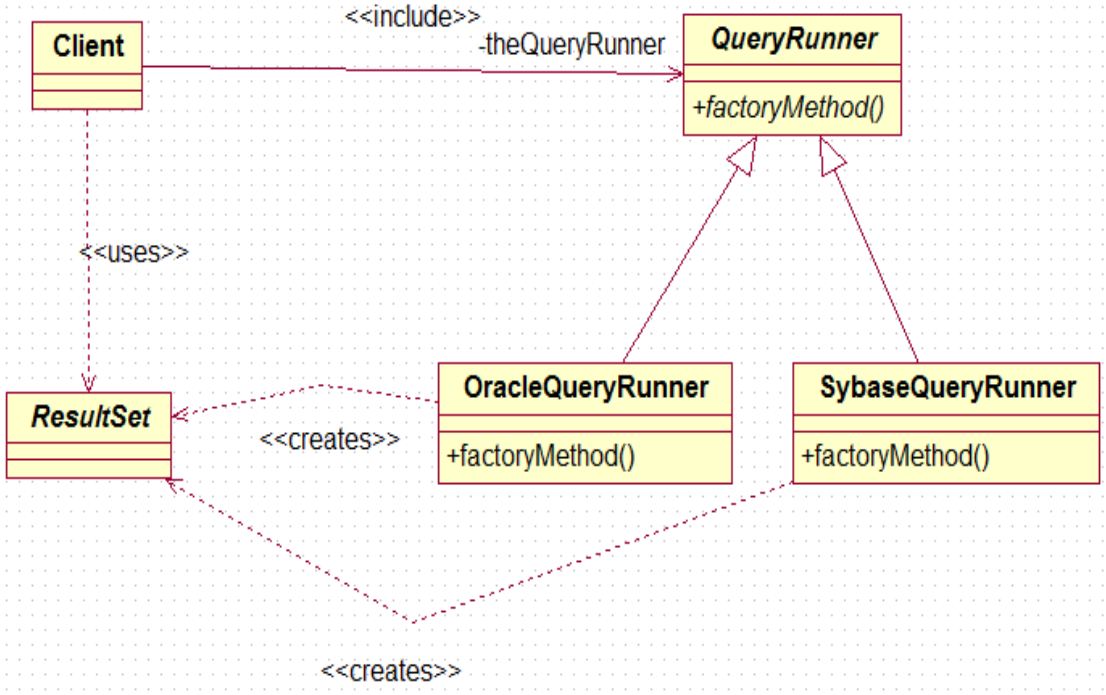
313. UML 是什么？UML 中有哪些图？

答：UML 是统一建模语言（Unified Modeling Language）的缩写，它发表于 1997 年，综合了当时已经存在的面向对象的建模语言、方法和过程，是一个支持模型化和软件系统开发的图形化语言，为软件开发的所有阶段提供模型化和可视化支持。使用 UML 可以帮助沟通与交流，辅助应用设计和文档的生成，还能够阐释系统的结构和行为。UML 定义了多种图形化的符号来描述软件系统部分或全部的静态结构和动态结构，包括：用例图（use case diagram）、类图（class diagram）、时序图（sequence diagram）、协作图（collaboration diagram）、状态图（statechart diagram）、活动图（activity diagram）、构件图（component diagram）、部署图（deployment diagram）等。在这些图形化符号中，有三种图最为重要，分别是：用例图（用来捕获需求，描述系统的功能，通过该图可以迅速的了解系统的功能模块及其关系）、类图（描述类以及类与类之间的关系，通过该图可以快速了解系统）、时序图（描述执行特定任务时对象之间的交互关系以及执行顺序，通过该图可以了解对象能接收的消息也就是说对象能够向外界提供的服务）。

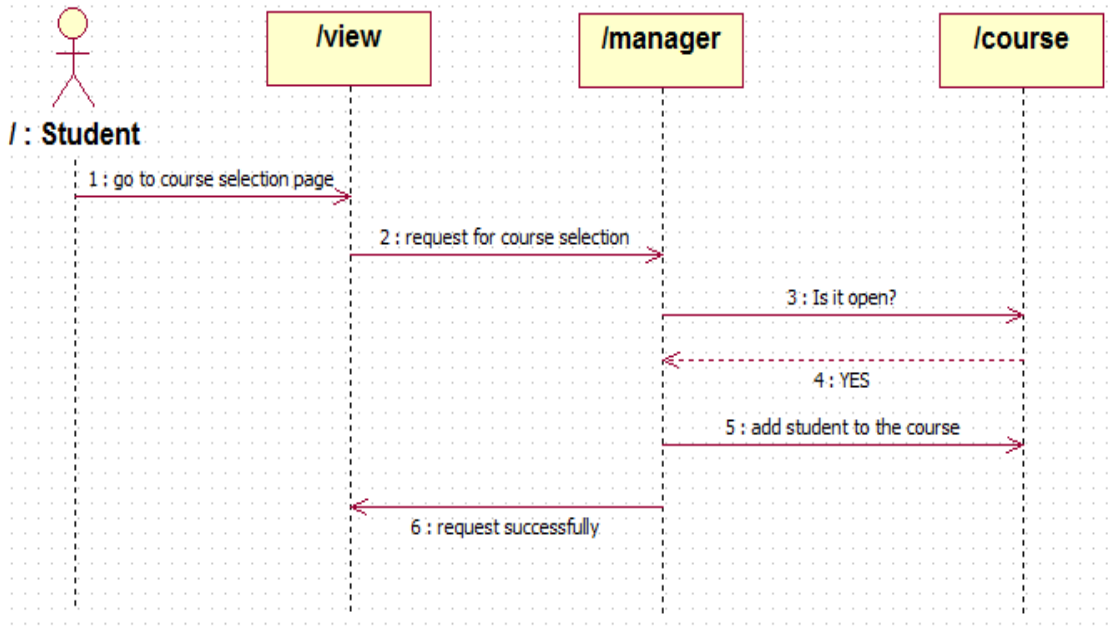
用例图：


Apply Book
Reader
Borrow Book
Return Book
Approver
Answer Apply
Manage Book
Librian
Lend Book
Collect Book
SysAdmin
Manage ACL

类图：


Client
<<include>>
-theQueryRunner
QueryRunner
+factoryMethod()
<<uses>>
ResultSet
<<creates>>
OracleQueryRunner
+factoryMethod()
SybaseQueryRunner
+factoryMethod()
<<creates>>

时序图：



314. 写一个单例类

答：单例模式主要作用是保证在 Java 应用程序中，一个类只有一个实例存在。下面给出两种不同形式的单例：

第一种形式：饿汉式单例

```
package com.bjsxt;


public class Singleton {

    private Singleton(){}

    private static Singleton instance = new Singleton();

    public static Singleton getInstance(){

        return instance;

    }

}
```

第二种形式：懒汉式单例

```
package com.bjsxt;


public class Singleton {
	private static Singleton instance = null;
	private Singleton() {}
	public static synchronized Singleton getInstance(){
		if (instance==null) instance = newSingleton();
		return instance;
	}
}
```

单例的特点：外界无法通过构造器来创建对象，该类必须提供一个静态方法向外界提供该类的唯一实例。

【补充】用 Java 进行服务器端编程时，使用单例模式的机会还是很多的，服务器上的资源都是很宝贵的，对于那些无状态的对象其实都可以单例化或者静态化（在内存中仅有唯一拷贝），如果使用了 spring 这样的框架来进行对象托管，Spring 的 IoC 容器在默认情况下对所有托管对象都是进行了单例化处理的。

315. 说说你所熟悉或听说过的设计模式以及你对设计模式的看法

答：在 GoF 的《Design Patterns: Elements of Reusable Object-Oriented Software》中给出了三类（创建型[对类的实例化过程的抽象化]、结构型[描

述如何将类或对象结合在一起形成更大的结构]、行为型[对在不同的对象之间划分责任和算法的抽象化]）共 23 种设计模式，包括：Abstract Factory（抽象工厂模式），Builder（建造者模式），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 Responsibility（责任链模式）。

所谓设计模式，就是一套被反复使用的代码设计经验的总结（情境中一个问题经过证实的一个解决方案）。使用设计模式是为了可重用代码、让代码更容易被他人理解、保证代码可靠性。设计模式使人们可以更加简单方便的复用成功的设计和体系结构。将已证实的技术表述成设计模式也会使新系统开发者更加容易理解其设计思路。

【补充】设计模式并不是像某些地方吹嘘的那样是遥不可及的编程理念，说白了设计模式就是对面向对象的编程原则的实践，面向对象的编程原则包括：

- 单一职责原则：一个类只做它该做的事情。（单一职责原则想表达的就是“高内聚”，写代码最终极的原则只有六个字“高内聚、低耦合”，就如同葵花宝典或辟邪剑谱的中心思想就八个字“欲练此功必先自宫”，

所谓的高内聚就是一个代码模块只完成一项功能，在面向对象中，如果只让一个类完成它该做的事，而不涉及与它无关的领域就是践行了高内聚的原则，这个类就只有单一职责。我们都知道一句话叫“因为专注，所以专业”，一个对象如果承担太多的职责，那么注定它什么都做不好。这个世界上任何好的东西都有两个特征，一个是功能单一，好的相机绝对不是电视购物里面卖的那种一个机器有一百多种功能的，它基本上只能照相；另一个是模块化，好的自行车是组装车，从减震叉、刹车到变速器，所有的部件都是可以拆卸和重新组装的，好的乒乓球拍也不是成品拍，一定是底板和胶皮可以拆分和自行组装的，一个好的软件系统，它里面的每个功能模块也应该是可以轻易的拿到其他系统中使用的，这样才能实现软件复用的目标。）

- 开闭原则：软件实体应当对扩展开放，对修改关闭。（在理想的状态下，当我们需要为一个软件系统增加新功能时，只需要从原来的系统派生出一些新类就可以，不需要修改原来的任何一行代码。要做到开闭有两个要点：①抽象是关键，一个系统中如果没有抽象类或接口系统就没有扩展点；②封装可变性，将系统中的各种可变因素封装到一个继承结构中，如果多个可变因素混杂在一起，系统将变得复杂而换乱，如果不清楚如何封装可变性，可以参考《设计模式精解》一书中对桥梁模式的讲解的章节。）
- 依赖倒转原则：面向接口编程。（该原则说得直白和具体一些就是声明方法的参数类型、方法的返回类型、变量的引用类型时，尽可能使用抽象

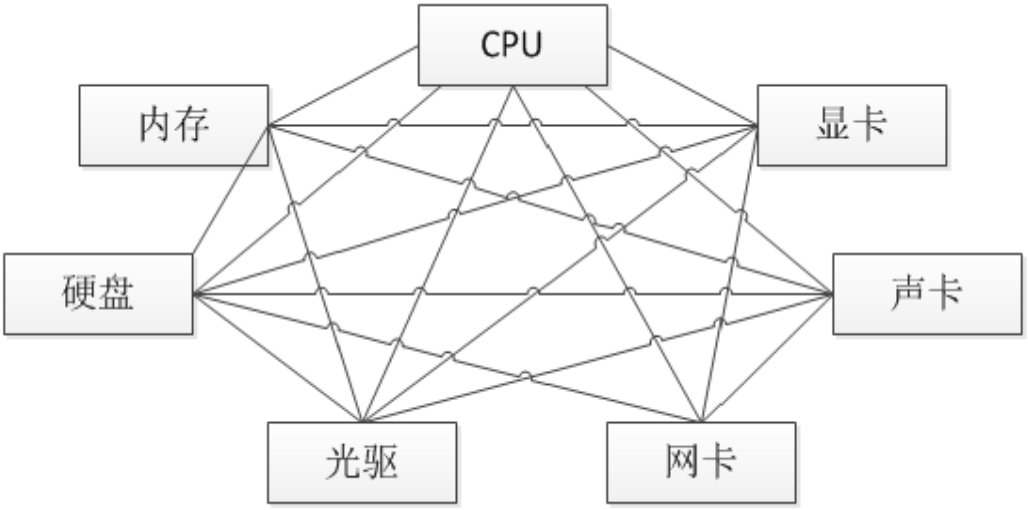
类型而不用具体类型，因为抽象类型可以被它的任何一个子类型所替代，请参考下面的里氏替换原则。）

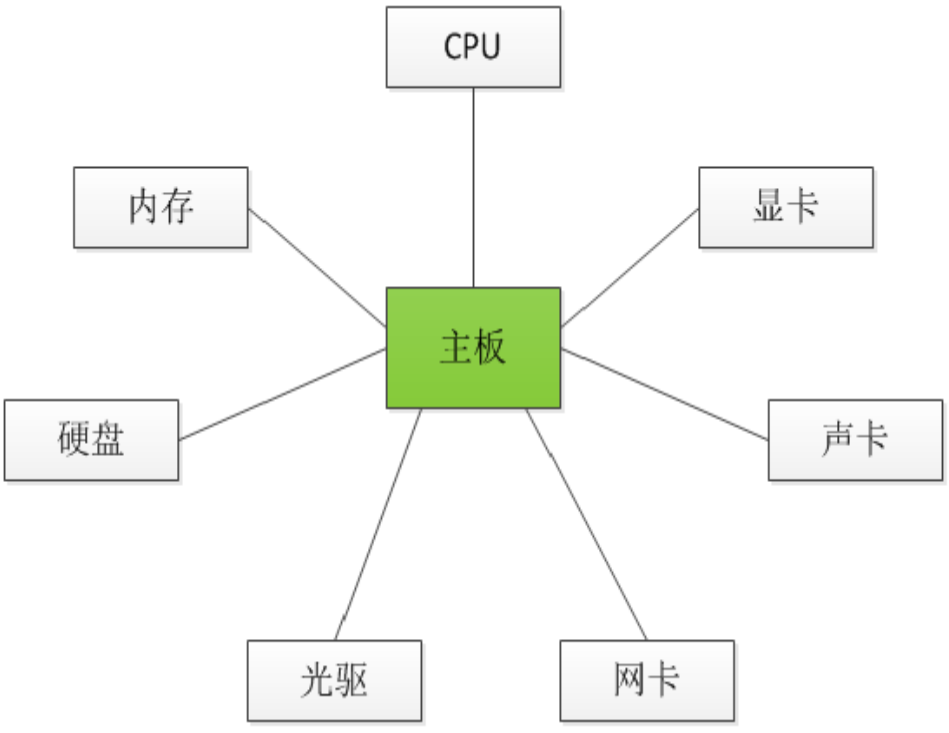
- 里氏替换原则：任何时候都可以用子类型替换掉父类型。（关于里氏替换原则的描述，Barbara Liskov 女士的描述比这个要复杂得多，但简单的说就是能用父类型的地方就一定能使用子类型。里氏替换原则可以检查继承关系是否合理，如果一个继承关系违背了里氏替换原则，那么这个继承关系一定是错误的，需要对代码进行重构。例如让猫继承狗，或者狗继承猫，又或者让正方形继承长方形都是错误的继承关系，因为你很容易找到违反里氏替换原则的场景。需要注意的是：子类一定是增加父类的能力而不是减少父类的能力，因为子类比父类的能力更多，把能力多的对象当成能力少的对象来用当然没有任何问题。）
- 接口隔离原则：接口要小而专，绝不能大而全。（臃肿的接口是对接口的污染，既然接口表示能力，那么一个接口只应该描述一种能力，接口也应该是高度内聚的。例如，琴棋书画就应该分别设计为四个接口，而不应设计成一个接口中的四个方法，因为如果设计成一个接口中的四个方法，那么这个接口很难用，毕竟琴棋书画四样都精通的人还是少数，而如果设计成四个接口，会几项就实现几个接口，这样的话每个接口被复用的可能性是很高的。Java 中的接口代表能力、代表约定、代表角色，能否正确的使用接口一定是编程水平高低的重要标识。）
- 合成聚合复用原则：优先使用聚合或合成关系复用代码。（通过继承来复用代码是面向对象程序设计中被滥用得最多的东西，因为所有的教科书都无一例外的对继承进行了鼓吹从而误导了初学者，类与类之间简单的

说有三种关系，IS-A 关系、HAS-A 关系、USE-A 关系，分别代表继承、关联和依赖。其中，关联关系根据其关联的强度又可以进一步划分为关联、聚合和合成，但说白了都是 HAS-A 关系，合成聚合复用原则想表达的是优先考虑 HAS-A 关系而不是 IS-A 关系复用代码，原因嘛可以自己从百度上找到一万个理由，需要说明的是，即使在 Java 的 API 中也有不少滥用继承的例子，例如 Properties 类继承了 Hashtable 类，Stack 类继承了 Vector 类，这些继承明显就是错误的，更好的做法是在 Properties 类中放置一个 Hashtable 类型的成员并且将其键和值都设置为字符串来存储数据，而 Stack 类的设计也应该是在 Stack 类中放一个 Vector 对象来存储数据。记住：任何时候都不要继承工具类，工具是可以拥有并可以使用的（HAS/USE），而不是拿来继承的。）

- 迪米特法则：迪米特法则又叫最少知识原则，一个对象应当对其他对象有尽可能少的了解。（迪米特法则简单的说就是如何做到“低耦合”，门面模式和调停者模式就是对迪米特法则的践行。对于门面模式可以举一个简单的例子，你去一家公司洽谈业务，你不需要了解这个公司内部是如何运作的，你甚至可以对这个公司一无所知，去的时候只需要找到公司入口处的前台美女，告诉她们你要做什么，她们会找到合适的人跟你接洽，前台的美女就是公司这个系统的门面。再复杂的系统都可以为用户提供一个简单的门面，Java Web 开发中作为前端控制器的 Servlet 或 Filter 不就是一个门面吗，浏览器对服务器的运作方式一无所知，但是通过前端控制器就能够根据你的请求得到相应的服务。调停者模式也可以举一个简单的例子来说明，例如一台计算机，CPU、内存、硬盘、显卡、

声卡各种设备需要相互配合才能很好的工作，但是如果这些东西都直接连接到一起，计算机的布线将异常复杂，在这种情况下，主板作为一个调停者的身份出现，它将各个设备连接在一起而不需要每个设备之间直接交换数据，这样就减小了系统的耦合度和复杂度。迪米特法则用通俗的话来将就是不要和陌生人打交道，如果真的需要，找一个自己的朋友，让他替你和陌生人打交道。）





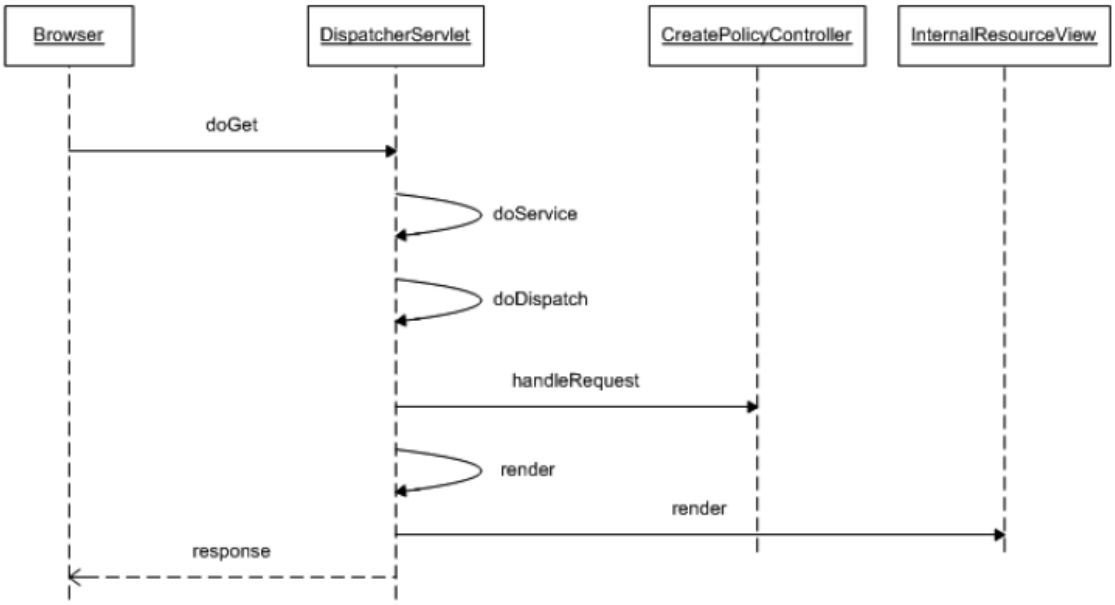
316. Java 企业级开发中常用的设计模式有哪些？

答： 按照分层开发的观点，可以将应用划分为：表示层、业务逻辑层和持久层，每一层都有属于自己类别的设计模式。

表示层设计模式：

1)Interceptor Filter：拦截过滤器，提供请求预处理和后处理的方案，可以对请求和响应进行过滤。

2)Front Controller：通过中央控制器提供请求管理和处理，管理内容读取、安全性、视图管理和导航等功能。Struts 2 中的 StrutsPrepareAndExecuteFilter、Spring MVC 中的 DispatcherServlet 都是前端控制器，后者如下图所示：



3)View Helper：视图帮助器，负责将显示逻辑和业务逻辑分开。显示的部分放在视图组件中，业务逻辑代码放在帮助器中，典型的功能是内容读取、验证与适配。

4)Composite View：复合视图。

业务逻辑层设计模式：

1)Business Delegate：业务委托，减少表示层和业务逻辑层之间的耦合。

2)Value Object：值对象，解决层之间交换数据的开销问题。

3)Session Façade：会话门面，隐藏业务逻辑组件的细节，集中工作流程。

4)Value Object Assembler：灵活的组装不同的值对象

5)Value List Handler：提供执行查询和处理结果的解决方案，还可以缓存查询结果，从而达到提升性能的目的。

6)Service Locator：服务定位器，可以查找、创建和定位服务工厂，封装其实现细节，减少复杂性，提供单个控制点，通过缓存提高性能。

持久层设计模式：

1)Data Access Object：数据访问对象，以面向对象的方式完成对数据的增删改查。

【补充】如果想深入的了解 Java 企业级应用的设计模式和架构模式，可以参考这些书籍：《Pro Java EE Spring Patterns》、《POJO in Action》、《Patterns of Enterprise Application Architecture》。

你在开发中都用到了那些设计模式？用在什么场合？

答：面试被问到关于设计模式的知识时，可以拣最常用的作答，例如：

1)工厂模式：工厂类可以根据条件生成不同的子类实例，这些子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（多态方法）。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。

2)代理模式：给一个对象提供一个代理对象，并由代理对象控制原对象的引

用。实际开发中，按照使用目的的不同，代理可以分为：远程代理、虚拟代理、保护代理、Cache 代理、防火墙代理、同步化代理、智能引用代理。

3)适配器模式：把一个类的接口变换成客户端所期待的另一种接口，从而使原本因接口不匹配而无法在一起使用的类能够一起工作。

4)模板方法模式：提供一个抽象类，将部分逻辑以具体方法或构造器的形式实现，然后声明一些抽象方法来迫使子类实现剩余的逻辑。不同的子类可以以不同的方式实现这些抽象方法（多态实现），从而实现不同的业务逻辑。

除此之外，还可以讲讲上面提到的门面模式、桥梁模式、单例模式、装潢模式(Collections 工具类里面的 synchronizedXXX 方法把一个线程不安全的容器变成线程安全容器就是对装潢模式的应用，而 Java IO 里面的过滤流(有的翻译成处理流）也是应用装潢模式的经典例子）等，反正原则就是拣自己最熟悉的用得最多的作答，以免言多必失。

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

答：XML 文档定义分为 DTD 和 Schema 两种形式，其本质区别在于 Schema 本身也是一个 XML 文件，可以被 XML 解析器解析。对 XML 的解析主要有 DOM（文档对象模型）、SAX、StAX（JDK 1.6 中引入的新的解析 XML 的方式，Streaming API for XML） 等，其中 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（典型的用空间换取时间的策略）；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；如其名称所暗示的那样，StAX 把重点放在流上。实际上，StAX 与其他方法的区别就在于应用程序能够把 XML 作为一个事件流来处理。将 XML 作为一组事件来处理的想法并不新颖（事实上 SAX 已经提出来了），但不同之处在于 StAX 允许应用程序代码把这些事件逐个拉出来，而不用提供在解析器方便时从解析器中接收事件的处理程序。

你在项目中哪些地方用到了 XML？

答:XML 的主要作用有两个方面：数据交换（曾经被称为业界数据交换的事实标准，现在此项功能在很多时候都被 JSON 取代）和信息配置。在做数据交换时，XML 将数据用标签组装成起来，然后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从 XML 文件中还原相关信息进行处理。目前很多软件都使用 XML 来存储配置信息，很多项目中我们通常也会将作为配置的硬代码（hard code）写在 XML 文件中，Java 的很多框架也是这么做的。

在进行数据库编程时，连接池有什么作用？

答：由于创建连接和释放连接都有很大的开销（尤其是数据库服务器不在本地时，每次建立连接都需要进行 TCP 的三次握手，再加上网络延迟，造成的开销是不可忽视的），为了提升系统访问数据库的性能，可以事先创建若干连接置于连接池中，需要时直接从连接池获取，使用结束时归还连接池而不必关闭连接，从而避免频繁创建和释放连接所造成的开销，这是典型的用空间换取时间的策略（浪费了空间存储连接，但节省了创建和释放连接的时

间）。池化技术在 Java 开发中是很常见的，在使用线程时创建线程池的道理与此相同。基于 Java 的开源数据库连接池主要有：C3P0、Proxool、DBCP、BoneCP、Druid 等。

【补充】在计算机系统中时间和空间是不可调和的矛盾，理解这一点对设计满足性能要求的算法是至关重要的。大型网站性能优化的一个关键就是使用缓存，而缓存跟上面讲的连接池道理非常类似，也是使用空间换时间的策略。可以将热点数据置于缓存中，当用户查询这些数据时可以直接从缓存中得到，这无论如何也快过去数据库中查询。当然，缓存的置换策略等也会对系统性能产生重要影响，对于这个问题的讨论已经超出了这里要阐述的范围。

什么是 DAO 模式？

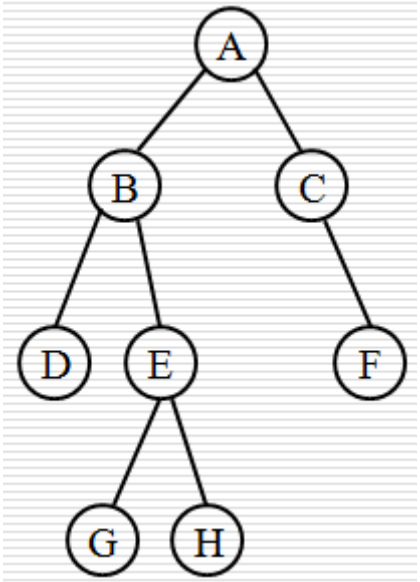
答：DAO（DataAccess Object）顾名思义是一个为数据库或其他持久化机制提供了抽象接口的对象，在不暴露数据库实现细节的前提下提供了各种数据操作。为了建立一个健壮的 Java EE 应用，应该将所有对数据源的访问操作进行抽象化后封装在一个公共 API 中。用程序设计语言来说，就是建立一个接口，接口中定义了此应用程序中将会用到的所有事务方法。在这个应用程序中，当需要和数据源进行交互的时候则使用这个接口，并且编写一个单独的类来实现这个接口，在逻辑上该类对应一个特定的数据存储。DAO 模式实际上包含了两个模式，一是 Data Accessor（数据访问器），二是 Data Object（数据对象），前者要解决如何访问数据的问题，而后者要解决的是如何用对象封装数据。

什么是 ORM？

答：对象关系映射（Object-Relational Mapping，简称 ORM）是一种为

了解决程序的面向对象模型与数据库的关系模型互不匹配问题的技术；简单的说，ORM 是通过使用描述对象和数据库之间映射的元数据（可以用 XML 或者是注解），将 Java 程序中的对象自动持久化到关系数据库中或者将关系数据库表中的行转换成 Java 对象，其本质上就是将数据从一种形式转换到另外一种形式。

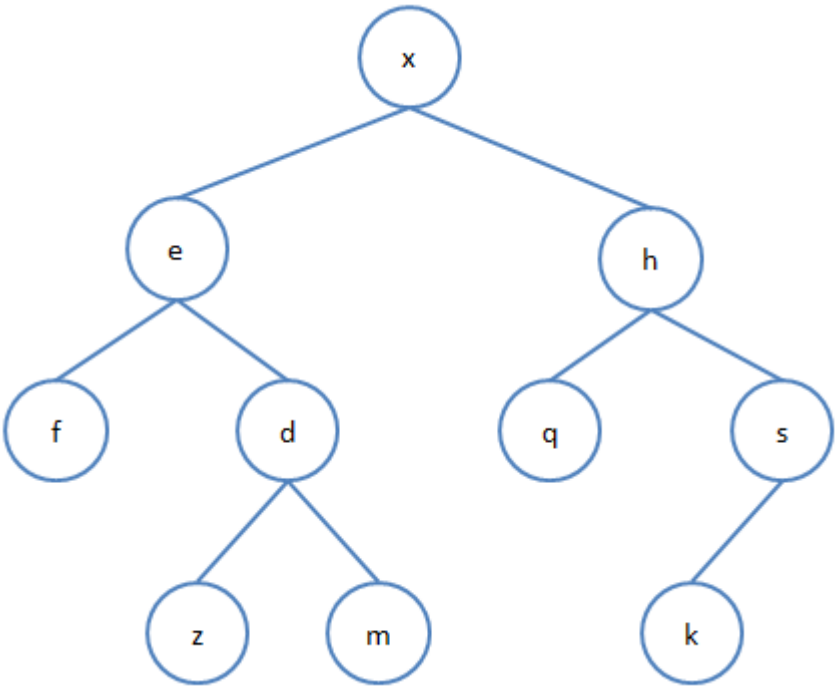
给出下面的二叉树先序、中序、后序遍历的序列？



答：先序序列：ABDEGHCF；中序序列：DBGEHACF；后序序列：DGHEBFCA。

补充：二叉树也称为二分树，它是树形结构的一种，其特点是每个结点至多有二棵子树，并且二叉树的子树有左右之分，其次序不能任意颠倒。二叉树的遍历序列按照访问根节点的顺序分为先序（先访问根节点，接下来先序访问左子树，再先序访问右子树）、中序（先中序访问左子树，然后访问根节点，最后中序访问右子树）和后序（先后序访问左子树，再后序访问右子树，最后访问根节点）。如果知道一棵二叉树的先序和中序序列或者中序和后序序列，那么也可以还原出该二叉树。

例如，已知二叉树的先序序列为：xefdzmhqsk，中序序列为：fezdmxqhks，那么还原出该二叉树应该如下图所示：



不同点	Struts1	Struts2
Action 类	要求 Action 类继承一个抽象基类。Struts1 的一个普遍问题是使用抽象类编程而不是接口。	Action 类可以实现一个 Action 接口，也可实现其他接口，使可选和定制的服务成为可能。Struts2 提供一个 ActionSupport 基类去实现 常用的接口。Action 接口不是必须的，任何有 execute 标识的 POJO 对象都可以用作 Struts2 的 Action 对象。
线程模式	Action 是单例模式并且必须是线程安全的，因为仅有 Action 的一个实例来处理所有的请求。单例策略限制了 Struts1 Action 能作的事，并且要在开发时特别小心。Action 资源必须是线程安全的或同步的。	Action 对象为每一个请求产生一个实例，因此没有线程安全问题。（实际上，Servlet 容器给每个请求产生许多可丢弃的对象，并且不会导致性能和垃圾回收问题）
Servlet 依赖	Action 依赖于 Servlet API ,当一个 Action 被调用时 HttpServletRequest 和 HttpServletResponse 被传递给 execute 方法。	Action 不依赖于容器，允许 Action 脱离容器单独被测试。如果需要，Struts2 Action 仍然可以访问初始的 request 和 response。但是，其他的元素减少或者消除了直接访问 HttpServetRequest 和 HttpServletResponse 的必要性。

可测性	Action 的一个主要问题是 execute 方法暴露了 Servlet API（这使得测试要依赖于容器）。	Action 可以通过初始化、设置属性、调用方法来测试，“依赖注入”支持也使测试更容易。
捕获输入	使用 ActionForm 对象捕获输入。所有的 ActionForm 必须继承一个基类。因为其他 JavaBean 不能用作 ActionForm，开发者经 常创建多余的类捕获输入。	直接使用 Action 属性作为输入属性，消除了对第二个输入对象的需求。输入属性可能是有自己(子)属性的对象类型。Struts2 也支持 ActionForm 模式。

表达式语言	整合了 JSTL，因此使用 JSTL EL。这种 EL 有基本对象图遍历，但是对集合和索引属性的支持很弱。	可以使用 JSTL，但是也支持一个更强大和灵活的表达式语言－－"Object Graph Notation Language" (OGNL).
绑定值到页面	使用标准 JSP 机制把对象绑定到页面中来访问	使用 "ValueStack"技术，使 taglib 能够访问值而不需要把你的页面（view）和对象绑定起来。ValueStack 策略允许通过一系列名称相同但类型不同的属性重用页面（view）。
类型转换	使用 Commons-Beanutils 进行类型转换。每个类一个转换器，对每一个实例来说是不可配置的。	使用 OGNL 进行类型转换。提供基本和常用对象的转换器。
校验	支持在 ActionForm 的 validate 方法中手动校验，或者通过 Commons Validator 的扩展来校验。同一个类可以有不同的校验内容，但不能校验子对象。	支持通过 validate 方法和 XWork 校验框架来进行校验。XWork 校验框架使用为属性类类型定义的校验和内容校验，来支持 chain 校验子属性
Action 执行控制	支持每一个模块有单独的 Request Processors（生命周期），但是模块中的所有 Action 必须共享相同的生命周期。	支持通过拦截器堆栈（Interceptor Stacks）为每一个 Action 创建不同的生命周期。堆栈能够根据需要和不同的 Action 一起使用。

你知道的排序算法都哪些？用 Java 写一个排序系统

答：稳定的排序算法有：插入排序、选择排序、冒泡排序、鸡尾酒排序、归并排序、二叉树排序、基数排序等；不稳定排序算法包括：希尔排序、堆排序、快速排序等。

下面是关于排序算法的一个列表：

交换排序	冒泡排序 · 鸡尾酒排序 · 奇偶排序 · 梳排序 · 侏儒排序 · 快速排序 · 臭皮匠排序 · B
选择排序	选择排序 · 堆排序 · Smooth排序 · 笛卡尔树排序 · 锦标赛排序 · 循环排序
插入排序	插入排序 · 希尔排序 · 二叉查找树排序 · 图书馆排序 · Patience排序
归并排序	归并排序 · 梯级归并排序 · 振荡归并排序 · 多相归并排序 · Strand排序
分布排序	美国旗帜排序 · 珠排序 · 桶排序 · 爆炸排序 · 计数排序 · 鸽巢排序 · 相邻图排序 ·
并发排序	双调排序器 · Batcher归并网络 · 两两排序网络
混合排序	Tim排序 · 内省排序 · Spread排序 · 反移排序 · J排序
其他	拓扑排序 · 煎饼排序 · 意粉排序

下面按照策略模式给出一个排序系统，实现了冒泡、归并和快速排序。

```
package com.bjsxt;


import java.util.Comparator;


/**
 * 排序器接口(策略模式: 将算法封装到具有共同接口的独立的类中使得它们可以相互替换)
 * @author SXT李端阳
 *
 */
    public interface Sorter {


        /**
         * 排序
         * @param list 待排序的数组
         */
```

```
    public <T extends Comparable<T>> void sort(T[] list);

    /**
     * 排序
     * @param list 待排序的数组
     * @param comp 比较两个对象的比较器
     */
    public <T> void sort(T[] list, Comparator<T> comp);
}
```

BubbleSorter.java

```
package com.bjsxt;

import java.util.Comparator;

/**
 * 冒泡排序
 * @author SXT李端阳
 *
 */
    public class BubbleSorter implements Sorter {
```

```
@Override
public <T extends Comparable<T>> void sort(T[] list) {
    boolean swapped = true;
    for(int i = 1; i < list.length && swapped;i++) {
       swapped= false;
       for(int j = 0; j < list.length - i; j++) {
           if(list[j].compareTo(list[j+ 1]) > 0 ) {
               T temp = list[j];
               list[j]= list[j + 1];
               list[j+ 1] = temp;
               swapped= true;
           }
       }
    }
}

public <T> void sort(T[] list,Comparator<T> comp) {
    boolean swapped = true;
    for(int i = 1; i < list.length && swapped; i++) {
       swapped = false;
       for(int j = 0; j < list.length - i; j++) {
```

```
                if(comp.compare(list[j], list[j + 1]) > 0 ) {
                    T temp = list[j];
                    list[j]= list[j + 1];
                    list[j+ 1] = temp;
                    swapped= true;
                }
            }
        }
    }
```

```
package com.bjsxt;


import java.util.Comparator;
/**
 * 归并排序
 * 归并排序是建立在归并操作上的一种有效的排序算法。
 * 该算法是采用分治法（divide-and-conquer）的一个非常典型的应用，
 * 先将待排序的序列划分成一个一个的元素，再进行两两归并，
    * 在归并的过程中保持归并之后的序列仍然有序。
    * @author SXT李端阳
```

```
 *
 */
public class MergeSorter implements Sorter {

    @Override
    public <T extends Comparable<T>> void sort(T[] list) {
        T[] temp = (T[]) new Comparable[list.length];
        mSort(list,temp, 0, list.length- 1);
    }

    private <T extends Comparable<T>> void mSort(T[] list, T[]
temp, int low, int high) {
        if(low == high) {
            return ;
        }
        else {
            int mid = low + ((high -low) >> 1);
            mSort(list,temp, low, mid);
            mSort(list,temp, mid + 1, high);
            merge(list,temp, low, mid + 1, high);
        }
    }
```

```
        private <T extends Comparable<T>> void merge(T[] list, T[]
temp, int left, int right, int last) {
                int j = 0;
                int lowIndex = left;
                int mid = right - 1;
                int n = last - lowIndex + 1;
                while (left <= mid && right <= last){
                    if (list[left].compareTo(list[right]) < 0){
                        temp[j++] = list[left++];
                    } else {
                        temp[j++] = list[right++];
                    }
                }
                while (left <= mid) {
                    temp[j++] = list[left++];
                }
                while (right <= last) {
                    temp[j++] = list[right++];
                }
                for (j = 0; j < n; j++) {
                    list[lowIndex + j] = temp[j];
```

```
            }
        }

        @Override
        public <T> void sort(T[] list, Comparator<T> comp) {
            T[]temp = (T[])new Comparable[list.length];
            mSort(list,temp, 0, list.length- 1, comp);
        }

        private <T> void mSort(T[] list, T[] temp, int low, int high,
Comparator<T> comp) {
            if(low == high) {
                return ;
            }
            else {
                int mid = low + ((high -low) >> 1);
                mSort(list,temp, low, mid, comp);
                mSort(list,temp, mid + 1, high, comp);
                merge(list,temp, low, mid + 1, high, comp);
            }
        }
```

```
        private <T> void merge(T[] list, T[]temp, int left, int right, int last,
Comparator<T> comp) {
                int j = 0;
                int lowIndex = left;
                int mid = right - 1;
                int n = last - lowIndex + 1;
                while (left <= mid && right <= last){
                    if (comp.compare(list[left], list[right]) <0) {
                        temp[j++] = list[left++];
                    } else {
                        temp[j++] = list[right++];
                    }
                }
                while (left <= mid) {
                    temp[j++] = list[left++];
                }
                while (right <= last) {
                    temp[j++] = list[right++];
                }
                for (j = 0; j < n; j++) {
                    list[lowIndex + j] = temp[j];
                }
```

```
        }


}
```

QuickSorter.java

```
package com.bjsxt;


import java.util.Comparator;


/**
 * 快速排序
 * 快速排序是使用分治法（divide-and-conquer）依选定的枢轴
 * 将待排序序列划分成两个子序列，其中一个子序列的元素都小于枢轴，
 * 另一个子序列的元素都大于或等于枢轴，然后对子序列重复上面的方法，
 * 直到子序列中只有一个元素为止
 * @author Hao
 *
 */
public class QuickSorter implements Sorter {


    @Override
```

```
    public <T extends Comparable<T>> void sort(T[] list) {
        quickSort(list, 0, list.length- 1);
    }

    @Override
    public <T> void sort(T[] list, Comparator<T> comp) {
        quickSort(list, 0, list.length- 1, comp);
    }

    private <T extends Comparable<T>> void quickSort(T[] list, int first,
int last) {
        if (last > first) {
          int pivotIndex = partition(list, first, last);
          quickSort(list, first, pivotIndex - 1);
          quickSort(list, pivotIndex, last);
        }
    }

    private <T> void quickSort(T[] list, int first, int last,Comparator<T>
comp) {
        if (last > first) {
          int pivotIndex = partition(list, first, last, comp);
```

```
            quickSort(list, first, pivotIndex - 1, comp);
            quickSort(list, pivotIndex, last, comp);
        }
    }

    private <T extends Comparable<T>> int partition(T[] list, int first,
int last) {
        T pivot = list[first];
        int low = first + 1;
        int high = last;

        while (high > low) {
            while (low <= high && list[low].compareTo(pivot) <= 0) {
                low++;
            }
            while (low <= high && list[high].compareTo(pivot) >= 0) {
                high--;
            }
            if (high > low) {
                T temp = list[high];
                list[high]= list[low];
                list[low]= temp;
```

```
            }
        }

        while (high > first&& list[high].compareTo(pivot) >= 0) {
            high--;
        }
        if (pivot.compareTo(list[high])> 0) {
            list[first]= list[high];
            list[high]= pivot;
            return high;
        }
        else {
            return low;
        }
    }

    private <T> int partition(T[] list, int first, int last, Comparator<T>
comp) {
        T pivot = list[first];
        int low = first + 1;
        int high = last;
```

```
        while (high > low) {
            while (low <= high&& comp.compare(list[low], pivot) <= 0) {
                low++;
            }
            while (low <= high&& comp.compare(list[high], pivot) >= 0) {
                high--;
            }
            if (high > low) {
                T temp = list[high];
                list[high] = list[low];
                list[low]= temp;
            }
        }

        while (high > first&& comp.compare(list[high], pivot) >= 0) {
            high--;
        }
        if (comp.compare(pivot,list[high]) > 0) {
            list[first]= list[high];
            list[high]= pivot;
            return high;
              }
```

```
            else {
                return low;
            }
        }

}
```

写一个二分查找（折半搜索）的算法。

答：折半搜索，也称二分查找算法、二分搜索，是一种在有序数组中查找某一特定元素的搜索算法。搜素过程从数组的中间元素开始，如果中间元素正好是要查找的元素，则搜素过程结束；如果某一特定元素大于或者小于中间元素，则在数组大于或小于中间元素的那一半中查找，而且跟开始一样从中间元素开始比较。如果在某一步骤数组为空，则代表找不到。这种搜索算法每一次比较都使搜索范围缩小一半。

```
package com.bjsxt;
import java.util.Comparator;

public class MyUtil1 {

    public static <T extends Comparable<T>> int binarySearch(T[] x, T key)
{
```

```
    return binarySearch(x, 0, x.length- 1, key);
}


public static <T> int binarySearch(T[] x, T key, Comparator<T> comp) {
    int low = 0;
    int high = x.length - 1;
    while (low <= high) {
        int mid = (low + high) >>> 1;
        int cmp = comp.compare(x[mid], key);
        if (cmp < 0) {
          low = mid + 1;
        }
        else if (cmp > 0) {
          high = mid - 1;
        }
        else {
          return mid;
        }
    }
    return -1;
}
```

```
    private static <T extends Comparable<T>> int binarySearch(T[] x, int
low, int high, T key) {
        if(low <= high) {
            int mid = low + ((high -low) >> 1);
            if(key.compareTo(x[mid]) == 0) {
                return mid;
            }
            else if(key.compareTo(x[mid])< 0) {
                return binarySearch(x,low, mid - 1, key);
            }
            else {
                return binarySearch(x, mid + 1, high, key);
            }
        }
        return -1;
    }
}
```

说明：两个版本一个用递归实现，一个用循环实现。需要注意的是计算中间位置时不应该使用(high+ low) / 2 的方式，因为加法运算可能导致整数越界 ,这里应该使用一下三种方式之一 :low+ (high – low) / 2 或 low + (high

– low) >> 1 或 (low + high) >>> 1 (注：>>>是逻辑右移，不带符号位的右移)

统计一篇英文文章单词个数。

答：

```
package com.bjsxt;


import java.io.FileReader;


public class WordCounting {
    public static void main(String[] args) {
      try(FileReader fr = new FileReader("a.txt")) {
          int counter = 0;
          boolean state = false;
          int currentChar;
          while((currentChar= fr.read()) != -1) {
            if(currentChar== ' ' || currentChar == '\n'
                || currentChar == '\t' || currentChar == '\r') {
                state = false;
            }
            else if(!state) {
                state = true;
                counter++;
```

```
            }
        }
        System.out.println(counter);
    }
    catch(Exception e) {
        e.printStackTrace();
    }
  }
}
```

补充：这个程序可能有很多种写法，这里选择的是 Dennis M. Ritchie 和 Brian W. Kernighan 老师在他们不朽的著作《The C Programming Language》中给出的代码，向两位老师致敬。下面的代码也是如此。

输入年月日，计算该日期是这一年的第几天。

```
package com.bjsxt;

import java.util.Scanner;

public class DayCounting {

    public static void main(String[] args) {
```

```
        int[][] data = {
                {31,28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
                {31,29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}
        };
            Scanner sc = new Scanner(System.in);
            System.out.print("请输入年月日(1980 11 28): ");
            int year = sc.nextInt();
            int month = sc.nextInt();
            int date = sc.nextInt();
            int[] daysOfMonth = data[(year % 4 == 0 && year % 100 != 0 ||
year % 400 == 0)?1 : 0];
            int sum = 0;
            for(int i = 0; i < month -1; i++) {
               sum += daysOfMonth[i];
            }
            sum += date;
            System.out.println(sum);
            sc.close();
        }
}
```

答：

```
package com.bjsxt;


public class Josephu {
        private static final int DEAD_NUM = 9;


        public static void main(String[] args) {
            boolean[] persons = new boolean[30];
            for(int i = 0; i < persons.length; i++) {
               persons[i] = true;
            }


            int counter = 0;
            int claimNumber = 0;
            int index = 0;
            while(counter < 15) {
               if(persons[index]) {
                    claimNumber++;
                    if(claimNumber == DEAD_NUM) {
                        counter++;
                        claimNumber= 0;
```

```
                    persons[index]= false;
                }
            }
            index++;
            if(index >= persons.length) {
                index= 0;
            }
        }
        for(boolean p : persons) {
            if(p) {
                System.out.print("基");
            }
            else {
                System.out.print("非");
            }
        }
    }
}
```

回文素数：

317. 所谓回文数就是顺着读和倒着读一样的数(例如：11，121，1991...)，回文素数就是既是回文数又是素数(只能被 1 和自身整除的数)的数。编程找出 11～9999 之间的回文素数。

答：

```
package com.bjsxt;


public class PalindromicPrimeNumber {


	public static void main(String[] args) {
		for(int i = 11; i <= 9999; i++) {
			if(isPrime(i) && isPalindromic(i)) {
				System.out.println(i);
			}
		}
	}


	public static boolean isPrime(int n) {
		for(int i = 2; i <= Math.sqrt(n); i++) {
			if(n % i == 0) {
				return false;
			}
		}
```

```
            return true;
        }

        public static boolean isPalindromic(int n) {
            int temp = n;
            int sum = 0;
            while(temp > 0) {
                sum= sum * 10 + temp % 10;
                temp/= 10;
            }
            return sum == n;
        }
    }
```

318. 全排列：给出五个数字 12345 的所有排列。

答：

```
package com.bjsxt;

public class FullPermutation {

    public static void perm(int[] list) {
```

```
        perm(list,0);
    }

    private static void perm(int[] list, int k) {
        if (k == list.length) {
            for (int i = 0; i < list.length; i++) {
                    System.out.print(list[i]);
                }
                 System.out.println();
            }else{
                for (int i = k; i < list.length; i++) {
                    swap(list, k, i);
                    perm(list, k + 1);
                    swap(list, k, i);
                }
            }
        }

        private static void swap(int[] list, int pos1, int pos2) {
            int temp = list[pos1];
            list[pos1] = list[pos2];
            list[pos2] = temp;
```

```
        }


        public static void main(String[] args) {
            int[] x = {1, 2, 3, 4, 5};
            perm(x);
        }
}
```

319. 对于一个有 N 个整数元素的一维数组，找出它的子数组（数组中下标连续的元素组成的数组）之和的最大值。

答：下面给出几个例子（最大子数组用粗体表示）：

数组：{ 1, -2, **3**,**5**, -3, 2 }，结果是：8

2) 数组：{ 0, -2, **3**, **5**, **-1**, **2** }，结果是：9

3) 数组：{ -9, **-2**,-3, -5, -3 }，结果是：-2

可以使用动态规划的思想求解：

```
package com.bjsxt;


public class MaxSum {


    private static int max(int x, int y) {
        return x > y? x: y;
```

```
    }


    public static int maxSum(int[] array) {
        int n = array.length;
        int[] start = new int[n];
            int[] all = new int[n];
            all[n - 1] = start[n - 1] = array[n - 1];
            for(int i = n - 2; i >= 0;i--) {
               start[i] = max(array[i], array[i] + start[i + 1]);
               all[i] = max(start[i], all[i + 1]);
            }
            return all[0];
        }


        public static void main(String[] args) {
            int[] x1 = { 1, -2, 3, 5,-3, 2 };
            int[] x2 = { 0, -2, 3, 5,-1, 2 };
            int[] x3 = { -9, -2, -3,-5, -3 };
            System.out.println(maxSum(x1));     // 8
            System.out.println(maxSum(x2));     // 9
            System.out.println(maxSum(x3));     //-2
        }
```

```
    }
```

320. 用递归实现字符串倒转

```
package com.bjsxt;


public class StringReverse {


        public static String reverse(String originStr) {
            if(originStr == null || originStr.length()== 1) {
                return originStr;
            }
            return reverse(originStr.substring(1))+ originStr.charAt(0);
        }


             public static void main(String[] args) {
                System.out.println(reverse("hello"));
             }
}
```

321. 输入一个正整数，将其分解为素数的乘积。

答：

```
package com.bjsxt;


import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;


public class DecomposeInteger {


        private static List<Integer> list = new ArrayList<Integer>();


        public static void main(String[] args) {

            System.out.print("请输入一个数: ");

            Scanner sc = new Scanner(System.in);

            int n = sc.nextInt();

            decomposeNumber(n);

                System.out.print(n + " = ");

                for(int i = 0; i < list.size() - 1; i++) {

                    System.out.print(list.get(i) + " * ");

                }

               System.out.println(list.get(list.size() - 1));
```

```
    }

    public static void decomposeNumber(int n) {
        if(isPrime(n)) {
            list.add(n);
            list.add(1);
        }
        else {
            doIt(n, (int)Math.sqrt(n));
        }
    }

    public static void doIt(int n, int div) {
        if(isPrime(div) && n % div == 0) {
            list.add(div);
            decomposeNumber(n / div);
        }
        else {
            doIt(n, div - 1);
        }
    }
```

```
        public static boolean isPrime(int n) {
            for(int i = 2; i <= Math.sqrt(n);i++) {
                if(n % i == 0) {
                    return false;
                }
            }
            return true;
        }
    }
```

322.

一个有 n 级的台阶，一次可以走 1 级、2 级或 3 级，问走完 n 级台阶有多少种走法。

答：可以通过递归求解。

```
package com.bjsxt;

public class GoSteps {

    public static int countWays(int n) {
        if(n < 0) {
            return 0;
        }
```

```
            else if(n == 0) {
                return 1;
            }
            else {
                return countWays(n - 1) + countWays(n - 2) +
countWays(n -3);
            }
        }

        public static void main(String[] args) {
            System.out.println(countWays(5));     // 13
        }
}
```

323. 写一个算法判断一个英文单词的所有字母是否全都不同(不区分大小写)。

答：

```
package com.bjsxt;
```

```
public class AllNotTheSame {


        public static boolean judge(String str) {
            String temp = str.toLowerCase();
            int[] letterCounter = new int[26];
            for(int i = 0; i <temp.length(); i++) {
              int index = temp.charAt(i)- 'a';
              letterCounter[index]++;
              if(letterCounter[index] > 1) {
                      return false;
                  }
               }
               return true;
            }

             public static void main(String[] args) {
                System.out.println(judge("hello"));
                System.out.print(judge("smile"));
            }
    }
```

324. 有一个已经排好序的整数数组，其中存在重复元素，请将重复元素删除掉，例如，A= [1, 1, 2, 2, 3]，处理之后的数组应当为 A= [1, 2, 3]。

答：

```
package com.bjsxt;


import java.util.Arrays;


public class RemoveDuplication {


    public static int[] removeDuplicates(int a[]) {
        if(a.length <= 1) {
            return a;
        }
        int index = 0;
            for(int i = 1; i < a.length; i++) {
                if(a[index] != a[i]) {
                    a[++index] = a[i];
                }
            }
            int[] b = new int[index + 1];
            System.arraycopy(a, 0, b, 0, b.length);
```

```
            return b;
        }

        public static void main(String[] args) {
            int[] a = {1, 1, 2, 2, 3};
            a = removeDuplicates(a);
            System.out.println(Arrays.toString(a));
        }
    }
```

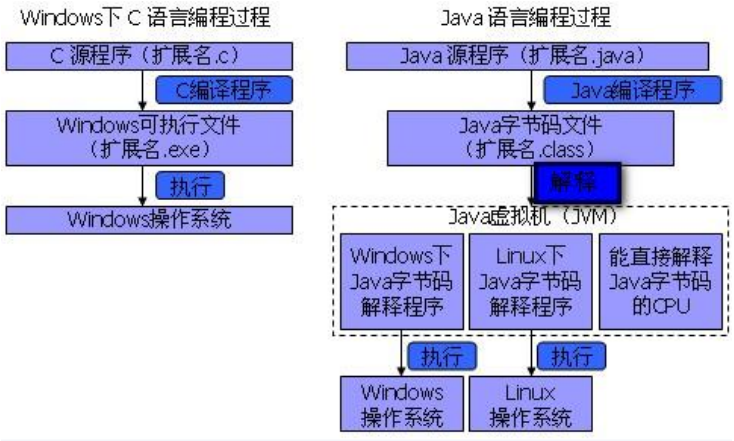
325. 给一个数组 ,其中有一个重复元素占半数以上 ,找出这个元素。

答：

```
package com.bjsxt;

public class FindMost {

    public static <T> T find(T[] x){
        T temp = null;
        for(int i = 0, nTimes = 0; i< x.length;i++) {
            if(nTimes == 0) {
                temp= x[i];
```

```
                    nTimes= 1;
                }
                    else {
                        if(x[i].equals(temp)) {
                            nTimes++;
                        }
                        else {
                            nTimes--;
                        }
                    }
                }
                return temp;
            }

            public static void main(String[] args) {
                String[]strs = {"hello","kiss","hello","hello","maybe"};
                System.out.println(find(strs));
            }
    }
```

326. Java 跨平台原理（字节码文件、虚拟机）



C/C++语言都直接编译成针对特定平台机器码。如果要跨平台，需要使用相应的编译器重新编译。

Java 源程序（.java）要先编译成与平台无关的字节码文件(.class)，然后字节码文件再解释成机器码运行。解释是通过 Java 虚拟机来执行的。

字节码文件不面向任何具体平台，只面向虚拟机。

Java 虚拟机是可运行 Java 字节码文件的虚拟计算机。不同平台的虚拟机是不同的，但它们都提供了相同的接口。

Java 语言具有一次编译，到处运行的特点。就是说编译后的.class 可以跨平台运行，前提是该平台具有相应的 Java 虚拟机。但是性能比 C/C++要低。

Java 的跨平台原理决定了其性能没有 C/C++高

327. Java 的安全性

语言层次的安全性主要体现在：

Java 取消了强大但又危险的指针 ,而代之以引用。由于指针可进行移动运算，指针可随便指向一个内存区域 ,而不管这个区域是否可用 ,这样做是危险的，因为原来这个内存地址可能存储着重要数据或者是其他程序运行所占用的，并且使用指针也容易数组越界。

垃圾回收机制：不需要程序员直接控制内存回收，由垃圾回收器在后台自动回收不再使用的内存。避免程序忘记及时回收，导致内存泄露。避免程序错误回收程序核心类库的内存，导致系统崩溃。

异常处理机制 :Java 异常机制主要依赖于 try、catch、finally、throw、throws 五个关键字。

强制类型转换：只有在满足强制转换规则的情况下才能强转成功。

底层的安全性可以从以下方面来说明

Java 在字节码的传输过程中使用了公开密钥加密机制(PKC)。

在运行环境提供了四级安全性保障机制：

字节码校验器 -类装载器 -运行时内存布局 -文件访问限制

328. Java 三大版本

Java2 平台包括标准版（J2SE）、企业版（J2EE）和微缩版（J2ME）三个版本：

Standard Edition(标准版) J2SE 包含那些构成 Java 语言核心的类。

比如：数据库连接、接口定义、输入/输出、网络编程

Enterprise Edition(企业版) J2EE 包含 J2SE 中的类，并且还包含用于开发企业级应用的类。

比如：EJB、servlet、JSP、XML、事务控制

Micro Edition(微缩版) J2ME 包含 J2SE 中一部分类，用于消费类电子产品的软件开发。

比如：呼机、智能卡、手机、PDA、机顶盒

他们的范围是：J2SE 包含于 J2EE 中，J2ME 包含了 J2SE 的核心类，但新添加了一些专有类

应用场合，API 的覆盖范围各不相同。

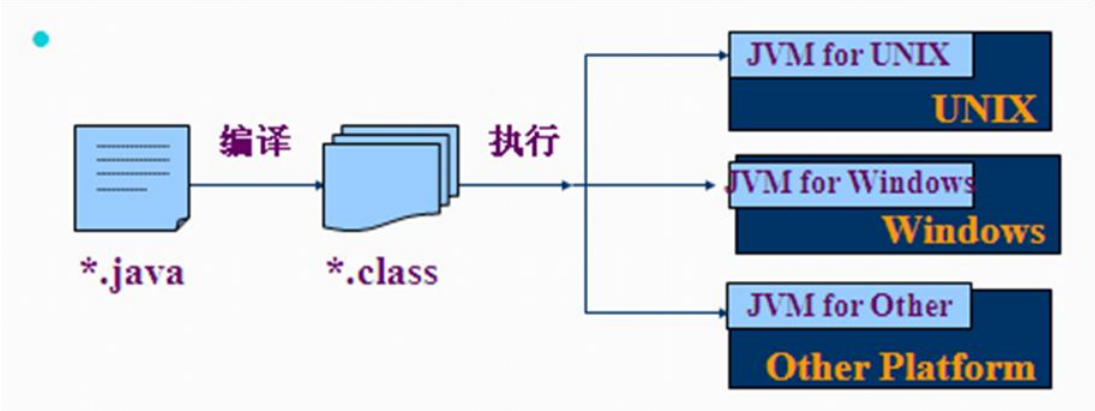
329. Java 开发运行过程

在安装好 JDK 并配置好 path、classpath 后开发运行步骤如下：

1、可以用任何文本编辑器创建并编辑 Java 源程序，Java 源程序用“.java”作为文件扩展名

2、编译 Java 源程序编译器，使用命令“javac”编译“java 源程序文件名.java”。最后编译成 Java 虚拟机能够明白的指令集合，且以字节码的形式保存在文件中。通常，字节码文件以“.class”作为扩展名。

3、执行 java 程序，使用“java”命令运行 class（字节码）文件“java 文件名”，Java 解释器会读取字节码，取出指令并且翻译成计算机能执行的机器码，完成运行过程。



330. Java 开发环境配置

具体配置步骤如下：

0）找到自己的 jdk 安装路径，如：C:\Java\jdk1.7.0_60\bin

1）右击桌面“我的电脑”，选择“属性”

2）选中“高级系统设置”-->高级-->环境变量设置

3）在系统变量中点击”新建”按钮，

变量名：JAVA_HOME

变量值：C:\Java\jdk1.7.0_60

4)在系统变量中找到“path”并选中，点击“编辑”，在最后面或最前面输入%JAVA_HOME%\bin;

5）在系统变量中点击”新建”按钮，

变量名:CLASSPATH

变量值: .;%JAVA_HOME%\lib\dt.jar;%JAVA_HOME%\lib\tools.jar;

6)一路点击三个”确定”按钮，配置完毕。

331. 什么是 JVM？什么是 JDK？ 什么是 JRE？

JVM :JVM 是 Java Virtual Machine(Java 虚拟机)的缩写，它是整个 java 实现跨平台的最核心的部分，所有的 java 程序会首先被编译为.class 的类文件，这种类文件可以在虚拟机上执行，也就是说 class 并不直接与机器的操作系统相对应，而是经过虚拟机间接与操作系统交互，由虚拟机将程序解释给本地系统执行。JVM 是 Java 平台的基础，和实际的机器一样，它也有自己的指令集，并且在运行时操作不同的内存区域。 JVM 通过抽象操作系统和 CPU 结构，提供了一种与平台无关的代码执行方法，即与特殊的实现方法、主机硬件、主机操作系统无关。JVM 的主要工作是解释自己的指令集(即字节码 ）到 CPU 的指令集或对应的系统调用，保护用户免被恶意程序骚扰。 JVM 对上层的 Java 源文件是不关心的，它关注的只是由源文件生成的类文件（ .class 文件)。

JRE :JRE 是 java runtime environment(java 运行环境)的缩写。光有 JVM 还不能让 class 文件执行，因为在解释 class 的时候 JVM 需要调用解释所需要的类库 lib。在 JDK 的安装目录里你可以找到 jre 目录，里面有两个文件夹 bin 和 lib,在这里可以认为 bin 里的就是 jvm ,lib 中则是 jvm 工作所需要的类库，而 jvm 和 lib 和起来就称为 jre。所以，在你写完 java 程序编译成.class 之后，你可以把这个.class 文件和 jre 一起打包发给朋友，这样你的朋友就可以运行你写程序了（ jre 里有运行.class 的 java.exe)。JRE 是 Sun 公司发布的一个更大的系统，它里面就有一个 JVM。JRE 就与具体的 CPU 结构和操作系统有关，是运行 Java 程序必不可少的（ 除非用其他一些编译环境编译成.exe 可执行文件……)，JRE 的地位就象一台 PC 机一样，我们写

好的 Win32 应用程序需要操作系统帮我们运行，同样的，我们编写的 Java 程序也必须要 JRE 才能运行。

JDK :JDK 是 java development kit(java 开发工具包)的缩写。每个学 java 的人都会先在机器上装一个 JDK，那 让我们看一下 JDK 的安装目录。在目录下面有六个文件夹、一个 src 类库源码压缩包、和其他几个声明文件。其中 ,真正在运行 java 时起作用的是以下四个文件夹 :bin、include、lib、jre。现在我们可以看出这样一个关系，JDK 包含 JRE，而 JRE 包含 JVM。

bin:最主要的是编译器(javac.exe)

include:java 和 JVM 交互用的头文件

lib：类库

jre:java 运行环境

（注意 :这里的 bin、lib 文件夹和 jre 里的 bin、lib 是不同的)总的来说 JDK 是用于 java 程序的开发,而 jre 则是只能运行 class 而没有编译的功能。eclipse、idea 等其他 IDE 有自己的编译器而不是用 JDK bin 目录中自带的，所以在安装时你会发现他们只要求你选 jre 路径就 ok 了。

4、JDK,JRE,JVM 三者关系概括如下：

jdk 是 JAVA 程序开发时用的开发工具包，其内部也有 JRE 运行环境 JRE。JRE 是 JAVA 程序运行时需要的运行环境，就是说如果你光是运行 JAVA 程序而不是去搞开发的话 ,只安装 JRE 就能运行已经存在的 JAVA 程序了。JDk、JRE 内部都包含 JAVA 虚拟机 JVM，JAVA 虚拟机内部包含许多应用程序的类的解释器和类加载器等等。

Java 三种注释类型

共有单行注释、多行注释、文档注释 3 种注释类型。使用如下：

单行注释，采用“//”方式.只能注释一行代码。如：//类成员变量

多行注释，采用“/*...*/”方式，可注释多行代码，其中不允许出现嵌套。如：

```
    /*System.out.println("a");
    System.out.println("b");
    System.out.println("c");*/
```

文档注释，采用“/**...*/”方式。如：

```
  /**
 * 子类   Dog
 * @author Administrator
 *
 */
public class Dog extends Animal{}
```

332. 8 种基本数据类型及其字节数

数据类型		关键字	字节数
数值型	整数型	byte	1
		short	2
		int	4
		long	8
	浮点型	float	4
		double	8
布尔型		boolean	1（位）
字符型		char	2

333. i++和++i 的异同之处

共同点：

1、i++和++i 都是变量自增 1，都等价于 i=i+1

2、如果 i++,++i 是一条单独的语句，两者没有任何区别

3、i++和++i 的使用仅仅针对变量。 5++和++5 会报错，因为 5 不是变量。

不同点：

如果 i++,++i 不是一条单独的语句，他们就有区别

i++ ：先运算后增 1。如：

```
int x=5;

int y=x++;
```

```
System.out.println("x="+x+", y="+y);
//以上代码运行后输出结果为：x=6, y=5
```

++i ： 先增 1 后运算。如：

```
int x=5;
int y=++x;
System.out.println("x="+x+", y="+y);
//以上代码运行后输出结果为：x=6, y=6
```

334. &和&&的区别和联系，|和||的区别和联系

&和&&的联系(共同点)：

&和&&都可以用作逻辑与运算符，但是要看使用时的具体条件来决定。

```
操作数 1&操作数 2 ,操作数 1&&操作数 2，
表达式 1&表达式 2 ,表达式 1&&表达式 2，
```

情况 1：当上述的操作数是 boolean 类型变量时，&和&&都可以用作逻辑与运算符。

情况 2：当上述的表达式结果是 boolean 类型变量时，&和&&都可以用作逻辑与运算符。

表示逻辑与(and)，当运算符两边的表达式的结果或操作数都为 true 时，整个运算结果才为 true，否则，只要有一方为 false，结果都为 false。

&和&&的区别(不同点)：

(1)、&逻辑运算符称为逻辑与运算符，&&逻辑运算符称为短路与运算

符，也可叫逻辑与运算符。

对于&：无论任何情况，&两边的操作数或表达式都会参与计算。

对于&&：当&&左边的操作数为 false 或左边表达式结果为 false 时，&&右边的操作数或表达式将不参与计算，此时最终结果都为 false。

综上所述，如果逻辑与运算的第一个操作数是 false 或第一个表达式的结果为 false 时，对于第二个操作数或表达式是否进行运算，对最终的结果没有影响，结果肯定是 false。推介平时多使用&&，因为它效率更高些。

、&还可以用作位运算符。当&两边操作数或两边表达式的结果不是 boolean 类型时，&用于按位与运算符的操作。

|和||的区别和联系与&和&&的区别和联系类似

335. 用最有效率的方法算出 2 乘以 8 等于多少

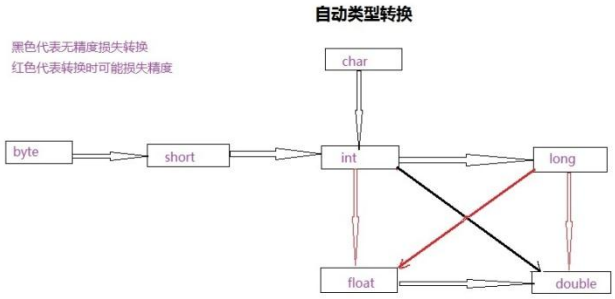
使用位运算来实现效率最高。位运算符是对操作数以二进制比特位为单位进行操作和运算，操作数和结果都是整型数。对于位运算符“<<”，是将一个数左移 n 位，就相当于乘以了 2 的 n 次方，那么，一个数乘以 8 只要将其左移 3 位即可，位运算 cpu 直接支持的，效率最高。所以，2 乘以 8 等于几的最效率的方法是 2 << 3。

336. 基本数据类型的类型转换规则

基本类型转换分为自动转换和强制转换。

自动转换规则：容量小的数据类型可以自动转换成容量大的数据类型，也可以说低级自动向高级转换。这儿的容量指的不是字节数，而是指类型表述的

范围。



强制转换规则：高级变为低级需要强制转换。

如何转换：

（1）、赋值运算符“=”右边的转换，先自动转换成表达式中级别最高的数据类型，再进行运算。

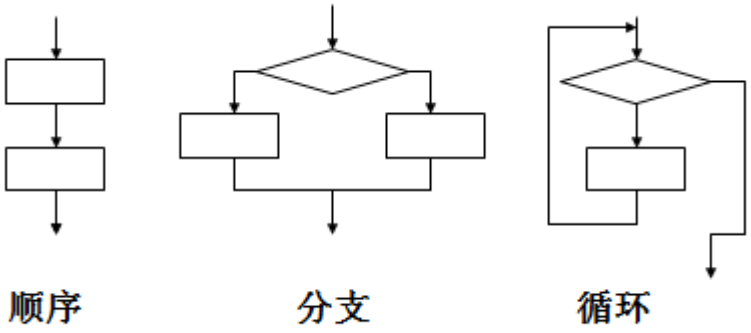
（2）、赋值运算符“=”两侧的转换，若左边级别>右边级别，会自动转换；若左边级别 == 右边级别，不用转换；若左边级别 < 右边级别，需强制转换。

(3)、可以将整型常量直接赋值给 byte, short, char 等类型变量，而不需要进行强制类型转换，前提是不超出其表述范围，否则必须进行强制转换。

337. 三种流程控制结构

其流程控制方式采用结构化程序设计中规定的三种基本流程结构，

即：顺序结构、分支结构和循环结构



338. if 多分支语句和 switch 多分支语句的异同之处

相同之处：都是分支语句，多超过一种的情况进行判断处理。

不同之处：

switch 更适合用于多分支情况，就是有很多种情况需要判断处理，判断条件类型单一，只有一个入口，在分支执行完后（如果没有 break 跳出），不加判断地执行下去;而 if—elseif---else 多分枝主要适用于分支较少的分支结构，判断类型不是单一，只要一个分支被执行后，后边的分支不再执行。

switch 为等值判断（不允许比如>= <=），而 if 为等值和区间都可以，if 的使用范围大。

339. while 和 do-while 循环的区别

while 先判断后执行，第一次判断为 false,循环体一次都不执行

do while 先执行 后判断，最少执行 1 次。

如果 while 循环第一次判断为 true，则两种循环没有区别。

340. break 和 continue 的作用

break: 结束当前循环并退出当前循环体。

break 还可以退出 switch 语句

continue: 循环体中后续的语句不执行，但是循环没有结束，继续进行循环条件的判断（for 循环还会 i++ ）。continue 只是结束本次循环。

341. 请使用递归算法计算 n！

```
package com.bjsxt;


public class Test {
    public int factorial(int n) {
        if (n == 1 || n == 0){
            return n;
        }else{
            return n * factorial(n - 1);
        }
    }
    public static void main(String[] args) {
        Test test = new Test();
        System.out.println(test.factorial(6));
    }

}
```

342. 递归的定义和优缺点

递归算法是一种直接或者间接地调用自身算法的过程。在计算机编写程序中，递归算法对解决一大类问题是十分有效的，它往往使算法的描述简洁而且易于理解。

递归算法解决问题的特点：

(1) 递归就是在过程或函数里调用自身。

(2) 在使用递归策略时，必须有一个明确的递归结束条件，称为递归出口。

(3) 递归算法解题通常显得很简洁，但运行效率较低。所以一般不提倡用递归算法设计程序。

(4) 在递归调用的过程当中系统为每一层的返回点、局部量等开辟了栈来存储。递归次数过多容易造成栈溢出等。所以一般不提倡用递归算法设计程序。

343. 数组的特征

数组是（相同类型数据）的（有序）（集合）

数组会在内存中开辟一块连续的空间，每个空间相当于之前的一个变量，称为数组的元素 element

元素的表示 数组名[下标或者索引] scores[7] scores[0] scores[9]

索引从 0 开始

每个数组元素有默认值 double 0.0 boolean false int 0

数组元素有序的，不是大小顺序，是索引 的顺序

数组中可以存储基本数据类型，可以存储引用数据类型；但是对于一个数组而言，数组的类型是固定的，只能是一个

length:数组的长度

数组的长度是固定的，一经定义，不能再发生变化（数组的扩容）

344. 请写出冒泡排序代码

```
package com.bjsxt;


public class TestBubbleSort {

    public static void sort(int[] a) {

        int temp = 0;

        // 外层循环，它决定一共走几趟

        for (int i = 0; i <a.length-1; ++i) {

            //内层循环，它决定每趟走一次

            for (int j = 0; j <a.length-i-1 ; ++j) {

                //如果后一个大于前一个

                if (a[j + 1] < a[j]) {

                    //换位

                    temp = a[j];a[j] = a[j + 1];a[j + 1] = temp;

                }

            }

    }


    public static void sort2(int[] a) {

        int temp = 0;
```

```
        for (int i = 0; i <a.length-1; ++i) {
            //通过符号位可以减少无谓的比较，如果已经有序了，就退出循环
            int flag = 0;
            for (int j = 0; j <a.length-1-i ; ++j) {
                if (a[j + 1] < a[j]) {
                    temp = a[j];
                                a[j] = a[j + 1];
                                a[j + 1] = temp;
                    flag = 1;
                }
            }
            if(flag == 0){
                    break;
                }
        }
    }
}
```

345. 请写出选择排序的代码

```
package com.bjsxt;
```

```
public class TestSelectSort {

    public static void sort(int arr[]) {

        int temp = 0;

        for (int i = 0; i < arr.length - 1; i++) {

            // 认为目前的数就是最小的，记录最小数的下标

            int minIndex = i;

            for (int j = i + 1; j < arr.length; j++) {

                if (arr[minIndex] > arr[j]) {

                    // 修改最小值的下标

                    minIndex = j;

                }

            }

            // 当退出for就找到这次的最小值

            if (i != minIndex) {

                temp = arr[i];

                arr[i] = arr[minIndex];

                arr[minIndex] = temp;

            }

        }

    }

}
```

346. 请写出插入排序的代码

```
package com.bjsxt;


public class TestInsertSort {

    public static void sort(int arr[]) {

        int i, j;

        for (i = 1; i < arr.length; i++) {

            int temp = arr[i];

            for (j = i; j > 0 && temp < arr[j - 1]; j--) {

                arr[j] = arr[j - 1];

            }

            arr[j] = temp;

        }

    }

}
```

347. 可变参数的作用和特点

总结 1：可变参数

1.可变参数的形式 ...

2.可变参数只能是方法的形参

3.可变参数对应的实参可以 0,1,2.....个，也可以是一个数组

4.在可变参数的方法中，将可变参数当做数组来处理

5.可变参数最多有一个，只能是最后一个

6.可变参数好处：方便　简单　减少重载方法的数量

7.如果定义了可变参数的方法 ,不允许同时定义相同类型数组参数的方法

总结 2：数组做形参和可变参数做形参联系和区别

联系：

1.实参都可以是数组；2.方法体中，可变参数当做数组来处理

区别：

1.个数不同 可变参数只能有一个数组参数可以多个

2.位置不同 可变参数只能是最后一个　数组参数位置任意

3.实参不同 可变参数实参可以 0,1,2.....个，也可以是一个数组，数组的实参只能是数组

348. 类和对象的关系

类是对象的抽象，而对象是类的具体实例。类是抽象的，不占用内存，而对象是具体的，占用存储空间。类是用于创建对象的蓝图，它是一个定义包括在特定类型的对象中的方法和变量的软件模板。

类和对象好比图纸和实物的关系，模具和铸件的关系。

比如人类就是一个概念，人类具有身高，体重等属性。人类可以做吃饭、说话等方法。

小明就是一个具体的人，也就是实例，他的属性是具体的身高 200cm，体

重 180kg，他做的方法是具体的吃了一碗白米饭，说了“12345”这样一句话。

349. 面向过程和面向对象的区别

两者都是软件开发思想，先有面向过程，后有面向对象。在大型项目中，针对面向过程的不足推出了面向对象开发思想。

	面向过程	面向对象
区别	**事物比较简单，可以用线性的思维去解决**	**事物比较复杂，使用简单的线性思维无法解决**
共同点	面向过程和面向对象都是解决实际问题的一种思维方式	
	二者相辅相成，并不是对立的。 解决复杂问题，通过面向对象方式便于我们从宏观上把握事物之间复杂的关系、方便我们分析整个系统；具体到微观操作，仍然使用面向过程方式来处理	

比喻

蒋介石和毛泽东分别是面向过程和面向对象的杰出代表，这样充分说明，在解决复制问题时，面向对象有更大的优越性。

面向过程是蛋炒饭，面向对象是盖浇饭。盖浇饭的好处就是“菜” “饭”分离，从而提高了制作盖浇饭的灵活性。饭不满意就换饭，菜不满意换菜。用软件工程的专业术语就是“可维护性”比较好，“饭” 和“菜”的耦合度比较低。

区别

编程思路不同： 面向过程以实现功能的函数开发为主，而面向对象要首先抽象出类、属性及其方法，然后通过实例化类、执行方法来完成功能。

封装性：都具有封装性，但是面向过程是封装的是功能，而面向对象封装的是数据和功能。

面向对象具有继承性和多态性，而面向过程没有继承性和多态性，所以面向对象优势是明显。

方法重载和方法重写（覆盖）的区别

	英文	位置不同	作用不同
重载	overload	同一个类中	在一个类里面为一种行为提供多种实现方式并提高可读性
重写	override	子类和父类间	父类方法无法满足子类的要求，子类通过方法重写满足要求

	修饰符	返回值	方法名	参数	抛出异常
重载	无关	无关	相同	不同	无关
重写	大于等于	小于等于	相同	相同	小于等于

350. this 和 super 关键字的作用

this 是对象内部指代自身的引用,同时也是解决成员变量和局部变量同名问题；this 可以调用成员变量，不能调用局部变量；this 也可以调用成员方法，但是在普通方法中可以省略 this，在构造方法中不允许省略，必须是构造方法的第一条语句。，而且在静态方法当中不允许出现 this 关键字。

super 代表对当前对象的直接父类对象的引用，super 可以调用直接父类的成员变量（注意权限修饰符的影响，比如不能访问 private 成员）

super 可以调用直接父类的成员方法（注意权限修饰符的影响，比如不能访

问 private 成员）；super 可以调用直接父类的构造方法，只限构造方法中使用，且必须是第一条语句。

351. static 关键字的作用（修饰变量、方法、代码块）

static 可以修饰变量、方法、代码块和内部类

static 属性属于这个类所有，即由该类创建的所有对象共享同一个 static 属性。可以对象创建后通过对象名.属性名和类名.属性名两种方式来访问。也可以在没有创建任何对象之前通过类名.属性名的方式来访问。

.static 变量和非 static 变量的区别(都是成员变量，不是局部变量)

1.在内存中份数不同

不管有多少个对象，static 变量只有 1 份。对于每个对象，实例变量都会有单独的一份

static 变量是属于整个类的，也称为类变量。而非静态变量是属于对象的，也称为实例变量

2.在内存中存放的位置不同

静态变量存在方法区中，　实例变量存在堆内存中 *

3.访问的方式不同

实例变量： 对象名.变量名　stu1.name="小明明";

静态变量：对象名.变量名　stu1.schoolName="西二旗小学"; 不推荐如此使用

类名.变量名　Student.schoolName="东三旗小学"; 推荐使用

4.在内存中分配空间的时间不同

实例变量：创建对象的时候才分配了空间。静态变量：第一次使用类的时候

Student.schoolName="东三旗小学";或者 Student stu1 = new Student("小明","男",20,98);

static 方法也可以通过对象名.方法名和类名.方法名两种方式来访问

static 代码块。当类被第一次使用时（可能是调用 static 属性和方法，或者创建其对象）执行静态代码块，且只被执行一次，主要作用是实现 static 属性的初始化。

static 内部类：属于整个外部类，而不是属于外部类的每个对象。不能访问外部类的非静态成员（变量或者方法），.可以访问外部类的静态成员

352. final 和 abstract 关键字的作用

final 和 abstract 是功能相反的两个关键字，可以对比记忆

abstract 可以用来修饰类和方法，不能用来修饰属性和构造方法；使用 abstract 修饰的类是抽象类，需要被继承，使用 abstract 修饰的方法是抽象方法，需要子类被重写。

final 可以用来修饰类、方法和属性，不能修饰构造方法。使用 final 修饰的类不能被继承，使用 final 修饰的方法不能被重写，使用 final 修饰的变量的值不能被修改，所以就成了常量。

特别注意：final 修饰基本类型变量，其值不能改变，由原来的变量变为常量；但是 final 修饰引用类型变量，栈内存中的引用不能改变，但是所指向的堆内存中的对象的属性值仍旧可以改变。例如

```
package com.bjsxt;


class Test {
```

```
    public static void main(String[] args) {
        final Dog dog = new Dog("欧欧");
        dog.name = "美美";//正确
        dog = new Dog("亚亚");//错误
    }
}
```

353. final、finally、finalize 的区别

final 修饰符（关键字）如果一个类被声明为 final，意味着它不能再派生出新的子类，不能作为父类被继承例如：String 类、Math 类等。将变量或方法声明为 final，可以保证它们在使用中不被改变。被声明为 final 的变量必须在声明时给定初值，而在以后的引用中只能读取，不可修改。被声明为 final 的方法也同样只能使用，不能重写，但是能够重载。 使用 final 修饰的对象，对象的引用地址不能变，但是对象的值可以变！

finally 在异常处理时提供 finally 块来执行任何清除操作。如果有 finally 的话，则不管是否发生异常，finally 语句都会被执行。一般情况下，都把关闭物理连接(IO 流、数据库连接、Socket 连接)等相关操作，放入到此代码块中。

finalize 方法名。Java 技术允许使用 finalize() 方法在垃圾收集器将对象从内存中清除出去之前做必要清理工作。finalize() 方法是在垃圾收集器删除对象之前被调用的。它是在 Object 类中定义的，因此所有的类都继承了它。子类覆盖 finalize() 方法以整理系统资源或者执行其他清理工作。一般情况下，此方法由 JVM 调用，程序员不要去调用！

354. 写出 java.lang.Object 类的六个常用方法

(1)public boolean equals(java.lang.Object)

比较对象的地址值是否相等，如果子类重写，则比较对象的内容是否相等；

(2)public native int hashCode() 获取哈希码

(3)public java.lang.String toString() 把数据转变成字符串

(4)public final native java.lang.Class getClass() 获取类结构信息

(5)protected void finalize() throws java.lang.Throwable

垃圾回收前执行的方法

(6)protected native Object clone() throws java.lang.CloneNotSupportedException 克隆

(7)public final void wait() throws java.lang.InterruptedException

多线程中等待功能

(8)public final native void notify() 多线程中唤醒功能

(9)public final native void notifyAll() 多线程中唤醒所有等待线程的功能

355. private/默认/protected/public 权限修饰符的区别

	同一个类	同一个包中	子类	所有类
private	*			
default	*	*		
protected	*	*	*	
public	*	*	*	*

类的访问权限只有两种

public 公共的 可被同一项目中所有的类访问。(必须与文件名同名)

default 默认的 可被同一个包中的类访问。

成员（成员变量或成员方法）访问权限共有四种：

public 公共的　可以被项目中所有的类访问。(项目可见性)

protected 受保护的 可以被这个类本身访问；同一个包中的所有其他的类访问；被它的子类（同一个包以及不同包中的子类）访问。（子类可见性）

default 默认的被这个类本身访问 ;被同一个包中的类访问。(包可见性）

private 私有的　　只能被这个类本身访问。（类可见性）

356. 继承条件下构造方法的执行过程

继承条件下构造方法的调用规则如下：

情况 1：如果子类的构造方法中没有通过 super 显式调用父类的有参构造方法，也没有通过 this 显式调用自身的其他构造方法，则系统会默认先调用父类的无参构造方法。在这种情况下，写不写“super();”语句，效果是一样的。

情况 2：如果子类的构造方法中通过 super 显式调用父类的有参构造方法，那将执行父类相应构造方法，而不执行父类无参构造方法。

情况 3：如果子类的构造方法中通过 this 显式调用自身的其他构造方法，在相应构造方法中应用以上两条规则。

特别注意的是，如果存在多级继承关系，在创建一个子类对象时，以上规则会多次向更高一级父类应用，一直到执行顶级父类 Object 类的无参构造方法为止。

357. ==和 equals 的区别和联系

“==”是关系运算符，equals()是方法，同时他们的结果都返回布尔值；

“==”使用情况如下：

a) 基本类型，比较的是值

b) 引用类型，比较的是地址

c) 不能比较没有父子关系的两个对象

equals()方法使用如下：

a) 系统类一般已经覆盖了 equals()，比较的是内容。

b) 用户自定义类如果没有覆盖 equals()，将调用父类的 equals（比如是 Object），而 Object 的 equals 的比较是地址（return (this == obj);）

c) 用户自定义类需要覆盖父类的 equals()

注意：Object 的==和 equals 比较的都是地址，作用相同

358. 多态的技能点（前提条件，向上转型、向下转型）

实现多态的三个条件

1、继承的存在；（继承是多态的基础，没有继承就没有多态）

2、子类重写父类的方法。（多态下会调用子类重写后的方法）

3、父类引用变量指向子类对象。（涉及子类到父类的类型转换）

向上转型 Student person = new Student()

将一个父类的引用指向一个子类对象，成为向上转型，自动进行类型转换。此时通过父类引用变量调用的方法是子类覆盖或继承父类的方法，而不是父类的方法此时通过父类引用变量无法调用子类特有的方法

向下转型 Student stu = (Student)person;

将一个指向子类对象的引用赋给一个子类的引用，成为向下转型，此时必须进行强制类型转换。向下转型必须转换为父类引用指向的真实子类类型，，否则将出现 ClassCastException，不是任意的强制转换向下转型时可以结合使用 instanceof 运算符进行强制类型转换，比如出现转换异常---ClassCastException

359. 接口和抽象类的异同之处

相同点

1、抽象类和接口均包含抽象方法，子类必须实现所有的抽象方法，否则该类是抽象类；

2、抽象类和接口都不能实例化，他们位于继承树的顶端，用来被其他类继承和实现，需要通过对象的向上转型方式，为父类实例化对象。

不同点：

1、两者的区别主要体现在两方面：语法方面和设计理念方面

2、语法方面的区别是比较低层次的，非本质的，主要表现在：

1）接口中只能定义全局静态常量，不能定义变量。

2）抽象类中可以定义常量和变量。

3）接口中所有的方法都是全局抽象方法。

4）抽象类中可以有 0 个、1 个或多个，甚至全部都是抽象方法。

5）抽象类中可以有构造方法，但不能用来实例化，而在子类实例化是执行，完成属于抽象类的初始化操作。接口中不能定义构造方法。

6）一个类只能有一个直接父类（可以是抽象类），但可以充实实现多

个接口。一个类使用 extends 来继承抽象类，使用 implements 来实现接口。

二者的主要区别还是在设计理念上，其决定了某些情况下到底使用抽象类还是接口。

抽象类体现了一种继承关系，目的是复用代码，抽象类中定义了各个子类的相同代码，可以认为父类是一个实现了部分功能的“中间产品”，而子类是“最终产品”。父类和子类之间必须存在“is-a”的关系，即父类和子类在概念本质上应该是相同的。

接口并不要求实现类和接口在概念本质上一致的，仅仅是实现了接口定义的约定或者能力而已。接口定义了“做什么”，而实现类负责完成“怎么做”，体现了功能（规范）和实现分离的原则。接口和实现之间可以认为是一种“has-a 的关系”

360. 简述 Java 的垃圾回收机制

传统的 C/C++语言，需要程序员负责回收已经分配内存。

显式回收垃圾回收的缺点：

1）程序忘记及时回收，从而导致内存泄露，降低系统性能。

2）程序错误回收程序核心类库的内存，导致系统崩溃。

Java 语言不需要程序员直接控制内存回收，是由 JRE 在后台自动回收不再使用的内存，称为垃圾回收机制，简称 GC；

1）可以提高编程效率。

2）保护程序的完整性。

3）其开销影响性能。Java 虚拟机必须跟踪程序中有用的对象，确定哪

些是无用的。

垃圾回收机制的**特点**

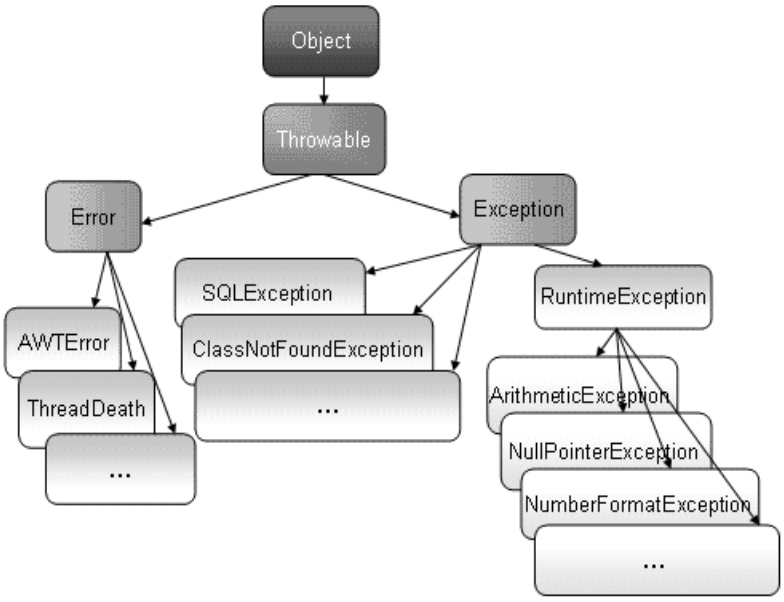
1）垃圾回收机制回收 JVM 堆内存里的对象空间,不负责回收栈内存数据。2）对其他物理连接，比如数据库连接、输入流输出流、Socket 连接无能为力。3）垃圾回收发生具有不可预知性，程序无法精确控制垃圾回收机制执行。4）可以将对象的引用变量设置为 null，暗示垃圾回收机制可以回收该对象。

现在的 JVM 有多种垃圾回收**实现算法，表现各异**。

垃圾回收机制回收任何对象之前，总会先调用它的 finalize 方法（如果覆盖该方法，让一个新的引用变量重新引用该对象，则会重新激活对象）。程序员可以通过 System.gc()或者 Runtime.getRuntime().gc()来通知系统进行垃圾回收，会有一些效果，但是系统是否进行垃圾回收依然不确定。

永远不要主动调用某个对象的 finalize 方法，应该交给垃圾回收机制调用。

361. Error 和 Exception 的区别



Error 类，表示仅靠程序本身无法恢复的严重错误，比如说内存溢出、动态链接异常、虚拟机错误。应用程序不应该抛出这种类型的对象。假如出现这种错误，除了尽力使程序安全退出外，在其他方面是无能为力的。所以在进行程序设计时，应该更关注 Exception 类。

Exception 类，由 Java 应用程序抛出和处理的非严重错误，比如所需文件没有找到、零作除数，数组下标越界等。它的各种不同子类分别对应不同类型异常。可分为两类：Checked 异常和 Runtime 异常

362. Checked 异常和 Runtime 异常的区别

运行时异常：包括 RuntimeaException 及其所有子类。不要求程序必须对它们作出处理，比如 InputMismatchException、ArithmeticException、NullPointerException 等。即使没有显示使用 try-catch 或 throws 进行处理，仍旧可以进行编译和运行(其实是 JVM 隐式的使用 try-catch 或 throws 进行处理)。如果运行时发生异常，会输出异常的堆栈信息并中止程序执行。

Checked 异常（非运行时异常）：除了运行时异常外的其他异常类都是 Checked 异常，程序必须捕获或者声明抛出这种异常，否则出现编译错误，无法通过编译。处理方式包括两种：通过 try-catch 捕获异常，通过 throws 声明抛出异常从而交给上一级调用方法处理。

363. Java 异常处理 try-catch-finally 的执行过程

try-catch-finally 程序块的执行流程以及执行结果比较复杂。

基本执行过程如下：

1）程序首先执行可能发生异常的 try 语句块。

2）如果 try 语句没有出现异常则执行完后跳至 finally 语句块执行；

3）如果 try 语句出现异常，则中断执行并根据发生的异常类型跳至相应的 catch 语句块执行处理。

4）catch 语句块可以有多个，分别捕获不同类型的异常。

5）catch 语句块执行完后程序会继续执行 finally 语句块。

finally 语句是可选的，如果有的话，则不管是否发生异常，finally 语句都会被执行。需要注意的是即使 try 和 catch 块中存在 return 语句，finally 语句也会执行，是在执行完 finally 语句后再通过 return 退出。

364. 异常处理中 throws 和 throw 的区别

1）**作用不同：**

throw 用于程序员自行产生并抛出异常；

throws 用于声明在该方法内抛出了异常

2） **使用的位置不同**：

throw 位于方法体内部，可以作为单独语句使用；

throws 必须跟在方法参数列表的后面，不能单独使用。

3）**内容不同**：

throw 抛出一个异常对象，且只能是一个；

throws 后面跟异常类，而且可以有多个。

365. 基本数据类型和包装类

1) 八个基本数据类型的包装类

基本数据类型　　包装类

byte Byte

boolean Boolean

short Short

char Character

int Integer

long Long

float Float

double Double

2) 为什么为基本类型引入包装类

2.1、基本数据类型有方便之处，简单、高效。

2.2、 但是 Java 中的基本数据类型却是不面向对象的（没有属性、方法），这在实际使用时存在很多的不便（比如集合的元素只能是 Object）。

为了解决这个不足，在设计类时为每个基本数据类型设计了一个对应的类进行包装，这样八个和基本数据类型对应的类统称为包装类(Wrapper Class)。

3) 包装类和基本数据类型之间的转换

3.1 包装类------ wrapperInstance.xxxValue() ------>基本数据类型

3.2 包 装 类 <---new WrapperClass(primitive) new WrapperClass(string)------基本数据类型

4)自动装箱和自动拆箱

JDK1.5 提供了自动装箱（autoboxing）和自动拆箱（autounboxing）

功能，从而实现了包装类和基本数据类型之间的自动转换

5)、包装类还可以实现基本类型变量和字符串之间的转换

基本类型变量----------String.valueof()---------->字符串

基本类型变量<----------WrapperClass.parseXxx(string)----------字符串

366. 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。

367. String 类为什么是 final 的

1)为了效率。若允许被继承，则其高度的 被使用率可能会降低程序的性能。

2 ）为了安全。JDK 中提供的好多核心类比如 String，这类的类的内部好多方法的实现都不是 java 编程语言本身编写的，好多方法都是调用的操作系

统本地的 API，这就是著名的“本地方法调用”，也只有这样才能做事，这种类是非常底层的， 和操作系统交流频繁的，那么如果这种类可以被继承的话，如果我们再把它的方法重写了，往操作系统内部写入一段具有恶意攻击性质的代码什么的， 这不就成了核心病毒了么？

不希望别人改，这个类就像一个工具一样，类的提供者给我们提供了， 就希望我们直接用就完了，不想让我们随便能改，其实说白了还是安全性， 如果随便能改了，那么 java 编写的程序肯定就很不稳定，你可以保证自己不乱改， 但是将来一个项目好多人来做，管不了别人，再说有时候万一疏忽了呢？他也不是估计的， 所以这个安全性是很重要的，java 和 C++相比，优点之一就包括这一点；

368. String、StringBuffer、StringBuilder 区别与联系

1）String 类是字符序列不可变的类，即一旦一个 String 对象被创建后，包含在这个对象中的字符序列是不可改变的，直至这个对象销毁。

2） StringBuffer 类则代表一个字符序列可变的字符串，可以通过 append、insert、reverse、setChartAt、setLength 等方法改变其内容。一旦生成了最终的字符串，调用 toString 方法将其转变为 String

3）JDK1.5 新增了一个 StringBuilder 类，与 StringBuffer 相似，构造方法和方法基本相同。不同是 StringBuffer 是线程安全的，而 StringBuilder 是线程不安全的，所以性能略高。通常情况下，创建一个内容可变的字符串，应该优先考虑使用 StringBuilder

369. String 类型是基本数据类型吗？基本数据类型有哪些

1)基本数据类型包括 byte、short/char、int、long、float、double、boolean

2)java.lang.String 类是引用数据类型，并且是 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

370. String s="Hello";s=s+"world!";执行后，s 内容是否改变？

答：没有改变。

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。

371. String s = new String("xyz");创建几个 String Object?

答：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。

372. 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。

373. java.sql.Date 和 java.util.Date 的联系和区别

1） java.sql.Date 是 java.util.Date 的子类，是一个包装了毫秒值的瘦包装器 ,允许 JDBC 将毫秒值标识为 SQL DATE 值。毫秒值表示自 1970 年 1 月 1 日 00:00:00 GMT 以来经过的毫秒数。 为了与 SQL DATE 的定义一致，由 java.sql.Date 实例包装的毫秒值必须通过将时间、分钟、秒和毫秒设置为与该实例相关的特定时区中的零来 “规范化” 。 说白了，java.sql.Date 就是与数据库 Date 相对应的一个类型，而 java.util.Date 是纯 java 的 Date。

2）JAVA 里提供的日期和时间类，java.sql.Date 和 java.sql.Time,只会从数据库里读取某部分值，这有时会导致丢失数据。例如一个包含 2002/05/22 5:00:57 PM 的字段，读取日期时得到的是 2002/05/22,而读取时间时得到的是 5:00:57 PM. 你需要了解数据库里存储时间的精度。有些数据库，比如 MySQL,精度为毫秒，然而另一些数据库，包括 Oracle,存储 SQL DATE 类型数据时，毫秒部分的数据是不保存的。以下操作中容易出现不易被发现的 BUG：获得一个 JAVA 里的日期对象。 从数据库里读取日期 试图比较两个日期对象是否相等。如果毫秒部分丢失，本来认为相等的两个日期对象用 Equals 方法可能返回 false。.sql.Timestamp 类比 java.util.Date 类精确度要高。这个类包了一个 getTime()方法，但是它不会返回额外精度部分的数据，因此必须使用...

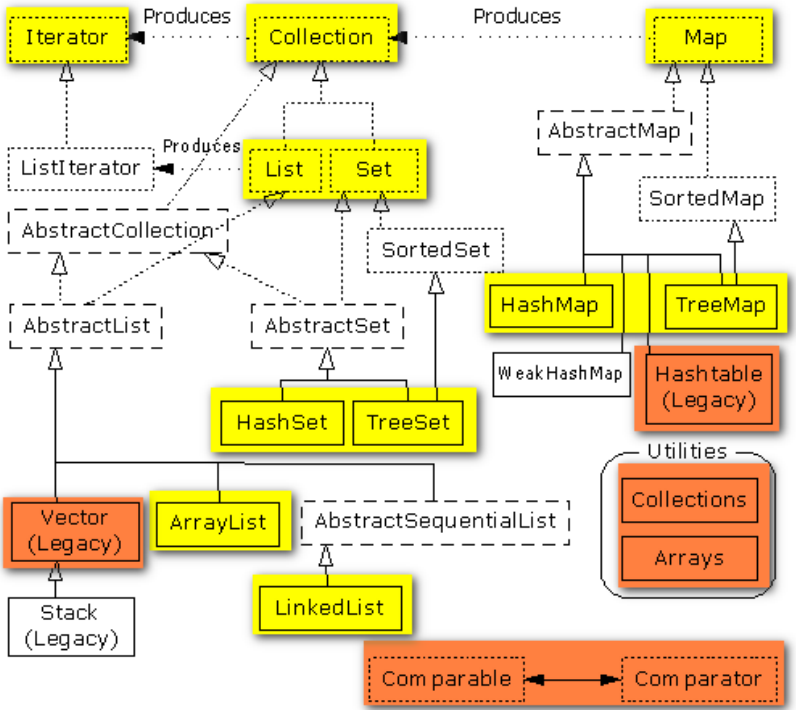
总之，java.util.Date 就是 Java 的日期对象，而 java.sql.Date 是针对 SQL 语句使用的，只包含日期而没有时间部分。

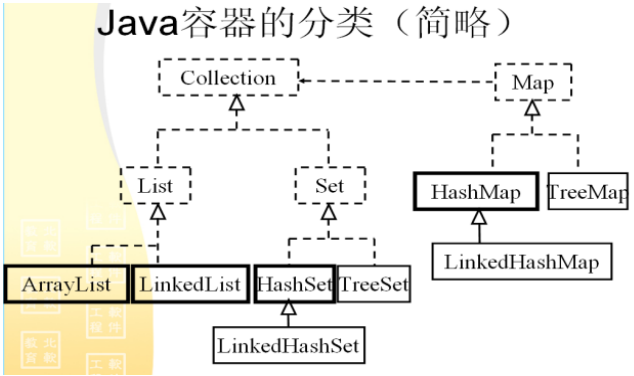
374. 使用递归算法输出某个目录下所有文件和子目录列表

```
package com.bjsxt;

import java.io.File;

public class $ {

    public static void main(String[] args) {

        String path = "D:/301SXT";

        test(path);

    }

    private static void test(String path) {

        File f = new File(path);

        File[] fs = f.listFiles();

        if (fs == null) {

            return;

        }

        for (File file : fs) {

            if (file.isFile()) {

                System.out.println(file.getPath());

            } else {

                test(file.getPath());

            }
```

```
        }
    }
```

375. Java 集合体系结构（List、Set、Collection、Map 的区别和联系）





1、Collection 接口存储一组不唯一，无序的对象

2、List 接口存储一组不唯一，有序（插入顺序）的对象

3、Set 接口存储一组唯一，无序的对象

4、Map 接口存储一组键值对象，提供 key 到 value 的映射。Key 无序，唯一。value 不要求有序，允许重复。（如果只使用 key 存储，而不使用 value，那就是 Set）

376. Vector 和 ArrayList 的区别和联系

相同点：

1）实现原理相同---底层都使用数组

2）功能相同---实现增删改查等操作的方法相似

3）都是长度可变的数组结构，很多情况下可以互用

不同点：

1）Vector 是早期 JDK 接口，ArrayList 是替代 Vector 的新接口

2）Vector 线程安全，ArrayList 重速度轻安全，线程非安全

长度需增长时，Vector 默认增长一倍，ArrayList 增长 50%

377. ArrayList 和 LinkedList 的区别和联系

相同点：

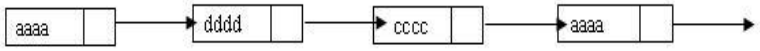
两者都实现了 List 接口，都具有 List 中元素有序、不唯一的特点。

不同点：

ArrayList 实现了长度可变的数组，在内存中分配连续空间。遍历元素和随机访问元素的效率比较高；

0	1	2	3	4	5	
aaaa	dddd	cccc	aaaa	eeee	dddd	

LinkedList 采用链表存储方式。插入、删除元素时效率比较高



378. HashMap 和 Hashtable 的区别和联系

相同点：

实现原理相同，功能相同，底层都是哈希表结构，查询速度快，在很多情况下可以互用

不同点：

1、Hashtable 是早期提供的接口，HashMap 是新版 JDK 提供的接口

2、Hashtable 继承 Dictionary 类，HashMap 实现 Map 接口

3、Hashtable 线程安全，HashMap 线程非安全

4、Hashtable 不允许 null 值，HashMap 允许 null 值

379. HashSet 的使用和原理（hashCode()和 equals()）

1）哈希表的查询速度特别快，时间复杂度为 O（1）。

2）HashMap、Hashtable、HashSet 这些集合采用的是哈希表结构，需要用到 hashCode 哈希码，hashCode 是一个整数值。

3）系统类已经覆盖了 hashCode 方法 自定义类如果要放入 hash 类集合，必须重写 hashcode。如果不重写，调用的是 Object 的 hashcode，而 Object 的 hashCode 实际上是地址。

4）向哈希表中添加数据的原理：当向集合 Set 中增加对象时，首先集合计算要增加对象的 hashCode 码，根据该值来得到一个位置用来存放当前对象，如在该位置没有一个对象存在的话，那么集合 Set 认为该对象在集合中不存在，直接增加进去。如果在该位置有一个对象存在的话，接着将准备增加到集合中的对象与该位置上的对象进行 equals 方法比较，

如果该 equals 方法返回 false,那么集合认为集合中不存在该对象，在进行一次散列，将该对象放到散列后计算出的新地址里。如果 equals 方法返回 true，那么集合认为集合中已经存在该对象了，不会再将该对象增加到集合中了。

5）在哈希表中判断两个元素是否重复要使用到 hashCode()和 equals()。hashCode 决定数据在表中的存储位置，而 equals 判断是否存在相同数据。

6）Y=K(X) ：K 是函数，X 是哈希码，Y 是地址

380. TreeSet 的原理和使用（Comparable 和 comparator）

1）TreeSet 集合，元素不允许重复且有序(**自然顺序**)

2）TreeSet 采用树结构存储数据，存入元素时需要和树中元素进行对比，需要指定比较策略。

3）可以通过 Comparable(外部比较器)和 Comparator(内部比较器)来指定比较策略，实现了 Comparable 的系统类可以顺利存入 TreeSet。自定义类可以实现 Comparable 接口来指定比较策略。

4）可创建 Comparator 接口实现类来指定比较策略，并通过 TreeSet 构造方法参数传入。这种方式尤其对系统类非常适用。

381. 集合和数组的比较（为什么引入集合）

数组不是面向对象的，存在明显的缺陷，集合完全弥补了数组的一些缺点，比数组更灵活更实用，可大大提高软件的开发效率而且不同的集合框架类可适用于不同场合。具体如下：

1）数组的效率高于集合类.

2）数组能存放基本数据类型和对象，而集合类中只能放对象。

3）数组容量固定且无法动态改变，集合类容量动态改变。

4）数组无法判断其中实际存有多少元素，length 只告诉了 array 的容量。

5）集合有多种实现方式和不同的适用场合，而不像数组仅采用顺序表方式。

6）集合以类的形式存在，具有封装、继承、多态等类的特性，通过简单的方法和属性调用即可实现各种复杂操作，大大提高软件的开发效率。

382. Collection 和 Collections 的区别

1）Collection 是 Java 提供的集合接口，存储一组不唯一，无序的对象。它有两个子接口 List 和 Set。

2）Java 中还有一个 Collections 类，专门用来操作集合类 ，它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。

383. 输入流和输出流联系和区别，节点流和处理流联系和区别

首先，你要明白什么是“流”。直观地讲，流就像管道一样，在程序和文件之间，输入输出的方向是针对程序而言，向程序中读入东西，就是输入流，从程序中向外读东西，就是输出流。

输入流是得到数据，输出流是输出数据，而节点流，处理流是流的另一种划分，按照功能不同进行的划分。节点流，可以从或向一个特定的地方(节点)读写数据。处理流是对一个已存在的流的连接和封装，通过所封装的流的功能调用实现数据读写。如 BufferedReader。处理流的构造方法总是要带一个其他的流对象做参数。一个流对象经过其他流的多次包装，称为流

的链接。

384. 字符流字节流联系区别；什么时候使用字节流和字符流?

字符流和字节流是流的一种划分，按处理照流的数据单位进行的划分。两类都分为输入和输出操作。在字节流中输出数据主要是使用 OutputStream 完成，输入使的是 InputStream，在字符流中输出主要是使用 Writer 类完成，输入流主要使用 Reader 类完成。这四个都是抽象类。

字符流处理的单元为 2 个字节的 Unicode 字符，分别操作字符、字符数组或字符串，而字节流处理单元为 1 个字节，操作字节和字节数组。字节流是最基本的 ,所有的 InputStrem 和 OutputStream 的子类都是,主要用在处理二进制数据，它是按字节来处理的 但实际中很多的数据是文本，又提出了字符流的概念，它是按虚拟机的编码来处理，也就是要进行字符集的转化 这两个之间通过 InputStreamReader,OutputStreamWriter 来关联，实际上是通过 byte[]和 String 来关联的。

385. 列举常用字节输入流和输出流并说明其特点，至少 5 对。

答：

FileInputStream 从文件系统中的某个文件中获得输入字节。

FileOutputStream 从程序当中的数据，写入到指定文件。

ObjectInputStream 对以前使用 ObjectOutputStream 写入的基本数据和对象进行反序列化。

ObjectOutputStream 和 ObjectInputStream 分别与 FileOutputStream 和 FileInputStream 一起使用时，可以为应用程序提供对对象图形的持久存储。ObjectInputStream 用于恢复那些以前序列化

的对象。其他用途包括使用套接字流在主机之间传递对象，或者用于编组和解组远程通信系统中的实参和形参。

ByteArrayInputStream 包含一个内部缓冲区，该缓冲区包含从流中读取的字节。内部计数器跟踪 read 方法要提供的下一个字节。

FilterInputStream 包含其他一些输入流，它将这些流用作其基本数据源，它可以直接传输数据或提供一些额外的功能。FilterInputStream 类本身只是简单地重写那些将所有请求传递给所包含输入流的 InputStream 的所有方法。FilterInputStream 的子类可进一步重写这些方法中的一些方法，并且还可以提供一些额外的方法和字段。

StringBufferInputStream 此类允许应用程序创建输入流，在该流中读取的字节由字符串内容提供。应用程序还可以使用 ByteArrayInputStream 从 byte 数组中读取字节。 只有字符串中每个字符的低八位可以由此类使用。

ByteArrayOutputStream 此类实现了一个输出流，其中的数据被写入一个 byte 数组。缓冲区会随着数据的不断写入而自动增长。可使用 toByteArray() 和 toString() 获取数据。

FileOutputStream 文件输出流是用于将数据写入 File 或 FileDescriptor 的输出流。文件是否可用或能否可以被创建取决于基础平台。特别是某些平台一次只允许一个 FileOutputStream（或其他文件写入对象）打开文件进行写入。在这种情况下，如果所涉及的文件已经打开，则此类中的构造方法将失败。

FilterOutputStream 类是过滤输出流的所有类的超类。这些流位于已存在的输出流（基础 输出流）之上，它们将已存在的输出流作为其基本数据接收器，但可能直接传输数据或提供一些额外的功能。 FilterOutputStream 类本身只是简单地重写那些将所有请求传递给所包含输出流的 OutputStream 的所有方法。FilterOutputStream 的子类可进一步地重写这些方法中的一些方法，并且还可以提供一些额外的方法和字段。

ObjectOutputStream 将 Java 对象的基本数据类型和图形写入 OutputStream。可以使用 ObjectInputStream 读取（重构）对象。通过在流中使用文件可以实现对象的持久存储。如果流是网络套接字流，则可以在另一台主机上或另一个进程中重构对象。

PipedOutputStream 可以将管道输出流连接到管道输入流来创建通信管道。管道输出流是管道的发送端。通常，数据由某个线程写入 PipedOutputStream 对象，并由其他线程从连接的 PipedInputStream 读取。不建议对这两个对象尝试使用单个线程，因为这样可能会造成该线程死锁。如果某个线程正从连接的管道输入流中读取数据字节，但该线程不再处于活动状态，则该管道被视为处于毁坏状态。

386. 说明缓冲流的优点和原理

不带缓冲的流的工作原理：

它读取到一个字节/字符，就向用户指定的路径写出去，读一个写一个，所以就慢了。

带缓冲的流的工作原理：

读取到一个字节/字符，先不输出，等凑足了缓冲的最大容量后一次

性写出去，从而提高了工作效率

优点：减少对硬盘的读取次数，降低对硬盘的损耗。

387. 序列化的定义、实现和注意事项

想把一个对象写在硬盘上或者网络上，对其进行序列化，把他序列化成为一个字节流。

实现和注意事项：

1）实现接口 Serializable Serializable 接口中没有任何的方法，实现该接口的类不需要实现额外的方法。

2）如果对象中的某个属性是对象类型，必须也实现 Serializable 接口才可以，序列化对静态变量无效

3）如果不希望某个属性参与序列化，不是将其 static，而是 transient 串行化保存的只是变量的值，对于变量的任何修饰符，都不能保存序列化版本不兼容

388. 使用 IO 流完成文件夹复制（结合递归）

```
package com.bjsxt;


import java.io.*;
/**
 * CopyDocJob定义了实际执行的任务，即
 * 从源目录拷贝文件到目标目录
*/
public class CopyDir2 {
```

```
    public static void main(String[] args) {
        try {
            copyDirectiory("d:/301sxt","d:/301sxt2");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    /**
     * 复制单个文件
     * @param sourceFile 源文件
     * @param targetFile 目标文件
     * @throws IOException
     */
    private static void copyFile(File sourceFile, File targetFile) throws
IOException {
        BufferedInputStream inBuff = null;
        BufferedOutputStream outBuff = null;
        try {
            // 新建文件输入流
            inBuff = new BufferedInputStream(new
FileInputStream(sourceFile));
            // 新建文件输出流
```

```
                outBuff = new BufferedOutputStream(new
FileOutputStream(targetFile));
                // 缓冲数组
                byte[] b = new byte[1024 * 5];
                int len;
                while ((len = inBuff.read(b)) != -1) {
                    outBuff.write(b, 0, len);
                }
                // 刷新此缓冲的输出流
                outBuff.flush();
            } finally {
                // 关闭流
                if (inBuff != null)
                    inBuff.close();
                if (outBuff != null)
                    outBuff.close();
            }
        }

        /**
         * 复制目录
         * @param sourceDir 源目录
```

```
     * @param targetDir  目标目录
     * @throws IOException
     */
    private static void copyDirectiory(String sourceDir, String targetDir)
throws IOException {
        // 检查源目录
        File fSourceDir = new File(sourceDir);
        if(!fSourceDir.exists() || !fSourceDir.isDirectory()){
            return;
        }
        //检查目标目录，如不存在则创建
        File fTargetDir = new File(targetDir);
        if(!fTargetDir.exists()){
            fTargetDir.mkdirs();
        }
        // 遍历源目录下的文件或目录
        File[] file = fSourceDir.listFiles();
        for (int i = 0; i < file.length; i++) {
            if (file[i].isFile()) {
                // 源文件
                File sourceFile = file[i];
                // 目标文件
```

```
                    File targetFile = new File(fTargetDir, file[i].getName());
                    copyFile(sourceFile, targetFile);
                }
                //递归复制子目录
                if (file[i].isDirectory()) {
                    // 准备复制的源文件夹
                    String subSourceDir = sourceDir + File.separator +
file[i].getName();
                    // 准备复制的目标文件夹
                    String subTargetDir = targetDir + File.separator +
file[i].getName();
                    // 复制子目录
                    copyDirectiory(subSourceDir, subTargetDir);
                }
            }
        }
}
```

389. 进程和线程有什么联系和区别？

下面我以一个日常生活中简单的例子来说明**进程**和线程之间的区别和联系：

这副图是一个双向多车道的道路图，假如我们把整条道路看成是一个"**进程**"的话，那么图中由白色虚线分隔开来的各个车道就是**进程**中的各个"线程"了。

①这些线程(车道)共享了**进程**(道路)的公共资源(土地资源)。

②这些线程(车道)必须依赖于**进程**(道路)，也就是说，线程不能脱离于**进程**而存在(就像离开了道路，车道也就没有意义了)。

③这些线程(车道)之间可以并发执行(各个车道你走你的，我走我的)，也可以互相同步(某些车道在交通灯亮时禁止继续前行或转弯，必须等待其它车道的车辆通行完毕)。

④这些线程(车道)之间依靠代码逻辑(交通灯)来控制**运行**，一旦代码逻辑控制有误(死锁，多个线程同时竞争唯一资源)，那么线程将陷入混乱，无序之中。

⑤这些线程(车道)之间谁先**运行**是未知的，只有在线程刚好被分配到 CPU 时间片(交通灯变化)的那一刻才能知道

1.定义：

进程是具有一定独立功能的程序关于某个数据集合上的一次运行活动，是系统进行资源分配和调度的一个独立单位。

线程是进程的一个实体，是 CPU 调度和分派的基本单位，他是比进程更小的能独立运行的基本单位，线程自己基本上不拥有系统资源，只拥有一点在运行中必不可少的资源（如程序计数器，一组寄存器和栈），一个线程可以创建和撤销另一个线程；

2.进程和线程的关系：

（1）一个线程只能属于一个进程，而一个进程可以有多个线程，但至少有一个线程。

（2）资源分配给进程，同一进程的所有线程共享该进程的所有资源。

（3）线程在执行过程中，需要协作同步。不同进程的线程间要利用消息通信的办法实现同步。

（4）处理机分给线程，即真正在处理机上运行的是线程。

（5）线程是指进程内的一个执行单元，也是进程内的可调度实体。

3.线程与进程的区别：

（1）调度：线程作为调度和分配的基本单位，进程作为拥有资源的基本单位。

（2）并发性：不仅进程之间可以并发执行，同一个进程的多个线程之间也可以并发执行。

（3）拥有资源：进程是拥有资源的一个独立单位，线程不拥有系统资源，但可以访问隶属于进程的资源。

（4）系统开销：在创建或撤销进程的时候，由于系统都要为之分配和回收资源，导致系统的明显大于创建或撤销线程时的开销。但进程有独立的地址空间，进程崩溃后，在保护模式下不会对其他的进程产生影响，而线程只是一个进程中的不同的执行路径。线程有自己的堆栈和局部变量，但线程之间没有单独的地址空间，一个线程死掉就等于整个进程死掉，所以多进程的程序要比多线程的程序健壮，但是在进程切换时，耗费的资源较大，效率要差些

390. 创建线程的两种方式分别是什么,优缺点是什么？

方式 1：继承 Java.lang.Thread 类，并覆盖 run() 方法。

优势：编写简单；

劣势：单继承的限制----无法继承其它父类，同时不能实现资源共享。

```
package com.bjsxt;


public class ThreadDemo1 {
    public static void main(String args[]) {
        MyThread1 t = new MyThread1();
        t.start();
        while (true) {
            System.out.println("兔子领先了，别骄傲");
        }
    }
}
class MyThread1 extends Thread {
    public void run() {
        while (true) {
            System.out.println("乌龟领先了，加油");
        }
    }
```

```
}
```

方式 2：实现 Java.lang.Runnable 接口，并实现 run()方法。

优势：可继承其它类，多线程可共享同一个 Thread 对象；

劣势：编程方式稍微复杂，如需访问当前线程，需调用 Thread.currentThread()方法

```
package com.bjsxt;


public class ThreadDemo2 {
    public static void main(String args[]) {
        MyThread2 mt = new MyThread2();
        Thread t = new Thread(mt);
        t.start();
        while (true) {
            System.out.println("兔子领先了，加油");
        }
    }
}
class MyThread2 implements Runnable {
    public void run() {
        while (true) {
```

```
            System.out.println("乌龟超过了，再接再厉");
        }
    }
}
```

391. Java 创建线程后，调用 start()方法和 run()的区别

两种方法的区别

1） start 方法：

用 start 方法来启动线程，真正实现了多线程运行，这时无需等待 run 方法体代码执行完毕而直接继续执行下面的代码。通过调用 Thread 类的 start()方法来启动一个线程，这时此线程处于就绪（可运行）状态，并没有运行，一旦得到 cpu 时间片，就开始执行 run()方法，这里方法 run()称为线程体，它包含了要执行的这个线程的内容，Run 方法运行结束，此线程随即终止。

2） run（ ）:

run()方法只是类的一个普通方法而已，如果直接调用 run 方法，程序中依然只有主线程这一个线程，其程序执行路径还是只有一条，还是要顺序执行，还是要等待

run 方法体执行完毕后才可继续执行下面的代码，这样就没有达到写线程的目的。

总结：调用 start 方法方可启动线程，而 run 方法只是 thread 的一

个普通方法调用，还是在主线程里执行。这两个方法应该都比较熟悉，把需要并行处理的代码放在 run()方法中，start()方法启动线程将自动调用 run()方法，这是由 jvm 的内存机制规定的。并且 run()方法必须是 public 访问权限，返回值类型为 void。

两种方式的比较 ：

实际中往往采用实现 Runable 接口，一方面因为 java 只支持单继承，继承了 Thread 类就无法再继续继承其它类，而且 Runable 接口只有一个 run 方法；另一方面通过结果可以看出实现 Runable 接口才是真正的多线程。

392. 线程的生命周期

线程是一个动态执行的过程，它也有一个从产生到死亡的过程。

(1)生命周期的五种状态

新建（new Thread）

当创建 Thread 类的一个实例（对象）时，此线程进入新建状态（未被启动）。

例如：Thread t1=new Thread();

就绪（runnable）

线程已经被启动，正在等待被分配给 CPU 时间片，也就是说此时线程正在就绪队列中排队等候得到 CPU 资源。例如：t1.start();

运行（running）

线程获得 CPU 资源正在执行任务（run()方法），此时除非此线程自动放弃 CPU 资源或者有优先级更高的线程进入，线程将一直运行到结束。

死亡（dead）

当线程执行完毕或被其它线程杀死，线程就进入死亡状态，这时线程不可能再进入就绪状态等待执行。

自然终止：正常运行 run()方法后终止

异常终止：调用 stop()方法让一个线程终止运行

堵塞（blocked）

由于某种原因导致正在运行的线程让出 CPU 并暂停自己的执行，即进入堵塞状态。

正在睡眠：用 sleep(long t) 方法可使线程进入睡眠方式。一个睡眠着的线程在指定的时间过去可进入就绪状态。

正在等待：调用 wait()方法。（调用 motify()方法回到就绪状态）

被另一个线程所阻塞：调用 suspend()方法。（调用 resume()方法恢复）

393. 如何实现线程同步？

当多个线程访问同一个数据时，容易出现线程安全问题，需要某种方式来确保资源在某一时刻只被一个线程使用。需要让线程同步，保证数据安全

线程同步的实现方案：

1）同步代码块，使用 synchronized 关键字

同步代码块：

```
synchronized (同步锁) {
        授课代码;
    }
```

同步方法：

public synchronized void makeWithdrawal(int amt) {}

线程同步的好处：解决了线程安全问题

线程同步的缺点：性能下降，可能会带来死锁

注意： 同步代码块，所使用的同步锁可以是三种，

1、this 2、 共享资源 3、 字节码文件对象

同步方法所使用的同步锁，默认的是 this

394. 关于同步锁的更多细节

Java 中每个对象都有一个内置锁。

当程序运行到非静态的 synchronized 同步方法上时，自动获得与正在执行代码类的当前实例（ this 实例 ）有关的锁。获得一个对象的锁也称为获取锁、锁定对象、在对象上锁定或在对象上同步。

当程序运行到 synchronized 同步方法或代码块时才该对象锁才起作用。

一个对象只有一个锁。所以，如果一个线程获得该锁，就没有其他线程可以获得锁，直到第一个线程释放（或返回）锁。这也意味着任何其他线程都不能进入该对象上的 synchronized 方法或代码块，直到该锁被释放。

释放锁是指持锁线程退出了 synchronized 同步方法或代码块。

关于锁和同步，有一下几个要点：

1）、只能同步方法，而不能同步变量和类；

2）、每个对象只有一个锁；当提到同步时，应该清楚在什么上同步？也就是说，在哪个对象上同步？

3）、不必同步类中所有的方法，类可以同时拥有同步和非同步方法。

4）、如果两个线程要执行一个类中的 synchronized 方法，并且两个线程使用相同的实例来调用方法，那么一次只能有一个线程能够执行方法，另一个需要等待，直到锁被释放。也就是说：如果一个线程在对象上获得一个锁，就没有任何其他线程可以进入（该对象的）类中的任何一个同步方法。

5）、如果线程拥有同步和非同步方法，则非同步方法可以被多个线程自由访问而不受锁的限制。

6）、线程睡眠时，它所持的任何锁都不会释放。

7）、线程可以获得多个锁。比如，在一个对象的同步方法里面调用另外一个对象的同步方法，则获取了两个对象的同步锁。

8）、同步损害并发性，应该尽可能缩小同步范围。同步不但可以同步整个方法，还可以同步方法中一部分代码块。

9）、在使用同步代码块时候，应该指定在哪个对象上同步，也就是说要获取哪个对象的锁。

395. 简述 sleep( )和 wait( )有什么区别？

1）sleep()线程休眠，是让某个线程暂停运行一段时间,其控制范围是由当前线程决定,也就是说,在线程里面决定.好比如说,我要做的事情是 "点火->烧水->煮面",而当我点完火之后我不立即烧水,我要休息一段时间再烧.对于运行的主动权是由我的流程来控制。

2）wait()线程等待,首先,这是由某个确定的对象来调用的,将这个对象理解成一个传话的人,当这个人在某个线程里面说"暂停!",也是 thisObj.wait(),这里的暂停是阻塞,还是"点火->烧水->煮饭",thisObj 就好比一个监督我的人站

在我旁边,本来该线 程应该执行 1 后执行 2,再执行 3,而在 2 处被那个对象喊暂停,那么我就会一直等在这里而不执行 3,但正个流程并没有结束,我一直想去煮饭,但还没被允许，直到那个对象在某个地方说"通知暂停的线程启动!",也就是 thisObj.notify()的时候,那么我就可以煮饭了,这个被暂停的线程就会从暂停处 继续执行。

3）其实两者都可以让线程暂停一段时间,但是本质的区别是一个线程的运行状态控制,一个是线程之间的通讯的问题

396. Java 中实现线程通信的三个方法的作用是什么？

Java 提供了 3 个方法解决线程之间的通信问题，均是 java.lang.Object 类的方法，都只能在同步方法或者同步代码块中使用，否则会抛出异常。

方法名	作　用
final void wait()	表示线程一直等待，直到其它线程通知
void wait(long timeout)	线程等待指定毫秒参数的时间
final void wait(long timeout,int nanos)	线程等待指定毫秒、微妙的时间
final void 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。
final void notifyAll()	唤醒同一个对象上所有调用 wait()方法的线程，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

397. IP 地址和端口号

1）IP 地址

用来标志网络中的一个通信实体的地址。通信实体可以是计算机，路由器等。

2）IP 地址分类

IPV4：32 位地址，以点分十进制表示，如 192.168.0.1

IPV6：128 位（16 个字节）写成 8 个 16 位的无符号整数，每个整数用四个十六进制位表示，数之间用冒号（：）分开，如：3ffe:3201:1401:1280:c8ff:fe4d:db39:1984

3）特殊的 IP 地址

127.0.0.1 本机地址

192.168.0.0--192.168.255.255 私有地址，属于非注册地址，专门为组织机构内部使用。

4）端口:port

IP 地址用来标志一台计算机，但是一台计算机上可能提供多种应用程序，使用端口来区分这些应用程序。 端口是虚拟的概念，并不是说在主机上真的有若干个端口。通过端口，可以在一个主机上运行多个网络

应用程序。 端口范围 0---65535,16 位整数

5）端口分类

公认端口 0—1023 比如 80 端口分配给 WWW ,21 端口分配给 FTP，22 端口分配给 SSH,23 端口分配给 telnet，25 端口分配给 smtp

注册端口 1024—49151 分配给用户进程或应用程序

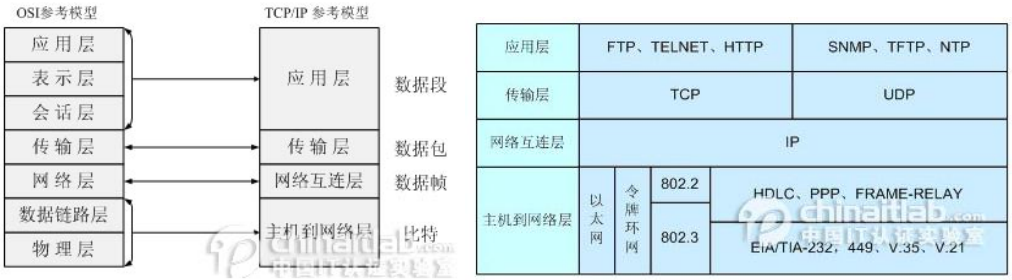
动态/私有端口 49152--65535

6）理解 IP 和端口的关系

IP 地址好比每个人的地址（门牌号），端口好比是房间号。必须同时指定 IP 地址和端口号才能够正确的发送数据

IP 地址好比为电话号码，而端口号就好比为分机号。

398. 介绍 OSI 七层模型和 TCP/IP 模型



OSI(Open System Interconnection)，开放式系统互联参考模型 。是一个逻辑上的定义，一个规范，它把网络协议从逻辑上分为了 7 层。每一层都有相关、相对应的物理设备，比如常规的路由器是三层交换设备，常规的交换机是二层交换设备。OSI 七层模型是一种框架性的设计方法，建立七层模型的主要目的是为解决异种网络互连时所遇到的兼容性问题，其最主要的功能就是帮助不同类型的主机实现数据传输。它的最大优点是将服务、接口和协议这三个概念明确地区分开来，通过七个层次化的结构模型使不同的系统不

同的网络之间实现可靠的通讯。

TCP/IP 协议是 Internet 最基本的协议、Internet 国际互联网络的基础，主要由网络层的 IP 协议和传输层的 TCP 协议组成。TCP/IP 定义了电子设备如何连入因特网，以及数据如何在它们之间传输的标准。协议采用了 4 层的层级结构，每一层都呼叫它的下一层所提供的协议来完成自己的需求。

ISO 制定的 OSI 参考模型的过于庞大、复杂招致了许多批评。伴随着互联网的流行，其本身所采用的 TCP/IP 协议栈获得了更为广泛的应用和认可。在 TCP/IP 参考模型中，去掉了 OSI 参考模型中的会话层和表示层（这两层的功能被合并到应用层实现）。同时将 OSI 参考模型中的数据链路层和物理层合并为主机到网络层。

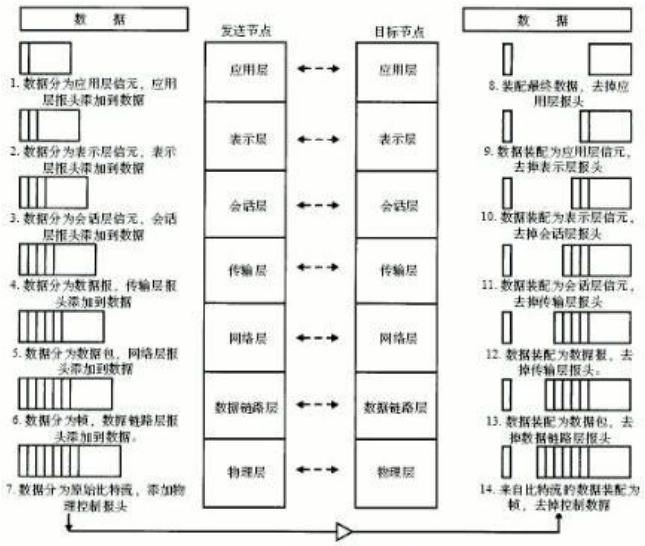


图1-2 OSI模型如何传输数据

399. TCP 协议和 UDP 协议的比较

TCP 和 UDP 是 TCP/IP 协议栈中传输层的两个协议，它们使用 IP 路由功能把数据包发送到目的地，从而为应用程序及应用层协议（包括：HTTP、SMTP、SNMP、FTP 和 Telnet）提供网络服务。

TCP 的 server 和 client 之间通信就好比两个人打电话，需要互相知道

对方的电话号码，然后开始对话。所以在两者的连接过程中间需要指定端口和地址。

UDP 的 server 和 client 之间的通信就像两个人互相发信。我只需要知道对方的地址，然后就发信过去。对方是否收到我不知道，也不需要专门对口令似的来建立连接。具体区别如下：

1）TCP 是面向连接的传输。UDP 是无连接的传输

2）TCP 有流量控制、拥塞控制，检验数据数据按序到达，而 UDP 则相反。

3）TCP 的路由选择只发生在建立连接的时候，而 UDP 的每个报文都要进行路由选择

4）TCP 是可靠性传输，他的可靠性是由超时重发机制实现的，而 UDP 则是不可靠传输

5）UDP 因为少了很多控制信息，所以传输速度比 TCP 速度快

6）TCP 适合用于传输大量数据，UDP 适合用于传输小量数据

400. 什么是 Socket 编程

Socket 编程的定义如下：

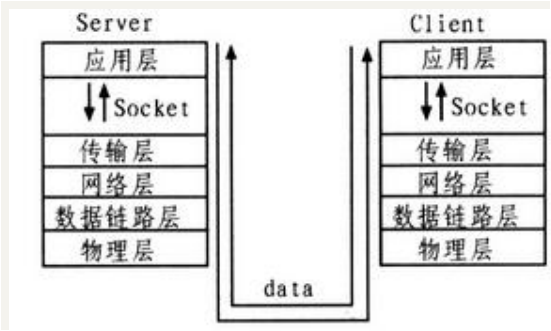
所谓 socket 通常也称作"套接字"，用于描述 IP 地址和端口，是一个通信链的句柄。应用程序通常通过"套接字"向网络发出请求或者应答网络请求。

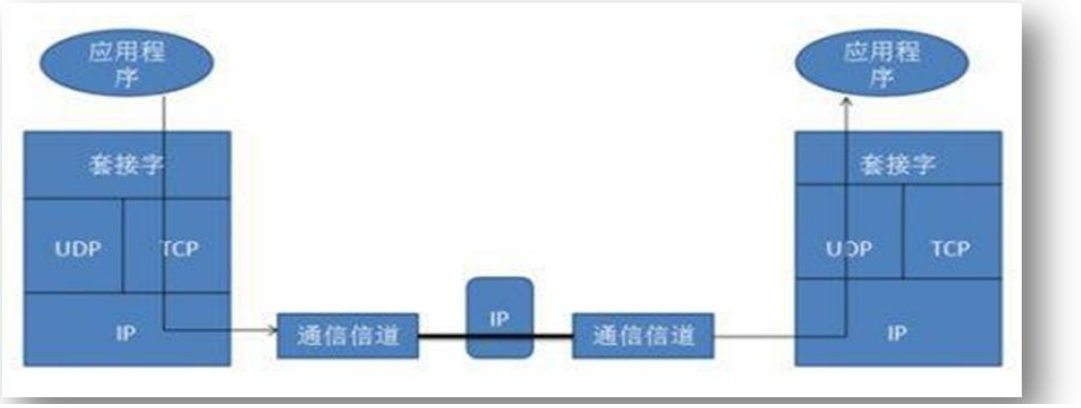
我们开发的网络应用程序位于应用层，TCP 和 UDP 属于传输层协议，在应用层如何使用传输层的服务呢？在应用层和传输层之间，则是使用套接字来进行分离。

套接字就像是传输层为应用层开的一个小口，应用程序通过这个小口向远程发送数据，或者接收远程发来的数据；而这个小口以内，也就是数据进入这个口之后，或者数据从这个口出来之前，是不知道也不需要知道的，也不会关心它如何传输，这属于网络其它层次的工作。

Socket 实际是传输层供给应用层的编程接口。传输层则在网络层的基础上提供进程到进程问的逻辑通道，而应用层的进程则利用传输层向另一台主机的某一进程通信。Socket 就是应用层与传输层之间的桥梁

使用 Socket 编程可以开发客户机和服务器应用程序，可以在本地网络上进行通信，也可通过 Internet 在全球范围内通信。





生活案例 1 如果你想写封邮件发给远方的朋友，如何写信、将信打包，属于应用层。信怎么写，怎么打包完全由我们做主；而当我们将信投入邮筒时，邮筒的那个口就是套接字，在进入套接字之后，就是传输层、网络

层等（邮局、公路交管或者航线等）其它层次的工作了。我们从来不会去关心信是如何从西安发往北京的，我们只知道写好了投入邮筒就 OK 了。

生活案例 2：可以把 Socket 比作是一个港口码头，应用程序只要将数据交给 Socket，就算完成了数据的发送，具体细节由 Socket 来完成，细节不必了解。同理，对于接收方，应用程序也要创建一个码头，等待数据的到达，并获取数据。

401. 简述基于 TCP 和 UDP 的 Socket 编程的主要步骤

Java 分别为 TCP 和 UDP 两种通信协议提供了相应的 Socket 编程类，这些类存放在 java.net 包中。与 TCP 对应的是服务器的 ServerSocket 和客户端的 Socket，与 UDP 对应的是 DatagramSocket。

基于 TCP 创建的套接字可以叫做流套接字，服务器端相当于一个监听器，用来监听端口。 服务器与客服端之间的通讯都是输入输出流来实现的。基于 UDP 的套接字就是数据报套接字，• 两个都要先构造好相应的数据包。基于 TCP 协议的 Socket 编程的主要步骤

服务器端（server）:

1. 构建一个 ServerSocket 实例，指定本地的端口。这个 socket 就是用来监听指定端口的连接请求的。

2.重复如下几个步骤：

a. 调用 socket 的 accept()方法来获得下面客户端的连接请求。通过 accept()方法返回的 socket 实例，建立了一个和客户端的新连接。

b.通过这个返回的 socket 实例获取 InputStream 和 OutputStream,

可以通过这两个 stream 来分别读和写数据。

c.结束的时候调用 socket 实例的 close()方法关闭 socket 连接。

客户端（client）:

1.构建 Socket 实例，通过指定的远程服务器地址和端口来建立连接。

2.通过 Socket 实例包含的 InputStream 和 OutputStream 来进行数据的读写。

3.操作结束后调用 socket 实例的 close 方法，关闭。

2.2 UDP

服务器端（server）:

1. 构造 DatagramSocket 实例，指定本地端口。
2. 通过 DatagramSocket 实例的 receive 方法接收 DatagramPacket.DatagramPacket 中间就包含了通信的内容。
3. 通过 DatagramSocket 的 send 和 receive 方法来收和发 DatagramPacket.

客户端（client）:

1. 构造 DatagramSocket 实例。

2.通过 DatagramSocket 实例的 send 和 receive 方法发送 DatagramPacket 报文。

3.结束后，调用 DatagramSocket 的 close 方法关闭。

402. Java 反射技术主要实现类有哪些，作用分别是什么？

在 JDK 中，主要由以下类来实现 Java 反射机制，这些类都位于 java.lang.reflect 包中

1）Class 类：代表一个类

2）Field 类：代表类的成员变量(属性)

3）Method 类：代表类的成员方法

4）Constructor 类：代表类的构造方法

5）Array 类：提供了动态创建数组，以及访问数组的元素的静态方法

403. Class 类的作用？生成 Class 对象的方法有哪些？

Class 类是 Java 反射机制的起源和入口，用于获取与类相关的各种信息，提供了获取类信息的相关方法。Class 类继承自 Object 类

Class 类是所有类的共同的图纸。每个类有自己的对象，好比图纸和实物的关系；每个类也可看做是一个对象，有共同的图纸 Class，存放类的结构信息，能够通过相应方法取出相应信息：类的名字、属性、方法、构造方法、父类和接口

方 法	示 例
对 象	String str="bdqn";

名.getClass()	Class clazz = str.getClass();
对象名 .getSuperClass()	Student stu = new Student(); Class c1 = stu.getClass(); Class c2 = stu.getSuperClass();
Class.forName()	Class clazz = Class.forName("java.lang.Object"); Class.forName("oracle.jdbc.driver.OracleDriver");
类名.class	Class c1 = String.class; Class c2 = Student.class; Class c2 = int.class
包装类.TYPE	Class c1 = Integer.TYPE; Class c2 = Boolean.TYPE;

404. 反射的使用场合和作用、及其优缺点

1）使用场合：

在编译时根本无法知道该对象或类可能属于哪些类，程序只依靠运行时信息来发现该对象和类的真实信息。

2）主要作用：通过反射可以使程序代码访问装载到 JVM 中的类的内部信息，获取已装载类的属性信息，获取已装载类的方法，获取已装载类的构造方法信息

3）反射的优点

反射提高了 Java 程序的灵活性和扩展性，降低耦合性，提高自适应能力。它允许程序创建和控制任何类的对象，无需提前硬编码目标类；反射是其它一些常用语言，如 C、C++、Fortran 或者 Pascal 等都不具备的

4） Java 反射技术应用领域很广，如软件测试、 EJB、JavaBean 等；许多流行的开源框架例如 Struts、Hibernate、Spring 在实现过程中都采用了该技术

5）反射的缺点

性能问题：使用反射基本上是一种解释操作，用于字段和方法接入时要远慢于直接代码。因此 Java 反射机制主要应用在对灵活性和扩展性要求很高的系统框架上,普通程序不建议使用。

使用反射会模糊程序内部逻辑：程序人员希望在源代码中看到程序的逻辑，反射等绕过了源代码的技术，因而会带来维护问题。反射代码比相应的直接代码更复杂。

405. 设计模式入什么是设计模式，设计模式的作用。

设计模式是一套被反复使用的、多数人知晓、经过分类编目的优秀代码设计经验的总结。特定环境下特定问题的处理方法。

1）重用设计和代码　重用设计比重用代码更有意义，自动带来代码重用

2）提高扩展性　大量使用面向接口编程，预留扩展插槽，新的功能或特性很容易加入到系统中来

3）提高灵活性　通过组合提高灵活性，可允许代码修改平稳发生，

对一处修改不会波及到其他模块

4） 提高开发效率　正确使用设计模式，可以节省大量的时间

406. 面向对象设计原则有哪些

面向对象设计原则是面向对象设计的基石，面向对象设计质量的依据和保障，设计模式是面向对象设计原则的经典应用

1）单一职责原则 SRP

2）开闭原则 OCP

3）里氏替代原则 LSP

4）依赖注入原则 DIP

5）接口分离原则 ISP

6）迪米特原则 LOD

7）组合/聚合复用原则 CARP

8）开闭原则具有理想主义的色彩，它是面向对象设计的终极目标。其他设计原则都可以看作是开闭原则的实现手段或方法

407. 23 种经典设计模式都有哪些，如何分类。

面向对象设计原则	创建型模式 5+1	结构型模式 7	行为型模式 11
•单一职责原则 •开闭原则 •里氏替代原则 •依赖注入原则 •接口分离原则 •迪米特原则 •组合/聚合复用原则	•简单工厂模式 •工厂方法模式 •抽象工厂模式 •创建者模式 •原型模式 •单例模式	•外观模式 •适配器模式 •适配器模式 •代理模式 •装饰模式 •桥接模式 •组合模式 •享元模式	•模板方法模式 •观察者模式 •状态模式 •策略模式 •职责链模式 •命令模式 •访问者模式 •调停者模式 •备忘录模式 •迭代器模式 •解释器模式

表 设计模式空间

		目 的		
		创 建 型	结 构 型	行 为 型
范围	类	Factory Method	Adapter(类)	Interpreter Template Method
	对象	Abstract Factory Builder Prototype Singleton	Adapter (对象) Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

408. 写出简单工厂模式的示例代码

```
package com.bjsxt;


public class SimpleFactory {

    public static Product createProduct(String pname){

        Product product=null;

        if("p1".equals(pname)){

            product = new Product();

        }else if("p2".equals(pname)){

            product = new Product();

        }else if("pn".equals(pname)){

            product = new Product();

        }

        return product;
```

```
    }
}
```

基本原理：由一个工厂类根据传入的参数（一般是字符串参数），动态决定应该创建哪一个产品子类(这些产品子类继承自同一个父类或接口)的实例，并以父类形式返回

优点：客户端不负责对象的创建，而是由专门的工厂类完成；客户端只负责对象的调用，实现了创建和调用的分离，降低了客户端代码的难度；

缺点：如果增加和减少产品子类，需要修改简单工厂类，违背了开闭原则；如果产品子类过多，会导致工厂类非常的庞大，违反了高内聚原则，不利于后期维护

409. 写出单例模式的示例代码

```
package com.bjsxt;
/**
 * 饿汉式的单例模式
 * 在类加载的时候创建单例实例，而不是等到第一次请求实例的时候的时候创建
* 1、私有 的无参数构造方法Singleton()，避免外部创建实例
 * 2、私有静态属性instance
 * 3、公有静态方法getInstance()
 */
```

```
public class Singleton {
    private static Singleton instance = new Singleton();
    private Singleton(){    }
    public static Singleton getInstance(){
        return instance;
    }
}
```

```
package com.bjsxt;
/**
 * 懒汉式的单例模式
 *在类加载的时候不创建单例实例，只有在第一次请求实例的时候的时候创建
*/
public class Singleton {
    private static Singleton instance;
    private Singleton(){    }
    /**
     * 多线程情况的单例模式，避免创建多个对象
```

```
    */
    public static Singleton getInstance(){
        if(instance ==null){//避免每次加锁，只有第一次没有创建对象时
才加锁
            synchronized(Singleton.class){//加锁，只允许一个线程进入
                if(instance==null){ //只创建一次对象
                    instance = new Singleton();
                }
            }
        }
        return instance;
    }
}
```

410. 请对你所熟悉的一个设计模式进行介绍

分析：建议挑选有一定技术难度，并且在实际开发中应用较多的设计模式。

可以挑选装饰模式和动态代理模式。此处挑选动态代理设计模式。

讲解思路：生活案例引入、技术讲解、优缺点分析、典型应用。

1、生活案例引入：送生日蛋糕：

MM 们要过生日了，怎么也得表示下吧。最起码先送个蛋糕。蛋糕多种多样了。巧克力，冰淇淋，奶油等等。这都是基本的了，再加点额外的装饰，如

蛋糕里放点花、放贺卡、放点干果吃着更香等等。

分析：

方案 1:如果采用继承会造成大量的蛋糕子类

方案 2、蛋糕作为主体，花，贺卡，果仁等是装饰者，需要时加到蛋糕上。要啥我就加啥。

技术讲解

装饰模式（别名 Wrapper）是在不必改变原类文件和使用继承的情况下，动态的扩展一个对象的功能。它通过创建一个包装对象，也就是装饰来包裹真实对象，提供了比继承更具弹性的代替方案。

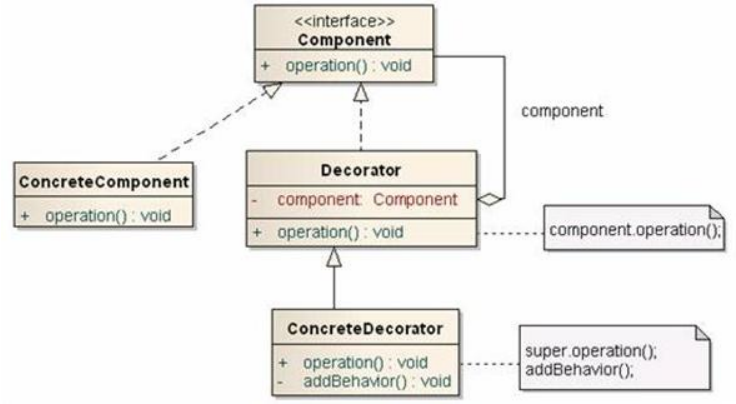
装饰模式一般涉及到的角色

抽象构建角色(Component):给出一个抽象的接口，以规范准备接受附加责任的对象。

具体的构建角色(ConcreteComponent) :定义一个将要接受附加责任的类。

抽象的装饰角色 (Decorator):持有一个抽象构建(Component)角色的引用，并定义一个与抽象构件一致的接口。

具体的装饰角色(ConcreteDecorator):负责给构建对象“贴上”附加的责任。



3、优缺点分析

优点

1）Decorator 模式与继承关系的目的都是要扩展对象的功能，但是 Decorato 更多的灵活性。

2）把类中的装饰功能从类中搬移出去，这样可以简化原有的类。有效地把类的核心功能和装饰功能区分开了。

3）通过使用不同的具体装饰类以及这些装饰类的排列组合，可创造出很多不同行为的组合。

缺点

这种比继承更加灵活机动的特性，也同时意味着更加多的复杂性。

装饰模式会导致设计中出现许多小类，如果过度使用，会使程序变得很复杂。

符合的设计原则：

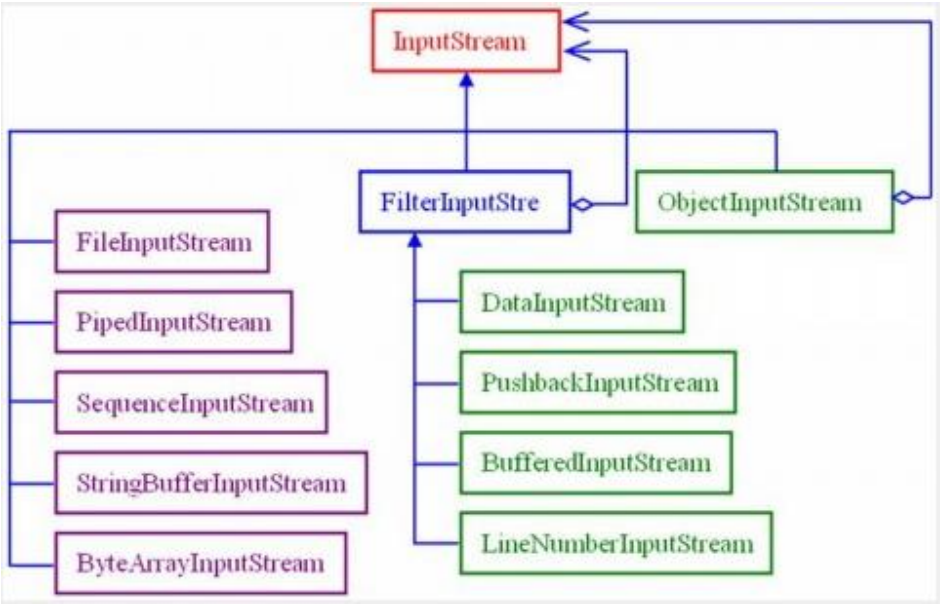
多用组合，少用继承。利用继承设计子类的行为是在编译时静态决定的，且所有的子类都会继承到相同的行为。如能够利用组合扩展对象的行为，就可在运行时动态进行扩展。

类应设计的对扩展开放，对修改关闭。

4、典型应用

java IO 中需要完成对不同输入输出源的操作，如果单纯的使用继承这一方式，无疑需要很多的类。比如说，我们操作文件需要一个类，实现文件的字节读取需要一个类，实现文件的字符读取又需要一个类....一次类推每个特定的操作都需要一个特定的类。这无疑会导致大量的 IO 继承类的出现。显然对于编程是很不利的。而是用装饰模式则可以很好的解决这一问题，

在装饰模式中：节点流（如 FileInputStream）直接与输入源交互，之后通过过滤流（FilterInputStream）进行装饰，这样获得的 io 对象便具有某几个的功能，很好的拓展了 IO 的功能。



411. 编写一个方法求一个字符串的字节长度?

```
public int getWordCount(String s)
  {
      int length = 0;
      for(int i = 0; i < s.length(); i++)
      {
          int ascii = Character.codePointAt(s, i);
          if(ascii >= 0 && ascii <=255)
              length++;
          else
              length += 2;
```

```
        }
        return length;

    }
```

412. 用 JavaScript 实现用正则表达式验证，某个字符串是合法的 6 位数字的邮编的函数

```
Function testE(ss){
   var reg=/^[1-9][0-9]{5}$/ ;
   if(req.test(ss)){
      alert(“邮编 OK”)
}else{
   alert(“邮编格式不正确”)；
   }
}
```

413. 请使用 JQuery 将页面上的所有元素边框设置为 2pix 宽的虚线？

```
$(“*”).css(“border”,” 2px dashed”)
```

414. 如何设定 JQuery 异步调用还是同步调用？

答案：

调用 jQuery 中的 ajax 函数，设置其 async 属性来表明是异步还是同步，如下：

```
$.ajax({
async:true//表示异步，false 表示同步
})
```

415. 说出 3 条以上 firefox 和 IE 的浏览器兼容问题？

答案：

兼容 firefox 的 outerHTML，FF 中没有 outerHtml 的方法

IE 下,可以使用()或[]获取集合类对象;Firefox 下,只能使用[]获取集合类对象.解决方法:统一使用[]获取集合类对象.

IE 下,可以使用获取常规属性的方法来获取自定义属性,也可以使用 getAttribute()获取自定义属性;Firefox 下,只能使用 getAttribute()获取自定义属性.解决方法:统一通过 getAttribute()获取自定义属性

416. 请用 Jquery 语言写出 ajax 请求或者 post 请求代码

```
$.post(“show”,{uname=”张三”，pwd=”123”},function(data){
  alert(data)
})
```

417. body 中的 onload()函数和 jQuery 中的 document.ready()有什么区别？

答案：

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。

418. jQuery 中有哪几种类型的选择器？

答案：

基本选择器

层次选择器

基本过滤选择器

内容过滤选择器

可见性过滤选择器

属性过滤选择器

子元素过滤选择器

表单选择器

表单过滤选择器

419. EasyUI 中 datagrid 刷新当前数据的方法？

答案：使用 reload()即可

420. 数据库 MySQL，Oracle，SqlServer 分页时用的语句

Mysql:使用 limit 关键字 Select * from 表名 where 条件 limit 开始位置，结束位置。通过动态的改变开始和结束位置的值来实现分页。
Oracle：通过 rownum 来实现 select * from (select rownum rn,t.* from addressbook where rownum<=

20) where rownum > 10
Sqlserver: select top 20 * from addressbook where id not in (select top 10 id from addressbook)

421. 分别写出一个 div 居中和其中的内容居中的 css 属性设置

Div 居中：

margin:auto 0px;

内容居中:

text-align:center;

422. 概述一下 session 与 cookie 的区别

答案：

存储角度：

Session 是服务器端的数据存储技术 ,cookie 是客户端的数据存储技术

解决问题角度：

Session 解决的是一个用户不同请求的数据共享问题，cookie 解决的是不同请求的请求数据的共享问题

生命周期角度：

Session 的 id 是依赖于 cookie 来进行存储的，浏览器关闭 id 就会失效

Cookie 可以单独的设置其在浏览器的存储时间。'

423. JavaScript 中 null 和 undefined 是否有区别？有哪些区别？

答案：

赋值角度说明：

null 表示此处没有值，undefined 表示此处定义了但是没有赋值

从数据转换角度：

Null 在做数值转换时会被转换为 0，undefined 会被转换为 NaN

424. Servlet 中的 doPost 和 doGet 方法有什么区别?它们在传递和获取参数上有什么区别？

答案：

区别：

doPost 用来处理 post 请求，doGet 用来处理 get 请求

获取参数：获取的参数是相同的都是 HttpServletRequest \HttpServletResponse

425. 请写出一段 jQuery 代码，实现把当前页面中所有的 a 元素中 class 属性为“view-link”的链接都改为在新窗口中打开

答案：

$(“a[class=view-link]”).attr(“target”,”_blank”)

426. 如下 JavaScript 代码的输出为：

```
var scope ="global scope";
    function checkscope() {
    var scope ="local scope”  ;
```

```
    return function() { return scope}
  }
  console.log (checkscope()());
```

```
global scope
```

427. 对于 Department 表的数据格式，请用 JQuery 提交下面的数据

```
Var options={
  url:’ /updateUrl’ ,
  type:’ post’ ,
  dataType:’ text’ ,
  data:_____,
  success:function(data){
    alert(“提交成功”) ;
    }
} ;
$.ajax(options);
```

1. 用 JQuery 提交一条科室的信息？
2. 用 JQuery 提交两条科室信息？

428. EasyUI 中 Datagrid 刷新当前数据的方法？

Datagrid 分页属性 pagination 起作用所需要的 json 数据中，必须的参数

是什么？

429. 什么是 Web 容器？常见 Web 容器有哪些？

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 直接跟容器中的环境变量交互，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

430. Jquery 中’.get()’与’.eq()’的区别

eq 返回的是一个 jquery 对象　get 返回的是一个 html 对象

431. 如何给 weblogic 定内存的大小？

在启动 Weblogic 的脚本中（位于所在 Domian 对应服务器目录下的 startServerName），增加 set MEM_ARGS=-Xms32m -Xmx200m，可以调整最小内存为 32M，最大 200M

432. TCP 为何采用三次握手来建立连接，若采用二次握手可以吗，请说明理由？

三次握手是为了防止已失效的连接请求再次传送到服务器端。 二次握手不可行，因为：如果由于网络不稳定，虽然客户端以前发送的连接请求以到达服务方，但服务方的同意连接的应答未能到达客户端。则客户方要重新发送连接请求，若采用二次握手，服务方收到重传的请求连接后，会以为是新的请求，就会发送同意连接报文，并新开进程提供服务，这样会造成服务方资源的无谓浪费

433. 以下 HTTP 相应状态码的含义描述正确的是（D）

A.	200ok表示请求成功
B.	400 不良请求表示服务器未发现与请求 URL 匹配内容
C.	404未发现表示由于语法错误儿导致服务器无法理解请求信息
D.	500 内部服务器错误，无法处理请求

434. JSP 页面包括哪些元素？（C）

A.	JSP命令
B.	JSP Action
C.	JSP脚本
D.	JSP 控件

435. Ajax 有四种技术组成：DOM,CSS,JavaScript，XmlHttpRequest，其中控制文档结构的是（A）

A.	DOM
B.	CSS
C.	JavaScript
D.	XmlHttpRequest

436. 下面关于 session 的用法哪些是错误的？（A）

A.	HttpSession session=new HttpSession();
B.	String haha=session getPaeameler(:haha”);
C.	session.removeAttribute(“haha”);

D.	session.setAttribute(:haha:);XmlHttpRequest

437. Jsp 九大内置对象

答案：

pageContext

request

session

application

config

page

exception

438. 如何配置一个 servlet?

在 web.xml 中使用如下标签:

```
<servlet>
 <servlet-name></servlet-name>
 <servlet-class></servlet-class>
</servlet>
<servlet-mapping>
 <servlet-name></servlet-name>
 <url-pattern></url-pattern>
</servlet-mapping>
```

439. JAVASCRIPT 部分

如何定义含有数值 1 至 8 的数组？

答案　var arr=[1,2,3,4,5,6,7,8]

2. 请写函数计算上题数组元素的合计计值

3. 如何在数组的头部和尾部添加一个元素？

答案：使用 shift 和 unshift 方法

4. 如何将数组组合成字符串，分隔符用逗号？

答案:使用 concat 函数进行数组合并，使用 join 方法进行间隔符的改变

440. 以下 javascipt 语句会产生运行错误的是_A_____

A.	var obj=( );
B.	var obj=[ ];
C.	var obj=//;
D.	var obj=1;

441. 在 JSP 中，下面__B__块中可以定义一个新类：

A.	<%　%>
B.	<%！%>
C.	<%@ %>
D.	<%=%>

442. 求 y 和 z 的值是多少？（JavaScript）

```
<script　type="text / javascript">

    var x=1;
```

```
var y=0;
var z=0;
Function add(n){n=n+1;}
        y=add(x);
function add(n){n=n+3;}
z=add(x);
</script>
```

443. 解释下面关于 J2EE 的名词：

JNDI、JDBC、JMS、JTA、EJB、RMI

444. 写一段 js，遍历所有的 li，将每个 li 的内容逐个 alert 出来

```
<body>
    <ul>
      <li>张三：123</li>
      <li>李四：456</li>
      <li>王五：789</li>
      <li>赵六：147</li>
  <ul>
 <body>
```

445. HTML 含义和版本变化

HTML 含义：

Hyper Text Markup Language 超文本标记语言，是一种用来制作“网页”的简单标记语言 ;用 HTML 编写的超文本文档称为 HTML 文档 ,HTML 文档的扩展名是 html 或者 htm

版本变化：

HTML1.0——在 1993 年 6 月作为 IETF 工作草案发布（并非标准）

HTML 2.0——1995 年 11 月作为 RFC 1866 发布

HTML 3.2——1997 年 1 月 14 日，W3C 推荐标准

HTML 4.0——1997 年 12 月 18 日，W3C 推荐标准

HTML 4.01（微小改进）——1999 年 12 月 24 日，W3C 推荐标准

HTML 5—2014 年 10 月 28 日，W3C 推荐标准 HTML 文档结构

446. 什么是锚链接

锚链接是带有文本的超链接。可以跳转到页面的某个位置，适用于页面内容较多，超过一屏的场合 。分为页面内的锚链接和页面间的锚链接 。

例如：

```
<a name=”1F”>1F</a><a name=”2F”>2F</a>
<a href=”#2F”>跳转到 2F 标记位置</a>
```

说明：

1.在标记位置利用 a 标签的 name 属性设置标记。

2.在导航位置通过 a 标签的 href 属性用#开头加 name 属性值即可跳转锚点位置。

447. HTML 字符实体的作用及其常用字符实体

有些字符，比如说“<”字符，在 HTML 中有特殊的含义，因此不能在文本

中使用。想要在 HTML 中显示一个小于号“<”，需要用到字符实体：<或者<

字符实体拥有三个部分：一个 and 符号（&），一个实体名或者一个实体号，最后是一个分号（;）

常用字符实体：

显示结果	描述	实体名	实体号
	空格		
<	小于	<	<
>	大于	>	>
&	and 符号	&	&
'	单引号	' (IE 不支持)	'
"	引号	"	"
£	英镑	£	&#

			163;
¥	人民币元	¥	¥
§	章节	§	§
©	版权	©	©

448. HTML 表单的作用和常用表单项类型

表单的作用：

利用表单可以收集客户端提交的有关信息。

常用表单项类型：

input 标签 type 属性	功能	input 标签 type 属性	功能
text	单行本框	reset	重置按钮
password	密码框	submit	提交按钮
radio	单选按钮	textarea	文本域
checkbox	复选框	select	下拉框
button	普通按	hidden	隐藏域

	钮		

449. 表格、框架、div 三种 HTML 布局方式的特点

	优点	缺点	应用场合
表格	方便排列有规律、结构均匀的内容或数据	产生垃圾代码、影响页面下载时间、灵活性不大难于修改	内容或数据整齐的页面
框架	支持滚动条、方便导航 节省页面下载时间等	兼容性不好 ,保存时不方便、应用范围有限	小型商业网站、论坛后台管理
Div	代码精简、提高页面下载速度、表现和内容分离	比较灵活、难于控制	复杂的不规则页面、业务种类较多的大型商业网站

450. form 中 input 设置为 readonly 和 disabled 的区别

	readonly	disabled
有效对象	. 只针对 type 为 text/password 有效	对所有表单元素有效
表单提交	当表单元素设置 readonly 后，表单提交能将该表单元素的值传递出去。	当表单元素设置 disabled 后 ,表单提交不能将该表单元素的值传递出去。

451. CSS 的定义和作用

CSS 的定义：CSS 是 Cascading Style Sheets(层叠样式表)的简称。

CSS 是一系列格式规则，它们控制网页内容的外观。CSS 简单来说就是用来

美化网页用的。

CSS 的具体作用包括：

使网页丰富多彩，易于控制。

页面的精确控制，实现精美、复杂页面 。

样式表能实现内容与样式的分离，方便团队开发。

样式复用、方便网站的后期维护。

452. CSS2 常用选择器类型及其含义

选择器名称	案例	语法格式
标签选择器	h3{font-size:24px;font-family:"隶书 “; } JSP	元素标签名{样式属性}
类选择器	.red {color:#F00;} <li class="red">Oracle	. 元素标签 class 属性值 {样式属性}
ID 选择器	#p1 {background-color:#0F0;} content	#元素标签 id 属性值{样式属性}

包含选择器	div h3{color:red;}
	
	 CSS 层叠样式表
		父元素标签 子元素标签{
		样式属性
		}

子选择器	div>ul{color:blue;}
	
	 - 测试 1
	
	 - 嵌 套 元 素
	
	 - 嵌套元素
	 - 嵌套元素
	 - 嵌套元素
	
	 - 测试 1
	 - 测试 1
	
		父元素标签名>子元素名{
		样式属性
		}

453. 引入样式的三种方式及其优先级别

三种引用方式：

1. 外部样式表（存放.css 文件中）

不需要 style 标签

<link rel=”stylesheet” href=”引用文件地址” />

2. 嵌入式样式表

<style type=“text/css”>

p{color:red;}

</style>

3.内联样式

标签属性名为 style

优先级级别：在一个页面上，自上而下，后定义优先级高。

454. 盒子模型

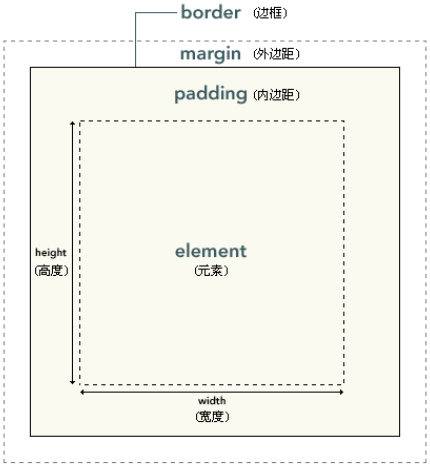
盒子模型类似于生活中的盒子，具有 4 个属性，外边距，内边距，边框，内容。

外边距：margin，用于设置元素和其他元素之间的距离。

内边距：padding,用于设置元素内容和边框之间的距离。

边框：border,用于设置元素边框粗细，颜色，线型。

内容：width,height,用于设置元素内容显示的大小。



例如：

```
<style>
        body{
            margin: 0;   /*取消body默认的外边距*/
        }
        #img1{
            width:200px;     /*设置图片的宽度*/
            border: 2px solid black; /*设置图片边框*/
            margin: 5px;
/*设置图片外边距（表示该图片与其他图片的距离为5px）*/
            padding:10px; /*设置图片与边框之间的距离*/
        }
        #img2{
            height: 200px; /*  设置图片的高度*/
            border: 2px solid black; /*设置图片的边框*/
            margin: 5px; /*设置图片外边距*/
```

```
            padding: 20px; /*设置图片与边框之间的距离*/
        }
</style>
    <img id="img1" src="img/2.jpg" />
        <img id="img2" src="img/lss.jpg" />
```

455. JavaScript 语言及其特点

Javascript 一种基于对象(object-based)和事件驱动(Event Driven)的简单的并具有安全性能的脚本语言。特点：

解释性： JavaScript 不同于一些编译性的程序语言，例如 C、C++等，它是一种解释性的程序语言，它的源代码不需要经过编译，而直接在浏览器中运行时被解释。

基于对象： JavaScript 是一种基于对象的语言。这意味着它能运用自己已经创建的对象。因此，许多功能可以来自于脚本环境中对象的方法与脚本的相互作用。

事件驱动 :JavaScript 可以直接对用户或客户输入做出响应，无须经过 Web 服务程序。它对用户的响应，是以事件驱动的方式进行的。所谓事件驱动，就是指在主页中执行了某种操作所产生的动作，此动作称为“事件”。比如按下鼠标、移动窗口、选择菜单等都可以视为事件。当事件发生后，可能会引起相应的事件响应。

跨平台:JavaScript 依赖于浏览器本身，与操作环境无关，只要能运行浏览器的计算机，并支持 JavaScript 的浏览器就可正确执行。

456. JavaScript 常用数据类型有哪些

数值型：整数和浮点数统称为数值。例如 85 或 3.1415926 等。

字符串型：由 0 个,1 个或多个字符组成的序列。在 JavaScript 中，用双引号或单引号括起来表示，如“您好”、‘学习 JavaScript’等。 不区分单引号、双引号。

逻辑（布尔）型：用 true 或 false 来表示。

空（null）值：表示没有值，用于定义空的或不存在的引用。

要注意，空值不等同于空字符串""或 0。

未定义（undefined）值：它也是一个保留字。表示变量虽然已经声明，但却没有赋值。

除了以上五种基本的数据类型之外，JavaScript 还支持复合数据类型，包括对象和数组两种。

457. html 语法中哪条命令用于使一行文本折行，而不是插入一个新的段落？ (B)

A.	<TD>
B.	
C.	<P>
D.	<H1>

458. Ajax 的优点和缺点

优点：减轻服务器的负担,按需取数据,最大程度的减少冗余请求

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

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：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.

459. 怎样防止表单刷新重复提交问题？（说出思路即可）

JS 脚本方式:

第一种：定义全局变量，在 form 提交前判断是否已有提交过

```
<script>
    var checkSubmitFlg = false;
    function checkSubmit(){
        if(checkSubmitFlg == true){
   return false;
        }
        checkSubmitFlg = true;
        return true;
   }
  </script>
<form action="" onsubmit="return checkSubmit();">
</form>
```

第二种：单击提交按钮后，立刻禁用改按钮

第三种：单击提交按钮后，弹出屏蔽层，防止用户第二次点击

460. 求 y 和 z 的值是多少？（JavaScript）(undefined, undefined)

```
<script   type="text / javascript">
    var x=1;
    var y=0;
    var z=0;
    function add(n){
          n=n+1;
      }
       y=add(x);
    function add(n){
          n=n+3;
    }
    z=add(x);
</script>
```

461. JavaScript window.onload 事件和 JQuery ready 函数有何不同？

window.onload 是在页面所需资源加载完成触发，包括图片等资源

jquery ready 是在 Domcontentloaded 下触发，如果浏览器不支持会退化

到 onload

区别在于，DomContentLoaded 是在 dom 解析完成下触发，它不要求图片已经下载完成

所以整体来说它要比 onload 考前一些，同时又是在 dom 节点可用的情况下触发

462. JQuery.get()和 JQuery.ajax()方法之间的区别是什么？

JQuery.ajax()是对原生的 javaScript 的 ajax 的封装，简化了 ajax 的步骤，用户可用 JQuery.ajax()发送 get 或者 post 方式请求,Jquery.get()是对 ajax 的 get 方式的封装，只能发送 get 方式的请求。

463. Jquery 里的缓存问题如何解决？例如($.ajax()以及$.get())

$.ajax()请求时候加上 cache:false 的参数，如：

```
$.ajax({
        type : "get",
        url : "XX",
        dataType : "json",
        cache:false,
        success : function(json) {
        }
});
```

$.get()请求时候加上时间，如:

$.get("url","data"+new Date(),function(data){

);

464. form 中的 input 可以设置为 readonly 和 disable，请问 2 者有什么区别？

readonly:设置 input 表单域只能读，不能编辑，但是可以提交给服务器。

disable：设置 input 表单域只读，不能编辑，也不能提交。

465. Javascript 是面向对象的，怎么体现 Javascript 的继承关系？

Javascript 里面没有像 java 那样的继承，javascript 中的继承机制仅仅是靠模拟的，可以使用 prototype 原型来实现

466. Javascript 的有几种种变量。变量范围有什么不同？

可以分为三种

1:原生类型（string,number,boolean）

2:对象（Date,Array）

3:特殊类型（var vara;(只什么没有定义),var varb = null;(定义一个变量并赋值为 null)）

467. Js 如何获取页面的 dom 对象

1:直接获取

```
//1.1 -- id 方式获取
var varid = document.getElementById("unameid");
//1.2 -- name 获取(获取的是数组对象)
var varname = document.getElementsByName("sex");
```

```
//1.3 -- 元素获取(获取的是数组对象)
var varinput = document.getElementsByTagName("input");
```

2:间接方式获取

```
//2.1 父子关系 --childNodes
var varchilds = document.getElementById("div01").childNodes;
//2.2 子父关系--parentNode
var varfather2 = document.getElementById("unameid").parentNode;
```

//2.3 兄弟之间相互获取 nextSibling：下一个节点 previousSibling：上一个节点

468. SERVLET API 中 forward() 与 redirect()的区别？

```
<h2>请求转发和请求重定向 :服务器内部资源跳转的两种方式 </h2>
<h4> 请 求 转 发 :request.getRequestDispatcher( “ url").forward(request, response)</h4>
<ol>
    <li>请求转发只发生了一次请求，url 地址栏不会发生改变</li>
    <li>可以携带请求参数信息</li>
    <li>请求转发之后不能再次请求转发，可以在后面加 return 来避免 500 错误</li>
    <li>效率相对较高</li>
```

当访问资源在服务器内部优先选用请求转发

<h4>请求重定:response.sendRedirect(URL)</h4>

请求转发只发生了两次次请求，url 地址栏会发生改变

不可以携带请求参数信息,但可以把请求参数信息写 url 地址后面（不建议使用）

请求重定向后不能 再次请求重定向，可以在后面加 return 来避免 500 错误

当访问的资源在请求的服务器外部时，只能用重定向

469. 列举 10 个以上 HTML 元素，列举 5 个网页事件

h,hr,br,font,p,table,div,span onclick,ondblclick,onmousemove,onmouseenter,onmouseout,onkeyup, onfocus,onblur

470. Session 域和 request 域什么区别？

作用域：存放数据，获取数据（传递数据）

有效的作用域：生命周期，作用范围

httpServeltRequest:

生命周期 ：一次请求之间

作用范围：所有被请求转发过的 servlet 都能获取到

httpSession:

生命周期：一次会话

作用范围：所有的 servlet 都可以获取到

servletContex:

生命周期：从项目开始运行到服务器关闭

作用范围：所有的 servlet 都可以获取到

作用域如何选用？

httpServeltRequest：和当前请求有关的信息

httpSession：和当前用户有关的信息

servletContex：访问量比较大，不易更改

471. 解释下面关于 J2EE 的名词：

JNDI、JDBC、JMS、JTA、EJB、RMI

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。

472. JAVA SERVLET API 中 forward()与 redirect()的区别？

<h2>请求转发和请求重定向：服务器内部资源跳转的两种方式 </h2>

<h4> 请 求 转 发 :request.getRequestDispatcher(“ url").forward(request, response)</h4>

请求转发只发生了一次请求，url 地址栏不会发生改变

可以携带请求参数信息

请求转发之后不能再次请求转发，可以在后面加 return 来避免 500 错误

效率相对较高

当访问资源在服务器内部优先选用请求转发

<h4>请求重定:response.sendRedirect(URL)</h4>

请求转发只发生了两次次请求，url 地址栏会发生改变

不可以携带请求参数信息,但可以把请求参数信息写 url 地址后面（不建议使用）

请求重定向后不能 再次请求重定向，可以在后面加 return 来避免 500 错误

当访问的资源在请求的服务器外部时，只能用重定向

473. 页面中有一个命名为 bankNo 的下拉列表，写 js 脚本获取当前选项的索引值，如果用 jquery 如何获取

```
var a = document.getElementsByName("bankNo")[0].value;

var b = $("select[name=bankNo]").val();
```

474. 写出要求 11 位数字的正则表达式

```
^\d{11}$
```

475. 分别获取指定 name、Id 的 javascript 对象，如果用 jquey 如何获取

```
js:
   id--document.getElementById("id");
   name--document.getElementsByName("name");
jquery
   id--$("#id");
   name--$("元素名称[name="name 值"]" )；
```

476. 一个页面有两个 form，如何获取第一个 form

用 id 方式获取；document.getElementById("id");

477. 如何设置一个层的可见/隐藏

可见: document.getElementById("divid").style.display = "block";

隐藏: document.getElementById("divid").style.display = "none";

478. 描述 JSP 中动态 INCLUDE 与静态 INCLUDE 的区别？

动态导入

1：会将多个 jsp 页面分别再编写成 java 文件，编译成 class 文件

2：jsp 文件中允许有相同的变量名，每个页面互不影响

3: 当 java 代码比较多优先选用动态导入

4：效率相对较低，耦合性低

静态导入

1：会将多个 jsp 页面合成一个 jsp 页面，再编写成 java 文件，编译成 class 文件

2：jsp 文件中不允许有相同的变量名

3: 当 java 代码比较少或者没有 java 代码是优先选用静态导入

4：效率相对较高，耦合性高

479. 列举 JSP 的内置对象及方法

<h4>四大作用域</h4>

<h5>request--对应 servlet 中的 htttpServletRequest</h5>

<h5>session--对应 servlet 中的 httpSession</h5>

<h5>application--对应 servlet 中的 servletContext</h5>

<h5>pageContext 页面上下文--可以用 pageContext 对象来获取其它的八个对象</h5>

<h4>两个输入</h4>

<h5>response</h5>

<h5>out--缓冲流（不需要自己手写）</h5>

<h4>三个打酱油</h4>

<h5>config--对应 servlet 中的 servleConfig</h5>

<h5>page</h5>

<h5>exception--需要配置编译器指令才会显示出来 <%@ page isErrorPage="true" %></h5>

480. 列举 jsp 的四种范围

page、request、session、application

481. html 和 xhtml 的区别是什么？

HTML 与 XHTML 之间的差别，粗略可以分为两大类比较：一个是功能上的差别，另外是书写习惯的差别。关于功能上的差别，主要是 XHTML 可兼容各大浏览器、手机以及 PDA，并且浏览器也能快速正确地编译网页。

因为 XHTML 的语法较为严谨，所以如果你是习惯松散结构的 HTML 编写者，那需要特别注意 XHTML 的规则。但也不必太过担心，因为 XHTML 的规则并不太难。下面列出了几条容易犯的错误，供大家引用。

1:所有标签都必须小写

在 XHTML 中，所有的标签都必须小写，不能大小写穿插其中，也不能全部都是大写。看一个例子。

错误：<Head></Head><Body></Body>

正确：<head></head><body></body>

2:标签必须成双成对

像是 ... 、<a>...、 ... 标签等，当出现一个标签时，必须要有对应的结束标签，缺一不可，就像在任何程序语言中的括号一样。

错误：大家好 我是 muki

正确： 大家好 我是 muki

3:标签顺序必须正确

标签由外到内，一层层包覆着，所以假设你先写 div 后写 h1，结尾就要先

写 h1 后写 div。只要记住一个原则“先进后出”，先弹出的标签要后结尾。

错误： <h1>大家好 </h1>

正确： <h1>大家好</h1>

4:所有属性都必须使用双引号

在 XHTML 1.0 中规定连单引号也不能使用，所以全程都得用双引号。

错误： hello

正确： hello</div

5:不允许使用 target="_blank"

从 XHTML 1.1 开始全面禁止 target 属性，如果想要有开新窗口的功能，就必须改写为 rel="external"，并搭配 JavaScript 实现此效果。

错误：MUKI space

正确：MUKI space

482. 描述一下 css 的盒子模型

网页设计中常听的属性名：内容(content)、填充(padding)、边框(border)、边界(margin)， CSS 盒子模式都具备这些属性。

这些属性我们可以用日常生活中的常见事物——盒子作一个比喻来理解，所以叫它盒子模式。

CSS 盒子模型就是在网页设计中经常用到的 CSS 技术所使用的一种思维模型

483. 你做的页面用哪些浏览器测试过？这些测试的内核分别是什么？

1、Trident 内核代表产品 Internet Explorer，又称其为 IE 内核。

Trident（又称为 MSHTML），是微软开发的一种排版引擎。使用 Trident 渲染引擎的浏览器包括：IE、傲游、世界之窗浏览器、Avant、腾讯 TT、Netscape 8、NetCaptor、Sleipnir、GOSURF、GreenBrowser 和 KKman 等。

2、Gecko 内核代表作品 Mozilla

FirefoxGecko 是一套开放源代码的、以 C++编写的网页排版引擎。Gecko 是最流行的排版引擎之一，仅次于 Trident。使用它的最著名浏览器有 Firefox、Netscape6 至 9。

3、WebKit 内核代表作品 Safari、Chromewebkit

是一个开源项目，包含了来自 KDE 项目和苹果公司的一些组件，主要用于 Mac OS 系统，它的特点在于源码结构清晰、渲染速度极快。缺点是对网页代码的兼容性不高，导致一些编写不标准的网页无法正常显示。主要代表作品有 Safari 和 Google 的浏览器 Chrome。

4、Presto 内核代表作品 OperaPresto

是由 Opera Software 开发的浏览器排版引擎，供 Opera 7.0 及以上使用。它取代了旧版 Opera 4 至 6 版本使用的 Elektra 排版引擎，包括加入动态功能，例如网页或其部分可随着 DOM 及 Script 语法的事件而重新排版。

484. 你遇到了哪些浏览器的兼容性问题？怎么解决的？

1:针对不同的浏览器写不同的代码

2:使用 jquery 屏蔽浏览器差异

485. 你知道的常用的 js 库有哪些？

1.moment.js

举个例子：

用 js 转换时间戳为日期

```
let date = new Date(1437925575663);
        let year = date.getFullYear() + '-';
        let month = ( date.getMonth() + 1 < 10 ? '0' +
(date.getMonth() + 1) :
date.getMonth() + 1 ) + '-';
        let day = date.getDate();
...
        return year + month + day;
```

用 moment.js

```
return moment(1437925575663).format('YYYY-MM-DD HH:mm:ss')
```

2.chart.js

绘制简单的柱状图，曲线图，蛛网图，环形图，饼图等完全够用，用法比较简单。

3.D3.js

功能太强大了，看首页就知道了，感觉没有什么图 d3 绘不出来的。

4.Rx.js

很好的解决了异步和事件组合的问题。

5.lodash.js

486. Js 中的三种弹出式消息提醒（警告窗口、确认窗口、信息输入窗口）的命令是什么？

alter(),confirm(),prompt()

487. 谈谈 js 的闭包

闭包无处不在，比如：jQuery、zepto 的核心代码都包含在一个大的闭包中，所以下面我先写一个最简单最原始的闭包，以便让你在大脑里产生闭包的画面：

function A(){ function B(){ console.log("Hello Closure!"); } return B; } var C = A(); C();//Hello Closure!

这是最简单的闭包。

有了初步认识后，我们简单分析一下它和普通函数有什么不同，上面代码翻译成自然语言如下：

（1）定义普通函数 A

（2）在 A 中定义普通函数 B

（3）在 A 中返回 B

（4）执行 A，并把 A 的返回结果赋值给变量 C

（5）执行 C

把这 5 步操作总结成一句话就是：

函数 A 的内部函数 B 被函数 A 外的一个变量 c 引用。

把这句话再加工一下就变成了闭包的定义：

当一个内部函数被其外部函数之外的变量引用时，就形成了一个闭包。

因此，当你执行上述 5 步操作时，就已经定义了一个闭包！

这就是闭包。

488. 写一段 js，遍历所有的 li，将每个 li 的内容逐个 alert 出来

```
<body>
        <ul>
          <li>张三：123</li>
          <li>李四：456</li>
          <li>王五：789</li>
          <li>赵六：147</li>
     <ul>
    <body>
```

```
function test(){
   var varli = document.getElementsByTagName("li");
   for (var i=0;i<varli.length;i++) {
      alert(varli[i].innerText);
   }
}
```

489. 页面上如何用 JavaScript 对多个 checkbox 全选

```
//全选
function checkAll(){
```

```
    //获取复选框对象--数组对象
    var varcheck = document.getElementsByName("name");
    //alert(varcheck.length);
    //遍历 for
    for(var i=0;i<varcheck.length;i++){
        varcheck[i].checked = true;
    }
}
```

490. 写一个简单的 JQuery 的 ajax

```
    <script type="text/javascript" src="js/jquery-1.9.1.js"
charset="utf-8"></script>

    <script type="text/javascript">
        function testJqAjax(){
            //url   :请求地址
            //type :请求的方式  get/post
            //data :请求的参数(json/String)
            //cache:true(走缓存 )  false(不走缓存)
            //result:当 ajax 发送成功后会调用 success 后面的函数，result :相
当于形参，返回的数据
            //async:是否为异步请求 默认 true 异步，false 同步
```

```
            $.ajax({
                url:"TestJqAjax",
                type:"get",
                /* data:"uname=zhangsan&realname=张三丰", */
                data:{uname:"zhangsan",realname:"张三丰"},
                cache:false,
                async:false,
                success:function(result){
                    alert(result);
                }
            });
        }

        //ajax 的 get 方式的请求
        function jqAjaxGet(){
            //url,[data],[callback](当 ajax 发送成功后调用的函数)
            $.get("TestJqAjax",{uname:"zhangsan",realname:"  张  三  丰
"},function(result){
                alert(result);
            });

        }
```

```
        function    jqAjaxPost() {
            //url,[data],[callback](当 ajax 发送成功后调用的函数)
            $.post("TestJqAjax",{uname:"zhangsan",realname:"  张  三  丰
"},function(result){
                alert(result);
            });
        }
    </script>
```

491. Js 截取字符串 abcdefg 的 efg

```
function test2(){
    var str = "abcdefg";
    var substr = str.substring(4);
    alert(substr);
}
```

492. Windows.load 和$（document）.read（ ）区别？

window.onload 是在页面所需资源加载完成触发，包括图片等资源

jquery ready 是在 Domcontentloaded 下触发，如果浏览器不支持会退化到 onload

区别在于，DomContentLoaded 是在 dom 解析完成下触发，它不要求图片已经下载完成

所以整体来说它要比 onload 考前一些，同时又是在 dom 节点可用的情况下触发

493. http 的请求头信息可包含?

请求行（请求方式，资源路径，协议和协议版本号）

若干请求头

请求实体内容

494. http 的响应码 200，404，302，500 表示的含义分别是？

200 - 确定。客户端请求已成功

302 - 临时移动转移，请求的内容已临时移动新的位置

404 - 未找到文件或目录

500 - 服务器内部错误

495. Servlet 中 request 对象的方法有？

```
    //获取网络信息
    private void getNet(HttpServletRequest req, HttpServletResponse resp) {
        System.out.println("TestHttpRequest.getNet(获取客户端的ip):"+req.getRemoteAddr());
        System.out.println("TestHttpRequest.getNet(获取客户端的端口):"+req.getRemotePort());
        System.out.println("TestHttpRequest.getNet(获取服务器的ip):"+req.getLocalAddr());
```

```
        System.out.println("TestHttpRequest.getNet(获取服务器的端口):"+req.getLocalPort());


    }


    //获取实体内容
    private void getContent(HttpServletRequest req, HttpServletResponse resp) {
        //获取单条信息
        String uname = req.getParameter("uname");
        //获取多条信息，数组格式
        String[] favs = req.getParameterValues("fav");
        //遍历数组
        //判断
        if(favs!=null&&favs.length>0){
            for (int i = 0; i < favs.length; i++) {

    System.out.println("TestHttpRequest.getContent(fav):"+favs[i]);
            }
        }

        String un = req.getParameter("un");
```

```
    System.out.println("TestHttpRequest.getContent():"+uname+"--"+favs+"--"+un);

    }

    //获取请求头信息
    private void getHeads(HttpServletRequest req, HttpServletResponse resp) {
        //获取单条头信息
        //System.out.println("TestHttpRequest.getHeads(获取请求头信息-浏览器头信息)："+req.getHeader("User-Agent"));
        //获取所有头信息--返回枚举类型
        Enumeration strHeads = req.getHeaderNames();
        //遍历枚举类型
        while (strHeads.hasMoreElements()) {
            String strhead = (String) strHeads.nextElement();
            System.out.println("TestHttpRequest.getHeads(获取头信息):"+req.getHeader(strhead));
        }

    }
```

```
    //获取请求行的信息
    private void getLines(HttpServletRequest req, HttpServletResponse
resp) {
        System.out.println("TestHttpRequest.getLines(请求方式
***):"+req.getMethod());
        System.out.println("TestHttpRequest.getLines(资源路
径):"+req.getRequestURI());
        System.out.println("TestHttpRequest.getLines(地
址):"+req.getRequestURL());
        System.out.println("TestHttpRequest.getLines(协
议):"+req.getScheme());
        System.out.println("TestHttpRequest.getLines(协议的版本
号):"+req.getProtocol());
        System.out.println("TestHttpRequest.getLines(获取参数信
息):"+req.getQueryString());
        System.out.println("TestHttpRequest.getLines(项目名称
***):"+req.getContextPath());
    }
```

496. HTML 含义和版本变化

HTML 含义：

Hyper Text Markup Language 超文本标记语言，是一种用来制作“网页”

的简单标记语言；用 HTML 编写的超文本文档称为 HTML 文档，HTML 文档的扩展名是 html 或者 htm

版本变化：

HTML1.0——在 1993 年 6 月作为 IETF 工作草案发布（并非标准）

HTML 2.0——1995 年 11 月作为 RFC 1866 发布

HTML 3.2——1997 年 1 月 14 日，W3C 推荐标准

HTML 4.0——1997 年 12 月 18 日，W3C 推荐标准

HTML 4.01（微小改进）——1999 年 12 月 24 日，W3C 推荐标准

HTML 5—2014 年 10 月 28 日，W3C 推荐标准 HTML 文档结构

497. 什么是锚链接

锚链接是带有文本的超链接。可以跳转到页面的某个位置，适用于页面内容较多，超过一屏的场合 。分为页面内的锚链接和页面间的锚链接 。

例如：

```
<a name=”1F”>1F</a><a name=”2F”>2F</a>
<a href=”#2F”>跳转到 2F 标记位置</a>
```

说明：

1.在标记位置利用 a 标签的 name 属性设置标记。

2.在导航位置通过 a 标签的 href 属性用#开头加 name 属性值即可跳转锚点位置。

498. HTML 字符实体的作用及其常用字符实体

有些字符，比如说“<”字符，在 HTML 中有特殊的含义，因此不能在文本中使用。想要在 HTML 中显示一个小于号“<”，需要用到字符实体：<或者

<

字符实体拥有三个部分：一个 and 符号（&），一个实体名或者一个实体号，最后是一个分号（;）

常用字符实体：

显 示结果	描述	实体名	实体号
	空格		
<	小于	<	<
>	大于	>	>
&	and 符号	&	&
'	单引号	' (IE 不支持)	'
"	引号	"	"
£	英镑	£	£
¥	人民币元	¥	¥
§	章节	§	§
©	版权	©	©

499. HTML 表单的作用和常用表单项类型

表单的作用：

利用表单可以收集客户端提交的有关信息。

常用表单项类型：

input 标签 type 属性	功能	input 标签 type 属性	功能
text	单行本框	reset	重置按钮
password	密码框	submit	提交按钮
radio	单选按钮	textarea	文本域
checkbox	复选框	select	下拉框
button	普通按钮	hidden	隐藏域

500. 表格、框架、div 三种 HTML 布局方式的特点

	优点	缺点	应用场合
表格	方便排列有规律、结构均匀的内容或数据	产生垃圾代码、影响页面下载时间、灵活性不大难于修改	内容或数据整齐的页面
框架	支持滚动条、方便导航 节省页面下载时间等	兼容性不好 ,保存时不方便、应用范围有限	小型商业网站、论坛后台管理
Div	代码精简、提高页面	比较灵活、难于控制	复杂的不规则页面、

	下载速度、表现和内容分离		业务种类较多的大型商业网站

501. form 中 input 设置为 readonly 和 disabled 的区别

	readonly	disabled
有效对象	. 只针对 type 为 text/password 有效	对所有表单元素有效
表单提交	当表单元素设置 readonly 后，表单提交能将该表单元素的值传递出去。	当表单元素设置 disabled 后 ,表单提交不能将该表单元素的值传递出去。

502. CSS 的定义和作用

CSS 的定义：CSS 是 Cascading Style Sheets(层叠样式表)的简称。

CSS 是一系列格式规则，它们控制网页内容的外观。CSS 简单来说就是用来美化网页用的。

CSS 的具体作用包括：

使网页丰富多彩，易于控制。

页面的精确控制，实现精美、复杂页面 。

样式表能实现内容与样式的分离，方便团队开发。

样式复用、方便网站的后期维护。

503. CSS2 常用选择器类型及其含义

选择器名称	案例	语法格式
标签选择器	h3{font-size:24px;font-family:"隶书 “; }	元素标签名{样式属性}

	JSP	
类选择器	.red {color:#F00;} <li class="red">Oracle	. 元素标签 class 属性值 {样式属性}
ID 选择器	#p1 {background-color:#0F0;} content	#元素标签 id 属性值{样式属性}

包含选择器	div h3{color:red;}
	
	 CSS 层叠样式表
	
		父元素标签 子元素标签{
		样式属性
		}

子选择器	div>ul{color:blue;}
	
	 - 测试 1
	
	 - 嵌套元素
	
	 - 嵌套元素
		父元素标签名>子元素名{
		样式属性
		}

	- 嵌套元素 - 嵌套元素 - 测试 1 - 测试 1 	

504. 引入样式的三种方式及其优先级别

三种引用方式：

1. 外部样式表（存放.css 文件中）

不需要 style 标签

<link rel=”stylesheet” href=”引用文件地址” />

2. 嵌入式样式表

<style type=“text/css”>

p{color:red;}

</style>

3.内联样式

标签属性名为 style

优先级级别：在一个页面上，自上而下，后定义优先级高。

505. 盒子模型

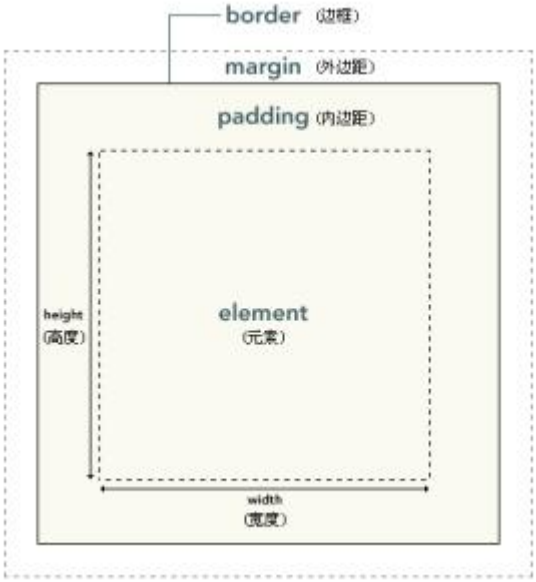
盒子模型类似于生活中的盒子，具有 4 个属性，外边距，内边距，边框，内容。

外边距：margin，用于设置元素和其他元素之间的距离。

内边距：padding,用于设置元素内容和边框之间的距离。

边框：border,用于设置元素边框粗细，颜色，线型。

内容：width,height,用于设置元素内容显示的大小。



例如：

```
<style>
        body{
            margin: 0;   /*取消body默认的外边距*/
```

```
        }
        #img1{
            width:200px;    /*设置图片的宽度*/
            border: 2px solid black; /*设置图片边框*/
            margin: 5px;
/*设置图片外边距（表示该图片与其他图片的距离为5px）*/
            padding:10px; /*设置图片与边框之间的距离*/
        }
        #img2{
            height: 200px; /*  设置图片的高度*/
            border: 2px solid black; /*设置图片的边框*/
            margin: 5px; /*设置图片外边距*/
            padding: 20px; /*设置图片与边框之间的距离*/
        }
</style>
    <img id="img1" src="img/2.jpg" />
        <img id="img2" src="img/lss.jpg" />
```

506. JavaScript 语言及其特点

Javascript 一种基于对象(object-based)和事件驱动(Event Driven)的简单的并具有安全性能的脚本语言。特点：

解释性： JavaScript 不同于一些编译性的程序语言，例如 C、C++等，它

是一种解释性的程序语言，它的源代码不需要经过编译，而直接在浏览器中运行时被解释。

基于对象： JavaScript 是一种基于对象的语言。这意味着它能运用自己已经创建的对象。因此，许多功能可以来自于脚本环境中对象的方法与脚本的相互作用。

事件驱动 :JavaScript 可以直接对用户或客户输入做出响应，无须经过 Web 服务程序。它对用户的响应，是以事件驱动的方式进行的。所谓事件驱动，就是指在主页中执行了某种操作所产生的动作，此动作称为“事件”。比如按下鼠标、移动窗口、选择菜单等都可以视为事件。当事件发生后，可能会引起相应的事件响应。

跨平台:JavaScript 依赖于浏览器本身，与操作环境无关，只要能运行浏览器的计算机，并支持 JavaScript 的浏览器就可正确执行。

507. JavaScript 常用数据类型有哪些

数值型：整数和浮点数统称为数值。例如 85 或 3.1415926 等。

字符串型：由 0 个,1 个或多个字符组成的序列。在 JavaScript 中，用双引号或单引号括起来表示，如“您好”、‘学习 JavaScript’等。 不区分单引号、双引号。

逻辑（布尔）型：用 true 或 false 来表示。

空（null）值：表示没有值，用于定义空的或不存在的引用。

要注意，空值不等同于空字符串""或 0。

未定义（undefined）值：它也是一个保留字。表示变量虽然已经声明，但却没有赋值。

除了以上五种基本的数据类型之外，JavaScript 还支持复合数据类型，包括对象和数组两种。

508. Javascript 的常用对象有哪些

常用对象包括日期对象Date，字符串对象String，数组对象Array

//获取并显示系统当前时间

```
function testDate(){
    var date = new Date();
    var fmtDate = date.getFullYear()+"-"+(date.getMonth()+1)+
        "-"+date.getDate()+"-"+date.getHours()
        +":"+date.getMinutes()+":"+date.getSeconds();
    alert(fmtDate);
}
```

//获取出’sxt’的下标位置

```
function testString(){
        var str = 'welcome to beijingsxt';
        alert(str.indexOf('sxt'));
 }
```

//遍历数组信息

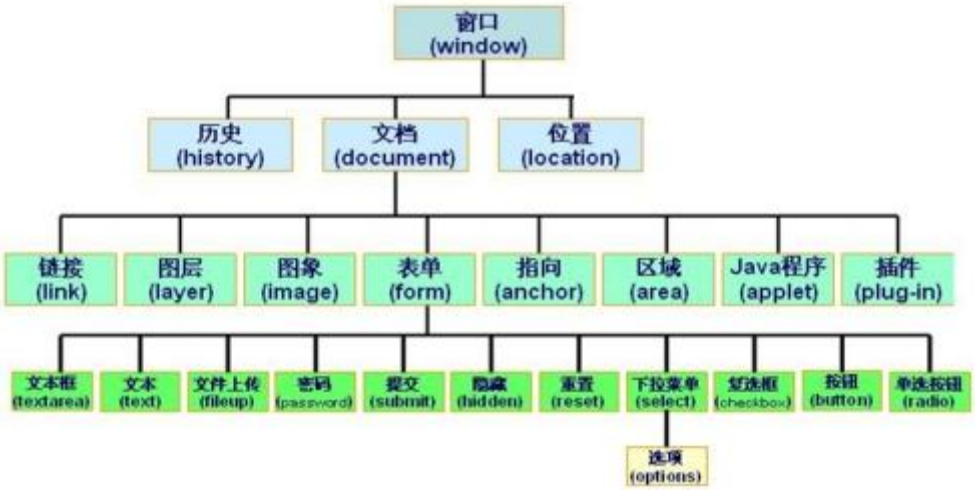
```
        function testArray(){
            var arr = new Array('a',123,'c',true,'e');
```

```
            for(var item in arr){
                document.write(arr[item]+" ");
            }
        }
```

509. DOM 和 BOM 及其关系

BOM 浏览器对象模型，由一系列对象组成，是访问、控制、修改浏览器的属性的方法。

DOM 文档对象模型，由一系列对象组成，是访问、检索、修改 XHTML 文档内容与结构的标准方法。



关系：

- BOM 描述了与浏览器进行交互的方法和接口
- DOM 描述了处理网页内容的方法和接口
- DOM 属于 BOM 的一个属性

510. JavaScript 中获取某个元素的三种方式 JavaScript 中的三种弹出式消息提醒命令是什么?

window.alert() 显示一个提示信息

window.confirm() 显示一个带有提示信息、确定和取消按钮的对话框

window.prompt()显示可提示用户输入的对话框

setTimeout 与 setInterval 的区别

setTimeout 和 setInterval 的语法相同。它们都有两个参数，一个是将要执行的代码字符串，还有一个是以毫秒为单位的时间间隔，当过了那个时间段之后就将执行那段代码。

不过这两个函数还是有区别的，setInterval 在执行完一次代码之后，经过了那个固定的时间间隔，它还会自动重复执行代码，而 setTimeout 只执行一次那段代码。

window.setTimeout("function",time)；//设置一个超时对象，只执行一次,无周期

window.setInterval("function",time)；//设置一个超时对象，周期='交互时间'

511. JavaScript 操作 CSS 的两种方式

第一种方式：操作元素的属性（对象.style.样式名=样式值;）

//改变直接样式

```
var child2 = document.createElement("div");
child2.innerHTML = "child2";
child2.style.fontWeight = "bold";
```

```
        parent.appendChild(child2);
```

第二种方式：操作元素的类（对象.className=类;）

例如：

```
    var parent = document.getElementById("parent");
        //改变 className
        var child0 = document.createElement("div");
        child0.innerHTML = "child0";
        child0.className = "newDiv";
      parent.appendChild(child0);
```

512. 静态网页和动态网页的联系和区别

联系：

1）静态网页是网站建设的基础，静态网页和动态网页都要使用到 HTMl 语言。

2）静态网页是相对于动态网页而言，指没有后台数据库、不含程序和不可交互的网页、是标准的 HTML 文件，它的文件扩展名是.htm 或.html。你编的是什么它显示的就是什么、不会有任何改变。

3）静态网页和动态网页之间并不矛盾，为了网站适应搜索引擎检索的需要，动态网站可以采用静动结合的原则，适合采用动态网页的地方用动态网页，如果必要使用静态网页，则可以考虑用静态网页的方法来实现，在同一个网站上，动态网页内容和静态网页内容同时

存在也是很常见的事情。

区别：

1）程序是否在服务器端运行，是重要标志。在服务器端运行的程序、网页、组件，属于动态网页，它们会随不同客户、不同时间，返回不同的网页，例如 ASP、PHP、JSP、ASP.net、CGI 等。运行于客户端的程序、网页、插件、组件，属于静态网页，例如 html 页、Flash、javascript、VBscript 等等，它们是永远不变的。

2）编程技术不同。静态网页和动态网页主要根据网页制作的语言来区分。静态网页使用语言：HTML。 动态网页使用语言：HTML + ASP 或 HTML + PHP 或 HTML + JSP 等其它网站动态语言。

3）被搜索引擎收录情况不同。由于编程技术不容，静态网页是纯粹 HTML 格式的网页，页面内容稳定，不论是网页是否被访问，页面都被保存在网站服务器上，很容易被搜索引擎收录。而动态网页的内容是当用户点击请求时才从数据库中调出返回给用户一个网页的内容，并不是存放在服务器上的独立文件，相比较于静态网页而言，动态网页很难被搜索引擎收录。

4）用户访问速度不同。用户访问动态网页时，网页在获得搜索指令后经过数据库的调查匹配，再将与指令相符的内容传递给服务器，通过服务器的编译将网页编译成标准的 HTML 代码，从而传递给用户浏览器，多个读取过程大大降低了用户的访问速度。而静态网页不同，由于网页内容直接存取在服务器上，省去了服务器的编译过程，用户访问网页速度很快。

5）制作和后期维护工作量不同。动态网页的设计以数据库技术为基础，可以实现多种功能，降低了网站维护的工作量。而静态网页由于没有数据库的支持，网页内容更改时需要直接修改代码，在网站内容制作和维护中，所需的工作量更大。

动态网页与静态网页各有特点，网站设计师在网页设计时，主要根据网站的功能需求和网站内容多少选择不同网页。如，网站包含信息量太大时，就需要选择动态网页，反之，则选择静态网页。

513. JSP/ASP/PHP 的比较

ASP(Active Server Pages),JSP(JavaServer Pages),PHP(Hypertext Preprocessor)是目前主流的三种动态网页语言。

ASP 是微软（Microsoft）所开发的一种后台脚本语言，它的语法和 Visual BASIC 类似，可以像 SSI（Server Side Include）那样把后台脚本代码内嵌到 HTML 页面中。虽然 ASP 简单易用，但是它自身存在着许多缺陷，最重要的就是安全性问题。

PHP 是一种跨平台的服务器端的嵌入式脚本语言。它大量地借用 C,Java 和 Perl 语言的语法，并耦合 PHP 自己的特性,使 WEB 开发者能够快速地写出动态产生页面。它支持目前绝大多数数据库。

JSP 是一个简化的 Servlet，它是由 Sun 公司倡导、许多公司参与一起建立的一种动态网页技术标准。JSP 技术有点类似 ASP 技术，它是在传统的网页 HTML 中插入 Java 程序段和 JSP 标记(tag)，从而形成 JSP 文件，后缀名为(*.jsp)。 用 JSP 开发的 Web 应用是跨平台的，既能在 Linux 下运行，也能在其他操作系统上运行。

ASP 优点:无需编译、易于生成、独立于浏览器、面向对象、与任何 ActiveX scripting 语言兼容、源程序码不会外漏。

缺点:

1） Windows 本身的所有问题都会一成不变的也累加到了它的身上。安全性、稳定性、跨平台性都会因为与 NT 的捆绑而显现出来。

2） ASP 由于使用了 COM 组件所以它会变的十分强大，但是这样的强大由于 Windows NT 系统最初的设计问题而会引发大量的安全问题。只要在这样的组件或是操作中一不注意，那么外部攻击就可以取得相当高的权限而导致网站瘫痪或者数据丢失。

3） 还无法完全实现一些企业级的功能：完全的集群、负载均横。

PHP 优点：

1）一种能快速学习、跨平台、有良好数据库交互能力的开发语言。

2）简单轻便，易学易用。

3) 与 Apache 及其它扩展库结合紧密。

缺点：

1) 数据库支持的极大变化。

2) 不适合应用于大型电子商务站点。

JSP 优点：

1) 一处编写随处运行。

2) 系统的多台平支持。

3) 强大的的可伸缩性。

4) 多样化和功能强大的开发工具支持。

缺点：

1) 与 ASP 一样，Java 的一些优势正是它致命的问题所在。

2) 开发速度慢

514. CGI/Servlet/JSP 的比较

CGI(Common Gateway Interface)，通用网关接口,是一种根据请求信息动态产生回应内容的技术。

通过 CGI，Web 服务器可以将根据请求不同启动不同的外部程序，并将请求内容转发给该程序，在程序执行结束后，将执行结果作为回应返回给客户端。也就是说，对于每个请求，都要产生一个新的进程进行处理。

Servlet 是在服务器上运行的小程序。在实际运行的时候 Java Servlet 与 Web 服务器会融为一体。与 CGI 不同的是，Servlet 对每个请求都是单独启动一个线程，而不是进程。这种处理方式大幅度地降低了系统里的进程数量，提高了系统的并发处理能力。

比较：

1）JSP 从本质上说就是 Servlet。JSP 技术产生于 Servlet 之后，两者分工协作，Servlet 侧重于解决运算和业务逻辑问题，JSP 则侧重于解决展示问题。

2) 与 CGI 相比，Servlet 效率更高。Servlet 处于服务器进程中，它通过多线程方式运行其 service 方法，一个实例可以服务于多个请求，并且其实例一般不会销毁。而 CGI 对每个请求都产生新的进程，服务完成后就销毁，所以效率上低于 Servlet 。

3）与 CGI 相比，Servlet 更容易使用，功能更强大，具有更好的可

移植性，更节省投资。在未来的技术发展过程中，Servlet 有可能彻底取代 CGI。

515. Tomcat/Jboss/WebSphere/WebLogic 的作用和特点

作用：

Tomcat：目前应用非常广泛的免费 web 服务器，支持部分 j2ee。

JBoss：JBoss 是一个管理 EJB 的容器和服务器，支持 EJB 1.1、EJB 2.0 和 EJB3.0 的规范。但 JBoss 核心服务不包括支持 servlet/JSP 的 WEB 容器，一般与 Tomcat 或 Jetty 绑定使用。

WebSphere：是 IBM 集成软件平台。可做 web 服务器，WebSphere 提供了可靠、灵活和健壮的集成软件。

Weblogic：是美国 bea 公司出品的一个基于 j2ee 架构的中间件。BEA WebLogic 是用于开发、集成、部署和管理大型分布式 Web 应用、网络应用和数据库应用的 Java 应用服务器。

特点（区别）：

1）价位不同：JBoss 与 Tomcat 的是免费的；WebLogic 与 WebSphere 是收费的，而且价格不菲。

2）开源性不同：JBoss 与 Tomcat 的是完全开源的，而其他两个不是。

3）对技术的支持：Tomcat 不支持 EJB，JBoss 是实现了 EJB 容器，再集成了 Tomcat。

WebLogic 与 WebSphere 都是对业内多种标准的全面支持 ,包括 EJB、JSB、JMS、JDBC、XML 和 WML，使 Web 应用系统实施更简单，且保护投资，同时也使基于标准的解决方案的开发更加简便。

4）扩展性的不同：WebLogic 和 WebSphere 都是以其高扩展的架构体系闻名于业内，包括客户机连接的共享、资源 pooling 以及动态网页和 EJB 组件群集。

5）应用范围的区别：Tomcat 是一个小型的轻量级应用服务器，在中小型系统和并发访问用户不是很多的场合下被普遍使用，是开发和调试 JSP 程序的首选。WebLogic 和 WebSphere 是商业软件,功能齐全强大,主要应用于大型企业的大型项目。JBOSS 主要应用于 EJB 服务的中小型公司。

6）安全性问题区别：因为 JBoss 和 Tomcat 都是开源的，所以它们的安全性相对来说比较低，万一应用服务器本身有什么漏洞，你是没办法向 Apache 索赔的。而 WebLogic 和 WebSphere 其容错、系统管理和安全性能已经在全球数以千记的关键任务环境中得以验证。

516. B/S 和 C/S 的含义及其区别

C/S 结构，即 Client/Server(客户机/服务器)结构，通过将任务合理分配到 Client 端和 Server 端，降低了系统的通讯开销，可充分利用两端硬件环境优势。早期软件系统多以此作为首选设计标准。

B/S 结构，即 Browser/Server(浏览器/服务器)结构，是随着 Internet 技术的兴起，对 C/S 结构的一种变化或者改进的结构。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，节约了开发成本，便于软件维护。

区别

1、C/S 是建立在局域网的基础上的。B/S 是建立在广域网的基础上的，但并不是说 B/S 结构不能在局域网上使用。

2、B/S 业务扩展简单方便，通过增加页面即可增加服务器功能。C/S 的客户端还需要安装专用的客户端软件，不利于扩展。

3、B/S 维护简单方便。开发、维护等几乎所有工作也都集中在服务器端，当企业对网络应用进行升级时，只需更新服务器端的软件就可以，这减轻了异地用户系统维护与升级的成本。。

4、B/S 响应速度不及 C/S；

5、B/S 用户体验效果不是很理想

517. 容器的理解

容器也是 java 程序，它的主要作用是为应用程序提供运行环境。容器用来接管安全性、并发性、事务处理、交换到辅助存储器和其它服务的责任

以 tomcat 为例 :Tomcat 是一个后台服务进程 ,其它的 servlet(相当于 DLL) 是在 Tomcat 容器内运行,Broswer 只与 Tomcat 通迅; Tomcat 接受 browser 的请求，经过一系列动作（如果是静态网页，那么装载，按 http 协议形成响应流;如果是动态的如 JSP，那就要调用 JDK 中的 servlet.jsp 接口，解释形成静态网页，按 http 协议生成响应流发送回 browser）后，形成静态网页，返回响应。

518. HTTP 协议工作原理及其特点

超文本传输协议（HTTP：Hypertext Transport Protocol）是万维网应用层的协议，它通过两个程序实现：一个是客户端程序（各种浏览器），另一个是服务器 （常称 Web 服务器）。这两个通常运行在不同的主机上，通过

交换报文来完成网页请求和响应，报文可简单分为请求报文和响应报文。

工作原理（流程）：

客户机与服务器建立连接后，浏览器可以向 web 服务器发送请求并显示收到的网页，当用户在浏览器地址栏中输入一个 URL 或点击一个超连接时，浏览器就向服务器发出了 HTTP 请求，请求方式的格式为：统一资源标识符、协议版本号，后边是 MIME（Multipurpose Internet Mail Extensions）信息包括请求修饰符、客户机信息和可能的内容。该请求被送往由 URL 指定的 WEB 服务器，WEB 服务器接收到请求后，进行相应反映，其格式为：一个状态行包括信息的协议版本号、一个成功或错误的代码，后边服务器信息、实体信息和可能的内容。即以 HTTP 规定的格式送回所要求的文件或其他相关信息，再由用户计算机上的浏览器负责解释和显示。

特点：

1）支持客户/服务器模式。

2）简单快速：客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有 GET、HEAD、POST。每种方法规定了客户与服务器联系的类型不同。由于 HTTP 协议简单，使得 HTTP 服务器的程序规模小，因而通信速度很快。

3）灵活：HTTP 允许传输任意类型的数据对象。正在传输的类型由 Content-Type 加以标记。

4）无连接：无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。

5）无状态：HTTP 协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。

519. get 和 post 的区别

1. Get 是不安全的，因为在传输过程，数据被放在请求的 URL 中；Post 的所有操作对用户来说都是不可见的。

2. Get 传送的数据量较小，这主要是因为受 URL 长度限制；Post 传送的数据量较大，一般被默认为不受限制。

3. Get 限制 Form 表单的数据集的值必须为 ASCII 字符；而 Post 支持整个 ISO10646 字符集。

4. Get 执行效率却比 Post 方法好。Get 是 form 提交的默认方法。

520. 如何解决表单提交的中文乱码问题

1）设置页面编码，若是 jsp 页面，需编写代码

```
<%@page language="java" pageEncoding="UTF-8"
contentType="text/html;charset=UTF-8" %>
```

若是 html 页面，在网页头部（<head></head>）中添加下面这段代码

```
<meta http-equiv="Content-Type" content="text/html;
```

charset=utf-8" />

2）将 form 表单提交方式变为 post 方式，即添加 method="post"；）在 Servlet 类中编写代码 request.setCharacterEncoding("UTF-8")，而且必须写在第一行。

3）如果是 get 请求，在 Servlet 类中编写代码

```
byte [] bytes = str.getBytes("iso-8859-1");
String  cstr =  new String(bytes,"utf-8");
```

或者直接修改 Tomcat 服务器配置文件 server.xml 增加内容：

URIEncoding="utf-8"

521. 绝对路径、根路径、相对路径的含义及其区别

绝对路径指对站点的根目录而言某文件的位置，相对路径指以当前文件所处目录而言某文件的位置,相对路径-以引用文件之网页所在位置为参考基础，而建立出的目录路径。绝对路径-以 Web 站点根目录为参考基础的目录路径。

先给出一个网站结构图做实例加深理解，A 网站（域名为 http://www.a.com）:/include/a-test.html，/img/a-next.jpg；B 网站（域名为 http://www.b.com）: /include/b-test.html，/img/b-next.jpg。

相对路径是从引用的网页文件本身开始构建的，如果在 A 网站中的 a-test.html 中要插入图片 a-next.jpg，可以这样做：，重点是 img 前面的../，表示从 html 处于的 include 开始起步，输入一个../表示回到上面一级父文件夹下，然后再接着 img/表示又从父级文件夹下的 img 文件开始了，最后定位 img 下面的

next.jpg。

根路径是从网站的最底层开始起，一般的网站的根目录就是域名下对应的文件夹，就如 D 盘是一个网站，双击 D 盘进入到 D 盘看到的就是网站的根目录，这种路径的链接样式是这样的：如果在 A 网站中的 a-test.html 中要插入图片 a-next.jpg，可以这样做:<img src="/img/a-next.jpg" >，以/开头表示从网站根目录算起 ,找到根目录下面的 img 文件夹下的 next.jpg。

绝对路径就很好理解了，这种路径一般带有网站的域名，如果在 A 网站中的 a-test.html 中要插入图片 a-next.jpg，需要这样这样写:<img src="http://www.a.com/img/a-next.jpg" > ,将图片路径上带有了域名信息，再打个比方：如果在 A 网站中的 a-test.html 中要插入 B 网站的图片 b-next.jpg ，就需要这样写：<img src="http://www.b.com/img/b-next.jpg" >，这种方法适用与在不同网站之间插入外部网站的图片。

522. 如实现 servlet 的单线程模式

实现 servlet 的单线程的 jsp 命令是： <%@ page isThreadSafe=”false”%>。默认 isThreadSafe 值为 true。

属性 isThreadSafe=false 模式表示它是以 Singleton 模式运行，该模式 implements 了接口 SingleThreadMode, 该模式同一时刻只有一个实例，不会出现信息同步与否的概念。若多个用户同时访问一个这种模式的页面，那么先访问者完全执行完该页面后，后访问者才开始执行。

属性 isThreadSafe=true 模式表示它以多线程方式运行。该模式的信息同步，需访问同步方法(用 synchronized 标记的)来实现。 一般格式如下：

```
public synchronized void syncmethod(...){
        while(...) {
           this.wait();
        }
        this.notifyAll();
}
```

523. Servlet 的生命周期

1) 加载：在下列时刻加载 Servlet：（1）如果已配置自动加载选项，则在启动服务器时自动加载（web.xml 中设置<load-on-start>）；（2）在服务器启动后，客户机首次向 Servlet 发出请求时;（3）重新加载 Servlet 时（只执行一次）

2) 实例化：加载 Servlet 后，服务器创建一个 Servlet 实例。（只执行一次）

3) 初始化：调用 Servlet 的 init() 方法。在初始化阶段，Servlet 初始化参数被传递给 Servlet 配置对象 ServletConfig。 （只执行一次）

4) 请求处理：对于到达服务器的客户机请求，服务器创建针对此次请求的一个“请求”对象和一个“响应”对象。服务器调用 Servlet 的 service() 方法，该方法用于传递“请求”和“响应”对象。service() 方法从“请求”对象获得请求信息、处理该请求并用“响应”对象的方法以将响应传回客户机。service() 方法可以调用其它方法来处理请求，例如 doGet()、doPost() 或其它的方法。（每次请求都执行该步骤）

5) 销毁：当服务器不再需要 Servlet, 或重新装入 Servlet 的新实例时，

服务器会调用 Servlet 的 destroy() 方法。（只执行一次）

524. session 和 cookie 的区别

联系

http 是无状态的协议，客户每次读取 web 页面时，服务器都打开新的会话，而且服务器也不会自动维护客户的上下文信息，那么要怎么才能在多次请求之间共享信息呢（比如实现网上商店中的购物车）？session 和 cookie 就是为了解决 HTTP 协议的无状态而采用的两种解决方案。

原理（通过比喻形象说明，真正原理自己总结）

Cookie :发给顾客一张卡片，上面记录着消费的数量，一般还有个有效期限。每次消费时，如果顾客出示这张卡片，则此次消费就会与以前或以后的消费相联系起来。这种做法就是在客户端保持状态。 【卡上记录所有信息，而店家只认卡不认人。】

Session：发给顾客一张会员卡，除了卡号之外什么信息也不纪录，每次消费时，如果顾客出示该卡片，则店员在店里的纪录本上找到这个卡号对应的纪录添加一些消费信息。这种做法就是在服务器端保持状态。 【只记用户 ID，而 ID 的详细记录放在店家的数据库里；每次凭 ID 检索服务器的记录。】

区别

cookie 数据存放在客户的浏览器上，session 数据放在服务器上(sessionid 可以通过 cookie 保存在客户端，也可以使用 URL 重写方式)。

cookie 不是很安全，别人可以分析存放在本地的 COOKIE 并进行 COOKIE 欺骗，考虑到安全应当使用 session

session 会在一定时间内保存在服务器上。当访问增多，会比较占用你服务

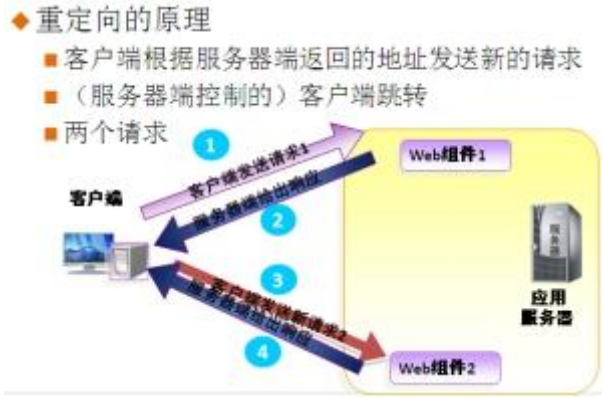
器的性能，考虑到减轻服务器性能方面，应当使用 COOKIE

单个 cookie 在客户端的限制是 3K，就是说一个站点在客户端存放的 COOKIE 不能 3K。

个人建议：将登陆信息等重要信息存放为 SESSION；其他信息如需保留，可放在 COOKIE

525. 转发和重定向的区别

转发是在服务端直接做的事情，是对客户端的同一个 request 进行传递，浏览器并不知道。重定向是由浏览器来做的事情。重定向时，服务端返回一个 response，里面包含了跳转的地址，由浏览器获得后，自动发送一个新 request。转发像呼叫转移或者 110 报警中心，重定向似 114 查号台。



a) 区别 1:跳转效率的不同

转发效率相对高；重定向效率相对低

b) 区别 2:实现语句不同

转 发

request.getRequestDispatcher("xxxx").forward(request,response) ;

重定向 response.sendRedirect("xxxx")

c) 区别 3:是否共有同一个 request 的数据

转发源组件与目标组件共有同一个 request 数据

重定向源组件与目标组件不共有同一个 request 数据（可使用 session 共有数据）

d) 区别 4:浏览器 URL 地址的不同

转发后浏览器 URL 地址保持不变（源组件地址）

重定向后浏览器 URL 地址改变为重定向后的地址（目标组件地址）

e) 区别 5:"/"路径的含义不同

转发时"/"代表当前项目的根路径 ；重定向时"/"代表当前服务器的根路径

f) 区别 6:跳转范围的不同

只能转发到同一应用中的 URL（默认） ；可以重定向任何服务器、任何应用的 URL

g) 区别 7:刷新是否导致重复提交

转发会导致重复提交(可以通过同步令牌解决)；重定向不会导致重复提交

h) 区别 8:是否经过过滤器

转发不经过过滤器（默认情况）; 重定向经过过滤器

526. JSP 的执行过程

在 JSP 运行过程中，首先由客户端发出请求，Web 服务器接收到请求后，如

果是第一次访问某个 jsp 页面，Web 服务器对它进行以下 3 个操作。

1) 翻译：由.jsp 变为.java,由 JSP 引擎实现。

2) 编译：由.java 变为.class,由 Java 编译器实现。

3) 执行：由.class 变为.html,用 Java 虚拟机执行编译文件,然后将执行结果返回给 Web 服务器，并最终返回给客户端

如果不是第一次访问某个 JSP 页面，则只执行第三步。所以第一次访问 JSP 较慢。

527. JSP 的 9 个内置对象及其含义

内部对象	所属类型	用途
request	javax.servlet.httpServletRequest	包含了请求方的信息
response	javax.servlet.httpServletResponse	封装了对客户端的响应
out	javax.servlet.jsp.JspWriter	响应信息流的标准输出
session	javax.servlet.Http.HttpSession	在同一请求中所产生的session资料，目前只对Http协议有定义
application	javax.servlet.ServletContext	提供安全信息
config	javax.servlet.ServletConfig	提供配置信息
pageContext	javax.servlet.jsp.PageContext	提供当前页面属性
page	java.lang.Object	同于java的this
exception	java.lang.Throwable	异常处理

1) request 表示 HttpServletRequest 对象。它包含了有关浏览器请求的信息

2) response 表示 HttpServletResponse 对象，并提供了几个用于设置浏览器的响应的方法

3) 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。

4) pageContext 表示一个 javax.servlet.jsp.PageContext 对象。是用于方便存取各种范围的名字空间

5) 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

6) 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

7) 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。

8) page 表示从该页面产生的一个 servlet 实例

9) exception：exception 对象用来处理错误异；如果使用 exception，则必须指定 page 中的 isErrorPage 为 true

528. JSP 动作有哪些,简述作用?

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。

529. JSP 中动态 INCLUDE 与静态 INCLUDE 的区别

回答这个问题得先清楚 Jsp 的执行过程。第一次访问该 jsp 页面时，需经 3 个步骤。即翻译（由.jsp 变为.java）--->编译（由.java 变为.class）--->执行（由.class 变为.html）。若不是第一次访问，只需经过步骤中的执行过程

就行。

区别：

1）使用语法不同。动态 INCLUDE 用 jsp:include 动作实现，称为行动元素。

如：<jsp:include page="included.jsp" flush="true" />

静态 INCLUDE 用 include 伪码实现,称为指令元素。

如：<%@ include file="included.html" %>。

2）执行时间不同。<jsp:include page="included.jsp" flush="true" />在请求处理阶段执行；会先解析所要包含的页面（included.jsp），解析后在和主页面放到一起显示；先编译，后包含。<%@ include file="included.html" %>在翻译阶段执行；不会解析所要包含的页面（included.html），先合成一个文件后再转换成 servlet，即先包含，后统一编译。

3）访问时产生文件个数不同。<jsp:include page="included.jsp" flush="true" />共产生 2 个 java 文件和 2 个 class 文件。<%@ include file="included.html" %>共产生 1 个 java 文件和 1 个 class 文件。

4）引入内容不同。动态 INCLUDE 主要是对动态页面的引入，它总是会检查所引入的页面的变化，如果被引入的文件内容在请求间发生变化，则下一次请求包含<jsp:include>动作的 jsp 时，将显示被引入文件的新内容。静态 INCLUDE 适用于引入静态页面（.html），include 指令它不会检查所引入的页面的变化，在 jsp 页面转换成 servlet 前与当前 jsp 文件融合在一起，之后与主页面一起显示，常使用静态 INCLUDE 引入导航条、页脚等。

530. page/request/session/application 作用域区别

page：当前页面范围

request：当前页面范围+转发页面（forward）+包含页面（include）

session：当前会话：session 在以下几种情况下失效

1) 销毁 session：Session.invalidate();
2) 超过最大非活动间隔时间
3) 手动关闭浏览器（session 并没有立刻失效，因为服务器端 session 仍旧存在，超过最大非活动间隔时间后真正失效）

application：当前应用；服务器重新启动前一直有效

531. JSP 和 Servlet 的区别和联系

区别：

1) JSP 是在 HTML 代码里写 JAVA 代码,框架是 HTML;而 Servlet 是在 JAVA 代码中写 HTML 代码，本身是个 JAVA 类。
2) JSP 使人们把显示和逻辑分隔成为可能，这意味着两者的开发可并行进行；而 Servlet 并没有把两者分开。
3) Servlet 独立地处理静态表示逻辑与动态业务逻辑.这样,任何文件的变动都需要对此服务程序重新编译;JSP 允许用特殊标签直接嵌入到 HTML 页面, HTML 内容与 JAVA 内容也可放在单独文件中,HTML 内容的任何变动会自动编译装入到服务程序.
4) Servlet 需要在 web.xml 中配置，而 JSP 无需配置。
5) 目前 JSP 主要用在视图层，负责显示，而 Servlet 主要用在控制层，负责调度

联系：

1) 都是 Sun 公司推出的动态网页技术。
2) 先有 Servlet，针对 Servlet 缺点推出 JSP。JSP 是 Servlet 的一种特殊形式，每个 JSP 页面就是一个 Servlet 实例——JSP 页面由系统翻译成 Servlet，Servlet 再负责响应用户请求。

532. 为什么要使用连接池？

- 传统的数据库连接方式

一个连接对象对应一个物理连接

每次操作都打开一个物理连接，

使用完都关闭连接，造成系统性能低下。

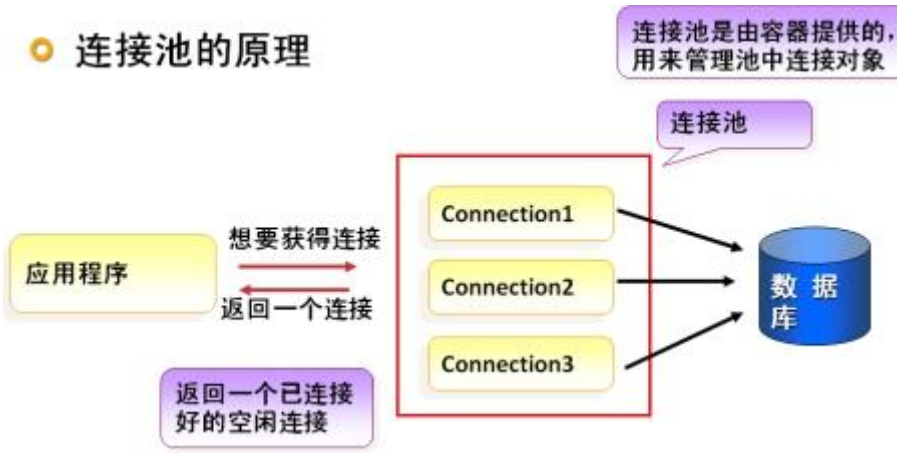
- 连接池技术

客户程序得到的连接对象是连接池中物理连接的一个句柄

调用连接对象的 close()方法,物理连接并没有关闭,数据源的实现只是删除了客户程序中的连接对象和池中的连接对象之间的联系.

- 数据库连接的建立及关闭是耗费系统资源的操作 ,在大型应用中对系统的性能影响尤为明显。为了能重复利用数据库连接对象,缩短请求的响应时间和提高服务器的性能,支持更多的客户，应采用连接池技术.

533. 数据库连接池的原理。



数据库连接池的原理。

- 传统连接方式:

首先调用 Class.forName()方法加载数据库驱动，

然后调用 DriverManager.getConnection()方法建立连接.

- 连接池技术:

连接池解决方案是在应用程序启动时就预先建立多个数据库连接对象,然后将连接对象保存到连接池中。

当客户请求到来时,从池中取出一个连接对象为客户服务。

当请求完成时,客户程序调用 close()方法,将连接对象放回池中.

对于多于连接池中连接数的请求，排队等待。

应用程序还可根据连接池中连接的使用率，动态增加或减少池中的连接数。

534. 谈谈过滤器原理及其作用?

原理:

- 过滤器是运行在服务器端的一个拦截作用的 web 组件，一个请求来到时，web 容器会判断是否有过滤器与该信息资源相关联，如果有则交给过滤

器处理，然后再交给目标资源，响应的时候则以相反的顺序交给过滤器处理，最后再返回给用户浏览器

- 一般用于日志记录、性能、安全、权限管理等公共模块。

过滤器开发:

- 过滤器是一个实现了 javax.servlet.Filter 接口的 java 类
- 主要业务代码放在 doFilter 方法中
- 业务代码完成后要将请求向后传递，即调用 FilterChain 对象的 doFilter 方法

配置:

在web.xml中增加如下代码

```
<filter>
    <filter-name>MyFilter</filter-name>
    <filter-class>Filter完整类名</filter-class>
</filter>
<filter-mapping>
    <filter-name>MyFilter</filter-name>
    <url-pattern>/*(要过虑的url，此处*表示过虑所有的url)</url-pattern>
</filter-mapping>
```

谈谈监听器作用及其分类?

监听器也叫 Listener ,是一个实现特定接口的 java 类 ,使用时需要在 web.xml 中配置，它是 web 服务器端的一个组件，它们用于监听的事件源分别为

SerlvetConext,HttpSession 和 ServletRequest 这三个域对象

主要有以下三种操作:

- 监听三个域对象创建和销毁的事件监听器
- 监听域对象中属性的增加和删除的事件监听器
- 监听绑定到 HttpSession 域中的某个对象的状态的时间监听器

接口分类:

- ServletContextListener
- HttpSessionListener
- ServletRequestListener
- ServletContextAttributeListener
- HttpSessionAttributeListener
- ServletRequestAttributeListener
- HttpSessionBindingListener(不需要配置)
- HttpSessionActivationListener(不需要配置)

配置：

<listener><listener-class>实现以上任意接口的 java 类全名</listener-class></listener>

535. jQuery 相比 JavaScript 的优势在哪里

jQuery 的语法更加简单。

jQuery 消除了 JavaScript 跨平台兼容问题。

相比其他 JavaScript 和 JavaScript 库，jQuery 更容易使用。

jQuery 有一个庞大的库/函数。

jQuery 有良好的文档和帮助手册。

jQuery 支持 AJAX

536. DOM 对象和 jQuery 对象的区别及其转换

DOM 对象，是我们用传统的方法(javascript)获得的对象，jQuery 对象即是用 jQuery 类库的选择器获得的对象，它是对 DOM 对象的一种封装，jQuery 对象不能使用 DOM 对象的方法 ,只能使用 jQuery 对象自己的方法。

普通的 dom 对象一般可以通过$()转换成 jquery 对象

如：var cr=document.getElementById("cr"); //dom 对象

var $cr = $(cr); //转换成 jquery 对象

由于 jquery 对象本身是一个集合。所以如果 jquery 对象要转换为 dom 对象则必须取出其中的某一项，一般可通过索引取出

如：$("#msg")[0]，$("div").eq(1)[0]，$("div").get()[1]，$("td")[5]这几种语法在 jQuery 中都是合法的

537. jQuery 中$的作用主要有哪些

1) $用作选择器

例如:根据 id 获得页面元素$("#元素 ID")

2) $相当于 window.onload 和 $(document).ready(...)

例如:$(function(){...}); function(){...}会在 DOM 树加载完毕之后执行。

3) $用作 JQuery 的工具函数的前缀

例如： var str = ' Welcome to shanghai.com ';

str = $.trim(str);去掉空格

4) $(element)：把 DOM 节点转化成 jQuery 节点

例如：var cr=document.getElementById("cr"); //dom 对象

var $cr = $(cr); //转换成 jquery 对象

5) $(html)：使用 HTML 字符串创建 jQuery 节点

例如：var obj = $(" 尚学堂，实战化教学第一品牌 ")

538. Ajax 含义及其主要技术

Ajax (Asynchronous JavaScript and XML 阿贾克斯)不是一个新的技术，事实上 ,它是一些旧有的成熟的技术以一种全新的更加强大的方式整合在一起。

Ajax 的关键技术：

1) 使用 CSS 构建用户界面样式，负责页面排版和美工
2) 使用 DOM 进行动态显示和交互，对页面进行局部修改
3) 使用 XMLHttpRequest 异步获取数据
4) 使用 JavaScript 将所有元素绑定在一起

539. Ajax 的工作原理

Ajax 的原理简单来说通过 XmlHttpRequest 对象来向服务器发异步请求，从服务器获得数据，然后用 javascript 来操作 DOM 而更新页面。这其中最关键的一步就是从服务器获得请求数据。要清楚这个过程和原理，我们必须对 XMLHttpRequest 有所了解。

XMLHttpRequest 是 ajax 的核心机制，它是在 IE5 中首先引入的，是一种支持异步请求的技术。简单的说，也就是 javascript 可以及时向服务器提出请求和处理响应，而不阻塞用户。达到无刷新的效果。

540. JSON 及其作用

JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式,采用完全独立于语言的文本格式，是理想的数据交换格式。同时，JSON 是 JavaScript 原生格式,这意味着在 JavaScript 中处理 JSON 数据不须要任何特殊的 API 或工具包。

在 JSON 中，有两种结构：对象和数组。

1) 一个对象以“{”（左括号）开始，“}”（右括号）结束。每个“名称”后跟一个“:”（冒号）;“‘名称/值’ 对”之间运用 “,”（逗号）分隔。 名称用引号括起来；值如果是字符串则必须用括号，数值型则不须要。例如：

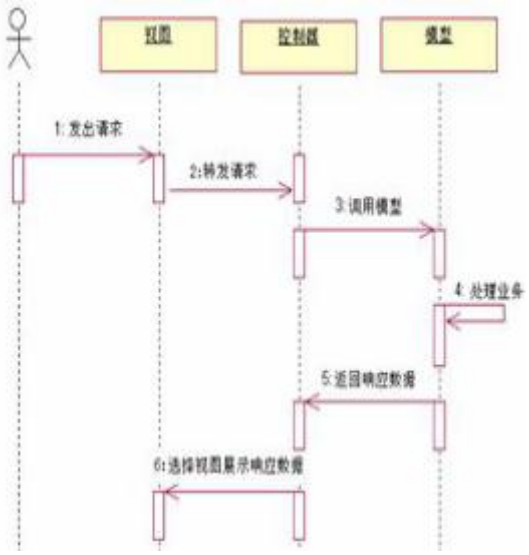
var o={"xlid":"cxh","xldigitid":123456,"topscore":2000,"topplaytime":"2009-08-20"};

2) 数组是值（value）的有序集合。一个数组以“[”（左中括号）开始，“]”（右中括号）结束。值之间运用 “,”（逗号）分隔。例如：

var jsonranklist=[{"xlid":"cxh","xldigitid":123456,"topscore":2000,"topplaytime":"2009-08-20"},
{"xlid":"zd","xldigitid":123456,"topscore":1500,"topplaytime":"2009-11-20"}];

541. MVC 模式及其优缺点

用户	
视图层V	JSP
控制层C	Servlet
模型层M	JavaBean
数据库	



一、MVC 原理

MVC 是一种程序开发设计模式,它实现了显示模块与功能模块的分离。提高了程序的可维护性、可移植性、可扩展性与可重用性，降低了程序的开发难度。它主要分模型、视图、控制器三层。

1、模型(model)它是应用程序的主体部分，主要包括业务逻辑模块和数据模块。模型与数据格式无关，这样一个模型能为多个视图提供数据。由于应用于模型的代码只需写一次就可以被多个视图重用，所以减少了代码的重复性

2、视图(view) 用户与之交互的界面、在 web 中视图一般由 jsp,html 组成

3、控制器(controller)接收来自界面的请求 并交给模型进行处理 在这个过程中控制器不做任何处理只是起到了一个连接的作用

二、MVC 的优点

1、降低代码耦合性。在 MVC 模式中，三个层各施其职，所以如果一旦哪一层的需求发生了变化，就只需要更改相应的层中的代码而不会影响到其他层中的代码。

2、有利于分工合作。在 MVC 模式中，由于按层把系统分开，那么就能更好的实现开发中的分工。网页设计人员可进行开发视图层中的 JSP，而对业务熟悉的人员可开发业务层，而其他开发人员可开发控制层。

3、有利于组件的重用。如控制层可独立成一个能用的组件，表示层也可做成通用的操作界面。可以为一个模型在运行时同时建立和使用多个视图。

三、MVC 的不足之处

1、增加了系统结构和实现的复杂性。对于简单的界面，严格遵循 MVC，使模型、视图与控制器分离，会增加结构的复杂性，并可能产生过多的更新操作，降低运行效率。

2、视图与控制器间的过于紧密的连接。视图与控制器是相互分离，但确实联系紧密的部件，视图没有控制器的存在，其应用是很有限的，反之亦然，这样就妨碍了他们的独立重用。

3、视图对模型数据的低效率访问。依据模型操作接口的不同，视图可能需要多次调用才能获得足够的显示数据。对未变化数据的不必要的频繁访问，也将损害操作性能。

4、目前，一般高级的界面工具或构造器不支持模式。改造这些工具以适应 MVC 需要和建立分离的部件的代价是很高的，从而造成 MVC 使用的困难。

542. EJB 与 JavaBean 的区别？

Java Bean 是可复用的组件，对 Java Bean 并没有严格的规范，理论上讲，任何一个 Java 类都可以是一个 Bean。但通常情况下，由于 JavaBean 是被容器所创建（如 Tomcat)的，所以 Java Bean 应具有一个无参的构造器。另外，通常 Java Bean 还要实现 Serializable 接口用于实现 Bean 的持久性。Java Bean 实际上相当于微软 COM 模型中的本地进程内 COM 组件，它是不能被跨进程访问的。

Enterprise Java Bean 相当于 DCOM，即分布式组件。它是基于 Java 的远程方法调用（RMI）技术的，所以 EJB 可以被远程访问（跨进程、跨计算机）。但 EJB 必须被布署在诸如 Webspere、WebLogic 这样的容器中，EJB 客户从不直接访问真正的 EJB 组件，而是通过其容器访问。EJB 容器是 EJB 组件的代理，EJB 组件由容器所创建和管理。客户通过容器来访问真正的 EJB 组件。

543. Oracle 完成分页功能的三层子查询语句及其含义？

如：

select * from (select t.*,rownum r from (select * from A) t where rownum < 10) where r >5

select * from A:要查询的数据

select t.*,rownum r from (select * from A) t where rownum < 10：取前 10 行

select * from (select t.*,rownum r from (select * from A) t where rownum < 10) where r >5：取 5-10 行

544. MVC 模式完成分页功能的基本思路是什么？

1）页面提交页码(第几页)到 Servlet 中

2）Servlet 接收到页码后，将页码传递给分页工具类(PageBean)

3）Servlet 中调用 Service 层传入 PageBean 对象

4）Service 层调用 DAO 层传入 PageBean 对象

5）Servlet 中得到查询出来的数据，并 setAttrivute 保存

6）在页面中得到(getAttribute)数据，遍历输出

545. 文件上传组件 Common-fileUpload 的常用类及其作用？

DiskFileItemFactory：磁盘文件工厂类，设置上传文件保存的磁盘目录，缓冲区大小。

ServletFileUpload：上传处理类，此类真正读取客户上传的文件，同时可以设置最大接收大小。

FileItem：上传的文件对象，可以是多个文件，每个上传的文件都是一个单独的 FileItem 对象。

546. 说出 Servlet 的生命周期，并说出 Servlet 和 CGI 的区别?

答：Web 容器加载 Servlet 并将其实例化后，Servlet 生命周期开始，容器运行其 init()方法进行 Servlet 的初始化 ;请求到达时调用 Servlet 的 service 方法，service 方法会调用与请求对应的 doGet 或 doPost 等方法；当服务器关闭会项目被卸载时服务器会将 Servlet 实例销毁，此时会调用 Servlet 的 destroy 方法。Servlet 与 CGI 的区别在于 Servlet 处于服务器进程中，

它通过多线程方式运行其 service 方法，一个实例可以服务于多个请求，并且其实例一般不会销毁，而 CGI 对每个请求都产生新的进程，服务完成后就销毁，所以效率上低于 Servlet。

【补充 1】SUN 公司在 1996 年发布 Servlet 技术就是为了和 CGI 进行竞争，Servlet 是一个特殊的 Java 程序，一个基于 Java 的 Web 应用通常包含一个或多个 Servlet 类。 Servlet 不能够自行创建并执行，它是在 Servlet 容器中运行的，容器将用户的请求传递给 Servlet 程序，此外将 Servlet 的响应回传给用户。通常一个 Servlet 会关联一个或多个 JSP 页面。以前 CGI 经常因为性能开销上的问题被诟病，然而 Fast CGI 早就已经解决了 CGI 效率上的问题，所以面试的时候大可不必诟病 CGI，腾讯的网站就使用了 CGI 技术，相信你也没感觉它哪里不好。

【补充 2】Servlet 接口定义了 5 个方法，其中前三个方法与 Servlet 生命周期相关：

void init(ServletConfig config) throws ServletException

void service(ServletRequest req, ServletResponse resp) throws ServletException, java.io.IOException

void destory()

java.lang.String getServletInfo()

ServletConfig getServletConfig()

547. 转发（forward）和重定向（redirect）的区别?

答：forward 是容器中控制权的转向，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 就是服务器端根据逻辑，发送一个状态码，告诉浏览器重新去请求那个地址，因此从浏览器的地址栏中可以看到跳转后的链接地址。前者更加高效，在前者可以满足需要时，尽量使用转发（通过 RequestDispatcher 对象的 forward 方法，RequestDispatcher 对象可以通过 ServletRequest 对象的 getRequestDispatcher 方法获得），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，则必须使用重定向（通过 HttpServletResponse 对象调用其 sendRedirect 方法）。

548. JSP 有哪些内置对象？作用分别是什么？

答：JSP 有 9 个内置对象：

request：封装客户端的请求，其中包含来自 GET 或 POST 请求的参数；

response：封装服务器对客户端的响应；

pageContext：通过该对象可以获取其他对象；

session：封装用户会话的对象；

application：封装服务器运行环境的对象；

out：输出服务器响应的输出流对象；

config：Web 应用的配置对象；

page：JSP 页面本身（相当于 Java 程序中的 this）；

exception：封装页面抛出异常的对象。

【补充】如果用 Servlet 来生成网页中的动态内容无疑是非常繁琐的工作，另一方面，所有的文本和 HTML 标签都是硬编码，即使做出微小的修改，都

需要进行重新编译。JSP 解决了 Servlet 的这些问题，它是 Servlet 很好的补充，可以专门用作呈现给用户的视图（View），而 Servlet 作为控制器（Controller）专门负责处理用户请求并转发或重定向到某个页面。基于 Java 的 Web 开发很多都同时使用了 Servlet 和 JSP。JSP 页面其实是一个 Servlet，能够运行 Servlet 的服务器（Servlet 容器）通常也是 JSP 容器，可以提供 JSP 页面的运行环境，Tomcat 就是一个 Servlet/JSP 容器。第一次请求一个 JSP 页面时，Servlet/JSP 容器首先将 JSP 页面转换成一个 JSP 页面的实现类，这是一个实现了 JspPage 接口或其子接口 HttpJspPage 的 Java 类。JspPage 接口是 Servlet 的子接口，因此每个 JSP 页面都是一个 Servlet。转换成功后，容器会编译 Servlet 类，之后容器加载和实例化 Java 字节码，并执行它通常对 Servlet 所做的生命周期操作。对同一个 JSP 页面的后续请求，容器会查看这个 JSP 页面是否被修改过，如果修改过就会重新转换并重新编译并执行。如果没有则执行内存中已经存在的 Servlet 实例。我们可以看一段 JSP 代码对应的 Java 程序就知道一切了，而且 9 个内置对象的神秘面纱也会被揭开。

549. get 和 post 请求的区别？

答：

①get 请求用来从服务器上获得资源，而 post 是用来向服务器提交数据；

②get 将表单中数据按照 name=value 的形式，添加到 action 所指向的 URL 后面，并且两者使用“?”连接，而各个变量之间使用“&”连接；post 是将表单中的数据放在 HTML 头部（header），传递到 action 所指向 URL；

③get 传输的数据要受到 URL 长度限制（1024 字节）；而 post 可以传输大

量的数据，上传文件只能使用 post 方式；

④使用 get 时参数会显示在地址栏上，如果这些数据不是敏感数据，那么可以使用 get；对于敏感数据还是应用使用 post；

⑤get 使用 MIME 类型 application/x-www-form-urlencoded 的 URL 编码（URL encoding，也叫百分号编码）文本的格式传递参数，保证被传送的参数由遵循规范的文本组成，例如一个空格的编码是"%20"。

550. 常用的 Web 容器

答：Unix 和 Linux 平台下使用最广泛的免费 HTTP 服务器是 Apache 服务器，而 Windows 平台的服务器通常使用 IIS 作为 Web 服务器。选择 Web 服务器应考虑的因素有：性能、安全性、日志和统计、虚拟主机、代理服务器、缓冲服务和集成应用程序等。下面是对常用服务器的简介：

IIS：Microsoft 的 Web 服务器产品为 Internet Information Services。IIS 是允许在公共 Intranet 或 Internet 上发布信息的 Web 服务器。IIS 是目前最流行的 Web 服务器产品之一，很多著名的网站都是建立在 IIS 的平台上。IIS 提供了一个图形界面的管理工具，称为 Internet 服务管理器，可用于监视配置和控制 Internet 服务。IIS 是一种 Web 服务组件，其中包括 Web 服务器、FTP 服务器、NNTP 服务器和 SMTP 服务器，分别用于网页浏览、文件传输、新闻服务和邮件发送等方面，它使得在网络（包括互联网和局域网）上发布信息成了一件很容易的事。它提供 ISAPI(Intranet Server API）作为扩展 Web 服务器功能的编程接口；同时，它还提供一个 Internet 数据库连接器，可以实现对数据库的查询和更新。

Kangle：Kangle Web 服务器是一款跨平台、功能强大、安全稳定、易操作

的高性能 Web 服务器和反向代理服务器软件。此外，Kangle 也是一款专为做虚拟主机研发的 Web 服务器。实现虚拟主机独立进程、独立身份运行。用户之间安全隔离，一个用户出问题不影响其他用户。支持 PHP、ASP、ASP.NET、Java、Ruby 等多种动态开发语言。

WebSphere：WebSphere Application Server 是功能完善、开放的 Web 应用程序服务器，是 IBM 电子商务计划的核心部分，它是基于 Java 的应用环境，用于建立、部署和管理 Internet 和 Intranet Web 应用程序，适应各种 Web 应用程序服务器的需要，范围从简单到高级直到企业级。

WebLogic：BEA WebLogic Server 是一种多功能、基于标准的 Web 应用服务器，为企业构建自己的应用提供了坚实的基础。各种应用开发、部署所有关键性的任务，无论是集成各种系统和数据库，还是提交服务、跨 Internet 协作，Weblogic 都提供了相应的支持。由于它具有全面的功能、对开放标准的遵从性、多层架构、支持基于组件的开发，基于 Internet 的企业都选择它来开发、部署最佳的应用。BEA WebLogic Server 在使应用服务器成为企业应用架构的基础方面一直处于领先地位，为构建集成化的企业级应用提供了稳固的基础，它们以 Internet 的容量和速度，在连网的企业之间共享信息、提交服务，实现协作自动化。

Apache：目前 Apache 仍然是世界上用得最多的 Web 服务器，市场占有率约为 60%左右。世界上很多著名的网站都是 Apache 的产物，它的成功之处主要在于它的源代码开放、有一支强大的开发团队、支持跨平台的应用（可以运行在几乎所有的 Unix、Windows、Linux 系统平台上）以及它的可移植性等方面。

Tomcat：Tomcat 是一个开放源代码、运行 Servlet 和 JSP 的容器。TomcatServer 实现了 Servlet 和 JSP 规范。此外，Tomcat 还实现了 Apache-Jakarta 规范而且比绝大多数商业应用软件服务器要好，因此目前也有不少的 Web 服务器都选择了 Tomcat。

Nginx：读作"engine x"，是一个高性能的 HTTP 和反向代理服务器，也是一个 IMAP/POP3/SMTP 代理服务器。 Nginx 是由 Igor Sysoev 为俄罗斯访问量第二的 Rambler.ru 站点开发的，第一个公开版本 0.1.0 发布于 2004 年 10 月 4 日。其将源代码以类 BSD 许可证的形式发布，因它的稳定性、丰富的功能集、示例配置文件和低系统资源的消耗而闻名。

551. JSP 和 Servlet 有有什么关系？

答：其实这个问题在上面已经阐述过了，Servlet 是一个特殊的 Java 程序，它运行于服务器的 JVM 中，能够依靠服务器的支持向浏览器提供显示内容。JSP 本质上是 Servlet 的一种简易形式， JSP 会被服务器处理成一个类似于 Servlet 的 Java 程序，可以简化页面内容的生成。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 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（有人说，Servlet 就是在 Java 中写 HTML，而 JSP 就是在 HTML 中写 Java 代码，当然，这个说法还是很片面的）。JSP 侧重于视图，Servlet 更侧重于控制逻辑，在 MVC 架构模式中，JSP 适合充当视图（view）而 Servlet 适合充当控制器（controller）。

552. JSP 中的四种作用域？

答：page、request、session 和 application，具体如下：

①page 代表与一个页面相关的对象和属性。

②request 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；需要在页面显示的临时数据可以置于此作用域

③session 代表与某个用户与服务器建立的一次会话相关的对象和属性。跟某个用户相关的数据应该放在用户自己的 session 中

④application 代表与整个 Web 应用程序相关的对象和属性，它实质上是跨越整个 Web 应用程序，包括多个页面、请求和会话的一个全局作用域。

553. 如何实现 JSP 或 Servlet 的单线程模式？

```
<%@page isThreadSafe=”false”%>
```

【补充】Servlet 默认的工作模式是单实例多线程，如果 Servlet 实现了标识接口 SingleThreadModel 又或是 JSP 页面通过 page 指令设置 isThreadSafe 属性为 false，那么它们生成的 Java 代码会以单线程多实例方式工作。显然，这样做会导致每个请求创建一个 Servlet 实例，这种实践将导致严重的性能问题。

554. 实现会话跟踪的技术有哪些？

答：由于 HTTP 协议本身是无状态的，服务器为了区分不同的用户，就需要对用户会话进行跟踪，简单的说就是为用户进行登记，为用户分配唯一的 ID，下一次用户在请求中包含此 ID，服务器据此判断到底是哪一个用户。

①URL 重写：在 URL 中添加用户会话的信息作为请求的参数，或者将唯一的会话 ID 添加到 URL 结尾以标识一个会话。

②设置表单隐藏域：将和会话跟踪相关的字段添加到隐式表单域中，这些信

息不会在浏览器中显示但是提交表单时会提交给服务器。

这两种方式很难处理跨越多个页面的信息传递，因为如果每次都要修改 URL 或在页面中添加隐式表单域来存储用户会话相关信息，事情将变得非常麻烦。

③cookie：cookie 有两种，一种是基于窗口的，浏览器窗口关闭后，cookie 就没有了；另一种是将信息存储在一个临时文件中，并设置存在的时间。当用户通过浏览器和服务器建立一次会话后，会话 ID 就会随响应信息返回存储在基于窗口的 cookie 中，那就意味着只要浏览器没有关闭，会话没有超时，下一次请求时这个会话 ID 又会提交给服务器让服务器识别用户身份。会话中可以为用户保存信息。会话对象是在服务器内存中的，而基于窗口的 cookie 是在客户端内存中的。如果浏览器禁用了 cookie，那么就需要通过下面两种方式进行会话跟踪。当然，在使用 cookie 时要注意几点：首先不要在 cookie 中存放敏感信息；其次 cookie 存储的数据量有限（4k），不能将过多的内容存储 cookie 中；再者浏览器通常只允许一个站点最多存放 20 个 cookie。当然，和用户会话相关的其他信息（除了会话 ID）也可以存在 cookie 方便进行会话跟踪。

④HttpSession：在所有会话跟踪技术中，HttpSession 对象是最强大也是功能最多的。当一个用户第一次访问某个网站时会自动创建 HttpSession，每个用户可以访问他自己的 HttpSession。可以通过 HttpServletRequest 对象的 getSession 方法获得 HttpSession，通过 HttpSession 的 setAttribute 方法可以将一个值放在 HttpSession 中，通过调用 HttpSession 对象的 getAttribute 方法，同时传入属性名就可以获取保存在

HttpSession 中的对象。与上面三种方式不同的是，HttpSession 放在服务器的内存中，因此不要将过大的对象放在里面，即使目前的 Servlet 容器可以在内存将满时将 HttpSession 中的对象移到其他存储设备中 ,但是这样势必影响性能。添加到 HttpSession 中的值可以是任意 Java 对象，这个对象最好实现了 Serializable 接口，这样 Servlet 容器在必要的时候可以将其序列化到文件中，否则在序列化时就会出现异常。

555. 过滤器有哪些作用和用法？

答： Java Web 开发中的过滤器（filter）是从 Servlet 2.3 规范开始增加的功能，并在 Servlet 2.4 规范中得到增强。对 Web 应用来说，过滤器是一个驻留在服务器端的 Web 组件，它可以截取客户端和服务器之间的请求与响应信息，并对这些信息进行过滤。当 Web 容器接受到一个对资源的请求时，它将判断是否有过滤器与这个资源相关联。如果有 ,那么容器将把请求交给过滤器进行处理。在过滤器中，你可以改变请求的内容，或者重新设置请求的报头信息，然后再将请求发送给目标资源。当目标资源对请求作出响应时候，容器同样会将响应先转发给过滤器，再过滤器中，你可以对响应的内容进行转换，然后再将响应发送到客户端。

常见的过滤器用途主要包括：对用户请求进行统一认证、对用户的访问请求进行记录和审核、对用户发送的数据进行过滤或替换、转换图象格式、对响应内容进行压缩以减少传输量、对请求或响应进行加解密处理、触发资源访问事件、对 XML 的输出应用 XSLT 等。

和过滤器相关的接口主要有：Filter、FilterConfig、FilterChain

556. 监听器有哪些作用和用法？

答：Java Web 开发中的监听器（listener）就是 application、session、request 三个对象创建、销毁或者往其中添加修改删除属性时自动执行代码的功能组件，如下所示：

①ServletContextListener：对 Servlet 上下文的创建和销毁进行监听。

②ServletContextAttributeListener：监听 Servlet 上下文属性的添加、删除和替换。

③HttpSessionListener：对 Session 的创建和销毁进行监听。

补充：session 的销毁有两种情况：1session 超时（可以在 web.xml 中通过<session-config>/<session-timeout>标签配置超时时间）;2 通过调用 session 对象的 invalidate()方法使 session 失效。

④HttpSessionAttributeListener：对 Session 对象中属性的添加、删除和替换进行监听。

⑤ServletRequestListener：对请求对象的初始化和销毁进行监听。

⑥ServletRequestAttributeListener：对请求对象属性的添加、删除和替换进行监听。

557. 你的项目中使用过哪些 JSTL 标签？

答：项目中主要使用了 JSTL 的核心标签库，包括<c:if>、<c:choose>、<c: when>、<c: otherwise>、<c:forEach>等，主要用于构造循环和分支结构以控制显示逻辑。

【说明】虽然 JSTL 标签库提供了 core、sql、fmt、xml 等标签库，但是实际开发中建议只使用核心标签库（core），而且最好只使用分支和循环标签

并辅以表达式语言（EL），这样才能真正做到数据显示和业务逻辑的分离，这才是最佳实践。

558. 使用标签库有什么好处？如何自定义 JSP 标签？

答：使用标签库的好处包括以下几个方面：

分离 JSP 页面的内容和逻辑，简化了 Web 开发；

开发者可以创建自定义标签来封装业务逻辑和显示逻辑；

标签具有很好的可移植性、可维护性和可重用性；

避免了对 Scriptlet（小脚本）的使用（很多公司的项目开发都不允许在 JSP 中书写小脚本）

自定义 JSP 标签包括以下几个步骤：

编写一个 Java 类实现实现 Tag/BodyTag/IterationTag 接口(通常不直接实现这些接口而是继承 TagSupport/BodyTagSupport/SimpleTagSupport 类，这是对适配器模式中缺省适配模式的应用)

重写 doStartTag()、doEndTag()等方法，定义标签要完成的功能

编写扩展名为 tld 的标签描述文件对自定义标签进行部署，tld 文件通常放在 WEB-INF 文件夹或其子目录

在 JSP 页面中使用 taglib 指令引用该标签库

下面是一个例子：

```
package com.bjsxt;

package com.lovo.tags;



import java.io.IOException;
```

```
import java.text.SimpleDateFormat;
import java.util.Date;

import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.TagSupport;

public class TimeTag extends TagSupport {
    private static final long serialVersionUID = 1L;

    private String format = "yyyy-MM-dd hh:mm:ss";
    private String foreColor = "black";
    private String backColor = "white";

    public int doStartTag() throws JspException {
        SimpleDateFormat sdf = new SimpleDateFormat(format);
        JspWriter writer = pageContext.getOut();
        StringBuilder sb = new StringBuilder();
        sb.append(String.format("<span
style='color:%s;background-color:%s'>%s</span>",
            foreColor, backColor, sdf.format(new Date())));
        try {
```

```
                writer.print(sb.toString());
            } catch(IOException e) {
                e.printStackTrace();
            }
            return SKIP_BODY;
        }

    public void setFormat(String format) {
        this.format = format;
    }

    public void setForeColor(String foreColor) {
        this.foreColor = foreColor;
    }

    public void setBackColor(String backColor) {
        this.backColor = backColor;
    }
}
```

标签库描述文件（该文件通常放在 WEB-INF 目录或其子目录下）

```
<?xml version="1.0" encoding="UTF-8" ?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd"
    version="2.0">

    <description>定义标签库</description>
    <tlib-version>1.0</tlib-version>
    <short-name>MyTag</short-name>
    <tag>
        <name>time</name>
        <tag-class>com.lovo.tags.TimeTag</tag-class>
        <body-content>empty</body-content>
        <attribute>
            <name>format</name>
            <required>false</required>
        </attribute>
        <attribute>
            <name>foreColor</name>
        </attribute>
```

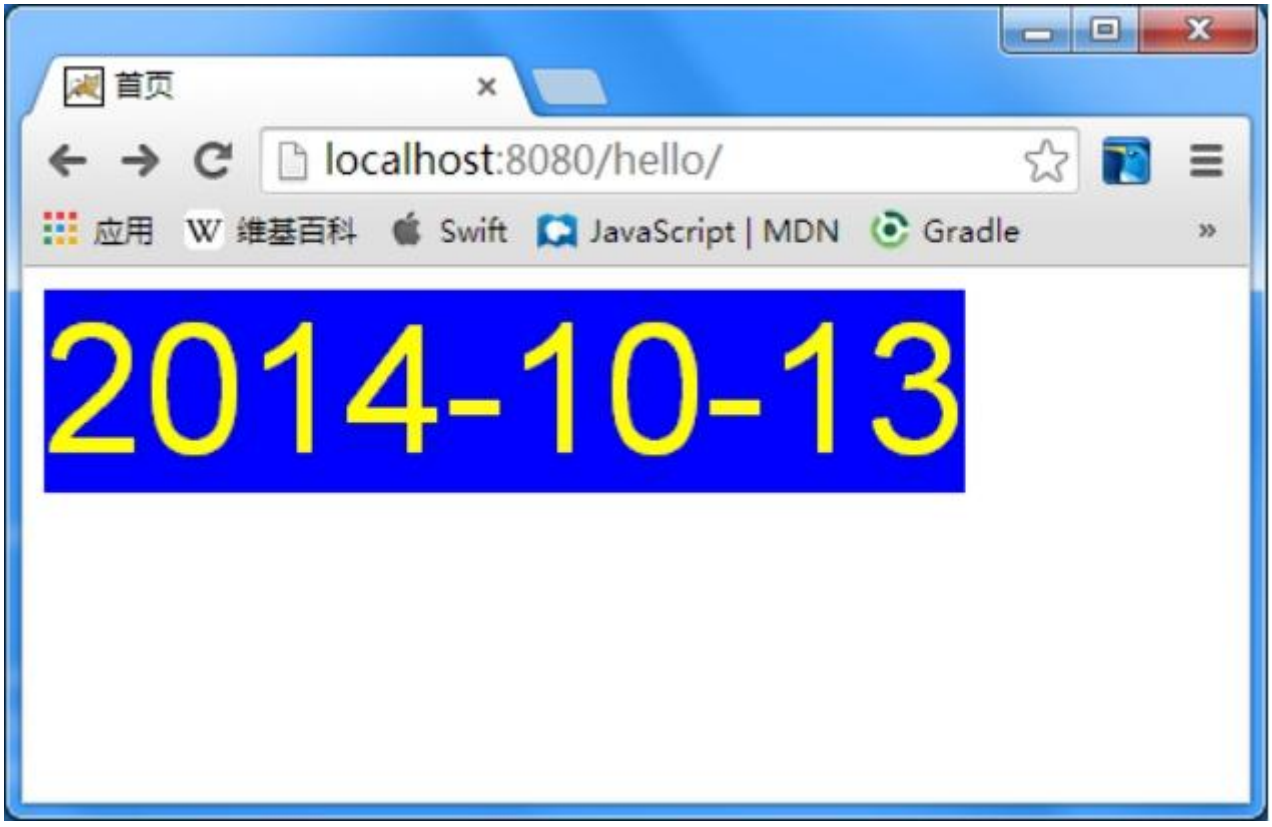
```
        <attribute>
            <name>backColor</name>
        </attribute>
    </tag>
</taglib>
```

JSP 页面

```
<%@ page pageEncoding="UTF-8"%>
<%@ taglib prefix="my" uri="/WEB-INF/tld/my.tld" %>
<%
String path = request.getContextPath();
String basePath =
request.getScheme()+"://"+request.getServerName()+":"+request.getSe
rverPort()+path+"/";
%>

<!DOCTYPE html>
<html>
  <head>
    <base href="<%=basePath%>">
    <title>首页</title>
    <style type="text/css">
```

```
            * { font-family: "Arial"; font-size:72px; }
        </style>
    </head>


    <body>
        <my:time format="yyyy-MM-dd" backColor="blue"
foreColor="yellow"/>
    </body>
</html>
```

运行结果



【注意】如果要将自定义的标签库发布成 JAR 文件，需要将标签库描述文件

（tld 文件）放在 JAR 文件的 META-INF 目录下，可以 JDK 自带的 jar 工具完成 JAR 文件的生成。

559. 表达式语言（EL）的隐式对象及其作用？

答：pageContext、initParam（访问上下文参数）、param（访问请求参数）、paramValues、header（访问请求头）、headerValues、cookie（访问 cookie）、applicationScope（访问 application 作用域）、sessionScope（访问 session 作用域）、requestScope（访问 request 作用域）、pageScope（访问 page 作用域）。用法如下所示：

${pageContext.request.method}

${pageContext["request"]["method"]}

${pageContext.request["method"]}

${pageContext["request"].method}

${initParam.defaultEncoding}

${header["accept-language"]}

${headerValues["accept-language"][0]}

${cookie.jsessionid.value}

${sessionScope.loginUser.username}

【补充】表达式语言的.和[]运算作用是一致的，唯一的差别在于如果访问的属性名不符合 Java 标识符命名规则，例如上面的 accept-language 就不是一个有效的 Java 标识符，那么这时候就只能用[]运算符而不能使用.获取它的值

560. 表达式语言（EL）支持哪些运算符？

答：除了.和[]运算符，EL 还提供了：

算术运算符：+、-、*、/或 div、%或 mod

关系运算符：==或 eq、!=或 ne、>或 gt、>=或 ge、<或 lt、<=或 le

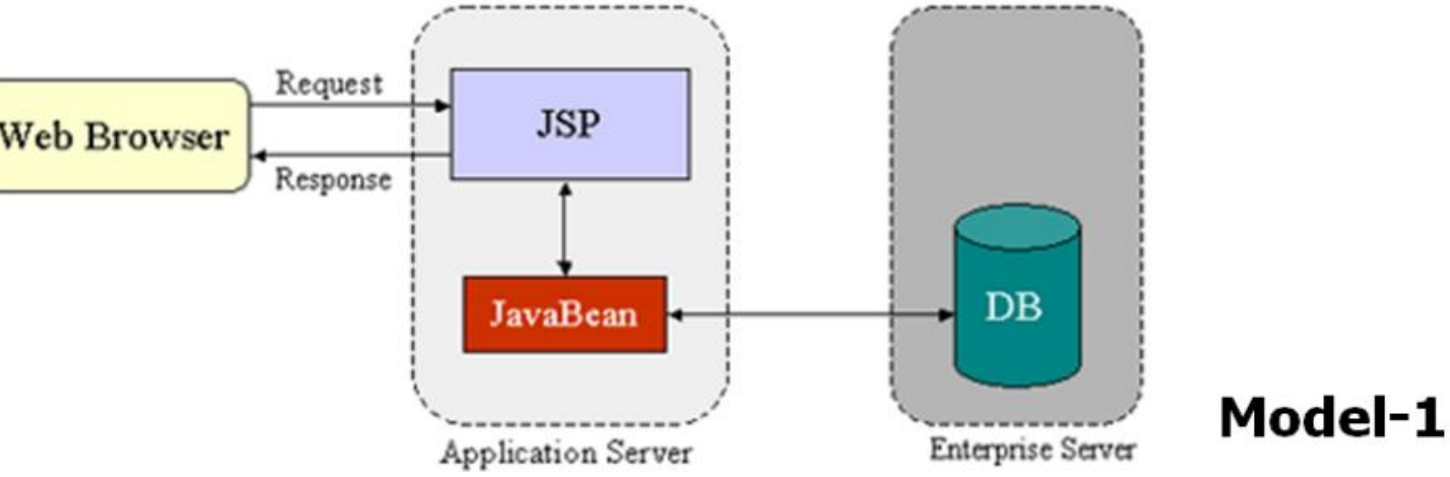
逻辑运算符：&&或 and、||或 or、!或 not

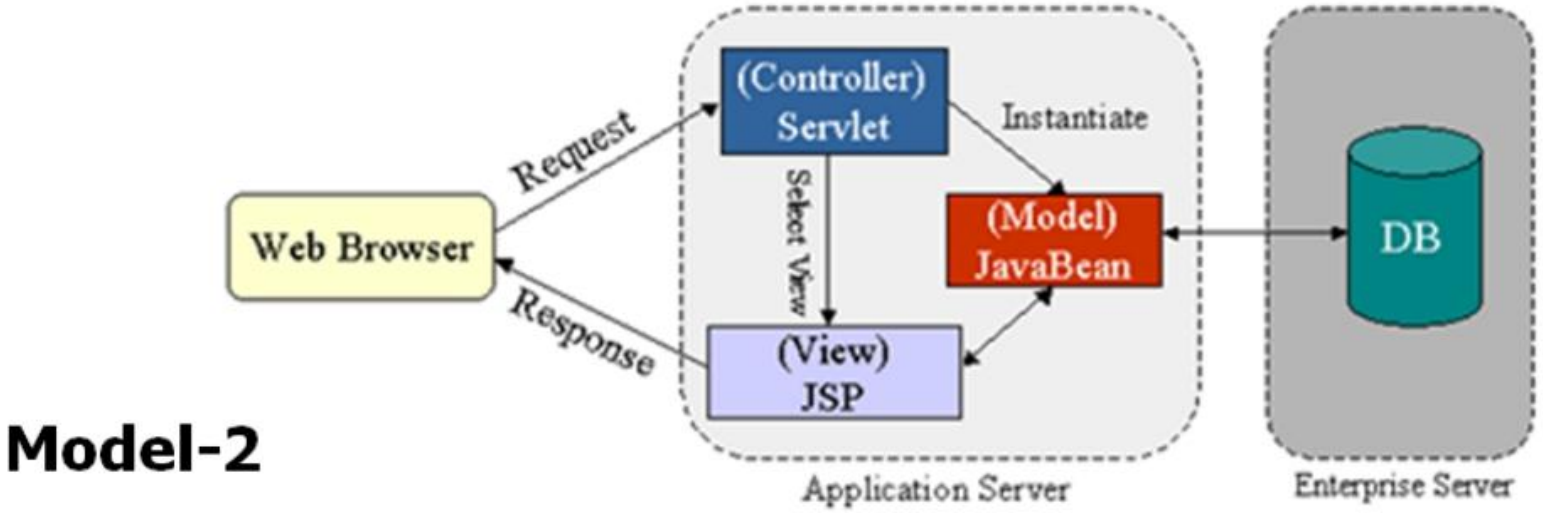
条件运算符：${statement? A : B}（跟 Java 的条件运算符类似）

empty 运算符：检查一个值是否为 null 或者空（数组长度为 0 或集合中没有元素也返回 true）

561. Java Web 开发的 Model 1 和 Model 2 分别指的是什么？

答：Model 1 是以页面为中心的 Java Web 开发，只适合非常小型的应用程序，Model 2 是基于 MVC 架构模式的应用，这一点在前文的面试题中已经详细讲解过了。





562. Servlet 3 中的异步处理指的是什么？

答：在 Servlet 3 中引入了一项新的技术可以让 Servlet 异步处理请求。有人可能会质疑，既然都有多线程了，还需要异步处理请求吗？答案是肯定的，因为如果一个任务处理时间相当长，那么 Servlet 或 Filter 会一直占用着请求处理线程直到任务结束，随着并发用户的增加，容器将会遭遇线程超出的风险，这这种情况下很多的请求将会被堆积起来而后续的请求可能会遭遇拒绝服务，直到有资源可以处理请求为止。异步特性可以帮助应用节省容器中的线程，特别适合执行时间长而且用户需要得到结果的任务，如果用户不需要得到结果则直接将一个 Runnable 对象交给 Executor(如果不清楚请查看前文关于多线程和线程池的部分）并立即返回即可。

【补充】多线程在 Java 诞生初期无疑是一个亮点，而 Servlet 单实例多线程的工作方式也曾为其赢得美名，然而技术的发展往往会颠覆我们很多的认知，就如同当年爱因斯坦的相对论颠覆了牛顿的经典力学一般。事实上，异步处理绝不是 Serlvet 3 首创，如果你了解 Node.js 的话，对 Servlet 3 的这个重

要改进就不以为奇了。

563. 如何在基于 Java 的 Web 项目中实现文件上传和下载？

答：（稍后呈现，我准备用 HTML5 写一个带进度条的客户端，然后再用 Servlet 3 提供的文件上传支持来做一个多文件上传的例子）

564. 简述值栈(Value-Stack)的原理和生命周期

答：Value-Stack 贯穿整个 Action 的生命周期，保存在 request 作用域中，所以它和 request 的生命周期一样。当 Struts 2 接受一个请求时，会创建 ActionContext、Value-Stack 和 Action 对象，然后把 Action 存放进 Value-Stack，所以 Action 的实例变量可以通过 OGNL 访问。由于 Action 是多实例的，和使用单例的 Servlet 不同， 每个 Action 都有一个对应的 Value-Stack，Value-Stack 存放的数据类型是该 Action 的实例，以及该 Action 中的实例变量，Action 对象默认保存在栈顶。

565. SessionFactory 是线程安全的吗？Session 是线程安全的吗，两个线程能够共享同一个 Session 吗？

答：SessionFactory 对应 Hibernate 的一个数据存储的概念，它是线程安全的，可以被多个线程并发访问。SessionFactory 一般只会在启动的时候构建。对于应用程序，最好将 SessionFactory 通过单例的模式进行封装以便于访问。Session 是一个轻量级非线程安全的对象（线程间不能共享 session），它表示与数据库进行交互的一个工作单元。Session 是由 SessionFactory 创建的，在任务完成之后它会被关闭。Session 是持久层服务对外提供的主要接口。Session 会延迟获取数据库连接（也就是在需要的时候才会获取）。为了避免创建太多的 session，可以使用 ThreadLocal 来

取得当前的 session，无论你调用多少次 getCurrentSession()方法，返回的都是同一个 session。

566. Session 的 load 和 get 方法的区别是什么？

答：主要有以下三项区别：

① 如果没有找到符合条件的记录, get 方法返回 null,load 方法抛出异常

②get 方法直接返回实体类对象, load 方法返回实体类对象的代理

③ 在 Hibernate 3 之前 ,get 方法只在一级缓存(内部缓存)中进行数据查找, 如果没有找到对应的数据则越过二级缓存, 直接发出 SQL 语句完成数据读取; load 方法则可以充分利用二级缓存中的现有数据；当然从 Hibernate 3 开始，get 方法不再是对二级缓存只写不读，它也是可以访问二级缓存的

简单的说 ,对于 load()方法 Hibernate 认为该数据在数据库中一定存在可以放心的使用代理来实现延迟加载，如果没有数据就抛出异常，而通过 get()方法去取的数据可以不存在。

567. Session 的 save()、update()、merge()、lock()、saveOrUpdate()和 persist()方法有什么区别？

答：Hibernate 的对象有三种状态：瞬态、持久态和游离态。游离状态的实例可以通过调用 save()、persist()或者 saveOrUpdate()方法进行持久化；脱管状态的实例可以通过调用 update()、0saveOrUpdate()、lock()或者 replicate()进行持久化。save()和 persist()将会引发 SQL 的 INSERT 语句，而 update()或 merge()会引发 UPDATE 语句。save()和 update()的区别在于一个是将瞬态对象变成持久态，一个是将游离态对象变为持久态。merge 方法可以完成 save()和 update()方法的功能 ,它的意图是将新的状态合并到

已有的持久化对象上或创建新的持久化对象。按照官方文档的说明：(1)persist()方法把一个瞬态的实例持久化，但是并"不保证"标识符被立刻填入到持久化实例中，标识符的填入可能被推迟到 flush 的时间；(2) persist"保证"，当它在一个事务外部被调用的时候并不触发一个 Insert 语句，当需要封装一个长会话流程的时候，一个 persist 这样的函数是需要的。(3)save"不保证"第 2 条,它要返回标识符，所以它会立即执行 Insert 语句，不管是不是在事务内部还是外部。update()方法是把一个已经更改过的脱管状态的对象变成持久状态；lock()方法是把一个没有更改过的脱管状态的对象变成持久状态。

568. 阐述 Session 加载实体对象的过程。

答：Session 加载实体对象的步骤是：

① Session 在调用数据库查询功能之前，首先会在缓存中进行查询，在一级缓存中，通过实体类型和主键进行查找，如果一级缓存查找命中且数据状态合法，则直接返回

③ 如果一级缓存没有命中，接下来 Session 会在当前 NonExists 记录(相当于一个查询黑名单，如果出现重复的无效查询可以迅速判断，从而提升性能)中进行查找，如果 NonExists 中存在同样的查询条件,则返回 null

③ 对于 load 方法，如果一级缓存查询失败则查询二级缓存，如果二级缓存命中则直接返回

④ 如果之前的查询都未命中，则发出 SQL 语句，如果查询未发现对应记录则将此次查询添加到 Session 的 NonExists 中加以记录，并返回 null

⑤ 根据映射配置和 SQL 语句得到 ResultSet,并创建对应的实体对象

⑥ 将对象纳入 Session(一级缓存)管理

⑦ 执行拦截器的 onLoad 方法(如果有对应的拦截器)

⑧将数据对象纳入二级缓存

⑨返回数据对象

569. 给了一大堆 SQL 语句。问这个 SQL 怎么优化执行效率更高

1. SQL 优化的原则是 :将一次操作需要读取的 BLOCK 数减到最低,即在最短的时间达到最大的数据吞吐量。

调整不良 SQL 通常可以从以下几点切入：

1).检查不良的 SQL，考虑其写法是否还有可优化内容

2).检查子查询 考虑 SQL 子查询是否可以用简单连接的方式进行重新书写

3).检查优化索引的使用

4).考虑数据库的优化器

2. 避免出现 SELECT * FROM table 语句，要明确查出的字段。

3. 在一个 SQL 语句中，如果一个 where 条件过滤的数据库记录越多，定位越准确，则该 where 条件越应该前移。

4. 查询时尽可能使用索引覆盖。即对 SELECT 的字段建立复合索引，这样查询时只进行索引扫描，不读取数据块。

5. 在判断有无符合条件的记录时建议不要用 SELECT COUNT（ *)和 select top 1 语句。

6. 使用内层限定原则，在拼写 SQL 语句时，将查询条件分解、分类，并尽量在 SQL 语句的最里层进行限定，以减少数据的处理量。

7. 应绝对避免在 order by 子句中使用表达式。

8. 如果需要从关联表读数据，关联的表一般不要超过 7 个。

9. 小心使用 IN 和 OR，需要注意 In 集合中的数据量。建议集合中的数据不超过 200 个。

10. <> 用 < 、 > 代替，>用>=代替，<用<=代替，这样可以有效的利用索引。

11. 在查询时尽量减少对多余数据的读取包括多余的列与多余的行。

12. 对于复合索引要注意，例如在建立复合索引时列的顺序是 F1，F2，F3，则在 where 或 order by 子句中这些字段出现的顺序要与建立索引时的字段顺序一致，且必须包含第一列。只能是 F1 或 F1，F2 或 F1，F2，F3。否则不会用到该索引。

570. 怎么防止重复提交

1.禁掉提交按钮。表单提交后使用 Javascript 使提交按钮 disable。这种方法防止心急的用户多次点击按钮。但有个问题，如果客户端把 Javascript 给禁止掉，这种方法就无效了。

2.Post/Redirect/Get 模式。在提交后执行页面重定向，这就是所谓的 Post-Redirect-Get (PRG)模式。简言之，当用户提交了表单后，你去执行一个客户端的重定向，转到提交成功信息页面。

这能避免用户按 F5 导致的重复提交，而其也不会出现浏览器表单重复提交的警告，也能消除按浏览器前进和后退按导致的同样问题。

3.在 session 中存放一个特殊标志。当表单页面被请求时，生成一个特殊的字符标志串，存在 session 中，同时放在表单的隐藏域里。接受处理表单数据时，检查标识字串是否存在，并立即从 session 中删除它，然后正常处理

数据。

如果发现表单提交里没有有效的标志串，这说明表单已经被提交过了，忽略这次提交。

4.在数据库里添加约束。在数据库里添加唯一约束或创建唯一索引，防止出现重复数据。这是最有效的防止重复提交数据的方法。

571. 数据库去空格

一、表中字符串带空格的原因

1、空格就是空格。

2、控制符 显示为 空格。

二、解决方法

第一种情况，去空格的处理的比较简单，Replace(column,' ','') 就可以解决。

第二种情况，解决方法就比较麻烦点：需要先查出相应的 ASCII 码，再用 Replace(column,char(ascii 码),'')解决，以下举个栗子：

```
CREATE TABLE #temp
(NAME NVARCHAR(50))

INSERT INTO #temp SELECT '明天就是国庆了'+CHAR(10)  --换行符

SELECT * FROM #temp                                  --末尾显示为空格
SELECT REPLACE(NAME,' ','') FROM #temp     --去不掉这个空格
SELECT REPLACE(NAME,CHAR(10),'') FROM #temp   --去掉空格
```

SELECT REPLACE(NAME,CHAR(ASCII(RIGHT(NAME,1))),'') FROM #temp --在不知道是最后一位是什么字符导致空格的情况下 ,先转 ASCII 码，在替换

DROP TABLE #temp

----下面是查询结果:

--'明天就是国庆了 '

--'明天就是国庆了 '

--'明天就是国庆了'

--'明天就是国庆了'

572. 根据你以往的经验简单叙述一下 MYSQL 的优化

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记录,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 indert 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.

573. 以 Oracle11R 为例简述数据库集群部署

命令行工具

–crsctl 管理集群相关的操作：

-启动和关闭 Oracle 集群

-启用和禁用 Oracle 集群后台进程

-注册集群资源

-srvctl 管理 Oracle 资源相关操作

-启动和关闭数据库实例和服务

在 Oracle Grid 安装的 home 路径下的命令行工具 crsctl 和 srvctl 用来管理 Oracle 集群。使用 crsctl 可以监控和管理任何集群节点的集群组件和资源。srvctl 工具提供了类似的功能，来监控和管理 Oracle 相关的资源，例如数据库实例和数据库服务。crsctl 命令只能是集群管理者来运行，srvctl 命令可以是其他用户，例如数据库管理员来使用。

574. 说一下数据库的存储过程？

一、存储过程与函数的区别：

1.一般来说，存储过程实现的功能要复杂一点，而函数的实现的功能针对性比较强。

2.对于存储过程来说可以返回参数(output)，而函数只能返回值或者表对象。

3.存储过程一般是作为一个独立的部分来执行，而函数可以作为查询语句的一个部分来调用，由于函数可以返回一个表对象，因此它可以在查询语句中位于 FROM 关键字的后面。

二、存储过程的优点：

1.执行速度更快 – 在数据库中保存的存储过程语句都是编译过的

2.允许模块化程序设计 – 类似方法的复用

3.提高系统安全性 – 防止 SQL 注入

4.减少网络流通量 – 只要传输存储过程的名称

系统存储过程一般以 sp 开头，用户自定义的存储过程一般以 usp 开头

575. 数据库创建索引的缺点？

缺点：

第一，创建索引和维护索引要耗费时间，这种时间随着数据量的增加而增加。

第二，索引需要占物理空间，除了数据表占数据空间之外，每一个索引还要占一定的物理空间，如果要建立聚簇索引，那么需要的空间就会更大。

第三，当对表中的数据进行增加、删除和修改的时候，索引也要动态的维护，这样就降低了数据的维护速度。

576. 有两张表；请用 SQL 查询，所有的客户订单日期最新的前五条订单记录。（分别注明 MySQL. Oracle 写法）

客户信息表(c CUSTOM)有以下字段：

id、name、mobile

客户订单表(C_ORDER)有以下字段：

id、custom_id、commodity、count、order _date

Mysql:

```
Select * from c_order order by order_date desc limit 0,5;
```

Oracle:

```
Select o.*,rownum n
    from c_order order by order_date desc   where n<6;
```

577. 数据库表 t1 内容如下

id	name	age
1	张三	20
2	李四	21
3	王五	22

请用标准的 json 格式描述 ,并用 Java 或者 JavaScript 写出构建。读取此 json 中的代码

```
var p1 = [{
```

```
    Id:1,
    Name:’张三’,
    Age:20
},{
    Id:2,
    Name:’李四’,
    Age:21
},{
    Id:3,
    Name:’王五’,
    Age:22
}]
```

578. 关于 HQL 与 SQL,以下哪些说法正确？(BC)

A.	HQL与SQL没什么差别
B.	HQL 面向对象，而 SQL 操纵关系数据库
C.	在 HQL 与 SQL 中，都包含 select,insert,update,delete 语句
D.	HQL 仅用于查询数据，不支持 insert,update 和 delete 语句

579. T 表: (字段：ID. NAME. ADDRESS. PHONE，LOGDATE)

E 表：(字段: NAME, ADDRESS，PHONE)

a.将表 T 中的字段 LOGDATE 为 2014-02-11 的数据更新为 2015-01-01，请写出相应的 sql 语句. (该字段类型为日期类型）

Update T set logdate=to_date（‘2015-01-01’,’ yyyy-mm-dd’）;

b.请写出将 T 表中第 301-350 行数据列出来的 SQL 语句(按照 NAME 排序)

Select * from T order by name limit 300,50;

c.请写出将 E 表中的 ADDRESS、PHONE 更新到 T 表中的 SQL 语句（按 NAME 相同进行关联>

Update T,E set T.ADDRESS=E.ADDRESS,T.PHONE=E.PHONE where T.NAME = E.NAME;

d.请写出将表 T 中 NAME 存在重复的记录都列出来的 SQL 语句（按 NAME 排序）

```
SELECT T1.*
FROM    T   T1
WHERE   1=1
AND     1< (SELECT COUNT(*)
            FROM    T T2
            WHERE   T2.ID=T1.ID)
ORDER BY T1.NAME;
```

e.请写出上一个语句中，只保留重复记录的第一条，删除其余记录的 SQL 语句（即使该表不存在重复记录）

```
DELETE T
WHERE   ROWID NOT IN (SELECT MIN(ROWID) FROM T GROUP BY NAME);
```

580. 请分别画出添加新操作员和修改操作员信息的程序流程图，并写出程序中应用的 SQL，先添加新用户（操作员标识为：12，登录名：text，登录口令为：12345678），然后再将该用户的登录名修改为：tester。

操作员信息储存在 operator（主键：operator_id,附键为：login_name[即登录名不可重复]），如下表所示：

字段名	字段名	类型	长度	是否可为空	是否为主键	取值是否唯一
操作员标识	Operator_id	int		N	Y	Y
登录名	Login_name	varchar	32	N	N	Y
登录口令	Logina_password	varchar	64	N	N	
…						

SQL 语句：

```
Insert into operator values(12,’text’,’12345678’);
Commit;
Update operator set login_name=’tester’ where operator_id = 12;
Commit;
```

581. 下面是学生表（student）的结构说明

字段名称	字段解释	字段类型	字 段 长	约束

			度	
s_id	学号	字符	10	PK
s_name	学生姓名	字符	50	Not null
s_age	学生年龄	数值	3	Not null
s-sex	学生性别	字符(男:1 女:0)	1	Not null

下面是教师表（Teacher ）的结构说明

字段名称	字段解释	字段类型	字段长度	约束
t_id	教师编号	字符	10	PK
t_name	教师名字	字符	50	Not null

下面是课程表（Course）的结构说明

字段名称	字段解释	字段类型	字段长度	约束
c_id	课程编号	字符	10	Pk
c_name	课程名字	字符	50	Not null
t_id	教师编号	字符	10	Not null

下面是成绩表（SC）的结构说明

字段名称	字段解释	字段类型	字段长度	约束
s_id	学号	字符	10	Pk
c_id	课程编号	字符	10	Pk
score	成绩	数值	3	Not null

查询同名同学学生名单，并统计同名人数；

Select count(s_name) from student group by s_name;

查询平均成绩大于 6 分的学生的学号和平均成绩；

Select s_id,avg(score) from sc group by s_id having avg(score)>6;

582. 为管理岗位业务培训信息，有如下 3 个表：

S（S#,SN,SD,SA)，其中 S#，SN,SD,SA 分别代表学号、学员姓名、所属单位、学员年龄。

C (C#,CN)，其中 C#,CN 分别代表课程编号、课程名称

SC(S#,C#，G)，其中 S#,C#，G 分别代表学号、所选修的课程编号、学习成绩

请使用 2 种标准 SQL 语句查询选修课程名称为“税收基础”的学员学号和姓名，并说明其优缺点 。

SQL92 标准：

```
SELECT SN,SD FROM S
  WHERE [S#] IN(
  SELECT [S#] FROM C,SC
  WHERE C.[C#]=SC.[C#]
  AND CN=N'税收基础')
```

SQL99 标准：

```
select s.s#,s.sn from s
join sc on s.s#=sc.s#
join c on sc.c#=c.c#
```

where c.cn='税收基础'

优点：

SQL99 将连接条件和过滤条件分开，显得代码清晰。

SQL92 书写简单易于理解。

缺点：

SQL92 连接条件和过滤条件都写在一起，不利于查看。

SQL99 书写相对麻烦不易于理解。

583. 用 Java 怎么实现有每天有 1 亿条记录的 DB 储存？MySQL 上亿记录数据量的数据库如何设计？

1.这么大数据量首先建议 使用大数据的 DB，可以用 spring batch 来做类似这样的处理。定量向 DB 存储数据。如果需要定时，可以考虑 quartz。

Mysql 数据库设计:

1.读写分离；

2.纵向横向拆分库、表。

MySQL 的基本功能中包括 replication（复制）功能。所谓 replication，就是确定 master 以及与之同步的 slave 服务器，再加上 slave 将 master 中写入的内容 polling 过来更新自身内容的功能。这样 slave 就是 master 的 replica（复制品）。这样就可以准备多台内容相同的服务器。

通过 master 和 salve 的 replication，准备好多台服务器之后，让应用程序服务器通过负载均衡器去处理查询 slave。这样就能将查询分散到多台服务器上。

应用程序实现上应该只把 select 等读取之类的查询发送给负载均衡器，

而更新应当直接发送给 master。要是在 slave 上执行更新操作，slave 和 master 的内容就无法同步。MySQL 会检测到 master 和 slave 之间内容差异，并停止 replication，这回导致系统故障。Slave 可以采用 LVS（linux 系统自带的负载均衡器）实现查询的负载均衡。

使用 MySQL 的 replication 是利用的冗余化，实现冗余化需要实现的最小服务器数量是 4 台 ,三台 slave 和一台 master ,slave 为什么是需要三台呢，比如一台 slave 死机了，现在需要修复再次上线，那么意味着你必须停止一台 slave 来复制 MySQL 的数据，如果只有两台 slave，一台坏了，你就必须停止服务，如果有三台，坏了一台，你复制数据时停止一台，还有一台可以运维。

对于数据的处理是能放入到内存中就尽量放入到内存中如果不能放入到内存中，可以利用 MySQL 的 Partitioning。

Partitioning 就是表分割也就是讲 A 表和 B 表放在不同的服务器上。简单来说，Partitioning 就是充分利用局部性进行分割，提高缓存利用效率，从而实现 Partitioning 的效果。其中最重要的一点就是以 Partitioning 为前提设计的系统将表分割开，用 RDBMS 的方式的话，对于一对多的关系经常使用 JOIN 查询将两张表连接起来。但是如果将表分割开了之后，也就是两张表不在同一个数据库，不在同一个服务器上怎样使用 JOIN 操作，这里需要注意的是如果是用 where in 操作不是省了一些麻烦了嘛。

584. Mysql 的引擎有哪些？支持事物么？DB 储存引擎有哪些？

MySQL 有多种存储引擎，每种存储引擎有各自的优缺点，可以择优选择使

用：

MyISAM、InnoDB、MERGE、MEMORY(HEAP)、BDB(BerkeleyDB)、EXAMPLE、FEDERATED、ARCHIVE、CSV、BLACKHOLE。

MySQL 支持数个存储引擎作为对不同表的类型的处理器。MySQL 存储引擎包括处理事务安全表的引擎和处理非事务安全表的引擎。

· MyISAM 管理非事务表。它提供高速存储和检索，以及全文搜索能力。MyISAM 在所有 MySQL 配置里被支持，它是默认的存储引擎，除非你配置 MySQL 默认使用另外一个引擎。

· MEMORY 存储引擎提供“内存中”表。MERGE 存储引擎允许集合将被处理同样的 MyISAM 表作为一个单独的表。就像 MyISAM 一样，MEMORY 和 MERGE 存储引擎处理非事务表，这两个引擎也都被默认包含在 MySQL 中。

注释：MEMORY 存储引擎正式地被确定为 HEAP 引擎。

· InnoDB 和 BDB 存储引擎提供事务安全表。BDB 被包含在为支持它的操作系统发布的 MySQL-Max 二进制分发版里。InnoDB 也默认被包括在所 有 MySQL 5.1 二进制分发版里，你可以按照喜好通过配置 MySQL 来允许或禁止任一引擎。

· EXAMPLE 存储引擎是一个“存根”引擎，它不做什么。你可以用这个引擎创建表，但没有数据被存储于其中或从其中检索。这个引擎的目的是服务，在 MySQL 源代码中的一个例子，它演示说明如何开始编写新存储引擎。同样，它的主要兴趣是对开发者。

· NDB Cluster 是被 MySQL Cluster 用来实现分割到多台计算机上的表的

存储引擎。它在 MySQL-Max 5.1 二进制分发版里提供。这个存储引擎当前只被 Linux, Solaris, 和 Mac OS X 支持。在未来的 MySQL 分发版中，我们想要添加其它平台对这个引擎的支持，包括 Windows。

· ARCHIVE 存储引擎被用来无索引地，非常小地覆盖存储的大量数据。

· CSV 存储引擎把数据以逗号分隔的格式存储在文本文件中。

· BLACKHOLE 存储引擎接受但不存储数据，并且检索总是返回一个空集。

· FEDERATED 存储引擎把数据存在远程数据库中。在 MySQL 5.1 中，它只和 MySQL 一起工作，使用 MySQL C Client API。在未来的分发版中，我们想要让它使用其它驱动器或客户端连接方法连接到另外的数据源。

585. 以下是学生考试结果表

fname	kecheng	fenshu
张三	语文	81
张三	数学	65
李四	语文	76
李四	数学	90
王五	语文	61
王五	数学	100
王五	英语	90

1.请用一条 sql 语句从 t_result 表中查询出每门课都大于 75 分的学生姓名；

select b.fname from

(select fname,count(kecheng) c from t_result group by fname)a,

(Select fname,kecheng,count(fname) c from t_result where

fenshu >75 group by fname)b

where a.fname = b.fname and a.c = b.c

2.请用一条 sql 写出总分排名前三的学生姓名，总分，平均分

select fname,sum(fenshu),avg(fenshu) from t_result GROUP By fname order by SUM(fenshu) desc;

586. 下列属于关系型数据库的是（AB）

A.	Oracle
B.	MySql
C.	IMS
D.	MongoDB

587. 库中已经存在雇用表表名：

org_employee;表中字段：雇员编号（emp_id）,雇员姓名（em_name），雇员年龄（emp_age）,雇员部门（depart_name）;请写出执行以下操作的 sql 语句：

1） 向表中增加一条数据：雇员编号（1001），雇员姓名（张三），雇员年龄（24），雇员部门（研发部）;

INSERT INTO org_employee

VALUES(‘1001’,’张三’,’24’,’研发部’);

2） 查询雇员年龄在 55（包含）至 60（不包含）岁之间的雇员数据

SELECT * FROM org_employee

WHERE emp_age>=55 and emp_age <60;

3） 分部门查询各个部门的雇员数量

SELECT COUNT(*),depart_name group by depart_name;

4） 删除姓名为张三的雇员数据

Delete from org_employee where em_name =’张三’；

5） 在表中增加一个日期类型的字段雇员出生日期，字段为 emp_brithday

Alter table org_employee add(emp_brithday date);

6） 将表 org_employee 删除

drop org_employee;

588. body 中的 onload()函数和 jQuery 中的 document.ready()有什么区别？

1.执行时间

window.onload 必须等到页面内包括图片的所有元素加载完毕后才能执行。

$(document).ready()是 DOM 结构绘制完毕后就执行，不必等到加载完毕。

2.编写个数不同

window.onload 不能同时编写多个，如果有多个 window.onload 方法，只会执行一个

$(document).ready()可以同时编写多个，并且都可以得到执行

3.简化写法

window.onload 没有简化写法

$(document).ready(function(){})可以简写成$(function(){});

589. $(document).ready(function(){})jQuery(document).ready(function(){})；有什么区别？

window.jQuery = window.$ = jQuery;

这两者可以互换使用。一般建议优先使用$

590. 写出输出结果

```
<script>
function Foo() {
    getName = function (){alert(1);};
    return this;
}
    Foo.getName = function() {alert (2);};
    Foo.prototype.getName = function (){ alert (3);};
    var getName = function (){alert (4);};
    function getName(){alert (5);}
</script>
```

```
//请写出以下输出结果：
    Foo.getName();   //   2
    getName();   // 4
    Foo().getName();   // 1
    getName(); // 1
    new Foo.getName(); // 2
    new Foo().getName();   // 3
    new new Foo().getName(); // 3
```

591. 如下表 1 中的数据，表名为：t_test,记录某场比赛的结果。

请用 sql 语句实现表 2 的查询结果

ID	matchdate	result
1	2015-02-04	胜
2	2015-02-04	负
3	2015-02-04	胜
4	2015-04-07	负
5	2015-04-07	胜
6	2015-04-07	负

表 1

比赛日期	胜	负
2015-02-04	2	1
2015-04-07	1	2

表 2

SQL 语句：

```
create table t_second(
  matchdate date,
   win varchar(3),
   lose varchar(3)
);
```

insert into t_second (matchdate,win) select matchdate,count(result) from t_test where result ='胜' GROUP BY matchdate;

update t_second,(select matchdate,count(result) as lose from t_test where result ='负' GROUP BY matchdate)s set t_second.lose = s.lose where t_second.matchdate = s.matchdate;

592. 请将如下数据库语句进行优化，使其执行效率更高（提示：...不需要更改）

```
SELECT…
    FROM EMP
    WHERE DEPT_NO NOT IN (SELECT DEPT_NO
                FROM DEPT
                WHERE DEPT_CAT=’A’);
```

优化后：

```
SELECT…
    FROM EMP
    WHERE DEPT_NO NOT EXISTS(SELECT DEPT_NO
                FROM DEPT
                WHERE DEPT_CAT=’A’);
```

593. TCP 为何采用三次握手来建立连接，若采用二次握手可以么，请说明理由

三次握手是为了防止已失效的连接请求再次传送到服务器端。 二次握手不可行，因为：如果由于网络不稳定，虽然客户端以前发送的连接请求已到达服务方，但服务方的同意连接的应答未能到达客户端。则客户方要重新发送连接请求，若采用二次握手，服务方收到重传的请求连接后，会以为是新的请求，就会发送同意连接报文，并新开进程提供服务，这样会造成服务方资源的无谓浪费。

594. web 项目从浏览器发起交易响应缓慢，请简述从哪些方面如数分析

从前端后端分别取考虑，后台是不是数据库死锁等。

前台看看是不是 js 错误，或者图片过大，dom 渲染 dom 树，画面优化。

cmd amd 规范等

595. 描述一下 jvm 加载 class 文件的原理机制？

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

由于 Java 的跨平台性，经过编译的 Java 源程序并不是一个可执行程序，而是一个或多个类文件。当 Java 程序需要使用某个类时，JVM 会确保这个类已经被加载、连接（验证、准备和解析）和初始化。类的加载是指把类的.class 文件中的数据读入到内存中，通常是创建一个字节数组读入.class 文件，然后产生与所加载类对应的 Class 对象。加载完成后，Class 对象还

不完整，所以此时的类还不可用。当类被加载后就进入连接阶段，这一阶段包括验证、准备（为静态变量分配内存并设置默认的初始值）和解析（将符号引用替换为直接引用）三个步骤。最后 JVM 对类进行初始化，包括：1)如果类存在直接的父类并且这个类还没有被初始化，那么就先初始化父类；2)如果类中存在初始化语句，就依次执行这些初始化语句。

类的加载是由类加载器完成的，类加载器包括：根加载器（BootStrap）、扩展加载器（Extension）、系统加载器（System）和用户自定义类加载器（java.lang.ClassLoader 的子类）。 从 Java 2（JDK 1.2）开始，类加载过程采取了父亲委托机制（PDM）。PDM 更好的保证了 Java 平台的安全性，在该机制中，JVM 自带的 Bootstrap 是根加载器，其他的加载器都有且仅有一个父类加载器。类的加载首先请求父类加载器加载，父类加载器无能为力时才由其子类加载器自行加载。JVM 不会向 Java 程序提供对 Bootstrap 的引用。下面是关于几个类加载器的说明：

Bootstrap：一般用本地代码实现，负责加载 JVM 基础核心类库（rt.jar）；

Extension：从 java.ext.dirs 系统属性所指定的目录中加载类库，它的父加载器是 Bootstrap；

System：又叫应用类加载器，其父类是 Extension。它是应用最广泛的类加载器。它从环境变量 classpath 或者系统属性 java.class.path 所指定的目录中记载类，是用户自定义加载器的默认父加载器。

596. 请简述如何将 Oracle 中的数据库转至 DB2 中，需要保证表结构和数据不变

使用 ETL 工具，如 infomatic,datastage,kettle 等，可以完成异构数据库的迁移

以 kettle 为例

表输入选择 oracle 库

表输出选择 DB 库

循环执行可以进行全库迁移

597. 学生成绩表

姓名：name 课程：subject 分数：score 学号：stuid

张三 数学 89 1

张三 语文 80 1

张三 英语 70 1

李四 数学 90 2

李四 语文 70 2

李四 英语 80 2

1.计算每个人的总成绩并排名（要求显示字段：姓名，总成绩）

```
select name,sum(score) s from t_stu GROUP BY name;
```

2.列出各门课程成绩最好的学生（要求显示字段：学号，姓名，科目，成绩）

```
select t1.stuid,t1.name,t1.subject,t1.score from t_stu t1,(
select subject,MAX(score) as maxscore from t_stu group by subject)t2
where t1.subject = t2.subject and t1.score = t2.maxscore;
```

3.列出各个课程的平均成绩（要求显示字段；课程，平均成绩）

```
select subject,AVG(score)平均成绩  from t_stu
group by subject;
```

598. Oracl 数据库中有两张表 Stu（学生表）和 Grade（分数表），如下图所示：

Stu 表

sid（学生 ID）	sname(姓名)	sage（年龄）
1	张三	23
2	李四	25
3	王五	24

Grade 表

gid（分数主键）	cid（课程 ID）	sid(学生主键)	grade（分数）
1	2	3	86
2	2	2	79
3	1	2	80
4	1	1	81
5	1	3	70
6	2	1	78

请写 sql 统计出有两门以上的课的分数在 80 分以上的学生的姓名和年龄？

```
Select sname,sage from Stu where Stu.sid in (
    Select sid from Grade where grade >80
```

)

599. 下面是学生表（Student）的结构说明：

字段名称	字段解释	字段类型	字段长度	约束
s_id	学号	字符	10	PK
s_name	学生姓名	字符	50	Not null
s_age	学生年龄	数值	3	Not null
s_sex	学生性别	字符(男:1 女:0)	1	Not null

下面是教师表（Teacher）

字段名称	字段解释	字段类型	字段长度	约束
t_id	教师编号	字符	10	PK
t_name	教师名字	字符	50	Not null

下面是课程表（Course）的结构说明：

字段名称	字段解释	字段类型	字段长度	约束
c_id	课程编号	字符	10	PK
c_name	课程名字	字符	50	Not null
t_id	教师编号	字符	10	Not null

下面是成绩表（SC）的结构说明：

字段名称	字段解释	字段类型	字段长度	约束
s_id	字符	字符	10	PK
c_id	课程编号	字符	10	PK
score	成绩	数值	3	Not null

查询同名同姓学生名单，并统计同名人数

select 姓名，count(学号) as num

from 学生表

group by 姓名

having count(学号)>1 --保证查找到的都是存在 2 个以上（包括 2）的同名同姓的姓名及人数。

查询平均成绩大于 60 分的学生的学号和平均成绩；

Select s_id,avg(score) from sc groupby s_id having avg(score)>60

查询姓“李”的老师的个数；

Select count(*),teacher.t_name from teacher where teacher.t_name like '李%'

600. 取出 sql 表中低 31 到 40 的记录（以自动增长 ID 为主键）

Sql server 方案：

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

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

Oracle 方案：

```
select rownum num,tid from (select rownum num,tid from t_test)
    where num>=30 and num<=41;
```

601. 判断身份证：要么是 15 位，要么是 18 位，最后一位可以为字母，并写出程序提出其中年月日。要求：

写出合格的身份证的正则表达式，

^(\d{15}|\d{17}[\dx])$

写程序提取身份证中的年月日

```
public class IdCard
{
    private String idCard;//私有变量
    public IdCard(){}//构造方法
   //构造方法
   public IdCard(String idCard){
        this.idCard=idCard;
     }

    public void setIdCard(String idCard)
    {
        this.idCard=idCard;
    }

    public String getIdCard()
    {
        return idCard;
```

```
    }

    //从身份证号码中截取生日
    public String getBirthday()
    {
       return this.getIdCard().substring(6, 14);
    }

    public static void main(String args[])
    {
        ShenFenZheng sfz = new ShenFenZheng("420154199908157841");
         //调用 getBirthday()方法获取生日
        System.out.println("生日：" + sfz.getBirthday());
    }
}
```

对于一个字符串，请设计一个高效算法，找到第一次重复出现的字符保证字符串中有重复的字符，字符串的长度小于等于 500.

```
package com.bjsxt;

import java.util.ArrayList;
```

```
import java.util.List;
public class FirstRepeat {


    public static void main(String[] args) {
    System.out.println(findFirstRepeat("pmedmitjtckhxwhvpwemznhmh
zhpueainchqrftkmbjlradhmjekcqzansyzkvqhwnrdgzdbzewdmxkzrscikda
ugbvygntrifnolehdtrqjlasofuvzeijbmzehkxknmjekcxswqldknysfsxrqaqzp",
152));
    }
    //返回:y
     public static char findFirstRepeat(String A, int n) {
        String[] str=A.split("");
        for(int x=0;x<n;x++){
            int index=0;
            int num=0;
            //对于每一个值，都需要从前开始遍历
            while(index<=x){
                if(str[index].equals(str[x])){
                    num++;
                }
                index++;
            }
```

```
            //该值出现了两次，说明重复了
            if(num>1){
                char flag='x';
                flag=str[x].toCharArray()[0];
                return flag;
            }
        }
        //返回该值说明已经没有重复的
        return 'p';
    }
}
```

602. 下列两个表 ,需要用一条 sql 语句把 b 表中的 ID 和 NAME 字段的数值复制到 A 表中

A 表

ID	NAME

B 表

ID	NAME	OTHER
1	Aaa	Ddd
2	Bbb	Eee

insert into a select id,name from b;

603. 什么是基本表，什么是视图，两者的区别和联系是什么？

它是从一个或几个基本表中导出的 表，是从现有基本表中抽取若干子集组成用户的“专用表”。

基本表：基本表的定义指建立基本关系模式,

而变更则是指对数据库中已存在的基本表进行删除与修改。

区别：

1、视图是已经编译好的 sql 语句。而表不是

2、视图没有实际的物理记录。而表有。

3、表是内容，视图是窗口

4、表只用物理空间而视图不占用物理空间，

视图只是逻辑概念的存在，表可以及时对它进行修改，

但视图只能有创建的语句来修改

5、表是内模式，试图是外模式

6、视图是查看数据表的一种方法，

可以查询数据表中某些字段构成的数据，

只是一些 SQL 语句的集合。从安全的角度说，

视图可以不给用户接触数据表，从而不知道表结构。

7、表属于全局模式中的表，是实表；视图属于局部模式的表，

是虚表。

8、视图的建立和删除只影响视图本身，不影响对应的基本表。

联系：视图（view）是在基本表之上建立的表，它的结构（

即所定义的列）和内容（即所有数据行）都来自基本表，

它依据基本表存在而存在。一个视图可以对应一个基本表，

也可以对应多个基本表。

视图是基本表的抽象和在逻辑意义上建立的新关系

604. 什么是事务？什么是锁？

事务与锁是不同的。事务具有 ACID（

原子性、一致性、隔离性和持久性），锁是用于解决隔离性的一种机制。事务的隔离级别通过锁的机制来实现。另外锁有不同的粒度，同时事务也是有不同的隔离级别的（一般有四种：读未提交 Read uncommitted，

读已提交 Read committed，

可重复读 Repeatable read，

可串行化 Serializable）。

在具体的程序设计中，开启事务其实是要数据库支持才行的，如果数据库本身不支持事务，那么仍然无法确保你在程序中使用的事务是有效的。

锁可以分为乐观锁和悲观锁：

悲观锁：认为在修改数据库数据的这段时间里存在着也想修改此数据的事务；

乐观锁：认为在短暂的时间里不会有事务来修改此数据库的数据；

我们一般意义上讲的锁其实是指悲观锁，在数据处理过程中，将数据置于锁定状态（由数据库实现）。

如果开启了事务，在事务没提交之前，别人是无法修改该数据的；如果 rollback，你在本次事务中的修改将撤消（不是别人修改的会没有，因为别

人此时无法修改）。当然，前提是你使用的数据库支持事务。还有一个要注意的是，部分数据库支持自定义 SQL 锁覆盖事务隔离级别默认的锁机制，如果使用了自定义的锁，那就另当别论。

重点：一般事务使用的是悲观锁（具有排他性）。

605. Student 学生表(学号 ,姓名、性别、年龄、组织部门),Course 课程表（编号，课程名称），Sc 选课表（学号，课程编号，成绩）

写一个 SQL 语句，查询选修了计算机原理的学生学号和姓名

select 学号，姓名 from Student where 学号 in

（select 学号 from Sc where 课程编号 in

(Select 课程编号 from Course where 课程名称 = ‘计算机原理’））

写一个 SQL 语句，查询“周星驰”同学选修了的课程名字

select 课程名称 from Course where 编号 in（

select Sc.课程编号 from Student,Sc where Student.姓名=’周星驰’

and Student.学号 = Sc.学号)

写一个 SQL 语句，查询选修了 5 门课程的学生学号和姓名

Select 学号，姓名 from Student where 学号 in（

Select 学号，count(课程编号) from Sc group by 学号 having count(课程编号)>=5)

606. sql 查询

Student(S#,Sname,Sage,Ssex)学生表

S#：学号

Sname:学生姓名

Sage：学生年龄

Ssex: 学生性别

Course(C#,Cname,T#)课程表

C#,课程编号；

Cname:课程名字；

T#：教师编号；

SC(S#,C#,score)成绩表

S#:学号；

C#,课程编号；

Score：成绩；

Teacher(T#,Tname)教师表

T#:教师编号；

Tname:教师名字

查询“001”课程比“002”课程成绩高的所有学生学号

```
select SC1.S#
   from SC SC1 JOIN SC SC2 ON SC1.S#=SC2.S#
      WHERE SC1.C#='001' AND SC2.C#='002' AND
SC1.score>SC2.score
```

查询平均成绩大于 60 分的同学的学号和平均成绩

```
select S#,AVG(score) 平均成绩
  from SC
    group by S#
      having AVG(score)>60
```

查询所有同学的学号、姓名、选课数、总成绩

```
select Student.S#,Sname,COUNT(*) 选课数,SUM(score) 总成绩
   from Student JOIN SC on Student.S#=SC.S#
      group by Student.S#,Sname
```

查询姓“李”的老师的个数

```
Select count(*) from Teacher where Tname like ‘李%’;
```

查询没学过“叶平”老师课的同学的学号、姓名

```
SELECT stu2.s#,stu2.stuname FROM Student stu2 WHERE stu2.s# NOT IN
      (SELECT DISTINCT stu.s# FROM student stu, course c,teacher
    tea,score score

      WHERE stu.s#= score.s# AND course.c#= score.c#
        AND tea.t#= course.t#AND tea.tname= '叶平' )
```