2015网易校招Java开发工程师在线笔试题

1 程序和进程的本质区别是？
A 在外存和内存存储　B 非顺序和顺序执行机器指令　C 独占使用和分时使用计算机资源
D 静态和动态特征

正确答案：D
题目解析：
进程与应用程序的区别：
进程（Process）是最初定义在Unix等多用户、多任务操作系统环境下用于表示应用程序在内存环境中基本执行单元的概念。以Unix操作系统为例，进程是Unix操作系统环境中的基本成分、是系统资源分配的基本单位。Unix操作系统中完成的几乎所有用户管理和资源分配等工作都是通过操作系统对应用程序进程的控制来实现的。
C、C++、Java等语言编写的源程序经相应的编译器编译成可执行文件后，提交给计算机处理器运行。这时，处在可执行状态中的应用程序称为进程。从用户角度来看，进程是应用程序的一个执行过程。从操作系统核心角度来看，进程代表的是操作系统分配的内存、CPU时间片等资源的基本单位，是为正在运行的程序提供的运行环境。进程与应用程序的区别在于应用程序作为一个静态文件存储在计算机系统的硬盘等存储空间中，而进程则是处于动态条件下由操作系统维护的系统资源管理实体。
多任务环境下应用程序进程的主要特点包括：
●进程在执行过程中有内存单元的初始入口点，并且进程存活过程中始终拥有独立的内存地址空间；
●进程的生存期状态包括创建、就绪、运行、阻塞和死亡等类型；
●从应用程序进程在执行过程中向CPU发出的运行指令形式不同，可以将进程的状态分为用户态和核心态。处于用户态下的进程执行的是应用程序指令、处于核心态下的应用程序进程执行的是操作系统指令。
在Unix操作系统启动过程中，系统自动创建swapper、init等系统进程，用于管理内存资源以及对用户进程进行调度等。在Unix环境下无论是由操作系统创建的进程还要由应用程序执行创建的进程，均拥有唯一的进程标识（PID）

2 假设某算法的时间复杂度符合递推关系式T(n)=2T(n/2)+n，那么该算法的时间复杂度相当于
A O(n)　B O(lgn)　C O(nlgn)　D O(n2)

正确答案：C
题目解析：解析：由时间代价严格推出时间复杂度比较复杂，对于这种题，可用特例验证，不过需要注意的是特例不能取太少，至少n取到5，这样规律基本就可以确定了。
T(1)=1
T(2)=2T(1)+2=4
T(3)=2T(1)+3=5
T(4)=2T(2)+4=12

T(5)=2T(2)+5=13
很容易排除D选项，其递增速率介于O(n)和O(n2)之间，实际上可以参考归并排序的递推公式。

3
下图一个非确定有限自动机（NFA）的状态转换，其等价的正规式为（）
A 0*|(0|1)0　B (0|10)*　C 0*((0|1)0)*　D 0*(10)*

正确答案：B
题目解析：根据分析题目中给出的状态转换图可知，该NFA可识别空串以及任意数目0组成的串，但若出现1，则其后至少要有1个0才能到达终态，因此，该自动机识别的串等价于正规式(0|10)*。

4 IPv6地址占____个字节
A 4　B 6　C 8　D 16

正确答案：D
题目解析：
IPv4采用32位地址长度，约有43亿地址，IPv6地址为128位长，但通常写作8组，每组为四个十六进制数的形式。
例如：FE80:0000:0000:0000:AAAA:0000:00C2:0002 是一个合法的IPv6地址。
IPv4映像地址布局如下：
| 80bits |16 | 32bits |
0000.....0000 | FFFF | IPv4 address |
IPv4兼容地址写法如下：::192.168.89.9
如同IPv4映像地址，这个地址仍然是一个IPv6地址，只是0000:0000:0000:0000:0000:0000:c0a8:5909的另外一种写法罢了

5 以下关于RARP协议的说法中，正确的是（）？
A RARP协议根据主机IP地址查询对应的MAC地址　B RARP协议用于对IP协议进行差错控制
C RARP协议根据MAC地址求主机对应的IP地址　D RARP协议根据交换的路由信息动态改变路由表

正确答案：C
题目解析：
逆地址解析协议（Reverse Address Resolution Protocol，RARP），是一种网络协议，互联网工程任务组（IETF）在RFC903中描述了RARP[1]。RARP使用与ARP相同的报头结构，作用与ARP相反。RARP用于将MAC地址转换为IP地址。其因为较限于IP地址的运用以及其他的一些缺点，因此渐为更新的BOOTP或DHCP所取代。
RARP的工作原理：
1. 发送主机发送一个本地的RARP广播，在此广播包中，声明自己的MAC地址并且请求任何收到此请求的RARP服务器分配一个IP地址；
2. 本地网段上的RARP服务器收到此请求后，检查其RARP列表，查找该MAC地址对应的IP地址；
3. 如果存在，RARP服务器就给源主机发送一个响应数据包并将此IP地址提供给对方主机使用；
4. 如果不存在，RARP服务器对此不做任何的响应；
5. 源主机收到从RARP服务器的响应信息，就利用得到的IP地址进行通讯；如果一直没有收到RARP服务器的响应信息，表示初始化失败

6 请描述JAVA异常类的继承体系结构，以及JAVA异常的分类，并为每种类型的异常各举三个例子？

正确答案：

粉红色的是受检查的异常(checked exceptions),其必须被try{}catch语句块所捕获,或者在方法签名里通过throws子句声明.受检查的异常必须在编译时被捕捉处理,命名为 CHeckedException 是因为Java编译器要进行检查,Java虚拟机也要进行检查,以确保这个规则得到遵守。

绿色的异常是运行时异常(runtime exceptions),需要程序员自己分析代码决定是否捕获和处理,比如 空指针,被0除...

而声明为Error的，则属于严重错误,需要根据业务信息进行特殊处理,Error不需要捕捉。

比如：AWTError、ThreadDeath、OutOfMemoryError.

题目解析：　　在 Java 中，所有的异常都有一个共同的祖先 Throwable（可抛出）。Throwable 指定代码中可用异常传播机制通过 Java 应用程序传输的任何问题的共性。

Throwable： 有两个重要的子类：Exception（异常）和 Error（错误），二者都是 Java 异常处理的重要子类，各自都包含大量子类。

Error（错误）:是程序无法处理的错误，表示运行应用程序中较严重问题。大多数错误与代码编写者执行的操作无关，而表示代码运行时 JVM（Java 虚拟机）出现的问题。例如，Java虚拟机运行错误（Virtual MachineError），当 JVM 不再有继续执行操作所需的内存资源时，将出现 OutOfMemoryError。这些异常发生时，Java虚拟机（JVM）一般会选择线程终止。

。这些错误表示故障发生于虚拟机自身、或者发生在虚拟机试图执行应用时，如Java虚拟机运行错误（Virtual MachineError）、类定义错误（NoClassDefFoundError）等。这些错误是不可查的，因为它们在应用程序的控制和处理能力之 外，而且绝大多数是程序运行时不允许出现的状况。对于设计合理的应用程序来说，即使确实发生了错误，本质上也不应该试图去处理它所引起的异常状况。在 Java中，错误通过Error的子类描述。

Exception（异常）:是程序本身可以处理的异常。

Exception 类有一个重要的子类 RuntimeException。RuntimeException 类及其子类表示“JVM 常用操作”引发的错误。例如，若试图使用空值对象引用、除数为零或数组越界，则分别引发运行时异常（NullPointerException、ArithmeticException）和 ArrayIndexOutOfBoundException。

注意：异常和错误的区别：异常能被程序本身可以处理，错误是无法处理。

7 描述Java类加载器的原理及其组织结构？

正确答案：

jvm classLoader architecture :

a、Bootstrap ClassLoader/启动类加载器

主要负责jdk_home/lib目录下的核心 api 或 -Xbootclasspath 选项指定的jar包装入工作.

B、Extension ClassLoader/扩展类加载器

主要负责jdk_home/lib/ext目录下的jar包或 -Djava.ext.dirs 指定目录下的jar包装入工作

C、System ClassLoader/系统类加载器

主要负责java -classpath/-Djava.class.path所指的目录下的类与jar包装入工作.

B、 User Custom ClassLoader/用户自定义类加载器(java.lang.ClassLoader的子类)

在程序运行期间, 通过java.lang.ClassLoader的子类动态加载class文件, 体现java动态实时类装入特性.

题目解析：

类加载器的树状组织结构

Java 中的类加载器大致可以分成两类，一类是系统提供的，另外一类则是由 Java 应用开发人员编写的。系统提供的类加载器主要有下面三个：

引导类加载器（bootstrap class loader）：它用来加载 Java 的核心库，是用原生代码来实现的，并不继承自 java.lang.ClassLoader。

扩展类加载器（extensions class loader）：它用来加载 Java 的扩展库。Java 虚拟机的实现会提供一个扩展库目录。该类加载器在此目录里面查找并加载 Java 类。
系统类加载器（system class loader）：它根据 Java 应用的类路径（CLASSPATH）来加载 Java 类。一般来说，Java 应用的类都是由它来完成加载的。可以通过 ClassLoader.getSystemClassLoader()来获取它。
除了系统提供的类加载器以外，开发人员可以通过继承 java.lang.ClassLoader类的方式实现自己的类加载器，以满足一些特殊的需求。
除了引导类加载器之外，所有的类加载器都有一个父类加载器。通过 表 1中给出的 getParent()方法可以得到。对于系统提供的类加载器来说，系统类加载器的父类加载器是扩展类加载器，而扩展类加载器的父类加载器是引导类加载器；对于开发人员编写的类加载器来说，其父类加载器是加载此类加载器 Java 类的类加载器。因为类加载器 Java 类如同其它的 Java 类一样，也是要由类加载器来加载的。一般来说，开发人员编写的类加载器的父类加载器是系统类加载器。类加载器通过这种方式组织起来，形成树状结构。树的根节点就是引导类加载器。图 1中给出了一个典型的类加载器树状组织结构示意图，其中的箭头指向的是父类加载器。

8 请简述Spring架构中IOC的实现原理？

正确答案：IoC在系统运行中，动态的向某个对象提供它所需要的其他对象。通过依赖注入来实现的。比如对象A需要操作数据库，以前我们总是要在A中自己编写代码来获得一个Connection对象，有了 spring我们就只需要告诉spring，A中需要一个Connection，至于这个Connection怎么构造，何时构造，A不需要知道。在系统运行时，spring会在适当的时候制造一个Connection，然后像打针一样，注射到A当中，这样就完成了对各个对象之间关系的控制。A需要依赖 Connection才能正常运行，而这个Connection是由spring注入到A中的。那么DI是如何实现的呢？spring是通过反射来实现注入的。
题目解析：　　那么IoC是如何做的呢？有点像通过婚介找女朋友，在我和女朋友之间引入了一个第三者：婚姻介绍所。婚介管理了很多男男女女的资料，我可以向婚介提出一个列表，告诉它我想找个什么样的女朋友，比如长得像李嘉欣，身材像林熙雷，唱歌像周杰伦，速度像卡洛斯，技术像齐达内之类的，然后婚介就会按照我们的要求，提供一个mm，我们只需要去和她谈恋爱、结婚就行了。简单明了，如果婚介给我们的人选不符合要求，我们就会抛出异常。整个过程不再由我自己控制，而是有婚介这样一个类似容器的机构来控制。Spring所倡导的开发方式就是如此，所有的类都会在spring容器中登记，告诉spring你是个什么东西，你需要什么东西，然后spring会在系统运行到适当的时候，把你要的东西主动给你，同时也把你交给其他需要你的东西。所有的类的创建、销毁都由 spring来控制，也就是说控制对象生存周期的不再是引用它的对象，而是spring。对于某个具体的对象而言，以前是它控制其他对象，现在是所有对象都被spring控制，所以这叫控制反转。

9 一栋大楼共及200层，某种类型的鸡蛋从某一楼层及其以上楼层下来时会被打破，从该楼层（即临界楼层）以下楼层摔下该鸡蛋，鸡蛋不会出现破损。现在给你2个完全一样的该种类型的鸡蛋，问：如何通过这2个鸡蛋找到该临界楼层

正确答案：　假设我们从第2个楼层开始试探，往楼层号码渐次增长的方向，每隔数个楼层，试探一次，并在试探到第1个鸡蛋摔破的地方停下来，用第2个鸡蛋来从最近一次试探没有摔破的楼层之上的那个楼层开始逐个楼层进行试探，知道鸡蛋被摔破，我们就得到了临界楼层，如果鸡蛋一直没有被摔破，那么第一个鸡蛋摔破的楼层就是临界楼层。现在假设第2个楼层和下一次试探的楼层之间有x个楼层，即第2次试探的楼层号是A(2)=x+3，以后试探的楼层间隔分别减少1，那么我们第3次试探的楼层号为A(3)=2x+3，第4次为A(4)=3x+2，第5次为A(5)=4x，第n次为A(n)=(n-1)*x-(1/2)*n*n+(5/2)*n，这里需要注意，我们试探的n不能超过x+1，可以这么想来：跳跃测试的n不应超过第一次最大的跨度（也即第一种需要连续测试的区间大小），及n<=x+1。那么把x替换为n-1，得到A(n)=(1/2)*n*(n+1)+1。楼层为100，那么A(n)=(1/2)*n*(n+1)+1>=100，得到n(n+1)>=198，得n=14，x=13，那么A(n)=(31*n-n*n-26)/2. 即通过楼层2,16,29,41,52,62,71,79,86,92,97,101,(104,106).作为间隔就可以在使用2个鸡蛋，不超过14次测试的情况下找到临界楼层。
题目解析：给了我们2个鸡蛋，意思就很明显，有1个鸡蛋起到关键作用，它可以被打破，以告诉我们临界楼层大

致在什么位置。如果给了我们99个及更多的鸡蛋，我们大致可以正序（2~100）或者倒序地（100~2）来逐层摔鸡蛋，在能摔破鸡蛋和不能摔破鸡蛋的临近两层楼的地方确定了临界楼层的楼层号。给了2个鸡蛋，那么我们可以大胆地想象通过摔破一个鸡蛋来提供给我们有用的信息以评估临界楼层。比如，我们很容易想到折半查找，从51楼摔一下鸡蛋，如果不碎，那么再从76，如果在76碎了，那么，从76和52的中值64处来试探貌似是不ok的，因为如果在64层，第2个鸡蛋也碎了，那么还有区间[53,63]都是可能摔碎鸡蛋的楼层。所以这种折半的方法不太靠谱，除非我们的鸡蛋远远超过2个。从刚才可以看到，我们将区间[2,100]通过折半的方法分成了两个段[2,50]及[51,100]，效率是比顺序和倒序的情况（即不分段）好很多的，在折半分段的情况下，我们可以比不分段的情况少打破至少一半以上的鸡蛋，就可以测试出临街楼层。只不过在摔破鸡蛋的每个小段区间里，每个楼层都应该被逐层的试探。好了，这就是我们要的思想。那么，现在的问题是：怎么来合理地分段呢？刚才的折半的方法是非常粗糙的，在鸡蛋很少（只有2个）的情况下，是不可能使用折半查找的。我们大胆地估计一下，如果把100分成10段，那么要找到临界楼层，我们大致最多需要20次测试，我们通过2,12,22,32,...，不断地试探，确定了某个区间，就进入该区间逐个测试。试看：如果临界楼层在10层一下，我测试2楼的时候鸡蛋正常，测试12楼的时候鸡蛋破了1个鸡蛋，我们就回头从3楼，4楼测试到11楼，直到第2个鸡蛋被摔破，就测试出来了（如果鸡蛋不摔破，那么临界楼层就是第12楼），总共试探了11次。同理，如果临界楼层在90几楼，那么，我们从2,12,22,32,...，不断测试应该不超过20次就可以测试出来。到目前为止我们已经取得比较好的结果了。再想想，正方形的面积要比长方形多吧，我们可以不以平均一下，不管临界楼层是在2~12之间还是90~100之间都能在一个比较平均的测试次数下面来把它测试出来呢？试想，我们在90几的临界楼层时，区间大小如果都是固定的，那么对于临界楼层比较大的情形，就多产生了测试次数。我们很容易就想到，可以设想，让区间大小从前往后，逐步减少，让相邻的两个区间，楼层较低的区间段比楼层号较高的区间段多1，这样就保证了无论临界楼层在哪个区间段，我们总能通过同样的试探次数将能将其找出来，平均来说是最优的方案。

10 在Web开发中，如何实现会话的跟踪？

正确答案：SSH会话、Cookies、URL重定向
题目解析：
A、SSL会话（Secure Socket Layer）
安全套接字层，是一种运行在TCP/IP之上和像HTTP这种应用层协议之下的加密技术。SSL是在HTTPS协议中使用的加密技术。SSL可以让采用SSL的服务器认证采用SSL的客户端，并且在客户端和服务器之间保持一种加密了连接，在建立了加密连接的过程中，客户端和服务器都可以产生一种名为“会话密钥”的东西，它是一种用于加密和解密的对称密钥。基于HTTPS协议的服务器可以使用这个客户端的对称密钥来建立会话
B、Cookies
中文译为小甜饼，由Netscape公司发明，是最常用的跟踪用户会话的方式。Cookies是一种由服务器发送给客户端的片段信息，存储在客户端的内存或者硬盘上，在客户随后对该服务器的请求中发回它。其实主要就是把服务器为客户端分配的session ID保存在Cookies中，每次发送请求时把Cookies附加到请求对象中一起发过去，服务器得到这个唯一的session ID，从而可以唯一的标识一个客户端
3、URL重写
如果客户端禁用了Cookies，那么就只能用URL重写机制了。就是在URL中附加标识客户端的sessionID，web容器解析URL，取出session ID，根据这个session ID将请求与特定的session关联起来。
注意如果采用了URL重写，那么代码里面的所有url都要经过编码，response.sendRedirect(url)中的url用response.encodeRedirectURL(url)编码，其他的用response.encodeURL(url)来编码

11

```
package algorithms.com.guan.javajicu;
public class TestDemo {
   public static String output ="";
   public static void foo(inti){
```

```
    try{
        if(i == 1){
          throw new Exception();
        }
    }catch(Exception e){
        output += "2";
        return ;
    }finally{
        output += "3";
    }
    output += "4";
  }
  public static void main(String[] args) {
    foo(0);
    foo(1);
    System.out.println(output); //3423
  }
}
```

正确答案：

3423

题目解析：

final**语句是在**catch**之后执行，在**try**语句块**return**结束之前执行**final**语句**。

foo(0)**执行了**final**块之后再执行**try**后面的语句，**

foo(1)**执行了**catch**语句之后再执行**final**块，然后在**catch**中**return**了**。

12

```
package algorithms.com.guan.javajicu;
public class TestDemo {
  public static String output ="";
  public static void foo(inti){
    try{
        if(i == 1){
          throw new Exception();
        }
    }catch(Exception e){
        output += "2";
        return ;
    }finally{
        output += "3";
    }
    output += "4";
  }
  public static void main(String[] args) {
    foo(0);
    foo(1);
```

```
        System.out.println(output); //3423
    }
}
```

正确答案：

3423

题目解析：

final**语句是在**catch**之后执行，在**try**语句块**return**结束之前执行**final**语句。**

foo(0)**执行了**final**块之后再执行**try**后面的语句，**

foo(1)**执行了**catch**语句之后再执行**final**块，然后在**catch**中**return**了。**

13

检查程序，是否存在问题，如果存在指出问题所在，如果不存在，说明输出结果。

```
package algorithms.com.guan.javajicu;
public class HelloB extends HelloA{
    public HelloB(){
        System.out.println("HelloB");
    }
    {
        System.out.println("I'm B class");
    }
    static{
        System.out.println("static B");
    }
    public static void main(String[] args) {
        new HelloB();
    }
}
class HelloA{
    public HelloA(){
        System.out.println("HelloA");
    }
    {
        System.out.println("I'm A class");  //这句话是什么时候加载？
    }
    static{
        System.out.println("static A");
    }
}
```

正确答案： /**输出结果参考答案：

```
     * static A
       staticB
       I'mA class
       HelloA
       I'mB class
```

```
        HelloB
      */
```

题目解析：

一个类的初始化过程如下：

1.**先顺序执行父类的**static**初始化代码块，再执行子类的**static**初始化代码块；**

2.**先顺序执行父类的非**static**初始化代码块，再执行子类的非**static**初始化代码块；**

3.**先执行父类的构造函数，再执行子类的构造函数；**

14

检查程序，是否存在问题，如果存在指出问题所在，如果不存在，说明输出结果。

```
package algorithms.com.guan.javajicu;
public class Inc {
    public static void main(String[] args) {
        Inc inc = new Inc();
        int i = 0;
        inc.fermin(i);
        i= i ++;
        System.out.println(i);  //输出结果为0
    }
    void fermin(int i){
        i++;
    }
}
```

正确答案：

//输出结果为0

题目解析：

i=i++ **是把**i**赋值给**i**之后，再对**i**的被赋值临时对象执行**+1**；**int**类型在函数调用上是传值，不是传引用。**

15

检查程序，是否存在问题，如果存在指出问题所在，如果不存在，说明输出结果。

```
package algorithms.com.guan.javajicu;
public class Example {
  Stringstr = new String("good");
  char[] ch = {'a','b','c'};
  public static void main(String[] args) {
     Exampleex = new Example();
     ex.change(ex.str, ex.ch);
     System.out.print(ex.str +"and");
     System.out.print(ex.ch);
     //参考答案输出结果：goodandgbc
  }
  public void change(Stringstr, char ch[]){
     str= "test ok";
     ch[0]= 'g';
  }
```

}

正确答案：输出结果：goodandgbc
题目解析：
当一个string**对象作为参数传递给函数时，其实传递的是引用的一个**copy**，每当把**string**对象作为方法的参数时，都会复制一份引用，原来的引用没有改变，这和数组不一样，如果将数组作为参数传递给一个函数，就是传递的这个数组的引用，对它进行函数处理，就是对原本的数组进行处理，**

16
检查程序，是否存在问题，如果存在指出问题所在，如果不存在，说明输出结果。

```
packagealgorithms.com.guan.javajicu;
import java.util.Date;
public class SuperTest extends Date{
   private static final long serialVersionUID = 1L;
   private void test(){
     System.out.println(super.getClass().getName());
   }
   public static void main(String[]args){
     newSuperTest().test();
   }
}
```

正确答案：//参考答案输出：algorithms.com.guan.javajicu.SuperTest
题目解析：
Class<? extends Date> java.lang.Object.getClass()
Returns the runtime class of this Object. The returned Class object is the object that is locked by static synchronized methods of the represented class.
The actual result type is Class<? extends |X|> where |X| is the erasure of the static type of the expression on which getClass is called. For example, no cast is required in this code fragment:
Number n = 0;
Class<? extends Number> c = n.getClass();
Returns:
The Class object that represents the runtime class of this object.
看getClass**方法的说明，返回当前的执行的类对象**

17 任意2n个整数，从其中选出n个整数，使得选出的n个整数和同剩下的n个整数之和的差最小

正确答案：假设数组A[1..2N]**所有元素的和是**SUM**。模仿动态规划解**0-1**背包问题的策略，令**S(k, i)**表示前**k**个元素中任意**i**个元素的和的集合。显然：**
S(k, 1) = {A[i] | 1<= i <= k}
S(k, k) = {A[1]+A[2]+…+A[k]}
S(k, i) = S(k-1, i) U {A[k] + x | x**属于**S(k-1, i-1) }
按照这个递推公式来计算，最后找出集合S(2N, N)**中与**SUM/2**最接近的那个和，这便是答案。这个算法的时间复杂度是**O(2^N).
因为这个过程中只关注和不大于SUM/2**的那个子数组的和。所以集合中重复的和以及大于**SUM/2**的和都是没有意义的。把这些没有意义的和剔除掉，剩下的有意义的和的个数最多就是**SUM/2**个。所以，我们不需要记录**S(2N,N)

中都有哪些和，只需要从SUM/2到1遍历一次，逐个询问这个值是不是在S(2N,N)中出现，第一个出现的值就是答案。我们的程序不需要按照上述递推公式计算每个集合，只需要为每个集合设一个标志数组，标记SUM/2到1这个区间中的哪些值可以被计算出来。
题目解析：

程序员面试100题之十五：数组分割

```
#include<iostream>
using namespace std;
//有一个没有排序，元素个数为2N的正整数数组。要求把它分割为元素个数为N的两个数组，并使两个子数组的和最接近。
int arr[] = {0,1,5,7,8,9,6,3,11,20,17};
const int N=5;
const int SUM = 87;
// 模仿动态规划解0-1背包问题的策略
int solve1()
{
int i , j , s;
int dp[2*N+1][N+1][SUM/2+2];
/*
用dp(i,j,c)来表示从前i个元素中取j个、且这j个元素之和不超过c的最佳(大)方案，在这里i>=j,c<=S
状态转移方程：
限第i个物品不取
dp(i,j,c)=max{dp(i-1,j-1,c-a[i])+a[i]，dp(i-1,j,c)}
dp(2N,N,SUM/2+1)就是题目的解。
*/
//初始化
memset(dp,0,sizeof(dp));
for(i = 1 ; i <= 2*N ; ++i)
{
for(j = 1 ; j <= min(i,N) ; ++j)
{
for(s = SUM/2+1 ; s >= arr[i] ; --s)
{
dp[i][j][s] = max(dp[i-1][j-1][s-arr[i]]+arr[i] , dp[i-1][j][s]);
}
}
}
//因为这为最终答案dp[2*N][N][SUM/2+1];
i=2*N , j=N , s=SUM/2+1;
while(i > 0)
{
if(dp[i][j][s] == dp[i-1][j-1][s-arr[i]]+arr[i])  //判定这个状态是由哪个状态推导出来的
{
cout<<arr[i]<<" ";   //取中arr[i]
j--;
s -= arr[i];
```

```
}
i--;
}
cout<<endl;
return dp[2*N][N][SUM/2+1];
}
int solve2()
{
int i , j , s;
int dp[N+1][SUM/2+2];    //取N+1件物品，总合不超过SUM/2+2，的最大值是多少
memset(dp,0,sizeof(dp));   //初始状态都为0
for(i = 1 ; i <= 2*N ; ++i)
{
for(j = 1 ; j <= min(i,N) ; ++j)
{
for(s = SUM/2+1 ; s >= arr[i] ; --s)   //01背包从大到小，可以省空间，即最外层的空间
{
dp[j][s] = max(dp[j-1][s-arr[i]]+arr[i] , dp[j][s]);
}
}
}
//要求最优解则空间不能优化,
return dp[N][SUM/2+1];
}
int solve3()
{
int i , j , s;
int isOK[N+1][SUM/2+2]; //isOK[i][v]表示是否可以找到i个数，使得它们之和等于v
memset(isOK,0,sizeof(isOK));   //都不合法
//注意初始化
isOK[0][0] = 1; //可以,取0件物品，总合为0，是合法的
for(i = 1 ; i <= 2*N ; ++i)
{
for( j = 1 ; j <= min(i,N) ; ++j)
{
for(s = SUM/2+1 ; s >= arr[i] ; --s) //从大到小，数组少了一维
{
if( isOK[j-1][s-arr[i]] )
isOK[j][s] = 1;
}
}
}
for(s = SUM/2+1 ; s >= 0 ; --s)
{
if(isOK[N][s])
return s;
```

```
}
//要求最优解则空间不能优化
return 0;
}
int main(void)
{
int s1 = solve1();
int s2 = solve2();
int s3 = solve3();
cout<<"s1="<<s1<<endl;
cout<<"s2="<<s2<<endl;
cout<<"s3="<<s3<<endl;
system("pause");
return 0;
}
```

18 有两个有序的集合，集合的每个元素都是一段范围，求其交集，例如集合{[4,8],[9,13]}和{[6,12]}的交集为{[6,8],[9,12]}

正确答案：
设置两个指针p和q，同时指向集合A和集合B的最小值，不相等的话移动*p和*q中较小值的指针，相等的话同时移动指针p和q，并且记下相等的数字，为交集的元素之一，依次操作，直到其中一个集合没有元素可比较为止。
对排好序的集合做查找操作，时间复杂度为O（N）
题目解析：参考：http://blog.csdn.net/jie1991liu/article/details/13168255

19 一个文件中有10000个数，用Java实现一个多线程程序将这个10000个数输出到5个不用文件中（不要求输出到每个文件中的数量相同）。要求启动10个线程，两两一组，分为5组。每组两个线程分别将文件中的奇数和偶数输出到该组对应的一个文件中，需要偶数线程每打印10个偶数以后，就将奇数线程打印10个奇数，如此交替进行。同时需要记录输出进度，每完成1000个数就在控制台中打印当前完成数量，并在所有线程结束后，在控制台打印”Done”.

正确答案：其实就是多线程的操作，参考 http://blog.csdn.net/vstar283551454/article/details/17362709
题目解析：
参考网上的java多线程处理，文件读写操作。