## 1.ArrayList list = new ArrayList(20);语句中的 list 集合大小扩充了几次（ A ）

A.0
B.1
C.2

D.3

## 2.如果去掉了 main 方法的 static 修饰符会怎样（ B ）

A.程序无法翻译
B.程序能正常编译，运行时或抛出 NoSuchMethodError 异常
C.程序能正常编译，正常运行
D.程序能正常编译，正常运行一会会立刻退出

## 3.启动 java 程序进程时，输入一下哪个参数可以实现年轻代的堆大小为 50M(C )

A.-Xms50M
B.-Xmx50M
C.-Xmn50M
D.-Xss50M

## 4.下面程序输出的结果是（ ）

```
static boolean foo(char c) {
    System.out.print(c);
    return true;
}
public static void main(String[] args) {
    int i = 0;
    for (foo('A'); foo('B') && (i < 2); foo('C')) {
        i++;
        foo('D');
    }
}
```

输出结果为：(A)

A. ABDCBDCB
B. ABDCDBCB
C. ABDBCDCB
D. ABDBCDCB

## 5.下面哪些是 Thread 类的方法（ A， B）

A.start()
B.run()
C.exit()
D.getPriority()

## 6.以下语句输出的结果是什么（ C）

System.out.print(Integer.MAX_VALUE*2);*
*System.out.print(Integer.MIN_VALUE*2);
A. -2-1
B. -1-2
C. -20
D. -1-1

## 7.log4j 的优先级从高到低的排序为（ A）

A. error>warn>info>debug
B. warn>info>debug>error

C. warn >debug>error>info
D. error>warn>debug>info

## 8.下列哪些方法可以使线程从运行状态进入到阻塞状态（ BCD）

A.notify
B.wait
C.sleep
D.yield

## 9.下列关于 Thread 类提供的线程控制的方法中，错误的一项是（ A）

A. 在线程 A 中执行线程 B 的 join()方法，则线程 A 等待直到 B 执行完成
B. 线程 A 通过调用 interrupt()方法来中断其阻塞状态。
C. currentThread()方法返回当前线程的引用
D. 若线程 A 调用方法 isAlive（ ）返回为 true,则说明 A 正在执行中

## 10.设 String s1 =”Topwalk”;String s2 =”Company”; 以下方法可以得到字符串“ TopwalkCompany” 有：（ ABD）

A. s2+s1;
B. s1.concat(s2)
C. s1.append(s2);
D.StringBuffer buf = new StringBuffer(s1);buf.append(s2);

## 11.String a = new String(“1”+”2”)最终创建了几个对象（ B ）

A.1
B.2
C.3
D.4

## 12.int 类型占用（ C ）个字节？

A.2
B.4
C.8
D.16

## 13.下列那一条语句可以实现快速的复制一张数据库表（ C ）

A. select * into b from a where 1<>1;
B. creat table b as select * from a where 0=1;
C. insert into b as select * from a where 1<>1;
D. insert into b select * from a where 1<>1;

## 14.属于单利模式的特点的是（ ACD ）

A. 提供了对唯一实现的受控访问
B. 允许可变数目的实例

C. 单利模式的抽象层会导致单例类扩展有和那的困难
D. 单利模式很容易导致数据库的连接池溢出

## 15.选择 Oracle 的分页语句的关键字（ A ）

A. rownum
B. limit
C.TOP
D. pagenum

## 16.选出可以查询出所有的表和视图的方法：（ B ）

A.preparedStatement.getMetaData().getTables***();***
***B.connection.getMetaData().getTables()***;
C.result.getMetaData().getTables***();***
***D..DiverManager.getMeta().getTables()***;

## 17.可以监控到数据库变化的机制有哪些（ AB ）

A. 存储过程
B. 数据库日志
C. 触发器
D. 物化视图

## 18.清空表所有数据的性能最优的语句是哪一个(B)

A. delete from tsuer;
B. truncate table tuser;ss
C. drop table tuser;
D. delete tuser;

## 19.文件对外共享的协议有哪几个（ AB ）

A. FTP
B. Windows 共享
C. TCP
D.SSH

## 20.关于 Java 中国特殊符号的用法正确的是（ AD ）

A. 判断一个字符串 str 中是否含有“.” ,可以根据 str.indexOf(“.”)是否等于-1 判断。
B. 判断一个字符串 str 是否含有“.” ,可以根据 str.indexOf(“\.”)是否等于-1 判断。
C. 根据“.”分隔字符串 str 的写法可以是 str.split(“\.”)
D. 根据“.”分隔字符串 str 的写法可以是 str.split(“.”)

## 21.根据以下代码回答问题，放置什么方法在地 6 行，会引起编译错误的是（ B ）

```
class Super{
    public float getNum(){
    }
    }
    public class Sub extends Super{

    }
```

A. public float getNum{return 4,0f;}
B.public void getNum(){};
C.public void getNum(double d()){}
D.public double getNum（float d）{return 4,0d;}

## 22.根据以下代码回答问题：输出结果是什么？(B)

```
public class Foo{
    public static void main(String args[]){
        try{return;}
        finally{System.out.println(“Finally”);}
    }
}
```

A. print out nothing;
B. print out “Finally”
C. 编译错误
D. 以上都不对

## 23.根据以下代码回答问题，请问输出 i 和 j 的值是多少（ D ）

```
int i=1,j=10;
do{
if(i++>--j) continue;
}while(i<5)
```

A. i=6 j=5

B i=5 j=5
C i=6 j=4
D i=5 j=6

## 24.请问 java 关键字？（ CD ）

A. run
B. low
C. import
D. implement

## 25.以下哪些不属于约束（ CD ）

A.主键
B.外键
C.索引
D.唯一索引
E.not null

## 26.下列关于数据库连接池的说法中哪个是错误的（ D ）

A. 服务器启动时会初始建立一定数量的池连接，并一直维持不少于此数目的池连接
B.客户端程序需要连接时，池驱动程序会返回一个使用的池连接并将其使用计数加 1；
C. 如果当前没有空闲连接，驱动程序就会再新建一定数量的连接，新建连接的数量可以由配置参数决定。

D. 当使用池连接调用完成后，池驱动程序将此连接标记为空间，其他调用就可以使用这个连接

## 28.以下哪句是对索引的错误描述（ C ）

A. 选择性差的索引只会降低 DML 语句的执行速度
B. 选择性强的索引只有被 Access Path 使用到才是有用的索引
C. 过多的索引只会阻碍性能的提升，而不是加速性能
D.在适当的时候将最常用的列放在复合索引的最前面
E. 索引和表的数据都存储在同一个 Segment 中

## 29.关于锁 locks，描述正确的是（ A ）

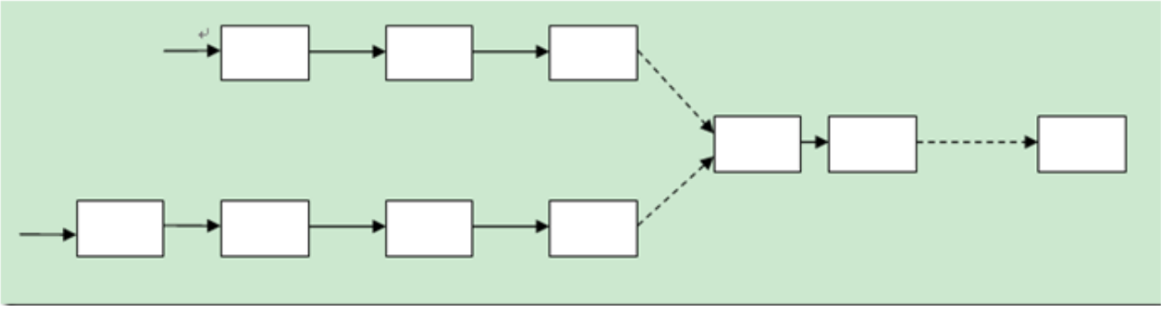
A. 当一个事务在表上防止了共享锁（shared lock）,其他事务，能阅读表里的数据
B. 当一个事务在表上防止了共享锁（shared lock）,其他事务，能更新表里的数据
C. 当一个事务在表上防止了排他锁（exclusive lock）,其他事务，能阅读表里的数据
D. 当一个事务在表上防止了排他锁（exclusive lock）,其他事务，能更新表里的数据

## 30.如下那种情况下， Oracle 不会使用 Full Table Scean(D)

A.缺乏索引，特别是在列上使用了函数，如果要利用索引，则需要使用函数索引。
B.当访问的数据占整个表中的大部分数据时
C.如果时一个表的 high water mark 数据块数少于初始化参数 DB_FILE_MULTIBLOCK_READ_COUNT
D.本次查询可以用到该张表的一个引用，但是该表具有多个索引包含用于过滤的字段

## 31.如何判断两个单向链表相交，如何找相交点？

第一种情况：两个链表均不含有环



思路：
1)、直接法
采用暴力的方法，遍历两个链表，判断第一个链表的每个结点是否在第二个链表中，时间复杂度为 O(len1*len2)，耗时很大。

2)、 hash 计数法

如 果 两个链表相交，则两个链表就会有共同的结点；而结点地址又是结点唯一标识。因而判断两个链表中是否存在地址一致的节点，就可以知道是否相交了。可以对第一 个链表的节点地址进行 hash 排序，建立 hash 表，然后针对第二个链表的每个节点的地址查询 hash 表，如果它在 hash 表中出现，则说明两个链表有共 同的结点。这个方法的时间复杂度为： O(max(len1+len2)；但同时还得增加 O(len1)的存储空间存储哈希表。这样减少了时间复杂度，增加 了存储空间。

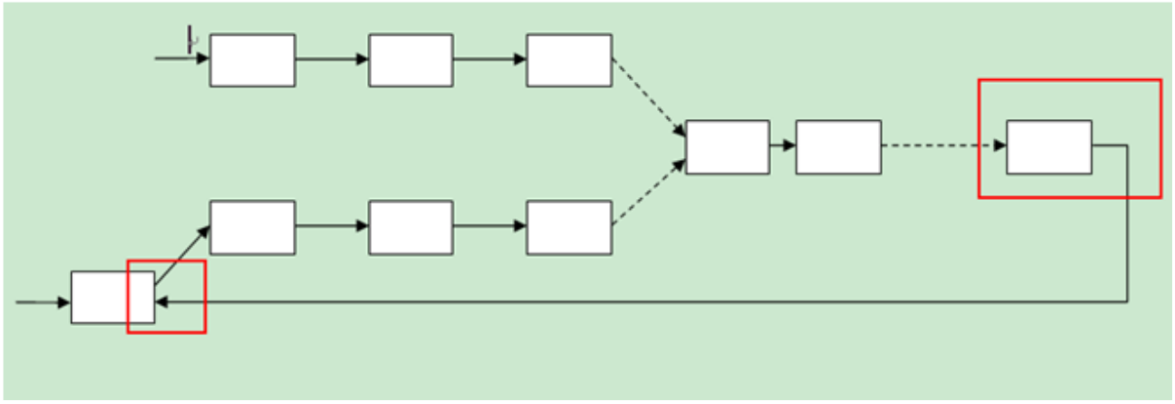
以链表节点地址为值，遍历第一个链表，使用 Hash 保存所有节点地址值，结束条件为到最后一个节点（无环）或 Hash 中该地址值已经存在（有环）。

再遍历第二个链表，判断节点地址值是否已经存在于上面创建的 Hash 表中。

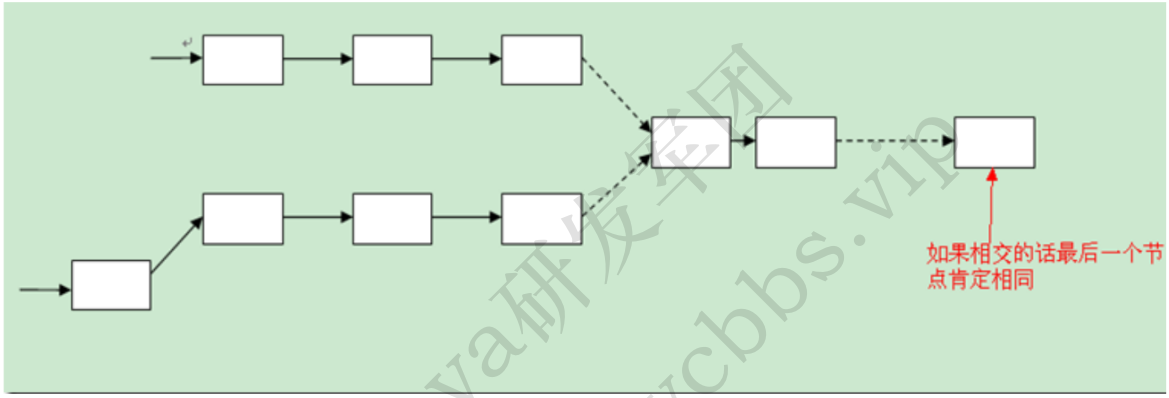
这个方面可以解决题目中的所有情况，时间复杂度为 O(m+n)， m 和 n 分别是两个链表中节点数量。由于节点地址指针就是一个整型，假设链表都是在堆中动态创建的，可以使用堆的起始地址作为偏移量，以地址减去这个偏移量作为 Hash 函数

3)、第三种思路是比较奇特的，在编程之美上看到的。先遍历第一个链表到他的尾部，然后将尾部的 next 指针指向第二个链表(尾部指针的 next 本来指向的是 null)。这样两个链表就合成了一个链表，判断原来的两个链表是否相交也就转变成了判断新的链表是否有环的问题了：即判断单链表是否有环？

这样进行转换后就可以从链表头部进行判断了，其实并不用。通过简单的了解我们就很容易知道，如果新链表是有环的，那么原来第二个链表的头部一定在环上。因此我们就可以从第二个链表的头部进行遍历的，从而减少了时间复杂度(减少的时间复杂度是第一个链表的长度)。
下图是一个简单的演示：



这种方法可以判断两个链表是否相交，但不太容易找出他们的交点。
4)、仔细研究两个链表，如果他们相交的话，那么他们最后的一个节点一定是相同的，否则是不相交的。因此判断两个链表是否相交就很简单了，分别遍历到两个链表的尾部，然后判断他们是否相同，如果相同，则相交；否则不相交。示意图如下：




判断出两个链表相交后就是判断他们的交点了。假设第一个链表长度为 len1，第二个问 len2，然后找出长度较长的，让长度较长的链表指针向后移动|len1 - len2| (len1-len2 的绝对值)，然后在开始遍历两个链表，判断节点是否相同即可。

## 32.如何从 cookie 中拿到 session?

答案： session 在服务器端， cookie 在客户端（浏览器）， session 默认被存在在服务器的一个文件里（不是内存）， session 的运行依赖 session id，而 session id 是存在 cookie 中的，也就是说，如果浏览器禁用了 cookie ，同时 session 也会失效（但是可以通过其它方式实现，比如在 url 中传递 session_id ）；session 可以放在 文件、数据库、或内存中都可以。
用户验证这种场合一般会用 session ，因此，维持一个会话的核心就是客户端的唯一标识，即 session id

## 33.System.out.println(3/2); System.out.println(3.0/2); System.out.println(3.0/2.0); 分别会打印什么结果？

答案： 1, 1.5， 1.5

## 34.打印出下面两个数组的交集，结果不能重复

String[] arr1 = {"112","wqw","2121"};
String[] arr2 = {"112","aad","ewqw"};
打印出两个数组的交集，结果不能重复
答案

```
public class test {
@Test//测试程序
public void test()
String[] arr1 = {"112","wqw","2121"};
String[] arr2 = {"112","aad","ewqw"};
String[] result=StringIntersection(arr1,arr2);
for (String str:result){
    System.out.printf(str);
}
}
//取两个 string 数组的交集
public String[] StringIntersection(String[] arr1,String[] arr2){
    Map<String,Boolean> map = new HashMap<String,Boolean>();
    List<String> list = new LinkedList<String>();
//取出 str1 数组的值存放到 map 集合中，将值作为 key，所以的 value 都设置为 false
    for (String str1:arr1){
        if (!map.containsKey(str1)){
            map.put(str1,Boolean.FALSE);
        }
    }
//取出 str2 数组的值循环判断是否有重复的 key，如果有就将 value 设置为 true
    for (String str2:arr2){
        if (map.containsKey(str2)){
            map.put(str2,Boolean.TRUE);
        }
    }
//取出 map 中所有 value 为 true 的 key 值，存放到 list 中
    for (Map.Entry<String,Boolean> entry:map.entrySet()){
        if (entry.getValue().equals(Boolean.TRUE)){
            list.add(entry.getKey());
        }
    }
//声明 String 数组存储交集
    String[] result={};
    return list.toArray(result);
}
}
```



## 35.SpringMvc 拦截器用过吗？什么场景会用到，过滤器，拦截器，监听器有什么区别？

拦截器是指通过统一拦截从浏览器发往服务器的请求来完成功能的增强。
使用场景：解决请求的共性问题（乱码问题、权限验证问题）

**过滤器**

Servlet 中的过滤器 Filter 是实现了 javax.servlet.Filter 接口的服务器端程序，主要的用途是过滤字符编码、做一些业务逻辑判断等。其工作原理是，只要你在 web.xml 文件配置好要拦截的客户端请求，它都会帮你拦截到请，此时你就可以对请求或响应(Request、 Response)统一设置编码，简化操作；同时还可进行逻辑判断，如用户是否已经登陆、有没有权限访问该页面等等工作。它是随你的 web 应用启动而启动的，只初始化一次，以后就可以拦截相关请求，只有当你的 web 应用停止或重新部署的时候才销毁。

**监听器**
现在来说说 Servlet 的监听器 Listener，它是实现了 javax.servlet.ServletContextListener 接口的服务器端程序，它也是随 web 应用的启动而启动，只初始化一次，随 web 应用的停止而销毁。主要作用是： 做一些初始化的内容添加工作、设置一些基本的内容、比如一些参数或者是一些固定的对象等等

**拦截器**
拦截器是在面向切面编程中应用的，就是在你的 service 或者一个方法前调用一个方法，或者在方法后调用一
个方法。是基于 JAVA 的反射机制。拦截器不是在 web.xml

1).过滤器：所谓过滤器顾名思义是用来过滤的，在 java web 中，你传入的 request,response 提前过滤掉一些信息，或者提前设置一些参数，然后再传入 servlet 或者 struts 的 action 进行业务逻辑，比如过滤掉非法 url（不是 login.do 的地址请求，如果用户没有登陆都过滤掉），或者在传入 servlet 或者 struts 的 action 前统一设置字符集，或者去除掉一些非法字符（聊天室经常用到的，一些骂人的话）。filter 流程是线性的， url 传来之后，检查之后，可保持原来的流程继续向下执行，被下一个 filter, servlet接收等.

2).监听器：这个东西在 c/s 模式里面经常用到，他会对特定的事件产生产生一个处理。监听在很多模式下用到。比如说观察者模式，就是一个监听来的。又比如 struts 可以用监听来启动。 Servlet 监听器用于监听一些重要事件的发生，监听器对象可以在事情发生前、发生后可以做一些必要的处理。

3).java 的拦截器 主要是用在插件上，扩展件上比如 hivernate spring struts2 等 有点类似面向切片的技术，在用之前先要在配置文件即 xml 文件里声明一段的那个东西

## 36.ThreadLocal 的原理和应用场景

每一个 ThreadLocal 能够放一个线程级别的变量，可是它本身能够被多个线程共享使用，并且又能够达到线程安
全的目的，且绝对线程安全。

ThreadLocal 的应用场景：
最常见的 ThreadLocal 使用场景为 用来解决数据库连接、 Session 管理等

## 37.简述 TCP 的三次握手

在 TCP/IP 协议中,TCP 协议提供可靠的连接服务,采用三次握手建立一个连接。
1).第一次握手：建立连接时,客户端发送 syn 包(syn=j)到服务器,并进入 SYN_SEND 状态,等待服务器确认； SYN：同步序列编号(Synchronize Sequence Numbers)
2).第二次握手：服务器收到 syn 包,必须确认客户的 SYN（ack=j+1）,同时自己也发送一个 SYN 包（syn=k）,即
SYN+ACK 包,此时服务器进入 SYN_RECV 状态；

3)第三次握手：客户端收到服务器的 SYN + ACK 包,向服务器发送确认包 ACK(ack=k+1), 此包发送完毕,客户端
和服务器进入 ESTABLISHED 状态,完成三次握手。完成三次握手,客户端与服务器开始传送数据

## 38.SpringMVC request 接收设置是线程安全的吗？

答案：是线程安全的， request、 response 以及 requestcontext 在使用时不需要进行同步。而根据 spring 的默认规则， controller 对于 beanfactory 而言是单例的。即 controller 只有一个， controller 中的request 等实例对象也只有一个

## 39.列举 Maven 常见的六种依赖范围

答案：
1.compile:编译依赖范围(默认),对其三种都有效
2.test:测试依赖范围,只对测试 classpath 有效
3.runtime:运行依赖范围,只对测试和运行有效,编译主代码无效,例如 JDBC
4.provided:已提供依赖范围,只对编译和测试有效,运行时无效,例如 selvet-api

5.system:系统依赖范围.谨慎使用.例如本地的,maven 仓库之外的类库文件
6.import(maven2.0.9 以上):导入依赖范围,不会对其他三种有影响

## 40.Mybatis 如何防止 sql 注入？ mybatis 拦截器了解过吗，应用场景是什么？

答案： Mybatis 使用#{}经过预编译的，是安全的，防止 sql 注入。 Mybatis 拦截器只能拦截四种类型的接口： Executor、 StatementHandler、ParameterHandler 和 ResultSetHandler。这是在 Mybatis 的 Configuration 中写死了的，如果要支持拦截其他接口就需要我们重写 Mybatis 的 Configuration。 Mybatis 可以对这四个接口中所有的方法进行拦截。 Mybatis 拦截器常常会被用来进行分页处理。

## 41.简单解释自动装配的各种模式，或者叫装配方式。

在 Spring 框架中共有 5 种自动装配:
no：这是 Spring 框架的默认设置，在该设置下自动装配是关闭的，开发者需要自行在 bean 定义中用标签明确的设置依赖关系。

byName：该选项可以根据 bean 名称设置依赖关系。当向一个 bean 中自动装配一个属 性时，容器将根据 bean 的名称自动在在配置文件中查询一个匹配的 bean。如果找到的 话，就装配这个属性，如果没找到的话就报错。

byType：该选项可以根据 bean 类型设置依赖关系。当向一个 bean 中自动装配一个属性时，容器将根据bean 的类型自动在在配置文件中查询一个匹配的 bean。如果找到的话，就装配这个属性，如果没找到的话就报错

constructor：造器的自动装配和 byType 模式类似，但是仅仅适用于与有构造器相同参数 的 bean，如果在容器中没有找到与构造器参数类型一致的 bean，那么将会抛出异常。

autodetect：该模式自动探测使用构造器自动装配或者 byType 自动装配。首先，首先会 尝试找合适的带参数的构造器，如果找到的话就是用构造器自动装配，如果在 bean 内部没有找到相应的构造器或者是无参构造器，容器就会自动选择 byTpe 的自动装配方式

## 42.mvc 的各个部分都有哪些技术来实现？如何实现的？

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

## 43.反射机制一般应用在什么场景？

答案： 反射机制的应用场景：
1)逆向代码 ，例如反编译
3)与注解相结合的框架 例如 Retrofit
4)单纯的反射机制应用框架 例如 EventBus 2.x
5)动态生成类框架 例如 Gson

## 44.设计 Java 程序，假设有 50 瓶饮料，喝完三个空瓶可以换一瓶饮料，依次类推，请问总共喝了多少饮料。

```
public class Buy {
    public static void main(String[] args) {
        int n = 50; //初始饮料总数
        int i=0; //兑换次数
        while(true){
            n -= 3; //喝 3 瓶
            n++; //兑换 1 瓶
            i++; //兑换次数+1
            if(n<3)
            {
                System.out.println ("共喝了"+(50+i)+"瓶");
                break;
            }
        }
    }
}
```

## 45.根据某年某月某日，输出这是一年中第几天

```
/定义存储年月日的变量
int year = 0, month = 0, day = 0;
//提示用户输入
printf("请输入年月日(比如 1990-1-1):");
//接受用户输入, 切记,scanf 中""内的格式是什么, 输入的格式必须一致
scanf("%d-%d-%d", &year,&month, &day);
//定义一个数组存放每个月的天数
int dayOfMonth[12] = {31, 28, 31, 30, 31,30, 31, 31, 30, 31, 30, 31};
//判断是否不是闰年
if (year % 400 == 0 || (year % 4 == 0&& year % 100 != 0)) {
//闰年二月 29 天
    dayOfMonth[1] = 29;
}
//定义变量记录是第几天
int whichDay = 0;
//例如 3 月 20 日是一年的第几天, 计算方法 1 月的天数 + 2 月的天数 + 20
for (int i = 0; i < month - 1; i++) {
    whichDay += dayOfMonth[i];
}
whichDay += day;
printf("是一年的%d 天\n", whichDay);
```

## 46.利润与奖金，某公司销售 10 万元到 20 万元的奖金 10%，在 20 万元的奖金 10 万元以上的奖金 7.5%，到 40 万元超出 20 万元的部分奖金为 5%,到 60 万元的超出 40 万元的部分奖金 3%，到 100 万元的超出 60 万元部分奖金 1%，请输出说的奖金

```
public class Test1 {
    public static void main(String[] args) {
        float jiangjin=0;
        Scanner scan=new Scanner(System.in);
        System.out.print("请输入利润：  ");
        float num=scan.nextInt();
        if(num<=100000){
            jiangjin=(float) (num*0.1);
        }
        else if(num<=200000){
            jiangjin=(float) ((num-100000)*0.075+100000*0.1);
        }
        else if(num<=400000){
            jiangjin=(float) ((num-200000)*0.5 +100000*0.175);
        }
        else if(num<=600000){
            jiangjin=(float) ((num-400000)*0.3 +100000*0.175+200000*0.5);
        }
        else if(num<=1000000){
            jiangjin=(float) ((num-600000)*0.015
+100000*0.175+200000*0.5+200000*0.3);
        }
        else{
            jiangjin=(float) ((num-1000000)*0.01
+100000*0.175+200000*0.5+200000*0.3+400000*0.015);
        }
        System.out.println("奖金：  "+jiangjin);
    }
}
```

## 47.请描述工作流机制（ JBPM/BPEL ）等

参考： http://blog.csdn.net/leroy008/article/details/8058187

## 48.除了懒汉式和饿汉式你还了解那些单利模式？

```
第一种： 双重查锁模式
public class DoubleCheckLock {
    private static DoubleCheckLock instance = null;
    private DoubleCheckLock() {
    }
    public static DoubleCheckLock getInstance() {
        if (instance == null) {
            synchronized (DoubleCheckLock.class) {
                if (instance == null) {
                    instance = new DoubleCheckLock();
                }
            }
        }
```

```
            return instance;
        }
    }

    第二种：枚举单例
    public enum SingleEnum {
        INSTANCE;
        public void doSomething() {
            ToastUtils.showLongToast("do something...");
        }
    }

    第三种：静态内部类方式
    public class StaticInner {
        private static StaticInner instance;
        public static StaticInner getInstance() {
            return SingletonHolder.STATIC_INNER;
        }
        private static class SingletonHolder {
            private static final StaticInner STATIC_INNER = new StaticInner();
        }
    }
```

## 49.简述 SSH 的概念以及中主要的设计思想？

SSH 是 struts+spring+hibernate 的一个集成框架，是目前比较流行的一种 Web 应用程序开源框架。集成 SSH 框架的系统从职责上分为四层：表示层、业务逻辑层、数据持久层和域模块层，以帮助开发人员在短期内搭建结构清晰、可复用性好、维护方便的 Web 应用程序。其中使用 Struts 作为系统的整体基础架构，负责 MVC 的分离，在 Struts 框架的模型部分，控制业务跳转，利用 Hibernate 框架对持久层提供支持， Spring 做管理，管理 struts
和 hibernate。具体做法是：用面向对象的分析方法根据需求提出一些模型，将这些模型实现为基本的 Java 对象，然后编写基本的 DAO(Data Access Objects)接口，并给出 Hibernate 的 DAO 实现，采用 Hibernate 架构实现的DAO类来实现 Java 类与数据库之间的转换和访问，最后由 Spring 做管理

## 50.unix 下如何让命令在后台执行？

要让程序在后台执行，只需在命令行的最后加上“&”符号。
例如： $ find . -name abc -print& ；

## 51.rm-i 与 rm-r 个实现什么功能？

rm -i --interactive 交互模式删除文件，删除文件前给出提示。
rm -r -r 或-R：递归处理，将指定目录下的所有文件与子目录一并处理

## 52.什么是乐观锁，什么是悲观锁，两者的区别是什么

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。它指的是对数据被外界（包括本系统当前的其他事务，以及来自外部系统的事务处理）修改持保守态度，因此，在整个数据处理过程中，将数据处于锁定状态。

悲观锁的实现，往往依靠数据库提供的锁机制（也只有数据库层提供的锁机制才能真正保证数据访问的排他性，否则，即使在本系统中实现了加锁机制，也无法保证外部系统不会修改数据）。

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

## 53.日志打印的 log4j 的配置中%t 表示什么？

答案： %t 输出产生该日志事件的线程名
扩展： %M 是输出方法的名字、 %m 是输出代码指定的日志信息。
指定的打印信息的具体格式 ConversionPattern，具体参数：
%m 输出代码中指定的消息
%p 输出优先级，即 DEBUG， INFO， WARN， ERROR， FATAL
%r 输出自应用启动到输出该 log 信息耗费的毫秒数
%c 输出所属的类目，通常就是所在类的全名
%t 输出产生该日志事件的线程名
%n 输出一个回车换行符， Windows 平台为"rn”， Unix 平台为"n”
%d 输出日志时间点的日期或时间，默认格式为 ISO8601，也可以在其后指定格式，比如： %d{yyyy MM dd HH:mm:ss,SSS}，输出类似： 2002 年 10 月 18 日 22： 10： 28， 921 %l 输出日志事件的发生位置，包括类目名、发生的线程，以及在代码中的行数。
%x: 输出和当前线程相关联的 NDC(嵌套诊断环境),尤其用到像 java servlets 这样的多客户多线程的应用中。
%%: 输出一个” %”字符
%F: 输出日志消息产生时所在的文件名称
%M: 输出执行方法
%L: 输出代码中的行号

## 54.Spring 中什么时候引起 NotWritablePropertyException 和 Could not open calss path resource[ApplicationContext.xml]

出现 NotWritablePropertyException 异常的原因一般是在 ApplicationContext.xml 中 property name 的错误
等相关错误