1. J2EE（总分20分。每题2分）
   1. 简述String和StringBuffer的区别。

String 对一串字符进行操作。不可变。一旦被创建，就不能修改它的值。

StringBuffer 也是对一串字符进行操作，但是可变类。

* 1. 运行时异常（RuntimeException）与一般异常有何不同？

一般异常是可捕捉到的。运行时异常是不可预知的异常。

* 1. Sleep（）和wait（）有什么区别？

sleep是线程类（Thread）的方法，导致此线程暂停执行指定时间，把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().会释放锁

* 1. 实现多线程有几种方法？

三种，

1.继承Thread类，重写run函数

2.实现Runnable接口，重写run函数

3.实现Callable接口，重写call函数

* 1. 简述线程的基本状态及状态之间的关系。

Thread.ThreadState 属性提供一个位掩码，用它指示线程的当前状态。一个线程至少总是处于 ThreadState 枚举中一个可能状态，并且可以同时处于多个状态。

一旦线程由于调用 Thread.Start 而离开 Unstarted 状态，则它将永远无法返回到 Unstarted 状态。同样，线程也永远无法离开 Stopped 状态。

* 1. 什么是线程同步，何如实现线程的同步？

当两个或多个线程需要访问同一资源时，它们需要以某种顺序来确保该资源某一时刻只能被一个线程使用的方式称为同步。要想实现同步操作，必须要获得每一个线程对象的锁。获得它可以保证在同一时刻只有一个线程访问对象中的共享关键代码，并且在这个锁被释放之前，其他线程就不能再进入这个共享代码。此时，如果还有其他线程想要获得该对象的锁，只得进入等待队列等待。只有当拥有该对象锁的线程退出共享代码时，锁被释放，等待队列中第一个线程才能获得该锁，从而进入共享代码区。

* 1. JSP或Servlet中的forward和redirect有什么区别？

forward是服务器内部重定向，程序收到请求后重新定向到另一个程序，客户机并不知道；redirect则是服务器收到请求后发送一个状态头给客户，客户将再请求一次，这里多了两次网络通信的来往。redirect 是送到客户端后再一次 request , 所以资料不被保留.

* 1. 简述JDBC调用数据库的基本步骤。

1,加载驱动  
2,创建连接  
3,获取语句对象  
4,执行sql语句  
5,如果是查询,还可以使用结果集  
6,关闭连接  
7,捕捉和处理异常

* 1. 简述Servlet的生命周期。

Servlet被服务器实例化后，容器运行其init方法，请求到达时运行其service方法，service方法自动派遣运行与请求对应的do方法（doGet，doPost）等，当服务器决定将实例销毁的时候调用其destroy方法。

1. 算法（总共20分）
   1. 请列举几种排序算法，并用JAVA实现快速排序算法。（6分）

冒泡排序，快速排序，shaker排序，[堆排序](http://zhidao.baidu.com/search?word=%E5%A0%86%E6%8E%92%E5%BA%8F&fr=qb_search_exp&ie=utf8)。

public void quick(Integer[] str) {

if (str.length > 0) { //查看数组是否为空

\_quickSort(str, 0, str.length – 1);

}

}

public void \_quickSort(Integer[] list, int low, int high) {

if (low < high) {

int middle = getMiddle(list, low, high); //将list数组进行一分为二

\_quickSort(list, low, middle – 1); //对低字表进行递归排序

\_quickSort(list, middle + 1, high); //对高字表进行递归排序

}

}

public int getMiddle(Integer[] list, int low, int high) {

int tmp = list[low]; //数组的第一个作为中轴

while (low < high) {

while (low < high && list[high] > tmp) {

high–;

}

list[low] = list[high]; //比中轴小的记录移到低端

while (low < high && list[low] < tmp) {

low++;

}

list[high] = list[low]; //比中轴大的记录移到高端

}

list[low] = tmp; //中轴记录到尾

return low; //返回中轴的位置

}

* 1. 用Java实现二叉树前序遍历、中序遍历和后序遍历。（8分）

public class Tree {

     private int data;// 数据节点  
     private Tree left;// 左子树  
     private Tree right;// 右子树

 public Tree(int data) {  
           this.data = data;  
           this.left = null;  
           this.right = null;  
    }

 /\*\*  
  \* 创建二叉树，返回根结点  
  \*/  
 public static Tree createTree(int[] input) {  
       Tree root = null;  
       Tree temp = null;  
       for (int i = 0; i < input.length; i++) {  
             // 创建根节点  
             if (root == null) {  
                root = temp = new Tree(input[i]);  
             } else {  
             // 回到根结点  
                  temp = root;  
            // 添加节点  
                 while (temp.data != input[i]) {  
                           if (input[i] <= temp.data) {  
                                 if (temp.left != null) {  
                                       temp = temp.left;  
                                 } else {  
                                       temp.left = new Tree(input[i]);  
                                }  
                           } else {  
                                if (temp.right != null) {  
                                     temp = temp.right;  
                                } else {  
                                     temp.right = new Tree(input[i]);  
                                }  
                           }  
                   }  
          }  
    }  
  return root;  
 }

 /\*\*  
  \* 前序遍历  
  \*/  
 public static void preOrder(Tree tree) {  
  if (tree != null) {  
   System.out.print(tree.data + “”);  
   preOrder(tree.left);  
   preOrder(tree.right);  
  }  
 }

 /\*\*  
  \* 中序遍历  
  \*/  
 public static void midOrder(Tree tree) {  
  if (tree != null) {  
   midOrder(tree.left);  
   System.out.print(tree.data + “”);  
   midOrder(tree.right);  
  }  
 }

 /\*\*  
  \* 后序遍历  
  \*/  
 public static void posOrder(Tree tree) {  
  if (tree != null) {  
   posOrder(tree.left);  
   posOrder(tree.right);  
   System.out.print(tree.data + “”);  
  }  
 }  
   
 /\*\*  
  \* 求二叉树的深度  
  \*/  
 public static int length(Tree tree){  
  int depth1;  
  int depth2;  
  if(tree == null) return 0;  
  //左子树的深度  
  depth1 = length(tree.left);  
  //右子树的深度  
  depth2 = length(tree.right);  
  if(depth1>depth2)  
   return depth1+1;  
  else  
   return depth2+1;  
 }  
 public static void main(String[] args) {  
  int[] input = { 4, 2, 6, 1, 3, 5, 7,8,10 };  
  Tree tree = createTree(input);  
  System.out.print(“前序遍历：“);  
  preOrder(tree);  
  System.out.print(“\n中序遍历：“);  
  midOrder(tree);  
  System.out.print(“\n后序遍历：“);  
  posOrder(tree);  
 }  
}

* 1. 阅读代码，并回答问题（6分）

|  |
| --- |
| Public String listToString( Vector strList) {  String str = new String();  SortedSet set = new TreeSet();  set.addAll(strList)；  for(Iterator iter=set.iterator(); iter.hasNext();  String currStr = (String) iter.next();  Str += currStr + “;”;  }  return str; |

问题1：

这段代码完成什么功能？如果输入列表{“To” , “SAIC” , “Welcome”}，输入结果是什么？

数组元素拼接。TreeSet是按字母顺序存储的，结果是：SAIC;To;Welcome;

问题2这段代码中有什么地方有错误或者可以改善的地方吗？

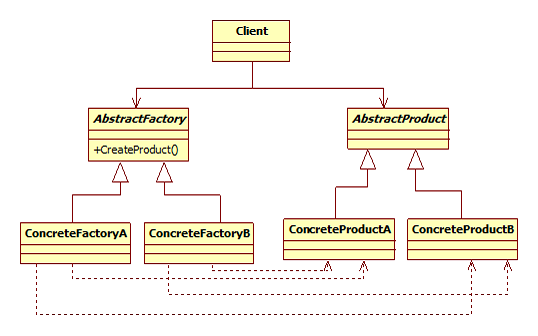
错误：for(Iterator iter=set.iterator(); iter.hasNext();后面缺少“)”

改善：拼接到最后时，最后一个分号应去掉。

1. 设计模式（总分10分，每题5分）
   1. 编程实现设计模式：Singleton（单例模式）。

class Singleton {  
private static Singleton s;  
private Singleton(){  
}  
public static Singleton getSigleton()  
{  
if(s==null)s=new Singleton();  
return s;  
}  
}

* 1. 画出抽象工厂（Abstract Factory）的UML类图。



1. 数据库（总分20分）
   1. SOL查询语句的左外连接、右外连接、全外连接、内联接之间的区别。

左外连接的结果集包括LEFT OUTER子句中指定的左表的所有行，而不仅仅是连接列所匹配的行

右外连接是左向外连接的反向连接。将返回右表的所有行。

全外连接返回左表和右表中的所有行。当某行在另一个表中没有匹配行时，则另一个表的选择列表列包含空值。如果表之间有匹配行，则整个结果集行包含基表的数据值。

内连接使用比较运算符根据每个表共有的列的值匹配两个表中的行。

* 1. 举例说明GROUP BY 和HAVING 子句的意义和使用方法。

每个部门有多少人  
就要用到GROUP BY  
select DepartmentID as ‘部门名称‘,  
COUNT(\*) as ‘个数‘ from BasicDepartment group by DepartmentID

1. 显示每个地区的总面积．仅显示那些面积超过1000000的地区。
2. SELECT region, SUM(area)
3. FROM bbc
4. GROUP BY region
5. HAVING SUM(area)>1000000
   1. IN和 EXISTS的区别（4分）

比如in(1,2) 就是 = 1 or = 2的一种简单写法，所以一般在元素少的时候使用IN，如果多的话就用exists指定一个子查询。

* 1. 编写SQL语句

为管理岗位业务培训信息，建立3个表。

S（S#，SN，SD，SA）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)C#，CN分别代表课程编号，课程名称。

SC（S#，C#，G）S#，C#，G 分别代表学号，课程编号、学习成绩

问题1：使用标准嵌套SQL语句查询选修课程名称为“税收基础“的学员学号和姓名。

Select

ssc s#, ssc.sn

from c

left join

( select s.\*,sc.c#

From sc

Left join s

on s.s#=sc.s#

) as ssc

on ssc.c#=c.c#

where c.cn=’税收基础‘

问题2：查询选修课程为“C2″的所有学员的学号、姓名、所属单位、成绩，并按照所属单位升序、成绩降序排列。

select sc.c#,s.sn,s.sd,sc.g

From sc

Left join s

on s.s#=sc.s#

where c. c#=’ C2′

order by s.sd asc,sc.g.desc

问题3：查询所有学员选修课程的数量，列出学号、姓名、所选课程数，并按照课程数降序排列。

Select sc.s#,s.sn, sc.count

(

select count(s#) as count,s#

from sc

group by sc.s#

) sc

Left join s

on s.s#=sc.s#

order by sc.count desc

问题4：查询选修课程数超过5次学员的学号、姓名、所属单位、所选课程数，并按照学号升序排列。

SELECT

S#,

SN,

SD,

(SELECTCOUNT(DISTINCT C#)FROM SC

Where S#=s. S#

) as count

FROM S

WHERE S# IN(

        SELECT [S#] FROM SC

        GROUP BY [S#]

        HAVING COUNT(DISTINCT [C#])>5)

1. 框架（总分30分）

JSF

* 1. 以JSF为例简述MVC设计模式。
  2. 简述JSF请求处理三命周期

1. 恢复视图  
2. 应用请求的值；处理验证  
3. 更新模型值；处理事件  
4. 调用程序；处理事件  
5. 进行响应；处理事件

* 1. JSF中的Backing Bea是干什么用的？

Backing bean定义了页面上的UI组件的属性和处理逻辑。每一个backing bean的属性对应一个组件或者组件的值。Backing bean同时定义了一组执行组件功能的方法，比如验证组件的数据，处理组件触发的事件，当组件activate时处理与导航相关的操作。

* 1. JSF中的Converter是干什么用的？

输入变换：在用户提交时对用户输入值进行变换

输出变换：在画面显示前对显示值进行变换

* 1. JSF中的Validator是干什么用的？

对用户输入的数据进行验证。

* 1. 简述Value-change Events、Action Events、Data Mooel Events、Phase Events。

值改变事件，动作事件，数据模型事件，阶段事件

* 1. JSF Request Processing Lifecycle 的六个阶段。

回复画面（Restore View）

套用申请值（Apply Request Values）

执行验证（Process Validations）

更新模型值（Update Model Values）

唤起应用程序（Invoke Application）

绘制回应（Render Response）

* 1. CommandButton 的immediate属性起什么作用？

JSF视图组件在取得请求中该取得的值之后，即立即处理指定的事件，而不再进行后续的转换器处理、验证器处理、更新模型值等流程。

* 1. <h：panelGroup>的用途

将封装在内的元件作为一个元件来看待

* 1. <f：verbatim>的用途

想要放入非JSF元件,例如简单的模板(template)文字，那么就需要使用<f:verbatim /> 标签来处理。

Spring

1. 简述依赖注入（Dependency Injeciton，DI）和控制反转（Inversion ofControl，IOC ）的基本概念。

依赖注入DI是一个程序设计模式和架构模型，  
一些时候也称作控制反转，尽管在技术上来讲，依赖注入是一个IOC的特殊实现，依赖注入是指一个对象应用另外一个对象来提供一个特殊的能力，例如：把一个数据库连接已参数的形式传到一个对象的结构方法里面而不是在那个对象内部自行创建一个连接。控制反转和依赖注入的基本思想就是把类的依赖从类内部转化到外部以减少依赖  
应用控制反转，对象在被创建的时候，由一个调控系统内所有对象的外界实体，将其所依赖的对象的引用，传递给它。也可以说，依赖被注入到对象中。所以，控制反转是，关于一个对象如何获取他所依赖的对象的引用，这个责任的反转。

1. 简述AOP基本概念（Advice、Joinpoint、Polntcut、Aspect）以及在Spring AOP实现的特点和局限。

Advice：用于定义拦截行为

JoinPoint：提供访问当前被通知方法的目标对象、代理对象、方法参数等数据

Polntcut：捕获所有的连接点在指定的方法执行中，包括执行方法本身

Aspect：切入点指示符用来指示切入点表达式目的。

1. Spring Autorwiring by name 和Autowiring by type是什么意思？

byType是通过类型在“MyIdProviderManager”类里找属性名字为“myIdProviderManager”的对象，然后为其注入  
byName是通过名称找的。