Neusoft 东软

Java 面试笔试题汇总

一、编程语言（Java）

1、Java 多态的实现

答案：继承、重载、覆盖

2、String 类是否能被继承

答案：不能，是个 final 修饰的类。

3、访问修饰符“public/private/protected/缺省的修饰符”的使用

答案：

作用域 当前类 同一 package 子孙类 其他 package

public √ √ √ √
protected √ √ √ ×
friendly √ √ × ×
private √ × × ×

不写时默认为 friendly

4、用关键字 final 修饰一个类或者方法时，有何意义？

答案：用 final 修饰的类不能被继承，使用 final 修饰的方法不能被覆盖。

5、谈谈 final，finally，finalize 的区别。

答案：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。因此一个类不能既被声明为 abstract 的，又被声明为 final 的。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?再异常处理时提供 finally 块来执行任何清除操作。如果抛出一个异常，那么相匹配的 catch 子句就会执行，然后控制就会进入 finally 块（如果有的话）。

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的清理工作。这个方法是由垃圾收集器在确定这个对象没有被引用时对这个对象调用的。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。finalize() 方法是在垃圾收集器删除对象之前对这个对象调用的。

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

答案: 匿名的内部类是没有名字的内部类。不能 extends(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。

7、HashMap 和 Hashtable 的区别。

答案: 都属于 Map 接口的类，实现了将惟一键映射到特定的值上。

HashMap 类没有分类或者排序。它允许一个 null 键和多个 null 值。

Hashtable 类似于 HashMap，但是不允许 null 键和 null 值。它也比 HashMap 慢，因为它是同步的。

8、Collection 和 Collections 的区别。

答案: Collections 是个 java.util 下的类，它包含有各种有关集合操作的静态方法。

Collection 是个 java.util 下的接口，它是各种集合结构的父接口。

9、 sleep() 和 wait() 有什么区别? 搞线程的最爱

答案: sleep()方法是使线程停止一段时间的方法。在 sleep 时间间隔期满后，线程不一定立即恢复执行。这是因为在那个时刻，其它线程可能正在运行而且没有被调度为放弃执行，除非(a)“醒来”的线程具有更高的优先级

(b)正在运行的线程因为其它原因而阻塞。

wait()是线程交互时，如果线程对一个同步对象 x 发出一个 wait()调用，该线程会暂停执行，被调对象进入等待状态，直到被唤醒或等待时间到。

10、Java 有没有 goto?

答案：Goto?java 中的保留字，现在没有在 java 中使用。

11、 数组有没有 length()这个方法? String 有没有 length()这个方法？

答案：数组没有 length()这个方法，有 length 的属性。

String 有有 length()这个方法

12、 构造器 Constructor 是否可被 override?

答案：构造器 Constructor 不能被继承，因此不能重写 Overriding，但可以被重载 Overloading。

13、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?

答案：会执行，在 return 前执行。

14、 面向对象的特征有哪些方面

答案:（1）.抽象:

抽象就是忽略一个主题中与当前目标无关的那些方面,以便更充分地注意与当前目标有关的方面。抽象并不打算了解全部问题，而只是选择其中的一部分，暂时不用部分细节。抽象包括两个方面，一是过程抽象，二是数据抽象。

（2）.继承:

继承是一种联结类的层次模型，并且允许和鼓励类的重用，它提供了一种明确表述共性的方法。对象的一个新类可以从现有的类中派生，这个过程称为类继承。新类继承了原始类的特性，新类称为原始类的派生类（子类），而原始类称为新类的基类（父类）。派生类可以从它的基类那里继承方法和实例变量，并且类可以修改或增加新的方法使之更适合特殊的需要。

（3）.封装:

封装是把过程和数据包围起来，对数据的访问只能通过已定义的界面。面向对象计算始于这个基本概念,即现实世界可以被描绘成一系列完全自治、封装的对象,这些对象通过一个受保护的接口访问其他对象。

（4）. 多态性:

多态性是指允许不同类的对象对同一消息作出响应。多态性包括参数化多态性和

包含多态性。多态性语言具有灵活、抽象、行为共享、代码共享的优势，很好的解决了应用程序函数同名问题。

15、说出 ArrayList,Vector, LinkedList 的存储性能和特性

答案: ArrayList 和 Vector 都是使用数组方式存储数据，此数组元素数大于实际存储的数据以便增加和插入元素，它们都允许直接按序号索引元素，但是插入元素要涉及数组元素移动等内存操作,所以索引数据快而插入数据慢,Vector 由于使用了 synchronized 方法（线程安全），通常性能上较 ArrayList 差，而 LinkedList 使用双向链表实现存储，按序号索引数据需要进行前向或后向遍历，但是插入数据时只需要记录本项的前后项即可，所以插入速度较快。

16、多线程有几种实现方法,都是什么?同步有几种实现方法,都是什么?

答: 多线程有两种实现方法，分别是继承 Thread 类与实现 Runnable 接口

同步的实现方面有两种，分别是 synchronized,wait 与 notify

17、jsp 有哪些内置对象?作用分别是什么?

答:JSP 共有以下 9 种基本内置组件（可与 ASP 的 6 种内部组件相对应）:

request 用户端请求，此请求会包含来自 GET/POST 请求的参数

response 网页传回用户端的回应

pageContext 网页的属性是在这里管理

session 与请求有关的会话期

application servlet 正在执行的内容

out 用来传送回应的输出

config servlet 的构架部件

page JSP 网页本身

exception 针对错误网页，未捕捉的例外

18、jsp 有哪些动作?作用分别是什么?

答:JSP 共有以下 6 种基本动作

jsp:include: 在页面被请求的时候引入一个文件。

jsp:useBean: 寻找或者实例化一个 JavaBean。

jsp:setProperty: 设置 JavaBean 的属性。

jsp:getProperty: 输出某个 JavaBean 的属性。

jsp:forward: 把请求转到一个新的页面。

jsp:plugin: 根据浏览器类型为 Java 插件生成 OBJECT 或 EMBED 标记

19、两种跳转方式分别是什么?有什么区别?

答: 有两种，分别为:

```
<jsp:include page="included.jsp" flush="true">
<jsp:forward page= "nextpage.jsp"/>
```

前者页面不会转向 include 所指的页面，只是显示该页的结果，主页面还是原来的页面。执行完后还会回来，相当于函数调用。并且可以带参数.后者完全转向新页面，不会再回来。相当于 go to 语句。

20、forward 和 redirect 的区别

答案: forward 是服务器请求资源，服务器直接访问目标地址的 URL，把那个 URL 的响应内容读取过来，然后把这些内容再发给浏览器，浏览器根本不知道服务器发送的内容是从哪儿来的，所以它的地址栏中还是原来的地址。
redirect 就是服务端根据逻辑,发送一个状态码,告诉浏览器重新去请求那个地址，一般来说浏览器会用刚才请求的所有参数重新请求，所以 session,request 参数都可以获取。

21、说出 Servlet 的生命周期，并说出 Servlet 和 CGI 的区别。

答案: Servlet 被服务器实例化后，容器运行其 init 方法，请求到达时运行

其 service 方法，service 方法自动派遣运行与请求对应的 doXXX 方法（doGet，doPost）等，当服务器决定将实例销毁的时候调用其 destroy 方法。

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。

22、EJB 是基于哪些技术实现的？并说出 SessionBean 和 EntityBean 的区别，StatefulBean 和 StatelessBean 的区别。

答案：EJB 包括 Session Bean、Entity Bean、Message Driven Bean，基于 JNDI、RMI、JAT 等技术实现。

SessionBean 在 J2EE 应用程序中被用来完成一些服务器端的业务操作，例如访问数据库、调用其他 EJB 组件。EntityBean 被用来代表应用系统中用到的数据。

对于客户机，SessionBean 是一种非持久性对象，它实现某些在服务器上运行的业务逻辑。

对于客户机，EntityBean 是一种持久性对象，它代表一个存储在持久性存储器中的实体的对象视图，或是一个由现有企业应用程序实现的实体。

Session Bean 还可以再细分为 Stateful Session Bean 与 Stateless Session Bean，这两种的 Session Bean 都可以将系统逻辑放在 method 之中执行，不同的是 Stateful Session Bean 可以记录呼叫者的状态，因此通常来说，一个使用者会有一个相对应的 Stateful Session Bean 的实体。Stateless Session Bean 虽然也是逻辑组件，但是他却不负责

记录使用者状态，也就是说当使用者呼叫 Stateless Session Bean 的时候，EJB Container 并不会找寻特定的 Stateless Session Bean 的实体来执行这个 method。换言之，很可能数个使用者在执行某个 Stateless Session Bean 的 methods 时，会是同一个 Bean 的 Instance 在执行。从内存方面来看， Stateful Session Bean 与 Stateless Session Bean 比较， Stateful Session Bean 会消耗 J2EE Server 较多的内存，然而 Stateful Session Bean 的优势却在于他可以维持使用者的状态。

23、EJB 与 JAVA BEAN 的区别？

答: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 Java Bean 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。

24、简单介绍您所了解的 MVC。

答案：略

二、软件工程方面

1.软件开发生命周期有哪几个阶段?

答案：1)、问题的定义及规划

2)、需求分析

3)、软件设计

4)、程序编码

5)、软件测试

6)、运行维护

2、什么是 CMM？划分为哪几级、分别是什么？

答案：CMM 是软件能力成熟度模型，是一种用于评价软件承包能力并帮助其改善软件质量的方法，侧重于软件开发过程的管理及工程能力的提高与评估。CMM 分为五个等级：一级为初始级，二级为可重复级，三级为已定义级，四级为已管理级，五级为优化级。

三、智力测试

1 、有两根不均匀分布的香，香烧完的时间是一个小时，你能用什么方法来确定一段 15 分钟的时间？

答案： 一只两头点燃，另一只一头点燃，当第一只烧完后，第二只丙再头点燃，就可以得到 15`

2、有三个人去住旅馆，住三间房，每一间房$10 元，于是他们一共付给老板$30，第二天，老板觉得三间房只需要$25 元就够了于是叫小弟退回$5 给三位客人，

谁知小弟贪心,只退回每人$1，自己偷偷拿了$2，这样一来便等于那三位客人每人各花了九元，

于是三个人一共花了$27，再加上小弟独吞了不$2，总共是$29。可是当初他们三个人一共付出$30 那么还有$1 呢？

答案：怎么会是每人第天九元呢,每人每天 (25/3) + 1，那一元差在

25 - 24 = 1

3、有两位盲人，他们都各自买了两对黑袜和两对白袜，八对袜了的布质、大小完全相同，
而每对袜了都有一张商标纸连着。两位盲人不小心将八对袜了混在一起。他们每人怎样才能取回黑袜和白袜各两对呢？
答案：每人取每双中的一只就可以了

4、有一辆火车以每小时 15 公里的速度离开洛杉矶直奔纽约，另一辆火车以每小时 20 公里的速度从纽约开往洛杉矶。如果有一只鸟，以 30 公里每小时的速度和两辆火车同时启动，从洛杉矶出发，碰到另一辆车后返回，依次在两辆火车来回飞行，直到两辆火车相遇，请问，这只小鸟飞行了多长距离？

答案：(D / 35) * 30 = D

5、你有两个罐子，50 个红色弹球，50 个蓝色弹球，随机选出一个罐子，随机选取出一个弹球放入罐子，怎么给红色弹球最大的选中机会？在你的计划中，得到红球的准确几率是多少？

答案：自己睁着眼睛挑一个红色的啊，这样是给红色最大的机会了，除了你是色盲，呵呵 ,当然他们的几率都是 1/2。

6、你有四个装药丸的罐子，每个药丸都有一定的重量，被污染的药丸是没被污染的重量＋1.只称量一次，如何判断哪个罐子的药被污染了？

答案：一个中取一个编号，然后称一下就知道

7、你有一桶果冻，其中有％％，绿色，红色三种，闭上眼睛，抓取两个同种颜色的果冻。抓取多少个就可以确定你肯定有两个同一颜色的果冻？

答案：4 个

8、对一批编号为 1～100，全部开关朝上(开)的灯进行以下*作：凡是 1 的倍数反方向拨一次开关；2 的倍数反方向又拨一次开关；3 的倍数反方向又拨一次开关……问：最后为关熄状态的灯的编号。

9 想象你在镜子前，请问，为什么镜子中的影像可以颠倒左右，却不能颠倒上下？

答案：因为照镜子时，镜子是与你垂直平行的，但在水平方向刚好转了 180 度。

10 一群人开舞会，每人头上都戴着一顶帽子。帽子只有黑白两种，黑的至少有一顶。每个人都能看到其它人帽子的颜色，却看不到自己的。主持人先让大家看看别人头上戴的是什么帽子，然后关灯，如果有人认为自己戴的是黑帽子，就打自己一个耳光。第一次关灯，没有声音。于是再开灯，大家再看一遍，关灯时仍然鸦雀无声。一直到第三次关灯，才有劈劈啪啪打耳光的声音响起。问有多少人戴着黑帽子？

答案：应该是三个人：

1，若是两个人，设 A、B 是黑帽子,第二次关灯就会有人打耳光。原因是 A 看到 B 第一次没打耳光，就知道 B 也一定看到了有带黑帽子的人，可 A 除了知道 B 带黑帽子外，其他人都是白帽子，就可推出他自己是带黑帽子的人！同理 B 也是这么想的，这样第二次熄灯会有两个耳光的声音。

2，如果是三个人，A,B,C．A 第一次没打耳光，因为他看到 B,C 都是带黑帽子的；而且假设自己带的是白帽子，这样只有 BC 戴的是黑帽子；按照只有两个人带黑帽子的推论，第二次应该有人打耳光；可第二次却没有。。。于是他知道 B 和 C 一定看到了除 BC 之外的其他人带了黑帽子，于是他知道 BC 看到的那个人一定是他，所以第三次有三个人打了自己一个耳光！

3，若是第三次也没有人打耳光，而是第四次有人打了耳光，那么应该有几个人带了黑猫子呢？大家给个结果看看^_^

11 两个圆环，半径分别是 1 和 2，小圆在大圆内部绕大圆圆周一周，问小圆自身转了几周？如果在大圆的外部，小圆自身转几周呢？

答案：可以把圆看成一根绳子，大绳是小绳的 2 倍长，所以应该是 2 圈吧

12 1元钱一瓶汽水，喝完后两个空瓶换一瓶汽水，问：你有20元钱，最多可以喝到几瓶汽水？

答案：一开始20瓶没有问题，随后的10瓶和5瓶也都没有问题，接着把5瓶分成4瓶和1瓶，前4个空瓶再换2瓶，喝完后2瓶再换1瓶，此时喝完后手头上剩余的空瓶数为2个，把这2个瓶换1瓶继续喝，喝完后把这1个空瓶换1瓶汽水,喝完换来的那瓶再把瓶子还给人家即可,所以最多可以喝的汽水数为:

20＋10＋5＋2＋1＋1＝40

四、数据库

1、设有关系R(S,D,M)其函数一览集F={S→D，D→M}。则关系R至多满足_____。

A. INF B. 2NF C. 3NF D. 4NF

答案：B

2、which are DML statements(choose all that apply)(下面哪个是DML语句多选)

A. commit B. merge C.update D.delete E. creat F. drop

答案：C，D

3、whice select statement will the result ‘ello world’from the string’Hello world’?（如果要从字符串“Hello world”中提取出“ello world”这样的结果，下面的哪条SQL语句适合？）

```
A. select substr('Hello World',1)from dual;
B. select substr('Hello World',1,1)from dual;
C. select lower(substr('Hello world',1))from dual;
D. select lower(trim('H' from 'Hello world') )from dual;
```

答案：D

3、储存过程和函数的区别是什么

答案：存储过程是用户定义的一系列 SQL 语句的集合，涉及特定表或其他对象的任务，用户可以调用存储过程。而函数通常是数据库已经定义的方法，它接受参数并返回某种类型的值，并且不涉及特定用户表。

4、事务是什么？

答案：1）原子性：事务必须是原子工作单元。对于其数据修改，要么全都执行，要么全都不执行。

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

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

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

5、游标的作用是什么？如何知道游标已经到了最后？

答案：游标用于定位结果集的行。通过判断全局变量@@FETCH_STATUS 可以判断其是否到了最后。通常此变量不等于 0 表示出错或到了最后。

6、触发器分为事前触发和事后触发，这两种触发有什么区别？语句级出发和行级触发有何区别？

答案：事前触发器运行于触发事件运行之前，而事后触发器运行于触发事件发生之后。语句级触发器可以在语句执行前或后执行，而行级触发在触发器所影响的每一行触发一次。

7、找出表 ppp 里面 num 最小的数，不能使用 min 函数

答案：
```
select * from ppp where num<=all(select num from ppp)
```
或者
```
Select top 1 num from ppp order by num
```

8、找出表 ppp 里面最小的数，可以使用 min 函数

答案：
```
select * from ppp where num = (select Min(num)from ppp)
```

9、选择表 ppp2 中 num 重复的纪录

答案：select * from ppp2 where num in(select num from ppp2 group by num having(count(num)>1))

10、为了防止在查询记录的时候被其他用户更改记录，应该采用什么方法？如何用查询语句实现该方法？

答案：添加一个"时间戳"类型的字段就可以了。 Timestamp 这种数据类型会根据当前时间自动产生一个时间字符串，确保这些数在数据库中是唯一的。Timetamp 一般用做给表行加版本戳的机制，存储大小为 8 字节。一个标志能有一个 timestamp 列。每次插入或更新包含 timestamp 列的行时，timestamp 列中的值均会更新。这一属性使 timestamp 列不适合为关键是用，尤其是不能作为主键是用。对行的任何更新都会更改 timestamp 值，从而更改键值。

五、Java 框架方面

1、关键的 Struts 标签种类？

Bean 标签、html 标签、logic 标签、nested 标签。

2、EJB 的三种 bean？

会话 bean、实体 bean、消息驱动 bean。

3、什么是 Ajax？

Ajax 的全称是：AsynchronousJavaScript+XML

Ajax（AsynchronousJavaScriptandXML）是结合了 Java 技术、XML 以及 JavaScript 等编程技术，可以让开发人员构建基于 Java 技术的 Web 应用，并打破了使用页面重载的惯例。

4、简单地说一下什么是 Hibernate?

Hibernate 是 ORmaping 的一种实现

是目前在 JAVA 界使用非常广泛的 ORMapping 的一种实现

可以实现关系型数据库和对象之间的映射。
用来开发数据库系统非常方便。
可以将数据库和程序的设计融合在一起
不会在出现以前那样程序是面向对象的，但是一到数据库那里就乱套了的想象。

5、Struts 工作流程是什么？

一个用户的请求是通 ActionServlet 来处理和转发的。那么，ActionServlet 如何决定把用户请求转发给哪个 Action 对象呢？这就需要一些描述用户请求路径和 Action 衍射关系的配置信息了。在 Struts 中，这些配置映射信息都存储在特定的 XML 文件 Struts－config.xml 中。在该配置文件中，每一个 Action 的映射信息都通过一个<Action>元素来配置。

这些配置信息在系统启动的时候被读入内存，供 Struts 在运行期间使用。在内存中，每一个 <action> 元素都对应一个 org.apache.struts.action.ActionMapping 类的实例。

对于采用 Struts 框架的 web 应用，在 web 应用启动时就会加载并初始化 ActionServlet，ActionServlet 从 struts-config.xml 文件中读取配置信息，把它们存放到各个配置对象中，例如 Action 的映射信息存放在 ActionMapping 对象中。

当 ActionServlet 接收到一个客户请求时，将执行如下流程:

1. 检索和用户请求相匹配的 ActionMapping 实例，如果不存在，就返回用户请求路径无效信息。
2. 如 ActionForm 实例不存在，就创建一个 ActionForm 对象，把客户提交

的表单数据保存到 ActionForm 对象中。

3. 根据配置信息决定是否需要表单验证。如果需要验证，就调用 ActionForm 的 Validate（）方法。

4. 如果 ActionForm 的 Validate()方法返回 null 或返回一个不包含 ActionMessage 的 ActionErrors 对象，就表示表单验证成功。

5. ActionServlet 根据 ActionMapping 实例包含的映射信息决定将请求转发给哪个 Action。如果相应的 Action 实例不存在，就先创建这个实例，然后调用 Action 的 execute（）方法。

6. Action 的 execute（）方法返回一个 ActionForward 对象，ActionServlet 再把客户请求转发给 ActionForward 对象指向的 JSP 组件。

7. ActionForward 对象指向的 JSP 组件生成动态网页，返回给客户。

6、谈一谈你对 Spring 的认识？

Spring 是 Rod 主创的一个应用于 J2EE 领域的轻量应用程序框架，其核心是一个 IOC 容器以及 AOP 实现，在核心上面的一个主要部件是数据访问 DAO 框架，包括一个自己的 JDBC 数据访问封装以及对众多 ORM 系统的集成支持。Spring 还内置一个功能强大、灵活的 Web MVC 框架，以提供快速的 Java Web 应用程序开发，同时 Spring 还提供了以其它各种 MVC 框架或视图技术的集成。通过 Spring 的核心容器及 AOP 的应用，Spring 提供了统一的声明式系统级服务支持。

7、白盒测试一般所以的工具是什么？

Junit

8、什么是 Ioc 和 AOP?

控制反转、面向切面编程

六、XML 方面

1、解析 xml 的四种解析器？区别是什么？

DOM 是基于平台、语言无关的官方 W3C 标准。基于树的层次，其优点是可以移植，编程容易，开发人员只需要调用建树的指令。其缺点是加载大文件不理想。

SAX 是基于事件模型的，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。类似与流媒体的解析方式，所以在加载大文件时效果不错。

JDOM 是想成为 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。使用的是具体类不使用接口，运用了大量的 Collections 类，方便程序员。

DOM4J 是一个独立的开发结果。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。推荐使用。

2、XML 文档定义有几种形式？它们之间有何本质区别？

两种形式 dtd,schema

本质区别:schema 本身是 xml 的,可以被 XML 解析器解析(这也是从 DTD 上发展 schema 的

根本目的)

七、网络基础方面

1、网络的 7 层协议是?

第一层 是物理层
第二层 是数据链路层

第三层 是网络层
第四层 是运输层
第五层 是会话层
第六层 是表示层
第七层 是应用层

第一层，物理层

OSI 模型最低层的“劳苦大众”。它透明地传输比特流，就是传输的信号。该层上的设备包括集线器、发送器、接收器、电缆、连接器和中继器。
第二层，数据链路层
这一层是和包结构和字段打交道的和事佬。一方面接收来自网络层（第三层）的数据帧并为物理层封装这些帧;另一方面数据链路层把来自物理层的原始数据比特封装到网络层的帧中。起着重要的中介作用。

数据链路层由 IEEE802 规划改进为包含两个子层：介质访问控制（MAC）和逻辑链路控制（LLC）。

第三层，网络层
这一层干的事就比较多了。它工作对象，概括的说就是：电路、数据包和信息交换。
网络层确定把数据包传送到其目的地的路径。就是把逻辑网络地址转换为物理地址。如果数据包太大不能通过路径中的一条链路送到目的地，那么网络层的任务就是把这些包分成较小的包。
这些光荣的任务就派给了路由器、网桥路由器和网关。
第四层，传输层。
确保按顺序无错的发送数据包。传输层把来自会话层的大量消息分成易于管理的包以便向网络发送。
第五层，会话层。
在分开的计算机上的两种应用程序之间建立一种虚拟链接，这种虚拟链接称为会话（session）。会话层通过在数据流中设置检查点而保持应用程序之间的同步。允许应用程序进行通信的名称识别和安全性的工作就由会话层完成。
第五层，会话层。
在分开的计算机上的两种应用程序之间建立一种虚拟链接，这种虚拟链接称为会话（session）。会话层通过在数据流中设置检查点而保持应用程序之间的同步。允许应用程序进行通信的名称识别和安全性的工作就由会话层完成。
第六层，表示层。
定义由应用程序用来交换数据的格式。在这种意义上，表示层也称为转换器（translator）。该层负责协议转换、数据编码和数据压缩。转发程序在该层进行服务操作。

第七层，应用层，该层是 OSI 模型的最高层。应用层向应用进程展示所有的网

络服务。当一个应用进程访问网络时，通过该层执行所有的动作。

2、电子邮件传输协议是？

SMTP

3、文件传输是基于哪种协议的？

TCP

4、一个 C 类网络最多能容纳多少台主机？

254

八、算法方面

1、插入排序:

```
package org.rut.util.algorithm.support;
import org.rut.util.algorithm.SortUtil;
public class InsertSort implements SortUtil.Sort{

/* (non-Javadoc)
* @see org.rut.util.algorithm.SortUtil.Sort#sort(int[])
*/ public void sort(int[] data) {
int temp;
for(int i=1;i<data.length;i++){
for(int j=i;(j>0)&&(data[j]<data[j-1]);j--){
SortUtil.swap(data,j,j-1);
}
}
}
}
```

2、冒泡排序:

```
package org.rut.util.algorithm.support;
import org.rut.util.algorithm.SortUtil;
public class BubbleSort implements SortUtil.Sort{

/* (non-Javadoc)
* @see org.rut.util.algorithm.SortUtil.Sort#sort(int[])
*/ public void sort(int[] data) {
```

```
int temp;
for(int i=0;i<data.length;i++){
for(int j=data.length-1;j>i;j--){
if(data[j]<data[j-1]){
SortUtil.swap(data,j,j-1);
}
}
}
}
}
```

3、选择排序:

```
package org.rut.util.algorithm.support;

import org.rut.util.algorithm.SortUtil;

public class SelectionSort implements SortUtil.Sort {

/*
* (non-Javadoc)
*
* @see org.rut.util.algorithm.SortUtil.Sort#sort(int[])
*/ public void sort(int[] data) {
int temp;
for (int i = 0; i < data.length; i++) {
int lowIndex = i;
for (int j = data.length - 1; j > i; j--) {
if (data[j] < data[lowIndex]) {
lowIndex = j;
}
}
SortUtil.swap(data,i,lowIndex);
}
}

}
```

4、Shell 排序:

```
package org.rut.util.algorithm.support;
import org.rut.util.algorithm.SortUtil;
public class ShellSort implements SortUtil.Sort{
```

```
/* (non-Javadoc)
* @see org.rut.util.algorithm.SortUtil.Sort#sort(int[])
*/ public void sort(int[] data) {
for(int i=data.length/2;i>2;i/=2){
for(int j=0;j<i;j++){
insertSort(data,j,i);
}
}
insertSort(data,0,1);
}
/**
* @param data
* @param j
* @param i
*/ private void insertSort(int[] data, int start, int inc)
{
int temp;
for(int i=start+inc;i<data.length;i+=inc){
for(int j=i;(j>=inc)&&(data[j]<data[j-inc]);j-=inc){
SortUtil.swap(data,j,j-inc);
}
}
}
}
```

5、快速排序:

```
package org.rut.util.algorithm.support;
import org.rut.util.algorithm.SortUtil;
public class QuickSort implements SortUtil.Sort{

/* (non-Javadoc)
* @see org.rut.util.algorithm.SortUtil.Sort#sort(int[])
*/ public void sort(int[] data) {
quickSort(data,0,data.length-1);
}
private void quickSort(int[] data,int i,int j){
int pivotIndex=(i+j)/2;
//swap
SortUtil.swap(data,pivotIndex,j);

int k=partition(data,i-1,j,data[j]);
SortUtil.swap(data,k,j);
```

```
if((k-i)>1) quickSort(data,i,k-1);
if((j-k)>1) quickSort(data,k+1,j);

}
/**
* @param data
* @param i
* @param j
* @return
*/ private int partition(int[] data, int l, int r,int pivot)
 {
do{
while(data[++l]<pivot);
while((r!=0)&&data[--r]>pivot);
SortUtil.swap(data,l,r);
}
while(l<r);
SortUtil.swap(data,l,r);
return l;
}
}
```

6、改进后的快速排序:

```
package org.rut.util.algorithm.support;
import org.rut.util.algorithm.SortUtil;
public class ImprovedQuickSort implements SortUtil.Sort {

private static int MAX_STACK_SIZE=4096;
private static int THRESHOLD=10;
/* (non-Javadoc)
* @see org.rut.util.algorithm.SortUtil.Sort#sort(int[])
*/ public void sort(int[] data) {
int[] stack=new int[MAX_STACK_SIZE];

int top=-1;
int pivot;
int pivotIndex,l,r;

stack[++top]=0;
stack[++top]=data.length-1;

while(top>0){
int j=stack[top--];
```

```
int i=stack[top--];

pivotIndex=(i+j)/2;
pivot=data[pivotIndex];

SortUtil.swap(data,pivotIndex,j);

//partition
l=i-1;
r=j;
do{
while(data[++l]<pivot);
while((r!=0)&&(data[--r]>pivot));
SortUtil.swap(data,l,r);
}
while(l<r);
SortUtil.swap(data,l,r);
SortUtil.swap(data,l,j);

if((l-i)>THRESHOLD){
stack[++top]=i;
stack[++top]=l-1;
}
if((j-l)>THRESHOLD){
stack[++top]=l+1;
stack[++top]=j;
}

}
//new InsertSort().sort(data);
insertSort(data);
}
/**
* @param data
*/ private void insertSort(int[] data) {
int temp;
for(int i=1;i<data.length;i++){
for(int j=i;(j>0)&&(data[j]<data[j-1]);j--){
SortUtil.swap(data,j,j-1);
}
}
}

}
```

7、归并排序:

```
package org.rut.util.algorithm.support;

import org.rut.util.algorithm.SortUtil;

public class MergeSort implements SortUtil.Sort{

/* (non-Javadoc)
* @see org.rut.util.algorithm.SortUtil.Sort#sort(int[])
*/ public void sort(int[] data) {
int[] temp=new int[data.length];
mergeSort(data,temp,0,data.length-1);
}

private void mergeSort(int[] data,int[] temp,int l,int r){
int mid=(l+r)/2;
if(l==r) return ;
mergeSort(data,temp,l,mid);
mergeSort(data,temp,mid+1,r);
for(int i=l;i<=r;i++){
temp=data;
}
int i1=l;
int i2=mid+1;
for(int cur=l;cur<=r;cur++){
if(i1==mid+1)
data[cur]=temp[i2++];
else if(i2>r)
data[cur]=temp[i1++];
else if(temp[i1]<temp[i2])
data[cur]=temp[i1++];
else
data[cur]=temp[i2++];
}
}
}
```

8、改进后的归并排序:

```
package org.rut.util.algorithm.support;
import org.rut.util.algorithm.SortUtil;
```

```
public class ImprovedMergeSort implements SortUtil.Sort {

private static final int THRESHOLD = 10;

/*
* (non-Javadoc)
*
* @see org.rut.util.algorithm.SortUtil.Sort#sort(int[])
*/ public void sort(int[] data) {
int[] temp=new int[data.length];
mergeSort(data,temp,0,data.length-1);
}

private void mergeSort(int[] data, int[] temp, int l, int r)
 {
int i, j, k;
int mid = (l + r) / 2;
if (l == r)
return;
if ((mid - l) >= THRESHOLD)
mergeSort(data, temp, l, mid);
else
insertSort(data, l, mid - l + 1);
if ((r - mid) > THRESHOLD)
mergeSort(data, temp, mid + 1, r);
else
insertSort(data, mid + 1, r - mid);

for (i = l; i <= mid; i++) {
temp = data;
}
for (j = 1; j <= r - mid; j++) {
temp[r - j + 1] = data[j + mid];
}
int a = temp[l];
int b = temp[r];
for (i = l, j = r, k = l; k <= r; k++) {
if (a < b) {
data[k] = temp[i++];
a = temp;
} else {
data[k] = temp[j--];
b = temp[j];
}
```

```
}
}

/**
* @param data
* @param l
* @param i
*/ private void insertSort(int[] data, int start, int len)
{
for(int i=start+1;i<start+len;i++){
for(int j=i;(j>start) && data[j]<data[j-1];j--){
SortUtil.swap(data,j,j-1);
}
}
}
```