# 企业面试题总汇(13-51)

## Java基础部分

### 请列举至少3个JDK安装目录下的可执行程序(javac)，并列举几个常用的命令行参数

|  |  |
| --- | --- |
| 1. javac 将Java源代码换成字节代 | javac <选项><源文件> 可能的选项包括：  -g 生成所有调试信息  -g:none 生成无调试信息  -g:{lines,vars,source} 生成只有部分调试信息  -O 优化；可能妨碍调试或者增大类文件  -nowarn 生成无警告  -verbose 输出关于编译器正在做的信息  -deprecation 输出使用了不鼓励使用的API的源程序位置  -classpath <路径> 指定用户类文件的位置  -sourcepath <路径> 指定输入源文件的位置  -bootclasspath <路径> 覆盖自举类文件的位置  -extdirs <目录(多个)> 覆盖安装的扩展类的位置  -d <目录> 指定输出类文件的位置 |
| **用法：** |
| 2.jar 打架包，查看jar包 | jar {ctxu}[vfm0M] [jar-文件] [manifest-文件] [-C 目录] 文件名 ...选项：  -c 创建新的存档  -t 列出存档内容的列表  -x 展开存档中的命名的（或所有的）文件  -u 更新已存在的存档  -v 生成详细输出到标准输出上  -f 指定存档文件名  -m 包含来自标明文件的标明信息  -0 只存储方式；未用ZIP压缩格式  -M 不产生所有项的清单（manifest）文件  -i 为指定的jar文件产生索引信息 |
| **用法：** |
| 3.javadoc：根据Java源代码及其说明语句生成的HTML文档 | javadoc [options] [packagenames] [sourcefiles] [classnames] [@files] -overview<file>读取 HTML 格式的概述文档  -public 仅显示 public 类和成员  -protected 显示 protected/public 类和成员（缺省）  -package 显示 package/protected/public 类和成员  -private 显示所有类和成员  -help 显示命令行选项  -doclet <class> 通过候选 doclet 生成输出  -docletpath <path> 指定 doclet 类文件的查找位置  -sourcepath <pathlist> 指定源文件的查找位置  -classpath <pathlist> 指定用户类文件的查找位置 |
| **用法：** |
| 4.javah：产生可以调用Java过程的C过程，或建立能被Java程序调用的C过程的头文件 | javah [options] <classes> 其中 [options] 包括：  -help 打印该帮助信息 -classpath <path> 类的加载路径  -bootclasspath <path> 自举类的加载路径  -d <dir> 输出目录  -o <file> 输出文件（仅能使用 -d 或 -o 之一）  -jni 生成 JNI 风格的头文件（缺省）  -old 生成 JDK1.0 风格的头文件  -stubs 生成 stubs 文件  -version 打印版本信息  -verbose 输出有关本命令所做工作的信息  -force 始终写输出文件 |
| **用法：** |

### 请分析命题：“Java采用自动垃圾回收技术（GC），因此不会出现内存泄露”

对象的建立和放置都是在内存堆栈上面进行的。程序或者其他的对象可以锁定一

块堆栈地址来进行其他对象的引用。

当一个对象没有任何引用的时候，Java的自动垃圾回收机制就发挥作用，自动

删除这个对象所占用的空间，释放内存以避免内存泄漏。但是内存泄漏并不是就此而完全避免了，当程序员疏忽大意地忘记解除一个对象不应该有的引用的时候，内存泄漏仍然不可避免，不过发生的几率要比不启用垃圾回收机制的C++程序少很多。但总体来讲，自动垃圾回收机制要安全和简单许多。

### 简单描述单子模式(单例模式)的各种不同实现方式，极其各自优缺点。请列举至少2种其他的设计模式及应用场景。

1. /\*单例设计模式主要有2种实现方式：懒汉式、饿汉式。
2. 他们分别有如下实现：
3. 饿汉式：
4. //饿汉式单例类.在类初始化时，已经自行实例化   \*/
5. **public** **class** Singleton1 {
6. //私有的默认构造子
7. **private** Singleton1() {}
8. //已经自行实例化
9. **private** **static** **final** Singleton1 single = **new** Singleton1();
10. //静态工厂方法
11. **public** **static** Singleton1 getInstance() {
12. **return** single;
13. }
14. }
15. 懒汉式：
16. //懒汉式单例类.在第一次调用的时候实例化
17. **public** **class** Singleton2 {
18. //私有的默认构造子
19. **private** Singleton2() {}
20. //注意，这里没有final
21. **private** **static** Singleton2 single=**null**;
22. //静态工厂方法
23. **public** **synchronized**  **static** Singleton2 getInstance() {
24. **if** (single == **null**) {
25. single = **new** Singleton2();
26. }
27. **return** single;
28. }
29. }
30. /\*其他设计模式：
31. 1.适配器模式
32. 应用场景：比如说在朋友聚会上遇到一个德国人sarsh，可以我不会德语，她不会中文，我只好求助于我的朋友mary了，他作为我和sarsh之间的adapt，让我和sarsh可以交流了。
33. 2.装饰模式
34. 应用场景：想要自己做礼物送给朋友，拿出去年在华山顶上的照片，在背面写上“最好的礼物”再到街上礼品店买个相框，在找隔壁学美术的朋友设计一个漂亮的盒子装起来，我们都是Decorator。\*/

### 输出结果

1. **public** **static** **void** main(String[] args)**throws** Exception{
2. String str = "中国";
3. System.out.println(str.getBytes("UTF-8").length);
4. System.out.println(str.getBytes("GBK").length);
5. System.out.println(str.getBytes("ISO-8859-1").length);
6. System.out.println(**new** String(str.getBytes("ISO-8859-1"),("ISO-8859-1"));
7. System.out.println(**new** String(str.getBytes("UTF-8"),("UTF-8"));
8. System.out.println(**new** String(str.getBytes("GBK"),("GBK"));
9. }
10. 输出结果：
11. 6
12. 4
13. 2
14. ??
15. 中国
16. 中国

### 使用二分查找法找字符串数组{"a","b","c","d","e","f","g","h"}中"g"元素

1. **public** **class** Query4Half {
2. **static** **int** bsearch( String[] a, String v ) {
3. **int** l, r;
4. l = 0; r = a.length-1;
5. **while** ( l <= r ) {
6. **int** m = (l+r)/2;
7. **if** ( a[m].compareTo(v)==0 ) **return** m; **else**
8. **if** ( a[m].compareTo(v)>0 ) r = m-1; **else**
9. **if** ( a[m].compareTo(v)<0 ) l = m+1;
10. }
11. **return** -1;
12. }
13. **public** **static** **void** main(String[] args) {
14. String str[] = {"a","b","c","d","e","f","g","h"};
15. **int** bsearch = bsearch(str, "g");
16. System.out.println(bsearch);
17. }

### Java代码实现一个链表结构

1. **public** **class** MyList<AnyType>  {
2. **private** **class** Node<AnyType>{
3. **public**  Node<AnyType> pre;
4. **public**  Node<AnyType> next;
5. **public**  AnyType      data;
6. **public** Node(AnyType d, Node<AnyType>p, Node<AnyType> n){}
7. **public** Node(){}
8. }
9. **private** **int** theSize;
10. **private** Node<AnyType> Header;
11. **private** Node<AnyType> Tail;
12. **public** MyList(){}
13. **public** **void** add(AnyType item){}
14. **public** **boolean** isEmpty(){}
15. **public** **int** size(){}
16. **public** AnyType get( **int** idx){}
17. **public** **void** print(){}
18. }
19. /\*Node<AnyType>类定义了双向链表中节点的结构，它是一个私有类，而其属性和构造函数都是公有的，这样，其父类可以直接访问其属性，而外部类根本不知道Node类的存在。Data是节点中的数据与，pre指向前一个Node节点，next指向后一个Node节点。其构造函数的实现如下，不解释：
20. View Code\*/
21. **public** Node(AnyType d, Node<AnyType>p, Node<AnyType> n){
22. **this**.data = d;
23. **this**.pre = p;
24. **this**.next = n;
25. }
26. **public** Node(){
27. **this**.data = **null**;
28. **this**.pre = **null**;
29. **this**.next = **null**;
30. }
31. /\*下面我们看一下链表的构造函数实现：
32. View Code\*/
33. **public** MyList(){
34. theSize = 0;
35. Header = **new** Node<AnyType>(**null**,**null**,**null**);
36. Tail   =  **new** Node<AnyType>(**null**,Header,**null**);
37. Header.next = Tail;
38. }
39. /\*我们构造了一个带有头、尾节点的双向链表，头节点的Next指向尾节点，为节点的pre指向头节点。链表长度起始为0。
40. 继续贴上链表类其它方法的实现，不解释了，应该比较清楚：\*/
41. //View Code
42. **public** **void** add(AnyType item){
43. Node<AnyType> aNode = **new** Node<AnyType>(item,**null**,**null**);
44. Tail.pre.next = aNode;
45. aNode.pre = Tail.pre;
46. aNode.next = Tail;
47. Tail.pre = aNode;
48. theSize++;
49. }
50. **public** **boolean** isEmpty(){
51. **return** ( theSize == 0);
52. }
53. **public** **int** size(){
54. **return** theSize;
55. }
56. **public** AnyType get( **int** idx){
57. **if**(idx > theSize-1 || idx < 0)
58. **throw** **new** IndexOutOfBoundsException();
59. Node<AnyType> current = **new** Node<AnyType>(**null**,Header,**null**);
60. **for**(**int** i = 0; i<idx; i++)
61. current = current.next;
62. **return** current.data;
63. }
64. **public** **void** print(){
65. Node<AnyType> current = Header.next;
66. **while**(current.next != **null**){
67. //如果AnyType是你自己定义 //的数据类型，那么请务必提供
68. //一个toString方法，要么就不
69. //要在链表里实现print方法。
70. System.out.println(current.data.toString());
71. current = current.next;
72. }
73. }

### Object类（或子类）的finalize()方法在什么情况下被调用？

当某个对象被系统收集为无用信息的时候,finalize()将被自动调用,但是jvm不保证finalize()一定被调用,也就是说,finalize()的调用是不确定的...

### 请列出你所了解的若干个不同类型的runtime exception

NullPointerException - 空指针引用异常

ClassCastException - 类型强制转换异常。

IllegalArgumentException - 传递非法参数异常。

ArithmeticException - 算术运算异常

ArrayStoreException - 向数组中存放与声明类型不兼容对象异常

IndexOutOfBoundsException - 下标越界异常

NegativeArraySizeException - 创建一个大小为负数的数组错误异常

NumberFormatException - 数字格式异常

SecurityException - 安全异常

UnsupportedOperationException - 不支持的操作异常

### 数组String[] str和String str分别怎么获取长度？

size 和 length()

### JAVA实现快速排序算法

1. **public** **class** QuickSort {
2. **public** **static** **int** Partition(**int** a[],**int** p,**int** r){
3. **int** x=a[r-1];
4. **int** i=p-1;
5. **int** temp;
6. **for**(**int** j=p;j<=r-1;j++){
7. **if**(a[j-1]<=x){
8. // swap(a[j-1],a[i-1]);
9. i++;
10. temp=a[j-1];
11. a[j-1]=a[i-1];
12. a[i-1]=temp;
13. }
14. }
15. //swap(a[r-1,a[i+1-1]);
16. temp=a[r-1];
17. a[r-1]=a[i+1-1];
18. a[i+1-1]=temp;
19. **return** i+1;
20. }
21. **public** **static** **void** QuickSort(**int** a[],**int** p,**int** r){
22. **if**(p<r){
23. **int** q=Partition(a,p,r);
24. QuickSort(a,p,q-1);
25. QuickSort(a,q+1,r);
26. }
27. }
28. **public** **static** **void** main(String[] args){
29. **int** a[]={23,53,77,36,84,76,93,13,45,23};
30. QuickSort(a,1,10);
31. **for** (**int** i=1;i<=10;i++)
32. System.out.println(a[i-1]);
33. }
34. }

### 请简单阐述一下Socket通讯原理，以及Socket基于TCP和UDP协议实现的优缺点

Socket通信的原理大致分为以下几个步骤。

服务器端的步骤如下。

（1）建立服务器端的Socket，开始侦听整个网络中的连接请求。

（2）当检测到来自客户端的连接请求时，向客户端发送收到连接请求的信息，建立与客户端之间的连接。

（3）当完成通信后，服务器关闭与客户端的Socket连接。

客户端的步骤如下。

（1）建立客户端的Socket，确定要连接的服务器的主机名和端口。

（2）发送连接请求到服务器，并等待服务器的回馈信息。

（3）连接成功后，与服务器进行数据的交互。

（4）数据处理完毕后，关闭自身的Socket连接。

TCP：面向连接、传输可靠(保证数据正确性,保证数据顺序)、用于传输大量数据(流模式)、速度慢，建立连接需要开销较多(时间，系统资源)。

UDP：面向非连接、传输不可靠、用于传输少量数据(数据包模式)、速度快。

### 对象封装的原则是什么？

封装原则：隐藏对象的属性和实现细节，仅对外公开接口，并且控制访问级别

在OOP（面向对象）编程中，用类来实现上面的要求。用类实现封装，用封装来实现高内聚，低耦合

高聚合是说一个对象的功能都在内部搞定，紧紧地结合在一起

低耦合是说对象之间的依赖要松散，不要牵一发动全身

只有这样，当系统规模扩大时，才能比较方便

### 什么是java序列化，如何实现java序列化？

我们有时候将一个java对象变成字节流的形式传出去或者从一个字节流中恢复成一个java对象，例

如，要将java对象存储到硬盘或者传送给网络上的其他计算机，这个过程我们可以自己写代码去把一个java对象变成某个格式的字节流再传输，但是，jre本身就提供了这种支持，我们可以调用OutputStream的writeObject方法来做，如果要让java 帮我们做，要被传输的对象必须实现serializable接口，这样，javac编译时就会进行特殊处理，编译的类才可以被writeObject方法操作，这就是所谓的序列化。需要被序列化的类必须实现Serializable接口，该接口是一个mini接口，其中没有需要实现的方法，implements Serializable只是为了标注该对象是可被序列化的。

例如，在web开发中，如果对象被保存在了Session中，tomcat在重启时要把Session对象序列化

到硬盘，这个对象就必须实现Serializable接口。如果对象要经过分布式系统进行网络传输或通过rmi等远程调用，这就需要在网络上传输对象，被传输的对象就必须实现Serializable接口。

### 序列化的作用，应用场景

所谓java对象序列化就是将对象的状态转换成字节流，以后可以通过这些值再生成相同状态的对象！对象序列化是对象持久化的一种实现方法,它是将一个对象的属性和方法转化为一种序列化的格式以用于存储和传输,反序列化就是根据这些保存的信息重建对象的过程。

运用场景1：

Java的JavaBeans：Bean的状态信息通常是在设计时配置的，Bean的状态信息必须被存起来，以便当程序运行时能恢复这些状态信息，这需要将对象的状态保存到文件中，而后能够通过读入对象状态来重新构造对象，恢复程序状态。例如Java.io包有两个序列化对象的类。ObjectOutputStream负责将对象写入字节流，ObjectInputStream从字节流重构对象。

运用场景2：

activiti工作流设置流程变量

### Collection和Collections有什么区别？

Collection是集合类的上级接口，继承与他的接口主要有Set 和List.

Collections是针对集合类的一个帮助类，他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

### String s = new String("abc")；请问有几个String对象？分别是

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。如果以前就用过‘xyz’直接从缓冲区拿。如果没有就在字符串缓冲区创建这个“xyz”字符串对象，New String每写一遍，就创建一个新的对象。

### try{}里有一个return语句，那么紧跟在这个try后的finally{}里德code会不会被执行？什么时候被执行，在return之前还是后？

1. **public**  **class** Test {
2. **public** **static** **void** main(String[] args) {
3. System.out.println(**new** Test().test());;
4. }
5. **static** **int** test(){
6. **int** x = 1;
7. **try**{
8. **return** x;
9. }**finally**{
10. ++x;
11. }
12. }
13. }
14. /\*---------执行结果 ---------\*/
15. 1
16. /\*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。 \*/

### java中可见的访问修饰符的含义

public：访问不受限制。

protected：访问仅限于包含类或从包含类派生的类型。只有包含该成员的类以及继承的类可以存取.

Internal：访问仅限于当前程序集。只有当前工程可以存取.

protected internal：访问仅限于当前程序集或从包含类派生的类型。

private：访问仅限于包含类型。只有包含该成员的类可以存取.

### java用什么来判断对象是否是属于某个类或借口的事例？

Instanceof

### Java中，int和Integer有什么区别

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

### Java中，HashMap和Hashtable的区别？

HashMap把Hashtable的contains方法去掉了，改成containsvalue和containsKey。因为contains方法容易让人引起误解。

Hashtable继承自Dictionary类，而HashMap是Java1.2引进的Map interface的一个实现。

最大的不同是，Hashtable的方法是Synchronize的，而HashMap不是，在多个线程访问Hashtable时，不需要自己为它的方法实现同步，而HashMap 就必须为之提供外同步。

Hashtable和HashMap采用的hash/rehash算法都大概一样，所以性能不会有很大的差异。

就HashMap与HashTable主要从三方面来说。

1.历史原因:Hashtable是基于陈旧的Dictionary类的，HashMap是Java 1.2引进的Map接口的一个实现

2.同步性:Hashtable是线程安全的，也就是说是同步的，而HashMap是线程序不安全的，不是同步

3.值：只有HashMap可以让你将空值作为一个表的条目的key或value

### Java中，String和StringBuffer的区别？

1, String : 字符串常量,字符串长度不可变!!!

2, StringBuffer : 字符串变量(Synchronized，即线程安全),如果要频繁对字符串内容进行修改，出于效率考虑最好使用StringBuffer，如果想转成String类型，可以调用StringBuffer的toString()；StringBuffer 上的主要操作是 append 和 insert 方法，可重载这些方法，以接受任意类型的数据。append方法始终将这些字符添加到缓冲区的末端；而insert方法则在指定的点添加字符。

3, (\*延伸) StringBuilder 字符串变量（非线程安全）,这是在JDK5.0新增加的!!!,该对象是一个线程不安全的字符串容器,在单线程操作的情况下此对象可以用于取代StringBuffer!!!

4，(String 和 StringBuffer )两者者之间的区别 :

这两者之间的区别主要区别在性能上,String,每次对存放进String当中的字符串进行更改的时候,都会生成一个新的String对象,然后指针指向该对象(虽然Java不承认自己有指针这回事),而一旦当产生的没有引用指向对象过多的时候 ,这个时候GC就会出动,对其进行清理,这个时候,性能就会有所降低!!!而使用 StringBuffer的时候,每次都会对该对象本身进行操作,而不是去创建一个新的对象,所以在多数情况下还是推荐使用该对象!!!

但是某些特殊的情况下,使用String 并不会比StringBuffer慢!!!

例如 : String str = "String1" + "String2"；&& StringBuffer sb =

new StringBuffer(); sb.append("String1").append("String2");

在这里的String str = "String1" + "String2"实际上被翻译成了 str = "String1String2";

### Thread和Runnable有什么不同？

首先,Java 中Thread 和 Runnable是实现并发线程的两种方式!!!,

通过两者中的任意一个都可以达到线程并发的目的!!!

区别 :

0,Thread 是一個 Class,Runnable 是一個 interface!!!

1,通过extends Thread，可获得start(), sleep(), 并覆盖run(), 使用时直接用start()，但有时不太方便通过继承(extends)而进行多线程，例如已继承了其它Class(Java 不允许多重继承)时，就应该实现(implements) Runnable写多线程程序。

2,注意，implements Runnable创建的只有run()方法而没有start()，因此要利用其它机制来启动。这要创建两个对象，

第一个是Runnable的，

第二个是Thread的，然后把第一个当成参数传给第二个!!!

3,由下面的程序可以看出,实现Runnable的多线程程序要比,继承Thread的多线程程序灵活的多!!!

4,这里还有一点,就是说使用Runnable实现多线程可以达到资源共享目的,让多个Thread对象接收同一个Runnable实现类的参数,这样就是多个线程操作同一块资源!!!

例如 :

1. //Runnable 接口的多线程程序举例!!!
2. **private** **class** RunnableThread **implements** Runnable{
3. /\*这个方法中写,多线程执行的程序!!!,因为这里是实现了Runnable所以就算再去继承其他的class也不会违反java成不能多继承的规定!!!\*/
4. **public** **void** run(){
5. }
6. }
7. /\*通过将Runnable的实现类作为参数传递给一个Thread对象,然后通过开启Thread对象,就能让其执行自己复写的方法,并且是并发操作!!!\*/
8. Thread runnThread = **new** Thread(**new** RunnableThread());
9. RunnThread.start();
10. /\*1，继承自Thread类的多线程程序
11. **private** **class** GenThread **extends** Thread  {
12. //这个方法中写,多线程需要执行的方法!!!
13. **public** **void** run() {
14. }
15. }
16. Thread theGenThread = **new** GenThread();
17. theGenThread.start();

### GC是什么，为什么要有GC?

1,GC : 垃圾回收 (GarbageCollection),Java的内存管理,其内置的垃圾回收,是Java语言的优秀成果,它允许开发者创建新的对象不需要明确内存分配和释放,因为垃圾回收器自动回收内存当中的对象。

2,垃圾收集释放从手工处理内存释放的程序员。其结果是，某些类别的缺陷被消除或显着减少：这也正是Java的特点(相对于C++来说,程序员不再需要通过痛苦的记忆哪里的内存是释放掉的,而哪一些是没有释放掉的)

内存不安全释放的问题 :

1,当一块内存被释放，同时还存在着指向它，而那些指针中的一个被废弃发生。届时内存可能已被重新分配到其他使用，具有不可预知的结果。

2,当程序试图发生释放的内存已被释放，并可能已经被再次分配的区域。

3,某些种类的内存泄漏，其中一个程序无法释放由已成为对象占用的内存可达，从而导致内存消耗殆尽。（垃圾收集通常不处理的数据是可到达的无界积累，但实际上将不被使用的程序）……

### 是否可以从一个static方法内部发出对非static方法的调用？

不可以。因为非static方法是要与对象关联在一起的，必须创建一个对象后，才可以在该对象上进行方法调用，而static方法调用时不需要创建对象，可以直接调用。也就是说，当一个static方法被调用时，可能还没有创建任何实例对象，如果从一个static方法中发出对非static方法的调用，那个非static方法是关联到哪个对象上的呢？这个逻辑无法成立，所以，一个static方法内部发出对非static方法的调用。

### 如何把一段逗号分隔的字符串转换成一个数组？

1. //方式一：使用String的split 方法
2. **public** **static** String[] convertStrToArray(String str){
3. String[] strArray = **null**;
4. strArray = str.split(",");
5. **return** strArray;
6. }
7. //方式二：使用StringTokenizer实现
8. //api说明：StringTokenizer 是出于兼容性的原因而被保留的遗留类
9. //（虽然在新代码中并不鼓励使用它）。建议所有寻求此功能的人使用
10. //String 的 split 方法或 java.util.regex 包
11. **public** **static** String[] convertStrToArray2(String str){
12. StringTokenizer st = **new** StringTokenizer(str,",");
13. String[] strArray = **new** String[st.countTokens()];
14. **int** i=0;
15. **while**(st.hasMoreTokens()){
16. strArray[i++] = st.nextToken();
17. }
18. **return** strArray;
19. }

### HashMap和Hashtable的区别

1、同步或线程安全：这是两者最重要的区别。HashMap的是不同步的，不是线程安全的，另一方面，Hashtable是线程安全的，同步的。

1-1、何时使用HashMap：答案是，如果您的应用程序不需要任何多线程任务，换句话说HashMap

的是对于非线程安全应用程序的更好的选择。哈希表应当在多线程应用中使用。

2、null键和空值：HashMap是允许空键空值的,而Hashtable是不允许空键空值的。

3、迭代器类型：HashTable使用Enumeration，HashMap使用Iterator。

4、性能：HashMap因为是线程不安全的,效率要比Hashtable高。

5、Hashtable是Dictionary类现在是在JDK 1.7过时的子类，因此，它已不再使用。

（

hashmap与hashtable  
Hashtable继承自Dictionary类，而HashMap是Java1.2引进的Map interface的一个实现   
HashMap允许将null作为一个entry的key或者value，而Hashtable不允许   
还有就是，HashMap把Hashtable的contains方法去掉了，改成containsvalue和containsKey。因为contains方法容易让人引起误解。   
最大的不同是，Hashtable的方法是Synchronize的，而HashMap不是，在   
多个线程访问Hashtable时，不需要自己为它的方法实现同步，而HashMap   
就必须为之提供外同步）

### java堆和栈的区别

Java栈是与每一个线程关联的，JVM在创建每一个线程的时候，会分配一定的栈空间给线程，主要存放线程执行过程中的局部变量，方法的返回值，基本类型的变量（,int, short, long, byte, float, double, boolean, char）以及方法调用的上下文。栈空间随着线程的终止而释放，栈的优势是，存取速度比堆要快，栈数据可以共享。但缺点是，存在栈中的数据大小与生存期必须是确定的，缺乏灵活性。栈有一个很重要的特殊性，就是存在栈中的数据可以共享。

Java中堆是由所有的线程共享的一块内存区域，堆用来保存各种JAVA对象，比如数组，线程对象等,

java的堆是一个运行时数据区,类的(对象从中分配空间。这些对象通过new、newarray、anewarray和multianewarray等 指令建立，它们不需要程序代码来显式的释放。堆是由垃圾回收来负责的，堆的优势是可以动态地分配内存大小，生存期也不必事先告诉编译器，因为它是在运行时 动态分配内存的，Java的垃圾收集器会自动收走这些不再使用的数据。但缺点是，由于要在运行时动态分配内存，存取速度较慢。

### 当一个线程进入一个对象的synchronized方法后，其他线程是否可进入此对象的其他方法？为什么？

分情况而定：

1.其他方法前是否加了synchronized关键字，如果没加，则能。

2.如果这个方法内部调用了wait，则可以进入其他synchronized方法。

3.如果其他个方法都加了synchronized关键字，并且内部没有调用wait，则能。

4.如果其他方法是static，它用的同步锁是当前类的字节码，与非静态的方法不能同步，因为非静态的方法用的是this

### 99乘法表

1. /\*主要是利用双层for循环来实现，外层循环9次，内层每次循环的次数与外层是第几次循环有关，还要考虑字符串的拼接及换行和制表符的问题，具体代码如下：\*/
2. **public** **class** s99{
3. **public** **static** **void** main(String[] args){
4. **for** (**int** i=1;i<=9;i++) {
5. **for** (**int** j=1;j<=i;j++){
6. System.out.print(i+"\*"+j+"="+(i\*j)+"\t");
7. }
8. System.out.print("\n");
9. }
10. }
11. }

### 常用的字符转码方式什么？举例说明

1.escape方法对String对象编码,escape方法返回一个包含了"转义序列"的字符串值。

除了ASCII字母和数字，以及这几个符号 \*+-/.@\_外(共有10+52+7=69个字符不会被编码)，

其他所有的字符都会被转义序列替换，标准ASCII码内的字符(0-127)和扩展的ASCII码内

的字符(128-255)都用%xx编码形式代替。字符值大于255的以unicode%uxxxx格式存储。

注意 :escape方法不能够用来对统一资源标示符URI进行编码,因为=&?#等会被编码.

对URI编码应使用encodeURI,对URI参数用encodeURIComponent方法。

2.encodeURI除了ASCII字母和数字，以及这几个符号 \*+-/.@\_!#$&\'(),:=?~外(共有10+52+20=82个字符不会被编码)，

其他所有的字符都会被转义序列替换，标准ASCII码内的字符(0-127)和扩展的ASCII码内

的字符(128-255)都用%xx编码形式代替。字符值大于255的以utf-8多字节%xx格式存储。

在URI中有特殊意义的字符如?=&等不会被编码。

3.encodeURIComponent除了ASCII字母和数字，以及这几个符号 \*-\_.~!'()外(共有10+52+9=71个字符不会被编码)，其他所有的字符都会被转义序列替换，标准ASCII码内的字符(0-127)和扩展的ASCII码内的字符(128-255)都用%xx编码形式代替。字符值大于255的以utf-8多字节%xx格式存储。可以看到?=&等在URI中有特殊意义的字符也会被编码。

encodeURIComponent方法在编码单个URIComponent（指请求参数）应当是最常用的，

它可以将参数中的中文、特殊字符进行转义，而不会影响整个URL。

### 如何给tomcat指定大小的内存

Tomcat默认可以使用的内存为128MB，在较大型的应用项目中，这点内存是不够的，需要调大。有以下几种方法可以选用：

第一种方法：

Windows下，在文件/bin/catalina.bat，Unix下，在文件/bin/catalina.sh的前面，增加如下设置：

JAVA\_OPTS='-Xms【初始化内存大小】-Xmx【可以使用的最大内存】'

需要把这个两个参数值调大。例如：

JAVA\_OPTS='-Xms256m-Xmx512m'

表示初始化内存为256MB，可以使用的最大内存为512MB。

第二种方法：环境变量中设

变量名：JAVA\_OPTS

变量值：-Xms512m-Xmx512m

第三种方法：前两种方法针对的是bin目录下有catalina.bat的情况（比如直接解压的Tomcat等），但是有些安装版的Tomcat下没有catalina.bat，这个时候可以采用如下方法，当然这个方法也是最通用的方法：

打开tomcatHome/\bin/\tomcat5w.exe，点击Java选项卡，然后将会发现其中有这么两项：Initialmemorypool和Maximummemorypool.Initialmemorypool这个就是初始化设置的内存的大小。Maximummemorypool这个是最大JVM内存的大小设置完了就按确定然后再重启TOMCAT你就会发现tomcat中jvm可用的内存改变了。

### final，finally，finalize的区别

1.final为关键字，final定义基本类型变量时，要求变量初始化必须在声明时或者构造函数中，不能用于其它地方。该关键字定义的常量，除了初始化阶段，不能更改常量的值； final定义对象的引用，该引用的初始化与定义常量时的要求一致；该关键字定义的对象内容可以改变，但是引用指向的地址不能改变；final修饰类时无法被其他任何类继承。

2.finalize()方法在Object中进行了定义，用于在对象“消失”时，由JVM进行调用用于对对象进行垃圾回收。

3.finally为区块标志，用于try语句中，表示必须运行的区块部分。

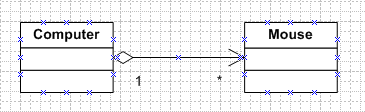
### Collection和Collections的区别

Collection是集合类的上级接口，继承与他的接口主要有Set 和List.

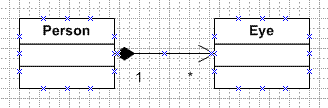
Collections是针对集合类的一个帮助类，他提供一系列静态方法实现对各种集合的搜索、排序、线程安全化等操作

### 组合和聚合的区别

聚合：表示两个对象之间是整体和部分的**弱**关系，部分的生命周期可以超越整体。电脑和鼠标的关系



组合：表示两个对象之间是整体和部分的强关系，部分的生命周期不能超越整体，或者说不能脱离整体而存在。组合的关系“部分”是不能再整体之间进行共享的。如人跟眼睛



### 关联和依赖的区别

关联：简单而言，如果一个类作为了另外一个类的属性，那一定是关联关系。但是聚合是一种特殊的关联，而组合

又是一种特殊的聚合。一般的关联我们认为是平级、无整体部分关系的，如朋友关系。

依赖：对象之间的临时关系，临时性体现在不超越方法的周期。简单地说，在方法使用到的，如参数类，就一定是

依赖关系

### JDK中用到的设计模式

1、创建模式

Singleton（单例）：只允许创建一个实例

1. Runtime.getRuntime();
2. java.awt.Toolkit.getDefaultToolkit();
3. java.awt.GraphicsEnvironment.getLocalGraphicsEnvironment();
4. java.awt.Desktop.getDesktop();

Abstract factory(抽象工厂):创建一组有关联的对象实例

1. java.util.Calendar.getInstance();
2. java.util.Arrays.asList();
3. java.util.ResourceBundle.getBundle(String baseName);
4. java.sql.DriverManager.getConnection(String url);
5. java.sql.Connection.createStatement();
6. java.sql.Statement.executeQuery(String sql);
7. java.text.NumberFormat.getInstance();
8. javax.xml.transform.TransformerFactory.newInstance();

Factory(工厂方法):按照需求返回一个类型的实例

1. java.lang.Proxy.newProxyInstance();
2. java.lang.Object.toString();
3. java.lang.Class.newInstance();
4. java.lang.reflect.Array.newInstance();
5. java.lang.reflect.Constructor.newInstance();
6. java.lang.Boolean.valueOf(String s);
7. java.lang.Class.forName(String className);

Builder(建造者):主要用来简化一个复杂的对象的创建

1. java.lang.StringBuilder.append(Object obj);
2. java.lang.StringBuffer.append(Object obj);
3. java.sql.PreparedStatement
4. javax.swing.GroupLayout.Group.addComponent();

Prototype(原型):使用自己的实例创建另一个实例

1. java.lang.Object.clone();
2. java.lang.Cloneable;
3. **结构模式**

Adapter(适配器):个接口或是类变成另外一种

1. java.util.Arrays.asList();
2. javax.swing.JTable(TableModel)
3. java.io.InputStreamReader(InputStream)
4. java.io.OutputStreamWriter(OutputStream)
5. javax.xml.bind.annotation.adapters.XmlAdapter.marshal();
6. javax.xml.bind.annotation.adapters.XmlAdapter.unmarshal();

Composite(组合):让使用者把单独的对象和组合对象混用

1. javax.swing.JComponent.add(Component comp);
2. java.awt.Container.add(Component comp);
3. java.util.Map.putAll(Map m);
4. java.util.List.addAll(Collection c);
5. java.util.Set.addAll(Collection c);

Decorator(装饰器):为一个对象动态的加上一系列的动作，而不需要因为这些动作的不同而产生大量的继承类

1. java.io.BufferedInputStream(InputStream);
2. java.io.DataInputStream(InputStream);
3. java.io.BufferedOutputStream(OutputStream);
4. java.util.zip.ZipOutputStream(OutputStream);
5. java.util.Collections.checkedList(List list, Class type) ;

Facade(门面):一个简单的接口包状一组组件，接口，抽象或是子系统。

1. java.lang.Class
2. javax.faces.webapp.FacesServlet

Flyweight(共享元):有效率地存储大量的小的对象

1. java.lang.Integer.valueOf(**int**)
2. java.lang.Boolean.valueOf(**boolean**)
3. java.lang.Byte.valueOf(**byte**)
4. java.lang.Character.valueOf(**char**)

Proxy(代理):用一个简单的对象来代替一个复杂的对象

1. java.lang.reflect.Proxy
2. 行为模式

Chain of Responsibility(责任链):一个对象在一个链接传递直到被处理

1. java.util.logging.Logger.log()
2. javax.servlet.Filter.doFilter()

Command(命令):将一系列的命令封装成在一个类中

1. java.lang.Runnable
2. javax.swing.Action

Interpreter(解释器):定义语言的文法 ,并且建立一个解释器来解释该语言中的句子

1. java.util.Pattern
2. java.text.Normalizer
3. java.text.Format

Iterator(迭代器):对象遍历

1. java.util.Iterator
2. java.util.Enumeration

Mediator(中介):用一个中介对象来封装一系列关于对象交互行为

1. java.util.Timer
2. java.util.concurrent.Executor.execute()
3. java.util.concurrent.ExecutorService#submit()
4. java.lang.reflect.Method.invoke()

Memento(快照):保存另外一个对象内部状态拷贝的对象

1. java.util.Date
2. java.io.Serializable  额

Observer(观察者):事件监听器

1. java.util.EventListener
2. javax.servlet.http.HttpSessionBindingListener
3. javax.servlet.http.HttpSessionAttributeListener
4. javax.faces.event.PhaseListener

State(状态):不同的状态,不同的行为;或者说,每个状态有着相应的行为.

1. java.util.Iterator
2. javax.faces.lifecycle.LifeCycle.execute()

Strategy(策略):定义一系列的算法,把这些算法一个个封装成单独的类

1. java.util.Comparator.compare()
2. javax.servlet.http.HttpServlet
3. javax.servlet.Filter.doFilter()

Template(模板):定义一个操作中算法的骨架,将一些步骤的执行延迟到其子类中

1. java.util.Collections.sort()
2. java.io.InputStream.skip()
3. java.io.InputStream.read()
4. java.util.AbstractList.indexOf()

Visitor(访问者):作用于某个对象群中各个对象的操作. 它可以使你在不改变这些对象本身的情况下,定义作用于这些对象的新操作.

1. javax.lang.model.element.Element
2. javax.lang.model.element.ElementVisitor
3. javax.lang.model.type.TypeMirror
4. javax.lang.model.type.TypeVisitor

### 在Java中，什么是interface，abstractclass？Interface和abstract class有何区别？

interface : java中的接口是，程序的规范,当中可以定义常量和抽象方法(它允许使用者确定方法名，参数列表和返回值,但是没有方法体)，接口提供了形式,但是没有提供具体的实现。

abstract Class：抽象类，抽象类是接口和普通类之间的一种实现方式，因为不能总是使用纯接口，但是又必须使用抽象方法的时候，抽象类就成了用于此目的的重要工具。

abstract method：仅有声明而没有方法体的方法，就成为抽象方法，包含抽象方法的类就叫抽象类，这个时候的类就不需要加上abstract的限定，否则编译错误。

Interface和abstract class的区别：

* 1. 接口中不允许有方法实现，抽象类中，可以存在抽象或者非抽象方法。
  2. 接口中只能有静态的不可修改的成员(static final)。
  3. 因为java只允许单集成，所以抽象作为类型定义，受到了极大的限制，但是接口可以多实现。
  4. 设计理念不同，abstract class 表示的是“is-a”关系，而interface表示“like-a”关系。
  5. 抽象类中的变量默认是friendly型，其值可以在子类中重新定义，也可以重新赋值。
  6. 接口中的方法默认都是public abstract类型的。

### String和StringBuffer有什么区别？

1. String是字符串常量，字符串长度不可变！
2. StringBuffer字符串变量(Synchronized,线程安全)，如果要频繁对字符串内容进行修改，处于效率考虑最好使用StringBuffer，如果想转换成String类型，可以调用StringBuffer的toString();StringBuffer上的主要操作是append和insert方法，可重载这些方法，已接受任意类型的数据。
3. （\*延伸）StringBuilder字符串变量（非现场安全），这是JDK5.0新增，该对象是一个线程不安全的字符串容器，在单线程操作的情况下此对象可以用于取代StringBuffer。
4. 两者区别：主要体现在性能上，每次对存放进String当中的字符串进行更改的时候，都会生成一个新的String对象，然后指针指向该对象，而一旦当产生的没有引用指向对象过多的时候，GC就会回收，降低性能；而使用StringBuffer的时候，每次都会对该对象本身进行操作，而不是去创建一个新的对象，所以在多数情况下还是推荐使用。
5. String某些情况下不比StringBuffer慢；如：

String str = "string1"+"string2";

StringBuffer sb = **new** StringBuffer();

sb.append("string1").append("string2");

在这里String str = "string1"+"string2";<==> str = "string1string2";

### 说明链表、哈希表、数组的特点（靖城远景公司）

1. 链表是一种物理存储单元上非连续，非顺序的存储结构，数据元素的逻辑顺序是通过链表中的指针链接次序实现的。
   1. 链表的存储在内存中可以是非连续性的，因为链表结构可以通过自身节点中的数据域指针域来找到下一个节点。
   2. 对链表进行删除插入操作时，只需要将其指针域进行修改即可（相对于数组来说内部操作更便捷）
   3. 链表本身不存在下标，所有查询效率略低。

\*：一般而言进行删除修改等操作的时候使用的是链表结构，而查询的时候则使用数组结构

\*：Java中由于linked\*的内部实现是采用链表结构。

1. 散列表（Hashtable，也叫哈希表），是根据关键码值（Key Value）而直接进行访问的数据结构
   1. 哈希表最大的优势，就是把数据的存储和查询消耗的时间大大降低，几乎可以看成是常数时间。损失的代价仅仅是消耗比较多的内存，然而在当前可利用内存越来越多的情况下，用空间换时间的做法是相当值得的。而且编码也比较容易。
   2. 散列表查询速度快的原因：
      1. 将键值保存在某处，以便于能很快找到（数组中，这里保存的不是键本身而是键的信息，数组的下标就是这个对象的hashCode）
      2. 查询的过程就变成了，首先生产该对象的HashCode，然后查询数组，，然后再去保存值的list当中查询
2. 数组是一种物理存储单元上连续，顺序的存储结构，可以通过下标访问数组元素。
   1. 数组的保存效率高并且具备保存基本类型的能力。
   2. 数组是一种简单的线性序列，这使得访问速度非常快。
   3. 数组在定义时其大小被固定，并且在其声明周期中不可改变。
   4. 数组的查询速度，相对来说是比较快的，因为可以对其索引进行快速便利。

### ArrayList、LinkedList和Vector的区别

ArrayList：底层数据结构是数组，运行所有元素，包括null，查询快，增删慢，线程不安全(非同步)，效率高！

Vector：底层数据结构是数组，查询快，增删慢，线程安全（同步）,效率低~！

LinkedList：底层数据结构是链表，允许null元素，查询慢，增删快，线程不安全（非同步），效率高！

### 多线程有几种实现方法，都是什么？启动一个线程调用的是哪个方法？

Java中Thread和Runnable是实现并发线程的两种方式！

区别：

* 1. Thread是一个Class，Runnable是一个interface
  2. 通过extends Thread，可获得start(),sleep(),并覆盖run()，使用时直接调用start()方法即可，但是有时不方便通过继承而进行多线程，如已经继承其他的class时，就需要实现实现Runnable接口了，因为Java是不允许多继承形式的出现，但是可以使用多实现。
  3. 而implement Runnable接口的时候，创建的只有run()方法，而没有start()方法，因此要利用其他机制来启动。这就要创建两个对象，第一个是Runnable的，第二个是Thread的，然后把第一个当成参数传递给第二个；
  4. \*：使用Runnable实现多线程可以到达资源共享的目的，让多个Thread对象接收同一个Runnable实现类的参数，这样就是多个线程同时操作同一块资源。

**private** **class** RunnableThread **implements** Runnable {

//这个方法中写,多线程执行的程序!!!,因为这里是实现了Runnable所以就算再去继承其他的class也不会违反java成不能多继承的规定!!!

**public** **void** run(){

    }

}

//通过将Runnable的实现类作为参数传递给一个Thread对象,然后通过开启Thread对象,就能让其执行自己复写的方法,并且是并发操作!!!

    Thread runnThread = **new** Thread(**new** RunnableThread());

          RunnThread.start();

          //1,继承自 Thread类的多线程程序!!!

**private** **class** GenThread **extends** Thread{

        //这个方法中写,多线程需要执行的方法!!!

**public** **void** run() {

            }

        }

        Thread theGenThread = **new** GenThread();

        theGenThread.start();

### Class.forName()的作用？

1. 装载一个类并且对其进行实例化的操作。
2. 装载过程中使用到的类加载器是当前类。

\*：为了详细描述，Class.forName()方法，这里使用classLoader.loadClass()进行对比描述

* 1. Class.forName(String className)使用装载当前类的类装载器来装载制定的类，因为class.forName(String name)

方法内部调用了Class.forName(className,true,this.getClass().getClassLoader())方法。

* 1. classLoader.loadClass(StringclassName,boolean resolve);需要手动制定装载器的实例。
  2. Class.forName(className)装载的class已经被实例化，classLoader.loadClass().则只是将信息装载给JVM。
  3. 在JDBC中Class.forName(“com.mysql.jdbc.Driver”)，如果换成getClass().getClassLoader().loadClass(“com.mysql.jdbc.Driver”)，就不可以，因为它只是想JVM装载了Driver的类信息，但是没有实例化，也就不能执行相应的操作，因为Driver是需要被初始化才能被使用的。

### 请列出最少3中J2EE 体系内组件、服务或API

### 请列出至少5中使用过关的开源软件

1. tomcat – Apache极具影响力的开源软件
2. struts
3. jdbc
4. maven
5. activiti
6. jfreechart
7. jxl

### 完成一个Map的toString()方法，要求输出该Map中所有键-值信息

**publicclass** MyMap<K, V>**extends** AbstractMap<K, V> {

**private** List<K>keys = **new** ArrayList<K>();

**private** List<V>values = **new** ArrayList<V>();

//put操作.

**public** V put(K key, V value) {

V oldValue = get(key);

**if** (!keys.contains(key)) {

keys.add(key);

values.add(value);

} **else**

values.set(keys.indexOf(key), value);

**return** oldValue;

}

//get操作,进行键值输出的必要方法.

**public** V get(Object key) {

**if** (!keys.contains(key))

**returnnull**;

**return**values.get(keys.indexOf(key));

}

@Override

**public** String toString() {

**return**"MyMap [keys=" + keys + ", values=" + values + "]";

}

@Override

**public** Set<java.util.Map.Entry<K, V>> entrySet() {

**thrownew** RuntimeOperationsException(**null**, "不可操作异常.");

}

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

MyMap<String, String> myMap = **new** MyMap<String,String>();

myMap.put("信息1","大");

myMap.put("信息2","大");

myMap.put("信息3","大");

myMap.put("信息4","大");

myMap.put("信息5","大");

myMap.put("信息6","大");

System.*out*.println(myMap);

}

}

### 一个类是否可以继承多个接口？

1、在Java中接口是允许多实现的，接口和接口之间可以进行多实现，大部分的类也都是实现多个接口。

### 一个类是否可以继承多个超类？

1、类的继承，同一层次结构只能继承一个类。

### 接口是否可继承接口？

1、接口之间不能实现，因为接口本身就是抽象的，但是接口可以继承接口的，并且可以进行多继承（因为抽象，所以不担心方法的冲突）

### 抽象类是否可以实现（implement）接口？

1、可以，抽象类也是类，类是可以实现接口的。

### 如何将字符串转换为Integer？

1、Integer parseInt = Integer.parseInt(“123”);

### 如何将日期类型转换为”xxxx年xx月xx日“格式的字符串？

@Test

**publicvoid**formatDate(){

Date time = **new** Date();

DateFormat df = **new** SimpleDateFormat("yyyy年MM月dd日-hh:mm:ss");

System.*err*.println(df.format(time));

}

### 如何将字符串bbbwlirbbb转换为bbbhhtccc?

@Test

**publicvoid** change(){

String str = "bbbwlirbbb";

String str1 = str.replaceAll(str, "bbbhhtccc");

System.*err*.println(str1);

}

### 文件拷贝，把一个文件的内容拷贝到另外一个文件里

### 项目中使用过哪些JSTL标签

导入库：

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>

<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>

core库：（最常用）

1. 表达式控制标签：out-输出、set、remove、catch
2. 流程控制标签：if – 判断标签、choose、when – 条件选择、otherwise
3. 循环标签：ForEach – 遍历标签、forTokens
4. URL操作标签：import、url、redirect

fmt库：格式化系列

functions库：主要用来处理一些字符串的操作系列

### 现在输入n个数字，以逗号，分开；然后可选择升或降序排序

**publicclass** SortNum {

@Test

**publicvoid**SortNum() {

@SuppressWarnings("resource")

Scanner sc = **new** Scanner(System.*in*);

System.*out*.println("请输入,需要进行排序的数字(格式 : 233,53,36),回车结束输入.");

String nextLine = sc.nextLine();

String[] split = nextLine.split(",");

Integer[] temp = **new** Integer[split.length];

**int** index = 0;

**try** {

**for** (String string : split) {

temp[index++] = Integer.*parseInt*(string);

}

} **catch** (NumberFormatException numberFormatException) {

System.*out*.println("请输入正确的内容以及格式.");

**return**;

}

System.*out*.println("你输入的是 : " + Arrays.*toString*(temp));

System.*out*.println("请选择,升序(0)还是降序(1)");

**int** nextInt = sc.nextInt();

Arrays.*sort*(temp);

**if**(nextInt == 1) {

StringBuilder stringBuilder = **new** StringBuilder(Arrays.*toString*(temp));

String substring = stringBuilder.substring(1,stringBuilder.length()-1);

StringBuilder sTemp = **new** StringBuilder(substring).reverse();

System.*out*.println("降序 : " + sTemp);

}**else** {

System.*out*.println("升序 : " + Arrays.*toString*(temp));

}

}

}

### 编程现实某一文件目录下的文件名

### 编程计算N！的程序，一个使用递归方法，一个不用递归方法

方法一：递归思想（自调）

@Test

**publicint** stratum(**int** num){

**if**(num == 1){

**return** 1;

}**elseif**(num == 2){

**return** 2;

}**else**{

**return** num\**stratum*(num-1);

}

}

方法二：非递归思想

**publicstaticint** stratum1(**int** num){

**int** jiecheng = 1;

**if**(num == 0){

**return** 0;

}

**for** (**int** i = 1; i <= num; i++) {

jiecheng\*=i;

}

**return** jiecheng;

}

### 将一个正整数分解质因数。例如：输入90，打印出90=2\*3\*3\*5

@Test

**publicstaticvoid** printNum(**int** n){

System.*err*.print(n+"=");

**for** (**int** j = 2; j <=n; j++) {

**while**(n%j == 0){

n/=j;

System.*err*.print(j);

**if**(n!=1){

System.*err*.print("\*");

}

}

}

}

### 请编写代码循环输出一个对象中所有的key和value

**publicclass** GetAll {

@Test

**publicvoid**getAll(){

Map<String, String> map = **new** LinkedHashMap<String,String>();

map.put("1", "张三");

map.put("2", "李四");

map.put("3", "王五");

//1、使用for循环遍历

**for** (Map.Entry<String, String> entry:map.entrySet()) {

System.*err*.println(entry.getKey()+":"+entry.getValue());

}

//2、使用迭代的方式

Set<Entry<String, String>> set = map.entrySet();

Iterator<Entry<String, String>> i = set.iterator();

**while**(i.hasNext()){

Map.Entry<String, String> entry = i.next();

System.*err*.println(entry.getKey()+"---"+entry.getValue());

}

//3、使用keySet的方式

Iterator<String> it = map.keySet().iterator();

**while**(it.hasNext()){

String key = it.next().toString();

String value = map.get(key);

System.*err*.println(key+">>>>>>>>"+value);

}

}

}

### switch是否能作用在byte上，是否能作用在long上，是否能作用在String上？

Only convertible int values, strings or enum variables are permitted

Int、Integer、String、enum、byte

测试String类型，JDK1.7新特性

**publicclass** SwitchDemo {

@Test

**publicvoid** testSwitch(){

String str = "abc";

**switch** (str) {

**case**"a":System.*err*.println("ERROR!!!");

**case**"ab":System.*err*.println("ERROR~~~");

**case**"abc":System.*err*.println(str == "abc");

**break**;

**default**:System.*err*.println("NO RIGTH NUMBER!");

**break**;

}

}

}

### J2EE是技术还是平台还是框架？

J2EE本身是一个标准，一个为企业分布式应用的开发提供的标准平台。

J2EE 也是一个框架，包括JDBC、JNDI、RMI、JMS、EJB、JTA等技术。

### 创建对象的方式有几种？

1. new
2. 反射
3. 反序列化
4. Clone

## 前台

### Javascript中，对象person有name属性，写出两种获取name属性的方法

1. <!--第一种：使用“ . ”来访问对象属性-->
2. Person.name
3. <!--第二种：使用“[]”来访问对象属性-->
4. Person[name]

### 在HTML中如何包含JavaScript？

1. <!-- 方法一：通过外部源文件来包含javascript,这是最正确的方式,把结构从行为中分离出来 -->
2. **<head>**
3. **<script** type=”text/javascript”src=”source.jsp”**></script>**
4. **</head>**
6. <!-- 方法二：在<head>中包含js代码 -->
7. **<head>**
8. **<script**  type=”text/javascript” **>**
9. //javascript代码
10. **</script>**
11. **</head>**
13. <!-- 方式三：把嵌入式的脚本添加到文档的<body>中, 维一的理由是要使用document.write()方法,但我们有更好方法来替代他。如:createElement()方法,  appendChild()方法, 非EMCAscript标准的 innerHTML属性-->
14. **<body>**
15. **<script**  type=”text/javascript” **>**
16. document.write(“html....”);
17. **</script>**
18. **</body>**
20. <!-- 方式四：非官方EMCA标准的 javascript:前缀 ;可以成功打开一个新的窗口-->
21. **<a** href="javascript:window.open('http://www.google.com');" **>**进入**</a>**

### 使用js和JQuery的方式获取表达中对象

1. Js  :
2. document.getElementById(“aa”)查询id为aa的元素，
3. document.getElementsByName(“aa”)取name为aa的元素，
4. document.getElementsByTagName("aa")获取标签名
5. jquery对应有$("#aa"),$("div[name=aa]"),$("div")
6. 如$(".aa")取**class**为aa的元素，$(":text")取页面type=text的输入元素等很多，功能比js强大很多
7. //document.getElementById(“xxx”)
8. //jQuery(“#xxx”);

### JQuery中什么方法用来把内容追加到指定元素的末尾？

1. JQeuery可以使用
2. after() 方法在被选元素后插入指定的内容。
3. 语法：  $(selector).after(content)
4. $(“p”).after(**function**(){
5. //为 p元素的后面需要添加的内容
6. });

### JQuery给按钮添加点击事件？

1. JQuery给 按钮添加事件的语法格式是：$(selector).click()
2. 例如：$(selector).click(**function**(){
3. //添加JQuery代码
4. });

### 同步和异步有何异同？什么在情况下分别使用？举例说明

1. /\*1、举个例子：普通B/S模式（同步）AJAX技术（异步）
2. 同步：通常的意义就是当做了一件与服务器进行交互的事情之后,需要等待服务的响应,然后再进行下一步的操作,也就是常规的与服务器交互的操作机制!!!
3. 步骤 : 提交请求->等待服务器处理->处理完毕返回 这个期间客户端浏览器不能干任何事
4. 异步: 当用户进行了与服务器的交互操作之后可以继续做其他的事情,而不需要进行等待,因为其所作的更改对于整个页面来说也是局部操作!!!
5. 步骤 : 请求通过事件触发->服务器处理（这是浏览器仍然可以作其他事情）->处理完毕 \*/
7. //2、
8. $.ajax({
9. type: "POST",
10. url: "DoServlet",
11. //传输格式,这里是false,同步传输!!!,通过该参数,修改 传输类型//true则表示为异步传输
12. async: **false**,
13. success: **function**(data){
14. //获取data数据!!!
15. }
16. });
18. /\*3、使用场景!!!
19. 第一、请求的提交是为了页面数据的显示，这时候用户一般不希望看到页面的刷新，是使用AJAX的一个最佳时候。--异步
20. 第二、如果请求提交后，用户能从页面感觉到提交结果，这时候，也最好不要有页面刷新，推荐使用AJAX技术。--异步
21. 第三、如果请求提交后，用户不能从页面感觉到提交动作，如绝大多数时候的数据的增加和修改，这时候则需要页面刷新，不能使用AJAX技术。
22. 第四、复杂的UI，以前对于复杂的C/S模式的UI，B/S模式一向采取逃避的方法，现在则可以放心大胆的使用AJAX来加以解决。\*/

### Extjs使用中firebug返回一个TypeError：c is undfined，请问意思？

第一你没有按照extjs的标准传入参数或者传入的参数是空的

第二你的对象在当前页面不存在。

可以确定是你的代码出错，调试方法：逐步注释自己写的js代码，直到无错误，再逐步恢复至有错误，

重复持过程不断缩小范围，直到找到错误位置，再仔细分析错误原因当然对自己水平很有信心可以加载压缩前的ext代码，那样根据错误位置可以大致判断错误原因

### 什么是Ajax

1，AJAX =异步JavaScript和XML。

AJAX是用于创建快速，动态网页的技术。

AJAX允许网页可以通过异步交换少量数据与幕后的服务器进行更新。这意味着，有可能更新网页的部分，无需刷新整个页面。

经典的网页，（不使用AJAX）必须重新加载整个页面如果内容应该改变。

谷歌地图，Gmail，YouTube和Facebook的标签：利用AJAX应用的例子，当然最常见的百度搜索框也是使用的AJax

1. 2，AJAX使用
2. 1,//原始使用方式!!!
3. **function** showHint(str) {
4. //判断字符串是不是为空,如果是空,那就直接返回!!!
5. **if** (str.length == 0){
6. document.getElementById("txtHint").innerHTML = "";
7. **return**;
8. } **else** {
9. //创建一个AJax引擎对象!!!
10. **var** xmlhttp = **new** XMLHttpRequest();
11. //设置回调函数!!!
12. xmlhttp.onreadystatechange = **function**(){
13. //判断请求状态是不是成功!!!,成功就获取响应的数据,否则数据获取是没有意义的!!!
14. **if** (xmlhttp.readyState == 4 && xmlhttp.status == 200){
15. document.getElementById("txtHint").innerHTML = xmlhttp.responseText;
16. }
17. }
18. //打开链接,第一个参数是请求方式,第二个参数是请求地址!!!
19. xmlhttp.open("GET", "gethintServlet?q=" + str);
20. //请求体,如果求情方式是Post这里必须填写!!!
21. xmlhttp.send();
22. }
23. }
24. 2,//JQuery使用方式!!!
25. $.ajax({
26. //请求地址
27. url: "gethintServlet",
28. //请求方式
29. type:"Get",
30. //回调函数
31. succeed:**function** (mes){
32. //这个mes就是
33. }
34. });

### ext的常用panel

一、属性

frame：布尔类型true 或 false ，为true时表示边框为圆角且具有背景色，取false则反之。

xtype：在EXTJS的可视化组件部署中的一种机制，即通过指定xtype的值，来告诉容量如何初始化所包含的级件，如xtype:"textfiled",表示使用Ext.form.TextFile来进行初始化当前组件。

二、方法

add(Ext.Component/Object\_component)：添加一个组件到面板中，运行时方法。

addButton( String/Object config, Function handler, Object scope ) :添加一个按钮到面板中，设计时方法（对象在未构造之前调用）。

\_panel.addButton({text:"确 定"}); //form按默认宽度自适应创建

\_panel.addButton(new Ext.Button({text:"取消",minWidth:200}));

三、事件

render：当前对象被构造后触发（设计时事件）。

renderto（构造参数）：将当前对象所生成的HTML对象存放在指定的对象中（运时时事件）。

两者不能同进使用，否则render不起作用。

四、构造参数

items：指定包含在面板中组件的配置数组如textField。

buttons：指定包含面板中按钮的配置数组。

### 简述Ajax原理、以及如何实现刷新数据及优点？请简述JVM原理？内存泄露与溢出的区别，何时产生内存泄露？（靖城远景公司）

### JavaScript的数据类型有哪些？

### JavaScript中对象和函数的创建都有哪种方法

### JavaScript中描述一下闭包的作用

### 编写JavaScript实现点击某添加按钮后表格中自动添加一行的功能。

## Javaweb部分

### Servlet中forward()与redirect()的区别是什么？

redirect() 会丢失request的所有信息 它属于页面级的重定向，仅仅让你的浏览器重新访问一个新的url，作为浏览者，能很明显的看到浏览器url地址的变化，这和点击了一个普通的超链接的后果是一样的。

而 RequestDispatcher 的forward() 方法，是转发，需要request 和 response最为参数，就是将用户的请求，连同请求信息等内容，一起转发到服务器的另外一个servlet去处理，它不会丢失request信息。这一过程是服务器内部完成的，作为访问者，是感觉不到了，或者说是透明的，因此访客浏览器的url 是不会发生变化的。

forward是转发，redirect是跳转。相同点都是为了两个组件之间的相互调用。forward的运行原理是服务器端内部的调用，所以它不需要通过浏览器来请求，所url地址不会改变，request的作用范围也没有中断过，它的作用域仍然有效。而redirect是重定向，是服务器发送消息告诉浏览器，让浏览器重新向新的地址请求，所以，url地址是会改变的，由于重新请求了，因而request也中断了，重新请求了，它的作用范围也失效了，放在request里面的信息也随之而清空了。

### 列举JSP的内置对象，同一应用中页面有哪些方法？

九大内置对象:

1.输出输入对象:request对象、response对象、out对象

2.通信控制对象:pageContext对象、session对象、application对象

3.Servlet对象:page对象、config对象

4.错误处理对象:exception对象

① 、直接在URL请求后添加 如：

< a href="thexuan.jsp?action=transparams&detail=directe">直接传递参数< /a> 特别的在使用response.sendRedirect做页面转向的时候，也可以用如下代码： response.sendRedirect("thexuan.jsp?action=transparams&detail=directe") ，可用request.getParameter(name)取得参数

②、jsp:param 它可以实现主页面向包含页面传递参数，如下：

1. **<** **jsp:include** page="Relative URL"**>**
2. **<** **jsp:param** name="param name" value="paramvalue" **/>**
3. **<** /**jsp:include >**

还可以实现在使用jsp:forward动作做页面跳转时传递参数，如下：

1. **<** **jsp:forward** page="Relative URL"**>**
2. **<** **jsp:param** name="paramname" value="paramvalue" **/>**
3. **<** /**jsp:forward >**

通过这种方式和一般的表单参数一样的，也可以通过request.getParameter(name)取得参数

3、设置session和request通过显示的把参数放置到session和request中，以达到传递参数的目的 session.setAttribute(name,value); request.setAttribute(name,value) 取参数： value=(value className)session.getAttribute(name);

value=(value className)request.getAttribute(name);

大家肯定已经注意到了，在取参数的时候，做了类型转换，这是因为放置在session和

request中的对象的属性被看作 java.lang.Object类型的了，如果不转换，在将直付给value时会报classcastexception异常。

### JSP如何获取HTML FORM中的数据？

html页面的表单

1. **<form** action="myjsp.jsp" method="post"**>**
2. 你要输入的数据：**<input** type="text" name="data "**>**
3. **</form>**

在表单中，action动作和method合起来表示把数据交给mujsp.jsp页面去处理，

在jsp页面呢，用

**<**% request.getParmerter("data");%**>**获取html页面传递过来的数据，

要注意的是，有时候连个页面之间的传递会出现中文乱码现象，

解决乱码的做法是：在jsp页面加上

1. **<**% request.setCharacterEncoding("GB2312"); %**>**

这样就不用担心编码问题了

另一种：是form表单提交方式获取表单数据。

### JSP中redirect和forward的区别是什么？（类似1）

forward是使用同一个request，也就是说用forward传递后可以在页面中获取request

中的变量或属性。

redirect是新建一个request用新建的request传递。

比如有三个jsp文件，第一个里面有<input name="name"> 提交到第二个页面进行

处理，若想在第三个页面中获取name的值就要用forward。

### JSP中对象的传递有哪些方法？

request，session，application，cookie

区别：page的生命周期最短,依次是request，session，application.他们保存在服务器端.

cookie是保存在客户端的,用来解决HTTP无状态问题.

范围不同,最小的是page然后request，session，application范围依次增大

### Http协议中返回状态200、404、302、301、500、502各式什么含义？

200 (成功) 服务器已成功处理了请求。 通常，这表示服务器提供了请求的网页。

301 (永久移动) 请求的网页已永久移动到新位置。 服务器返回此响应(对 GET 或

HEAD 请求的响应)时，会自动将请求者转到新位置。

302 (临时移动) 服务器目前从不同位置的网页响应请求，但请求者应继续使用原有位置

来进行以后的请求。

404 (未找到) 服务器找不到请求的网页。

500 (服务器内部错误) 服务器遇到错误，无法完成请求。

502 (错误网关) 服务器作为网关或代理，从上游服务器收到无效响应。

### 描述Cookie和Session的作用，区别和各自的应用范围，Session工作原理

实现sessionid的保存和传输。是cookie，将http的请求变得有状态。session的实现是web服务

器的事情，不过很显然默认是利用会话cookie，也就是存放在浏览器内存里的那种cookie来实现的，在cookie里只是存放了SessionID，然后在服务器上建立了一张表，对应客户端的SessionID。这张表是存放在服务器上的进程中的，也就是服务器的内存里，也就是Application里。当客户端请求服务器的时候，如果服务器发现这是一个新的请求，就会分配给他一个SessionID，也就是在浏览器写入一个sessionID的cookie。你可以试一下，在你请求过一次服务器之后，在服务器遍历客户端所有的cookie，就会发现这个叫做SessionID的cookie。

但是，这只是session实现sessionid状态记录的一种方法，如果客户端浏览器禁用了cookie，很多

网站还是可以正常登录，还是可以正确识别浏览器的身份，这就是通过url重写的方式，每次交互都携带sessionid的信息。

还有一种方式，就是表单重写，有些服务器会在表单中增加隐藏域，来回传递sessionid。

所以在了解了这个session的原理之后我们可以自己来造一个自己的session，可以使用cookie，也

可以使用url重写方式，也可以使用表单方式。

session的结束问题

还有知道了session的原理就会明白一个session会话如何结束，除非服务器端可以清除一个session

会话，客户端是没有能力清除session的，关闭浏览器的时候并不能够结束当前的会话(忽略浏览器关闭事件)，在不依赖于cookie的session中最明显，你可以记下你的sessionID那串字符串，关闭浏览器，打开，把sessionID自己替换，你就可以恢复刚才的会话了。所谓的20分钟session超时，是服务器每隔20分钟对于不活动的session清除而已。

还有要指出的是依赖于cookie的session是存放在浏览器内存中的，所以一个浏览器的进程就会产

生一个session,可以通过任务管理器查看进程，对于IE来说，只有双击桌面上的浏览器快捷方式才能新启动一个进程的，当然其他类似启动iexplorer.exe的行为也可以

到这里你会发现在依赖于cookie的session中，是需要浏览器的支持的！

对于js的ajax请求，确切的说是xmlhttp这个组件发送的http请求，虽然是你使用js的代码自己制

造的，但是依然是借助于浏览器发送的，所以可以获得session。

### 请列举几种重复提交的方式？

F5重复提交、浏览器前进和后退按钮重复提交、使用frameset框架右键的此框架重新

载入：1、使用遮罩，让用户提交完成之前等待。提交成功后执行页面重定向（redirect）。转到提交成功信息页面；

### XML是什么？针对它常用的工具包是什么？

XML全称为Extensible Markup Language，意思是可扩展的标记语言。XML语法上和HTML比较相似，但HTML中的元素是固定的，而XML的标签是可以由用户自定义的，常用在保存关系型数据和配置文件。针对它常用的工具包是DOM4J

### 在web开发中，将GBK字符串，以utf-8的格式进行转换输出

new String("string".getBytes("GBK"),"UTF-8")；

### 有一个长字符串，它的内部包含多个子字符串、子字符串之间用空格隔离，给出算法：如何去掉里面重复的字符串

### java递归调用输出一个目录下的所有子目录及文件名称

1. **public** **void** fun(File file){
2. File[] files = file.listFiles();
3. **for**(File \_file : files){
4. **if**(\_file.isDirectory()){
5. fun2(\_file);
6. }**else**{
7. System.out.println(\_file);
8. }
9. }
10. }

### 一个月的图片的网站，用户每检索一次、阅读一张、下载一张图片就会扣除相应的签署，如果让你来设计计费模板、采用何种方式最能节省各方面的资源，谈谈想法！

### XML文档定义有几种形式？他们之间有何本质区别？解析XML文档有几种方式？

XML文档定义有几种形式：dtd文档类型定义和SchemaXML模式

本质区别：

XML Schema 和DTD都用于文档验证

XML Schema 是内容开放模型，可扩展，功能性强，而DTD可扩展性差。

XML Schema 支持丰富的数据类型，而 DTD不支持元素的数据类型，对属性的类型定义也很有限。

XML Schema 支持命名空间机制，而DTD不支持。

XML Schema 可针对不同情况对整个XML 文档或文档局部进行验证；而 DTD缺乏这种灵活性。

XML Schema 完全遵循XML规范，符合XML语法，可以和DOM结合使用，功能强大；而DTD 语法本身有自身的语法和要求，难以学习。

解析XML文档有几种方式：

DOM：所有内容都封装成对象，并且装入内存中，如果是大本文处理的时候，性能下降。可以做增删改查等一系列操作。

Sax:事件驱动.在读取的过程中.将遇到的文档的各个部分的内容.拆分各个事件.当我们遇到什么事件的时候,可以选择做相应事.sax只能做查询。

### XML的解析有哪几种方式，并用其中一种写出解析XML实例

XML的解析方式有很多种，目前常用的主要有:SAX解析和DOM解析

1、SAX解析是基于事件驱动的解析，处理的优点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。另一方面，由于应用程序没有以任何方式存储数据，所以sax解析不能对文档进行增删改的操作。

2、DOM 解析是的基于文档驱动的解析，类似树的处理，具有以下几个特点。首先，由于树在内存中是持久的，因此可以修改它以便应用程序能对数据和结构作出更改。它还可以在任何时候在树中上下导航，而不是像 SAX 那样是一次性的处理。DOM 使用起来也要简单得多。另一方面，在内存中构造这样的树涉及大量的开销。大型文件完全占用系统内存容量的情况并不鲜见。此外，创建一棵 DOM 树可能是一个缓慢的过程，因此不适合解析大文档。

### getParameter与getAttribute的区别？

HttpServletRequest类既有getAttribute()方法，也有getParameter()方法，这两个方法有以下区别：

（1）getAttribute是返回对象,必须先存再取才能取到值，否则取不到，即在域空间中获取值；getParameter返回字符串，一般用在两个Web组件之间为链接关系时，类似post，get等方式传入的数据，在被链接的组件中通过getParameter()方法来获得请求参数，即在请求的过程中获取值。

（2）HttpServletRequest类有setAttribute()方法，而没有setParameter()方法。

### xml有哪些解析技术？区别是什么？

### 你在项目中用到了XML技术的哪些方面？如何实现的？请写出常用的XML操作的API函数

### JSP中动态INCLUDE与静态的INCLUDE的区别

动态INCLUDE用jsp:include动作实现   
<jsp:include page="included.jsp" flush="true" />  
它总是会检查所含文件中的变化，适合用于包含动态页面，并且可以带参数   
<%@ include file="included.htm" %>  
静态INCLUDE用include伪码实现,定不会检查所含文件的变化，适用于包含静态页面

### [iframe,Frame中Session丢失的解决方法](http://blog.csdn.net/happydaisy1985/article/details/4492089)

在开发中，我们经常会遇到使用Frame来工作，而且有时是为了跟其他网站集成，应用到多域的情况下，而Iframe是不能保存Session的。因此，网上可以找到很多相关的文章，如果网站可以采用设置Web.Config中的配置：<sessionState mode="StateServer" stateConnectionString="tcpip=127.0.0.1:42424"   
sqlConnectionString="data source=127.0.0.1;Trusted\_Connection=yes" cookieless="false"   
timeout="40" />   
把cookieless="false"改成"true"就可以了。但也同样有个小问题，就是如果页面中采用Javascript的window.location.href=''这样的方式来重定向的话，系统会认为这是另一个新的请求，产生一个新的SessionId，导致原Session同样的丢失。所以对于重定向，还是使用Response.Redirect()为好。

除了Ifrmae有丢Session问题外，frameset也有同样的问题。Frameset的问题更不移定，是有时会丢，有时不会丢，这更认人头痛，在网上找到了一个方法，在页面page\_onload里添加一语句：   
Response.AddHeader("P3P","CP=CAO PSA OUR")

IE框架中访问不同域的网页不能带入COOKIE————是IE浏览器的一个BUG？   
偶然中发现一个IE浏览器的问题：   
站点aa.com的页面a.aspx中存在一个iframe，iframe的src为站点bb.com的b.aspx。b.aspx设置了一个COOKIE（不管临时保存还是长期保存），然后刷新aa.com/a.aspx，bb.com/b.aspx也会被刷新，这时候会发现第二次向bb.com/b.aspx的请求里面并没有带入第一次设置的COOKIE。   
更确切地说：IE浏览器丢掉了框架中不同域的COOKIE。   
  
查看浏览器的隐私选项，默认是“中”。现在把隐私的等级修改成“低”，再次做上面的测试，然后发现框架中的COOKIE可以被带入了。   
从这个现象来看，应该是IE出于安全考虑而做了限制。但是！！！处于不同的两个域的的页面，应该被看成打开了两个互不相关的浏览器窗口，大家各自设置自己的COOKIE，各自带入到下次请求中，只要互不影响，不应该受到限制。   
恩，那么，为什么要限制呢？看来还是微软的程序员对自己浏览器的安全没什么信心，担心同一浏览器窗口不同域的情况下，黑客仍可能获取另一域的COOKIE，所以干脆把其中一个域的COOKIE干掉！！！人家FireFox为什么就没这种限制？

IE6/IE7支持的P3P(Platform for Privacy Preferences Project (P3P) specification)协议默认阻止第三方无隐私安全声明的cookie，Firefox目前还不支持P3P安全特性，firefox中自然也不存在此问题了。

解决方法:   
很简单，在要嵌入的内容中(iframe指向的站点)输出P3P的主机头声明，步骤如下(Session variables are lost if you use FRAMESET in Internet Explorer 6)：   
1.打开IIS管理器 inetmgr   
2.选择被嵌入iframe源站点或者目录，右键点击打开属性框   
3.切换到HTTP头   
4.添加   
5.自定义HTTP头名: P3P   
6.自定义HTTP头值: CP="CAO PSA OUR"   
7.关闭属性框退出，即刻生效

## 框架部分

### Struts

#### Struts采用什么设计模式？并详细说明各个组成部分？

struts采用了MVC设计模式，严格按照mvc设计模式进行设计，实现解耦和，struts2经过前端控

制器来转发请求，控制请求流向(也就是我们的控制层)。然后Struts经过模型层去处理数据，把处理结果再交给控制层进行视图解析。这样一个过程严格按照MVC设计模式

#### Struts的Action是不是线程安全的？如果不是，有什么方法可以保证Action的线程安全，如果是，请说明原因

Struts1的实现底层是servlet，是单利的，所以在Struts1的开发过程中有很多的时

候是需要注意单例导致的线程安全性的问题。

Struts2核心是拦截器，是多例的，每一次的请求都会创建一个新的实例出来，所以它

本身是没有线程安全性问题的

#### struts的工作原理

Struts的流程

　　服务器启动后，根据web.xml加载ActionServlet读取struts-config.xml文件内容到内存。

　　以登陆为例:第一次进login.jsp会先实例化Form、把默认值赋给表单元素。

输入用户名密码提交表单、提交到action属性的login.do,通过ActionServlet读struts-config.xml文件找到 action下的path属性找到.do,通过name属性找form-beans中的form-bean的name属性得到ActionForm的包名类名,先实例化form，把表单的值填充给form，调用form的validate方法验证、ActionErrors返回null表示验证通过，否则失败返回input指定的页面.验证通过会实例化Action，执行Action的excute方法。

#### Struts2的值栈结构（ValueStack结构）

Action

JavaBean

Session

Request

Request内存地址

session内存地址

List

Map

### Hibernate

#### Hibernate的特点及其工作原理，为什么大多数公司都使用Hibernate？

Hibernate是一个开放源代码的对象关系映射框架，它对JDBC进行了非常轻量级的对象封装，使

得Java程序员可以随心所欲的使用对象编程思维来操纵数据库。 Hibernate可以应用在任何使用JDBC的场合，既可以在Java的客户端程序使用，也可以在Servlet/JSP的Web应用中使用,

另外hibernate还提供一级缓存和二级缓存。可以进行简单的数据库优化和简洁的HQL编写。使用

hibernate需要根据项目来确定。不能根据公司确定。

Hibernate缺点

(1) Hibernate在批量数据处理时有弱势

(2) 针对单一对象简单的增删查改，适合于Hibernate,而对于批量的修改，删除，不适合用Hibernate,

这也是OR框架的弱点；要使用数据库的特定优化机制的时候，不适合用Hibernate。

#### Hibernate中怎么样实现类之间的关系？

实体类间存在三种关系，一对一，一对多（或者说多对一），多对多。类与类之间的关系主

要体现在表与表之间的关系进行操作，它们都是对对象进行操作，我们程序中把所有的表与类都映射在一起，它们通过配置文件中的many-to-one、one-to-many、many-to-many、来实现关系的。

### Spring

#### 细阐述Spring事务机制的实现原理？

Spring的事务管理机制实现的原理，就是通过AOP，使用动态代理对所有需要事务管理的Bean进行加载，并根据配置在invoke方法中对当前调用的 方法名进行判定，并在method.invoke方法前后为其加上合适的事务管理代码，这样就实现了Spring式的事务管理。

#### Spring声明式事务的优点

执行情况提交或者回滚事务。

声明式事务最大的优点就是不需要通过编程的方式管理事务，这样就不需要在业务逻

辑代码中掺杂事务管理的代码，只需在配置文件中做相关的事务规则声明（或通过等价的基于标注的方式），便可以将事务规则应用到业务逻辑中。因为事务管理本身就是一个典型的横切逻辑，正是 AOP 的用武之地。Spring 开发团队也意识到了这一点，为声明式事务提供了简单而强大的支持。

声明式事务管理曾经是 EJB 引以为傲的一个亮点，如今 Spring 让 POJO 在事务管

理方面也拥有了和 EJB 一样的待遇，让开发人员在 EJB 容器之外也用上了强大的声明式事务管理功能，这主要得益于 Spring 依赖注入容器和 Spring AOP 的支持。依赖注入容器为声明式事务管理提供了基础设施，使得 Bean 对于 Spring 框架而言是可管理的；而 Spring AOP 则是声明式事务管理的直接实现者，这一点通过清单8可以看出来。

通常情况下，笔者强烈建议在开发中使用声明式事务，不仅因为其简单，更主要是因

为这样使得纯业务代码不被污染，极大方便后期的代码维护。

和编程式事务相比，声明式事务唯一不足地方是，后者的最细粒度只能作用到方法级

别，无法做到像编程式事务那样可以作用到代码块级别。但是即便有这样的需求，也存在很多变通的方法，比如，可以将需要进行事务管理的代码块独立为方法等等。

#### Spring的工作流程、以及核心机制？

核心机制：IOC-控制翻转。对象的创建交由spring容器创建。常用方式有构造器创建、静态工厂、实例工厂。底层使用反射技术实现。

DI-依赖注入。对象的依赖属性由容器注入。一般使用set方法注入。

#### Spring中的事物隔离级别有哪些？事物隔离和传播的特点

ISOLATION\_DEFAULT:默认级别，根据数据库选择对应隔离级别ISOLATION\_READ\_UNCOMMITTED :可以读取未提交数据ISOLATION\_READ\_COMMITTED:只能读取已提交数据，解决脏读问题(Oracle默认级别)ISOLATION\_REPEATABLE\_READ:是否读取其他事务提交修改后的数据，解决不可重复读问题(MySQL默认级别)ISOLATION\_SERIALIZABLE:是否读取其他事务提交添加后的数据，解决幻影读问题

#### Spring的工作机制以及为什么要使用？

Spring的工作记机制：

* 1. springmvc将所有的请求都提交给DispatcherServlet，它会委托应用系统的其他模块负责对请求进行真正的处理工作。
  2. DispatcherServlet查询一个或多个HandlerMapping，找到处理请求的Controller。
  3. DispatcherServlet请求提交到目标Controller
  4. Controller进行业务逻辑处理后，会返回一个ModelAndView
  5. Dispatcher查询一个或多个ViewResolver视图解析器，找到ModelAndView对象制定的视

图对象

* 1. 视图对象负责渲染返回给客户端。

AOP：让开发人员可以创建非行为性的关注点，称为横切关注点，并将它们插入到应用程序代码

中。使用AOP后，公共服务（日志、持久性、事务等）就可以分解成方面并应用到域对象上。同时不会增加域对象的对象模型的复杂性。

IoC：允许创建一个可以构造对象的应用环境，然后向这些对象传递他们的协作对象。如倒置所表

明的，IoC就像反过来的JNDI。没有使用一堆抽象工厂、服务定位器、单元素（singleton）和直接构造（straight construction），每一个对象都是用其协作对象构造的。因此由容器负责管理协作对象（collaborator）

Spring既是一个AOP框架，也是一个IoC容器。Spring最好的地方是它有助于您替换对象。有

了Spring，只要用JavaBean属性和配置文件加入依赖性（协作对象）。然后可以很容易地在需要时替换具有类似接口的写作对象。

#### AOP和IOC的概念以及在Spring中是如何应用？

1.Spring实现了工厂模式的工厂类（在这里有必要解释清楚什么是工厂模式），这个类名为BeanFactory（实际上是一个接口），在程序中通常BeanFactory的子类ApplicationContext。Spring相当于一个大的工厂类，在其配置文件中通过<bean>元素配置用于创建实例对象的类名和实例对象的属性。

2.Spring提供了对IOC良好支持，IOC是一种编程思想，是一种架构艺术，利用这种思想可以很好地实现模块之间的解耦。IOC也称为DI（Depency Injection），什么叫依赖注入呢？

1. Class Programmer
2. {
3. Computer computer = **null**;
4. **public** **void** code()
5. {
6. //Computer computer = new IBMComputer();
7. //Computer computer = beanfacotry.getComputer();
8. computer.write();
9. }
10. **public** **void** setComputer(Computer computer)
11. {
12. **this**.computer = computer;
13. }
14. }
15. <!--另外两种方式都由依赖，第一个直接依赖于目标类，第二个把依赖转移到工厂上，第三个彻底与目标和工厂解耦了。在spring的配置文件中配置片段如下： -->
16. **<bean** id=”computer” class=”cn.itcast.interview.Computer”**></bean>**
17. **<bean** id=”programmer” class=”cn.itcast.interview.Programmer”**>**
18. **<property** name=”computer”  ref=”computer”**></property>**
19. **</bean>**
20. <!-- 3. Spring提供了对AOP技术的良好封装， AOP称为面向切面编程，就是系统中有很多各不相干的类的方法，在这些众多方法中要加入某种系统功能的代码，例如，加入日志，加入权限判断，加入异常处理，这种应用称为AOP。实现AOP功能采用的是代理技术，客户端程序不再调用目标，而调用代理类，代理类与目标类对外具有相同的方法声明，有两种方式可以实现相同的方法声明，一是实现相同的接口，二是作为目标的子类在，JDK中采用Proxy类产生动态代理的方式为某个接口生成实现类，如果要为某个类生成子类，则可以用CGLI B。在生成的代理类的方法中加入系统功能和调用目标类的相应方法，系统功能的代理以Advice对象进行提供，显然要创建出代理对象，至少需要目标类和Advice类。spring提供了这种支持，只需要在spring配置文件中配置这两个元素即可实现代理和aop功能，例如，-->
21. **<bean** id=”proxy” type=”org.spring.framework.aop.ProxyBeanFactory”**>**
22. **<property** name=”target” ref=””**></property>**
23. **<property** name=”advisor” ref=””**></property>**
24. **</bean>**

#### Spring3中依赖注入方式？

依赖注入的3种实现方式分别是：

接口注入（interface injection）接口注入指的就是在接口中定义要注入的信息，并通

过接口完成注入。

Set注入（setter injection）Set注入指的就是在接受注入的类中定义一个Set方法，

并在参数中定义需要注入的元素。

构造注入（constructor injection）构造注入指的就是在接受注入的类中定义一个构造

方法，并在参数中定义需要注入的元素。

### SpringMVC

### ibatis/Mybatis

#### mybaits与Hibernate对比有哪些不同点

相同点：屏蔽jdbc api的底层访问细节，使用我们不用与jdbc api打交道，就可以

访问数据。jdbc api编程流程固定，还将sql语句与java代码混杂在了一起，经常需要拼凑sql语句，细节很繁琐。

mybatis的好处：屏蔽jdbc api的底层访问细节；将sql语句与java代码进行分

离;提供了将结果集自动封装称为实体对象和对象的集合的功能，queryForList返回对象集合，用queryForObject返回单个对象；提供了自动将实体对象的属性传递给sql语句的参数。

Hibernate是一个全自动的orm映射工具，它可以自动生成sql语句,mybatis需

要我们自己在xml配置文件中写sql语句，hibernate要比mybatis功能负责和强大很多。因为hibernate自动生成sql语句，我们无法控制该语句，我们就无法去写特定的高效率的sql。对于一些不太复杂的sql查询，hibernate可以很好帮我们完成，但是，对于特别复杂的查询，hibernate就很难适应了，这时候用mybatis就是不错的选择，因为mybatis还是由我们自己写sql语句。

### 框架组合

#### SSH/SSI中的工作流程

SSH 为 struts+spring+hibernate的一个集成框架，是目前较流行的一种Web应

用程序开源框架。

集成SSH框架的系统从职责上分为三层：表示层、业务逻辑层、数据持久层，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，管理struts和hibernate。

SSI的框架主要是由struts2，spring以及ibatis组成，或springMVC，spring以

及ibatis组成，他们负责各层之间的交互与协作，从而实现整个web端的功能实现与整合。Struts目前主要负责数据传递和控制方面，spring则依靠其强大的依赖注入技术实现了类似bean托管和整合等功能，当然这只是spring功能中的冰山一角，而ibatis作为一种轻量级的OR Mapping框架，提供了半自动化对象关系映射的实现，自由度相对于hibernate更高。SSI框架能够降低我们代码的耦合度，增强了代码的健壮性和可重用性，加快了开发速度，但是也有一些不足之处，比如由于三种框架的配置文件较多，也给我们带来了一些不便，特别是对于较小的应用来说更是如此。

### MVC

#### 描述在MVC架构模式下，表单从页面提交内容，最后到内容保存到数据持久层中间要做哪些步骤处理

#### MVC是什么?请简述在WEB程序中MVC如何实现？

#### 列举出UML语言常用的5中图名称

1、用例视图   
UML语言中的用例视图描述系统应具备的功能，也就是被成为参与者的外部用户所能观察到的功能。用例是系统的一个功能单元，可以被描述为参与者与系统之间的一次交互作用。参与者可以是一个用户或者另外一个系统。客户对系统要求的功能被当作多个用例在用例视图中进行描述，一个用例就是对系统的一个用法的通用描述。用例模型的用途就是列出系统中的用例和参与者，并显示哪个参与者参与了哪个用例的执行。用例视图是其他视图的核心，它的内容直接驱动其他视图的开发。   
2、逻辑视图   
逻辑视图描述用例视图中提出的系统功能的实现。与用例视图相比，逻辑视图主要关注系统内部，它既描述系统的静态结构（类、对象以及他们之间的关系），也描述系统内部的动态协作关系。系统的静态结构在类图和对象图中进行描述，而动态模型则在状态图、时序图、协作图以及活动图中进行描述。逻辑视图的使用者主要是设计人员和开发人员。   
3、并发视图   
UML语言中的并发视图主要考虑资源的有效利用、代码的并行执行以及系统环境中异步事件的处理。除了将系统划分为并发执行的控制以外，并发视图还需要处理线程之间的通信和同步。并发视图的使用者是开发人员和系统集成人员。并发视图由状态图、协作图、以及活动图组成。   
4、组件视图   
组件是不同类型的代码模块，它是构造应用的软件单元。组件视图描述系统的实现模块以及它们之间的依赖关系。组件视图中也可以添加组件的其他附加信息，例如资源分配或者其他管理信息。组件视图主要由组件图构成，它的使用者主要是开发人员。   
5、配置视图   
  
配置视图显示系统的物理部署，它描述位于节点上的运行实例的部署情况。配置视图主要由配置图表示，它的使用者是开发人员、系统集成人员和测试人员。配置视图还允许评估分配结果和资源分配。

## 数据库

### 明数据库主键、外键的作用。以及独立主键的好处及坏处

SQL的主键和外键的作用：

外键取值规则：空值或参照的主键值。

(1)插入非空值时，如果主键表中没有这个值，则不能插入。

(2)更新时，不能改为主键表中没有的值。

(3)删除主键表记录时，你可以在建外键时选定外键记录一起级联删除还是拒绝删除。

(4)更新主键记录时，同样有级联更新和拒绝执行的选择。

简而言之，SQL的主键和外键就是起约束作用。

### 简述什么是事物？事物存在哪些属性？

数据库事务是指作为单个逻辑工作单元执行的一系列操作。属性：ACID

1、一致性：将事务中所做的操作捆绑成一个原子单元，即对于事务所进行的数据修改等操作，要么全部执行，要么全部不执行。

2、原子性：事务在完成时，必须使所有的数据都保持一致状态，而且在相关数据中，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应该是正确的。

3、隔离性：由并发事务所做的修改必须与任何其他事务所做的修改相隔离。事务查看数据时数据所处的状态，要么是被另一并发事务修改之前的状态，要么是被另一并发事务修改之后的状态，即事务不会查看由另一个并发事务正在修改的数据。这种隔离方式也叫可串行性。

4、持久性：事务完成之后，它对系统的影响是永久的，即使出现系统故障也是如此。

### 如何写出高性能的SQL语句？

(1)整合简单,无关联数据库访问: 如果你有几个简单数据库查询语句,你可以把它们整合到个查询

中(即使它们的间没有关系)

(2)删除重复记录: 最高效删除重复记录思路方法 ( 使用了ROWID)例子: DELETE FROM

EMP E WHERE E.ROWID > (SELECT MIN(X.ROWID) FROM EMP X WHERE X.EMP\_NO = E.EMP\_NO);

(3)用TRUNCATE替代DELETE: 当删除表中记录时,在通常情况下, 回滚段(rollback

segments ) 用来存放可以被恢复信息. 如果你没有 COMMIT事务,ORACLE会将数据恢复到删除的前状态(准确地说是恢复到执行删除命令的前状况) 而当运用TRUNCATE时, 回滚段不再存放任何可被恢复信息.当命令运行后,数据不能被恢复.因此很少资源被,执行时间也会很短. (译者按: TRUNCATE只在删除全表适用,TRUNCATE是DDL不是DML)

(4)选择最有效率表名顺序(只在基于规则优化器中有效): ORACLE 解析器按照从右到左顺序

处理FROM子句中表名FROM子句中写在最后表(基础表 driving table)将被最先处理在FROM子句中包含多个表情况下,你必须选择记录条数最少表作为基础表如果有3个以上表连接查询,那就需要选择交叉表(ersection table)作为基础表, 交叉表是指那个被其它表所引用表.

(5)WHERE子句中连接顺序．: ORACLE采用自下而上顺序解析WHERE子句,根据这个原理,

表的间连接必须写在其它WHERE条件的前, 那些可以过滤掉最大数量记录条件必须写在WHERE子句末尾.

(6)SELECT子句中避免使用 ' \* ': ORACLE在解析过程中, 会将'\*' 依次转换成所有列名, 这个

工作是通过查询数据字典完成, 这意味着将耗费更多时间

(7)减少访问数据库次数: ORACLE在内部执行了许多工作: 解析SQL语句, 估算索引利用率,

绑定变量 , 读数据块等；

(8)在SQL\*Plus , SQL\*Forms和Pro\*C中重新设置ARRAYSIZE参数, 可以增加每次数据库访问

检索数据量 ,建议值为200 (9)使用DECODE来减少处理时间: 使用DECODE可以避免重复扫瞄相同记录或重复连接相同表. (11) 用Where子句替换HAVING子句；

### 请列举几种SQL优化的方法？

1:mysql所在服务器内核 优化；此优化可由系统运维人员完成

2：mysql配置参数优化（my.cnf）；此优化需进行压力测试来进行参数调整

3：sql语句及表优化

①、当只要一行数据时使用 LIMIT 1

②、为搜索字段建索引

③、在Join表的时候使用相当类型的列，并将其索引

④、千万不要 ORDER BY RAND()

⑤、避免 SELECT \*

⑥、永远为每张表设置一个ID

使用 ENUM 而不是 VARCHAR

⑧、尽可能的使用 NOT NULL

⑨、固定长度的表会更快

垂直分割

拆分大的 DELETE 或 INSERT 语句

越小的列会越快

### 一个表中的ID有多个记录，把所有这个id的记录查出，并显示共有多少条记录数（SQL实现）

1. **SELECT** id,COUNT(id) **FROM** `**table**` **GROUP** **BY** id **HAVING** COUNT(id) > 1
2. **SELECT** \* **FROM** (**SELECT** id,COUNT(id) sumid **FROM** `**table**` **GROUP** **BY** id) t **WHERE** t.sumid > 1

### 查询表A中存在ID重复三次以上的记录。（使用SQL实现）

1. **SELECT** id,COUNT(id) **FROM** `**table**` **GROUP** **BY** id **HAVING** COUNT(id) > 3
2. **SELECT** \* **FROM** (**SELECT** id,COUNT(id) sumid **FROM** `**table**` **GROUP** **BY** id) t **WHERE** t.sumid > 3

### 数据库连接池的工作机制是什么？

连接池的实现是以空间换时间。

J2EE服务器启动时会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。客户

端程序需要连接时，池驱动程序会返回一个未使用的池连接并将其表记为忙。如果当前没有空闲连接，池驱动程序就新建一定数量的连接，新建连接的数量有配置参数决定。当使用的池连接调用完成后，池驱动程序将此连接表记为空闲，其他调用就可以使用这个连接。

### 什么是JDBC

### PreparedStatement是什么？

PreparedStatement是java.sql包下面的一个借口，用来执行SQL语句查询，通过调用

connection.preparedStatement(sql)方法可以或得到PreparedStatement对象。数据库系统会对sql语句继续拧预编译处理，预处理语句预先编译好，者条预编译的sql查询语句能在建立的查询中重用，这样一来，比Statement对象的查询速度更快。

### 预编译处理的优势（9中的）

企业开发中很多地方都强制性的使用PreparedStatement来做SQL查询

* 1. PreparedStatement可以写动态参数化的查询，可以写带参数的sql查询语句，通过使用相同的sql语句和不同的参数值来做查询比创建一个不同的查询语句要好，如：

select num from tb\_user where id = ?

* 1. PreparedStatement比 Statement 更快

使用PreparedStatement最重要的一点好处是它拥有更佳的性能优势，SQL语句会预编译在数据库系统中。执行计划同样会被缓存起来，他允许数据库做了参数化查询。使用预处理语句比普通语句查询更快，因为做的工作更少（数据库对SQL语句的分析，编译，优化已经在第一次查询前完成）。为了减少数据库的负载，生产环节中的JDBC代码应该总是使用PreparedStatement。

* 1. PreparedStatement可以防止SQL注入式攻击，

1. strSQL
2. = "**SELECT**
3. \* **FROM** users **WHERE** **name** = '"
4. + userName + "'
5. and pw = '"+
6. **passWord** +"';"

恶意填入：

1. userName
2. = "1'
3. OR '1'='1";
5. **passWord**
6. = "1'
7. OR '1'='1";

最终变成了：

1. strSQL
2. = "**SELECT**
3. \* **FROM** users **WHERE** **name** = '1' OR '1'='1' and pw = '1' OR '1'='1';"

因为WHERE条件恒为真，这就相当于执行：

1. strSQL
2. = "**SELECT**
3. \* **FROM** users;"';"

因此可以达到无账号密码亦可登录网站。如果恶意用户要是更坏一点，用户填入：

1. strSQL
2. = "**SELECT**
3. \* **FROM** users;"

SQL语句变成了：

1. strSQL
2. = "**SELECT**
3. \* **FROM** users **WHERE** **name** = 'any\_value' and pw = ''; **DROP** **TABLE** users"

这样一来，虽然没有登录，但是数据表都被删除了。

然而使用PreparedStatement的参数化的查询可以阻止大部分的SQL注入。在使用参数化查询的情况下，数据库系统不会讲参数的内容视为SQL指令的一部分来处理，

而是在数据库完成SQL指令的编译后，才套用参数运行，因此就算参数中含有破坏性的指令，也不会被数据库所运行。

避免SQL注入的第二种方式：在组合SQL字符串的时候，先对所传入的参数做字符取代（将单引号字符取代为连续2个单引号字符，因为连续两个单引号字符在SQL数据库中会视为字符中的一个单引号字符），如下：

1. strSQL
2. = "**SELECT**
3. \* **FROM** users **WHERE** **name** = '"
4. + userName + "';"

传入字符串：

1. userName
2. = "
3. 1' OR 1=1 "

把userName做字符替换后变成：

1. userName
2. = "
3. 1'' OR 1=1"

最后生成的SQL查询语句为：

1. strSQL
2. = "**SELECT** \* **FROM** users **WHERE** **name** = '1''
3. OR 1=1'
   1. 可读性、安全性高

PreparedStatement的局限性

1. 为了防止SQL注入攻击，PreparedStatement不允许一个占位符?有多个值，在执行有IN子句查询的时候这个问题变得棘手，例如：以下使用PreparedStatement就不会返回任何结果。
2. **SELECT**
3. \* **FROM** loan **WHERE** loan\_type IN (?)
5. preparedSatement.setString(1,"'personal
6. loan', 'home loan', 'gold loan'");

### 左连接、右连接和内连接的区别是什么？

假设数据库中有A、B两张表

left join是以A表的记录为基础的,A可以看成左表,B可以看成右表,left join是以左表为准的.换句话说,左表(A)的记录将会全部表示出来,而右表(B)只会显示符合搜索条件的记录(例子中为: A.aID = B.bID).B表记录不足的地方均为NULL.

Right join以右表(B)为基础的,A表不足的地方用NULL填充.

inner join并不以谁为基础,它只显示符合条件的记录

### 表和视图的区别以及各自的用途

### when is more faster IN and EXISTS? Why?

### SQL编程题

#### 表结构：

1. --Student(S#,Sname,Sage,Ssex) 学生表
2. Student(sid,Sname,Sage,Ssex) 学生表
3. **CREATE** **TABLE** student (
4. sid **varchar**(10) NOT NULL,
5. sName **varchar**(20) **DEFAULT** NULL,
6. sAge datetime **DEFAULT** '1980-10-12 23:12:36',
7. sSex **varchar**(10) **DEFAULT** NULL,
8. **PRIMARY** **KEY** (sid)
9. ) ENGINE=InnoDB **DEFAULT** CHARSET=utf8;
10. Course(cid,Cname,tid) 课程表
11. **CREATE** **TABLE** course (
12. cid **varchar**(10) NOT NULL,
13. cName **varchar**(10) **DEFAULT** NULL,
14. tid **int**(20) **DEFAULT** NULL,
15. **PRIMARY** **KEY** (cid)
16. ) ENGINE=InnoDB **DEFAULT** CHARSET=utf8;
17. SC(sid,cid,score) 成绩表
18. **CREATE** **TABLE** sc (
19. sid **varchar**(10) **DEFAULT** NULL,
20. cid **varchar**(10) **DEFAULT** NULL,
21. score **int**(10) **DEFAULT** NULL
22. ) ENGINE=InnoDB **DEFAULT** CHARSET=utf8;
23. Teacher(tid,Tname) 教师表
24. **CREATE** **TABLE** taacher (
25. tid **int**(10) **DEFAULT** NULL,
26. tName **varchar**(10) **DEFAULT** NULL
27. ) ENGINE=InnoDB **DEFAULT** CHARSET=utf8;

#### 表数据：

1. **insert**  **into** taacher(tid,tName) **values** (1,'李老师'),(2,'何以琛'),(3,'叶平');
2. **insert**  **into** student(sid,sName,sAge,sSex) **values** ('1001','张三丰','1980-10-12 23:12:36','男'),('1002','张无极','1995-10-12 23:12:36','男'),('1003','李奎','1992-10-12 23:12:36','女'),('1004','李元宝','1980-10-12 23:12:36','女'),('1005','李世明','1981-10-12 23:12:36','男'),('1006','赵六','1986-10-12 23:12:36','男'),('1007','田七','1981-10-12 23:12:36','女');
3. **insert**  **into** sc(sid,cid,score) **values** ('1','001',80),('1','002',60),('1','003',75),('2','001',85),('2','002',70),('3','004',100),('3','001',90),('3','002',55),('4','002',65),('4','003',60);
4. **insert**  **into** course(cid,cName,tid) **values** ('001','企业管理',3),('002','马克思',3),('003','UML',2),('004','数据库',1),('005','英语',1);

Oracle表+数据

1. --ORACLE(表+数据)
2. **CREATE** **TABLE** student (
3. sid varchar2(10) NOT NULL,
4. sName varchar2(20) **DEFAULT** NULL,
5. sAge **date** ,
6. sSex varchar2(10) **DEFAULT** NULL,
7. **PRIMARY** **KEY** (sid)
8. )
10. **CREATE** **TABLE** course (
11. cid varchar2(10) NOT NULL,
12. cName varchar2(10) **DEFAULT** NULL,
13. tid number(20) **DEFAULT** NULL,
14. **PRIMARY** **KEY** (cid)
15. )
17. **CREATE** **TABLE** sc (
18. sid varchar2(10) **DEFAULT** NULL,
19. cid varchar2(10) **DEFAULT** NULL,
20. score number(10) **DEFAULT** NULL
21. )

24. **CREATE** **TABLE** teacher (
25. tid number(10) **DEFAULT** NULL,
26. tName varchar2(10) **DEFAULT** NULL
27. )
29. **insert**  **into** course(cid,cName,tid) **values** ('001','企业管理',3);
30. **insert**  **into** course(cid,cName,tid) **values** ('002','马克思',3);
31. **insert**  **into** course(cid,cName,tid) **values** ('004','数据库',1);
32. **insert**  **into** course(cid,cName,tid) **values** ('005','英语',1);
34. **insert**  **into** sc(sid,cid,score) **values** ('1001','001',80);
35. **insert**  **into** sc(sid,cid,score) **values** ('1001','002',60);
36. **insert**  **into** sc(sid,cid,score) **values** ('1001','003',70);
37. **insert**  **into** sc(sid,cid,score) **values** ('1002','001',85);
38. **insert**  **into** sc(sid,cid,score) **values** ('1002','002',70);
39. **insert**  **into** sc(sid,cid,score) **values** ('1003','004',90);
40. **insert**  **into** sc(sid,cid,score) **values** ('1003','001',90);
41. **insert**  **into** sc(sid,cid,score) **values** ('1003','002',99);
42. **insert**  **into** sc(sid,cid,score) **values** ('1004','002',65);
43. **insert**  **into** sc(sid,cid,score) **values** ('1004','003',50);
44. **insert**  **into** sc(sid,cid,score) **values** ('1005','005',80);
45. **insert**  **into** sc(sid,cid,score) **values** ('1005','004',70);
46. **insert**  **into** sc(sid,cid,score) **values** ('1003','003',10);
47. **insert**  **into** sc(sid,cid,score) **values** ('1003','005',10);

50. **insert**  **into** student(sid,sName,sAge,sSex) **values** ('1001','张三丰',to\_date('1980-10-12 23:12:36','YYYY-MM-DD HH24:MI:SS'),'男');
51. **insert**  **into** student(sid,sName,sAge,sSex) **values** ('1002','张无极',to\_date('1995-10-12 23:12:36','YYYY-MM-DD HH24:MI:SS'),'男');
52. **insert**  **into** student(sid,sName,sAge,sSex) **values** ('1003','李奎',to\_date('1992-10-12 23:12:36','YYYY-MM-DD HH24:MI:SS'),'女');
53. **insert**  **into** student(sid,sName,sAge,sSex) **values** ('1004','李元宝',to\_date('1980-10-12 23:12:36','YYYY-MM-DD HH24:MI:SS'),'女');
54. **insert**  **into** student(sid,sName,sAge,sSex) **values** ('1005','李世明',to\_date('1981-10-12 23:12:36','YYYY-MM-DD HH24:MI:SS'),'男');
55. **insert**  **into** student(sid,sName,sAge,sSex) **values** ('1006','赵六',to\_date('1986-10-12 23:12:36','YYYY-MM-DD HH24:MI:SS'),'男');
56. **insert**  **into** student(sid,sName,sAge,sSex) **values** ('1007','田七',to\_date('1981-10-12 23:12:36','YYYY-MM-DD HH24:MI:SS'),'女');
58. **insert**  **into** teacher(tid,tName) **values** (1,'李老师');
59. **insert**  **into** teacher(tid,tName) **values** (2,'何以琛');
60. **insert**  **into** teacher(tid,tName) **values** (3,'叶平');

#### SQL问题：

1. --1.查询“001”课程比“002”课程成绩高的所有学生的学号;
2. **select** a.sid **from** (**select** sid,score **from** SC **where** cid='001') a,(**select** sid,score
3. **from** SC **where** cid='002') b
4. **where** a.score>b.score and a.sid=b.sid;
6. --2、查询平均成绩大于60分的同学的学号和平均成绩;
7. **select** sid,avg(score)
8. **from** sc
9. **group** **by** sid **having** avg(score) >60;
11. --3、查询所有同学的学号、姓名、选课数、总成绩;
12. **select** Student.sid,Student.Sname,count(SC.cid),sum(score)
13. **from** Student left Outer join SC **on** Student.sid=SC.sid
14. **group** **by** Student.sid,Sname
16. --4、查询姓“李”的老师的个数;
17. **select** count(**distinct**(Tname))
18. **from** Teacher
19. **where** Tname like '李%';
21. --5、查询没学过“叶平”老师课的同学的学号、姓名;
22. **select** Student.sid,Student.Sname
23. **from** Student
24. **where** sid not in (**select** **distinct**( SC.sid) **from** SC,Course,Teacher **where** SC.cid=Course.cid and Teacher.tid=Course.tid and Teacher.Tname='叶平');
26. --6、查询学过“001”并且也学过编号“002”课程的同学的学号、姓名;
27. A:**select** Student.sid,Student.Sname **from** Student,SC **where** Student.sid=SC.sid and SC.cid='001'and exists( **Select** \* **from** SC **as** SC\_2 **where** SC\_2.sid=SC.sid and SC\_2.cid='002');
28. B:**SELECT**    s.sid,s.sName
29. **FROM**    student s,  (**SELECT** sid,COUNT(cid) **FROM**  sc **WHERE** cid IN ('001','002') **GROUP** **BY** sid **HAVING** COUNT(cid)>=2) t **WHERE** s.sid = t.sid
31. --7、查询学过“叶平”老师所教的所有课的同学的学号、姓名;
32. **select** sid,Sname
33. **from** Student
34. **where** sid in (**select** sid **from** SC ,Course ,Teacher **where** SC.cid=Course.cid and Teacher.tid=Course.tid and Teacher.Tname='叶平' **group** **by** sid **having** count(SC.cid)=(**select** count(cid) **from** Course,Teacher **where** Teacher.tid=Course.tid and Tname='叶平'));
36. --8、查询课程编号“002”的成绩比课程编号“001”课程低的所有同学的学号、姓名;
37. 1>**Select** sid,Sname **from** (**select** Student.sid,Student.Sname,score ,(**select** score **from** SC SC\_2 **where** SC\_2.sid=Student.sid and SC\_2.cid='002') score2
38. **from** Student,SC **where** Student.sid=SC.sid and cid='001') S\_2 **where** score2 <score;
39. 2>**SELECT** s.sid,s.sName **FROM** student s,
40. (**SELECT** sid,score **FROM** sc **WHERE** cid = '001') sc\_1,
41. (**SELECT** sid,score **FROM** sc **WHERE** cid = '002') sc\_2
42. **WHERE** sc\_1.sid = sc\_2.sid AND s.sid = sc\_2.sid AND sc\_2.score < sc\_1.score
44. --9、查询所有课程成绩小于60分的同学的学号、姓名;
45. **select** sid,Sname
46. **from** Student
47. **where** sid not in (**select** Student.sid **from** Student,SC **where** S.sid=SC.sid and score>60);
49. --10、查询没有学全所有课的同学的学号、姓名;
50. 1>
51. **select** Student.sid,Student.Sname
52. **from** Student,SC
53. **where** Student.sid=SC.sid **group** **by** Student.sid,Student.Sname **having** count(cid) <(**select** count(cid) **from** Course);
54. 2>
55. **SELECT** s.sid,s.sname **FROM** student s,
56. (**SELECT** sid,COUNT(cid) **FROM** sc **GROUP** **BY** sid **HAVING** COUNT(cid) < (**SELECT** COUNT(cid) **FROM** course) )t
57. **WHERE** s.sid  = t.sid
58. --11、查询至少有一门课与学号为“1001”的同学所学相同的同学的学号和姓名;
59. **select** sid,Sname **from** Student,SC **where** Student.sid=SC.sid and cid in (**select** cid **from** SC **where** sid='1001');
61. --12、把“SC”表中“叶平”老师教的课的成绩都更改为此课程的平均成绩;
62. **UPDATE** sc,(**SELECT** c.cid,AVG(score) avgs **FROM** sc,course c,teacher t **WHERE** sc.cid = c.cid AND
63. c.tid = t.tid AND t.tName = '叶平' **GROUP** **BY** c.cid)sc\_2 **SET** sc.score = sc\_2.avgs **WHERE** sc.cid = sc\_2.cid
65. --13、查询和“1002”号的同学学习的课程完全相同的其他同学学号和姓名;
66. **select** sid **from** SC **where** cid in (**select** cid **from** SC **where** sid='1002')
67. **group** **by** sid **having** count(\*)=(**select** count(\*) **from** SC **where** sid='1002');
69. --14、删除学习“叶平”老师课的SC表记录;
70. **DELETE**  **FROM** sc  **WHERE** sc.cid IN (**SELECT** sc.cid **FROM** course c ,teacher t **WHERE** sc.cid = c.cid AND c.tid = t.tid AND t.tName = '叶平')
72. --15、按平均成绩从高到低显示所有学生的“数据库”、“企业管理”、“英语”三门的课程成绩，按如下形式显示： 学生ID,,数据库,企业管理,英语,有效课程数,有效平均分
73. **SELECT** sid **as** 学生ID
74. ,(**SELECT** score **FROM** SC **WHERE** SC.sid=t.sid AND cid='004') **AS** 数据库
75. ,(**SELECT** score **FROM** SC **WHERE** SC.sid=t.sid AND cid='001') **AS** 企业管理
76. ,(**SELECT** score **FROM** SC **WHERE** SC.sid=t.sid AND cid='005') **AS** 英语
77. ,COUNT(\*) **AS** 有效课程数, AVG(t.score) **AS** 平均成绩
78. **FROM** SC **AS** t
79. **GROUP** **BY** sid
80. **ORDER** **BY** avg(t.score)
82. --16、查询各科成绩最高和最低的分：以如下形式显示：课程ID，最高分，最低分
83. **select** cid "课程ID",**max**(score) "最高分",**min**(score) "最低分" **from** sc **group** **by** cid
85. --17、按各科平均成绩从低到高和及格率的百分数从高到低排序
86. oracle>
87. **SELECT** t.cid **AS** 课程号,**MAX**(course.Cname)**AS** 课程名,nvl(AVG(score),0) **AS** 平均成绩
88. ,100 \* SUM(CASE **WHEN** nvl(score,0)>=60 **THEN** 1 **ELSE** 0 **END**)/COUNT(\*) **AS** 及格百分数
89. **FROM** SC T,Course
90. **WHERE** t.cid=course.cid
91. **GROUP** **BY** t.cid
92. **ORDER** **BY** 100 \* SUM(CASE **WHEN** nvl(score,0)>=60 **THEN** 1 **ELSE** 0 **END**)/COUNT(\*) **DESC**
93. Mysql>
94. **SELECT** t.cid **AS** 课程号,**MAX**(course.Cname)**AS** 课程名,IFNULL(AVG(score),0) **AS** 平均成绩
95. ,100 \* SUM(CASE **WHEN** IFNULL(score,0)>=60 **THEN** 1 **ELSE** 0 **END**)/COUNT(\*) **AS** 及格百分数
96. **FROM** SC T,Course
97. **WHERE** t.cid=course.cid
98. **GROUP** **BY** t.cid
99. **ORDER** **BY** 100 \* SUM(CASE **WHEN** IFNULL(score,0)>=60 **THEN** 1 **ELSE** 0 **END**)/COUNT(\*) **DESC**
101. --18、查询如下课程平均成绩和及格率的百分数(用"1行"显示): 企业管理（001），马克思（002），OO&UML （003），数据库（004）
102. **SELECT** SUM(CASE **WHEN** cid ='001' **THEN** score **ELSE** 0 **END**)/SUM(CASE cid **WHEN** '001' **THEN** 1 **ELSE** 0 **END**) **AS** 企业管理平均分
103. ,100 \* SUM(CASE **WHEN** cid = '001' AND score >= 60 **THEN** 1 **ELSE** 0 **END**)/SUM(CASE **WHEN** cid = '001' **THEN** 1 **ELSE** 0 **END**) **AS** 企业管理及格百分数
104. ,SUM(CASE **WHEN** cid = '002' **THEN** score **ELSE** 0 **END**)/SUM(CASE cid **WHEN** '002' **THEN** 1 **ELSE** 0 **END**) **AS** 马克思平均分
105. ,100 \* SUM(CASE **WHEN** cid = '002' AND score >= 60 **THEN** 1 **ELSE** 0 **END**)/SUM(CASE **WHEN** cid = '002' **THEN** 1 **ELSE** 0 **END**) **AS** 马克思及格百分数
106. ,SUM(CASE **WHEN** cid = '003' **THEN** score **ELSE** 0 **END**)/SUM(CASE cid **WHEN** '003' **THEN** 1 **ELSE** 0 **END**) **AS** UML平均分
107. ,100 \* SUM(CASE **WHEN** cid = '003' AND score >= 60 **THEN** 1 **ELSE** 0 **END**)/SUM(CASE **WHEN** cid = '003' **THEN** 1 **ELSE** 0 **END**) **AS** UML及格百分数
108. ,SUM(CASE **WHEN** cid = '004' **THEN** score **ELSE** 0 **END**)/SUM(CASE cid **WHEN** '004' **THEN** 1 **ELSE** 0 **END**) **AS** 数据库平均分
109. ,100 \* SUM(CASE **WHEN** cid = '004' AND score >= 60 **THEN** 1 **ELSE** 0 **END**)/SUM(CASE **WHEN** cid = '004' **THEN** 1 **ELSE** 0 **END**) **AS** 数据库及格百分数
110. **FROM** SC
112. --19、查询不同老师所教不同课程平均分从高到低显示 要求显示：教师ID,教师姓名，课程ID,课程名称，平均成绩
113. **SELECT** **MAX**(t.tid) "教师ID",**MAX**(t.tName) "教师姓名",c.cid "课程ID", **MAX**(c.cName) "课程名称" ,AVG(sc.score) "平均成绩"
114. **FROM** sc,course c,teacher t **WHERE** sc.cid = c.cid AND c.tid = t.tid **GROUP** **BY** c.tid,c.cid
115. **ORDER** **BY** AVG(sc.score) **DESC**
117. --20、统计列印各科成绩,各分数段人数:课程ID,课程名称,[100-85],[85-70],[70-60],[ <60]
118. **SELECT** SC.cid **as** 课程ID, Cname **as** 课程名称
119. ,SUM(CASE **WHEN** score BETWEEN 85 AND 100 **THEN** 1 **ELSE** 0 **END**) **AS** [100 - 85]
120. ,SUM(CASE **WHEN** score BETWEEN 70 AND 85 **THEN** 1 **ELSE** 0 **END**) **AS** [85 - 70]
121. ,SUM(CASE **WHEN** score BETWEEN 60 AND 70 **THEN** 1 **ELSE** 0 **END**) **AS** [70 - 60]
122. ,SUM(CASE **WHEN** score < 60 **THEN** 1 **ELSE** 0 **END**) **AS** [60 -]
123. **FROM** SC,Course
124. **where** SC.cid=Course.cid
125. **GROUP** **BY** SC.cid,Cname;
127. --21、查询每门课程被选修的学生数
128. **select** cid,count(sid) **from** sc **group** **by** cid;
130. --22、查询出只选修了一门课程的全部学生的学号和姓名
131. **select** SC.sid,Student.Sname,count(cid) **AS** 选课数
132. **from** SC ,Student
133. **where** SC.sid=Student.sid **group** **by** SC.sid ,Student.Sname **having** count(cid)=1;
135. --23、查询男生、女生人数
136. **Select** count(Ssex) **as** 男生人数 **from** Student **group** **by** Ssex **having** Ssex='男';
137. **Select** count(Ssex) **as** 女生人数 **from** Student **group** **by** Ssex **having** Ssex='女';
139. --24、查询姓“张”的学生名单
140. **SELECT** Sname **FROM** Student **WHERE** Sname like '张%';
141. 30、查询同名同性学生名单，并统计同名人数
142. **SELECT** sName,sSex ,COUNT(\*) **FROM** student **GROUP** **BY** sName,sSex **HAVING** COUNT(\*) > 1
144. --25、1981年出生的学生名单(注：Student表中Sage列的类型是datetime)
145. Mysql>
146. **select** Sname, CONVERT(**char** (11),DATEPART(year,Sage)) **as** age
147. **from** student
148. **where** CONVERT(**char**(11),DATEPART(year,Sage))='1981';
149. Oracle>
150. **select** \* **from** student **where** substr(to\_char(sage,'yyyy-MM-dd'),1,4)= '1981'
152. --26、查询每门课程的平均成绩，结果按平均成绩升序排列，平均成绩相同时，按课程号降序排列
153. **Select** cid,Avg(score) **from** SC **group** **by** cid **order** **by** Avg(score),cid **DESC** ;
155. --27、查询平均成绩大于85的所有学生的学号、姓名和平均成绩
156. **select** Sname,SC.sid ,avg(score)
157. **from** Student,SC
158. **where** Student.sid=SC.sid **group** **by** SC.sid,Sname **having** avg(score)>85;
160. --28、查询课程名称为“数据库”，且分数低于60的学生姓名和分数
161. **Select** Sname,isnull(score,0)
162. **from** Student,SC,Course
163. **where** SC.sid=Student.sid and SC.cid=Course.cid and Course.Cname='数据库'and score <60;
165. --29、查询所有学生的选课情况;
166. **SELECT** SC.sid,SC.cid,Sname,Cname
167. **FROM** SC,Student,Course
168. **where** SC.sid=Student.sid and SC.cid=Course.cid ;
170. --30、查询任何一门课程成绩在70分以上的姓名、课程名称和分数;
171. **SELECT** **distinct** student.sid,student.Sname,SC.cid,SC.score
172. **FROM** student,Sc
173. **WHERE** SC.score>=70 AND SC.sid=student.sid;
175. --31、查询不及格的课程，并按课程号从大到小排列
176. **select** cid **from** sc **where** scor e <60 **order** **by** cid ;
178. --32、查询课程编号为003且课程成绩在80分以上的学生的学号和姓名;
179. **select** SC.sid,Student.Sname **from** SC,Student **where** SC.sid=Student.sid and Score>80 and cid='003';
181. --33、求选了课程的学生人数
182. **select** count(\*) **from** sc;
184. --34、查询选修“叶平”老师所授课程的学生中，成绩最高的学生姓名及其成绩
185. **select** Student.Sname,score
186. **from** Student,SC,Course C,Teacher
187. **where** Student.sid=SC.sid and SC.cid=C.cid and C.tid=Teacher.tid and Teacher.Tname='叶平' and SC.score=(**select** **max**(score)**from** SC **where** cid=C.cid );
189. --35、查询各个课程及相应的选修人数
190. **select** count(\*) **from** sc **group** **by** cid;
192. --36、查询不同课程成绩相同的学生的学号、课程号、学生成绩
193. **select** **distinct** A.sid,B.score **from** SC A ,SC B **where** A.Score=B.Score and A.cid <>B.cid ;
195. --37、查询每门功课成绩最好的前两名
196. **SELECT** \*
197. **FROM** sc t1
198. **WHERE** (
199. **SELECT** COUNT(\*)
200. **FROM** sc t2
201. **WHERE** t1.cid=t2.cid
202. AND t2.score>=t1.score
203. ) <=2 **ORDER** **BY** t1.cid
205. --38、统计每门课程的学生选修人数（超过10人的课程才统计）。要求输出课程号和选修人数，查询结果按人数降序排列，若人数相同，按课程号升序排列
206. **SELECT**  cid "课程号",COUNT(\*) "选修人数" **FROM** sc **GROUP** **BY** cid **HAVING** COUNT(\*) >10  **ORDER** **BY** COUNT(\*) **DESC**,cid
208. --39、检索至少选修两门课程的学生学号
209. **select** sid
210. **from** sc
211. **group** **by** sid
212. **having** count(\*) > = 2
214. --40、查询全部学生都选修的课程的课程号和课程名
215. **SELECT** s.sName,c.cName, COUNT(\*) **FROM** student s,course c, sc **WHERE** s.sid = sc.sid AND sc.cid = c.cid **GROUP** **BY** sc.cid **HAVING** COUNT(\*) = (**SELECT** COUNT(\*) **FROM** student)
217. --41、查询没学过“叶平”老师讲授的任一门课程的学生姓名
218. **SELECT** **DISTINCT** Sname **FROM** Student **WHERE** sid NOT IN (**SELECT** sid **FROM** Course,Teacher,SC **WHERE** Course.tid=Teacher.tid AND SC.cid=course.cid AND Tname='叶平');
220. --42、查询两门以上不及格课程的同学的学号及其平均成绩
221. **select** sid,avg(ifnull(score,0)) **from** SC **where** sid in (**select** sid **from** SC **where** score <60 **group** **by** sid **having** count(\*)>2)**group** **by** sid;
223. --43、检索“004”课程分数小于60，按分数降序排列的同学学号
224. **select** sid **from** SC **where** cid='004'and score <60 **order** **by** score **desc**;
226. --44、删除“1002”同学的“001”课程的成绩
227. **delete** **from** Sc **where** sid='1002' and cid='001';

#### 行转列

1. **CREATE** **TABLE** stu\_score (
2. grade\_id **varchar**(10) **DEFAULT** NULL,
3. subject\_name **varchar**(10) **DEFAULT** NULL,
4. max\_score **int**(10) **DEFAULT** NULL
5. )
6. **insert**  **into** `stu\_score`(`grade\_id`,`subject\_name`,`max\_score`) **values**('1','语文',98);
7. **insert**  **into** `stu\_score`(`grade\_id`,`subject\_name`,`max\_score`) **values**('2','数学',95);
8. **insert**  **into** `stu\_score`(`grade\_id`,`subject\_name`,`max\_score`) **values**('2','政治',87);
9. **insert**  **into** `stu\_score`(`grade\_id`,`subject\_name`,`max\_score`) **values**('5','语文',97);
10. **insert**  **into** `stu\_score`(`grade\_id`,`subject\_name`,`max\_score`) **values**('5','数学',100);
11. **insert**  **into** `stu\_score`(`grade\_id`,`subject\_name`,`max\_score`) **values**('5','政治',92);

转换步骤

1. --1.查询出要求的结果格式
2. **SELECT**
3. CASE grade\_id **WHEN** 1 **THEN** '一年级'
4. **WHEN** 2 **THEN** '二年级'
5. **WHEN** 5 **THEN** '五年级'
6. **END** **AS** '年级',
7. CASE subject\_name **WHEN** '语文' **THEN** max\_score **END** **AS** '语文',
8. CASE subject\_name **WHEN** '数学' **THEN** max\_score **END** **AS** '数学',
9. CASE subject\_name **WHEN** '政治' **THEN** max\_score **END** **AS** '政治'
10. **FROM** stu\_score
11. --2,去除null
12. **SELECT**
13. CASE grade\_id **WHEN** 1 **THEN** '一年级'
14. **WHEN** 2 **THEN** '二年级'
15. **WHEN** 5 **THEN** '五年级'
16. **END** **AS** '年级',
17. IFNULL(CASE subject\_name **WHEN** '语文' **THEN** max\_score **END**,0) **AS** '语文',
18. IFNULL(CASE subject\_name **WHEN** '数学' **THEN** max\_score **END**,0) **AS** '数学',
19. IFNULL(CASE subject\_name **WHEN** '政治' **THEN** max\_score **END**,0) **AS** '政治'
20. **FROM** stu\_score
21. --3.合并结果。
22. **SELECT**
23. CASE grade\_id **WHEN** 1 **THEN** '一年级'
24. **WHEN** 2 **THEN** '二年级'
25. **WHEN** 5 **THEN** '五年级'
26. **END** **AS** '年级',
27. **MAX**(IFNULL(CASE subject\_name **WHEN** '语文' **THEN** max\_score **END**,0)) **AS** '语文',
28. **MAX**(IFNULL(CASE subject\_name **WHEN** '数学' **THEN** max\_score **END**,0)) **AS** '数学',
29. **MAX**(IFNULL(CASE subject\_name **WHEN** '政治' **THEN** max\_score **END**,0)) **AS** '政治'
30. **FROM** stu\_score
31. **GROUP** **BY** grade\_id

## webservice

### 请简要说明WebService的组成部分？以及各个部分使用的协议？

WebService框架核心是基于简单对象访问协议（Simple Object Access Protocol，

SOAP）、Web 服务描述语言（Web Service Description Language，WSDL）以及通用描述、发现和集成（Universal Description Discovery and Integration，UDDI）。web服务中三种发送接受协议SOAP/HTTP GET/HTTP POST

在web服务中，有三种可供选择的发送和接受信息的协议：SOAP,HTTP GET,HTTP

POST,但是SOAP支持的数据类型更为广泛

SOAP=RPC+HTTP+XMLSOAP简单的理解，就是这样的一个开放协议

SOAP=RPC+HTTP+XML：采用HTTP作为底层通讯协议；RPC作为一致性的调用途径，ＸＭＬ作为数据传送的格式，允许服务提供者和服务客户经过防火墙在INTERNET进行通讯交互。

### 什么是WEB SERVICE?SOAP和WSDL是什么？

Web service是一个平台独立的，低耦合的，自包含的、基于可编程的web的应用程

序，可使用开放的XML（标准通用标记语言下的一个子集）标准来描述、发布、发现、协调和配置这些应用程序，用于开发分布式的互操作的应用程序。Web Service技术，能使得运行在不同机器上的不同应用无须借助附加的、专门的第三方软件或硬件， 就可相互交换数据或集成。依据Web Service规范实施的应用之间， 无论它们所使用的语言、 平台或内部协议是什么， 都可以相互交换数据。

SOAP是简单对象访问协议，是一种轻量的、简单的、基于 XML 的结构化数据交换的

一个协议规范，用于WebService。它被设计成在 WEB 上交换结构化的信息。 SOAP 可以和现存的许多因特网协议和格式结合使用，包括超文本传输协议（ HTTP），简单邮件传输协议（SMTP），多用途网际邮件扩充协议（MIME）。它还支持从消息系统到远程过程调用（RPC）等大量的应用程序。SOAP在HTTP协议的基础上时，把编写成XML的REQUEST参数, 放在HTTP BODY上提交给WEB SERVICE服务器(SERVLET，ASP等) 处理完成后，结果也写成XML作为RESPONSE送回用户端，为了使用户端和WEB SERVICE可以相互对应，可以使用WSDL作为这种通信方式的描述文件，利用WSDL工具可以自动生成WS和用户端的框架文件，SOAP具备把复杂对象序列化捆绑到XML里去的能力。

### WebService中的SOAP是什么协议

SOAP是简单对象访问协议，是一种轻量的、简单的、基于 XML 的结构化数据交换的一个协议

规范，用于WebService。它被设计成在 WEB 上交换结构化的信息。 SOAP 可以和现存的许多因特网协议和格式结合使用，包括超文本传输协议（ HTTP），简单邮件传输协议（SMTP），多用途网际邮件扩充协议（MIME）。它还支持从消息系统到远程过程调用（RPC）等大量的应用程序。SOAP在HTTP协议的基础上时，把编写成XML的REQUEST参数, 放在HTTP BODY上提交给WEB SERVICE服务器(SERVLET，ASP等) 处理完成后，结果也写成XML作为RESPONSE送回用户端，为了使用户端和WEB SERVICE可以相互对应，可以使用WSDL作为这种通信方式的描述文件，利用WSDL工具可以自动生成WS和用户端的框架文件，SOAP具备把复杂对象序列化捆绑到XML里去的能力。

## 通信

### 不同语言开发的程序之间如何进行通信？

webService通信技术

### 如果client已经有一个接口来访问所需要的服务，我们已经有现成的类来提供这样的服务，但是服务提供的方法和client访问所需要的方法不匹配，那么如何设计变成来提供client服务？

可以写一个代理类，将client中接口的方法写在代理类中，在代理类中调用服务提供的方法。如果用

的spring的话，通过配置就能解决这个问题。

## 设计模式

### 写出两种Java设计模式中不同的单例模式singleton

1. /\*第一种（懒汉，线程不安全）：所谓懒汉,就是说当这个对象在需要进行获取的时候,再去对该对象进行创建\*/
2. **public** **class** Singleton{
3. **private** **static** Singleton instance;
4. //通过将构造方法私有化达到,不能直接通过new 类名(),创建类的目的!!!
5. **private** Singleton(){}
6. **public** **static** Singleton getInstance(){
7. **if** (instance == **null**){
8. instance = **new** Singleton();
9. }
10. **return** instance;
11. }
12. }
14. /\*第二种(懒汉,线程安全):这里的线程安全看起来十分的美好,但是效率却十分的低,所以一般的情况下不会使用这种方式\*/
15. **public** **class** Singleton {
16. **private** **static** Singleton instance;
17. **private** Singleton(){}
18. //通过在此使用同步方法,进行将其变成线程安全的!!!
19. **public** **static** **synchronized** Singleton getInstance(){
20. **if** (instance == **null**) {
21. instance = **new** Singleton();
22. }
23. **return** instance;
24. }
25. }
27. /\*第三种(饿汉 : 线程安全) : 当该类一旦进行加载(classLoader机制),就需要对其该类中的对象进行创建,同样因此也就使其避免了线程的安全问题\*/
28. **public** **class** Singleton {
29. **private** **static** Singleton instance = **new** Singleton();
30. **private** Singleton(){}
31. **public** **static** Singleton getInstance() {
32. **return** instance;
33. }
34. }
36. /\*第四种(枚举):该方法是Effective Java作者Josh Bloch 提倡的方式,它不仅能避免多线程同步问题,保证只有一个实例,即使使用反射机制也无法多次实例化一个枚举量\*/
37. **public** **enum** Singleton {
38. INSTANCE;
39. **private** Singleton();
40. }

### j2ee常用的设计模式？描述Singleton的作用

0,Java当中总共包含以下23种设计模式!!!

创建型模式，共五种：工厂方法模式、抽象工厂模式、单例模式、建造者模式、原型模式。

结构型模式，共七种：适配器模式、装饰器模式、代理模式、外观模式、桥接模式、组合模式、

享元模式。

行为型模式，共十一种：策略模式、模板方法模式、观察者模式、迭代子模式、责任链模式、命令模式、备忘录模式、状态模式、访问者模式、中介者模式、解释器模式。

1,常用的有 :

0，工厂方法模式、工厂设计模式中包含三种,这里是指普通工厂、就是建立一个工厂类，对实现

了同一接口的一些类进行实例的创建。

1，单例模式、单例对象（Singleton）是一种相对来说更常用的设计模式。在Java应用中，单

例对象能保证在一个JVM中，该对象只有一个实例存在。

2，适配器模式、适配器模式将某个类的接口转换成客户端期望的另一个接口表示，目的是消除

由于接口不匹配所造成的类的兼容性问题。

主要分为三类：类的适配器模式、对象的适配器模式、接口的适配器模式。 3，装饰器模式、顾名思义，装饰模式就是给一个对象增加一些新的功能，而且是动态的，要求装饰对象和被装饰对象实现同一个接口，装饰对象持有被装饰对象的实例

4，代理模式、代理模式就是多一个代理类出来，替原对象进行一些操作，同样也是对类进行动态增强的一种方式

5，策略模式等、策略模式定义了一系列算法，并将每个算法封装起来，使他们可以相互替换，且算法的变化不会影响到使用算法的客户

-----------------------------以上是常用的六种设计模式----------------------------------

Singleton的作用,也就是Singleton的好处,利用这个单例实际模式可以做到什么

1、某些类创建比较频繁，对于一些大型的对象，这是一笔很大的系统开销。(这个时候如果不需要多个对象,就可以使用单例设计模式进行搞定)!!!

2、省去了new操作符，降低了系统内存的使用频率，减轻GC压力。-- 这条与第一条,的作用就是减少系统开销,降低内存使用!!!

3、有些类如交易所的核心交易引擎，控制着交易流程，如果该类可以创建多个的话，系统完全乱了。--有效的对系统进行控制!!!

4、再比如有些软件的配置文件,不管使用该软件当中什么工具,都会对同一个配置文件进行读取和修改,这就是单例设计的思想!!!--使用该思想,可以实现某些业务需求!!!

### 举例你所熟悉的设计模式，单例、工厂除外

装饰模式是在不必改变原类文件和使用继承的情况下，动态地扩展一个对象的功能。它是通过创

建一个包装对象，也就是装饰来包裹真实的对象。装饰模式的特点

（1） 装饰对象和真实对象有相同的接口。这样客户端对象就能以和真实对象相同的方式和装饰

对象交互。

（2） 装饰对象包含一个真实对象的引用。

（3） 装饰对象接受所有来自客户端的请求。它把这些请求转发给真实的对象。

（4） 装饰对象可以在转发这些请求以前或以后增加一些附加功能。这样就确保了在运行时，不用修改给定对象的结构就可以在外部增加附加的功能。在面向对象的设计中，通常是通过继承来实现对给定类的功能扩展。

## Linux

### 查看某个日志文件的最后25行的内容的命令

tail -f 是查看变化了的内容

tail -n 查看最后n行的内容

## 其他

Servlet的理解

Servlet已经出现了很长一段时间，它先于J2EE平台出现。在过去的一段时间内，Servlet曾经得到广泛的应用，如今它在J2EE项目开发中仍然广泛使用。JSP基于Servlet技术构架，可见Servlet功能的强大。

1.什么是Servlet    Servlet(Java 服务器小程序)是用Java编写的服务器程序，是由服务器调用和执行的，按照Servlet自身规范编写的Java类。Servlet可以看成是用Java编写的CGI。但是它的功能和性能比CGI更加强大。    Servlet是使用Java Servlet应用程序设计接口(API)及相关类和方法的Java程序。除了Java Servlet API,Servlet还可以使用用于扩展和添加到API的Java类软件包。

2.Servlet技术的特点    Servlet带给开发人员最大的好处是它可以处理客户端传来的HTTP请求，并返回一个响应。Servlet是一个Java的类，Java语言能够实现的功能，Servlet基本上都能实现（除了图形界面外）。   Servlet有以下优势

a. 可移植  因为Servlet由Java开发并符合规范定义和广泛接收的API，它可以再不提的操作系统平台和不同的应用服务器平台下移植。

b. 功能强大 Servlet可以使用Java API核心的所有功能，这些功能包括Web和URL访问、图像处理、数据压缩、多线程、JDBC和序列化对象等。

c. 安全 有几个不同层次为Servlet的安全提供了保障。首先，它是Java编写的，所以它可以使用Java的安全框架；其次，ServletAPI被实现为类型安全的；另外容器也会对Servlet的安全进行管理。在Servlet安全策略中，可以使用编程的安全也可以使用声明性的安全，声明性的安全由容器进行统一管理

d. 简洁 Servlet代码面向对象，在封装方面具有先天的优势。   e. 集成  Servlet和服务器紧密集成，它们可以密切合作完成特定的任务。

f. 模块化 每一个Servlet可以执行一个特定任务，并且可以讲他们并在一起工作。Servlet之间是可以相互交流的

g. 扩展性和灵活性  Servlet本身的接口设计得非常精简，使得它有很强的扩展性。需要指出的是，Servlet不等于HttpServlet，后者是前者的一个常见扩展。

h. 高效耐久 Servlet一旦载入，它就驻留在内存中，这样加快了响应的速度。在服务器上仅有一个Java虚拟机在运行，它的优势在于，当Servlet被客户端发送的第一个请示激活，以后它将继续运行于后台，等待以后的请求。每个请求将生成一个线程而不是进程。