杭州归谷培训中心

笔面试题库

目 录

一、数据结构、算法、计算机基础

I、单选题

1、（D）	2、（B）	3、（D）
4、（D）	5、（C）	6、（C）
7、（B）	8、（C）	9、（B）
10、（D）	11、（C）	12、（D）
13、（A）	14、（D）	15、（B）
16、（D）	17、（C）	18、（B）

II 问答题

1、答：8bit。

2、答：ls, pwd, mkdir, rm, cp, mv, cd, ps, ftp, telnet, ping, env, more, echo

3、答：栈是一种线形集合，其添加和删除元素的操作应在同一段完成，栈按照后进先出的方式进行处理；堆是栈的一个组成元素。

4、答：顺序为：DJGEBKNIFCA。

5、答：排序的方法有：插入排序（直接插入排序、希尔排序），交换排序（冒泡排序、快速排序），选择排序（直接选择排序、堆排序），归并排序，分配排序（箱排序、基数排序）；

快速排序的伪代码：

使用快速排序方法对 a[0 :n-1]排序,从 a[0 :n-1]中选择一个元素作为 middle，该元素为支点；把余下的元素分割为两段 left 和 right，使得 left 中的元素都小于等于支点，而 right 中的元素都大于等于支点；递归地使用快速排序方法对 left 进行排序；递归地使用快速排序方法对 right 进行排序；所得结果为 left + middle + right。

6、答案：只允许在端点处插入和删除元素。

7.答案：线性存储结构和链表存储结构。

8. 答案：便于插入和删除操作。

9.答案：方便运算的实现。

10.答案：从表中任一结点出发都能访问到整个链表。

11. 答案：随机存取的存储结构和顺序存取的存储结构。

12.答案：有且只有 1

13. 答案：31
14. 答案：5 种形态。
15. 答案：13
16. 答案：cedba
17. 答案：DGEBHFCA
18. 答案：gdbehfca
19.答案：解题方案的准确而完整的描述。

20. 答案：无穷性。

21. 答案：顺序、选择、循环。

22.答案：算法执行过程中所需要的基本运算次数。

23.答案：执行过程中所需要的存储空间。

24. 答案：分析算法的效率以求改进。

25. 答案：数据的逻辑结构在计算机中的表示。

26. 答案：反映数据元素之间逻辑关系的数据结构。

27.答案：有且只有 1。

28. 答案：128

29.答案：16

30. 答案：31

31. 答案：350
说明：完全二叉树总结点数为 N，若 N 为奇数，则叶子结点数为（N+1）/2；若 N 为偶数，则叶子结点数为 N/2。

32.答案：线性结构和非线性结构。

33.答案：gdbehfca

34.答案：串中所含字符的个数。

35. 答案：模式匹配。

36.答案：N-1

37.答案：N

38. 答案：N

39. 答案：冒泡排序

40. 答案：n(n-1)/2

41.答案：冒泡排序

42.答案：堆排序

43.答案：插入类排序

44.答案：选择类排序

45.答案：归并排序

46.答案：直接插入排序

47.答案：连续不连续都可以。

48.答案：一是对数据对象的运算和操作，二是算法的控制结构。

49.答案：时间复杂度和空间复杂度。实现算法所需的存储单元多少和算法的工作量大小分别称为算法的空间复杂度和时间复杂度 。

50.答案：所谓数据处理是指对数据集合中的各元素以各种方式进行运算，包括插入、删除、查找、更改等运算，也包括对数据元素进行分析。

51.答案：数据结构是指相互有关联的数据元素的集合。

52.答案：数据结构分为逻辑结构与存储结构，线性链表属于存储结构。

53.答案：数据结构包括数据的逻辑结构和数据的存储结构。

54. 答案：用前趋和后继关系来描述。

55.答案：有线性结构和非线性结构两大类。

56.答案：顺序、链接、 索引等存储结构。

57.答案：顺序存储是把逻辑上相邻的结点存储在物理位置相邻的存储单元中。

58.答案：入栈、退栈与读栈顶元素。

59.答案：入队运算与退队运算。

60.答案：链式存储和顺序存储 。

61. 答案：逻辑结构中相邻的结点在存储结构中仍相邻。

62.答案：入队运算与退队运算。每进行一次入队运算，队尾指针就进 1。

63.答案：上溢 。

64 答案：下溢。

65.答案：主要研究数据的逻辑结构、对各种数据结构进行的运算，以及数据的存储结构。

66.答案：队列

67. 答案：节省存储空间，降低上溢发生的机率。

III、编程题

1、答：C++中冒泡排序：

```
void swap( int& a, int& b ){
    int c=a;
    a = b;
    b = c;
}
void bubble( int* p, int len ){
    bool bSwapped;
    do {
        bSwapped = false;
        for( int i=1; i<len; i++ ){
            if( p[i-1]>p[i] ){
                swap( p[i-1], p[i] );
                bSwapped = true;
            }
        }
    }while( bSwapped);
}
```

2、答：代码如下：

```
#include <math.h>
bool prime(int n ){
    if(n<=0) {
        exit(0);
```

```
    }
    for( int i=2; i<=n; i++ )
        for( int j=2; j<=sqrt(i); j++) {
            if((n%j==0) && (j!=n)) {
                return false;
            }
            return true;
        }
    }
}
```

采用 C++，因为其运行效率高。

3. 答：代码如下：

```
package test;
public class CountGame {
    private static boolean same(int[] p,int l,int n){
        for(int i=0;i<l;i++){
            if(p[i]==n){
                return true;
            }
        }
        return false;
    }
    public static void play(int playerNum, int step){
        int[] p=new int[playerNum];
        int counter = 1;
        while(true){
            if(counter > playerNum*step){
                break;
            }
            for(int i=1;i<playerNum+1;i++){
                while(true){
                    if(same(p,playerNum,i)==false) {
                        break;
                    }else{
                        i=i+1;
                    }
                }
                if(i > playerNum){break;}
                if(counter%step==0){
                    System.out.print(i + " ");
                    p[counter/step-1]=i;
                }
                counter+=1;
```

```
            }
        }
        System.out.println();
    }
    public static void main(String[] args) {
        play(10, 7);
    }
}
```

4. 答：C++的代码实现如下：

```
#include <iostream>
#include <iomanip>
#include <vector>
using namespace std;
class longint {
private:
    vector<int> iv;
public:
    longint(void) {
        iv.push_back(1);
    }
    longint& multiply(const int &);
    friend ostream& operator<<(ostream &, const longint &);
};
ostream& operator<<(ostream &os, const longint &v) {
    vector<int>::const_reverse_iterator iv_iter = v.iv.rbegin();
    os << *iv_iter++;
    for ( ; iv_iter < v.iv.rend(); ++iv_iter) {
        os << setfill('0') << setw(4) << *iv_iter;
    }
    return os;
}

longint& longint::multiply(const int &rv) {
    vector<int>::iterator iv_iter = iv.begin();
    int overflow = 0, product = 0;
    for ( ; iv_iter < iv.end(); ++iv_iter) {
        product = (*iv_iter) * rv;
        product += overflow;
        overflow = 0;
        if (product > 10000) {
            overflow = product / 10000;
            product -= overflow * 10000;
        }
```

```
        iv_iter = product;
    }
    if (0 != overflow) {
        iv.push_back(overflow);
    }
    return *this;
}
int main(int argc, char **argv) {
    longint result;
    int l = 0;
    if(argc==1){
        cout << "like:
        multiply 1000" << endl;
        exit(0);
    }
    sscanf(argv[1], "%d", &l);
    for (int i = 2; i <= l; ++i) {
        result.multiply(i);
    }
    cout << result << endl;
    return 0;
}
```

5.答：
```
public class num5 {
  public static void main(String[] args) {
    for (int i = 1; i < 9; i++) {
      for (int j = 1; j <= i; j++) {
        System.out.print(j+”*”+i+”=”+i*j+”\t”);
      }
      System.out.println();
    }
  }
}
```

6.答:
```
public class num2{
public static void main(String[] args){
int ywzm,space,num,other;
ywzm=space=num=other=0;
BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
String str = null;
try {
str = br.readLine();
} catch (IOException ex) {
```

```
Logger.getLogger(num2.class.getName()).log(Level.SEVERE, null,
ex);
}
char[] carr=str.toCharArray();
for (int i = 0; i < carr.length; i++) {
if((carr>=' 0' )&&(carr<=' 9' ))
num++;
else if((carr>=' A'&&carr<=' Z')||(carr>=' a'&&carr<=' z'))
ywzm++;
else if(carr==' ')
space++;
else
other++;
}
System.out.println("英文字母个数:"+ywzm+"\t 空格个
数"+space+"\t 数字个数"+num+"\t 其他字符个数:"+other);
}
}
```

7.答：折半查找法也称为二分查找法，它充分利用了元素间的次序关系，采用分治策略，可在最坏的情况下用 O(log n)完成搜索任务。它的基本思想是，将 n 个元素分成个数大致相同的两半，取 a[n/2]与欲查找的 x 作比较，如果 x=a[n/2]则找到 x，算法终止。如果 xa[n/2]，则我们只要在数组 a 的右半部继续搜索 x。二分搜索法的应用极其广泛，而且它的思想易于理解，但是要写一个正确的二分搜索算法也不是一件简单的事。第一个二分搜索算法早在 1946 年就出现了，但是第一个完全正确的二分搜索算法直到1962年才出现。Bentley在他的著作《Writing Correct Programs》中写道，90%的计算机专家不能在 2 小时内写出完全正确的二分搜索算法。问题的关键在于准确地制定各次查找范围的边界以及终止条件的确定，正确地归纳奇偶数的各种情况，其实整理后可以发现它的具体算法是很直观的，我们可用 C++描述如下：

```
template
int BinarySearch(Type a[],const Type& x,int n)
{
int left=0;
int right=n-1;
while(left<=right){
int middle=(left+right)/2;
if (x==a[middle]) return middle;
if (x>a[middle]) left=middle+1;
else right=middle-1;
}
```

```
return -1;
}
```

模板函数 BinarySearch 在 a[0]<=a[1]<=...<=a[n-1]共 n 个升序排列的元素中搜索 x,找到 x 时返回其在数组中的位置，否则返回-1。容易看出，每执行一次 while 循环，待搜索数组的大小减少一半,因此整个算法在最坏情况下的时间复杂度为 O(log n)。在数据量很大的时候，它的线性查找在时间复杂度上的优劣一目了然。

java 中的实例如下：

```
public class TestSearch {
public static void main(String[] args) {
int a[] = { 1, 3, 6, 8, 9, 10, 12, 18, 20, 34 };
int i = 12;
//System.out.println(search(a, i));
System.out.println(binarySearch(a, i));
}

public static int search(int[] a, int num) {
for(int i=0; i
if(a == num) return i;
}
return -1;
}

public static int binarySearch(int[]a, int num) {
if (a.length==0) return -1;

int startPos = 0;
int endPos = a.length-1;
int m = (startPos + endPos) / 2;
while(startPos <= endPos){
if(num == a
) return m;
if(num > a
) {
startPos = m + 1;
}
if(num < a
) {
endPos = m -1;
}
m = (startPos + endPos) / 2;
}
return -1;
```

```
}
}
```

8.答 1 List reverse(List l) {
2 if(!l) return l;
3 list cur = l.next;
4 list pre = l;
5 list tmp;
6 pre.next = null;
7 while (cur) {
8 tmp = cur;
9 cur = cur.next;
10 tmp.next = pre
11 pre = tmp;
12 }
13 return tmp;
14 }

9. 答：1 List resverse(list l) {
2 if(!l || !l.next) return l;
3
4 List n = reverse(l.next);
5 l.next.next = l;
6 l.next=null;
7 }
8 return n;
9 }

10.答： 1 void BST(Tree t) {
2 Queue q = new Queue();
3 q.enque(t);
4 Tree t = q.deque();
5 while(t) {
6 System.out.println(t.value);
7 q.enque(t.left);
8 q.enque(t.right);
9 t = q.deque();
10 }
11 }

————————————-

```
1class Node {
2 Tree t;
3 Node next;
4 }
5class Queue {
6 Node head;
7 Node tail;
8 public void enque(Tree t){
9 Node n = new Node();
10 n.t = t;
11 if(!tail){
12 tail = head = n;
13 } else {
14 tail.next = n;
15 tail = n;
16 }
17 }
18 public Tree deque() {
19 if (!head) {
20 return null;
21 } else {
22 Node n = head;
23 head = head.next;
24 return n.t;
25 }
26}
```

11. 答：

```
1char[] p;
2void perm(char s[], int i, int n){
3 int j;
4 char temp;
5 for(j=0;j<N;++J){
6 if(j!=0 && s[j]==s[j-1]);
7 elseif(s[j]!='@'){
8 p[i]=s[j];
9 s[j]='@';
10 if(i==n-1){
11 p[n]='\0';
12 printf("%s", p);
13 }else{
14 perm(s,i+1,n);
15 }
```

```
16 s[j]=p[i];
17 }
18 }
19}
```

```
1void main() {
2 char s[N];
3 sort(s);
4 perm(s,0,strlen(s));
5}
```

12.答：

```
1 long atol(char *str){
2 char *p = str;
3 long l=1;m=0;
4 if (*p=='-') {
5 l=-1;
6 ++p;
7 }
8 while(isDigit(*p)){
9 m = m*10 + p;
10 ++p;
11 }
12 if(!p) return m*l;
13 else return error;
14}
```

13.答：

```
1 int isLoop(List l) {
2 if ( ! l) return - 1 ;
3 List s = l.next;
4 while (s && s != l) {
5 s = s.next;
6 }
7 if ( ! s) return - 1 ;
8 else reutrn 1 ;
9 }
```

```
1int isLoop(List l){
2 if(!l) return 0;
3 p=l.next;
4 wihle(p!=l&&p!=null) {
5 l.next=l;
6 l=p;p=p.next;
7 }
```

```
8 if(p=l) return 1;
9 return 0;
10}
```

14.答：
```
1 void reverse( char * str) {
2 char tmp;
3 int len;
4 len = strlen(str);
5 for ( int i = 0 ;i < len / 2 ; ++ i) {
6 tmp = char [i];
7 str[i] = str[len - i - 1 ];
8 str[len - i - 1 ] = tmp;
9 }
10 }
```

15.答：
```
1int strstr(char[] str, char[] par){
2 int i=0;
3 int j=0;
4 while(str[i] && str[j]){
5 if(str[i]==par[j]){
6 ++i;
7 ++j;
8 }else{
9 i=i-j+1;
10 j=0;
11 }
12 }
13 if(!str[j]) return i-strlen(par);
14 else return -1;
15}
```

16.答：
```
1int strcmp(char* str1, char* str2){
2 while(*str1 && *str2 && *str1==*str2){
3 ++str1;
4 ++str2;
5 }
6 return *str1-*str2;
7}
```

17.答：
```
1 int f( int x) {
2 int n = 0 ;
```

```
3 while (x) {
4 ++ n;
5 x &= x - 1 ;
6 }
7 return n;
8 }
```

18.答：
```
1void tower(n,x,y,z){
2 if(n==1) move(x,z);
3 else {
4 tower(n-1, x,z,y);
5 move(x,z);
6 tower(n-1, y,x,z);
7 }
8}
```

19.答：三柱汉诺塔最小步数。

```
1 int f3(n) {
2 if (f3[n]) return f3[n];
3 else {
4 if (n == 1 ) {
5 f3[n] = 1 ;
6 return 1 ;
7 }
8 f3[n] = 2 * f3(n - 1 ) + 1 ;
9 return f3[n];
10 }
11 }
```

四柱汉诺塔最小步数。

```
1int f4(n){
2 if(f4[n]==0){
3 if(n==1) {
4 f4[1]==1;
5 return 1;
6 }
7 min=2*f4(1)+f3(n-1);
8 for(int i=2;i<N;++I){
9 u=2*f4(i)+f3(n-i);
10 if(u 11 }
12 f4[n]=min;
13 return min;
14 } else return f4[n];
15}
```

20.答：1void delete(List m, List n) {
2 if(!m || !n) return;
3 List pre = new List();
4 pre.next=m;
5 List a=m, b=n,head=pre;
6 while(a && b){
7 if(a.value < b.value) {
8 a=a.next;
9 pre=pre.next;
10 }else if(a.value > b.value){
11 b=b.next;
12 }else{
13 a=a.next;
14 pre.next=a;
15 }
16 }
17 m=head.next;
18}

21.答：1int hasDuplicate(int[] a, int n){
2 for(int i=0;i<N;++I){
3 while(a[i]!=i && a[i]!=-1){
4 if(a[a[i]]==-1) return 1;
5 a[i]=a[a[i]];
6 a[a[i]]=-1;
7 }
8 if(a[i]==i) {a[i]=-1;}
9 }
10 return 0;
11}

22.答：1int isB(Tree t){
2 if(!t) return 0;
3 int left=isB(t.left);
4 int right=isB(t.right);
5 if(left >=0 && right >=0 && left - right <= 1 || left -right >=-1)
6 return (left
7 else return -1;
8}
9

23.答：1int depth(Tree t){
2 if(!t) return 0;

```
3 else {
4 int a=depth(t.right);
5 int b=depth(t.left);
6 return (a>b)?(a+1):(b+1);
7 }
8}
```

24.答：
```
1 List merge(List a, List d) {
2 List a1 = reverse(d);
3 List p = q = new List();
4 while ( a && a1 ) {
5 if (a.value < a1.value) {
6 p.next = a;
7 a = a.next;
8 } else {
9 p.next = a1;
10 a1 = a1.next;
11 }
12 p = p.next;
13 }
14 if (a) p.next = a;
15 elseif(a1) p.next = a1;
16 return q.next;
17 }
```

25.答：
```
1char* ltoa(long l){
2 char[N] str;
3 int i=1,n=1;
4 while(!(l/i<10)){i*=10;++n}
5 char* str=(char*)malloc(n*sizeof(char));
6 int j=0;
7 while(l){
8 str[j++]=l/i;
9 l=l%i;
10 i/=10;
11 }
12 return str;
13}
```

26.答：
```
1 if (x == 0) y = a;
2 else y = b;
```

```
1 j[] = {a,b};
2 y=j[x];
```

27.答：1void del(List head, List node){
2 List pre=new List();
3 pre.next = head;
4 List cur = head;
5 while(cur && cur!=node){
6 cur=cur.next;
7 pre=pre.next;
8 }
9 if(!cur) return;
10 List post = cur.next;
11 pre.next=cur.next;
12 post.last=cur.last;
13 return;
14}

28.答：1 void outputUnique(char [] str, int n) {
2 if (n <= 0) return ;
3 elseif(n == 1) putchar(str[0]);
4 else {
5 int i = 0 ,j = 1 ;
6 putchar(str[0]);
7 while (j < n) {
8 if (str[j] !== str[i]) {
9 putchar(str[j]);
10 i = j;
11 }
12 ++ j;
13 }
14 }
15 }

29.答：判断一个链表是否存在环，例如下面这个链表就存在一个环：
例如：N1->N2->N3->N4->N5->N2 就是一个有环的链表，环的开始结点是 N5 这里有一个比较简单的解法。
设置两个指针 p1，p2。每次循环 p1 向前走一步，p2 向前走两步。直到 p2 碰到 NULL 指针或者两个指针相等结束循环。如果两个指针相等则说明存在环。
struct link
{
int data;
link* next;
};

bool IsLoop(link* head)
{

```
link* p1=head, *p2 = head;
if (head ==NULL || head->next ==NULL)
{
return false;
}
do{
p1= p1->next;
p2 = p2->next->next;
} while(p2 && p2->next && p1!=p2);
if(p1 == p2)
return true;
else
return false;
}
```

30.答：单向链表的反转是一个经常被问到的一个面试题，也是一个非常基础的问题。
例如：一个链表是这样的： 1->2->3->4->5 通过反转后成为 5->4->3->2->1。最容易想到的方法遍历一遍链表，利用一个辅助指针，存储遍历过程中当前指针指向的下一个元素，然后将当前节点元素的指针反转后，利用已经存储的指针往后面继续遍历。源代码如下：

```
struct linka {
int data;
linka* next;
};

void reverse(linka*& head)
{
if(head ==NULL)
return;
linka*pre, *cur, *ne;
pre=head;
cur=head->next;
while(cur)
{
ne = cur->next;
cur->next = pre;
pre = cur;
cur = ne;
}
head->next = NULL;
head = pre;
}
```

还有一种利用递归的方法。这种方法的基本思想是在反转当前节点之前

先调用递归函数反转后续节点。源代码如下。不过这个方法有一个缺点，就是在反转后的最后一个结点会形成一个环，所以必须将函数的返回的节点的 next 域置为 NULL。因为要改变 head 指针，所以我用了引用。算法的源代码如下：

```
linka* reverse(linka* p,linka*& head)
{
if(p == NULL || p->next == NULL)
{
head=p;
return p;
}
else
{
linka* tmp = reverse(p->next,head);
tmp->next = p;
return p;
}
}
```

31.答：给定两个排好序的数组，怎样高效得判断这两个数组中存在相同的数字？

这个问题首先想到的是一个 O(nlogn)的算法。就是任意挑选一个数组，遍历这个数组的所有元素，遍历过程中，在另一个数组中对第一个数组中的每个元素进行 binary search。用 C++实现代码如下：

```
bool findcommon(int a[],int size1,int b[],int size2)
{
int i;
for(i=0;i<SIZE1;I++)
{
int start=0,end=size2-1,mid;
while(start<=end)
{
mid=(start+end)/2;
if(a[i]==b[mid])
return true;
else if (a[i]<B[MID])
end=mid-1;
else
start=mid+1;
}
}
return false;
}
```

后来发现有一个 O(n)算法。因为两个数组都是排好序的。所以只要一

次遍历就行了。首先设两个下标，分别初始化为两个数组的起始地址，依次向前推进。推进的规则是比较两个数组中的数字，小的那个数组的下标向前推进一步，直到任何一个数组的下标到达数组末尾时，如果这时还没碰到相同的数字，说明数组中没有相同的数字。

```
bool findcommon2(int a[], int size1, int b[], int size2)
{
int i=0,j=0;
while(i<size2)
{
if(a[i]==b[j])
return true;
if(a[i]>b[j])
j++;
if(a[i]<B[J])
i++;
}
return false;
}
```

32.答：给定一整数序列 A1， A2，... An （可能有负数），求 A1~An 的一个子序列 Ai~Aj，使得 Ai 到 Aj 的和最大。

例如：整数序列-2, 11, -4, 13, -5, 2, -5, -3, 12, -9 的最大子序列的和为 21。

对于这个问题，最简单也是最容易想到的那就是穷举所有子序列的方法。利用三重循环，依次求出所有子序列的和然后取最大的那个。当然算法复杂度会达到 O(n^3)。显然这种方法不是最优的，下面给出一个算法复杂度为 O(n)的线性算法实现，算法的来源于 Programming Pearls 一书。 在给出线性算法之前，先来看一个对穷举算法进行优化的算法，它的算法复杂度为 O(n^2)。其实这个算法只是对对穷举算法稍微做了一些修改：其实子序列的和我们并不需要每次都重新计算一遍。假设 Sum(i, j)是 A[i]... A[j]的和，那么 Sum(i, j+1) = Sum(i, j) + A[j+1]。利用这一个递推，我们就可以得到下面这个算法：

```
int max_sub(int a[],int size)
{
int i,j,v,max=a[0];
for(i=0;i<SIZE;I++)
{
v=0;
for(j=i;j<SIZE;J++)
{
v=v+a[j];//Sum(i, j+1) = Sum(i, j) + A[j+1]
if(v>max)
```

```
max=v;
}
}
return max;
}
```
那怎样才能达到线性复杂度呢？这里运用动态规划的思想。先看一下源代码实现：
```
int max_sub2(int a[], int size)
{
int i,max=0,temp_sum=0;
for(i=0;i<SIZE;I++)
{
temp_sum+=a[i];
if(temp_sum>max)
max=temp_sum;
else if(temp_sum<0)
temp_sum=0;
}
return max;
}
```

33.答：并不是简单的字符串反转，而是按给定字符串里的单词将字符串倒转过来，就是说字符串里面的单词还是保持原来的顺序，这里的每个单词用空格分开。
例如：Here is www.fishksy.com.cn
经过反转后变为：
www.fishksy.com.cn is Here
如果只是简单的将所有字符串翻转的话，可以遍历字符串，将第一个字符和最后一个交换，第二个和倒数第二个交换，依次循环。其实按照单词反转的话可以在第一遍遍历的基础上，再遍历一遍字符串，对每一个单词再反转一次。这样每个单词又恢复了原来的顺序。
```
char* reverse_word(const char* str)
{
int len = strlen(str);
char* restr = new char[len+1];
strcpy(restr,str);
int i,j;
for(i=0,j=len-1;i<J;I++,J--)
{
char temp=restr[i];
restr[i]=restr[j];
restr[j]=temp;
```

```
}
int k=0;
while(k<LEN)
{
i=j=k;
while(restr[j]!=' ' && restr[j]!='' )
j++;
k=j+1;
j--;
for(;i<J;I++,J--)
{
char temp=restr[i];
restr[i]=restr[j];
restr[j]=temp;
}
}
return restr;
}
```

如果考虑空间和时间的优化的话，当然可以将上面代码里两个字符串交换部分改为异或实现。

例如将

```
char temp=restr[i];
restr[i]=restr[j];
restr[j]=temp;
```

改为

```
restr[i]^=restr[j];
restr[j]^=restr[i];
restr[i]^=restr[j];
```

34.答：一个动态长度可变的数字序列，以数字 0 为结束标志，要求将重复的数字用一个数字代替。

例如：将数组 1,1,1,2,2,2,2,2,7,7,1,5,5,5,0 转变成 1,2,7,1,5,0

问题比较简单，要注意的是这个数组是动态的。所以避免麻烦我还是用了 STL 的 vector。

```
#include
#include
using namespace std;
//remove the duplicated numbers in an intger array, the array
was end with 0;
//e.g. 1,1,1,2,2,5,4,4,4,4,1,0 —>1,2,5,4,1,0
void static remove_duplicated(int a[], vector& _st)
{
_st.push_back(a[0]);
for(int i=1;_st[_st.size()-1]!=0;i++)
```

```
{
if(a[i-1]!=a[i])
_st.push_back(a[i]);
}
}
```
当然如果可以改变原来的数组的话，可以不用 STL，仅需要指针操作就可以了。下面这个程序将修改原来数组的内容。
```
void static remove_duplicated2(int a[])
{
if(a[0]==0 || a==NULL)
return;
int insert=1,current=1;
while(a[current]!=0)
{
if(a[current]!=a[current-1])
{
a[insert]=a[current];
insert++;
current++;
}
else
current++;
}
a[insert]=0;
}
```

35.答：判断一个二叉排序树是否是平衡二叉树 解决方案：根据平衡二叉树的定义，如果任意节点的左右子树的深度相差不超过 1，那这棵树就是平衡二叉树。
首先编写一个计算二叉树深度的函数，利用递归实现。
```
template
static int Depth(BSTreeNode* pbs)
{
if (pbs==NULL)
return 0;
else
{
int ld = Depth(pbs->left);
int rd = Depth(pbs->right);
return 1 + (ld >rd ? ld : rd);
}
}
```
下面是利用递归判断左右子树的深度是否相差 1 来判断是否是平衡二叉树的函数：

```
template
static bool isBalance(BSTreeNode* pbs)
{
if (pbs==NULL)
return true;
int dis = Depth(pbs->left)  -  Depth(pbs->right);
if (dis>1 || dis<-1 )
return false;
else
return isBalance(pbs->left) && isBalance(pbs->right);
}
```

二、SQL

I、单选题

1.C	2.C	3.B	4.D	5.D	6.B
7.A	8.D	9.B	10.A	11.B	12.A
13.A	14.A	15.D	16.C	17.C	18.D
19.C	20.D	21.D	22.B	23.C	24.D
25.C	26.C	27.B	28.A	29.A	30.B
31.C	32.D	33.D	34.C	35. A	36. D
37. F	38.A	39.B	40. A	41. A	42.B
43.A	44.D	45. A	46. B	47. A	48. C_
49. C	50. B	51. A	52. D	53. B	54. D
55. B	56. C	57. A	58. D	59. C	60. B
61. A	62. A	63. B	64. A	65. A	66. B
67. A	68.C	69. C	70. B	71. B	72. A
73. B	74. B	75. C	76. C	77. _B	78. D
79. D	80. A	81. A	82. B	83. C	84. B
85. B	86. D	87. D	88. C	89. B	90. B
91. C	92. C	93. D	94. D	95. C	96. C
97. A	98. D	99. C	100. B	101. C	102. A
103. B	104. B	105. D	106 A	107.C	108. C
109. D	110. C	111. B	112. D	113. A	114. C
115. B	116. B	117. A	118. A	119. A	120. C
121. C	122. B	123. A	124. D	125. C	126. C
127. A	128. A	129. B	130. B	131. D	132. C
133. A	134. A	135. D			

II、不定项选题

1.C	2. B	3. C	4. A	5. B	6. C
7.A	8. C	9.B	10. B	11.BC	12. D
13. D	14. C	15. A	16. C	17.B	18. B
19. B	20. B	21. A	22. A	23. A	24. B
25. B	26. B	27. C	28. C	29. A	30. C
31. B	32. A	33. C	34. B、C	35. ABCD	36. C
37. B	38. D	39. C	40. C	41. ABCD	42. AB
43. B	44. A	45. A	46. B	47. ABCD	48. BD _
49. AD	50. B	51. CD	52. A	53. A	54. B
55. BCD	56. AB	57. C	58. ABC	59. ABC	60. B
61. BD	62. B	63. B	64. ABC	65. B、E	66. A
67. A、E	68 B	69. B	70. C	71. C	72. C
73. C	74. B	75. B	76. C	77. _ C	78. C
79. c	80. a	81. bcd	82. b	83. b	84. c
85. c	86. c	87. a	88. b	89. C	90. B
91. B	92. AC	93. C	94. D	95. A	96. B
97. BC	98. BC				

III.填空题

1. 答案: 1)ID2 in 2) 3) =

select * from T1 A where _ Exists_____ (select * from T2 B where B.ID = A.ID2)

select * from T1 A where A.ID __in____ (select ID from T2 B)

select A.* from T1 A, T2 B where A.ID2 __＝____ B.ID

2. 答案:22k

3. 答案: HAVING COUNT(*)>1

4. 答案: 时间

5. 答案:模式_

6. 答案:黑盒

7. 答案：一对多_

8. 答案：投影

9. 答案：15、into cursor_

10. 答案：_列数

11. 答案：应用程序生成器

12. 答案：数据库管理系统、DBMS

13 答案：联接类型

14. 答案：CREATE TABLE 借阅
(借书证号 C(4),
总编号 C(6),
借书日期 D(8))
或 CREA TABL 借阅
(借书证号 C(4),
总编号 C(6),
借书日期 D(8))

15. 答案：单价<=20
或 (出版单位=”清华大学出版社” OR 出版单位=”电子工业出版社”)
或 (出版单位=”电子工业出版社” OR 出版单位=”清华大学出版社”)
或 (出版单位=’清华大学出版社’ OR 出版单位=’电子工业出版社’) 与
(出版单位=”清华大学出版社” OR 出版单位=”电子工业出版社”)
或 (出版单位=’清华大学出版社‘) 49. COUNT(DISTINCT 总编号)
或 COUN(DISTINCT 总编号) 或 COUNT(DIST 总编号) 或
COUN(DIST 总编号)

16. 答案：概念

17. 答案：n(n-1)*2

18. 答案：查询

19. 答案：文件位置

20. 答案：字符

21. 答案：逻辑表达式

22. 答案：数据源

23. 答案：内部联接

24. 答案：单价 BETWEEN 15 AND 25
或 单价 BETW 15 AND 25
或 单价 BETWE 15 AND 25
或 单价>=15 and 单价<=25
或 单价>=15 and 单价=<25
或 单价=>15 and 单价<=25
或 单价=>15 and 单价=<25 与 分类号 ASC
或 分类号

25 答案：SELECT * FROM R UNION SELECT * FROM T
或 SELE * FROM R UNIO SELE * FROM T
或 SELECT * FROM R UNIO SELECT * FROM T
或 SELE * FROM R UNION SELE * FROM T

26 答案：INSERT INTO R(NO,NAME,CLASS) VALUES(30,”郑和”,”95031″)
或 INSE INTO R(NO,NAME,CLASS) VALUES(30,”郑和”,”95031″)

27. 答案：DELETE FROM R WHERE NO=20
或 DELE FROM R WHERE NO=20
或 DELE FROM R WHER NO=20
或 DELETE FROM R WHER NO=20

IV、简答题

1. 答案：使用 distinct 关键字

2.：快照 Snapshot 是一个文件系统在特定时间里的镜像，对于在线实时数据备份非常有用。快照对于拥有不能停止的应用或具有常打开文件的文件系统的备份非常重要。对于只能提供一个非常短的备份时间而言，快照能保证系统的完整性。

3. 答案：
存储过程是一组 Transact-SQL 语句，在一次编译后可以执行多次。因为不必重新编译 Transact-SQL 语句，所以执行存储过程可以提高性能。
触发器是一种特殊类型的存储过程，不由用户直接调用。创建触发器时会对其进行定义，以便在对特定表或列作特定类型的数据修改时执行。

4. 答案：支持动态行级锁定
SQL Server 2000 动态地将查询所引用的每一个表的锁定粒度调整到合适的级别。当查询所引用的少数几行分散在一个大型表中时，优化数据并行访问的最佳办法是使用粒度锁，如行锁。但是，如果查询引用的是一个表中的大多数行或所有行，优化数据并行访问的最佳办法可以是锁定整个表，以尽量减少锁定开销并尽快完成查询。
SQL Serve 2000 通过为每个查询中的每个表选择适当的锁定级别，在总体上优化了数据并发访问。对于一个查询，如果只引用一个大型表中的几行，则数据库引擎可以使用行级锁定；如果引用一个大型表的几页中的多行，则使用页级锁定；如果引用一个小型表中的所有行，则使用表级锁定。

5. 答案：每个数据库都有事务日志，用以记录所有事务和每个事务对数据库所做的修改。

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

7. 答案：事务是作为一个逻辑单元执行的一系列操作，一个逻辑工作单元必须有四个属性，称为 ACID（原子性、一致性、隔离性和持久性）属性，只有这样才能成为一个事务：
(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状态。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（如 B 树索引或双向链表）都必须是正确的。
(3) 隔离性
由并发事务所作的修改必须与任何其它并发事务所作的修改隔离。事务查看数据时数据所处的状态，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，事务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。

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

9. 答案：
事前触发器运行于触发事件发生之前，而事后触发器运行于触发事件发生之后。通常事前触发器可以获取事件之前和新的字段值。
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。

10. 答案：
解决手段一：SQL Server 自动检测和消除死锁
解决手段二：设置死锁优先级
解决手段三：设置锁定超时
解决手段四：使用更新锁避免死锁

11. 答案：在给定的系统硬件和系统软件条件下，提高数据库系统的运行效率的办法是：
(1) 在数据库物理设计时，降低范式，增加冗余，少用触发器，多用存储过程。
(2) 当计算非常复杂、而且记录条数非常巨大时(例如一千万条)，复杂计算要先在数据库外面，以文件系统方式用 C++语言计算处理完成之后，最后才入库追加到表中去。这是电信计费系统设计的经验。
(3) 发现某个表的记录太多，例如超过一千万条，则要对该表进行水平分割。水平分割的做法是，以该表主键 PK 的某个值为界线，将该表的记录水平分割为两个表。若发现某个表的字段太多，例如超过八十个，则垂直分割该表，将原来的一个表分解为两个表。
(4) 对数据库管理系统 DBMS 进行系统优化，即优化各种系统参数，如缓冲区个数。
(5) 在使用面向数据的 SQL 语言进行程序设计时，尽量采取优化算法。总之，要提高数据库的运行效率，必须从数据库系统级优化、数据库设计级优化、程序实现级优化，这三个层次上同时下功夫。

12. 答案：通俗地理解三个范式，对于数据库设计大有好处。在数据库设计中，为了更好地应用三个范式，就必须通俗地理解三个范式(通俗地理解是够用的理解，并不是最科学最准确的理解)：
第一范式：1NF 是对属性的原子性约束，要求属性具有原子性，不可再分解； 第二范式：2NF 是对记录的惟一性约束，要求记录有惟一标识，即实体的惟
一性；
第三范式：3NF 是对字段冗余性的约束，即任何字段不能由其他字段派生出来，它要求字段没有冗余。没有冗余的数据库设计可以做到。但是，没有冗余的数据库未必是最好的数据库，有时为了提高运行效率，就必须降低范式标准，适当保留冗余数据。具体做法是：在概念数据模型设计时遵守第三范式，降低范式标准的工作放到物理数据模型设计时考虑。降低范式就是增加字段，允许冗余。

13. 优点：
1. 更快的执行速度：存储过程只在创造时进行编译，以后每次执行存储过程都不需再重新编译，而一般 SQL 语句每执行一次就编译一次,所以使用存储过程可提高数据库执行速度；
2. 与事务的结合，提供更好的解决方案：当对数据库进行复杂操作时(如对多个表进行 Update、Insert、Query 和 Delete 时），可将此复杂操作用存储过程封装起来与数据库提供的事务处理结合一起使用；
3. 支持代码重用:存储过程可以重复使用,可减少数据库开发人员的工作量;4. 安全性高：可设定只有某此用户才具有对指定存储过程的使用权。

缺点：
1. 如果更改范围大到需要对输入存储过程的参数进行更改，或者要更改由其返回

的数据，则您仍需要更新程序集中的代码以添加参数、更新 GetValue() 调用，等等，这时候估计比较繁琐了。
2. 可移植性差由于存储过程将应用程序绑定到 SQL Server，因此使用存储过程封装业务逻辑将限制应用程序的可移植性。如果应用程序的可移植性在您的环境中非常重要，则将业务逻辑封装在不特定于 RDBMS 的中间层中可能是一个更佳的选择。

14 答案：
相同点:它们都属于实体完整性约束。
不同点:
(1) 唯一性约束所在的列允许空值，但是主键约束所在的列不允许空值。
(2) 可以把唯一性约束放在一个或者多个列上，这些列或列的组合必须有唯一的。但是，唯一性约束所在的列并不是表的主键列。
(3) 唯一性约束强制在指定的列上创建一个唯一性索引。在默认情况下，创建唯一性的非聚簇索引，但是，也可以指定所创建的索引是聚簇索引。
(4) 建立主键的目的是让外键来引用。
(5) 一个表最多只有一个主键，但可以有很多唯一键。

15. 作用：加快查询速度。
原则：
(1) 如果某属性或属性组经常出现在查询条件中，考虑为该属性或属性组建立索引；
(2) 如果某个属性常作为最大值和最小值等聚集函数的参数，考虑为该属性建立索引；
(3) 如果某属性经常出现在连接操作的连接条件中，考虑为该属性或属性组建立索引；

16. 数据库设计分为五个阶段：
需求分析：主要是准确收集用户信息需求和处理需求，并对收集的结果进行整理和分析，形成需求说明。
概念结构设计：对用户需求进行综合、归纳、抽象，形成一个与与具体的 DBMS 无关概念模型（一般为 ER 模型）。
逻辑结构设计：将概念结构设计的概念模型转化为某个特定的 DBMS 所支持的数据模型，建立数据库逻辑模式，并对其进行优化，同时为各种用户和应用设计外模式。
物理结构设计：为设计好的逻辑模型选择物理结构，包括存储结构和存取方法等，建立数据库物理模式。
实施和维护：实施就是使用 DLL 语言建立数据库模式，将实际数据载入数据库，建立真正的数据库。维护阶段是对运行中的数据库进行评价、调整和修改。

17. 答案：一般我们所说的内存泄漏指的是堆内存的泄漏。堆内存是程序从堆中为其分配的，大小任意的，使用完后要显示释放内存。当应用程序用关键字 new 等创建对象时，就从堆中为它分配一块内存，使用完后程序调用 free 或者 delete 释放该内存，否则就说该内存就不能被使用，我们就说该内存被泄漏了。

18. 答案：基本表是本身独立存在的表，在 SQL 中一个关系就对应一个表。
视图是从一个或几个基本表导出的表。视图本身不独立存储在数据库中，是一个虚表

19. (1) 视图能够简化用户的操作
(2) 视图使用户能以多种角度看待同一数据；
(3) 视图为数据库提供了一定程度的逻辑独立性；
(4) 视图能够对机密数据提供安全保护。

20. 答案：不是。
视图是不实际存储数据的虚表，因此对视图的更新，最终要转换为对基本表的更新。因为有些视图的更新不能惟一有意义地转换成对相应基本表的更新，所以，并不是所有的视图都是可更新的。

21.0 答案：基本表的行列子集视图一般是可更新的。若视图的属性来自集函数、表达式，则该视图肯定是不可以更新的。

22. 答案：尽可能用约束（包括 CHECK、主键、唯一键、外键、非空字段）实现，这种方式的效率最好；其次用触发器，这种方式可以保证无论何种业务系统访问数据库都能维持数据库的完整性、一致性；最后再考虑用自写业务逻辑实现，但这种方式效率最低、编程最复杂，当为下下之策。

23. (1) 视图以及视图中引用的所有表都必须在同一数据库中，并具有同一个所有者
(2) 索引视图无需包含要供优化器使用的查询中引用的所有表。
(3) 必须先为视图创建唯一群集索引，然后才可以创建其它索引。
(4) 创建基表、视图和索引以及修改基表和视图中的数据时，必须正确设置某些 SET 选项（在本文档的后文中讨论）。另外，如果这些 SET 选项正确，查询优化器将不考虑索引视图。
(5) 视图必须使用架构绑定创建，视图中引用的任何用户定义的函数必须使用 SCHEMABINDING 选项创建。
(6) 另外，还要求有一定的磁盘空间来存放由索引视图定义的数据。

24. 答案：只要使用特定的输入值集并且数据库具有相同的状态，不管何时调用，始终都能范围相同结果的函数叫确定性函数。几十访问的数据库的状态不变，每次书用特定的输入值都可能范围不同结果的函数叫非确定性函数。

25. 答案：Shared pool(共享池)，DataBase Buffer Cache(数据缓冲区)

Redo Log Buffer(重做日志缓冲区)，Large Pool

26. 答案：字典管理方式和本地管理方式

27. Select 语句

Order by 另外在查询语句中 where 子句中，判断函数尽量不要放在左边

V、编程题

1.答：KEYS:

1. SELECT AVG(SCOST) FROM SOFTWARE WHERE DEVIN = ‘ORACLE’;
2. SELECT PNAME,TRUNC(MONTHS_BETWEEN(SYSDATE,DOB)/12) “AGE”, TRUNC(MONTHS_BETWEEN(SYSDATE,DOJ)/12) “EXPERIENCE” FROM PROGRAMMER;
3. SELECT PNAME FROM STUDIES WHERE COURSE = ‘PGDCA’;
4. SELECT MAX(SOLD) FROM SOFTWARE;
5. SELECT PNAME, DOB FROM PROGRAMMER WHERE DOB LIKE ‘%APR%’;
6. SELECT MIN(CCOST) FROM STUDIES;
7. SELECT COUNT(*) FROM STUDIES WHERE COURSE = ‘DCA’;
8. SELECT SUM(SCOST*SOLD-DCOST) FROM SOFTWARE GROUP BY DEVIN HAVING DEVIN = ‘C’;
9. SELECT * FROM SOFTWARE WHERE PNAME = ‘RAKESH’;
10. SELECT * FROM STUDIES WHERE SPLACE = ‘PENTAFOUR’;
11. SELECT * FROM SOFTWARE WHERE SCOST*SOLD-DCOST > 5000;
12. SELECT CEIL(DCOST/SCOST) FROM SOFTWARE;
13. SELECT * FROM SOFTWARE WHERE SCOST*SOLD >= DCOST;
14. SELECT MAX(SCOST) FROM SOFTWARE GROUP BY DEVIN HAVING DEVIN = ‘VB’;
15. SELECT COUNT(*) FROM SOFTWARE WHERE DEVIN = ‘ORACLE’;
16. SELECT COUNT(*) FROM STUDIES WHERE SPLACE = ‘PRAGATHI’;
17. SELECT COUNT(*) FROM STUDIES WHERE CCOST BETWEEN 10000 AND 15000;
18. SELECT AVG(CCOST) FROM STUDIES;
19. SELECT * FROM PROGRAMMER WHERE PROF1 = ‘C’ OR PROF2 = ‘C’;
20. SELECT * FROM PROGRAMMER WHERE PROF1 IN (‘C’,’PASCAL’) OR PROF2 IN (‘C’,’PASCAL’);
21. SELECT * FROM PROGRAMMER WHERE PROF1 NOT IN (‘C’,’C++’) AND PROF2 NOT IN (‘C’,’C++’);
22. SELECT TRUNC(MAX(MONTHS_BETWEEN(SYSDATE,DOB)/12)) FROM PROGRAMMER WHERE SEX = ‘M’;
23. SELECT TRUNC(AVG(MONTHS_BETWEEN(SYSDATE,DOB)/12)) FROM PROGRAMMER WHERE SEX = ‘F’;
24. SELECT PNAME, TRUNC(MONTHS_BETWEEN(SYSDATE,DOJ)/12) FROM PROGRAMMER ORDER BY PNAME DESC;

25. SELECT PNAME FROM PROGRAMMER WHERE TO_CHAR(DOB,' MON') = TO_CHAR(SYSDATE,' MON');
26. SELECT COUNT(*) FROM PROGRAMMER WHERE SEX = 'F' ;
27. SELECT DISTINCT(PROF1) FROM PROGRAMMER WHERE SEX = 'M' ;
28. SELECT AVG(SAL) FROM PROGRAMMER;
29. SELECT COUNT(*) FROM PROGRAMMER WHERE SAL BETWEEN 5000 AND 7500;
30. SELECT * FROM PROGRAMMER WHERE PROF1 NOT IN ('C' ,'C++' ,'PASCAL') AND PROF2 NOT IN ('C' ,'C++' ,'PASCAL');
31. SELECT PNAME,TITLE,SCOST FROM SOFTWARE WHERE SCOST IN (SELECT MAX(SCOST) FROM SOFTWARE GROUP BY PNAME);

32. SELECT 'Mr.' || PNAME || ' - has ' || TRUNC(MONTHS_BETWEEN(SYSDATE,DOJ)/12) || ' years of experience' "Programmer" FROM PROGRAMMER WHERE SEX = 'M' UNION SELECT 'Ms.' || PNAME || ' - has ' || TRUNC (MONTHS_BETWEEN (SYSDATE,DOJ)/12) || ' years of experience' "Programmer" FROM PROGRAMMER WHERE SEX = 'F' ;

2.答：KEYS:

1. SELECT DISTINCT(A.ENAME) FROM EMP A, EMP B WHERE A.EMPNO = B.MGR; or SELECT ENAME FROM EMP WHERE EMPNO IN (SELECT MGR FROM EMP);
2. SELECT * FROM EMP WHERE DEPTNO IN (SELECT DEPTNO FROM EMP GROUP BY DEPTNO HAVING COUNT(EMPNO)>10 AND DEPTNO=10);
3. SELECT A.ENAME "EMPLOYEE", B.ENAME "REPORTS TO" FROM EMP A, EMP B WHERE A.MGR=B.EMPNO;
4. SELECT * FROM EMP WHERE EMPNO IN (SELECT EMPNO FROM EMP MINUS SELECT MGR FROM EMP);
5. SELECT * FROM EMP WHERE SAL > (SELECT MIN(SAL) FROM EMP GROUP BY DEPTNO HAVING DEPTNO=20);
6. SELECT * FROM EMP WHERE SAL > (SELECT MAX(SAL) FROM EMP GROUP BY JOB HAVING JOB = 'MANAGER');
7. SELECT JOB, MAX(SAL) FROM EMP GROUP BY JOB;
8. SELECT * FROM EMP WHERE (DEPTNO, HIREDATE) IN (SELECT DEPTNO, MAX(HIREDATE) FROM EMP GROUP BY DEPTNO);
9. SELECT TO_CHAR(HIREDATE,' YYYY') "YEAR", COUNT(EMPNO) "NO. OF EMPLOYEES" FROM EMP GROUP BY TO_CHAR(HIREDATE,' YYYY') HAVING COUNT(EMPNO) = (SELECT MAX(COUNT(EMPNO)) FROM EMP GROUP BY TO_CHAR(HIREDATE,' YYYY'));
10. SELECT DEPTNO, LPAD(SUM(12*(SAL+NVL(COMM,0))),15) "COMPENSATION" FROM EMP GROUP BY DEPTNO HAVING SUM(12*(SAL+NVL(COMM,0))) = (SELECT MAX(SUM(12*(SAL+NVL(COMM,0)))) FROM EMP GROUP BY DEPTNO);

11. SELECT ENAME, HIREDATE, LPAD(‘*’ ,8) “RECENTLY HIRED” FROM EMP WHERE HIREDATE = (SELECT MAX(HIREDATE) FROM EMP) UNION SELECT ENAME NAME, HIREDATE, LPAD(‘ ‘,15) “RECENTLY HIRED” FROM EMP WHERE HIREDATE != (SELECT MAX(HIREDATE) FROM EMP);
12. SELECT ENAME,SAL FROM EMP E WHERE SAL > (SELECT AVG(SAL) FROM EMP F WHERE E.DEPTNO = F.DEPTNO);
13. SELECT ENAME, SAL FROM EMP A WHERE &N = (SELECT COUNT (DISTINCT(SAL)) FROM EMP B WHERE A.SAL<=B.SAL);
14. SELECT * FROM EMP A WHERE A.EMPNO IN (SELECT EMPNO FROM EMP GROUP BY EMPNO HAVING COUNT(EMPNO)>1) AND A.ROWID!=MIN (ROWID));
15. SELECT ENAME “EMPLOYEE”,TO_CHAR(TRUNC(MONTHS_BETWEEN(SYSDATE,HIREDATE)/12))||’ YEARS ‘||TO_CHAR(TRUNC(MOD(MONTHS_BETWEEN (SYSDATE, HIREDATE),12)))||’ MONTHS ‘ “LENGTH OF SERVICE” FROM EMP;

3 答：

3.1 答：select a.S#
from (select s#,score from SC where C#=’ 001′) a,
(select s#,score from SC where C#=’ 002′) b
where a.score>b.score and a.s#=b.s#;

3.2 答：select S#,avg(score)
from sc
group by S# having avg(score) >60;

3.3 答：select Student.S#,Student.Sname,count(SC.C#),sum(score)
from Student left Outer join SC on Student.S#=SC.S#
group by Student.S#,Sname

3.4 答：select count(distinct(Tname))
from Teacher
where Tname like ‘李%’;

3.5 答：select Student.S#,Student.Sname
from Student
where S# not in (select distinct(SC.S#) from SC,Course,Teacher where SC.C#=Course.C# and Teacher.T#=Course.T# and Teacher.Tname=’ 叶平’);

3.6 答：select Student.S#,Student.Sname
from Student,SC
where Student.S#=SC.S# and SC.C#=’ 001′ and exists(Select * from SC as

```
SC_2 where SC_2.S#=SC.S# and SC_2.C#=’002′);
```

3.7 答：
```
select S#,Sname
from Student
where S# in
(select S#
from SC ,Course ,Teacher
where SC.C#=Course.C# and Teacher.T#=Course.T# and Teacher.Tname=’叶平’ group by S# having count(SC.C#)=(select count(C#) from Course,Teacher where Teacher.T#=Course.T# and Tname=’叶平’));
```

3.8 答：
```
select S#,Sname
from Student
where S# not in (select Student.S# from Student,SC where S.S#=SC.S# and score>60);
```

3.9 答：
```
select Student.S#,Student.Sname
from Student,SC
where Student.S#=SC.S#
group by Student.S#,Student.Sname having count(C#) <(select count(C#) from Course);
```

3.10 答：
```
select S#,Sname
from Student,SC
where Student.S#=SC.S# and C# in （select C# from SC where S#=’1001’）;
```

3.11 答：
```
Delect SC
from course ,Teacher
where Course.C#=SC.C# and Course.T#= Teacher.T# and Tname=’叶平’;
```

3.12 答：
```
SELECT L.C# 课程 ID,L.score 最高分,R.score 最低分
FROM SC L ,SC R
WHERE L.C# = R.C#
and
L.score = (SELECT MAX(IL.score)
FROM SC IL,Student IM
WHERE IL.C# = L.C# and IM.S#=IL.S#
GROUP BY IL.C#)
and
R.Score = (SELECT MIN(IR.score)
FROM SC IR
WHERE IR.C# = R.C#
GROUP BY IR.C# );
```

3.13 答：SELECT 1+(SELECT COUNT(distinct 平均成绩)
FROM (SELECT S#,AVG(score) 平均成绩
FROM SC
GROUP BY S#) T1
WHERE 平均成绩 > T2.平均成绩) 名次, S# 学生学号,平均成绩
FROM (SELECT S#,AVG(score) 平均成绩 FROM SC GROUP BY S#) T2
ORDER BY 平均成绩 desc;

3.14 答：SELECT t1.S# as 学生 ID,t1.C# as 课程 ID,Score as 分数
FROM SC t1
WHERE score IN (SELECT TOP 3 score
FROM SC
WHERE t1.C#= C#
ORDER BY score DESC)
ORDER BY t1.C#;

3.15 答：SELECT t1.S# as 学生 ID,t1.C# as 课程 ID,Score as 分数
FROM SC t1
WHERE score IN (SELECT TOP 2 score
FROM SC
WHERE t1.C#= C#
ORDER BY score DESC)
ORDER BY t1.C#;

4 答： select Name,Course,Score from T1,T2,T3 where T3.ID=T1.NameID and T1.CourseID=T2.ID

5.答： (T-SQL)：

－利用存储过程查找看本日期和操作类型有没有数据，如果没有则插入数据，如果有则更新数据使操作次数加 1

create or replace procedure update_pro(v_DateStr in varchar,v_OprType in integer) as

vv_DateStr T1.DateStr%type;

vv_OprType T1.OprType%type;

cursor my_cursor is(select DateStr,OprType,OprCount from T1 where DateStr= v_DateStr and OprType= v_OprType);－声明游标

```
begin

open my_cursor;

fetch my_cursor into vv_DateStr,vv_OprType;

if my_cursor%notfound then

insert into T1(DateStr,OprType,OprCount) values(v_DateStr,v_OprType,1);

else

update T1 set OprCount=OprCount+1 where DateStr=v_DateStr and
OprType=v_OprType;

end if;

commit;

close my_cursor;

end;
```

6.答： ALTER TABLE T1

ADD CONSTRAINTS CHK_FILED CHECK (Field1 like ‘aaa%’ or Field1 like

‘bbb%’ or Field1 like ‘ccc%’);

7.答：（T-SQL）：此题应用存储过程分批删除并提交，如下是每次删除 10000

```
create or replace procedure Del_pro as

v_Boolean Boolean :=true;

v_count integer :=0;

begin

while v_Boolean loop

delete from t_Log where Add_months(LogDateTime,6)<sysdate and rownum<10000;

commit;

select count(*) into v_count from t_log   where
Add_months(LogDateTime,6)<sysdate;
```

```
if v_count=0 then

v_Boolean=false;

end if;

end loop;

end;
```

数据量比较大的情况，可以考虑分批删除，效率会高一些。可使用循环控制语句中，使用 rownum<10000 来分 300 次来删除。注意每次删除后 commit。

8.答：答：1）SQL 语句如下：

```
select stu.sno, stu.sname
    from Student stu
    where (select count(*)
        from sc
        where sno=stu.sno and cno = (select cno
            from Course
            where cname='计算机原理')) != 0;
```

2）SQL 语句如下：

```
select cname
    from Course
    where cno in ( select cno
        from sc
        where sno = (select sno
            from Student
            where sname='周星驰'));
```

3）SQL 语句如下：

```
select stu.sno, stu.sname
    from student stu
    where (select count(*)
        from sc
where sno=stu.sno) = 5;
```

9.答：答：1）建表语句如下（mysql 数据库）：

```
create table s(id integer primary key, name varchar(20));
create table c(id integer primary key, name varchar(20));
create table sc( sid integer references s(id), cid integer references c(id),
primary key(sid,cid) );
```

2）SQL 语句如下：

```
select stu.id, stu.name
    from s stu
    where (select count(*)
        from sc
```

```
        where sid=stu.id) = (select count(*)
            from c);
```

3）SQL 语句如下：

```
select stu.id, stu.name
    from s stu
    where (select count(*)
        from sc
        where sid=stu.id)>=5;
```

10.答： SQL 语句如下：

```
select employee.name
    from test employee
    where employee.age > (select manager.age
        from test manager
        where manager.id=employee.manager);
```

11.答： SQL 语句为：

```
SELECT C.CITYNO, C.CITYNAME, C.STATENO, S.STATENAME
    FROM CITY C, STATE S
WHERE C.STATENO=S.STATENO(+) ORDER BY(C.CITYNO);
```

三、Java 基础、J2SE

I 单选题

1、(B)	2、(B)	3、(B)
4、(A)	5、(C)	6、(B)
7、(C)	8、(A)	9、(E)
10、(C)	11、(B)	12、(C)
13、(B)	14、(B)	15、(D)
16、(C)	17、(A)	18、(D)
19. (A)	20. (C)	21 (.B)
22.(A)		

II 多选题（共 6 题）

1、(CD)	2、(ACD)	3、(CD)
4、(ACD)	5、(AD)	6、(BC)

III 填空题

1. 编码、编译、运行

2. JApplet、Applet、MyApplet.java

3. javac、3、class
4. 2、2
5. 0
6. 1、0
7. 抽象
8. _120
9. Hello! I love JAVA. I love JAVA.
10. S S
11. userA_
12. super();
13. A B A ___
14. 123

IV、 判断题

1、对 2、错 3、对 4. 错 5. 对。因为这样存储的结构为散乱结构。正确定义为 Map map=new HashMap()。
6. 错 7. 对

V、问答题

1. 答：boolean char byte short int long float double
2. 答： int double
3. 代表隐含参数的调用、调用本类的其它的构造器
4. 单继承。
5. 多线程 ； 对象 clone
6. final
7. Unicode 2
8. 答：8
9. 覆盖方法返回类型与父类返回类型一样
   覆盖方法的参数类型、次序、方法名称与被覆盖方法一致
   覆盖方法的访问级别不能比被覆盖方法访问级别低
   覆盖方法不能比被覆盖方法抛出的异常多 8
10. 会
11. 运行时异常能够进行编译；一般异常不能进行编译
12. a.抽象性：
    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。

b.继承性：
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。
c.封装性：
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。
d. 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。
13. 答：不是。
基本数据类型包括 byte、int、char、long、float、double、boolean 和 short。

14. 异常表示程序运行过程中可能出现的非正常状态，运行时异常表示虚拟机的通常操作中可能遇到的异常，是一种常见运行错误。java 编译器要求方法必须声明抛出可能发生的非运行时异常，但是并不要求必须声明抛出未被捕获的运行时异常。

15. 答：区别如下：

作用域	当前类	同包	子类	其他
public	√	√	√	√
protected	√	√	√	×
Default	√	√	×	×
private	√	×	×	×

不写时默认为 default。

16. 答：(注：修饰符是影响类、变量及成员方法的生存空间和可访问性的关键字)

修饰符	类	成员方法	成员变量	局部变量
abstract	√	√	—	—
static	—	√	√	—
public	√	√	√	—
protected	—	√	√	—
private	—	√	√	—
synchronized	—	√	—	—
native	—	√	—	—
volatile	—	—	√	—
final	√	√	√	—
transient	—	—	√	√

以下是访问控制修饰符： 默认为 friendly

修饰符	同类	同包	子孙类	不同包
public	√	√	√	√
protected	√	√	√	—
friendly	√	√	—	—
private	√	—	—	—

17. 答：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。

18. 答:不正确；精度不准确,应该用强制类型转换，如下所示：
`float f=(float)3.4;`

19. 答：`short s1 = 1; s1 =s1 + 1;`s1+1 运算结果是 int 型，需要强制转换类型；`short s1 =1; s1 += 1;`可以正确编译,自动类型提升。

20. 答：goto 是 java 中的保留字，现在没有在 java 中使用。

21. 答：在 Java 中类变量在局部中一定要初始化，因为局部变量会覆盖全局变量，否则会报错：变量未初始化。全局变量则可以不初始化，而到具体的内部方法或其他的类成员中初始化。

22. 答：数组是作为一种对象实现的。数组元素可以包含作何类型值，但数组里面的每个元素的类型必须一致，创建数组步聚如下：
    - 声明
    - 构造
    - 初始化

23. 答：Java 提供两种不同的类型：引用类型和原始类型（或内置类型）；

int 是 java 的原始数据类型，Integer 是 java 为 int 提供的封装类。Java 为每个原始类型提供了封装类：
原始类型：boolean,char,byte,short,int,long,float,double
封装类型：Boolean，Character，Byte，Short，Integer，Long，Float，Double
引用类型和原始类型的行为完全不同，并且它们具有不同的语义。引用类型和原始类型具有不同的特征和用法，它们包括：大小和速度问题，这种类型以哪种类型的

数据结构存储，当引用类型和原始类型用作某个类的实例数据时所指定的缺省值。对象引用实例变量的缺省值为 null，而原始类型实例变量的缺省值与它们的类型有关。

24. 答：因为每个类都继承了 Object 类，所以都实现了 toString()方法。
通过 toString()方法可以决定所创建对象的字符串表达形式。

25. 答：Object 类是所有其他的类的超类，Object 的一个变量可以引用任何其他类的对象。因为数组是作为类实现的，所以 Object 的一个变量也可以引用任何数组，它包括以下几种方法：
clone() 、 equals() 、 finalize() 、 getClass() 、 hashCode()、notify() 、 notifyAll() 、toString() 、 wait()。

26. 答：java 具有以下几个主要特点：
- 简单性
- 面向对象：JAVA 是完全面向对象的，它支持静态和动态风格的代码继承及重用
- 分布式：包括数据分布和操作分布
- 健壮性：java 系统仔细检测对内存的每次访问，确认它是否合法，而且在编译和运行程序时，都要对可能出现的问题进行检查，以消除错误的产生。
- 结构中立
- 安全性：java 不支持指针，一切对内存的访问都必须通过对象的实例变量来实现，这样就防止程序员使用木马等欺骗手段访问对象的私有成员，同时也避免了指针操作中容易产生的错误。
- 与平台无关：java 写的应用程序不用修改就可在不同的软硬平台上运行。平台无关性有两种：源代码级和目标代码级。 Java 主要靠 JAVA 虚拟机在目标代码级上实现平台无关性
- 解释性：运行 JAVA 程序时，它首先被编译成字节代码，字节代码非常类似机器码，执行效率非常高。
- 高性能
- 多线程
- 动态性：它允许程序动态的装入运行时需要的类。

27. 答：&是位运算符，表示按位与运算，&&是逻辑与运算符，表示逻辑与（and）。

28. 答：区别主要有两点：a.条件操作只能操作布尔型的,而逻辑操作不仅可以操作布尔型,而且可以操作数值型 b.逻辑操作不会产生短路。

29. 答：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。

30. 答： Math.round(11.5)==12 Math.round(-11.5)== -11 ，round 方法返回与

参数最接近的长整数，参数加 1/2 后求其 floor。

31. 答： switch（expr1）中，expr1 是一个整数表达式。因此传递给 switch 和 case 语句的参数应该是 int、 short、 char 或者 byte。long,String 都不能作用于 switch。

32. 答： 2 << 3。

33. 答：数组没有 length()这个方法，有 length 的属性。String 有 length()这个方法。

34. 答：在最外层循环前加 label 标识,然后用 break:label 方法即可跳出多重循环。

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

36. 答：不对，有相同的 hash code。

37. 答：String 类是 final 类，故不可以继承。

38. A:“beijing”==“beijing”;
B:“beijing”.equalsIgnoreCase（newString（“beijing”））;
答：A 和 B。

39. 答：是值传递。Java 编程语言只有值传递参数。当一个对象实例作为一个参数被传递到方法中时，参数的值就是对该对象的引用。对象的内容可以在被调用的方法中改变，但对象的引用是永远不会改变的。

40. 答：String 类的值在初始后不能改变，如果要改变，可转换为 StringBuffer 类，这个类的值是可以动态改变的。(这里主要考 String 和 StringBuffer 的区别)

41. 答：String 的长度是不可变的；
StringBuffer 的长度是可变的，如果你对字符串中的内容经常进行操作，特别是内容要修改时，那么使用 StringBuffer，如果最后需要 String，那么使用 StringBuffer 的 toString()方法；线程安全；
StringBuilder 是从 JDK 5 开始，为 StringBuffer 该类补充了一个单个线程使用的等价类；通常应该优先使用 StringBuilder 类，因为它支持所有相同的操作，但由于它不执行同步，所以速度更快。

42. 答：方法的重写 Overriding 和重载 Overloading 是 Java 多态性的不同表现。重写 Overriding 是父类与子类之间多态性的一种表现，重载 Overloading 是一个类中多态性的一种表现。如果在子类中定义某方法与其父类有相同的名称和参数，我们说该方法被重写(Overriding)。子类的对象使用这个方法时，将调用子类中的定义，对它而言，父类中的定义如同被“屏蔽

”了。如果在一个类中定义了多个同名的方法，它们或有不同的参数个数或有不同的参数类型，则称为方法的重载(Overloading)。Overloaded 的方法是可以改变返回值的类型。

43. 答：输出结果为：
1） Class A: a=1 d=2.0；
2） Class A: a=1 d=2.0
Class B: a=3.0 d=Java program。

44. 答：JVM 中类的装载是由 ClassLoader 和它的子类来实现的，Java ClassLoader 是一个重要的 Java 运行时系统组件。它负责在运行时查找和装入类文件的类。
45. 答：能够定义成为一个中文的，因为 java 中以 unicode 编码，一个 char 占 2 个字节，所以放一个中文是没问题的。

46. 答：声明方法的存在而不去实现它的类被叫做抽象类（abstract class），它用于要创建一个体现某些基本行为的类，并为该类声明方法，但不能在该类中实现该类的情况。不能创建 abstract 类的实例。然而可以创建一个变量，其类型是一个抽象类，并让它指向具体子类的一个实例。不能有抽象构造函数或抽象静态方法。Abstract 类的子类为它们父类中的所有抽象方法提供实现，否则它们也是抽象类。取而代之，在子类中实现该方法。知道其行为的其它类可以在类中实现这些方法。接口（interface）是抽象类的变体。新型多继承性可通过实现这样的接口而获得。接口中的所有方法都是抽象的，所有成员变量都是 public static final 的。一个类可以实现多个接口，当类实现特殊接口时，它定义（即将程序体给予）所有这种接口的方法。然后，它可以在实现了该接口的类的任何对象上调用接口的方法。由于有抽象类，它允许使用接口名作为引用变量的类型。通常的动态联编将生效。引用可以转换到接口类型或从接口类型转换，instanceof 运算符可以用来决定某对象的类是否实现了接口。

47. 答：Static Nested Class 是被声明为静态（static）的内部类，它可以不依赖于外部类实例被实例化。而通常的内部类需要在外部类实例化后才能实例化。

48. 答：会；存在无用但可达的对象，这些对象不能被 GC 回收，导致耗费内存资源。

49. 答：都不能。

50. 答：静态变量也称为类变量，归全类共有，它不依赖于某个对象，可通过类名直接访问；而实例变量必须依存于某一实例，只能通过对象才能访问到它。

51. 答：不可以，如果其中包含对象的 method()，不能保证对象初始化。

52. 答：Clone 有缺省行为：super.clone()，他负责产生正确大小的空间，并逐位复制。

53. 答：GC 是垃圾收集的意思（Gabage Collection）,内存处理是编程人员容易出现问题的地方，忘记或者错误的内存回收会导致程序或系统的不稳定甚至崩溃，Java 提供的 GC 功能可以自动监测对象是否超过作用域从而达到自动回收内存的目的，Java 语言没有提供释放已分配内存的显示操作方法。Java 程序员不用担心内存管理，因为垃圾收集器会自动进行管理。要请求垃圾收集，可以调用下面的方法之一：System.gc()或 Runtime.getRuntime().gc()。

54. 答：Java 语言中一个显着的特点就是引入了垃圾回收机制，使 c++程序员最头疼的内存管理的问题迎刃而解，它使得 Java 程序员在编写程序的时候不再需要考虑内存管理。由于有个垃圾回收机制，Java 中的对象不再有“作用域”的概念，只有对象的引用才有“作用域”。垃圾回收可以有效的防止内存泄露，有效的使用可以使用的内存。垃圾回收器通常是作为一个单独的低级别的线程运行，不可预知的情况下对内存堆中已经死亡的或者长时间没有使用的对象进行清除和回收，程序员不能实时的调用垃圾回收器对某个对象或所有对象进行垃圾回收。回收机制有分代复制垃圾回收和标记垃圾回收，增量垃圾回收。

55. 答：对于 GC 来说，当程序员创建对象时，GC 就开始监控这个对象的地址、大小以及使用情况。通常，GC 采用有向图的方式记录和管理堆(heap)中的所有对象。通过这种方式确定哪些对象是"可达的"，哪些对象是"不可达的"。当 GC 确定一些对象为"不可达"时，GC 就有责任回收这些内存空间。程序员可以手动执行 System.gc()，通知 GC 运行，但是 Java 语言规范并不保证 GC 一定会执行。

56. 答：两个对象，一个是"xyz",一个是指向"xyz"的引用对象。

57. 答：接口可以继承接口。抽象类可以实现(implements)接口，抽象类可继承实体类，但前提是实体类必须有明确的构造函数。

58. 答：都不能。

59. 答：由于 Java 不支持多继承，而有可能某个类或对象要使用分别在几个类或对象里面的方法或属性，现有的单继承机制就不能满足要求。与继承相比，接口有更高的灵活性，因为接口中没有任何实现代码。当一个类实现了接口以后，该类要实现接口里面所有的方法和属性，并且接口里面的属性在默认状态下面都是 public static,所有方法默认情况下是 public.一个类可以实现多个接口。

60. 答：可以；必须只有一个类名与文件名相同。

61. 答：常用的类：BufferedReader， BufferedWriter，File， FileReader，FileWirter，String，Integer，Math，Socket，ArrayList，HashSet，HashMap；常用的包：java.lang， java.awt， java.io， java.util，java.sql， java.net；常用的接口：Runnable ， Remote，List，Set，Map，Document，NodeList ；

62. 答：可以继承其他类或实现其他接口，在 swing 编程中常用此方式。

63. 答：一个内部类对象可以访问创建它的外部类对象的内容。

64. 答：方法的覆盖 Overriding 和重载 Overloading 是 java 多态性的不同表现；覆盖 Overriding 是父类与子类之间多态性的一种表现，重载 Overloading 是一个类中多态性的一种表现。

65. 答：表示该类不能被继承，是顶级类。

66. 1）java.lang.Thread (T)
2）java.lang.Number(T)
3）java.lang.Double(F)
4）java.lang.Math(F)
5）java.lang.Void(F)
6）java.lang.Class(F)
7）java.lang.ClassLoader(T)
答：1、2、7 可以被继承。

67. 答：输出结果为 1a2b2b；类的 static 代码段,可以看作是类首次加载(虚拟机加载)执行的代码,而对于类加载,首先要执行其基类的构造,再执行其本身的构造。

68. 答：输出结果为：
FatherClass Create
FatherClass Create
ChildClass Create

69. 答：
InterClass
Create
OuterClass
Create

70. 答：答案为
C、E；说明如下：
1）静态内部类可以有静态成员，而非静态内部类则不能有静态成员；故 A、B 错；
2）静态内部类的非静态成员可以访问外部类的静态变量，而不可访问外部类的非静态变量；故 D 错；
3）非静态内部类的非静态成员可以访问外部类的非静态变量；故 C 正确。

71. 答：1)调用数值类型相应包装类中的方法 parse***(String)或 valueOf(String)即可返回相应基本类型或包装类型数值；
2)将数字与空字符串相加即可获得其所对应的字符串;另外对于基本类型数字还可调用 String 类中的 valueOf(…)方法返回相应字符串,而对于包装类型数字则可调用其 toString()方法获得相应字符串；
3)可用该数字构造一 java.math.BigDecimal 对象,再利用其 round()方法进行四舍五入

到保留小数点后两位,再将其转换为字符串截取最后两位。

72. 答：可用字符串构造一 StringBuffer 对象,然后调用 StringBuffer 中的 reverse 方法即可实现字符串的反转,调用 replace 方法即可实现字符串的替换。

73. 答：示例代码如下:

```
String s1 ="你好";
String s2 = new String(s1.getBytes("GB2312"), "ISO8859-1");
```

74. 答:1)创建 java.util.Calendar 实例(Calendar.getInstance()),调用其 get()方法传入不同的参数即可获得参数所对应的值,如：

```
calendar.get(Calendar.YEAR);//获得年
```

2)以下方法均可获得该毫秒数:

```
Calendar.getInstance().getTimeInMillis();
System.currentTimeMillis();
```

3)示例代码如下:

```
Calendar time = Calendar.getInstance();
time.set(Calendar.DAY_OF_MONTH,
time.getActualMaximum(Calendar.DAY_OF_MONTH));
```

4)利用 java.text.DataFormat 类中的 format()方法可将日期格式化。

75. 答：assertion(断言)在软件开发中是一种常用的调试方式，很多开发语言中都支持这种机制。一般来说，assertion 用于保证程序最基本、关键的正确性。assertion 检查通常在开发和测试时开启。为了提高性能，在软件发布后，assertion 检查通常是关闭的。在实现中，断言是一个包含布尔表达式的语句，在执行这个语句时假定该表达式为 true；如果表达式计算为 false，那么系统会报告一个 Assertionerror。断言用于调试目的：

```
assert(a > 0);
// throws an Assertionerror if a <= 0 断言可以有两种形式：
assert Expression1 ;
assert Expression1 : Expression2 ;
```

Expression1 应该总是产生一个布尔值。

Expression2 可以是得出一个值的任意表达式；这个值用于生成显示更多调试信息的 String 消息。断言在默认情况下是禁用的,要在编译时启用断言,需使用 source1.4 标记：

```
javac -source 1.4 Test.java
```

要在运行时启用断言，可使用-enableassertions 或者-ea 标记。

要在运行时选择禁用断言，可使用 -da 或者 -disableassertions 标记。

要在系统类中启用断言，可使用 -esa 或者 -dsa 标记。还可以在包的基础上启用或者禁用断言。可以在预计正常情况下不会到达的任何位置上放置断言。断言可以用于验证传递给私有方法的参数。不过,断言不应该用于验证传递给公有方法的参数,因为不管是否启用了断言，公有方法都必须检查其参数。不过，既可以在公有方法中，也可以在非公有方法中利用断言测试后置条件。另外，断言不应该以任何方式改变程序的状态。

76. 答：当 JAVA 程序违反了 JAVA 的语义规则时，JAVA 虚拟机就会将发生的错误表示为一个异常。违反语义规则包括 2 种情况。一种是 JAVA 类库内置的语义检查。例如数组下标越界,会引发 IndexOutOfBoundsException;访问 null 的对象时会引发 NullPointerException。另一种情况就是 JAVA 允许程序员扩展这种语义检查，程序员可以创建自己的异常，并自由选择在何时用 throw 关键字引发异常。所有的异常都是 java.lang.Thowable 的子类。

77. 答：error 表示系统级的错误和程序不必处理的异常，是恢复不是不可能但很困难的情况下的一种严重问题；比如内存溢出，不可能指望程序能处理这样的情况；exception 表示需要捕捉或者需要程序进行处理的异常，是一种设计或实现问题；也就是说，它表示如果程序运行正常，从不会发生的情况。

78. 答：会执行，在 return 前执行。

79. 答：Java 通过面向对象的方法进行异常处理，把各种不同的异常进行分类，并提供了良好的接口。在 Java 中，每个异常都是一个对象，它是 Throwable 类或其它子类的实例。当一个方法出现异常后便抛出一个异常对象，该对象中包含有异常信息，调用这个对象的方法可以捕获到这个异常并进行处理。Java 的异常处理是通过 5 个关键词来实现的：try、catch、throw、throws 和 finally。一般情况下是用 try 来执行一段程序，如果出现异常，系统会抛出（throws）一个异常，这时候你可以通过它的类型来捕捉（catch）它，或最后（finally）由缺省处理器来处理；
try 用来指定一块预防所有“异常”的程序；
catch 子句紧跟在 try 块后面，用来指定你想要捕捉的“异常”的类型；
throw 语句用来明确地抛出一个“异常”；
throws 用来标明一个成员函数可能抛出的各种“异常”；
Finally 为确保一段代码不管发生什么“异常”都被执行一段代码；
可以在一个成员函数调用的外面写一个 try 语句，在这个成员函数内部写另一个 try 语句保护其他代码。每当遇到一个 try 语句，“异常”的框架就放到堆栈上面，直到所有的 try 语句都完成。如果下一级的 try 语句没有对某种“异常”进行处理，堆栈就会展开，直到遇到有处理这种“异常”的 try 语句。

80. 答：异常表示程序运行过程中可能出现的非正常状态，运行时异常表示虚拟机的通常操作中可能遇到的异常，是一种常见运行错误。java 编译器要求方法必须声明抛出可能发生的非运行时异常，但是并不要求必须声明抛出未被捕获的运行时异常。

. . . . . .

杭州归谷培训中心

笔面试题库

目　录

一、数据结构、算法、计算机基础

I、单选题

1、（D）	2、（B）	3、（D）
4、（D）	5、（C）	6、（C）
7、（B）	8、（C）	9、（B）
10、（D）	11、（C）	12、（D）
13、（A）	14、（D）	15、（B）
16、（D）	17、（C）	18、（B）

II 问答题

1、答：8bit。

2、答：ls, pwd, mkdir, rm, cp, mv, cd, ps, ftp, telnet, ping, env, more, echo

3、答：栈是一种线形集合，其添加和删除元素的操作应在同一段完成，栈按照后进先出的方式进行处理；堆是栈的一个组成元素。

4、答：顺序为：DJGEBKNIFCA。

5、答：排序的方法有：插入排序（直接插入排序、希尔排序），交换排序（冒泡排序、快速排序），选择排序（直接选择排序、堆排序），归并排序，分配排序（箱排序、基数排序）；

快速排序的伪代码：

使用快速排序方法对 a[0 :n-1]排序,从 a[0 :n-1]中选择一个元素作为 middle，该元素为支点；把余下的元素分割为两段 left 和 right，使得 left 中的元素都小于等于支点，而 right 中的元素都大于等于支点；递归地使用快速排序方法对 left 进行排序；递归地使用快速排序方法对 right 进行排序；所得结果为 left + middle + right。

6、答案：只允许在端点处插入和删除元素。

7.答案：线性存储结构和链表存储结构。

8. 答案：便于插入和删除操作。

9.答案：方便运算的实现。

10.答案：从表中任一结点出发都能访问到整个链表。

11. 答案：随机存取的存储结构和顺序存取的存储结构。

12.答案：有且只有 1

13. 答案：31

14. 答案：5 种形态。

15. 答案：13

16. 答案：cedba
17. 答案：DGEBHFCA
18. 答案：gdbehfca19.答案：解题方案的准确而完整的描述。

20. 答案：无穷性。

21. 答案：顺序、选择、循环。

22.答案：算法执行过程中所需要的基本运算次数。

23.答案：执行过程中所需要的存储空间。

24. 答案：分析算法的效率以求改进。

25. 答案：数据的逻辑结构在计算机中的表示。

26. 答案：反映数据元素之间逻辑关系的数据结构。

27.答案：有且只有 1。

28. 答案：128

29.答案：16

30. 答案：31

31. 答案：350
说明：完全二叉树总结点数为 N，若 N 为奇数，则叶子结点数为（N+1）/2；若 N 为偶数，则叶子结点数为 N/2。

32.答案：线性结构和非线性结构。

33.答案：gdbehfca

34.答案：串中所含字符的个数。

35. 答案：模式匹配。

36.答案：N-1

37.答案：N

38. 答案：N

39. 答案：冒泡排序

40. 答案：n(n-1)/2

41.答案：冒泡排序

42.答案：堆排序

43.答案：插入类排序

44.答案：选择类排序

45.答案：归并排序

46.答案：直接插入排序

47.答案：连续不连续都可以。

48.答案：一是对数据对象的运算和操作，二是算法的控制结构。

49.答案：时间复杂度和空间复杂度。实现算法所需的存储单元多少和算法的工作量大小分别称为算法的空间复杂度和时间复杂度 。

50.答案：所谓数据处理是指对数据集合中的各元素以各种方式进行运算，包括插入、删除、查找、更改等运算，也包括对数据元素进行分析。

51.答案：数据结构是指相互有关联的数据元素的集合。

52.答案：数据结构分为逻辑结构与存储结构，线性链表属于存储结构。

53.答案：数据结构包括数据的逻辑结构和数据的存储结构。

54. 答案：用前趋和后继关系来描述。

55.答案：有线性结构和非线性结构两大类。

56.答案：顺序、链接、 索引等存储结构。

57.答案：顺序存储是把逻辑上相邻的结点存储在物理位置相邻的存储单元中。

58.答案：入栈、退栈与读栈顶元素。

59.答案：入队运算与退队运算。

60.答案：链式存储和顺序存储 。

61. 答案：逻辑结构中相邻的结点在存储结构中仍相邻。

62.答案：入队运算与退队运算。每进行一次入队运算，队尾指针就进 1。

63.答案：上溢 。

64 答案：下溢。65.答案：主要研究数据的逻辑结构、对各种数据结构进行的运算，以及数据的存储结构。

66.答案：队列

67. 答案：节省存储空间，降低上溢发生的机率。

III、编程题

1、答：C++中冒泡排序：

```
void swap( int& a, int& b ){
    int c=a;
    a = b;
    b = c;
}
void bubble( int* p, int len ){
    bool bSwapped;
    do {
        bSwapped = false;
        for( int i=1; i<len; i++ ){
            if( p[i-1]>p[i] ){
                swap( p[i-1], p[i] );
                bSwapped = true;
            }
        }
    }while( bSwapped);
}
```

2、答：代码如下：

```
#include <math.h>
bool prime(int n ){
    if(n<=0) {
        exit(0);
    }
    for( int i=2; i<=n; i++ )
        for( int j=2; j<=sqrt(i); j++) {
            if((n%j==0) && (j!=n)) {
                return false;
            }
            return true;
        }
```

```
    }
}
```

采用 C++，因为其运行效率高。

3. 答：代码如下：

```
package test;
public class CountGame {
    private static boolean same(int[] p,int l,int n){
        for(int i=0;i<l;i++){
            if(p[i]==n){
                return true;
            }
        }
        return false;
    }
    public static void play(int playerNum, int step){
        int[] p=new int[playerNum];
        int counter = 1;
        while(true){
            if(counter > playerNum*step){
                break;
            }
            for(int i=1;i<playerNum+1;i++){
                while(true){
                    if(same(p,playerNum,i)==false) {
                        break;
                    }else{
                        i=i+1;
                    }
                }
                if(i > playerNum){break;}
                if(counter%step==0){
                    System.out.print(i + " ");
                    p[counter/step-1]=i;
                }
                counter+=1;
            }
        }
        System.out.println();
    }
    public static void main(String[] args) {
        play(10, 7);
    }
}
```

4. 答：C++的代码实现如下：

```
#include <iostream>
#include <iomanip>
#include <vector>
using namespace std;
class longint {
    private: vector<int> iv;
public:
    longint(void) {
        iv.push_back(1);
    }
    longint& multiply(const int &);
    friend ostream& operator<<(ostream &, const longint &);
};
ostream& operator<<(ostream &os, const longint &v) {
    vector<int>::const_reverse_iterator iv_iter = v.iv.rbegin();
    os << *iv_iter++;
    for ( ; iv_iter < v.iv.rend(); ++iv_iter) {
        os << setfill( '0') << setw(4) << *iv_iter;
    }
    return os;
}

longint& longint::multiply(const int &rv) {
    vector<int>::iterator iv_iter = iv.begin();
    int overflow = 0, product = 0;
    for ( ; iv_iter < iv.end(); ++iv_iter) {
        product = (*iv_iter) * rv;
        product += overflow;
        overflow = 0;
        if (product > 10000) {
            overflow = product / 10000;
            product -= overflow * 10000;
        }
        iv_iter = product;
    }
    if (0 != overflow) {
        iv.push_back(overflow);
    }
    return *this;
}
int main(int argc, char **argv) {
    longint result;
    int l = 0;
```

```
    if(argc==1){
        cout << "like:
        multiply 1000" << endl;
        exit(0);
    }
    sscanf(argv[1], "%d", &l);
    for (int i = 2; i <= l; ++i) {
        result.multiply(i);
    }cout << result << endl;
    return 0;
}
```

5.答：
```
public class num5 {
  public static void main(String[] args) {
    for (int i = 1; i < 9; i++) {
      for (int j = 1; j <= i; j++) {
        System.out.print(j+”*”+i+”=”+i*j+”\t”);
      }
      System.out.println();
    }
  }
}
```

6.答:
```
public class num2{
public static void main(String[] args){
int ywzm,space,num,other;
ywzm=space=num=other=0;
BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
String str = null;
try {
str = br.readLine();
} catch (IOException ex) {
Logger.getLogger(num2.class.getName()).log(Level.SEVERE, null,
ex);
}
char[] carr=str.toCharArray();
for (int i = 0; i < carr.length; i++) {
if((carr>=’0′)&&(carr<=’9′))
num++;
else if((carr>=’A’&&carr<=’Z’)||(carr>=’a’&&carr<=’z’))
ywzm++;
else if(carr==’  ‘)
space++;
else
```

```
other++;
}
System.out.println(“英文字母个数:”+ywzm+”\t 空格个数”+space+”\t 数字个数” +num+” \t 其他字符个数:” +other);
}
}
```

7.答：折半查找法也称为二分查找法，它充分利用了元素间的次序关系，采用分治策略，可在最坏的情况下用 O(log n)完成搜索任务。它的基本思想是，将 n 个元素分成个数大致相同的两半，取 a[n/2]与欲查找的作比较，如果 x=a[n/2]则找到 x，算法终止。如果 xa[n/2]，则我们只要在数组 a 的右半部继续搜索 x。二分搜索法的应用极其广泛，而且它的思想易于理解,但是要写一个正确的二分搜索算法也不是一件简单的事。第一个二分搜索算法早在 1946 年就出现了,但是第一个完全正确的二分搜索算法直到 1962 年才出现。Bentley 在他的著作《Writing Correct Programs》中写道，90%的计算机专家不能在 2 小时内写出完全正确的二分搜索算法。问题的关键在于准确地制定各次查找范围的边界以及终止条件的确定，正确地归纳奇偶数的各种情况，其实整理后可以发现它的具体算法是很直观的，我们可用 C++描述如下：

```
template
int BinarySearch(Type a[],const Type& x,int n)
{
int left=0;
int right=n-1;
while(left<=right){
int middle=(left+right)/2;
if (x==a[middle]) return middle;
if (x>a[middle]) left=middle+1;
else right=middle-1;
}
return -1;
}
```

模板函数 BinarySearch 在 a[0]<=a[1]<=...<=a[n-1]共 n 个升序排列的元素中搜索 x,找到 x 时返回其在数组中的位置，否则返回-1。容易看出，每执行一次 while 循环，待搜索数组的大小减少一半,因此整个算法在最坏情况下的时间复杂度为 O(log n)。在数据量很大的时候，它的线性查找在时间复杂度上的优劣一目了然。

java 中的实例如下：

```
public class TestSearch {
public static void main(String[] args) {
int a[] = { 1, 3, 6, 8, 9, 10, 12, 18, 20, 34 };
int i = 12;
```

```
//System.out.println(search(a, i));
System.out.println(binarySearch(a, i));
}

public static int search(int[] a, int num) {
for(int i=0; i
if(a == num) return i;
}
return -1;
}

public static int binarySearch(int[]a, int num) {
if (a.length==0) return -1; int startPos = 0;
int endPos = a.length-1;
int m = (startPos + endPos) / 2;
while(startPos <= endPos){
if(num == a
) return m;
if(num > a
) {
startPos = m + 1;
}
if(num < a
) {
endPos = m -1;
}
m = (startPos + endPos) / 2;
}
return -1;
}
}
```

```
8.答1 List reverse(List l) {
2 if(!l) return l;
3 list cur = l.next;
4 list pre = l;
5 list tmp;
6 pre.next = null;
7 while ( cur ) {
8 tmp = cur;
9 cur = cur.next;
10 tmp.next = pre
11 pre = tmp;
```

```
12 }
13 return tmp;
14 }
```

9. 答：
```
1 List resverse(list l) {
2 if(!l || !l.next) return l;
3
4 List n = reverse(l.next);
5 l.next.next = l;
6 l.next=null;
7 }8 return n;
9 }
```

10.答：
```
1 void BST(Tree t) {
2 Queue q = new Queue();
3 q.enque(t);
4 Tree t = q.deque();
5 while(t) {
6 System.out.println(t.value);
7 q.enque(t.left);
8 q.enque(t.right);
9 t = q.deque();
10 }
11 }
```
——————————
```
1class Node {
2 Tree t;
3 Node next;
4 }
5class Queue {
6 Node head;
7 Node tail;
8 public void enque(Tree t){
9 Node n = new Node();
10 n.t = t;
11 if(!tail){
12 tail = head = n;
13 } else {
14 tail.next = n;
15 tail = n;
16 }
```

```
17 }
18 public Tree deque() {
19 if (!head) {
20 return null;
21 } else {
22 Node n = head;
23 head = head.next;
24 return n.t;
25 }
26}
```

11. 答：1char[] p;

```
2void perm(char s[], int i, int n){ 3 int j;
4 char temp;
5 for(j=0;j<N;++J){
6 if(j!=0 && s[j]==s[j-1]);
7 elseif(s[j]!='@'){
8 p[i]=s[j];
9 s[j]='@';
10 if(i==n-1){
11 p[n]='\0';
12 printf("%s", p);
13 }else{
14 perm(s,i+1,n);
15 }
16 s[j]=p[i];
17 }
18 }
19}
```

```
1void main() {
2 char s[N];
3 sort(s);
4 perm(s,0,strlen(s));
5}
```

12.答：1 long atol(char *str){

```
2 char *p = str;
3 long l=1;m=0;
4 if (*p=='-') {
5 l=-1;
6 ++p;
```

```
7 }
8 while(isDigit(*p)){
9 m = m*10 + p;
10 ++p;
11 }
12 if(!p) return m*1;
13 else return error;
14}
```

13.答：
```
1 int isLoop(List l) {
2 if ( ! l) return - 1 ;
3 List s = l.next;
4 while (s && s != l) {
5 s = s.next;
6 }7 if ( ! s) return - 1 ;
8 else reutrn 1 ;
9 }
```

```
1int isLoop(List l){
2 if(!l) return 0;
3 p=l.next;
4 wihle(p!=l&&p!=null) {
5 l.next=l;
6 l=p;p=p.next;
7 }
8 if(p=l) return 1;
9 return 0;
10}
```

14.答：
```
1 void reverse( char * str) {
2 char tmp;
3 int len;
4 len = strlen(str);
5 for ( int i = 0 ;i < len / 2 ; ++ i) {
6 tmp = char [i];
7 str[i] = str[len - i - 1 ];
8 str[len - i - 1 ] = tmp;
9 }
10 }
```

15.答：
```
1int strstr(char[] str, char[] par){
2 int i=0;
3 int j=0;
4 while(str[i] && str[j]){
```

```
5 if(str[i]==par[j]){
6 ++i;
7 ++j;
8 }else{
9 i=i-j+1;
10 j=0;
11 }
12 }
13 if(!str[j]) return i-strlen(par);
14 else return -1;
15}
```

16.答：

```
1int strcmp(char* str1, char* str2){
2 while(*str1 && *str2 && *str1==*str2){
3 ++str1; 4 ++str2;
5 }
6 return *str1-*str2;
7}
```

17.答：

```
1 int f( int x) {
2 int n = 0 ;
3 while (x) {
4 ++ n;
5 x &= x - 1 ;
6 }
7 return n;
8 }
```

18.答：

```
1void tower(n,x,y,z){
2 if(n==1) move(x,z);
3 else {
4 tower(n-1, x,z,y);
5 move(x,z);
6 tower(n-1, y,x,z);
7 }
8}
```

19.答：三柱汉诺塔最小步数。

```
1 int f3(n) {
2 if (f3[n]) return f3[n];
3 else {
```

```
4 if (n == 1 ) {
5 f3[n] = 1 ;
6 return 1 ;
7 }
8 f3[n] = 2 * f3(n - 1 ) + 1 ;
9 return f3[n];
10 }
11 }
```

四柱汉诺塔最小步数。

```
1int f4(n){
2 if(f4[n]==0){
3 if(n==1) {
4 f4[1]==1;
5 return 1;
6 }
7 min=2*f4(1)+f3(n-1);
8 for(int i=2;i<N;++I){ 9 u=2*f4(i)+f3(n-i);
10 if(u 11 }
12 f4[n]=min;
13 return min;
14 } else return f4[n];
15}
```

20.答：

```
1void delete(List m, List n) {
2 if(!m || !n) return;
3 List pre = new List();
4 pre.next=m;
5 List a=m, b=n,head=pre;
6 while(a && b){
7 if(a.value < b.value) {
8 a=a.next;
9 pre=pre.next;
10 }else if(a.value > b.value){
11 b=b.next;
12 }else{
13 a=a.next;
14 pre.next=a;
15 }
16 }
17 m=head.next;
18}
```

21.答：1int hasDuplicate(int[] a, int n){
2 for(int i=0;i<N;++I){
3 while(a[i]!=i && a[i]!=-1){
4 if(a[a[i]]==-1) return 1;
5 a[i]=a[a[i]];
6 a[a[i]]=-1;
7 }
8 if(a[i]==i) {a[i]=-1;}
9 }
10 return 0;
11}

22.答：1int isB(Tree t){
2 if(!t) return 0;
3 int left=isB(t.left);
4 int right=isB(t.right);
5 if(left >=0 && right >=0 && left - right <= 1 || left -right >=-1)
6 return (left
7 else return -1; 8}
9

23.答：1int depth(Tree t){
2 if(!t) return 0;
3 else {
4 int a=depth(t.right);
5 int b=depth(t.left);
6 return (a>b)?(a+1):(b+1);
7 }
8}

24.答：1 List merge(List a, List d) {
2 List a1 = reverse(d);
3 List p = q = new List();
4 while (a && a1) {
5 if (a.value < a1.value) {
6 p.next = a;
7 a = a.next;
8 } else {
9 p.next = a1;
10 a1 = a1.next;
11 }
12 p = p.next;
13 }
14 if (a) p.next = a;
15 elseif(a1) p.next = a1;

```
16 return q.next;
17 }
```

25.答：
```
1char* ltoa(long l){
2 char[N] str;
3 int i=1,n=1;
4 while(!(l/i<10)){i*=10;++n}
5 char* str=(char*)malloc(n*sizeof(char));
6 int j=0;
7 while(l){
8 str[j++]=l/i;
9 l=l%i;
10 i/=10;
11 }
12 return str;
13}
```

26.答：
```
1 if (x == 0) y = a;
2 else y = b; 1 j[] = {a,b};
2 y=j[x];
```

27.答：
```
1void del(List head, List node){
2 List pre=new List();
3 pre.next = head;
4 List cur = head;
5 while(cur && cur!=node){
6 cur=cur.next;
7 pre=pre.next;
8 }
9 if(!cur) return;
10 List post = cur.next;
11 pre.next=cur.next;
12 post.last=cur.last;
13 return;
14}
```

28.答：
```
1 void outputUnique( char [] str, int n) {
2 if (n <= 0 ) return ;
3 elseif(n == 1 ) putchar(str[ 0 ]);
4 else {
5 int i = 0 ,j = 1 ;
6 putchar(str[ 0 ]);
7 while (j < n) {
8 if (str[j] !== str[i]) {
```

```
9 putchar(str[j]);
10 i = j;
11 }
12 ++ j;
13 }
14 }
15 }
```

29.答：判断一个链表是否存在环，例如下面这个链表就存在一个环：
例如：N1->N2->N3->N4->N5->N2 就是一个有环的链表，环的开始结点是 N5 这里有一个比较简单的解法。
设置两个指针 p1，p2。每次循环 p1 向前走一步，p2 向前走两步。直到 p2 碰到 NULL 指针或者两个指针相等结束循环。如果两个指针相等则说明存在环。

```
struct link
{
int data;
link* next;
};bool IsLoop(link* head)
{
link* p1=head, *p2 = head;
if (head ==NULL || head->next ==NULL)
{
return false;
}
do{
p1= p1->next;
p2 = p2->next->next;
} while(p2 && p2->next && p1!=p2);
if(p1 == p2)
return true;
else
return false;
}
```

30.答：单向链表的反转是一个经常被问到的一个面试题，也是一个非常基础的问题。
例如：一个链表是这样的： 1->2->3->4->5 通过反转后成为 5->4->3->2->1。最容易想到的方法遍历一遍链表，利用一个辅助指针，存储遍历过程中当前指针指向的下一个元素，然后将当前节点元素的指针反转后，利用已经存储的指针往后面继续遍历。源代码如下：

```
struct linka {
int data;
```

```
linka* next;
};

void reverse(linka*& head)
{
if(head ==NULL)
return;
linka*pre, *cur, *ne;
pre=head;
cur=head->next;
while(cur)
{
ne = cur->next;
cur->next = pre;
pre = cur;
cur = ne;
}
head->next = NULL;
head = pre;
}
```

还有一种利用递归的方法。这种方法的基本思想是在反转当前节点之前先调用递归函数反转后续节点。源代码如下。不过这个方法有一个缺点就是在反转后的最后一个结点会形成一个环，所以必须将函数的返回的节点的 next 域置为 NULL。因为要改变 head 指针，所以我用了引用。算法的源代码如下：

```
linka* reverse(linka* p,linka*& head)
{
if(p == NULL || p->next == NULL)
{
head=p;
return p;
}
else
{
linka* tmp = reverse(p->next,head);
tmp->next = p;
return p;
}
}
```

31.答：给定两个排好序的数组，怎样高效得判断这两个数组中存在相同的数字？

这个问题首先想到的是一个 O(nlogn)的算法。就是任意挑选一个数组，遍历这个数组的所有元素，遍历过程中，在另一个数组中对第一个数

组中的每个元素进行 binary search。用 C++实现代码如下：

```
bool findcommon(int a[],int size1,int b[],int size2)
{
int i;
for(i=0;i<SIZE1;I++)
{
int start=0,end=size2-1,mid;
while(start<=end)
{
mid=(start+end)/2;
if(a[i]==b[mid])
return true;
else if (a[i]<B[MID])
end=mid-1;
else
start=mid+1;
}
}
return false;
}
```

后来发现有一个 O(n)算法。因为两个数组都是排好序的。所以只要一次遍历就行了。首先设两个下标，分别初始化为两个数组的起始地址，依次向前推进。推进的规则是比较两个数组中的数字，小的那个数组的下标向前推进一步，直到任何一个数组的下标到达数组末尾时，如果这时还没碰到相同的数字，说明数组中没有相同的数字。

```
bool findcommon2(int a[], int size1, int b[], int size2)
{
int i=0,j=0;
while(i<size2)
{
if(a[i]==b[j])
return true;
if(a[i]>b[j])
j++;
if(a[i]<B[J])
i++;
}
return false;
}
```

32.答：给定一整数序列 A1， A2，... An （可能有负数），求 A1~An 的一个子序列 Ai~Aj，使得 Ai 到 Aj 的和最大。

例如：整数序列-2, 11, -4, 13, -5, 2, -5, -3, 12, -9 的最大子序列的和为 21。
对于这个问题，最简单也是最容易想到的那就是穷举所有子序列的方法。利用三重循环，依次求出所有子序列的和然后取最大的那个。当然算法复杂度会达到 O(n^3)。显然这种方法不是最优的，下面给出一个算法复杂度为 O(n)的线性算法实现，算法的来源于 Programming Pearls 一书。 在给出线性算法之前，先来看一个对穷举算法进行优化的算法，它的算法复杂度为 O(n^2)。其实这个算法只是对对穷举算法稍微做了一些修改：其实子序列的和我们并不需要每次都重新计算一遍。假设 Sum(i, j)是 A[i]... A[j]的和，那么 Sum(i, j+1) = Sum(i, j) + A[j+1]。利用这一个递推，我们就可以得到下面这个算法：

```
int max_sub(int a[],int size)
{
int i,j,v,max=a[0];
for(i=0;i<SIZE;I++)
{
v=0;
for(j=i;j<SIZE;J++)
{
v=v+a[j];//Sum(i, j+1) = Sum(i, j) + A[j+1]
if(v>max)
max=v;
}
}
return max;
}
```

那怎样才能达到线性复杂度呢？这里运用动态规划的思想。先看一下源代码实现：

```
int max_sub2(int a[], int size)
{
int i,max=0,temp_sum=0;
for(i=0;i<SIZE;I++)
{
temp_sum+=a[i];
if(temp_sum>max)
max=temp_sum;
else if(temp_sum<0)
temp_sum=0;
}
return max;
}
```

33.答：并不是简单的字符串反转，而是按给定字符串里的单词将字符串倒转过来，就是说字符串里面的单词还是保持原来的顺序，这里的每个单词用空格分开。
例如：Here is www.fishksy.com.cn
经过反转后变为：
www.fishksy.com.cn is Here
如果只是简单的将所有字符串翻转的话，可以遍历字符串，将第一个字符和最后一个交换，第二个和倒数第二个交换，依次循环。其实按照单词反转的话可以在第一遍遍历的基础上，再遍历一遍字符串，对每一个单词再反转一次。这样每个单词又恢复了原来的顺序。

```
char* reverse_word(const char* str)
{
int len = strlen(str);
char* restr = new char[len+1];
strcpy(restr,str);
int i,j;
for(i=0,j=len-1;i<J;I++,J--)
{
char temp=restr[i];
restr[i]=restr[j];
restr[j]=temp;
}
int k=0;
while(k<LEN)
{
i=j=k;
while(restr[j]!=' ' && restr[j]!='' )
j++;
k=j+1;
j--;
for(;i<J;I++,J--){
char temp=restr[i];
restr[i]=restr[j];
restr[j]=temp;
}
}
return restr;
}
```

如果考虑空间和时间的优化的话，当然可以将上面代码里两个字符串交换部分改为异或实现。
例如将

```
char temp=restr[i];
restr[i]=restr[j];
restr[j]=temp;
```

改为
```
restr[i]^=restr[j];
restr[j]^=restr[i];
restr[i]^=restr[j];
```
34.答：一个动态长度可变的数字序列，以数字 0 为结束标志，要求将重复的数字用一个数字代替。
例如：将数组 1,1,1,2,2,2,2,2,7,7,1,5,5,5,0 转变成 1,2,7,1,5,0
问题比较简单，要注意的是这个数组是动态的。所以避免麻烦我还是用了 STL 的 vector。
```
#include
#include
using namespace std;
//remove the duplicated numbers in an intger array, the array was end with 0;
//e.g. 1,1,1,2,2,5,4,4,4,4,1,0 —>1,2,5,4,1,0
void static remove_duplicated(int a[], vector& _st)
{
_st.push_back(a[0]);
for(int i=1;_st[_st.size()-1]!=0;i++)
{
if(a[i-1]!=a[i])
_st.push_back(a[i]);
}
}
```
当然如果可以改变原来的数组的话，可以不用 STL，仅需要指针操作就可以了。下面这个程序将修改原来数组的内容。
```
void static remove_duplicated2(int a[])
{
if(a[0]==0 || a==NULL)
return;
int insert=1,current=1;
while(a[current]!=0) {
if(a[current]!=a[current-1])
{
a[insert]=a[current];
insert++;
current++;
}
else
current++;
}
a[insert]=0;
}
```

35.答：判断一个二叉排序树是否是平衡二叉树 解决方案：根据平衡二叉树的定义，如果任意节点的左右子树的深度相差不超过 1，那这棵树就是平衡二叉树。
首先编写一个计算二叉树深度的函数，利用递归实现。

```
template
static int Depth(BSTreeNode* pbs)
{
if (pbs==NULL)
return 0;
else
{
int ld = Depth(pbs->left);
int rd = Depth(pbs->right);
return 1 + (ld >rd ? ld : rd);
}
}
```

下面是利用递归判断左右子树的深度是否相差 1 来判断是否是平衡二叉树的函数：

```
template
static bool isBalance(BSTreeNode* pbs)
{
if (pbs==NULL)
return true;
int dis = Depth(pbs->left)  -  Depth(pbs->right);
if (dis>1 || dis<-1 )
return false;
else
return isBalance(pbs->left) && isBalance(pbs->right);
}
```

二、SQL

I、单选题

1.C	2.C	3.B	4.D	5.D	6.B
7.A	8.D	9.B	10.A	11.B	12.A
13.A	14.A	15.D	16.C	17.C	18.D
19.C	20.D	21.D	22.B	23.C	24.D
25.C	26.C	27.B	28.A	29.A	30.B
31.C	32.D	33.D	34.C	35. A	36. D
37. F	38.A	39.B	40. A	41. A	42.B
43.A	44.D	45. A	46. B	47. A	48. C_

49. C	50. B	51. A	52. D	53. B	54. D
55. B	56. C	57. A	58. D	59. C	60. B
61. A	62. A	63. B	64. A	65. A	66. B
67. A	68.C	69. C	70. B	71. B	72. A
73. B	74. B	75. C	76. C	77. _B	78. D
79. D	80. A	81. A	82. B	83. C	84. B
85. B	86. D	87. D	88. C	89. B	90. B
91. C	92. C	93. D	94. D	95. C	96. C
97. A	98. D	99. C	100. B	101. C	102. A
103. B	104. B	105. D	106 A	107.C	108. C
109. D	110. C	111. B	112. D	113. A	114. C
115. B	116. B	117. A	118. A	119. A	120. C
121. C	122. B	123. A	124. D	125. C	126. C
127. A	128. A	129. B	130. B	131. D	132. C
133. A	134. A	135. D			

II、不定项选题

1.C	2. B	3. C	4. A	5. B	6. C
7.A	8. C	9.B	10. B	11.BC	12. D
13. D	14. C	15. A	16. C	17.B	18. B
19. B	20. B	21. A	22. A	23. A	24. B
25. B	26. B	27. C	28. C	29. A	30. C
31. B	32. A	33. C	34. B、C	35. ABCD	36. C
37. B	38. D	39. C	40. C	41. ABCD	42. AB
43. B	44. A	45. A	46. B	47. ABCD	48. BD _
49. AD	50. B	51. CD	52. A	53. A	54. B
55. BCD	56. AB	57. C	58. ABC	59. ABC	60. B
61. BD	62. B	63. B	64. ABC	65. B、E	66. A
67. A、E	68 B	69. B	70. C	71. C	72. C
73. C	74. B	75. B	76. C	77. _ C	78. C
79. c	80. a	81. bcd	82. b	83. b	84. c
85. c	86. c	87. a	88. b	89. C	90. B
91. B	92. AC	93. C	94. D	95. A	96. B
97. BC	98. BC				

III.填空题

1. 答案: 1)ID2 in 2) 3) =

select * from T1 A where _ Exists_____ (select * from T2 B where B.ID = A.ID2)

select * from T1 A where A.ID __in____ (select ID from T2 B)

select A.* from T1 A, T2 B where A.ID2 __＝____ B.ID

2. 答案:22k

3. 答案: HAVING COUNT(*)>1

4. 答案: 时间

5. 答案:模式_

6. 答案:黑盒

7. 答案: 一对多_

8. 答案: 投影

9. 答案: 15、into cursor_

10. 答案: _列数

11. 答案: 应用程序生成器

12. 答案: 数据库管理系统、DBMS

13 答案: 联接类型

14. 答案: CREATE TABLE 借阅
(借书证号 C(4),
总编号 C(6),
借书日期 D(8))
或 CREA TABL 借阅
(借书证号 C(4), 总编号 C(6),
借书日期 D(8))

15. 答案: 单价<=20
或 (出版单位=”清华大学出版社” OR 出版单位=”电子工业出版社”)
或 (出版单位=”电子工业出版社” OR 出版单位=”清华大学出版社”)
或 (出版单位=’清华大学出版社’ OR 出版单位=’电子工业出版社’) 与
(出版单位=”清华大学出版社” OR 出版单位=”电子工业出版社”)
或 (出版单位=’清华大学出版社‘) 49. COUNT(DISTINCT 总编号)

或 COUN(DISTINCT 总编号) 或 COUNT(DIST 总编号) 或 COUN(DIST 总编号)

16. 答案：概念

17. 答案：n(n-1)*2

18. 答案：查询

19. 答案：文件位置

20. 答案：字符

21. 答案：逻辑表达式

22. 答案：数据源

23. 答案：内部联接

24. 答案：单价 BETWEEN 15 AND 25
或 单价 BETW 15 AND 25
或 单价 BETWE 15 AND 25
或 单价>=15 and 单价<=25
或 单价>=15 and 单价=<25
或 单价=>15 and 单价<=25
或 单价=>15 and 单价=<25 与 分类号 ASC
或 分类号

25 答案：SELECT * FROM R UNION SELECT * FROM T
或 SELE * FROM R UNIO SELE * FROM T
或 SELECT * FROM R UNIO SELECT * FROM T
或 SELE * FROM R UNION SELE * FROM T

26 答案：INSERT INTO R(NO,NAME,CLASS) VALUES(30,”郑和”,”95031″)
或 INSE INTO R(NO,NAME,CLASS) VALUES(30,”郑和”,”95031″)

27. 答案：DELETE FROM R WHERE NO=20
或 DELE FROM R WHERE NO=20
或 DELE FROM R WHER NO=20
或 DELETE FROM R WHER NO=20

IV、简答题

1. 答案：使用 distinct 关键字

2.：快照 Snapshot 是一个文件系统在特定时间里的镜像，对于在线实时数据备份非常有用。快照对于拥有不能停止的应用或具有常打开文件的文件系统的备份非常重要。对于只能提供一个非常短的备份时间而言，快照能保证系统的完整性。

3. 答案：
存储过程是一组 Transact-SQL 语句，在一次编译后可以执行多次。因为不必重新编译 Transact-SQL 语句，所以执行存储过程可以提高性能。
触发器是一种特殊类型的存储过程，不由用户直接调用。创建触发器时会对其进行定义，以便在对特定表或列作特定类型的数据修改时执行。

4. 答案：支持动态行级锁定
SQL Server 2000 动态地将查询所引用的每一个表的锁定粒度调整到合适的级别。当查询所引用的少数几行分散在一个大型表中时，优化数据并行访问的最佳办法是使用粒度锁，如行锁。但是，如果查询引用的是一个表中的大多数行或所有行，优化数据并行访问的最佳办法可以是锁定整个表，以尽量减少锁定开销并尽快完成查询。
SQL Serve 2000 通过为每个查询中的每个表选择适当的锁定级别，在总体上优化了数据并发访问。对于一个查询，如果只引用一个大型表中的几行，则数据库引擎可以使用行级锁定；如果引用一个大型表的几页中的多行，则使用页级锁定；如果引用一个小型表中的所有行，则使用表级锁定。

5. 答案：每个数据库都有事务日志，用以记录所有事务和每个事务对数据库所做的修改。

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

7. 答案：事务是作为一个逻辑单元执行的一系列操作，一个逻辑工作单元必须有四个属性，称为 ACID（原子性、一致性、隔离性和持久性）属性，只有这样才能成为一个事务：
(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状态。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（如 B 树索引或双向链表）都必须是正确的。
(3) 隔离性
由并发事务所作的修改必须与任何其它并发事务所作的修改隔离。事务查看数据时数据所处的状态，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，事务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。

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

9. 答案：
事前触发器运行于触发事件发生之前，而事后触发器运行于触发事件发生之后。通常事前触发器可以获取事件之前和新的字段值。
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。

10. 答案：
解决手段一：SQL Server 自动检测和消除死锁
解决手段二：设置死锁优先级
解决手段三：设置锁定超时
解决手段四：使用更新锁避免死锁

11. 答案：在给定的系统硬件和系统软件条件下，提高数据库系统的运行效率的办法是：
(1) 在数据库物理设计时，降低范式，增加冗余，少用触发器，多用存储过程。
(2) 当计算非常复杂、而且记录条数非常巨大时(例如一千万条)，复杂计算要先在数据库外面，以文件系统方式用 C++语言计算处理完成之后，最后才入库追加到表中去。这是电信计费系统设计的经验。
(3) 发现某个表的记录太多，例如超过一千万条，则要对该表进行水平分割。水平分割的做法是，以该表主键 PK 的某个值为界线，将该表的记录水平分割为两个表。若发现某个表的字段太多，例如超过八十个，则垂直分割该表，将原来的一个表分解为两个表。
(4) 对数据库管理系统 DBMS 进行系统优化，即优化各种系统参数，如缓冲区个数。
(5) 在使用面向数据的 SQL 语言进行程序设计时，尽量采取优化算法。总之，要提高数据库的运行效率，必须从数据库系统级优化、数据库设计级优化、程序实现级优化，这三个层次上同时下功夫。

12. 答案：通俗地理解三个范式，对于数据库设计大有好处。在数据库设计中，为了更好地应用三个范式，就必须通俗地理解三个范式(通俗地理解是够用的理解，并不是最科学最准确的理解)：
第一范式：1NF 是对属性的原子性约束，要求属性具有原子性，不可再分解；二范式：2NF 是对记录的惟一性约束，要求记录有惟一标识，即实体的惟一性；
第三范式：3NF 是对字段冗余性的约束，即任何字段不能由其他字段派生出来，它要求字段没有冗余。没有冗余的数据库设计可以做到。但是，没有冗余的数据库未

必是最好的数据库，有时为了提高运行效率，就必须降低范式标准，适当保留冗余数据。具体做法是：在概念数据模型设计时遵守第三范式，降低范式标准的工作放到物理数据模型设计时考虑。降低范式就是增加字段，允许冗余。

13. 优点：
1. 更快的执行速度：存储过程只在创造时进行编译，以后每次执行存储过程都不需再重新编译，而一般 SQL 语句每执行一次就编译一次,所以使用存储过程可提高数据库执行速度；
2. 与事务的结合，提供更好的解决方案：当对数据库进行复杂操作时(如对多个表进行 Update、Insert、Query 和 Delete 时），可将此复杂操作用存储过程封装起来与数据库提供的事务处理结合一起使用；
3. 支持代码重用:存储过程可以重复使用,可减少数据库开发人员的工作量;4. 安全性高：可设定只有某此用户才具有对指定存储过程的使用权。

缺点：
1. 如果更改范围大到需要对输入存储过程的参数进行更改，或者要更改由其返回的数据，则您仍需要更新程序集中的代码以添加参数、更新 GetValue() 调用，等等，这时候估计比较繁琐了。
2. 可移植性差由于存储过程将应用程序绑定到 SQL Server，因此使用存储过程封装业务逻辑将限制应用程序的可移植性。如果应用程序的可移植性在您的环境中非常重要,则将业务逻辑封装在不特定于 RDBMS 的中间层中可能是一个更佳的选择。

14 答案：
相同点:它们都属于实体完整性约束。
不同点:
(1) 唯一性约束所在的列允许空值，但是主键约束所在的列不允许空值。
(2) 可以把唯一性约束放在一个或者多个列上，这些列或列的组合必须有唯一的。但是，唯一性约束所在的列并不是表的主键列。
(3) 唯一性约束强制在指定的列上创建一个唯一性索引。在默认情况下，创建唯一性的非聚簇索引，但是，也可以指定所创建的索引是聚簇索引。
(4) 建立主键的目的是让外键来引用。
(5) 一个表最多只有一个主键，但可以有很多唯一键。

15. 作用：加快查询速度。
原则：
(1) 如果某属性或属性组经常出现在查询条件中，考虑为该属性或属性组建立索引；
(2) 如果某个属性常作为最大值和最小值等聚集函数的参数,考虑为该属性建立索引；
(3) 如果某属性经常出现在连接操作的连接条件中,考虑为该属性或属性组建立索引；

16. 数据库设计分为五个阶段：
需求分析：主要是准确收集用户信息需求和处理需求，并对收集的结果进行整理分析，形成需求说明。

概念结构设计：对用户需求进行综合、归纳、抽象，形成一个与与具体的 DBMS 无关概念模型（一般为 ER 模型）。
逻辑结构设计：将概念结构设计的概念模型转化为某个特定的 DBMS 所支持的数据模型，建立数据库逻辑模式，并对其进行优化，同时为各种用户和应用设计外模式。
物理结构设计：为设计好的逻辑模型选择物理结构，包括存储结构和存取方法等，建立数据库物理模式。
实施和维护：实施就是使用 DLL 语言建立数据库模式，将实际数据载入数据库，建立真正的数据库。维护阶段是对运行中的数据库进行评价、调整和修改。

17. 答案：一般我们所说的内存泄漏指的是堆内存的泄漏。堆内存是程序从堆中为其分配的，大小任意的，使用完后要显示释放内存。当应用程序用关键字 new 等创建对象时，就从堆中为它分配一块内存，使用完后程序调用 free 或者 delete 释放该内存，否则就说该内存就不能被使用，我们就说该内存被泄漏了。

18. 答案：基本表是本身独立存在的表，在 SQL 中一个关系就对应一个表。
视图是从一个或几个基本表导出的表。视图本身不独立存储在数据库中，是一个虚表

19. (1) 视图能够简化用户的操作
(2) 视图使用户能以多种角度看待同一数据；
(3) 视图为数据库提供了一定程度的逻辑独立性；
(4) 视图能够对机密数据提供安全保护。

20. 答案：不是。
视图是不实际存储数据的虚表，因此对视图的更新，最终要转换为对基本表的更新。因为有些视图的更新不能惟一有意义地转换成对相应基本表的更新，所以，并不是所有的视图都是可更新的。

21.0 答案：基本表的行列子集视图一般是可更新的。若视图的属性来自集函数、表达式，则该视图肯定是不可以更新的。

22. 答案：尽可能用约束（包括 CHECK、主键、唯一键、外键、非空字段）实现，这种方式的效率最好；其次用触发器，这种方式可以保证无论何种业务系统访问数据库都能维持数据库的完整性、一致性；最后再考虑用自写业务逻辑实现，但这种方式效率最低、编程最复杂，当为下下之策。

23. (1) 视图以及视图中引用的所有表都必须在同一数据库中，并具有同一个所有者
(2) 索引视图无需包含要供优化器使用的查询中引用的所有表。
(3) 必须先为视图创建唯一群集索引，然后才可以创建其它索引。
(4) 创建基表、视图和索引以及修改基表和视图中的数据时，必须正确设置某些 SET 选项（在本文档的后文中讨论）。另外，如果这些 SET 选项正确，查询优化器将不考虑索引视图。
(5) 视图必须使用架构绑定创建，视图中引用的任何用户定义的函数必须使用 SCHEMABINDING 选项创建。

(6) 另外，还要求有一定的磁盘空间来存放由索引视图定义的数据。24. 答案：只要使用特定的输入值集并且数据库具有相同的状态，不管何时调用，始终都能范围相同结果的函数叫确定性函数。几十访问的数据库的状态不变，每次书用特定的输入值都可能范围不同结果的函数叫非确定性函数。

25. 答案：Shared pool(共享池)，DataBase Buffer Cache(数据缓冲区)

Redo Log Buffer(重做日志缓冲区)，Large Pool

26. 答案：字典管理方式和本地管理方式

27. Select 语句

Order by 另外在查询语句中 where 子句中，判断函数尽量不要放在左边

V、编程题

1.答：KEYS:

1. SELECT AVG(SCOST) FROM SOFTWARE WHERE DEVIN = ‘ORACLE’;
2. SELECT PNAME,TRUNC(MONTHS_BETWEEN(SYSDATE,DOB)/12) “AGE”, TRUNC(MONTHS_BETWEEN(SYSDATE,DOJ)/12) “EXPERIENCE” FROM PROGRAMMER;
3. SELECT PNAME FROM STUDIES WHERE COURSE = ‘PGDCA’;
4. SELECT MAX(SOLD) FROM SOFTWARE;
5. SELECT PNAME, DOB FROM PROGRAMMER WHERE DOB LIKE ‘%APR%’;
6. SELECT MIN(CCOST) FROM STUDIES;
7. SELECT COUNT(*) FROM STUDIES WHERE COURSE = ‘DCA’;
8. SELECT SUM(SCOST*SOLD-DCOST) FROM SOFTWARE GROUP BY DEVIN HAVING DEVIN = ‘C’;
9. SELECT * FROM SOFTWARE WHERE PNAME = ‘RAKESH’;
10. SELECT * FROM STUDIES WHERE SPLACE = ‘PENTAFOUR’;
11. SELECT * FROM SOFTWARE WHERE SCOST*SOLD-DCOST > 5000;
12. SELECT CEIL(DCOST/SCOST) FROM SOFTWARE;
13. SELECT * FROM SOFTWARE WHERE SCOST*SOLD >= DCOST;
14. SELECT MAX(SCOST) FROM SOFTWARE GROUP BY DEVIN HAVING DEVIN = ‘VB’;
15. SELECT COUNT(*) FROM SOFTWARE WHERE DEVIN = ‘ORACLE’;
16. SELECT COUNT(*) FROM STUDIES WHERE SPLACE = ‘PRAGATHI’;
17. SELECT COUNT(*) FROM STUDIES WHERE CCOST BETWEEN 10000 AND 15000;

18. SELECT AVG(CCOST) FROM STUDIES;
19. SELECT * FROM PROGRAMMER WHERE PROF1 = 'C' OR PROF2 = 'C';
20. SELECT * FROM PROGRAMMER WHERE PROF1 IN ('C','PASCAL') OR PROF2 IN ('C','PASCAL'); SELECT * FROM PROGRAMMER WHERE PROF1 NOT IN ('C','C++') AND PROF2 NOT IN ('C','C++');
21. SELECT TRUNC(MAX(MONTHS_BETWEEN(SYSDATE,DOB)/12)) FROM PROGRAMMER WHERE SEX = 'M';
22. SELECT TRUNC(AVG(MONTHS_BETWEEN(SYSDATE,DOB)/12)) FROM PROGRAMMER WHERE SEX = 'F';
23. SELECT PNAME, TRUNC(MONTHS_BETWEEN(SYSDATE,DOJ)/12) FROM PROGRAMMER ORDER BY PNAME DESC;
24. SELECT PNAME FROM PROGRAMMER WHERE TO_CHAR(DOB,'MON') = TO_CHAR(SYSDATE,'MON');
25. SELECT COUNT(*) FROM PROGRAMMER WHERE SEX = 'F';
26. SELECT DISTINCT(PROF1) FROM PROGRAMMER WHERE SEX = 'M';
27. SELECT AVG(SAL) FROM PROGRAMMER;
28. SELECT COUNT(*) FROM PROGRAMMER WHERE SAL BETWEEN 5000 AND 7500;
29. SELECT * FROM PROGRAMMER WHERE PROF1 NOT IN ('C','C++','PASCAL') AND PROF2 NOT IN ('C','C++','PASCAL');
30. SELECT PNAME,TITLE,SCOST FROM SOFTWARE WHERE SCOST IN (SELECT MAX(SCOST) FROM SOFTWARE GROUP BY PNAME);

32. SELECT 'Mr.' || PNAME || ' - has ' || TRUNC(MONTHS_BETWEEN(SYSDATE,DOJ)/12) || ' years of experience' "Programmer" FROM PROGRAMMER WHERE SEX = 'M' UNION SELECT 'Ms.' || PNAME || ' - has ' || TRUNC (MONTHS_BETWEEN (SYSDATE,DOJ)/12) || ' years of experience' "Programmer" FROM PROGRAMMER WHERE SEX = 'F';

2.答：KEYS:

1. SELECT DISTINCT(A.ENAME) FROM EMP A, EMP B WHERE A.EMPNO = B.MGR; or SELECT ENAME FROM EMP WHERE EMPNO IN (SELECT MGR FROM EMP);
2. SELECT * FROM EMP WHERE DEPTNO IN (SELECT DEPTNO FROM EMP GROUP BY DEPTNO HAVING COUNT(EMPNO)>10 AND DEPTNO=10);
3. SELECT A.ENAME "EMPLOYEE", B.ENAME "REPORTS TO" FROM EMP A, EMP B WHERE A.MGR=B.EMPNO;
4. SELECT * FROM EMP WHERE EMPNO IN (SELECT EMPNO FROM EMP MINUS SELECT MGR FROM EMP);
5. SELECT * FROM EMP WHERE SAL > (SELECT MIN(SAL) FROM EMP GROUP BY DEPTNO HAVING DEPTNO=20);
6. SELECT * FROM EMP WHERE SAL > (SELECT MAX(SAL) FROM EMP GROUP BY JOB HAVING JOB = 'MANAGER');
7. SELECT JOB, MAX(SAL) FROM EMP GROUP BY JOB;

8. SELECT * FROM EMP WHERE (DEPTNO, HIREDATE) IN (SELECT DEPTNO, MAX(HIREDATE) FROM EMP GROUP BY DEPTNO);
9. SELECT TO_CHAR(HIREDATE,' YYYY') "YEAR", COUNT(EMPNO) "NO. OF EMPLOYEES" FROM EMP GROUP BY TO_CHAR(HIREDATE,'YYYY') HAVING COUNT(EMPNO) = (SELECT MAX(COUNT(EMPNO)) FROM EMP GROUP BY TO_CHAR(HIREDATE,' YYYY'));
10. SELECT DEPTNO, LPAD(SUM(12*(SAL+NVL(COMM,0))),15) "COMPENSATION" FROM EMP GROUP BY DEPTNO HAVING SUM(12*(SAL+NVL(COMM,0))) = (SELECT MAX(SUM(12*(SAL+NVL(COMM,0)))) FROM EMP GROUP BY DEPTNO);
11. SELECT ENAME, HIREDATE, LPAD('*',8) "RECENTLY HIRED" FROM EMP WHERE HIREDATE = (SELECT MAX(HIREDATE) FROM EMP) UNION SELECT ENAME NAME, HIREDATE, LPAD(' ',15) "RECENTLY HIRED" FROM EMP WHERE HIREDATE != (SELECT MAX(HIREDATE) FROM EMP);
12. SELECT ENAME,SAL FROM EMP E WHERE SAL > (SELECT AVG(SAL) FROM EMP F WHERE E.DEPTNO = F.DEPTNO);
13. SELECT ENAME, SAL FROM EMP A WHERE &N = (SELECT COUNT (DISTINCT(SAL)) FROM EMP B WHERE A.SAL<=B.SAL);
14. SELECT * FROM EMP A WHERE A.EMPNO IN (SELECT EMPNO FROM EMP GROUP BY EMPNO HAVING COUNT(EMPNO)>1) AND A.ROWID!=MIN (ROWID));
15. SELECT ENAME "EMPLOYEE",TO_CHAR(TRUNC(MONTHS_BETWEEN(SYSDATE,HIREDATE)/12))||' YEARS '||TO_CHAR(TRUNC(MOD(MONTHS_BETWEEN (SYSDATE, HIREDATE),12)))||' MONTHS ' "LENGTH OF SERVICE" FROM EMP;

3 答：

3.1 答：select a.S#
from (select s#,score from SC where C#=' 001') a,
(select s#,score from SC where C#=' 002') b
where a.score>b.score and a.s#=b.s#;

3.2 答：select S#,avg(score)
from sc
group by S# having avg(score) >60;

3.3 答：select Student.S#,Student.Sname,count(SC.C#),sum(score)
from Student left Outer join SC on Student.S#=SC.S#
group by Student.S#,Sname

3.4 答：select count(distinct(Tname))
from Teacher
where Tname like '李%';

3.5 答：select Student.S#,Student.Sname
from Student
where S# not in (select distinct(SC.S#) from SC,Course,Teacher where SC.C#=Course.C# and Teacher.T#=Course.T# and Teacher.Tname=’叶平’); 3.6 答：select Student.S#,Student.Sname
from Student,SC
where Student.S#=SC.S# and SC.C#=’001′ and exists(Select * from SC as SC_2 where SC_2.S#=SC.S# and SC_2.C#=’002′);

3.7 答：select S#,Sname
from Student
where S# in
(select S#
from SC ,Course ,Teacher
where SC.C#=Course.C# and Teacher.T#=Course.T# and Teacher.Tname=’叶平’ group by S# having count(SC.C#)=(select count(C#) from Course,Teacher where Teacher.T#=Course.T# and Tname=’叶平’));

3.8 答：select S#,Sname
from Student
where S# not in (select Student.S# from Student,SC where S.S#=SC.S# and score>60);

3.9 答：select Student.S#,Student.Sname
from Student,SC
where Student.S#=SC.S#
group by Student.S#,Student.Sname having count(C#) <(select count(C#) from Course);

3.10 答：select S#,Sname
from Student,SC
where Student.S#=SC.S# and C# in （select C# from SC where S#=’1001’）;

3.11 答：Delect SC
from course ,Teacher
where Course.C#=SC.C# and Course.T#= Teacher.T# and Tname=’叶平’;

3.12 答：SELECT L.C# 课程 ID,L.score 最高分,R.score 最低分
FROM SC L ,SC R
WHERE L.C# = R.C#
and
L.score = (SELECT MAX(IL.score)
FROM SC IL,Student IM
WHERE IL.C# = L.C# and IM.S#=IL.S#

```
GROUP BY IL.C#)
and
R.Score = (SELECT MIN(IR.score)
FROM SC IR
WHERE IR.C# = R.C#
GROUP BY IR.C# ); 3.13 答：SELECT 1+(SELECT COUNT( distinct 平均成绩)
FROM (SELECT S#,AVG(score) 平均成绩
FROM SC
GROUP BY S# ) T1
WHERE 平均成绩 > T2.平均成绩) 名次, S# 学生学号,平均成绩
FROM (SELECT S#,AVG(score) 平均成绩 FROM SC GROUP BY S# ) T2
ORDER BY 平均成绩 desc;

3.14 答：SELECT t1.S# as 学生 ID,t1.C# as 课程 ID,Score as 分数
FROM SC t1
WHERE score IN (SELECT TOP 3 score
FROM SC
WHERE t1.C#= C#
ORDER BY score DESC)
ORDER BY t1.C#;

3.15 答：SELECT t1.S# as 学生 ID,t1.C# as 课程 ID,Score as 分数
FROM SC t1
WHERE score IN (SELECT TOP 2 score
FROM SC
WHERE t1.C#= C#
ORDER BY score DESC )
ORDER BY t1.C#;
```

4 答： select Name,Course,Score from T1,T2,T3 where T3.ID=T1.NameID and T1.CourseID=T2.ID

5.答： (T-SQL)：

－利用存储过程查找看本日期和操作类型有没有数据，如果没有则插入数据，如果有则更新数据使操作次数加 1

```
create or replace procedure update_pro(v_DateStr in varchar,v_OprType in integer) as

vv_DateStr T1.DateStr%type;
```

```
vv_OprType T1.OprType%type;

cursor my_cursor is(select DateStr,OprType,OprCount from T1 where DateStr=
v_DateStr and OprType= v_OprType);-声明游标

beginopen my_cursor;

fetch my_cursor into vv_DateStr,vv_OprType;

if my_cursor%notfound then

insert into T1(DateStr,OprType,OprCount) values(v_DateStr,v_OprType,1);

else

update T1 set OprCount=OprCount+1 where DateStr=v_DateStr and
OprType=v_OprType;

end if;

commit;

close my_cursor;

end;
```

6.答： ALTER TABLE T1

ADD CONSTRAINTS CHK_FILED CHECK (Field1 like ‘aaa%’ or Field1 like

‘bbb%’ or Field1 like ‘ccc%’);

7.答：（T-SQL）：此题应用存储过程分批删除并提交，如下是每次删除 10000

```
create or replace procedure Del_pro as

v_Boolean Boolean :=true;

v_count integer :=0;

begin

while v_Boolean loop

delete from t_Log where Add_months(LogDateTime,6)<sysdate and rownum<10000;

commit;
```

```
select count(*) into v_count from t_log   where
Add_months(LogDateTime,6)<sysdate;

if v_count=0 then

v_Boolean=false; end if;

end loop;

end;
```

数据量比较大的情况，可以考虑分批删除，效率会高一些。可使用循环控制语句中，使用 rownum<10000 来分 300 次来删除。注意每次删除后 commit。

8.答：答：1）SQL 语句如下：

```
select stu.sno, stu.sname
    from Student stu
    where (select count(*)
        from sc
        where sno=stu.sno and cno = (select cno
            from Course
            where cname='计算机原理')) != 0;
```

2）SQL 语句如下：

```
select cname
    from Course
    where cno in ( select cno
        from sc
        where sno = (select sno
            from Student
            where sname='周星驰'));
```

3）SQL 语句如下：

```
select stu.sno, stu.sname
    from student stu
    where (select count(*)
        from sc
where sno=stu.sno) = 5;
```

9.答：答：1）建表语句如下（mysql 数据库）：

```
create table s(id integer primary key, name varchar(20));
create table c(id integer primary key, name varchar(20));
create table sc( sid integer references s(id), cid integer references c(id),
primary key(sid,cid) );
```

2）SQL 语句如下：

```
select stu.id, stu.name
    from s stu
    where (select count(*)
```

```
        from sc
        where sid=stu.id) = (select count(*)
            from c);
```
3）SQL 语句如下：
```
select stu.id, stu.name
    from s stu
        where (select count(*)from sc
        where sid=stu.id)>=5;
```
10.答： SQL 语句如下：
```
select employee.name
    from test employee
    where employee.age > (select manager.age
        from test manager
        where manager.id=employee.manager);
```
11.答： SQL 语句为：
```
SELECT C.CITYNO, C.CITYNAME, C.STATENO, S.STATENAME
    FROM CITY C, STATE S
WHERE C.STATENO=S.STATENO(+) ORDER BY(C.CITYNO);
```

三、Java 基础、J2SE

I单选题

1、(B)	2、(B)	3、(B)
4、(A)	5、(C)	6、(B)
7、(C)	8、(A)	9、(E)
10、(C)	11、(B)	12、(C)
13、(B)	14、(B)	15、(D)
16、(C)	17、(A)	18、(D)
19. (A)	20. (C)	21 (.B)
22.(A)		

II 多选题（共 6 题）

1、(CD)	2、(ACD)	3、(CD)
4、(ACD)	5、(AD)	6、(BC)

III 填空题

1. 编码、编译、运行
2. JApplet、Applet、MyApplet.java
3. javac、3、class
4. 2、2

5. 0
6. 1、0
7. 抽象
8. _120
9. Hello! I love JAVA. I love JAVA.
10. S S_
11. userA_
12. super();13. A_B A___
14. 123

IV、 判断题

1、对 2、错 3、对 4. 错 5. 对。因为这样存储的结构为散乱结构。正确定义为 Map map=new HashMap()。
6. 错 7. 对

V、问答题

1. 答：boolean char byte short int long float double
2.答： int double
3. 代表隐含参数的调用、调用本类的其它的构造器
4. 单继承。
5. 多线程 ； 对象 clone
6. final
7. Unicode 2
8. 答：8
9. 覆盖方法返回类型与父类返回类型一样
   覆盖方法的参数类型、次序、方法名称与被覆盖方法一致
   覆盖方法的访问级别不能比被覆盖方法访问级别低
   覆盖方法不能比被覆盖方法抛出的异常多 8
10. 会
11. 运行时异常能够进行编译；一般异常不能进行编译
12. a.抽象性：
    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。
    b.继承性：
    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。

c.封装性：

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。

d. 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，很好的解了应用程序函数同名问题。

13. 答：不是。

基本数据类型包括 byte、int、char、long、float、double、boolean 和 short。

14. 异常表示程序运行过程中可能出现的非正常状态，运行时异常表示虚拟机的通常操作中可能遇到的异常，是一种常见运行错误。java 编译器要求方法必须声明抛出可能发生的非运行时异常，但是并不要求必须声明抛出未被捕获的运行时异常。

15. 答：区别如下：

作用域	当前类	同包	子类	其他
public	√	√	√	√
protected	√	√	√	×
Default	√	√	×	×
private	√	×	×	×

不写时默认为 default。

16. 答：(注：修饰符是影响类、变量及成员方法的生存空间和可访问性的关键字)

修饰符	类	成员方法	成员变量	局部变量
abstract	√	√	—	—
static	—	√	√	—
public	√	√	√	—
protected	—	√	√	—
private	—	√	√	—
synchronized	—	√	—	—
native	—	√	—	—
volatile	—	—	√	—
final	√	√	√	—
transient	—	—	√	√

以下是访问控制修饰符： 默认为 friendly

修饰符	同类	同包	子孙类	不同包
public	√	√	√	√
protected	√	√	√	—

friendly	√	√	—	—
private	√	—	—	—

17. 答：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。

18. 答:不正确；精度不准确,应该用强制类型转换，如下所示：
```
float f=(float)3.4;
```

19. 答：short s1 = 1; s1 =s1 + 1;s1+1 运算结果是 int 型，需要强制转换类型；short s1 =1; s1 += 1;可以正确编译,自动类型提升。

20. 答：goto 是 java 中的保留字，现在没有在 java 中使用。

21. 答：在 Java 中类变量在局部中一定要初始化，因为局部变量会覆盖全局变量，否则会报错：变量未初始化。全局变量则可以不初始化，而到具体的内部方法或其他的类成员中初始化。

22. 答：数组是作为一种对象实现的。数组元素可以包含作何类型值，但数组里面的每个元素的类型必须一致，创建数组步聚如下：
    - 声明
    - 构造
    - 初始化

23. 答：Java 提供两种不同的类型：引用类型和原始类型（或内置类型）；

int 是 java 的原始数据类型，Integer 是 java 为 int 提供的封装类。Java 为每个原始类型提供了封装类：
原始类型：boolean,char,byte,short,int,long,float,double
封装类型：Boolean，Character，Byte，Short，Integer，Long，Float，Double
引用类型和原始类型的行为完全不同，并且它们具有不同的语义。引用类型和原始类型具有不同的特征和用法，它们包括：大小和速度问题，这种类型以哪种类型的数据结构存储，当引用类型和原始类型用作某个类的实例数据时所指定的缺省值。对象引用实例变量的缺省值为 null，而原始类型实例变量的缺省值与它们的类型有关。

24. 答：因为每个类都继承了 Object 类，所以都实现了 toString()方法。
通过 toString()方法可以决定所创建对象的字符串表达形式。

25. 答：Object 类是所有其他的类的超类，Object 的一个变量可以引用任何其他类的对象。因为数组是作为类实现的，所以 Object 的一个变量也可以引用任何数组，它包括以下几种方法：
clone() 、 equals() 、 finalize() 、 getClass() 、 hashCode()、 notify() 、 notifyAll() 、toString() 、 wait()。

26. 答：java 具有以下几个主要特点：
- 简单性
- 面向对象：JAVA 是完全面向对象的，它支持静态和动态风格的代码继承及重用
- 分布式：包括数据分布和操作分布健壮性：java 系统仔细检测对内存的每次访问，确认它是否合法，而且在编译和运行程序时，都要对可能出现的问题进行检查，以消除错误的产生。
- 结构中立
- 安全性：java 不支持指针，一切对内存的访问都必须通过对象的实例变量来实现，这样就防止程序员使用木马等欺骗手段访问对象的私有成员，同时也避免了指针操作中容易产生的错误。
- 与平台无关：java 写的应用程序不用修改就可在不同的软硬平台上运行。平台无关性有两种：源代码级和目标代码级。 Java 主要靠 JAVA 虚拟机在目标代码级上实现平台无关性
- 解释性：运行 JAVA 程序时，它首先被编译成字节代码，字节代码非常类似机器码，执行效率非常高。
- 高性能
- 多线程
- 动态性：它允许程序动态的装入运行时需要的类。

27. 答：&是位运算符，表示按位与运算，&&是逻辑与运算符，表示逻辑与（and）。

28. 答：区别主要有两点：a.条件操作只能操作布尔型的,而逻辑操作不仅可以操作布尔型,而且可以操作数值型 b.逻辑操作不会产生短路。

29. 答：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。

30. 答： Math.round(11.5)==12 Math.round(-11.5)== -11 ，round 方法返回与参数最接近的长整数，参数加 1/2 后求其 floor。

31. 答： switch（expr1）中，expr1 是一个整数表达式。因此传递给 switch 和 case 语句的参数应该是 int、 short、 char 或者 byte。long,String 都不能作用于 switch。

32. 答： 2 << 3。

33. 答：数组没有 length()这个方法，有 length 的属性。String 有 length()这个方法。

34. 答：在最外层循环前加 label 标识,然后用 break:label 方法即可跳出多重循环。

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

36. 答：不对，有相同的 hash code。

37. 答：String 类是 final 类，故不可以继承。

38. A:“beijing”==“beijing”;
B:“beijing”.equalsIgnoreCase（newString（“beijing”)); 答：A 和 B。

39. 答：是值传递。Java 编程语言只有值传递参数。当一个对象实例作为一个参数被传递到方法中时，参数的值就是对该对象的引用。对象的内容可以在被调用的方法中改变，但对象的引用是永远不会改变的。

40. 答：String 类的值在初始后不能改变，如果要改变，可转换为 StringBuffer 类，这个类的值是可以动态改变的。(这里主要考 String 和 StringBuffer 的区别)

41. 答：String 的长度是不可变的；
StringBuffer 的长度是可变的，如果你对字符串中的内容经常进行操作，特别是内容要修改时，那么使用 StringBuffer，如果最后需要 String，那么使用 StringBuffer 的 toString()方法；线程安全；
StringBuilder 是从 JDK 5 开始，为 StringBuffer 该类补充了一个单个线程使用的等价类；通常应该优先使用 StringBuilder 类，因为它支持所有相同的操作，但由于它不执行同步，所以速度更快。

42. 答：方法的重写 Overriding 和重载 Overloading 是 Java 多态性的不同表现。重写 Overriding 是父类与子类之间多态性的一种表现，重载 Overloading 是一个类中多态性的一种表现。如果在子类中定义某方法与其父类有相同的名称和参数，我们说该方法被重写(Overriding)。子类的对象使用这个方法时，将调用子类中的定义，对它而言，父类中的定义如同被“屏蔽
”了。如果在一个类中定义了多个同名的方法，它们或有不同的参数个数或有不同的参数类型，则称为方法的重载(Overloading)。Overloaded 的方法是可以改变返回值的类型。

43. 答：输出结果为：

1） Class A: a=1 d=2.0;
2） Class A: a=1 d=2.0
Class B: a=3.0 d=Java program。

44. 答:JVM 中类的装载是由 ClassLoader 和它的子类来实现的,Java ClassLoader 是一个重要的 Java 运行时系统组件。它负责在运行时查找和装入类文件的类。
45. 答：能够定义成为一个中文的，因为 java 中以 unicode 编码，一个 char 占 2 个字节，所以放一个中文是没问题的。

46. 答：声明方法的存在而不去实现它的类被叫做抽象类（abstract class），它用于要创建一个体现某些基本行为的类，并为该类声明方法，但不能在该类中实现该类的情况。不能创建 abstract 类的实例。然而可以创建一个变量，其类型是一个抽象类，并让它指向具体子类的一个实例。不能有抽象构造函数或抽象静态方法。Abstract 类的子类为它们父类中的所有抽象方法提供实现，否则它们也是抽象类。取而代之，在子类中实现该方法。知道其行为的其它类可以在类中实现这些方法。接口（interface）是抽象类的变体。新型多继承性可通过实现这样的接口而获得。接口中的所有方法都是抽象的，所有成员变量都是 public static final 的。一个类可以实现多个接口，当类实现特殊接口时，它定义（即将程序体给予）所有这种接口的方法。然后，它可以在实现了该接口的类的任何对象上调用接口的方法。由于抽象类，它允许使用接口名作为引用变量的类型。通常的动态联编将生效。引用可以转换到接口类型或从接口类型转换，instanceof 运算符可以用来决定某对象的类是否实现了接口。

47. 答：Static Nested Class 是被声明为静态（static）的内部类，它可以不依赖于外部类实例被实例化。而通常的内部类需要在外部类实例化后才能实例化。

48. 答：会；存在无用但可达的对象，这些对象不能被 GC 回收，导致耗费内存资源。

49. 答：都不能。

50. 答：静态变量也称为类变量，归全类共有，它不依赖于某个对象，可通过类名直接访问；而实例变量必须依存于某一实例，只能通过对象才能访问到它。

51. 答：不可以,如果其中包含对象的 method()，不能保证对象初始化。

52. 答：Clone 有缺省行为：super.clone()，他负责产生正确大小的空间，并逐位复制。

53. 答：GC 是垃圾收集的意思（Gabage Collection）,内存处理是编程人员容易出现问题的地方，忘记或者错误的内存回收会导致程序或系统的不稳定甚至崩溃，Java 提供的 GC 功能可以自动监测对象是否超过作用域从而达到自动回收内存的目的，Java 语言没有提供释放已分配内存的显示操作方法。Java 程序员不用担心内存管理，因为垃圾收集器会自动进行管理。要请求垃圾收集，可以调用下面的方法之

一：System.gc()或 Runtime.getRuntime().gc()。

54. 答：Java 语言中一个显着的特点就是引入了垃圾回收机制，使 c++程序员最头疼的内存管理的问题迎刃而解，它使得 Java 程序员在编写程序的时候不再需要考虑内存管理。由于有个垃圾回收机制，Java 中的对象不再有“作用域”的概念，只有对象的引用才有“作用域”。垃圾回收可以有效的防止内存泄露，有效的使用可以使用的内存。垃圾回收器通常是作为一个单独的低级别的线程运行，不可预知的情况下对内存堆中已经死亡的或者长时间没有使用的对象进行清除和回收，程序员不能实时的调用垃圾回收器对某个对象或所有对象进行垃圾回收。回收机制有分代复制垃圾回收和标记垃圾回收，增量垃圾回收。

55. 答：对于 GC 来说，当程序员创建对象时，GC 就开始监控这个对象的地址、大小以及使用情况。通常，GC 采用有向图的方式记录和管理堆(heap)中的所有对象。通过这种方式确定哪些对象是"可达的"，哪些对象是"不可达的"。当 GC 确定一些对象为"不可达"时，GC 就有责任回收这些内存空间。程序员可以手动执行 System.gc()，通知 GC 运行，但是 Java 语言规范并不保证 GC 一定会执行。

56. 答：两个对象，一个是"xyz",一个是指向"xyz"的引用对象。

57. 答：接口可以继承接口。抽象类可以实现(implements)接口，抽象类可继承实体类，但前提是实体类必须有明确的构造函数。
58. 答：都不能。

59. 答：由于 Java 不支持多继承，而有可能某个类或对象要使用分别在几个类或对象里面的方法或属性，现有的单继承机制就不能满足要求。与继承相比，接口有更高的灵活性，因为接口中没有任何实现代码。当一个类实现了接口以后，该类要实现接口里面所有的方法和属性，并且接口里面的属性在默认状态下面都是 public static,所有方法默认情况下是 public.一个类可以实现多个接口。

60. 答：可以；必须只有一个类名与文件名相同。

61. 答：常用的类：BufferedReader， BufferedWriter，File， FileReader，FileWirter，String，Integer，Math，Socket，ArrayList，HashSet，HashMap；常用的包：java.lang， java.awt， java.io， java.util，java.sql， java.net；常用的接口：Runnable ， Remote，List，Set，Map，Document，NodeList ；

62. 答：可以继承其他类或实现其他接口，在 swing 编程中常用此方式。

63. 答：一个内部类对象可以访问创建它的外部类对象的内容。

64. 答：方法的覆盖 Overriding 和重载 Overloading 是 java 多态性的不同表现；覆盖 Overriding 是父类与子类之间多态性的一种表现，重载 Overloading 是一个类中多态性的一种表现。

65. 答：表示该类不能被继承，是顶级类。

66. 1）java.lang.Thread (T)
2）java.lang.Number(T)
3）java.lang.Double(F)
4）java.lang.Math(F)
5）java.lang.Void(F)
6）java.lang.Class(F)
7）java.lang.ClassLoader(T)
答：1、2、7 可以被继承。

67. 答：输出结果为 1a2b2b；类的 static 代码段,可以看作是类首次加载(虚拟机加载)执行的代码,而对于类加载,首先要执行其基类的构造,再执行其本身的构造。

68. 答：输出结果为：
FatherClass Create
FatherClass Create
ChildClass Create

69. 答：
InterClass
Create
OuterClass
Create
70. 答：答案为
C、E；说明如下：
1）静态内部类可以有静态成员，而非静态内部类则不能有静态成员；故 A、B 错；
2）静态内部类的非静态成员可以访问外部类的静态变量，而不可访问外部类的非静态变量；故 D 错；
3）非静态内部类的非静态成员可以访问外部类的非静态变量；故 C 正确。

71. 答：1)调用数值类型相应包装类中的方法 parse***(String)或 valueOf(String)即可返回相应基本类型或包装类型数值；
2)将数字与空字符串相加即可获得其所对应的字符串;另外对于基本类型数字还可调用 String 类中的 valueOf(…)方法返回相应字符串,而对于包装类型数字则可调用其 toString()方法获得相应字符串；
3)可用该数字构造一 java.math.BigDecimal 对象,再利用其 round()方法进行四舍五入到保留小数点后两位,再将其转换为字符串截取最后两位。

72. 答：可用字符串构造一 StringBuffer 对象,然后调用 StringBuffer 中的 reverse 方法即可实现字符串的反转,调用 replace 方法即可实现字符串的替换。

73. 答：示例代码如下:
String s1 =″你好″;

```
String s2 = new String(s1.getBytes("GB2312"), "ISO8859-1");
```

74. 答:1)创建 java.util.Calendar 实例(Calendar.getInstance()),调用其 get()方法传入不同的参数即可获得参数所对应的值,如:

```
calendar.get(Calendar.YEAR);//获得年
```

2)以下方法均可获得该毫秒数:

```
Calendar.getInstance().getTimeInMillis();
System.currentTimeMillis();
```

3)示例代码如下:

```
Calendar time = Calendar.getInstance();
time.set(Calendar.DAY_OF_MONTH,
time.getActualMaximum(Calendar.DAY_OF_MONTH));
```

4)利用 java.text.DataFormat 类中的 format()方法可将日期格式化。

75. 答：assertion(断言)在软件开发中是一种常用的调试方式，很多开发语言中都支持这种机制。一般来说，assertion 用于保证程序最基本、关键的正确性。assertion 检查通常在开发和测试时开启。为了提高性能，在软件发布后，assertion 检查通常是关闭的。在实现中，断言是一个包含布尔表达式的语句，在执行这个语句时假定该表达式为 true；如果表达式计算为 false，那么系统会报告一个 Assertionerror。断言用于调试目的：

```
assert(a > 0);
// throws an Assertionerror if a <= 0 断言可以有两种形式：
assert Expression1 ;
assert Expression1 : Expression2 ;
```

Expression1 应该总是产生一个布尔值。

Expression2 可以是得出一个值的任意表达式；这个值用于生成显示更多调试信的 String 消息。断言在默认情况下是禁用的，要在编译时启用断言，需使用 source1.4 标记：

```
javac -source 1.4 Test.java
```

要在运行时启用断言，可使用-enableassertions 或者-ea 标记。

要在运行时选择禁用断言，可使用 -da 或者 -disableassertions 标记。

要在系统类中启用断言，可使用 -esa 或者 -dsa 标记。还可以在包的基础上启用或者禁用断言。可以在预计正常情况下不会到达的任何位置上放置断言。断言可以用于验证传递给私有方法的参数。不过，断言不应该用于验证传递给公有方法的参数，因为不管是否启用了断言，公有方法都必须检查其参数。不过，既可以在公有方法中，也可以在非公有方法中利用断言测试后置条件。另外，断言不应该以任何方式改变程序的状态。

76. 答：当 JAVA 程序违反了 JAVA 的语义规则时，JAVA 虚拟机就会将发生的错误表示为一个异常。违反语义规则包括 2 种情况。一种是 JAVA 类库内置的语义检查。例如数组下标越界,会引发 IndexOutOfBoundsException;访问 null 的对象时会引发 NullPointerException。另一种情况就是 JAVA 允许程序员扩展这种语义检查，程序员可以创建自己的异常，并自由选择在何时用 throw 关键字引发异常。所有的异常都是 java.lang.Thowable 的子类。

77. 答：error 表示系统级的错误和程序不必处理的异常，是恢复不是不可能但很困难的情况下的一种严重问题；比如内存溢出，不可能指望程序能处理这样的情况；exception 表示需要捕捉或者需要程序进行处理的异常，是一种设计或实现问题；也就是说，它表示如果程序运行正常，从不会发生的情况。

78. 答：会执行，在 return 前执行。

79. 答：Java 通过面向对象的方法进行异常处理，把各种不同的异常进行分类，并提供了良好的接口。在 Java 中，每个异常都是一个对象，它是 Throwable 类或其它子类的实例。当一个方法出现异常后便抛出一个异常对象，该对象中包含有异常信息，调用这个对象的方法可以捕获到这个异常并进行处理。Java 的异常处理是通过 5 个关键词来实现的：try、catch、throw、throws 和 finally。一般情况下是用 try 来执行一段程序，如果出现异常，系统会抛出（throws）一个异常，这时候你可以通过它的类型来捕捉（catch）它，或最后（finally）由缺省处理器来处理；
try 用来指定一块预防所有“异常”的程序；
catch 子句紧跟在 try 块后面，用来指定你想要捕捉的“异常”的类型；
throw 语句用来明确地抛出一个“异常”；
throws 用来标明一个成员函数可能抛出的各种“异常”；
Finally 为确保一段代码不管发生什么“异常”都被执行一段代码；
可以在一个成员函数调用的外面写一个 try 语句，在这个成员函数内部写另一个 try 语句保护其他代码。每当遇到一个 try 语句，“异常”的框架就放到堆栈上面，直到所有的 try 语句都完成。如果下一级的 try 语句没有对某种“异常”进行处理，堆栈就会展开，直到遇到有处理这种“异常”的 try 语句。

80. 答：异常表示程序运行过程中可能出现的非正常状态，运行时异常表示虚拟机的通常操作中可能遇到的异常，是一种常见运行错误。java 编译器要求方法必须声明抛出可能发生的非运行时异常，但是并不要求必须声明抛出未被捕获的运行时异常。

81. 答：

ArithmeticException,
ArrayStoreException,
BufferOverflowException,
BufferUnderflowException,
CannotRedoException,
CannotUndoException,
ClassCastException,
CMMException,
ConcurrentModificationException,
DOMException,
EmptyStackException,
IllegalArgumentException,
IllegalMonitorStateException,
IllegalPathStateException,

IllegalStateException,
ImagingOpException,
IndexOutOfBoundsException,
MissingResourceException,
NegativeArraySizeException,
NoSuchElementException,
NullPointerException,
ProfileDataException,
ProviderException,
RasterFormatException,
SecurityException,
SystemException,
UndeclaredThrowableException,
UnmodifiableSetException,
UnsupportedOperationException

82. 答：输出为 A。

83. 答：Collection FrameWork 如下：
Collection
├List
│ ├LinkedList
│ ├ArrayList
│ └Vector
│
└Stack
└Set
Map
├Hashtable
├HashMap
└WeakHashMap
.....

Collection 是最基本的集合接口，一个 Collection 代表一组 Object，即 Collection 的元素（Elements）；Map 提供 key 到 value 的映射。

84. 答：List,Set 是；Map 不是。

85. 答：最常用的集合类是 List 和 Map。List 的具体实现包括 ArrayList 和 Vector，它们是可变大小的列表，比较适合构建、存储和操作任何类型对象的元素列表。List 适用于按数值索引访问元素的情形。Map 提供了一个更通用的元素存储方法。Map 集合类用于存储元素对（称作“键”和“值”），其中每个键映射到一个值。

86. 答：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。

87. 答：二者都实现了 Map 接口，是将唯一键映射到特定的值上；主要区别在于：
1)HashMap 没有排序，允许一个 null 键和多个 null 值,而 Hashtable 不允许；
2)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；
3)Hashtable 继承自 Dictionary 类，HashMap 是 Java1.2 引进的 Map 接口的实现；
4)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 算法大致一样，所以性能不会有很大的差异。

88. 答：接口有以下优点：

- 接口只是一个框架而没有实现,因此在接口定义时不需要考虑接口中的方法如何实现。
- 利用接口可达到实现多继承的目地。
- 可以在不暴露对象的类的前提下，暴露对象的编程接口。
- 不用强迫类关系在无关类中截获相似处(采用适配器就可以了)。
- 声明想执行的一个或多个方法。

89. 答：就 ArrayList 与 Vector 主要从二方面来说：
1）安全、效率方面：如果要实现同步安全，则要用 Vector，否则则用 ArrayList，因为 ArrayList 不考虑同步安全的问题，所以效率要高些。
2）资源方面：当两者的容量已满时，它们都会自动增长其容量，但 Vector 是按其容量的一倍增长,而 ArrayList 则按其容量的 50%增加,所以 Vector 更能节省资源。

90. 答：List 以特定次序来持有元素，可有重复元素。Set 无法拥有重复元素,内部排序。Map 保存 key-value 值，value 可多值。

91. 答：两者主要区别如下：

List 用来处理序列，而 Set 用来处理集。
List 中的内容可以重复，而 Set 则不行。

91. 答：Set 里的元素是不能重复的，用 equals()方法来区分重复与否。覆盖 equals()方法用来判断对象的内容是否相同，而”==”判断地址是否相等,用来决定引用值是否指向同一对象。

92. 答：Set 里的元素是不能重复的，用 equals()方法来区分重复与否。覆盖 equals()方法用来判断对象的内容是否相同，而”==”判断地址是否相等,用来决定引用值是否指向同一对象。

93. 答：sleep 是线程类（Thread）的方法，导致此线程暂停执行指定时间，给执行机会给其他线程，但是监控状态依然保持，到时后会自动恢复。调用 sleep 不会释放对象锁。wait 是 Object 类的方法，对此对象调用 wait 方法导致本线程放弃对象锁,进入等待此对象的等待锁定池,只有针对此对象发出 notify 方法(或 notifyAll)后本线程才进入对象锁定池准备获得对象锁进入运行状态。

94. 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

95. 答：其它线程只能访问该对象的其它非同步方法，同步方法则不能进入。

96. 答：wait():使一个线程处于等待状态，并且释放所持有的对象的 lock；
sleep():使一个正在运行的线程处于睡眠状态，是一个静态方法，调用此方法要捕捉 InterruptedException 异常；
notify():唤醒一个处于等待状态的线程，注意的是在调用此方法的时候，并不能确切的唤醒某一个等待状态的线程，而是由 JVM 确定唤醒哪个线程，而且不是按优先级；
notityAll():唤醒所有处入等待状态的线程，注意并不是给所有唤醒线程一个对象的锁，而是让它们竞争。

97. 答:如果数据将在线程间共享。例如正在写的数据以后可能被另一个线程读到，或者正在读的数据可能已经被另一个线程写过了，那么这些数据就是共享数据，必须进行同步存取。当应用程序在对象上调用了一个需要花费很长时间来执行的方法，并且不希望让程序等待方法的返回时，就应该使用异步编程，在很多情况下采用异步途径往往更有效率。

98. 答：可分两方面来答：

- 相对于单线程而言 ：
  可以响应多任务的并发操作。
  多线程取消了主循环和轮流检测机制，一个线程可以暂停而不阻止系统其他的部分的执行，而且当程序中一个线程阻塞时，只有那个被阻塞的线程暂停，所有其他的线程继续执行。
- 相对于进程而言：(可以答也可以不答)

它所要求的开销比较小，转换成本较小。
所有线程共享同一地址空间，相互协作。
彼此之间通信很容易。

99. 答：启动一个线程是调用 start()方法，使线程所代表的虚拟处理机处于可运行状态，这意味着它可以由 JVM 调度并执行。这并不意味着线程就会立即运行。run()方法可以产生必须退出的标志来停止一个线程。

100. 答：线程指在程序执行过程中，能够执行程序代码的一个执行单位，每个程序至少都有一个线程，也就是程序本身；
Java 中的线程有四种状态分别是：运行、就绪、挂起、结束。
101 答：主要相同点：Lock 能完成 synchronized 所实现的所有功能；
主要不同点：Lock 有比 synchronized 更精确的线程语义和更好的性能。
synchronized 会自动释放锁，而 Lock 一定要求程序员手工释放，并且必须在 finally 从句中释放。

102. 答：有两种实现方法，分别是继承 Thread 类与实现 Runnable 接口；
用 synchronized 关键字修饰同步方法；
反对使用 stop()，是因为它不安全。它会解除由线程获取的所有锁定，而且如果对象处于一种不连贯状态，那么其他线程能在那种状态下检查和修改它们。结果很难检查出真正的问题所在；
suspend()方法容易发生死锁。调用 suspend()的时候，目标线程会停下来，但却仍然持有在这之前获得的锁定。此时，其他任何线程都不能访问锁定的资源，除非被“挂起”的线程恢复运行。对任何线程来说，如果它们想恢复目标线程，同时又试图使用任何一个锁定的资源，就会造成死锁。故不应该使用 suspend()，而应在自己的 Thread 类中置入一个标志，指出线程应该活动还是挂起。若标志指出线程应该挂起，便用 wait()命其进入等待状态。若标志指出线程应当恢复，则用一个 notify()重新启动线程。

103. 答：此方法已过时。该方法具有固有的不安全性。用 Thread.stop 来终止线程将释放它已经锁定的所有监视器（作为沿堆栈向上传播的未检查 ThreadDeath 异常的一个自然后果）。如果以前受这些监视器保护的任何对象都处于一种不一致的状态，则损坏的对象将对其他线程可见，这有可能导致任意的行为。stop 的许多使用都应由只修改某些变量以指示目标线程应该停止运行的代码来取代。目标线程应定期检查该变量，并且如果该变量指示它要停止运行，则从其运行方法依次返回。如果目标线程等待很长时间（例如基于一个条件变量），则应使用 interrupt 方法来中断该等待。

104. 答：序列化就是一种用来处理对象流的机制，所谓对象流也就是将对象的内容进行流化。可以对流化后的对象进行读写操作，也可将流化后的对象传输于网络之间。序列化是为了解决在对对象流进行读写操作时所引发的问题；
序列化的实现：将需要被序列化的类实现 Serializable 接口，该接口没有需实现的方法，implements Serializable 只是为了标注该对象是可被序列化的，然后使用一个输出流(如 FileOutputStream)来构造一个 ObjectOutputStream(对象流)对象，接着，

使用
ObjectOutputStream 对象的 writeObject(Object obj)方法就可以将参数为 obj 的对象写出(即保存其状态)，要恢复的话则用输入流。

105. 答：字节流，字符流。字节流继承于 InputStream、OutputStream，字符流继承于 Reader、Writer。在 java.io 包中还有许多其他的流，主要是为了提高性能和使用方便。

106. 具有关键字 abstract ，在实现内容上没有完全定义的类（即包含一个或多个方法抽象方法的类）叫做抽象类。

抽象类和接口的区别如下：

错误！未找到引用源。抽象类含有一个或多个抽象方法,接口只含抽象方法。

在类来继承抽象类时，只需实现部分具体方法和全部抽象方法，而实现接口则要实现里面的全部方法。

错误！未找到引用源。在接口中无成员变量，而抽象类中可有成员变量。

在 Java 中引进接口主要是为了解决多继承的问题。

107. 答：

Ex,no-args

Fx,no-args

Fx,int

108. 答：1
2
5
1
2
5
1
2
5
1
2
5
1
2
5
1
2
5

109.
答：
2
1
6

2
10
3
14
4
18
5

110. 答：
my favorateandy
hellomy favorate

111. 答案：错。abstract method 必须以分号结尾，且不带花括号。

112. 答案：错。局部变量前不能放置任何访问修饰符 (private，public，和 protected)。final 可以用来修饰局部变量
(final 如同 abstract 和 strictfp，都是非访问修饰符，strictfp 只能修饰 class 和 method 而非 variable)。

113. 答案：错。abstract 的 methods 不能以 private 修饰。abstract 的 methods 就是让子类 implement(实现)具体细节的，怎么可以用 private 把 abstract
method 封锁起来呢? (同理，abstract method 前不能加 final)。

114. 答案：错。int x 被修饰成 final，意味着 x 不能在 addOne method 中被修改。

115. 答案：正确。在 addOne method 中，参数 o 被修饰成 final。如果在 addOne method 里我们修改了 o 的 reference
(比如：o = new Other();)，那么如同上例这题也是错的。但这里修改的是 o 的 member vairable
(成员变量)，而 o 的 reference 并没有改变。
116. 答案：正确。输出的是"i = 0"。int i 属于 instant variable (实例变量，或叫成员变量)。instant variable 有 default value。int 的 default value 是 0。

117. 答案：错。final int i 是个 final 的 instant variable (实例变量，或叫成员变量)。final 的 instant variable 没有 default value，必须在 constructor (构造器)结束之前被赋予一个明确的值。可以修改为"final int i = 0;"。

118. 答案：错。看上去在 main 里 call doSomething 没有什么问题，毕竟两个 methods 都在同一个 class 里。但仔细看，main 是 static 的。static method 不能直接 call non-static methods。可改成"System.out.println("s.doSomething() returns " + s.doSomething());"。同理，static method 不能访问 non-static instant variable。

119. 答案：正确。从来没有人说过 Java 的 Class 名字必须和其文件名相同。但 public class 的名字必须和文件名相同。

120. 答案：错误。在编译时会发生错误(错误描述不同的 JVM 有不同的信息，意思就是未明确的 x 调用，两个 x 都匹配（就像在同时 import java.util 和 java.sql 两个包时直接声明 Date 一样）。对于父类的变量,可以用 super.x 来明确，而接口的属性默认隐含为 public static final.所以可以通过 A.x 来明确。

121. 答案: 错。"interface Rollable extends Playable, Bounceable"没有问题。interface 可继承多个 interface,所以这里没错。问题出在 interface Rollable 里的"Ball ball = new Ball("PingPang");"。任何在 interface 里声明的 interface variable (接口变量，也可称成员变量)，默认为 public static final。也就是说"Ball ball = new Ball("PingPang");"实际上是"public static final Ball ball = new Ball("PingPang");"。在 Ball 类的 Play()方法中，"ball = new Ball("Football");"改变了 ball 的 reference，而这里的 ball 来自 Rollable interface，Rollable interface 里的 ball 是 public static final 的，final 的 object 是不能被改变 reference 的。因此编译器将在"ball = new Ball("Football");"这里显示有错。

122. 答：

```
WindX.play(NoteX n)
InstrumentX.play()
```

123.

答：

```
static Insect.x1 initialized
static Beetle.x2 initialized
Beetle constructor
Tags(1)
Tags(2)
i = 9, j = 0
Tags(3)
Beetle.k initialized
k = 63
j = 39
```

124. 答：

```
str = [1,2,3,4,5,6]
After increase:
str = [2,3,4,5,6,7]
```

125. 答：第 11 行有空指针异常

```
if(nullStr.length() == 0){//这行
```

126. 答：clone 方法中只定义了浅克龙,没有把原始对象(这里是一个链表)完全复制下来,s2 的每个元素都是 s 的对应元素的引用（除了第一个）。

127. 答：不能，gc 线程是被虚拟机调度的，在对象没有被引用的时候按调度机制进行收集

的。

128. 答：251346

129. 答：

```
int i=5,j=6;
 i=i+j;//i=11
 j=i-j;//j=11-6=5
 i=i-j;//i=11-5=6
 或者
 i=i^j;
 j=i^j;
 i=i^j;
```

130. 答：队列的取出和放入方法必须是 synchronized,取出和放入方法内必须使用 wait 、notify 机制进行多线程同步。

131. 答：反射机制/reflection

132. 答：

```
s=:a:b:c:d:e
s2=:a:b:c:d:e
after s.increment
s=:b:c:d:e:f
s2=:a:c:d:e:f
```

133. 答：

```
父类 3
子类 5
33
父类构造器
55
子类构造器
```

VI、编程题

1. 答：函数代码如下：

```
public String[] split(String str, int chars){
    int n = (str.length()+ chars -1)/chars;
    String ret[] = new String[n];
    for(int i=0; i<n; i++){
        if(i < n-1){
            ret[i] = str.substring(i*chars , (i+1)*chars);
        }else{
```

```
            ret[i] = str.substring(i*chars);
        }
    }
    return ret;
```

2. 答：代码如下：

```
public String subString(String str, int subBytes) {
    int bytes =0;
    //用来存储字符串的总字节数
    for (int i = 0; i<str.length(); i++) {
        if (bytes == subBytes){
            return str.substring(0, i);
        }
        char c = str.charAt(i);
        if (c < 256) {
            bytes += 1; //英文字符的字节数看作 1
        } else {
            bytes += 2; //中文字符的字节数看作 2
            if(bytes -subBytes == 1){
                return str.substring(0, i);
            }
        }
    }
    return str;
}
```

3. 答：

```
public class YesterdayCurrent{
    public static void main(String[] args){
        Calendar cal = Calendar.getInstance();
        cal.add(Calendar.DATE, -1);
        System.out.println(cal.getTime());
    }
}
```

4. 答：注意 private static 关键字。

```
public class Singleton{
    private static final Singleton s=new Singleton();
    private Singleton(){}
    public static Singleton getInstance(){
        return s;
    }
}
```

5. 答：代码如下：

```
package test;
```

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;
import java.util.Random;
public class RandomSort {
    public static void printRandomBySort() {
        Random random = new Random();
        //创建随机数生成器
        List list = new ArrayList();
        //生成 10 个随机数，并放在集合 list 中
        for (int i = 0; i < 10; i++) {
            list.add(random.nextInt(1000));
        }
        Collections.sort(list);
        //对集合中的元素进行排序
        Iterator it = list.iterator();
        int count = 0;
        while (it.hasNext()) {
        //顺序输出排序后集合中的元素
            System.out.println(++count + ": " + it.next());
        }
    }
    public static void main(String[] args) {
        printRandomBySort();
    }
}
```

6. 答：用插入法进行排序代码如下：

```
package test;
import java.util.*;
class InsertSort {
    ArrayList al;
    public InsertSort(int num,int mod) {
        al = new ArrayList(num);
        Random rand = new Random();
        System.out.println("The ArrayList Sort Before:");
        for (int i=0;i<num ;i++ ){
            al.add(new Integer(Math.abs(rand.nextInt()) % mod + 1));
            System.out.println("al["+i+"]="+al.get(i));
        }
    }
```

```
    public void SortIt(){
        tempInt;
        int MaxSize=1;
        for(int i=1;i<al.size();i++){
            tempInt = (Integer)al.remove(i);
            if(tempInt.intValue()                                    >=
        ((Integer)al.get(MaxSize-1)).intValue()){
                al.add(MaxSize,tempInt);
                MaxSize++;
                System.out.println(al.toString());
            }else{
                for (int j=0;j<MaxSize ;j++ ){
                    if(((Integer)al.get(j)).intValue() >=tempInt.intVal
                ue()){
                        al.add(j,tempInt);
                        MaxSize++;
                        System.out.println(al.toString());
                        break;
                    }
                }
            }
        }
        System.out.println("The ArrayList Sort After:");
        for(int i=0;i<al.size();i++){
            System.out.println("al["+i+"]="+al.get(i));
        }
    }
    public static void main(String[] args){
        InsertSort is = new InsertSort(10,100);
        is.SortIt();
    }
}
```

JAVA 类实现序例化的方法是实现 java.io.Serializable 接口；
Collection 框架中实现比较要实现 Comparable 接口和 Comparator 接口。

7. 答：代码如下：

```
packagetest;
public class TestThread {
    private int j;
    public TestThread(int j){
        this.j = j;
    }
    private synchronized void inc(){
```

```
        j++;
        System.out.println(j + "--Inc--" +
    Thread.currentThread().getName());
    }
    private synchronized void dec(){
        j--;
        System.out.println(j + "--Dec--"+
    Thread.currentThread().getName());
    }
    public void run(){
        (new Dec()).start();
        new Thread(new Inc()).start();
        (new Dec()).start();
        new Thread(newInc()).start();
    }
class Dec extends Thread {
    public void run(){
        for(int i=0; i<100; i++){
            dec();
        }
    }
}
class Inc implements Runnable{
    public void run(){
        for(int i=0; i<100; i++){
            inc();
        }
    }
}
    public static void main(String[] args){
        (new TestThread(5)).run();
    }
}
```

8. 答：

```
public String translate(String str){
    String tempStr = "";
    try{
        tempStr = new String(str.getBytes("ISO8859-1"), "GBK");
        tempStr = tempStr.trim();
    }catch(Exception e){
        System.err.println(e.getMessage());
    }
    return tempStr;
```

```
}
```

9. 答：
```
import java.io.*;
    public  class  test{
        public static void main(String args[]){
            String A="" ;
            String B="" ;
            String C="" ;
            Try{
               BufferedReader          br=new          BufferedReader(new
InputStreamReader(System.in));
               If ((B=br.readLine())!=null){
                  A=A+B;
               }

               for(int I=str.length()-1;I>=0;I--){
                  C=C+A.substring(I,I+1);
                }
                System.out.println(C);
            }
            catch(Exception e){
                System.out.println(e.getMessage());
            }
        }
    }
```

10. 答：1)示例代码如下:
```
File file = newFile("e:\\总结");
File[] files = file.listFiles();
for(int i=0; i<files.length; i++){
    if(files[i].isFile())
        System.out.println(files[i]);
}
```
2)示例代码如下:
```
File file = newFile("e:\\总结");
File[] files = file.listFiles();
for(int i=0; i<files.length; i++){
    if(files[i].isDirectory())
        System.out.println(files[i]);
}
```
3)创建 File 对象,调用其 exsit()方法即可返回是否存在,如:
```
System.out.println(new File("d:\\t.txt").exists());
```

4)示例代码如下:

```
//读文件:
FileInputStream fin = new FileInputStream("e:\\tt.txt");
byte[] bs = new byte[100];
while(true){ int len = fin.read(bs);
    if(len <= 0)
        break;
    System.out.print(new String(bs,0,len));
}
fin.close();
//写文件:
FileWriter fw = new FileWriter("e:\\test.txt");
fw.write("hello world!" + System.getProperty("line.separator"));
fw.write("你好！北京！");
fw.close();
```

11. 答：代码如下：

```
public int countWords(String file, String find) throws Exception {
    int count = 0;
    Reader in = new FileReader(file);
    int c;
    while ((c = in.read()) != -1) {
        while (c == find.charAt(0)) {
            for (int i = 1; i < find.length(); i++) {
                c = in.read();
                if (c != find.charAt(i))
                    break;
                if (i == find.length() -1)
                    count++;
            }
        }
    }
    return count;
}
```

12. 答：Server 端程序:

```
package test;
import java.net.*;
import java.io.*;
public class Server{
    private ServerSocket ss;
```

```
    private Socket socket;
    private BufferedReader in;
    private PrintWriter out;
    public Server(){
        try {
            ss=new ServerSocket(10000);
            while(true){
                socket = ss.accept();
                String                    RemoteIP                    =
            socket.getInetAddress().getHostAddress();
                String RemotePort = ":"+socket.getLocalPort();
                System.out.println("A    client    come    in!IP:"    +
            RemoteIP+RemotePort);
                in          =          new          BufferedReader(new
            InputStreamReader(socket.getInputStream()));
                String line = in.readLine();
                System.out.println("Cleint send is :" + line);
                out = new PrintWriter(socket.getOutputStream(),true);
                out.println("Your Message Received!");
                out.close();
                in.close();
                socket.close();
            }
        }catch (IOException e){
            out.println("wrong");
        }
    }
    public static void main(String[] args){
        new Server();
    }
}
```

Client 端程序:

```
package test;
import java.io.*;
import java.net.*;
public class Client {
    Socket socket;
    BufferedReader in;
    PrintWriter out;
    public Client(){
        try {
            System.out.println("Try to Connect to 127.0.0.1:10000");
            socket = new Socket("127.0.0.1",10000);
            System.out.println("The Server Connected!");
```

```
            System.out.println("Please enter some Character:");
            BufferedReader line = new BufferedReader(new
        InputStreamReader(System.in));
            out = new PrintWriter(socket.getOutputStream(),true);
            out.println(line.readLine());
            in = new BufferedReader( new
        InputStreamReader(socket.getInputStream()));
            System.out.println(in.readLine());
            out.close();
            in.close();
            socket.close();
        }catch(IOException e){
            out.println("Wrong");
        }
    }
    public static void main(String[] args) {
        new Client();
    }
}
```

13.
```
import java.util.*;
public class bycomma{
 public static String[] splitStringByComma(String source){
   if(source==null||source.trim().equals(""))
    return null;
   StringTokenizer commaToker = new StringTokenizer(source,",");
   String[] result = new String[commaToker.countTokens()];
   int i=0;
   while(commaToker.hasMoreTokens()){
    result[i] = commaToker.nextToken();
    i++;
   }
   return result;
 }
 public static void main(String args[]){
 String[] s = splitStringByComma("5,8,7,4,3,9,1");
 int[] ii = new int[s.length];
 for(int i = 0;i<s.length;i++){
 ii[i] =Integer.parseInt(s[i]);
 }
    Arrays.sort(ii);
    //asc
```

```
    for(int i=0;i<s.length;i++){
    System.out.println(ii[i]);
    }
    //desc
    for(int i=(s.length-1);i>=0;i--){
    System.out.println(ii[i]);
    }
 }
}
```

14.
```
package test.format;
import java.text.NumberFormat;
import java.util.HashMap;
public class SimpleMoneyFormat {
  public static final String EMPTY = "";
  public static final String ZERO = "零";
  public static final String ONE = "壹";
  public static final String TWO = "贰";
  public static final String THREE = "三";
  public static final String FOUR = "肆";
  public static final String FIVE = "伍";
  public static final String SIX = "陆";
  public static final String SEVEN = "柒";
  public static final String EIGHT = "捌";
  public static final String NINE = "玖";
  public static final String TEN = "拾";
  public static final String HUNDRED = "佰";
  public static final String THOUSAND = "仟";
  public static final String TEN_THOUSAND = "万";
  public static final String HUNDRED_MILLION = "亿";
  public static final String YUAN = "元";
  public static final String JIAO = "角";
  public static final String FEN = "分";
  public static final String DOT = ".";

  private static SimpleMoneyFormat formatter = null;
  private HashMap chineseNumberMap = new HashMap();
  private HashMap chineseMoneyPattern = new HashMap();
  private NumberFormat numberFormat = NumberFormat.getInstance();

  private SimpleMoneyFormat() {
    numberFormat.setMaximumFractionDigits(4);
```

```
    numberFormat.setMinimumFractionDigits(2);
    numberFormat.setGroupingUsed(false);

    chineseNumberMap.put("0", ZERO);
    chineseNumberMap.put("1", ONE);
    chineseNumberMap.put("2", TWO);
    chineseNumberMap.put("3", THREE);
    chineseNumberMap.put("4", FOUR);
    chineseNumberMap.put("5", FIVE);
    chineseNumberMap.put("6", SIX);
    chineseNumberMap.put("7", SEVEN);
    chineseNumberMap.put("8", EIGHT);
    chineseNumberMap.put("9", NINE);
    chineseNumberMap.put(DOT, DOT);

    chineseMoneyPattern.put("1", TEN);
    chineseMoneyPattern.put("2", HUNDRED);
    chineseMoneyPattern.put("3", THOUSAND);
    chineseMoneyPattern.put("4", TEN_THOUSAND);
    chineseMoneyPattern.put("5", TEN);
    chineseMoneyPattern.put("6", HUNDRED);
    chineseMoneyPattern.put("7", THOUSAND);
    chineseMoneyPattern.put("8", HUNDRED_MILLION);
  }

  public static SimpleMoneyFormat getInstance() {
    if (formatter == null)
      formatter = new SimpleMoneyFormat();
    return formatter;
  }

  public String format(String moneyStr) {
    checkPrecision(moneyStr);
    String result;
    result = convertToChineseNumber(moneyStr);
    result = addUnitsToChineseMoneyString(result);
    return result;
  }

  public String format(double moneyDouble) {
    return format(numberFormat.format(moneyDouble));
  }

  public String format(int moneyInt) {
```

```
    return format(numberFormat.format(moneyInt));
  }

  public String format(long moneyLong) {
    return format(numberFormat.format(moneyLong));
  }

  public String format(Number moneyNum) {
    return format(numberFormat.format(moneyNum));
  }

  private String convertToChineseNumber(String moneyStr) {
    String result;
    StringBuffer cMoneyStringBuffer = new StringBuffer();
    for (int i = 0; i < moneyStr.length(); i++) {

cMoneyStringBuffer.append(chineseNumberMap.get(moneyStr.substring(i, i +
1)));
    }
    //拾佰仟万亿等都是汉字里面才有的单位，加上它们
    int indexOfDot = cMoneyStringBuffer.indexOf(DOT);
    int moneyPatternCursor = 1;
    for (int i = indexOfDot - 1; i > 0; i--) {
      cMoneyStringBuffer.insert(i,    chineseMoneyPattern.get(EMPTY    +
moneyPatternCursor));
      moneyPatternCursor = moneyPatternCursor == 8 ? 1 : moneyPatternCursor
+ 1;
    }

    String                          fractionPart                          =
cMoneyStringBuffer.substring(cMoneyStringBuffer.indexOf("."));
    cMoneyStringBuffer.delete(cMoneyStringBuffer.indexOf("."),
cMoneyStringBuffer.length());
    while (cMoneyStringBuffer.indexOf("零拾") != -1) {
      cMoneyStringBuffer.replace(cMoneyStringBuffer.indexOf("零拾"),
cMoneyStringBuffer.indexOf("零拾") + 2, ZERO);
    }
    while (cMoneyStringBuffer.indexOf("零佰") != -1) {
      cMoneyStringBuffer.replace(cMoneyStringBuffer.indexOf("零佰"),
cMoneyStringBuffer.indexOf("零佰") + 2, ZERO);
    }
    while (cMoneyStringBuffer.indexOf("零仟") != -1) {
      cMoneyStringBuffer.replace(cMoneyStringBuffer.indexOf("零仟"),
cMoneyStringBuffer.indexOf("零仟") + 2, ZERO);
```

```
    }
    while (cMoneyStringBuffer.indexOf("零万") != -1) {
      cMoneyStringBuffer.replace(cMoneyStringBuffer.indexOf(" 零 万 "),
cMoneyStringBuffer.indexOf("零万") + 2, TEN_THOUSAND);
    }
    while (cMoneyStringBuffer.indexOf("零亿") != -1) {
      cMoneyStringBuffer.replace(cMoneyStringBuffer.indexOf(" 零 亿 "),
cMoneyStringBuffer.indexOf("零亿") + 2, HUNDRED_MILLION);
    }
    while (cMoneyStringBuffer.indexOf("零零") != -1) {
      cMoneyStringBuffer.replace(cMoneyStringBuffer.indexOf(" 零 零 "),
cMoneyStringBuffer.indexOf("零零") + 2, ZERO);
    }
    if              (cMoneyStringBuffer.lastIndexOf(ZERO)              ==
cMoneyStringBuffer.length() - 1)
      cMoneyStringBuffer.delete(cMoneyStringBuffer.length()     -     1,
cMoneyStringBuffer.length());
    cMoneyStringBuffer.append(fractionPart);

    result = cMoneyStringBuffer.toString();
    return result;
  }
private String addUnitsToChineseMoneyString(String moneyStr) {
    String result;
    StringBuffer cMoneyStringBuffer = new StringBuffer(moneyStr);
    int indexOfDot = cMoneyStringBuffer.indexOf(DOT);
    cMoneyStringBuffer.replace(indexOfDot, indexOfDot + 1, YUAN);
```

15.答：

```
import java.util.*;
        public class PrimeTest {

            public static void main(String[] args){
                PrimeTest pt=new PrimeTest();
                int data=100;
                pt.addList(data);
            }
            public void addList(int data)
            {
                List list =new ArrayList();
                int sum = 0;
                for(int i=1;i<=data;i++)
                {
```

```
                int j=2;
                for( ;j <i ;j++)
                {
                    if(i % j == 0)
                    break;
                }
                if(j ==i)
                {
                    sum+=i;
                    list.add((Object)(i));
                }
            }
            System.out.println(list);
            System.out.println(sum);
        }
    }
```

16.答：
```
import java.io.*;
public class ControlInput {
    public static void main(String[] args) throws Exception{
        System.out.println("Enter:");
        InputStream in = System.in;
        String str="";
        StringBuffer buf=new StringBuffer();
        int result;
        while ((result = in.read()) != -1){
            buf.append((char)result);
            byte[] temp=buf.toString().getBytes("ISO8859-1");
            str =new String(temp,"gb2312");
            System.out.println(str);

        }
    }
}
```

17. 答：
```
public class TestClass {
    int num;
    String str = " ";
    IncludeClass inClass;
    public TestClass(){
        inClass = new IncludeClass();
    }
```

```
    public boolean equals(TestClass tc){
        if (this.num == tc.num && this.str.equals(tc.str) &&
this.inClass.equals(tc.inClass)){
            return true;
        }
        return false;
    }
    public TestClass clone (){
        TestClass tc = new TestClass();
        tc.num = this.num;
        tc.str = this.str;
        tc.inClass.fnum = this.inClass.fnum;
        return tc;
    }
    public static void main(String[] args){
        try{
            TestClass tc1 = new TestClass();
            TestClass tc2 = tc1.clone();
            TestClass tc3 = new TestClass();
            System.out.println(tc2.equals(tc1));
            System.out.println(tc3.equals(tc1));
        }catch (Exception ef){
            ef.printStackTrace();
        }
    }
}
class IncludeClass{
    double fnum;
    public boolean equals(IncludeClass inClass){
        return this.fnum == inClass.fnum;
    }
}
```

四、Web、JavaScript

I、简答题

1.答：Web 容器加载 Servlet 并将其实例化后，Servlet 生命周期开始，容器运行其 init 方法进行 Servlet 的初始化，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。

2. 答：

```
public class ServletName extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletResponse
response)
        throws ServletException, IOException{
    }
    public void doGet(HttpServletRequest request, HttpServletResponse
response)
        throws ServletException, IOException {
    }
}
```

3. 答：0。

4. 答：不输出，设 scope="request"。

5. 答：动作在运行时发生作用，指令在编译时发生作用。动作：<jsp:include>、<jsp:forward>、<jsp:useBean>
指令：<%@page%> <%@include%> <%@plugin%>

6. 答：<%@page session="false"%>

application, out, request, response, page, config, exception, pagecontext

7. 答：a1.jsp 输出 10，a2.jsp 访问不成功。

8. 答：request.getPathInfo(); PathInfo

9. 答：先实例化，init，service，doXXX，容器调度 destroy；init 被调用 1 次

10. 答：覆盖 init 方法<%! public init(){…..}%>

11. 答：<%@page isThreadSafe="false"%>，完成 SingleThreadMode 接口。

12. 答：response.sendRedirect("a.html");不在同一个 request 作用域。

13. 答：

```
(1)<jsp:forward page="a.jsp?id=1"/>
(2)<jsp:forward page="a.jsp">
        <jsp:param name="id" value="1"/>
   </jsp:forward>
```

14. 答：session, getServletContext();

15. 答：forward 是容器中控制权的转向，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，并且从浏览器的地址栏中可以看到跳转后的链接地址。前者更加高效，在前者可以满足需要时，尽量使用 forward()方法，并且，这样也有助于隐藏实际的链接；在有些情况下，比如，需要跳转到一个其它服务器上的资源，则必须使用 sendRedirect()方法。

16. 答：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；静态 INCLUDE 用 include 伪码实现,它不会检查所含文件的变化，适用于包含静态页面<%@ include file="included.htm" %>

17. 答：容器是一个软件的概念，提供了一个组件运行的环境。Web 容器或 ejb 容器都是其中的一种。例如 tomcat 就是一个开源的服务器，是 jsp/servlet 的容器。

18. 答：JSP 共有以下 9 种基本内置组件：

Jsp 内置对象	功能	主要方法
out	向客户端输出数据	print() println() flush() clear() isAutoFlush() getBufferSize() close() ……

request	向客户端请求数据	getAttributeNames() getCookies() getParameter() getParameterValues() setAttribute() getServletPath() ……
response	封装了 jsp 产生的响应，然后被发送到客户端以响应客户的请求	addCookie() sendRedirect() setContentType() flushBuffer() getBufferSize() getOutputStream() sendError() containsHeader()……
application		
config	表示 Servlet 的配置,当一个 Servlet 初始化时，容器把某些信息通过此对象传递给这个 Servlet	getServletContext() getServletName() getInitParameter() getInitParameterNames()……
page	Jsp 实现类的实例，它是 jsp 本身，通过这个可以对它进行访问	flush()……
pagecontext	为 JSP 页面包装页面的上下文。管理对属于 JSP 中特殊可见部分中已经命名对象的该问	forward() getAttribute() getException() getRequest() getResponse() getServletConfig() getSession() getServletContext() setAttribute() removeAttribute() findAttribute() ……
session	用来保存每个用户的信息，以便跟踪每个用户的操作状态	getAttribute() getId() getAttributeNames() getCreateTime() getMaxInactiveInterval() invalidate() isNew()
exception	反映运行的异常	getMessage()……

JSP 共有以下 6 种基本动作：

1) jsp:include：在页面被请求的时候引入一个文件。
2) jsp:useBean：寻找或者实例化一个 JavaBean。
3) jsp:setProperty：设置 JavaBean 的属性。
4) jsp:getProperty：输出某个 JavaBean 的属性。
5) jsp:forward：把请求转到一个新的页面。
6) jsp:plugin：根据浏览器类型为 Java 插件生成 OBJECT 或 EMBED 标记。

19. 答：<%@page language=”java” contenType=”text/html;charset=gb2312” session=” true” buffer=” 64kb” autoFlush=” true” isThreadSafe=” true” info=” text” errorPage=” error.jsp” isErrorPage=” true” isELIgnored=” true” pageEncoding=” gb2312” import=” java.sql.*” %>

isErrorPage：是否能使用 Exception 对象；isELIgnored：是否忽略 EL 表达式；
<%@include file=”filename”%>
<%@taglib prefix=”c”uri=”http://……”%>

20. 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。

21. 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。

22. 答：Form 中的 GET 和 POST 方法，在数据传输过程中分别对应了 HTTP 协议中的 GET 和 POST 方法。二者主要区别如下：
1）GET 是用来从服务器上获得数据，而 POST 是用来向服务器上传递数据；
2）GET 将表单中数据按照 variable=value 的形式，添加到 action 所指向的 URL 后面，并且两者使用“?”连接，而各个变量之间使用“&”连接；POST 是将表单中的数据放在 form 的数据体中，按照变量和值相对应的方式，传递到 action 所指向 URL；
3）GET 是不安全的，因为在传输过程，数据被放在请求的 URL 中；POST 的所有操作对用户来说都是不可见的；
4）GET 传输的数据量小，这主要是因为受 URL 长度限制；而 POST 可以传输大量的数据，所以在上传文件只能使用 POST；
5）GET 限制 Form 表单的数据集必须为 ASCII 字符，而 POST 支持整个 ISO10646 字符集；
6）GET 是 Form 的默认方法。

23. 答： doPost 会把地址作为一个文件写到服务器，而 doGet 不会，而只显示在地址栏。

24. 答：其中 doGet 方法的访问限制修饰符应该为 public，因为 protected 修饰的方法只能被同包中的类或其子类才可访问，这将导致 web 容器无法调用该方法。

25. 答：Jsp 页面中的 form 标签里的 method 属性为 get 时调用 doGet()，为 post 时调用

doPost()。

26. 答：可在页面把 checkbox 的 name 属性取同一个，value 属性取每个条目的 id,后台用 getParamter(“name”)能取到 checkbox 的一组值。

27. 答：用 java 或 javaScript 的处理方式分别如下：

Java： request.getParameter(“bank No”);

javaScript： var selectItems = document.getElementsByName(“bank No”);
selectItems[0].value;

28. 答：JavaScript 与 Java 是两个公司开发的不同的两个产品。Java 是 SUN 公司推出的新一代面向对象的程序设计语言，特别适合于 Internet 应用程序开发；而 JavaScript 是 Netscape 公司的产品，其目的是为了扩展 Netscape Navigator 功能,而开发的一种可以嵌入 Web 页面中的基于对象和事件驱动的解释性语言,它的前身是 Live Script；而 Java 的前身是 Oak 语言。下面对两种语言间的异同作如下比较：

1）基于对象和面向对象：

Java 是一种真正的面向对象的语言，即使是开发简单的程序，必须设计对象。

JavaScript 是种脚本语言，它可以用来制作与网络无关的，与用户交互作用的复杂软件。它是一种基于对象（Object Based）和事件驱动（Event Driver）的编程语言。因而它本身提供了非常丰富的内部对象供设计人员使用。

2）解释和编译：

Java 的源代码在执行之前，必须经过编译；

JavaScript 是一种解释性编程语言，其源代码不需经过编译，由浏览器解释执行。

3）强类型变量和弱类型变量：

Java 采用强类型变量检查，即所有变量在编译之前必须作声明；

JavaScript 中变量声明，采用其弱类型。即变量在使用前不需作声明，而是解释器在运行时检查其数据类型。

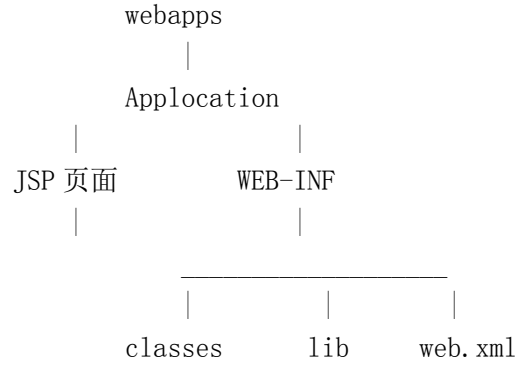
4）代码格式不一样。

29. 答：常用于数据输入校验和页面特殊效果等。

30. 答：最常用的容器包括：tomcat、weblogic；

开发工具有：eclipse,jbuilder。

31. 答：目录结构如下图所示：



32. 答：JSP 是 Servlet 技术的扩展，本质上是 Servlet 的简易方式，更强调应用的外表表达。JSP 编译后是"类 servlet"。Servlet 和 JSP 最主要的不同点在于，Servlet 的应用逻辑是在 Java 文件中，并且完全从表示层中的 HTML 里分离开来。而 JSP 的情况是 Java 和 HTML 可以组合成一个扩展名为.jsp 的文件。JSP 侧重于视图，Servlet 主要用于控制逻辑。

33. 答：A . page 是代表与一个页面相关的对象和属性。一个页面由一个编译好的 Java servlet 类（可以带有任何的 include 指令，但是没有 include 动作）表示。这既包括 servlet 又包括被编译成 servlet 的 JSP 页面。
B . request 是代表与 Web 客户机发出的一个请求相关的对象和属性。一个请求可能跨越多个页面，涉及多个 Web 组件（由于 forward 指令和 include 动作的关系）。
C . session 是代表与用于某个 Web 客户机的一个用户体验相关的对象和属性。一个 Web 会话可以，也经常会跨越多个客户机请求。
D . application 是代表与整个 Web 应用程序相关的对象和属性。这实质上是跨越整个 Web 应用程序，包括多个页面、请求和会话的一个全局作用域。

34. 答：setAttribute(String name,Object)：设置名字为 name 的属性值;
getAttribute(String name)：返回由 name 指定的属性值;
getAttributeNames()：返回 request 对象所有属性的名字集合(枚举);
getCookies()：返回客户端的所有 Cookie 对象，结果是一个 Cookie 数组;
getCharacterEncoding()：返回请求中的字符编码方式;
getContentLength()：返回请求的 Body 的长度;
getHeader(String name)：获得 HTTP 协议定义的文件头信息;
getHeaders(String name)：返回指定名的 request Header 的所有值(枚举);
getHeaderNames()：返回所有 request Header 的名字(枚举);
getInputStream()：返回请求的输入流，用于获得请求中的数据;
getMethod()：获得客户端向服务器端传送数据的方法;
getParameter(String name)：获得客户端请求中传送的 name 指定的参数值;
getParameterNames()：获得客户端传送给服务器端的所有参数的名字(枚举);
getParameterValues(String name)：获得有 name 指定的参数的所有值;
getProtocol()：获取客户端向服务器端传送数据所依据的协议名称;
getQueryString()：获得查询字符串;
getRequestURI()：获取发出请求字符串的客户端地址;
getRemoteAddr()：获取客户端的 IP 地址;
getRemoteHost()：获取客户端的名字;
getSession([Boolean create])：返回和请求相关 ；
SessiongetServerName()：获取服务器的名字;
getServletPath()：获取客户端所请求的脚本文件的路径;
getServerPort()：获取服务器的端口号;
removeAttribute(String name)：删除请求中的一个属性。

35. 答：<%@page isThreadSafe=”false”%>。

36. 答：request,session,application,cookie 等。

37. 答：基于 Java 的 Web 应用系统采用 MVC 架构模式，即 model（模型）、view（视图）、control（控制）分离设计；这是目前 WEB 应用服务系统的主流设计方向。

model：即处理业务逻辑的模块，每一种处理一个模块；

view：负责页面显示，显示 model 处理结果给用户，主要实现数据到页面转换过程；

control：负责每个请求的分发，把 form 数据传递给 model 处理，把处理结果的数据传递给 view 显示。

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

39. 答：BEA WebLogic Server, IBM WebSphere Application Server, Oracle9iApplication Server, JBoss, Tomcat。

40. 答：

```
public void init(ServletConfig config){};
public ServletConfig getServletConfig(){} ;
public String getServletInfo(){} ;
public void service(ServletRequest request,ServletResponse response){} ;
public void destroy(){}。
```

41. 答：cookie、URL 重写、设置表单隐藏域。

42. 答：C/S 是 Client/Server 的缩写，是客户机与服务器结构的应用程序，服务器通常采用高性能的 PC、工作站或小型机，并采用大型数据库系统，如 Oracle、Sybase、Informix 或 SQL Server。客户端需要安装专用的客户端软件。

B/S 是 Brower/Server 的缩写，是浏览器和服务器结构的应用程序，即 Web 应用程序，客户机上只要安装一个浏览器（Browser），如 Netscape Navigator 或 InternetExplorer，服务器安装 Oracle、Sybase、Informix 或 SQL Server 等数据库。在这种结构下，用户界面完全通过 WWW 浏览器实现，一部分事务逻辑在前端实现，但是主要事务逻辑在服务器端实现。浏览器通过Web Server 同数据库进行数据交互。

C/S 与 B/S 区别：

1)硬件环境不同:

C/S 一般建立在专用的网络上,小范围里的网络环境,局域网之间再通过专门服务器

提供连接和数据交换服务；
B/S 建立在广域网之上的,不必是专门的网络硬件环境,例如：电话上网,租用设备.信息自己管理.有比 C/S 更强的适应范围,一般只要有操作系统和浏览器就行；
2)对安全要求不同：
C/S 一般面向相对固定的用户群,对信息安全的控制能力很强.一般高度机密的信息系统采用 C/S 结构适宜.可以通过 B/S 发布部分可公开信息；
B/S 建立在广域网之上,对安全的控制能力相对弱,可能面向不可知的用户；
3)对程序架构不同：
C/S 程序可以更加注重流程,可以对权限多层次校验,对系统运行速度可以较少考虑；
B/S 对安全以及访问速度的多重的考虑,建立在需要更加优化的基础之上.比 C/S 有更高的要求 B/S 结构的程序架构是发展的趋势,从 MS 的.Net 系列的 BizTalk 2000 Exchange 2000 等,全面支持网络的构件搭建的系统.SUN 和 IBM 推的 JavaBean 构件技术等,使 B/S 更加成熟；
4)软件重用不同：
C/S 程序可以不可避免的整体性考虑,构件的重用性不如在 B/S 要求下的构件的重用性好；
B/S 对的多重结构,要求构件相对独立的功能.能够相对较好的重用.就入买来的餐桌可以再利用,而不是做在墙上的石头桌子；
5)系统维护不同：
C/S 程序由于整体性,必须整体考察,处理出现的问题以及系统升级.升级难.可能是再做一个全新的系统；
B/S 构件组成,方面构件个别的更换,实现系统的无缝升级.系统维护开销减到最小.用户从网上自己下载安装就可以实现升级；
6)处理问题不同：
C/S 程序可以处理用户面固定,并且在相同区域,安全要求高需求,与操作系统相关.应该都是相同的系统；
B/S 建立在广域网上,面向不同的用户群,分散地域,这是 C/S 无法作到的.与操作系统平台关系最小；

7)用户接口不同：
C/S 多是建立的 Window 平台上,表现方法有限,对程序员普遍要求较高；
B/S 建立在浏览器上,有更加丰富和生动的表现方式与用户交流.并且大部分难度减低,减低开发成本；

8)信息流不同：
C/S 程序一般是典型的中央集权的机械式处理,交互性相对低；
B/S 信息流向可变化,B-B B-C B-G 等信息、流向的变化,更像交易中心。

43. 答：可以验证客户是否来自可信的网络，可以对客户提交的数据进行重新编码，可以从系统里获得配置的信息，可以过滤掉客户的某些不应该出现的词汇，可以验证用户是否登录，可以验证客户的浏览器是否支持当前的应用，可以记录系统的日志等等。

44. 答：首先要实现（implements）Filter 接口，同时覆盖 Filter 接口的三个方法：
init(FilterConfig config)
//用于获得 FilterConfig 对象；
doFilter(ServletRequest request, ServletResponse response,FilterChain chain)
//进行过滤处理一些业务；
destroy()
//销毁 Filter。

45. 答：HttpSession 中可以跟踪并储存用户信息，把值设置到属性中，有 2 个方法：
setAttribute(),getAttrribute()；
例如：在一个方法中用 session.setAttribute(“student”,student);在 session 中设置一个属性名为 student,值为一个名为 student 的对象。而后可在同一 session 范围内用 getAttribute(“student”)取出该属性，得到 student 对象。

46. 答：在 JSP 中使用 JavaBean 常用的动作有：
1）<jsp:useBean />：用来创建和查找 bean 对象；
2）<jsp:setProperty />：用来设置 bean 的属性，即调用其 setXxx()方法；
3）<jsp:getProperty />：用来获得 bean 的属性，即调用其 getXxx()方法。

47. 答：JSP 中的请求转发可利用 forward 动作实现：<jsp:forward />；
Serlvet 中实现请求转发的方式为：
getServletContext().getRequestDispatcher(path).forward(req,res)。

48. 答：用于配置 web 应用的信息；如 listener、filter 及 servlet 的配置信息等。

49. 答： <c:if>、<c:choose>、<c: when>、<c: otherwise>、<c:forEach>、<c:set>。

50. 答：<bean:message />
<html:errors />
<bean:include />
<html:messages />
<bean:define />
<html:html>
<bean:write />
<html:form>
<bean:resource />
<html:link>
<bean:cokkie />
<html:text>
<bean:heder />
<logic:present />

```
<bean:parameter />
<logic:equal />
<bean:size />
```

51. 答：

作用：分离 jsp 页面的内容和逻辑；

业务逻辑开发者可以创建自定义标签；

封装业务逻辑；

可重用并且易维护；

易于手工修改、易于工具维护；

提供简洁的语法；

定义:

写标签处理器；

写 tld 文件；

将标签处理器和 tld 文件放到同一个包里面；

把 jsp 页面和标签库配置部署在一起。

52. 答：1）优点：简单易用，与 Java 有类似的语法，可以使用任何文本编辑工具编写，只需要浏览器就可执行程序，并且事先不用编译，逐行执行，无需进行严格的变量声明，而且内置大量现成对象，编写少量程序可以完成目标；

2）缺点：不适合开发大型应用程序；

3）Javascript 有 11 种内置对象：Array、String、Date、Math、Boolean、Number、Function、Global、Error、RegExp、Object。

53. 答：jsp 页面实现跳转主要有 jsp:forward 和 sendRedirect 两种方法

jsp:forward：在本容器内跳转 。 跳转后，地址栏地址不变 。效率高。跳转后立即结束本页的内容。

sendRedirect：在容器之间的跳转，跳转后地址栏地址为跳转后的地址，效率较低。通常采用 jsp:forward 方式跳转。

II、编程题

1. 答：代码如下：

```
import javax.servlet.jsp.tagext.*;
import javax.servlet.jsp.*;
import java.io.*;
public class TimeTag extends SimpleTagSupport{
    private boolean isServer = true;
    public void setServer(boolean isServer){
        this.isServer = isServer;
    }
    public void doTag() throws JspException, IOException{
        JspWriter out = getJspContext().getOut();
        if(isServer) {
```

```
            out.println(new java.util.Date());
        }else{
            out.println("<script language=\"javascript\">");
            out.println("document.write(new Date());");
            out.println("</script>");
        }
    }
}
```

五、JDBC、Struts、Hibernate、Spring 及其它 J2EE 技术

I、简答题

1. 答：可采用连接池。

2. 答：对象关系映射（Object—Relational Mapping，简称 ORM）是一种为了解决面向对象与面向关系数据库存在的互不匹配的现象的技术；简单的说，ORM 是通过使用描述对象和数据库之间映射的元数据，将 java 程序中的对象自动持久化到关系数据库中；本质上就是将数据从一种形式转换到另外一种形式。

3. 答：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。

4. 答：是 ActionServlet，所有的 struts 请求都经由该类转发处理。

5. 答：Struts 是采用 Java Servlet/JavaServerPages 技术，开发 Web 应用程序的开放源码的 framework。采用 Struts 能开发出基于 MVC(Model-View-Controller)设计模式的应用构架；Struts 有如下的主要功能：

1)包含一个 controller servlet，能将用户的请求发送到相应的 Action 对象。

2) JSP 自由 tag 库，并且在 controller servlet 中提供关联支持，帮助开发员创建交互式表单应用。

3)提供了一系列实用对象：XML 处理、通过 Java reflection APIs 自动处理 JavaBeans 属性、国际化的提示和消息。

6. 答：Struts 是采用 Java Servlet/JavaServer Pages 技术开发 Web 应用程序的开放源码的 framework。采用 Struts 能开发出基于 MVC(Model-View-Controller)设计模式的应用构架。

Struts 有如下的主要功能：

1)包含一个 controller servlet，能将用户的请求发送到相应的 Action 对象；

2)JSP 自由 tag 库，并且在 controller servlet 中提供关联支持，帮助开发人员创建交互式表单应用；

3)提供了一系列实用对象：XML 处理、通过 Java reflection APIs 自动处理 JavaBeans 属性、国际化的提示和消息。

7. 答：struts framework 是一种基于 java 的技术，Web 应用程序开发人员通过 struts framework 即可充分利用面向对象设计、代码重用以及“编写一次、到处运行”的优点。Struts 提供了一种创建 Web 应用程序的框架，其中对应用程序的显示、表示和数据的后端代码进行了抽象。Struts 采用 jsp 作为 MVC 的视图,由 ActionServlet 具体指定的 action 动作类作为控制器即 MVC 中的 C,负责视图与模型之间的交互。控制器的每个入口点都由名为 struts-config.xml 的配置文件设置。该文件把来自视图的请求映射为特定的 JAVA 类以进行相应的处理，控制器还指定下一个视图的位置。Struts 中的模型主要指的就是 javabean，它是模型的代表，主要封装数据和业务逻辑。

Struts 的处理流程：

控制器进行初始化工作，读取配置文件，为不同的 Struts 模块初始化相应的 ModulConfig 对象。

控制器接收 Http 请求，并从 ActionConfig 中找出对应于该请求的 Action 子类，如果没有对应的 Action，控制器直接将请求转发给 JSP 或者静态页面，否则控制器将请求分发至具体的 Action 类进行处理。

在控制器调用具体的 Action 的 Execute 方法之前，ActionForm 对象将利用 Http 请求中的参数来填充自己。还可以在 ActionForm 类中调用 Validate 方法来检查请求参数的合法性，并且可以返回一个包含所有错误信息的 ActionErrors 对象。

执行具体的 Execute 的方法，它负责执行相应的业务逻辑。执行完后，返回一个 ActionForward 对象，控制器通过该 ActionForward 对象来进行转发工作。也可以把 Action 要处理的业务逻辑封装在 JavaBean 中，如果系统中还有 EJB，那么通过

JavaBean 调用 EJB 以完成业务处理；如果没有 EJB，那么就直接在 JavaBean 中连接数据库，进行数据库相关的操作。

8. 答：MVC 包括三类对象，model 是应用对象，view 是视图，controller 是控制器，它定义用户界面对用户输入的响应方式。

在 MVC 体系中，模型通常被称为“业务逻辑”，是真正完成任务的代码，视图就是使用界面，反映数据的变化。控制器控制着模型和视图之间的交互过程，它决定着向用户返回怎样的视图、检查通过界面输入的信息以及选择处理输入信息的模型在 MVC 中，表示层和逻辑层分离，各部分可相互独立进行开发，便于开发和维护，提高了开发效率。

9. 答：Configuration 接口：配置 Hibernate，根据其启动 Hibernate，创建 SessionFactory 对象；SessionFactory 接口：初始化 Hibernate，充当数据存储源的代理，创建 session 对象，SessionFactory 是线程安全的，意味着它的同一个实例可以被应用的多个线程共享，是重量级、二级缓存；Session 接口：负责保存、更新、删除、加载和查询对象，是线程不安全的，避免多个线程共享同一个 session，是轻量级、一级缓存；Transaction 接口：管理事务；Query 和 Criteria 接口：执行数据库的查询。

10. 1）在 Hibernate 中，在配置文件呈标题一对多，多对多的标签是什么；

2）Hibernate 的二级缓存是什么；

3）Hibernate 是如何处理事务的；

答：1）一对多的标签为<one-to-many>；多对多的标签为<many-to-many>；

2）SessionFactory 的缓存为 Hibernate 的二级缓存；

3）Hibernate 的事务实际上是底层的 JDBC Transaction 的封装或者是 JTATransaction 的封装；默认情况下使用 JDBCTransaction。

11. 答：//首先获得 SessionFactory 的对象

```
SessionFactory sessionFactory = new Configuration().configure().
buildSessionFactory();
//然后获得 session 的对象
Session session = sessionFactory.openSession();
//其次获得 Transaction 的对象
Transaction tx = session.beginTransaction();
//执行相关的数据库操作:增,删,改,查
session.save(user); //增加,user 是 User 类的对象
session.delete(user); //删除
session.update(user); //更新
Query query =session.createQuery(“from User”); //查询
List list = query.list();
//提交事务
tx.commit();
//如果有异常,我们还要作事务的回滚,恢复到操作之前
tx.rollback();
//最后还要关闭 session,释放资源
session.close();
```

12. 答：轻量级是指它的创建和销毁不需要消耗太多的资源，意味着可以在程序中经常创建和销毁 session 的对象；重量级意味不能随意的创建和销毁它的实例，会

占用很多的资源。

13. 答：Connection 类中提供了 3 个事务处理方法：
setAutoCommit(Boolean autoCommit):设置是否自动提交事务，默认为自动提交事务，即为 true，通过设置 false 禁止自动提交事务；
commit():提交事务；
rollback():回滚事务。
14. 答：Java 中访问数据库的步骤如下：
1）注册驱动；
2）建立连接；
3）创建 Statement；
4）执行 sql 语句；
5）处理结果集（若 sql 语句为查询语句）；
6）关闭连接。
PreparedStatement 被创建时即指定了 SQL 语句，通常用于执行多次结构相同的 SQL 语句。
15. 答：方法分别为：
1) Hibernate 的分页：
Query query = session.createQuery("from Student");
query.setFirstResult(firstResult);//设置每页开始的记录号
query.setMaxResults(resultNumber);//设置每页显示的记录数
Collection students = query.list();
2) JDBC 的分页：根据不同的数据库采用不同的 sql 分页语句例如:
Oracle 中的 sql 语句为: "SELECT * FROM (SELECT a.*, rownum r FROM TB_STUDENT)
WHERE r between 2 and 10"查询从记录号 2 到记录号 10 之间的所有记录
16. 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。
17. 答：EJB 是企业级的 JavaBean，它提供了构建企业级业务逻辑的一种组件模型。EJB 分为三种：Session Bean Entity Bean Message-Driven Bean 三种，其中 Session Bean 分为有状态和无状态 Session Bean 两种，Entity Bean 分为容器管理的 Entity Bean（ CMP ）和 Bean 管理的 Entity Bean（ BMP ）。每一个 EJB 由一个远程接口、一个本地接口和一个 EJB 容器实现组成，远程接口声明了提供给 EJB 客户调用的各种应用方法，本地接口声明了创建新的 EJB 实例的 create 方法、寻找 EJB 实例的查找(finder)方法以及删除 EJB 实例的 remove 方法。EJB 容器提供了 EJB 的运行环境和生命周期的管理。

18. 答：会话（Session）Bean、实体（Entity）Bean、消息驱动的（Message Driven）Bean；会话 Bean 又可分为有状态（Stateful）和无状态（Stateless）两种；实体 Bean 可分为 Bean 管理的持续性（BMP）和容器管理的持续性（CMP）两种。

19. 答：remote 接口定义了业务方法，用于 EJB 客户端调用业务方法；
home 接口是 EJB 工厂用于创建和移除查找 EJB 实例。

20. 答：设置 JNDI 服务工厂以及 JNDI 服务地址系统属性，查找 Home 接口，从 Home 接口调用 Create 方法创建 Remote 接口，通过 Remote 接口调用其业务方法。
21. 答：一个完整的基于 EJB 的分布式计算结构由六个角色组成，这六个角色可以由不同的开发商提供，每个角色所作的工作必须遵循 Sun 公司提供的 EJB 规范，以保证彼此之间的兼容性。这六个角色分别是 EJB 组件开发者（Enterprise Bean Provider）、应用组合者（Application Assembler）、部署者（Deployer）、EJB 服务器提供者（EJB Server Provider）、EJB 容器提供者（EJB Container Provider）、系统管理员（System Administrator），这里面，EJB 容器是 EJB 之所以能够运行的核心。EJB 容器管理着 EJB 的创建，撤消，激活，去活，与数据库的连接等等重要的核心工作；三个对象是 Remote（Local）接口、Home（LocalHome）接口，Bean 类。
22. 答：EJB 包括 Session Bean、Entity Bean、Message Driven Bean，基于 JNDI、RMI、JTA 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。StatelessSession 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. 答：对于 Stateless Session Bean、Entity Bean、Message Driven Bean 一般存在缓冲池管理，而对于 Entity Bean 和 Statefull Session Bean 存在 Cache 管理，通常包含创建实例，设置上下文、创建 EJB Object（create）、业务方法调用、remove 等过程，对于存在缓冲池管理的 Bean，在 create 之后实例并不从内存清除，而是采用缓冲池调度机制不断重用实例，而对于存在 Cache 管理的 Bean 则通过激活和去激活机制保持 Bean 的状态并限制内存中实例数量。

24. 答：以 Stateful Session Bean 为例：其 Cache 大小决定了内存中可以同时存在的

Bean 实例的数量，根据 MRU 或 NRU 算法，实例在激活和去激活状态之间迁移，激活机制是当客户端调用某个 EJB 实例业务方法时，如果对应 EJB Object 发现自己没有绑定对应的 Bean 实例则从其去激活 Bean 存储中（通过序列化机制存储实例）回复（激活）此实例。状态变迁前会调用对应的 ejbActive 和 ejbPassivate 方法。

25. 答： SessionBean：Stateless Session Bean 的生命周期是由容器决定的，当客户机发出请求要建立一个 Bean 的实例时，EJB 容器不一定要创建一个新的 Bean 的实例供客户机调用，而是随便找一个现有的实例提供给客户机。当客户机第一次调用一个 Stateful Session Bean 时，容器必须立即在服务器中创建一个新的 Bean 实例，并关联到客户机上，以后此客户机调用 Stateful Session Bean 的方法时容器会把调用分派到与此客户机相关联的 Bean 实例。
EntityBean：EntityBeans 能存活相对较长的时间，并且状态是持续的。只要数据库中的数据存在，Entity beans 就一直存活。而不是按照应用程序或者服务进程来说的。即使 EJB 容器崩溃了，Entity beans 也是存活的。Entity Beans 生命周期能够被容器或者 Beans 自己管理。EJB 通过以下技术管理事务：对象管理组织（OMG）的对象实务服务（OTS），Sun Microsystems 的 Transaction Service（JTS）、JavaTransaction API（JTA），开发组（X/Open）的 XA 接口。

26. 答：是通过使用容器或 Bean 自身管理事务的；当产生一个系统异常时容器就自动回滚事务。

27. 答：主要提供生命周期管理、代码产生、持续性管理、安全、事务管理、锁和并发行管理等服务。
28. 答：远程接口和 Home 接口不需要直接实现，他们的实现代码是由服务器产生的，程序运行中对应实现类会作为对应接口类型的实例被使用。
29. 答：是 java 命名和目录接口服务。
30. 答：web 容器：给处于其中的应用程序组件（JSP，SERVLET）提供一个环境，使 JSP,SERVLET 直接跟容器中的环境变量接口交互，不必关注其它系统问题。主要由 WEB 服务器来实现。例如：TOMCAT,WEBLOGIC,WEBSPHERE 等。该容器提供的接口严格遵守 J2EE 规范中的 WEB APPLICATION 标准。我们把遵守以上标准的 WEB 服务器就叫做 J2EE 中的 WEB 容器；
EJB 容器：Enterprise java bean 容器。更具有行业领域特色。他提供给运行在其中的组件 EJB 各种管理功能。只要满足 J2EE 规范的 EJB 放入该容器，马上就会被容器进行高效率的管理。并且可以通过现成的接口来获得系统级别的服务。例如邮件服务、事务管理；
JNDI：（Java Naming & Directory Interface）JAVA 命名目录服务。主要提供的功能是：提供一个目录系统，让其它各地的应用程序在其上面留下自己的索引，从而满足快速查找和定位分布式应用程序的功能；
JMS：（Java Message Service）JAVA 消息服务。主要实现各个应用程序之间的通讯。包括点对点和广播；
JTA：（Java Transaction API）JAVA 事务服务。提供各种分布式事务服务。应用程序只需调用其提供的接口即可；
JAF：（Java Action FrameWork）JAVA 安全认证框架。提供一些安全控制方面的框架。让开发者通过各种部署和自定义实现自己的个性安全控制策略；

RMI/IIOP:（Remote Method Invocation /internet 对象请求中介协议）他们主要用于通过远程调用服务。例如，远程有一台计算机上运行一个程序，它提供股票分析服务，我们可以在本地计算机上实现对其直接调用。当然这是要通过一定的规范才能在异构的系统之间进行通信。RMI 是 JAVA 特有的。

31. 答：J2EE 是 Sun 公司提出的多层(multi-diered),分布式(distributed),基于组件(component-base)的企业级应用模型(enterpriese application model).在这样的一个应用系统中，可按照功能划分为不同的组件，这些组件又可在不同计算机上，并且处于相应的层次(tier)中。所属层次包括客户层(clietn tier)组件,web 层和组件,Business 层和组件,企业信息系统(EIS)层。

32. 答：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。

33. 答：三种机制为：通过 setter 方法注入、通过构造方法注入和接口注入。

34. 答：框架：hibernate、spring、struts；
Hibernate 主要用于数据持久化；
Spring 的控制反转能起到解耦合的作用；
Struts 主要用于流程控制。

35. 答：1)不能操作线程和线程 API(线程 API 指非线程对象的方法，如 notify,wait 等)；
2)不能操作 awt；
3)不能实现服务器功能；
4)不能对静态属性存取；
5)不能使用 IO 操作直接存取文件系统；
6)不能加载本地库；
7)不能将 this 作为变量和返回；
8)不能循环调用。

36. (1)JNDI:Java Naming & Directory Interface,JAVA 命名目录服务.主要提供的功能是：提供一个目录系统，让其它各地的应用程序在其上面留下自己的索引，从而满足快速查找和定位分布式应用程序的功能.
(2)JMS：Java Message Service,JAVA 消息服务.主要实现各个应用程序之间的通讯.包括点对点和广播.
(3)JTA：Java Transaction API,JAVA 事务服务.提供各种分布式事务服务.应用程序只需调用其提供的接口即可.
(4)JAF: Java Action FrameWork,JAVA 安全认证框架.提供一些安全控制方面的框架.让开发者通过各种部署和自定义实现自己的个性安全控制策略.
(5)RMI:Remote Method Interface,远程方法调用

II 、 编程题

1. 答：MS SQL Server//第二种连接方式

```
Class.forName( “ com.microsoft.jdbc.sqlserver.SQLServerDriver ” ).
newInstance();
conn =
DriverManager.getConnection(“jdbc:Microsoft:sqlserver://localhost:1433
;DatabaseName=pubs”,” sa”,””);
//Oracle
Class.forName(“oracle.jdbc.driver.OracleDriver”).newInstance();
conn =
DriverManager.getConnection(“jdbc:oracle:thin:@localhost:1521:sid”,ui
d,pwd);
//Mysql
Class.forName(“org.git.mm.mysql.Driver”).newInstance();
conn = DriverManager.getConnection(“jdbc:mysql://localhost:3306/pubs”,”
root”,””);
//处理中文的问题:
jdbc:mysql://localhost:3306/pubs?useUnicode=true&characterEncoding=GB23
12
```

2. 答：JDBC 示例程序如下：

```
public void testJdbc(){
    Connection con = null;
    PreparedStatement ps = null;
    ResultSet rs = null;
    try{
        //step1：注册驱动；
        Class.forName("oracle.jdbc.driver.OracleDriver");
        //step2：获取数据库连接；
        con=DriverManager.getConnection("jdbc:oracle:thin:@192.168.0.39
    :1521:TARENADB", "sd0605","sd0605");
        /************************查询************************/
        //step3：创建 Statement；
        String sql = "SELECT id, fname, lname, age, FROM Person_Tbl";
        ps = con.prepareStatement(sql);
        //step4：执行查询语句，获取结果集；
        rs = ps.executeQuery();
        //step5：处理结果集—输出结果集中保存的查询结果；
        while (rs.next()){
            System.out.print("id = " + rs.getLong("id"));
            System.out.print(" , fname = " + rs.getString("fname"));
            System.out.print(" , lname = " + rs.getString("lname"));
            System.out.print(" , age = " + rs.getInt("age"));
        }
/************************JDBC 修改********************/
```

```
        sql = "UPDATE Person_Tbl SET age=23 WHERE id = ?";
        ps = con.prepareStatement(sql);
        ps.setLong(1, 88);
        int rows = ps.executeUpdate();
        System.out.println(rows + " rows affected.");
    }catch(Exception e){
        e.printStackTrace();
    }finally{
        try{
            con.close(); //关闭数据库连接，以释放资源。
        }catch(Exception e1) {
        }
    }
```

3. 答:一种分页方法

```
<%
    int i=1;
    int numPages=14;
    String pages = request.getParameter("page") ;
    int currentPage = 1;
    currentPage = (pages==null)?(1):{Integer.parseInt(pages)}
    sql = "select count(*) from tables";
    ResultSet rs = DBLink.executeQuery(sql) ;
    while(rs.next()){
        i = rs.getInt(1) ;
    }
    int intPageCount=1;
    intPageCount=(i%numPages==0)?(i/numPages):(i/numPages+1);
    int nextPage;
    int upPage;
    nextPage = currentPage+1;
    if(nextPage>=intPageCount){
        nextPage=intPageCount;
    }
    upPage = currentPage-1;
    if(upPage<=1){
        upPage=1;
    }
    rs.close();
    sql="select * from tables";
    rs=DBLink.executeQuery(sql);
    i=0;
    while((i<numPages*(currentPage-1))&&rs.next()){
```

```
        i++;
    }
%>
//输出内容
//输出翻页连接
合计:<%=currentPage%>/<%=intPageCount%>页
<a href="List.jsp?page=1">第一页</a>
<a href="List.jsp?page=<%=upPage%>">上一页</a>
<%
    for(int j=1;j<=intPageCount;j++){
        if(currentPage!=j){
%>
            <a href="list.jsp?page=<%=j%>">[<%=j%>]</a>
<%
        }else{
            out.println(j);
        }
    }
%>
<a href="List.jsp?page=<%=nextPage%>">下一页</a>
<a href="List.jsp?page=<%=intPageCount%>">最后页</a>
```

六、XML

I、简答题

1.答：1）两种形式：dtd 以及 schema；
2）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)；
3）解析方式：有 DOM,SAX,STAX 等：
DOM:处理大型文件时其性能下降的非常厉害。这个问题是由 DOM 的树结构所造成的，这种结构占用的内存较多，而且 DOM 必须在解析文件之前把整个文档装入内存,适合对 XML 的随机访问；
SAX:不同于 DOM,SAX 是事件驱动型的 XML 解析方式。它顺序读取 XML 文件，不需要一次全部装载整个文件。当遇到像文件开头，文档结束，或者标签开头与标签结束时，它会触发一个事件，用户通过在其回调事件中写入处理代码来处理 XML 文件，适合对 XML 的顺序访问；
STAX:Streaming API for XML (StAX)。

2. 答:用到了数据存储，信息配置两方面。在做数据交换平台时，将不能数据源的数据组装成 XML 文件，然后将 XML 文件压缩打包加密后通过网络传送给接收者，接收解密与解压缩后再同 XML 文件中还原相关信息进行处理。在做软件配置时，利用 XML 可以很方便的进行，软件的各种配置参数都存储在 XML 文件中。

II. 编程题

1. 答：示例代码如下：

```
package xml;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.transform.OutputKeys;
import javax.xml.transform.Transformer;
```

```
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
public class DOMTest{
    private String outFile ="c:\\test.xml";
    public static void main(String args[]) {
        DOMTesttest =new DOMTest();
    try{
        DocumentBuilder builder =DocumentBuilderFactory.newInstance()
        .newDocumentBuilder();
        Document doc= builder.newDocument();
        Element root =doc.createElement("Tarena");
        Element zhang =doc.createElement("张");
        zhang.appendChild(doc.createTextNode("我是张丽芳"));
        root.appendChild(zhang);
        doc.appendChild(root);
        Transformertransformer=TransformerFactory.newInstance()
        .newTransformer();
        //设置 xml 的编码
        transformer.setOutputProperty(OutputKeys.ENCODING,"gb2312");
        //设置缩近格式
        transformer.setOutputProperty(OutputKeys.INDENT,"yes");
        transformer.transform(new DOMSource(doc),
        new StreamResult(test.outFile));
    }catch(Exception e) {
        System.out.println(e.getMessage());
    }
    }
}
```

2. 答：看如下代码,用编码方式加以解决 package test;

```
import java.io.*;
public class DOMTest{
    private String inFile = "c:\people.xml";
    private String outFile = "c:\people.xml";
    public static void main(String args[]){
        new DOMTest();
    }
    public DOMTest(){
        try{
            javax.xml.parsers.DocumentBuilder builder =
```

```
javax.xml.parsers.DocumentBuilderFactory.
newInstance().newDocumentBuilder();
            org.w3c.dom.Document doc = builder.newDocument();
            org.w3c.dom.Element root =doc.createElement("老师");
            org.w3c.dom.Element wang =doc.createElement("王");
            org.w3c.dom.Element liu =doc.createElement("刘");
            wang.appendChild(doc.createTextNode("我是王老师"));
            root.appendChild(wang);
            doc.appendChild(root);
            javax.xml.transform.Transformer transformer =
            javax.xml.transform.TransformerFactory.
        newInstance().newTransformer();
transformer.setOutputProperty( javax.xml.transform.OutputKeys.ENCODING,
"gb2312");
            transformer.setOutputProperty(
            javax.xml.transform.OutputKeys.INDENT, "yes");
transformer.transform(new javax.xml.transform.dom.DOMSource(doc),
new javax.xml.transform.stream.StreamResult(outFile));
        }catch(Exception e){
            System.out.println (e.getMessage());
        }
    }
}
```

3.答：用 SAX 方式解析 XML，XML 文件如下：

```
<?xml version="1.0" encoding="gb2312"?>
<person>
    <name>王小明</name>
    <college>信息学院</college>
    <telephone>6258113</telephone>
    <notes>男,1955 年生,博士，95 年调入海南大学</notes>
</person>
```

事件回调类 SAXHandler.java：

```
import java.io.*;
import java.util.Hashtable;
import org.xml.sax.*;
public class SAXHandler extends HandlerBase{
    private Hashtable table = new Hashtable();
    private String currentElement = null;
    private String currentValue = null;
    public void setTable(Hashtable table){
        this.table = table;
    }
```

```
    public Hashtable getTable(){
        return table;
    }
    public void startElement(String tag, AttributeList attrs) throws
SAXException{
        currentElement = tag;
    }
    public void characters(char[] ch, int start, int length) throws
SAXException{
        currentValue = new String(ch, start, length);
    }
    public void endElement(String name) throws SAXException{
        if (currentElement.equals(name))
            table.put(currentElement, currentValue);
    }
}
```

JSP 内容显示源码,SaxXml.jsp:

```
<HTML>
    <HEAD>
        <TITLE>剖析 XML 文件 people.xml</TITLE>
    </HEAD>
    <BODY>
    <%@                    page                    errorPage="ErrPage.jsp"
contentType="text/html;charset=GB2312" %>
    <%@ page import="java.io.*" %>
    <%@ page import="java.util.Hashtable" %>
    <%@ page import="org.w3c.dom.*" %>
    <%@ page import="org.xml.sax.*" %>
    <%@ page import="javax.xml.parsers.SAXParserFactory" %>
    <%@ page import="javax.xml.parsers.SAXParser" %>
    <%@ page import="SAXHandler" %>
    <%
        File file = new File("c:\people.xml"); FileReader reader = new
    FileReader(file);
        Parser parser;
        SAXParserFactory spf = SAXParserFactory.newInstance();
        SAXParser sp = spf.newSAXParser();
        SAXHandler handler = new SAXHandler();
        sp.parse(new InputSource(reader), handler);
        Hashtable hashTable = handler.getTable();
        out.println("<TABLE  BORDER=2><CAPTION>"  +" 教 师 信 息 表
    </CAPTION>");
        out.println("<TR><TD>姓名</TD>" + "<TD>"
        + (String)hashTable.get(new String("name")) + "</TD></TR>");
```

```
        out.println("<TR><TD>学院</TD>" + "<TD>"
        + (String)hashTable.get(new String("college")) +"</TD></TR>");
        out.println("<TR><TD>电话</TD>" + "<TD>"
        + (String)hashTable.get(new String("telephone")) + "</TD></TR>");
        out.println("<TR><TD>备注</TD>" + "<TD>"
        + (String)hashTable.get(new String("notes")) + "</TD></TR>");
        out.println("</TABLE>");
    %>
    </BODY>
</HTML>
```

七、UML、OOAD

I、简答题

1.答：UML 是标准建模语言；常用图包括：用例图,静态图(包括类图、对象图和包图),行为图,交互图(顺序图,合作图),实现图。

2. 答：Session Facade Pattern：使用 SessionBean 访问 EntityBean；
Message Facade Pattern：实现异步调用；EJB Command Pattern：使用 Command JavaBeans 取代 SessionBean，实现轻量级访问；Data Transfer Object Factory：通过 DTO Factory 简化 EntityBean 数据提供特性；Generic Attribute Access：通过 AttibuteAccess 接口简化 EntityBean 数据提供特性；Business Interface：通过远程（本地）接口和 Bean 类实现相同接口规范业务逻辑一致性；EJB 架构的设计好坏将直接影响系统的性能、可扩展性、可维护性、组件可重用性及开发效率。项目越复杂，项目队伍越庞大则越能体现良好设计的重要性。

3. 答：Java 中的 23 种设计模式：
Factory（工厂模式），Builder（建造模式），Factory Method（工厂方法模式），Prototype（原始模型模式），Singleton（单例模式），Facade（门面模式），Adapter（适配器模式），Bridge（桥梁模式），Composite（合成模式），Decorator（装饰模式），Flyweight（享元模式），Proxy（代理模式），Command（命令模式），Interpreter

（解释器模式），Visitor（访问者模式），Iterator（迭代子模式），Mediator（调停者模式），Memento（备忘录模式），Observer（观察者模式），State（状态模式），Strategy(策略模式),Template Method(模板方法模式),Chain Of Responsibleity（责任链模式）。
工厂模式：工厂模式是一种经常被使用到的模式，根据工厂模式实现的类可以根据提供的数据生成一组类中某一个类的实例，通常这一组类有一个公共的抽象父类并且实现了相同的方法，但是这些方法针对不同的数据进行了不同的操作。首先需要定义一个基类，该类的子类通过不同的方法实现了基类中的方法。然后需要定义一个工厂类，工厂类可以根据条件生成不同的子类实例。当得到子类的实例后，开发人员可以调用基类中的方法而不必考虑到底返回的是哪一个子类的实例。

4. 答：每个模式都描述了一个在我们的环境中不断出现的问题，然后描述了该问题的解决方案的核心。通过这种方式，你可以无数次地使用那些已有的解决方案，无需在重复相同的工作。主要用到了 MVC 的设计模式，用来开发 JSP/Servlet 或者 J2EE 的相关应用；及简单工厂模式等。

5. 答：软件开发中，各个开发阶段不是顺序执行的，应该是并行执行,也就是迭代的意思。这样对于开发中的需求变化，及人员变动都能得到更好的适应。

6. 答：需求分析、系统设计、代码实现、测试、发布

II、 编程题

1. 答：Singleton 模式主要作用是保证在 Java 应用程序中，一个类 Class 只有一个实例存在。举例：定义一个类，它的构造函数为 private 的，它有一个 static 的 private 的该类变量，在类初始化时实例话，通过一个 public 的 getInstance 方法获取对它的引用,继而调用其中的方法。
第一种形式:

```
public class Singleton {
    private Singleton(){
    }
    private static Singleton instance = new Singleton();
    public static Singleton getInstance(){
        return instance;
    }
}
```

第二种形式:

```
public class Singleton {
    private static Singleton instance = null;
    public static synchronized Singleton getInstance(){
        if (instance==null)
```

```
            instance＝new Singleton();
        return instance;
    }
}
```

其他形式:定义一个类，它的构造函数为 private 的，所有方法为 static 的。一般认为第一种形式要更加安全些。

八、Weblogic、Apache、Tomcat 及其它

I、简答题

1.答：在启动 Weblogic 的脚本中（位于所在 Domian 对应服务器目录下的 startServerName），增加 set MEM_ARGS=-Xms32m -Xmx200m，可以调整最小内存为 32M，最大 200M。

2. 答：可以在管理控制台中修改对应服务器的启动模式为开发或产品模式之一，或者修改服务的启动文件或者 commenv 文件，增加 set PRODUCTION_MODE=true。

3. 答:修改服务启动文件,增加WLS_USER和WLS_PW项;也可以在boot.properties文件中增加加密过的用户名和密码。

4.答：保存在此 Domain 的config.xml 文件中，它是服务器的核心配置文件。

5. 答：Domain 目录\服务器目录\applications，将应用目录放在此目录下将可以作为应用访问，如果是Web 应用，应用目录需要满足Web 应用目录要求，jsp 文件可以直接放在应用目录中,Javabean 需要放在应用目录的WEB-INF 目录的classes 目录中，设置服务器的缺省应用将可以实现在浏览器上无需输入应用名。

6. 答：不同类型的 EJB 涉及的配置文件不同，都涉及到的配置文件包括ejbjar.xml,
weblogic-ejb-jar.xml，CMP 实体Bean 一般还需要weblogic-cmprdbms-jar.xml

7. 答：缺省安装中使用DemoIdentity.jks 和DemoTrust.jksKeyStore 实现SSL，需要配置服务器使用Enable SSL，配置其端口，在产品模式下需要从CA 获取私有密钥和数字证书，创建identity 和trust keystore，装载获得的密钥和数字证书。可以配置此SSL 连接是单向还是双向的。

8. 答：可以使用管理控制台，在它的Deployment 中可以查看所有已发布的EJB。

9. 答：CORBA 标准是公共对象请求代理结构(Common Object Request BrokerArchitecture)，由对象管理组织(Object Management Group，缩写为OMG)标准化。它的组成是接口定义语言(IDL),语言绑定(binding:也译为联编)和允许应用程序间互操作的协议。其目的为：用不同的程序设计语言书写在不同的进程中运行，为不同的操作系统开发。解决面向对象的异构应用之间的互操作问题，并提供分布式计算所需的一些其他的服务。ORB 是CORBA 的核心。CORBA 重新调整了客户与服务器之间的关系。客户可以向服务器提出事务请求，同时也可以为下一个请求充当服务器角色。由于CORBA 系统引入了中间件的概念，即事件代理，由中间件完成客户机与服务器之间的通信，使得服务器对于客户机的位置相对透明,取消了原有分布式计算机模型中客户机----服务器之间的一一对应关系，CORBA 客户机可以在运行时动态获得服务对象的位置，并且可以对多个服务对象提交事务请求，所以它极大的推动了分布计算的发展。另外，CORBA 规范约束采用面向对象的分布式方法，以接口定义语言的形式实现对象内部细节的完整封装，从而降低了软件系统的复杂度，增加了软件功能的可重用性。CORBA 提供到C C++ JAVA 等高级语言的映射，极大程度地减小了程序设计语言的依赖性。

10. 答：persistent 方式的MDB 可以保证消息传递的可靠性,也就是如果EJB 容器出现问题而 JMS 服务器依然会将消息在此 MDB 可用的时候发送过来，而nonpersistent 方式的消息将被丢弃。

11. 答：LINUX 实现的就是基于核心轻量级进程的"一对一"线程模型，一个线程实体对应一个核心轻量级进程，而线程之间的管理在核外函数库中实现；GDI 类为图像设备编程接口类库。

12. 答：JDO 是 Java 对象持久化的新的规范，为 java data object 的简称,也是一个用于存取某种数据仓库中的对象的标准化 API。JDO 提供了透明的对象存储，因此对开发人员来说，存储数据对象完全不需要额外的代码（如 JDBC API 的使用）。这些繁琐的例行工作已经转移到 JDO 产品提供商身上，使开发人员解脱出来，从而集中时间和精力在业务逻辑上。另外，JDO 很灵活，因为它可以在任何数据底层上运行。JDBC 只是面向关系数据库（RDBMS）JDO 更通用，提供到任何数据底层的存储功能，比如关系数据库、文件、XML 以及对象数据库（ODBMS）等等，使得应用可移植性更强。

13. 答：Web Service 是基于网络的、分布式的模块化组件，它执行特定的任务，遵守具体的技术规范，这些规范使得 Web Service 能与其他兼容的组件进行互操作；
JAXP(Java API for XML Parsing)定义了在 Java 中使用 DOM, SAX, XSLT 的通用的接口，这样在你的程序中你只要使用这些通用的接口，当你需要改变具体的实现时候也不需要修改代码；
JAXM(Java API for XML Messaging)是为 SOAP 通信提供访问方法和传输机制的 API;
WSDL 是一种 XML 格式，用于将网络服务描述为一组端点，这些端点对包含面向文档信息或面向过程信息的消息进行操作。这种格式首先对操作和消息进行抽象描述，然后将其绑定到具体的网络协议和消息格式上以定义端点。相关的具体端点即组合成为抽象端点（服务）；
SOAP 即简单对象访问协议(Simple Object Access Protocol)，它是用于交换 XML 编码信息的轻量级协议；
UDDI 的目的是为电子商务建立标准；UDDI 是一套基于 Web 的、分布式的、为 Web Service 提供的、信息注册中心的实现标准规范，同时也包含一组使企业能将自身提供的 Web Service 注册，以使别的企业能够发现的访问协议的实现标准

九、C、C++

I、 简答题

1. 答：输出为：false、false、true。

2. 答：for 循环中的变量 i 的类型不应定义为 vector<int>::size_type，因为该

类型为无符号数值类型，故循环条件将恒成立，为死循环，应将其类型定义为有符号的 int 类型。

3. 答：运算符中两个可选值的类型不同。

4. 答：for 循环中的 if 语句后的 array.erase(itor)语句，它将迭代器 itor 所指向的元素删除后会自动下移一位，故应在其后加上语句：itor--;

5. 答：在 upperCase 方法中，for 循环的 sizeof(str)的值将总是 4，所以该方法只能将参数中的字符串的前四个字符转换成大写字母。

6. 答：能；

7. 答：main 函数的返回类型应为 int；不能对 b 调用 fun（）方法。

8. 答：输出不是 0；

9. 答：空类中默认包含的成员函数如下:

```
class Empty{
public:
    Empty();
    //缺省构造函数
    Empty( const Empty& );
    //拷贝构造函数
    ~Empty();
    //析构函数
    Empty& operator=( const Empty& );
    //赋值运算符
    Empty* operator&();
    //取址运算符
    const Empty* operator&() const;
    //取址运算符 const
};
```

10. 答：a）class B : public A{……}//B 公有继承自 A，可以是间接继承的
b）class B{operator A();}//B 实现了隐式转化为 A 的转化
c）class A{ A(const B&);}//A 实现了 non-explicit 的参数为 B 构造函数(可以有其他带带默认值的参数)
d）A& operator= (const A&);//赋值操作，虽不是正宗的隐式类型转换，但也可以勉强算一个

11. 答：第一处输出 false，第二处输出 true。

II、编程题

1. 答：代码如下：

```
include<iostream>
#include<fstream>
using namespace std;
int main(){
    ifstream fin("t.txt");
    if(!fin){
        cout<<"can't open file"<<endl;
        return -1;
    }
    int count = 0;
    char buf[256];
    memset(buf, 0, 256);
    while(1){
        fin2>>buf;
        if(fin2.eof()) {
            break;
        }
        count++;
    }
    cout<<"The number of the words is : "<<count<<endl;
    fin2.close();
    return 0;
}
```

2. 答：代码如下：

```
void* mymemcpy(void* dest, const void* src, size_t count){
    char* pdest = static_cast<char*>(dest);
    const char* psrc = static_cast<const char*>(src);
    if(pdest>psrc && pdest<psrc+count){
        //能考虑到这种情况就行了
        for(size_t i=count-1; i!=-1; --i){
            pdest[i] = psrc[i];
        }
    }else{
        for(size_t i=0; i<count; ++i){
            pdest[i] = psrc[i];
```

```
        }
    }
    return dest;
}
int main(){
    char str[] = "0123456789";
    mymemcpy(str+1, str+0, 9);
    cout << str << endl;
    //将输出"0012345678"
    return 0;
}
```

十、英语题

I、单选题

1、（D）	2、（B）	3、（D）
4、（C）	5、（A）	6、（B）
7、（B）	8、（A）	9、（D）
10、（D）	11、（E）	12、（A）
13、（D）	14、（D）	15、（C）
16、（C）	17、（C）	18、（C）
19.C	20.A	21.C
22.D	23.C	24.AB
25.D	26.C	27.A
28.A	29.C	30.B
31.B	32.A	33.B
34.D	35.D	36.B
37.C		

II、　多选题

1、（BC）	2、（BC）	3、（ABCB）
4、（BC）	5、（BC）	6、（AB）