一篇完整的网易笔试题

发布时间:2010-11-04 来源:应届毕业生求职网

卷(研发类笔试题)

第一部分(必做)：计算机科学基础

1. (单选)软件设计中模块划分应该遵循的准则是:

A.低内聚低耦合 **B.高内聚低耦合** C.低内聚高耦合 D.高内聚高耦合

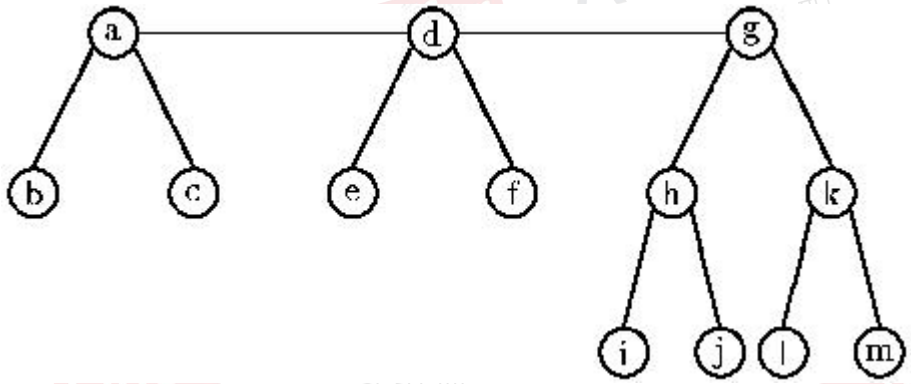
2. (单选)最坏情况下时间复杂度不是 n(n-1)/2 的排序算法是:

A.快速排序(n^2) B.冒泡排序(n^2) C.直接插入排序(n^2) **D.堆排序(nlogn)**

3. 哈希表中解决冲突的方法通常可以分为 open addressing 和 chaining 两类，请分别解释这两类冲突解决方法的大致实现原理

//见书

4. 简单的链表结构拥有很好的插入 删除节点性能，但随机定位(获取链表第 n 个节点)操作性能不佳，请你设计一种改进型的链表结构优化随机定位操作的性能，给出设计思路及其改进后随机定位操作的时间复杂度



具体参见 Purely Functional Random-Access Lists.pdf

大概地说，节点构成多棵相连的完全二叉树来表示（为了不浪费节点），存取顺序为前序遍历。

复杂度为 O(log n)

这里有代码

http://www.cs.oberlin.edu/~jwalker/ra-list/

5. 什么是 NP 问题?列举典型的 NP 问题(至少两个)?对于一个给定的问题你通常如何判断它是否为 NP 问题?

NP(Nondeterministic Polynomial 问题)。但是对于很多问题来说，他们找不到一个多项式的解决方法，

，只能“尝试”很多种方案

才能够得出一个答案，这显然是很费时的，这种问题未 NP 问题。

NPC(NP Complete)问题，可以这么认为，这种问题只有把解域里面的所有可能都穷举了之后才能得出答案，这样的问题是 NP 里面最难

旅行商问题 TSP Travelling Salesman Problem

子集和问题

Hamilton 回路

要满足两个条件：

1.封闭的环

2.是一个连通图，且图中任意两点可达

经过图（有向图或无向图）中所有顶点一次且仅一次的通路称为哈密顿通路。

经过图中所有顶点一次且仅一次的回路称为哈密顿回路。

最大团问题

6. 以下是一个 tree 的遍历算法，queue 是 FIFO 队列，请参考下面的 tree，选择正确的输出.

```
1
/ \
2     3
/ \ / \
4 5 6 7
queue.push(tree.root)
while(true){
node=queue.pop();
output(node.value);//输出节点对应数字
if(null==node)
      break;
for(child_node in node.children){
      queue.push(child_node);
}
}
```

A. 1234567

B. 1245367

C. 1376254

D. 1327654

第二部分(选作)：C/C++程序设计

1. 有三个类 A B C 定义如下，请确定 sizeof(A) sizeof(B) sizeof(C)的大小顺序，并给出理由

```
struct A{
A() {}
~A() {}
int m1;
int m2;
};
```

```
struct B{
B() {}
~B() {}
int m1;
char m2;
static char m3;
};
struct C{
C() {}
virtual~C() {}
int m1;
short m2;
};
//8 //8 //12
```

2. 请用 C++实现以下 print 函数,打印链表 I 中的所有元素, 每个元素单独成一行

```
void print(const std::list<int> &I){
}
```

```
#include <list>
#include <iostream>

void print(const std::list<int> &I)
{
    std::list<int>::const_iterator iter;
    for(iter=I.begin();iter!=I.end();iter++)
        printf("%d\n",*iter);
}


int main()
{
std::list<int> L;
L.push_back(1);
L.push_back(2);
L.push_back(3);
print(L);
return 0;
```

}

3. 假设某C工程包含a.c和b.c两个文件,在a.c中定义了一个全局变量foo，在b.c中想访问这一变量时该怎么做?

增加一个a.h,写上 extern int foo,然后让a.c和b.c都包含a.h

4. C++中的new操作符通常完成两个工作，分配内存及其调用相应的构造函数初始化
请问:
1）如何让new操作符不分配内存，只调用构造函数?
2）这样的用法有什么用?

解答：（要求new显式调用构造函数，但不分配内存。）

题目要求不能生成内存 还要调用构造函数 说明这个类里面没有对内部操作 但可以对外部操作 比如static的数

摘录：如果我是用new分配对象的，可以显式调用析构函数吗？
可能不行。除非你使用定位放置 new.

```
class Fred
{public:
    Fred()
    {
        cout<<"fuck";
        }

    };
int main()
{

   Fred*f=new((void*)10000)Fred();
```

```
    system("pause");
}
```

其中这个 10000 可以是任意数，但不能为 0

2）

定位放置 new（placement new）有很多作用。最简单的用处就是将对象放置在内存中的特殊位置。这是依靠 new 表达式部分的指针参数的位置来完成的：

```
#include <new>          // 必须 #include 这个，才能使用 "placement new"
#include "Fred.h"       // class Fred 的声明

void someCode()
{
   char memory[sizeof(Fred)];     // Line #1
   void* place = memory;          // Line #2

   Fred* f = new(place) Fred();   // Line #3 (详见以下的"危险")
   // The pointers f and place will be equal

   // ...
}
```

Line #1 在内存中创建了一个 sizeof(Fred)字节大小的数组，足够放下 Fred 对象。Line #2 创建了一个指向这块内存的首字节的 place 指针（有经验的 C 程序员会注意到这一步是多余的，这儿只是为了使代码更明显）。Line #3 本质上只是调用了构造函数 Fred::Fred()。Fred 构造函数中的 this 指针将等于 place。因此返回的 f 将等于 place。Line #3 本质上只是调用了构造函数 Fred::Fred()。

****

placement new 的作用就是：创建对象但是不分配内存，而是在已有的内存块上面创建对象。

用于需要反复创建并删除的对象上，可以降低分配释放内存的性能消耗。

```
#include <iostream>
#include <new>

const intchunk = 16;
class Foo
{
public :
int val( ) { return _val; }
Foo( ) { _val = 0; }
```

```
private :
int_val;
};

//预分配内存，但没有 Foo 对象
char*buf = new char[ sizeof(Foo) * chunk ];

int
main( void )
{
//在 buf 中创建一个 Foo 对象
Foo*pb = new (buf) Foo;

//检查一个对象是否被放在 buf 中
if ( pb->val() == 0 )
{
cout <<"new expressio worked!" <<endl;
}

//到这里不能再使用 pb
delete[] buf;

return 0;
}
```

http://www.bearcave.com/software/c++_mem.html

```
#include <iostream>
using namespace std;

class a
{
    int mV;

public:
    void *operator new(size_t size,void *mem)
    {
```

```
        return mem;
    }

    a(int v)
    {
        printf("a constructor\n");
        mV=v;
    }

    ~a()
    {
        printf("a destructor\n");
    }

    void pr()
    {
        printf("%d\n",mV);
    }

};


int main()
{
    a a1(1);
    a1.pr();
    void *p=&a1;
    a *a2=new(p)a(2);
    a2->pr();
    a1.pr();
    return 0;
}
```

5. 下面这段程序的输出是什么?为什么?

```
class A{
public:
A(){p();}
virtual void p(){print("A")}
```

```
virtual ~A(){p();}
};
class B{
public:
B(){p();}
void p(){print("B")}
~B(){p();}
};
int main(int, char**){
A* a=new B();
delete a;
}
```

6. 什么是 C++ Traits? 并举例说明

http://accu.org/index.php/journals/442

Think of a trait as a small object whose main purpose is to carry information used by another object or algorithm to determine "policy" or "implementation details"

Traits 可以说是一个小小的信息体，其它对象或算法可以根据它来选择不同的执行政策功实现细节。

比如 std 中 `numeric_limits< double >::max()`

第三部分(选作): JAVA 程序设计

1. (单选)以下 Java 程序运行的结构是:

```
public class Tester{
public static void main(String[] args){
    Integer var1=new Integer(1);
    Integer var2=var1;
    doSomething(var2);
    System.out.print(var1.intValue());
    System.out.print(var1==var2);
}
```

```
public static void doSomething(Integer integer){
integer=new Integer(2);
}
}
```

A. 1true

B. 2true

C. 1false

D. 2false

java 中的值传递和引用传递

值传递：方法调用时，实际参数把它的值传递给对应的形式参数，方法执行中形式参数值的改变不影响实际参 数的值。

引用传递：也称为传地址。方法调用时，实际参数的引用(地址，而不是参数的值)被传递给方法中相对应的形式参数，在方法执行中，对形式参数的操作实际上就是对实际参数的操作，方法执行中形式参数值的改变将会影响实际参数的值。

下面举例说明：

传值---传递基本数据类型参数

```
public   class      PassValue{
  static void exchange(int a, int b){//静态方法，交换 a,b 的值
     int temp;
     temp = a;
     a = b;
     b = temp;
  }
  public static void main(String[] args){
    int i = 10;
```

```
        int j = 100;
       System.out.println("before call: " + "i=" + i + "\t" + "j = " + j);//调用前
        exchange(i, j);                                    //值传递，main 方法只能调用静态方法
        System.out.println("after call: " + "i=" + i + "\t" + "j = " + j);//调用后
    }
}
```

运行结果：

```
        before call: i = 10      j = 100
        after    call: i = 10      j = 100
```

说明：调用 exchange(i, j)时，实际参数 i，j 分别把值传递给相应的形式参数 a，b,在执行方法 exchange()时，形式参数 a，b 的值的改变不影响实际参数 i 和 j 的值，i 和 j 的值在调用前后并没改变。

引用传递---对象作为参数

如果在方法中把对象（或数组）作为参数，方法调用时，参数传递的是对象的引用（地址），即在方法调用时，实际参数把对对象的引用（地址）传递给形式参数。这是实际参数与形式参数指向同一个地址，即同一个对象（数组），方法执行时，对形式参数的改变实际上就是对实际参数的改变，这个结果在调用结束后被保留了下来。

```
class Book{
    String name;
    private folat price;
    Book(String n,    float ){              //构造方法
        name = n;
        price = p;
    }
    static  void   change(Book a_book,    String n,    float p){    //静态方法，对象作为参数
        a_book.name = n;
        a_book.price = p;
    }
    public void output(){       //实例方法，输出对象信息
        System.out.println("name: " + name + "\t" + "price: " + price);
    }
```

```
}
 public class PassAddr{
   public static void main(String [] args){
     Book b = new Book("java2",  32.5f);
     System.out.print("before call:\t");      //调用前
     b.output();
     b.change(b,  "c++",  45.5f);          //引用传递，传递对象 b 的引用，修改对象 b 的值
     System.out.print("after call:\t");          //调用后
     b.output();
   }
}
```

运行结果：

```
before   call:   name:java2      price:32.5
after    call:   name:c++        price:45.5
```

说明：调用 change(b,"c++",45.5f)时，对象 b 作为实际参数，把引用传递给相应的形式参数 a_book,实际上 a_book 也指向同一个对象，即该对象有两个引用名：b 和 a_book。在执行方法 change()时，对形式参数 a_book 操作就是对实际参数 b 的操作。

而题目中的 Integer 属于基础类型，和 int 一样对待

2. （单选）往 OuterClass 类的代码段中插入内部类声明，哪一个是正确的：

```
public class OuterClass{
private float f=1.0f;
//插入代码到这里
}
```

A.

```
class InnerClass{
```

```
public static float func(){return f;} //错，static 函数不能在非静态的内嵌类中定义，另外不能引用非静态的 f
}
B.
abstract class InnerClass{
public abstract float func(){} //错，abstract 函数无 body
}
C.
static class InnerClass{
protected static float func(){return f;} ///错，不能引用非静态的 f
}
D.
public class InnerClass{
static static float func(){return f;} ///错，
}
```

3. Java 中的 interface 有什么作用？举例说明哪些情况适合用 interface，哪些情况下适合用抽象类.

When To Use Interfaces

An **interface** allows somebody to start from scratch to implement your **interface** or implement your **interface** in some other code whose original or primary purpose was quite different from your **interface**. To them, your **interface** is only incidental, something that have to add on to the their code to be able to use your package. The disadvantage is every method in the interface must be **public**. You might not want to expose everything.

When To Use Abstract classes

An **abstract** class, in contrast, provides more structure. It usually defines some default implementations and provides some tools useful for a full implementation. The catch is, code using it **must** use your class as the base. That may be highly inconvenient if the other programmers wanting to use your package have already developed their own class hierarchy independently. In Java, a class can inherit from only one base class.

更具体

http://mindprod.com/jgloss/interfacevsabstract.html

4. Java 多线程有哪几种实现方式？Java 中的类如何保证线程安全？请说明 ThreadLocal 的用法和适用场景

书

java synchronized 详解(一)

文章分类: Java 编程

Java 语言的关键字，当它用来修饰一个方法或者一个代码块的时候，能够保证在同一时刻最多只有一个线程执行该段代码。

一、当两个并发线程访问同一个对象 object 中的这个 synchronized(this)同步代码块时，一个时间内只能有一个线程得到执行。另一个线程必须等待当前线程执行完这个代码块以后才能执行该代码块。

二、然而，当一个线程访问 object 的一个 synchronized(this)同步代码块时，另一个线程仍然可以访问该 object 中的非 synchronized(this)同步代码块。

三、尤其关键的是，当一个线程访问 object 的一个 synchronized(this)同步代码块时，其他线程对 object 中所有其它 synchronized(this)同步代码块的访问将被阻塞。

四、第三个例子同样适用其它同步代码块。也就是说，当一个线程访问 object 的一个 synchronized(this)同步代码块时，它就获得了这个 object 的对象锁。结果，其它线程对该 object 对象所有同步代码部分的访问都被暂时阻塞。

五、以上规则对其它对象锁同样适用.

举例说明：

一、当两个并发线程访问同一个对象 object 中的这个 synchronized(this)同步代码块时，一个时间内只能有一个线程得到执行。另一个线程必须等待当前线程执行完这个代码块以后才能执行该代码块。

```
package ths;

public class Thread1 implements Runnable {
    public void run() {
        synchronized(this) {
            for (int i = 0; i < 5; i++) {
                System.out.println(Thread.currentThread().getName() + " synchronized loop " + i);
            }
        }
    }
    public static void main(String[] args) {
        Thread1 t1 = new Thread1();
        Thread ta = new Thread(t1, "400 电话 ");
        Thread tb = new Thread(t1, "800 电话 ");
        ta.start();
        tb.start();
    }
}
```

结果：

```
400 电话 synchronized loop 0
400 电话 synchronized loop 1
400 电话 synchronized loop 2
400 电话 synchronized loop 3
400 电话 synchronized loop 4
800 电话 synchronized loop 0
800 电话 synchronized loop 1
800 电话 synchronized loop 2
800 电话 synchronized loop 3
800 电话 synchronized loop 4
```

二、然而，当一个线程访问 object 的一个 synchronized(this)同步代码块时，另一个线程仍然可以访问该 object 中的非 synchronized(this)同步代码块。

```
package ths;

public class Thread2 {
    public void m4t1() {
        synchronized(this) {
            int i = 5;
            while( i-- > 0) {
                System.out.println(Thread.currentThread().getName() + " : " + i);
                try {
                    Thread.sleep(500);
                } catch (InterruptedException ie) {
                }
            }
        }
    }
    public void m4t2() {
        int i = 5;
        while( i-- > 0) {
            System.out.println(Thread.currentThread().getName() + " : " + i);
            try {
                Thread.sleep(500);
            } catch (InterruptedException ie) {
            }
        }
    }
    public static void main(String[] args) {
        final Thread2 myt2 = new Thread2();
        Thread t1 = new Thread(   new Runnable() {   public void run() {   myt2.m4t1();   }   }, "t1"   );
        Thread t2 = new Thread(   new Runnable() {   public void run() { myt2.m4t2();   }   }, "t2"   );
        t1.start();
        t2.start();
    }
}
```

结果：

```
    t1 : 4
    t2 : 4
    t1 : 3
    t2 : 3
    t1 : 2
    t2 : 2
    t1 : 1
```

t2 : 1
t1 : 0
t2 : 0

三、尤其关键的是，当一个线程访问 object 的一个 synchronized(this)同步代码块时，其他线程对 object 中所有其它 synchronized(this)同步代码块的访问将被阻塞(http://www.my400800.cn)。

```
//修改 Thread2.m4t2()方法：
public void m4t2() {
    synchronized(this) {
        int i = 5;
        while( i-- > 0) {
            System.out.println(Thread.currentThread().getName() + " : " + i);
            try {
                Thread.sleep(500);
            } catch (InterruptedException ie) {
            }
        }
    }
}
```

结果：

```
t1 : 4
t1 : 3
t1 : 2
t1 : 1
t1 : 0
t2 : 4
t2 : 3
t2 : 2
t2 : 1
t2 : 0
```

四、第三个例子同样适用其它同步代码块。也就是说，当一个线程访问 object 的一个 synchronized(this)同步代码块时，它就获得了这个 object 的对象锁。结果，其它线程对该 object 对象所有同步代码部分的访问都被暂时阻塞。

```
//修改 Thread2.m4t2()方法如下：
public synchronized void m4t2() {
    int i = 5;
    while( i-- > 0) {
        System.out.println(Thread.currentThread().getName() + " : " + i);
        try {
            Thread.sleep(500);
```

```
            } catch (InterruptedException ie) {
            }
        }
    }
```

结果：

```
    t1 : 4
    t1 : 3
    t1 : 2
    t1 : 1
    t1 : 0
    t2 : 4
    t2 : 3
    t2 : 2
    t2 : 1
    t2 : 0
```

五、以上规则对其它对象锁同样适用:

```
package ths;

public class Thread3 {
    class Inner {
        private void m4t1() {
            int i = 5;
            while(i-- > 0) {
                System.out.println(Thread.currentThread().getName() + " : Inner.m4t1()=" + i);
                try {
                    Thread.sleep(500);
                } catch(InterruptedException ie) {
                }
            }
        }
        private void m4t2() {
            int i = 5;
            while(i-- > 0) {
                System.out.println(Thread.currentThread().getName() + " : Inner.m4t2()=" + i);
                try {
                    Thread.sleep(500);
                } catch(InterruptedException ie) {
                }
            }
        }
    }
    private void m4t1(Inner inner) {
```

```
        synchronized(inner) { //使用对象锁
        inner.m4t1();
    }

    private void m4t2(Inner inner) {
        inner.m4t2();
    }

    public static void main(String[] args) {
        final Thread3 myt3 = new Thread3();
        final Inner inner = myt3.new Inner();
        Thread t1 = new Thread( new Runnable() {public void run() { myt3.m4t1(inner);} }, "t1");
   Thread t2 = new Thread( new Runnable() {public void run() { myt3.m4t2(inner);} }, "t2");
    t1.start();
    t2.start();
  }
}
```

结果：

尽管线程 t1 获得了对 Inner 的对象锁，但由于线程 t2 访问的是同一个 Inner 中的非同步部分。所以两个线程互不干扰。

```
    t1 : Inner.m4t1()=4
    t2 : Inner.m4t2()=4
    t1 : Inner.m4t1()=3
    t2 : Inner.m4t2()=3
    t1 : Inner.m4t1()=2
    t2 : Inner.m4t2()=2
    t1 : Inner.m4t1()=1
    t2 : Inner.m4t2()=1
    t1 : Inner.m4t1()=0
    t2 : Inner.m4t2()=0
```

现在在 Inner.m4t2()前面加上 synchronized：

```
    private synchronized void m4t2() {
        int i = 5;
        while(i-- > 0) {
            System.out.println(Thread.currentThread().getName() + " : Inner.m4t2()=" + i);
            try {
                Thread.sleep(500);
            } catch(InterruptedException ie) {
            }
```

```
        }
    }
```

结果：

尽管线程 t1 与 t2 访问了同一个 Inner 对象中两个毫不相关的部分,但因为 t1 先获得了对 Inner 的对象锁，所以 t2 对 Inner.m4t2()的访问也被阻塞，因为 m4t2()是 Inner 中的一个同步方法。

```
    t1 : Inner.m4t1()=4
    t1 : Inner.m4t1()=3
    t1 : Inner.m4t1()=2
    t1 : Inner.m4t1()=1
    t1 : Inner.m4t1()=0
    t2 : Inner.m4t2()=4
    t2 : Inner.m4t2()=3
    t2 : Inner.m4t2()=2
    t2 : Inner.m4t2()=1
    t2 : Inner.m4t2()=0
```

早在 JDK 1.2 的版本中就提供 java.lang.ThreadLocal，ThreadLocal 为解决多线程程序的并发问题提供了一种新的思路。使用这个工具类可以很简洁地编写出优美的多线程程序。

ThreadLocal 很容易让人望文生义，想当然地认为是一个“本地线程”。其实，ThreadLocal 并不是一个 Thread，而是 Thread 的局部变量，也许把它命名为 ThreadLocalVariable 更容易让人理解一些。

当使用 ThreadLocal 维护变量时，ThreadLocal 为每个使用该变量的线程提供独立的变量副本，所以每一个线程都可以独立地改变自己的副本，而不会影响其它线程所对应的副本。

从线程的角度看，目标变量就象是线程的本地变量，这也是类名中“Local”所要表达的意思。

线程局部变量并不是 Java 的新发明，很多语言（如 IBM IBM XL FORTRAN）在语法层面就提供线程局部变量。在 Java 中没有提供在语言级支持，而是变相地通过 ThreadLocal 的类提供支持。

所以，在 Java 中编写线程局部变量的代码相对来说要笨拙一些，因此造成线程局部变量没有在 Java 开发者中得到很好的普及。

ThreadLocal 的接口方法

ThreadLocal 类接口很简单，只有 4 个方法，我们先来了解一下：

void set(Object value)

设置当前线程的线程局部变量的值。

public Object get()

该方法返回当前线程所对应的线程局部变量。

public void remove()

将当前线程局部变量的值删除，目的是为了减少内存的占用，该方法是 JDK 5.0 新增的方法。需要指出的是，当线程结束后，对应该线程的局部变量将自动被垃圾回收，所以显式调用该方法清除线程的局部变量并不是必须的操作，但它可以加快内存回收的速度。

protected Object initialValue()

返回该线程局部变量的初始值，该方法是一个 protected 的方法，显然是为了让子类覆盖而设计的。这个方法是一个延迟调用方法，在线程第 1 次调用 get()或 set(Object)时才执行，并且仅执行 1 次。ThreadLocal 中的缺省实现直接返回一个 null。

值得一提的是，在 JDK5.0 中，ThreadLocal 已经支持泛型，该类的类名已经变为 ThreadLocal。API 方法也相应进行了调整，新版本的 API 方法分别是 void set(T value)、T get()以及 T initialValue()。

ThreadLocal 是如何做到为每一个线程维护变量的副本的呢？其实实现的思路很简单：在 ThreadLocal 类中有一个 Map，用于存储每一个线程的变量副本，Map 中元素的键为线程对象，而值对应线程的变量副本。

实例

下面，我们通过一个具体的实例了解一下 ThreadLocal 的具体使用方法。

代码清单 2 SequenceNumber

```
package com.baobaotao.basic;
public class SequenceNumber {
①通过匿名内部类覆盖 ThreadLocal 的 initialValue()方法，指定初始值
private static ThreadLocal seqNum = new ThreadLocal(){
public Integer initialValue(){
return 0;
}
```

```
};
②获取下一个序列值
public int getNextNum(){
seqNum.set(seqNum.get()+1);
return seqNum.get();
}
public static void main(String[] args)
{
SequenceNumber sn = new SequenceNumber();
③ 3 个线程共享 sn，各自产生序列号
TestClient t1 = new TestClient(sn);
TestClient t2 = new TestClient(sn);
TestClient t3 = new TestClient(sn);
t1.start();
t2.start();
t3.start();
}
private static class TestClient extends Thread
{
private SequenceNumber sn;
public TestClient(SequenceNumber sn) {
this. sn = sn;
}
public void run()
{
for (int i = 0; i < 3; i++) {④每个线程打出 3 个序列值
System.out.println("thread["+Thread.currentThread().getName()+
"] sn["+sn.getNextNum()+"]");
}
}
}
}
```

通常我们通过匿名内部类的方式定义 ThreadLocal 的子类，提供初始的变量值，如例子中①处所示。TestClient 线程产生一组序列号，在③处，我们生成 3 个 TestClient，它们共享同一个 SequenceNumber 实例。运行以上代码，在控制台上输出以下的结果：

```
thread[Thread-2] sn[1]
thread[Thread-0] sn[1]
```

thread[Thread-1] sn[1]
thread[Thread-2] sn[2]
thread[Thread-0] sn[2]
thread[Thread-1] sn[2]
thread[Thread-2] sn[3]
thread[Thread-0] sn[3]
thread[Thread-1] sn[3]

考察输出的结果信息，我们发现每个线程所产生的序号虽然都共享同一个 SequenceNumber 实例，但它们并没有发生相互干扰的情况，而是各自产生独立的序列号，这是因为我们通过 ThreadLocal 为每一个线程提供了单独的副本。

5. 线程安全的 Map 在 JDK 1.5 及其更高版本环境 有哪几种方法可以实现?

Map 线程安全几种实现方法

文章分类: Java 编程 关键字: map 线程安全

如果需要使 Map 线程安全，大致有这么四种方法：

1、使用 synchronized 关键字，代码如下

Java 代码

```
1.    synchronized(anObject) {
2.        value = map.get(key);
3.    }
```

2、使用 JDK1.5 提供的锁（java.util.concurrent.locks.Lock）。代码如下

Java 代码

```
1.    lock.lock();
2.    value = map.get(key);
3.    lock.unlock();
```

3、使用 JDK1.5 提供的读写锁（java.util.concurrent.locks.ReadWriteLock）。代码如下

Java 代码

```
1.    rwlock.readLock().lock();
2.    value = map.get(key);
3.    rwlock.readLock().unlock();
```

这样两个读操作可以同时进行，理论上效率会比方法 2 高。

4、使用 JDK1.5 提供的 java.util.concurrent.ConcurrentHashMap 类。该类将 Map 的存储空间分为若干块，每块拥有自己的锁，大大减少了多个线程争夺同一个锁的情况。代码如下

value = map.get(key); //同步机制内置在 get 方法中

6.

1) 简述 Java ClassLoader 的模型，说明其层次关系及其类加载的主要流程即可.

2) TypeA.class 位于 classpath 下，/absolute_path/TypeA.class 为其在文件系统中的绝对路径，且类文件小于 1k，MyClassLoader 为一个自定义的类加载器，下面的这段类加载程序是否正确，如果有错请指出哪一行有错，简述理由

```
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
public class Tester{
public static void main(String[] args){
    MyClassLoader cl1=new MyClassLoader();
    try{
      File f=new File("/absolute_path/TypeA.class");
      byte[] b=new byte[1024];
      InputStream is=new FileInputStream(f);
      int I=is.read(b);
      Class c=cl1.defineMyClass(null,b,0,1);
      TypeA a=(TypeA)c.newInstance();
    }catch(Exception e){
```

```
            e.printStacktrace();
        }
    }
}
```

第四部分(选作)：Linux 应用与开发

1. 写出完成以下功能的 Linux 命令：

1) 在当前目录及其子目录所有的.cpp 文件中查找字符串"example"，不区分大小写；

2) 使用 sed 命令，将文件 xyz 中的单词 AAA 全部替换为 BBB；

3) 用一条命令创建 aa bb cc 三个子目录

4) mount cdrom.iso 至/dev/cdrom 目录

5) 设置 ulimit 使得程序在 Segment fault 等严重错误时可以产生 coredump；

2. 设 umask 为 002，则新建立的文件的权限是什么？

A. -rw-rwr--

B. rwxrwx-w-

C. -------w-

D. rwxrwxr-x

3. 用户 HOME 目录下的.bashrc 和.bash_profile 文件的功能有什么区别？

4. 写出完成以下功能的 gdb 命令(可以使用命令简写形式)：

1) 使用 gdb 调试程序 foo，使用 coredump 文件 core.12023；

2) 查看线程信息

3) 查看调用堆栈

4) 在类 ClassFoo 的函数 foo 上设置一个断点

5) 设置一个断点，当表达式 expr 的值被改变时触发

5.

1) 例举 Linux 下多线程编程常用的 pthread 库提供的函数名并给出简要说明(至少给出 5 个)

2) pthread 库提供哪两种线程同步机制，列出主要 API

3) 使用 pthread 库的多线程程序编译时需要加什么连接参数？

第五部分(选作)：Windows 开发

1. DC(设备上下文)有哪几类？区别在哪里？

2. 碰撞检测是游戏中经常要用到的基本技术 对于二维情况，请回答以下问题：

1)．如何判断一个点在一个多边形内

2). 如何判断两个多边形相交

3). 如何判断两个点集所形成的完全图所围的区域是否相交

3. PostMessage SendMessage 和 PostThreadMessage 的区别是什么

4. 什么叫 Alpha 混合？当前流行的图片格式中哪些支持 alpha 通道？Layered Window 和普通 Window 有什么区别？

5. 如果要实现一个多线程(非 MFC)程序，选择多线程 CRT，创建线程的时候应该用 CreateThread 还是_beginthreadex()，为什么？

第六部分(选作)：数据库开发

1. 基于哈希的索引和基于树的索引有什么区别？

2. User 表用于记录用户相关信息，Photo 表用于记录用户的照片信息，两个表的定义如下：

```
CREATE TABLE User( --用户信息表
UserId bigint,      --用户唯一 id
Account varchar(30)      --用户唯一帐号
);
CREATE TABLE Photo(      --照片信息表
PhotoId bigint, --照片唯一 id
UserId bigint,      --照片所属用户 id
AccessCount int,      --访问次数
Size bigint      --照片文件实际大小
)
```

1) 请给出 SQL 打印帐号为"dragon"的用户访问次数最多的 5 张照片的 id;

2) 给出 SQL 打印拥有总的照片文件大小(total_size)最多的前 10 名用户的 id，并根据 total_size 降序排列

3) 为优化上面两个查询，需要在 User 和 Photo 表上建立什么样的索引？

4) 简述索引对数据库性能的影响？

3. 什么是两阶段提交协议？

4. 数据库事务基本概念：

1) 什么是事务的 ACID 性质？

2) SQL 标准中定义的事务隔离级别有哪四个？

3) 数据库中最常用的是哪两种并发控制协议？

4) 列举你所知的数据库管理系统中采用的并发控制协议

5. 数据库中有表 User(id, name, age):

表中数据可能会是以下形式:

id　　name　　age

001 张三　　56

002 李四　　25

003 王五　　56

004 赵六　　21

005 钱七　　39

006 孙八　　56

.....

由于人员年龄有可能相等，请写出 SQL 语句，用于查询 age 最大的人员中，id 最小的一个记录

6. 并发访问数据库时常使用连接池，请问使用连接池的好处是什么？对于有多台应用服务器并发访问一台中心数据库的情况，数据库访问往往成为系统瓶颈，请问在应用服务器上设计和使用连接池时该注意哪些问题，以保证系统的可靠性 正确性和整体性能. 假设每台应用服务器都执行相同的任务并且负载均衡.

第七部分(选作): Web 开发

1. 以下哪一条 Javascript 语句会产生运行错误:

A. var obj=( );

B. var obj=[ ];

C. var obj={ };

D. var obj=/ /;

2. 如下页面代码(示例代码 DOCTYPE 为 Strict)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="zh" lang="zh">
<head>
<title>测试</title>
<meta http-equiv="content-type" content="text/html; charset=gbk" />
<meta http-equiv="content-style-type" content="text/cee" />
<meta http-equiv="content-script-type" content="text/javascript" />
<script type="text/css">
*{margin:0; padding:0}
```

```
    html{width:100%; height:100%;
      overflow:scroll; overflow-x:auto;
      text-align:center; border:0}
    .test{height:200px}
</script>
</head>
<body>
<div class="text"> </div>
</body>
</html>
```

假设 a.jpg 图片的规格是 200pxX100px，请给出当前背景图片距 div.a 顶部距离的计算方式和结果(css)

3. HTTP 协议相关知识

A) 常见的 HTTP Request 头字段有哪些?

B) web 服务器如何区分访问者是普通浏览用户还是搜索引擎的 Spider?

C) cookie 按生命周期分类分为哪两类? 其生命周期分别是多长? 向浏览器设置 cookie 时 cookie 有哪些属性可以设置，分别起到什么作用?

D) HTTP 协议中 Keep-Alive 是什么意思? 使用 Keep-Alive 有何好处，对服务器会有什么不利的影响? 对于不利的影响有什么解决方案

4. 简述你最常用的 Web 服务器的一种或者几种，并说明如何在 Web 服务器和应用服务器之间建立反向代理

5. 简述你所了解的 MVC 各层次的常用开发框架，说明其特点

6. 简述 Web 应用环境下远程调用的几种方式，并且从性能 异构性等方面比较其优劣

第八部分(选作): Flash 开发

7. flash 和 js 如何交互?

8. flash 中的事件处理分哪几个过程 Event 对象的 target 和 currentTarget 有什么区别?

第九部分(选作): 软件测试

1. 请描述你对测试的了解，内容可以涉及测试流程，测试类型，测试方法，测试工具等

2. 如果有一天你早上上班，发现不能上网了，你会用什么步骤找出问题所在?

3. Web 应用中实现了好友功能，用户可以给别人发"加为好友"的请求，发了请求后可以取消请求，对方收到请求后，可以选择接受或者拒绝. 互为好友的两个人，每个人都可以单方面删除对方，请设想尽可能多的路径对此功能设计测试用例，每个用例包括测试步骤和

预期结果

4. 公司开发了一个 web 聊天工具，用于网络用户之间的聊天，一个人同时可以和多个人聊天，功能类似于 MSN 等等 IM 工具

要求该系统能承受 1 万个在线用户，平均每个用户会和 3 个人同时聊天，在网络条件正常的情况下，要求用户收到消息的延迟时间不超过 1 分钟．现在需要对系统进行性能测试，验证系统是否达到预定要求，请你写一个性能测试方案．提示如下：

1）性能测试的过程一般都是模拟大量客户端操作，同时监控服务器的性能和客户端相应，根据服务器的性能指标和客户端响应状况进行分析和判断

2）系统的性能问题可以从两个角度考虑，一个是服务器问题，设计得不好的程序，在大负载或者长时间运行情况下，服务器会 down 机；另一个是客户端问题，在负载大的时候，客户端响应会变慢

3）在答题中，可以不涉及性能测试工具，监控工具等细节，把你的测试思路说清楚就可以

5. 自动功能测试中会将测试用例组织成测试集合来统一运行，测试集合 suite 按功能分类可以有若干个模块module，每个模块module下包含若干个测试用例test．现测试集合已经运行完毕，但是需要在测试报告中统计各个模块的用例失败率，将失败率超过 20%的模块名与其失败率记录下来报警，请编写实现上述功能的 getTestReport 函数．可使用 Java 或 C++等您熟悉的编程语言，提供的接口及方法如下：

测试集合接口 Isuite:

```
Collection<ITest>getTests()      //得到测试集合下的所有测试用例 test
```

测试用例接口 Itest:

```
String getModule()      //得到该用例对应的模块名称 module
int getResult()      //得到该用例的执行结果:0 失败 1 成功
```

报警函数:

```
void alertMessage(String message)
public static void getTestReport(ISuite suite){
    //你的实现写在这里
}
```