

### 一个类包含：

- 构造方法  
构造方法与类同名，一个类可以有多个构造方法
- 成员变量
- 成员方法

### 封装

1. 修改属性可见性
2. 类内局部属性 **private**
3. 每个值属性提供对外的 **public** 方法访问

### 继承

1. 通常格式：**class 子类 extends 父类 {}**
2. 为什么要继承？减少代码重复部分，提高可维护性和复用性，将重复的部分提取出来组成父类

企鹅类：属性（姓名，id），方法（吃，睡）

老鼠类：属性（姓名，id），方法（吃，睡）



动物类：属性（姓名，id），方法（吃，睡）

3. 构造器：子类不继承父类的构造器，而是调用；①父类的构造器有参数：**super（参数）**  
②父类构造器无参：无需 **super** 关键字，直接调用父类无参关键字。
4. 单继承 ✓ 多重继承 ✓ 不同类继承同一个类 ✓ 多继承 ×
5. **extends** 只能继承一个类
6. **implement** 变相成为多继承，类继承接口。**public class C implements A,B{}**
7. **super** 实现引用父类
8. **final class A** 指这个类不能被继承

### 重写

子类对父类的允许访问的方法的实现过程进行重写

```
Animal a= new Animal()
```

```
Animal b= new Dog()
```

**Dog** 是 **Animal** 的子类，都有 **move** 方法

编译成功是因为 **Animal** 里有 **move** 方法。但是 **b.move()** 会调用 **Dog** 里的方法，因为 **JVM** 运行的是特定对象的方法。

#### 方法的重写规则：

1. 参数列表必完全与被重写方法的相同。
2. 返回类型与被重写方法的返回类型可以不相同,但是必须是父类返回值的派生类 java5 及更早版本返回类型要一样, java7 及更高版本可以不同)。
3. 访问权限不能比父类中被重写的方法的访问权限更低。例如:如果父类的一个方法被声明为 **public**,那么在子类中重写该方法就不能声明为 **protected**。
4. 父类的成员方法只能被它的子类重写。
5. 声明为 **final** 的方法不能被重写。
6. 声明为 **static** 的方法不能被重写，但是能够被再次声明。
7. 子类和父类在同一个包中，那么子类可以重写父类所有方法，除了声明为 **private** 和 **final** 的方法。
8. 子类和父类不在同一个包中，那么子类只能重写父类的声明为 **public** 和 **protected** 的非 **final** 方法。

9. 重写的方法能够抛出任何非强制异常, 无论被重写的方法是否抛出异常。但是, 写的方法不能抛出新的强制性异常, 或者比被重写方法声明的更泛的强制性异常, 反之则可以。
10. 构造方法不能被重写。
11. 如果不能继承一个方法, 则不能重写这个方法。

### 重载

一个类里, 方法名相同, 参数列表一定不同, 返回类型可相同可不同。常用于构造器重载

区别点	重载方法	重写方法
参数列表	必须修改	一定不能修改
返回类型	可以修改	一定不能修改
异常	可以修改	可以减少或删除, 一定不能抛出新的或者更广的异常
访问	可以修改	一定不能做更严格的限制 (可以降低限制)

### 多态 (继承+重写+父类引用指向子类对象)

同一个接口, 用不同的实例实现不同的操作

先检查父类里是否有调用的方法, 没有则编译出错, 有就在 JVM 中运行子类同名方法!

### 抽象类

- **抽象类不能被实例化**(初学者很容易犯的错), 如果被实例化, 就会报错, 编译无法通过。只有抽象类的非抽象子类可以创建对象。
- 抽象类中不一包含抽象方法, 但是有抽象方法的类必定是抽象类。
- 抽象类中的抽象方法只是声明, 包含方法体, 就是不给出方法的具体实现也就是方法的具体功能。
- 构造方法, 方法(用 `static` 修饰的方法)不能声明为抽象方法。
- **抽象类的子类必须给出抽象类中的抽象方法的具体实现**, 除非该子类也是抽象类。

### 接口

一组抽象方法的集合。接口无法被实例化, 他是要告诉类要实现哪些方法。

继承了某接口的类必须要实现接口里描述的所有方法, 否则就必须声明为抽象类。

接口是隐形抽象的。接口中的方法都是公有的

接口中的成员变量只能是 `public static/final` 变量

接口是可以多继承的 `A extends B,C`

### 接口和抽象类有什么区别?

实现: 抽象类的子类使用 `extends` 来继承; 接口必须使用 `implements` 来实现接口。

构造函数: 抽象类可以有构造函数; 接口不能有。

实现数量: 类可以实现很多个接口; 但是只能继承一个抽象类。

访问修饰符: 接口中的方法默认使用 `public` 修饰; 抽象类中的方法可以是任意访问修饰符。

海隆的面试:

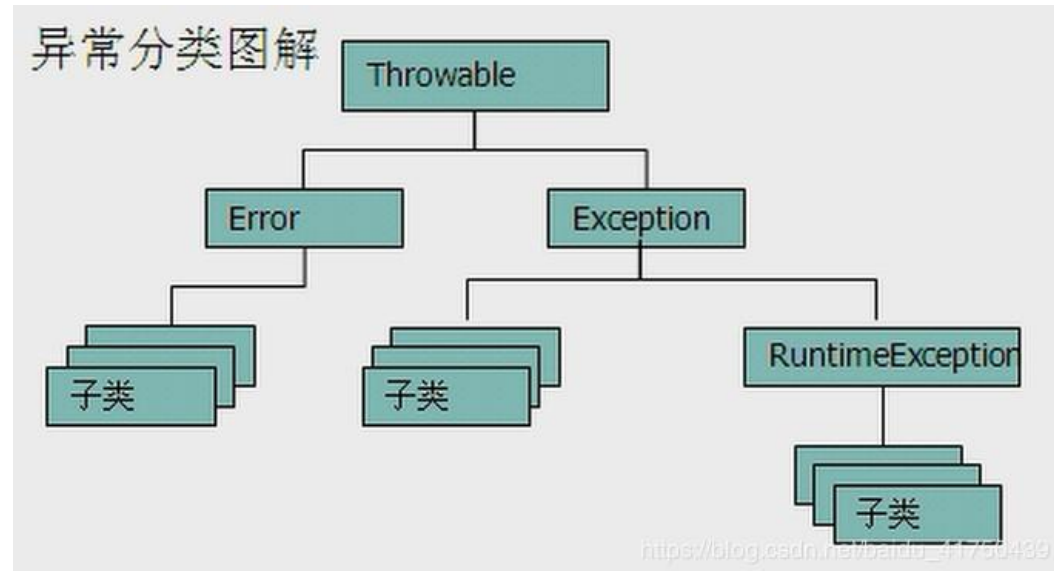
- **mysql 语句 聚合函数 有哪些?**  
sum avg count max min
- **重写和重载是什么?**
- **switch 语句里的参数是什么类型?**  
在 Java 7 之前, switch 只能支持 `byte`、`short`、`char`、`int` 或者其对应的封装类以及 `Enum` 类型。在 Java 7 中, `String` 支持被加上了。

Switch 对于 String 类型的支持原理是把 string 转为 hashcode 去做的路由，当 hashcode 相同时在使用 equals 方法去区分路由。

- **static** 方法有什么特征？

- 编译没有出现错误，但是运行出错时，怎么找到错误：

1. 设置断点，单步运行
2. ?



- 软件工程生命周期

需求分析，分析设计，集成，部署，测试（黑盒，白盒），评估

- **mysql** 事务处理

事务必须具有 **ACID** 特性：原子性，一致性，隔离性，持久性

隔离级别：

1. 读未提交 **read uncommitted** 可能脏读，不可重复读，幻读
2. 读已提交 **read committed** 可能不可重复读，幻读
3. 可重复读 **repeatable read** 可能幻读 **mysql** 默认的隔离级别
4. 串行化 **serializable** 都不会发生 回锁表，并发性差

脏读（对未提交的数据的读取），不可重复读（两次读到的数据不一致），幻读（同样的检索条件读出来的东西多了）

- （操作系统）用户态切换至核心态：1.系统调用 2.异常 3.外围设备中断

- （网络）**UDP** 报头没有序号，因为是无连接的，不保序

- （算法）排序：稳定的：冒泡，插入，桶，归并

- （数据库）事务的隔离级别

读未提交：可能造成脏读，不可重复读（两次读到数据不一致），幻读（前后多次读，数据总量不一致）

读取提交内容：不可重复读，幻读

可重复读：幻读

- （Linux）命令

- 编译过程

预处理，展开头文件/宏替换/去掉注释/条件编译 （test.i main .i）

No such file or directory

编译，	检查语法，生成汇编	(test.s main.s)
	变量'variable' 没有声明 (第一次使用此变量)	
汇编，	汇编代码转换机器码	(test.o main.o)
链接	链接到一起生成可执行程序	a.out
	文件不可识别：文件格式不可识别	
	undefined reference to 'foo'程序中使用了在本文件和其它库中没有定义的函数或变量。有可能是丢失了链接库，或使用了不正确的名字。	
	<b>• 运行过程段错误：</b> 错误原因：企图访问受保护的内容或覆盖重要的数据！它指明内存访问错误。 通常的原因如下： <ol style="list-style-type: none"> <li>1、反向引用一个空指针或没初始化的指针；</li> <li>2、超出数组访问的下标；</li> <li>3、对 malloc, free 和相关函数不正确的使用；</li> <li>4、使用 scanf 时的参数（数量、类型）不正确。</li> </ol>	
	<b>• 单例模式</b> 意图：保证一个类仅有一个实例，并提供一个访问它的全局访问点。 主要解决：一个全局使用的类频繁地创建与销毁。 何时使用：当您想控制实例数目，节省系统资源的时候。 如何解决：判断系统是否已经有这个单例，如果有则返回，如果没有则创建。 关键代码：构造函数是私有的。 优点： <ol style="list-style-type: none"> <li>1、在内存里只有一个实例，减少了内存的开销，尤其是频繁的创建和销毁实例</li> <li>2、避免对资源的多重占用（比如写文件操作）。</li> </ol> 缺点：没有接口，不能继承。	
	<b>• 程序的局部性</b> 时间局部性和空间局部性	

<b>static 关键字</b>	
<ul style="list-style-type: none"> <li>• 被 static 修饰的变量和类属于静态资源，可以在类实例之间共享，用类名.变量名可以直接引用，不需要 new 出一个类，一处变，处处变。</li> <li>• 不同的静态资源放在不同的类中的好处：①清晰②避免重名③防止膨胀</li> <li>• JVM 的类加载机制：静态资源在类初始化时加载，而非静态资源在 new（实例化）时才加载。</li> </ul>	
<ol style="list-style-type: none"> <li>1、静态方法能不能引用非静态资源？不能</li> <li>2、静态方法里面能不能引用静态资源？可以</li> <li>3、非静态方法里面能不能引用静态资源？可以</li> </ol>	
<ul style="list-style-type: none"> <li>• 静态块如</li> </ul>	
<pre>static {     System.out.println("Enter A.static block"); }</pre>	
静态块里的代码只执行一次且只在类初始化时执行。	
<ul style="list-style-type: none"> <li>• 静态资源的加载顺序是严格按照静态资源的定义顺序来加载的</li> </ul>	

<ul style="list-style-type: none"> <li>• 静态代码块对于定义在它之后的静态变量，可以赋值，但是不能访问。Cannot reference a field before it is defined</li> <li>• 父类静态代码块-&gt;子类静态代码块的顺序加载的，且只加载一次</li> </ul>
<p>StringUtils 里面的 isEmpty 和 isBlank 的区别</p> <p>对空格判断：isEmpty 判断字符串不为空，isBlank 判断字符串为空</p>

<p>输入： 第 1 行为 n 代表用户的个数 第 2 行为 n 个整数，第 i 个代表用户标号为 i 的用户对某类文章的喜好度 第 3 行为一个正整数 q 代表查询的组数 第 4 行到第 (3+q) 行，每行包含 3 个整数 l,r,k 代表一组查询，即标号为 l&lt;=i&lt;=r 的用户中对这类文章喜好值为 k 的用户的个数。 数据范围 n &lt;= 300000,q&lt;=300000 k 是整型</p>
<p>输出描述：</p> <p>输出：一共 q 行，每行一个整数代表喜好值为 k 的用户的个数</p> <p>输入例子 1:</p> <pre>5 1 2 3 3 5 3 1 2 1 2 4 5 3 5 3</pre> <p>输出例子 1:</p> <pre>1 0 2</pre>
<p>Scanner 类：用来从输入流获得</p> <pre>Scanner sc = new Scanner(System.in); int n = sc.nextInt();//将下一个扫描为 int 型 string name=sc.nextLine();//此执行器执行当前行，并返回跳过的输入信息</pre> <p>①nextInt()只读取数值，读取完后\n 没有读取并且光标放在本行</p> <p>②next()方法遇到第一个扫描有效字符，即第一个非空格非换行符后面开始，一直获取到下一个空格，换行符之前的，单个字符串</p> <p>③nextLine()可以扫描到一行内容并作为一个字符串而被获取到</p>
<p>(常用)方法</p> <ol style="list-style-type: none"> <li>1. 存值：put(key,value) 返回这个 key 以前的值，没有则 null</li> <li>2. 取值：get(key) 没有对应则 null</li> <li>3. 判断 Hashmap 是否为空：isEmpty</li> <li>4. 判断是否含有这个 key: containsKey(key);</li> <li>5. 判断是否含有这个 value: containsValue(value);</li> <li>6. 删除这个 key 值下的 value: remove("1")会返回删除的值</li> <li>7. 删除这个 key 和 value: remove("DEMO2", 1)</li> <li>8. 显示所有的 value 值: map.values()      {}→{1,2}</li> <li>9. 显示所有的 key 值: map.keySet()      {}→{demo1,demo2}</li> <li>10. 元素个数: map.size()</li> </ol>

11. 显示所有的 key 和 value: `map.entrySet()`
  12. 转化为字符串: `map.toString()`
  13. 添加另一个同一类型的 map 下的所有制: `map.putAll(map1);` //map1 里的内容放入 map 里, 插到前面 ( ? )
  14. 替换这个 key 的 value: `.repalce(key,new value)`会返回改之前的 value
  15. 清空 map: `clear()`
  16. 克隆: `Object clone=map.clone()`
  17. 如果 Map 不存在键 key 或者该 key 关联的值为 null(返回这个值), 那么就执行 `put(key, value)`; 否则, 便不执行 put 操作: `putIfAbsent(key,new value)`
- ```
System.out.println(map); //{DEMO1=1, DEMO2=2}

System.out.println(map.putIfAbsent("DEMO1", 12222)); //1

System.out.println(map.putIfAbsent("DEMO3", 12222)); //null

System.out.println(map); //{DEMO1=1, DEMO2=2, DEMO3=12222}
```
18. `compute` 的方法,指定的 key 在 map 中的值进行操作 不管存不存在, 操作完成后保存到 map 中 `Integer integer = map.compute("3", (k,v) -> v+1 );`

什么是反射?

反射就是运行时获得对象类型, 能提高可移植性。

1.`getClass()` 获得**实例**的类型, class 是一个类获得类型

对象 a 是 A 的一个实例, A 是某一个类,

```
A a = new A();
if(a.getClass()==A.class) {
    System.out.println("equal");
} else {
    System.out.println("unequal");
}
//输出 equal;
```

ArrayList 常用方法

1. 定义一个 ArrayList

```
//默认创建一个 ArrayList 集合
List<String> list = new ArrayList<>();
//创建一个初始化长度为 100 的 ArrayList 集合
List<String> initlist = new ArrayList<>(100);
//将其他类型的集合转为 ArrayList
List<String> setList = new ArrayList<>(new HashSet());
```

2. `add (E element)`

过程: 使用了 `ensureCapacityInternal(size+1)`来保证有足够空间存放数据

`add("铁扇", 0);`在首位插入

3. `set(int index, E element)`返回改之前的数据

4. `get(int index)`下标查找

5. remove(int index)
6. isEmpty()
7. size()
8. contains()
9. clear()
10. Collection.reverse(str)

数据库操作语句类型

1. 数据定义语言 DDL

CREATE TABLE/VIEW/INDEX/SYN/CLUSTER 不能 rollback

2. 数据查询语言 DQL

SELECT FROM WHERE

3. 数据操纵语言 DML

插入 INSERT

更新 Update

删除 Delete

4. 数据控制语言 DCL:

授予或回收访问数据库的某种**特权**，并控制数据库**操纵事务发生的时间及效果**，对数据库实行监视

GRANT ROLLBACK

显式提交 COMMIT

隐式提交 ALTER, AUDIT, COMMENT, CONNECT, CREATE, DISCONNECT, DROP, EXIT, GRANT, NOAUDIT, QUIT, REVOKE, RENAME。

自动提交 SET AUTOCOMMIT ON

数据库操作的事务四大特征是：ACID

Atomic(原子性)、Consistency(一致性)、Isolation(隔离性)和 Durability(持久性)

面向对象的 SOLID 原则：

s( Single-Responsibility Principle )：单一职责原则

o( Open-Closed principle )：开放封闭原则

l( Liskov-Substitution Principle )：里氏原则

i( Interface-Segregation Principle )：接口隔离原则

d( Dependency-Inversion Principle )：依赖倒置原则

二维数组的长度：a.length 行数 a[0].length 列数

【剑指 offer】【查找 数组】

1. 在一个二维数组中（每个一维数组的长度相同），每一行都按照从左到右递增的顺序排序，每一列都按照从上到下递增的顺序排序。请完成一个函数，输入这样的一个二维数组和一个整数，判断数组中是否含有该整数。

```
public class Solution {
    //对每一行都进行二分查找
    public boolean Find(int target, int [][] array) {
        //int i=0,j=0;
        int rowLen=array.length;
        int colLen=array[0].length;
```

```

        for(int i=0;i<rowLen;i++){
            int left=0,right=colLen-1,mid;
            while(left<=right){
                mid=(left+right)/2;
                if(array[i][mid]<target)
                    left=mid+1;
                else if(array[i][mid]>target)
                    right=mid-1;
                else
                    return true;
            }
        }
        return false;
    }
}

```

【备注】重复进行二分查找，有不必要的比较

```

public class Solution {
    /*从左下角的元素 array[rowLen-1][0]开始查找
    如果目标比它大就在右边找 col++，比它小就在上方找 row--
    直到下标超出界限，如果还没找到则 false
    */
    public boolean Find(int target, int [][] array) {
        //int rowLen=array.length;
        //int colLen=array[0].length;
        int row=array.length-1,col=0;
        while(row>=0 && col<array[0].length){
            if(target>array[row][col])
                col++;
            else if(target<array[row][col])
                row--;
            else
                return true;
        }
        return false;
    }
}

```

【剑指 offer】【字符串 数组】

2.请实现一个函数，将一个字符串中的每个空格替换成“%20”。例如，当字符串为 We Are Happy.则经过替换之后的字符串为 We%20Are%20Happy。

```

public class Solution {
    public String replaceSpace(StringBuffer str) {
        String str2=str.toString();

```



```

        //str.toString();
        str2=str2.replace(" ", "%20");
        //str2=new StringBuffer(str3);
        return str2;
    }
}

```

【备注】String 有 replace(old,new)方法，StringBuffer 有 replace(start,end,str)方法  
可以通过构造的方法和 toString 方法来将 StringBuffer 转化为 String 类型

### 【剑指 offer】【链表】

3.输入一个链表，按链表从尾到头的顺序返回一个 ArrayList。

```

/**
 * public class ListNode {
 *     int val;
 *     ListNode next = null;
 *
 *     ListNode(int val) {
 *         this.val = val;
 *     }
 * }
 */
import java.util.ArrayList;
import java.util.Collections;

public class Solution {
    public ArrayList<Integer> printListFromTailToHead(ListNode n) {
        ArrayList<Integer> list = new ArrayList<>();
        while(n!=null){
            list.add(n.val);
            n=n.next;
        }
        Collections.reverse(list);
        return list;
    }
}

```

【备注】Collections 是一个集合类的包装类，包含很多静态方法：Collections.reverse(str)、sort(str)

### 【剑指 offer】【树】

4.输入某二叉树的前序遍历和中序遍历的结果，请重建出该二叉树。假设输入的前序遍历和中序遍历的结果中都不含重复的数字。例如输入前序遍历序列{1,2,4,7,3,5,6,8}和中序遍历序列{4,7,2,1,5,3,8,6}，则重建二叉树并返回。

```

/**

```

```

* Definition for binary tree
* public class TreeNode {
*     int val;
*     TreeNode left;
*     TreeNode right;
*     TreeNode(int x) { val = x; }
* }
*/
import java.util.*;
public class Solution {
    public TreeNode reConstructBinaryTree(int [] pre,int [] in) {
        /*前序第一个数字是根，在中序中找到其位置，前面的是左子树，后面的是右子树
        采用递归的思想，只要想递归到最后的时候要做什么操作，即先找到根节点，然后安上左右子树即可。
        递归的出口是只剩下一个元素的时候
        */
        //数组长度为 0 和 1 时
        if(pre.length==0)
            return null;
        if(pre.length==1)
            return new TreeNode(pre[0]);

        //试图用 arraylist 的 indexOf 方法
        ArrayList<Integer> inlist = new ArrayList<>();
        for(int i=0;i<in.length;i++){
            inlist.add(in[i]);
        }
        int rootIndex=inlist.indexOf(pre[0]);
        TreeNode root=new TreeNode(pre[0]);

        root.left=reConstructBinaryTree(Arrays.copyOfRange(pre,1,rootIndex+1),Arrays.copyOfRange(in,0,rootIndex));

        root.right=reConstructBinaryTree(Arrays.copyOfRange(pre,rootIndex+1,pre.length),Arrays.copyOfRange(in,rootIndex+1,in.length));

        return root;
    }
}

```

运行时间: 225ms

占用内存: 22756k

Arrays.asList(String 数组)

Arrays.copyOfRange(int[],int from,int to);//左闭右开

Arrays.toString(参数可以是 int [],byte[], int []等等)//因为 out.println(数组)会输出地址而不是内容

【剑指 offer】【树】

5.给定一个二叉树和其中的一个结点，请找出中序遍历顺序的下一个结点并且返回。注意，树中的结点不仅包含左右子结点，同时包含指向父结点的指针。

【第一次三次就 ac 了，爱了爱了】

```
/*
public class TreeLinkNode {
    int val;
    TreeLinkNode left = null;
    TreeLinkNode right = null;
    TreeLinkNode next = null;

    TreeLinkNode(int val) {
        this.val = val;
    }
}
*/
public class Solution {
    public TreeLinkNode GetNext(TreeLinkNode pNode)
    {
        /*
        1.如果有右子树，则下一个节点是右子树的最左节点，
        2.如果没有右子树，且父结点的左子树是自己，则下一个结点是父结点
        3.如果没有右子树，且父节点的右子树是自己，则下一个结点是：对父结点进行从
        2-3 再判断【即用父结点递归,直到根节点】
        */
        if(pNode==null)
            return null;

        if(pNode.right!=null){
            TreeLinkNode tNode=pNode.right;
            while(tNode.left!=null){
                tNode=tNode.left;
            }
            return tNode;
        }
        else{
            while(pNode.next!=null){
                if(pNode.next.left==pNode)
                    return pNode.next;
                if(pNode.next.right==pNode)
```

```

        pNode=pNode.next;
    }
}
return null;
}
}

```

### 【剑指 offer】【树】

6.请实现一个函数，用来判断一颗二叉树是不是对称的。注意，如果一个二叉树同此二叉树的镜像是一样的，定义其为对称的。

```

public class Solution {
    boolean isSymmetrical(TreeNode pRoot)
    {
        /*
        递归算法：
        1.只要知道 pRoot.left 和 pRoot.right 是否对称
        2. 则 递 归 到 最 后 要 判 断 的 是 ： 左 右 结 点 的 值 是 否 相 等 ， 且
        left.left=right.right,left.right=right.left
        */
        if(pRoot==null)
            return true;
        return isSym(pRoot.left,pRoot.right);
    }

    boolean isSym(TreeNode left,TreeNode right)
    {
        if(left == null && right == null) return true;
        if(left == null || right == null) return false;
        return (left.val==right.val) && isSym(left.left,right.right) && isSym(left.right,right.left);
    }
}

```

这个递归的解法很强，递归的重点在于想到最后需要做什么操作。还有出口。

```

import java.util.*;
public class Solution {
    boolean isSymmetrical(TreeNode pRoot)
    {
        /*
        非递归算法 BFS:
        使用队列，
        1.成对出队,若都为空，则继续
        2.若有一个为空，则返回 false
        3.若 val 不相等，则返回 false
        4.成对入队顺序， left.left 和 right.right,left.right 和 right.left
        */
    }
}

```

```

        if(pRoot==null) return true;
        Queue<TreeNode> q = new LinkedList<>();
        q.offer(pRoot.left);
        q.offer(pRoot.right);

        while(!q.isEmpty()){
            TreeNode left=q.poll();
            TreeNode right=q.poll();
            if(left==null && right==null) continue;
            if(left==null || right==null) return false;
            if(left.val==right.val){
                q.offer(left.left);
                q.offer(right.right);
                q.offer(left.right);
                q.offer(right.left);
            }
            else return false;
        }
        return true;
    }
}

```

#### 【Queue 的用法】

1. add 和 remove 当参数是 null 是会抛出错误
2. offer 和 poll 当参数是 null 会返回 false

**LinkedList** 是采用 **链表** 的方式来实现 List 接口的,它本身有自己特定的方法,如: addFirst(),addLast(),getFirst(),removeFirst()等. 由于是采用链表实现的,因此在进行 insert 和 remove 动作时在效率上要比 ArrayList 要好得多!**适合用来实现 Stack(堆栈)与 Queue(队列)**,前者先进后出,后者是先进先出.

```

import java.util.*;

public class Solution {
    boolean isSymmetrical(TreeNode pRoot)
    {
        /*
        非递归算法 DFS:
        使用栈,
        1.成对出栈, 如果都为空, 则继续
        2.如果有一个为空, 返回 false
        3.如果值不相等, 返回 false
        4.成对进栈: left.left 和 right.right, left.right 和 right.left
        */
        if(pRoot==null)
            return true;
        Stack<TreeNode> s=new Stack<>();
    }
}

```

```

s.push(pRoot.left);
s.push(pRoot.right);
while(!s.isEmpty()){
    TreeNode right=s.pop();
    TreeNode left=s.pop();
    if(left==null && right==null) continue;
    if(left==null || right==null) return false;
    if(left.val==right.val){
        s.push(left.left);
        s.push(right.right);
        s.push(left.right);
        s.push(right.left);
    }
    else return false;
}
return true;
}
}

```

#### Linux 命令

|          |                        |                                           |
|----------|------------------------|-------------------------------------------|
| 新建目录     | mkdir filename         | mkdir test 创建 test 文件夹                    |
| 进入指定目录   | cd filename            | cd test 切换到 test 文件夹                      |
| 新建文件     | touch filename         | touch 1.php 新建一个 1.php 的文件 支持创建多文件，一直写就行了 |
| 编辑文件     | vi filename i          | vi 1.php 进入编辑环境，按 i 执行编辑                  |
| 退出编辑     | Esc :wq!               | 先按键盘左上角的退出键，然后输入:wq!,保存并退出编辑环境            |
| 查看编辑后效果  | cat filename           | cat 1.php 查看编辑后的 1.php 文件                 |
| 删除指定文件   | rm filename            | rm 1.php 删除 1.php 文件                      |
| 返回上层目录   | cd ...                 | cd ... 从 test 目录切换到上一目录                   |
| 返回到根目录   | cd /                   | 注意，根目录不等于桌面                               |
| 返回上次停留位置 | cd -                   | 类似返回键                                     |
| 删除空目录    | rmdir filename         | rmdir test 删除空文件夹 test                    |
| 强制删除     | rm -rf filename        | 自带循环的删除，即便目录不是空的 ,注意 Linux 命令下的删除不可恢复     |
| 复制文件并重命名 | cp filename1 filename2 | cp 1.php 2.php 当前目录下，复制 1.php，重命名为 2.php  |
| 移动文件     | mv filename path       | mv 1.php ../ 移动 1.php 到上一层目录              |

|             |                        |                                                                         |
|-------------|------------------------|-------------------------------------------------------------------------|
| 重命名文件       | mv filename1 filename2 | mv 1.php 2.php 将 1.php 更名为 2.php, 支持改拓展名 window 下不支持创建.开头的文件, 可以用这个实现更名 |
| 通配符*        | rm *.txt               | 删除以.txt 为拓展名的文件                                                         |
| 显示自己当前目录    | pwd                    | 绝对路径返回<br>->/c/Users/Administrator/Desktop                              |
| 退出          | exit                   | 直接关了小黑窗口也行                                                              |
| 查看磁盘空间情况    | df -h                  | 会显示使用量, 总量, 剩余量等信息                                                      |
| 查看网络通讯情况    | ping 域名或 ip            | ping www.baidu.com                                                      |
| 检查 ip 地址的配置 | ifconfig               | 会出现 ip 相关信息, 网关, 子网掩码, ipv4, ipv6 等                                     |

| 命令      | 用法                           | 功能                                                                                    |
|---------|------------------------------|---------------------------------------------------------------------------------------|
| ls      | ls                           | 列出目录中的内容                                                                              |
| cat     | cat 文件名                      | 显示文件内容                                                                                |
| echo    | echo abc                     | 显示一行文本或者字符串                                                                           |
| cd      | cd 目录                        | 进入指定目录                                                                                |
| type    | type 命令名                     | 显示命令的类型（外部/内建）                                                                        |
| tab 键   | more pa<tab><br>echo fi<tab> | ①按下 Tab 键一次, 自动补足完整的命令变成了 more pasaage<br>②按下 Tab 键两次, 系统会显示当前目录下所有具有相同前缀的文件名称, 供用户选择 |
| history | ①history<br>②history n       | ①查看所有历史执行的命令<br>②查看最近执行的 n 条命令                                                        |
| !       | ①! n<br>②!!                  | ①运行第 n 条历史记录<br>②运行上一条历史记录                                                            |
| alias   | alias dir='ls -l'            | 创建别名                                                                                  |
| unalias | unalias dir                  | 取消别名                                                                                  |
| &       | ls -l &                      | 后台的方式执行该命令, 显示后台运行程序的进程 PID, shell 返回命令提示符状态                                          |
| >       | ls -l > File1                | 将 ls -l 命令的结果送至 File1 文件中                                                             |
| <       | cat < File1                  | cat 从 File1 获得输入, 察看 File1 文件的内容                                                      |
|         | ls   grep .c                 | 建立一个 ls 进程和 grep 进程之间的管道                                                              |
| rename  | rename a.c b.c a.c           | 将文件 a.c 的名字改成 b.c                                                                     |
| grep    | 较少单独使用, 一般配合管道使用             | 使用正则表达式搜索文本, 并把匹配的行打印出来                                                               |
| clear   | clear                        | 清空屏幕                                                                                  |

|      |          |        |
|------|----------|--------|
| date | date     | 显示当前时间 |
| pwd  | 显示当前所在位置 |        |

- 结合实际项目经验回顾软件工程的知识，例如：如何从需求推导出系统设计，如何衡量两个不同设计的优劣，如何在各种限制下（人员、时间、资源等）选择其中更合适的设计，以及提升该设计的可拓展性等。

- 在白板上练习算法题目，写出清晰、简洁、bug free 的代码，并衡量时间和空间复杂度以及可能存在的副作用。

- 尝试用不同的方法，思路或数据结构去解决同一个问题，并且衡量不同解法之间的优劣。

- Linux Shell 知识

- REST API

- SOA

- SSH, SFTP

- Web 基础知识 HTML, JS, CSS

- 应用程序服务器 tomcat 和 Web 服务器 httpd

兴业数金笔试题：

1. 【数据库】最小粒度的逻辑数据层次：数据块

2. 【web】servlet 的生命周期：

①加载和实例化 servlet

②初始化 init(ServletConfig)生命周期中只调用一次，一般在请求前（容器启动时）预加载预初始化

③为客户提供服务 service()：

客户通过点击链接或提交表单来请求访问 Servlet

容器接收到请求，根据 url 找到正确的 servlet，创建两个对象：HttpServletRequest 请求对象，HttpServletResponse 响应对象。

创建一个新的线程，在线程中调用 service

然后 service 根据 HTTP 请求的类型(get/post)来决定调用 Servlet 的 doGet()或 doPost()

④销毁和卸载 destroy()

3. 【web】Servlet 程序的入口点是？ service()

4. 【Linux】-m 设置用户权限

-p 如果路径中某些目录不存在，系统会自动创建

mkdir 没有-d 和-f 选项

5. 创建 Cookie 的方法：setCookie？

6. 【Java】

ArrayList list=new ArrayList();

这种是默认创建大小为 10 的数组，每次扩容大小为 1.5 倍

ArrayList list=new ArrayList(20);

使用的 ArrayList 的有参构造函数，创建时直接分配其大小，没有扩充。

7. 【web】HTTP 状态响应码：

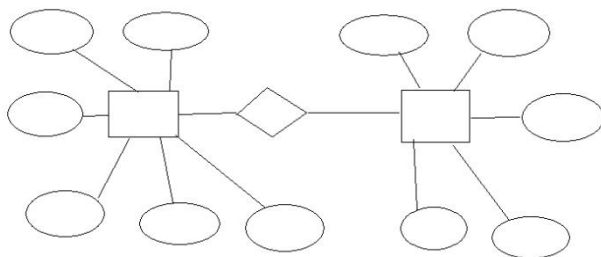
8. 【web】每个网站访问用户都要访问的变量，应该将其设为 Application 变量

9. 【web】当一个用户访问某个网页时，在传输层报头中封装了相应的源端口号和目的端口号，下列端口号中正确的是 TCP 21 端口：FTP 文件传输服务



|             |                                    |
|-------------|------------------------------------|
| TCP 23 端口   | TELNET 终端仿真服务                      |
| TCP 25 端口   | SMTP 简单邮件传输服务                      |
| UDP 53 端口   | DNS 域名解析服务                         |
| TCP 80 端口   | HTTP 超文本传输服务                       |
| TCP 110 端口  | POP3 “邮局协议版本 3”使用的端口               |
| TCP 443 端口  | HTTPS 加密的超文本传输服务                   |
| TCP 1521 端口 | Oracle 数据库服务                       |
| TCP 1863 端口 | MSN Messenger 的文件传输功能所使用的端口        |
| TCP 3389 端口 | Microsoft RDP 微软远程桌面使用的端口          |
| TCP 5631 端口 | Symantec pcAnywhere 远程控制数据传输时使用的端口 |
| UDP 5632 端口 | Symantec pcAnywhere 主控端扫描被控端时使用的端口 |
| TCP 5000 端口 | MS SQL Server 使用的端口                |
| UDP 8000 端口 | 腾讯 QQ                              |

10. 【数据库】两个实体 m:n 联系到关系模型的转化，原有的实体关系表不变，再单独建立一个关系表，分别用两个实体的关键属性作为外键即可，并且，如果联系有属性，也要归入这个关系中。m:n 联系转化为关系模型：如图



11. 【数据库】在下列模式中，能够给出数据库物理存储结构与物理存取方法的是（内模式）。

12. 【Java】Java 中的新生代、老年代、永久代和各种 GC

13. 【数据库】linux 系统中某文件的组外成员的权限为只读；所有者有全部权限；组内的权限为读与写，则该文件的权限为：764

u-g-o: 用户-组内用户-其他用户

r-w-x: 用 4-2-1 表示。

14.

mysql

• 创建语句

- 学生表 student(id,name)
- 课程表 course(id,name)
- 学生课程表 student\_course(sid,cid,score)

```
create table student(
  id int unsigned primary key auto_increment, //自动增加序号
  name char(10) not null
);
```

```

create table course(
    id int unsigned primary key auto_increment,
    name char(20) not null
);
insert into course(name) values('语文'),('数学');

create table student_course(
    sid int unsigned,
    cid int unsigned,
    score int unsigned not null,
    foreign key (sid) references student(id),
    foreign key (cid) references course(id),
    primary key(sid, cid)
);
insert into student_course values(1,1,80),(1,2,90),(2,1,90),(2,2,70);

```

- 插入语句

```
insert into student(name) values('eika'),('zhj');
```

- 查询语句

1. 查询 student 表中重名的学生，结果包含 id 和 name，按 name,id 升序

```

select id,name
from student
where name in(
    select name from student
    group by name
    having( count(*)>1
)order by name;

```

2. 在 student\_course 表中查询平均分不及格的学生，列出学生 id 和平均分

```

select id,avg(score) as avg_score
from student_course
group by sid
having (avg_score<60);

```

**where 子句中不能用聚集函数作为条件表达式**

3. 在 student\_course 表中查询每门课成绩都不低于 80 的学生 id

```

select distinct sid
from student_course
where sid not in(
    select sid from student_course
    where score <80
);

```

4. 查询每个学生的总成绩，结果列出学生姓名和总成绩

```

select name , sum(score) as total
from student a left join student_course b

```

```
on a.sid=b.sid
```

```
group by sid;
```

5. 总成绩最高的学生

```
select sid,sum(score) as total
```

```
from course
```

```
group by sid
```

```
order by total desc limit 1;
```

**order by** 中可以使用聚集函数

6. 在 student\_course 表查询课程 1 成绩第 2 高的学生，如果第 2 高的不止一个则列出所有的学生

```
select *from student_course
```

```
where cid=1 and score=(
```

```
    select score from student_course
```

```
    where cid=1 group by score desc limit 1,1
```

```
);
```

```
import java.util.Scanner;
```

```
public class Main {
```

```
    public static void main(String[] args) {
```

```
        Scanner in = new Scanner(System.in);
```

```
        String inlist=in.next();
```

```
        String [] num=inlist.split(",");
```

```
        if(ifShui(num[0]) && ifShui(num[1]))
```

```
            sort(num[0],num[1]);
```

```
        else if(ifShui(num[0]))
```

```
            System.out.println(num[0]);
```

```
        else if(ifShui(num[1]))
```

```
            System.out.println(num[1]);
```

```
        else
```

```
            System.out.println("NO DATA");
```

```
    }
```

```
    static boolean ifShui(String a){
```

```
        int b=Integer.parseInt(a);
```

```
        int gw,sw,bw,total;
```

```
        bw=b/100;
```

```
        sw=(b-bw*100)/10;
```

```
        gw=(b-bw*100-sw*10);
```

```
        total=(bw*bw*bw)+(sw*sw*sw)+(gw*gw*gw);
```

```
        if(total==b)
```

```
            return true;
```

```
        else
```

```
            return false;
```

```
    }
```

```
    static void sort(int a,int b){
```

```
        if(a<b)
```

```
        System.out.println(a+","+b);
    else
        System.out.println(b+","+a);
    }
}
```

字节跳动面试：测试

## 1.linux 的一些命令：

管道 | ， 怎么使用 ls | grep .c

后台的方式执行 ls -l &

后台的方式执行该命令，显示后台运行程序的进程 PID，shell 返回命令提示符状态

grep 正则搜索式

## 2.git 工具的使用

### 普通功能

git init 把当前目录变为可以管理的仓库

git clone 远程仓库地址 将远程仓库克隆到本地仓库

git add . 将所有修改了的代码添加到暂存区

git commit -m "note" 将暂存区的内容放入缓存区准备提交到远程仓库

git push 将缓存区的内容更新到远程仓库

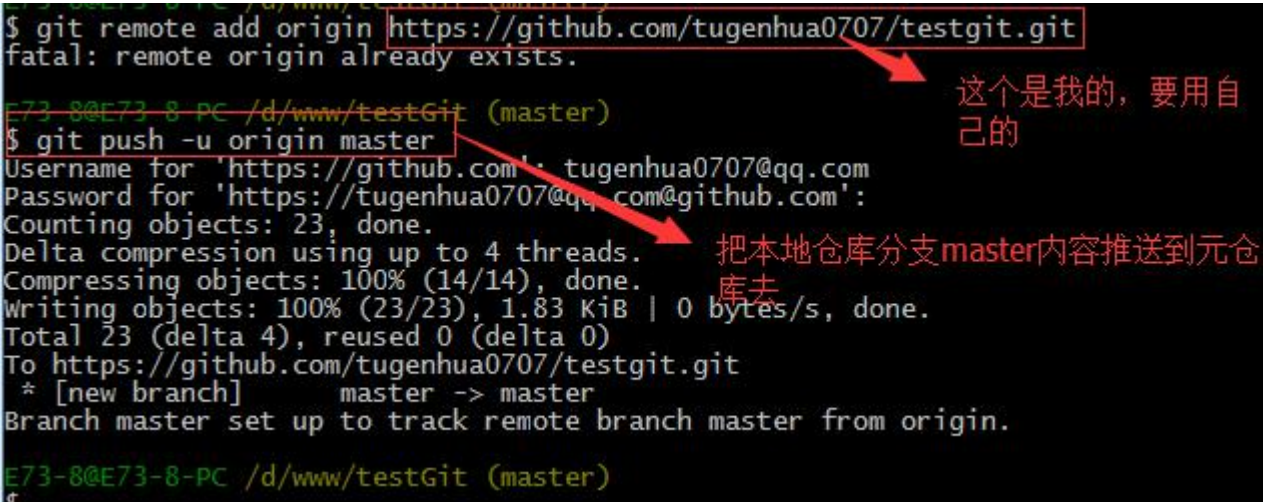
git pull 将远程仓库同步到本地仓库（可用 git fetch, get merge 代替更加安全）

git fetch 不会自动 merge

git remote add origin https://github.com/tughenhu0707/testgit.git 第一次把本地仓库上传到远程仓库如果想改远程仓库地址：git remote rm origin

### 正确步骤

1. git init //初始化仓库
2. git add .(文件 name) //添加文件到本地仓库
3. git commit -m "first commit" //添加文件描述信息
4. git remote add origin + 远程仓库地址 //链接远程仓库，创建主分支
5. git pull origin master // 把本地仓库的变化连接到远程仓库主分支
6. git push -u origin master //把本地仓库的文件推送到远程仓库



```
$ git remote add origin https://github.com/tughenhu0707/testgit.git
fatal: remote origin already exists.

E73-8@E73-8-PC /d/www/testGit (master)
$ git push -u origin master
Username for 'https://github.com': tughenhu0707@qq.com
Password for 'https://tughenhu0707@qq.com@github.com':
Counting objects: 23, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (14/14), done.
Writing objects: 100% (23/23), 1.83 KiB | 0 bytes/s, done.
Total 23 (delta 4), reused 0 (delta 0)
To https://github.com/tughenhu0707/testgit.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.

E73-8@E73-8-PC /d/www/testGit (master)
```

这个是我的，要用自己的

把本地仓库分支master内容推送到元仓库去

版本跟踪功能:

git status 查看是否有被更改过

```
E73-8@E73-8-PC /d/www/testgit (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   readme.txt

no changes added to commit (use "git add" and/or "git commit -a")
E73-8@E73-8-PC /d/www/testgit (master)
$
```

git diff readme.txt 查看到底更改了什么内容

版本回退

git log (--pretty=online)查看历史修改版本

git reset --hard HEAD~100 回退到上 n 个版本

git reflog 查看版本号

git reset --hard 版本号

撤销操作

1. 未使用 git add 加入暂存区

// 放弃单个文件修改,注意不要忘记中间的"--",不写就成了检出分支了!

git checkout -- filepathname

// 放弃所有的文件修改

git checkout .

2. 已使用 git add

git reset HEAD filepathname 返回到没有 add 之前的状态

3. 已使用 git commit

git reset --hard HEAD^ 来回退到上一次 commit 的状态。

分支问题

创建新分支: git branch branchName

切换到新分支: git checkout branchName

上面两个命令也可以合成为一个命令: git checkout -b branchName

3. 关于项目的内容

1. 不想让类被继承可以使用什么修饰符: 只有 final

静态类不能被继承 (其继承类不能被实例化, 等于不能继承)

关于 static 变量和方法的继承: 父类中的静态成员变量和方法是**可以被子类继承的,但是不能被自己重写**,无法形成多态.

2. JAVA 的垃圾回收机制是什么样的

垃圾回收 (Garbage Collection, GC)

引用计数法 (python 在用, 但 java 里无法解决循环引用)

Java 中采取了 可达性分析法. 该方法的基本思想是通过一系列的 “GC Roots” 对象作为起点进行搜索, 如果在 “GC Roots” 和一个对象之间没有可达路径, 则称该对象是不可达的, 不过要注意的是被判定为不可达的对象不一定会成为可回收对象. 被判定为不可达的对象要成为可回收对象必须至少经历两次标记过程, 如果在这两次标记过程中仍然没有逃脱成为

可回收对象的可能性，则基本上就真的成为可回收对象了。

1) 显示地将某个引用赋值为 `null` 或者将已经指向某个对象的引用指向新的对象，比如下面的代码：

2) 局部引用所指向的对象，比如下面这段代码：

```
void fun() {  
  
.....  
  
    for(int i=0;i<10;i++) {  
        Object obj = new Object();  
  
        System.out.println(obj.getClass());  
  
    }  
}
```

循环每执行完一次，生成的 `Object` 对象都会成为可回收的对象。

3.JDBC 链接溢出的话，如何关闭，如何保证不会出现异常

4.制作的项目使用的可视化界面用的是 `swing`，其中使用的是什么 `layout`

5.如何建立与数据库的连接

6.修饰符什么都不写的情况下默认访问权限

缺省情况下，是 `default`，只能被同包的访问。

类中默认是 `default`，接口默认 `public`

| 修饰符       | 同一个类 | 同一个包 | 不同包的子类 | 不同包的非子类 |
|-----------|------|------|--------|---------|
| Private   | ✓    | ×    | ×      | ×       |
| Default   | ✓    | ✓    | ×      | ×       |
| Protected | ✓    | ✓    | ✓      | ×       |
| Public    | ✓    | ✓    | ✓      | ✓       |

判断一个范围内的数是不是素数回文

```
import java.util.*;  
public class Main{  
    public static void main(String[] args){  
        Scanner sc = new Scanner(System.in);  
        int a=sc.nextInt();  
        int b=sc.nextInt();  
        int count=0;  
        for(int i=a;i<=b;i++){  
            if(ifHuiwen(i) && ifSushu(i))  
                count++;  
        }  
    }  
}
```

```

        System.out.println(count);
    }
    static boolean ifHuiwen(int a){
        String text=Integer.toString(a);
        StringBuffer sb=new StringBuffer(text);
        String text2=sb.reverse().toString();
        for(int i=0;i<text.length();i++){
            if(text.charAt(i)==text2.charAt(i)) i++;
            else return false;
        }
        return true;
    }

    static boolean ifSushu(int a){
        int i = 2;
        while(i<a){
            if(a%i!=0) i++;
            else return false;
        }
        return true;
    }
}

```

1.如何判断回文：使用 StringBuffer 的 reverse 方法，然后用 str.charAt()进行判断。要注意 String 类没有 reverse 方法。需要先将 String 类转化成 StringBuffer 类型：StringBuffer sb=new StringBuffer(str)

2.如何将 int 转化为 String：Integer.toString(int)

3.如何将 String 化为 char 数组：str.toCharArray();