**JAVA笔记**

目录

[一．简单运算 5](#_Toc464941532)

[二．运算符 6](#_Toc464941533)

[1. 位运算 6](#_Toc464941534)

[2. & | ^ ~ 的位运算 6](#_Toc464941535)

[3. y=(x>1)?100:200; 7](#_Toc464941536)

[4. switch() 7](#_Toc464941537)

[三. 循环、堆、栈 7](#_Toc464941538)

[1. for循环 7](#_Toc464941539)

[2. 最简单无限循环形式 7](#_Toc464941540)

[3. 重载 7](#_Toc464941541)

[4. new 7](#_Toc464941542)

[5. java程序运行时的五块存储区 8](#_Toc464941543)

[四. 排序 8](#_Toc464941544)

[1. 数组赋值 8](#_Toc464941545)

[2. 选择排序 8](#_Toc464941546)

[3. 冒泡排序 9](#_Toc464941547)

[4. 希尔排序 9](#_Toc464941548)

[五. 对象、封装 9](#_Toc464941549)

[1. 万物皆对象 9](#_Toc464941550)

[2. 匿名对象 9](#_Toc464941551)

[3. 封装 9](#_Toc464941552)

[4. 构造函数 9](#_Toc464941553)

[5. 构造代码块 9](#_Toc464941554)

[6. this 10](#_Toc464941555)

[六. 静态、文档 10](#_Toc464941556)

[1. static: 10](#_Toc464941557)

[2. static注意事项： 10](#_Toc464941558)

[3. static利弊： 11](#_Toc464941559)

[4. 主函数main是static的 11](#_Toc464941560)

[5. 不能创建对象的类 11](#_Toc464941561)

[6. 文档 11](#_Toc464941562)

[7. 类的默认构造函数权限与类的权限一致 12](#_Toc464941563)

[8. 静态代码块 12](#_Toc464941564)

[9. 对象的初始化过程 p a = new p(); 12](#_Toc464941565)

[10. 单例设计模式 12](#_Toc464941566)

[七. 继承、抽象、接口 13](#_Toc464941567)

[1. 继承 extends 13](#_Toc464941568)

[2. 聚集（关系） 14](#_Toc464941569)

[3. 子父类中变量的特点 14](#_Toc464941570)

[4. 子父类中函数的特点 14](#_Toc464941571)

[5. 字父类中构造函数的特点 14](#_Toc464941572)

[6. final关键字 15](#_Toc464941573)

[7. 抽象类 15](#_Toc464941574)

[8. 模板方法设计模式 15](#_Toc464941575)

[9. 接口 interface 15](#_Toc464941576)

[八. 多态、Object中的方法 16](#_Toc464941577)

[1. 多态 16](#_Toc464941578)

[2. 多态的体现 16](#_Toc464941579)

[3. 多态的前提 16](#_Toc464941580)

[4. 多态的好处 16](#_Toc464941581)

[5. 多态的弊端 16](#_Toc464941582)

[6. 自动转型 16](#_Toc464941583)

[7. if(a instanceof b) 17](#_Toc464941584)

[8. 在多态中非静态成员函数的特点 17](#_Toc464941585)

[9. 接口提高功能扩展性即程序扩展性——多态 17](#_Toc464941586)

[10. Object 18](#_Toc464941587)

[11. Object.equals() 19](#_Toc464941588)

[12. Object.toString() 19](#_Toc464941589)

[13. Object.getClass(); 19](#_Toc464941590)

[九. 内部类、异常机制 20](#_Toc464941591)

[1. 内部类 20](#_Toc464941592)

[2. 访问格式 20](#_Toc464941593)

[4. 匿名内部类（匿名子类对象） 21](#_Toc464941594)

[5. 异常机制 22](#_Toc464941595)

[6. 异常处理 （多异常的处理） 22](#_Toc464941596)

[7. 自定义异常 23](#_Toc464941597)

[8. Exception子类中的RuntimeException 24](#_Toc464941598)

[十. 异常、jar包 24](#_Toc464941599)

[1. finally代码块 24](#_Toc464941600)

[2. catch用于处理异常 25](#_Toc464941601)

[3. 异常在子父类覆盖时的体现 25](#_Toc464941602)

[4. 包（package） 25](#_Toc464941603)

[5. import（简化类名的书写） 26](#_Toc464941604)

[6. jar包（Java的压缩包） 26](#_Toc464941605)

[十一. 多线程 27](#_Toc464941606)

[1. 多线程 27](#_Toc464941607)

[2. 创建线程 27](#_Toc464941608)

[4. 线程的多种状态 28](#_Toc464941609)

[5. 多线程的安全问题 28](#_Toc464941610)

[6. 安全问题的解决办法——同步（锁） 30](#_Toc464941611)

[7. 单例设计模型中的同步--\*\* 31](#_Toc464941612)

[8. 死锁--导致程序停止运行 32](#_Toc464941613)

[十二. 线程间通讯 32](#_Toc464941614)

[1. 线程间通讯 32](#_Toc464941615)

[2. 代码优化后 35](#_Toc464941616)

[3. 同一资源被多个线程共同操作时 38](#_Toc464941617)

[4. JDK1.5及之后（JDK5.0）synchroized被替代 39](#_Toc464941618)

[十三. String类 41](#_Toc464941619)

[1. String类（final class String） 41](#_Toc464941620)

[1.1 概述 41](#_Toc464941621)

[1.2用于描述字符串这类事物，常见操作有： 41](#_Toc464941622)

[2. StringBuffer类 43](#_Toc464941623)

[3. StringBuilder类 43](#_Toc464941624)

[4. 基本数据类型对象封装类 43](#_Toc464941625)

[十四. 集合框架 44](#_Toc464941626)

[1. 集合框架 44](#_Toc464941627)

[2. Collection超类接口 45](#_Toc464941628)

[3. interface Iterator;迭代器 45](#_Toc464941629)

[4. collection子接口List 46](#_Toc464941630)

[4.1 List特有方法： 46](#_Toc464941631)

[4.2 LinkedList 特有方法： 47](#_Toc464941632)

[5. Collection子接口Set 47](#_Toc464941633)

[5.1 HashSet 48](#_Toc464941634)

[5.2 TreeSet 48](#_Toc464941635)

[6. 泛型 49](#_Toc464941636)

[6.1 何时定义泛型类？ 50](#_Toc464941637)

[6.2 何时定义泛型方法？ 50](#_Toc464941638)

[6.3 泛型接口 51](#_Toc464941639)

[6.4 <?>占位符 51](#_Toc464941640)

[十五. (十六) Collection子接口Map 52](#_Toc464941641)

[1. Map集合 52](#_Toc464941642)

[2. 方法 52](#_Toc464941643)

[3. HashMap 53](#_Toc464941644)

[4. TreeMap进行排序 53](#_Toc464941645)

[4.1 用法 53](#_Toc464941646)

[4.2 TreeMap练习，统计一个字符串中各个字母的出现次数 53](#_Toc464941647)

[5. Map集合扩展知识？ 54](#_Toc464941648)

[6. 集合框架的构成及分类 56](#_Toc464941649)

[十七. 集合和数组工具类 56](#_Toc464941650)

[1. Collections 集合工具类 56](#_Toc464941651)

[1.1 <T extends Comparable<? super T>> void sort(List<T> list) 56](#_Toc464941652)

[1.2 <T> void sort(List<T> list,Comparator<? super T> c) 56](#_Toc464941653)

[1.3 <T> T max(Collection<? extends T> coll,Comparator<? super T> com) 58](#_Toc464941654)

[1.4 <T extends Object & Comparable<? super T>> T max(Collection<? extends T> coll) 58](#_Toc464941655)

[1.5<T> int binarySearch(List<? extends Comparable<? super T>> list, T key) 58](#_Toc464941656)

[1.6 <T> int binarySearch(List<? extends T> list,T key,Comparator<? super T> c) 58](#_Toc464941657)

[1.7 <T> void fill(List<? super T> list, T obj) 58](#_Toc464941658)

[1.8 static <T> boolean replaceAll(List<T> list, T oldVal, T newVal) 58](#_Toc464941659)

[1.9static void reverse(List<?> list) 59](#_Toc464941660)

[1.10 static <T> Comparator<T> reverseOrder() 59](#_Toc464941661)

[1.11 static <T> Comparator<T> reverseOrder(Comparator<T> cmp) 59](#_Toc464941662)

[1.12 static <T> Collection<T> synchronizedCollection(Collection<T> c) 59](#_Toc464941663)

[1.13 static void swap(List<?> list, int i, int j) 59](#_Toc464941664)

[2. Arrays 数组工具类 59](#_Toc464941665)

[3 数组变集合 60](#_Toc464941666)

[3. 集合变数组 60](#_Toc464941667)

[4. 增强for循环（foreach） 60](#_Toc464941668)

[5. 可变参数 61](#_Toc464941669)

[6. 静态导入 61](#_Toc464941670)

[十八.其他对象 61](#_Toc464941671)

[1. System 61](#_Toc464941672)

[2. Runtime 61](#_Toc464941673)

[3. Date日期 62](#_Toc464941674)

[4. Calendar 日历 62](#_Toc464941675)

[5. Math 62](#_Toc464941676)

[6. IO（Input Output） 63](#_Toc464941677)

[7. 字符流的两个基类：Writer Reader 64](#_Toc464941678)

[1. Writer 64](#_Toc464941679)

[2. Reader 66](#_Toc464941680)

[8. 复制文件原理： 67](#_Toc464941681)

[十九. IO 69](#_Toc464941682)

[1. 字符流的缓冲区 69](#_Toc464941683)

[1. BufferedWriter 69](#_Toc464941684)

[2. BufferedReader 70](#_Toc464941685)

[3. 装饰设计模式 71](#_Toc464941686)

[3. 字节流（操作图片、MP3等文件） 72](#_Toc464941687)

[4. 键盘输入模拟 73](#_Toc464941688)

[5. 读取转换流 74](#_Toc464941689)

[6. 流操作的基本规律 74](#_Toc464941690)

[二十二. GUI 75](#_Toc464941691)

[1. GUI 75](#_Toc464941692)

[2. 继承关系 75](#_Toc464941693)

[3. 布局管理器 76](#_Toc464941694)

[二十三. 网络编程 77](#_Toc464941695)

[1. 网络编程 77](#_Toc464941696)

[2. 网络参考模型 77](#_Toc464941697)

[3. 网络通讯要素 78](#_Toc464941698)

[4. TCP和UDP 78](#_Toc464941699)

[5. java.net.InetAddress简单使用 78](#_Toc464941700)

[6. Socket 79](#_Toc464941701)

[7. UDP传输 80](#_Toc464941702)

[7.1 简单聊天 81](#_Toc464941703)

[8. TCP传输 84](#_Toc464941704)

[9. PrintWriter自动有效刷新和添加换行 87](#_Toc464941705)

[10. 开启多个进程同时可接受来自不同主机的图片上传请求 87](#_Toc464941706)

[11. 多线程处理图片上传 90](#_Toc464941707)

[12. 自定义简单服务端 92](#_Toc464941708)

[13. Tomcat7.0 93](#_Toc464941709)

[14. java.net.URL 93](#_Toc464941710)

[15. java.net.URLConnection 94](#_Toc464941711)

[16. 域名解析 94](#_Toc464941712)

[二十五. 正则表达式 95](#_Toc464941713)

[1. 正则表达式 95](#_Toc464941714)

[2. 匹配 95](#_Toc464941715)

[3. 切割 95](#_Toc464941716)

[4. 替换 96](#_Toc464941717)

[5. 获取 96](#_Toc464941718)

[6. 实例 97](#_Toc464941719)

[7. 网页爬虫（蜘蛛） 97](#_Toc464941720)

# 一．简单运算

**int x=3;**

**short s=5;**

**s=s+2; //编译报错---先加法后赋值---加法运算时自动类型提升可能损失精度**

**s+=2; //编译通过---一次运算--左右两边的和赋给左边--只进行赋值时的自动类型转换 int -> short**

**&(单与): int a=4; a>5 & a<6; //两次运算 a>5为false，再a<6为true。则表达式结果为false**

**&&(双与--短路): int a=4; a>5 && a<6; //一次运算 a>5为false，则表达式结果为false（第一次判断为false就得出最终结果）**

**|:同理**

**||:同理**

# 二．运算符

**\* 一个数a异或同一个数b两次，结果还是a。(可用于“加密”)a公共可见，b作为秘钥**

## 1. 位运算

**物理计算速度最快**

**左移 越移越大**

**3<<2 = 12; 3\*2^2 = 12；结果 = 原数\*二的移动位数次幂**

**3 = 0000-0000 0000-0000 0000-0000 0000-0011**

**移位：00|00-0000 0000-0000 0000-0000 0000-0011**

**00-0000 0000-0000 0000-0000 0000-001100 = 12**

**右移 越移越小**

**6>>2 = 1; 6 / 2^2 = 1;结果 = 原数 除以 二的移动位数次幂**

**6 = 0000-0000 0000-0000 0000-0000 0000-0110**

**移位：0000-0000 0000-0000 0000-0000 0000-01|10**

**000000-0000 0000-0000 0000-0000 0000-01 = 1**

## 2. & | ^ ~ 的位运算

**&:6&3=2；**

**110 （0视为false，1视为true）**

**&011**

**-----**

**010**

**|: 6 | 5=7；**

**110 （0视为false，1视为true）**

**| 101**

**-----**

**111**

**^:同理（异或）**

**~:即对应二进制的反码（反）**

## 3. y=(x>1)?100:200;

**三元运算符 可以简化(if...else)代码，其本质为运算符 必须有“返回值”**

## 4. switch()

**不顺序执行 case和default全加载进内存，与if形成对比**

**只能为byte short int char String enum**

**case之间与default没有顺序，检查完case没有匹配的才执行default**

**若case内没有break则会继续检查case，直到遇到break或switch结尾**

# 三. 循环、堆、栈

## 1. for循环

**不一定都为int i=0;可以为其他表达式 如system.out.println()**

**for(System.out.println("a");x<4;System.out.println("c"))**

**for(System.out.println("a");x<4;System.out.println("c"),x++)**

## 2. 最简单无限循环形式

**for：for( ; ; )**

**while: while(true)**

## 3. 重载

**同名函数，通过参数列表（有顺序）区别**

**与返回值类型无关，可以不同**

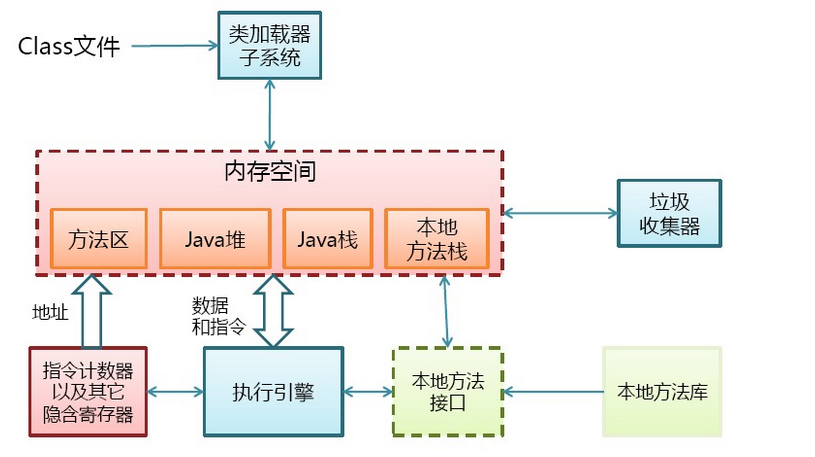
**当定义的功能相同，但参与运算的内容未知或不同时使用**

## 4. new

**new操作符用于在内存中产生一个容器实体（开辟一块空间通常在堆内存中）**

## 5. java程序运行时的五块存储区

**java程序运行时会在内存中开辟五块存储区**

****

**1.栈内存：用于存储局部变量，当数据使用完，所占空间会自动释放**

**2.堆内存：数组、对象和一切new创建的实例**

**每个实体都有内存地址值**

**\* 实体中的变量都有默认初始化值（默认值由数据类型决定）**

**实体不再被使用时，会在不确定的时间内被垃圾回收器回收**

**（内存垃圾c++程序员手动清除，而java由JVM检测后不定时被垃圾回收机制自动清除）**

**3.方法区：**

**4.本地方法区：**

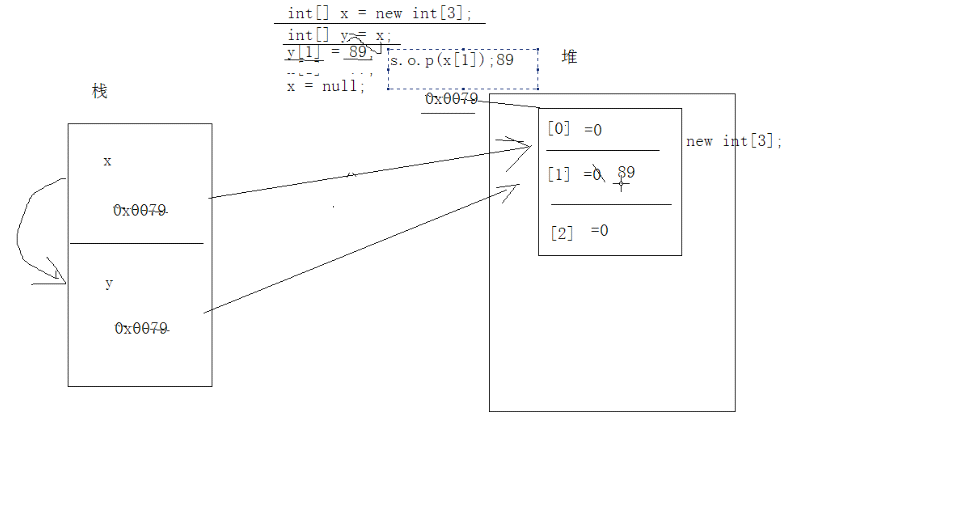
**5.寄存器：**

**这是最快的存储区，因为它位于不同于其他存储区的地方——处理器内部。但是寄存器的数量极其有限，所以寄存器由编译器根据需求进行分配。你不能直接控制，也不能在程序中感觉到寄存器存在的任何迹象。**

# 四. 排序

## 1. 数组赋值

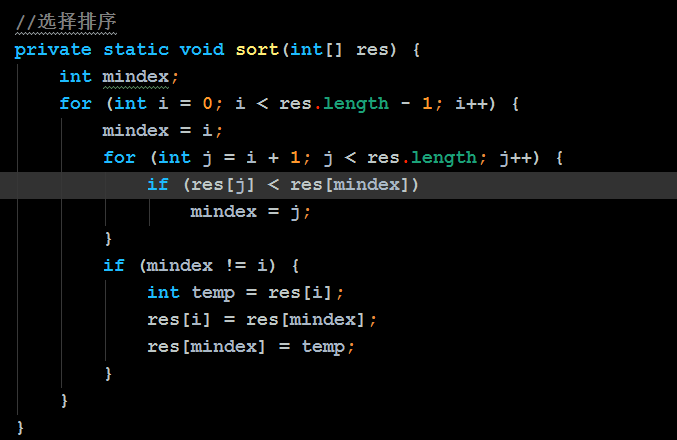
**指向同一块在堆中的内存区间——引用型数据类型共有特点**

****

## 2. 选择排序

**数组a的a[mindex]与a[i+1]比较，若后一个数小于前一个数则交换位置(升序)**

**两层循环，每一次循环（外层）找出当前剩余的（内层里）最大/最小（有的话），并与剩余里的第一个位置的交换，选择排序是不稳定的排序方法（比如序列[5， 5， 3]第一次就将第一个[5]与[3]交换，导致第一个5挪动到第二个5后面）。**

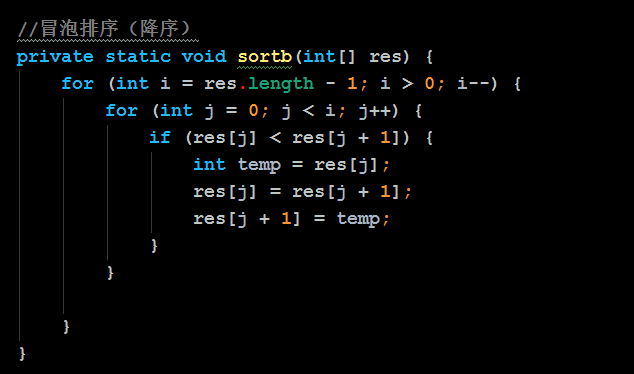


## 3. 冒泡排序

**相邻的两个比较，若符合条件就交换位置，一次比较把第一大(第一小)的移动到最后，**

**参与比较元素个数依次减少**

**两层循环，从最后一个开始（外层循环），依次比较前一个和后一个（内存循环），大小不等就交换，一次循环（外层）会把最大/最小移动到最后面。 前后元素相等不会交换（两数相等时位置不会变化）——稳定的排序算法**



## 4. 希尔排序

**三层循环+位运算（速度最快，效率最高）**

**开发中数组排序用 Arrays.sort(int[] array)**

# 五. 对象、封装

## 1. 万物皆对象

## 2. 匿名对象

当对象的方法只调用一次时，可以用匿名对象完成。当对对象进行多个成员调用时，必须给对象起一个名字。

**可以将匿名对象作为实际参数进行传递。 showCar( new Car() );**

## 3. 封装

**1.将不需要对外提供的内容都隐藏起来，把属性都隐藏，提供公共方法对齐方法**

**2.私有仅仅是封装的一种形式**

**3.之所以对外提供访问方式，是因为可以在访问方式中加入逻辑判断语句，提高健壮性。**

## 4. 构造函数

**给对象初始化，构造函数可以重载**

**区别于一般方法。对象一建立就运行，且只执行一次**

## 5. 构造代码块

**对象一建立就运行，且优先于构造函数**

**给所有该类下对象统一的初始化，而构造函数是给不同对象初始化**

**{**

**System.out.println("empty");**

**}**

## 6. this

**哪个对象在调用this所在的函数，this就代表那个对象**

**构造函数间调用用this()传的参数类型要对应（避免出现调用死循环）**

**\* this()只能放在第一行(先执行更细的初始化)，只能用一次，只能在构造函数间使用，不允许在一般函数中使用。**

**class p**

**{**

**int a;**

**String name;**

**p(int a)**

**{**

**this.a=a;**

**}**

**p(String n,int a)**

**{**

**this(a); //传入的a又通过p(int a)初始化**

**this.name=n;**

**}**

**}**

# 六. 静态、文档

## 1. static:

**修饰成员变量或（函数），不能修饰局部。**

**1.被对象共享，在内存中只存在一份，存在于方法区中。**

**2.除被对象调用外还可以直接被类名调用。类名.静态成员**

**3.随类的加载而加载，随类的消失而消失（类中其生命周期最长）**

**4.优先于对象存在**

## 2. static注意事项：

**静态方法只能访问静态成员（方法/变量）**

**静态方法中不能存在this关键字（此时对象还未new出来）**

**非静态可以访问静态（静态优先于对象存在）**

## 3. static利弊：

**利：对类的共享数据进行单独空间存储，省空间，不需要每个对象都生成一份**

**弊：访问局限(只能访问静态)，生命周期过长。**

## 4. 主函数main是static的

**public static void main(String[] args)**

**JVM识别的入口main格式固定唯一（参数类型），但可以重载**

**JVM调用main时传的args为 new String[0]**

**被JVM调用作为程序入口**

**public： 访问权限最大**

**static ：随类的加载必定存在**

**void ：主函数没有具体返回值**

**main ：不是关键字，可以被JVM唯一识别**

**String[] args：字符串数组**

## 5. 不能创建对象的类

**如果类中所有的方法都为静态方法（用类名可以直接调用方法），可以将类的构造函数初始为私有private，以使类不能创建对象。（能隐藏的都隐藏起来）**

## 6. 文档

**Java的说明书通过文档注释来完成(api文档)**

**cmd命令：javadoc -d myhelp -author -version 123.java**

**可被文档注释工具自动识别，生成.html文件，打开index索引文件查看**

**/\*\***

**功能**

**@author 作者**

**@param arr 数组**

**@return 返回值**

**..............**

**\*/**

## 7. 类的默认构造函数权限与类的权限一致

**private class p{**

**private p(){}**

**}**

## 8. 静态代码块

**1.static**

**{**

**语句**

**}**

**随着类的加载而执行，只执行一次。用于给类初始化。**

**只能访问静态变量，不能使用this**

**2.**

**class p**

**{**

**static{} //静态构造代码块**

**{} //构造代码块**

**}**

**p a = null; //类并未加载进内存**

**new p();//先类加载进内存，再在内存中创建无实体对象引用的对象内存**

**p a = new p(); //先类加载进内存，再在堆内存中创建有实体对象引用的内存**

**用到类中内容才会加载类到内存中**

## 9. 对象的初始化过程 p a = new p();

**1.JVM找到并加载p.class加载进内存**

**2.若有static代码块，则先执行进行类的初始化**

**3.在堆内开辟空间，分配内存地址**

**4.在堆内存中建立对象的特有属性，并进行默认初始化**

**5.对属性进行显示初始化**

**6.对对象进行构造代码块初始化**

**7.对对象进行对应的构造函数初始化、**

**8.将内存地址赋给栈内存中的a变量**

**4.对象调用成员过程 p a = new p(); a.setName("duan");**

**..........................**

## 10. 单例设计模式

**（在堆内存中只有一个对象空间，栈使用时引用同一个地址）**

**class test**

**{**

**private int number;**

**private test(){} //构造函数私有，使类外无法创建对象**

**static test t = new test(); //类内创建本类对象**

**public static test getInstance() //对外提供获取该类唯一对象的方法**

**{**

**return t;**

**}**

**public void set(int a)**

**{**

**number = a;**

**}**

**public int get()**

**{**

**return number;**

**}**

**}**

**class main**

**{**

**public static void main(String[] args)**

**{**

**test s = test.getInstance();**

**s.set(20);**

**test s2 = test.getInstance();**

**System.out.println(s2.get()); //输出值为20**

**}**

**}**

**内存中有且只有一份static test t;对象**

# 七. 继承、抽象、接口

## 1. 继承 extends

**----is a**

**1.不能为了获取类的方法或简化代码而继承，必须类与类间有所属关系is a**

**2.Java只支持单个继承（一个类只能有一个父类），不支持多继承（多继承容易带来安全隐患，产生歧义）**

**3.Java支持多重继承（一个类的父类还可以有父类....），也就是一个继承体系。要想使用体系，先查阅父类的描述。**

## 2. 聚集（关系）

**----has a**

**聚合：个体与班级，个体球队**

**组合：心脏与身体**

## 3. 子父类中变量的特点

**有非私有同名变量时子类对象默认访问子类变量，要访问父类时使用关键字super**

## 4. 子父类中函数的特点

**当子类出现与父类一模一样的函数时，子类调用该方法时会默认调用子类的方法。——覆写**

**class Fu**

**{**

**void show()**

**{ ......**

**}**

**}**

**class Zi extends Fu**

**{**

**void show()**

**{**

**super.show();**

**.....**

**}**

**}**

**子类覆盖父类必须保证子类权限不小于父类，才可以覆盖。**

**静态只能覆盖静态**

## 5. 字父类中构造函数的特点

**子类构造函数第一行默认隐式调用父类的默认空参构造函数语句——super()。**

**为什么子类构造函数一定会访问父类空参构造函数？**

**因为父类中的数据子类可以直接获取，所有子类在对象建立时需要查看父类时如何对数据进行初始化的，如果要访问父类中指定的构造函数可以手动指定要访问的构造函数。**

## 6. final关键字

**可以修饰：  
类：修饰的类不可以被继承，避免被继承被覆写。**

**方法：方法不可以被覆写**

**变量：是一个常量，只能赋值一次，可修饰成员也可修饰局部。**

**(1).内部类是外部类的一个成员，就像外部类的成员方法一样，所以内部类有权限访问外部类的所有成员，包括private的。  
(2).内部类不能访问外部类方法中的局部变量，除非变量是final的(一般发生在方法中定义的内部类)。这是因为局部变量的生命周期原因。http://blog.csdn.net/cpp597455873/article/details/46287853**

## 7. 抽象类

**abstract class stuff**

**{**

**abstract void study();**

**abstract boolean judge(boolean b);**

**void show();**

**}**

**向上抽取，只取功能定义，而不抽取功能主体**

**1.抽象方法一定定义在抽象类中**

**2.抽象方法和抽象类都必须被abstract修饰**

**3.抽象类不可以用new创建对象，因为调用抽象方法没有意义**

**4.抽象类中的方法要被使用，必须由子类复写所有的抽象方法后才能建立子类对象调用，如果子类只覆盖了部分抽象方法，那么该子类还是一个抽象类。**

**5.抽象类也可以定义非抽象方法，这样可以阻止该抽象类建立对象**

## 8. 模板方法设计模式

**在定义功能时，功能的一部分是确定的，但有一部分不确定，而确定的部分在使用不确定的部分，这时就把不确定的部分暴露出去，由子类完成。**

## 9. 接口 interface

**初期可理解为：所有方法都为抽象方法的抽象类**

**1.接口中常见定义：常量，方法**

**2.接口中成员都有固定修饰符：**

**常量：public static final \*；**

**方法：public abstract \* \*()；**

**3.接口中成员都默认且固定是public**

**4.接口不可创建对象，需被子类有关键字implements实现且覆盖接口内抽象方法，子类才可以被实例化，否则子类也是抽象类。**

**5.接口可以被类多实现。**

**6.接口与接口间支持多继承。（类支持单继承多实现，接口支持多继承）**

# 八. 多态、Object中的方法

## 1. 多态

**涵盖：对象，方法（重载、覆盖）......**

**事物存在的多种不同体现形态。**

**猫 x = new 猫() == 动物 x = new 猫()**

## 2. 多态的体现

**父类的引用指向了自己的子类对象**

**即父类的引用也可以接収自己的子类对象**

**class A;**

**class B extends A;**

**A a = new B();**

## 3. 多态的前提

**1.必须是类与类之间有关系的，通过或继承，或实现的方式完成。**

**2.通常存在覆盖**

## 4. 多态的好处

**可以提高代码的扩展性**

## 5. 多态的弊端

**只能使用父类的引用父类中的成员。**

## 6. 自动转型

**class Animal;**

**class Cat extends Animal;**

**~~Animal a = new Cat(); //类型提升，自动向上转型。~~**

**~~a.eat(); //调用父类Animal的eat方法~~**

**~~//如果想调用子类Cat的特有方法，需要向下转型（前提为转型对象为~~*~~父类的~~*~~指向子类引用的~~*~~对象~~*~~）。~~**

**~~Cat c = (Cat)a; //强制类型转换~~**

**~~c.catchMouse(); //子类特有方法~~**

## 7. if(a instanceof b)

**instanceof 用于判断a对象引用指向类型是否为b**

**注意父类 b为父类a为子类时满足**

## 8. 在多态中非静态成员函数的特点

**1.在编译时期：参阅引用型变量所属的类中是否有调用的方法，如果有，编译通过，如果没有编译失败。**

**2.在运行时期：参阅创建对象（new）所属的类中是否有调用的方法**

**成员函数在多态调用时，编译看左边，运行看右边。**

**3.多态中，成员变量的特点，无论编译或运行都参考左边（引用型变量所属类）**

**4.在多态中，静态方法无论编译或运行都参考左边。**

## 9. 接口提高功能扩展性即程序扩展性——多态

**interface PCI**

**{**

**abstract public void open();**

**abstract public void close();**

**}**

**class MainBoard**

**{**

**public void run()**

**{**

**System.out.println("MainBoard run.");**

**}**

**public void usePCI(PCI p) //接口型引用指向自己的子类对象**

**{**

**if(p != null)**

**{**

**p.open();**

**p.close();**

**}**

**}**

**}**

**class NetBoard implements PCI**

**{**

**public void open()**

**{**

**System.out.println("NetBoard open.");**

**}**

**public void close()**

**{**

**System.out.println("NetBoard close.");**

**}**

**}**

**class MusicBoard implements PCI**

**{**

**public void open()**

**{**

**System.out.println("MusicBoard open.");**

**}**

**public void close()**

**{**

**System.out.println("MusicBoard close.");**

**}**

**}**

**class main**

**{**

**public static void main(String[] args)**

**{**

**MainBoard mainBoard = new MainBoard();**

**mainBoard.run();**

**mainBoard.usePCI(new NetBoard());**

**mainBoard.usePCI(new MusicBoard());**

**}**

**}**

**MainBoard run.**

**NetBoard open.**

**NetBoard close.**

**MusicBoard open.**

**MusicBoard close.**

## 10. Object

**定义了所有对象都具有的功能**

## 11. Object.equals()

比较是否为同一对象（可重写）

**所有对象都有的方法equals**

**默认比较引用型对象的地址引用值。**

**demo d1 = new demo();**

**demo d2 = new demo();**

**demo d3 = d1;**

**d1.equals(d2); //结果为false**

**d1.equals(d3); //结果为true**

**equals复写**

**public int num;**

**public boolean equals(Object obj)**

**{**

**if( !(obj instanceof demo) )**

**return false;**

**demo d = (demo)obj;**

**return this.num == d.num;**

**}**

## 12. Object.toString()

**将对象变为字符串**

**person p = new person();**

**p.toString() ——>person@659e0bfd //类名@哈希值（十六进制）**

**p.HashCode();**

**........**

## 13. Object.getClass();

**person p = new person();**

**Class c = p.getClass;**

**c.getName();**

**c.getMethod();**

**.........**

# 九. 内部类、异常机制

## 1. 内部类

**class Outer //外部类不能私有**

**{**

**private int x = 3;**

**public class Inner //内部类可以私有private**

**{**

**void function()**

**{ //内部类可以直接访问外部类中的成员包括私有**

**//Outer.this被默认省略，当内部类有与外部类同名成员时且局 部也有同 名成员时查找顺序：局部——this.\*——Outer.this.\***

**System.out.println("inner:"+Outer.this.x);**

**}**

**}**

**void method()**

**{ //外部类要访问内部类，必须建立内部类对象**

**Inner in = new Inner();**

**in.function();**

**}**

**}**

**class main**

**{**

**public static void main(String[] args)**

**{ //直接访问内部类**

**Outer.Inner in = new Outer().new Inner();**

**in.function();**

**}**

**}**

## 2. 访问格式

**1.当内部类定义在外部类的成员位置上，而且非私有，则可以在外部其他类中直接 建立内部类对象对其进行访问。**

**格式：Outer.Inner in = new Outer().new Inner();**

**2.内部类可以被修饰符修饰，private static**

**private：将内部类在外部类中进行封装**

**static：对应static的特性，只能直接访问外部类中的静态成员**

**在外部其他类中访问静态内部类的非静态成员：new Outer.Inner().function()**

**在外部其他类中访问静态内部类的静态成员：Outer.Inner.function()**

**3.当内部类中有静态成员时该内部类必须为静态内部类**

**3.当描述事物时，事物内部还有事物**

**4.内部类定义在局部位置**

**class Outer**

**{**

**int x = 3;**

**void method()**

**{**

**final int y = 5;//局部内部类访问外部类的成员方法中的局部变量时要用 final修饰**

**class Inner //内部类不在外部类的成员位置不能使用修饰符private或 static**

**{**

**void function()**

**{**

**System.out.println(Outer.this.x+y);**

**}**

**}**

**}**

**}**

## 4. 匿名内部类（匿名子类对象）

**new 父类(或接口)（） //构造函数**

**{**

**定义子类的内容 // 实现**

**1.要用的方法(父类已有的)或**

**2.必须实现的方法(抽象方法)或**

**3.自己的方法(自己添加方法)**

**}.方法名（调用方法）；**

**1.其实就是内部类的简写格式（就是一个匿名子类对象）——把定义类和new对象 以及调用方法封装成一体**

**2.匿名内部类必须继承一个类或实现接口**

**new AbsDemo()**

**{**

**void show()**

**{**

**........**

**}**

**........ //实现其他方法**

**}.show();**

**3.一个匿名内部类只能调用一个方法（也可以起个名字调用多个方法，但极少用到）**

**4.匿名内部类内部方法一般不超过三个**

## 5. 异常机制

**Throwable**

**|--Error**

**|--Exception**

**|--RuntimeException**

**1.程序在运行时出现的不正常情况**

**2.由来：问题也是现实生活中具体的事物，也可以封装为类（Java对不正常情况进行描述后的对象体现）**

**3.分类：1.严重问题：Java通过Error类进行描述，一般不编写针对性代码对其处理**

**2.非严重问题：Java通过Exception类进行描述，使用针对性处理方法处理**

**4.共性：引发原因，向上抽取得Throwable超类（已知子类Error和Exception）**

**2.异常分类**

**1.编译时被检测的异常**

**2.编译时不被检测的异常（运行时异常RuntimeException及其子类）**

## 6. 异常处理 （多异常的处理）

**class Demo**

**{**

**//在方法上通过throws关键字声明该功能可能会出现问题并抛出**

**//声明时建议声明更为具体的异常类型，这样处理得更为具体**

**//(多异常).....throws ArithmeticException , throws ArrayIndexOutOfBoundsException**

**//抛几个并且必须catch几个并处理，没throws的由JVM处理（异常停止）**

**public int deal(int a,int b) throws Exception**

**{**

**return a/b;**

**}**

**}**

**public static void main(String[] args[])**

**{**

**..............**

**try**

**{**

**//需要被检测的代码**

**;**

**System.out.println("run success.");**

**}**

**//如果多个catch块中的异常出现继承关系，父类catch块放在最下面**

**//throws几个就catch几个，不要定义多余的catch块**

**//建议进行catch处理时，一定要定义具体处理方法：**

**//处理方法：在本地保存异常日志等**

**catch(Exception e) //捕捉抛出的异常（Exception类没有针对性，会捕获所有异常）**

**{**

**//处理异常的代码（处理方式）**

**System.out.println("deal with exception.")**

**String str = e.getMessage(); //获取异常信息（ / by zero）**

**String str2 = e.toString(); //获取异常名称、异常信息**

**e.printStackTrace(); //打印出异常名称、异常信息、异常出现位置（JVM默认异 常处理语句）**

**//..........**

**}**

**catch(ArithmeticException e) //被0除异常（能具体处理）**

**{**

**...................**

**}**

**catch(ArrayIndexOutOfBoundsException e) //角标越界异常（能具体处理）**

**{**

**...................**

**}**

**finally**

**{**

**//一定会执行的语句**

**}**

**//异常成功处理之后继续运行**

**..............**

**}**

## 7. 自定义异常

**（有些特有问题未被Java封装为异常类因而可以定义自己的异常类用throw关键字抛出）**

**1.当在函数中用throw抛出异常对象，那么必须要给对应的处理动作。**

**要么在内部进行try() catch()处理**

**要么在函数上声明让调用者处理**

**2.如何定义异常信息？**

**因为父类已经把异常信息的操作都完成了，子类在构造是只需把异常信息通过 super语句传给父类即可。那么对象就可以直接调用getMessage()方法了。**

**FushuException extends Exception**

**{**

**FushuException(String message)**

**{**

**super(message);**

**}**

**}**

**3.必须是自定义类继承Exception**

**异常体系有一个特点，异常类和异常对象都需要被抛出，因而都具有可抛性，而 可抛性是Throwable超类体系独有特点，只有这个体系对象和类才可以被throw和 throws操作。**

**4.如果该异常的发生会使程序无法继续运行，可让自定义异常类继承RuntimeException**

## 8. Exception子类中的RuntimeException

**特点：**

**1.如果在函数内抛出该异常或其子类，函数上可以不用声明，编译可以通过**

**2.如果在函数上声明该异常或其子类，调用者可以不用处理，遍译可以通过**

**3.不用声明和不用处理的原因：因为该异常发生时，表示程序无法继续，JVM在程序运行时将停止程序以让开发人员对代码进行修正。**

# 十. 异常、jar包

## 1. finally代码块

**以数据库操作为例**

**定义一定会执行的代码 通常用于关闭资源**

**try**

**{**

**连接数据库**

**数据操作 //throw new SQLException();**

**}**

**catch(SQLException e)**

**{**

**会对数据库进行异常处理**

**...........**

**// 此位置有return;语句时finally0仍会被执行**

**//此位置有System.exit(0);语句时finally不会被执行（JVM结束，仅此一种情况finally不会执行）**

**}**

**finally**

**{**

**关闭数据库**

**}**

## 2. catch用于处理异常

**如果没有catch语句块代表异常没有被处理过**

## 3. 异常在子父类覆盖时的体现

**子类在覆盖父类方法时，如果父类的方法抛出异常，那么子类的覆盖方法只能抛出父类的异常或者该异常的子类**

**Exception //继承关系**

**|--AException**

**|--BException**

**|--CException**

**class demo**

**{**

**public void show() throws AException**

**{}**

**}**

**class des extends demo**

**{**

**public void show() //1.thows AException 抛父类的异常**

**{} //2.throws BException 抛父类异常的子类**

**//3.不抛**

**}**

## 4. 包（package）

***1.特点***

**1.对类文件进行分类管理**

**2.对类提供多层命名空间**

**3.写在程序文件的第一行**

**4.类名的全称是 包名.类名**

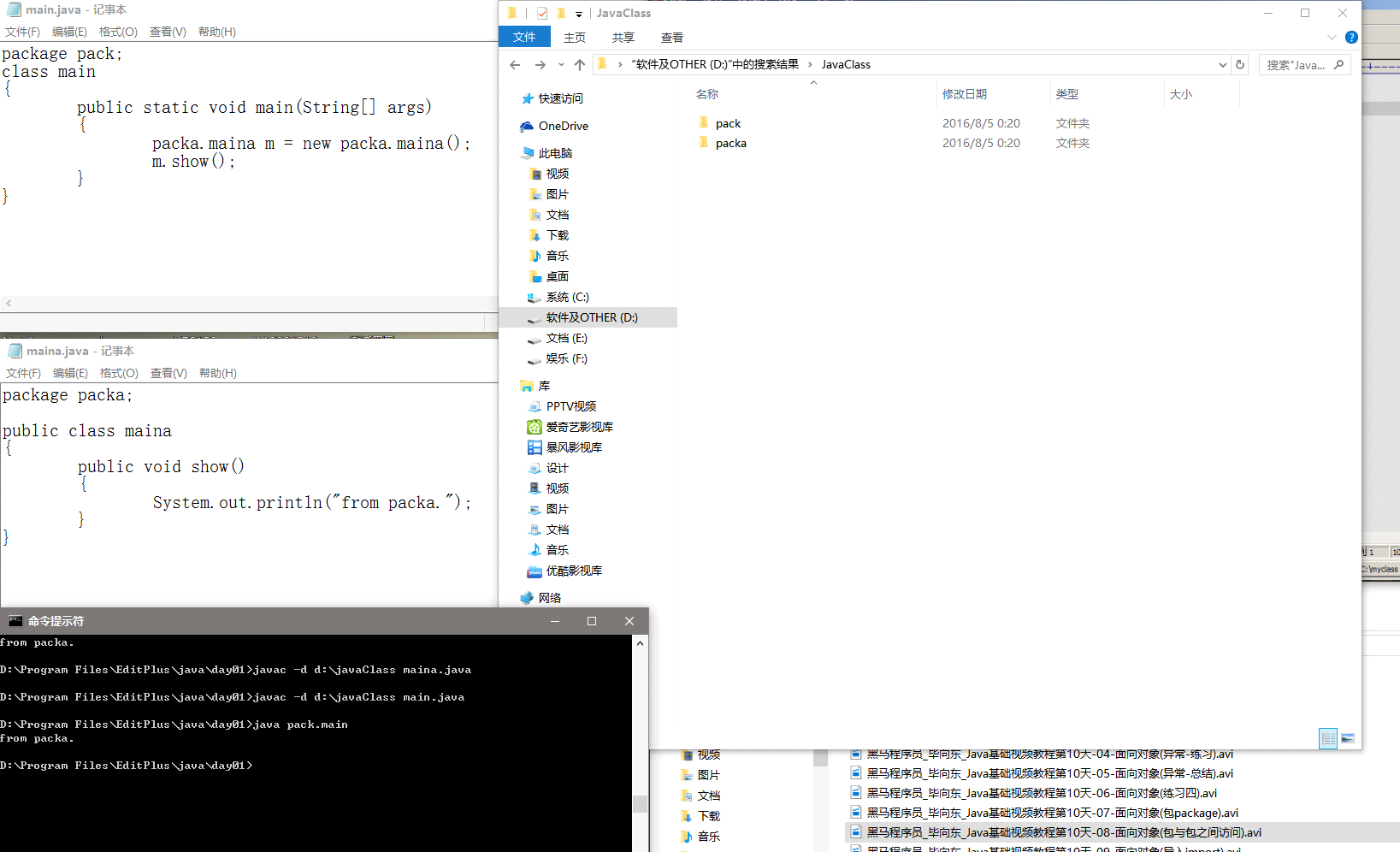
**5.包也是一种封装形式**

***2.包与包之间的访问***

**有了包，范围变大，一个包中的类要被访问，必须有足够大的权限，被访问的类要被public修饰**

**类公有，被访问的成员也要公有。**

**包也是一种封装**

****

***3.总结***

**包与包之间访问，被访问的包中的类以及类中的成员，需要被public修饰**

**不同包中的子类可以访问父类中被protected修饰的成员**

**4.权限从大到小：public protected default(默认权限) private**

**同一类中： 1 1 1 1**

**同一包中： 1 1 1 0**

**子类 ： 1 1 0 0**

**不同包中： 1 0 0 0**

## 5. import（简化类名的书写）

**1.import pack.b.demo.class //导入包pack.b下的类demo**

**2.import pack.b.\* //导入包pack.b下的所有类（不建议使用）**

**3.导入不同包中有同名类时必须用包名区分开**

**4.建议定义包名不重复，使用URL完成定义**

## 6. jar包（Java的压缩包）

**1.cmd命令：D:\JavaClass>jar -cf javaJar.jar pack packa**

**cmd输入jar课查看命令提示**

# 十一. 多线程

## 1. 多线程

**1.进程：正在进行中的程序，每一个进程执行都有一个执行顺序，该顺序是一个执行路 径，（或者叫一个控制单元）**

**2.线程：进程中的独立的控制单元，线程控制着进程的执行。**

**3.一个进程中至少有一个线程。**

**4.JVM启动时，除主线程外至少还有垃圾回收的线程**

## 2. 创建线程

**创建线程的第一种方式：**

**1.使类继承Thread类**

**2.复写Thread中的run方法**

**3.调用start方法（作用）：**

**(1).调用run方法以启动线程**

**(2).启动线程**

**若直接调用run方法，作用同于调用普通对象方法，线程并没创建。**

**class main**

**{**

**public static void main(String[] args)**

**{**

**Demo d = new Demo(); //new一个对象即创建了一个线程**

**d.start(); //调用start方法**

**}**

**}**

**class Demo extends Thread**

**{**

**public void run()**

**{**

**System.out.println("Thread run");**

**}**

**}**

**//执行多次发现远行结果每次都不一样，谁抢到cpu谁就远行。**

**//明确，在同一时刻，一个cpu只能运行一个程序（多核除外）**

**//多线程的随机性，谁抢到谁执行，cpu不断切换，目前执行多长时间由cpu控制**

**为什么覆盖run()方法？**

**创建线程的目的用于同时运行一些代码，run方法存储线程要运行的代码。（主线程由同理为main方法）**

**创建线程的第二种方法**

**定义类实现Runnable接口**

**覆盖Runnable接口中的run方法**

**将Runnable接口的子类对象作为实际参数传递给Thread类的构造器**

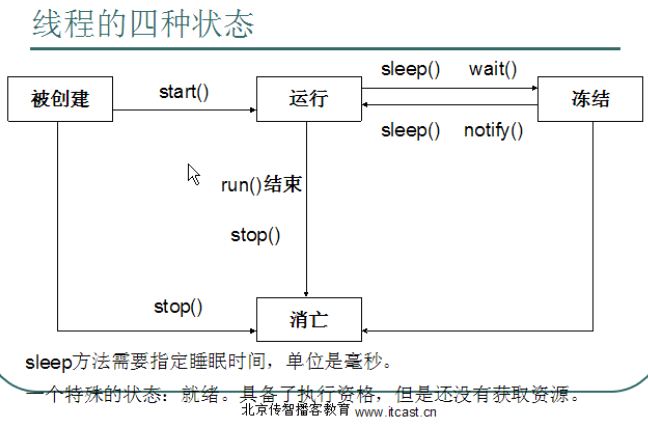
**调用Thread类的start方法开启线程以调用Runnable接口的子类的run方法**

**该实现方式与继承方式的区别：避免java单继承的局限性,建议使用。**

**3.两种方式的区别：  
继承Thread：线程代码存放在Thread的run方法中**

**实现Runnable：线程代码存放在实现Runnable接口的子类的run方法中**

## 4. 线程的多种状态

****

**除图中状态之外还有一个临时状态（阻塞状态）——具备运行资格但没有执行权（CPU忙于执行其他进程）**

**冻结时进程自主暂停运行，暂时放弃使用（不使用）CPU。**

**1.线程都有默认名称Thread-编号（0，1，.................）**

**也可创建时构造方法Thread(String name)，或setName(String name)方法**

## 5. 多线程的安全问题

**例子：当多条语句在操作同一个线程共享数据时，一个线程对多条语句只执行了一部分，这时另一个线程参与进来，导致共享数据出错**

**解决办法：对多条操作共享数据的语句，只能让一个线程都执行完。再执行过程中其他线程不可以打断并参与操作此线程代码。**

**import java.lang.\*;**

**public class main**

**{**

**public static void main(String[] args)**

**{**

**CustomThread ct = new CustomThread();**

**Thread t1 = new Thread(ct);**

**Thread t2 = new Thread(ct);**

**Thread t3 = new Thread(ct);**

**Thread t4 = new Thread(ct);**

**Thread t5 = new Thread(ct);**

**t1.start();**

**t2.start();**

**t3.start();**

**t4.start();**

**t5.start();**

**}**

**}**

**class CustomThread implements Runnable**

**{**

**private int num = 100;**

**public void run()**

**{**

**//System.out.println("a new Thread is run.");**

**while ( true)**

**{**

**if(num > 0)**

**try**

**{**

**//此处用于模拟出错，假设线程t1线程运行到这（CPU）离开处理t2线程，此时t2改变了num的值，而t1，再次唤醒时直接在此处开始此时num的值已经改变了，而不是t1sleep前的值了**

**Thread.sleep(20);**

**}**

**catch (InterruptedException e)**

**{**

**}**

**System.out.println(Thread.currentThread().getName()+"......num = "+num--);**

**}**

**}**

**}**

**}**

## 6. 安全问题的解决办法——同步（锁）

**java的专业解决方式：同步代码块**

**synchronized(对象) //—— 锁**

**{}**

**同步的前提：**

**1.必须要有两个及以上线程**

**2.必须两个及以上线程会操作同一个锁内的代码**

**同步的弊端：线程要判断锁的状态，消耗资源，慢。**

**当某一个线程进入，若锁开启状态则进入并置锁关闭，期间其他线程进入前得知锁关闭，无法进入，所内代码执行完该线程置锁开启，此时其他线程可进入。类似（火车上的卫生间）**

**private Object obj = new Object();**

**while (true)**

**{**

**synchronized(obj )//**

**{**

**if (num > 0)**

**{**

**//try**

**//{Thread.sleep(20);}catch (InterruptedException e){} System.out.println(Thread.currentThread().getName()+"......num = "+num--);**

**}**

**}**

**}**

**synchronized(任意对象){} 可以用在任何地方 ---------同步代码块**

**或直接作为修饰符加在任何方法上：--------------------同步函数**

**public synchronized void add() synchronized(this) 任意对象为this**

**{}**

**如果同步函数被static修饰，它使用的锁不再是this（静态进内存时还没有对象）**

**静态进内存，内存中没有本类对象，但一定有该类对应的字节码文件对象：类名.class，该对象类型为Class synchronized(类名.class)**

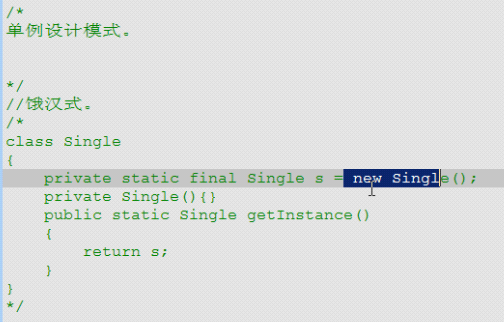
**怎么判断代码是否要同步：**

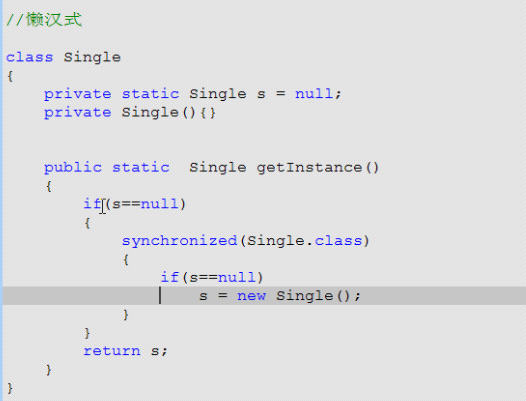
**1.明确哪些代码是多线程运行代码**

**2.明确共享数据**

**3.明确多线程运行代码中哪些代码操作共享数据，同步那些代码即可。**

## 7. 单例设计模型中的同步--\*\*

****

****

**懒汉式（相较饿汉式）特点：**

**延时加载因而导致其被多个线程使用时不安全，可以用加同步锁（同步代码块/同步函数）的方式去除安全隐患，但比较低效，因而采用同步代码块加双重判断的方式减少判断次数解决低效问题（当有一个线程执行并创建了一个实例，他之后的线程就不会再次判断锁了），锁加载静态方法中因而使用的锁时该类所属的字节码对象。**

## 8. 死锁--导致程序停止运行

**同步中嵌套同步，而两个同步的锁不同**

****

**flag为boolean**

**main中代码为：  
Thread loa = new Thread(new Test(true));**

**Thread lob = new Thread(new Test(false));**

**loa.start();**

**lob.start();**

**试图打开和所持有锁（钥匙）不同的锁----打不开---死锁**

# 十二. 线程间通讯

## 1. 线程间通讯

**就是多个线程操作同一资源，只是操作方式不同。**

**生产者和消费者的例子**

**import java.lang.\*;**

**public class main**

**{**

**public static void main(String[] args)**

**{**

**Res r = new Res();**

**Input in = new Input(r);**

**Output out = new Output(r);**

**Thread ra = new Thread(in);**

**Thread rb = new Thread(out);**

**ra.start();**

**rb.start();**

**}**

**}**

**class Res**

**{**

**String name;**

**String sex;**

**boolean flag = false;**

**}**

**class Input implements Runnable**

**{**

**private Res res;**

**Input(Res r)**

**{**

**this.res = r;**

**}**

**public void run()**

**{**

**int x=0;**

**while (true)**

**{**

**synchronized(res)**

**{**

**if (res.flag)**

**{**

**try**

**{**

**res.wait();**

**}**

**catch (InterruptedException e)**

**{**

**}**

**}**

**if (x==0)**

**{**

**res.name = "mike";**

**res.sex = "man";**

**}**

**else**

**{**

**res.name = "丽丽";**

**res.sex = "女";**

**}**

**x = (x+1)%2;**

**res.flag = true;**

**res.notify();**

**}**

**}**

**}**

**}**

**class Output implements Runnable**

**{**

**private Res res;**

**Output(Res r)**

**{**

**this.res = r;**

**}**

**public void run()**

**{**

**while (true)**

**{**

**synchronized(res)**

**{**

**if(!res.flag)**

**{**

**try**

**{**

**res.wait();**

**}**

**catch (InterruptedException e)**

**{**

**}**

**}**

**System.out.println(res.name+"...."+res.sex);**

**res.flag = false;**

**res.notify();**

**}**

**}**

**}**

**}**

****

**notif() notifAll()都只能使用在同步中？**

**同步才有锁的概念，因为要对持有监视器（锁）的线程操作**

**为什么wait() notif()......等都定义在Object中？**

**因为这些方法在操作同步中线程时，都必须要标识他们所操作线程所持有的锁，只有同一个锁上的被等待线程，才可以被同一个锁上的notify唤醒。不可以对不同锁中的线程进行唤醒。也就是说，等待和唤醒必须是同一个锁。**

## 2. 代码优化后

**import java.lang.\*;**

**public class main**

**{**

**public static void main(String[] args)**

**{**

**Res r = new Res();**

**new Thread( new Input(r) ).start();**

**new Thread( new Output(r) ).start();**

**}**

**}**

**class Res**

**{**

**private String name;**

**private String sex;**

**private boolean flag = false;**

**public synchronized void set(String name,String sex)**

**{**

**if (flag)**

**{**

**try**

**{**

**this.wait();**

**}**

**catch (InterruptedException e)**

**{**

**}**

**}**

**this.name = name;**

**this.sex = sex;**

**this.notify();**

**this.flag = true;**

**}**

**public synchronized void out()**

**{**

**if(!flag)**

**{**

**try**

**{**

**this.wait();**

**}**

**catch (InterruptedException e)**

**{**

**}**

**}**

**System.out.println(name+"...."+sex);**

**this.flag = false;**

**this.notify();**

**}**

**}**

**class Input implements Runnable**

**{**

**private Res res;**

**Input(Res r)**

**{**

**this.res = r;**

**}**

**public void run()**

**{**

**int x=0;**

**while (true)**

**{**

**if (x==0)**

**{**

**res.set("make","man");**

**}**

**else**

**{**

**res.set("丽丽","女女女女女女");**

**}**

**x = (x+1)%2;**

**}**

**}**

**}**

**class Output implements Runnable**

**{**

**private Res res;**

**Output(Res r)**

**{**

**this.res = r;**

**}**

**public void run()**

**{**

**while (true)**

**{**

**res.out();**

**}**

**}**

**}**

## 3. 同一资源被多个线程共同操作时

**class Resource**

**{**

**private String name;**

**private int count = 1;**

**private boolean flag = false;**

**public synchronized void set(String name)**

**{**

**//当有多个（不小于两个）线程操作同步代码块（同步函数）时要用while判断标记**

**//因为线程在哪里wait，被notify时就从那里开始往下执行，多线程时就要对此判断标记，notify时默认值唤醒线程池中排第一个的冻结线程**

**while(flag)**

**try{this.wait();}catch(Exception e){}**

**this.name = name+"-------"+count++;**

**System.out.println("..........");**

**flag = true;**

**//多线程时要唤醒所有冻结的线程**

**//防止出现所有线程都冻结（都有执行资格而都放弃执行权）而出现假死锁（死锁是都有执行资格却因为没有持有正确的钥匙而取不到执行权）现象**

**this.notifAll();**

**}**

**//同理**

**public synchronized void out()**

**{**

**while(flag)**

**try{this.wait();}catch(Exception e){}**

**//this.name = name+"-------"+count++;**

**System.out.println(".........."+name);**

**flag = true;**

**this.notifAll();**

**}**

**}**

## 4. JDK1.5及之后（JDK5.0）synchroized被替代

**以下示例（生产者-消费者）中本方只唤醒对方的操作**

**import java.lang.concurrent.locks.\***

**class Resource**

**{**

**private String name;**

**private int count = 1;**

**private boolean flag = false;**

**//替代synchronized的接口Lock实例已知实现Lock接口的类ReentrantLock**

**//以调用其lock和unLock方法**

**private Lock lock = new ReentrantLock();**

**//Condition 将 Object 监视器方法（wait、notify 和 notifyAll）分解成截然不同的对象，**

**//以便通过将这些对象与任意 Lock 实现组合使用，为每个对象提供多个等待 set（wait-set）。**

**//其中，Lock 替代了 synchronized 方法和语句的使用，Condition 替代了 Object 监视器方法**

**//的使用。**

**private Condition condition\_pro = lock.newConition();**

**private Condition condition\_con = lock.newConition();**

**public synchronized void set(String name) throws InterruptedException**

**{**

**lock.lock(); //锁上**

**try**

**{**

**while(flag)**

**condition\_pro .await();**

**this.name = name+"-------"+count++;**

**System.out.println("..........");**

**flag = true;**

**condition\_con .signal();**

**}**

**finally**

**{**

**lock.unLock(); //一定要解锁**

**}**

**}**

**public void out()**

**{**

**lock.lock();**

**try**

**{**

**while(flag)**

**condition\_con .await(); //这里会抛异常，不在这catch在方法上抛出，需要在调用处catch处理**

**System.out.println(".........."+name);**

**flag = true;**

**condition\_pro .signal();**

**}**

**finally**

**{**

**lock.unLock();**

**}**

**}**

**}**

**5.停止线程**

**stop方法（不管线程处于什么状态下，直接杀死该线程）已经过时，**

**停止线程只有一种方法，结束run方法。开启多线程运行，运行代码通常都是循环结构，只要控制住循环，就可以让run方法结束，即结束线程。**

**当没有指定的方式让冻结的线程恢复到运行状态时，这时需要强制对冻结状态进行清除，就强制让线程恢复到运行状态，但此时会抛出中断异常，此时就可以在捕获该异常处修改标记，以结束线程。**

**Thread类提供了方法interrupt强制中断线程的冻结状态恢复到运行状态同时抛出中断异常，**

**5.其他部分方法**

**setDaemon(boolean on)**

**该方法必须在启动线程前调用，即在调用start方法前调用。线程设置为守护线程后，当启动守护线程的主线程结束时，其守护线程也都结束。**

**join()**

**跟主线程申请CPU执行权，此时主线程冻结，使用join的线程结束时主线程被唤醒。**

**t1.start();**

**t1.join();//主线程冻结，此时只有t1线程在运行**

**t2.start();**

**----------------------**

**t1.start();**

**t2.start();**

**t1.join();//此时主线程冻结，t1,t2同时运行，当t2结束时主线程不会被唤醒，只有t1结束时主线程才会被唤醒。**

**setPriority(int);**

**设置线程优先级；**

**优先级等级共有10等， 1~10**

**其中1、5、10跨度最大最明显，因而有对应的静态int值，MAX\_PRIORITY、MIN\_PRIORITY、NORM\_PRIORITY。**

**yield();**

**临时释放执行权。减缓线程占用CPU的频率。可以借之时多个线程平均运行。**

# 十三. String类

## 1. String类（final class String）

### 1.1 概述

**String s1 = "abc";**

**String s2 = new String("abc");**

**String s3 = "abc";**

**System.out.println(s1 == s2); //false**

**System.out.println(s1 == s3); //true**

**一旦被初始化就不可以被改变**

**String.equals()：String复写了Object类中的equals方法：该方法用于判断字符串是否相 同.**

**String s1 = "abc"; //内存中只有一个对象**

**String s2 = new String("abc"); //内存中有两个对象**

### 1.2用于描述字符串这类事物，常见操作有：

**(1) 获取**

**int length(); //获取字符串长度**

**char charAt(int index); //根据位置获取位置上的字符(没有找到返回-1)**

**int indexOf(int ch)://根据字符获取该字符在字符串中的位置(ch为ASCAL码)**

**int indexOf(int ch,int fromIndex); //从指定位置开始查找**

**int indexOf(String str);**

**int lastIndexOf(String str); //从后往前找**

**....................**

**(2) 判断**

**字符串中是否包含给定子串**

**boolean contains(String str); //功能有时可以用indexOf()替代**

**字符串中是否有内容**

**boolean isEmpty(); //.length()==0时返回true**

**字符串是否以指定内容开头**

**boolean startsWith(String str);**

**字符串是否以指定内容结尾**

**boolean endsWith(String str);**

**字符串内容是否相同**

**boolean equals(String str);**

**字符串内容是否相同，并忽略大小写**

**boolean equalsIgnoreCase(String str);**

**(3)转换**

**将字符数组转换为字符串**

**构造函数：**

**String(char[ ]);**

**String(char[ ],offset,count); //一部分字符数组转换为字符串offset开始取count个**

**静态方法：**

**static String copyValueOf(char[ ]);**

**static String copyValueOf(char[ ] data,int offset,int count);**

**static String valueOf(char[ ]);**

**将字符串转换为字符数组**

**char[ ] toCharArray();**

**将字节数组装换为字符串**

**String(byte[ ]);**

**String(byte[ ],offset,count);**

**将字符串转换为字节数组**

**byte[ ] getBytes(); //可以指定编码表**

**将基本数据类型转换为字符串**

**static String valueOf(int a);**

**static String valueOf(double a);**

**(4)替换**

**替换单个字符**

**String replace(char old,char new); //如果替换的不存在则返回原串**

**替换字符串**

**String replace(String old,String new);**

**(5)切割**

**String[] split(String regex); //regex用于切割的字符串(regex被舍弃)**

**(6)获取**

**获取字符串中的一部分**

**String subString(int begin); //包括begin处**

**String subString(int begin,int end); //包含begin不包含end**

**(7)转换、去除空格，比较**

**将字符串转成大写或小写**

**String toUperCase();**

**String toLowerCase();**

**将字符串两端的多个空格去除**

**String trim();**

**对两个字符串进行自然顺序比较**

**int compareTo(String str);**

## 2. StringBuffer类

**(1)存储**

**StringBuffer append();//将指定数据添加到已有数据结尾处**

**StringBuffer insert(index 数据); //将数据插入到指定位置**

**(2)删除**

**StringBuffer delete(int start,int end);//包含start不包含end**

**StringBuffer deleteChar At(int index);**

**(3)获取**

**.....................................**

**(4)修改**

**StringBuffer replace(int start,int end,String str);**

**void setCharAt(int index,char c);**

**(5)反转**

**StringBuffer reverse();**

**(6)**

**将缓冲区中指定数据存储到指定字符数组中**

**void getChars(int srcBegin,int srcEnd,char[] ast,int dstBegin);**

## 3. StringBuilder类

**StringBuffer是线程同步**

**StringBuilder是线程不同步**

**开发建议使用StringBuilder**

**升级三要素：**

**提高效率**

**简化书写**

**提高安全性**

## 4. 基本数据类型对象封装类

**(通常用于基本数据类型与字符串间的转换)**

**1.基本数据类型 引用数据类型（类）**

**byte Byte**

**short short**

**int Integer**

**long Long**

**boolean Boolean**

**float Float**

**double Double**

**char Character**

**2.基本数据 ——> 字符串：**

**基本+""**

**Integer.toString(34); //把34变为"34"**

**3.字符串——> 基本数据 ：**

**static int parseInt("123");//返回值为123；**

**...............**

**static boolean parseBoolean("true");//true**

**4.进制转换**

**//将十进制转换为二进制**

**String str = Integer.toBinaryString(21);**

**//str是radix指定的进制，转换为二进制**

**static int = parseInt(String str,int radix);**

**5.JDK之后（自动装箱，注意为nullshi出现空指针异常）：  
Integer x = new Integer(4);**

**等价于Integer x = 4; //自动装箱**

**x = x+2; //自动拆箱（x变为int型）**

**Integer m = 128;**

**Integer n = 128;**

**判断 m == n 结果为false**

**Integer a = 127;**

**Integer b = 127;**

**判断a == b 结果为true**

**//原因：当数值在byte范围（-128 —— +127）内时，对于新特性，如果数值已经存在，则不会再开辟新空间**

# 十四. 集合框架

## 1. 集合框架

**（集合类、容器）interface Collection**

**1.集合是存储对象的一种最常用方式**

**数组长度固定，集合长度可变，**

**数组内只能存储相同基本数据类型，集合内可存储不同对象类型**

## 2. Collection超类接口

**import java.util.\*;**

**class main**

**{**

**public static void main(String[] args)**

**{**

**//创建集合容器，使用Collection的子类**

**ArrayList a = new ArrayList();**

**a.add("java01");**

**a.add("java02");**

**a.add(25); //集合对象在堆中存储的都为对象的引用（地址值）**

**System.out.println(a.size()); //结果为3**

**}**

**}**

**boolean add(Object o); //添加**

**boolean remove(Object o);//移除**

**boolean clear();//清空集合**

**boolean contains(Object o);//是否包含某个对象**

**int size(); //个数**

**boolean isEmpty();//判断size是否为0**

**arrayList01.retainAll(arrayList02);//只取两者交集并把交集存于arraList01中**

**arrayList01.removeAll(arrayList02);//在arrayList01中去除与arrayList02相同的并存剩下的于arraList01中**

## 3. interface Iterator;迭代器

**用于取出集合内元素**

**私有内部类(内部事物)对象的获取**

**interface myClass**

**{**

**int num(int a)；**

**}**

**public class main**

**{**

**public myClass getMyClass() //用于获取内部类对象**

**{**

**return mc();**

**}**

**private class mc implements myClass //私有内部类**

**{**

**public int num(int a)**

**{**

**return 13;**

**}**

**}**

**}**

**Iterator it = a.iterator(); //全局变量**

**while(it.hasNext)**

**{**

**System.out.println(it.next());**

**}**

**/\**

**| 下面for循环写法更节省内存**

**\/**

**for(Iterator it = a.iterator() ; it.hasNext() ; ) //局部变量**

**{**

**System.out.println(it.next());**

**}**

## 4. collection子接口List

**Collcetion**

**\* |-List 元素是有序的，元素可以重复，因为该体系有索引**

**|-ArrayList JDK1.2 底层数据结构为数组结构，查询速度快，增删较慢。线 程不同步**

**|-LinkedList 底层使用链表结构，查询慢，增删快。**

**~~|-Vector JDK1.0 底层为数组数据结构（同ArrayList一模一样）被ArrayList替 代，接口 Enumeration （枚举），Vector特有的取出方式，枚举用法同于Interator（因单词过长而被Intertor替代>-<）。~~**

**|-Set 元素是无序的，元素不可以重复**

### 4.1 List特有方法：

**增：**

**add(int index,E element);**

**addAll(int index,Collection c);**

**删：**

**remove(int index);**

**改：**

**set(int index,E element)  
查：**

**get(int index);**

**subList(int from,int to);//返回指定区间的元素集合，含头不含尾。**

**listInterator();**

**\*List集合特有迭代器listIterator是Iterator的子接口**

**用迭代器时只能用Iterator迭代器定义的方法操作元素，如果有其他操作时若使用list的方法会抛出并发操作异常，因而使用定义了更多方法的listIterator的子接口对象。**

### 4.2 LinkedList 特有方法：

**//添加元素，在JDK1.6出现替代方法：offerFirst(),offerLast()**

**addFIrst();**

**addLast();**

**//只获取元素，在JDK1.6出现替代方法：peekFirst(),peekLast()**

**getFIrst();**

**getLast();**

**//获取元素后删除元素，在JDK1.6出现替代方法：pollFirst(),pollLast()**

**removeFirst();**

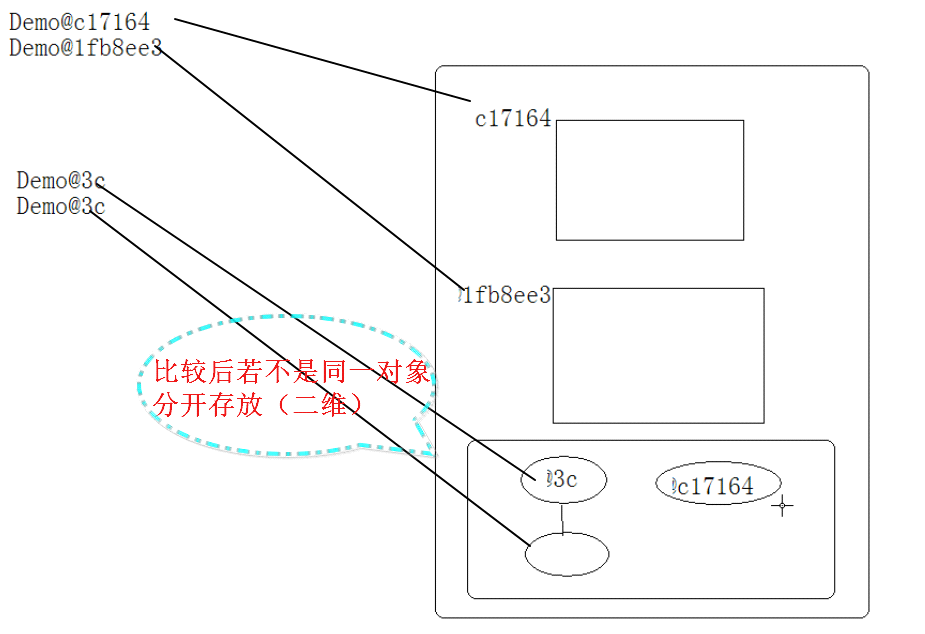
**removeLast();**

## 5. Collection子接口Set

**元素是无序（存入与取出顺序不一定一致）的，元素不可以重复**

**Set定义方法与Collection一致**

**哈希表，表中按地址值（哈希值）顺序存和取。当两对象哈希值相同时再判断该两对象是否为同一对象，不是就纵向延伸挂在下面，是则不再新增哈希值。（哈希值的生成可复写HashCode，比较可复写equals）**

****

**|--Set**

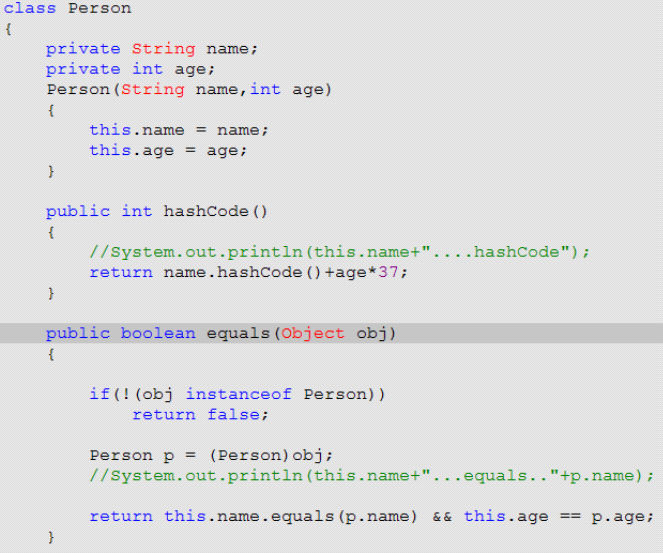
**|--HashSet 底层数据结构是哈希表**

**|--TreeSet**

### 5.1 HashSet

**HashSet是如何保证元素的唯一性？**

**答：底层自动通过调用HashCode()方法比较若一样，再调用equals()方法比较对象内容是否一致。**

****

### 5.2 TreeSet

**1.Set**

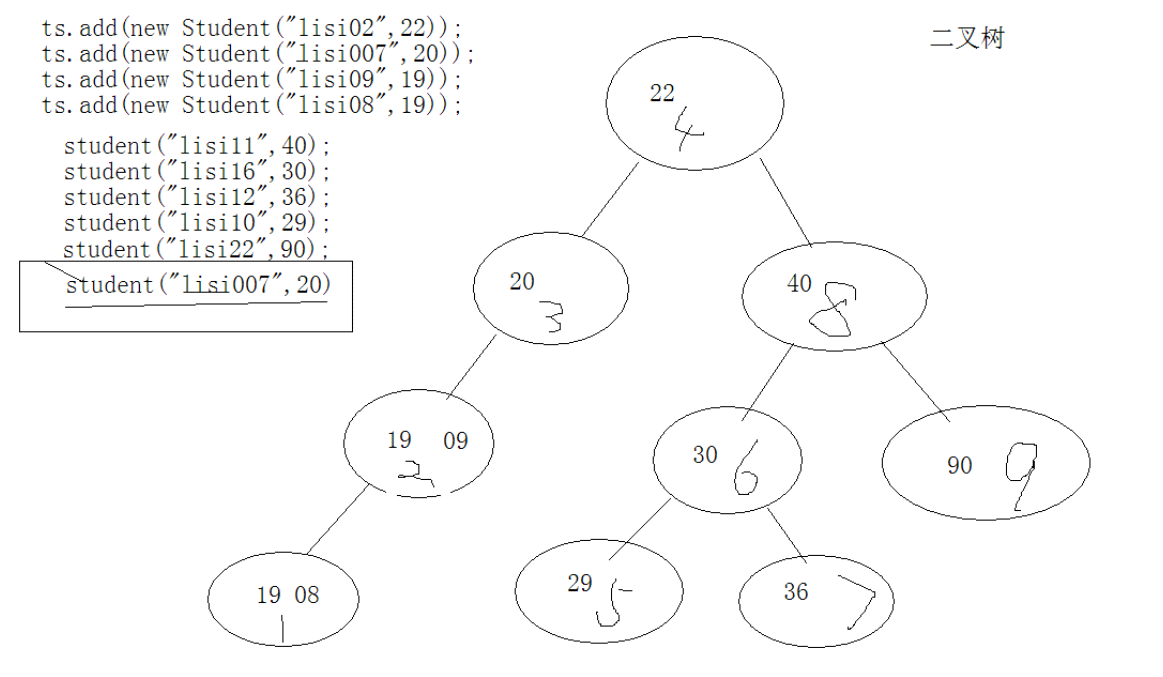
**|--TreeSet**

**可以对Set中的元素进行排序：**

1. **TreeSet的第一种排序方式：**

**让元素自身具备比较性，依靠调用接口Comparable并实现compareTo()方法进行比较，可复写（当主要比较条件相等时比较次要条件）。**

**底层数据结构为二叉树**

****

1. **TreeSet的第二种排序方式：**

**元素自身不具备比较性，或比较性不是所需的，这是让集合自身具备比较性。在集合初始化时就有了比较方式。（定义一个比较器，将比较器作为参数传递给集合）**

**class MyCompare implements Comparator**

**{**

**public int compare(Object o1,Object o2)**

**{**

**}**

**}**

**当两种排序都存在时，优先使用自定义的比较器。**

## 6. 泛型

格式：通过<Object>来定义操作的引用数据类型，在集合中很常见

**JDK1.5之后出现的新特性，是一个安全机制，将远行时期的ClassCastException异常转移到编译时期、避免了强制转换的麻烦。**

### 6.1 何时定义泛型类？

**当类要操作的引用数据类型不确定时，以前用Object 完成扩展，现在用泛型完成。**

**import java.util.\*;**

**class main**

**{**

**public static void main(String[] args)**

**{**

**Utils<Worker> ut = new Utils();**

**ut.setObject(new Worker("LiLI"));**

**System.out.println(ut.getObject().name);**

**}**

**}**

**class Worker**

**{**

**public String name;**

**public Worker(String name)**

**{**

**this.name = name;**

**}**

**}**

**//自定义的泛型类**

**class Utils<UT>**

**{**

**private UT ut;**

**public void setObject(UT ut)**

**{**

**this.ut = ut;**

**}**

**public UT getObject()**

**{**

**return ut;**

**}**

**}**

### 6.2 何时定义泛型方法？

**泛型类定义在类上时，类内所有使用了该泛型的所对应的数据类型就固了，为了让同一方法可以操作不同类型，可以将泛型定义在方法上。**

**class Demo**

**{**

**// <>放在返回值类型前面**

**public <T> void show(T t)**

**{**

**}**

**}**

**静态方法不可以访问类上定义的泛型，**

### 6.3 泛型接口

**interface Inter<T>**

**{**

**void show(T t);**

**}**

**class InterImpl<T> implements Inter<T>**

**{**

**vdoi show(T t)**

**{**

**...............**

**}**

**}**

**mian()**

**{**

**InterImpl<String> i = new InterImpl<String>();**

**i.show("hah");**

**}**

### 6.4 <?>占位符

**不能使用具体类的特有方法**

**public static void printColl(ArrayList<?> al)**

**{**

**Iterator<?> it = al.iterator();**

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

**{**

**System.out.println( it.next() );**

**}**

**}**

**//泛型限定**

**public static void printColl(ArrayList<? extends Person> al)**

**结：**

**1.？ 通配符。**

**2.泛型的限定。**

**3.？ extends E：可以接收E或者E的子类类型，上限。**

**4.？ super E：可以接收E或者E的父类类型，下限。**

# 十五. (十六) Collection子接口Map

## Map集合

**Map ---**

**|--Hashtable：底层为哈希表数据结构，不可存null键null值，线程同步**

**|--HashMap：底层为哈希表数据结构，可存null键null值，线程不同步，效率高**

**|--TreeMap：二叉树数据结构，线程不同步，可以给集合中的键排序。**

**Set底层就是使用了Map集合**

**该集合存储键对值，一对一对往里存，键是唯一的。**

## 2. 方法

**添加：**

**V put(K key,V value);//当存了相同键的值时，后添加的会覆盖原有键对应的值，并返回被覆盖的值。**

**void putAll(Map<? extends K>,? extends V> m);**

**判断：**

**boolean containsValue(Object value)**

**boolean containsKey(Object key)**

**boolean isEmpty()**

**删除：**

**clear()**

**reomve(Object key)**

**获取：**

**V get(Object k);**

**int size();**

**Set<Map.Entry<K,V>> entrySet() //将map集合中的映射关系取出，存入到set集合中**

**Set<Map.Entry<K,V>> entrySet = map.entrySet();**

**Iterator<Map.Entry<K,V>> it = entrySet.iterator();**

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

**{**

**Map.Entry<K,V> me = it.next();**

**K key = me.getKey();**

**V value = me.getValue();**

**}**

**Map.Entry Entry是Map接口中的内部接口**

**interface Map**

**{**

**public static interface Entry**

**{}**

**}**

**Set<K> keySet()//获得所有的键到集合**

**Collection<V> values(); //获得所有值到集合 //可配合Iterator使用**

## 3. HashMap

**需要覆写hashCode方法和equals方法,类对象过多时可能需要用二叉树结构存放，因而需要时也应实现comparable<T>接口，并覆写compareTo(T c)方法，使之具有自然顺序。**

**public int hashCode()**

**{**

**return \*\*;**

**}**

**public boolean equals(Object obj)**

**{**

**if( !(obj instanceof \*\*) )**

**throw ClassCastException("类型不匹配");**

**return \*\*;**

**}**

## TreeMap进行排序

### 4.1 用法

**1. 实现Comparable接口：Class Student implements Comparable{}，并覆写int compareTo()方法。**

**2. 定义比较器作为参数实例TreeMap：Class stuComparator implements Comparator<Student>{}，并覆写int compare()方法，按需覆写equals方法。**

**类似于TreeSet的排序实现。**

### 4.2 TreeMap练习，统计一个字符串中各个字母的出现次数

**private static String charCount(String str)**

**{**

**char[] cha = str.toCharArray();**

**TreeMap<Character,Integer> tree = new TreeMap<Character,Integer>();**

**for(int i = 0 ; i < cha.length ; i++)**

**{**

**Integer in = tree.get(cha[i]);**

**if(in == null)**

**tree.put(cha[i],1);**

**else**

**tree.put(cha[i],++in);**

**}**

**return tree.toString();**

**}**

**判断部分可优化为：**

**private static String charCount(String str)**

**{**

**char[] cha = str.toCharArray();**

**TreeMap<Character,Integer> tree = new TreeMap<Character,Integer>();**

**int count = 0;**

**for(int i = 0 ; i < cha.length ; i++)**

**{**

**Integer in = tree.get(cha[i]);**

**if(in != null)**

**count = in;**

**tree.put(cha[i],++count);**

**}**

**return tree.toString();**

**}**

## Map集合扩展知识？

Map集合中可以嵌套Map集合

import java.util.\*;

**public class main**

**{**

**public static void main(String[] args)**

**{**

**HashMap<String,HashMap<String,String>> stu = new HashMap<String,HashMap<String,String>> ();**

**HashMap<String,String> yure = new HashMap<String,String>();**

**HashMap<String,String> jiuye = new HashMap<String,String>();**

**yure.put("01","yure111");**

**yure.put("02","yure222");**

**yure.put("03","yure333");**

**yure.put("04","yure444");**

**jiuye.put("01","jiuye111");**

**jiuye.put("02","jiuye222");**

**jiuye.put("03","jiuye333");**

**jiuye.put("04","jiuye444");**

**stu.put("yure",yure);**

**stu.put("jiuye",jiuye);**

**get(stu,"jiuye");**

**//System.out.println(charCount("adsfasdfgfdsdsafadsfadfadsfdsawerafds"));**

**}**

**private static void get(HashMap hm,String room)**

**{**

**HashMap<String,String> ro = (HashMap<String,String>)hm.get(room);**

**Iterator<String> id = ro.keySet().iterator();**

**Iterator<String> name = ro.values().iterator();**

**while (name.hasNext())**

**{**

**String na = name.next();**

**String i = id.next();**

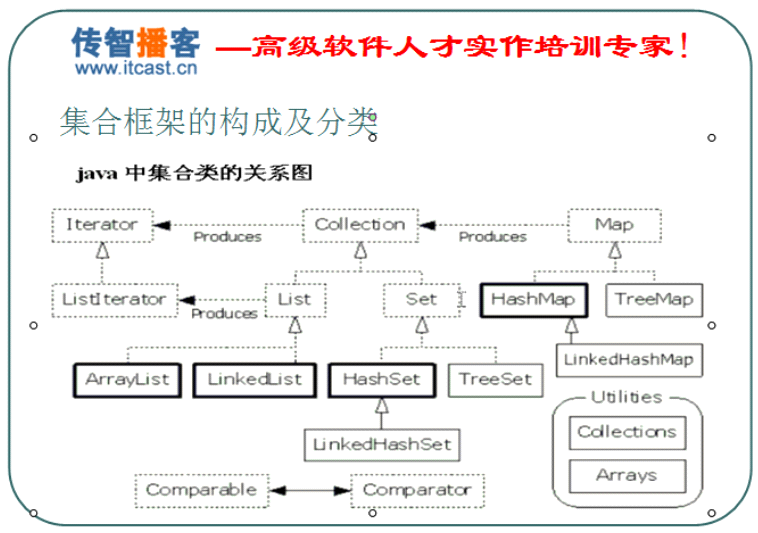
**System.out.println("id="+i+" name="+na);**

**}**

**}**

**}**

## 集合框架的构成及分类

****

# 十七. 集合和数组工具类

## 1. Collections 集合工具类

该类中方法全为静态方法

### 1.1 <T extends Comparable<? super T>> void sort(List<T> list)

泛型T要求要继承于实现了Comparable接口的子类，而Comparable接口的泛型要求必须是T或T的父类

### 1.2 <T> void sort(List<T> list,Comparator<? super T> c)

泛型要求T没有要求，而作为参数传入的比较器（实现了Comparator接口的类）的泛型要求必须是是T或T的父类

**根据指定比较器产生的顺序对指定列表进行排序。 的自然顺序 对指定列表按升序进行排序。**

**import java.util.\*;**

**public class main**

**{**

**public static void main(String[] args)**

**{**

**sortDemo();**

**}**

**private static void sortDemo()**

**{**

**//String类已实现了Comparable接口**

**List<String> list = new ArrayList<String>();**

**list.add("adfa");**

**list.add("8798asdgaewrtqetrewtyreyerwygtui");**

**list.add("glkj;");**

**list.add("l");**

**list.add("wqeru");**

**list.add("4354tr");**

**list.add("8798gtui");**

**list.add("asdfasdgagagag");**

**sop(list);**

**Collections.sort(list,new stringLengthCompare())//自定义比较器;**

**Collections.sort(list）//默认比较方式比较;**

**sop(list);**

**}**

**private static void sop(Object obj)**

**{**

**System.out.println(obj.toString());**

**}**

**}**

**class stringLengthCompare implements Comparator<String>**

**{**

**public int compare(String s,String sto)**

**{**

**if (s.length() > sto.length())**

**return 1;**

**if (s.length() < sto.length())**

**return -1;**

**return s.compareTo(sto);**

**}**

**}**

### 1.3 <T> T max(Collection<? extends T> coll,Comparator<? super T> com)

泛型T没有要求，但传入的集合参数要求其泛型是T或T的子类，传入的比较器参数要求其泛型是T或T的父类

### 1.4 <T extends Object & Comparable<? super T>> T max(Collection<? extends T> coll)

### 1.5<T> int binarySearch(List<? extends Comparable<? super T>> list, T key)

### 1.6 <T> int binarySearch(List<? extends T> list,T key,Comparator<? super T> c)

**使用二分搜索法搜索指定列表，以获得指定对象。在进行此调用之前，必须根据列表元素的自然顺序对列表进行升序排序（通过 sort(List) 方法）。如果没有对列表进行排序，则结果是不确定的。**

**存在返回对应角标，不存在返回负数。**

### 1.7 <T> void fill(List<? super T> list, T obj)

**使用指定元素替换指定列表中的所有元素。**

**如list<String> lis = {“afdfa”,”s”,”w”,”ss”};**

**Lis = Collections.fill(lis,”00”);**

**//则lis变为 {“00”,”00”,”00”,”00”};**

### static <T> boolean replaceAll(List<T> list, T oldVal, T newVal)

**使用另一个值替换列表中出现的所有某一指定值。**

### 1.9static void reverse(List<?> list)

**反转指定列表中元素的顺序。**

### 1.10 static <T> Comparator<T> reverseOrder()

一返回一个比较器，它强行逆转实现了 Comparable 接口的对象 collection 的自然顺序。（自然顺序是通过对象自身的 compareTo 方法强行排序的。）此方法允许使用单个语句，以逆自然顺序对实现了 Comparable 接口的对象 collection（或数组）进行排序（或维护）。例如，假设 a 是一个字符串数组。那么：

Arrays.sort(a, Collections.reverseOrder());

将按照逆字典（字母）顺序对数组进行排序。

因为Arrays.sort(…)方法中的泛型在这里是String，因而Collections.reverseOrder()逆自然顺序对实现了Comparable接口的对象（这里即String）的数组实现逆序。

### 1.11 static <T> Comparator<T> reverseOrder(Comparator<T> cmp)

**返回一个比较器，它强行逆转指定比较器的顺序。**

### 1.12 static <T> Collection<T> synchronizedCollection(Collection<T> c)

**返回指定 collection 支持的同步（线程安全的）collection。**

**………. <T> List<T> synchronizedList(List<T> list)**

**………<K,V> Map<K,V> synchronizedMap(Map<K,V> m)**

**………. <T> Set<T> synchronizedSet(Set<T> s)**

### 1.13 static void swap(List<?> list, int i, int j)

**在指定列表的指定位置处交换元素。**

## Arrays 数组工具类

**此类包含用来操作数组（比如排序和搜索）的各种方法。此类还包含一个允许将数组作为列表来查看的静态工厂。**

**除非特别注明，否则如果指定数组引用为 null，则此类中的方法都会抛出 NullPointerException。**

**二分查找、替换、排序、...................................................**

**参看文档：**

## 3 数组变集合

**public static <T> List<T> asList(T... a)**

**使可以使用集合的方法操作数组**

**注意：**

**1.此时不能使用集合的增删方法**

**2.如果数组中的元素都是对象，那么编变成集合时，数组中的元素就直接转成集合中的元素，而当数组中的元素都是基本数据类型时，那么会将该数组作为集合中的元素存在**

**Integer[] in = new Integer[] {5,6,8,9};**

**int[] it = {5,6,8,9};**

**List<Integer> inl = Arrays.asList(in); //集合中有4个元素：5,6,8,9**

**List<int[]> itl = Arrays.asList(it); //集合中有一个元素：{5,6,8,9}**

**返回一个受指定数组支持的固定大小的列表。（对返回列表的更改会“直接写”到数组。）此方法同 Collection.toArray() 一起，充当了基于数组的 API 与基于 collection 的 API 之间的桥梁。返回的列表是可序列化的，并且实现了 RandomAccess。**

**此方法还提供了一个创建固定长度的列表的便捷方法，该列表被初始化为包含多个元素：**

**List<String> stooges = Arrays.asList("Larry", "Moe", "Curly");**

## 集合变数组

<T> T[] toArray(T[] a)

指定类型数组（参数）要定义多长呢？

当list的size大于传参时给的数组的length时返回list的size大小的数组

当list的size小于传参时给的数组的length时返回传参所给大小，用null填补

返回包含此 collection 中所有元素的数组；返回数组的运行时类型与指定数组的运行时类型相同。如果指定的数组能容纳该 collection，则返回包含此 collection 元素的数组。否则，将分配一个具有指定数组的运行时类型和此 collection 大小的新数组。

为什么集合变数组？

为了限定对集合元素的操作，增和删操作是没有意义的。

## 4. 增强for循环（foreach）

**JDK1.5出现——简化书写**

**格式：  
for(数据类型 变量名：被遍历的集合（Collection）或数组)**

**{**

**}**

**对被遍历对象只有取功能，不能进行其他操作**

## 5. 可变参数

**JDK1.5之后出现**

**其实就是传递数组参数的简写形式，不用new数组再作为实参传递，隐式将参数封装成数组。**

**show("adsfas","sad","s","a");//strings.length() = 4**

**show(); //strings.length() = 0**

**public void show(String... strings)**

**注意：**

**1.public void show(String str，String... strings);//错误的**

**2.方法有多个参数时，可变参数一定要放在参数列表的最后面**

## 6. 静态导入

**import static java.util.Arrays;//导入的是java.util.Arrays中所有的静态成员**

**import java.lang.\*; //导入包里的所有类**

# 十八.其他对象

## System

在 System 类提供的设施中，有标准输入、标准输出和错误输出流；对外部定义的属性和环境变量的访问；加载文件和库的方法；还有快速复制数组的一部分的实用方法，类中方法都是静态

## Runtime

每个 Java 应用程序都有一个 Runtime 类实例，使应用程序能够与其运行的环境相连接。可以通过 getRuntime 方法获取当前运行时。

应用程序不能创建自己的 Runtime 类实例

该类未提供构造器，不能以new方式创建对象，可以通过调用该类的静态方法getRuntime方法获得该类对象。单例设计模式，使一个程序只有一个Runtime实例。

Runtime r = Runtime.getRuntime();

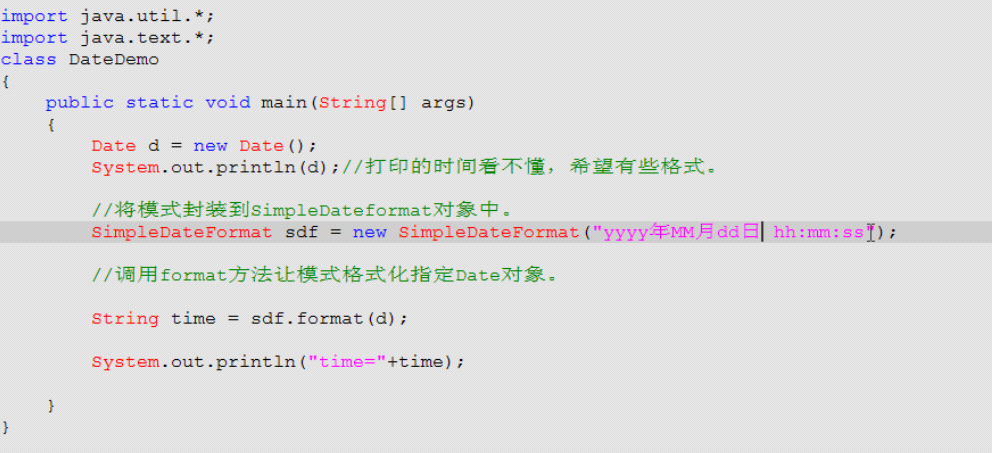
Process p =r.exec("C:\\windows\\notepad.exe"); //可以开启进程执行JVM运行对应平台的命令

Thread.sleep(4000);

p.destroy(); //结束p进程

## Date日期

其大部分方法都已过时被Calendar，DateFormat，取代或完善。

****

SimpleDateFormat 是一个以与语言环境有关的方式来格式化和解析日期的具体类。它允许进行格式化（日期 -> 文本）、解析（文本 -> 日期）和规范化。

## Calendar 日历

**查表法**

**import java.text.\*;**

**import java.util.\*;**

**public class main**

**{**

**public static void main(String[] args)**

**{**

**Calendar c =Calendar.getInstance();**

**String[] strs = {"一月","二月","三月","四月",**

**"五月","六月","七月","八月"**

**,"九月","十月","十一月","十二月"};**

**int index = c.get(Calendar.MONTH);**

**System.out.println("MONTH: "+strs[index]);**

**}**

**}**

## Math

**ceil方法返回大于指定数据的最小整数**

**double d = Math.ceil(12.34);**

**//d = 13；**

**floor方法返回小于指定数据的最大整数**

**double d = Math.floor(12.34);**

**//d=12;**

**四舍五入**

**long l = Math.round(12.34); //l = 12;**

**long f = Math.round(12.54); //f = 13;**

**幂运算**

**double d = Math.pow(2,3); //d = 2\*2\*2 = 8;**

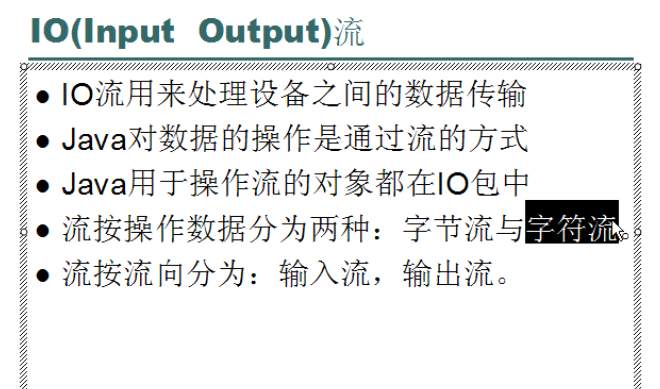
**求真abs()...............................**

**产生随机数x（0.0 =< x < 1.0）的伪随机数（根据算法生成，只有次数足够多就可以找到规律）**

**double d = Math.random();**

**也有Random类可使用**

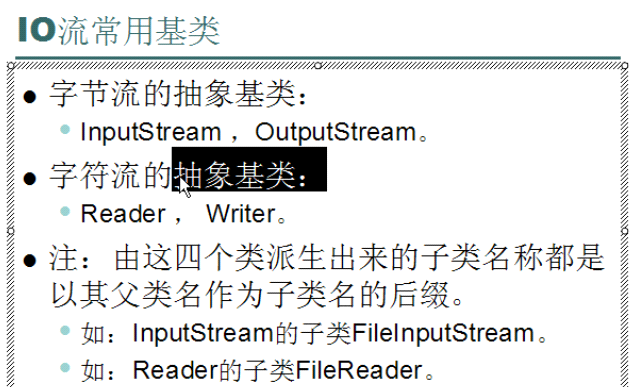
## IO（Input Output）

****

**字符流基于字节流：针对文字编码**

**GBK：（汉字内码扩展规范）GBK 规范收录了 ISO 10646.1 中的全部 CJK 汉字和符号，并有所补充。具体包括：**

**UTF-8：（8-bit Unicode Transformation Format）是一种针对Unicode的可变长度字符编码，又称万国码。由Ken Thompson于1992年创建。现在已经标准化为RFC 3629。UTF-8用1到4个字节编码Unicode字符。用在网页上可以统一页面显示中文简体繁体及其它语言（如英文，日文，韩文）。**

****

## 字符流的两个基类：Writer Reader

**数据的最常见体现形式是文件，以操作文件来演示：**

**找到一个转梦用于操作文件的Writer子类对象**

### Writer

**|——OutputStreamWriter**

**|——FileWriter**

**import java.io.\*;**

**public class main**

**{**

**public static void main(String[] args) throws IOException**

**{**

**//创建FileWriter对象，该对象一被创建就必须明确被操作的文件**

**//该文件将创建到指定目录下（或当前目录下）**

**//如果指定目录已存在同名文件则该文件会被清空**

**FileWriter fw = new FileWriter("text.txt");**

**//write操作将字符串写入流中（内存中）**

**fw.write("dsafdasfa\nasdfsadf");**

**//刷新当前流中数据到目的地**

**fw.flush();**

**//关闭流资源，关闭之前会自动刷新一次内部的缓冲区**

**//将数据刷到目的地**

**//flush刷新时流可以继续使用，close刷新后流关闭**

**fw.close();**

**}**

**}**

**write时的异常处理**

**FileWriter fw = null;**

**try**

**{**

**fw = new FileWriter("text.txt");**

**fw.write("dsafasdf");**

**}**

**catch ( IOException e)**

**{**

**System.out.println( e.toString() );**

**}**

**finally**

**{**

**try**

**{**

**if (fw != null)**

**fw.close();**

**}**

**catch (IOException e)**

**{**

**System.out.println( e.toString() );**

**}**

**}**

**write对已有文件的数据续写**

**public static void main(String[] args) throws IOException**

**{**

**//传递一个true参数，代表不覆盖已有的文件，并在已有文件的末尾处进行数据续写**

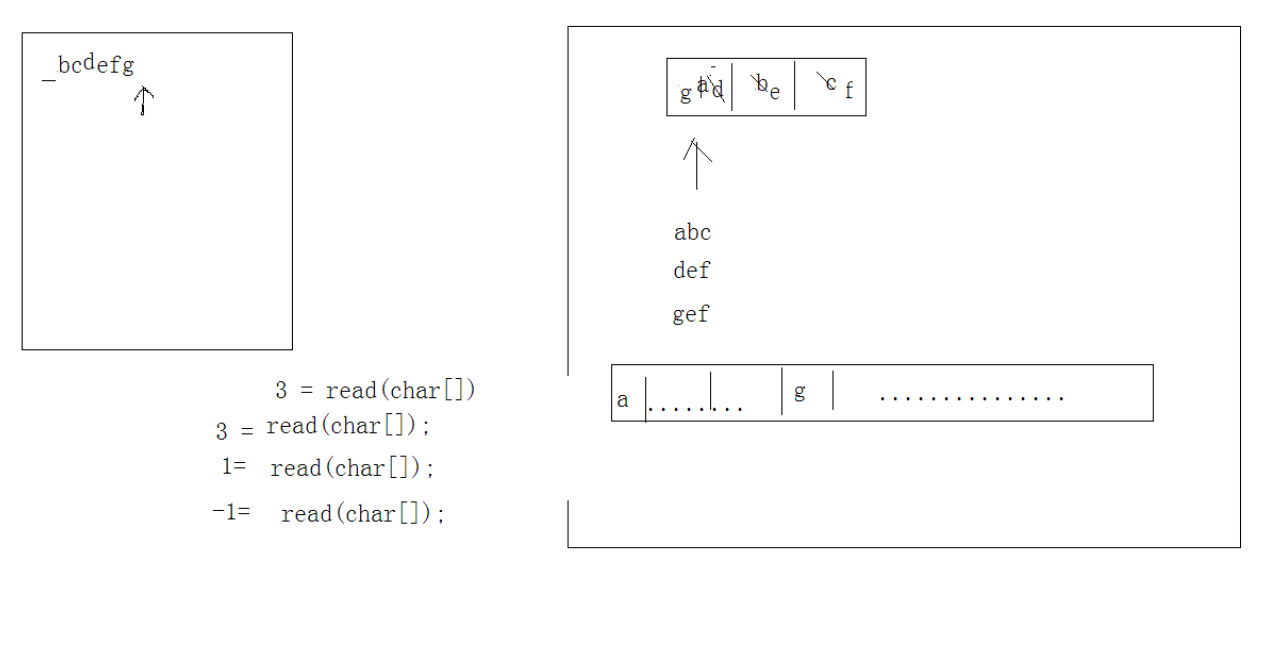
**FileWriter fw = new FileWriter("text.txt",true);**

**//Windows系统下换行为\r\n，linux为\n,否则无法识别**

**fw.write("ewrt...\r\n......sdfwerwer");**

**fw.close();**

**}**

****

### Reader

**|——InputStreamReader**

**|——FileReader**

**一个字符一个字符读**

**//创建一个文件读取流对象，和指定文件名相关联**

**FileReader fr = new FileReader("text.txt");**

**//调用read方法**

**//read方法一次读一个字符而且会自动往下读**

**//读到最后会返回-1**

**/\*int ch = fr.read();**

**int x = fr.read();**

**System.out.println("ch= "+(char)ch+" x= "+(char)x);**

**\*/**

**while (fr.read() != -1)**

**System.out.println((char)fr.read());**

**用char[]一次读取多个字符**

**public static void main(String[] args) throws IOException**

**{**

**FileReader fr = new FileReader("text.txt");**

**//定义一个字符数组，用于存储读到的字符**

**//而read(char[])返回的是读到的字符个数**

**//文件超大时，定义1024的整数倍**

**char[] buf = new char[1024];**

**int num =0;**

**while ( (num = fr.read(buf)) != -1 )**

**{**

**System.out.println("num= "+num);**

**System.out.println(new String(buf,0,num));**

**}**

**fr.close();**

**}**

## 复制文件原理：

在D盘创建一个文件，用于存储C盘文件中的数据

定义读取流和C盘文件关联

通过不断的读写完成数据存储

关闭资源

**private static void copy1()**

**{**

**FileReader fr = null;**

**FileWriter fw = null;**

**try**

**{**

**fr = new FileReader("C:**[**\\Users\\ai\\Desktop\\java\\java**](file:///\\Users\\ai\\Desktop\\java\\java)**笔记\\毕向东java笔记.rtf");**

**fw = new FileWriter("笔记copy.txt",true);**

**char[] buf = new char[1024\*100];**

**int len = 0;**

**while ( (len = fr.read(buf)) != -1 )**

**{**

**fw.write(buf,0,len);**

**System.out.println("copy "+len+" byte ");**

**}**

**}**

**catch (IOException e)**

**{**

**throw new RuntimeException("读写失败");**

**}**

**finally**

**{**

**try**

**{**

**if (fr != null)**

**{**

**fr.close();**

**}**

**}**

**catch (IOException e)**

**{**

**}**

**try**

**{**

**if (fw != null)**

**{**

**fw.close();**

**}**

**}**

**catch (IOException e)**

**{**

**}**

**}**

**}**

# 十九. IO

## 字符流的缓冲区

****

### BufferedWriter

**缓冲区的出现时为了提高流的操作效率，所以再创建缓冲区之前必须现有流对象。**

**该缓冲区中提高一个跨平台的换行操作BufferedWriter.newLine();**

**public static void main(String[] args) throws IOException**

**{**

**//创建一个字符写入流对象**

**FileWriter fr = new FileWriter("buf.txt");**

**//为了提高字符写入流效率，加入缓冲技术**

**//只要将需要提高效率的流对象作为参数传递给缓冲区对象的构造参数即可**

**BufferedWriter bufw = new BufferedWriter(fr);**

**bufw.write("adsfa");**

**//Windows下等同于\r\n，linux下等同于\n**

**bufw.newLine();**

**bufw.write("......");**

**//只要用到缓冲区就要记得刷新**

**bufw.flush();**

**//其实关闭缓冲区就是在关闭缓冲区中的流对象 fr.close()**

**bufw.close();**

**}**

### BufferedReader

**包含一个一次直接读取一行的方法：BufferedRead.readLine()**

**读取一个文本行。通过下列字符之一即可认为某行已终止：换行 ('\n')、回车 ('\r') 或回车后直接跟着换行。**

**返回：**

**包含该行内容的字符串，不包含任何行终止符，如果已到达流末尾，则返回 null**

**抛出：**

**IOException - 如果发生 I/O 错误**

**public static void main(String[] args) throws IOException**

**{**

**//创建一个读取流对象和文件相关联**

**FileReader fr = new FileReader("buf.txt");**

**//缓冲提高读取效率**

**BufferedReader bufr = new BufferedReader(fr);**

**String line = null;**

**while ( (line = bufr.readLine()) != null)**

**{**

**System.out.println(line);**

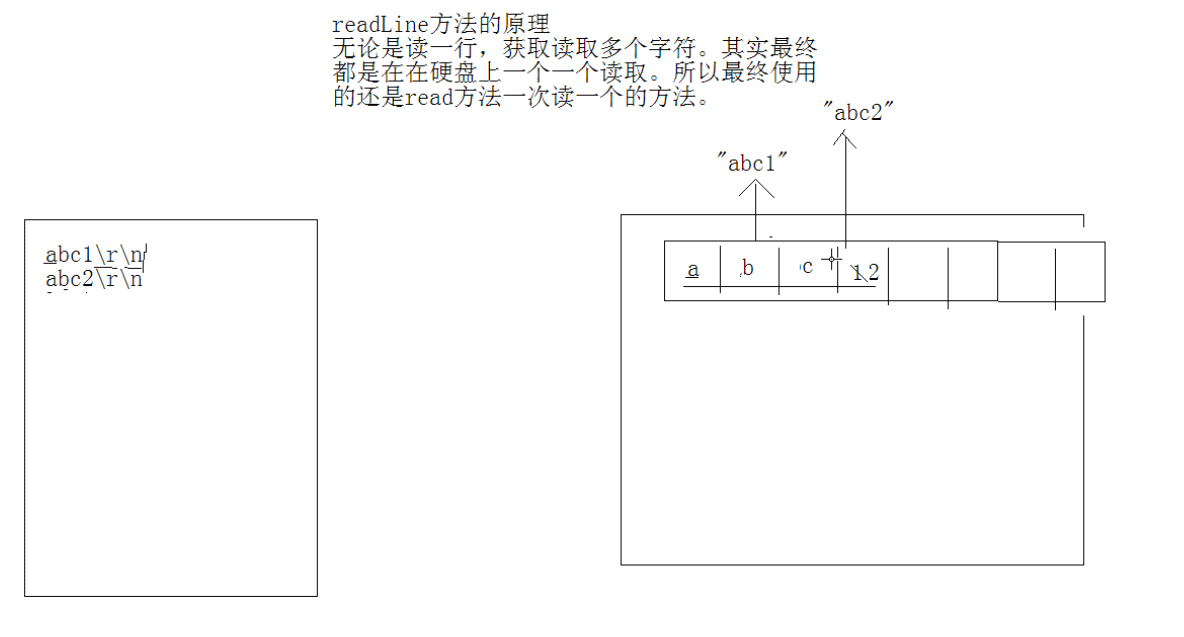
**}**

**//关闭缓冲区（同时自动关闭了读取流）**

**bufr.close();**

**}**

**readLine()返回时只返回回车符之前的数据，不带任何终止符。**

****

**自定义的readLine方法，内部实际调用的是BufferedReader父类Reader的read方法。**

### 3. 装饰设计模式

**当想要对已有的对象进行功能增强时，可以定义类，将已有对象传入，基于已有的功能，并提供加强功能，那么自定义的类称为装饰类。如BufferedReader的readLine()方法。**

**装饰类一般通过构造方法接收被装饰对象，并基于被装饰对象的功能提供更强的功能。**

**装饰模式比继承要灵活，避免了继承体系臃肿，而且降低了类与类间的关系**

**MyReader**

**|——MyTextReader**

**|——MyDataReader**

**|——MyBufferedReader**

**class MyBufferedReader extends MyReader**

**{**

**MybufferedReader(MyReader r)**

**{}**

**}**

## 字节流（操作图片、MP3等文件）

**OutputStream**

**|——FileOutputStream 用于数据写入**

**InputStream**

**|——FIleInputStream 用于数据读出**

**对应缓冲区的使用类似于字符流**

**import java.io.\*;**

**class main2**

**{**

**public static void main(String[] args) throws IOException**

**{**

**FileOutputStream fos = new FileOutputStream("fos.txt");**

**//字节流写时直接写到目的地，不需要刷新操作**

**fos.write("字节流数据".getBytes());**

**FileInputStream fis = new FileInputStream("main2.java");**

**//int ch = 0;**

**//while ( (ch = fis.read()) != -1)**

**//{**

**// System.out.print((char)ch);**

**//}**

**int len = 0;**

**byte[] buf = new byte[1024];**

**while ( ( len = fis.read(buf) ) != -1)**

**{**

**System.out.print(new String(buf,0,len));**

**}**

**//fis.available()获得文件内字节总数，当数据不是很大才能使用否则内存溢出**

**//byte[] by = new byte[fis.available()];**

**//fis.read(by);**

**//System.out.println(new String(by));**

**fos.close();**

**fis.close();**

**}**

**}**

**read方法读时读：一个字节(byte)/次，此时可能出现读到11111111的情况，即-1但并不是结束标志而是有效字节，而之所以返回int类型（四个字节），**

**byte——>int（值不变仍为-1）**

**11111111——>11111111 11111111 11111111 11111111**

**因而返回的int做了如下操作**

**return byte&0xff;**

**即：**

**byte——>int&0xff**

**11111111——>11111111 11111111 11111111 11111111**

**& 00000000 00000000 00000000 11111111**

**——————————————————————**

**int： 00000000 00000000 00000000 11111111**

**值改变了变为255了，因而在写时同样要加些操作：（int——>byte）只取低8位**

**因而有效避免了非结束标志的-1被误认为结束标志。**

## 键盘输入模拟

**当键盘输入一行数据后打印**

**输入over时结束**

**public static void main(String[] args) throws IOException**

**{**

**InputStream in = System.in;**

**StringBuilder sb = new StringBuilder();**

**while (true)**

**{**

**int by = in.read();**

**if (by == '\r')**

**{**

**continue;**

**}**

**if (by == '\n')**

**{**

**String s = sb.toString();**

**if (s.equals("over"))**

**{**

**break;**

**}**

**System.out.println(s.toUpperCase());**

**sb.delete(0,s.length());**

**}**

**else**

**{**

**sb.append((char)by);**

**}**

**}**

**//System.out.println((char)by);**

**}**

## 读取转换流

**public static void main(String[] args) throws IOException**

**{**

**InputStream in = System.in;**

**InputStreamReader isr = new InputStreamReader(in);**

**BufferedReader bufr = new BufferedReader(isr);**

**String line = null;**

**while ( (line = bufr.readLine()) != null )**

**{**

**if ("over".equals(line))**

**{**

**break;**

**}**

**System.out.println(line.toUpperCase());**

**}**

**bufr.close();**

**}**

## 流操作的基本规律

两个明确：

明确源和目的

明确操作的数据是否是纯文本

如：

源：输入流、InputStream、Reader

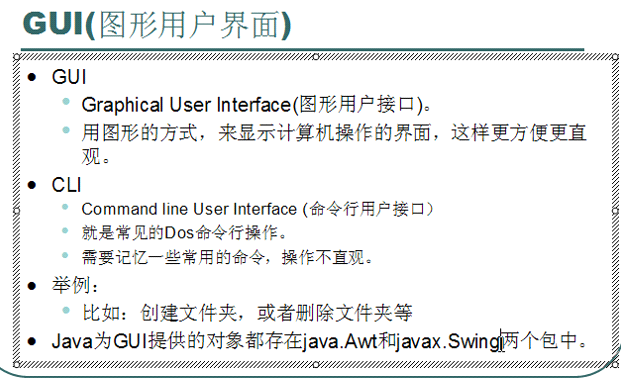
目的：输出流、OutPutStream、Writer

是纯文本则用字符流

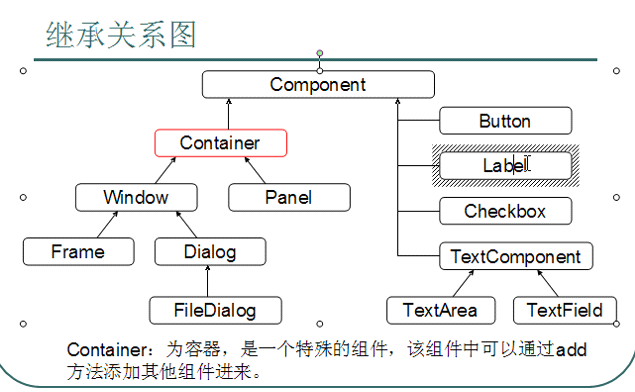
不是用字节流

# 二十二. GUI

## GUI

****

## 继承关系

****

## 布局管理器

****

**public static void main(String[] args)**

**{**

**//创建窗体**

**Frame frame = new Frame("test");**

**//设置窗体属性**

**frame.setSize(500,300);**

**frame.setLocation(400,200);**

**frame.setLayout(new FlowLayout());**

**//定义组件**

**Button b = new Button("我是一个按钮");**

**//将组件加入到窗体中**

**frame.add(b);**

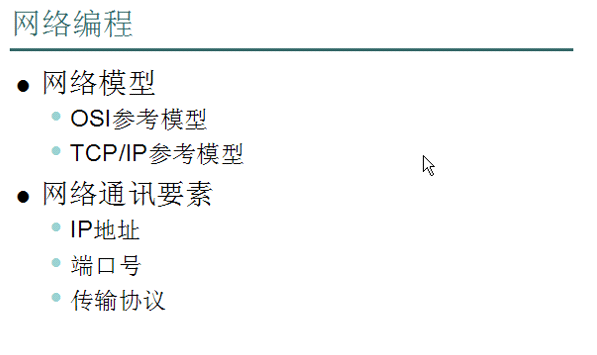
**//让窗体显示出来**

**frame.setVisible(true);**

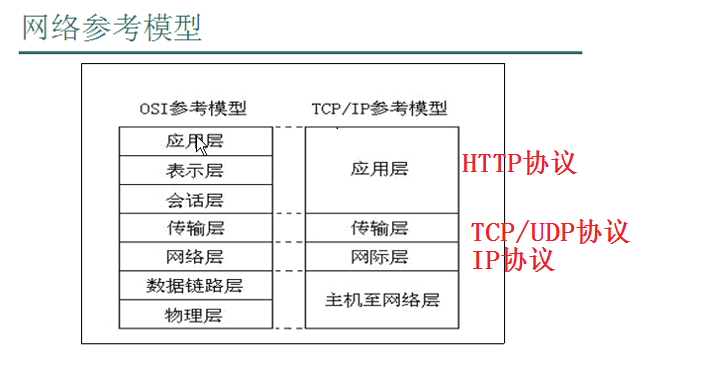
**}**

# 二十三. 网络编程

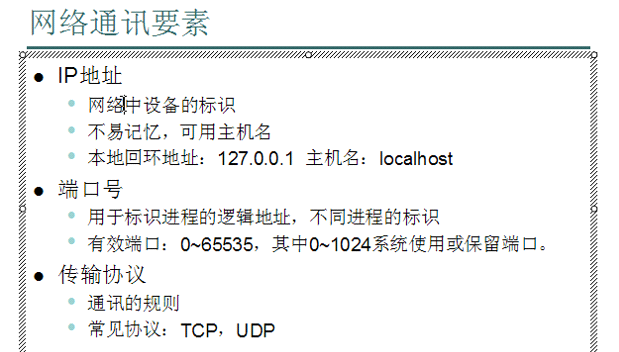
## 网络编程

****

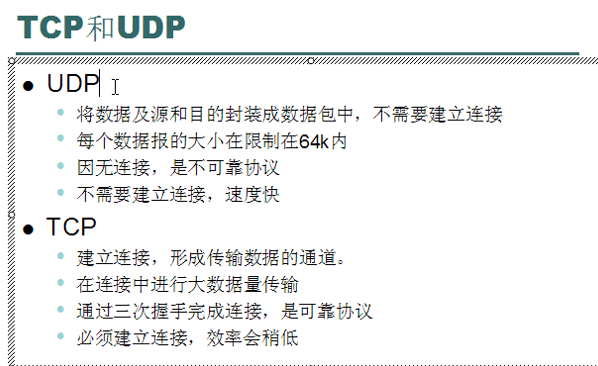
## 网络参考模型

****

## 网络通讯要素

****

## TCP和UDP

****

## java.net.InetAddress简单使用

**public static void main(String[] args) throws UnknownHostException**

**{**

**InetAddress in = InetAddress.getLocalHost();**

**InetAddress ia = InetAddress.getByName("223.252.199.6");**

**InetAddress[] ib = InetAddress.getAllByName("**[**www.iqiyi.com**](www.iqiyi.com)**");**

**System.out.println(in.toString());**

**System.out.println(ia.toString());**

**for (int i =0;i<ib.length ;i++ )**

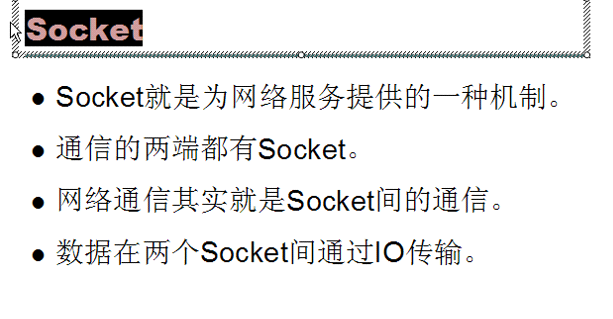
**{**

**System.out.println(ib[i].toString());**

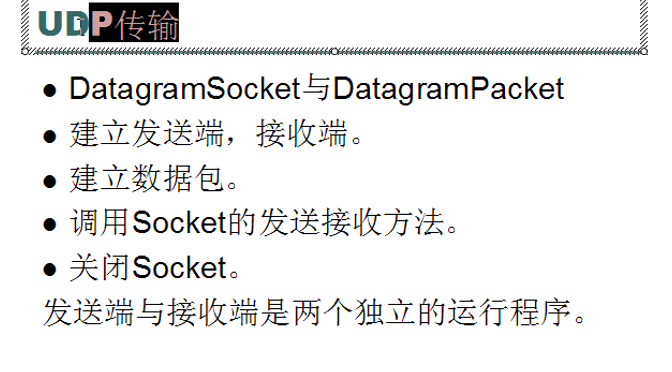
**}**

**}**

## Socket

****

## UDP传输

****

**/\***

**通过UDP传输方式将一段文字数据发送出去**

**1.建立udpsocket服务**

**2.提供数据并封装到数据报包中**

**3.发送**

**4.关闭资源**

**\*/**

**class sendUDP**

**{**

**public static void main(String[] args) throws Exception**

**{**

**DatagramSocket ds = new DatagramSocket();**

**byte[] data = "UDP is coming".getBytes();**

**DatagramPacket dp = new DatagramPacket(**

**data,**

**data.length,**

**InetAddress.getLocalHost(),**

**10000);**

**ds.send(dp);**

**ds.close();**

**}**

**}**

**/\***

**定义一个应用程序用于接收UDP传输的数据**

**定义接收端时要给这个网络应用程序定义一个数字标识 new DatagramSocket(10000)**

**方便明确接收哪些发送而来的数据包**

**1.定义udpsocket、建立端点**

**2.定义一个数据包用于存储接收到的字节数据和提取**

**3.通过socket的receive接收**

**4.数据包对象的特有功能提取需要的信息**

**\*/**

**class receiveUDP**

**{**

**public static void main(String[] args) throws Exception**

**{**

**DatagramSocket ds = new DatagramSocket(10000);**

**byte[] buf = new byte[1024];**

**DatagramPacket dp = new DatagramPacket(buf,buf.length);**

**ds.receive(dp);**

**String ip = dp.getAddress().getHostAddress();**

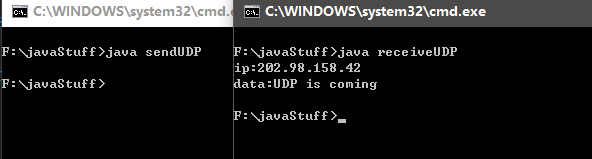
**String data = new String(dp.getData(),0,dp.getLength());**

**System.out.println("ip:"+ip+"\r\ndata:"+data);**

**ds.close();**

**}**

**}**

****

### 7.1 简单聊天

**在两个进程中完成，开启两个cmd窗口**

**import java.net.\*;**

**import java.io.\*;**

**class sendUDP2**

**{**

**public static void main(String[] args) throws Exception**

**{**

**//使用系统默认给分配的端口**

**DatagramSocket ds = new DatagramSocket();**

**BufferedReader bufr = new BufferedReader(**

**new InputStreamReader(System.in));**

**String line = null;**

**//readLine也是阻塞式方法**

**while ( (line = bufr.readLine() ) != null)**

**{**

**if ("886".equals(line))**

**break;**

**byte[] buf = line.getBytes();**

**DatagramPacket dp = new DatagramPacket(**

**buf,**

**buf.length,**

**InetAddress.getLocalHost(),**

**456);**

**ds.send(dp);**

**}**

**ds.close();**

**}**

**}**

**class receiveUDP2**

**{**

**public static void main(String[] args) throws Exception**

**{**

**DatagramSocket ds = new DatagramSocket(456);**

**while (true)**

**{**

**byte[] buf = new byte[1024];**

**DatagramPacket dp = new DatagramPacket(**

**buf,**

**buf.length);**

**ds.receive(dp);**

**String id = dp.getAddress().getHostAddress();**

**String data = new String(dp.getData(),0,dp.getLength());**

**System.out.println("id:"+id+"\r\ndata:"+data);**

**}**

**}**

**}**

**import java.net.\*;**

**import java.io.\*;**

**在一个cmd窗口中完成**

**/\***

**编写一个聊天程序，有收数据部分和发数据部分**

**这两部分需要同时执行（用到多线程）**

**因为收和发动作不一致因而要定义两个run方法，而且**

**两个方法要封装到不同的类中**

**\*/**

**class chat**

**{**

**public static void main(String[] args) throws Exception**

**{**

**DatagramSocket send = new DatagramSocket();**

**DatagramSocket receive = new DatagramSocket(1002);**

**new Thread(new Send(send)).start();**

**new Thread(new Receive(receive)).start();**

**}**

**}**

**class Send implements Runnable**

**{**

**private DatagramSocket ds;**

**public Send(DatagramSocket ds)**

**{**

**this.ds = ds;**

**}**

**public void run()**

**{**

**try**

**{**

**BufferedReader bufr = new BufferedReader(new InputStreamReader(System.in));**

**String line = null;**

**while ((line = bufr.readLine()) != null)**

**{**

**if ("stop".equals(line))**

**break;**

**byte[] buf = line.getBytes();**

**DatagramPacket dp = new DatagramPacket(**

**buf,**

**buf.length,**

**InetAddress.getLocalHost(),**

**1002);**

**ds.send(dp);**

**}**

**}**

**catch (Exception e)**

**{**

**e.printStackTrace();**

**}**

**}**

**}**

**class Receive implements Runnable**

**{**

**private DatagramSocket ds;**

**public Receive(DatagramSocket ds)**

**{**

**this.ds = ds;**

**}**

**public void run()**

**{**

**try**

**{**

**while (true)**

**{**

**byte[] buf = new byte[1024];**

**DatagramPacket dp = new DatagramPacket(buf,buf.length);**

**ds.receive(dp);**

**String ip = dp.getAddress().getHostAddress();**

**String data = new String(dp.getData(),0,dp.getLength());**

**System.out.println(ip+":"+data);**

**}**

**}**

**catch (Exception e)**

**{**

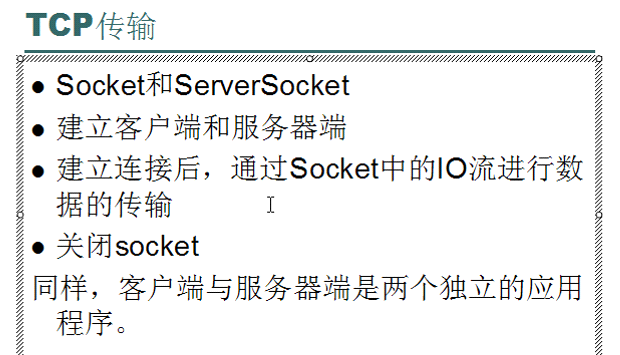
**e.printStackTrace();**

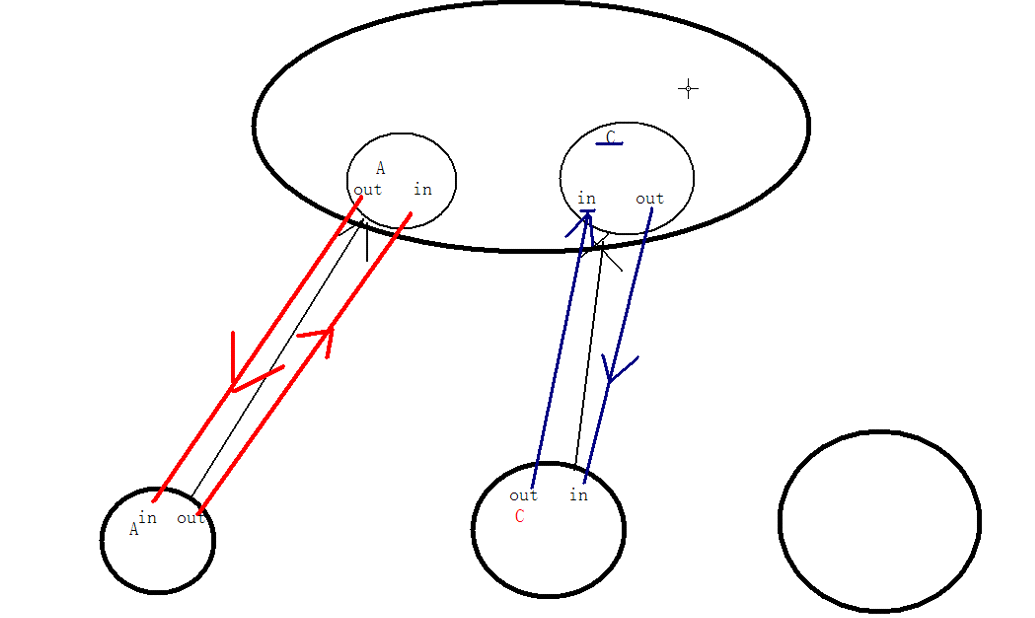
**}**

**}**

**}**

## TCP传输

****

****

**import java.net.\*;**

**import java.io.\*;**

**/\***

**TCP传输**

**1.TCP分客户端和服务端**

**2.客户端对应Socket**

**3.服务端对应ServerSocket**

**\*/**

**/\***

**客户端向指定IP地址的指定端口发送数据**

**\*/**

**class TCPClient**

**{**

**public static void main(String[] args) throws Exception**

**{**

**//建立时就可通过选择构造方法去连接指定的主机**

**//内部封装了输入流和输出流**

**Socket s = new Socket(InetAddress.getLocalHost(),1003);**

**OutputStream out = s.getOutputStream();**

**out.write("TCP is coming".getBytes());**

**s.close();**

**}**

**}**

**/\***

**服务端监听本机的指定端口以接收数据**

**\*/**

**class TCPServer**

**{**

**public static void main(String[] args) throws Exception**

**{**

**//指定绑定端口或使用默认端口建立ServerSocket监听**

**ServerSocket ss = new ServerSocket(1003);**

**//获取连接过来的客户端对象，没有连接就会阻塞**

**Socket s = ss.accept();**

**//通过客户端对象获得读取流**

**InputStream in = s.getInputStream();**

**byte[] buf = new byte[1024];**

**int len = in.read(buf);**

**System.out.println(s.getInetAddress()+":"+new String(buf,0,len));**

**//关闭客户端**

**s.close();**

**//关闭服务端（可选操作）**

**ss.close();**

**}**

**}**

**import java.net.\*;**

**import java.io.\*;**

**/\***

**TCP客户端和服务端互相访问**

**服务端收到数据后给客户端反馈信息**

**\*/**

**class TCPClient**

**{**

**public static void main(String[] args) throws Exception**

**{**

**Socket s = new Socket(InetAddress.getLocalHost(),1003);**

**OutputStream out = s.getOutputStream();**

**out.write("TCP is coming again".getBytes());**

**//获取socket流中的输入流，获取服务端反馈数据**

**InputStream in = s.getInputStream();**

**byte[] buf = new byte[1024];**

**int len = in.read(buf);**

**System.out.println(new String(buf,0,len));**

**s.close();**

**}**

**}**

**class TCPServer**

**{**

**public static void main(String[] args) throws Exception**

**{**

**ServerSocket ss = new ServerSocket(1003);**

**Socket s = ss.accept();**

**InputStream in = s.getInputStream();**

**byte[] buf = new byte[1024];**

**int len = in.read(buf);**

**System.out.println(s.getInetAddress()+":"+new String(buf,0,len));**

**Thread.sleep(5000);**

**s.getOutputStream().write("nice to meet you".getBytes());**

**}**

**}**

## PrintWriter自动有效刷新和添加换行

**Socket s = ss.accept();**

**BufferedReader bufr = new BufferedReader(new InputStream(s.getInputStream()));**

**BufferedWriter bufw = new BufferedWriter(new OutputStream(s.getOutputStream()));**

**String line;**

**while ( (line =bufr.readLine()) != null)**

**{**

**bufw.write(line.toUpperCase());**

**bufw.newLine();**

**bufw.flush();**

**}**

**可优化为：**

**PrintWriter out = new PrintWriter(s.getOutputStream(),true);**

**String line;**

**while ( (line =bufr.readLine()) != null)**

**{**

**out.println(line.toUpperCase());**

**}**

**import java.io.\*;**

**import java.net.\*;**

## 开启多个进程同时可接受来自不同主机的图片上传请求

**客户端**

**\*/**

**class PicClient**

**{**

**public static void main(String[] args) throws Exception**

**{**

**if (args.length != 1)**

**{**

**System.out.println("请选择一个jpg格式的图片文件");**

**return;**

**}**

**File file = new File(args[0]);**

**if (!(file.exists() && file.isFile() && file.getName().endsWith(".jpg")))**

**{**

**System.out.println("不是标准文件");**

**return;**

**}**

**if (file.length() > 1024\*1024\*5)**

**{**

**System.out.println("文件大小不能超过5M");**

**return;**

**}**

**Socket s = new Socket(**

**InetAddress.getLocalHost(),**

**1003);**

**FileInputStream fis = new FileInputStream("file");**

**OutputStream out = s.getOutputStream();**

**byte[] buf = new byte[1024];**

**int len = 0;**

**while ((len = fis.read(buf)) != -1)**

**{**

**out.write(buf,0,len);**

**}**

**//告诉服务端数据已写完**

**s.shutdownOutput();**

**InputStream in = s.getInputStream();**

**byte[] bufin = new byte[1024];**

**int num = in.read(bufin);**

**System.out.println(new String(bufin,0,num));**

**s.close();**

**fis.close();**

**}**

**}**

**/\***

**服务端（局限性：同一时刻只能处理一个客户端的上传请求）**

**改进：定义线程（多线程并行处理）**

**\*/**

**class PicServer**

**{**

**public static void main(String[] args) throws Exception**

**{**

**ServerSocket ss = new ServerSocket(1003);**

**while (true)**

**{**

**new Thread(new PicThread(ss.accept())).start();**

**}**

**//ss.close();//将服务端也关闭**

**}**

**}**

**class PicThread implements Runnable**

**{**

**private Socket socket;**

**public PicThread(Socket socket)**

**{**

**this.socket = socket;**

**}**

**public void run()**

**{**

**int count = 1;**

**String path = "C:**[**\\Users\\ai\\Desktop\\**](file:///\\Users\\ai\\Desktop\\)**图片\\";**

**String ip = socket.getInetAddress().getHostAddress();**

**System.out.println(Thread.currentThread().getName()+":"+ip);**

**try**

**{**

**InputStream in = socket.getInputStream();**

**File file = new File(path+ip+"("+(count)+")"+".jpg");**

**while (file.exists())**

**{**

**file = new File(path+ip+"("+(count++)+")"+".jpg");**

**}**

**FileOutputStream fos = new FileOutputStream(file);**

**byte[] buf = new byte[1024];**

**int len =0;**

**while ((len = in.read(buf)) != -1)**

**{**

**fos.write(buf,0,len);**

**}**

**OutputStream out = socket.getOutputStream();**

**out.write("上传成功".getBytes());**

**fos.close();**

**socket.close();//将客户端关闭**

**}**

**catch (Exception e)**

**{**

**throw new RuntimeException("上传失败");**

**}**

**}**

**}**

## 多线程处理图片上传

**import java.io.\*;**

**import java.net.\*;**

**/\***

**开启多个进程同时可接受来自不同主机的图片上传请求**

**客户端**

**\*/**

**class PicClient**

**{**

**public static void main(String[] args) throws Exception**

**{**

**if (args.length != 1)**

**{**

**System.out.println("请选择一个jpg格式的图片文件");**

**return;**

**}**

**File file = new File(args[0]);**

**if (!(file.exists() && file.isFile() && file.getName().endsWith(".jpg")))**

**{**

**System.out.println("不是标准文件");**

**return;**

**}**

**if (file.length() > 1024\*1024\*5)**

**{**

**System.out.println("文件大小不能超过5M");**

**return;**

**}**

**Socket s = new Socket(**

**InetAddress.getLocalHost(),**

**1003);**

**FileInputStream fis = new FileInputStream("file");**

**OutputStream out = s.getOutputStream();**

**byte[] buf = new byte[1024];**

**int len = 0;**

**while ((len = fis.read(buf)) != -1)**

**{**

**out.write(buf,0,len);**

**}**

**//告诉服务端数据已写完**

**s.shutdownOutput();**

**InputStream in = s.getInputStream();**

**byte[] bufin = new byte[1024];**

**int num = in.read(bufin);**

**System.out.println(new String(bufin,0,num));**

**s.close();**

**fis.close();**

**}**

**}**

**/\***

**服务端（局限性：同一时刻只能处理一个客户端的上传请求）**

**改进：定义线程（多线程并行处理）**

**\*/**

**class PicServer**

**{**

**public static void main(String[] args) throws Exception**

**{**

**ServerSocket ss = new ServerSocket(1003);**

**while (true)**

**{**

**new Thread(new PicThread(ss.accept())).start();**

**}**

**//ss.close();//将服务端也关闭**

**}**

**}**

**class PicThread implements Runnable**

**{**

**private Socket socket;**

**public PicThread(Socket socket)**

**{**

**this.socket = socket;**

**}**

**public void run()**

**{**

**int count = 1;**

**String path = "C:**[**\\Users\\ai\\Desktop\\**](file:///\\Users\\ai\\Desktop\\)**图片\\";**

**String ip = socket.getInetAddress().getHostAddress();**

**System.out.println(Thread.currentThread().getName()+":"+ip);**

**try**

**{**

**InputStream in = socket.getInputStream();**

**File file = new File(path+ip+"("+(count)+")"+".jpg");**

**while (file.exists())**

**{**

**file = new File(path+ip+"("+(count++)+")"+".jpg");**

**}**

**FileOutputStream fos = new FileOutputStream(file);**

**byte[] buf = new byte[1024];**

**int len =0;**

**while ((len = in.read(buf)) != -1)**

**{**

**fos.write(buf,0,len);**

**}**

**OutputStream out = socket.getOutputStream();**

**out.write("上传成功".getBytes());**

**fos.close();**

**socket.close();//将客户端关闭**

**}**

**catch (Exception e)**

**{**

**throw new RuntimeException("上传失败");**

**}**

**}**

**}**

## 自定义简单服务端

**可以在任意浏览器中输入**[**http://171.223.192.255:11007/**](http://171.223.192.255:11007/)**即可访问该服务器**

**\*/**

**import java.io.\*;**

**import java.net.\*;**

**class ServerDemo**

**{**

**public static void main(String[] args) throws Exception**

**{**

**System.out.println(InetAddress.getLocalHost());**

**ServerSocket ss = new ServerSocket(11007);**

**Socket s = ss.accept();**

**String ip = s.getInetAddress().getHostAddress();**

**System.out.print("来自："+ip+" ");**

**PrintWriter out = new PrintWriter(s.getOutputStream(),true);**

**out.println("<font color='red' size='60'>客户端你好</font>");**

**s.close(); //关闭客户端**

**ss.close();//关闭服务端**

**}**

**}**

## 13. Tomcat7.0

**客户端：浏览器**

**服务端：Tomcat7.0提供主机服务**

**InetAddress.getLocalHost();//获得当前主机地址**

**127.0.0.1也是当前主机地址**

## 14. java.net.URL

**String getFile()**

**获取此 URL 的文件名。**

**String getHost()**

**获取此 URL 的主机名（如果适用）。**

**String getPath()**

**获取此 URL 的路径部分。**

**int getPort()**

**获取此 URL 的端口号。**

**String getProtocol()**

**获取此 URL 的协议名称。**

**String getQuery()**

**获取此 URL 的查询部分。**

**public static void main(String[] args) throws Exception**

**{**

**URL url = new URL("**[**http://www.hao123.com:80/?tn=95771877\_hao\_pg**](http://www.hao123.com:80/?tn=95771877_hao_pg)**");**

**println("File:"+url.getFile());**

**println("Host:"+url.getHost());**

**println("Path:"+url.getPath());**

**println("Port:"+url.getPort());**

**println("Protocol:"+url.getProtocol());**

**println("Query:"+url.getQuery());**

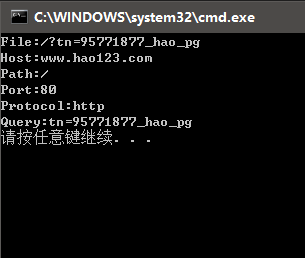
**}**

**private static void println(Object obj)**

**{**

**System.out.println(obj.toString());**

**}**

****

## 15. java.net.URLConnection

**public static void main(String[] args) throws Exception**

**{**

**URL url = new URL("**[**http://www.hao123.com:80/?tn=95771877\_hao\_pg**](http://www.hao123.com:80/?tn=95771877_hao_pg)**");**

**URLConnection con = url.openConnection();**

**InputStream in = con.getInputStream();**

**byte[] buf = new byte[1024];**

**int len;**

**while ( (len = in.read(buf)) != -1)**

**{**

**System.out.println(new String(buf,0,len));**

**}**

**}**

## 16. 域名解析

**DNS**

**hosts**

# 二十五. 正则表达式

## 正则表达式

**用特定的符号来表示一些代码操作，简化书写，符合一定规则的表达式**

**符号定义越多，正则越长、阅读性极差。**

**作用：**

**操作字符串 java,lang.String;**

**具体操作功能：**

## 匹配

**boolean matches(String regex)用规则匹配字符串**

## 切割

**String[] spilt(String regex);**

**class main**

**{**

**public static void main(String[] args) throws Exception**

**{**

**//按照多个空格来切割**

**String qq = "0 3 2 45 2 3 5";**

**String regex = " +";**

**String[] strs = qq.split(regex);**

**//out(strs);**

**//按‘.’切割**

**String q = "asfsa.gasdg.asgd.wewe";**

**String[] a = q.split("\\.");**

**//out(a);**

**String w = "C:**[**\\Users\\ai\\Desktop\\**](file:///\\Users\\ai\\Desktop\\)**图片\\截图**[**\\2016-9-23\\Image\_007.png**](file:///\\2016-9-23\\Image_007.png)**";**

**String[] aw = w.split("\\\\");**

**//out(aw);**

**//按照叠词切**

**String str = "qweertuufdgghvvghjkloo";**

**//为了让规则结果被重用可以运用组的理念**

**//组：这里把第一位设置为一个组"(.)"自动编号为1，第二位使用到第一组则用\1来获取组**

**String[] re = str.split("(.)**[**\\1+**](file:///\\1+)**");**

**out(re);**

**}**

**private static void out(String[] s)**

**{**

**System.out.println("length:"+s.length);**

**for (String str : s )**

**{**

**System.out.println(str);**

**}**

**}**

**}**

## 替换

**String replaceAll(String regex,String replacement)**

**class main**

**{**

**public static void main(String[] args) throws Exception**

**{**

**//将字符串中的非数字替换为’#‘**

**String str = "asdfqwre5234afghy67rwueit467rt3gyu";**

**//replace(str,"**[错误!超链接引用无效。](\\\\D{1,})**");**

**//将叠词替换为‘@’**

**String str1 = "uiojhkhhhrweqqnmzzxbccv";**

**//replace(str1,"(.)**[**\\1+","@**](file:///\\1+)**");**

**//将叠词替换为对应单个字母**

**String str2 = "uiojhkHHHrweQQnmZZxbccv";**

**replace(str2,"(.)**[**\\1+","$1**](file:///\\1+)**");**

**}**

**private static void replace(String source,String regex,String replacement)**

**{**

**System.out.println(source.replaceAll(regex,replacement));**

**}**

**}**

## 获取

**将字符串中符合规则的子串取出**

**1.将正则表达式封装成对象**

**2.正则表达式对象与字符串相关联**

**3.关联后，获取正则匹配引擎**

**4.通过引擎对符合规则的子串进行操作，比如取出**

**String str = "asdf asdf aa aa wer qerqw wer sdf";**

**String regex = "";**

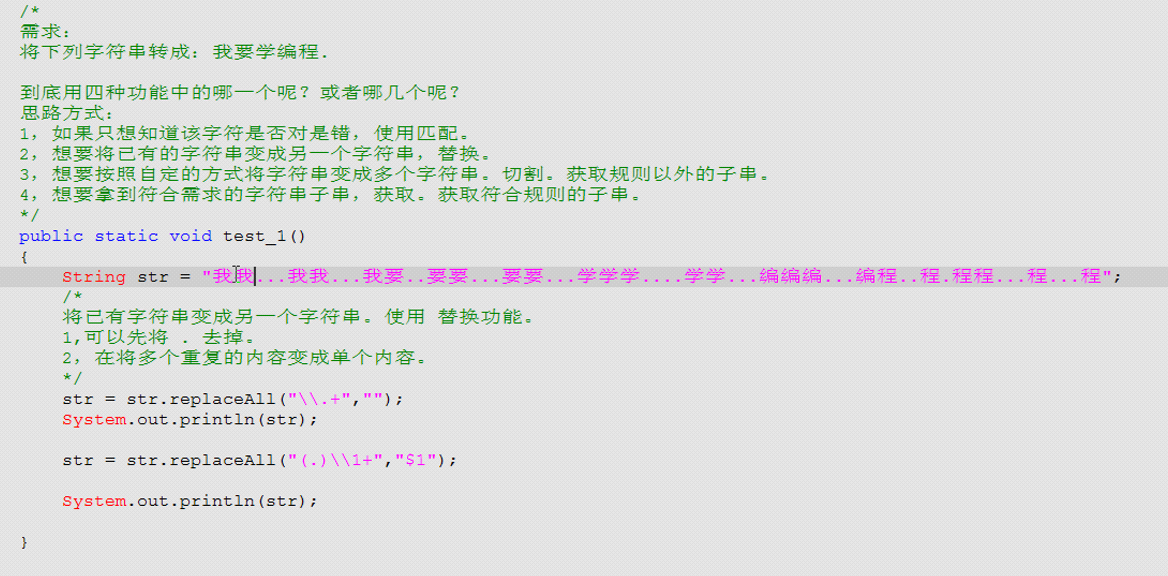
**//正则对象**

**Pattern p = Pattern.compile(regex);**

**//匹配器**

**Matcher m = p.matcher(str);**

## 实例

****

## 网页爬虫（蜘蛛）

**从网页上爬去邮箱地址**

**正则表达式不全面**

**import java.util.regex.\*;**

**import java.io.\*;**

**import java.net.\*;**

**class main**

**{**

**public static void main(String[] args) throws Exception**

**{**

**getMails();**

**}**

**//获取指定文档（网页）中的特定内容**

**private static void getMails() throws Exception**

**{**

**BufferedReader bufr = new BufferedReader(new FileReader("test.txt"));**

**show(bufr);**

**}**

**//获取指定文档（网页）中的特定内容**

**private static void getMailsFromWeb() throws Exception**

**{**

**URL url = new URL("**[**http://blog.sina.com.cn/s/blog\_515617e60101e151.html**](http://blog.sina.com.cn/s/blog_515617e60101e151.html)**");**

**URLConnection conn = url.openConnection();**

**BufferedReader bufr = new BufferedReader(new InputStreamReader(conn.getInputStream()));**

**show(bufr);**

**}**

**private static void show(BufferedReader bufr) throws Exception**

**{**

**String line = null;**

**String regex = "**[**\\w+@\\w+(\\.\\w+)+**](file:///\\w+@\\w+(\\.\\w+)+)**";**

**Pattern p = Pattern.compile(regex);**

**while ((line = bufr.readLine()) != null)**

**{**

**Matcher m = p.matcher(line);**

**while (m.find())**

**{**

**System.out.println(line);**

**}**

**}**

**}**

**}**