# 进制

Java针对整数常量提供了4种表现形式：二进制 、8进制、十进制、16进制

计算机存储单位：bit(位) ,byte(字节),KB,MB,GB,TB

bit(位): 二进制表现方式：0000 0000

1byte(字节)=8bit(位)

1TB=1024GB,1GB=1024MB,1MB=1024KB ,1KB=1024byte

二进制 只有0和1两个数码来表示的数。进位规则是“逢二进一”，借位规则是“借一当二”

一般都是0b开头后尾是1和0的都是二进制

0=0000 0000；1=0000 0001；2=0000 0010

十进制

0=0000 0000；1=0000 0001；2=0000 0002;………10=0000 0010

八进制 0,1,2,3,4,5,6,7,没有8和9。进位规则是“逢八进一”

0=00000000，1=00000001……….. 7=00000001, 8=00000010,9=00000011

十六进制 除了0-9数字还有： a，b，c，d，e ,f分别代表：10,11,12,13,14,15

内存地址都是用16进制形式表示的：0x251f

0=00000000, 1=0000 0001，………..10=0000 0000a ,11=0000 0000b, 16=0000 00010, 17=0000 00011

计算机存储格式与原码、反码、补码

计算机存”都是二进制形式进行存储的，计算机只认二进制

十进制的整数1在计算机存储格式： 0000 0001

十进制的整数2在计算机存储格式： 0000 0010

负数的表现方式：16位数最前面(符号位)变1 就是负数

符号位:即八位字符的第一位，1为负，0为正。

十进制的整数-2在计算机存储格式：1000 0010

这样导致了问题：在二进制 2+(-2)就不正确了：衍生了：原码、反码、补码

进行负数和整数相加规则：

正数：原码= 反码=补码

负数：需要最后的补码来进行操作

反码=符号位不变，其他位取反

补码=反码+1

十进制、二进制、八进制、十六进制的4个互转：记住公式即可(很简单)

系数\*基数的权次幂然后进行相加

给个例子：十进制到十进制的转换：先拆分相加再拆分成次方公式

12345=10000+2000+300+40+5 = 1\*10^4+2\*10^3+3\*10^2+4\*10^1+5\*10^0

系数：就是每一位上的数据：如上面：123456

基数：进制 如上面的是十进制：基数就是10

圈：固定的，从右边开始编号：即上面次方的0、1、2、3

所以公式是：系数\*基数的权次幂然后进行相加

二进制转10进制：注意0b开头是java中代表二进制

0b100=1\*2^2+0\*2^1+0\*^2^0=4;使用乘法口诀相加后最后得到4，

八进制转10进制：注意0开头是java中代表八进制

0100=1\*8^2+0\*8^1+0\*^8^0=64;

十六进制转10进制：注意0x开头代表16进制

0x100=1\*16^2+0\*16^1+0\*^16^0=256;

注意问题：有可能面试问到

byte by = 130;有没有问题?有问题如何解决?结果是多少呢? byte 的数值范围是 126~-127 130已经超出byte取值范围 运算是会砍掉前面的 三个8位结果会变成-126

byte b1=3,b2=4,b;

b=b1+b2;

b=3+4;

哪句是编译失败的呢？为什么呢？

//b=b1+b2出错在JAVA虚拟机中默认int类型转换，虚拟机不知道b1,b2中的数值是多少，所有会抛出可能损失精度的错误;

虚拟机会把 b1,b2 转换成int类型在进行运算，两个int类型的数值相加结果任然是int类型，int类型不能赋值给 byte类型

byte b = 10;

b++;//这句 java虚拟机会做一个自动转换动作相当于b=(byte)(b+1)所以不会报错

b = b + 1;//这句java虚拟机会把，b转换成int类型 相当于 int b + int 类型的1 然后赋值给Byte类型 int是4个字节 byte是1个字节 所有会抛出损失精度会报错需要强制类型转换

# 位运算符

位运算符：&,|,^,~,<<,>>,>>>7种

分析：因为是位运算，所以我们必须先把数据换算成二进制。

第一步 数字3和4在内存中二进制数据(java默认int类型)

3的二进制：11，转换为int类型4字节：00000000 00000000 00000000 00000011

4的二进制：100，转换为int类型4字节：00000000 00000000 00000000 00000100

第二步 使用为运算符的操作结果

(1) <<:左移 左边最高位丢弃，右边补齐0

//<< 把<<左边的数据乘以2的移动次幂

System.out.println(3 << 2); //3\*2^2 = 3\*4 = 12; 下面是运算原理：

注意：如果是负数需要转换为补码(正数是原码=反码=补码)来进行操作，操作完后再转换为原码

00000000 00000000 00000000 00000011 二进制 3

(00)000000 00000000 00000000 0000001100 进行左移后的二进制结果12

(2) >>:右移 最高位（符号位）是0，左边补齐0；最高为是1，左边补齐1

//>> 把>>左边的数据除以2的移动次幂

System.out.println(24 >> 2); //24 / 2^2 = 24 / 4 = 6 原理几乎和左移一样。3>>2值为0

(3) >>>:无符号右移 无论最高位是0还是1，左边补齐0

System.out.println(24 >>> 2);结果是6 原理几乎和左移一样

System.out.println(-24 >>> 2);结果是1073741818

(4)&位与运算：有0则0。

System.out.println(3 & 4);//结果为0，下面是运算原理：

00000000 00000000 00000000 00000011 二进制3

00000000 00000000 00000000 00000100 二进制4

&------------------------------------------------------- 两个运算只要一个有0最后就变为0

00000000 00000000 00000000 00000000结果是：0

(5) |位或运算：有1则1。

System.out.println(3 | 4);//结果为7,下面是运算原理：

00000000 00000000 00000000 00000011 二进制3

00000000 00000000 00000000 00000100 二进制4

|------------------------------------------------------- 两个运算只要一个有1最后就变为1

00000000 00000000 00000000 00000111 //折算为十进制为数字7

(6) ^位异或运算：相同则0，不同则1。

System.out.println(3 ^ 4); //结果为7,下面是运算原理：

00000000 00000000 00000000 00000011 二进制3

00000000 00000000 00000000 00000100 二进制4

^------------------------------------------------------- 两个运算只要相同为0，不同是1

00000000 00000000 00000000 00000111 //折算为十进制为数字7

^还有一个特点：一个数据对另一个数据位异或两次，该数本身不变。

int a = 10;int b = 20;

System.out.println(a ^ b ^ b); //结果等于10

System.out.println(a ^ b ^ a); //结果等于20

(7) ~按位取反运算符：0变1，1变0。

System.out.println(~3); 结果是：-4

00000000 00000000 00000000 00000011

~11111111 11111111 11111111 11111100 (补码) 取反后获得是补码

补码：11111111 11111111 11111111 11111100

反码：11111111 11111111 11111111 11111011 反码=补码-1

原码：10000000 00000000 00000000 00000100 原码=反码的符号位不变其它取反。所以最后结果是-4

# 码表

Java字符(char)采用Unicode 编码，每个字符占两个字节，因而可用十六进制编码形式表示。

为什么会乱码？因为每个码表设置的字节都不一样，好比gpk中文进行Iso8859-1进行转换（它的字节都是一位，又不支持中文），所以会造成乱码

ASCII 原生的码表

ASCII（American Standard Code for Information Interchange，它由128个字符组成（对应二进制：01111111，转成十进制127的数字）包括大小写字母、数字0-9、标点符号、非打印字符（换行符、制表符等4个）以及控制字符（退格、响铃等）组成。总共有 128 个，用一个字节的低 7 位表示，0~31 是控制字符如换行回车删除等；32~126 是打印字符，可以通过键盘输入并且能够显示出来。

需记住：48-57对应：十进制0~9；65~90对应：十进制A~Z；97~122对应：十进制a~z

ISO-8859-1 (Latin1是别名) 在ASCII基础上 扩展了

128 个字符显然是不够用的，于是 ISO 组织在 ASCII 码基础上又制定了一些列标准用来扩展 ASCII 编码，它们是 ISO-8859-1~ISO-8859-15，其中 ISO-8859-1 涵盖了大多数西欧语言字符，应用的最广泛。ISO-8859-1 仍然是单字节编码，它总共能表示 256 个字符（对应二进制：11111111，转成十进制255的数字）。

GBK与gb2312 通常采用EUC储存方法，以便兼容于ASCII

GB2312: 全称为信息交换用汉字编码字符集，是中国于1980年发布，主要用于计算机系统中的汉字处理。GB2312主要收录了6763个汉字、682个符号。

在使用GB2312的程序中，通常采用EUC储存方法，以便兼容于ASCII。浏览器编码表上的“GB2312”，通常都是指“EUC-CN”表示法。

GBK:全称为Chinese Internal Code Specification，即汉字内码扩展规范，于1995年制定。它主要是扩展了GB2312，在它的基础上又加了更多的汉字，它一共收录了21003个汉字。GBK是向下兼容GB2312编码的，也就是说GB2312编码的汉字可以用GBK正常解码不会出现乱码，但用GBK编码的汉字用GB2312解码就不一定了。

UTF-8和UNICODE（万国码） [\u4f60\u597d ]：你好

简单来说： Unicode 是「字符集」 UTF-8 是Unicode「编码规则」

UNICODE:为了自己的语言能在计算机中正常显示，每个国家和地区都有各自的编码，所以编码多了谁也不认识对方的编码，这时候ISO组织就提出了一种新的编码叫UNICODE编码让全球的文化、字符、符号都能支持。UNICODE在制定时计算机容量已不是问题，所以设计成了固定两个字节，所有的字符都用16位表示，包括之前只占8位的英文字符等，所以会造成空间的浪费，UNICODE在很长的一段时间内都没有得到推广应用。UNICODE 在网络传输中，出现了两个标准 UTF-8 和 UTF-16，分别每次传输 8个位和 16个位。

UTF-8: 虽然UTF-16很高效，但也是UNICODE最大的坏处，使得所有单字节字符一定要占两个字节，存储空间放大了一倍，这明显消耗了资源，不符合现在互联网高速发展的现状。所以有了UTF-8，它是UNICODE的一种可变长度字符编码的实现，它可以使用1～6个定长字节来编码UNICODE字符

UTF-8 既然能保存那么多文字、符号，为什么国内还有这么多使用 GBK 等编码的人？

因为 UTF-8 等编码体积比较大，占电脑空间比较多，如果面向的使用人群绝大部分都是中国人，用 GBK 等编码也可以。

各码表占用字节

ASCII： 美国标准信息交换码。用一个字节的7位可以表示，都是一个字节

ISO8859-1： 拉丁码表。欧洲码表，用一个字节的8位表示。跟ascii基础上增加了，从而支持了欧洲一些国家

GB2312： 英文占一个字节， 中文占两个字节。

GBK： 英文占一个字节， 中文占两个字节。

Unicode： 国际标准码，。所有文字都用两个字节来表示，

UTF-8： 英文占一个字节，中文占三个字节。 最多用三个字节来表示一个字符。

（我们以后接触最多的是iso8859-1、gbk、utf-8）

UTF-16: 不管英文中文都是占两个字节。

## 编码与解码

编码： 把看得懂的字符变成看不懂码值这个过程我们称作为编码。字符---数字

解码： 把码值查找对应的字符，我们把这个过程称作为解码。数字----字符

注意： 以后编码与解码一般我们都使用统一的码表。否则非常容易出乱码。

unicode实际上就是用了utf-16.它不管字母还是中文统一都是2个字节，它有一个恶心之处，就是打印出来的都有-1和-2进行开头

public class Demo7 {

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

/\*

String str = "中国";

byte[] buf = str.getBytes("utf-8");// 平台默认的编码表是gbk编码表。 编码

System.out.println("数组的元素："+Arrays.toString(buf)); //

str = new String(buf,"utf-8"); //默认使用了gbk码表去解码。

System.out.println("解码后的字符串："+ str);

\*/

/\*String str = "a中国"; // ,0, 97, 78, 45, 86, -3

byte[] buf = str.getBytes("unicode"); //编码与解码的时候指定 的码表是unicode实际上就是用了utf-16.

System.out.println("数组的内容："+ Arrays.toString(buf));

\*/

String str = "大家好";

byte[] buf = str.getBytes(); //使用gbk进行编码

System.out.println("字节数组："+ Arrays.toString(buf)); // -76, -13, -68, -46, -70, -61

str = new String(buf,"iso8859-1");

// 出现乱码之后都可以被还原吗？ 出现乱码不一定能还原

byte[] buf2 = str.getBytes("iso8859-1");

str = new String(buf2,"gbk");

System.out.println(str);

## URL编码

URL编码 也叫%编码方式：[%e4%bd%a0%e5%a5%bd ]：你好

由于HTTP协议规定URL路径中只能存在ASCII码中的字符，所以如果URL中存在中文或特殊字符需要进行URL编码。 利用将ascii码中没有字符转换为ascii码中有的字符表示 如%5a

在java中进行URL编码和解码，必须要用UTF-8 因为浏览器只认utf-8

URLencoder.encode("xxxx","utf-8");//编码

URLDecoder.decode(str,"utf-8"); //解码

# Base64

Base64 [5L2g5aW9] ：你好

base64是因为互联网发展而出现的，当电子邮件、web网页等互联网应用出现后，由于人们使用的自然语言（那时主要是英语）是用ASCII编码的， ASCII使用一个字节全部的8位，这样如果直接传输，就会被网关、路由器等设备把一个字节的最高位置零，从而影响传输内容的正确性。于是出现了base64编码

在一端发送GB2312编码－>根据Base64规则－>转换成ASCII码，接收端收到ASCII码－>根据Base64规则－>还原到GB2312编码。

Base64是一种任意二进制到文本字符串的编码方法，常用于在URL、Cookie、网页中传输少量二进制数据。

System.out.print("请输入用户名:");

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

String userName = in.readLine();

jdk1.8

Encoder encoder=Base64.getEncoder();

System.out.println("编码后的用户名为:" + encoder.encodeToString(userName.getBytes()));

JDK7

System.out.println(Base64.encode(userName.getBytes())); //打印：5L2g5aW9

System.out.println(new String(Base64.decode(encoder);,"utf-8");); //打印：你好

# 哈希表

用途：加密，解决冲突问题

哈希表（Hash table，也叫散列表），是根据关键码值(Key value)而直接进行访问的数据结构。也就是说，它通过把关键码值映射到表中一个位置来访问记录，以加快查找的速度。这个映射函数叫做散列函数，存放记录的数组叫做散列表。

哈希表hashtable(key，value) 的做法其实很简单，就是把Key通过一个固定的算法函数既所谓的哈希函数转换成一个整型数字，然后就将该数字对数组长度进行取余，取余结果就当作数组的下标，将value存储在以该数字为下标的数组空间里。

而当使用哈希表进行查询的时候，就是再次使用哈希函数将key转换为对应的数组下标，并定位到该空间获取value，如此一来，就可以充分利用到数组的定位性能进行数据定位

Object 方法的hashCode和equals，一般用来重写，hsahMap、hashSet的存储需要调用这两个方法判断是否同一个

# UUID

概念

UUID是由一组32位数的16进制数字所构成，所以UUID理论上的总数为16^32=2^128，约等于3.4 x 10^38。也就是说若每纳秒产生1兆个UUID，要花100亿年才会将所有UUID用完。

UUID是1.5中新增的一个类，在java.util下，用它可以产生一个号称全球唯一的ID

目前最广泛应用的UUID，是微软公司的全局唯一标识符（GUID）

作用

UUID 的目的是让分布式系统中的所有元素，都能有唯一的辨识资讯，而不需要透过中央控制端来做辨识资讯的指定。如此一来，每个人都可以建立不与其它人冲突的 UUID。在这样的情况下，就不需考虑数据库建立时的名称重复问题。

public static void main(String[] args) {

UUID uuid = UUID.randomUUID();

System.out.println (uuid);//打印：a57aaafc-bb7f-4550-98fc-d9d114c5b1e7

}

# MD5

MD5即Message-Digest Algorithm 5（信息-摘要算法5），用于确保信息传输完整一致。是计算机广泛使用的杂凑算法之一（又译摘要算法、哈希算法），主流编程语言普遍已有MD5实现。将数据（如汉字）运算为另一固定长度值，是杂凑算法的基础原理，MD5的前身有MD2、MD3和MD4。

MD5的作用是让大容量信息在用[数字签名](https://baike.baidu.com/item/%E6%95%B0%E5%AD%97%E7%AD%BE%E5%90%8D)软件签署私人[密钥](https://baike.baidu.com/item/%E5%AF%86%E9%92%A5)前被"[压缩](https://baike.baidu.com/item/%E5%8E%8B%E7%BC%A9/13032501)"成一种保密的格式（就是把一个任意长度的字节串变换成一定长的[十六进制](https://baike.baidu.com/item/%E5%8D%81%E5%85%AD%E8%BF%9B%E5%88%B6)数字串）。除了MD5以外，其中比较有名的还有[sha-1](https://baike.baidu.com/item/sha-1)、[RIPEMD](https://baike.baidu.com/item/RIPEMD)以及Haval等。

# 异常

Throwable 是一个正常类 可以创建对象，

此类3个方法的效果如下:

public static void main(String[] args) {

//创建了一个Throwable对象。

Throwable t = new Throwable("头晕，感冒");

System.out.println("message: "+ t.getMessage());打印： message: 头晕，感冒

System.out.println("toString: "+ t.toString();); //打印：toString: java.lang.Throwable: 头晕，感冒

test();

}

public static void test(){

Throwable t = new Throwable();

t.printStackTrace();//打印：java.lang.Throwable

at com.wwl.reflect.Test.test(Test.java:16)

at com.wwl.reflect.Test.main(Test.java:12)

}

## 自定义异常操作

Sun公司没有内置负数异常：可以自己定义：

第一步：继承Exception或者RuntimeException

public class FuShuException extends RuntimeException{ //一般都是继承运行时异常

public FuShuException(String s){

super(s); //如果需要消息：需要调父类的带String构造方法

}

}

public static void main(String[] args) {

int avg = getAvg(50,60,-70,80);

System.out.println(avg);

}

\* 成绩没有负数,需要抛出异常,停止运算

public static int getAvg(int...source){ //运行异常不用抛出

int sum = 0 ;

for(int s : source){

if( s < 0){

第二步 手动抛出异常自己定义的异常对象（因为是此类是继承了运行时异常类所以不用抛出）

throw new FuShuException("成绩错误 "+s);

}

sum = sum + s;

}

return sum/source.length;

}

# 时间类

## Date

计算两个日期的相隔天数(自己活了多少岁)

Public long getInterval(Date begin, Date end) throws Exception{

long day = 0;

SimpleDateFormat sdf = new SimpleDateFormat("yyyyMMdd"); //第一步 创建格式化时间对象

if(begin\_date != null){

String b= sdf.format(begin); //第二步 先将Date格式化为只有年、月、日的形式，返回String。

begin = sdf.parse(b); //第三步然后将此String再转化为Date(此时时、分、秒就成为了00:00:00）

}

if(end!= null){

String e= sdf.format(end);

end = sdf.parse(e);

}

//第四步 调用getTime()方法返回Date的1970的毫秒表示:将两个日期的毫秒相减，然后除以一天有（24\*60\*60\*1000）毫秒，得到日期差。

day = (e.getTime()-b.getTime())/(24\*60\*60\*1000);

return day;

}

## Calendar

判断是否闰年

public void isLeapYear(int years){

Calendar cal=Calendar.getInstance();

cal.set(years,Calendar.DECEMBER,31);//12月是固定有31天的

if(cal.get(Calendar.DAY\_OF\_YEAR)==366){

System.out.println(years+"年是闰年");

}else{

System.out.println(years+"年平年");

}

}

第二种方式

if(i % 4 == 0 && i % 100 != 0 || i % 400 == 0){ //是闰年}else{ //不是闰年}

获取今天或者之后多少天的日期

第一步创建对象它已经是当前时间

Calendar calendar=Calendar.getInstance();

第二步 获取今天的日期，如6月25，那就是25

System.out.println("今天的日期是："+calendar.get(Calendar.DAY\_OF\_MONTH));

System.out.println("今年的日期是："+calendar.get(Calendar.DAY\_OF\_YEAR));是获取本年的日期

calendar.clear();//此语句是把日期更改为本月第一天,网上的是不正确的不应该用此方法

第三步 参数：今天的日期和未来的日期

calendar.set(Calendar.DAY\_OF\_MONTH, calendar.get(Calendar.DAY\_OF\_MONTH)+10);

第四步 再进行调用第二步的方法

//注意：使用clear方法是打印：十天之后的日期是：11，是不正确的，它把日期变为了本月1号

没有使用方法是正确的显示：下个月5号

System.out.println("十天之后的日期是："+calendar.get(Calendar.DAY\_OF\_MONTH));

## 计时器

用到的类：Timer和它的子类TimerTask

Timer类用来执行

TimerTask类：继承了线程Runnable，需在run方法中写程序

public static void main(String[] args) throws InterruptedException {

Timer t = new Timer(); 第一步 创建对象Timer

//在指定时间安排指定任务

//第一个参数,是安排的任务,第二个参数是执行的时间,第三个参数是过多长时间再重复执行

t.schedule(new MyTimerTask(), new Date(188, 6, 1, 14, 22, 50),3000);第二步 使用方法调用子类和时间

while循环中没用，只是控制台看效果

while(true) {

Thread.sleep(1000);

System.out.println(new Date());

}

}

}

----------------------------------继承子类TimerTask开始----------------------------------------------

class MyTimerTask extends TimerTask {

@Override

public void run() {

System.out.println("起床背英语单词");

}

}

# 文件操作

1、每次运行到代码FileOutputStream、FileWrite对象都会将目标文件进行清空

创建目录：mkdirs为子目录也进行创建，mkdir只创建一个文件目录

File file = new File("F:\\ct.properties");

FileWriter f=new FileWriter(file); //运行完此代码：目标文件就已经清空了，加了true：不清空，追加内容

f.write("你好");

f.close();

实例 模拟加了true的操作

File file = new File("F:\\ct.properties");

int count=0;

for(int i=0;i<2;i++){

if(count ==0){

f=new FileWriter(file); 关键只运行一次创建对象

}

f.write("第"+(++count)+"次");打印：第1次第2次

}

f.close();

2、通过HashCode配合UUID获得8位字符创建8个子文件目录

String name=Integer.toHexString(UUID.randomUUID().hashCode()); //’ a’的hachCode是61;使用toHexString方法

String mkd="e:/update";

for(char c : name.toCharArray()) { //转成char数组

mkd +="/"+c;

}

new File(mkd).mkdirs();

## IO流

IO技术主要的作用是解决设备与设备之间 的数据传输问题。 比如：

硬盘--->内存，把键盘的数据------->内存中，内存的数据---->硬盘上，

引用场景：导出报表 ， 上传大头照 、 下载 、 解释xml文件 ...

### File

File类可以描述一个文件或者一个文件夹

构造方法

1、File(String pathname) 指定文件或者文件夹的路径创建一个File文件。

2、File(File parent, String child) 根据 parent 抽象路径名和 child 路径名字符串创建一个新 File 实例。

用途：如在window系统下的盘符当父File：

File file = new File("d:\\"); //复盘

File file2 = new File(file,"a.txt"); //就不用指定d盘了

if(!file2.exists()) {

file2.createNewFile();

}

3、File(String parent, String child)

一般用2种构造方法：父路径可以先做预处理动作：如先删除d盘没用的，第3种不怎么用

参数要注意问题

细节一：目录分隔符 此类有静态域：File.separator

使用File.separator：在windows机器上的目录分隔符是 \ ,在linux机器上的目录分隔符是/

注意：在windows上面\ 与 / 都可以使用作为目录分隔符。 而且，如果写/ 的时候只需要写一个即可。

细节二：路径

绝对路径: 该文件在硬盘上 的完整路径。绝对路径一般都是以盘符开头的。

相对路径: 相对路径就是资源文件相对于当前程序所在的路径。

如果不知道相对路径的所在路径：老师教了一个方法：

//第一步：在此参数中放入一个点

File file = new File(".");

//第二步使用：file.getAbsolutePath()打印出来

System.out.println("当前路径是："+ file.getAbsolutePath());打印：I: \eclipsev4.7\eclipse \project\MyJavaCode\_1.7

#### 创建

共4个方法

1、createNewFile() 在指定位置创建一个空文件，成功就返回true，如果已存在就不创建然后返回false

File file = new File("F:\\aa");

System.out.println("创建成功了吗？"+file.createNewFile());

2、mkdir() 在指定位置创建目录，这只会创建最后一级目录，如果上级目录不存在就抛异常。

//使用 mkdir 创建一个单级文件夹，注意只能创建单级文件夹

File dir = new File("F:\\a.txt");

System.out.println("创建文件夹成功吗？"+dir.mkdir());

3、mkdirs() 在指定位置创建目录，这会创建路径中所有不存在的目录。

//使用 mkdirs 创建多级文件夹，下面demo会创建aa文件夹和bb文件夹

dir = new File("F:\\aa\\bb");

System.out.println("创建多级文件夹："+ dir.mkdirs());

4、renameTo(File dest) 重命名文件或文件夹，也可以操作非空的文件夹，文件不同时相当于文件的剪切,剪切时候不能操作非空的文件夹。移动/重命名成功则返回true，失败则返回false。

//重命名文件名，也可以重命名文件夹名：路径相同

File file = new File("d:\\a.txt");

File file2 = new File("d:\\cccc");

file.renameTo(file2);//要注意需要被重命名的file进行调用方法

//只能移动(剪切)文件不能移动文件夹：路径不相同

File file = new File("d:\\a.txt");

File file2 = new File("e:\\cccc");

file.renameTo(file2);//要注意需要被重命名的file进行调用方法

#### 判断

共5个方法

1、exists() 文件或文件夹是否存在。存在返回true

File file = new File("..\\..\\a.txt");

System.out.println("存在吗？"+ file.exists());

2、isFile() 是否是一个文件，如果不存在，则始终为false。

3、isDirectory() 是否是一个目录，如果不存在，则始终为false。

4、isHidden() 是否是一个隐藏的文件或是否是隐藏的目录。

5、isAbsolute() 测试此抽象路径名是否为绝对路径名。

#### 删除

共2个方法

1、delete() 删除文件或一个空文件夹，如果是文件夹不为空，不能删除，成功true，失败false。

2、deleteOnExit() 在虚拟机终止时，请求删除此抽象路径名表示的文件或目录，保证程序异常时创建的临时文件也可以被删除

File file = new File("F:\\a.txt");

file.deleteOnExit(); //jvm退出的时候删除文件。 一般用于删除临时 文件。

#### 获取

共6个方法

1、getName() 获取文件或文件夹的名称，不包含上级路径。

2、getPath() 返回绝对路径，可以是相对路径，但是目录要指定

3、getAbsolutePath() 获取文件的绝对路径

4、length() 获取文件的大小（字节数），如果文件不存在则返回0L，如果是文件夹也返回0L。

5、getParent() 返回父目录的路径名字符串；如果此路径名没有指定父目录，则返回null。

6、lastModified() 获取最后一次被修改的时间。可用来判断此文件是否被修改

long lastModified = file.lastModified();

Date date = new Date(lastModified);

SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy年MM月dd日 HH:mm:ss");

System.out.println("获取最后一次的修改时间(毫秒值)："+ dateFormat.format(date) );

7、staic File[] listRoots() 列出所有的根目录（Window中就是所有系统的盘符）

File[] roots = File.listRoots(); //列出所有的根目录

for(File file : roots){

System.out.println(file);

}

8、list() 返回目录下的文件或者目录名，包含隐藏文件。指定到文件不是目录会返回null。

//把 当前文件夹下面的所有子文件名与子文件夹名 存储到一个String类型 的数组中 返回。

File file = new File("F:\\ ");

String[] fileNames = file.list();

for(String fileName : fileNames){

System.out.println(fileName);

}

9、listFiles() 返回目录下的文件或者目录对象，包含隐藏文件。指定到文件不是目录会返回null。

// 返回file对象：存储到file数组中

File file = new File("F:\\ ");

File[] files = file.listFiles();

for(File fileItem : files){

System.out.println("文件名："+ fileItem.getName());

}

10、listFiles(FilenameFilter filter) 目录中符合过滤条件的子文件或子目录。指定文件不是目录会返回null。

//获取a目录中的后缀名为java的文件

File dir = new File("d:\\a");

File[] files = dir.listFiles(new FilenameFilter() { //第一步 使用匿名内部类

public boolean accept(File dir, String name) { //第二步 接口中需要被实现的方法

//System.out.println("文件夹:"+dir+" 文件名："+ name);

return name.endsWith(".java"); //第三步 使用字符串的方法判断是否为java

}

});

for(File file : files){ //第四步 fareace 得到a目录下的所有java文件

System.out.println(file.getName());

}

11、list(fileNameFilter filter) 目录中符合过滤条件的子文件或子目录。指定到文件不是目录会返回null。

#### 例子

例子1：指定一个文件夹，获取该文件夹下面所有java文件。

这边以根目录开头操作(当然你可以直接指定到文件夹目录下操作更方便)：

File file = new File("d:\\");

File[] files = file.listFiles();第一步 获取盘目录下所有的子文件和目录

for (File f : files) {

if(f.getName().equals("a")) { 第二步：判断目录名称是否为a(建议还要判断是否为目录)

File[] listFiles = f.listFiles(); 第三步 获取a目录下的所有子文件和目录

for (File f2 : listFiles) {

if(f2.getName().endsWith("java")) { 第四步 使用字符串方法判断后缀名为java的文件

System.out.println(f2.getName());

}

}

}

}

需求2： 指定一个文件夹，然后列出文件夹下面的所有子文件与文件夹，但是格式要如下:

文件：

文件名1

文件名2

文件名3

..

文件夹：

文件夹名1

文件夹名2

文件夹名3

....

File dir = new File("d:\\");

File[] files = dir.listFiles();//获取到所有的子文件

System.out.println("文件：");

for(File fileItem : files){

if(fileItem.isFile()){

System.out.println("\t"+fileItem.getName());

}

}

System.out.println("文件夹：");

for(File fileItem : files){

if(fileItem.isDirectory()){

System.out.println("\t"+fileItem.getName());

}

}

}

### 字节流

字节流: 字节流读取得都是文件中二进制数据，读取到二进制数据不会经过任何的处理。

FileInputStream 输入流

InputStream 所有输入字节流的父类(基类) 它是抽象类

FileInputStream 读取文件数据的输入字节流

FileOutputStream 输出流

OutputStream 是所有输出字节流 的父类。 抽象类

FileOutStream 向文件输出数据的输出字节流。

拷贝一张图片

第一种 使用普通方式

//第一步 找到目标文件

File inFile = new File("F:\\美女\\1.jpg");

File destFile = new File("E:\\1.jpg");

//第二步 建立数据的输入输出通道

FileInputStream input = new FileInputStream(inFile);

FileOutputStream out = new FileOutputStream(destFile); //追加数据....

//每新创建一个FileOutputStream的时候，默认情况下FileOutputStream 的指针是指向了文件的开始的位置。 每写出一次，指向都会出现相应移动。

//第三步 建立缓冲数据，边读边写

byte[] buf = new byte[1024];

int length = 0 ; //获取数组的个数：初始为0，如第一次循环是存满的就是1024，最后一次824就是824

while((length = input.read(buf))!=-1){ //最后一次只剩下了824个字节

//参数：字节数组，数组开始位置，数组结束位置

out.write(buf,0,length); //写出很多次数据，所以就必须要追加。

}

//第四步 关闭资源 原则： 先开后关，后开先关。

input.close();

out.close();

第二种 使用缓冲流方式

//第一步 找到目标文件

File inFile = new File("F:\\美女\\1.jpg");

File outFile = new File("E:\\1.jpg");

//第二步 建立数据输入输出通道

FileInputStream fileInputStream = new FileInputStream(inFile);

FileOutputStream fileOutputStream = new FileOutputStream(outFile);

//第三步 建立缓冲输入输出流：内部都维护了8kb的字节数组

BufferedInputStream bufferedInput = new BufferedInputStream(fileInputStream);

BufferedOutputStream bufferedOutput = new BufferedOutputStream(fileOutputStream);

int content = 0;

//第四步 循环边读边写：最后返回值为-1证明已经读完

while((content = bufferedInput.read())!=-1){ // 如果使用read方法中无参数(没传入缓冲数组)，那么返回值是读取到的内容。

bufferedOutput.write(content);

}

//第5步：关闭资源

bufferedInput.close();

bufferedOutput.close();//先flush后关闭，所有循环中不用flush，不然效率很低

}

注意细节(如果忘记仔细看上面两种方法)：

read(byte): 内容是存储到缓冲数组中，返回值是存储到缓冲数组中的字节个数。

read();如果使用read方法中无参数(没传入缓冲数组)，那么返回值是读取到的内容.所以缓冲：直接读内容

记住：缓冲流都不具备读写文件的能力构造方法中需要传入FileInputStream

### 字符流

前提引申：使用字节流也可以读取txt内容

字节流之所以能够写中文是因为借助了字符串的getBytes方法对字符串进行了编码（字符---->数字）

使用字节流出现乱码的原因： 一个中文在gbk码表中默认是占两个字节，它只是一个字节一个字节的读所以不是一个完整的中文。

使用字符流的应用场景： 如果是读写字符数据的时候则使用字符流。

字符流会把读取到的二进制的数据进行对应 的编码与解码工作。 字符流 = 字节流 + 编码(解码)

FileReader 输入字符流 一个字符的读

reader 输入字符流的基类。 抽象类

FileReader 向文件数据数据的输入字符流

//找到目标文件

File file = new File("F:\\1208project\\day21\\src\\day21\\Demo1.java");

// 建立数据的输入通道

FileReader fileReader = new FileReader(file);

//建立缓冲字符数组读取文件数据

char[] buf = new char[1024];

int length = 0 ;

while((length = fileReader.read(buf))!=-1){

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

}

FileWriter 输出字符流

reader 输出字符流的基类。 抽象类

FileReader 向文件数据数据的输出字符流

要注意的事项：

1. 使用FileWriter写数据的时候，它爷爷writer内部是维护了一个1024个字符数组的，写数据的时候会先写入到它内部维护的字符数组中，如果需要把数据真正写到硬盘上：需要调用flush或者是close方法或者是填满了内部的字符数组。

2. 使用FileWriter的时候，如果目标文件不存在，那么会自动创建目标文件。

3．使用FileWriter的时候， 如果目标文件已经存在了，那么默认情况会先清空文件中的数据，然后再写入数据 ， 如果需要在原来的基础上追加数据，需要使用“new FileWriter(File , boolean)”的构造方法，第二参数为true。

//找到目标文件

File file = new File("F:\\a.txt");

//建立数据输出通道

FileWriter fileWriter = new FileWriter(file,true);

//准备数据，把数据写出

String data = "今天天气非常好！！";

fileWriter.write(data); //字符流具备解码的功能。

//刷新字符流

// fileWriter.flush();

//关闭资源

fileWriter.close();

拷贝txt文件

第一种 使用缓冲字符流操作 要注意”\r\n”问题:\r:13,\n:10

1. 输入输出内部维护了8196Kb的char数组
2. 缓冲输入流：读一行字符数据readLine();方法
3. 缓冲输出流：写入分隔符”\r\n”的功能newLine()方法

//第一步 创建缓冲字符输出、输入流

BufferedReader bufferedReader = new BufferedReader(new FileReader("F:\\Test.txt"));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter("E:\\Test.txt"));

//第二步 使用String接收内容，因为内部维护了char数组， String默认值是null

String line=null;

while((line = bufferedReader.readLine())!=null){ //读取一行

bufferedWriter.write(line);

}

//第三步 关闭流

bufferedWriter.close();

bufferedReader.close();

第二种 普通方式( 一般都不会用：和字节流普通方式差不多)

//找到目标文件

File file = new File("F:\\1208project\\day21\\src\\day21\\Demo1.java");

File file2 = new File("F:\\1208project\\day21\\src\\day21\\Demo1.java");

// 建立数据的输入通道

FileReader fileReader = new FileReader(file);

FileWrite fileWrite= new FileWrite (file2);

//建立缓冲字符数组读取文件数据

char[] buf = new char[1024];

int length = 0 ;

while((length = fileReader.read(buf))!=-1){

fileWrite .write(buf,0,length));

}

注意问题：

记住：缓冲流都不具备读写文件的能力

拓展FileReader和FileWriter

装饰者设计模式：增强一个类的功能，而且还可以让这些装饰类互相装饰。

使用步骤：继承BufferedReader缓冲类，重写readLine的方法

需求1： 编写一个类拓展BufferedReader的功能， 增强readLine方法返回 的字符串带有行号。

class BufferedLineNum extends BufferedReader{

//第一步 默认行号为1

int count = 1 ;

//第二步 创建带参构造函数

public BufferedLineNum(Reader in) {

super(in);

}

第二步 重写readLine()

public String readLine() throws IOException {

String line = super.readLine(); //首先获得父类的方法返回内容

if(line ==null){

return null;

}

line = count+" "+ line; //在内容最前面加上行号即可

count++;

return line;

}

}

调用：

File file = new File("F:\\Demo1.java");

//建立数据的输入通道

FileReader fileReader = new FileReader(file);

//建立带行号的缓冲输入字符流

BufferedLineNum bufferedLineNum = new BufferedLineNum(fileReader);

String line = null;

while((line = bufferedQuto.readLine())!=null){

System.out.println(line);

}

这里只写一个列子：详细请看2015基础33期第21天的课程

### 打印流

打印流（printStream）打印任意类型的数据，而且打印数据之前都会先把数据转换成字符串再进行打印。

指定标准的输出流显示

//打印流可以打印任何类型的数据，而且打印数据之前都会先把数据转换成字符串再进行打印。

File file = new File("F:\\a.txt");

//第一步 创建一个打印流

PrintStream printStream = new PrintStream(file);

//第二步 重新设置了标准的输出流对象(默认标准的输出流就是向控制台输出的)

System.setOut(printStream);

System.out.println("哈哈，猜猜我在哪里！！");

收集异常日志信息

File logFile = new File("F:\\2015年1月8日.log");

//第一步 创建一个打印流：追加内容需要用到字节流的构造方法设为true

PrintStream logPrintStream = new PrintStream( new FileOutputStream(logFile,true) );

try{

int c = 4/0;

System.out.println("c="+c);

int[] arr = null;

System.out.println(arr.length);

}catch(Exception e){

e.printStackTrace(logPrintStream); //第二步 把异常信息发送到打印流中

}

### 转换流

输入字节流的转换流：InputStreamReader 是字节流通向字符流的桥

输出字节流的转换流： OutputStreamWriter可以把输出字节流转换成输出字符流 。

转换流的作用：

1.获取到的是一个字节流需要转换字符流使用，这时候就可以使用转换流。 字节流----> 字符流

(1) 使用字符缓冲流接收字节输入流来写入数据的操作

InputStream in = System.in; //获取了标准的输入流。

//System.out.println("读到 的字符："+ (char)in.read()); //read()一次只能读取一个字节。

//第一步 需要把字节流转换成字符流。

InputStreamReader inputStreamReader = new InputStreamReader(in);

//第二步 使用字符流的缓冲类

BufferedReader bufferedReader = new BufferedReader(inputStreamReader);

String line = null;

while((line = bufferedReader.readLine())!=null){

System.out.println("内容："+ line);

}

2. 使用转换流可以指定编码表进行读写文件。

(1)使用输入字节流的转换流指定码表进行读取文件数据

public static void readTest2() throws IOException{

File file = new File("F:\\a.txt");

FileInputStream fileInputStream = new FileInputStream(file);

//创建字节流的转换流并且指定码表进行读取

InputStreamReader inputStreamReader = new InputStreamReader(fileInputStream,"utf-8");

char[] buf = new char[1024];

int length = 0;

while((length = inputStreamReader.read(buf))!=-1){

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

}

}

(2)使用输出字节流的转换流指定码表写出数据

public static void writeTest2() throws IOException{

File file = new File("F:\\a.txt");

//建立数据的输出通道

FileOutputStream fileOutputStream = new FileOutputStream(file);

//把输出字节流转换成字符流并且指定编码表。

OutputStreamWriter outputStreamWriter = new OutputStreamWriter(fileOutputStream, "utf-8");

outputStreamWriter.write("新中国好啊");

//关闭资源

outputStreamWriter.close();

}

### 对象流

对象的输入输出流 : 对象的输入输出流 主要的作用是用于写对象的信息与读取对象的信息。 对象信息一旦写到文件上那么对象的信息就可以做到持久化了

对象的输出流： ObjectOutputStream .

对象的输入流: ObjectInputStream

对象输入输出流要注意的细节：

1. 如果对象需要被写出到文件上，那么对象所属的类必须要实现Serializable接口。

2. 对象的反序列化创建对象的时候并不会调用到构造方法、

3. serialVersionUID 是用于记录class文件的版本信息的，serialVersionUID这个数字是通过一个类的类名、成员、包名、工程名算出的一个数字。

4. 使用ObjectInputStream反序列化的时候，ObjeectInputStream会先读取文件中的serialVersionUID，然后与本地的class文件的serialVersionUID

进行对比，如果这两个id不一致，那么反序列化就失败。

5. 如果序列化与反序列化的时候可能会修改类的成员，那么最好一开始就给这个类指定一个serialVersionUID，如果一类已经指定的serialVersionUID，然后在序列化与反序列化的时候，jvm都不会再自己算这个 class的serialVersionUID了。

6. 如果一个对象某个数据不想被序列化到硬盘上，可以使用关键字transient修饰。

7. 如果一个类维护了另外一个类的引用，那么另外一个类也需要实现Serializable接口。

使用步骤：

类实现Serializable接口

class User implements Serializable{

private static final long serialVersionUID = 1L;

String userName ;

String password;

transient int age; // transient 透明 不被系列化到硬盘上使用此关键字修饰

Address address ;

public User(String userName , String passwrod) {

this.userName = userName;

this.password = passwrod;

}

public User(String userName , String passwrod,int age,Address address) {

this.userName = userName;

this.password = passwrod;

this.age = age;

this.address = address;

}

@Override

public String toString() {

return "用户名："+this.userName+ " 密码："+ this.password+" 年龄："+this.age+" 地址："+this.address.city;

}

}

反系列化和系列化操作

public class Demo3 {

public static void main(String[] args) throws IOException, Exception {

writeObj();

// readObj();

}

//把文件中的对象信息读取出来-------->对象的反序列化

public static void readObj() throws IOException, ClassNotFoundException{

//第一步 找到目标文件

File file = new File("F:\\obj.txt");

//第二步 建立数据的输入通道

FileInputStream fileInputStream = new FileInputStream(file);

//第三步 建立对象的输入流对象

ObjectInputStream objectInputStream = new ObjectInputStream(fileInputStream);

// 第四步 读取对象信息

User user = (User) objectInputStream.readObject(); //创建对象肯定要依赖对象所属 的class文件。

System.out.println("对象的信息："+ user);

}

//定义方法把对象的信息写到硬盘上------>对象的序列化。

public static void writeObj() throws IOException{

//第一步 把user对象的信息持久化存储。

Address address = new Address("中国","广州");

User user = new User("admin","123",15,address);

//第二步 找到目标文件

File file = new File("F:\\obj.txt");

//第三步 建立数据输出流对象

FileOutputStream fileOutputStream = new FileOutputStream(file);

//第四步 建立对象的输出流对象

ObjectOutputStream objectOutputStream = new ObjectOutputStream(fileOutputStream);

//第五步 把对象写出

objectOutputStream.writeObject(user);

//第六步 关闭资源

objectOutputStream.close();

}

### properties

Properties（配置文件类）: 主要用于生产配置文件与读取配置文件的信息。

Properties要注意的细节：

1. 如果配置文件的信息一旦使用了中文，那么在使用store方法生成配置文件的时候只能使用字符流解决，如果使用字节流生成配置文件的话，默认使用的是iso8859-1码表进行编码存储，这时候会出现乱码。

2. 如果Properties中的内容发生了变化，一定要重新使用Properties生成配置文件，否则配置文件信息不会发生变化。

读取和修改

//第一步 创建Properties对象

Properties properties = new Properties();

//第二步 加载配置文件信息到Properties中

properties.load(new FileReader("F:\\persons.properties"));

//第三步 获取所有的键值信息：遍历

Set<Entry<Object, Object>> entrys = properties.entrySet();

for(Entry<Object, Object> entry :entrys){

System.out.println("键："+ entry.getKey() +" 值："+ entry.getValue());

}

//修改密码需要重新生成配置文件

//把修改后的Properties再生成一个配置文件

properties.setProperty("狗娃", "007");

properties.store(new FileWriter("F:\\persons.properties"), "hehe");

保存

//保存配置文件文件的信息。

public static void creatProperties() throws IOException{

//创建Properties

Properties properties = new Properties();

properties.setProperty("狗娃", "123");

properties.setProperty("狗剩","234");

properties.setProperty("铁蛋","345");

//使用Properties生产配置文件。

//第一个参数是一个输出流对象，第二参数是使用一个字符串描述这个配置文件的信息。

properties.store(new FileWriter("F:\\persons.properties"), "hehe");

### 系列流(合并流)

序列流，对多个流进行合并。SequenceInputStream

把三个文件合并成一个文件：需要用到Vector集合

public static void merge3() throws IOException{

//第一步 找到目标文件和被合并的位置

File file1 = new File("F:\\a.txt");

File file2 = new File("F:\\b.txt");

File file3 = new File("F:\\c.txt");

File file4 = new File("F:\\d.txt"); //被合并最后的路径名

//第二步 建立对应的输出流对象

FileOutputStream fileOutputStream = new FileOutputStream(file4);

//第三步 建立对应的输入流对象

FileInputStream fileInputStream1 = new FileInputStream(file1);

FileInputStream fileInputStream2 = new FileInputStream(file2);

FileInputStream fileInputStream3 = new FileInputStream(file3);

//第四步 创建序列流对象：Vector相当于arrayList，区别它是同步安全的，效率低

Vector<FileInputStream> vector = new Vector<FileInputStream>();

vector.add(fileInputStream1);

vector.add(fileInputStream2);

vector.add(fileInputStream3);

Enumeration<FileInputStream> e = vector.elements(); //java早期迭代器

SequenceInputStream sequenceInputStream = new SequenceInputStream(e);//把Vector放进合并流中

//第五步：读取文件数据

byte[] buf = new byte[1024];

int length = 0;

while((length = sequenceInputStream.read(buf))!=-1){ //使用合并流进行写数据

fileOutputStream.write(buf,0,length); //输出流把数据写到指定文件

}

//关闭资源

sequenceInputStream.close();

fileOutputStream.close();

}

合并只有两个文件的操作：直接使用构造方法带两个参数的

public static void merge2() throws IOException{

//找到目标文件

File inFile1 = new File("F:\\a.txt");

File inFile2 = new File("F:\\b.txt");

File outFile = new File("F:\\c.txt");

//建立数据的输入输出通道

FileOutputStream fileOutputStream = new FileOutputStream(outFile);

FileInputStream fileInputStream1 = new FileInputStream(inFile1);

FileInputStream fileInputStream2 = new FileInputStream(inFile2);

//建立序列流对象

SequenceInputStream inputStream = new SequenceInputStream(fileInputStream1,fileInputStream2);

byte[] buf = new byte[1024];

int length = 0 ;

while((length = inputStream.read(buf))!=-1){

fileOutputStream.write(buf,0,length);

}

//关闭资源

inputStream.close();

fileOutputStream.close();

}

#### 切割MP3

public static void cutFile() throws IOException{

File file = new File("F:\\美女\\1.mp3");

//目标文件夹

File dir = new File("F:\\music");

//建立数据的输入通道

FileInputStream fileInputStream = new FileInputStream(file);

//建立缓冲数组读取

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

int length = 0;

for(int i = 0 ; (length = fileInputStream.read(buf))!=-1 ; i++){

FileOutputStream fileOutputStream = new FileOutputStream(new File(dir,"part"+i+".mp3"));

fileOutputStream.write(buf,0,length);

fileOutputStream.close();

}

//关闭资源

fileInputStream.close();

}

#### 合并MP3

public static void mergeFlile() throws IOException{

//找到目标文件

File dir = new File("F:\\music");

//通过目标文件夹找到所有的MP3文件，然后把所有的MP3文件添加到vector中。

Vector<FileInputStream> vector = new Vector<FileInputStream>();

File[] files = dir.listFiles();

for(File file : files){

if(file.getName().endsWith(".mp3")){

vector.add(new FileInputStream(file));

}

}

//通过Vector获取迭代器

Enumeration<FileInputStream> e = vector.elements();

//创建序列流

SequenceInputStream inputStream = new SequenceInputStream(e);

//建立文件的输出通道

FileOutputStream fileOutputStream = new FileOutputStream("F:\\合并.mp3");

//建立缓冲数组读取文件

byte[] buf = new byte[1024];

int length = 0 ;

while((length = inputStream.read(buf))!=-1){

fileOutputStream.write(buf,0,length);

}

//关闭资源

fileOutputStream.close();

inputStream.close();

}

## 执行exe文本

通过notepad.exe 来打开后缀txt文件需要用到Runtime类

File[] root= File.listRoots(); //获得所有系统盘符

Runtime runtime= Runtime.getRuntime(); //用来启动windos系统exe文件类

for (File f : root) {

if("E:\\".equals(f.getPath())) { //进入E盘

String[] tr=f.list(); //获得e盘根目录所有的文件和文件夹

for(String f1 : tr) {

if("自己".equals(f1)) {//进入”自己”文件夹

File f2=new File(f.getPath()+File.separator+f1);//创建File路径：E:\自己

String [] s=f2.list();

for (String string : s) {

if("oracle.txt".equals(string)) { //判断此文件是oracle.txt

//注意 这里要打开后缀txt文件的都是通过notepad.exe 打开的，exe后面需要一个空格

runtime.exec("notepad.exe "+f.getPath()+File.separator+f1+File.separator+"oracle.txt");

# 网络编程

## UDP

### 群聊

接收端

public class ChatReceive extends Thread {

@Override

public void run() {

try {

//建立udp的服务,要监听一个端口

DatagramSocket socket = new DatagramSocket(9090);

//准备空的数据包存储数据

byte[] buf = new byte[1024];

DatagramPacket packet = new DatagramPacket(buf, buf.length);

boolean flag = true;

while(flag){

socket.receive(packet);

// packet.getAddress() 获取对方数据 包的IP地址对象。

System.out.println(packet.getAddress().getHostAddress()+"说:"+new String(buf,0,packet.getLength()));

}

//关闭资源

socket.close();

}catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

发送端

public class ChatSender extends Thread {

@Override

public void run() {

try {

//建立udp的服务

DatagramSocket socket = new DatagramSocket();

//准备数据，把数据封装到数据包中发送

BufferedReader keyReader = new BufferedReader(new InputStreamReader(System.in));

String line = null;

DatagramPacket packet = null;

while((line = keyReader.readLine())!=null){

//把数据封装 到数据数据包中，然后发送数据。

packet = new DatagramPacket(line.getBytes(), line.getBytes().length, InetAddress.getByName("192.168.15.255"), 9090);

//把数据发送出去

socket.send(packet);

}

//关闭 资源

socket.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

## TCP

TCP通讯协议特点：

1. tcp是基于IO流进行数据 的传输 的，面向连接。

2. tcp进行数据传输的时候是没有大小限制的。

3. tcp是面向连接，通过三次握手的机制保证数据的完整性。 可靠协议。

4. tcp是面向连接的，所以速度慢。

5. tcp是区分客户端与服务端 的。

比如： 打电话、 QQ\feiQ的文件传输、 迅雷下载....

tcp的客户端使用步骤：

1. 建立tcp的客户端服务。

2. 获取到对应的流对象。

3.写出或读取数据

4. 关闭资源。

### 群聊

客户端

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

//建立tcp的客户端服务

Socket socket = new Socket(InetAddress.getLocalHost(),9090);

//获取socket的输出流对象。

OutputStreamWriter socketOut = new OutputStreamWriter(socket.getOutputStream());

//获取socket的输入流对象

BufferedReader socketReader = new BufferedReader(new InputStreamReader(socket.getInputStream()));

//获取键盘的输入流对象，读取数据

BufferedReader keyReader = new BufferedReader(new InputStreamReader(System.in));

String line = null;

//不断的读取键盘录入的数据，然后把数据写出

while((line = keyReader.readLine())!=null){

socketOut.write(line+"\r\n");

//刷新

socketOut.flush();

//读取服务端回送的数据

line = socketReader.readLine();

System.out.println("服务端回送的数据是："+line);

}

//关闭资源

socket.close();

}

服务端

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

//建立tcp的服务端

ServerSocket serverSocket = new ServerSocket(9090);

//接受客户端的连接，产生一个SOcket

Socket socket = serverSocket.accept();

//获取到Socket的输入流对象

BufferedReader socketReader = new BufferedReader(new InputStreamReader(socket.getInputStream()));

//获取到Socket输出流对象

OutputStreamWriter socketOut = new OutputStreamWriter(socket.getOutputStream());

//获取键盘的输入流对象

BufferedReader keyReader = new BufferedReader(new InputStreamReader(System.in));

//读取客户端的数据

String line = null;

while((line = socketReader.readLine())!=null){

System.out.println("服务端接收到的数据："+ line);

System.out.println("请输入回送给客户端的数据：");

line = keyReader.readLine();

socketOut.write(line+"\r\n");

socketOut.flush();

}

//关闭资源

serverSocket.close();

}

模拟Tomcat服务器

1. 开启多线程
2. 不断的接受客户端的请求
3. 可以获取到客户的ip：String address = socket.getInetAddress().getHostAddress();
4. 使用set集合统计访问次数
5. 访问：开启服务后直接在浏览器上访问栏通过ip访问

public class TomcatDemo extends Thread {

private Socket socket;

//集合保存访问者ip

static Set<String> set = new HashSet<String>();

public TomcatDemo(Socket socket){

this.socket = socket;

}

public void run() {

try {

//第四步 获取socket的输出流对象

OutputStream outputStream = socket.getOutputStream();

//第五步 把数据写到浏览器上

outputStream.write("<html><head><title>aaa</title></head><body>你好啊浏览器</body></html>".getBytes());

String address = socket.getInetAddress().getHostAddress(); //第六步 获取ip

if(set.add(address)) { //第七步 把访问者的ip存到set中

System.out.println("IP:"+address+"当前访问的次数是："+set.size());

}

socket.close();

} catch (IOException e) {

e.printStackTrace();

}

}

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

//第一步：建立tcp的服务端

ServerSocket serverSocket = new ServerSocket(9090);

//第二步 不断的接受客户端的连接

while(true){

Socket socket = serverSocket.accept();

new TomcatDemo(socket).start();//第三步 开启多线程

}

}

}

# 动态代理

静态代理书写步骤:

1.要求被装饰者和装饰者实现同一个接口或者继承同一个类

2.在装饰者中要有被装饰者的引用

3.对需要加强的方法进行加强

4.对不需要加强的方法调用原来的方法

## Proxy

作用：

1、不改源码的情况下，执行此方法前增加代码，或执行之后增加代码等

2、不改源码的情况下，改造方法的实现效果，如之前改造colse效果

匿名InvocationHandler接口操作

步骤一：被代理类必须是实现接口的类:A类实现类与AInterfaces接口类

public class A implements AInterfaces {

public void method() {

System.*out*.println("A方法");

}

}

public interface AInterfaces {

public abstract void method();

}

步骤二：创建代理类：P类，创建方法：dd();内部使用Proxy.newProxyInstance()方法得到被改造后的对象：Object

**public** **class** P {

**private** AInterfaces a=**new** A();

创建方法

**public** Object dd() {

参数：类装载器，目标对象所有接口,匿名代理对象InvocationHandler，

**return** Proxy.*newProxyInstance*(a.getClass().getClassLoader(),a.getClass().getInterfaces(),

**new** InvocationHandler() {

代理对象，当前执行的方法，当前方法的参数

**public** Object invoke(Object proxy, Method method, Object[] args) **throws** Throwable {

**if**("method".equals(method.getName())) {

//在调用本类method方法前加一句话，（如果是全改此方法返回值不能是：method.invoke(ai, args);）

System.***err***.println("加你一句话进去");

**return** method.invoke(ai, args); ai这里必须是声明的对象，不可以A.class

}

//不改造的方法直接通过

**return** method.invoke(ai, args);

步骤三：得到改造后的对象：但还是Object，必须用接口类进行强转，再调用方法

ProxyTest pt=**new** ProxyTest();

Object o=pt.dd();//得到改造后的object对象

AInterfaces a=(AInterfaces) o;//必须用接口类进行强转和接收

a.method();

注意事项：

1、Proxy.newProxyInstance()：创建方式

要先进行创建对象AInterfaces a=new A();或A a=new A();用声明的名称

Proxy.newProxyInstance(类的装载,类的接口 , InvocationHandler(代理类接口))

2、method.invoke(ai, args)：不想改造的方法直接这样通过。ai 参数不能A.class

参数：1、声明的对象。2、方法中自带的Object[] args（返回值参数）

如要全新改造：被改造的方法就不要return method.invoke(ai, args)： 需return新方法的值，没有返回值就null代替.

3、调用方法：因为得到了被改造后的Object对象，强转要必须注意：

强转操作：必须用接口类进行强转和接收

4、然后进行调用要调用的方法，实现改造

传入对象操作

步骤一：被代理类必须是实现接口的类:A类实现类与AInterfaces接口类

public class A implements AInterfaces {

public void method() {

System.*out*.println("A方法");

}

}

public interface AInterfaces {

public abstract void method();

}

步骤二：创建代理类：P类，实现接口：InvocationHandler，创建instance方法：内部使用方法Proxy.newProxyInstance方法：有返回值Object（得到改造后的对象）

public class B implements InvocationHandler{

private Object obj;

public Object instance(Object o) {

//重要这步是把传入的对象传到成员变量，让下面方法要用到，因为调用操作是调用此方法的方法

this. obj =o;

//参数：目标对象A的类装载、获取被代理的所有接口，代理对象变成this

return Proxy.newProxyInstance(obj.getClass().getClassLoader(), obj.getClass().getInterfaces(), this);

}

//代理对象、当前调用的方法，当前传入的参数，

public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {

Object result=null;

System.out.println("执行方法前");

if("MyName".equals(method.getName())) {

System.out.println("MyName被我改造了");

//必须要有一个返回值，没有就null代替

return "真的啊";

}else {

result = method.invoke(obj, args);//可以说获得返回值

return result;

}

步骤三：调用操作：创建代理对象，调用代理对象的方法，得到改造后被代理Object对象的返回值，进行强转，调用方法

public static void main(String[] args) {

B b=new B();

//获得被改造后的被代理对象

Object obj=b.instance(new A());

// 使用Ainterfaces接口类进行强制转换和接收，

AInterfaces a=(AInterfaces) obj;

String c=a.MyName();

System.out.println(c);

a.MyName2();

## **Cglib代理**

此框架不要求目标类实现接口（被代理对象）Spring框架jar包已经有包含在内

第一步：导入jir包

经测试导入此cglib-nodep-3.1jar包就可以了

cglib-nodep-3.1.jar使用nodep包不需要关联asm的jar包,jar包内部包含asm的类.

cglib -3.1.jar 使用此jar包需要关联asm的jar包,否则运行时报错. asm-4.2.jar

第二步：创建被代理类：A

public class A {

public void add() {

System.out.println("此方法是A类add方法");

}

}

第三步：创建实现接口MethodInterceptor的代理类：B类，注意：invokeSuper

public class B implements MethodInterceptor{

public Object intercept(Object obj, Method method, Object[] params, MethodProxy proxy) throws Throwable {

if("add".equals(method.getName())) {

System.out.println("执行被增强的方法前执行");

//只想在改造方法的前后加代码的操作

Object o=null;

o=proxy.invokeSuper(obj, params);//注意此地方的方法和proxy不同哦

System.out.println("执行完被增强的方法后执行");

return o;

}

//不想改造的方法通过

return proxy.invokeSuper(obj, params);

}

第四步：调用操作：创建对象Enhancer，对象中的方法setSuperclass传入被代理对象，对象中的方法setCallback【回收】代理对象，对象中的方法create()得到改造后的对象

public static void main(String[] args) {

//生成对象Enhancer【增强子；强化剂；增加者】

Enhancer en = new Enhancer();

//传入被增强方法的类

en.setSuperclass(A.class);

//传入实现接口MethodInterceptor的类Callback

en.setCallback(new B());

A a=(A) en.create();

a.add();

}

# 正则表达式

类在：java.util.regex. Pattern

matches(String regex) 告知此字符串是否匹配给定的正则表达式。

String aa="45679";

System.out.println(aa.matches("[1-9]\\d{4,10}")?"合法":"不合法");

[1-9]\\d{4,10}意思是：开头只能在1到9取值，长度在5到9之间才是true

\\：双斜杠是因为单\在字符串有特殊意义，要进行转义

[1-9]：先写这个是让它开头第一个只能填1-9

\d：是正则中的预定义字符类：取值0-9，

{4,10}：是多少位

常用预定义字符类：

. 任何字符（与行结束符可能匹配也可能不匹配）

\d 数字：[0-9] , \D非数字 [^0-9]

\w 单词字符和[a-zA-Z\_0-9] 一样：大小写a到z还有下划线：\_和0到9数字，

\W 非单词字符：[^\w] 除了\w的其它都是true

Greedy 数量词 注意：任何预定义字符没有加上数量词之前都只能匹配一个字符

System.out.println("+:一次或者多次："+"1".matches("\\d+"));//返回true

System.out.println("?:一次或者零："+"".matches("\\d?"));//返回true

System.out.println("\*:零或者多次："+"123".matches("\\d\*"));//返回true

System.out.println("{3} 指定3位："+"456".matches("\\d{3}"));//返回true

System.out.println("{3,} 最少3位："+"567".matches("\\d{3,}"));//返回true

System.out.println("{3,9}指定3到9："+"123".matches("\\d{3,9}"));//返回true

范围词：注意：范围词里面不管内容多长，也一定要配合数量词，不然只能匹配一个字符

[abc] a、b 或 c（简单类）

[^abc] 任何字符，除了 a、b 或 c（否定）

[a-zA-Z] a 到 z 或 A 到 Z，两头的字母包括在内（范围）

正则表达式主要用于操作字符串的规则，正则表达式对字符串的操作主要有以下几种应用

匹配

matches()方法：手机号码：”[1][34578]\\d{9}”:11位数，前面占两位后面配合数量词9

没使用正则表达，就比较内容

切割

字符串的Split()也可以使用正则表达式的

把空格去掉：因为它有好几个空格，使用到正则数量词：+一次或者多次

String vv="你 好 啊 哈 哈";

String[] v=vv.split(" +");//空格再配合正则的数量词：+：至少一次或多次

System.out.println(Arrays.toString(v));//[你, 好, 啊, 哈, 哈]

切割掉重叠的字

String vv="大家明明天天天玩得得得得得开心点点";

()分组：分组是为了让正则的内容被复用起来，组号是从1开始的

((A)(B(C))):第一组：(A)(B(C))，第2组：(A)，第3组：(B(C)，第4组是(C)

String[] v=vv.split("(.)\\1+");//所匹配的内容：至少有相同或者多个：如大大成立

System.out.println(Arrays.toString(v));//[大家, , 玩, 开心]

替换

replaceAll(String regex, String replacement)：参数：regex：正则表达式，2要替换的内容

老师介绍了论坛打广告，输入电话替换成\*\*\*\*\*

replaceAll(“[1][34578]\\d{9}”,“让你打广告”)

替换重复的内容

String vv="大家明明天天天玩得得得得得开心点点";

//如果需要在正则的外部引用组的内容，那么是使用"$组号"

System.out.println(vv.replaceAll("(.)\\1+", "$1"));//大家明天玩得开心点

查找

这个放一放没有看，使用到类java.util.regex. Pattern对象，在18天就单一个视频

# 设计模式

## 单例模式

单例模式 ：饿汉模式、懒汉模式、fina修饰

一个类Class只有一个实例(对象)存在。 使用Singleton【单例】的好处还在于可以节省内存，因为它限制了实例的个数，有利于Java垃圾回收（garbage collection）

单例类确保自己只有一个实例、必须自己创建自己的实例、必须为其他对象提供唯一的实例

饿汉模式 对于初始化慢，占用资源多的重量级对象使用，首选饿汉模式：程序运行时快

因为单例的实例被声明成 static和 final变量了，在第一次加载类到内存中时就会初始化，所以创建实例本身是线程安全的。

public class A{

private static final A a= new A(); //类加载时就初始化

private A(){} //私有化构造方法

public static A getA(){ return a; }

懒汉模式

public class A {

private volatile static A a; //声明成 volatile 保证编译器不进行优化

private A (){} //私有化构造方法

public static A getA() {

if (a == null) {

synchronized (A.class) {

if (a == null) {

a = new A();}}}

return instance;

}

fina修饰 就不能改变值了(如下面A类就不能：A a=null;)

public class A{

public A{};

private static final A a= new A(); //类加载时就初始化

## 装饰模式

装饰模式和代理对象很像：也需要一个接口，创建类实现同样的接口，改造自己想要改造的方法，不想改造的调用回被装饰类的方法（当然前提要进行new被装饰的对象）

缺点：接口如果方法很多方法的话，每一个方法都需要进行调用操作累死你；

public interface AInterfaces {

public void MyName();

public void MyName2();

}

第一步 创建目标类：被装饰的类：A

public class A implements AInterfaces {

public void MyName() {

System.out.println("打印第1句话");

}

public void MyName2() {

System.out.println("打印第2句话");

}

}

第二步 自己创建一个类继承或实现A类同样的接口类： B类

public class B implements AInterfaces{

private AInterfaces aInterfaces; //第一步 在本类维护实现接口的类

//通过构造方法进行传入接口类，当然也可以传入被改造类，这为了通用才传接口类

public B(AInterfaces aInterfaces) { //第二步 在调用中：创建对象传入接口的实现类对象

this.aInterfaces=aInterfaces;

}

public void MyName() { //第三步 要改造的方法进行重写即可

System.out.println("我要改造你");

}

//第四步 不想改造的方法：在方法中调A类的方法

public void MyName2() {

aInterfaces.MyName2(); //调A类的方法

}

--------------------------调用---------------------------------

public static void main(String[] args) {

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

a.MyName();

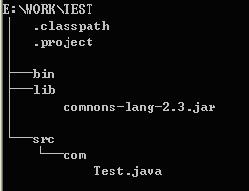
a.MyName2();

}

# DOS环境下导包操作

DOS环境下含包并引用第三方jar的java程序的编译及运行

1、程序目录机构

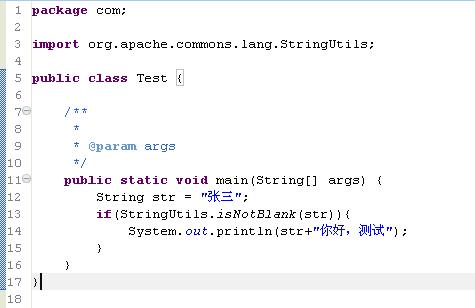


bin：class文件生成目录

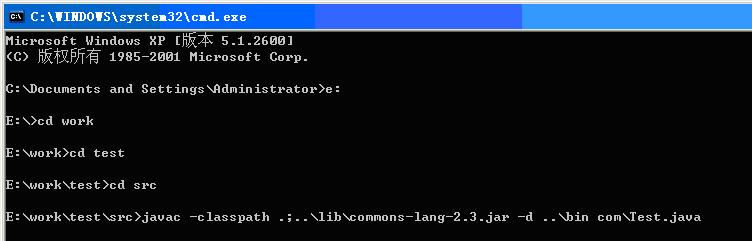
lib：第三方jar包目录

src：源程序文件目录

2、程序代码：

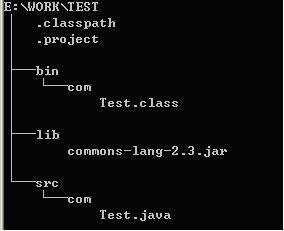


3、程序编译

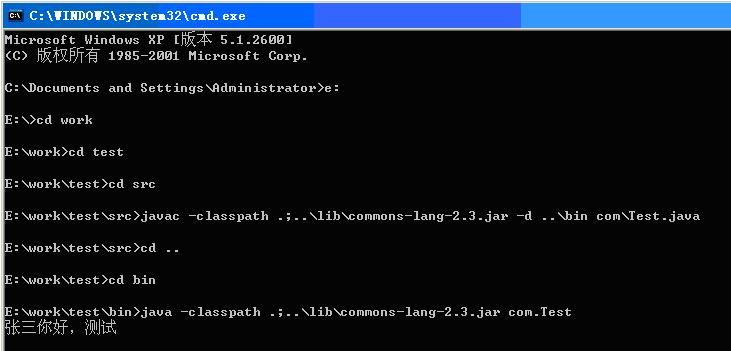


javac –classpath .;..\lib\commons-lang-2.3.jar -d ..\bin com\Test.java

4、编译后的程序目录：



5、运行程序及结果显示



java -classpath .;..\lib\commons-lang-2.3.jar com.Test

引用多个jar包

假设有个程序的启动方法在test.java里

运行Java程序是java test，但是如果要引用其他jar包，网上大多数的方法都是java -classpath xx.jar test

但是jar很多的话，就要一个一个写上去，非常麻烦。而且jar包可能经常变

现在用以下一句就能解决问题，假设jar包都放在lib文件夹里

java -Djava.ext.dirs=lib test

注意：要JDK1.6以上才可以，如果不怕麻烦也可以这样写：

javac -classpath c:\lib\log4j.jar;D:\lib\logging.jar src\org\myproject\\*.java

还可以指定输出目录，如输出到bin（bin目录要预告建立好）

javac -classpath c:\lib\log4j.jar;D:\lib\logging.jar -d bin src\org\myproject\\*.java

# 打jar包

需要使用到jdk的开发工具（jar.exe）.

jar的用法：

使用格式： jar cvf jar文件的名字 class文件或者是文件夹

第一种：直接通过class文件操作 jar cvf 新名字.jar 目标文件.class

命令：Jar cvf ww.jar a.class b.class 然后生成了：ww.jar

第二种：把上面两个文件放到qq文件夹下

命令：Jar cvf qq.jar qq 然后生成了qq.jar

通过包打开qq.jar此文件清单：

qq文件夹下有：a.class b.class a.class是主方法入口

META-INF文件夹：MANIFEST.MF

打开MANIFEST.MF：

最后面添加内容：Main-Class: c.a

注意：

1、英文冒号和c之间有空格的，少一个空格和多一个空格都不行

2、A.class中的package c;要和文件夹路径一致

打jar包要注意的事项：

1. 一个程序打完jar之后必须要在清单文件上指定入口类：格式Main-Class: 包名.类名

2. jar包双击运行仅对于图形化界面的程序起作用，对控制台的程序不起作用。

jar文件的作用：

1. 方便用户快速运行一个项目。

2. 提供工具类以jar包的形式给别人使用。

如果使用jar包里面的类必须要先设置classpath路径。

jre = jvm+ 核心类库

package qq;

import javax.swing.\*;

class Demo9 {

public static void main(String[] args) {

System.out.println("QQ程序..");

JFrame frame = new JFrame("QQ程序");

frame.setSize(400,500);

frame.setVisible(true); //设置窗口可见。

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

}

}

## Eclipse

Eclipse进行打jar包：选中工程包右键Export搜索jar选中JAR file第一个路径目标文件按两次Next进入选择Main class选择主线程操作然后完成

貌似不能进行jar中的文件进行修改，可以使用jar资源文件，路径操作：

InputStream in = UserDao.class.getClassLoader().get

# 类加载器

JDK中提供的类加载器：

\* 引导/系统类加载器 ：Java\jre7\lib\rt.jar

\* 扩展类加载器 ：Java\jre7\lib\ext\\*.jar

\* 应用类加载器 ：自定义的类,类路径下的所有class文件.

类加载

我们编写的.java文件，jvm会将java变成.class文件，该文件要想运行，必须加载到你内存中，然后会生成一个对象Class对象

类加载器层次结构：

引导类加载器：用来加载rt.jar,最核心的jia包：全部类都来源这里

扩展类加载器: 用来加载ext目录下的所有jar包：我们用不上



应用类加载器：自己编写的类

实验：

应用类加载器：自己编写的类

ClassLoader ap=Test.class.getClassLoader();

System.out.println(ap);//打印：sun.misc.Launcher$AppClassLoader@2d8eef25

引导类加载器：

ClassLoader st=String.class.getClassLoader();

System.out.println(st);//打印null，因为使用c语言编写的类，可以这样认为打印null的加载器类是引导加载器类

扩展类加载器:在exp的jar包找到类sun.net.spi.nameservice.dns.DNSNameService测试

ClassLoader ext=DNSNameService.class.getClassLoader();

System.out.println(ext);//打印：sun.misc.Launcher$ExtClassLoader@9cboF4

全盘负责委托机制：

class文件由应用类加载器获得到,没有加载,向上一层委托向扩展类加载器委托,向上一层进行委托委托给引导类加载器.引导类加载器查看class哪些它负责,将自己负责的这个class进行加载.不是其负责的就向下传递扩展类加载器.扩展类加载器查看是否是其管理的class,如果是加载,不是就再向下到应用类加载器.

当一个类运行的时候，有可能有其它的类:

应用类加载器询问扩展类加载器：你加载过这类吗？

扩展类加载器不搭理，直接问（引导类加载器）：你加载过这些类吗？

引导类加载器：我查查，有一个是我负责的，我加载

扩展类加载器：接下来我来查，有几个是我负责，我加载，还有几个类我已经加载完成了，你可以直接使用

应用类加载器：收到了 剩下的我来

# 性能优化

1、尽量指定类、方法的final修饰符

带有final修饰符的类是不可派生的。在Java核心API中，有许多应用final的例子，例如java.lang.String，整个类都是final的。为类指定final修饰符可以让类不可以被继承，为方法指定final修饰符可以让方法不可以被重写。如果指定了一个类为final，则该类所有的方法都是final的。Java编译器会寻找机会内联所有的final方法，内联对于提升Java运行效率作用重大，具体参见Java运行期优化。此举能够使性能平均提高50%。

2、尽量重用对象

特别是String对象的使用，出现字符串连接时应该使用StringBuilder/StringBuffer代替。由于Java虚拟机不仅要花时间生成对象，以后可能还需要花时间对这些对象进行垃圾回收和处理，因此，生成过多的对象将会给程序的性能带来很大的影响。

3、尽可能使用局部变量

调用方法时传递的参数以及在调用中创建的临时变量都保存在栈中速度较快，其他变量，如静态变量、实例变量等，都在堆中创建，速度较慢。另外，栈中创建的变量，随着方法的运行结束，这些内容就没了，不需要额外的垃圾回收。

4、及时关闭流

Java编程过程中，进行数据库连接、I/O流操作时务必小心，在使用完毕后，及时关闭以释放资源。因为对这些大对象的操作会造成系统大的开销，稍有不慎，将会导致严重的后果。

5、尽量减少对变量的重复计算

明确一个概念，对方法的调用，即使方法中只有一句语句，也是有消耗的，包括创建栈帧、调用方法时保护现场、调用方法完毕时恢复现场等。所以例如下面的操作：

for (int i = 0; i < list.size; i++){...}

建议替换为： 意思就是先算出共几条数据，再判断

for (int i = 0, int length = list.size; i < length; i++){...}

这样，在list.size很大的时候，就减少了很多的消耗

6、尽量采用懒加载的策略，即在需要的时候才创建

String str = "aaa";if (i == 1){list.add(str);}

建议替换为：先判断再 进行后续的创建对象，定义变量操作

if (i == 1){String str = "aaa";list.add(str);}

7、慎用异常 异常对性能不利，抛出异常首先要创建一个新的对象

Throwable接口的构造函数调用名为fillInStackTrace的本地同步方法，fillInStackTrace方法检查堆栈，收集调用跟踪信息。只要有异常被抛出，Java虚拟机就必须调整调用堆栈，因为在处理过程中创建了一个新的对象。异常只能用于错误处理，不应该用来控制程序流程。

8、循环内不要不断创建对象引用

for (int i = 1; i <= count; i++){Object obj = new Object;}

这种做法会导致内存中有count份Object对象引用存在，count很大的话，就耗费内存了，建议为改为：

Object obj = null;for (int i = 0; i <= count; i++) {obj = new Object; }

这样的话，内存中只有一份Object对象引用，每次new Object的时候，Object对象引用指向不同的Object罢了，但是内存中只有一份，这样就大大节省了内存空间了。

9、关于数组、集合的使用

①基于效率和类型检查的考虑，尽可能使数组array，无法确定数组大小才使用ArrayList

②优先使用HashMap、ArrayList、StringBuilder，

除非线程安全需要，否则不推荐使用Hashtable、Vector、StringBuffer，后三者由于使用同步机制而导致了性能开销

③不要将数组声明为public static final

因为这毫无意义，这样只是定义了引用为static final，数组的内容还是可以随意改变的，将数组声明为public更是一个安全漏洞，这意味着这个数组可以被外部类所改变。

④实现RandomAccess接口的集合比如ArrayList，应当使用最普通的for循环而不是foreach循环来遍历

⑤公用的集合类中不使用的数据一定要及时remove掉

如果一个集合类是公用的（也就是说不是方法里面的属性），那么这个集合里面的元素是不会自动释放的，因为始终有引用指向它们。所以，如果公用集合里面的某些数据不使用而不去remove掉它们，那么将会造成这个公用集合不断增大，使得系统有内存泄露的隐患。

⑥使用'system.arraycopy ()'代替通过来循环复制数组

10、尽量在合适的场合使用单例

使用单例可以减轻加载的负担、缩短加载的时间、提高加载的效率，但并不是所有地方都适用于单例，简单来说，单例主要适用于以下三个方面：

（1）控制资源的使用，通过线程同步来控制资源的并发访问

（2）控制实例的产生，以达到节约资源的目的

（3）控制数据的共享，在不建立直接关联的条件下，让多个不相关的进程或线程之间实现通信

11 不要创建一些不使用的对象，不要导入一些不使用的类

这毫无意义，如果代码中出现”The value of the local variable i is not used”、”The import java.util is never used”，那么请删除这些无用的内容

12 静态的变量、常量

①常量声明为static final，并以大写命名

这样在编译期间就可以把这些内容放入常量池中，避免运行期间计算生成常量的值。另外，将常量的名字以大写命名也可以方便区分出常量与变量

②尽量避免随意使用静态变量

当某个对象被定义为static的变量所引用，那么gc通常是不会回收这个对象所占有的堆内存的，如：

public class A{ private static B b = new B;}

此时静态变量b的生命周期与A类相同，如果A类不被卸载，那么引用B指向的B对象会常驻内存，直到程序终止

13、使用同步代码块替代同步方法

这点在多线程模块中的synchronized锁方法块一文中已经讲得很清楚了，除非能确定一整个方法都是需要进行同步的，否则尽量使用同步代码块，避免对那些不需要进行同步的代码也进行了同步，影响了代码执行效率。

# 生命周期

类的加载

当程序要使用某个类时，如果该类还未被加载到内存中，则系统会通过加载，连接，初始化三步来实现对这个类进行初始化。

加载

就是指将class文件读入内存，并为之创建一个Class对象。

任何类被使用时系统都会建立一个Class对象。

连接

验证 是否有正确的内部结构，并和其他类协调一致

准备 负责为类的静态成员分配内存，并设置默认初始化值

解析 将类的二进制数据中的符号引用替换为直接引用

初始化 就是我们以前讲过的初始化步骤

类的初始化有5种情况

1. 创建对象：new
2. 调用类中的静态方法
3. 为类或接口中的静态域赋值
4. 访问类或接口中声明的静态域

注意：继承关系：B类继承A类，A类有静态C成员、B类调C成员，初始化是A类，B不初始化

1. 调用Class类和反射API

类初始化顺序

(1)静态变量、静态代码块初始化

(2)构造函数

(3)自定义构造函数

生命周期

类的对象：7个阶段

1. 创建阶段(Created)：类的初始化

2. 应用阶段(In Use)

3. 不可见阶段(Invisible) ：对象的变量提示报错

4. 不可达阶段(Unreachable)：静态变量或县城或JNI等强引用持有者

5. 收集阶段(Collected)：垃圾回收器发现该对象已经处于第4阶段并且已做好重新分配准备

6. 终结阶段：等待垃圾回收器回收

7. 对象空间的重新分配(De-allocated)：

局部变量

离开大括号

成员变量

随着对象的消失而消失

静态 static是修饰①静态代码块②成员变量③方法

其方法或者代码块的生命周期横跨整个应用的生命周期。

加载：java虚拟机在加载类的过程中为静态变量分配内存。

类变量：static变量在内存中只有一个，存放在方法区，属于类变量，被所有实例所共享

销毁：类被卸载时，静态变量被销毁，并释放内存空间。static变量的生命周期取决于类的生命周期

结论：

(1)上面若final修饰的只是生命周期内不能更改值

(2)想要用static存一个变量，使得下次程序运行时还能使用上次的值是不可行的。因为静态变量生命周期虽然长（就是类的生命周期），但是当程序执行完，也就是该类的所有对象都已经被回收，或者加载类的ClassLoader已经被回收，那么该类就会从jvm的方法区卸载，即生命期终止。

更进一步来说，static变量终究是存在jvm的内存中的，jvm下次重新运行时，肯定会清空里边上次运行的内容，包括方法区、常量区的内容。

要实现某些变量在程序多次运行时都可以读取，那么必须要将变量存下来，即存到本地文件中。常用的数据存取格式：XML、JSON、Propertities类（类似map的键值对）等

# 集合|数组|properties

数组

1）arraycopy(Object src, int srcPos, Object dest, int destPos, int length)

从指定源数组中复制一个数组，复制从指定的位置开始，到目标数组的指定位置结束。

int[] i = {1,4,5,7,8};

int[] s=new int[4];

//源数组，索引起始位置，目标数组，索引起始位置，总共要复制几个索引

System.arraycopy(i, 1, s, 0, 3);

System.out.println(Arrays.toString(s));//打印：[4, 5, 7, 8]

Properties 继承Hashtable(和HashMap区别：线程是安全的)

需求： 使用properties实现本软件只能 运行三次，超过了三次之后就提示购买正版，退jvm.

第一步 判断配置文件是否存在

File file = new File("F:\\count.properties");

if(!file.exists()){

file.createNewFile();//如果配置文件不存在，则创建该配置文件

}

第二步 创建Properties对象 读取文件。

Properties properties = new Properties();

properties.load(new FileInputStream(file)); //把配置文件的信息加载到properties中

第三步 创建FileOutputStream对象重新生成配置文件对象：一定要先把目标文件加载到内存中，创建对象代码会把文件清空

注意：此FileOutputStream对象会先进行清空目标文件，构造方法中加true也不行，会内容追加多次

FileOutputStream fileOutputStream = new FileOutputStream(file);

int count = 0;

第四步 读取配置文件的运行次数

//在第一次进行取值的时候文件是空的，取到的值是null，没有也不会报错

String value = properties.getProperty("count");

if(value!=null){

count = Integer.parseInt(value);//转换成int类型

}

第五步 判断使用的次数是否已经达到了三次 并推出jvm：System.exit(0);

if(count==3){

System.out.println("你已经超出了试用次数，请购买正版软件！！");

properties.store(fileOutputStream,"runtime"); //保存参数：流，描述文件

System.exit(0); //退出java虚拟机

}

System.out.println("你已经使用了本软件第"+ (count++)+"次");

properties.setProperty("count",count+""); //修改key中的值

第六步 重新保存文件

properties.store(fileOutputStream,"runtime");

}

# Object

protected Object clone() 创建并返回此对象的一个副本。

使用用途：当你需要使用此对象中的数据，但是又不希望修改此对象的数据时，

浅克隆：复制被克隆的对象，等于拷贝对象内容，两个内存地址不一样的对象，如果被克隆对象维护了另一个对象，是克隆不了维护中的对象的，

此方法的修饰符是protected（只有他自身或者继承他的类可以用）

第一步：被克隆的对象中需重写Object中的clone的方法：把权限修饰符protected改为public即可(因为是自己要被别人克隆，权限要公共)：

public Object clone() throws CloneNotSupportedException {}

(重写父类的方法可以权限子类比父类大，但不能比父类小)

第二步：被克隆的对象要实现接口：标识符 Cloneable类：

public class User implements Cloneable{}

第三步：调用：如有异常，被克隆对象没实现Cloneable：CloneNotSupportedException

public static void main(String[] args) {

User u= new User();

u.setId(555);

u.setName("温文亮");

User p= new User();

try {

p=(User) u.clone();//返回的是Object

} catch (CloneNotSupportedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

p.setName("你好");

cachCode和equalse的运用 快捷键生成对比你指定的变量如都一致是重复的

没有重写hashCode方法的类，返回32位对象在JVM中的地址；Long类重写了hashCode方法，返回计算出的hashCode数值

ComHashcode a = new ComHashcode(); a.hashCode()//打印870919696

Long num = new Long(8); num1.hashCode()//打印 8

集合的SetHash、MapHash(键)存储机制都需要调用这两个方法：

第一步：先调用hashCode方法查询到哈希值的位置:没有值直接放入，有值才会调用equals方法

第二步：调用equals方法：拿参数进行对比如：返回true不让存储，返回false在同一个哈希值存储两个数据

**简单例子**：实体类User：根据id进行存储至set集合，id一致的话不让存储进hashSet集合中

**实体类**

public class User {

private String s;

private int id;

public User(int id,String s) {this.id=id; this.s=s; } //构造方法

public String getS() {return s; }

public void setS(String s) {this.s = s; }

public int getId() {return id; }

public void setId(int id) {this.id = id; } 两个属性getset封装方法

第一步 重写hashCode方法，固定让id的值进行查找内存中的哈希表位置

public int hashCode() {

return this.id;

}

第二步 重写equals方法，传进来的值和本类的id进行对比

public boolean equals(Object obj) {//此参数传进来的值是你添加的对象set.add(new User(1,"福福"));

User u = (User) obj;

return this.id==u.getId(); //id对比

}

**Main方法调用**

Set<User> set=new HashSet<User>();

//哈希表特点**桶式结构（1个坑多个萝卜）**：算出来的哈希码值一样，进行equals对比不一致就两个元素放在同一个地方

set.add(new User(1,"福福"));

set.add(new User(1,"阿福"));

System.out.println(set.size()); 打印：1，（没有重写则打印2： 没有重写hashCode方法都是拿内存地址进行对比的）

# 其它

1、Integer类内部维护了一个静态的数组，该数组内部存储了-128~127这些数据，当我们创建一个Interger对象的时候，如果数据是-128~127范围之内，那么会直接从数组中获取，而不会创建新的Integer对象，目的是为了节省内存：因为这个数据是常用的：享元模式

Integer i = 127;

Integer i1=127;

System.out.println(i==i1);//打印：false 如果为127的话就打印为true

2、JDk工具类：数组Arrays， 集合Collections

获取最大：int i=Collections.max(list); 获取最小：int c=Collections.min(list); 排序Collections.sort(list);

3、This就是本类对象的引用，A a=new A() a就是对象也就是this 静态不能有this

4、int超出范围会变成负数：时间毫秒都建议是long型

int i =1000\*60\*60\*30\*50\*10000\*1000\*100000;

System.out.println(i);//打印：-446693376

5、properties文件 value存中文问题 需要转义成Unicode（万国码）

p.properties文件内容：y=\u4F60\u597D\u554A\uFF0C\u66FE\u4E3D\u65B9

\u: Unicode编码的意思

中文转换Unicode：JDk有自带的命令 :cmd输入”native2ascii” 回车 然后输入中文名后再回车就有了

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

Properties p = new Properties();

p.load(Test.class.getResourceAsStream("/p.properties"));

System.out.println(p.getProperty("y"));//打印：你好啊，曾丽方

}

6增强for循环底层还是迭代器：不可以修改数据，普通for：可以修改数据

Map<String, String[]> v=request.getParameterMap();

Collection<String[]> l=v.values();

for(String[] e:l) {

//增强for循环是不能修改的，只能进行赋值后进行重新保存；

for(int i=0;i<e.length;i++) {

e[i]= new String(e[i].getBytes("iso8859-1"),"utf-8");//直接这样就可以保存数据了，不用再进行set

}

}要明白引用问题：栈中的是固定值无法修改，再次修改是拷贝，而堆:需要内存地址(有一个或多个指向同一个内存地址）

7 StringWriter—字符串字符流是writer装饰类

作用：把流中的字符串写进此字符串输出流中

场景：现只知道自定义标签获取标签体内的内容需要使用到

/\*PageContext pc= (PageContext) getJspContext();

//FileWriter是指向文件的输出流，StringWriter是指向字符串的输出流

StringWriter sw = new StringWriter();

//把它写进此(字符串输出流)中来获取标签体的内容

getJspBody().invoke(sw);

String str=sw.toString();//获取到内容

str=str.toUpperCase();//字符串转大写方法

pc.getOut().write(str);//输出到页面