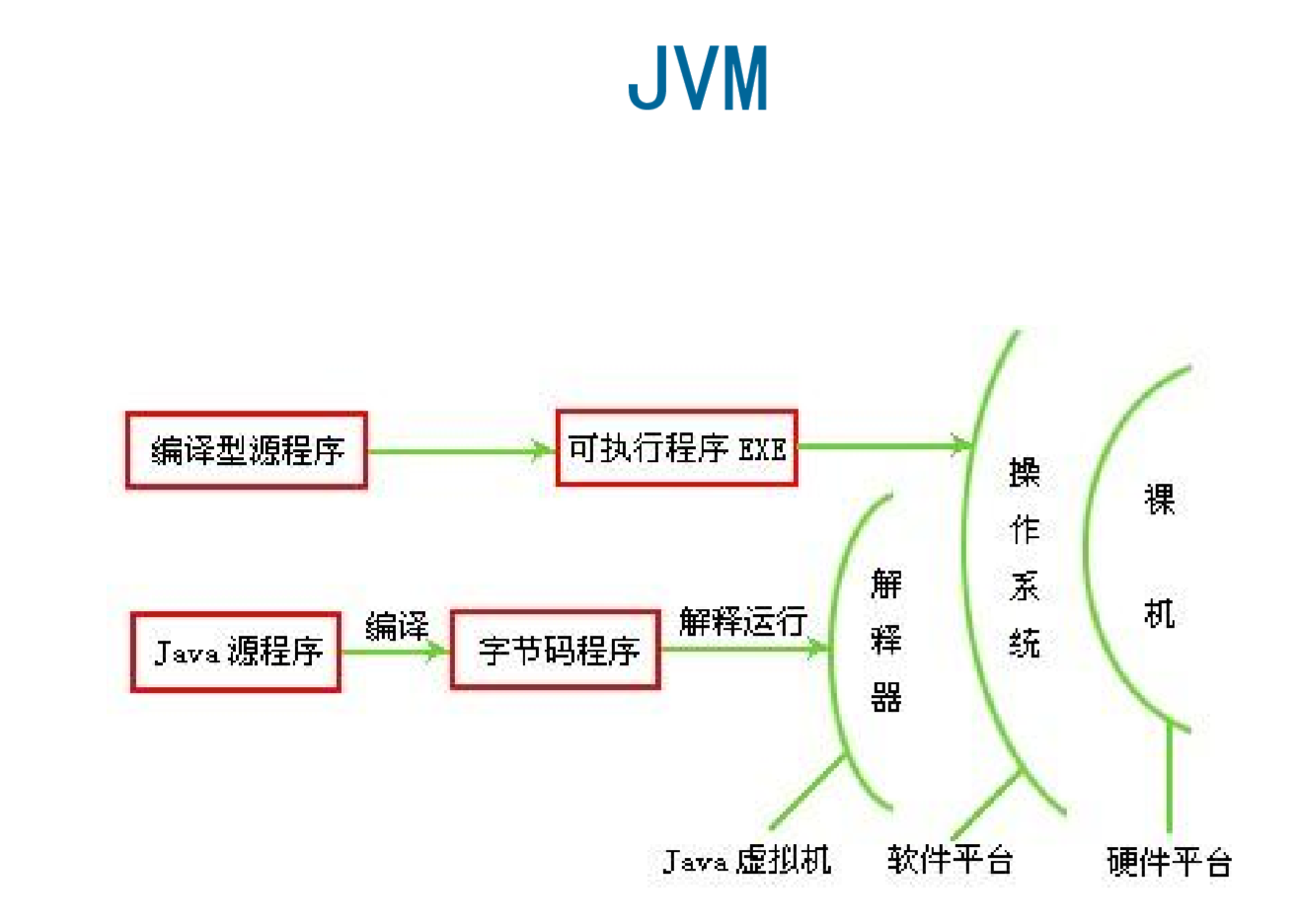
Java语言特点

Java语言特点：开源跨平台

Java是跨平台的，JVM不是跨平台的。不同的操作系统上有不同的JVM。

JDK（Java Development Kit） Java开发工具包

JDK是提供给开发人员使用的，其中包含了Java的开发工具包，也包括了JRE。

其中的开发工具：编译工具（Javac.exe）打包工具（Jar.exe）等

JRE（Java Runtime Environment） Java运行时环境

包括Java虚拟机（Java Virtual Machine）和Java程序所需的核心类库等，如果想要运行一个开发好的Java程序，计算机中只需要安装一个JRE即可。

JDK给开发人员用的，JRE给使用的人用的。

环境变量配置

path配置是为了在任意目录下都能使用JAVAC程序。

Path环境变量里面的记录是可执行性文件，如.exe文件，对可执行文件现在当前路径去找，如果没有找到就去path环境变量中配置的路径去找

classpath配置是为了在任意目录下执行class文件。

classpath环境变量记录的是JAVA类的运行文件（.class文件）所在的目录

Java规范

Java关键字都是小写字母

Java标识符：英文字母大小写，数字字符，$和\_

注意事项：不能以数字开头，不能是关键字，Java严格区分大小写

包：包的命名全部小写

单级：yan

多级：cn.com.yan

类或者接口：

一个单词：单词的首字母必须大写（举例：Student，Dog）

多个单词：每个单词的首字母必须大写（举例：HelloWorld，UserName）

方法和变量：

一个单词：单词的首字母小写（举例：main,age）

多个单词：从第二个单词开始，每个单词的首字母大写（举例：studentAge，showAllNames）

常量：

一个单词：全部大写（举例：PI）

多个单词：每个字母都大写，用下划线隔开（举例：STUDENT\_MAX\_AGE）

注释：单行：// 多行：/\* \*/ 文档注释：/\*\* \*/

常量：在程序执行过程中，其值不发生改变的量

常量分类： A：字面值常量 B：自定义常量

字面值常量：

A：字符串常量 用双引号括起来的内容。举例：“hello”,‘world’

B：整数常量 所有整数。举例：100,200

C：小数常量 所有的小数

D：字符常量 用单引号括起来的内容

E：布尔常量

F：空常量

进制

1byte=8bit 1K=1024byte 1M=1024K 1G=1024M 1T=1024G

以0b开头是2进制，0开头是8进制，十进制默认正常显示，0x开头是16进制

System.out.println(100);//输出100

System.out.println(0b100);//输出4

System.out.println(0100);//输出64

System.out.println(0x100);//输出256

二进制到十进制：

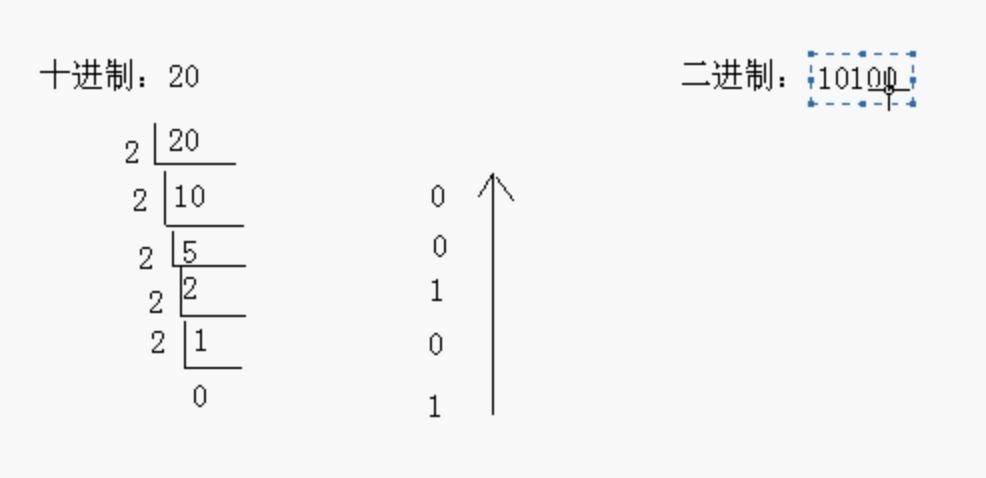
1010100=64+16+4=84

十进制到二进制：（其他进制同理）

100=64+32+4=1100100

X进制到Y进制的转换--用十进制作为桥梁

十进制转到其他进制：



除基取余，直到商为0，余数反转

原码，反码，补码（day2\_15）

在计算机内，有符号数有3种表示法：原码，反码，和补码

所有数据的运算都是采用补码进行的

原码：就是二进制定点表示法，即最高位为符号位，0表示正数，1表示负数，其余位表示数值的大小

反码：正数的反码与其原码相同；负数的反码是对其原码逐位取反，但符号位除外。

补码：正数的补码与其原码相同；负数的补码是在其反码的末位加1

数据类型

数据类型：Java是一种强数据类型的语言。针对每一种数据都定义了明确的数据类型。

数据类型分类：

A：基本数据类型

B：引用数据类型（类，接口，数组）

基本数据类型：4类，8种

A：整数 byte（1字节）short（2字节）int（4字节）long（8字节）

B：浮点数 float（4字节）double（8字节）

C：字符 char（2字节）

D：布尔 boolean（1字节）

注意事项：整数默认是int类型，浮点数默认是double类型。

定义long类型，末尾需要加L。定义float类型，末尾需要加F。

变量的作用域：

变量定义在哪一级大括号中，那个大括号的范围就是这个变量的作用域。相同的作用域中不能定义两个同名变量。

变量的初始化：

没有初始化值的变量不能直接使用。只需要在使用前赋值，不一定要在定义的时候赋值。

一般来说，要求参与运算的数据类型必须一致。

数据类型参与计算：

Boolean类型不能转换为其他的数据类型

如果在一个运算中，只要有大的类型存在，小的类型自动转换成大的类型，再参与计算

默认/隐形转换（从小到大）：byte.short.char-int-long-float-double

如果有byte.short.char他们参与运算，他们会首先转换为int类型，再参与运算

例如：byte a = 3 ; int b = 4 ; byte c = a + b ;

编译报错，因为运算时a和b均为int类型，要将Int赋值给byte，可能损失精度

数据的强制转换：从大的数据类型转换成小的数据类型，可能会有精度的损失，一般不建议

关系运算符：

&：按位与，二进制补码运算，二进制位上两个都为1，结果为1，最后求值

|：按位或，二进制补码运算，二进制位上有一个为1，则结果为1

^：按位异或，二进制补码运算，二进制位上，符号相同为0，符号不同为1

~：按位取反，二进制补码运算，二进制位上，按位取反

在定义long或者float类型变量的时候要加L或者F

证书默认是int类型，浮点数默认是double

byte,short在定义的时候，他们接收的其实是一个int类型的值

这个是自己做了一个数据监测的，如果不在它们的范围内，就会报错

byte值的问题:byte值的范围：-128 ~ 127

byte b1 = 127 ;

byte b2 = (byte)128; //-128

byte b3 = (byte)129; //-127

byte b4 = (byte)130; //-126

128：10000000

-128：10000000（这里的1既是符号位，也是数值位）

数据类型转换之默认转换

byte,short,char – int – long – float – double

解释：只要byte short char 参与运算，首先会转换成int类型，有long转换成long以此类推。

long是8个字节，float是4个字节。为啥Long在前？

因为它们底层的存储结构不同；float表示的数据范围比long要大

char可以存储中文汉字，char占用2个字节，Java采用Unicode字符集

算数运算符

++,-- 运算符的使用：对变量进行自增1或者自减1

单独使用（最常用）：x++ 和 ++x 没有区别

只有变量可以自增自减，常量不行

参与运算：a++ 先取值，再自增 ++a 先自增，再取值

**int** x = 4 ;

**int** y = (x++)+(++x)+(x\*10);

System.***out***.println(y);//输出y=70

参与计算时，第一个（x++）取的值为4

System.***out***.println(‘a’);//输出字符a

System.***out***.println(‘a’+1);//参与运算的时候，char转换成int，所以输出98

赋值运算符：

int x = 10 ;

int a,b; a = b = 10 ;

逻辑运算符：一般用于连接布尔类型的值或表达式

& 逻辑与：有false则false

| 逻辑或：有true则true

^ 逻辑异或：相同为false，不同为true

！逻辑非：非false则true，非true则false

&& 和 & 的区别，|| 和 | 的区别：

最终结果一样，但是&&和||具有短路效果

x++ = 3 & y ++ = 4 ;//如果左边不成立，右边也执行

x++ = 3 && y ++ = 4 ;//如果左边不成立，则右边不执行

位运算符（操作的是补码）：

做位运算时，首先要把数据转换成二进制，操作的是补码

<< 左移，空位补0,，被移除的高位丢弃（左边的数据乘以2的移动次幂）

>> 右移，被移位的二进制最高位是0，右移后，空缺位补0；最高位是1，最高位补1.

>>> 无符号右移 无论最高位是0还是1，左边补齐0。

& 与运算：有0则0 ，同1为1

| 或运算：有1则1，无1为0

^ 异或运算：相同则0，不同则1

~ 按位取反运算符：0变1,1变0

&和|：当两边是数据的时候，做的是位运算，当两边是布尔值的时候，做逻辑运算。

^的特点：一个数据对另一个数据异或两次，该数本身不变。

三目运算符：

(关系表达式)?表达式1:表达式2;

关系表达式需要提供一个布尔类型。否则报错

如果条件为true，就把表达式1作为结果;

如果条件为false，就把表达式2作为结果;

键盘录入的基本格式：

A:导入包:import java.util.Scanner ;

B:创建键盘录入对象：Scanner sc = new Scanner(System.in);

C:通过对象获取数据：int x = sc.nextInt();

流程控制语句

顺序结构：从上往下依次执行

选择结构：if语句，switch语句

if语句有三种结构

第一种格式：

if(结果为boolean的关系表达式){

语句体；

}

if语句的注意事项：

A:比较表达式无论简单还是复杂，结果必须是boolean类型

B:if的语句体如果是一条语句，大括号可以省略，如果是多条语句，则不能省略，建议永远不要省略。

C:if语句后的分号要注意，不要乱使用

第二种格式：

if(关系表达式){

语句体1;

}else{

语句体2;

}

if的第二种格式有时候也可以用三目运算符实现，这种情况下建议用三目运算符，因为简单

三目运算都可以用if改，但是if未必都可以用三目运算改

第三种格式：

if(关系表达式1){

语句体1;

}else if(关系表达式2){

语句体2;

}

……

else{

语句体n+1;

}

if语句适合做什么事情呢

格式1：适合做单个判断

格式2：适合做两个判断

格式3：适合做多个判断

选择结构：switch语句格式：

swtich(表达式){

case 值1:

语句体1;

break;

case 值2:

语句体2;

break;

…

default:

语句体n+1;

break;

}

表达式：这个地方的取值是有限定的

byte,short,int,char

JDK5以后可以是枚举

JDK7以后可以是字符串

case:后面跟的是要和表达式比较的值

语句体：要执行的代码

break:表示中断，结束的意思，可以控制switch语句的结束

default:当所有的值都和表达式不匹配的时候，就执行default控制的语句

switch语句注意事项：

先找case，再运行default，就算default在前面，也先找case

case后面只能跟常量，不能是变量；多个case后面的值不能相同

default可以省略，但是不建议省略

break可以省略，但要灵活使用

switch结束条件：遇到break结束，或者程序结尾结束

if语句和switch语句的区别？

if：针对结果是boolean类型的判断；针对一个范围判断；针对几个常量的判断

switch：针对几个常量的判断

循环结构：多次执行

for循环语句

while循环语句

do…while循环语句

while循环语句格式：

基本格式

while(判断条件语句){

循环体语句;

}

扩展格式

初始化语句：

while(判断条件语句){

循环体语句;

控制条件语句;

}

while循环和for循环的区别

使用区别：for循环结束之后不能再访问循环变量，但是while可以

理解：不知道具体循环次数的循环用while循环

do…while循环的基本格式：

do{

循环体语句;

}while(判断条件语句);

扩展格式：

初始化语句;

do{

循环体语句;

控制条件语句;

}while(判断条件语句);

循环语句的区别：

do…while循环只要要执行一次循环体

for,while循环必须先判断条件是否成立,然后决定是否执行循环体

死循环的注意事项，注意控制条件语句

两种死循环格式：

while(true){…}

for(;;){…}

跳转控制语句

break:中断

countinue:继续

return:返回

break:中断 使用场景：

1、在选择结构switch语句中

2、在循环语句中跳出循环

注意：离开以上的两个场景，在其他地方使用是没有意义的

在循环中使用：

A：跳出单层循环

B：跳出多层循环（需要给循环取个标签名），用的很少

wc:for(){

nc:for(){

//break nc;即可跳出内存循环

break wc;//跳出外层循环

}

}

countinue:继续 使用场景

在循环中使用，其他地方使用无意义

跳出一次循环，但是会继续下一次循环。而break会跳出整个循环

return:返回 使用场景

结束整个方法

方法

完成特定功能的代码块，在其他语言中，方法也称为函数

方法个是：

修饰符 返回值类型 方法名（参数列表）{

方法体语句；

return;

}

参数：实际参数，形式参数

要想写好一个方法，就必须明确两个东西

A：返回值类型：结果的数据类型

B：参数列表：要传递几个参数以及每个参数的数据类型

方法写在类里面

方法的执行特点是：不调用不执行

main方法是虚拟机调用，其他方法需要自行调用

调用方式：有明确返回值的时候

方式1：单独调用不进行操作一般来说没有意义，所以不推荐

方式2：输出调用，但是不够好，因为可能需要进一步操作

方式3：赋值调用。推荐

方法的注意事项：

方法不调用不执行

方法与方法是平级关系，不能嵌套定义（要在类中定义，不可在方法中固定义）

参数列表之间参数用逗号隔开

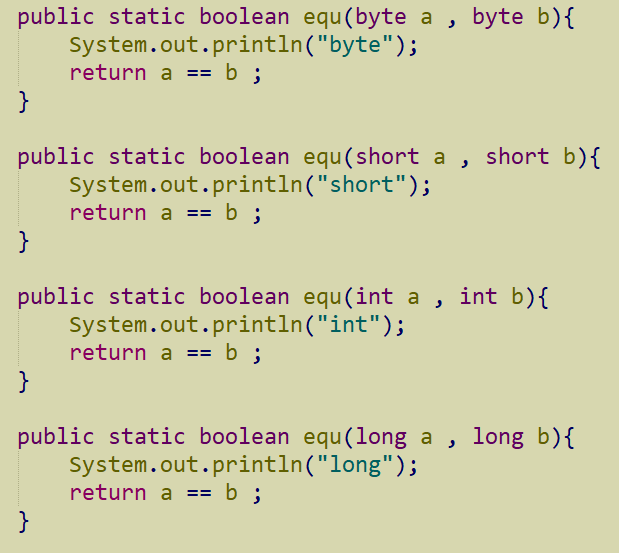
方法调用的时候不用再传递数据类型

如果方法有明确的返回值，一定要有return返回一个值

方法重载：

在同一个类中，方法名相同，参数列表不同的两个方法构成方法重载

与返回值和修饰符无关，只看方法名和参数列表，在调用时，虚拟机会根据参数列表不同来自动调用相应的函数



如上图，在调用时，会自动匹配数据类型相同的函数

数组

数组是存储同一数据类型的多个元素的集合也可以看成是一个容器

数组可以存储基本数据类型，也可以存储引用数据类型

数组定义：

格式1：数据类型[] 数组名; int[] arr;

格式2：数据类型 数组名[]; int arr[];

两种定义格式，常用第一种

数组初始化：

Java中的数组必须先初始化，然后才能使用，初始化就是为数组中的元素分配内存空间，并为每个数组元素赋值

动态初始化：初始化时只指定数组长度，由系统为数组分配初始值

静态初始化：初始化时指定每个数组元素的初始值，由系统决定长度

动态初始化：数据类型[] 数组名 = new 数据类型[数组长度]

int[] arr = new int[3];

定义了一个int类型的数组，这个数组可以存放三个int数据

Java中的内存分配

Java程序在运行是，需要在内存中分配空间。为了提高运算效率，就对空间进行了不同区域的划分，因为每一片区域都有特定的处理数据方式和内存管理模式

栈：存储局部变量

堆：存储所有new出来的东西

方法区（重点，面向对象部分详细讲解）

本地方法区（和系统相关）

寄存器（给CPU使用）

局部变量：在方法定义中或者方法声明上的变量都称为局部变量

栈内存特点：栈内存的数据用完就释放掉（脱离作用域就会释放）

堆内存的特点：

A:每一个new出来的东西都有地址值

B:每个变量都有默认值

byte,short,int,long 默认 0

float,double 默认 0.0

char 默认 ‘\u0000’

boolean 默认 false

引用类型 默认 null

C:使用完毕就会变成垃圾，但是并没有立即回收，会在垃圾回收器空闲的时候回收

数组在内存中的概述：（刘意课程Day5 17-19）

int[] arr = new int[2];

int[] arr 这个是局部变量，是存储在栈内存中的，如果直接输出arr,得到的则是一个地址值，这个地址值指向的是堆内存中对应的new出来的值的地址

new int[2]这个是new出来的，存放在堆内存中。而这个堆内存的地址则是给了栈内存中的arr

int[] arr = new int[3];

int[] arr2 = arr ;

上面的例子中，把arr给了arr2，实则是把arr的地址给了arr2，所以访问arr2[n]的时候，访问的是arr中的元素

栈内存的两个引用指向同一个堆内存空间，无论是他们谁的操作，都是针对同一个地方的操作

数组静态初始化：只给元素，不给数组长度

数据类型[] 数组名 = new 数据类型[]{元素1,元素2,…};

数据类型[] 数组名 = {元素1,元素2,…}

int[] arr = new int[]{1,2,3};

int[] arr = {1,2,3};//后面大括号里也是存放在堆内存里

数组操作的两个常见小问题：

数组越界:索引大于数组长度报错数组索引越界异常

原因：你访问了不存在的索引

空指针异常：数组已经不指向堆内存了，还用数组名去访问就报错空指针异常

数组常见练习：

数组遍历：(遍历巧用arr.length)

逐一输出；for循环输出；foreach循环遍历

int[] arr = new int[5];

for(int s : arr){

System.out.println(s);

}

数组获取最值：

二维数组

理解：二维数组就是元素为一维数组的一个数组

格式：

数据类型[][] 变量名 = new 数据类型[m][n];

int[][] arr = new int[3][2];

理解：定义了一个二维数组arr，这个二维数组有三个一维数组，名称是arr[0],arr[1],arr[2]。每个一维数组有2个元素

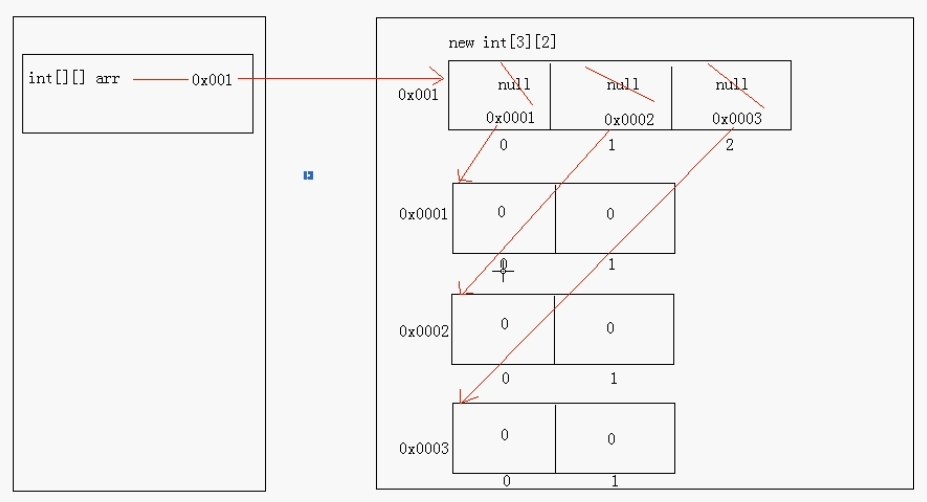
数据类型[] 变量名[] = new 数据类型[m][n];

上述也是定义一个二维数组

二维数组内存模型详解：

int[][] arr = new int[3][2];

int[][] arr存储在栈内存中

new int[3][2]存放在堆内存中，而在堆内存中，将int[3]看做是一维数组，有三个元素，每个元素中存放的是下一级一维数组的地址值

格式2：

数据类型[][] 变量名 = new 数据类型[m][];

m表示这个二维数组有多少个一维数组，这一次没有直接给出一维数组的元素个数，可以动态的给出，举例

int[][] arr = new int[3][];

arr[0] = new int[2];

arr[1] = new int[3];

arr[2] = new int[5];

格式3：

数据类型[][] 变量名 = {{元素…},{元素…},{元素…}};

int[][] arr = {{1,2,4},{5,7},{8}};

Java参数传递问题（刘意课程 day6\_10）

Java中只有值传递

基本类型传递的是数据值。实参把自己的值拷贝给形参，无论形参怎么变化，都无法改变实参本身

引用类型传递的是地址值。实参把自己所指向的地址给形参，所以形参可以操作地址所指向的堆内存中的数据

面向对象

面向对象思想：面向对象是基于面向过程的编程思想

面向过程：强调的是每一个功能的步骤

面向对象：强调的是对象，然后由对象去调用功能

面向对象的思想特点：

A：是一种更符合我们思想习惯的思想

B：可以将复杂的事情简单化

C：将我们从执行者变成了指挥者

举例：

面向过程：需要电脑-找配置-去电脑城-讨价还价-买回电脑

面向对象：需要电脑-助理去买-买回电脑

面向过程：需要吃菜-种菜-摘菜-洗菜-切菜-炒菜-吃菜

面向对象：需要吃菜-服务员上菜-吃菜

简述：模拟大象装进冰箱事件

面向过程：遇到问题，如果你首先想到的是如何去进行每一个步骤，那么就是面向过程

代码：

System.out.println(“打开冰箱”);//方法

System.out.println(“把大象放进去”);//方法

System.out.println(“关上冰箱”);//方法

遇到问题，如果首先想到的是这个事件中

A:有哪些类

B:每个类有哪些东西

C:类与类直接的关系是什么呢

这就是面向对象思想

把大象装进冰箱事件

如何分析有哪些类？名词提取法。

A:有大象类，冰箱类 Demo类

B:大象：进去 冰箱：开门关门

C:类与类直接的关系：Demo中使用大象和冰箱类的功能

代码：

冰箱类调用开门方法;//类调用方法

大象类调用进去方法;//类调用方法

冰箱类调用关门方法;//类调用方法

面向对象开发：不断的创建对象，使用对象，指挥对象做事情

面向对象设计：在管理和维护对象之间的关系

面向对象特征：封装继承多态

现实世界是如何描述一个事物的呢?

学生：----类

属性：姓名 性别 年龄----属性

行为：吃饭 睡觉 学习----方法

类是对象的抽象，对象是类的实例

一个对象的内存图（day6\_24）

成员变量和局部变量区别

A:在类中的位置不同

成员变量：类中方法外的变量

局部变量：类中方法中的变量

B:在内存中的位置不同

成员变量：存放在堆内存

局部变量：存放在栈内存

C:生命周期不同

成员变量：随着对象的创建而存在，随着对象的消失而消失

局部变量：随着方法的调用而存在，随着方法的调用完毕而消失

D:初始值不同

成员变量：有默认初始值

局部变量：没有默认初始值，必须定义赋值才能使用

注意事项：局部变量名称和成员变量名称一样，在方法中使用的时候采用的是就近原则

形式参数的问题：

基本类型：形式参数的改变不影响实际参数

引用类型：形式参数的改变直接影响实际参数

匿名对象：

就是没有名字的对象

A:正常调用

Student s = new Student();

s.study();

B:匿名对象调用（调用一次的时候使用此方法）

new Student().study;

缺点：匿名对象只适合调用一次

优点：匿名对象调用完毕就是垃圾，可以被垃圾回收器回收

匿名对象也可以作为实际参数传递

封装

引出：开发中类中的部分属性和方法不能随意访问，所以需要提供隐私保护

Java中提供不同权限的关键字进行修饰

封装：指隐藏对象的属性和实现细节，仅对外提供公共的访问方式

好处：隐藏实现细节，提供公共的访问方式

提高了代码的复用性 提高安全性

封装原则：将不需要对外提供的内容都隐藏起来

把属性隐藏，提供公共方法对其访问

private

是一个权限修饰符，可以修饰成员（成员变量和成员方法）

被private修饰的成员，只能在本类中访问

常用于修饰属性，并提供set和get方法来进行操作

this关键字

区分局部变量和成员变量

是当前类的对象引用，它就代表当前类的一个对象

注意：方法被哪个对象调用，this就代表那个对象

this关键字内存图解（day07\_09）

构造方法

给对象的数据进行初始化

格式:

方法名与类名相同

没有返回值类型，连void都没有

没有具体的返回值

构造方法的注意事项：

如果没有给出构造方法，系统将自动提供一个没有参数的构造方法

如果我们给出了构造方法，系统将不再提供任何构造方法，所以建议永远都要自己给出无参构造方法

类的初始化过程（day7\_16）

static关键字

特点：

可以修饰成员变量和，也可以修饰成员方法啊

随着类的加载而加载

优先于对象存在

属于类，被类的所有对象共享

可以通过类名调用（推荐）

static修饰的称为静态修饰的内容，一般称为类成员

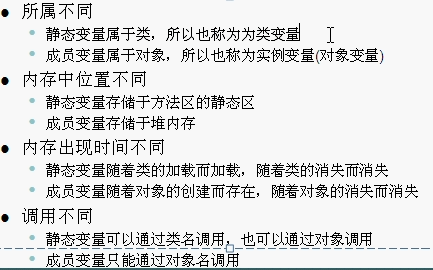
static内存图解（day7\_23）

static注意事项

在静态方法中没有this关键字

静态方法只能访问静态成员变量和静态成员方法

静态变量和成员变量的区别



main方法的格式讲解

public static void main(String[] args)

String[] args:这是一个字符串数组，用来接收键盘输入的字符串

在DOS中正常运行代码：java MainDemo

如果输入：java MainDemo hello world java

则此时hello world java这三个字符串将直接给args数组

工具类中使用静态

让构造方法私有化，外接则不能创建对象，把功能性的类用静态修饰，然后使用类名访问

工具类创建之后，一般不会给别人源码，所以需要制作一个说明书

做说明文档过程：

1.写一个工具类

2.对代码加文档注释

3.用javadoc把文档解析出来

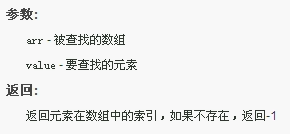
格式Javadoc -d 目录 -author -version 类名.java

目录：写一个文件夹的路径，自动创建文件夹

出错：找不可以文档化的公共或受保护的类

原因是类的权限不够，此时需要把类用public修饰

文档中注释请注意：

@author:作者

@version:版本

@param:对参数解释

@return:对返回值解释

JAVA帮助文档的使用:

1.打开帮助文档

2.点击显示，找到索引，看到输入框

3.找到需要找的类，例如Scanner

4.看包，java.lang包下的类不需要导入，其他的都需要导入

5.再简单的看看类的解释和说明和举例

6.看看该类的版本

7.看类的结构

成员变量 字段摘要

构造方法 构造方法摘要

成员方法 方法摘要

8.学习构造方法

有构造方法 就创建对象

无构造方法 成员可能都是静态的，可以直接使用

9.看成员方法

是否静态 返回值类型

看方法名 看参数列表

按照以上步骤，通过API学习Math类

代码块：在java中，使用{}括起来的代码被称为代码块根据位置和声明的不同，可以分为局部代码块；构造代码块；静态代码块和同步代码块

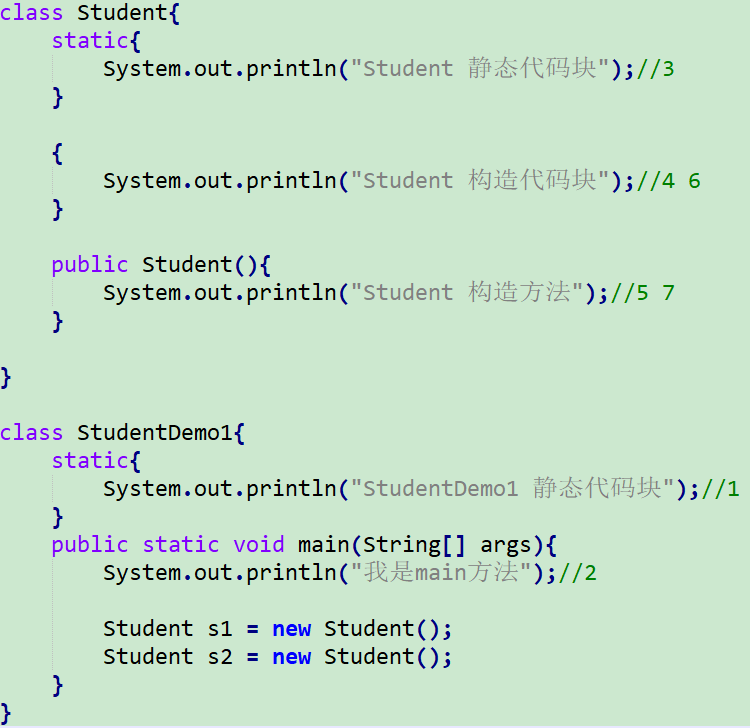
局部代码块：局部位置（在方法中）用于限定变量的生命周期，让他及早的释放，提高内存的利用率

构造代码块：在类中方法外出现，每次调用构造方法前都会先执行构造代码块

作用：把多个构造方法中的共同代码提取出来放到一起；也对对象进行初始化

静态代码块：在类中方法外出现，用static修饰，随着类的加载而加载，并且只加载一次

作用：一般是对类进行初始化

下面代码中假如不生产Student对象，则Student类不加载

继承

继承的格式：class子类 extends 父类{}

父类 基类 超类

子类 派生类

继承的好处：提高代码的复用性和维护性

继承的弊端：类的耦合性增强了

开发的原则：低耦合 高内聚

耦合：类与类的关系

内聚：自己完成某事的能力

继承的特点：

A:只支持单继承，不支持多继承

B:支持多层继承

继承的注意事项：

A:子类只能继承父类所有的非私有的成员

B:子类不能继承父类的构造方法，但是可以通过super关键字去访问父类的构造方法

C:不要为了部分功能而去继承

思考：什么时候考虑继承？

继承其实体现的是一种关系，采用假设法，如果有两个类A,B，如果他们符合A是B的一种或者B是A的一种，就可以考虑使用继承

继承中成员变量的关系：

A:子类中的成员变量和父类中的成员变量名称不一样时，正常使用

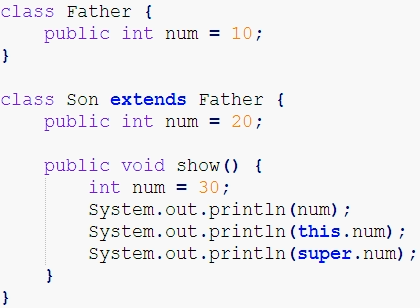
B:子类中的成员变量和父类中的成员变量名称一样时，采用就进原则，

在子类方法中访问一个变量的查找顺序：

a:在子类方法的局部范围找

b:在子类的成员范围找

c:在父类的成员范围找

以上代码，当三个范围都有相同名称的变量时，需要用this和super关键字来区别，第一个num采用就近原则，指向的是当前方法中的局部变量num，this是当前对象指向的num，也就是成员变量。super则表示的是父类中的成员变量num

this和super关键字的区别

this代表本类对象的引用

super代表父类存储空间的标识（可以理解为父类引用，可以操作父类成员）

用法：

A:调用成员变量

this.成员变量 调用本类的成员变量

super.成员变量 调用父类的成员变量

B:调用构造方法

this(…) 调用本类的构造方法

super(…) 调用父类的构造方法

C:调用成员方法

this.成员方法 调用本类的成员方法

super.成员方法 调用父类的成员方法

继承中的构造方法关系

子类中的所有构造方法默认都会访问父类中的空参的构造方法

原因：因为子类会继承父类中的数据，可能还会使用父类的数据，所以，子类初始化之前，一定要先完成父类数据的初始化

子类每一个构造方法的第一条语句默认都是:super();

注意事项：如果父类没有无参构造函数，那么子类构造方法将会报错，因为如果父类没有无参构造函数，那么子类中的super()将无法执行

解决方法：

A:给出父类无参构造方法

B:通过使用super()关键字去显示的调用父类其他构造方法

C:子类通过this()去调用本类的其他构造方法

子类中一定要有一个去访问父类的构造方法，否则父类数据就没有初始化

注意事项：this(…)或者super(…)必须出现在第一条语句上

如果不是放在第一条语句上，就可能对父类的数据进行了多次初始化

方法重写

继承中成员方法的关系

A:当子类和父类方法名不一样时，太简单

B:当子类和父类方法名一样时，先找子类，再找父类

方法重写：三同一大一小

方法名相同，形参列表相同，返回值类型相同

子类方法声明时抛出的异常类应比父类更小或相等

子类方法访问权限比父类大或相等

方法重载：两同一不同

方法名相同，同一个类，形参列表不同

方法重写的应用当子类需要父类的功能，而功能主体子类有自己特有的内容时，可以重写父类中的方法。这样既沿袭了父类的功能，又定义了子类特有的内容

方法重写的注意事项：

父类的私有方法不能被重写，因为子类无法继承

子类重写父类方法时，访问权限不能更低（最好一致）

父类静态方法，子类也必须通过静态方法进行重写

子类重写父类方法的时候，声明最好一模一样

final关键字

由于继承中方法会重写，有些时候，我们不想让子类覆盖父类的功能，只能让他使用，这时候需要把方法用final关键字修饰

final特点：

final修饰类:被final修饰的类无法被继承

final修饰方法:被final修饰的方法无法被重写

final修饰变量:该变量不能被重新赋值（因为这个变量是常量）

final面试题

多态

多态：某一事物，在不同时刻表现出来的不同状态

多态：同一种行为在不同的对象上有不同的表现形式

多态的前提：

A:要有继承或实现关系

B:要有方法重写

C:要有父类引用指向子类对象

父类 f = new 子类();

多态中的成员访问特点：父类 f = new 子类();

A:成员变量

编译看左边，运行看左边（父类 f = new 子类();）

解释：如果使用成员变量，编译的时候看父类，如果父类中没有此成员变量，会报错，运行的时候也在父类找

----属性没有多态，属性的值取决于引用类型

B:构造方法

创建子类对象的时候，访问父类的构造方法，对父类的数据初始化

C:成员方法

编译看左边，运行看右边

解释：如果有调用一个方法，编译的时候看父类，如果父类中没有此方法，会报错，运行的时候方法在子类找

----方法有多态性，方法取决于对象而不是引用类型

D:静态方法

编译看左边，运行看左边（静态和类相关，算不上重写，所以访问的还是左边）

结论：由于成员方法存在方法重写，所以他运行看右边其他没有特殊

多态的好处

A:提高代码的维护性（由继承保证）

B:提高代码的扩展性（由多态保证）案例：day9\_09

多态的弊端

不能使用子类的特有方法

弊端的解决方法：

A:另外创建一个子类对象然后调用子类方法（可行，但不可取）

B:把父类的引用强制转换成子类的引用(向下转型)

Fu f = new Zi();

Zi z = (Zi) f;

因为f是指向 Zi();这个对象，所以可以把f强制转换成Zi类型

-----------内存图解重要（day9\_13）

对象间的转型问题

向上转型:Fu f = new Zi();

向下转型:Zi z = (Zi) f;//要求该f必须是能够转换为Zi的

抽象类

抽象类

动物不应该定义为具体的东西，而且动物中的吃，睡等也不应该是具体的，我们把一个不是具体的功能称为抽象的功能，而一个类中如果有抽象的功能，该类必须是抽象类

抽象类的特点：

A:抽象类和抽象方法必须是用abstract关键字修饰的

B:抽象类不一定有抽象方法，但是有抽象方法的类必须定义为抽象类

C:抽象类不能实例化，因为它不是具体的

D:抽象类的子类

a:如果没有全部重写抽象方法，该子类还是一个抽象类

b:如果重写所有的抽象方法，这个时候子类是一个具体的类

c:子类在实现从抽象父类继承来的方法时，返回值类型，方法名，参数列表必须完全相同

抽象类的实例化其实是靠具体的子类实现的，是多态的方式

抽象类的成员特点：

成员变量：既可以是变量，也可以是常量

构造方法：有构造方法，用于子类访问父类数据的初始化

成员方法：既可以是抽象的，也可以是非抽象的

抽象类的成员方法特性：

A:抽象方法 强制要求子类做的事情

B:非抽象方法 子类继承的事情，提高代码的复用性

抽象类的几个小问题：

一个类如果没有抽象方法，可不可以定义为抽象类？如果可以，有什么意义？

答：可以;意义：不让创建对象

abstract不能和哪些关键字共存

private冲突：private类无法被继承

final冲突：final关键字修饰的方法无法被重写

static无意义：因为static可以通过类名访问，但是抽象方法没有方法体，所以无意义

接口

接口的特点：

A:用关键字interface表示

格式：interface 接口名 {}

B:类实现接口用implements表示

格式：class 类名 implements 接口名 {}

C:接口不能实例化

按照多态的方式类实例化

D:接口的实现类

a:可以是一个抽象类，但是意义不大

b:可以是一个具体类，要重写接口中的所有抽象方法（推荐）

接口的成员特点：

成员变量：只能是常量，并且是静态的

接口中的成员变量默认修饰符：public static final

构造方法：接口没有构造方法

成员方法：接口方法必须是抽象的

接口中的成员方法默认修饰符：public abstract

多态的方式：

A:具体类多态（几乎不用）

B:抽象类多态（常用）

C:接口多态（最常用）

类与类的关系：

继承关系，只能单继承，可以多层继承

类与接口的关系：

实现关系，可以单实现，也可以多实现

并且还可以在继承一个类的同时实现多个接口

接口与接口：

继承关系，可以单继承，也可以多继承

接口表示额外新增的功能

抽象类与接口的区别：

A:成员区别：

抽象类：

成员变量：可以是变量，也可以是常量

构造方法：有

成员方法：可以是抽象方法，也可以是非抽象方法

接口：

成员变量：只可以是静态常量

成员方法：无构造方法，成员方法只可以是抽象方法

B:关系区别

类与类：继承，单继承

类与接口：类实现接口，单实现，多实现

接口与接口：接口可以继承接口，单继承，多继承

C:设计理念的区别

抽象类被继承体现的是：is a的关系，抽象类中定义的是该继承体现的共性功能

接口被实现体现的是：like a的关系，接口中定义的是该继承体系的扩展功能

接口案例：day9\_30

类名作为形式参数

形式参数：

基本类型：简单，不讨论

引用类型

类：需要传递的是一个该类的对象

抽象类：需要传递的是一个该抽象类的子类对象

当一个抽象类作为形式参数的时候，编译不会报错，但是如果只是这样的话，是无法实现的，因为抽象类没有办法实例化，那么现在需要定义一个具体类来继承这个抽象类，并且利用多态的方式来实现抽象类作为形式参数

接口：需要传递一个该接口的实现类的对象

类名作为返回值类型

形式参数：

基本类型：简单，不讨论

引用类型

类：返回的是该类的一个对象

抽象类：返回的是该类的子类对象

接口：返回的是该接口的实现类对象

package关键字

package：包

A:其实就是文件夹

B:作用

a:把相同的类名放到不同的包中

b:对类进行分类管理

举例：

学生：增加，删除，修改，查询

老师：增加，删除，修改，查询

方案1：按功能分

cn.ylx5.add

AddStudent

AddTeacher

cn.ylx5.delete

cn.ylx5.update

cn.ylx5.find

方案2：按模块分

cn.ylx5.teacher

AddTeacher…

cn.ylx5.Student

AddStudent…

包的定义：package 包名;

多级包用 . 分开即可

注意事项：

A:package语句必须是程序的第一条可执行的代码

B:package语句在一个java文件中只能有一个

C:如果没有package，默认表示无包名

带包的编译和运行

A:手动式

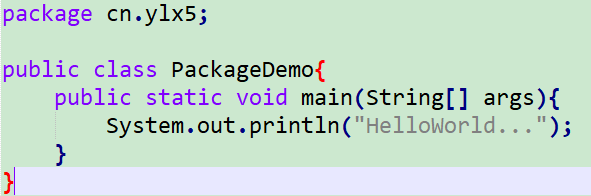
a:编写一个带包的java文件

b:通过javac命令编译该java文件

c:手动创建包名

d:把b步骤的class文件放到c步骤的最底层的包中

e:回到和包根目录在同一目录的地方，然后带包运行

例子：

在文件夹中创建cn文件夹，再在cn中创建ylx5，然后把class文件放到ylx5中，最后在dos中：java cn.ylx5.PackageDemo

B:自动式

a:javac编译的时候带上 -d 即可（注意格式）

javac -d . PackageDemo.java

b:通过java执行命令，带包运行

不同包下类直接的访问

cn.ylx5.PackageDemo pd = new cn.ylx5.PackageDemo();

导包

格式：import 包名;

这种方式导入是到类的名称

package import class有没有顺序关系

package > import > class

pacakge只能有一个，import可以有多个

class可以有多个，建议一个java类中只写一个

权限修饰符

类及其组成常见修饰符

修饰符：

权限修饰符：private 默认修饰符（default） protected public

状态修饰符：static final

抽象修饰符：abstract

类：

default public final abstract

成员变量：

private default protected public static final

构造方法：

private default protected public

成员方法：

private default protected public static final abstract

除以上意外的组合规则：

成员变量：public static final

成员方法：public static

public abstract

public final

内部类

把类定义在其他类的内部，这个类就被称为内部类



内部类的访问特点

A:内部类可以直接访问外部类的成员，包括私有

B:外部类要想访问内部类，必须创建内部类对象

内部类的位置

成员内部类：在成员位置定义的类被称为成员内部类

局部内部类：在局部位置定义的类被称为局部内部类

如何直接访问成员内部类的成员：

外部类名.内部类名 对象名 = 外部类对象.内部类对象;

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

成员内部类修饰符：day10

Eclipse

集成开发环境IDE

Eclipse是一个可扩展的开放源代码的IDE

Eclipse的特点描述

免费

纯JAVA语言编写

免安装

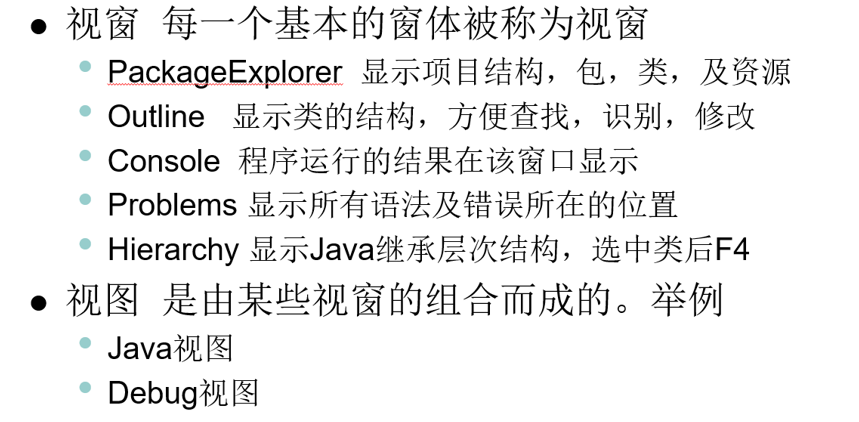
扩展性强

MyEclipse

在Eclipse基础上追加的功能性插件，对插件收费

在web开发中提供强大的系统架构平台

Eclipse组成之视窗与视图



Eclipse中打jar包并使用jar包

jar包是多个class文件的压缩包

jar包作用：用别人写好的东西

打jar包：选中项目，右键，Export-Java-Jar-自己制定一个路径和一个名称-Finish

使用jar包：复制到项目路径下并添加至构建路径

Eclipse中如何制作帮助文档

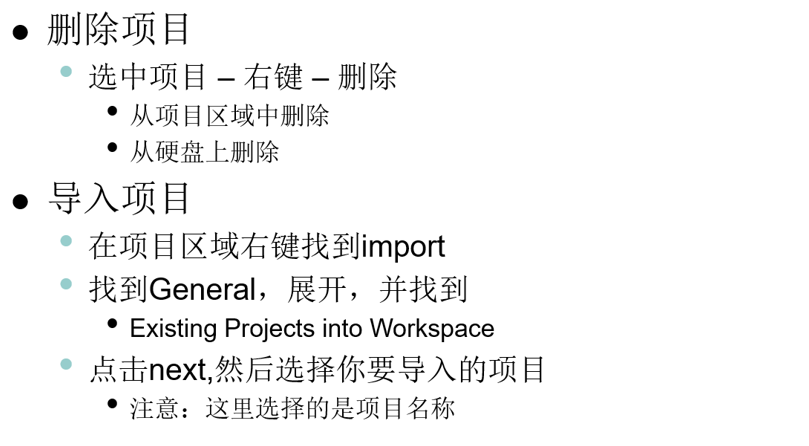
编写源程序（设计接口，抽象类等）

针对源程序添加文档注释（前面讲过了）

选中项目，右键 Export-Java-Javadoc-Finish

jar包和帮助文档必须配套使用

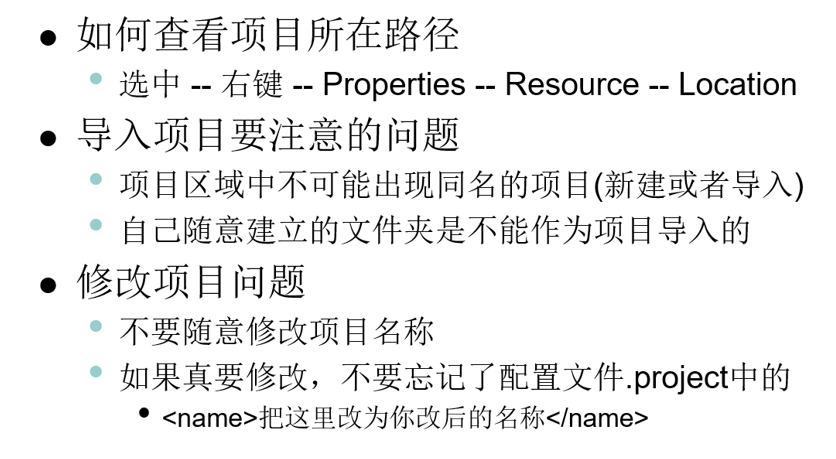
Eclipse中如何删除项目和导入项目



Eclipse中如何删除项目和导入项目



Eclipse中几个常见的小问题



Eclipse中代码的高级Debug调试

Debug的作用

调试程序 查看程序执行流程

如何查看程序执行流程

要查看必须设置断点

如何设置断点：

想看哪里的程序，就在那个有效程序的左边双击即可

在哪里设置：

哪里不会点哪里，学习阶段每个方法第一行都加上断点

如何运行设置断点后的程序：

右键—Debug as—Java Application

点完之后看什么地方：

Debug:断点测试的地方

在这个视图记住F6按键，一次看一行的执行过程，点红色按钮结束

Variables:查看程序的变量变化

ForDemo:被查看的源文件

Console:控制台

如何去断点：

1.再次双击即可取消

2.找到Variables视图，找到断点选项去除所有断点

Eclipse查看Java中的参数传递问题

通过程序查看断点来查看Java参数传递day11\_22

Object类概述及其方法

Object类的概述

是类层次结构的根类，所有类都直接或间接的继承自该类

构造方法

public Object(){}

Object类的hashCode()方法

public int hashCode()

返回该对象的哈希码值

哈希码值是根据地址值计算出来的一个值，这个值和地址值有关，但是不是实际地址值，可以理解为地址值，不同对象的哈希码值不一样

Object类的getClass()方法

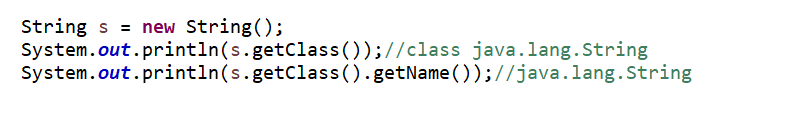
public final class getClass()

返回次Object的运行时类

class类的方法：public String getName()

以字符串形式返回此class对象所表示的实体名称

注意getClass()返回的是一个class



Class类重写了toString方法，所以第一行输出先打个class再打类名，而getName是取得的类名

Object类的toString()方法

该toString类方法Object返回一个由类的名称，其中所述对象是其实例，该符号字符`的字符串@ ”，并且对象的哈希码的无符号的十六进制表示，换句话说这个方法返回一个等于下列值的字符串：

this.getClass().getName()+'@'+Integer.toHexString(this.hashCode())

但是这组信息是没有意义的，建议所有子类都重写该方法

重写方式：把该类的所有成员变量值组成返回即可

重写举例：假如有一个学生类，成员变量有name,age

则此时重写toString方法

public String toString(){

return “姓名:”+name+“,年龄:”+age ;

}

此时，这种重写之后就变得有意义

重写最终版方案：利用eclipse自动生产toString()方法

当重写toString()方法之后，直接输出对象名称，此时显示的也是toString()方法的结果，因为直接输出一个对象的名称，其实就是调用该对象的toString()方法

Object类的equals()方法

public Boolean equals(Object obj)

指示其他某个对象是否与此对象“相等”

这个方法默认情况下比较的是地址值，但是比较地址值，一般来说，意义不大，一般都是用来比较对象的成员变量值是否相同，所以此时需要重写改方法

在Object类中，其实==和equals是一样的，都是比较地址值，但是在String类Integer等类中，重写了equals方法，所以String比较的是字符串内容而不是地址，所以在自己写类的时候，如果想要使用equals方法，需要自行重写该方法

==：

基本类型：比较值是否相同

引用类型：比较地址值是否相同

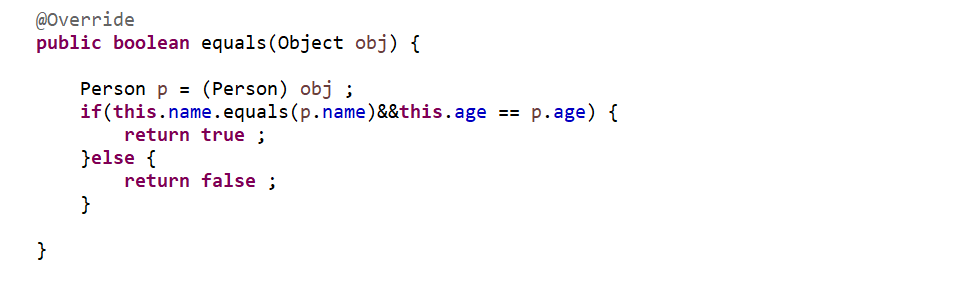
equals：

引用类型：默认情况下，比较地址值

不过，我们可以根据情况自己重写该方法，一般重写都是自动生成，比较对象的成员变量值是否相同

下面介绍几种重写的过程

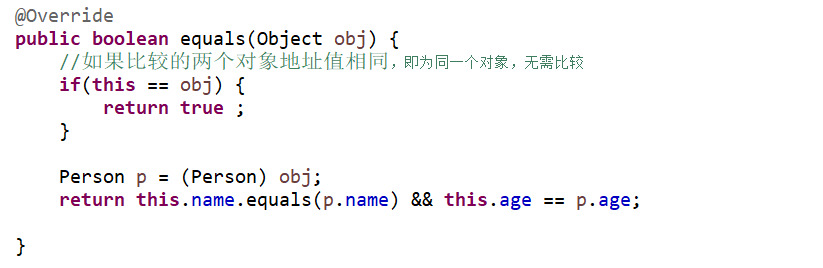
重写：



解释：先将形参obj向下转型为Person类，才可以访问Person类的成员变量，然后分别比较成员变量，相同返回true否则返回false

自此，重写完成，后续该类调用equals的时候，比较的是类的成员变量

重写equals方法代码改进：同一个对象无需比较

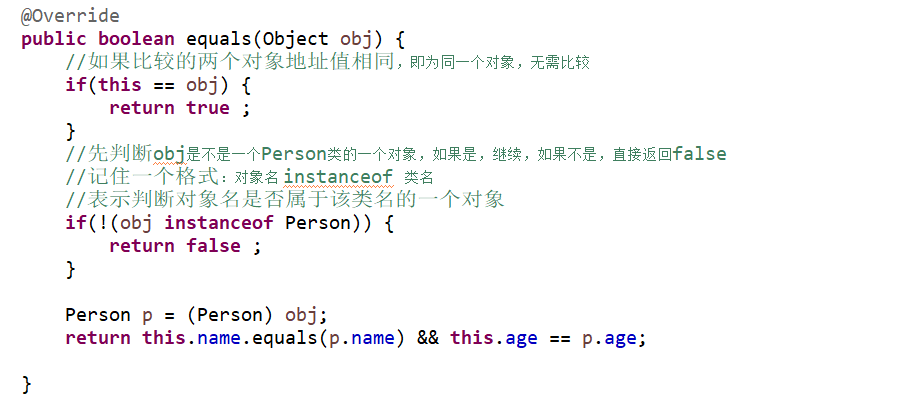


再改进（标准）：如果比较的不是Person对象会报错

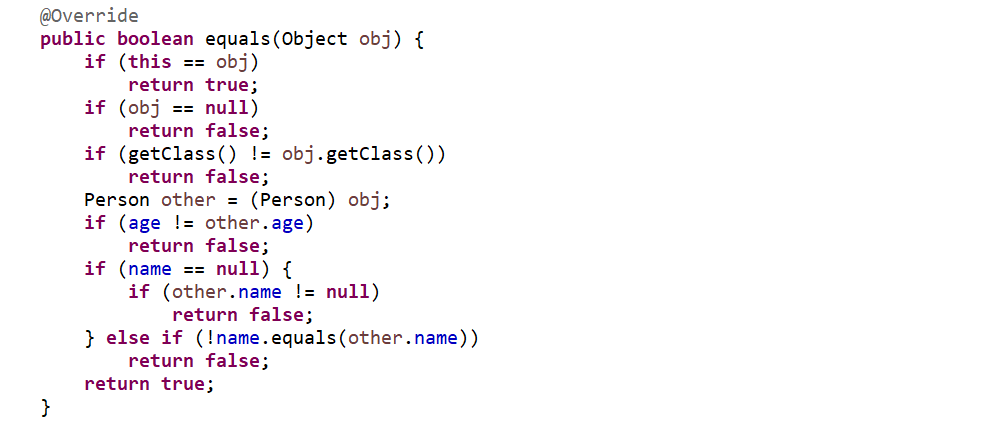
此时需要用到关键字instanceof

格式：对象名 instanceof 类名

表示判断该对象是否属于该类名的一个对象



最终版：eclipse自动生成equals方法



Object类的finalize()方法

protected void finalize()

当垃圾回收器确定不存在对该对象的更多引用时，由对象的垃圾回收器调用此方法，用于垃圾回收，但是什么时候回收不确定

Object类的clone()方法（day11\_30）

protected Object clone()

创建并返回对象的一个副本

前提:想要克隆一个类，首先让被克隆的类去实现Cloneable接口，此接口没有方法，称为标记接口，告诉我们实现该接口的类就可以实现对象的克隆了，因为该方法是protected修饰的，所以需要重写该方法（用eclipse默认重写即可）

Scanner类概述及其方法

java.util包下

public final calss Scanner

Scanner:用于接收键盘录入数据

Scanner sc = new Scanner(System.in);

System类下有一个静态的字段

public static final InputStream in;标准的输入流，对应键盘录入

上述调用的是用InputStream作为形参的构造方法

Scanner类的hasNextXxx()方法

public Boolean hasNextXxx()判断是否是某种类型的元素

判断是否还有下一个输入项，其中Xxx可以是int,double等，如果需要判断是否包含下一个字符串，则可以省略Xxx

Scanner类的NextXxx()方法

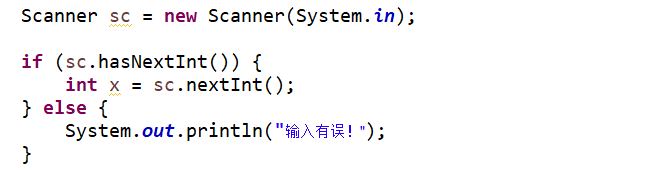
public Xxx nextXxx()获取该元素

nextXxx()获取下一个输入项，Xxx的含义与上个方法相同，默认情况下，Scanner使用空格，回车等作为分隔符

以上两个方法可以搭配使用：

举例：int x = sc.nextInt();

如果键盘录入不是int类型，会报错，为了避免此类情况发生可以首先判断输入的是否是int类型



两个常用方法(day12\_03)

public int nextInt():获取一个int类型的值

public String nextLine():获取一个String类型的值

String类的概述及其方法（非常重要）

String类的概述

**public final class String**

字符串：由多个字符组成的一串数据，也可以看出是一个字符数组

Java程序中所有字符串字面值(“abc”)都可以作为此类的对象

字符串是常量，他们的值在创建后不能更改，字符串缓冲区支持可变的字符串，因为String对象是不可变的，所以可以共享

String str = “abc”;等同于

char[] data = {‘a’,‘b’,‘c’};

String str = new String(data);

String类的构造方法

public String()：无参构造

public String(byte[] bytes)：把字节数组转换成字符串(byte表示-127-128，此时对应的是ASCII码表的字符赋值给String)

**public String(byte[] bytes,int offset,int length)：**

把字节数组的一部分转换成字符串(从索引offset开始往后length个)

public String(char[] value)：把字符数组转换成字符串

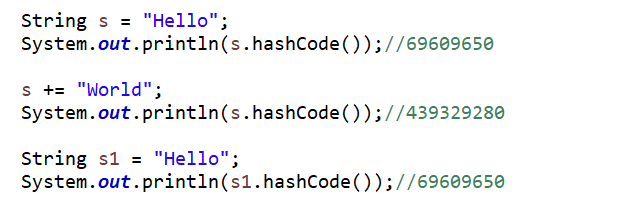
**public String(char[] value,int offset,int count)**

把字符数组的一部分转换成字符串

**public String(String original)**

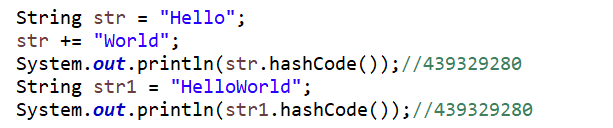
把字符串常量值转成字符串

String类的特点

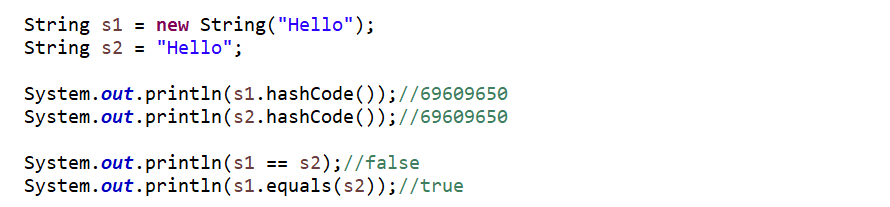


字符串直接赋值的方式是先到字符串常量池里去找，如果有就直接返回，没有，就创建并返回

一旦被赋值就不能改变



理解：新建str字符串，会首先去字符串常量池寻找是否有Hello，如果有，则str直接指向该字符串地址，如果没有，则创建Hello字符串(Hello字符串一旦被创建，则无法改变)然后又会去创建World字符串，接着把Hello和World拼接起来，把拼接起来的完整字符串地址给str，自始至终Hello字符串的值未曾改变，str指向的地址值可以变，当再创建一个str1并赋值HelloWorld时候，会找到之前已经存在的HelloWorld，而不会再创建，所以两个字符串输出的hashCode相同

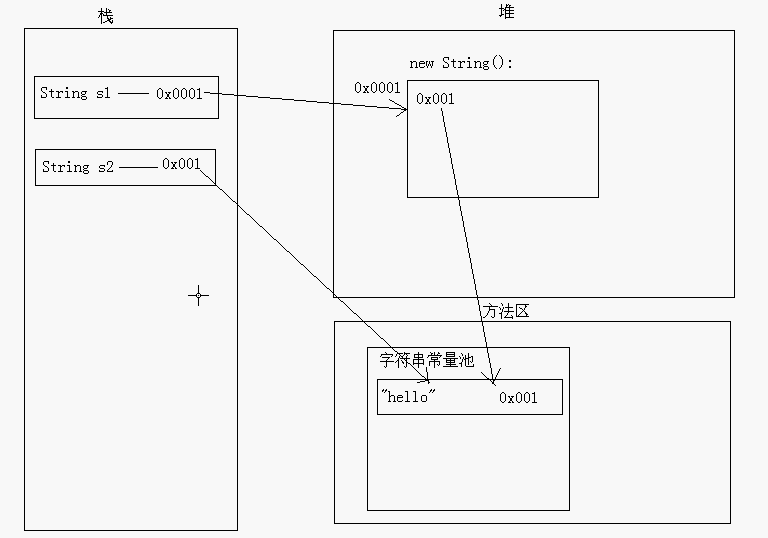


解释：第一行在堆内存中生成String对象，对象储存的值为”Hello”，此时new出来的String对象会去字符串常量池寻找”Hello”，如果有，则会把该字符串的地址值存在堆内存的new String()中，如果没有，则会在字符串常量池创建”Hello”然后把地址值给new String()并且此时s1指向的是堆内存中new String()对象的地址值，接下来创建s2字符串，直接赋值”Hello”，此时会去字符串常量池找”Hello”，之前有创建，所以直接让s2指向该地址值；虽然s1和s2的地址值不一样，但是String类重写了hashCode方法，String类的hashCode方法是返回字符串的哈希码值，所以此时s1和s2的hashCode是一样的，但是s1==s2比较地址值，所以是false，而s1.equals(s2)比较字符串内容是true

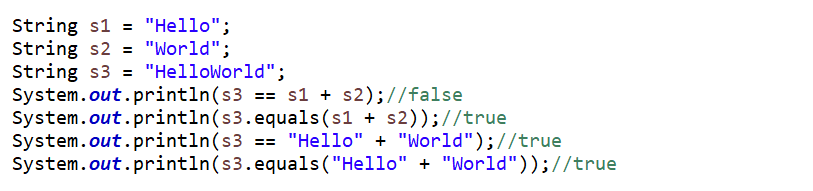


区别：如果独立运行前者会创建两个对象，后者会创建一个对象

内存模型图如下



面试题：看程序写结果



第一条语句，因为s1+s2是两个变量相加，会先开辟空间，所以新开辟的空间地址值与s3的地址值不同

第二条第四条略

第三条因为”Hello”和”World”是常量，所以会先相加再比较，相加之后是”HelloWorld”，所以会在字符串常量池中寻找，找到了，所以与s3指向的是同一个字符串，所以地址值相同

String类的判断功能

**boolean equals(Object obj):**

比较字符串的内容是否相同，区分大小写

**boolean equalsIgnoreCase(String str):**

比较字符串的内容是否相同，忽略大小写

**boolean contains(String str):**

判断大字符串是否包含小字符串

**boolean startsWith(String str):**

判断字符串是否以某个指定的字符串开头

**boolean endsWith(String str):**

判断字符串是否以某个指定的字符串结尾

**boolean isEmpty():**

判断字符串内容是否为空

注意：

String s = “”;//字符串内容为空，但是对象存在可以调用方法

String s = null;//字符串对象为空，无对象

String类的获取功能

**int length():**

获取字符串长度

**char charAt(int index):**

获取指定索引处位置的字符

**int indexOf(int ch):**

返回指定字符在字符串中第一次出现处的索引

**int indexOf(String str):**

返回指定字符串在字符串中第一次出现处的索引

**int indexOf(int ch,int fromIndex):**

返回指定字符在字符串中从指定位置后第一次出现处的索引

**int indexOf(String str,int fromIndex):**

返回指定字符串在字符串中从指定位置后第一次出现处的索引

**String substring(int start):**

从指定位置开始截取字符串，默认到末尾

**String substring(int start,int end):**

从指定位置开始到指定结束位置截取字符串

String类的转换功能

**byte[] getBytes()**

把字符串转换为字节数组

**char[] toCharArray()**

把字符串转换为字符数组

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

把字符数组转换成字符串

**static String valueOf(int i)**

把int类型转换成字符串

注意：String类的静态方法valueOf()可以把任意类型转成字符串

**String toLowerCase()**

把字符串转换成小写

**String toUpperCase()**

把字符串转换成大写

**String concat(String str)**

把字符串拼接，可以用+号代替

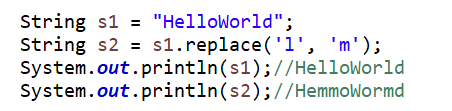
练习题：统计大串中小串出现的次数

String类的其他功能

替换功能

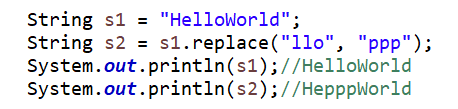
**String replace(char old,char new)**

把字符串的某个字符改成新的，并返回新的字符串



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

把字符串的某个子字符串改成新的，并返回新的字符串



去除字符串两端空格

**String trim()**

按字典顺序比较两个字符串

**int compareTo(String str)**

ASCII码表比较首字母区分大小写

**int compareToIgnoreCase(String str)**

ASCII码表比较首字母不区分大小写

特例：比较Hello和abc时，返回H-a

当比较”Hello”和”Hel”时，此时返回的是两个字符串的长度相减(由**compareTo源码得知**)

StringBuffer类的概述及其构造方法

StringBuffer类概述

我们如果对字符串进行拼接操作，每次拼接都会构建一个新的String对象，既耗时间，又浪费空间；而StringBuffer可以解决，StringBuffer是线程安全的可变字符串

StringBuffer和String的区别

前者长度内容可变，后者不可变；如果使用前者做字符串的拼接，不会浪费太多的资源

StringBuffer的构造方法

public StringBuffer()

无参构造，构造一个不带字符的字符串缓冲区初始容量为16个字符，一个字符串缓冲区的容量等于当前length+16

public StringBuffer(int capacity)

指定容量的字符串缓冲区对象

public StringBuffer(String str)

指定字符串内容的字符串缓冲区对象

StringBuffer类的方法

public int capacity()

返回当前容量，理论值

public int length()

返回长度（字符数），实际值

StringBuffer的添加功能

public StringBuffer append(String str)

可以把任意类型添加到字符串缓冲区里面，在自身添加，无需另外创建空间，并把自身返回

public StringBuffer insert(int offset,String str)

在指定位置把任意类型的数据插入到字符串缓冲区里面，并返回本身

StringBuffer的删除功能

public StringBuffer deleteCharAt(int index)

删除指定位置的一个字符，并返回本身

public StringBuffer delete(int start,int end)

删除从指定位置开始到指定位置结束的内容（包括start不包括end），并返回本身

StringBuffer的替换功能

public StringBuffer

replace(int start,int end,String str)

将字符串缓冲区从start开始到end结束，用str替换并返回本身

StringBuffer的反转功能

public StringBuffer reverse()

将字符串缓冲区的字符反转

StringBuffer的截取功能

public String substring(int start)

指定位置开始到最后截取字符串缓冲区并返回一个String类型的字符串，其本身不变

public String substring(int start,int end)

指定位置开始到指定位置结束截取字符串缓冲区并返回一个String类型的字符串，其本身不变

StringBuffer类练习

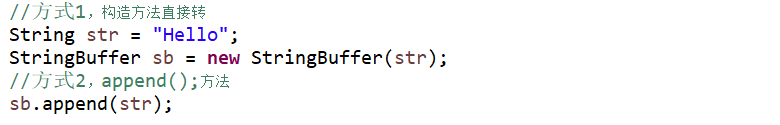
String和StringBuffer的相互转换

为什么要转换，我们把A转换成B是为了使用B的功能；

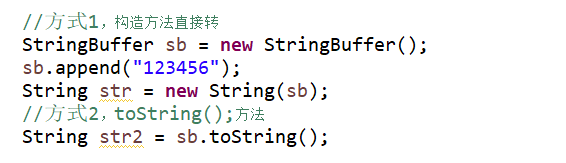
我们可能要的结果是A类型，所以还需要转回A



String转StringBuffer



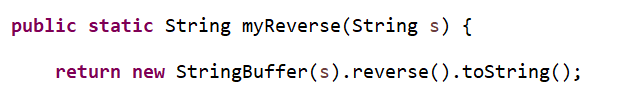
StringBuffer转String



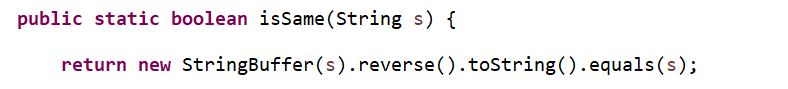
把数组拼接成一个字符串（用StringBuffer）



字符串反转功能



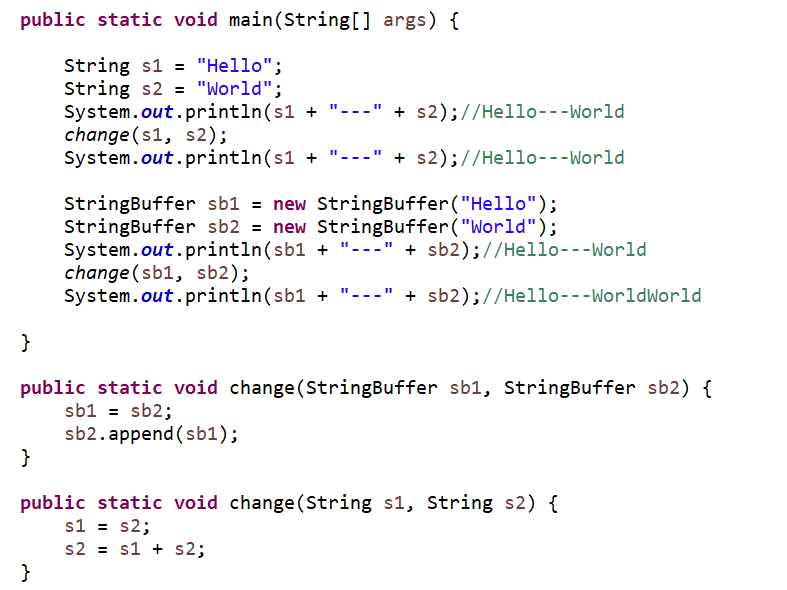
判断字符串是否对称



StringBuilder类概述

一个可变的字符序列。 该类提供了一个与StringBuffer兼容的API，但不能保证同步。 该类被设计为在字符串缓冲区被单个线程使用的地方（作为一般情况）用作StringBuffer替代替代。 在可能的情况下，建议使用这个类别优先于StringBuffer因为它在大多数实施中将更快。

面试题：看程序写结果



解释：第一个和第三个输出语句不做解释；

第一个change(s1,s2);方法调用，虽然传递的是String类型是引用类型，但是String比较特殊，他是常量值，所以String是特殊的引用类型，可以当做基本类型看，形参的改变对实参没有影响，形参复制的是字符串的值，并不是拿地址操作

第二个change(sb1.sb2);方法调用，当StringBuffer作为形参传递的时候，调用方法，是可以改变形参的，所以建议调用方法来改变

注意：String作为形参传递效果和基本类型作为形参传递是一样的

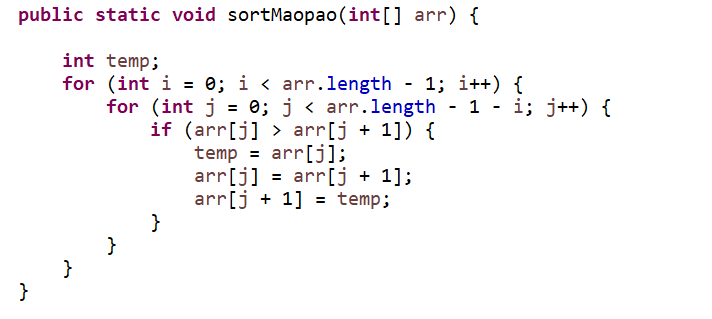
数组高级（排序和查找）

排序

冒泡排序：相邻元素两两比较，大的往后放，第一次完毕，最大值出现在最大索引处（每一次获得一个最大值）

选择排序：从0索引开始，依次和后面元素比较，小的往前方，第一次完毕，最小值出现在最小索引处（每一次获得一个最小值）

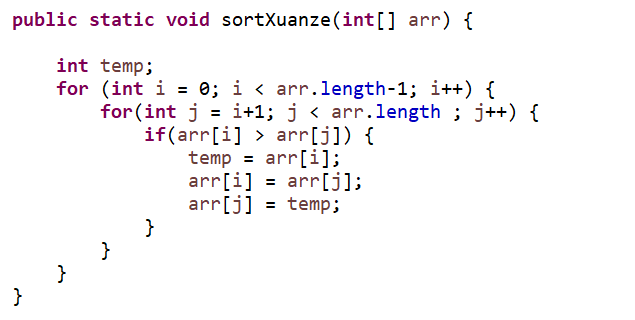
冒泡排序：



解释：外面的for循环控制比较次数，一共需要比较长度减1次；里面的for循环控制比较arr[0]和arr[1]比，1和2,2和3一直到倒数第二个和第一个比，大的往后放，第一次比较完之后，最大的在最后面；然后第二次比较，第二次比较的时候最后一位已经是最大，无需比较，因为比较了几次，就有几个最大的数无需比较所以每次要减去比较的次数，最后获得升序排序

记忆方法：循环体是交换，外层循环是次数，内层循环是比较方式，内层循环每次少比较一次所以需要减去外层循环次数

选择排序：

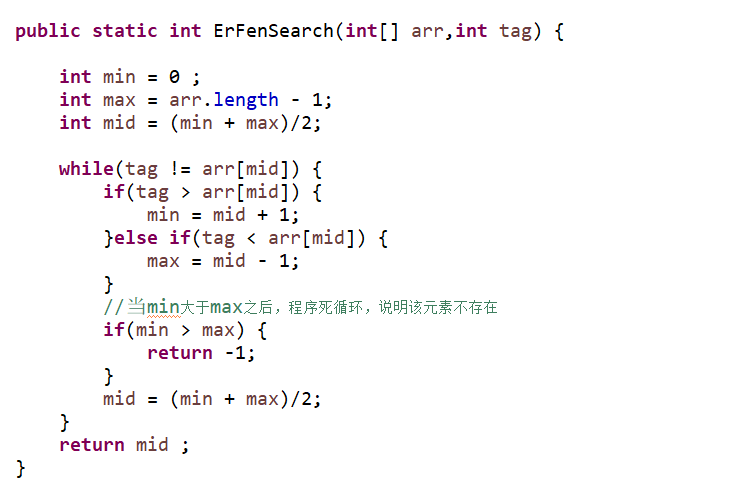


解释：先让arr[0]和arr[1]比，小的往前放，然后0和2比，然后0和3比，直到0和最后一个比，此时得到0处的值是最小的；外层循环是循环次数，每循环一次，都会得到最小值放在前面，需要循环length-1次，内层For循环比较的时候可以跳过已经得到的最小值，从i+1开始依次和后面的比较

查找

基本查找：数组元素无序（从头找到尾）

二分查找（折半查找）：数组元素有序



当数组是无序的时候，必须用基本查找；不能先排序用二分查找，因为排序之后改变了该元素的原始位置，再查找到的索引并不是元素原来所在位置的索引

Arrays类概述及其常用方法

Arrays类概述（具体看API）

针对数组进行操作的工具类

提供了排序，查找等功能

public static String toString(int[] a)

把数组转成字符串（仅以int数组举例，可以对任意数组操作）

public static void sort(int[] a)

把数组进行排序

public static int binarySearch(int[] a,int key)

对数组进行二分查找

基本类型包装类的引入

将基本数据类型封装成对象的好处在于可以在对象中定义更多的功能方法操作该数据

常用的操作之一：用于基本数据类型与字符串之间的转换

基本类型和包装类的对应

byte:Byte short:Short int:Integer

long:Long float:Float double:Double

char:Character boolean:Boolean

Integer类概述及其构造方法

Integer类的概述

Integer类在对象中包装了一个基本类型int的值

该类提供了多个方法，能在int类型和String类型之间相互转换，还提供了处理int类型时非常有用的其他一些常量和方法

Integer类的构造方法

public Integer(int value)

将一个int类型转换成Integer类的一个对象

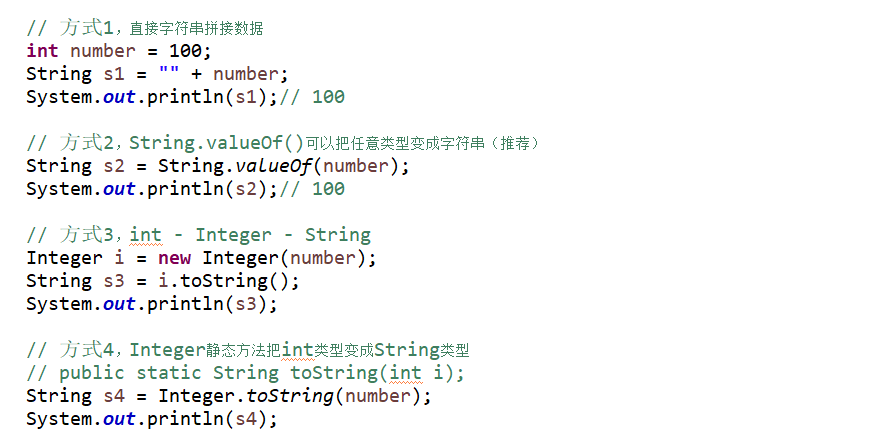
public Integer(String s)

将一个纯数字的字符串转换成一个Integer类的对象不能含有字母

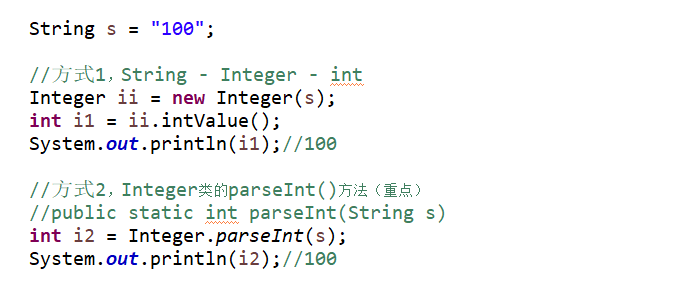
Integer类的成员方法

int类型和String类型的相互转换

int – String



String – int



总结：重点

String.valueOf(Object obj)

String类的静态方法valueOf可以把任意类型转换成字符串

Integer.parseInt(String s)

基本类型对应的类都有parseXxx(String s)可以把字符串转换成该类对应的基本类型，例如parseDouble(String s),但是字符串必须是纯数字或对应的要求

Integer类的成员方法

Integer类常用的基本进制转换

public static String toBinaryString(int i)

把一个int类型数据转换成二进制并以字符串形式返回

public static String toOctalString(int i)

把一个int类型数据转换成八进制并以字符串形式返回

public static String toHexString(int i)

把一个int类型数据转换成十六进制并以字符串形式返回

十进制到其他进制（重点）

public static String toString(int i,int radix)

把一个int类型数据转换成radix进制，radix范围是2-36，超过这个范围则返回的是十进制的

其他进制到十进制

public static int parseInt(String s,int radix)

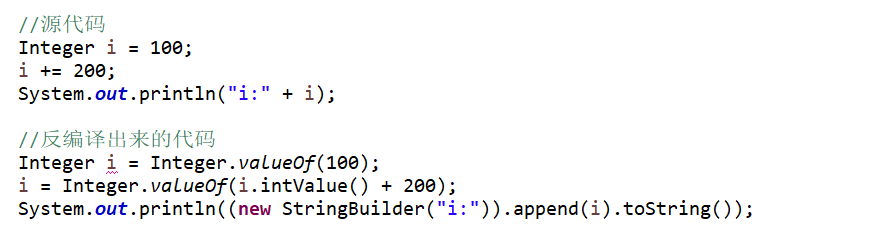
Integer.parseInt(“100”,16);

表示一个数用16进制表示是100，那么此方法返回他的十进制，也就是256

自动装箱和自动拆箱

自动装箱：把基本类型转换为包装类类型

自动拆箱：把包装类类型转换为基本类型



JDK5新特性，自动拆装箱

Integer i = null;

i += 100;//报错，空指针异常

因为自动拆装箱时需要调方法，空指针无法调用方法

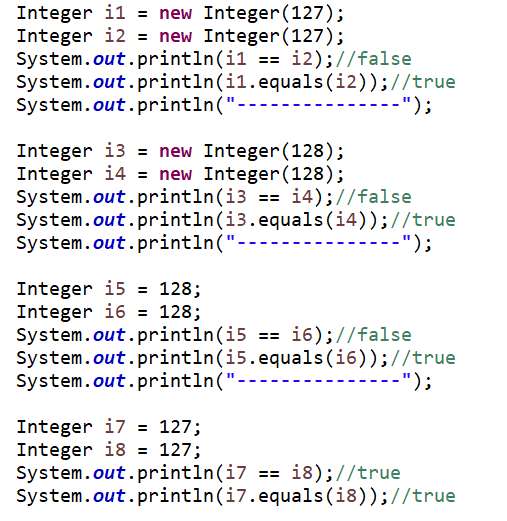
Integer面试题：下面代码做了哪些事情

Integer i = 1;//自动装箱

i += 1;//先自动拆箱，然后自动装箱

什么时候会拆箱？——基本数据类型和引用数据类型做运算时

什么时候会装箱？——基本数据类型赋值给引用数据类型的时候



正则表达式

符合一定规则的字符串，正则表达式是一个字符串

String类有一个方法

public boolean matches(String regex)

告知字符串是否匹配给定的正则表达式

正则表达式的组成规则

规则字符在java.util.regex Pattern类中

常见组成规则

字符：

x 表示字符x。举例：’a’表示字符a

\\表示反斜线字符

\r换行符

\n回车符

字符类

[abc]表示a,b或c，三个中选一

[^abc]除了abc外的任何字符

[a-zA-Z]a到z或者A到Z，两头字母包括在内的范围（这个表达包括所有的英文字母）

[0-9]0到9的字符

预定义字符类

. 表示任何字符，（\.表示.本身）

\d 表示数字[0-9]（在字符串中需要转义，需要写成[\\d](file:///\\d)）

\D表示非数字[^0-9]

\w 单词字符：[a-zA-Z\_0-9]在正则表达式里面组成单词的东西必须有这些东西组成

\W 非单词字符

边界匹配器

^ 行的开头

$ 行的结尾

\b 单词边界：就是不是单词字符的地方

举例：hello!world?haha;xixi(!,?,;三个都是单词边界，不是单词的地方)

Greedy 数量词

X? X出现一次或一次都没出现

X\* X出现零次或一次或多次

X+ X出现一次或多次

X{n} X恰好出现n次

X{n,} X出现至少n次

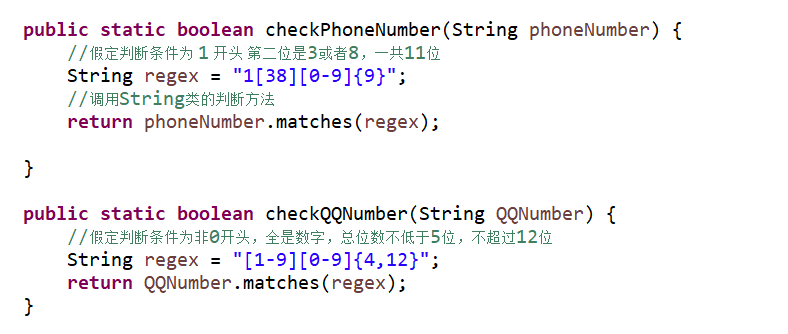
X{n,m} X至少n次，但是不超过m次

正则表达式的应用

判断功能用的方法是

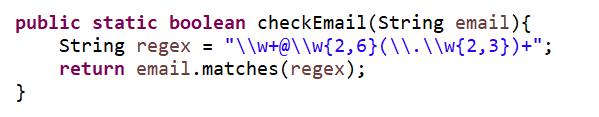
String类的public boolean matches(String regex)

例子：检查电话号码和QQ号码



正则表达式练习

判断功能：检验邮箱



解释：@前面是单词字符(a-zA-Z\_0-9)需要出现一次或多次，[然后@后面.前面单词字符出现2到6](mailto:然后@后面.前面单词字符出现2到6)次，[最后可能@sina.com.cn](mailto:最后可能@sina.com.cn)所以考虑到两个点的情况，需要把后面的点部分括号起来然后+出现一次或多次

分割功能用的方法也是String类的

public String[] split(String regex)

按照给定正则表达式的匹配拆分此字符串

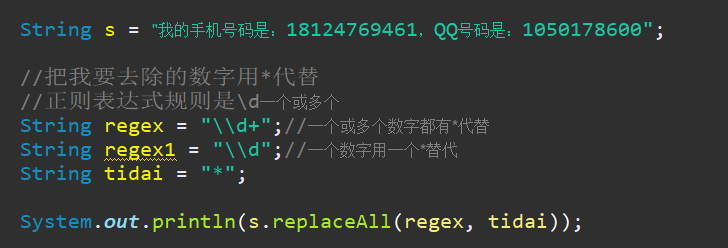
举例：



正则表达式的替换功能方法

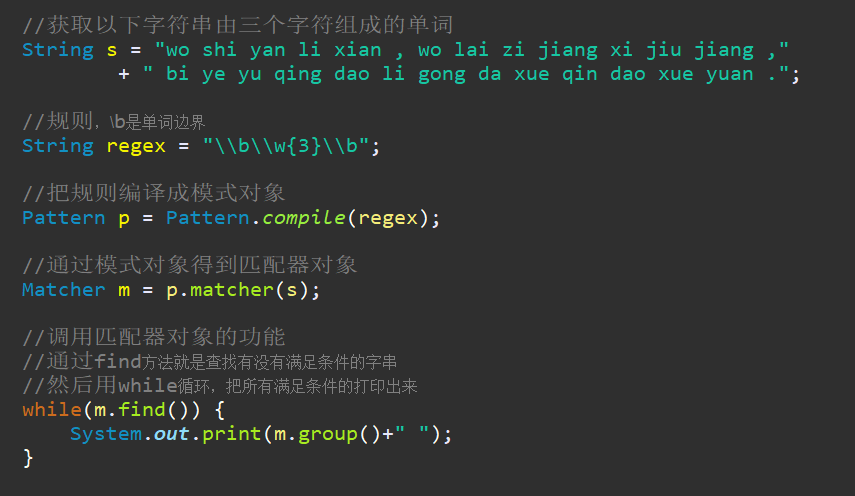
public String replaceAll(String regex,String replaceme)

用传过来的参数字符串按照规则替代原有字符串



正则表达式的获取功能

Pattern和Matcher的应用



注意事项：一定要先用find()然后才能group()

Math类

Math类概述

Math类包含用于执行基本数学运算的方法，如初等指数，对数，平方根和三角函数

成员方法

public static int abs(int a)

返回绝对值

public static double ceil(double a)

向上取整Math.ceil(10.1)=11.0;Math.ceil(-10.1)=-10.0;

public static double floor(double a)

向下取整Math.floor(10.1)=10.0;Math.floor(-10.1)=-11.0;

public static int max(int a,int b) min自学

返回最大值

public static double pow(double a,double b)

返回a的b次方

public static double random()

返回一个随机数[0.0,1.0)包括0.0，不包括1.0

public static int round(float a) 参数为double的自学

四舍五入

public static double sqrt(double a)

返回a的正平方根

成员变量

public static final double PI

public static final double E

案例：返回1-100的随机数

(int)(Math.random()\*100+1)

案例：写一个程序，输入min和max返回这两个数之间的随机数

return (int)(Math.random()\*(max-min+1)+min);

Random类及其构造方法

此类用于产生随机数

构造方法：

public Random()

没有给种子，用的是默认种子，是当前时间的毫秒值

public Random(long seed)

给出固定的种子，给定种子后，每次得到的随机数是相同的

成员方法

public int nextInt()

返回的是int范围内的随机数

public int nextInt(int n)

返回的是[0,n)范围的随机数

System类

System类包含一些有用的类字段和方法，它不能被实例化

成员方法

public static void gc()

运行垃圾回收器，调用该方法实际是调用finalize()方法

API解释：调用 gc 方法暗示着 Java 虚拟机做了一些努力来回收未用对象，以便能够快速地重用这些对象当前占用的内存。当控制权从方法调用中返回时，虚拟机已经尽最大努力从所有丢弃的对象中回收了空间。

调用 System.gc() 实际上等效于调用：

Runtime.getRuntime().gc()

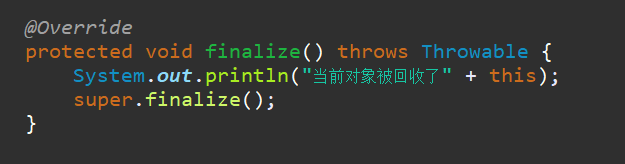
刘意课件：System.gc()可用于垃圾回收。当使用System.gc()回收某个对象所占用的内存之前，通过要求程序调用适当的方法来清理资源。在没有明确指定资源清理的情况下，Java提高了默认机制来清理该对象的资源，就是调用Object类的finalize()方法。finalize()方法的作用是释放一个对象占用的内存空间时，会被JVM调用。而子类重写该方法，就可以清理对象占用的资源，该方法由于没有链式调用，所以必须手动实现。

从程序的运行结果可以发现，执行System.gc()前，系统会自动调用finalize()方法清除对象占有的资源，通过super.finalize()方式可以实现从下到上的finalize()方法的调用，即先释放自己的资源，再去释放父类的资源。

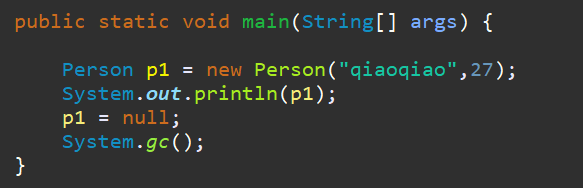
但是，不要在程序中频繁的调用垃圾回收，因为每一次执行垃圾回收，jvm都会强制启动垃圾回收器运行，这会耗费更多的系统资源，会与正常的Java程序运行争抢资源，只有在执行大量的对象的释放，才调用垃圾回收最好

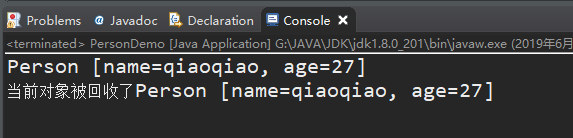
举例：

定义Person类，在Person类中重写了finalize方法



在Person类中new了一个对象并让p1指向它





运行之后，首先打印出重写过的toString()方法，然后让p1指向空，此时new出来的对象就被定义为垃圾，没有引用指向他，但是调用System.gc()的时候，发现打印出来的话是finalize方法里的，这足以说明执行System.gc()前，系统会自动调用finalize()方法清除对象占有的资源，通过super.finalize()方式可以实现从下到上的finalize()方法的调用，即先释放自己的资源，再去释放父类的资源

public static void exit(int status)

终止当前正在运行的JAVA虚拟机，参数用作状态码，根据惯例，非0的状态码表示异常终止，0表示正常终止，一般参数给0

public static long currentTimeMillis()

返回以毫秒为单位的当前时间；返回当前时间与协调世界时1970年1月1日午夜之间的时间差

单独得到这个时间目前对我们意义不大，所以运用的时候一般可以统计程序运行的时间



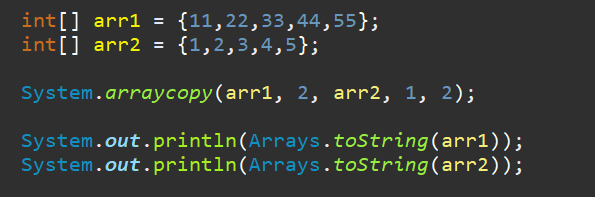
public static void arraycopy(Object src,

int srcPos,Object dest,int destPos,int length)

从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；源数组数据不变，目标数组指的位置被替代

src:源数组；srcPos:源数组中的起始位置，dest:目标数组；

destPos:目标数组中的起始位置length:要复制的数组元素的数量



BigInteger类

概述

因为Integer类范围有限，不足以满足需求，所以需要BigInteger

构造方法：（多个构造方法，只说一个）其他看API

[BigInteger](mk:@MSITStore:G:\JAVA\API\JDK_API_1_6_zh_CN.CHM::/java/math/BigInteger.html#BigInteger(java.lang.String))([String](mk:@MSITStore:G:\JAVA\API\JDK_API_1_6_zh_CN.CHM::/java/lang/String.html) val)

把纯数字字符串转换为BigInteger类型

public BigInteger add(BigInteger val)

加法，不能用基本运算“+”来计算

public BigInteger subtract(BigInteger val)

减法

public BigInteger multiply(BigInteger val)

乘法

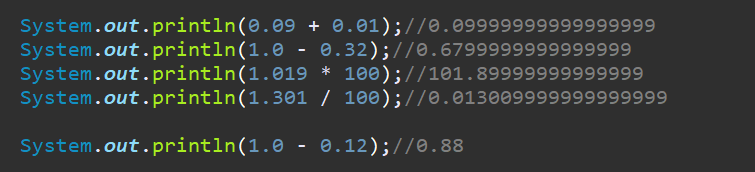
public BigInteger divide(BigInteger val)

除法

public BigInteger[] divideAndRemainder(BigInteger val)

返回包含 (this / val) 后跟 (this % val) 的两个 BigInteger 类型数据的数组其中改数组中第一个元素是商，第二个是余数

BigDecimal类



由于在运算的时候，float类型和double类型很容易丢失精度（原因是因为float类型的数据存储和整数不一样导致的，他们大部分的时候，都是带有效数字位），所以，为了能精确的表示和计算浮点数，Java提供了BigDecimal

BigDecimal类概述

不可变的，任意精度的有符号十进制数

构造方法

public BigDecimal(String val)//推荐此构造方法

成员方法

public BigDecimal add(BigDecimal augend)

加

public BigDecimal subtract(BigDecimal subtrahend)

减

public BigDecimal multiply(BigDecimal multiplicand)

乘

public BigDecimal divide(BigDecimal divisor)

除

public BigDecimal divide(BigDecimal divisor,int scale,int roundingMode)

除法，第一个参数是除数，第二个是几位小数，第三个是取舍方式第三个一般用ROUND\_HALF\_UP，其他取舍方式参考API

Date类概述及其方法

Date类表示特定的瞬间，精确到毫秒

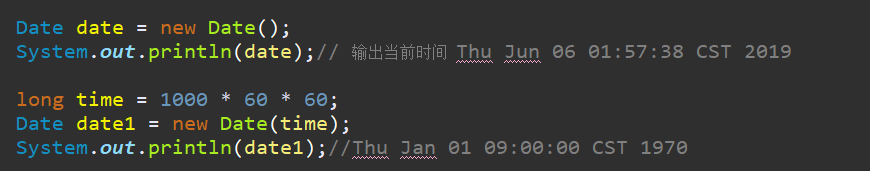
构造方法

Date()

根据当前的默认毫秒值创建日期对象

Date(long date)

根据给定的毫秒值创建日期对象



第一个，根据当前的默认毫秒值创建日期对象，这个毫秒值是1970年1月1日0点到现在的时间差

第二个，给的参数是一个小时的毫秒值，打印出的是1970年1月1日早上9点，因为我们所处东八区，时差8小时，所以8小时加1

成员方法

public long getTime()从date得到毫秒值

获取当前时间，以毫秒为单位

public void setTime(long time)从毫秒值到date

设置时间，以1970年1月1日8点为起点（东八区）

DateFormat类

一般直接输出日期格式，很难看懂，不是我们想要的

DateFormat可以进行日期和字符串的格式化和解析，但是由于是抽象类，所以使用具体子类SimpleDateFormat

SimpleDateFormat构造方法

SimpleDateFormat()

创建一个SimpleDateFormat对象（默认模式）

SimpleDateFormat(String pattern)

给定模式创建一个对象（具体模式看API）

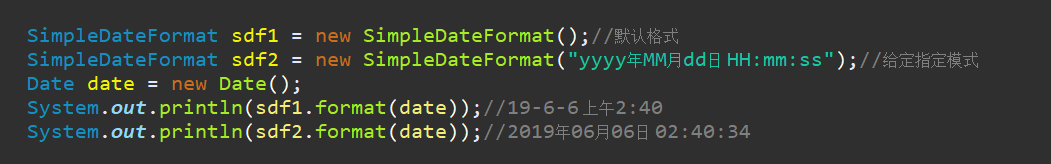
一般用格式”yyyy-MM-dd HH:mm:ss”（注意字母大小写）

DateFormat类成员方法

public final String format(Date date)

把日期类以字符串形式输出(需要用子类对象调用)

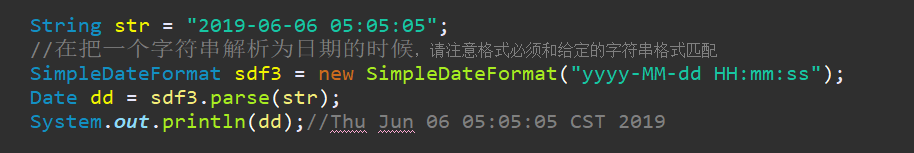
把日期格式化为字符串输出



public Date parse(String val)

把字符串解析为日期格式

把字符串解析为日期格式



Calendar类（抽象类）

Calendar 类是一个抽象类，它为特定瞬间与一组诸如 YEAR、MONTH、DAY\_OF\_MONTH、HOUR 等 日历字段之间的转换提供了一些方法，并为操作日历字段（例如获得下星期的日期）提供了一些方法。

成员方法

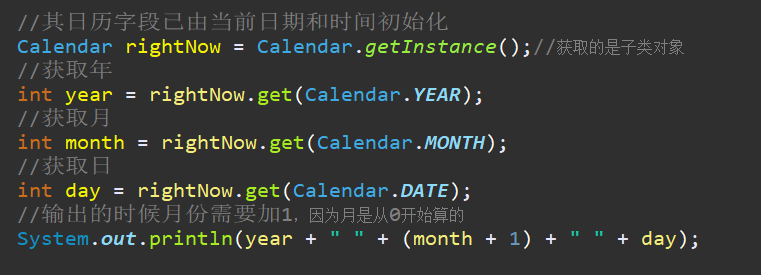
public int get(int field)

返回给定日历字段的值，日历类中的每个日历字段都是静态的成员变量，并且是int类型

因为Calendar是个抽象类，所以不能直接调用构造方法，但是其中提供了一个静态方法getInstance()方法，直接调用可以获得一个对象

Calendar rightNow = Calendar.getInstance();

底层原理其实getInstance返回的是一个子类对象



public static Calendar getInstance()

获取一个Calendar对象（其实是子类对象）

public int get(int field)

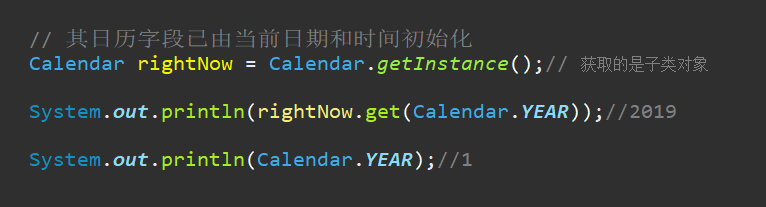
获取Calendar对象里的字段（年月日时分秒都有,都是int类型）

public void add(int field,int amount)

根据给定的日历字段和对应的时间来对当前的日历进行操作

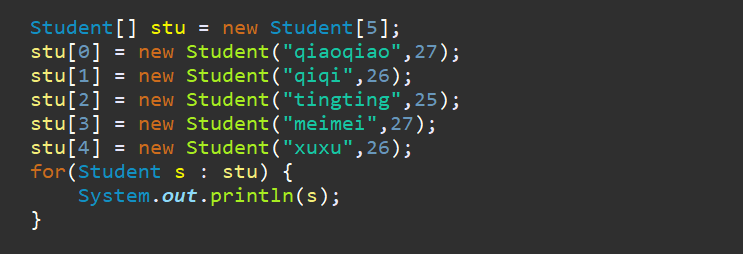
public final void set(int year,int month,int date)

设置当前的日历的年月日



注意：YEAR,MONTH等都是静态字段，由类所有；所以需要使用对象的get()方法，才能获得需要的值，如果直接输出Calendar.YEAR则得到的是静态类属性

对象数组的概述和使用



集合类

集合类的概述

面向对象语言对事物的体现都是以对象的形式，所以为了方便对多个对象的操作，我们就必须把这多个对象进行存储；首先我们可以考虑数组，这就是对象数组；但是因为数组的长度是固定的，不能满足变化的需求，所以Java就提供了集合类

数组和集合的区别：

A:长度区别

数组的长度固定，集合的长度可变

B:内容不同

数组存储的是同一种类型的元素

集合可以存储不同类型的元素

C:元素的数据类型问题

数组可以存储基本数据类型，也可以存储引用数据类型

集合只能存储引用类型

集合类的特点：

只能存储引用类型；集合的长度可变；集合可以存储不同类型的对象

集合类不能存储基本数据类型，当添加基本数据类型的元素时，会自动装箱，封装成对应的类，通过反编译软件可以知道

Java提供多种集合类，他们的数据结构不同，但是他们肯定有共性的能容（存储，获取，判断等）通过不断的向上提取，我们就能够得到一个集合的集成体系结构图

这个体系的老大是Collection

Collection

Set

HashSet

TreeSet

Collection接口

Collection是集合的顶层接口，他的子体系有重复的，有唯一的，有有序的，有无序的

Collection的功能概述

1:添加功能

boolean add(Object obj)

添加一个元素

boolean addAll(Collection c)

添加一个集合的元素

2:删除功能

void clear()

移除所有元素

boolean remove(Object obj)

移除一个元素

boolean removeAll(Collection c)

移除一个集合的元素，只要有一个被移除，就返回true

3:判断功能

boolean contains(Object obj)

判断集合是否包含指定的元素

boolean containsAll(Collection c)

判断集合中是否包含指定的集合元素，只有包含所有的元素才叫包含

boolean isEmpty()

判断集合是否为空

4:获取功能

Iterator<E> iterator()

获取一个在此集合的元素上进行迭代的迭代器

5:长度功能

int size()

元素的个数

6:交集功能

boolean retainAll(Collection c)

解释：假设有两个集合A,B;

A.retainAll(B)，最终的结果保存在A中，B不变，返回值表示的是A是否发生过改变

7: 把集合转换成数组

Object[] toArray()

把集合转换成数组

集合的遍历：依次获取集合中的每一个元素

Object[] toArray()

把集合转换成数组，可以实现集合的遍历

Iterator<E> iterator()

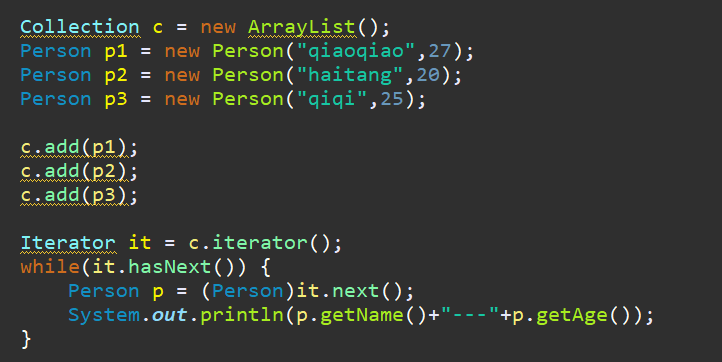
迭代器，集合的专用遍历方式

Object next()

获取元素，并移动到下一个位置

boolean hasNxet()

判断如果仍有元素可以迭代，则返回true



迭代器：是遍历集合的一种方式；迭代器是依赖于集合而存在的

迭代器的使用方式:



集合的使用步骤：

A:创建集合对象

B:创建元素对象

C:把元素添加到集合

D:遍历集合

a:通过集合对象获取迭代器对象

b:通过迭代器对象的hasNext()方法判断是否有元素

c:通过迭代器对象的next()获取元素并移动到下一个位置

迭代器为什么不定义成一个类，而是一个接口呢

假设迭代器定义的是一个类，这样我们就可以创建该类的对象，调用该类的方法来实现集合的遍历，但是Java中提供了很多的集合类，而这些集合类的数据结构是不同的。所以，存储的方式和遍历的方式应该是不同的，进而他们的遍历方式也应该不是一样的，最终，就没有定义迭代器类的

而无论你是哪种集合，你都应该具备获取元素的操作，并且，最好再辅助于判断功能，这样，在获取前先判断，这样就更不容易出错，也就是说，判断功能和获取功能应该是一个集合遍历所具备的，而每种集合的方式又不太一样，所以我们把这两个功能给提取出来，并不提供具体实现，这种方式就是接口

那么，真正的具体的实现类在哪里呢？

在真正的具体的子类中，以内部类的方式体现的

迭代器原理及源码（day15\_14）

List集合存储字符串并遍历

List概述

有序的collection，此接口的用户可以对列表中每个元素的插入位置进行精确的控制，用户可以根据元素的整数索引（在列表中的位置）访问元素，并搜索列表中的元素。与set不同，列表通常允许重复的元素

List集合的特点：

有序（存储和取出的元素一致），可重复的

add()方法在底层是直接返回true所以肯定是可以重复的添加元素

List集合的特有功能

A：添加功能

Void add(int index,Object element)

在指定位置添加元素,如果索引越界会报错，但是如果是在最后一个添加，则不会报错

例如：长度为3，在3的索引位置添加不会报错，会在最后一个添加，但是4就会报错

B:获取功能

Object get(int index)

获取指定位置的元素，越界会报错

C：列表迭代器

ListIterator listIterator()

List集合特有的迭代器

ListIterator listIterator(int index)

从指定位置开始获取迭代器

D：删除功能

Object remove(int index)

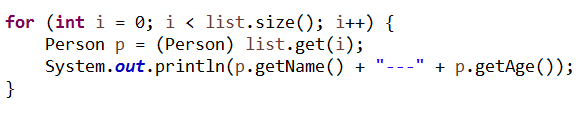
根据索引删除元素，返回删除的元素

E:修改功能

Object set(int index,Object element)

根据索引修改元素，返回被修改的元素

List集合特有的遍历功能



因为list列表是有序的，有索引，所以可以利用get()方法进行遍历

列表迭代器

ListIterator listIterator()

List集合特有的迭代器

该迭代器集成了Iteretor迭代器，所以就可以直接使用hasNext()和next()方法

特有功能：

Object previous()

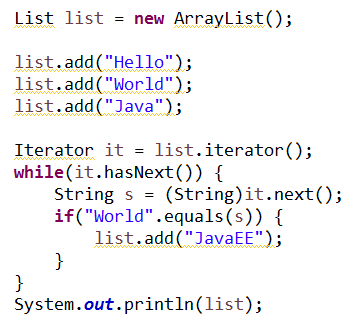
获取上一个元素（与next()相反）

Boolean hasPrevious()

判断是否有上一个元素

**注意：ListIterator可以实现逆向遍历，但是当创建迭代器时，指针是在最前面，所以必须先正向遍历，迭代器指向最后一个元素之后，才能逆向遍历，所以一般无意义，不实用。**

案例：在列表中查找”World”，如果找到了就添加字符串”JavaEE”



以上代码会报错

报错原因：因为迭代器是依赖于集合而存在的，在判断成功后，集合中新添加了元素，而迭代器却不知道，所以就报错了，这个错叫并发修改异常。其实这个问题描述的是：迭代器遍历元素的时候，通过集合是不能修改元素的

解决办法：

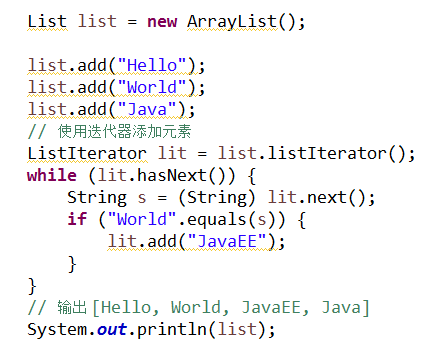
A：迭代器迭代元素，迭代器修改元素

添加的元素是跟在查找的元素后面

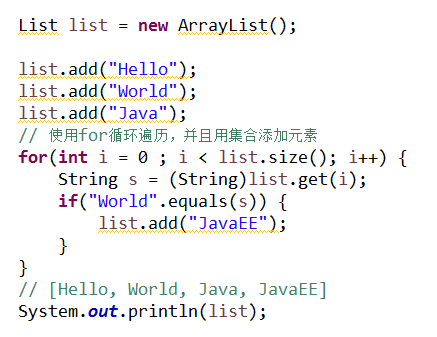
B：集合遍历元素，集合修改元素（for循环遍历）

添加的元素是添加在集合最后

第一种方法中，Iterator迭代器是没有添加功能的，所以我们使用其子接口”ListIterator”

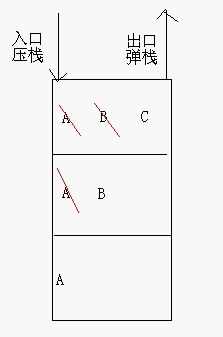


第二种方法，使用for循环遍历，并且用集合添加元素



常见的数据结构

面试题：常见的数据结构的优缺点

栈：先进后出

需求：把A B C存储到栈中

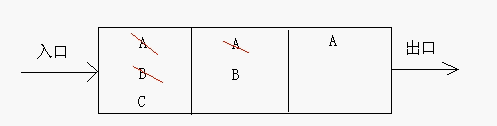
压栈：

首先存入A，B来了之后，会把A压到下一层，以此类推

弹栈：

最上面的先出来，所以取出是C B A

举例：子弹夹：先压进的子弹最后打出来



队列：先进先出

需求：把A B C存储到队列中

存储：

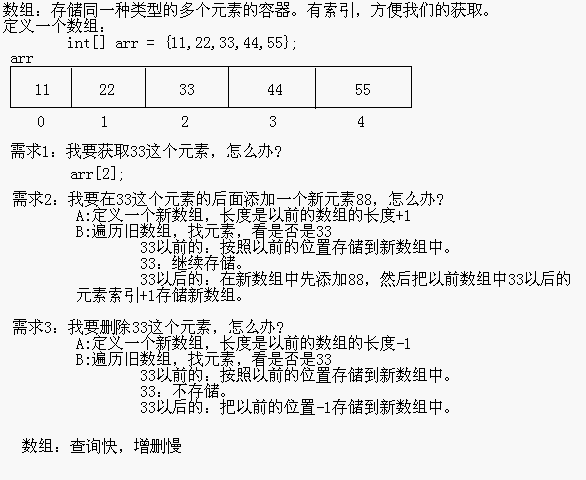
存入A，接着存入B会把A挤到下一个空间，以此类推

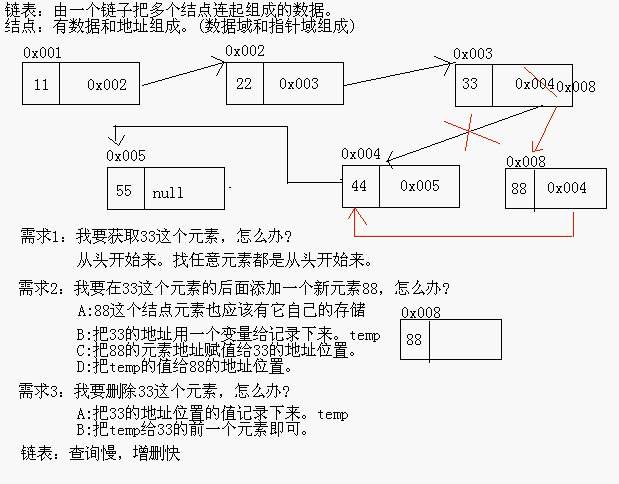
取出：

取出 A B C因为先进去的先出来

举例：

排队例子





上述是单向链表，如果把头元素的地址给了最后一个元素的地址位置，就是循环链表。如果把每个结点由3部分组成，我们就可以组成双向链表，如果再把前后的对应也连接起来，就成了双向循环链表

List的子类特点：（面试题）

ArrayList：

底层数据结构是数组；查询快；增删慢；线程不安全；效率高

Vector：

底层数据结构是数组；查询快；增删慢；线程安全；效率低

LinkedList：

底层数据结构是链表；查询慢；增删快；线程不安全；效率高

List有三个子类，我们到底使用谁？

看需求：

需要安全：Vector

（实际要安全的话也不用这个了，后面有替代的，这里的安全只是就这三个子类而言）

不需要安全：ArrayList或者LinkedList

查询多：ArrayList（用的多）

增删多：LinkedList

ArrayList的使用

迭代器和for循环遍历

Vector

Vector的特有功能

1:添加功能(一般不用，用add()替代)

public void addElement(Object obj)

2:获取功能(一般不用，用get()和Iterator替代)

public object elementAt(int index)

public Enumeration elements()

LinkedList

LinkedList特有功能：

A:添加功能

public void addFirst(Object e)

在最前面添加一个元素

public void addLast(Object e)

[无意义，与add()差不多]

B:获取功能

public Object getFirst()

获取第一个元素

public Object getLast()

获取最后一个元素

C:删除功能

public Object removeFirst()

移除第一个元素并返回

public Object removeLast()

移除最后一个元素并返回

ArrayList案例:

去除集合中字符串的重复值（字符串内容相同）

方法一思路：

A:创建集合对象

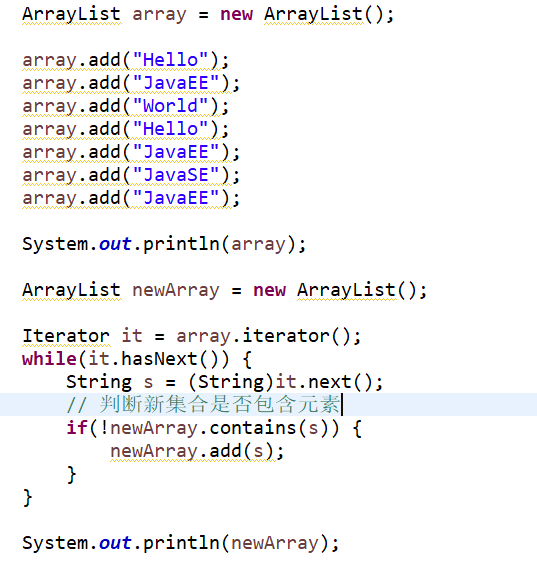
B:添加多个字符串元素（包含内容相同的）

C:创建新集合

D:遍历旧集合，获取得到每一个元素

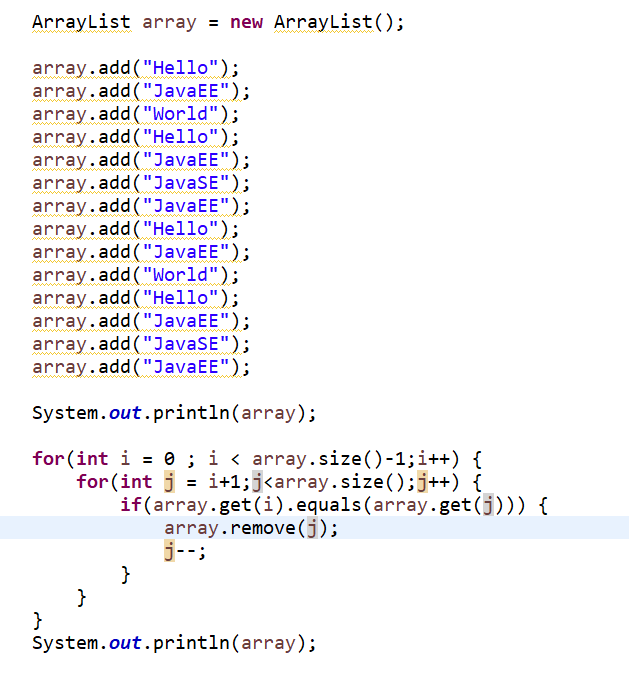
E:拿这个元素去新集合找，没有就添加

F:遍历新集合



方法二：

不创建新集合，利用选择排序思想，挨个比较，有相同的就去掉



解释：if语句里有个j--;因为当有重复元素时，例如第五个和第四个都是”JaveEE”,第四个被删除之后，第五个就会填充到第四个的位置，这样就会有遗漏，所以为了避免这样的情况，当移除一个元素时，把索引减一，回来前面一个位置再比一次

去除自定义对象的重复值（成员变量值相同）

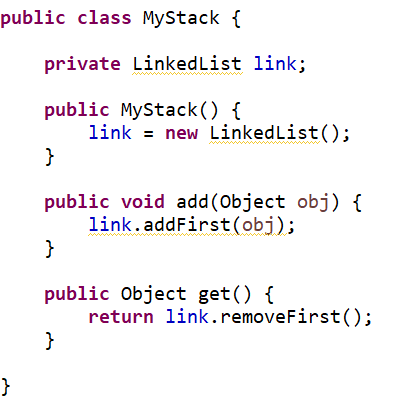
参考以上，注意重写equals()方法

面试题：

请用LinkedList模拟数据结构的集合，并测试

题目的意思是：

你自己定义一个集合类，在这个集合类内部使用LinkedList模拟



以上代码，很好的模拟了栈的数据结构，因为在代码中可以知道，压栈操作进去的时候是在第一个位置加，弹栈的时候也是得到第一个元素，弹栈之后需要删除该元素，所以get()方法中要用removeFirst()方法

此面试题的思想：让我们用自己设计的类来模拟Stack类的方法，此时我们在设计类的时候，其实代码里面包装的也是Stack类的方法，这种思想经常会用到

泛型

当一个集合同时存入不同类型的对象时

例如：

list.add(“Hello”);

list.add(“JaveSE”);

list.add(new Person());

编译的时候没有问题，但是当用迭代器遍历的时候会出现问题，这样不好，所以在创建对象的时候如果能明确集合中的元素类型，就不会出现这样的问题了，于是Java提供了泛型

泛型：是一种把明确的工作推迟到创建对象的时候才去明确的特殊的型。参数化类型，把类型当做参数一样的传递

格式：<数据类型>（这里的数据类型只能是引用类型）

注意：集合是什么类型，迭代器必须是一样的类型

泛型使用地方：看API，如果类，接口，抽象类后面跟了<E>，就说要用泛型，一般来说就在集合类中使用

好处

A:把运行时期的问题提前到了编译期间

B:避免了迭代器遍历时的强转

C:优化了程序设计，解决了黄色警告线

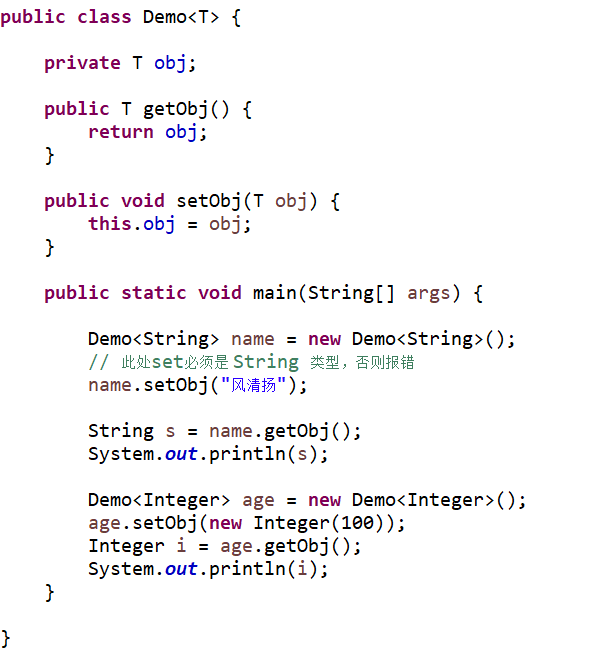
泛型的应用

泛型类

把泛型定义在类上

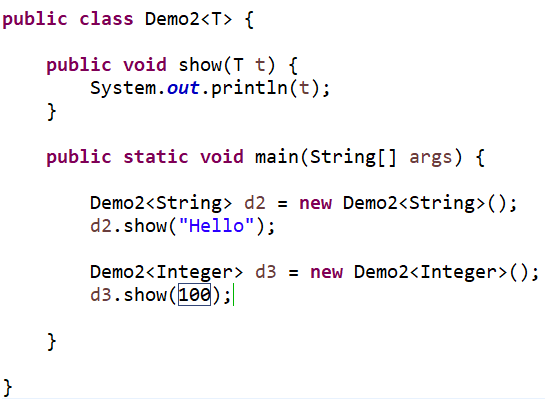
格式：public class 类名<泛型类型1…>

注意：泛型类型必须是引用类型



以上案例，当用泛型定义了一个类的时候，可以作为一种类型应用到成员变量和方法上

泛型类的另一个应用：



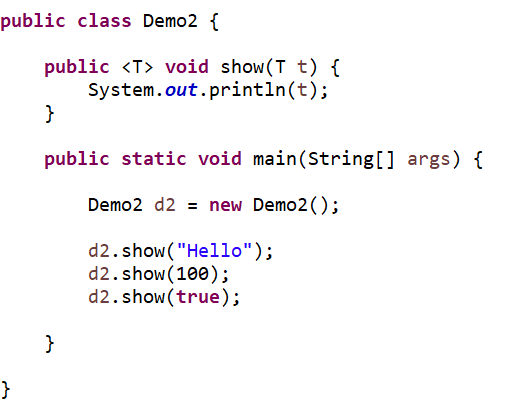
传入什么类型，都可以，否则无法输出多种类型，但是有一个弊端，每次调用不同的数据类型都有创建一个新的对象

泛型方法

定义：把泛型定义在方法中

格式：public <T> void show(){}

改进上述代码：



把上述案例中的泛型定义在类上，这时候不需要创建多个对象，直接可以使用任何类型

泛型接口

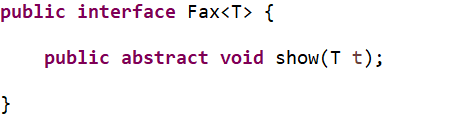
把泛型定义在接口上

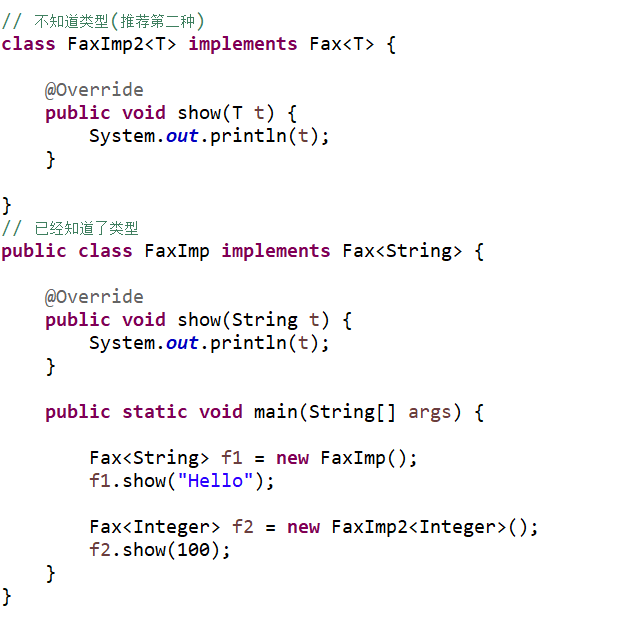
实现类在实现接口的时候有两种情况：

A:实现类定义的时候已经知道是什么类型了（了解）

B:实现类定义的时候还不知道是什么类型（常用）

接口：





泛型高级之通配符（day16\_18）

?:

任意类型，如果没有明确，那么就是Object以及任意的Java类了

? extends E:

向下限定，E及其子类

? super E:

向上限定，E及其父类

JDK5新特性：

自动拆装箱；泛型；增强for；静态导入；可变参数；枚举

增强for循环

格式：for(元素数据类型 变量 : 数组或者Collection集合){

使用变量即可，改变量就是元素;

}

好处：简化了数组和集合的遍历

弊端：增强for的目标不能为null

解决：对增强for的目标先进行不为null的判断，然后再使用

增强for其实是用来替代迭代器的，通过反编译代码可知

静态导入（了解，用不到）

格式：import static 包名…类名.方法名

可以直接导入到方法的级别

import static java.lang.Math.max;

System.out.println(max(20,30));

注意事项：

方法必须是静态的

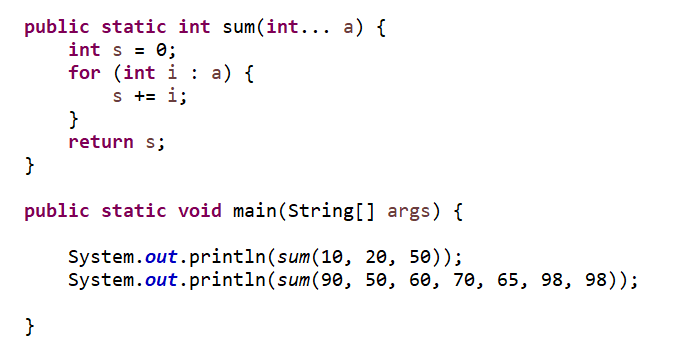
如果有多个同名的静态方法，容易不知道使用谁?这个时候要使用，必须加前缀。由此可见，意义不大，所以一般不用，但是要能看懂。

可变参数

定义方法的时候不知道该定义多少个参数

格式：修饰符 返回值类型 方法名（数据类型… 变量名）

这里的变量其实是一个数组



当不知道参数个数的时候，可以使用可变参数

注意：

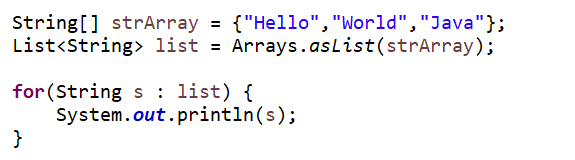
这里的变量其实是一个数组

如果一个方法有可变参数，并且有多个参数，那么，可变参数肯定是最后一个

Arrays类的asList()方法

public static <T> List<T> asList(T… a)

把数组转换成集合



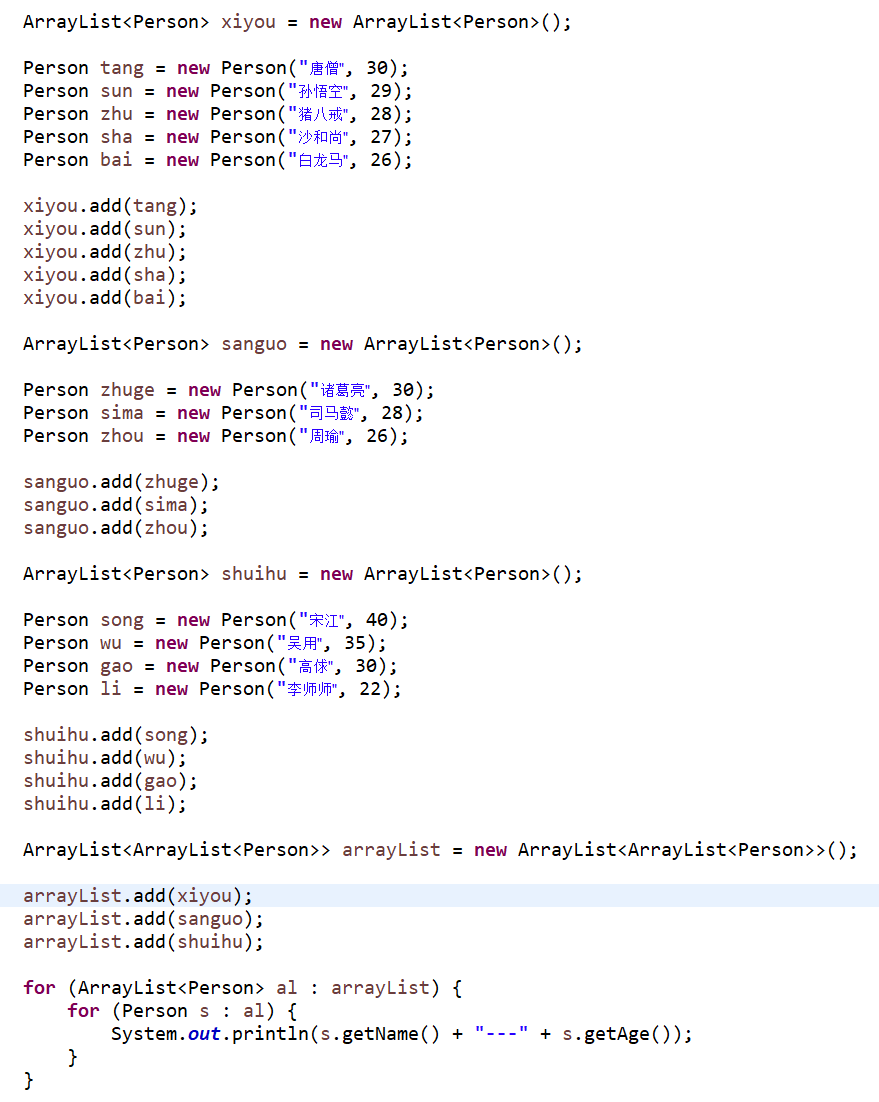
注意事项：虽然可以把数组转换成集合，但是集合长度不能改变（可以修改，不能增删），因为其本质还是数组

集合的嵌套遍历

需求：

一个班级的学生用集合表示：ArrayList<Student>

每个班级又用集合表示:ArrayList< ArrayList<Student>>



ArrayList对象的toString()方法

在源码里面没有看到toString()方法，这时候需要去其父类里面找

Set集合

Set集合概述：

一个无序的，不包含重复元素的collection

Set集合特点：

无序，唯一

HashSet类（掌握）

HashSet类概述

不保证 set 的迭代顺序

特别是它不保证该顺序恒久不变

HashSet如何保证元素唯一性

源码解析（day17\_12）

底层数据结构是哈希表(元素是链表的数组)

哈希表依赖于哈希值存储

添加功能底层依赖两个方法：

int hashCode()

boolean equals(Object obj)

执行顺序：

首先比较哈希值是否相同

哈希值相同：继续执行equals()方法

返回true:元素重复了，不添加

返回false:直接把元素添加到集合

哈希值不同：直接把元素添加到集合中

按照方法的步骤来说：

先看hashCode()是否相同，如果不同就直接把元素添加到集合，如果相同继续走equals()方法，返回true说明元素重复，就不添加，返回false说明元素不重复，就添加到集合，

如果类没有重写这两个方法，默认使用Object()，一般来说不相同，而String类重写了hashCode()和equals()方法，所以，它就可以把内容相同的字符串去掉，只留下一个

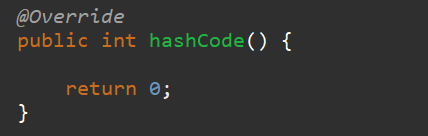
案例：

存储自定义对象，并保证元素的唯一性（如果两个对象的成员变量值都相同，则判定为同一个元素）

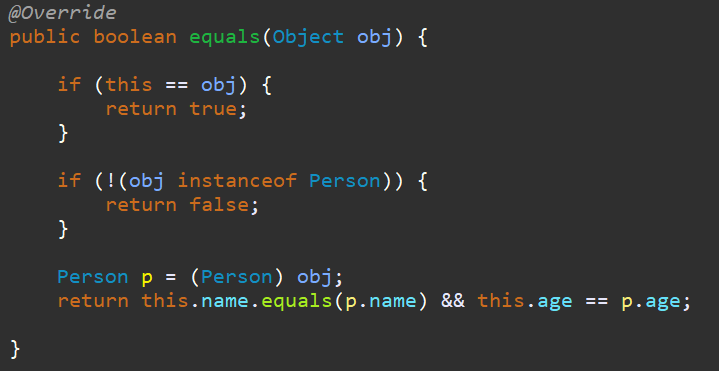
实现：

正常创建Person类，不重写hashCode()和equals()方法，然后直接添加到HashSet()集合中，这个时候，因为默认使用的是Object类的hashCode()和equals()方法，他们的哈希值是不会一样的，根本就不会继续判断，直接执行添加操作

所以现在需要做的是，将Person类中的hashCode()方法和equlas()方法重写，因为在底层源码中，我们知道，set集合添加元素的时候，首先判断的是hashCode值是否相同，这是，我们重写hashCode()



固定返回0，所以每个对象的hashCode值都相同了，再来重写equals()方法

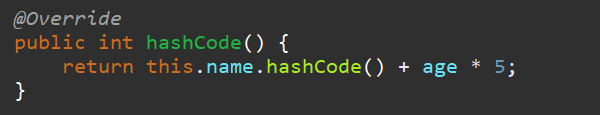


重写了以上两个方法之后，set在添加元素的时候，只要Person对象的name和age相同，就属于同一元素，就不会被添加进集合。

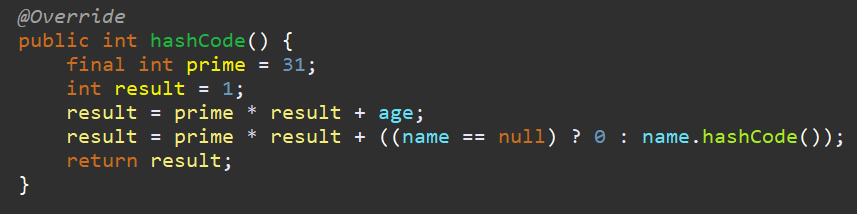
基本原理：先判断hashCode，然后equals

如果两个对象hashcode相等，他们不一定equals  
如果两个对象hashcode不相等，他们一定不equals

但上述重写hashCode()方法直接返回0，这样不够好，因为这样会把所有任务都交给后面的equals()放去逐个比较，效率很低，所以要在hashCode方法中添加算法，用Person类举例，我们可以把对象的name的hashCode值，加上age的值作为最终的hashCode值，但是这样也会有误差，因为20+50和30+40的值是一样的，所以，我们将对象的name的hashCode值，加上5倍的age的值，这个时候，就可以将hashCode不同的值直接判断为不是同一个对象，把相同的进一步equals，大大提高效率



Eclipse提供了自动重写



大概意思相同，都是利用name的hashCode值和age的值进行操作

LinkedHashSet类概述

元素有序唯一

底层数据结构由哈希表和链表组成

由链表保证元素有序（存储和取出一致）

由哈希表保证元素唯一

TreeSet类

TreeSet类概述

能够对元素按照某种规则进行排序（两种）

A:自然排序

B:比较器排序

具体排序方法取决于使用的构造方法

TreeSet是如何保证元素的排序和唯一性的？（day17\_19）

以Integer类的数据为例

底层的数据结构是红黑树（红黑树是一种自平衡的二叉树）

自平衡的二叉树，意思是自动平衡，每个节点下会自动平衡分配两个元素，不会出现某一分支有多个元素，而某一分支没有元素的情况

TreeSet的元素是如何进去的呢

第一个元素存储的时候，直接作为根节点存储

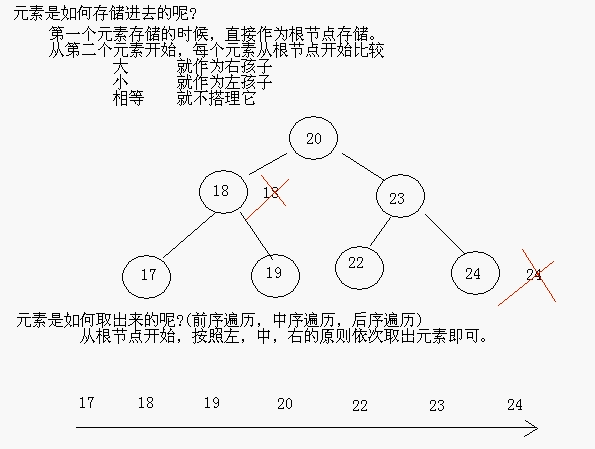
从第二个元素开始，每个元素从根节点开始比较

比根节点大，作为右孩子；比节点小，作为左孩子；相等就不添加

此时，存储的时候已经确定了唯一性

TreeSet的元素是如何取出的呢

从节点开始，按照左中右的原则依次取出元素即可，如下图所示



以自定义对象为例存入TreeSet详解(day17\_20)

TreeSet存储自定义对象并保证排序和唯一

正常操作步骤：

1. 定义Person类（name,age,getter,setter，构造方法）
2. 创建TreeSet<Person>集合
3. 创建多个Person对象
4. 把Person对象添加到集合中

但是这样会报错，因为没有给出自然排序标准

遇到问题：需要给出标准

A:按照年龄从小到大排序

B:成员变量都相同即为同一个元素

首先，我们需要知道：

真正的比较是依赖于元素的compareTo()方法，而这个方法是定义在Comparable接口里面的

compareTo()方法返回正数，则判断为大，返回负数，则判定为小，返回0，则判定两个元素相等。

如果一个类的元素想要能够进行自然排序，就必须实现自然排序接口Comparable<E>

实现接口并重写接口中的方法

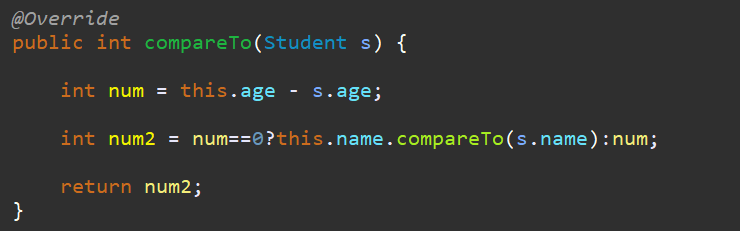
public int CompareTo(Person p){

int num = this.age – p.age;

return num;

}

上述重写了方法之后，元素会比较年龄，年龄大的排在后面，年龄小的排在前面，年龄相同的不添加（具体步骤前面说过了）但是这种方法并不合理，因为年龄相同就不添加了，没有考虑到姓名可能不同；所以还需要考虑姓名是否相同



上述代码解释：先比较年龄是否相同，当年龄相同时，还需要比较姓名是否相同，因为String类重写了compareTo()方法，所以可以直接调用，比较完name之后，如果返回一个int类型给num2，实际上的意思就是当年龄相同时，比较姓名，如果姓名相同，则整个函数返回0，不添加元素，如果姓名不同，再按照姓名的String排序

总结：TreeSet集合存储一个自定义对象

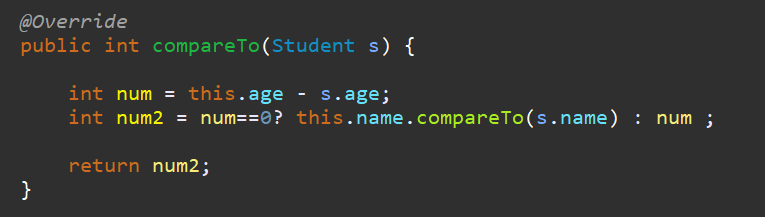
当我们需要用TreeSet集合存储一个自定义对象的时候按照正常操作逻辑，会报错：java.lang.ClassCastException

报错的原因是因为TreeSet是有序的，但是我们正常逻辑操作的时候并没有给出排序规则，所以这个时候我们定义规则：

A:根据年龄来排序

B:年龄和姓名相同，则判定为同一个元素

我们知道，实现自然排序依赖的是compareTo()方法，所以这个时候，我们需要把Student类实现Comparable接口并重写里面的compareTo()



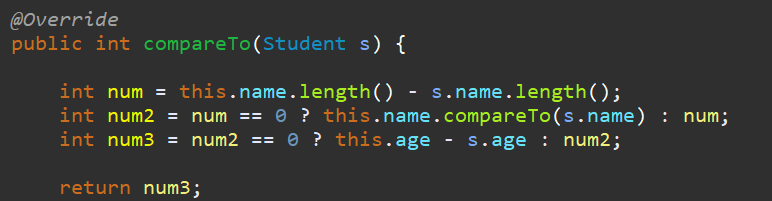
首先判断年龄是否相等，如果不相等，则返回num的值，根据这个值来进行排序；如果相等，再比较姓名，因为String类重写过了compareTo方法，所以可以直接调用，再把name的比较结果返回，如果name比较结果为0，说明姓名年龄都相等，不添加，如果不相等，再在年龄相等的基础上，按照姓名排序

案例:

定义学生类，根据姓名长度排序

此案例注意的问题是：

主要条件是姓名长度，所以需要比较姓名长度；但是仅此比较是不够的，因为qiqi和xuxu的长度一样；所以比较万长度，还需要比较内容是否一样，比较完姓名内容之后，还是不行，因为qiqi,25和qiqi,29也不是同一个人，所以还需要比较年龄那么代码如下：



这样才比较完整全面的进行了排序

以上是自然排序

下面介绍比较器排序：

因为两种排序取决于构造方法，上面说的都是用的无参构造，进行的是自然排序，下面要说用带参数的构造方法来实现比较器排序

day17\_22过程复杂，不说了，直接上代码

开发中偶尔用一次，使用匿名内部类来实现这个问题



这样就不需要自己再另外写一个类了

最后小总结：

TreeSet集合保证元素排序和唯一性的原理

唯一性：是根据比较的返回是否是0来决定的

因为添加元素的时候都是会调用compareTo方法的，这个方法通过给定的要求比较，返回一个int值，相同返回0

排序：两种方式

A:自然排序：（让元素具备比较性）

让元素所属的类实现自然排序接口Comparable

B:比较器排序：（让集合具备比较性）

让集合的构造方法接收一个比较器接口的子类对象Comparator

练习：录入五个学生信息（姓名，语文、数学、英语成绩）按照总分从高到低排序

Collection集合总结

Collection

Set

HashSet

TreeSet

Collection:

List 有序（指存取顺序，不是排序），可重复

ArrayList:

底层数据结构是数组，查询快，增删慢

线程不安全，效率高

Vector:

底层数据结构是数组，查询快，增删慢

线程安全，效率低

LinkedList:

底层数据结构是链表，查询慢，增删快

线程不安全，效率高

Set 无序（指存取顺序），唯一

HashSet:

底层数据结构是哈希表

如何保证元素唯一性呢？

依赖两个方法：hashCode()和equals()

开发中自动生成这两个方法即可

LinkedHashSet:

底层数据结构是链表和哈希表

由链表保证元素有序

由哈希表保证元素唯一

TreeSet:

底层结构是红黑树

如何保证元素排序的呢？

自然排序

比较器排序

如何保证元素唯一性呢？

根据比较的返回值是否是0来决定

针对Collection集合，我们到底使用谁？

唯一吗？

元素要求唯一：Set

要排序吗？

要排序：TreeSet

不排序：HashSet

如果你知道是Set，但是不知道是哪个Set，就用HashSet

元素不要求唯一：List

要安全吗？

要安全：Vector

不需要安全：ArrayList或者LinkedList

查询多：ArrayList

增删多：LinkedList

如果你知道是List，但不知道是哪个List，就用ArrayList

如果你知道是Collection集合，但是不知道用谁，就用ArrayList

如果你知道用集合，就用ArrayList

在集合中常见的数据结构：

ArrayXxx:底层数据结构是数组，查询快，增删慢

LinkedXxx:底层数据结构是链表，查询慢，增删快

HashXxx:

底层数据结构是哈希表；依赖两个方法：hashCode()和equals()

TreeXxx:

底层数据结构是二叉，两种排序方式：自然排序和比较器排序

Map集合

Map集合概述

将键映射到值的对象，一个映射不能包含重复的键；每个键最多只能映射到一个值；

Map集合的键可以理解为一个Set集合，键唯一，无序

Map集合的值可以理解为一个List集合，可以重复

Map集合和Collection集合的区别：

Map集合存储的元素是成对出现的

Map集合的键是唯一的，值是可以重复的

Map集合的数据结构只针对键有效，跟值无关

Collection集合存储元素是单独出现的

Collection的儿子Set是唯一的，List是可重复的

Collection集合的数据结构是针对元素有效的

Map集合的功能概述

1:添加功能

V put(K key,V value)

添加元素

如果键是第一次存储，就直接存储元素，返回null

如果键不是第一次存储，就用值把以前的值替换掉，返回以前的值

2:删除功能

void clear()

移除所有的键值对元素

V remove(Object key)

根据键删除键值对元素，并把值返回（没有该键返回null）

3:判断功能

boolean containsKey(Object key)

判断集合是否包含指定的键

boolean containsValue(Object value)

判断集合是否包含指定的值

boolean isEmpty()

判断集合是否为空

4:获取功能

Set<Map.Entry<K,V>> entrySet()

返回的是键值对对象的集合

V get(Object key)

根据键获取对应的值

Set<K> keySet()

获取集合中所有键的集合

Collection<V> values()

获取集合中所有值的集合

5:长度功能

int size()

返回集合中的键值对的对数

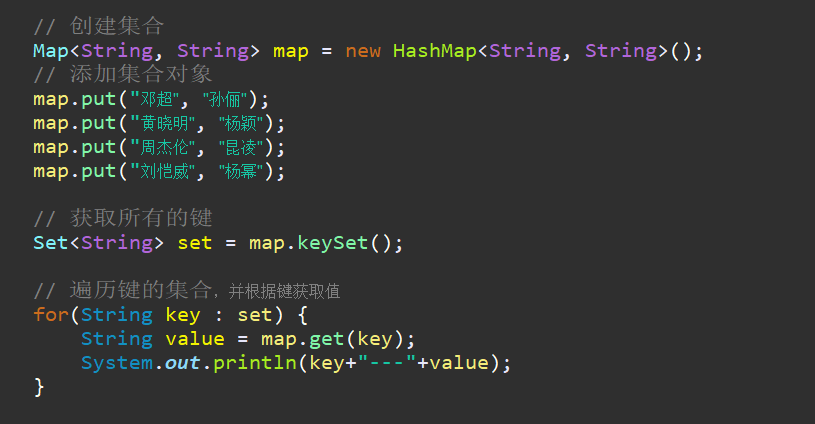
Map集合的遍历

方式一思路：

A:获取所有的键

B:遍历键的集合，获取得到每一个键

C:根据键去找值

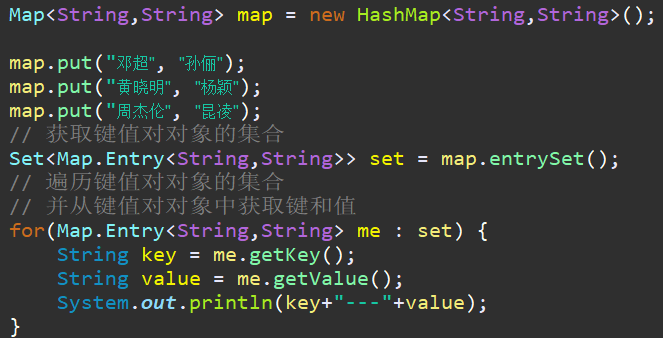


方式二思路：

A:获取所有键值对对象的集合

B:遍历键值对对象的集合，得到每一个键值对对象

C:根据键值对对象获取键和值



HashMap类

HashMap类的概述

特点:

线程不安全，效率高

是基于哈希表的Map接口实现

哈希表的作用是从来保证键的唯一性的

HashMap<String,String>

键：String 值：String

键唯一是通过hashCode()和equals()两个方法保证的

当自定义类Person作为键时候，要想键唯一（姓名年龄相同），只需重写两个方法即可

LinkedHashMap类

概述

是Map接口的哈希表和链表实现，具有可预知的迭代顺序

特点：

由哈希表保证键的唯一性

由链表保证键的有序（存储和取出顺序一致）

Hashtable类

底层数据结构是哈希表，线程安全，效率低

TreeMap类

TreeMap类概述

基于红黑树的Map接口的实现，可以保证键的排序和唯一性

集合的嵌套遍历

HashMap嵌套HashMap

HashMap嵌套ArrayList

ArrayList嵌套HashMap

HashMap嵌套HashMap嵌套HashMap(三层嵌套)

Collection和Collections的区别

Collection是单列集合的顶层接口，有两个子接口List和Set

Collections是针对集合进行操作的工具类，可以对集合进行排序和查找等

Collections工具类

Collections概述

针对集合操作的工具类，都是静态方法

Collections成员方法

public static <T> void sort(List<T> list)

对集合排序（默认情况下是自然排序,要求元素实现自然排序接口）

public static <T> void sort​

([List](mk:@MSITStore:G:\JAVA\API\JDK_1.9-ZH.CHM::/java/util/List.html)<T> list,[Comparator](mk:@MSITStore:G:\JAVA\API\JDK_1.9-ZH.CHM::/java/util/Comparator.html)<? super T> c)

对集合排序，需要传入一个比较器

public static <T> int binarySearch(List<?> list,T key)

二分查找

public static <T> T max(Collection<?> coll)

最大值

public static void reverse(List<?> list)

反转

public static void shuffle(List<?> list)

随机置换（每调用一次，就随机调换元素位置）

到底选用哪种集合：看需求

是否是键值对形式：

是：Map

键是否需要排序：

是：TreeMap

否：HashMap

不知道就使用HashMap

否：Collection

元素是否唯一：

是：Set

元素是否需要排序：

是：TreeSet

否：HashSet

不知道就是用HashSet

否：List

要安全吗:

是：Vector

否：ArrayList或者LinkedList

增删多：LinkedList

查询多：ArrayList

不知道就是用ArrayList

不知道就是用ArrayList

集合的常见方法及遍历方式：

Collection:

add()

remove()

contains()

iterator()

size()

遍历：增强for 迭代器

list多一个普通for遍历，通过get(i)方法

Map:

put()

remove()

containsKey(),containsValue()

keySet():获取键的集合

get(Object key):根据键获取值

values():获取值的集合

entrySet():获取键值对对象

size()

遍历：

A:根据键找值

B:根据键值对对象分别找键和值

异常

异常：异常就是Java程序在运行过程中出现的错误

程序的异常：Throwable

错误：Error

（错误我们不处理，这种问题一般都是很严重的问题，比如内存溢出）

异常：Exception

编译期问题：不是RuntimeException的异常，必须进行处理的

编译期问题不处理编译无法通过

运行期问题：RuntimeException

（RuntimeException这种问题不处理，因为是用户的问题，而且这个问题出现肯定是我们的代码不够严谨，需要修正代码）

Throwable

Error

Exception

子类

子类

RuntimeException

子类

如果程序出现问题，问我们没有做任何处理，最终JVM会做出默认的处理：把异常的名称，原因及出现的问题等信息输出在控制台；同时程序会结束

我们觉得JVM的默认处理方式不太好，因为会结束程序，我们不想结束，所以我们就可以自己处理

异常处理方案

A:try…catch…finally

B:throws----抛出

try…catch…finally处理方式

格式：

try{

可能出现问题的代码;

}catch(异常名 变量){

针对问题的处理;

}finally{

释放资源;

}

变形格式： try{

可能出现问题的代码;

}catch(异常名 变量){

针对问题的处理;

}

注意：

A:try里面的代码越少越好

B:catch里面必须有内容，哪怕是给出一个简单的提示

处理多个异常

多个异常：

a:给没一个异常写一个try…catch

b:写一个try，多个catch

try{…

}catch(异常类名1 变量名){…

}catch(异常类名2 变量名){…

}

注意：一旦try里面出现问题，就会在这里把问题抛出去（在try里的后面的语句不再执行），然后和catch里面的问题进行匹配，一旦有匹配的，就执行catch里面的处理，然后就结束了try…catch，继续执行try…catch后面的语句

注意：处理多个异常有多个catch时

A:能明确异常类型的在catch中尽量明确异常类型

B:平级关系的异常谁在前后无所谓，如果出现了父子关系的异常，父异常必须在后面

Exception是所有异常的父类，如果不明确异常类型，可以用Exception来匹配（只能放在最后）

JDK7新特性

try{

}catch(异常名1 | 异常名2 | … 变量){

出现异常的操作;

}

注意事项：JDK7新特性异常虽然简洁，但是不够好

A:处理方式是一致的（实际开发中，很多时候可能就是针对同类型的问题，给出同一个处理）

B:多个异常间必须是平级关系

编译期异常和运行期异常的区别：

编译期异常：Java程序必须显示处理，否则程序就无法通过编译

运行期异常：无需显示处理，也可以和编译时异常一样处理

注意：只要不是运行时异常，就都是编译期异常，必须处理

在try里面发现问题后，JVM会帮我们生产一个异常对象，然后把这个对象抛出，和catch里面的类进行匹配；如果该对象是某个类型的，就会执行该catch里面的处理信息

异常中需要了解的几个方法：

public String getMessage()

获取异常信息，返回异常的消息字符串

public String toString()

返回异常的简单信息描述，返回格式：

此对象的类的name(全路径名) “: ”(冒号和一个空格)

调用此对象getLocalizedMessage()方法（默认返回的是getMessage()的内容）

public void printStackTrace()

获取异常类名的异常信息，以及异常出现在程序中的位置，返回值void，把信息输出在控制台

printStackTrace(PrintStream s)

通常用改方法将异常内容保存在日志文件中，以便查阅

throws（第二种处理方式）

有些时候，我们是可以对异常进行处理的，但是又有些时候，我们根本就没有权限去处理某个异常，或者说，我处理不了，那就不处理；为了解决出错问题，Java针对这种情况，提供了另一种处理方案

抛出

格式：throws 异常类名

注意：这个格式必须跟在方法的括号后面，在方法声明上抛出，是为了告诉方法调用者，有问题

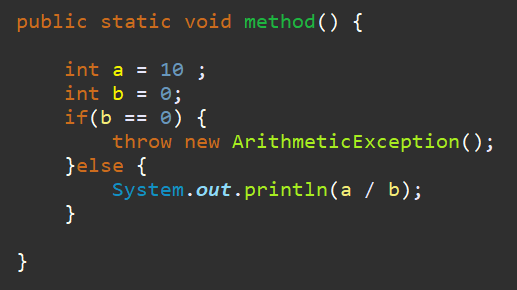
当我们没有能力或者没有权限去处理时，使用此方法，在方法的括号后面抛出异常，调用者必须解决该异常或者继续抛出，但是尽量不要在main方法上抛出异常

throws后面可以跟多个异常类型

throw

在功能方法内部出现某种情况，程序不能继续运行，需要进行跳转时，就用throw把异常对象抛出，这个时候抛出的应该是异常的对象

举例：



throw和throws区别

throws：

用在方法后面，跟的是异常类名

可以跟多个异常类名，用逗号隔开

表示抛出异常，由该方法的调用者来处理

throws表示出现异常的一种可能性，并不一定会发生这些异常

throw：

用在方法体内，跟的是异常对象名

只能抛出一个异常对象名

表示抛出异常，由方法体内的语句处理

throw则是抛出了异常，执行throw则一定抛出了某种异常

我们到底该如何处理异常呢？

原则:

如果该功能内部可以将问题处理,用try,如果处理不了,交由调用者处理,这是用throws

区别:

后续程序需要继续运行就try

后续程序不需要继续运行就throws

举例:

感冒了就自己吃点药就好了,try

吃了好几天药都没好结果得了H7N9,那就得throws到医院

如果医院没有特效药就变成Error了

finally

被finally控制的语句一定会执行

注意：如果在执行到finally之前JVM退出了，就不能执行了

A:格式 try…catch…finally

B:作用：用于释放资源，在IO流或者数据库操作会用到



自定义异常(非常简单)

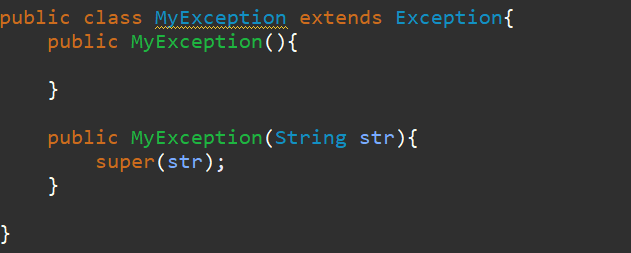
Java定义了很多常见的异常，但是有一些特定的异常他没有提供，所以我们需要自定义异常

自定义异常：重点要继承

继承自Exception

继承自RuntimeException

案例：考试成绩必须在0-100之间





异常的注意事项

A:子类重写父类方法时，子类的方法必须抛出相同的异常或父类异常的子类。(父亲坏了,儿子不能比父亲更坏)

B:如果父类抛出了多个异常,子类重写父类时,只能抛出相同的异常或者是他的子集,子类不能抛出父类没有的异常

C:如果被重写的方法没有异常抛出,那么子类的方法绝对不可以抛出异常,如果子类方法内有异常发生,那么子类只能try,不能throws

File类

File类概述

文件和目录路径名的抽象表示形式

构造方法

File(String pathname)

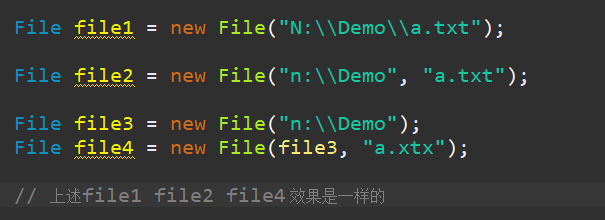
根据一个路径得到File对象

File(String parent,String child)

根据一个目录和一个子文件/目录得到File文件

File(File parent,String child)

根据一个父File对象和一个子文件/目录得到File对象



File类的方法

创建功能

public boolean mkdir()

创建文件夹，如果父文件夹不存在，就帮你创建出来

public boolean mkdirs()

创建文件夹，如果存在这样的文件夹，就不创建了

public boolean creatNewFile()

创建文件，如果存在这样的文件夹，就不创建了

要想在某个目录下创建内容，该目录必须存在，否则就会报错



注意：要注意方法的使用，到底是创建文件还是文件夹

删除功能

public boolean delete()

删除文件或者文件夹

注意：

A:如果创建文件或者文件夹没有盘符路径，那么，默认在项目路径下

B:Java中的删除不走回收站

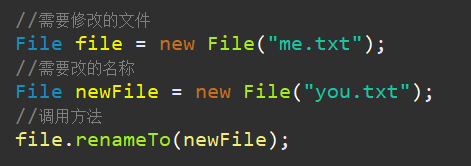
C:要删除一个文件夹，该文件夹下不能有东西



File类的重命名功能

boolean renameTo(File dest)

重命名文件，如果前后两个路径名相同，就是改名，如果路径不同，就是改名并剪切



路径以盘符开始：绝对路径 C:\\a.txt

路径不以盘符开始：相对路径 a.txt

File类判断功能

public boolean isDirectory()

判断是否是目录（文件夹）

public boolean isFile()

判断是否是文件

public boolean exists()

判断是否存在

public boolean canRead()

判断是否可读

public boolean canWrite()

判断是否可写

public boolean isHidden()

判断是否隐藏

File类获取功能

基本获取功能

public String getAbsolutePath()

获取绝对路径

public String getPath()

获取相对路径

public String getName()

获取名称

public long length()

获取长度，字节数

public long lastModified()

获取最后一次的修改时间，返回毫秒值

高级获取功能

public String[] list()

获取指定文件夹下的所有文件或者文件夹的名称数组

public File[] listFiles()

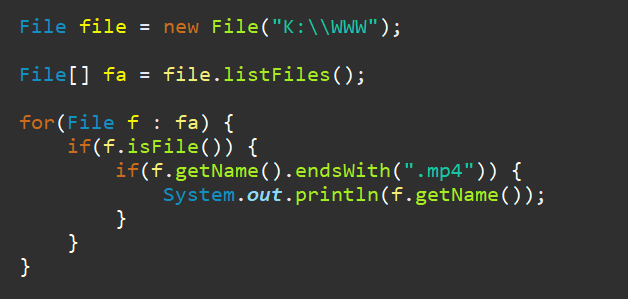
获取指定文件夹下的所有文件或者文件夹的File数组

练习：判断K:\WWW下是否有.mp4结尾的文件，有就输出文件名称

思路：

A:获取File数组，遍历File数组，判断是否是文件

B:判断文件名是否以 .mp4 结尾，是就输出



练习：判断K:\WWW下是否有.mp4结尾的文件，有就输出文件名称

思路

思路2：

A:将满足条件的 .mp4 结尾的文件添加到File数组

B:遍历数组，输出名称

要想实现这个效果，需要学习一个接口：

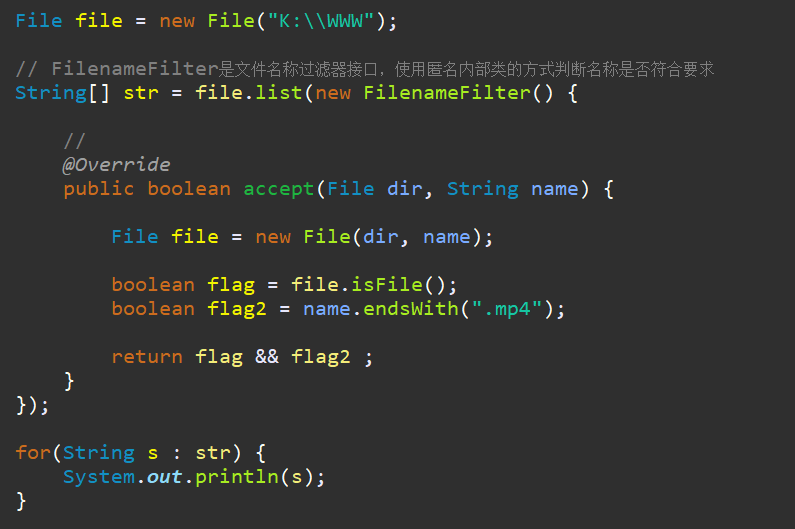
文件名称过滤器 🡪 FilenameFilter

public String[] list(FilenameFilter filter)

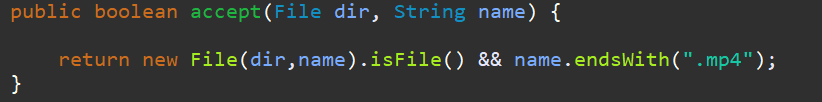
按照某种规则，获取一个目录所有文件和文件夹的名称数组

public File[] listFiles(FilenameFilter filter)

按照某种规则，获取指定文件夹下的所有文件或文件夹的File数组



改进版（简化accept方法里的代码）



案例：把某个文件夹下的名称改为xxx\_名称.MP4

A:封装目录

B:获取该目录下所有的文件的File数组

C:遍历File数组，得到每一个File对象

D:拼接一个新的名称，然后重命名即可

递归

递归概述(内存模型详解day20\_03)

方法定义中调用方法本身的现象

注意事项：

A:递归一定要有出口，否则就是死递归

B:递归的次数不能太多，否则就内存溢出

C:构造方法不能递归调用

案例需求：请用代码实现5的阶乘

A:循环实现（忽略）

B:递归实现

a:做递归要写一个方法

b:出口条件

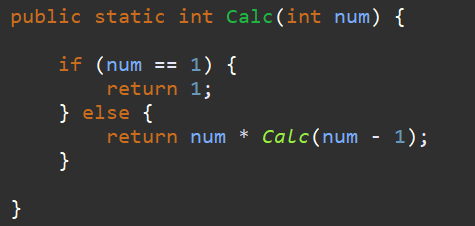
c:规律

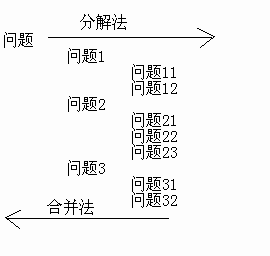
方法：返回值类型：int

参数列表：int n

出口条件：if(n == 1){return 1;}

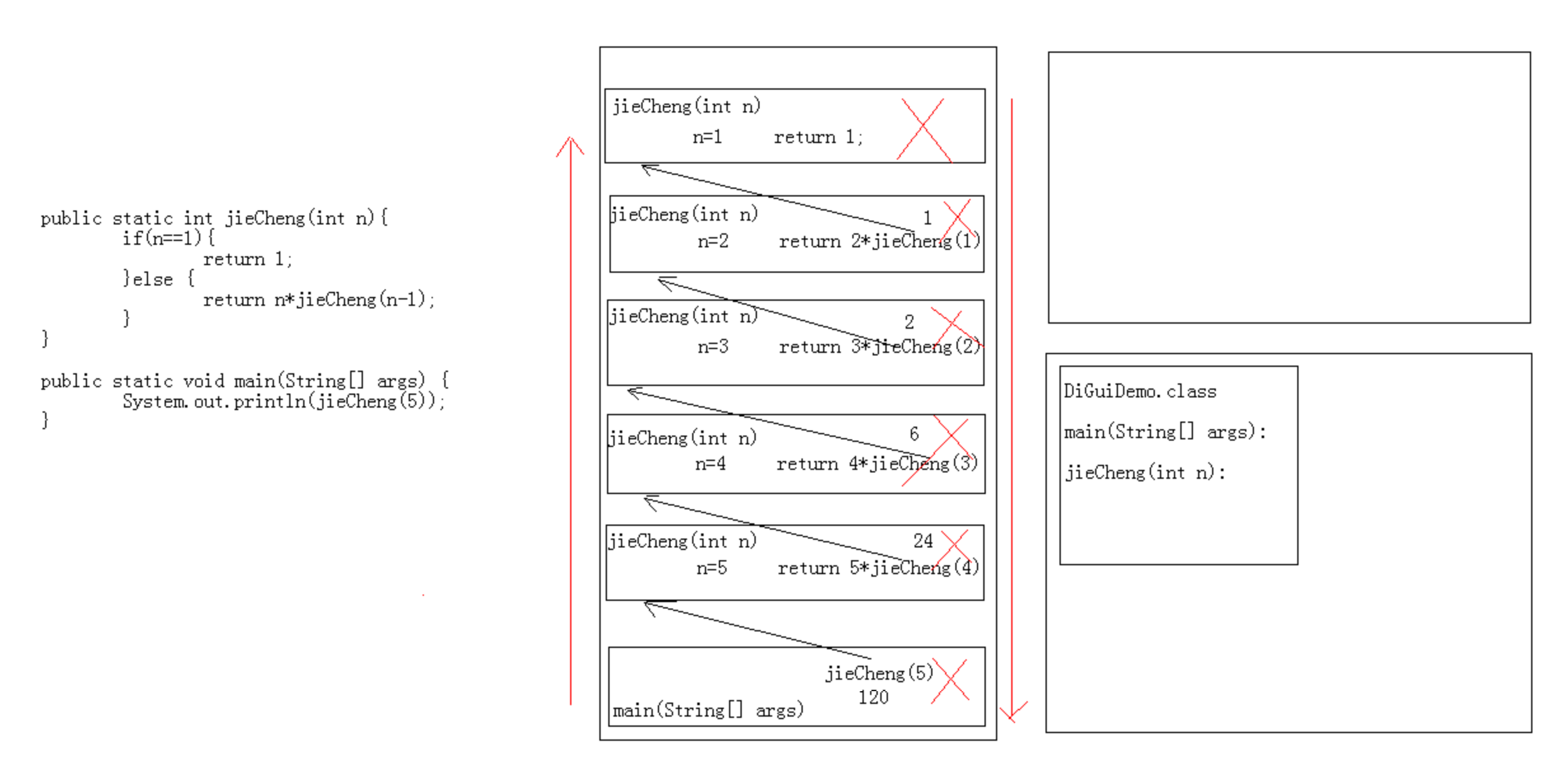
规律：if(n != 1){return n \* 方法名(n-1);}



递归解决问题的思想：

分解法 合并法

在内存中，每调用一次递归方法，在栈内都会开辟一个该方法的空间，遇到出口条件的时候，又会逐个的返回，之前开辟的空间会一个一个的销毁，通过断点查看，能非常好的理解递归

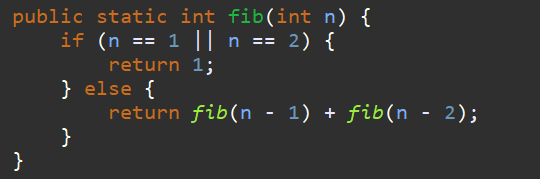


斐波那契数列用递归实现：

方法：返回值类型：int 参数列表：int n

出口条件：第一个月是1，第二个月是1

规律：从第三个月开始，每一个月是前两个月之和



需求：把N:\Project目录下所有的java结尾的文件的绝对路径输出在控制台

分析:

A:封装目录

B:获取该目录下所有的文件或者文件夹的File数组

C:遍历该File数组，得到每一个File对象

D:判断该File对象是否是文件夹

是：回到B

否：继续判断是否以.java结尾

是：输出绝对路径

否：不管它

需求：递归删除带内容的目录案例

分析：

A:封装目录

B:获取该目录下所有的文件或者文件夹的File数组

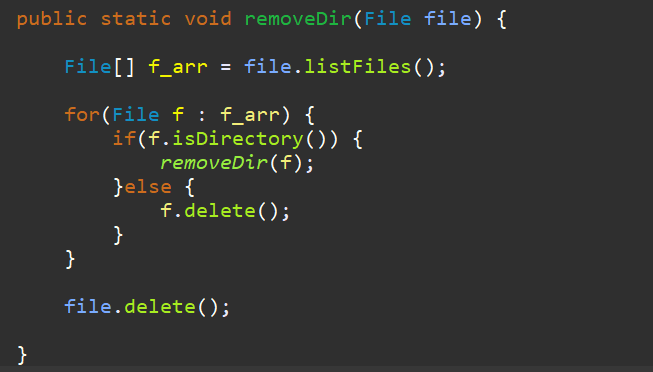
C:遍历该File数组，得到每一个File对象

D:判断该File对象是否是文件夹

是：回到B

否：删除

E:遍历一个文件夹，删除文件之后，即可删除该文件夹



IO流

IO流用来处理设备之间的数据传输

上传文件和下载文件

Java对数据的操作是通过流的方式

Java用于操作流的对象都在IO包下

Java中IO流的分了类

按流向分类：

输入流（读取数据）

输出流（写出数据）

默认按数据类型分类：

字节流

字节输入流（读取数据）

字节输出流（写出数据）

字符流：为了方便操作文本数据，Java就提供了字符流

字符输入流（读取数据）

字符输出流（写出数据）

那么我们到底使用哪种类型的流呢？

A:如果你操作的数据是文本数据，就用字符流；

B:把你要操作的文件用windows记事本打开，如果你打开数据后能读懂，就可以使用字符流，如果不能读懂，就用字节流；

C:如果什么都不知道，就用字节流

注意：没有明确说明按哪种分类，默认按照数据类型分

IO流常用的基类

字节流

字节输入流（读取数据）：**InputStream**

字节输出流（写出数据）：**OutputStream**

字符流：为了方便操作文本数据，Java就提供了字符流

字符输入流（读取数据）：**Reader**

字符输出流（写出数据）：**Writer**

需求：往一个文本文件里输入一句话：Hello,io;

分析：

A:这个操作最好是用字符流，但是这里先讲字节流

B:由于是往文件中写一句话，要采用字节输出流

通过分析：要使用OutputStream

但是这个是抽象类，不能实例化，所以我们需要一个他的子类

FileOutputStream:

注意：每种基类的子类都是以父类名作为后缀名

FileOutputStream类

构造方法：

FileOutputStream(File file)

利用文件对象创建

FileOutputStream(String name)

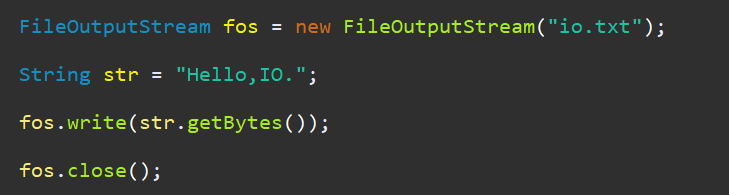
利用字符串创建（常用，查看源代码可知）

创建字节输出流对象做了几件事情？

A:调用系统功能去创建了文件

B:创建了字节输出流对象

C:把字节输出流对象指向了创建的文件



流对象操作完后，要释放资源，调用close()方法

调用close()方法作用：

A:让流对象变成垃圾，这样就可以被垃圾回收器回收了

B:通知系统去释放跟该文件相关的资源

字节输出流操作步骤：

A:创建字节输出流对象

B:写数据

C:释放资源

字节流写数据的方式

public void write(int b)

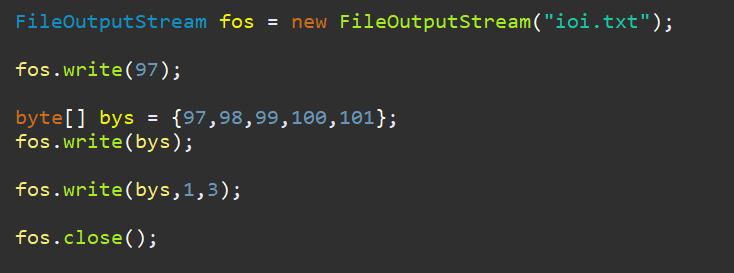
写入一个字节

public void write(byte[] b)

写入一个字节数组

public void write(byte[] b,int off,int len)

写入字节数组的一部分

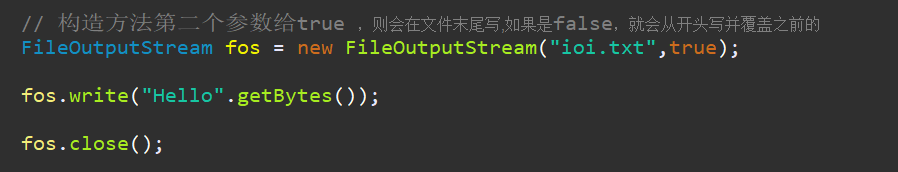


写入换行符

换行符的ACSII码是10，或者写入”\r\n”

追加写入

用构造方法带第二个参数是true



字节流写数据加入异常处理



FileInputStream类:字节读取流

步骤：

A:创建字节输入流对象

B:调用read()方法读取数据并显示在控制台

C:释放资源

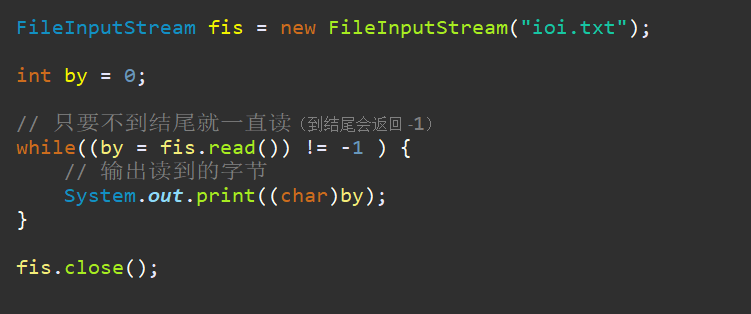
读取数据的方式：

int read()

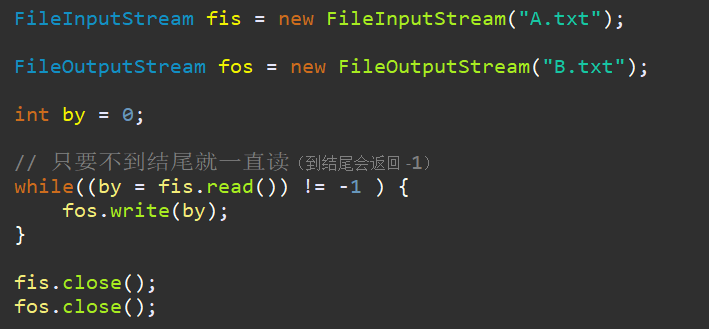
一次读取一个字节

int read(byte[] b)

一次读取一个字节数组



案例：把A.txt的内容复制到B.txt



计算机是如何识别什么时候该把两个字节转换为一个中文呢？

在计算机中中文的存储分为两个字节：

第一个字节肯定是负数；

第二个字节常见的是负数，也可能是正数，但是没影响；

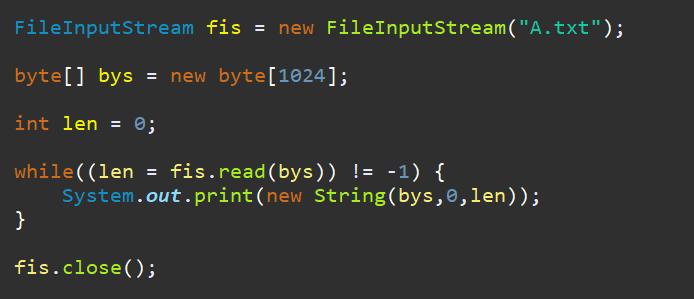
如果字节遇到负数，计算机就会自动拼接

上述方法一次读一个字节比较慢，所以需要一次读一个字节数组

int read(byte[] b)：一次读取一个字节数组

返回值其实是实际读取的字节个数：标准代码如下

理论上说，效率比之前一个一个的读高1024倍



视频详解：day20\_23

字节缓冲流

字节流一次读写一个数组的速度明显比一次读写一个字节的速度快很多，这是加入了数组这样的缓冲区效果，java本身在设计的时候，也考虑到了这样的设计思想(装饰设计模式后面讲解)，所以提供了字节缓冲区流

这种类被称为：缓冲区类（高效类）

写数据：BufferedOutputStream

读数据：BufferedInputStream



BufferedOutputStream类

构造方法：

[BufferedOutputStream](mk:@MSITStore:G:\JAVA\API\JDK_1.9-ZH.CHM::/java/io/BufferedOutputStream.html#BufferedOutputStream-java.io.OutputStream-)​([OutputStream](mk:@MSITStore:G:\JAVA\API\JDK_1.9-ZH.CHM::/java/io/OutputStream.html) out)

创建一个对象

[BufferedOutputStream](mk:@MSITStore:G:\JAVA\API\JDK_1.9-ZH.CHM::/java/io/BufferedOutputStream.html#BufferedOutputStream-java.io.OutputStream-int-)​([OutputStream](mk:@MSITStore:G:\JAVA\API\JDK_1.9-ZH.CHM::/java/io/OutputStream.html) out, int size)

这个构造方法可以指定缓冲区大小，但是我们一般用不上，因为默认缓冲区大小就足够了

构造方法参数是一个抽象类，为什么呢？

原因是因为字节缓冲区流仅仅提供缓冲区，为高效而设计的，但是呢，真正的读写操作还得靠基本的流对象实现

BufferedInputStream类

[BufferedInputStream](mk:@MSITStore:G:\JAVA\API\JDK_1.9-ZH.CHM::/java/io/BufferedInputStream.html#BufferedInputStream-java.io.InputStream-)​([InputStream](mk:@MSITStore:G:\JAVA\API\JDK_1.9-ZH.CHM::/java/io/InputStream.html) in)

创建一个对象

[BufferedInputStream](mk:@MSITStore:G:\JAVA\API\JDK_1.9-ZH.CHM::/java/io/BufferedInputStream.html#BufferedInputStream-java.io.InputStream-int-)​([InputStream](mk:@MSITStore:G:\JAVA\API\JDK_1.9-ZH.CHM::/java/io/InputStream.html) in, int size)

这个构造方法可以指定缓冲区大小，但是我们一般用不上，因为默认缓冲区大小就足够了

字符流

字节流读取中文可能出现的小问题：

由于字节流操作中文不是特别方便，所以，JAVA就提供了转换流

字符流=字节流+编码表

编码表：由字符及其对应的数值组成的一张表

常见编码表

ASCII：美国标准信息交换码，用一个字节的7位可以表示。

ISO8859-1：拉丁码表；欧洲码表，用一个字节的8位表示。

GB2312：中国的中文编码表。

GBK：中国的中文编码表升级，融合了更多的中文文字符号。

GB18030：GBK的取代版本

BIG-5码 ：台湾、香港地区的一个繁体字编码方案，俗称“大五码”。

Unicode：国际标准码，融合了多种文字；所有文字都用两个字节来表示,Java语言使用的就是unicode

UTF-8：最多用三个字节来表示一个字符，能用一个就用一个，一个表示不了的就用两个，两个不行就用三个

UTF-8不同，它定义了一种“区间规则”，这种规则可以和ASCII编码保持最大程度的兼容：

它将Unicode编码为00000000-0000007F的字符，用单个字节来表示  
它将Unicode编码为00000080-000007FF的字符用两个字节表示   
它将Unicode编码为00000800-0000FFFF的字符用3字节表示

String(byte[] bytes,String charsetName)

通过指定的字符集解码字节数组

byte[] getBytes(String charsetName)

使用指定的字符集合把字符串编码为字节数组

编码：把看得懂的变成看不懂的

String – byte[]

解码：把看不懂的变成看得懂的

byte[] – String

举例：发电报

编码问题简单，只要编码解码格式一致就不会有问题

OutputStreamWriter(OutputStream out)

根据默认编码把字节流的数据转换为字符流

OutputStreamWriter(OutputStream out，String charsetName)

根据指定编码把字节流数据转换为字符流

多线程

多线程概述：

要想了解多线程，必须先了解线程，而要想了解线程，必须先了解进程，因为您线程是依赖于进程而存在的

什么是进程：

通过任务管理器，我们看到了进程的存在，而通过观察，我们发现只有运行的程序才会出现进程

进程就是正在运行的程序，是系统进行资源分配和调用的独立单位，每一个进程都有它自己的内存空间和系统资源

多进程有什么意义呢？

单进程的计算机，只能做一件事情，而我们现在的计算机都可以做多件事情：比如说一边玩游戏（游戏进程），一边听音乐（音乐进程）

也就是说现在的计算机都是支持多进程的，可以在一个时间段内执行多个任务，并且可以提高CPU的使用率

问题：一边玩游戏，一边听音乐是同时进行的吗？

不是，因为CPU在某一个时间点上只能做一件事情，而我们在玩游戏或者在听音乐的时候，是CPU在做着程序间的高效切换，让我们觉得是同时进行的

什么是线程呢？

在一个进程内又可以执行多个任务，而每一个任务就是一个线程，线程是程序的执行单元，也叫执行路径，是程序使用CPU的最基本单位

单线程：如果程序只有一条执行路径

多线程：如果程序有多条执行路径

多线程有什么意义呢？

多线程的存在，不是提高程序的执行速度，其实是为了提高应用程序的使用率；程序的执行其实都是在抢CPU的资源，CPU的执行权，多个进程是在抢这个资源，而其中的某一个进程如果执行路径比较多，就会有更高的几率抢到CPU的执行权，我们是不敢保证哪一个线程能够在哪个时刻抢到，所以线程的执行有随机性

举例：扫雷程序，迅雷下载

并行和并发

前者是逻辑上同时发生，指在某一个时间内同时运行多个程序

后者是物理上同时发生，指在某一个时间点同时运行多个程序

Java程序运行原理

java 命令会启动 JVM，等于启动了一个应用程序，也就是启动了一个进程。该进程会创建启动一个 “主线程” ，然后主线程去调用某个类的 main 方法，所以 main方法运行在主线程中，在此之前的所有程序都是单线程的；

思考：JVM的启动是单线程的还是多线程的

答：多线程的，原因是垃圾回收线程也要先启动，否则很容易出现内存溢出，现在垃圾回收线程加上前面的主线程，最低启动了两个线程，所以JVM的启动是多线程的

需求：我们要实现多线程的程序

如果实现呢？

由于线程是依赖进程而存在的，所以我们应该先创建一个进程出来；而进程是有系统创建的，所以我们应该去调用系统功能创建一个进程；而Java是不能直接调用系统功能的，所以我们没有办法直接实现多线程程序，但是，Java可以去调用C/C++写好的程序来实现多线程程序，由C/C++去调用系统功能创建进程，然后由Java去调用这样的东西，然后提供一些类供我们使用，我们就可以实现多线程程序了

那么Java提供的是类是什么呢？

Thread类：有2种方式实现多线程程序，

方式1：继承Thread类

步骤：

A:自定义类MyThread继承Thread类

B:MyThread类里面重写run()

C:创建MyThread对象

D:启动线程对象

不是类中的所有代码都需要被线程执行，而这个时候，为了区分哪些代码能够被线程执行，Java提供了Thread类中的run()方法用来包含那些被线程执行的代码

一般来说，被线程执行的代码肯定是比较耗时的，过于简单的代码单独开一个线程来做有些浪费



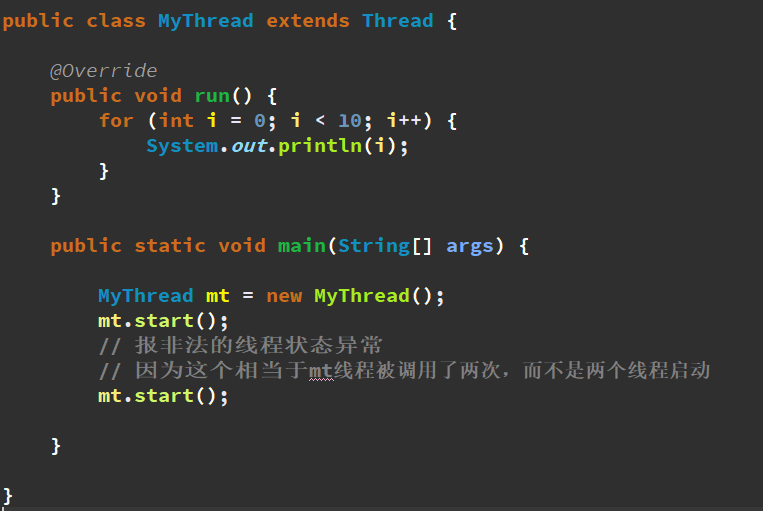
上述代码中，调用run()方法为什么是单线程的呢？

因为直接调用run()其实就相当于普通的方法调用，所以看到的就是单线程的效果

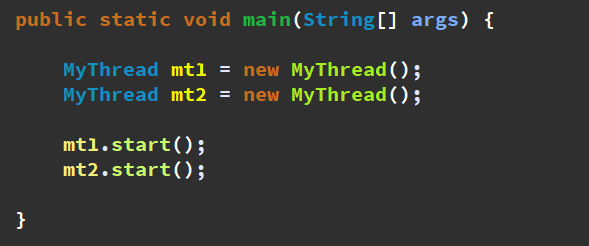
要想看到多线程的效果，就必须谈谈start()方法

void start()

首先启动了线程，然后由jvm去调用该线程的run()方法



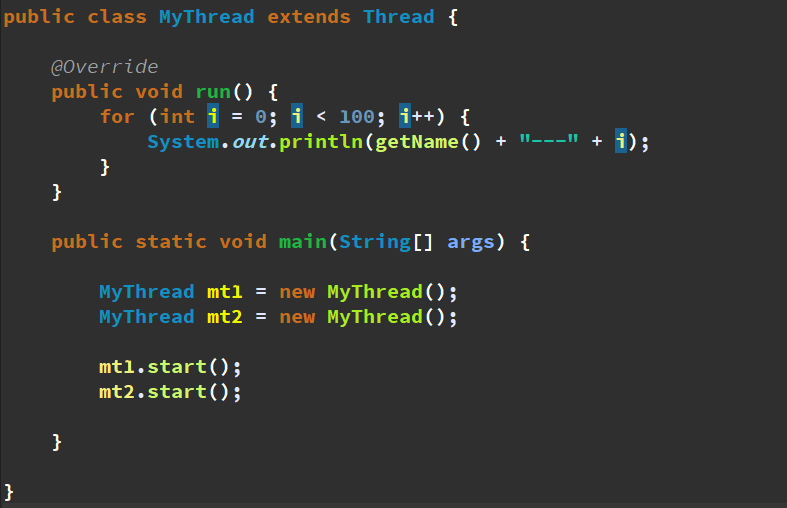
上述代码会报错，因为一个线程不能调用两次，要想实现多线程，需要创建两个对象



获取线程对象的名称：

public final String getName()

获取线程的名称

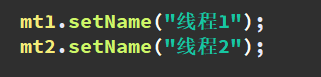


查看源码得知默认名称是：Thread - ?（编号）

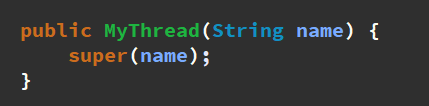
设置线程对象的名称：

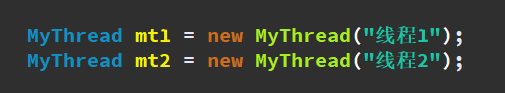
public final void setName(String name)

设置线程名称



把MyThread类定义一个带参构造方法可以直接设置名称

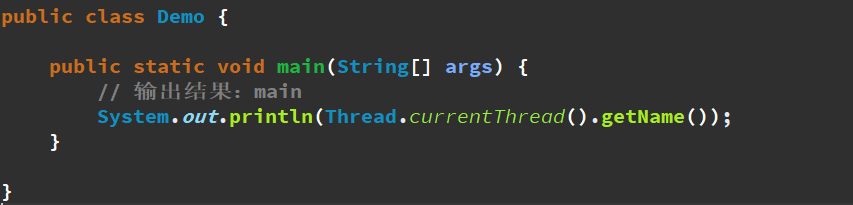




要获取main方法所在的线程的名称，怎么办呢？

public static Thread currentThread()

返回当前正在执行的线程的对象的引用



针对不是Thread类的子类中如何获取线程对象名称呢？

方法：Thread.*currentThread*().getName()

利用Thread类的currentThread()方法返回当前执行的线程的对象引用，然后getName()

线程调度

假如我们的计算机只有一个 CPU，那么 CPU 在某一个时刻只能执行一条指令，线程只有得到 CPU时间片，也就是使用权，才可以执行指令。那么Java是如何对线程进行调用的呢？

线程有两种调度模型：

分时调度模型：所有线程轮流使用 CPU 的使用权，平均分配每个线程占用 CPU 的时间片

抢占式调度模型：优先让优先级高的线程使用 CPU，如果线程的优先级相同，那么会随机选择一个，优先级高的线程获取的 CPU 时间片相对多一些。

Java使用的是抢占式调度模型，取决于线程本身的优先级

如何设置/获取线程的优先级

我们的线程没有设置优先级，那么他具有一个默认优先级5

public final int getPriority()

返回线程对象的优先级

public final void setPriority(int newPriority)

更改/设置线程的优先级（范围为1-10）

线程优先级范围

最高优先级是10；最低优先级是1；默认优先级是5

线程优先级高仅仅表示线程获取CPU获取时间片的几率，但是要在多次运行才能看到比较好的效果，一两次说明不了问题，因为存在随机性

线程控制

我们已经知道了线程的调度，接下来我们就可以使用如下方法对象线程进行控制

线程休眠

public static void sleep(long millis)

在指定的毫秒数内，让当前正在执行的线程休眠



线程加入

public final void join()

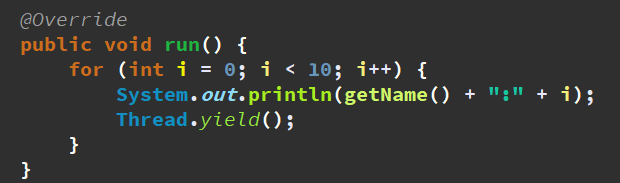
等待该线程中止（让该线程执行完毕，别人才能执行）



线程礼让

public static void yield()

暂停当前正在执行的线程对象，并执行其他线程；让多个线程的执行更和谐，但是不能靠他保证一个线程执行一次



守护线程

public final void setDaemon(boolean on)

将该线程标记为守护线程或用户线程，当正在运行的线程都是守护线程时，Java虚拟机退出；该方法必须在启动前调用



中断线程

public final void stop()

让线程终止，后面的代码不执行，具有不安全性，不建议使用已过时

t1.stop();

public void interrupt()

中断线程，把线程状态终止并抛出一个异常InterruptedException

后面的代码会继续执行

t1.interrupt()

线程的生命周期：（面试题）

新建：创建线程对象

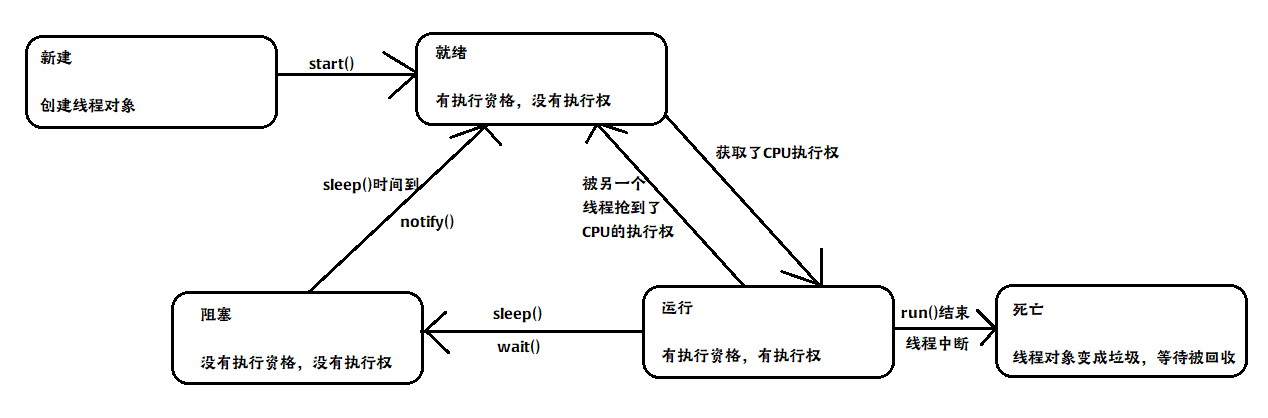
就绪：线程有执行资格，没有执行权

运行：线程有执行资格，有执行权

阻塞：由于一些操作让线程处于该状态，没有执行资格，没有执行权；而另一些操作可以把它给激活，激活后处于就绪状态

死亡：线程对象变成垃圾，等待被回收

图解：



创建线程方式2

实现Runnable接口

步骤：

A:自定义类MyRunnable实现Runnable接口

B:重写run()方法

C:创建MyRunnable类的对象

D:创建Thread类的对象，并把C步骤的对象作为构造参数传递



小总结：

实现多线程的方式：2种

方式1：继承Thread类

A:创建自定义类MyThread继承Thread类

B:在MyThread类中重写run()方法

C:创建MyThread类对象

D:启动线程对象

问题：

a:为什么要重写run()

run()里面封装的是被线程执行的代码

b:启动线程对象用的哪个方法

start()方法

c:run()和start()区别

run()直接调用仅仅是普通方法

start()是先启动一个线程，再由JVM调用run()方法

方式2：实现Runnable接口

A:自定义类MyRunnable实现Runnable接口

B:在MyRunnable里面重写run()方法

C:创建MyRunnable类的对象

D:创建Thread类的对象，并把C步骤的对象作为构造参数传递

实现接口方式（方式2）的好处

1：通过方式2接口可以避免Java单继承带来的局限性

2：适合多个相同程序的代码去处理同一个资源的情况，把线程同程序的代码，数据有效分离，较好的体现了面向对象的设计思想。

案例需求：

某电影院目前正在上映贺岁大片，共有100张票，而它有3个售票窗口售票，并且每次买出一张票延迟100毫秒，请设计一个程序模拟该电影院售票。

出现问题

1.相同的票出现多次

CPU的一次操作必须是原子性的

2.还出现了负数的票

随机性和延迟导致的

以上属于线程安全问题

解决线程安全问题的基本思想

首先想为什么出现问题？(也是我们判断是否有问题的标准)

A:是否是多线程环境

B:是否有共享数据

C:是否有多条语句操作共享数据

电影票案例出现了以上三张情况，但是AB情况无法改变，只能改变C情况

改变的思路：

把多条语句操作共享数据的代码给包成一个整体，让某个线程在执行的时候，别人不能来执行

Java提供了同步机制

同步代码块：

格式：

synchronized(对象){

需要同步的代码;

}

A:对象是什么？

任意对象

B:需要同步的代码是哪些？

多条语句操作共享数据的代码

注意：同步可以解决安全问题的根本原因就在那个对象上，该对象如同锁的功能，多个线程必须是同一把锁

举例：

火车上厕所，只能一个人进去

同步的特点：

多个线程

多个线程使用的是同一个锁对象

同步的好处：

解决了多线程的安全问题

同步的弊端：

当线程相当多时，因为每个线程都会去判断同步上的锁，这是很耗费资源的，无形中就会降低程序的运行效率

同步代码块的锁对象是谁呢？

任意对象

同步方法的格式及锁对象问题

把同步关键字加在方法上，同步方法锁是this

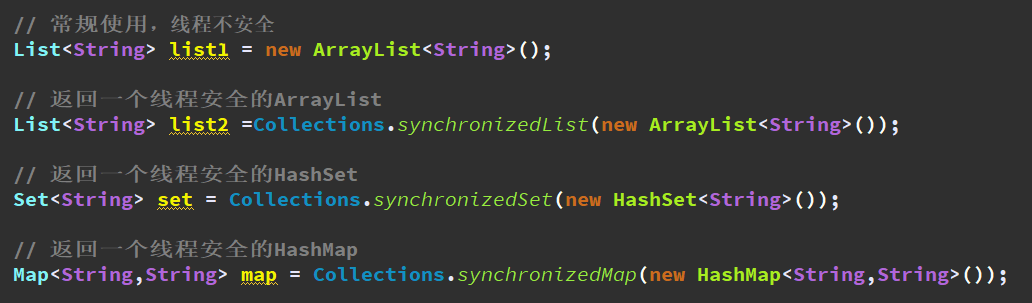
静态方法的锁对象问题

静态方法的锁对象是当前类的字节码(class)文件对象

Vector类集合是线程安全的，但是之前说过，即使需要线程安全，也不用Vector，那么到底用谁呢？

查看API可知Collections类下有许多的方法

举例：



day23总结：

1.多线程

进程：正在执行的应用程序

线程：进程的执行单元，执行路径

单线程：一个应用程序只有一条执行路径

多线程：一个应用程序有多条执行路径

多进程的意义：提高CPU的使用率

多线程的意义：提高应用程序的使用率

2.Java程序的运行原理及JVM的启动是多线程的吗？

A:Java命令去启动JVM，JVM会启动一个进程，该进程会启动一个主线程

B:JVM的启动是多线程的，因为他最低有两个线程启动了，主线程和垃圾回收线程

3.多线程的实现方案（2种方式）

A:继承Thread类

步骤：创建自定义类MyThread继承Thread类；重写run()方法；创建多个MyThread类对象；每个对象调用start()方法

B:实现Runnable接口

步骤：创建自定义类MyRunnable类实现Runnable接口；重写run()方法；创建MyRunnable对象；把创建的对象当做构造参数新建Thread类对象；启动Thread类对象

4.线程的调度和优先级问题

分时调度

抢占式调度（Java采用这种方式）

获取和设置线程优先级；范围是1-10；默认优先级是5

5.线程的控制

A:休眠线程

B:加入线程

C:礼让线程

D:后台(守护)线程

E:中止线程

6.线程的生命周期

A:新建：创建一个线程对象

B:就绪：有执行资格，没有执行权

C:运行：有执行资格，有执行权

D:阻塞：由于某些操作让线程对象处于阻塞状态，没有执行资格，没有执行权；但是通过某些操作可以把它激活，激活后处于就绪状态

E:死亡：线程对象变成垃圾，等待被回收

7.卖票程序的实现

8.线程安全问题

JDK5中Lock锁的使用

虽然我们可以理解同步代码块和同步方法的锁对象问题，但是我们并没有直接看到在哪里加上了锁，在哪里释放了锁，为了更清晰的表达如何加锁和释放锁，JDK5以后提供了一个新的锁对象Lock

接口：Lock

void lock();

获取锁

void unlock();

释放锁

Lock的实现类：ReentrantLock

使用方法：



先定义锁对象，然后加锁释放锁，为了防止中间代码出异常，所以把释放锁加在finally里，可以保证释放锁

死锁问题

同步弊端：

效率低

如果出现了同步嵌套，就容易产生死锁问题

死锁问题及其代码

是指两个或者两个以上的线程在执行的过程中，因争夺资源产生的一种互相等待现象

同步代码块的嵌套案例

线程间通信问题

不同种类的线程间针对同一个资源的操作

举例：店铺卖包子；包子能卖出，也会继续出炉；

通过设置线程（生产者）和获取线程（消费者）针对同一个学生对象进行操作

day24视频案例

等待唤醒机制

等待唤醒：

Object类中提供了三个方法：

wait()

线程等待，并且会立即释放锁，将来醒过来的时候是从wait()方法处醒来

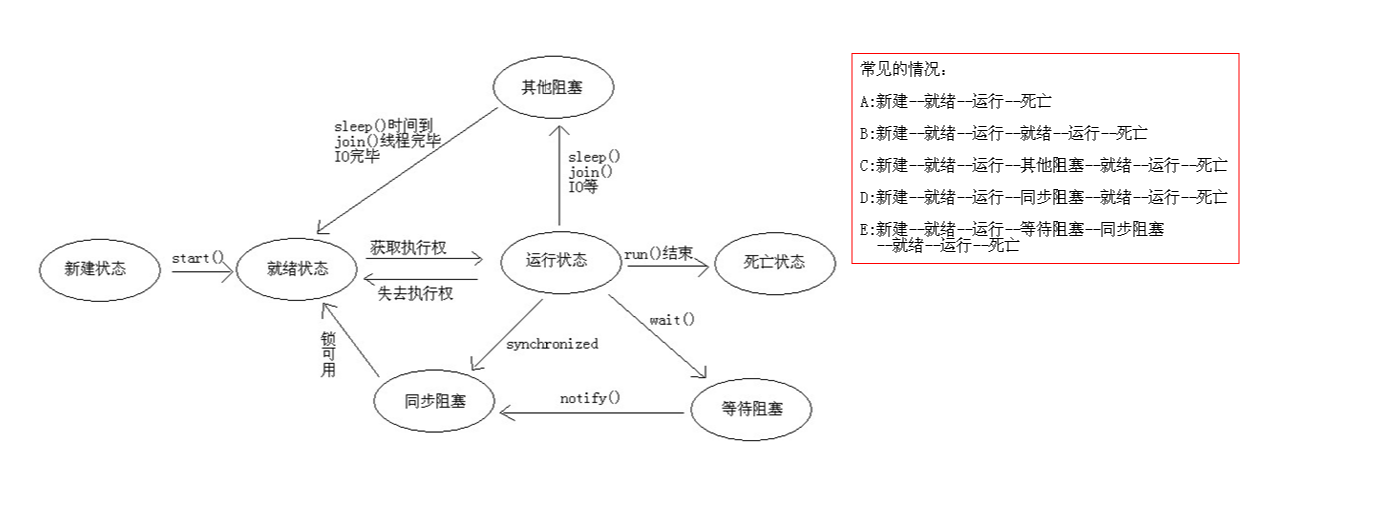
notify():唤醒单个线程

notifyAll():唤醒所有线程

为什么这些方法不定义在Thread里面呢？

这些方法的调用必须通过锁对象调用，而锁对象可以是任意对象，所以这些方法必须定义在Object方法中

线程状态转换图



线程组

Java中使用ThreadGroup来表示线程组，它可以对一批线程进行分类管理，Java允许程序直接对线程组进行控制

线程默认情况下属于main线程组

可以新建线程组，并把线程加入该线程组

线程组的好处是可以对一个组线程进行操作

线程池

程序启动一个新线程成本是比较高的，因为它涉及到要与操作系统进行交互。而使用线程池可以很好的提高性能，尤其是当程序中要创建大量生存期很短的线程时，更应该考虑使用线程池

线程池的好处:

线程池里的每一个线程代码结束后，并不会死亡，而是再次回到线程池中成为空闲状态，等待下一个对象来使用

在JDK5之前，我们必须手动实现自己的线程池，从JDK5开始，Java内置支持线程池

如何实现线程池的代码呢？

A:创建一个线程池对象，控制要创建几个线程对象

public static ExecutorService

newFixedThreadPool(int nThreads)

B:这种线程池的线程可以执行

可以执行Runnable对象或者Callable对象代表的线程

做一个类实现Runnable接口

C:调用如下方法即可

Future<?> submit(Runnable task)

<T> Future<T> submit(Callable<T> task)

JDK5新增了一个Executors工厂类来产生线程池，有如下几个方法

public static ExecutorService

newCachedThreadPool()

public static ExecutorService

newFixedThreadPool(int nThreads)

public static ExecutorService

newSingleThreadExecutor()