# 背景介绍

人类交流有自己的语言，同样，人类与计算机交流也需要有自己的语言。1995年，Sun公司开发了一门新的编程语言——JAVA。是由Java 之父 詹姆斯.高斯林 和一个小组的其他成员组成共同开发的。开发 Java 之初的目标是创建能嵌入消费类电子产品的软件，构建一种既可移植又可跨平台的语言。后来，人们慢慢意识到Internet应用具有同样的跨平台安全性和可移植性的问题，所以Java 很快取得了巨大的成功。

Java自1995诞生，至今已经20多年的历史。

Java的名字的来源：Java是印度尼西亚爪哇岛的英文名称，因盛产咖啡而闻名。Java语言中的许多库类名称，多与咖啡有关，如JavaBeans(咖啡豆)、NetBeans(网络豆)以及ObjectBeans (对象豆)等等。SUN和JAVA的标识也正是一杯正冒着热气的咖啡。



# 初识Java

## 学习目标

* 理解什么是程序
* 会使用记事本开发简单Java程序
* 会使用输出语句在控制台输出信息
* 熟悉Eclipse开发环境

## 什么是程序

1. 生活中的程序
2. 带上银行卡去银行取款
3. 取款机排队
4. 输入密码进行取款
5. 拿钱离开银行
6. 计算机中的程序

计算机程序：为了让计算机执行某些操作或解决某个问题而编写的一系列有序指令的集合。编写程序的工具就是计算机语言，Java就是多种语言中的一种。

System.out.println("从账户余额中扣除指定金额");

System.out.println("取款机吐出指定金额");

## 什么是Java

### 为什么学习Java

Java是Sun Microsystems于1995年推出的高级编程语言。

使用Java可以做什么？

* 开发桌面应用程序：银行软件、商场结算软件
* 开发面向Internet的应用程序：网上数码商城、阿里巴巴、易趣网

Java的几个优点：

* 一次编写，到处运行。
* 系统的多平台支持。
* 强大的可伸缩性。
* 多样化和功能强大的开发工具支持。

Java是一种面向对象编程语言，与面向过程编程不同。

面向过程:(){

吃饭()

喝水()

洗衣服()

看电影()

睡觉()

端茶倒水()

}

面向对象:

女朋友{

吃饭()

喝水()

洗衣服()

看电影()

睡觉()

端茶倒水()

}

Java技术平台介绍：

Java SE：标准版。

* Java技术的基础和核心
* 主要用于开发桌面应用程序

Java EE：企业版

* 提供了企业级应用开发的完整解决方案
* 很多的网站都是采用Java EE技术开发



## 搭建Java开发环境

1. 安装JDK

2. 配置环境变量

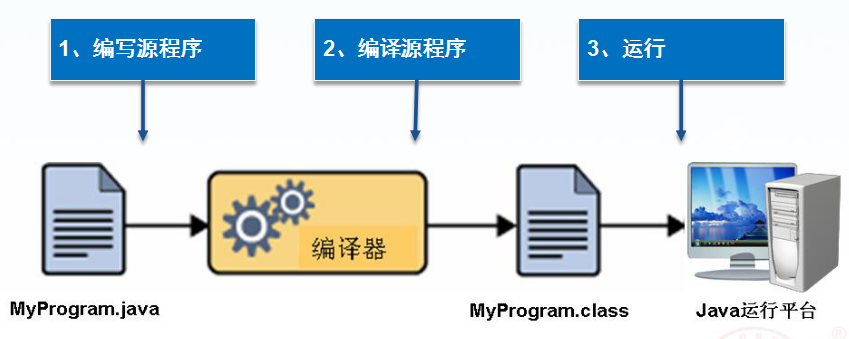
JAVA\_HOME ：C:\Program Files\Java\jdk1.7.0CLASSPATH：.;%JAVA\_HOME%\lib\dt.jar;%JAVA\_HOME%\lib\tools.jar;Path：%JAVA\_HOME%\bin;%JAVA\_HOME%\jre\bin;

JDK：java开发工具包。

JRE：java运行时环境。

## 第一个Java程序

### 开发Java程序的步骤



使用记事本开发Java程序的步骤如下：

1. 使用记事本编辑源程序，以.java为后缀名保存

public class HelloWorld {

public static void main(String[ ] args) {

System.out.println("Hello World!!!");

}

}

1. 使用javac命令编译.java文件，生成.class文件

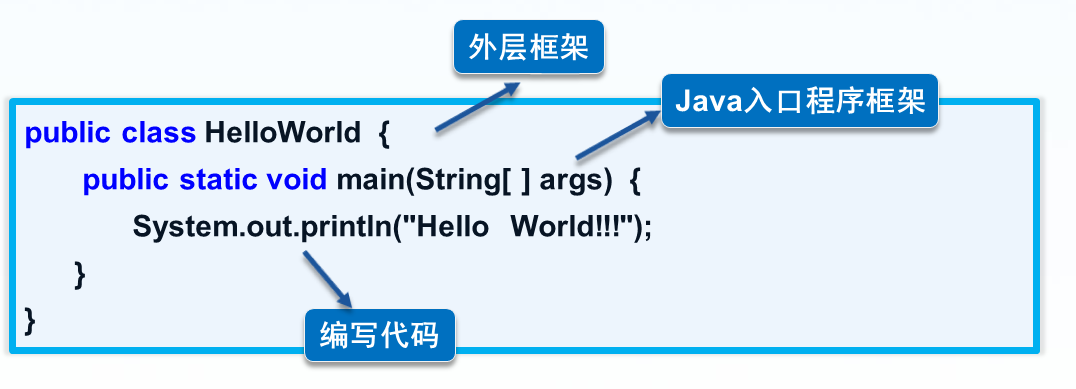
javac.tif

1. 使用java命令运行.class文件，输出程序结果

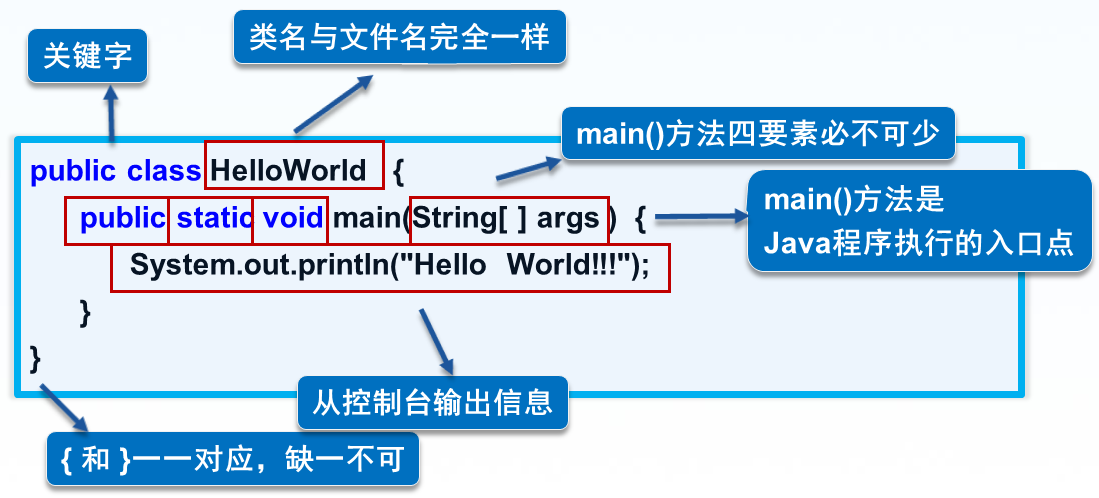
java.tif

### Java程序的结构

基本结构：



详细说明：



### 控制台输出

如何使下面2个语句达到同样的效果？

System.out.println()

System.out.print()

答：使用转义符。

|  |  |
| --- | --- |
| **转义符** | **说 明** |
| \n | 将光标移动到下一行的第一格 （回车键） |
| \t | 将光标移到下一个水平制表位置 （一个tab键的距离） |

示例如下：

public class HelloWorld{

public static void main(String[ ] args){

System.out.print("Hello World!!!\n");

}

}

练习：从控制台打印输出张三的姓名和年龄。

System.out.println("张三");

System.out.println("18");

System.out.print("张三\n");

System.out.print("18");

### 程序注释

单行注释以 // 开始：

public class HelloWorld{

public static void main(String[ ] args){

//输出消息到控制台

System.out.println("Hello World!!! ");

}

}

多行注释以“/\*”开头，以“\*/”结尾，每一行开头可写一个或多个\*：

/\*

\* HelloWorld.java

\* 2019-3-6

\* 第一个Java程序

\*/

public class HelloWorld {

public static void main(String[ ] args) {

/\*

System.out.println("Hello World!!!");

System.out.println("Hello World!!!");

\*/

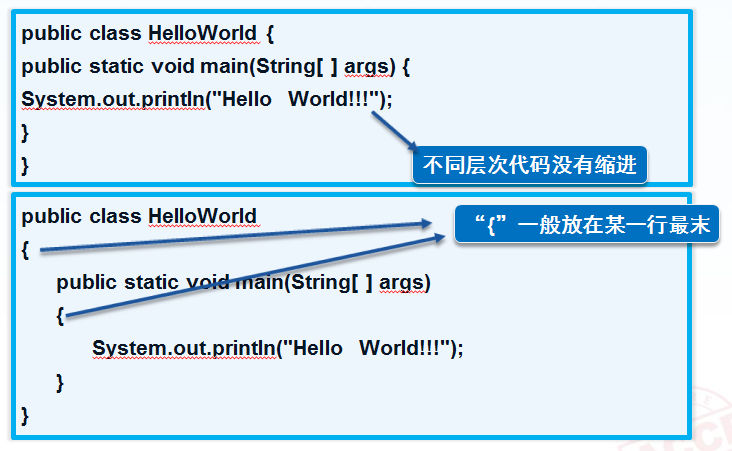
}

}

### 编码规范

Java编码规范需要注意以下几点：

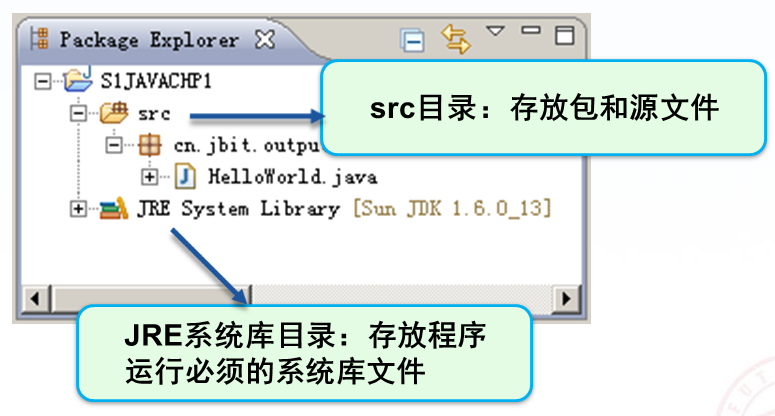
* 类名使用public修饰
* 一行只写一条语句
* { }的使用及位置
* 代码缩进



### 项目组织结构

包资源管理器：

* 用包组织Java源文件，类似于文件夹
* 选择菜单“Window→Show View→Package Explorer”打开



## 练习题

1. 指出下面代码中错误的地方：

public class helloWorld { //源文件名为HelloWorld.java

public static void main (String[ ] args){

System.out.println("我的第一个MyEclipse小程序！");

}

}

答案：public修饰的类的名称必须与Java文件同名!

2. 指出下面代码中错误的地方：

public class HelloWorld {

public static main (String[ ] args) {

System.out.println("我的第一个MyEclipse小程序！");

}

}

答案：main()方法作为程序入口，void必不可少！

3. 指出下面代码中错误的地方：

public class HelloWorld {

public static void main (String[ ] args) {

system.out.println("我的第一个MyEclipse小程序！");

}

}

答案：语法出错，无法解析system！Java对大小写敏感!

4. 指出下面代码中错误的地方：

public class HelloWorld {

public static void main (String[ ] args) {

System.out.println("我的第一个MyEclipse小程序！")

}

}

答案：每一条Java语句必须以分号结束!

## 编码实操

1. 使用Eclipse创建Java应用程序，实现从控制台输出多行信息：姓名、年龄、爱好。
2. 练习Eclipse相关操作：

* 显示行号
* 删除项目
* 导入项目
* 重命名项目
* 关闭、打开包资源管理器

1. 控制台输出购物清单，如下图：

购物清单.TIF

答案：

public class GoodsList {

public static void main(String[] args) {

System.out.println("\t购物清单");

System.out.print("商品名称\t");

System.out.print("购买数量\t");

System.out.print("商品单价\t");

System.out.print("金额\n");

System.out.print("鳄鱼夹克\t");

System.out.print(" 1\t");

System.out.print("1900\t");

System.out.print("1900\n");

System.out.print("ipodMP4\t");

System.out.print(" 2\t");

System.out.print("400\t");

System.out.print("800\n");

}

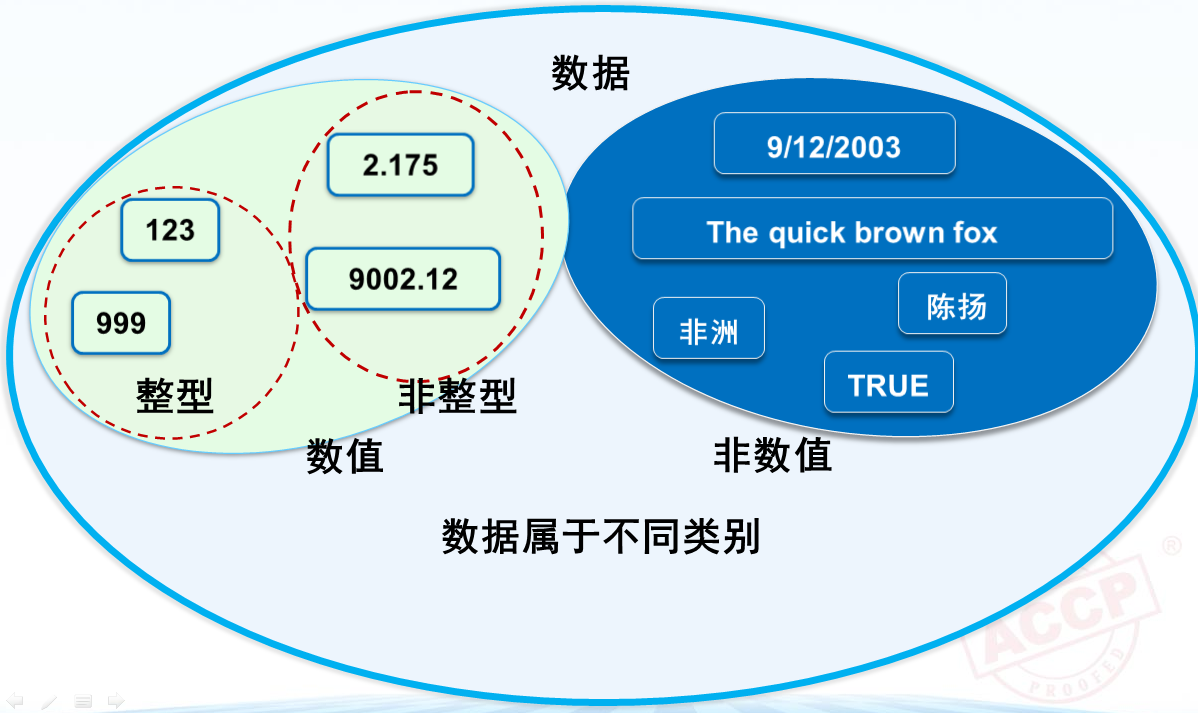
}

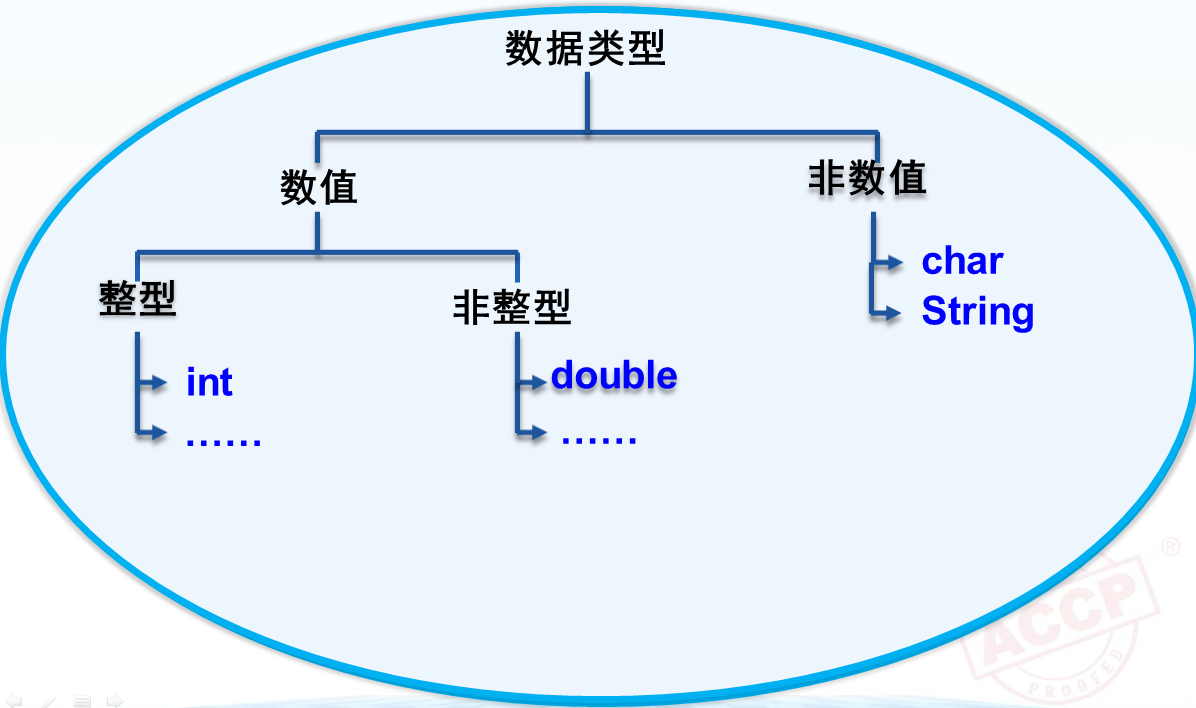
# Java基础知识

## 学习目标

* 掌握Java中的变量
* 掌握Java的基本数据类型
* 掌握Java的运算符和表达式
* 掌握Java的流程控制结构
* 掌握break、continue语句的用法和区别
* 掌握数组的使用

## 数据类型





数据类型说明：

|  |  |
| --- | --- |
| **数据类型** | **说 明** |
| char  （字符型） | 用于存储单个字符  如：性别‘男’、‘女’，电灯‘开’、‘关’ |
| int  （整型） | 用于存储整数  如：一天的时间是24小时，一月份有31天 |
| double  （双精度） | 用于存储小数  如：蒙牛早餐奶的价格1.3元，手机待机时间6.5小时 |
| String  （字符串） | 用于存储一串字符  如：“我的爱好是踢足球”，“我喜欢Java程序” |

## 变量

### 为什么要有变量

电脑使用内存来记忆计算时所使用的数据，内存如何存储数据？

内存好比旅馆，使用内存就好比在旅馆订一个房间：

1、开房间（单人间、双人间、总统套间）

2、入住



数据类型就好比旅馆中房间的类型，

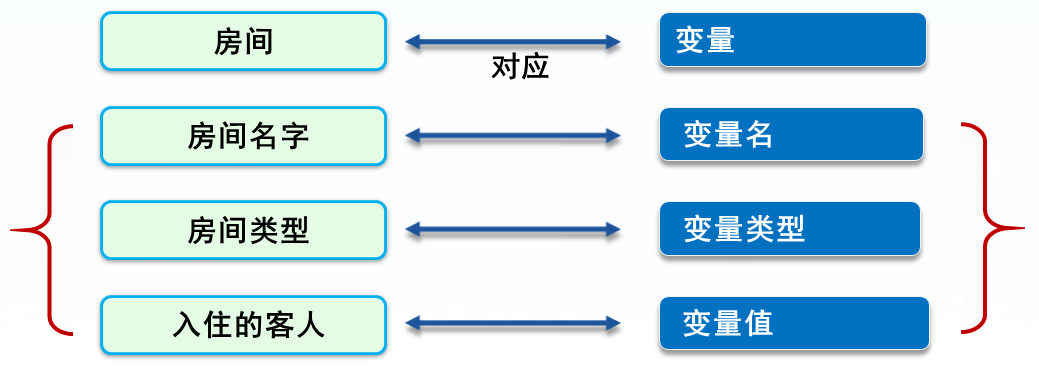
数据各式各样，要先根据数据的需求（即数据类型）为它申请一块合适的空间

在银行存1000元钱，银行一年的利息5%，那一年之后钱变成了多少？



已经将数据存入内存，但是： 怎么找到存入的数据？

可以通过内存中房间的别名找到数据存储的位置，即通过变量名可以简单快速地找到它存储的数据。



### 使用变量的步骤

第一步：声明变量，即“根据数据类型在内存申请空间”



第二步：赋值，即“将数据存储至对应的内存空间”



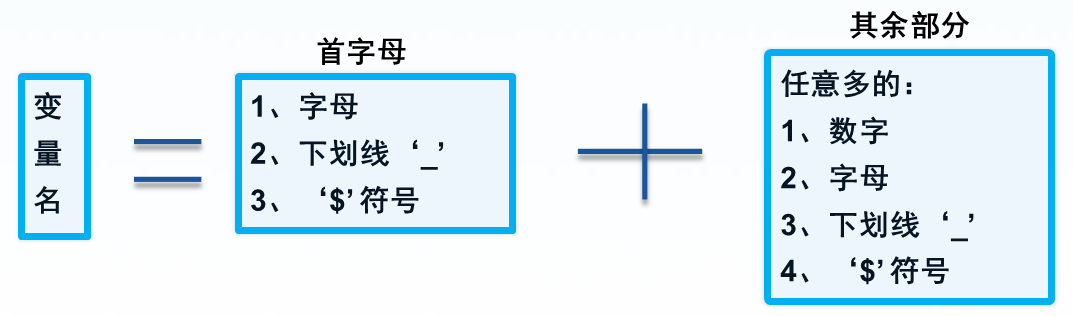
第一步和第二步可以合并：



第三步：使用变量，即“取出数据使用 ”

### 变量命名规则

Java语言中，变量命名要符合一定规则。简短且能清楚地表明变量的作用，通常第一个单词的首字母小写，其后单词的首字母大写。例如：myScore。



实际开发时，为了易于维护，尽量使用有意义的变量名。

检查变量名：



不同类型的变量存取：

public static void main(String[ ] args) {

double score = 98.5;

String name = "张三";

char sex = '男';

System.out.println("本次考试成绩最高分： " + score);

System.out.println("最高分得主： " + name);

System.out.println("性别： " + sex);

}

## 运算符

语法：

变量名=表达式;

例如：

a=( b+3)+(b–1)

表达式就是符号（如加号、减号）与操作数（如b，3等）的组合。

### 算术运算符

基本的算术运算：

+ - \* /

例如，从控制台输入学员王浩3门课程成绩，编写程序实现

1. Java课和SQL课的分数之差
2. 3门课的平均分

计算成绩.TIF

实现代码如下：

import java.util.Scanner;

public class ScoreStat {

/\*

\* 成绩统计

\*/

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

System.out.print("STB的成绩是：");

int stb = input.nextInt(); //STB分数

System.out.print("Java的成绩是：");

int java = input.nextInt(); //Java分数

System.out.print("SQL的成绩是：");

int sql = input.nextInt(); //SQL分数

int diffen; //分数差

double avg; //平均分

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

System.out.println("STB\tJava\tSQL");

System.out.println(stb + "\t" + java + "\t" + sql);

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

diffen = java - sql; //计算Java课和SQL课的成绩差

System.out.println("Java和SQL的成绩差：" + diffen);

avg = (stb + java + sql) / 3; //计算平均分

System.out.println("3门课的平均分是: " + avg);

}

}

### 关系运算符

为什么使用关系运算符？

比较高低、大小、长短等：

* 张三的考试成绩是否比李四高
* 大象是否比乌龟更长寿
* 篮球跟地球一样大吗

常用的关系运算符有哪些？

>，<

==，!=

>=，<=

关系运算符的比较结果使用boolean类型。boolean类型一共有两个值：

* 真：true
* 假：false

例如，控制台输入张三的成绩，与李四的成绩进行比较，输出比较结果：

import java.util.Scanner;

public class BoolTest {

public static void main(String[] args) {

int liSi = 80; //学员李四成绩

boolean isBig ; //声明一个boolean类型的变量

Scanner input = new Scanner(System.in); //Java输入的一种方法

System.out.print("输入学员张三成绩: "); //提示要输入学员张三的成绩

int zhangSan = input.nextInt(); //输入张三的成绩

isBig = zhangSan > liSi ; //将比较结果保存在boolean变量中

System.out.println( "张三成绩比李四高吗 ？ "+isBig );//输出比较结果

}

}

### 案例：打印购物小票并计算积分

需求说明：

* 结算时打印购物小票
* 计算此次购物获得的会员积分

积分计算规则：每消费100元计3积分。

打印小票.TIF

实现代码如下：

public class Pay {

/\*

\* 购物结算、打印小票并计算积分

\*/

public static void main(String[] args) {

int shirtPrice = 245; //Ｔ恤价格

int shoePrice = 570; //网球鞋价格

int padPrice = 320; //网球拍价格

int shirtNo=2; //Ｔ恤件数

int shoeNo =1; //运动鞋数目

int padNo = 1; //网球拍数目

double discount = 0.8;

/\*计算消费总金额\*/

double finalPay = (shirtPrice \* shirtNo + shoePrice \* shoeNo + padPrice \* padNo) \* discount;

//System.out.println("消费总金额：" + finalPay);

/\*计算找钱\*/

double returnMoney = 1500-finalPay;

/\*打印购物小票\*/

System.out.println("＊＊＊＊＊＊＊\*消费单\*＊＊＊＊＊＊＊");

System.out.println("购买物品\t" + "单价\t" + "个数\t"+ "金额\t");

System.out.println("Ｔ恤\t" + "￥"+shirtPrice+ "\t" + shirtNo+ "\t" + "￥"+(shirtPrice \* shirtNo)+"\t");

System.out.println("网球鞋\t" + "￥"+shoePrice + "\t"+ shoeNo+ "\t" + "￥"+(shoePrice \* shoeNo)+ "\t");

System.out.println("网球拍\t" + "￥"+padPrice + "\t"+ padNo+ "\t" + "￥"+(padPrice \* padNo)+ "\t\n");

System.out.println("折扣：\t8折");

System.out.println("消费总金额\t" + "￥" + finalPay);

System.out.println("实际交费\t￥1500");

System.out.println("找钱\t" + "￥" + returnMoney);

/\*计算本次购物所获积分\*/

int score = (int)finalPay / 100 \* 3; //强制类型转换

System.out.println("本次购物所获的积分是： " + score);

}

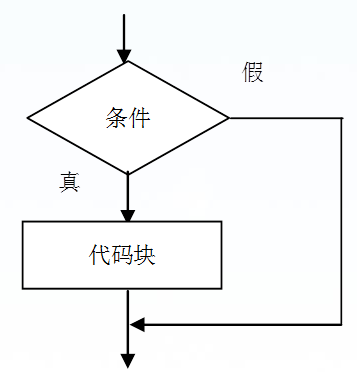
}

## 选择控制语句

### if else

1. if结构

if选择结构是根据条件判断之后再做处理



基本的if选择结构：

if ( 条件 ) {

//代码块

}

例如，如果张浩的Java考试成绩大于98分，张浩就能获得一个MP4作为奖励。

示例1.TIF

实现代码如下：

import java.util.Scanner;

public class GetPrize {

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

System.out.print("输入张浩的Java成绩: "); //提示输入Java成绩

int score = input.nextInt(); //从控制台获取张浩的Java成绩

if ( score > 90 ) { //判断是否大于90分

System.out.println("老师说:不错，奖励一个MP4！");

}

}

}

1. if...else if...else 结构

语法：

if(布尔表达式 1){

//如果布尔表达式 1的值为true执行代码

}else if(布尔表达式 2){

//如果布尔表达式 2的值为true执行代码

}else if(布尔表达式 3){

//如果布尔表达式 3的值为true执行代码

}else {

//如果以上布尔表达式都不为true执行代码

}

实例代码：

public class Test {

public static void main(String args[]){

int x = 30;

if( x == 10 ){

System.out.print("Value of X is 10");

}else if( x == 20 ){

System.out.print("Value of X is 20");

}else if( x == 30 ){

System.out.print("Value of X is 30");

}else{

System.out.print("这是 else 语句");

}

}

}

1. 嵌套if

使用嵌套的 if…else 语句是合法的。也就是说你可以在另一个 if 或者 else if 语句中使用 if 或者 else if 语句。

语法：

if(布尔表达式 1){

////如果布尔表达式 1的值为true执行代码

if(布尔表达式 2){

////如果布尔表达式 2的值为true执行代码

}

}

实例：

public class Test {

public static void main(String args[]){

int x = 30;

int y = 10;

if( x == 30 ){

if( y == 10 ){

System.out.print("X = 30 and Y = 10");

}

}

}

}

### switch case

switch case 语句判断一个变量与一系列值中某个值是否相等，每个值称为一个分支。

switch case 语句语法格式如下：

switch(expression){

case value :

//语句

break; //可选

case value :

//语句

break; //可选

//你可以有任意数量的case语句

default : //可选

//语句

}

表达式expression为整型或字符型。

switch case 执行时，一定会先进行匹配，匹配成功返回当前 case 的值，再根据是否有 break，判断是否继续输出，或是跳出判断。

示例如下：

public class Test {

public static void main(String args[]){

//char grade = args[0].charAt(0);

char grade = 'C';

switch(grade)

{

case 'A' :

System.out.println("优秀");

break;

case 'B' :

case 'C' :

System.out.println("良好");

break;

case 'D' :

System.out.println("及格");

break;

case 'F' :

System.out.println("你需要再努力努力");

break;

default :

System.out.println("未知等级");

}

System.out.println("你的等级是 " + grade);

}

}

如果 case 语句块中没有 break 语句时，JVM 并不会顺序输出每一个 case 对应的返回值，而是继续匹配，匹配不成功则返回默认 case。

public class Test {

public static void main(String args[]){

int i = 5;

switch(i){

case 0:

System.out.println("0");

case 1:

System.out.println("1");

case 2:

System.out.println("2");

default:

System.out.println("default");

}

}

}

以上代码编译运行结果如下：

default

如果 case 语句块中没有 break 语句时，匹配成功后，从当前 case 开始，后续所有 case 的值都会输出。

public class Test {

public static void main(String args[]){

int i = 1;

switch(i){

case 0:

System.out.println("0");

case 1:

System.out.println("1");

case 2:

System.out.println("2");

default:

System.out.println("default");

}

}

}

以上代码编译运行结果如下：

1

2

default

## 循环控制语句

### while/do while

1. while

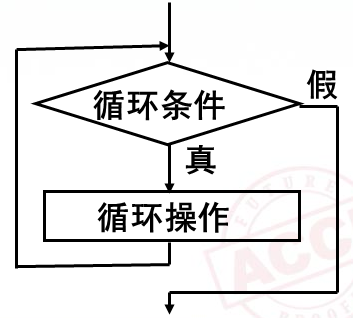
while是最基本的循环，它的结构为：

while( 布尔表达式 ) {

//循环内容

}

只要布尔表达式为 true，循环就会一直执行下去。



**特点：先判断，再执行。**

示例如下：

public class Test {

public static void main(String args[]) {

int x = 10;

while( x < 20 ) {

System.out.print("value of x : " + x );

x++;

System.out.print("\n");

}

}

}

以上实例编译运行结果如下：

value of x : 10

value of x : 11

value of x : 12

value of x : 13

value of x : 14

value of x : 15

value of x : 16

value of x : 17

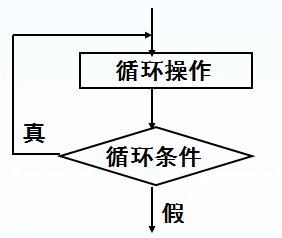
value of x : 18

value of x : 19

1. do while

为什么需要do-while循环？

经过几天的学习，老师给张浩一道测试题，让他先上机编写程序完成，然后老师检查是否合格。如果不合格，则继续编写。……



**特点：先执行，再判断**

do…while 循环和 while 循环相似，不同的是，do…while 循环至少会执行一次。

do…while 循环的结构如下：

do {

//代码语句

}while(布尔表达式);

注意：布尔表达式在循环体的后面，所以语句块在检测布尔表达式之前已经执行了。 如果布尔表达式的值为 true，则语句块一直执行，直到布尔表达式的值为 false。

示例如下：

public class Test {

public static void main(String args[]){

int x = 10;

do{

System.out.print("value of x : " + x );

x++;

System.out.print("\n");

}while( x < 20 );

}

}

以上实例编译运行结果如下：

value of x : 10

value of x : 11

value of x : 12

value of x : 13

value of x : 14

value of x : 15

value of x : 16

value of x : 17

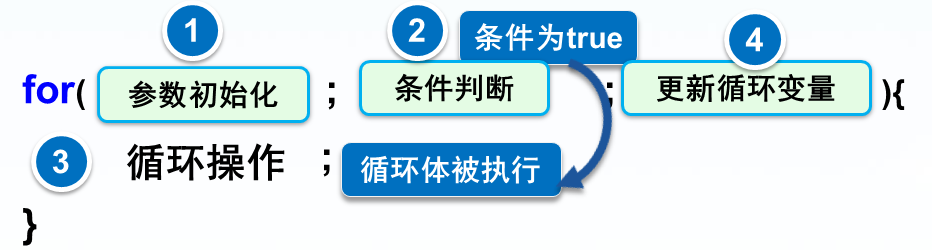
value of x : 18

value of x : 19

### for

Java 提供了另一种语句 —— for 循环，使一些循环结构变得更加简单。for循环执行的次数是在执行前就确定的。语法格式如下：

for循环的语法和执行顺序：



**特点：先判断，再执行。**

例如，输出100次“好好学习！”。

使用while循环结构：

int i=0;

while(i<100){

System.out.println("好好学习！");

i++;

}

使用for循环结构：

for(int i=0;i<100;i++){

System.out.println("好好学习！");

}

### break和continue

1. break

为什么要使用break？

假如在一次长跑比赛中，当运动员跑到第8圈的时候坚持不下去了，想退出比赛，可以使用break。



for (int i = 0; i<10; i++) {

//跑4000米；

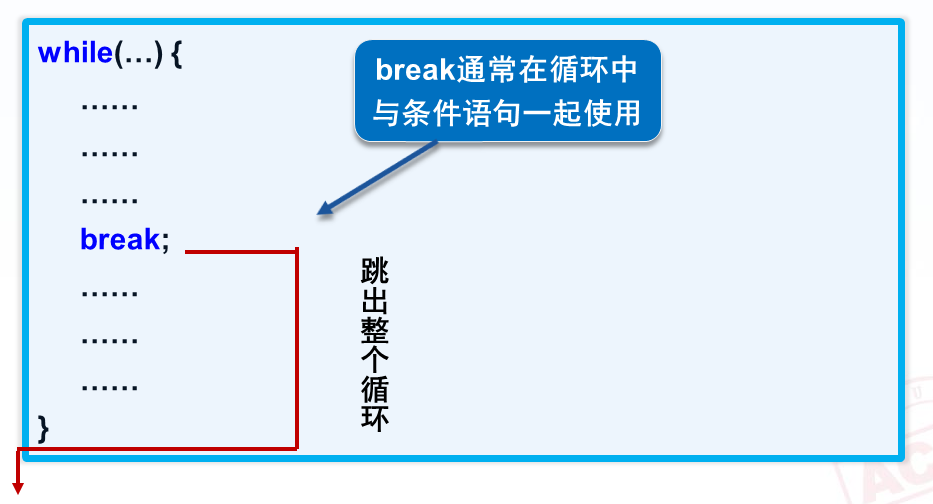
if ( 不能坚持 ) {

break; //退出比赛

}

}

break用于do-while、while、for中时，可跳出循环而执行循环后面的语句。



例如以下代码：

public class Test {

public static void main(String args[]) {

int [] numbers = {10, 20, 30, 40, 50};

for(int x : numbers ) {

// x 等于 30 时跳出循环

if( x == 30 ) {

break;

}

System.out.print( x );

System.out.print("\n");

}

}

}

以上实例编译运行结果如下：

10

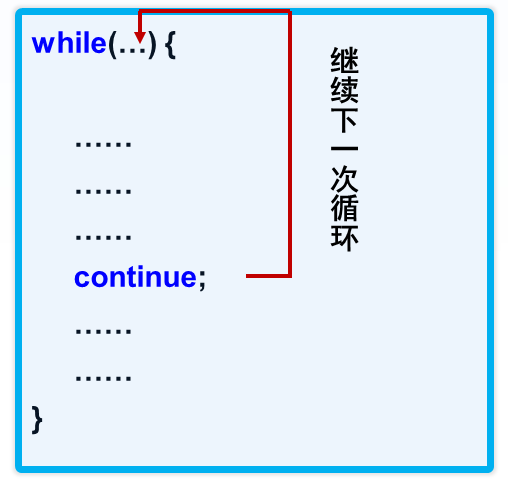
20

1. continue

为什么要使用continue？

假如需要循环录入Java课的学生成绩，统计分数大于等于80分的学生人数，如果成绩<80则不执行人数累加，直接执行下一次循环。此时可以使用continue。

continue 适用于任何循环控制结构中。作用是跳过循环体中剩余的语句而执行下一次循环。



例如以下代码，统计80分以上学生的比例：

import java.util.Scanner;

public class ContinueDemo {

/\*\*

\* 统计80分以上学生比例

\*/

public static void main(String[] args) {

int score; // 成绩

int total; // 班级总人数

int num = 0; // 成绩大于或等于80分的人数

Scanner input = new Scanner(System.in);

System.out.print("输入班级总人数: ");

total = input.nextInt(); // 输入班级总数

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

System.out.print("请输入第" + (i + 1) + "位学生的成绩： ");

score = input.nextInt();

if (score < 80) {

**continue**;

}

num++;

}

System.out.println("80分以上的学生人数是： " + num);

double rate = (double) num / total \* 100;

System.out.println("80分以上的学生所占的比例为：" + rate + "%");

}

}

## 数组

### 为什么需要数组

假如统计全班30人的平均分，若没有数组，需要新建30个变量，每个变量存储一个学生的成绩。

int stu1 = 95;

int stu2 = 89;

int stu3 = 79;

int stu4 = 64;

int stu5 = 76;

int stu6 = 88;

……

avg = (stu1+stu2+stu3+stu4+stu5…+stu30)/30;

如果使用数组，可以将30个学生的成绩放入一个数组中。

数组就是一个变量，存储相同数据类型的一组数据。



声明一个变量就是在内存空间划出一块合适的空间。

声明一个数组就是在内存空间划出一串**连续的**空间。

### 数组的基本要素

数组基本要素：

* 标识符：数组的名称，用于区分不同的数组
* 数组元素：向数组中存放的数据
* 元素下标：对数组元素进行编号，从0开始，数组中的每个元素都可以通过下标来访问
* 元素类型：数组元素的数据类型



### 数组的使用步骤

1. 声明数组：告诉计算机数据类型是什么

int[ ] score1; //Java成绩

int score2[ ]; //C#成绩

String[ ] name; //学生姓名

语法：

数据类型 数组名[ ] ;

数据类型[ ] 数组名 ;

1. 分配空间：告诉计算机分配几个连续的空间

score = new int[30];

avgAge = new int[6];

name = new String[30];

语法：

数据类型[ ] 数组名=new 数据类型[大小];

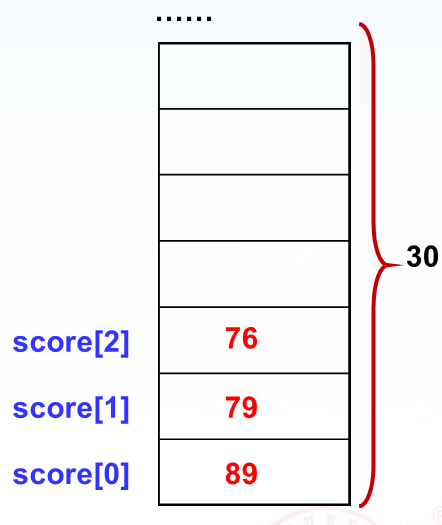
1. 赋值：向分配的空间里放数据

score[0] = 89;

score[1] = 79;

score[2] = 76;

……



此外也可以边声明边赋值：

int[ ] score = {89, 79, 76};

或

int[ ] score = new int[]{89, 79, 76};

1. 处理数组

取数组中的元素，求平均分：

int [ ] score = {60, 80, 90, 70, 85};

double avg;

avg = (score[0] + score[1] + score[2] + score[3] + score[4])/5;

结合循环：

数组的元素类型和数组的大小都是确定的，所以当处理数组元素时候，我们通常使用基本循环或者 For-Each 循环。

public class TestArray {

public static void main(String[] args) {

double[] myList = {1.9, 2.9, 3.4, 3.5};

// 打印所有数组元素

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

System.out.println(myList[i] + " ");

}

// 计算所有元素的总和

double total = 0;

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

total += myList[i];

}

System.out.println("Total is " + total);

// 查找最大元素

double max = myList[0];

for (int i = 1; i < myList.length; i++) {

if (myList[i] > max) max = myList[i];

}

System.out.println("Max is " + max);

}

}

以上实例编译运行结果如下：

1.9

2.9

3.4

3.5

Total is 11.7

Max is 3.5

## 编码实操

### 求1~100之间不能被3整除的数之和

提示：

1、循环条件：i<100

2、循环操作：

if (i % 3 != 0) {

sum = sum + i;

}

完整代码如下：

public class CalcSum {

public static void main(String[] args) {

int sum = 0;

for (int i = 0; i < 100; i++) {

if (i % 3 != 0) {

sum = sum + i;

}

}

System.out.println("和是：" + sum);

}

}

### 计算5位学员的平均成绩

import java.util.Scanner;

public class ArrayDemo {

/\*\*

\* 使用数组计算平均分

\*/

public static void main(String[] args) {

int[] scores = new int[5]; //成绩数组

int sum = 0; //成绩总和

Scanner input = new Scanner(System.in);

System.out.println("请输入5位学员的成绩：");

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

scores[i] = input.nextInt();

sum = sum + scores[i]; //成绩累加

}

//计算并输出平均分

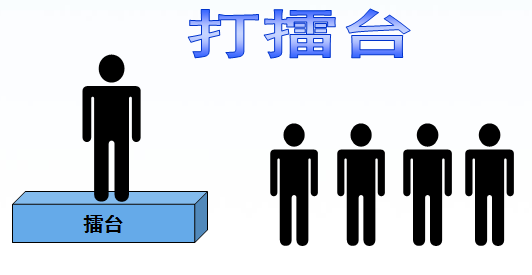
System.out.println("学员的平均分是：" + (double)sum/scores.length);

}

}

### 求成绩最高分

从键盘输入本次Java考试五位学生的成绩，求考试成绩最高分。



代码如下：

import java.util.Scanner;

public class MaxScore {

/\*\*

\* 求数组最大值

\*/

public static void main(String[] args) {

int[] scores = new int[5];

int max = 0; //记录最大值

System.out.println("请输入5位学员的成绩：");

Scanner input = new Scanner(System.in);

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

scores[i] = input.nextInt();

}

//计算最大值

max = scores[0];

for(int i = 1; i < scores.length; i++){

if(scores[i] > max){

max = scores[i];

}

}

System.out.println("考试成绩最高分为：" + max);

}

}

## 练习题

1. 在Java类中，使用以下（ ）声明语句来定义公有的int型常量MAX。

A. public int MAX = 100;

B. final int MAX = 100;

C. public static int MAX = 100;

D. public static final int MAX = 100;

答案：D

2. 你希望找到一个更优雅的方式给你的数组赋值而不使用for 循环语句，下面的哪一个能做到？

A. myArray{

[1]="One";

[2]="Two";

[3]="Three";

}

B. String s[5]=new String[]{"Zero","One","Two","Three","Four"};

C. String s[]=new String[]{"Zero","One","Two","Three","Four"};

D. String s[]=new String[]={"Zero","One","Two","Three","Four"};

答案：C

3. 你想用下面的代码查找数组最后一个元素的值，当你编译并运行它的时候，会发生什么？

public class MyAr{

public static void main(String argv[]){

int[]i = new int[5];

System.out.println(i[5]);

}

}

A. 编译完成并输出 0

B. 编译完成并输出null

C. 编译完成但运行时异常

D. 编译错误

答案：C

# 类与对象

## 学习目标

* 掌握类与对象的意义和概念
* 掌握属性与方法的意义和概念
* 掌握构造方法的意义和使用
* 掌握继承的意义和使用

## 为什么要有类和对象

举个例子：

生产每辆汽车都要按照图纸，经过生产组装，最终加工而成。而同样的车型，原始的生产图纸是同一个。那么这张生产图纸就相当于类。而生产出的具体的每辆车就相当于对象。类有一个，根据类实例化的对象可以有多个。类和对象就是这种关系。

再比如：

某某大学表白墙找对象的公告上写到这些条件“170cm以上的女生、学习好、会做饭、打篮球...”。

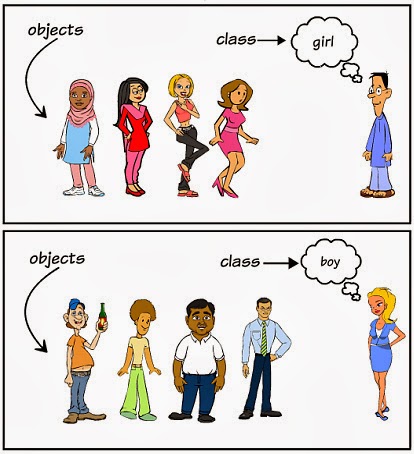
那么正巧一个名叫石榴姐的女孩说具备这个条件，而石榴姐是客观存在的事物，她就是对象（Object）。而这些条件就是一个类（Class），类是确定对象拥有的**特征（属性）和行为**（方法），它不是具体客观存在的东西。

## 基本概念

部门（软件研发部，设计部，销售部），对象

对象：对象表示现实世界中一个具体的事物。对象是类的一个实例，有状态和行为。例如，一条狗是一个对象，它的状态有：颜色、名字、品种；行为有：摇尾巴、叫、吃等。

类：类是一个模板，它描述一类对象的行为和状态。



1. 对象的特征——属性



1. 对象的特征——方法



1. 类与对象的关系

* 类是抽象的概念，仅仅是一个模板，比如说：“人”。
* 对象是一个你能够看得到、摸得着的具体实体（不准确）。
* 类定义了对象将会拥有的特征（属性）和行为（方法）
* 类是对象的类型，对象是类的实例

1. 类的使用步骤



创建一个Person类：

public class Person {

String name;

int age;

void eat() {

}

void sleep() {

}

}

创建Person对象：

public class Person {

String name;

int age;

void eat() {

}

void sleep() {

}

public static void main(String[] args) {

//创建Person对象

Person per = new Person();

//调用方法

per.eat();

}

}

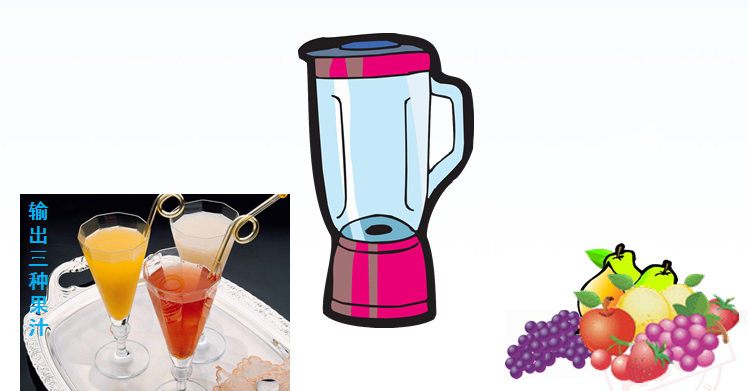
一个类可以包含以下几种变量：

* **局部变量**：在方法、构造方法或者语句块中定义的变量被称为局部变量。变量声明和初始化都是在方法中，方法结束后，变量就会自动销毁。
* **成员变量**：成员变量是定义在类中，方法体之外的变量。这种变量在创建对象的时候实例化。成员变量可以被类中方法、构造方法和特定类的语句块访问。
* **类变量**：类变量也声明在类中，方法体之外，但必须声明为static类型。

## 带参方法

### 为什么要用带参方法

例如，将榨汁机工作过程模拟到程序中：榨汁操作对应榨汁的方法；提供的水果原料对应方法参数；得到的果汁对应方法的返回值。传入不同的参数将得到不同的返回值。



### 带参方法的使用

1. 一个参数

定义带参方法：

public class ZhazhiJi {

public String zhazhi ( **String fruit** ) {

String juice = fruit + "汁";

return juice;

}

}

调用带参方法：

/\*调用zhazhi方法\*/

ZhazhiJi myZhazhiji = new ZhazhiJi();

String myFruit = "苹果";

String myJuice = **myZhazhi.zhazhi(myFruit)**;

System.out.println(myJuice);

1. 多个参数

定义带参方法：

public class ZhazhiJi {

public String zhazhi ( **String fruit,String fruit2** ) {

String juice = fruit + "汁和"+fruit2+"汁";

return juice;

}

}

调用带参方法：

/\*调用zhazhi方法\*/

ZhazhiJi myZhazhiji = new ZhazhiJi();

String myFruit = "苹果";

String myFruit2 = "桃子";

String myJuice = **myZhazhi.zhazhi(myFruit,myFruit2)**;

System.out.println(myJuice);

## 构造方法

构造方法作用就是对类进行**初始化**。每个类都有构造方法。如果没有显式地为类定义构造方法，Java编译器将会为该类提供一个默认不带任何参数的构造方法。

**在创建一个对象的同时，至少要调用一个构造方法**。构造方法的名称必须与类同名，一个类可以有多个构造方法。

例子如下：

class Person //人类{

public Person(String n,int a) //构造方法

{

name = n; age = a;

}

private string name;

private int age;

}

public static void main(String[] args){

Person p = new Person("张三",14);//这就是作用

}

new一个对象的时候要用到构造函数，例如Hello hello = new Hello();这时调用的是Hello的**无参数构造方法。**

Hello hello = new Hello("hi");这个是调用Hello的**有参数构造方法**。

在JAVA中如果不写构造方法的话，会默认加上一个无参数的构造方法，但是如果已经有了一个有参数的构造方法，那么无参数的构造方法就不会默认被加上（**失效**）。

如果Hello类中已经有了一个有参数的构造方法，这时再使用 Hello hello = new Hello();来创建对象的时候就会出错，这就是为什么要强调写了有参数的构造方法就最好加一个无参数的构造方法。

## 方法的重载

在Java程序中，如果同一个类中存在两个方法同名，方法的签名(参数个数、参数类型、类型排列次序)上也一样，将无法编译通过。 但在Java中多个方法重名是允许的，只要保证方法签名不同即可，这种特性称为方法重载（overload）。

方法重载需遵循如下两条规则：

* 方法名相同；
* 参数列表（个数、类型、顺序）不同；

注意：返回值不影响方法重载。

构造方法也可以重载 。

以下实例演示了如何重载 MyClass 类的 info 方法：

class MyClass {

int height;

MyClass() {

System.out.println("无参数构造函数");

height = 4;

}

MyClass(int i) {

System.out.println("房子高度为 " + i + " 米");

height = i;

}

void info() {

System.out.println("房子高度为 " + height + " 米");

}

void info(String s) {

System.out.println(s + ": 房子高度为 " + height + " 米");

}

}

public class MainClass {

public static void main(String[] args) {

MyClass t = new MyClass(3);

t.info();

t.info("重载方法");

//重载构造函数

new MyClass();

}

}

以上代码运行输出结果为：

房子高度为 3 米

房子高度为 3 米

重载方法: 房子高度为 3 米

无参数构造函数

## 访问修饰符

Java中定义了private（私有的）、protected（受保护的）和public（公共的）的访问修饰符，同时也定义了一个缺省的访问级别，用于声明类、属性、方法的访问权限。

* 使用public访问修饰符，类的成员可被同一包或不同包中的所有类访问，也就是说，public访问修饰符可以使类的特性公用于任何类；
* 使用protected访问修饰符允许类本身、同一包中的所有类和不同包中的子类访问；
* 如果一个类或类的成员前没有任何访问修饰符时，它们获得缺省的访问权限，缺省的可以被同一包中的其他类访问；
* private访问修饰符是限制性最大的一种访问修饰符,被声明为private的成员只能被此类中的其他成员访问，不能在类外看到。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **访问修饰符** | **本类** | **同包** | **子类** | **其他** |
| private | **√** |  |  |  |
| 默认(friendly) | **√** | **√** |  |  |
| protected | **√** | **√** | **√** |  |
| public | **√** | **√** | **√** | **√** |

实例：

首先，定义MyClass1类结构：

public class MyClass1 { public int a = 5; private int b = 10; protected int c = 20; int d = 30; public void func1() { System.out.println("func1"); } private void func2() { System.out.println("func2"); System.out.println(b); } protected void func3() { System.out.println("func3"); } void func4() { System.out.println("func4"); }}

然后，定义MyClass2类结构：

class MyClass2 {

public void func1() {

System.out.println("func1 of MyClass2");

}

}

最后，定义Test类，假如将Test类放在与MyClass1和MyClass2同一个包p1下，在Test中访问MyClass1、MyClass2及其成员的可行性如下 ：

public class Test { public void func() { MyClass1 obj1 = new MyClass1(); System.out.println(obj1.a); // 公共属性，任何地方都可以访问 System.out.println(obj1.b); // Error，私有属性类外无法访问 System.out.println(obj1.c); // 受保护属性同包的类或子类可以访问 System.out.println(obj1.d); // 缺省属性同包的类可以访问 obj1.func1(); // 公共方法，任何地方都可以访问 obj1.func2(); // Error，私有方法类外无法访问 obj1.func3(); // 受保护方法同包的类或子类可以访问 obj1.func4(); // 缺省方法同包的类可以访问 MyClass2 obj2 = new MyClass2(); // 同一包中的缺省类可以访问 }}

假如将Test类放在与MyClass1和MyClass2不同的包下，在Test中访问MyClass1、MyClass2及其成员的可行性如下 ：

public class Test {

public void func() {

MyClass1 obj1 = new MyClass1();

System.out.println(obj1.a); // 公共属性，任何地方都可以访问

System.out.println(obj1.b); // Error，私有属性类外无法访问

System.out.println(obj1.c); // Error，受保护属性同包的类或子类才可以访问

System.out.println(obj1.d); // Error，缺省属性同包的类才可以访问

obj1.func1(); // 公共方法，任何地方都可以访问

obj1.func2(); // Error，私有方法类外无法访问

obj1.func3(); // Error，受保护方法同包的类或子类才可以访问

obj1.func4(); // Error，缺省方法同包的类才可以访问

MyClass2 obj2 = new MyClass2(); // 缺省访问级别同包才可以访问

}

}

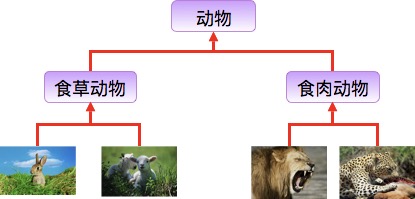
假如Test类继承了MyClass1，并且不在同一个包下，在Test中访问MyClass1、MyClass2及其成员的可行性如下 ：

public class Test extends MyClass1{ public void func() { System.out.println(a); // 公共属性，任何地方都可以访问 System.out.println(b); // Error，私有属性类外无法访问 System.out.println(c); // 受保护属性同包的类或子类可以访问 System.out.println(d); // Error，缺省属性同包的类才可以访问 func1(); // 公共方法，任何地方都可以访问 func2(); // Error，私有方法类外无法访问 func3(); // 受保护方法同包的类或子类可以访问 func4(); // Error，缺省方法同包的类才可以访问 MyClass2 obj2 = new MyClass2(); // 缺省访问级别同包才可以访问 }}

## 继承

### 为什么要有继承？

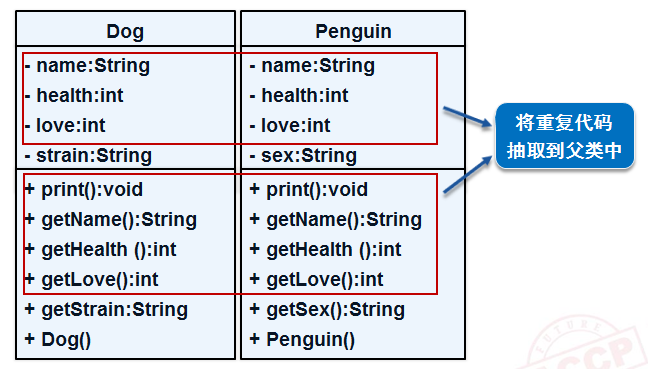
生活中的继承：



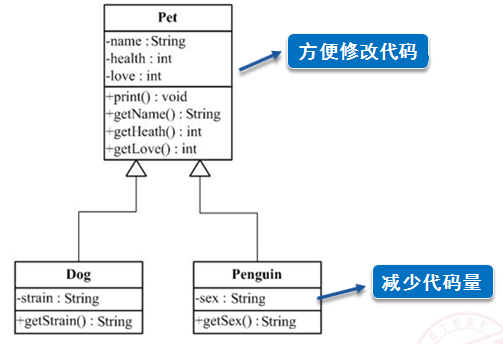
兔子和羊属于食草动物类，狮子和豹属于食肉动物类。食草动物和食肉动物又是属于动物类。所以继承需要符合的关系是：is-a，父类更通用，子类更具体。

虽然食草动物和食肉动物都是属于动物，但是两者的属性和行为上有差别，所以子类会具有父类的一般特性也会具有自身的特性。

使用继承优化设计：



优化后：



子类与父类的关系是is-a的关系，或者说是一种特殊和一般的关系。

例如Dog is a Pet。同样可以让学生继承人，让苹果继承水果，让三角形继承几何图形。

### Java中的继承

继承是Java面向对象编程技术的一块基石，它允许创建分等级层次的类。运用继承，可以创建一个通用类定义一系列一般特性。该类可以被更具体的类继承。

* 被继承的类称为父类
* 继承父类的类称为子类
* 执行继承时，子类将获得父类的属性，并具有自身特有的属性。

extends关键字用于继承类。声明一个继承父类的类的通常形式如下：

class 父类 {

}

class 子类 extends 父类 {

}

**注意：**

* 没有extends，默认父类继承了Object类
* 只能有一个父类，即单继承
* 子类继承父类的全部成员，除了private成员
* 子类与父类不在同包，使用默认访问权限的成员不能被继承
* 构造方法不能被继承

例如：

开发动物类，其中动物分别为企鹅以及老鼠，要求如下：

企鹅：属性（姓名，id），方法（吃，睡，自我介绍）

老鼠：属性（姓名，id），方法（吃，睡，自我介绍）

企鹅类：

public class Penguin {

private String name;

private int id;

public Penguin(String myName, int myid) {

name = myName;

id = myid;

}

public void eat(){

System.out.println(name+"正在吃");

}

public void sleep(){

System.out.println(name+"正在睡");

}

public void introduction() {

System.out.println("大家好！我是" + id + "号" + name + ".");

}

}

老鼠类：

public class Mouse {

private String name;

private int id;

public Mouse(String myName, int myid) {

name = myName;

id = myid;

}

public void eat(){

System.out.println(name+"正在吃");

}

public void sleep(){

System.out.println(name+"正在睡");

}

public void introduction() {

System.out.println("大家好！我是" + id + "号" + name + ".");

}

}

从这两段代码可以看出来，代码存在重复了，导致后果就是代码量大且臃肿，而且维护性不高(维护性主要是后期需要修改的时候，就需要修改很多的代码，容易出错)，所以要从根本上解决这两段代码的问题，就需要继承，将两段代码中相同的部分提取出来组成 一个父类。

公共父类：

public class Animal {

private String name;

private int id;

public Animal(String myName, int myid) {

name = myName;

id = myid;

}

public void eat(){

System.out.println(name+"正在吃");

}

public void sleep(){

System.out.println(name+"正在睡");

}

public void introduction() {

System.out.println("大家好！我是" + id + "号" + name + ".");

}

}

这个Animal类就可以作为一个父类，然后企鹅类和老鼠类继承这个类之后，就具有父类当中的属性和方法，子类就不会存在重复的代码，维护性也提高，代码也更加简洁，提高代码的复用性（复用性主要是可以多次使用，不用再多次写同样的代码） 继承之后的代码：

企鹅类：

public class Penguin extends Animal {

public Penguin(String myName, int myid) {

super(myName, myid);

}

}

老鼠类：

public class Mouse extends Animal {

public Mouse(String myName, int myid) {

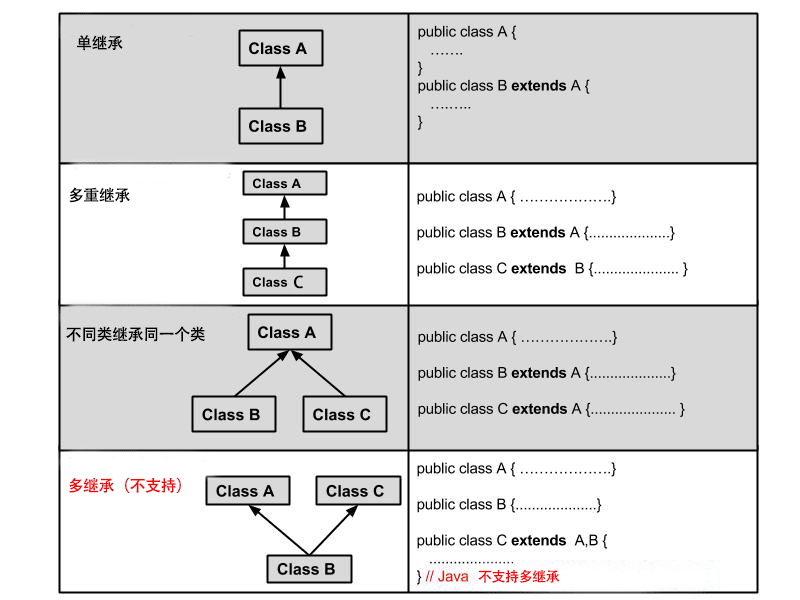
super(myName, myid);

}

}

### 继承的类型和特性

Java 不支持多继承，但支持多重继承。



继承的特性：

* 子类拥有父类非 private 的属性、方法。
* 子类可以拥有自己的属性和方法，即子类可以对父类进行扩展。
* 子类可以用自己的方式实现父类的方法。
* Java 的继承是单继承，但是可以多重继承，单继承就是一个子类只能继承一个父类，多重继承就是，例如 A 类继承 B 类，B 类继承 C 类，所以按照关系就是 C 类是 B 类的父类，B 类是 A 类的父类，这是 Java 继承区别于 C++ 继承的一个特性。
* 提高了类之间的耦合性（继承的缺点，耦合度高就会造成代码之间的联系越紧密，代码独立性越差）。

### super与this

**super**关键字：我们可以通过super关键字来实现对父类成员的访问，用来引用当前对象的父类。

**this**关键字：指向自己的引用。

例如：

class Animal {

void eat() {

System.out.println("animal : eat");

}

}

class Dog extends Animal {

void eat() {

System.out.println("dog : eat");

}

void eatTest() {

this.eat(); // this 调用自己的方法

super.eat(); // super 调用父类方法

}

}

public class Test {

public static void main(String[] args) {

Animal a = new Animal();

a.eat();

Dog d = new Dog();

d.eatTest();

}

}

输出结果：

animal : eat

dog : eat

animal : eat

**final** 关键字声明类可以把类定义为不能继承的，即最终类；或者用于修饰方法，该方法不能被子类重写：

声明类：

final class 类名 {//类体}

声明方法：

修饰符(public/private/default/protected) final 返回值类型 方法名(){//方法体}

### 构造方法继承

子类是不继承父类的构造器（构造方法或者构造函数）的，它只是调用（隐式或显式）。如果父类的构造器带有参数，则必须在子类的构造器中显式地通过 super 关键字调用父类的构造器并配以适当的参数列表。

如果父类构造器没有参数，则在子类的构造器中不需要使用 super 关键字调用父类构造器，系统会自动调用父类的无参构造器。

例如：

class SuperClass {

private int n;

SuperClass(){

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

}

SuperClass(int n) {

System.out.println("SuperClass(int n)");

this.n = n;

}

}

// SubClass 类继承

class SubClass extends SuperClass{

private int n;

SubClass(){ // 自动调用父类的无参数构造器

System.out.println("SubClass");

}

public SubClass(int n){

super(300); // 调用父类中带有参数的构造器

System.out.println("SubClass(int n):"+n);

this.n = n;

}

}

// SubClas2 类继承

class SubClass2 extends SuperClass{

private int n;

SubClass2(){

super(300); // 调用父类中带有参数的构造器

System.out.println("SubClass2");

}

public SubClass2(int n){ // 自动调用父类的无参数构造器

System.out.println("SubClass2(int n):"+n);

this.n = n;

}

}

public class TestSuperSub{

public static void main (String args[]){

System.out.println("------SubClass 类继承------");

SubClass sc1 = new SubClass();

SubClass sc2 = new SubClass(100);

System.out.println("------SubClass2 类继承------");

SubClass2 sc3 = new SubClass2();

SubClass2 sc4 = new SubClass2(200);

}

}

输出结果为：

------SubClass 类继承------

SuperClass()

SubClass

SuperClass(int n)

SubClass(int n):100

------SubClass2 类继承------

SuperClass(int n)

SubClass2

SuperClass()

SubClass2(int n):200

## 包

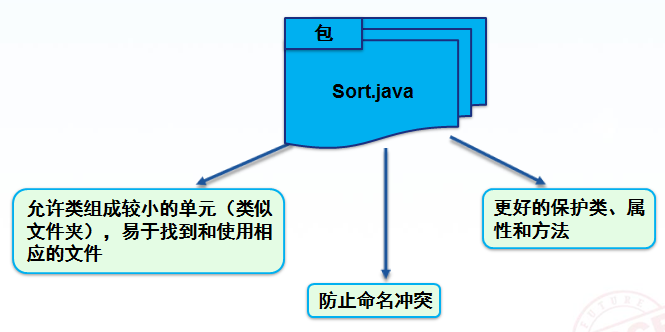
### 为什么需要包

Windows树形文件系统：

* 文档分门别类，易于查找和管理
* 使用目录解决文件同名冲突问题



Java中的包：解决类的同名问题。



### 包的创建

1. 包的创建

Java允许使用包将多个类组织在一起。借助于包可以方便的组织管理类，并将自定义的类与其它的类库分开管理。Java就是使用包来管理类库的 。 使用包维护类库比较简单，只要保证在同一个包下不存在同名的类即创建一个包也比较简单：只要将package命令作为一个Java源文件的第一句就可以，该文件中定义的任何类将属于指定的包。

* 用package声明包，以分号结尾。
* 作为Java程序的第一条语句。

**package com.hyg.classandobject;** //声明包

public class School {

包名

//……

public String toString() {

//……

}

}

1. 包中类的访问

一个类可以访问其所在包的所有类。访问其他包的类有如下两种方式访问 ：

使用import语句导入要访问的类 ，如：

import java.util.\*;

import mypackage.school.Student ;

使用的类名前直接添加完整的包名 ，如：

java.util.Date now = new java.util.Date();

mypackage.school.Student tom = new mypackage.school.Student();

注意：\* 指明导入当前包的所有类，不能使用类似于java. \* 的语句来导入以java为前缀的所有包的所有类。

### 包的命名规范

* 包名由小写字母组成，不能以圆点开头或结尾
* 包名之前最好加上唯一的前缀，通常使用组织倒置的网络域名

package net.javagroup.mypackage;

* 包名后续部分依不同机构内部的规范不同而不同

package net.javagroup.research.powerproject;

项目名

部门名

## 编码实操

### 根据年龄显示门票价格

根据输入的顾客年龄，显示相应的门票价格。

顾客类：

import java.util.Scanner;

public class Visitor {

String name; // 姓名

int age; // 年龄

public void show() {

Scanner input = new Scanner(System.in);

while (!"n".equals(name)) {

if (age >= 18 && age <= 60) { // 判断年龄

System.out.println(name + "的年龄为：" + age + "，门票价格为：20元\n");

} else {

System.out.println(name + "的年龄为：" + age + "，门票免费\n");

}

System.out.print("请输入姓名：");

name = input.next(); // 给name属性赋值

if (!"n".equals(name)) {

System.out.print("请输入年龄：");

age = input.nextInt(); // 给age属性赋值

}

}

System.out.print("退出程序");

}

}

程序运行主类：

import java.util.Scanner;

public class InitialVistor {

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

Visitor v = new Visitor(); // 创建对象

System.out.print("请输入姓名：");

v.name = input.next(); // 给name属性赋值

System.out.print("请输入年龄：");

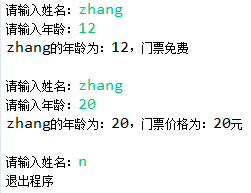
v.age = input.nextInt(); // 给age属性赋值

v.show(); // 调用显示信息方法

}

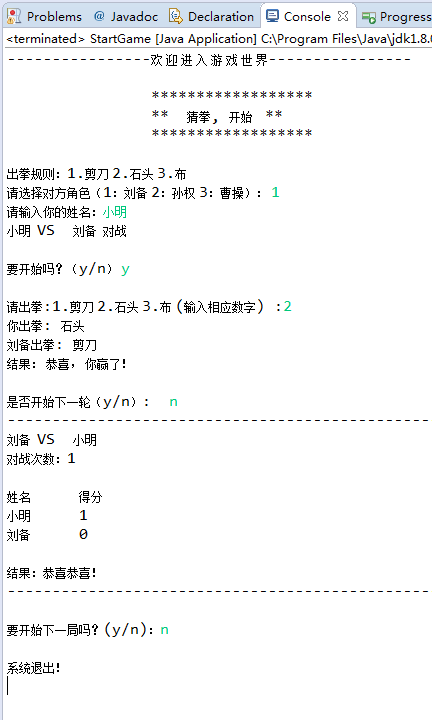
}

运行InitialVistor.java类，控制台显示效果如下：



### 人机猜拳对战游戏

1. 游戏效果



1. 程序编写

计算机类：

package com.game.guess;

/\*\*

\* 计算机类

\*/

public class Computer {

String name = "电脑"; // 名字

int score = 0;; // 积分

/\*\*

\* 出拳

\*

\* @return 出拳结果：1.剪刀 2.石头 3.布

\*/

public int showFist() {

// 产生随机数,表示电脑出拳

int show = (int) (Math.random() \* 10) % 3 + 1;

// 输出出拳结果并返回

switch (show) {

case 1:

System.out.println(name + "出拳: 剪刀");

break;

case 2:

System.out.println(name + "出拳: 石头");

break;

case 3:

System.out.println(name + "出拳: 布");

break;

}

return show;

}

}

人类：

package com.game.guess;

/\*\*

\* 人类

\*/

import java.util.Scanner;

public class Person {

String name = "匿名";

int score;

/\*\*

\* 出拳

\*/

public int showFirst() {

// 接收用户的选择

Scanner input = new Scanner(System.in);

System.out.print("\n请出拳:1.剪刀 2.石头 3.布 (输入相应数字) :");

int show = input.nextInt();

// 输出出拳结果，并返回

switch (show) {

case 1:

System.out.println("你出拳: 剪刀");

break;

case 2:

System.out.println("你出拳: 石头");

break;

case 3:

System.out.println("你出拳: 布");

break;

}

return show;

}

}

游戏类：

package com.game.guess;

import java.util.Scanner;

/\*\*

\* 游戏类

\*/

public class Game {

Person person; // 甲方

Computer computer; // 乙方

int count; // 对战次数

/\*\*

\* 初始化

\*/

public void initial() {

person = new Person();

computer = new Computer();

count = 0;

}

/\*\*

\* 开始游戏

\*/

public void startGame() {

System.out.println("----------------欢 迎 进 入 游 戏 世 界----------------");

System.out.println("\n\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println("\t\t\*\* 猜拳, 开始 \*\*");

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

System.out.println("\n出拳规则：1.剪刀 2.石头 3.布");

Scanner input = new Scanner(System.in);

String exit = "n"; // 退出系统

do {

initial(); // 初始化

/\* 选择对方角色 \*/

System.out.print("请选择对方角色（1：刘备 2：孙权 3：曹操）： ");

int role = input.nextInt();

if (role == 1) {

computer.name = "刘备";

} else if (role == 2) {

computer.name = "孙权";

} else if (role == 3) {

computer.name = "曹操";

}

// 扩展功能1：输入用户姓名

/\* 输入用户姓名 \*/

System.out.print("请输入你的姓名：");

person.name = input.next();

System.out.println(person.name + " VS " + computer.name + " 对战\n");

// 扩展功能1结束

System.out.print("要开始吗？（y/n） ");

String start = input.next(); // 开始每一局游戏

int perFist; // 用户出的拳

int compFist; // 计算机出的拳

while (start.equals("y")) {

/\* 出拳 \*/

perFist = person.showFirst();

compFist = computer.showFist();

/\* 裁决 \*/

if ((perFist == 1 && compFist == 1) || (perFist == 2 && compFist == 2)

|| (perFist == 3 && compFist == 3)) {

System.out.println("结果:和局,真衰！嘿嘿,等着瞧吧 !\n"); // 平局

} else if ((perFist == 1 && compFist == 3)

|| (perFist == 2 && compFist == 1)

|| (perFist == 3 && compFist == 2)) {

System.out.println("结果： 恭喜， 你赢了！"); // 用户赢

person.score++;

} else {

System.out.println("结果说:^\_^,你输了，真笨!\n"); // 计算机赢

computer.score++;

}

count++;

System.out.print("\n是否开始下一轮（y/n）: ");

start = input.next();

}

/\* 显示结果 \*/

showResult();

// 扩展功能3：循环游戏，直到退出系统

System.out.print("\n要开始下一局吗？(y/n)：");

exit = input.next();

System.out.println();

// 扩展功能3结束

} while (!exit.equals("n"));

System.out.println("系统退出！");

}

/\*\*

\* 显示比赛结果

\*/

public void showResult() {

/\* 显示对战次数 \*/

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

System.out.println(computer.name + " VS " + person.name);

System.out.println("对战次数：" + count);

// 扩展功能2：显示最终的得分

System.out.println("\n姓名\t得分");

System.out.println(person.name + "\t" + person.score);

System.out.println(computer.name + "\t" + computer.score + "\n");

// 扩展功能2结束

/\* 显示对战结果 \*/

int result = calcResult();

if (result == 1) {

System.out.println("结果：打成平手，下次再和你一分高下！");

} else if (result == 2) {

System.out.println("结果：恭喜恭喜！"); // 用户获胜

} else {

System.out.println("结果：呵呵，笨笨，下次加油啊！"); // 计算机获胜

}

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

}

/\*\*

\* 计算比赛结果

\*

\* @return 1:战平；2:用户赢；3:电脑赢

\*/

public int calcResult() {

if (person.score == computer.score) {

return 1; // 战平

} else if (person.score > computer.score) {

return 2; // 用户赢

} else {

return 3; // 电脑赢

}

}

}

游戏启动类：

package com.game.guess;

/\*\*

\* 人机互动版猜拳游戏

\* 程序入口

\*/

public class StartGame {

public static void main(String[] args) {

Game game = new Game();

game.startGame();

}

}

## 练习题

1. 在如下所示的JAVA代码中，this关键字是指（）。

public class Person{

private String name;

public void setName(String name){

this.name=name;

}

}

A) Person类

B) Person类自身对象的引用

C) setName()方法的参数name

D) setName()方法

答案：B

2. 在Java中，下列关于方法重载的说法中错误的是（ ）（多选）。

A. 方法重载要求方法名称必须相同

B. 重载方法的参数列表必须不一致

C. 重载方法的返回类型必须一致

D. 一个方法在所属的类中只能被重载一次

答案：CD

3. 给定Java代码如下所示，在横线处新增下列（ ）方法，是对cal方法的重载。（多选）

public class Test{

public void cal(int x, int y, int z) {}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

}

A. public int cal(int x, int y, float z){ return 0; }

B. public int cal(int x, int y, int z){ return 0; }

C. public void cal(int x, int z){ }

D. public void cal(int z, int y, int x){ }

答案：AC

4. 在Java中，下面对于构造函数的描述正确的是（ ）。

A. 类必须显式定义构造函数

B. 构造函数的返回类型是void

C. 构造函数和类有相同的名称，并且不能带任何参数

D. 一个类可以定义多个构造函数

答案：D

5. 下面Java代码的运行结果是（ ）。

class Penguin {

private String name=null; // 名字

private int health=0; // 健康值

private String sex=null; // 性别

public void Penguin() {

health = 10;

sex = "雄";

System.out.println("执行构造方法。");

}

public void print() {

System.out.println("企鹅的名字是" + name +

"，健康值是" + health + "，性别是" + sex+ "。");

}

public static void main(String[] args) {

Penguin pgn = new Penguin();

pgn.print();

}

}

A. 企鹅的名字是null，健康值是10，性别是雄。

B. 执行构造方法。

企鹅的名字是null，健康值是0，性别是null。

C. 企鹅的名字是null，健康值是0，性别是null。

D. 执行构造方法。

企鹅的名字是null，健康值是10，性别是雄。

答案：C

6. 分析如下所示的Java代码，其中this关键字的意思是（ ）。

public class Test {

private String name;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name; //this关键字所在的行

}

}

A. name属性

B. Test类的内部指代自身的引用

C. Test类的对象引用Test类的其他对象

D. 指所在的方法

答案：B

7. 在Java中，以下程序编译运行后的输出结果为（ ）。

public class Test {

int x, y;

Test(int x, int y) {

this.x = x;

this.y = y;

}

public static void main(String[] args) {

Test pt1, pt2;

pt1 = new Test(3, 3);

pt2 = new Test(4, 4);

System.out.print(pt1.x + pt2.x);

}

}

答案：D

8. 在Java语言中，下列关于类的继承的描述，正确的是（ ）。

A. 一个类可以继承多个父类

B. 一个类可以具有多个子类

C. 子类可以使用父类的所有方法

D. 子类一定比父类有更多的成员方法

9. Java中，如果类C是类B的子类，类B是类A的子类，那么下面描述正确的是（ ）。

A. C不仅继承了B中的公有成员，同样也继承了A中的公有成员

B. C只继承了B中的成员

C. C只继承了A中的成员

D. C不能继承A或B中的成员

10. 给定一个Java程序Test.java的代码如下所示，编译时，会出现以下（ ）情况。

class Parent {

public int count() { // 第1行

return 0;

}

}

public class Test extends Parent {

private int i;

public int count() { // 第2行

return i % 9; // 第3行

}

}

A. 编译通过

B. 在第1行引发编译错误

C. 在第2行引发编译错误

D. 在第3行引发编译错误

11. 给定如下一个Java源文件Child.java，编译并运行Child.java，以下结果正确的是（ ）。

class Parent1 {

Parent1(String s){

System.out.println(s);

}

}

class Parent2 extends Parent1{

Parent2(){

System.out.println("parent2");

}

}

public class Child extends Parent2 {

public static void main(String[] args) {

Child child = new Child();

}

}

A. 编译错误：没有找到构造器Child()

B. 编译错误：没有找到构造器Parent1()

C. 正确运行，没有输出值

D. 正确运行，输出结果为：parent2

12. 给定Java程序Child.java的代码如下所示，则编译运行该类的结果是（ ）。

class Parent {

Parent() {

System.out.println("parent");

}

}

public class Child extends Parent {

Child(String s) {

System.out.println(s);

}

public static void main(String[] args) {

Child child = new Child("child");

}

}

A. 输出：child

B. 输出：child

parent

C. 输出：parent

child

D. 编译错误

13. 下列选项中关于Java中super关键字的说法错误的是（ ）。

A. super关键字是在子类对象内部指代其父类对象的引用

B. super关键字不仅可以指代子类的直接父类，还可以指代父类的父类

C. 子类可以通过super关键字调用父类的方法

D. 子类可以通过super关键字调用父类的属性

14. 分析如下所示的Java代码，则选项中的说法正确的是（ ）（多选）。

class Parent{

public String name;

public Parent(String pName){

this.name = pName;

}

}

public class Test extends Parent { //1

public Test(String Name){ //2

name="hello"; //3

super("kitty"); //4

}

}

A. 第2行错误，Test类的构造函数中参数名称应与其父类构造函数中的参数名相同

B. 第3行错误，应使用super关键字调用父类的name属性，改为super.name="hello";

C. 第4行错误，调用父类构造方法的语句必须放在子类构造方法中的第一行

D. 程序编译通过，无错误

15. 以下关于Object类说法错误的是（ ）。

A. 一切类都直接或间接继承自Object类

B. 接口亦继承Object类

C. Object类中定义了toString()方法

D. Object类在java.lang包中

答案：8.B 9.A 10.A 11.B 12.C 13.B 14.AC 15.B

# 字符串

## 学习目标

* 掌握堆内存与栈内存的数据存储
* 掌握equals()和==的异同
* 掌握常用字符串方法

## 堆内存与栈内存

Java把内存分成两种，一种叫做**栈内存**，一种叫做**堆内存**。

在函数中定义的一些基本类型的变量和对象的引用变量都是在函数的栈内存中分配。当在一段代码块中定义一个变量时，java就在栈中为这个变量分配内存空间，当超过变量的作用域后，java会自动释放掉为该变量分配的内存空间，该内存空间可以立刻被另作他用。

堆内存用于存放由new创建的对象和数组。

在Java代码，常常会使用到这样的类的声明实例化：

Person per = new Person();

这其实是包含了两个步骤，声明和实例化：

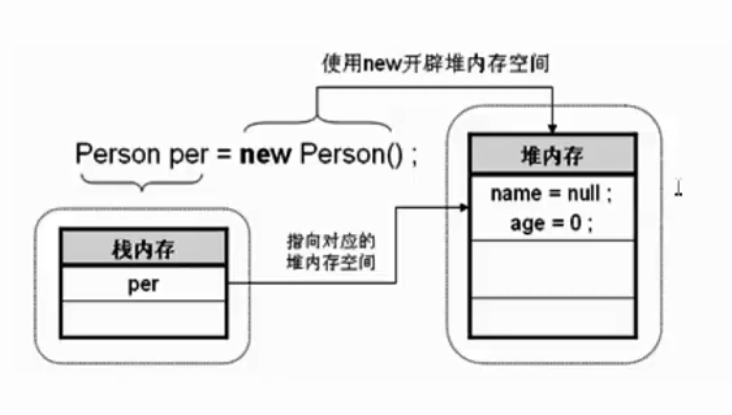
Person per = null; //声明一个名为Person类的对象per

per = new Person(); // 实例化这个per对象

**声明**指的是创建类的对象的过程；

实例化指的是用关键词new来开辟内存空间。

它们在内存中的划分是这样的：



## 字符串比较

String类提供了equals( )方法，比较存储在两个字符串对象的内容是否一致。

public class Login {

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

String uname,pwd;

System.out.print("请输入用户名： ");

uname=input.next();

System.out.print("请输入密码： ");

pwd=input.next();

if( **uname.equals("TOM") && pwd.equals("1234567")** ){

System.out.print("登录成功！ ");

}else{

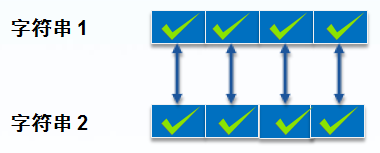
System.out.print("用户名或密码不匹配，登录失败！");

}

}

}

equals()方法比较原理：



equals()：检查组成字符串内容的字符是否完全一致

==：判断两个字符串在内存中的地址，即判断是否是同一个字符串对象。

例如以下代码：

public class StringDemo {

public static void main(String args[]) {

String str1 = "Hello";

String str2 = new String("Hello");

String str3 = str2; // 引用传递

System.out.println(str1 == str2); // false

System.out.println(str1 == str3); // false

System.out.println(str2 == str3); // true

System.out.println(str1.equals(str2)); // true

System.out.println(str1.equals(str3)); // true

System.out.println(str2.equals(str3)); // true

}

}

上方第4行代码中，new了一个对象，用“==”比较str1和str2，返回的结果却是false；而用用“equals”比较str1和str2，返回的结果是true。

为了分析上面的代码，必须首先分析堆内存空间和栈内存空间，这一点非常重要：



“==”和equals()的区别？

* ==：比较的是两个字符串内存地址（堆内存）是否相等；
* equals()：比较的是两个字符串的内容是否相等。

## 内存中的字符串

我们已经知道，创建字符串对象有两种方式：一种是通过初始化的方式创建，例如以下代码：

String str="Hello";

另一种是使用new关键字创建，例如以下代码：

String str = new String("Hello");

Java为了避免产生大量的字符串对象，设计了一个**字符串池**（String Pool），通过初始化方式创建的字符串对象都会存在于**字符串池**中，且字符串池中的字符串不会重复，以便可以被共享使用，提高存储效率。

其工作原理是：

当使用初始化的方式创建对象时，JVM会首先检查字符串池中是否存在值相等的字符串，如果存在，则不进行创建，而是直接返回字符串池中该字符串的引用地址。如果不存在，则创建该对象，并放入字符串池中，返回新创建的字符串的引用地址。

当使用new关键字创建对象时，JVM仍然首先检查字符串池中是否存在要创建的字符串，如果不存在，则在字符串池中创建一个字符串对象，**然后**在堆内存中继续创建一个字符串对象，返回该对象的引用地址。如果存在，则只在堆内存中创建一个字符串对象，返回该对象的引用地址。

使用new关键字创建字符串对象，实际上调用了String类的构造函数，构造函数的源码如下：

public String(String original) {

this.value = original.value;

this.hash = original.hash;

}

从上述代码可以看出，构造函数需要传入一个String对象，然后将该对象的值作为当前对象的值。因此，执行代码String str = new String("Hello");时，如果字符串池中不存在字符串“Hello”，则会创建两个对象，因为字符串“Hello”本身就是一个对象，通过关键字new又创建了一个对象。

接下来通过以下代码进一步讲解：

String str1 = new String("Hello");➊

String str2 = new String("Hello");➋

String str3 = "Hello"; ➌

String str4 = "Hello"; ➍

System.out.println(str1 == str2); //false

System.out.println(str3 == str4); //true

System.out.println(str2 == str3); //false

System.out.println(str1.equals(str2)); //true

System.out.println(str3.equals(str4)); //true

String str5=str4;➎

System.out.println(str4 == str5); //true

str5="World";➏

System.out.println(str4);//Hello

System.out.println(str5);//World

当代码运行到第➊行后，JVM首先会查找字符串池中是否存在值为“Hello”的对象，发现不存在，则在字符串池中创建一个“Hello”对象，在堆内存中也创建一个“Hello”对象，并将堆内存中的“Hello”对象的引用（内存地址）赋给变量str1，此时字符串在内存中的存储示意图如下：

String str1 = new String("Hello");



字符串在内存中的存储示意图

当代码运行到第➋行后，JVM首先会查找字符串池中是否存在值为“Hello”的对象，发现存在，则只在堆内存中创建一个值为“Hello”的对象，此时字符串在内存中的存储示意图如下：

String str2 = new String("Hello");



字符串在内存中的存储示意图

当代码运行到第➌行后，JVM首先会查找字符串池中是否存在值为“Hello”的对象，发现存在该对象，则将该对象的引用赋给变量str3，此时字符串在内存中的存储示意图如下：

String str3 = "Hello";



字符串在内存中的存储示意图

当代码运行到第➍行后，同理，会将“Hello”对象的引用赋给变量str4。

String str4 = "Hello";

当代码运行到第➎行后，会将str4的引用赋给str5，此时字符串在内存中的存储示意图如下：

String str5=str4;



字符串在内存中的存储示意图

当代码运行到第➏行后，JVM会首先查找字符串池中是否存在值为“World”的对象，发现不存在，则创建一个“World”对象，并将该对象的引用赋给变量str5，此时字符串在内存中的存储示意图如下：

str5="World";



字符串在内存中的存储示意图

到此，str5的引用是对象“World”；str4和str3的引用是同一个对象“Hello”；str2和str1的引用是不同的对象“Hello”。

## 常用方法

1. 字符串连接

使用“+”连接字符串：

int sqlScore = 80; //sql成绩

int javaScore = 90; //java成绩

double htmlScore = 86.7; //html成绩

String scoreSheet = "SQL:" + sqlScore + " Java:" + javaScore + " HTML:" + htmlScore;

使用String类的concat()方法，语法：A.concat(B)，B字符串将被连接到A字符串后面。

String s = new String("你好，");

String name = new String("张三！");

String sentence = s.concat(name);

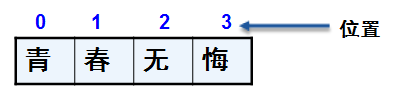
System.out.println(sentence);

上述代码的输出结果为：你好，张三！

1. 字符串查找

返回出现第一个匹配的位置，如果没有找到字符或字符串，则返回-1：

|  |  |
| --- | --- |
| **方 法** | **说 明** |
| public int indexOf(int ch) | 搜索第一个出现的字符ch（或字符串value） |
| public int indexOf(String value) |
| public int lastIndexOf(int ch) | 搜索最后一个出现的字符ch（或字符串value） |
| public int lastIndexOf(String value) |



截取字符串：

|  |  |
| --- | --- |
| **方 法** | **说 明** |
| public String substring(int index) | 截取从位置索引开始的字符串部分 |
| public String substring(int beginindex, int endindex) | 截取beginindex和endindex之间的字符串部分 |
| public String trim() | 返回一个前后不含任何空格的调用字符串的副本 |

substring(int beginindex, int endindex)：截取beginindex和endindex之间的字符串部分，包括位置beginindex但不包括位置endindex。

示例：检查文件和邮箱格式。

import java.util.\*;

public class Verify{

public static void main(String[] args) {

// 声明变量

boolean fileCorrect = false; //标识文件名是否正确

boolean emailCorrect = false; //标识E-mail是否正确

System.out.print("---欢迎进入作业提交系统---");

Scanner input = new Scanner(System.in);

System.out.println("请输入Java文件名: ");

String fileName = input.next();

System.out.print("请输入你的邮箱:");

String email = input.next();

//检查Java文件名

int index = fileName.lastIndexOf("."); //"."的位置

if(index!=-1 && index!=0 &&

fileName.substring(index+1,

fileName.length()).equals("java")){

fileCorrect = true; //标识文件名正确

}else{

System.out.println("文件名无效。");

}

//检查你的邮箱格式

if(email.indexOf('@')!=-1 &&

email.indexOf('.')>email.indexOf('@')){

emailCorrect = true; //标识E-mail正确

}else{

System.out.println("E-mail无效。");

}

//输出检测结果

if(fileCorrect && emailCorrect){

System.out.println("作业提交成功!");

}else{

System.out.println("作业提交失败!");

}

}

}

1. 字符串分割

String类提供了split()方法，将一个字符串分割为子字符串，结果作为字符串数组返回。

例如，将一行歌词按空格进行拆分成多行输出：

public class Lyric {

/\*\*

\* 拆分歌词

\*/

public static void main(String[] args) {

String words="长亭外 古道边 芳草碧连天 晚风扶 柳笛声残 夕阳山外山";

String[] printword=new String[100];//接收数组

System.out.println("\*\*\*原歌词格式\*\*\*\n"+words);

System.out.println("\n\*\*\*拆分后歌词格式\*\*\*");

printword=words**.split(" ")**;//按照空格进行拆分

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

System.out.println(printword[i]);//打印输出

}

}

}

1. StringBuffer类

对字符串频繁修改（如字符串连接）时，使用StringBuffer类可以大大提高程序执行效率。

StringBuffer的声明：

//创建空StringBuffer对象：

StringBuffer sb = new StringBuffer();

//创建一个变量存储字符串aaa：

StringBuffer sb = new StringBuffer("aaa");

StringBuffer的使用：

sb.append("\*\*"); //追加字符串

sb.insert (1, "\*\*"); //插入字符串

sb.toString(); //转化为String类型

示例如下：

public class sbAppend {

public static void main(String[ ] args) {

StringBuffer sb = new StringBuffer("青春无悔");

int num=110;

StringBuffer sb1 = sb.append("我心永恒");

System.out.println(sb1);

StringBuffer sb2 = sb1.append('啊');

System.out.println(sb2);

StringBuffer sb3 = sb2.append(num);

System.out.println(sb3);

}

}

1. 其它方法

charAt() : 返回字符串中某个特定位置的字符

indexOf() :返回字符串中某个特定字符或子字符串首次出现的索引

toUpperCase() : 将字符串内的所有字符从小写改为大写

toLowerCase() : 将字符串内的所有字符从大写改为小写

## 编码实操

将一个数字字符串转换成逗号分隔的数字串，即从右边开始每三个数字用逗号分隔。

例如，输入12345678，输出12,345,678。

代码如下：

import java.util.\*;

public class TestInsert {

/\*\*

\* 每隔三位插入逗号

\*

\*/

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

//接收数字串，存放于StringBuffer类型的对象中

System.out.print("请输入一串数字： ");

String nums = input.next();

StringBuffer str=new StringBuffer(nums);

//从后往前每隔三位添加逗号

for(int i=str.length()-3;i>0;i=i-3){

**str.insert(i,',');**

}

System.out.print(str);

}

}

## 练习题

1. 如果要打印输出“小鱼儿”，应填入的代码是什么？

String word = "Hello, ";

word = word.trim();

String s = word.concat("小鱼儿!");

int index1 = s.indexOf(',');

int index2 = s.indexOf('!');

System.out.println(s.substring(\_\_\_\_\_\_, \_\_\_\_\_\_\_));

答案：index1+1 index2

2. 给定一个字符串，判断该字符串中是否包含某个子串.如果包含，求出子串的所有出现位置。

如："abcbcbabcb34bcbd"中，"bcb"子串的出现位置为：1，7，12。字符串和子串均由用户输入。

3. .给定一个长度，随机产生一个该长度的字符串，由大写、小写字母以及数字组成。

Java中随机数的生成：

java.util.Random r = new java.util.Random();

int a = r.nextInt(100): a 0-99的随机数。

4. 给定一个由数字组成的字符串，如："1239586838923173478943890234092"，统计

出每个数字出现的次数。

2、3、4题答案：

package Chp10;

import java.util.HashMap;

import java.util.Map;

import java.util.Random;

import java.util.Scanner;

import java.util.Set;

public class TestString2{

public static void Method\_One(){

Scanner s = new Scanner(System.in);

System.out.println("请输入一个父串");

String str = s.next();

System.out.println("请输入一个子串");

String str1 = s.next();

int stat = 0;

while (true){

int i = str.indexOf(str1,stat);

if (i==-1){

break;

}

System.out.println(i);

stat = i+str1.length();

}

}

public static void Method\_Two(){

System.out.println("请输入一个长度");

Scanner s = new Scanner(System.in);

int n = s.nextInt();

char[] cs = new char[n];

Random random = new Random();

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

int a = random.nextInt(62);

if(a>=0 && a<=25){

a+=65;

}else if (a>25 && a<=51){

// 97 - 26 = 71

a+=71;

}else{

// 52-48 = 4

a-=4;

}

cs[i] = (char)a;

}

System.out.println(cs);

}

public static void Method\_Three(){

Scanner scanner = new Scanner(System.in);

String string = scanner.next();

int[] c = new int[10];

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

char c1 = string.charAt(i); //48-57

int a = c1-48; // 类型转换

c[a]++;

}

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

System.out.println(i+" 出现的次数： "+c[i]);

}

System.out.println("=========方法二==========");

String s = "12548954231351486214894562";

char[] ch = s.toCharArray();

Map<Character, Integer> map = new HashMap<Character, Integer>();

Set<Character> set = map.keySet();

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

if (set.contains(ch[i])){// 判断map中已经有的char值

map.put(ch[i], map.get(ch[i])+1);

} else {

map.put(ch[i], 1); // 第一次放入map中

}

}

Set<Character> set2 = map.keySet();

for (Character character : set2){

System.out.println(character+"=========="+map.get(character));

}

}

public static void main(String[] args){

System.out.println("第一题如下:");

Method\_One();

System.out.println("第二题如下:");

Method\_Two();

System.out.println("第三题如下:");

Method\_Three();

}

}

# 抽象类与多态

## 学习目标

* 掌握抽象类的意义和使用
* 掌握多态的意义和使用

## 抽象类

### 为什么要用抽象类

问你个问题，你知道什么是“东西”吗？什么是“物体”吗？

“麻烦你，小王。帮我把那个东西拿过来好吗”

在生活中，你肯定用过这个词－－东西。

小王：“你要让我帮你拿那个水杯吗？”

你要的是水杯类的对象。而东西是水杯的父类。通常东西类没有实例对象，但我们有时需要东西的引用指向它的子类实例。

在面向对象的概念中，所有的对象都是通过类来表述，但并不是所有的类都是用来描绘对象的，如果一个类中没有包含足够的信息来描绘一类具体的对象，这样的类就是抽象类。

抽象类往往用来表征对问题领域进行分析、设计中得出的抽象概念，是对一系列看上去不同，但是本质上相同的具体概念的抽象。

以下代码实例化没有意义：

Pet pet = new Pet ("贝贝",20,40);

pet.print();

可以使用抽象类限制实例化：

public **abstract** class Pet {

}

### 抽象类的使用

抽象类需要注意以下几点：

* abstract放在class前，指明该类是抽象类
* abstract放在方法声明中，则该方法是抽象方法，抽象方法没有方法体
* 一个抽象类可以含有多个抽象方法，也可以含有已实现的方法
* 抽象方法必须在**子类中**被实现，除非子类是抽象类

抽象类示例：

/\*\*形状抽象类\*\*/

public abstract class Shape { double dim; public Shape(double dim) { this.dim = dim; } // 抽象方法,获得面积 public abstract double callArea(); // 抽象方法，获得周长 public abstract double callPerimeter();}

定义Shape类的一个子类Circle：

/\*\*圆形类\*\*/

public class Circle extends Shape { public Circle(double dim) { super(dim); } // 实现抽象方法callArea() public double callArea() { // 返回圆的面积 return 3.14 \* dim \* dim; } // 实现抽象方法callPerimeter() public double callPerimeter() { // 返回圆的周长 return 2 \* 3.14 \* dim; } public static void main(String[] args) { // 声明一个Shape对象，指向实现它的子类对象 Shape shape = new Circle(10); // 调用callArea()求圆的面积，并输出 System.out.println("圆的面积是：" + shape.callArea() ); // 调用callPerimeter()求圆的周长，并输出 System.out.println("圆的周长是：" + shape.callPerimeter() ); }}

抽象类虽然具备类的形式，但由于其“抽象”性，不能定义抽象类的实例，即不能为抽象类分配具体空间，例如以下代码是错误的：

Shape circle= new Shape(3); //抽象类不能实例化

可以通过如下方式来实例一个抽象类：

Shape someShape;

//引用Circle类的实例对象

someShape = new Circle(5);

someShape.callArea();

注意：

* 抽象类不能实例化，但可以指向一个实现它的子类对象；
* abstract不能与final同时修饰一个类；
* abstract不能和static、private、final或native并列

### final用法

final关键字表示的不可变的。一个类不希望被其它类继承，可以使用final，Java中的String类就是一个final类。

public **final** class Penguin extends Pet {

//…

}

方法不希望被重写，可以使用final，final修饰的方法表示此方法已经是“最后的、最终的”含义，即父类的final方法是不能被子类所覆盖的，也就是说子类是不能够存在和父类一模一样的方法的。

public **final** void print () {

//…

}

属性不希望被修改，可以使用final：

public class Penguin {

**final** String home ="南极";// 居住地

public void setHome(String name){

this.home=home; //错误，不可再赋值

}

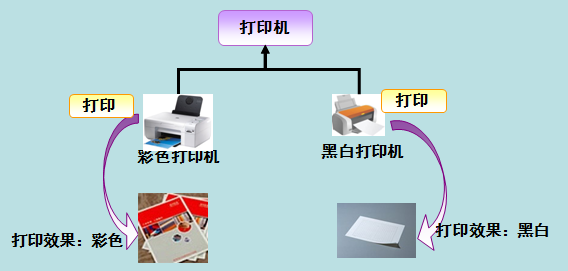
}

## 多态

### 为什么使用多态

多态是指，**同一个事件**发生在**不同的对象**上会产生不同的结果。

比如，同样是打印操作，不同打印机打印的照片效果不一样：



比如，我们按下 F1 键这个动作：

如果当前在 Flash 界面下弹出的就是 AS 3 的帮助文档；

如果当前在 Word 下弹出的就是 Word 帮助；

在 Windows 下弹出的就是 Windows 帮助和支持。

再比如，宠物饿了，需要主人给宠物喂食，不同宠物吃的东西不一样：

狗狗吃狗粮；

企鹅吃鱼。

### 多态的实现

多态的实现步骤：

1. 编写父类
2. 编写子类，子类重写父类方法
3. 运行时，使用父类的类型、子类的对象

Pet pet = new Dog();

实现多态的两种形式：

* 使用父类作为方法形参实现多态
* 使用父类作为方法返回值实现多态

1. 使用父类作为方法形参实现多态

以下实现一个主人喂食宠物例子：

1. 新建宠物类Pet，是一个抽象类

/\*\*

\* 宠物类，狗狗和企鹅的父类

\*/

public abstract class Pet {

/\*\*

\* 抽象方法eat(),负责宠物吃饭功能。

\*/

public abstract void eat();

}

1. 新建狗狗类，继承Pet，并实现吃食方法eat()

/\*\*

\* 狗狗类，宠物的子类

\*/

public class Dog extends Pet {

/\*\*

\* 实现吃食方法。

\*/

public void eat() {

System.out.println("狗狗吃饭中！");

}

}

1. 新建企鹅类，继承Pet，并实现吃食方法eat()

/\*\*

\* 企鹅类，宠物的子类

\*/

public class Penguin extends Pet {

/\*\*

\* 实现吃食方法。

\*/

public void eat() {

System.out.println("企鹅吃饭中！");

}

}

1. 新建主人类，负责给宠物喂食，将抽象类Pet作为方法形参

/\*\*

\* 主人类

\*/

public class Master {

/\*\*

\* 主人给宠物喂食

\*/

public void feed(**Pet pet**) {

pet.eat();

}

}

1. 新建测试类Test，执行主人的喂食方法

/\*\*

\* 测试类，给宠物喂食

\*/

public class Test {

public static void main(String[] args) {

Dog dog = new Dog();

Penguin pgn = new Penguin();

Master master=new Master();

master.feed(dog);//主人给狗狗喂食

master.feed(pgn);//主人给企鹅喂食

}

}

运行测试类Test，输出结果为：

狗狗吃饭中！

企鹅吃饭中！

1. 使用父类作为方法返回值实现多态

以下继续实现一个主人领养宠物例子：

1. 在主人类中添加领养宠物的方法

/\*\*

\* 主人领养宠物

\* @param typeId 宠物编号

\* @return

\*/

public Pet getPet(int typeId){

Pet pet=null;

if(typeId==1){

pet= new Dog();

}else if(typeId==2){

pet = new Penguin();

}

return pet;

}

1. 新建测试类Test2，根据控制台选择，领养宠物

import java.util.Scanner;

public class Test2 {

public static void main(String[] args) {

System.out.println("欢迎您来到宠物店！");

System.out.print("请选择要领养的宠物类型：（1、狗狗 2、企鹅）");

Scanner input = new Scanner(System.in);

//获取选择的宠物类型

int typeId=input.nextInt();

//领养宠物

Master master=new Master();

Pet pet=master.getPet(typeId);

if(pet!=null){

System.out.println("领养成功！");

//开始喂食

master.feed(pet);

}else{

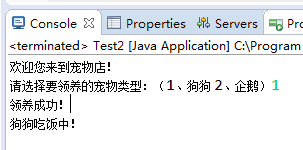
System.out.println("对不起，没有此类型的宠物，领养失败");

}

}

}

运行测试类Test2：



## 练习题

1. 在Java中，多态的实现不仅能减少编码的工作量，还能大大提高程序的可维护性及可扩展性，那么实现多态的步骤包括以下几个方面除了（ ）。

A. 子类重写父类的方法

B. 子类重载同一个方法

C. 定义方法时，把父类类型作为参数类型；调用方法时，把父类或子类的对象作为参数传入方法

D. 运行时，根据实际创建的对象类型动态决定使用哪个方法

2. 给定Java代码如下所示，则编译运行后，输出结果是（ ）。

class Parent {

public void count() {

System.out.println(10%3);

}

}

public class Child extends Parent{

public void count() {

System.out.println(10/3);

}

public static void main(String args[]) {

Parent p = new Child();

p.count();

}

}

A. 1

B. 1.0

C. 3

D. 3.3333333333333335

3. 分析如下Java代码，如果想在控制台上输出“B类的test()方法”，则在横线处应填入（ ）（多选）。

class A {

public void test() {

System.out.println("A类的test()方法");

}

}

class B extends A {

public void test() {

System.out.println("B类的test()方法");

}

public static void main(String args[]) {

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

}

}

A. A a = new B();

a.test();

B. A a = new A();

a.test();

C. B b = new A();

b.test();

D. B b = new B();

b.test();

4. 编译运行如下Java代码，输出结果是（ ）。

class Base {

public void method(){

System.out.print ("Base method");

}

}

class Child extends Base{

public void methodB(){

System.out.print ("Child methodB");

}

}

class Sample {

public static void main(String[] args) {

Base base= new Child();

base.methodB();

}

}

A. Base method

B. Child methodB

C. Base method Child MethodB

D. 编译错误

答案：1.B 2.C 3.AD 4.D

# 接口

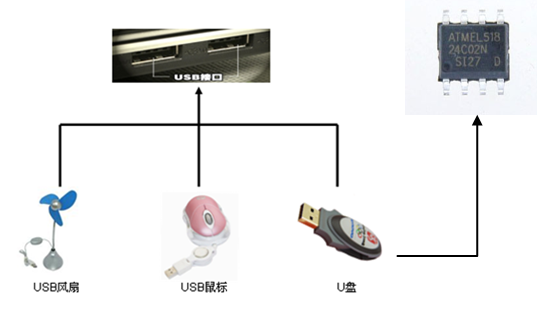
## 学习目标

* 掌握接口的意义和使用

## 为什么要用接口

计算机中的数据可以通过USB传输到外部存储器中，USB具有数据传输功能，存储器具有数据存储功能，将USB和存储器分别定义为抽象类，U盘可以同时继承USB和存储器么？

抽象类表示的是一种继承关系，一个类只能使用一次继承关系，这样限制了类的多重体现 。如何解决这个问题呢？使用接口。



接口的特点：

* Java是单继承的语言，利用接口可以模拟多继承；
* 接口是对抽象类的进一步抽象，是方法声明和常量的定义集合；
* 接口是一种行为规范，是一种“契约”；
* 接口不可以被实例化
* 实现类必须实现接口的所有方法
* 实现类可以实现多个接口
* 接口中的变量都是静态常量

为什么接口可以多继承，而类不可以？

如果有两个父类，两个父类里有一个相同的方法，那么作为子类应该怎么继承这个方法？父类1的还是父类2的？但是实现多个接口则没问题，因为不管哪个接口，调用的都是同一个实现，因为只有方法名！而且单继承的目的之一，就是降低复杂度，减少维护难度。

## 接口的使用

接口的定义格式如下：

<访问符> interface 接口名 { [访问符] <返回类型> 方法名([参数列表])； ......}

定义一个接口MultiInterface.java，代码如下：

public **interface** MultiInterface { public static final double PI = 3.1415926; public void callArea(); **default** void before(){ //仅限于JDK1.8以后可使用 System.out.println("现在开始计算："); }}

注意：在定义接口的时候，接口中的所有方法和常量自动定义为public。 接口中定义的变量默认是public static final 型，且必须赋初值。

从JDK1.8开始可以使用default关键字来定义一个默认的方法来扩展接口，default关键字修饰的默认方法可以直接调用，不用子类去实现。

## 编码实操

先声明USB接口：其中规定了要实现USB接口就必须实现接口规定实现的read( )和write( )这两个方法。

interface USB {

void read();

void write();

}

然后在写一个U盘类UDisk和一个键盘类Keyboard，这两个类都去实现USB接口。（实现其中的方法）

class UDisk **implements** USB {

    @Override

    public void read() {

        System.out.println("U盘正在通过USB功能读取数据");

    }

    @Override

    public void write() {

        System.out.println("U盘正在通过USB功能写入数据");

    }

}

class Keyboard implements USB {

    @Override

    public void read() {

        System.out.println("键盘正在通过USB功能读取数据");

    }

    @Override

    public void write() {

        System.out.println("键盘正在通过USB功能写入数据");

    }

}

测试类：

public class Main {

    public static void main(String[] args) {

        //生成一个实现可USB接口（标准）的U盘对象

       UDisk udisk = new UDisk();

        //调用U盘的read( )方法读取数据

        udisk.read();

        //调用U盘的write( )方法写入数据

        udisk.write();

        //生成一个实现可USB接口（标准）的键盘对象

       Keyboard keyboard = new Keyboard();

        //调用键盘的read( )方法读取数据

       keyboard.read();

        //调用键盘的write( )方法写入数据

       keyboard.write();

    }

}

## 练习题

1. 下面关于Java接口的说法错误的是（ ）。

A. 一个Java接口是一些方法特征的集合，但没有方法的实现

B. Java接口中定义的方法在不同的地方被实现，可以具有完全不同的行为

C. Java接口中可以声明私有成员

D. Java接口不能被实例化

2. 在Java接口中定义常量，下面语法错误的是（ ）。

A. static int MALE = 1;

B. final int MALE = 1;

C. int MALE = 1;

D. private int MALE = 1;

3. 在Java接口中，下列选项中属于有效的方法声明是（ ）（多选）。

A. public void aMethod();

B. void aMethod();

C. void aMethod(){}

D. private void aMethod();

4. 给定如下Java程序代码，在横线处加入（ ）语句，可以使这段代码编译通过（多选）。

interface Parent{

public int count(int i);

}

public class Test implements Parent {

public int count(int i){

return i % 9;

}

public static void main(String[] args){

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

int i = p.count(20);

}

}

A. Test p = new Test();

B. Parent p = new Test();

C. Parent p = new Parent();

D. Test p = new Parent();

答案：1.C 2.D 3.AB 4.AB

# 异常

## 学习目标

* 掌握异常的处理方法

## 什么是异常

异常是程序中的一些错误，但并不是所有的错误都是异常，并且错误有时候是可以避免的。

比如说，你的代码少了一个分号，那么运行出来结果是提示是错误 java.lang.Error；如果你用System.out.println(11/0)，那么你是因为你用0做了除数，会抛出 java.lang.ArithmeticException 的异常。

异常发生的原因有很多，通常包含以下几大类：

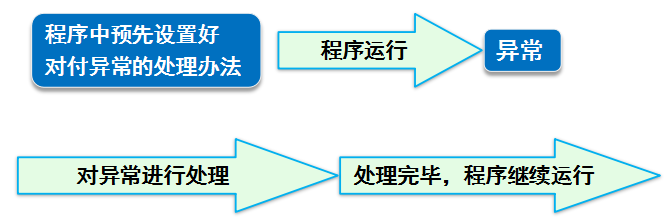
* 用户输入了非法数据。
* 要打开的文件不存在。
* 网络通信时连接中断，或者JVM内存溢出。

要理解Java异常处理是如何工作的，你需要掌握以下三种类型的异常：

* 检查性异常：最具代表的检查性异常是用户错误或问题引起的异常，这是程序员无法预见的。例如要打开一个不存在文件时，一个异常就发生了，这些异常在编译时不能被简单地忽略。
* 运行时异常： 运行时异常是可能被程序员避免的异常。与检查性异常相反，运行时异常可以在编译时被忽略。
* 错误： 错误不是异常，而是脱离程序员控制的问题。错误在代码中通常被忽略。例如，当栈溢出时，一个错误就发生了，它们在编译也检查不到的。

## 异常处理

异常处理的流程如下：



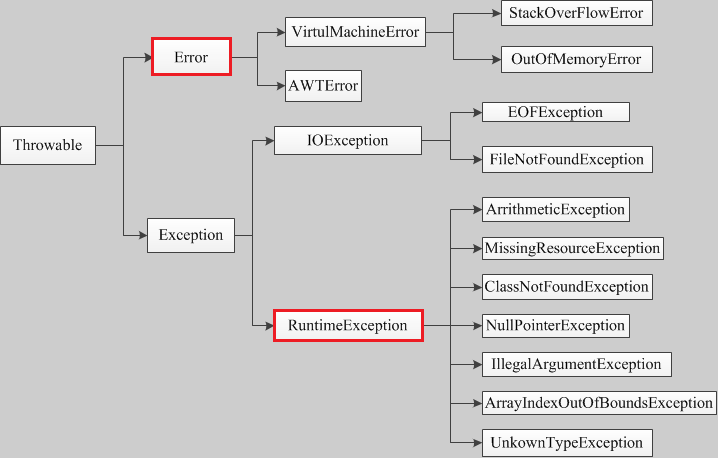
Java标准库内建了一些通用的异常，这些类以Throwable为顶层父类。

Throwable又派生出Error类和Exception类。

错误：Error类以及他的子类的实例，代表了JVM本身的错误。错误不能被程序员通过代码处理，Error很少出现。因此，程序员应该关注Exception为父类的分支下的各种异常类。

异常：Exception以及他的子类，代表程序运行时发送的各种不期望发生的事件。可以被Java异常处理机制使用，是异常处理的核心。

Java异常的分类和类结构图：



Java的异常处理是通过5个关键字来实现的：try、catch、 finally、throw、throws。

1. 捕获异常

try：执行可能产生异常的代码。

catch：捕获异常。

finally：无论是否发生异常，代码总能执行。

public void method(){try { // 代码段 1 // 产生异常的代码段 2 // 代码段 3} catch (异常类型 ex) { // 对异常进行处理的代码段4}// 代码段5}

|  |  |
| --- | --- |
| **异 常 类 型** | **说 明** |
| Exception | 异常层次结构的父类 |
| ArithmeticException | 算术错误情形，如以零作除数 |
| ArrayIndexOutOfBoundsException | 数组下标越界 |
| NullPointerException | 尝试访问 null 对象成员 |
| ClassNotFoundException | 不能加载所需的类 |
| IllegalArgumentException | 方法接收到非法参数 |
| ClassCastException | 对象强制类型转换出错 |
| NumberFormatException | 数字格式转换异常，如把"abc"转换成数字 |

异常的处理示例：

import java.util.Scanner;

/\*\*

\* 使用try-catch-finally进行异常处理。

\*/

public class Test4 {

public static void main(String[] args) {

Scanner in = new Scanner(System.in);

System.out.print("请输入被除数:");

try {

int num1 = in.nextInt();

System.out.print("请输入除数:");

int num2 = in.nextInt();

System.out.println(String.format("%d / %d = %d",

num1, num2, num1/ num2));

} catch (Exception e) {

System.err.println("出现错误:被除数和除数必须是整数," +

"除数不能为零。");

System.out.println(e.getMessage());

//System.exit(1); // finally语句块不执行的唯一情况

} finally {

System.out.println("感谢使用本程序!");

}

}

}

多重异常示例：

import java.util.Scanner;

import java.util.InputMismatchException;

/\*\*

\* 多重catch块。

\*/

public class Test7 {

public static void main(String[] args) {

Scanner in = new Scanner(System.in);

System.out.print("请输入被除数:");

try {

int num1 = in.nextInt();

System.out.print("请输入除数:");

int num2 = in.nextInt();

System.out.println(String.format("%d / %d = %d",

num1, num2, num1/ num2));

} catch (InputMismatchException e) {

System.err.println("被除数和除数必须是整数。");

} catch (ArithmeticException e) {

System.err.println("除数不能为零。");

} catch (Exception e) {

System.err.println("其他未知异常。");

} finally {

System.out.println("感谢使用本程序!");

}

}

}

多重异常的执行规则：

* 排列catch 语句的顺序：先子类后父类
* 发生异常时按顺序逐个匹配
* 只执行第一个与异常类型匹配的catch语句

1. 声明异常

throws：声明方法可能要抛出的各种异常。

如果一个方法没有捕获到一个检查性异常，那么该方法必须使用 throws 关键字来声明。throws 关键字放在方法签名的尾部。

1. 抛出异常

throw：手动抛出异常。

例子：

public class Person {

private String name = ""; // 姓名

private int age = 0; // 年龄

private String sex = "男"; // 性别

public void setSex(String sex) throws Exception {

if ("男".equals(sex) || "女".equals(sex))

this.sex = sex;

else {

throw new Exception(“性别必须是\"男\"或者\"女\"！");

}

}

}

# 集合

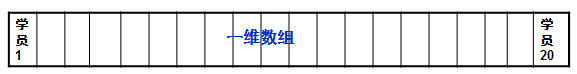
## 学习目标

* 掌握集合的意义
* 掌握常用集合的使用
* 掌握泛型集合的意义和使用

## 为什么要使用集合

假设需要存储一个班20个学生的信息，如何实现？

使用数组：



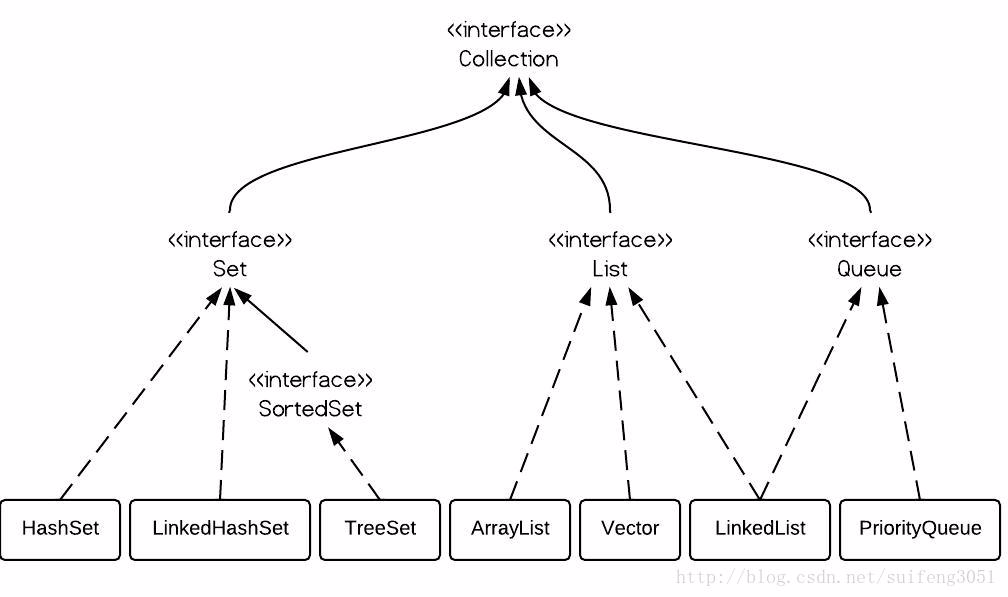
假设来了一个新同学，如何把新同学的信息添加到数组里？

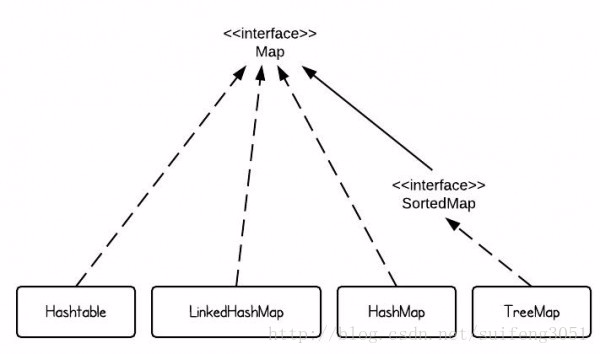
假设需要存储所有学生的姓名和成绩，并一一对应，如何存储在一个数组里？

上述问题，使用数组实现比较困难。可以使用Java的集合框架。

## 集合基础

Java集合框架提供了一套性能优良、使用方便的接口和类，它们位于java.util包中。





Java集合按照其存储结构可以分为两大类，即单列集合Collection和双列集合Map。

Collection是单列集合类的根接口，其有两个重要子接口 List 和Set。

* List接口的主要类有 ArrayList 和 LinkedList。
* Set接口的主要类有 HashSet和TreeSet。
* Map接口的主要实现类有HashMap 和 TreeMap

各接口的特点：

* List 接口存储一组不唯一，有序（插入顺序）的对象
* Set 接口存储一组唯一，无序的对象
* Map接口存储一组键值对象，提供key到value的映射

ArrayList实现了长度可变的数组，在内存中分配连续的空间。遍历元素和随机访问元素的效率比较高。

Snap4

LinkedList采用链表存储方式。插入、删除元素时效率比较高。

遍历ArrayList集合：

import java.util.\*;

public class Test{

public static void main(String[] args) {

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

list.add("Hello");

list.add("World");

list.add("HAHAHAHA");

//第一种遍历方法使用foreach遍历List

for (String str : list) {//也可以改写for(int i=0;i<list.size();i++)这种形式

System.out.println(str);

}

//第二种遍历，把链表变为数组相关的内容进行遍历

String[] strArray=new String[list.size()];

list.toArray(strArray);

//这里也可以改写为 foreach(String str:strArray)这种形式

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

System.out.println(strArray[i]);

}

//第三种遍历 使用迭代器进行相关遍历

Iterator<String> ite=list.iterator();

while(ite.hasNext())//判断下一个元素之后有值

{

System.out.println(ite.next());

}

}

}

遍历 Map：

import java.util.\*;

public class Test{

public static void main(String[] args) {

Map<String, String> map = new HashMap<String, String>();

map.put("1", "value1");

map.put("2", "value2");

map.put("3", "value3");

//第一种：普遍使用，二次取值

System.out.println("通过Map.keySet遍历key和value：");

for (String key : map.keySet()) {

System.out.println("key= "+ key + " and value= " + map.get(key));

}

//第二种

System.out.println("通过Map.entrySet使用iterator遍历key和value：");

Iterator<Map.Entry<String, String>> it = map.entrySet().iterator();

while (it.hasNext()) {

Map.Entry<String, String> entry = it.next();

System.out.println("key= " + entry.getKey() + " and value= " + entry.getValue());

}

//第三种：推荐，尤其是容量大时

System.out.println("通过Map.entrySet遍历key和value");

for (Map.Entry<String, String> entry : map.entrySet()) {

System.out.println("key= " + entry.getKey() + " and value= " + entry.getValue());

}

//第四种

System.out.println("通过Map.values()遍历所有的value，但不能遍历key");

for (String v : map.values()) {

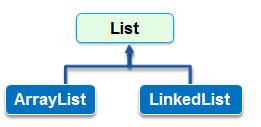
System.out.println("value= " + v);

}

}

}

## List



List集合代表一个有序集合，集合中每个元素都有其对应的顺序索引。List集合允许使用重复元素，可以通过索引来访问指定位置的集合元素。

ArrayList实现了长度可变的数组，在内存中分配连续的空间。遍历元素和随机访问元素的效率比较高。

LinkedList采用链表存储方式。插入、删除元素时效率比较高。

### ArrayList

使用ArrayList存储元素：

//创建四个狗狗对象 ...

List dogs = new ArrayList();

dogs.add(ououDog);

dogs.add(yayaDog);

dogs.add(meimeiDog);

dogs.add(2, feifeiDog); // 添加feifeiDog到指定位置

System.out.println("共计有" + dogs.size() + "条狗狗。");

System.out.println("分别是：");

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

Dog dog = (Dog) dogs.get(i);

… …

}

使用ArrayList移除、判断元素：

import java.util.ArrayList;

import java.util.List;

/\*\*

\* 测试ArryList的remove()、contains()方法

\*/

public class Test2 {

public static void main(String[] args) {

// 1、创建多个狗狗对象

Dog ououDog = new Dog("欧欧", "雪娜瑞");

Dog yayaDog = new Dog("亚亚", "拉布拉多");

Dog meimeiDog = new Dog("美美", "雪娜瑞");

Dog feifeiDog = new Dog("菲菲", "拉布拉多");

// 2、创建ArrayList集合对象并把多个狗狗对象放入其中

List dogs = new ArrayList();

dogs.add(ououDog);

dogs.add(yayaDog);

dogs.add(meimeiDog);

dogs.add(2, feifeiDog);

// 3、输出删除前集合中狗狗的数量

System.out.println("删除之前共计有" + dogs.size() + "条狗狗。");

// 4、删除集合中第一个狗狗和feifeiDog狗狗

dogs.remove(0);

dogs.remove(feifeiDog);

// 5、显示删除后集合中各条狗狗信息

System.out.println("\n删除之后还有" + dogs.size() + "条狗狗。");

System.out.println("分别是:");

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

Dog dog = (Dog) dogs.get(i);

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

}

//6、判断集合中是否包含指定狗狗信息

if(dogs.contains(meimeiDog))

System.out.println("\n集合中包含美美的信息");

else

System.out.println("\n集合中不包含美美的信息");

}

}

List接口常用方法：

|  |  |
| --- | --- |
| **方法名** | **说 明** |
| boolean add(Object o) | 在列表的末尾顺序添加元素，起始索引位置从0开始 |
| void add(int index,Object o) | 在指定的索引位置添加元素。索引位置必须介于0和列表中元素个数之间 |
| int size() | 返回列表中的元素个数 |
| Object get(int index) | 返回指定索引位置处的元素。取出的元素是Object类型，使用前需要进行强制类型转换 |
| boolean contains(Object o) | 判断列表中是否存在指定元素 |
| boolean remove(Object o) | 从列表中删除元素 |
| Object remove(int index) | 从列表中删除指定位置元素，起始索引位置从0开始 |

### LinkedList

使用LinkedList存储元素：

// 创建多个狗狗对象...

LinkedList dogs = new LinkedList();

dogs.add(ououDog);

dogs.add(yayaDog);

dogs.addLast(meimeiDog);

dogs.addFirst(feifeiDog);

Dog dogFirst= (Dog)dogs.getFirst();//获取集合第一条信息

System.out.println("第一条狗狗昵称是"+dogFirst.getName() );

Dog dogLast= (Dog)dogs.getLast();//获取集合最后一条信息

System.out.println("最后一条狗狗昵称是"+dogLast.getName());

dogs.removeFirst();//删除集合第一条信息

dogs.removeLast(); //删除集合最后一条信息

LinkedList的特殊方法：

|  |  |
| --- | --- |
| **方法名** | **说 明** |
| void addFirst(Object o) | 在列表的首部添加元素 |
| void addLast(Object o) | 在列表的末尾添加元素 |
| Object getFirst() | 返回列表中的第一个元素 |
| Object getLast() | 返回列表中的最后一个元素 |
| Object removeFirst() | 删除并返回列表中的第一个元素 |
| Object removeLast() | 删除并返回列表中的最后一个元素 |

### Vector和ArrayList的异同

实现原理、功能相同，可以互用。

主要区别：

* Vector线程安全，ArrayList重速度轻安全，线程非安全
* 长度需增长时，Vector默认增长一倍，ArrayList增长50%

## 迭代器Iterator

如何遍历一个Map集合？

方法一：使用迭代器Iterator

Set keys=dogMap.keySet(); //取出所有key的集合

Iterator it=keys.iterator(); //获取Iterator对象

while(it.hasNext()){

String key=(String)it.next(); //取出key

Dog dog=(Dog)dogMap.get(key); //根据key取出对应的值

System.out.println(key+"\t"+dog.getStrain());

}

方法二：使用for循环。

例子：使用HashMap存储和处理信息。

企鹅类：

/\*\*

\* 宠物企鹅

\*/

public class Penguin {

String name = "无名氏"; // 昵称

int health = 100; // 健康值

int love = 0; // 亲密度

String sex = "Q仔"; // 性别

/\*\*

\* 两个参数构造方法。

\*/

public Penguin(String name, String sex) {

this.name = name;

this.sex = sex;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getSex() {

return sex;

}

public void setSex(String sex) {

this.sex = sex;

}

}

测试类：

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Iterator;

import java.util.LinkedList;

import java.util.List;

import java.util.Map;

import java.util.Set;

/\*\*

\* 使用HashMap存储和处理企鹅信息。

\*/

public class Test3 {

public static void main(String[] args) {

// 1、创建多个企鹅对象

Penguin ououPenguin = new Penguin("欧欧", "Q仔");

Penguin yayaPenguin = new Penguin("亚亚", "Q妹");

Penguin meimeiPenguin = new Penguin("美美", "Q妹");

Penguin feifeiPenguin = new Penguin("菲菲", "Q妹");

/\* 2、创建HashMap集合对象并把多个企鹅对象放入其中\*/

Map penguinMap=new HashMap();

penguinMap.put(ououPenguin.getName(),ououPenguin);

penguinMap.put(yayaPenguin.getName(),yayaPenguin);

penguinMap.put(meimeiPenguin.getName(),meimeiPenguin);

penguinMap.put(feifeiPenguin.getName(),feifeiPenguin);

/\*3、通过迭代器依次输出集合中所有狗狗的信息\*/

System.out.println("使用Iterator遍历，所有企鹅的昵称和品种分别是：");

Set keys=penguinMap.keySet();//取出所有key的集合

Iterator it=keys.iterator();//获取Iterator对象

while(it.hasNext()){

String key=(String)it.next(); //取出key

Penguin penguin=(Penguin)penguinMap.get(key); //根据key取出对应的值

System.out.println(key+"\t"+penguin.getSex());

}

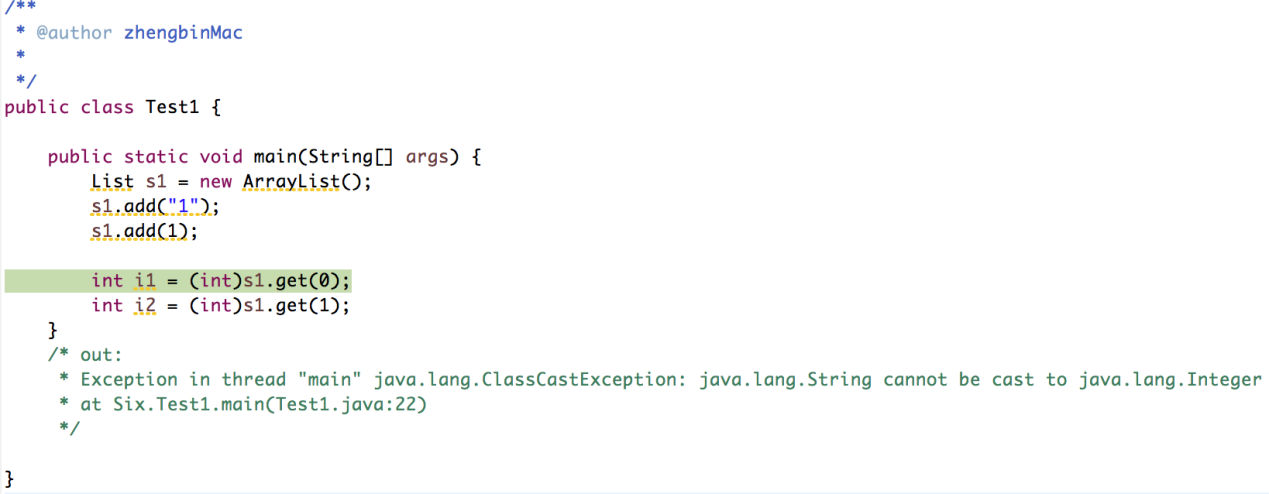
}

}

## 泛型集合

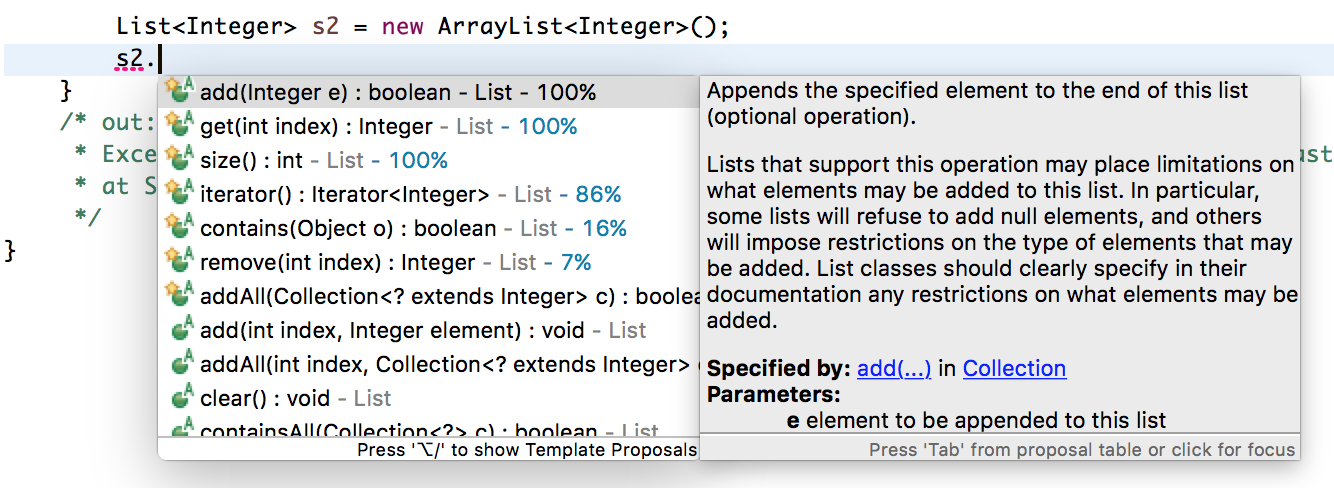
### 为什么要用泛型

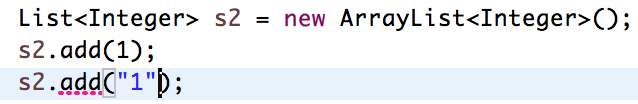
举个例子：



上述代码在编译期没有问题，但在运行期，将会报错。因为List的add方法的参数为Object，所以如果程序员在不知道List类型时，通过强转获值，这时就会报错了。

使用泛型，将会解决这个问题。





如上两个截图所示，当声明List所能装载的类型后，List的add方法就只能装载指定的类型，不然在编译期便会报错。而且在获值时，不再需要强制转换。

### 使用泛型的优点

Java语言引入泛型的好处是安全简单。泛型的好处是在编译的时候检查类型安全，并且所有的强制转换都是自动和隐式的，提高代码的重用率。

* **类型安全**。泛型的主要目标是提高 Java 程序的类型安全。通过知道使用泛型定义的变量的类型限制，编译器可以在一个高得多的程度上验证类型假设。没有泛型，这些假设就只存在于程序员的头脑中（或者如果幸运的话，还存在于代码注释中）。
* **消除强制类型转换**。泛型的一个附带好处是，消除源代码中的许多强制类型转换。这使得代码更加可读，并且减少了出错机会。
* **潜在的性能优势**。泛型为较大的优化带来可能。在泛型的初始实现中，编译器将强制类型转换（没有泛型的话，程序员会指定这些强制类型转换）插入生成的字节码中。但是更多类型信息可用于编译器这一事实，为未来版本的 JVM 的优化带来可能。由于泛型的实现方式，支持泛型（几乎）不需要 JVM 或类文件更改。所有工作都在编译器中完成，编译器生成类似于没有泛型（和强制类型转换）时所写的代码，只是更能确保类型安全而已。

### 泛型的使用

泛型使用示例一：

/\* 1、创建多个狗狗对象\*/

Dog ououDog = new Dog("欧欧", "雪娜瑞");

Dog yayaDog = new Dog("亚亚", "拉布拉多");

Dog meimeiDog = new Dog("美美", "雪娜瑞");

Dog feifeiDog = new Dog("菲菲", "拉布拉多");

/\* 2、创建ArrayList集合对象并把多个狗狗对象放入其中\*/

List<Dog> dogs = new ArrayList<Dog>();//标记元素类型

dogs.add(ououDog);

dogs.add(yayaDog);

dogs.add(meimeiDog);

dogs.add(2, feifeiDog); // 添加feifeiDog到指定位置

//dogs.add("hello"); //出现编译错误,元素类型不是Dog。

/\* 3、 显示第三个元素的信息\*/

Dog dog3 = dogs.get(2); //无需类型强制转换

System.out.println("第三个狗狗的信息如下:");

System.out.println(dog3.getName() + "\t" + dog3.getStrain());

/\*4、使用foreach语句遍历dogs对象\*/

System.out.println("\n所有狗狗的信息如下：");

for(Dog dog:dogs){//无需类型强制转换

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

}

泛型使用示例二：

Dog ououDog = new Dog("欧欧", "雪娜瑞");

Dog yayaDog = new Dog("亚亚", "拉布拉多");

Dog meimeiDog = new Dog("美美", "雪娜瑞");

Dog feifeiDog = new Dog("菲菲", "拉布拉多");

/\* 2、创建Map集合对象并把多个狗狗对象放入其中\*/

Map<String,Dog> dogMap=new HashMap<String,Dog>();

dogMap.put(ououDog.getName(),ououDog);

dogMap.put(yayaDog.getName(),yayaDog);

dogMap.put(meimeiDog.getName(),meimeiDog);

dogMap.put(feifeiDog.getName(),feifeiDog);

/\*3、通过迭代器依次输出集合中所有狗狗的信息\*/

System.out.println("使用Iterator遍历，所有狗狗的昵称和品种分别是：");

Set<String> keys=dogMap.keySet();//取出所有key的集合

Iterator<String> it=keys.iterator();//获取Iterator对象

while(it.hasNext()){

String key=it.next(); //取出key

Dog dog=dogMap.get(key); //根据key取出对应的值

System.out.println(key+"\t"+dog.getStrain());

}

/\*//使用foreach语句输出集合中所有狗狗的信息

for(String key:keys){

Dog dog=dogMap.get(key); //根据key取出对应的值

System.out.println(key+"\t"+dog.getStrain());

}\*/

## Map

Map接口专门处理键值映射数据的存储，可以根据键实现对值的操作，最常用的实现类是HashMap。

使用HashMap存储数据：

Map countries = new HashMap();

countries.put("CN", "中华人民共和国");

countries.put("RU", "俄罗斯联邦");

countries.put("FR", "法兰西共和国");

countries.put("US", "美利坚合众国");

String country = (String) countries.get("CN");//获取指定元素的值

//… …

System.out.println("Map中共有"+countries.size() +"组数据");//获取元素个数

countries.remove("FR");//删除指定元素

System.out.println("Map中包含FR的key吗？" +

countries.containsKey("FR"));

System.out.println( countries.keySet() ) ;//显示键集

System.out.println( countries.values() );//显示值集

System.out.println( countries );//显示键值对集

Map接口常用方法：

|  |  |
| --- | --- |
| **方法名** | **说 明** |
| Object put(Object key, Object val) | 以“键-值对”的方式进行存储 |
| Object get (Object key) | 根据键返回相关联的值，如果不存在指定的键，返回null |
| Object remove (Object key) | 删除由指定的键映射的“键-值对” |
| int size() | 返回元素个数 |
| Set keySet () | 返回键的集合 |
| Collection values () | 返回值的集合 |
| boolean containsKey (Object key) | 如果存在由指定的键映射的“键-值对”，返回true |

Hashtable和HashMap的异同：

实现原理、功能相同，可以互用。

主要区别：

* Hashtable继承Dictionary类，HashMap实现Map接口
* Hashtable线程安全，HashMap线程非安全
* Hashtable不允许null值，HashMap允许null值

注意：开发过程中，最好使用ArrayList和HashMap。

## 练习题

1. 下列选项中关于Java中ArrayList、LinkedList、HashMap三个类的描述错误的是（ ）。

A. ArrayList和LinkedList均实现了List接口

B. ArrayList的访问速度比LinkedList快

C. 添加和删除元素时，ArrayList的表现更佳

D. HashMap实现Map接口，它允许任何类型的键和值对象，允许将null用作键或值

2. 关于Java中的集合实现类，下列说法描述错误的是（ ）。

A. HashMap是Map接口的实现类

B. ArrayList对象是长度可变的对象引用数组

C. 集合框架都包含三大块内容：对外的接口、接口的实现和对集合运算的算法

D. Set中存储一组不允许重复、有序的对象

3. 对于Java中的集合类java.util.ArrayList，以下说法正确的是（ ）。

A. 这个集合中的元素是无序的

B. 可以通过键来获取这个集合中的元素

C. 在对这个集合的元素执行遍历操作时，效率比较高

D. 可以通过addFirst()方法，在列表的首部插入元素

4. 给定如下Java代码，其中第（ ）行代码有错误。

import java.util.\*;

public class MapTest {

public static void main(String[] args) {

List pList = new ArrayList();

pList.add("A"); //第五行

pList.add("B"); //第六行

for (int i = 0; i < pList.size(); i++) { //第七行

String str = pList.get(i); //第八行

System.out.print(str);

}

}

}

A. 五

B. 六

C. 七

D. 八

5. 给定如下Java代码，可以填入横线处的代码是（ ）。

import java.util.\*;

public class Test {

public static void main(String[] args) {

\_\_\_\_\_\_\_\_

list.addLast("001");

}

}

A. List list = new ArrayList();

B. List list = new List();

C. ArrayList list = new ArrayList();

D. LinkedList list = new LinkedList();

6. 在Java中，（ ）类的对象可被用作堆栈或者队列。

A. LinkedList

B. ArrayList

C. Collection

D. HashMap

7. 在Java中，关于HashMap类的描述，以下说法错误的是（ ）。

A. HashMap使用键/值的形式保存数据

B. HashMap能够保证其中元素的顺序

C. HashMap的存储方式是哈希表，查询指定元素效率高

D. HashMap通过get( key)获取对应的值

8. 阅读下面的Java代码，运行结果描述正确的是（ ）。

import java.util.\*;

public class TestMap{

public static void main(String[] args){

Map map=new HashMap( );

map.put(“first”,”football”);

map.put(“first”,”basketball”);

System.out.print(map.get(“first”));

}

}

A. 编译时发生错误

B. 编译通过，运行时发生错误

C. 正确运行，显示basketball

D. 正确运行，显示football

9. 在Java中，集合框架 API定义了一组用于批量操作数据的接口和类，它们包括在（ ）包中。

A. java.lang

B. java.sql

C. java.util

D. java.math

答案：1.C 2.D 3.C 4.D 5.D 6.A 7.B 8.C 9.C

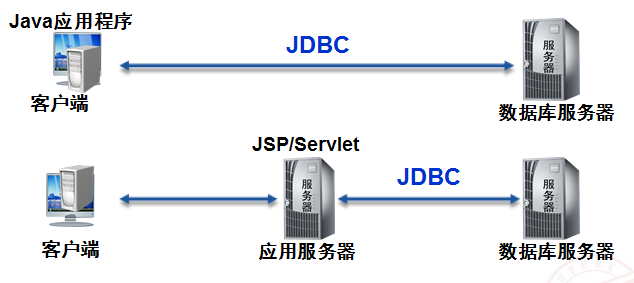
# JDBC

## 学习目标

* 掌握JDBC的意义和使用
* 掌握Java的分层开发

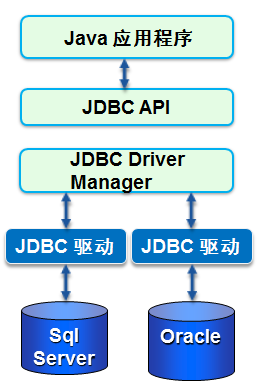
## 为什么需要JDBC

JDBC是Java数据库连接技术的简称，提供连接各种常用数据库的能力。



## JDBC的使用

### 工作原理

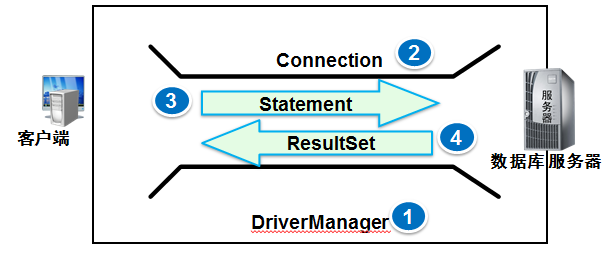


JDBC API：提供供程序员调用的接口与类，集成在java.sql和javax.sql包中，如：

* DriverManager类
* Connection接口
* Statement接口
* ResultSet接口

DriverManager：依据数据库的不同，管理各种不同的JDBC驱动。

JDBC 驱动：由数据库厂商提供，负责连接各种不同的数据库。



JDBC API主要功能：与数据库建立连接、执行SQL 语句、处理结果。

* DriverManager ：依据数据库的不同，管理JDBC驱动。
* Connection ：负责连接数据库并担任传送数据的任务。
* Statement ：由 Connection 产生、负责执行SQL语句。
* ResultSet：负责保存Statement执行后所产生的查询结果。

### 使用流程

JDBC的使用流程如下：

try {

//1.加载JDBC驱动

Class.forName(JDBC驱动类);

}

… …

try {

//2.与数据库建立连接

Connection con=DriverManager.getConnection(URL,数据库用户名,密码);

//3.发送SQL语句并得到结果

Statement stmt = con.createStatement();

ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Table1");

//4.处理返回结果

while (rs.next()) {

int x = rs.getInt("a");

String s = rs.getString("b");

float f = rs.getFloat("c");

}

//5.释放资源

rs.close();

stmt.close();

con.close();

}

数据库访问基类：

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

/\*\*

\* 数据库访问基类

\*/

public class BaseDao {

//数据库驱动

private String driver="com.mysql.jdbc.Driver";

//数据库连接地址

private String url="jdbc:mysql://localhost:3306/orcl";

//数据库用户名

private String name="root";

//数据库密码

private String password="";

public Connection conn;//数据库连接对象

public PreparedStatement ps;//数据库操作对象

public ResultSet rs;//存放结果数据

/\*\*

\* 获取数据库连接

\*/

public Connection getConn(){

try {

Class.forName(driver);

this.conn=DriverManager.getConnection(url,name,password);

} catch (Exception e) {

e.printStackTrace();

}

return conn;

}

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

//1. 获取连接

Connection conn=new BaseDao().getConn();

//2. 执行查询

String sql="select \* from k\_user";

PreparedStatement ps=conn.prepareStatement(sql);

ResultSet rs=ps.executeQuery();

//3. 循环输出数据

while(rs.next()){

System.out.println("用户名是："+rs.getString("username"));

}

}

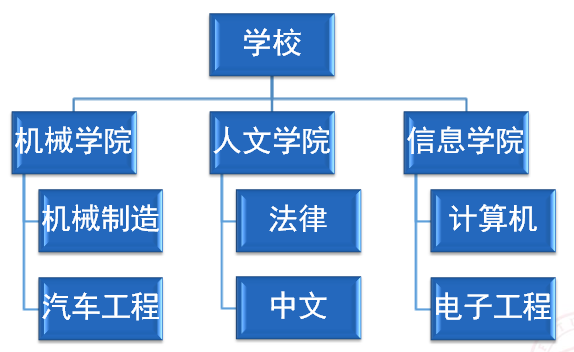
}

## 分层开发

### 为什么要使用分层开发

分层开发是一种化大为小，分而治之的软件开发方法。

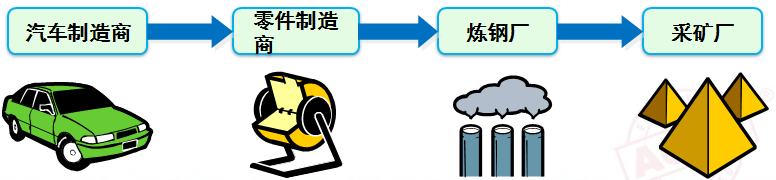
例如，一所学校被分为若干学院，各学院下面再分若干专业，这样的分层更易于管理。



一道好的菜放到客户的餐桌上需要经过以下几层：



制造一辆汽车需要经过以下几层：



### 分层开发的特点及好处

分层的特点：

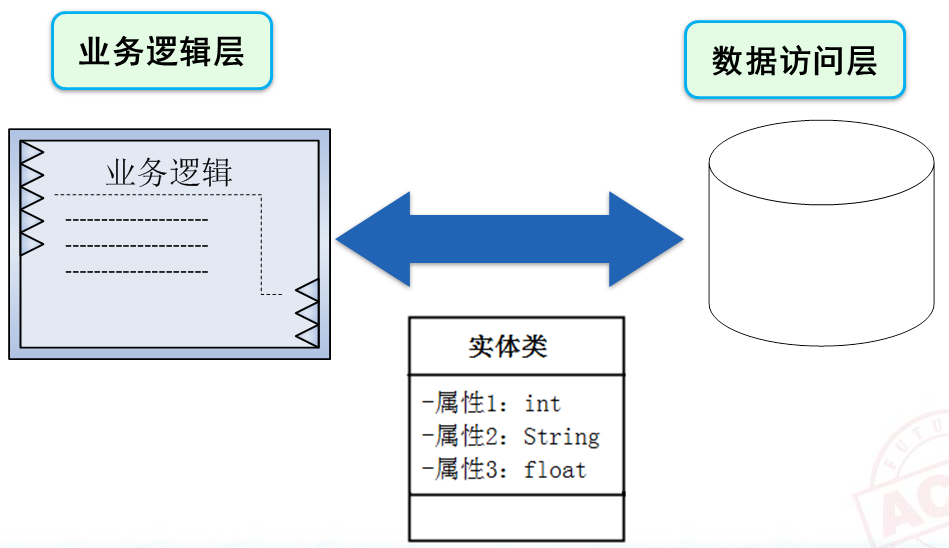
* 每一层都有自己的职责。
* 上一层不用关心下一层的实现细节，上一层通过下一层提供的对外接口来使用其功能。
* 上一层调用下一层的功能，下一层不能调用上一层功能。

分层开发的好处：

* 各层专注于自己功能的实现，便于提高质量。
* 便于分工协作，提高开发效率。
* 便于代码复用。
* 便于程序扩展。

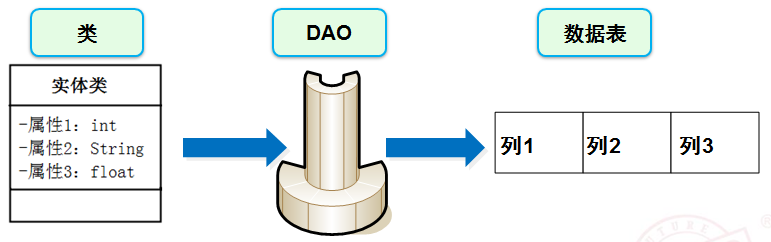
### Java中的分层开发

不同层之间通过实体类传输数据：



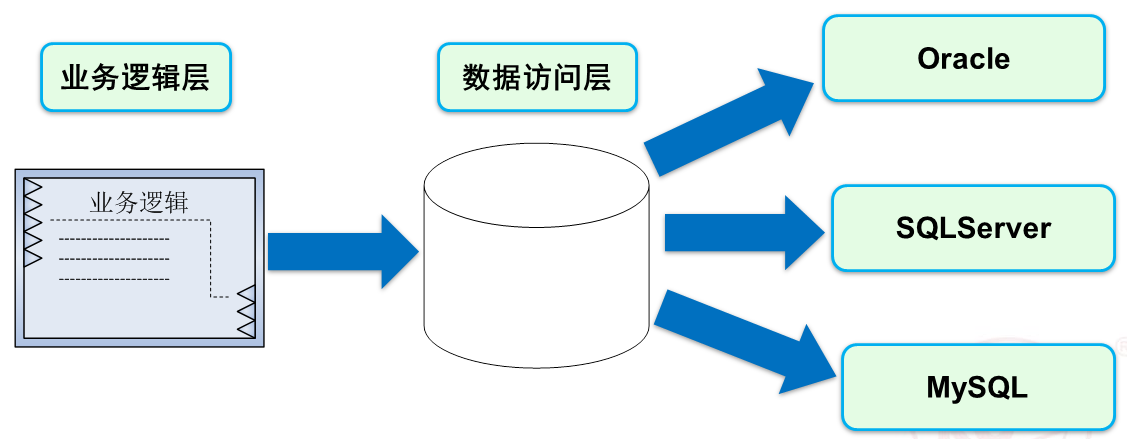
DAO，Data Access Object(数据存取对象)，位于业务逻辑和持久化数据之间，实现对持久化数据的访问。

DAO起着转换器的作用，把实体类转换为数据库中的记录：



DAO模式的作用：

* 隔离业务逻辑代码和数据访问代码
* 隔离不同数据库的实现



### 编码实操

使用JDBC分层开发，完成完成用户登录、用户数据增删改查功能。

1. 数据访问层代码

基类BaseDao的源码：

import java.sql.\*;

/\*\*

\* Dao层的基类，获取数据库连接

\*/

public class BaseDao {

private String driver = "com.mysql.jdbc.Driver";

private String url = "jdbc:mysql://localhost:3306/orcl";

private String name = "root";

private String pass = "123456";

public Connection conn;

public PreparedStatement ps;

public ResultSet rs;

/\*\*

\* 得到数据库连接

\*/

public Connection getConn() {

try {

Class.forName(driver);

this.conn = DriverManager.getConnection(url, name, pass);

} catch (Exception e) {

e.printStackTrace();

}

return conn;

}

/\*\* 测试连接 \*/

public static void main(String[] args) {

BaseDao bd = new BaseDao();

bd.getConn();

}

/\*\* 关闭资源 \*\*/

public void closeAll() {

try {

if (rs != null) {

rs.close();

rs = null;

}

if (ps != null) {

ps.close();

ps = null;

}

if (conn != null) {

conn.close();

conn = null;

}

} catch (SQLException e) {

e.printStackTrace();

}

}

/\*\* 增删改的公用方法 \*\*/

public int execSql(String sql, String param[]) {

int ret = 0;

this.conn = this.getConn();

try {

this.ps = conn.prepareStatement(sql);

if (param != null) {

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

ps.setString(i + 1, param[i]);

}

}

ret = ps.executeUpdate();

} catch (SQLException e) {

e.printStackTrace();

} finally {

this.closeAll();

}

return ret;

}

}

UserDao的源码：

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.List;

import com.stock.entity.User;

/\*\*

\* 与用户相关的数据库操作

\*/

public class UserDao extends BaseDao{

/\*\*

\* 用户登录

\* @param userNo 用户名(编号)

\* @param pwd 密码

\* @return 用户对象

\*/

public User login(String userNo, String pwd) {

String sql = "select \* from k\_user where userno='"+userNo+"' And pwd='"+pwd+"'";

this.conn=this.getConn();

User user=null;

try {

this.ps=conn.prepareStatement(sql);

ResultSet rs = ps.executeQuery();

while (rs.next()) {

user = new User();

user.setId(rs.getInt("id"));

user.setUsername(rs.getString("username"));

user.setUserno(rs.getString("userno"));

user.setPwd(rs.getString("pwd"));

}

} catch (SQLException e) {

e.printStackTrace();

}

this.closeAll();

return user;

}

/\*\*

\* 查询用户列表

\* @return 用户集合

\*/

public List<User> findUserList() {

String sql = "select \* from k\_user order by id asc";

this.conn=this.getConn();

List<User> list = new ArrayList<>();

try {

this.ps=conn.prepareStatement(sql);

ResultSet rs = ps.executeQuery();

while (rs.next()) {

User user = new User();

user.setId(rs.getInt("id"));

user.setUsername(rs.getString("username"));

user.setUserno(rs.getString("userno"));

user.setPwd(rs.getString("pwd"));

user.setAge(rs.getInt("age"));

user.setSex(rs.getString("sex"));

user.setPosition(rs.getString("position"));

list.add(user);

}

} catch (SQLException e) {

e.printStackTrace();

}

return list;

}

/\*\*

\* 添加用户

\* @param user 用户对象

\* @return 是否添加成功

\*/

public boolean addUser(User user) {

String sql = "insert into k\_user(username,userno,pwd) values('"+user.getUsername()+"','"+user.getUserno()+"','"+user.getPwd()+"')";

this.conn=this.getConn();

try {

this.ps=conn.prepareStatement(sql);

int count=ps.executeUpdate();

if(count>0){

return true;

}

} catch (SQLException e) {

e.printStackTrace();

}

return false;

}

/\*\*

\* 删除用户

\* @param id 用户ID

\* @return 是否删除成功

\*/

public boolean deleteUser(String id) {

String sql = "delete from k\_user where id="+id;

this.conn=this.getConn();

try {

this.ps=conn.prepareStatement(sql);

int count=ps.executeUpdate();

if(count>0){

return true;

}

} catch (SQLException e) {

e.printStackTrace();

}

return false;

}

}

User实体类：

public class User {

private int id;//用户ID

private String userno;//用户编号

private String username;//用户姓名

private String pwd;//密码

private String sex;//性别

private int age;//年龄

private String position;//职位

public String getSex() {

return sex;

}

public void setSex(String sex) {

this.sex = sex;

}

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

public String getPosition() {

return position;

}

public void setPosition(String position) {

this.position = position;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getUserno() {

return userno;

}

public void setUserno(String userno) {

this.userno = userno;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getPwd() {

return pwd;

}

public void setPwd(String pwd) {

this.pwd = pwd;

}

}

1. 业务逻辑层代码

UserService类：

import java.util.List;

import com.stock.dao.UserDao;

import com.stock.entity.User;

/\*\*

\* 业务类

\* 用户相关的业务操作

\*/

public class UserService {

UserDao userDao=new UserDao();

/\*\*

\* 用户登录

\* @param userNo 用户名(编号)

\* @param pwd 密码

\* @return 用户对象

\*/

public User login(String userNo, String pwd) {

return userDao.login(userNo,pwd);

}

/\*\*

\* 查询用户列表

\* @return 用户集合

\*/

public List<User> findUserList() {

return userDao.findUserList();

}

/\*\*

\* 添加用户

\* @param user 用户对象

\* @return 是否添加成功

\*/

public boolean addUser(User user) {

return userDao.addUser(user);

}

/\*\*

\* 删除用户

\* @param id 用户ID

\* @return 是否删除成功

\*/

public boolean deleteUser(String id) {

return userDao.deleteUser(id);

}

}

用户操作类：

import java.util.List;

import java.util.Scanner;

/\*\*

\* 用户视图层

\* 接收用户的选择并进行处理

\*/

public class UserView {

UserService userService = new UserService();

Scanner sc = new Scanner(System.in);

/\*\*

\* 用户登录

\*/

public User login() {

System.out.println("请输入用户名");

String userNo = sc.next();

System.out.println("请输入密码:");

String pwd = sc.next();

User user = userService.login(userNo, pwd);

if (user != null) {

System.out.println("登录成功！");

} else {

System.out.println("登录失败！！！");

}

return user;

}

/\*\*

\* 显示所有用户信息

\*/

public void showUserList() {

System.out.println("所有用户信息如下：");

List<User> list = userService.findUserList();

System.out.println("用户ID\t\t用户代码\t\t用户姓名\t\t密码\t\t性别\t\t年龄\t\t职位");

for (User user : list) {

System.out.println(user.getId() + "\t\t" + user.getUserno() + "\t\t"

+ user.getUsername() + "\t\t" + user.getPwd() + "\t\t"

+ user.getSex() + "\t\t" + user.getAge() + "\t\t"

+ user.getPosition());

}

}

public void addUser() {

System.out.println("输入学号：");

String userNo = sc.next();

System.out.println("输入用户名：");

String userName = sc.next();

System.out.println("输入密码：");

String pwd = sc.next();

User user = new User();

user.setUserno(userNo);

user.setUsername(userName);

user.setPwd(pwd);

if (userService.addUser(user)) {

System.out.println("添加用户成功");

} else {

System.out.println("添加用户失败");

}

}

public void deleteUser() {

System.out.println("请输入要删除的用户id：");

String id = sc.next();

if (userService.deleteUser(id)) {

System.out.println("删除成功。");

} else {

System.out.println("删除失败！！！");

}

}

}

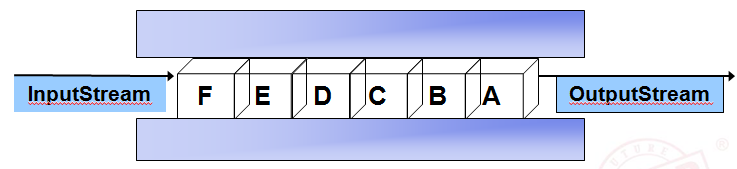
# IO流与文件读写

## 学习目标

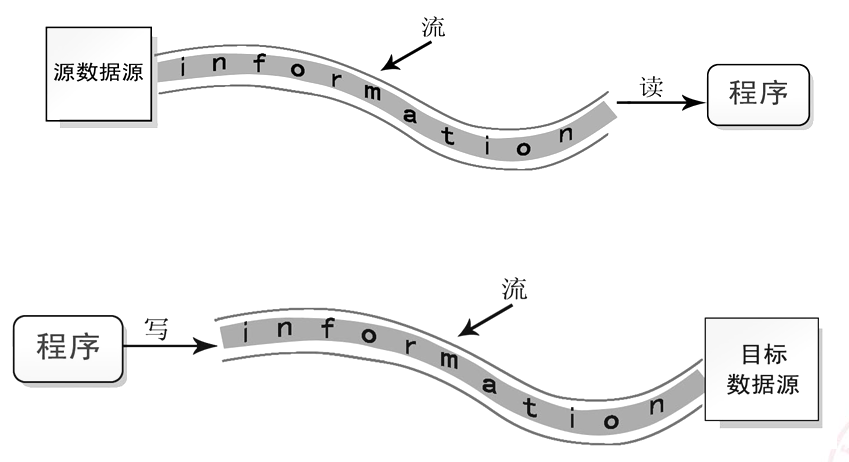
* 掌握File类的常用方法
* 掌握文件读写操作
* 掌握字节流与字符流的区别

## 文件读写

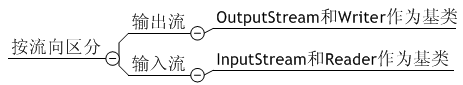
Java中通过流来读写文件，流是指一连串流动的字符，是以先进先出方式发送信息的通道。



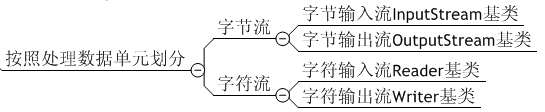
输入/输出流与数据源：



Java流的分类：



输入输出流是相对于计算机内存来说的。



### 字节流与字符流的区别

字符流的底层就是字节流。而字符流主要是读取文本文件内容的，可以一个字符一个字符的读取，也可以一行一行的读取文本文件内容，而字节流读取单位为byte。byte作为计算机存储最基本单位，可以用字节流来读取很多其他格式的文件，比如图片视频等等。基于B/S和C/S的文件传输都可以采用字节流的形式。在读写文件需要对内容按行处理，比如比较特定字符，处理某一行数据的时候一般会选择字符流。只是读写文件，和文件内容无关的，一般选择字节流。

字节流就是按照字节来进行传输，字符流是按照字符来传输.最基本的是字节流，字符流可以说是对字节流的一个包装流，比如你知道了一个字符是8个字节，那么你让字节流一次传输8个字节，那不就相当于一次传输一个字符，也就是字符流了。

字节流与和字符流的使用非常相似，两者除了操作代码上的不同之外，是否还有其他的不同呢？

实际上字节流在操作时本身不会用到缓冲区（内存），是文件本身直接操作的，而字符流在操作时使用了缓冲区，通过缓冲区再操作文件，如图：



下面以两个写文件的操作为主进行比较，但是在操作时字节流和字符流的操作完成之后都不关闭输出流。

范例：使用字节流不关闭执行：

// 通过字节流 写文件

public static void ByteWrite() {

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

try {

OutputStream outputStream = new FileOutputStream(file);

String textString = "Hello World!!!";

byte[] data = textString.getBytes();

outputStream.write(data);

//outputStream.close();

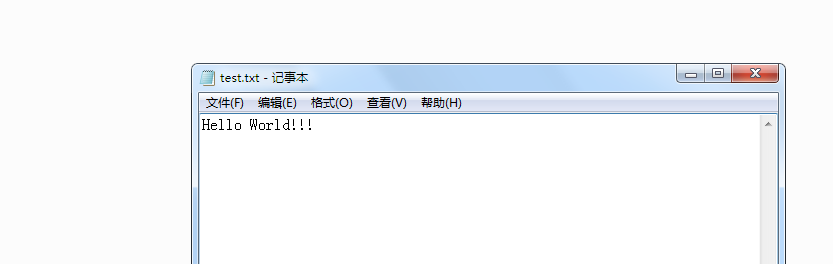
} catch (IOException e) {

System.out.println("字节流写入文件出错。");

e.printStackTrace();

}

}



此时没有关闭字节流操作，但是文件中也依然存在了输出的内容，证明字节流是直接操作文件本身的。而下面继续使用字符流完成，再观察效果。

范例：使用字符流不关闭执行：

// 通过字符流 写文件

public static void StringWrite() {

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

try {

FileWriter writer = new FileWriter(file);

String textString = "Hello World!!!";

writer.write(textString);

// writer.flush();

// writer.close();

} catch (IOException e) {

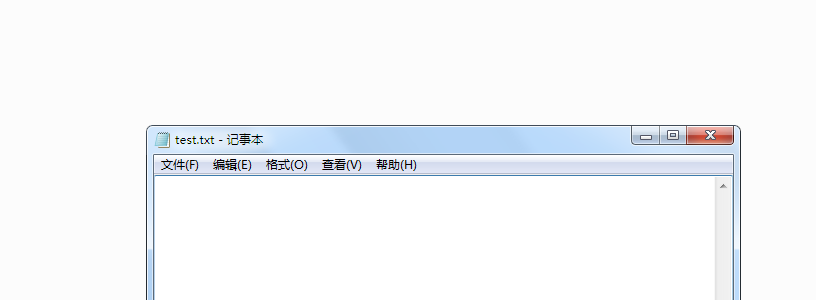
System.out.println("字符流写入文件出错.");

e.printStackTrace();

}

}

程序运行结果：



程序运行后会发现文件中没有任何内容，这是因为字符流操作时使用了缓冲区，而 在关闭字符流时会强制性地将缓冲区中的内容进行输出，但是如果程序没有关闭，则缓冲区中的内容是无法输出的，所以得出结论：字符流使用了缓冲区，而字节流没有使用缓冲区。

在开发中是使用字节流好还是字符流好呢？

回答：使用字节流更好。所有的文件在硬盘或在传输时都是以字节的方式进行的，包括图片等都是按字节的方式存储的，而字符是只有在内存中才会形成，所以在开发中，字节流使用较为广泛。

如果要java程序实现一个拷贝功能，应该选用字节流进行操作（可能拷贝的是图片），并且采用边读边写的方式（节省内存）。

### 使用FileInputStream读取文本文件

示例代码：

/\*\*

\* 文件读取

\*/

public static void readFile(){

File file=new File("D:/test/students.txt");

try {

InputStreamReader read = new InputStreamReader(new FileInputStream(file), "utf8");

BufferedReader bufferedReader = new BufferedReader(read);

String lineTxt = null;

//循环输出每一行内容

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

System.out.println(lineTxt);

}

read.close();

} catch (Exception e) {

e.printStackTrace();

}

}

或者：

/\*

\* 读取文件

\*/

public static String read(String fileName) {

File f = new File(fileName);

if (!f.exists()) {

return "File not found!";

}

FileInputStream fs;

String result = null;

try {

fs = new FileInputStream(f);

byte[] b = new byte[fs.available()];

fs.read(b);

fs.close();

result = new String(b);

} catch (Exception e) {

e.printStackTrace();

}

return result;

}

InputStream类常用方法及解析：

public void close() throws IOException{}

关闭此文件输入流并释放与此流有关的所有系统资源。抛出IOException异常。

protected void finalize()throws IOException {}

这个方法清除与该文件的连接。确保在不再引用文件输入流时调用其 close 方法。抛出IOException异常。

public int read(int r)throws IOException{}

这个方法从 InputStream 对象读取指定字节的数据。返回为整数值。返回下一字节数据，如果已经到结尾则返回-1。

public int read(byte[] r) throws IOException{}

这个方法从输入流读取r.length长度的字节。返回读取的字节数。如果是文件结尾则返回-1。

public int available() throws IOException{}

返回下一次对此输入流调用的方法可以不受阻塞地从此输入流读取的字节数。返回一个整数值。

### 使用FileOutputStream 写文本文件

示例代码：

import java.io.FileOutputStream;

import java.io.IOException;

public class FileOutputStreamTest {

public static void main(String[] args) {

FileOutputStream fos=null;

try {

String str ="好好学习Java";

byte[] words = str.getBytes();

fos = new FileOutputStream("D:\\myDoc\\hello.txt");

fos.write(words, 0, words.length);

System.out.println("hello文件已更新!");

}catch (IOException obj) {

System.out.println("创建文件时出错!");

}finally{

try{

if(fos!=null)

fos.close();

}catch (IOException e) {

e.printStackTrace();

}

}

}

}

OutputStream类常用方法及解析：

public void close() throws IOException{}

关闭此文件输入流并释放与此流有关的所有系统资源。抛出IOException异常。

protected void finalize()throws IOException {}

这个方法清除与该文件的连接。确保在不再引用文件输入流时调用其 close 方法。抛出IOException异常。

public void write(int w)throws IOException{}

这个方法把指定的字节写到输出流中。

public void write(byte[] w)

把指定数组中w.length长度的字节写到OutputStream中。

## File类

Java程序通过java.io.File 类对文件进行访问。File 类的常用方法如下：

|  |  |
| --- | --- |
| **方法名称** | **说 明** |
| boolean exists( ) | 判断文件或目录是否存在 |
| boolean isFile( ) | 判断是否是文件 |
| boolean isDirectory( ) | 判断是否是目录 |
| String getPath( ) | 返回此对象表示的文件的相对路径名 |
| String getAbsolutePath( ) | 返回此对象表示的文件的绝对路径名 |
| String getName( ) | 返回此对象表示的文件或目录的名称 |
| boolean delete( ) | 删除此对象指定的文件或目录 |
| boolean createNewFile( ) | 创建名称的空文件，不创建文件夹 |
| long length() | 返回文件的长度，单位为字节, 如果文件不存在，则返回 0L |

File类的操作示例如下：

import java.io.\*;

public class FileMethods {

public static void main(String[] args) {

FileMethods fm=new FileMethods();

File file=null;

file=new File("D:\\myDoc\\test.txt");

//fm.create(file);

fm.showFileInfo(file);

//fm.delete(file);

}

/\*\*

\* 创建文件的方法

\* @param file 文件对象

\*/

public void create(File file){

if(!file.exists()){

try {

file.createNewFile();

System.out.println("文件已创建！");

} catch (IOException e) {

e.printStackTrace();

}

}

}

/\*\*

\* 删除文件

\* @param file 文件对象

\*/

public void delete(File file){

if(file.exists()){

file.delete();

System.out.println("文件已删除！");

}

}

/\*\*

\* 显示文件信息

\* @param file 文件对象

\*/

public void showFileInfo(File file){

if(file.exists()){ //判断文件是否存在

if(file.isFile()){ //如果是文件

System.out.println("名称:" + file .getName());

System.out.println("相对路径: " + file.getPath());

System.out.println("绝对路径: " + file.getAbsolutePath());

System.out.println("文件大小:" + file.length()+ " 字节");

}

if(file.isDirectory()){

System.out.println("此文件是目录");

}

}else

System.out.println("文件不存在");

}

}

### 创建目录

File类中有两个方法可以用来创建文件夹：

* mkdir( )方法创建一个文件夹，成功则返回true，失败则返回false。失败表明File对象指定的路径已经存在，或者由于整个路径还不存在，该文件夹不能被创建。
* mkdirs()方法创建一个文件夹和它的所有父文件夹。

下面的例子创建 "/tmp/user/java/bin"文件夹：

import java.io.File;

public class CreateDir {

public static void main(String args[]) {

String dirname = "/tmp/user/java/bin";

File d = new File(dirname);

// 现在创建目录

d.mkdirs();

}

}

编译并执行上面代码来创建目录 "/tmp/user/java/bin"。

注意：Java 在 UNIX 和 Windows 自动按约定分辨文件路径分隔符。如果你在 Windows 版本的 Java 中使用分隔符 (/) ，路径依然能够被正确解析。

### 读取目录

一个目录其实就是一个 File 对象，它包含其他文件和文件夹。

如果创建一个 File 对象并且它是一个目录，那么调用 isDirectory() 方法会返回 true。可以通过调用该对象上的 list() 方法，来提取它包含的文件和文件夹的列表。下面展示的例子说明如何使用 list() 方法来检查一个文件夹中包含的内容：

import java.io.File;

public class DirList {

public static void main(String args[]) {

String dirname = "/tmp";

File f1 = new File(dirname);

if (f1.isDirectory()) {

System.out.println("目录 " + dirname);

String s[] = f1.list();

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

File f = new File(dirname + "/" + s[i]);

if (f.isDirectory()) {

System.out.println(s[i] + " 是一个目录");

} else {

System.out.println(s[i] + " 是一个文件");

}

}

} else {

System.out.println(dirname + " 不是一个目录");

}

}

}

### 删除目录或文件

删除文件可以使用 java.io.File.delete() 方法。

以下代码会删除目录 /tmp/java/，需要注意的是当删除某一目录时，必须保证该目录下没有其他文件才能正确删除，否则将删除失败。

import java.io.File;

public class DeleteFileDemo {

public static void main(String args[]) {

// 这里修改为自己的测试目录

File folder = new File("/tmp/java/");

deleteFolder(folder);

}

// 删除文件及目录

public static void deleteFolder(File folder) {

File[] files = folder.listFiles();

if (files != null) {

for (File f : files) {

if (f.isDirectory()) {

deleteFolder(f);

} else {

f.delete();

}

}

}

folder.delete();

}

}

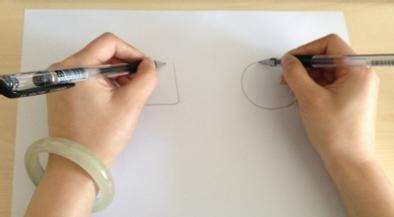
# 多线程

## 学习目标

* 理解线程的基本概念
* 理解Java的线程模型
* 掌握线程的创建和使用
* 理解多线程的概念
* 掌握Java的多线程实现
* 掌握线程的同步技巧

## 为什么要使用多线程

如果不考虑多线程的话，那么在程序只有一条执行路径，代码串行执行：顺序执行、选择或者循环。单线程就像你用你惯常的手去写字，多线程编程就要求你左手画圆，右手画方。



虽然电脑只有一个CPU，但是电脑却不只做一种工作，好比只有一个车间，一次只能容一个工人工作，但你要完成的工作种类很多，所以就让不同的工人在不同的时间干不同的工作，一个工人就是一个线程。当然你可以只用一个工人完成所有的工作，这样还省去换班的时间，不过前提是你得能找到这样全能的工人。



一个线程完不成所有的工作，通常Windows有数百个线程在运行。

多线程出现的原因：

为了解决负载均衡问题，充分利用CPU资源，为了提高CPU的使用率，采用多线程的方式去同时完成几件事情而不互相干扰，为了处理大量的IO操作时或处理的情况需要花费大量的时间等等。比如：读写文件、视频图像的采集等。

多线程的好处：

1.使用线程可以把占据时间长的程序中的任务放到后台去处理

2.用户界面更加吸引人,这样比如用户点击了一个按钮去触发某件事件的处理,可以弹出一个进度条来显示处理的进度

3.程序的运行效率可能会提高

4.在一些等待的任务实现上如用户输入,文件读取和网络收发数据等,线程就比较有用了.

多线程的缺点：

1.如果有大量的线程,会影响性能,因为操作系统需要在它们之间切换.

2.更多的线程需要更多的内存空间

3.线程中止需要考虑对程序运行的影响.

4.通常块模型数据是在多个线程间共享的,需要防止线程死锁情况的发生

## 基本概念

多线程：在一个程序中同时运行多个任务。

多进程：在操作系统中能同时运行多个任务(程序)。

线程和进程的区别：

* 进程是运行中的应用程序，拥有自己独立的内存空间和资源
* 一个进程可以有一个或多个线程组成，且至少有一个线程称为主线程
* 线程是最小的处理单位,多个线程共享一块内存和资源
* 当一个线程改变了所属进程的变量时，其它线程下次访问该变量时得到这种改变

## 线程的使用

每个Java程序至少包含一个线程：主线程。其他线程都是通过Thread构造器或实例化继承类Thread的类来创建的。

class MainThread { public static void main(String args[]) { // 调用Thread类的currentThread()方法获取当前线程 Thread t = Thread.currentThread(); System.out.println("主线程是: " + t); }}

创建线程有两种方法：

* 使用Thread类
* 使用Runnable接口

继承Thread类，重写run方法，可以创建线程类。

例如：

class Thread1 extends Thread { public void run() { // 获取当前线程的名字 System.out.println( Thread.currentThread().getName()); }}

Runnable接口由单个方法run()组成，它在线程激活时执行。

通过Runnable接口创建线程的步骤如下：

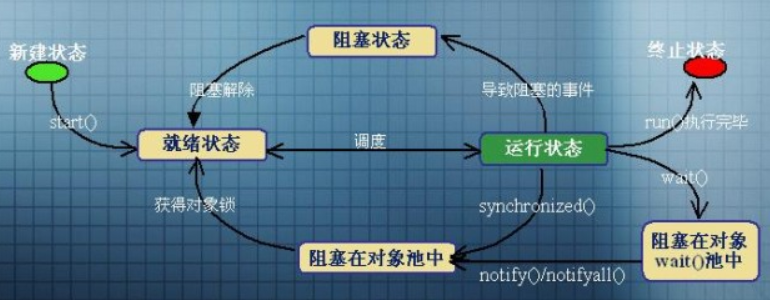
1. 实例化实现Runnable接口的类

2. 建立一个Thread对象，并将第一步实例化后的对象作为参数传入Thread类的构造方法

3. 通过Thread类的start()方法建立线程。

class <类\_名> implements Runnable{ public void run() { //线程的任务功能 }}

## 线程的生命周期



线程是一个动态执行的过程，它也有一个从产生到死亡的过程。

新建状态：

使用 new 关键字和 Thread 类或其子类建立一个线程对象后，该线程对象就处于新建状态。它保持这个状态直到程序 start() 这个线程。

就绪状态：

当线程对象调用了start()方法之后，该线程就进入就绪状态。就绪状态的线程处于就绪队列中，要等待JVM里线程调度器的调度。

运行状态：

如果就绪状态的线程获取 CPU 资源，就可以执行 run()，此时线程便处于运行状态。处于运行状态的线程最为复杂，它可以变为阻塞状态、就绪状态和死亡状态。

阻塞状态：

如果一个线程执行了sleep（睡眠）、suspend（挂起）等方法，失去所占用资源之后，该线程就从运行状态进入阻塞状态。在睡眠时间已到或获得设备资源后可以重新进入就绪状态。可以分为三种：

等待阻塞：运行状态中的线程执行 wait() 方法，使线程进入到等待阻塞状态。

同步阻塞：线程在获取 synchronized 同步锁失败(因为同步锁被其他线程占用)。

其他阻塞：通过调用线程的 sleep() 或 join() 发出了 I/O 请求时，线程就会进入到阻塞状态。当sleep() 状态超时，join() 等待线程终止或超时，或者 I/O 处理完毕，线程重新转入就绪状态。

死亡状态：

一个运行状态的线程完成任务或者其他终止条件发生时，该线程就切换到终止状态。

下面是一个创建线程并开始让它执行的实例：

class RunnableDemo implements Runnable {

private Thread t;

private String threadName;

RunnableDemo( String name) {

threadName = name;

System.out.println("Creating " + threadName );

}

public void run() {

System.out.println("Running " + threadName );

try {

for(int i = 4; i > 0; i--) {

System.out.println("Thread: " + threadName + ", " + i);

// 让线程睡眠一会

Thread.sleep(50);

}

}catch (InterruptedException e) {

System.out.println("Thread " + threadName + " interrupted.");

}

System.out.println("Thread " + threadName + " exiting.");

}

public void start () {

System.out.println("Starting " + threadName );

if (t == null) {

t = new Thread (this, threadName);

t.start ();

}

}

}

public class TestThread {

public static void main(String args[]) {

RunnableDemo R1 = new RunnableDemo( "Thread-1");

R1.start();

RunnableDemo R2 = new RunnableDemo( "Thread-2");

R2.start();

}

}

编译以上程序运行结果如下：

Creating Thread-1

Starting Thread-1

Creating Thread-2

Starting Thread-2

Running Thread-1

Thread: Thread-1, 4

Running Thread-2

Thread: Thread-2, 4

Thread: Thread-1, 3

Thread: Thread-2, 3

Thread: Thread-1, 2

Thread: Thread-2, 2

Thread: Thread-1, 1

Thread: Thread-2, 1

Thread Thread-1 exiting.

Thread Thread-2 exiting.

创建一个线程的第二种方法是创建一个新的类，该类继承 Thread 类，然后创建一个该类的实例。

继承类必须重写 run() 方法，该方法是新线程的入口点。它也必须调用 start() 方法才能执行。

该方法尽管被列为一种多线程实现方式，但是本质上也是实现了 Runnable 接口的一个实例。

class ThreadDemo extends Thread {

private Thread t;

private String threadName;

ThreadDemo( String name) {

threadName = name;

System.out.println("Creating " + threadName );

}

public void run() {

System.out.println("Running " + threadName );

try {

for(int i = 4; i > 0; i--) {

System.out.println("Thread: " + threadName + ", " + i);

// 让线程睡眠一会

Thread.sleep(50);

}

}catch (InterruptedException e) {

System.out.println("Thread " + threadName + " interrupted.");

}

System.out.println("Thread " + threadName + " exiting.");

}

public void start () {

System.out.println("Starting " + threadName );

if (t == null) {

t = new Thread (this, threadName);

t.start ();

}

}

}

public class TestThread {

public static void main(String args[]) {

ThreadDemo T1 = new ThreadDemo( "Thread-1");

T1.start();

ThreadDemo T2 = new ThreadDemo( "Thread-2");

T2.start();

}

}

编译以上程序运行结果如下：

Creating Thread-1

Starting Thread-1

Creating Thread-2

Starting Thread-2

Running Thread-1

Thread: Thread-1, 4

Running Thread-2

Thread: Thread-2, 4

Thread: Thread-1, 3

Thread: Thread-2, 3

Thread: Thread-1, 2

Thread: Thread-2, 2

Thread: Thread-1, 1

Thread: Thread-2, 1

Thread Thread-1 exiting.

Thread Thread-2 exiting.

## 线程同步

线程同步：保证某个资源在某一时刻只能由一个线程访问，以此保证共享数据及操作的完整性。

同步的关键字synchronized，可以使用synchronized关键字修饰同步方法或同步块

synchronized 返回类型 方法(参数) { //同步方法体}