目录

[第1章 Java介绍及使用 8](#_Toc50821299)

[1-1 Java版本 8](#_Toc50821300)

[1-2 Java特性 8](#_Toc50821301)

[1-3 搭建JDK 8](#_Toc50821302)

[1-4 配置环境变量 8](#_Toc50821303)

[1-5 编译器 8](#_Toc50821304)

[第2章 Java基础 9](#_Toc50821305)

[2-1 主类结构 9](#_Toc50821306)

[包声明 9](#_Toc50821307)

[成员变量和局部变量 9](#_Toc50821308)

[编写主方法 9](#_Toc50821309)

[导入API类库 9](#_Toc50821310)

[2-2 基本数据类型 9](#_Toc50821311)

[整数型 9](#_Toc50821312)

[浮点类型 10](#_Toc50821313)

[字符类型 10](#_Toc50821314)

[布尔类型 10](#_Toc50821315)

[2-3 变量和常量 10](#_Toc50821316)

[标识符和关键字 10](#_Toc50821317)

[变量 11](#_Toc50821318)

[常量 11](#_Toc50821319)

[变量的有效范围 11](#_Toc50821320)

[2-4 运算符 11](#_Toc50821321)

[赋值运算符 11](#_Toc50821322)

[算术运算符 11](#_Toc50821323)

[自增、自减 11](#_Toc50821324)

[比较运算符 12](#_Toc50821325)

[逻辑运算符 12](#_Toc50821326)

[位运算符 12](#_Toc50821327)

[三元运算符 13](#_Toc50821328)

[运算符优先级 13](#_Toc50821329)

[2-5 数据类型转换 13](#_Toc50821330)

[隐式转换 13](#_Toc50821331)

[显式转换 13](#_Toc50821332)

[2-6 代码注释和编写规范 13](#_Toc50821333)

[代码注释 13](#_Toc50821334)

[编写规范 14](#_Toc50821335)

[第3章 流程控制 15](#_Toc50821336)

[3-1 条件语句 15](#_Toc50821337)

[if条件语句 15](#_Toc50821338)

[switch多分支语句 15](#_Toc50821339)

[3-2 循环语句 16](#_Toc50821340)

[while循环语句 16](#_Toc50821341)

[do...while循环语句 16](#_Toc50821342)

[for循环语句 16](#_Toc50821343)

[foreach循环语句 16](#_Toc50821344)

[3-3 循环控制 16](#_Toc50821345)

[break语句 16](#_Toc50821346)

[continue语句 17](#_Toc50821347)

[第4章 字符串 18](#_Toc50821348)

[4-1 String类 18](#_Toc50821349)

[声明字符串 18](#_Toc50821350)

[创建字符串 18](#_Toc50821351)

[4-2 连接字符串 18](#_Toc50821352)

[使用”+”号连接 18](#_Toc50821353)

[连接其他类型数据 18](#_Toc50821354)

[4-3 获取字符串信息 19](#_Toc50821355)

[获取字符串长度 19](#_Toc50821356)

[字符串查找 19](#_Toc50821357)

[获取指定索引位置的字符 19](#_Toc50821358)

[4-4 字符串操作 19](#_Toc50821359)

[获取子字符串 19](#_Toc50821360)

[去除空格 20](#_Toc50821361)

[字符串替换 20](#_Toc50821362)

[判断字符串的开始和结尾 20](#_Toc50821363)

[判断字符串是否相等 20](#_Toc50821364)

[字母大小写转换 20](#_Toc50821365)

[字符串分割 20](#_Toc50821366)

[4-5 格式化字符串 21](#_Toc50821367)

[日期格式化 21](#_Toc50821368)

[时间格式化 21](#_Toc50821369)

[常见的日期时间组合 22](#_Toc50821370)

[常规类型格式化 22](#_Toc50821371)

[4-6 正则表达式 22](#_Toc50821372)

[4-7 字符串生成 23](#_Toc50821373)

[第5章 数组 24](#_Toc50821374)

[5-1 一维数组创建和使用 24](#_Toc50821375)

[创建一维数组 24](#_Toc50821376)

[初始化 24](#_Toc50821377)

[使用一维数组 24](#_Toc50821378)

[5-2 二维数组的创建和使用 24](#_Toc50821379)

[创建二维数组 24](#_Toc50821380)

[初始化 25](#_Toc50821381)

[使用二维数组 25](#_Toc50821382)

[5-3 数组的基本操作 25](#_Toc50821383)

[遍历数组 25](#_Toc50821384)

[替换数组元素 25](#_Toc50821385)

[数组排序 26](#_Toc50821386)

[复制数组 26](#_Toc50821387)

[数组查询 26](#_Toc50821388)

[5-4 数组排序算法 26](#_Toc50821389)

[冒泡排序 26](#_Toc50821390)

[直接选择排序 27](#_Toc50821391)

[反转排序 27](#_Toc50821392)

[第6章 类与对象 28](#_Toc50821393)

[6-1 面向对象 28](#_Toc50821394)

[对象 28](#_Toc50821395)

[类 28](#_Toc50821396)

[封装 28](#_Toc50821397)

[继承 28](#_Toc50821398)

[多态 28](#_Toc50821399)

[6-2 类 28](#_Toc50821400)

[成员变量 28](#_Toc50821401)

[成员方法 28](#_Toc50821402)

[权限修饰符 29](#_Toc50821403)

[局部变量 29](#_Toc50821404)

[this 29](#_Toc50821405)

[构造方法 29](#_Toc50821406)

[静态变量,常量和方法 30](#_Toc50821407)

[6-3 方法 31](#_Toc50821408)

[主方法 31](#_Toc50821409)

[方法重载 31](#_Toc50821410)

[方法递归 32](#_Toc50821411)

[代码块 32](#_Toc50821412)

[6-4 对象 33](#_Toc50821413)

[创建对象 33](#_Toc50821414)

[使用对象 33](#_Toc50821415)

[对象的引用 33](#_Toc50821416)

[对象的比较 34](#_Toc50821417)

[对象销毁 34](#_Toc50821418)

[第7章 包装类 35](#_Toc50821419)

[7-1 Integer、Byte、Short、long(整数类型) 35](#_Toc50821420)

[装和拆 35](#_Toc50821421)

[常量 35](#_Toc50821422)

[常用方法 35](#_Toc50821423)

[7-2 Boolean(布尔类型) 36](#_Toc50821424)

[装和拆 36](#_Toc50821425)

[常量 36](#_Toc50821426)

[常用方法 36](#_Toc50821427)

[7-3 Character(字符类型) 36](#_Toc50821428)

[装和拆 36](#_Toc50821429)

[常量 36](#_Toc50821430)

[常用方法 36](#_Toc50821431)

[7-4 Double、Float(浮点类型) 37](#_Toc50821432)

[装和拆 37](#_Toc50821433)

[常量 37](#_Toc50821434)

[常用方法 37](#_Toc50821435)

[7-5 Number 37](#_Toc50821436)

[第8章 数字处理类 38](#_Toc50821437)

[8-1 数字格式化 38](#_Toc50821438)

[8-2 数字运算 38](#_Toc50821439)

[最大(小)值、绝对值 39](#_Toc50821440)

[三角函数 39](#_Toc50821441)

[指函数 39](#_Toc50821442)

[取整函数 39](#_Toc50821443)

[8-3 随机数 39](#_Toc50821444)

[Math.random()方法 39](#_Toc50821445)

[Random类 40](#_Toc50821446)

[8-4 大数字运算 40](#_Toc50821447)

[BigInteger 40](#_Toc50821448)

[BigDecimal 42](#_Toc50821449)

[第9章 接口、继承、多态 43](#_Toc50821450)

[9-1 继承 43](#_Toc50821451)

[继承父类 43](#_Toc50821452)

[重写方法 43](#_Toc50821453)

[高级重写 44](#_Toc50821454)

[9-2 Object类 44](#_Toc50821455)

[用法 44](#_Toc50821456)

[方法 44](#_Toc50821457)

[9-3 对象类型转换 45](#_Toc50821458)

[向上转型 45](#_Toc50821459)

[向下转型 46](#_Toc50821460)

[instanceof 46](#_Toc50821461)

[上下转型的作用 46](#_Toc50821462)

[9-4 多态 46](#_Toc50821463)

[9-5 抽象类 47](#_Toc50821464)

[9-6 接口 48](#_Toc50821465)

[第10章 类的高级特性 49](#_Toc50821466)

[10-1 Java类包 49](#_Toc50821467)

[类名冲突 49](#_Toc50821468)

[类路径 49](#_Toc50821469)

[创建导入包 49](#_Toc50821470)

[导入静态变量 49](#_Toc50821471)

[10-2 final 49](#_Toc50821472)

[final方法 50](#_Toc50821473)

[final类 50](#_Toc50821474)

[10-3 内部类 50](#_Toc50821475)

[成员内部类 51](#_Toc50821476)

[局部内部类 51](#_Toc50821477)

[匿名内部类 52](#_Toc50821478)

[静态内部类 53](#_Toc50821479)

[内部类继承 53](#_Toc50821480)

[第11章 异常处理 55](#_Toc50821481)

[区分异常 55](#_Toc50821482)

[捕捉异常 55](#_Toc50821483)

[Java常见异常 56](#_Toc50821484)

[自定义异常 57](#_Toc50821485)

[运行时异常 58](#_Toc50821486)

[异常使用原则 58](#_Toc50821487)

[第12章 Swing 59](#_Toc50821488)

[12-1 Swing概述 59](#_Toc50821489)

[12-2 窗体 60](#_Toc50821490)

[JFrame 60](#_Toc50821491)

[JDialog 61](#_Toc50821492)

[12-3 容器 62](#_Toc50821493)

[12-4 标签和图 62](#_Toc50821494)

[12-5 布局管理 63](#_Toc50821495)

[绝对布局 63](#_Toc50821496)

[流布局 64](#_Toc50821497)

[边界布局 65](#_Toc50821498)

[网格布局 65](#_Toc50821499)

[网格组布局 66](#_Toc50821500)

[12-6 面板 73](#_Toc50821501)

[JPanel 73](#_Toc50821502)

[JScrollPane 74](#_Toc50821503)

[12-7 按钮组件 74](#_Toc50821504)

[提交按钮 74](#_Toc50821505)

[单选按钮 75](#_Toc50821506)

[多选按钮 75](#_Toc50821507)

[12-8 列表组件 76](#_Toc50821508)

[下拉列表 76](#_Toc50821509)

[列表框 77](#_Toc50821510)

[12-9 文本组件 78](#_Toc50821511)

[文本框 78](#_Toc50821512)

[密码框 78](#_Toc50821513)

[文本域 79](#_Toc50821514)

[12-10 事件监听 79](#_Toc50821515)

[动作事件监听 79](#_Toc50821516)

[焦点事件监听 80](#_Toc50821517)

# Java介绍及使用

## Java版本

1. Java SE: Java标准版,用于大多数的桌面程序开发,包含了(Java语言基础,Java数据库连接操作,I(输入)/O(输出),网络通信,多线程).
2. Java EE: Java企业版,开发企业级分布式网络程序.
3. Java ME: Java嵌入版,开发嵌入式系统,用于手机平板等终端设备.

## Java特性

1. 面向对象：万物皆可对象。
2. 分布性：在多个主机上布置操作，存放数据，通过URL对象访问网络对象。
3. 可移植性：程序可移植，类库可移植。
4. 解释型：任何移植了java解释器的计算器或其他设备都可以用java字节码编译执行。
5. 安全性：代码经过多次测试步骤才能运行，未经允许的java程序不可能出现损害系统的行为。
6. 健壮性：java可以编写出多方面的、可靠的程序。
7. 多线程：在同一时间运行多个程序，且数据共享。
8. 高性能：先编译字节码直接向机器发出指令，速度提升显著。
9. 动态：动态适应环境，调整库中方法。

## 搭建JDK

官网下载最新版

## 配置环境变量

安装: 我的电脑(右击)-属性-高级系统属性-环境变量-系统变量-Path-编辑文本-添加Java安装路径

测试: 命令提示符输入javac

## 编译器

Eclipse

Intellij IDE

新建文本文档（手动编译）

注: 编程里每一个标点符号都是英文

注意语句结束的分号

注意大括号小括号有没有收尾

区分大小写

# Java基础

## 主类结构

Java程序组成基本单元是类，类包括属性与方法，其中含有main()方法的为主类.

包声明

package (包名);

成员变量和局部变量

类中的属性为成员(全局)变量,方法中的属性为局部变量.

public class (类名){

成员变量;

public static void main(String[] args){

局部变量;

}

}

编写主方法

main()方法为类中主方法,{}大括号内的为方法内容,方法的声明为:

public(权限修饰符) static(静态修饰符) void(返回修饰符) String[] args(主方法参数)

main()为程序执行的开始.

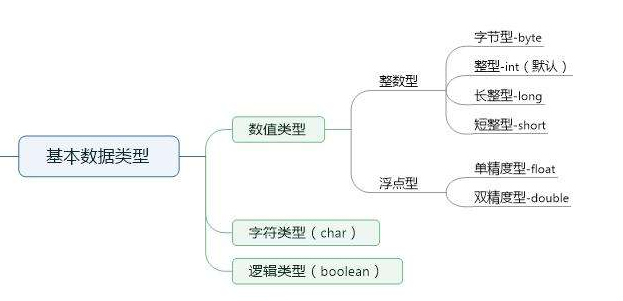
导入API类库

import java.\*.\*;

import java.util.Scanner; //导入Scanner类

导入需要用到的类库.

## 基本数据类型



整数型

取值范围:

byte字节型 -128~127

short短整型 -32768~32767

int整型 -2147483648~2147483647

long长整型 -9223372036854775808~9223372036854775807

定义变量:

int x; //定义x y

int x,y; //同时定义x y

int x=123.y=456; //同时定义并赋值

定义long类型需要在后面加上L或l

long x=123123123123L;

浮点类型

取值范围:

float(单精度浮点) 小数点往后七位,够用就行

double(双精度浮点) 同上

默认情况,小数都被看做double类型,若定义float,则需要在后面加上F或f

float x=1.23F

double y=1.23

字符类型

分为单字符和转义字符,使用单引号表示

char x=’a’;

char y=’你’;

由于字符在常量区中用unicode表传递,所以也可以使用排序位置表示

char x=97; //a在unicode表中排97位,可以用97代表a

也可以通过字符寻找数字位置:

public class 类名{ //定义类

public static void main(String[] args){ //定义主方法

char x=’你’; //定义char变量,赋值’你’

System.out.println((int)x); //控制台输出将char转为int后,’你’在unicode表中排序

int y=56; //同理,可以定义int类型查询该数字排序所对应的字符

}

}

转义字符,以反斜杠开头\,代表特定的含义

|  |  |
| --- | --- |
| \\*\*\* | 代表1~3位八进制代表的字符,\123 |
| \u\*\*\*\* | 代表4位十六进制代表的字符.\u0052 |
| \’ | 单引号 |
| \\ | 反斜杠 |
| \t | 制表符 |
| \r | 回车 |
| \n | 换行 |
| \b | 退格 |
| \r | 换页 |

布尔类型

布尔类型相当于判断,只有两个值,true”对”和false”错”.

boolean x; //定义一个布尔类型,默认false;

boolean y=true; //定义一个布尔类型,赋值true;

通常用于条件判断,流程控制.

## 变量和常量

标识符和关键字

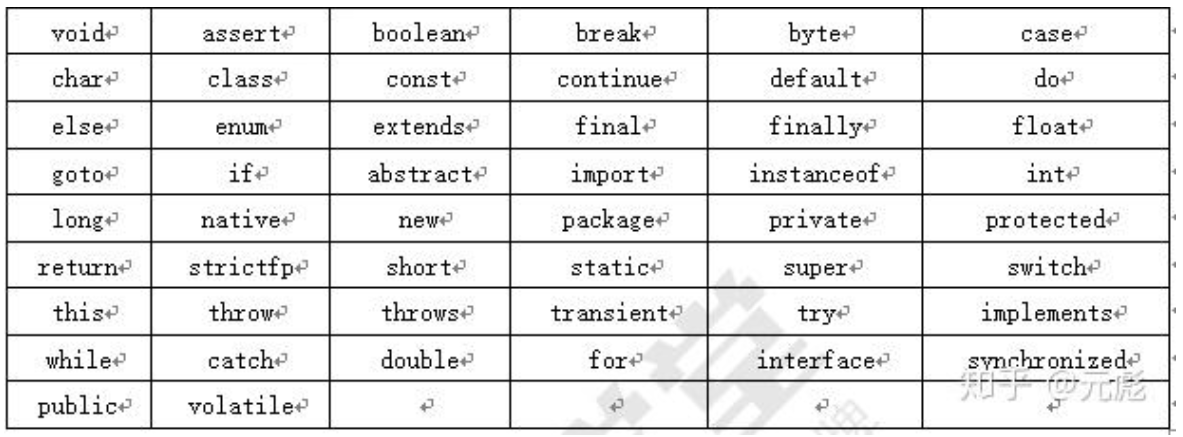
在java中,标识符可以理解为名字:

number //合格的标识符

5b //第一个字符不能是数字

int //不能是关键字

关键字就是java语言里的语法词语.被java赋予特殊意义的单词



关键字已经定义好的,所以标识符不能是关键字,否则程序无法编译.

变量

变量,可以改变的量.声明方式如下:

int x=1; //写法顺序是: 变量类型 变量名字 = 赋的值;

int y; //可以声明的同时不赋值

声明变量的名字(标识符)通常使用英文,且使用有含义的单词:

int number; //声明一个存储号码(number)的变量

int Password; //声明一个存储密码(Password)的变量

常量

常量就是变量前加一个限定单词final;

final int x=1; //声明一个常量,值为1

声明常量的同时必须进行赋值,不同于变量可以不赋值.

变量的有效范围

用变量的生命周期(使用范围)分类

|  |  |
| --- | --- |
| 成员变量 | 实例变量(本类变量)本类中使用,声明位置在类中public class方法外static |
| 静态变量(跨类变量)全程序使用在变量前加上static将变量变为类,即可跨类使用 |
| 局部变量 | 方法变量,在方法内声明的变量,仅在本方法内使用 |
| 逻辑变量,在逻辑语句中声明,仅在逻辑语句中使用 |

## 运算符

赋值运算符

= //将右边的值或表达式赋给左边的变量

int x=y=1+1; //多个赋值,先处理最右边的赋值,先计算1+1,然后赋值给y,然后y赋值给x

算术运算符

+ //加法

- //减法

\* //乘法

/ //除法

% //取余 8%3得2余2 结果为2

自增、自减

++或—代表值+1或-1,且使用自增自减的数据类型必须是整型或浮点型,

int a=2,b=5; //声明变量a赋值2 b赋值5;

b=++a; //先使a+1,然后再赋值给b,此时a=3,b=3

b=--a; //先使a-1,然后再赋值给b,此时a=1,b=1;

b=a++; //先将a赋值给b,再使a+1,此时a=3,b=2

b=a--; //先将a赋值给b,再使a-1,此时a=1,b=2;

比较运算符

比较运算符,结果为boolean类型(布尔),当结果成立时,boolean值为true,反之为false.

比较大小的使用于整数,浮点,字符

比较等于的适用于所有类型

> //比较左大于右

>= //比较左大等于右

< //比较右大于左

<= //比较右大等于左

== //比较两边等于

!= //比较两边不等于

逻辑运算符

判断两边表达式是否相等,结果为布尔值

int a=2;

int b=5;

& //与 两者同时满足则返回true

|| //或 两者有一者满足则返回true

! //非 false加!返回true true加!返回 false

&& //短路与 两者同时满足则返回true,如第一个不满足,直接返回false,节省了判断次数.

位运算符

由于整型数据在内存中以二进制的形式存在,所以位运算符是对内存中的二进制数据进行运算.

与逻辑运算符不同的是,逻辑运算的结果的布尔,位运算的结果是原来的数据,相当于加减乘除

5&-4 //按位与 5的二进制: 00000000 00000000 00000000 00000101

// -4的二进制: 11111111 11111111 11111111 11111100

//当上下同时等于1时才会得1: 00000000 00000000 00000000 00000100 结果为4

3|6 //按位或 3的二进制: 00000000 00000000 00000000 00000011

// 6的二进制: 00000000 00000000 00000000 00000110

//当上下有一个是1则返回1: 00000000 00000000 00000000 00000111 结果为7

10^3 //按位异或 10的二进制: 00000000 00000000 00000000 00001010

// 3的二进制: 00000000 00000000 00000000 00000011

//当上下相同返回0,上下不同返回1: 00000000 00000000 00000000 00001001 结果为9

~7 //按位取反 7的二进制: 00000000 00000000 00000000 00000111

//将1改为0 0改为1: 11111111 11111111 11111111 11111000 结果为-8

移位操作

23<<2 //左移 23的二进制: 00000000 00000000 00000000 00010111

//左移后多余的抛弃 00000000 00000000 00000000 00101110 结果为46

//一个数左移n位,就是将这个数乘以2ⁿ

23>>2 //右移 23的二进制: 00000000 00000000 00000000 00010111

//右移后多余的抛弃 00000000 00000000 00000000 00001011 结果为11

//一个数右移n位,就是将这个数除以2ⁿ

>>> //无符号右移

//右移前,最高位为1,则右移时的缺口填1,反之则0

三元运算符

表达式?值1:值2 //判断表达式 若true则返回值1 false则返回值2

boolean a = 1>2?true:false; //类似于if….else

boolean b; //声明变量b

if(1>2){ //如果1>2

b=true; //就把true赋值给b,也就是值1

}else{ //否则

b=false; //就把false赋值给b,也就是值2

}

运算符优先级



## 数据类型转换

隐式转换

低精度>>高精度

byte a=8; //声明byte变量并赋值8

int b=a; //声明int变量并赋值a,这时byte类型自动转换为int

long c =b; //声明long变量并赋值b,这时int类型自动转换为long

double d=c; //声明double变量并赋值c,这时long类型自动转换为double

显式转换

高精度>>低精度 可能导致精度损失，除boolean类型外的基本类型，都能进行显式转换

int a = (int)45.23 //将带有小数的值赋给int类型的a

long b = (long)456.7F //将float类型的数赋值给long类型的b

## 代码注释和编写规范

代码注释

// 单行注释,表示从符号”//”开始直到换行这一段内容都是注释

/\*

多行注释,从/\*开始到\*/结束,这一段内容都是注释

\*/

/\*\*

文档注释,当文档注释出现在声明,类,成员变量,方法之前,会被Javadoc文档工具读取做为Javadoc文档内容

\*/

编写规范

每条语句独占一行,尽量声明一个变量换一行;

关键字和关键字中只需要一个空格,多个无效;

多加注释,方便日后维护;

编写全程使用英文标点符号;

每一句使用分号结束;

声明变量的同时尽量赋值;

避免冗余,繁杂的代码,写完后检查是否可以更加精简地实现需求

# 流程控制

## 条件语句

if条件语句

if用于判断条件是否满足,满足则执行条件内语句,否则不执行,判断一定要返回boolean值

if(a=b){ //判断a是否等于b

System.out.println(“a=b”); //如果等于,则输出 “a=b” 这段话,否则跳过

}

if...else是if的加强版,进行一次判断 必要时可以使用三元运算符判断

if(a=b){ //判断a是否等于b

System.out.println(“a=b”); //如果相等,输出 “a=b” 这句话

}else{ //否则

System.out.println(“a不等于b”); //输出 “a不等于b” 这句话

}

if…else if语句是if的终极版,进行多次判断

if(a=b){ //判断a是否等于b

System.out.println(“a=b”); //如果相等,则输出 “a=b” 这句话,否则跳过

}else if(a<b){ //再判断a是否小于b

System.out.println(“a<b”); //如果小于,则输出 “a<b” 这句话,否则跳过

}else if(a>b){ //再判断a是否大于b

System.out.println(“a>b”); //如果大于,则输出 “a>b” 这句话,否则跳过

} //后面可以添加多次else if判断

switch多分支语句

与if不同的是,switch判断的是值是否相等,给定一个值,可以判断多个条件

switch(a){ //括号内填需要判断的值

case 1: //判断a的值是不是1,如果不是,则跳到下一个case

System.out.println(“a的值是1”); //如果是,则输出 “a的值是1”

break; //结束条件判断,跳出判断,不再执行下面语句

case 2: //判断a的值是不是2,如果不是,则跳到下一个case

System.out.println(“a的值是2”); //如果是,则输出 “a的值是2”

break; //结束条件判断,跳出判断,不再执行下面语句

case 3: //判断a的值是不是3,如果不是,则跳到下一个case

System.out.println(“a的值是3”); //如果是,则输出 “a的值是3”

break; //结束条件判断,跳出判断,不再执行下面语句

default: //如果上面的都不满足 default是必须的语句

System.out.println(“a的值不是1,2,3”); //则输出”a的值不是1,2,3”

} //每一个判断后可以不写东西 可以不带break,不带break程序会继续判断下面的case,增加程序负担(穿透性)

## 循环语句

while循环语句

利用一个条件来控制是否继续执行

int a = 1; //声明变量

while(a<10){ //判断a是否大于等于a

a++; //如果是,则a+1 如果不是,跳过语句

} //需要循环9次

do...while循环语句

与while不同的是,do...while先执行代码,再判断条件

int a=1; //声明变量

do{ //执行代码

a++; //a+1

}while(a<10) //判断是否继续执行

for循环语句

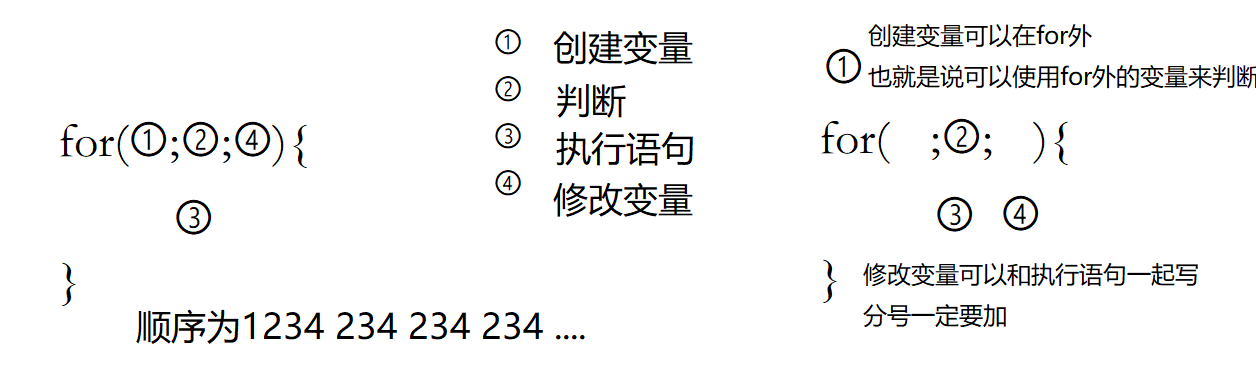
重复执行某个语句,直到条件满足

自定义循环次数,自定义声明变量

for(int x=0;x<=10;x++){ //for(创建变量 ; 循环条件,也就是变量满足条件 ; 修改变量,使变量逐渐满足条件)

//循环中执行的语句

} //顺序是:创建变量>>判断是否满足>>满足,执行下面的语句>>修改变量>>判断



foreach循环语句

for简化版，适用于数组

int a[]={1,2,3}; //声明数组

for(int x:a){ //for(变量 ; 遍历对象){

System.out.println(); //输出对象;

} //}

## 循环控制

break语句

跳出当前循环,中断循环

while(){ do{ for(){

break; break; break; //适用于所有循环结构

} } }

实例:

for(int i=1;i<20;i++){ //创建一个循环,每次循环i+1,直到i=20

if(i==10){ //判断i是否等于10

break; //如果是,中断循环,程序结束

} //

System.out.println(i); //输出i值,由于i循环到10时中断了,结果只有123456789

}

循环嵌套的情况下,break只可以跳出一层循环,若想跳出外层循环,则使用标签

a:for(){ //第一个for循环,标签为a

b:for(){ //第二个for循环,嵌套在a中,标签为b

if(条件满足){ //如果条件满足

break a; //跳出a循环

} //标签名: 循环体 为循环贴标签,命名

} //break 标签名 跳出指定循环体

}

continue语句

是break的兄弟,但不是终止循环,而是不执行下面的语句,开始新的循环

while(){ do{ for(){

continue; continue; continue; //适用于所有循环结构

} } }

实例:

for(int i=1;i<20;i++){ //创建一个循环,每次循环i+1,直到i=20

if(i%2==0){ //判断i/2的余数是否为0,也就是判断是不是偶数

continue; //如果是,则跳过下面的代码,重新循环,但i不会变,也就是说本次循环也计数

} //

System.out.println(i); //输出i值,这时偶数已经被if语句剔除了,结果是20以内的奇数

}

continue同样支持标签功能

# 字符串

## String类

索引第一个是0,不是1

声明字符串

声明变量

String a=“java”; //String 指定这是字符串类型 a为该字符串命名 =为这个字符串赋值

在java中,字符串必须包含在 “ “ 内

“123456” “字符串”

字符串属于引用类型,不能作为基本类型使用

“1+2”不等于”3”

创建字符串

创建对象

1. String(char a[])

char a[]={‘j’,’a’,’v’,’a’}; //声明一个字符数组,并初始化元素

String b=new String(a); //创建一个String对象

String b=new String(“good”); //上面的语句可以结合成这句

1. String(char a[],int offset,int length)

char a[]={‘s’,’t’,’u’,’d’,’e’,’n’,’t’}; //声明一个字符串数组,并初始化元素

String b=new String(a,2,4); //创建一个String对象,(提取数组a的元素 , 开始截取的位置 , 截取的长度)

String b=new String(“uden”); //上面的语句可以结合成这句

## 连接字符串

使用”+”号连接

String a=“ja”; //声明一个String类型变量,赋值”ja”

String b=“va”; //声明一个String类型变量,赋值”va”

String c=a+b; //通过”+”号连接两个字符串,输出结果为java

一句相连的字符串不能分开两行写,如果过长,则使用

String a=“ja”+

“va”

连接其他类型数据

基本数据和字符串连接时，将会把基本数据类型转换为字符串

String a=“java”; //声明一个String类型变量,赋值”java”

int b = 3; //声明一个int数据变量,赋值3

String c=a+b; /通过”+” 号连接,输出结果为java3

## 获取字符串信息

获取字符串长度

字符串名字.length()

String a=“123456789”; //声明一个String类型数据并赋值

int b=a.length(); //声明一个int数据类型,获取a的长度,输出结果为9

字符串查找

1. indexOf() 返回的是第一个被找到的值的索引

String a=“123456789”; //声明一个String数据

int b=a.intdexOf(“4”); //声明一个int变量,获取查找结果 注意要用双引号

System.out.println(b); //输出结果为3

1. lastIndexOf() 返回的值最后一个被找到的值的索引

String a=“12345678910”; //声明一个String数据

int b=a.lastIndexOf(“1”); //声明一个int变量,获取查找结果

System.out.println(); //输出结果为9

若没有查找到结果,则返回-1 若是查找字符 “ ,则返回的是字符串长度

获取指定索引位置的字符

charAt()

String a=“123456789”;

char b=a.charAt(5); //用char类型获取查找结果 查找索引6对应的字符

System.out.println(b); //输出结果为6

## 字符串操作

获取子字符串

substring() 截取指定索引后的字符串

String a=“123456789”;

String b=a.substring(5); //在a的索引5后截掉，获取5以后的字符串

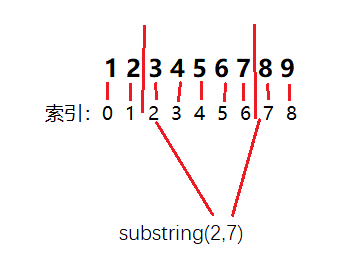
System.out.println(b); //输出结果为 789

同时,substring()可以截取一段指定的字符串

String a=“123456789”; //从a的索引2开始截取,同时在索引7前截取,然后取中间一段字符串

String b=a.substring(2,7); // 1 2 3 4 5 6 7 8 9

System.out.println(b); //输出结果为 34567



去除空格

trim()

String a=“1 2 3 456 78 9 “; //包含了许多空格

String b=a.trim(); //获取去除空格后的字符串

Sstem.out.println(b) //输出结果为 123456789

字符串替换

replace(char,char) 第一个是需要替换的字符,第二个是替换的字符

String a=“123456789”;

String b=a.replace(“1”,”0”); //将a中的字符1替换成”0”

System.out.println(b); //输出结果为 023456789

若有多个相同字符,则全部替换 区分大小写

判断字符串的开始和结尾

startsWith(String) 判断字符串开头与括号里的字符串是否相同

String a=“123456789”;

boolean b=a.startsWith(“12”); //判断a是不是以 “12” 开头 输出结果为true

endsWith(String) 判断字符串结尾与括号里的字符串是否相同

String a=“123456789”;

boolean b=a.endsWith(“12”); //判断a是不是以 “12” 结尾 输出结果为false

判断字符串是否相等

变量地址相同,对象是在堆内存开辟出来的,地址不同

“==“比较的是地址,所以

String a=“123456789”;

String b=new String(“123456789”);

boolean c=(a==b); //输出结果为false 由于地址不相等,所以结果为否

equals() 判断两个字符串长度和内容是否都相等

String a=“123456789”;

String b=new String(“123456789”);

boolean c=a.equals(b); //输出结果为true

equalsIgnoreCase() 在equals的基础上增加了忽略大小写

String a=“ABC”;

String b=“abc”;

boolean c=a. equalsIgnoreCase(b); //输出结果为true

括号内可以直接填字符串 “abc”

字母大小写转换

toLowerCase() 大写转小写

String a=“ABCdef”;

String b=a.toLowerCase(); //输出结果为”abcdef”

toUpperCase() 小写转大写

String a=“ABCdef”;

String b=a.toUpperCase(); //输出结果为”ABCEDF”

字符串分割

split(String) 分割字符串,String为分割的分割符

String a=“123.456.789”;

String[] b=a.split(“\\.”); //使用正则表达式表达 “.” 用 . 来分割字符串并使用数组保存

for(String c:a){ //遍历数组 输出结果为 [123] [456] [789]

System.out.print(“[“+c+”]”);

}

split(String,int) 同上,增加了int为限定分割次数

String a=“123.456.789”;

String[] b=a.split(“\\.”,2); //使用正则表达式表达 “.” 用 . 来分割字符串并使用数组保存

for(String c:a){ //遍历数组 输出结果为 [123}[456789]

System.out.print(“[“+c+”]”);

}

## 格式化字符串

将字符串按照指定格式进行转化 format方法必须使用对象.因此必须创建对象

日期格式化

Date d = new Date(); //创建Date对象

String a=String.format(“%te”,d); //用a接收d对象格式化后的结果 输出结果为 今天号数



时间格式化

用法同上.



常见的日期时间组合

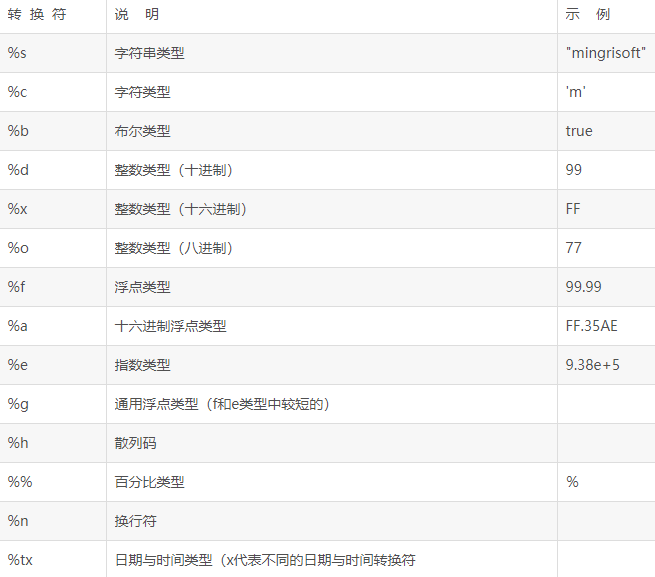


日期时间格式化要导入java.util.Date类

import java.util.Date;

常规类型格式化

String a = String.format(“%b”,3>5); //结果为false



## 正则表达式

用于判断语句,检查某一字符串是否满足某一格式.

相当于用正则表达式定义一个标准,判断某个字符串是否符合

String a = “\\d{6}”; //六个任意数字

String b = “\\d{8,10}@qq.com”;//八到十位数的QQ邮箱

String c = “\\w+@\\w+(\\.\\w{2,3})\*\\.\\w{2,3}”;

\\w+ //字母或数字出现异常或多次

@ //规定必须出现这个字符,可以是其他字符

(\\.\\w{2,3})\* //括号内的为自行定义的标准 一个点接上两到三个字母或数字 这个标准出现0~多次 也就是说可以不出现

\\.\\w{2,3} //一个点 接上 两到三个字母或数字 后面没有任何限定,那就必须出现

String j = “123456789@qq.com” // 创建一个字符串用于判断

boolean i = j.matches(b); //使用matches方法判断 用布尔变量接收判断值

## 字符串生成

StringBuilder是一个字符串对象，与String不同的是，StringBuilder不限制长度

String要扩容的话就需要加上新的Stirng实例，StringBuilder要扩容直接使用append

append(\*); 往StringBuilder中填入数据,包括基本类型和其他对象

StringBuilder a =new StringBuilder(“); //创建一个StringBuilder对象，命名为a

for(int i=0;i<=100;i++) { //用for循环100次

a.append(i); //使用append（）方法填入字符

} //输出a字符串的结果为123456….100

insert(int,\*); 在索引为int的位置后插入数据,包括基本类型或其他对象

StringBuilder a = new StringBuilder(“123456789”); //创建一个StringBuilder对象,并初始化

a.insert(5,”A”); //在字符串索引5的位置后插入”A”字符,索引必须大于等于0,小于等于字符串长度,输出结果为12345A6789

delete(int1,int2) 删除索引int1到int2中的所有字符

StringBuilder a = new StringBuilder(“123456789”);

a.delete(2,6); //删除包括2,6之前的所有字符,输出结果为 12789

# 数组

## 一维数组创建和使用

创建一维数组

1. 首先声明一个数组,任何为这个数组创建内存空间(对象)

int a[]; //数组的数据类型 数组名字[]

int[] a; //数组的数据类型[] 数组名字

数据类型可以是Java任何数据类型,它确定了该数组的元素类型

a = new int[5]; //为数组a创建内存空间,该数组元素有五个

1. 声明同时分配内存

int a[] = new int[5]; //数组类型 数组名字 [] = new 数组类型[数组元素数量]

初始化

直接初始化(填值)

int a[] = new int[]{1,2,3,4,5,6}; //用逗号隔开,大括号内是元素,中括号内为数量 不能在限制数量的同时初始化

一个个初始化(填值)

int a = new int[2];

a[0] = 1; //数组第一个元素的序号为0

a[1] =2;

如果先限制数组长度,再一个个赋值,若赋值次数大于数组长度,则报错,若次数小于长度,没有赋值的值为0/null(布尔类型为false)

使用一维数组

使用for循环读取

int a[]={0,1,2,3,4,5,6,7}; //创建数组

for(int i=0;i<a.length;i++){ //使用for数组一个个读取数组元素 length为数组长度 a.length为a数组的长度

System.out.println(a[i]); //输出时方括号内的数字为输出的元素索引

} //按照for方法执行顺序 判断i是否小于数组长度,然后执行输出语句a[i],也就是a数组的第i个元素

使用foreach循环读取

for(int i : a){ //变量的类型必须与数组的类型一样 中间那个是冒号

System.out.println(i); //输出时直接输出变量

}

## 二维数组的创建和使用

创建二维数组

方法1同一维数组

int a[][]; //声明一个二维数组

a = new int[2][4] //为这个数组分配空间 第一个方括号为这个数组内有两个小数组,第二个为每个小数组有4个元素

方法2同一维数组

int a[][] = new int[2][4]; //数组类型 数组名字 [] = new 数组类型[小数组数量][每个小数组内元素数量]

初始化

方法同一维数组

int a[][] = {{1,2,3,4},{5,6,7,8}}; //第一个括号为第一个小数组的元素,第二个为第二个小数组的元素

方法同一维数组

int a[][] = new int [2][2];

a[0][0] = 1; //第一个方括号为小数组的序号,从0开始

a[0][1] = 2; //第二个方括号为小数组的第几个元素,从0开始

a[1][0] = 3; //输出结果为 第一个小数组: 12

a[1][1] = 4; // 34

使用二维数组

方法同一维数组,多了一层for嵌套

for(int i=0;i<a.lenght;i++){

for(int j=0;j<a[i].lenght;j++){

System.out.println(a[i][j]);

}

}

同时，也可以使用foreach

for (int b[] : a) { //用b数组接收a数组的小数组

for (int c : b) { //用c接收b接收a数组的小数组中的元素

System.out.print+(c); //输出c

}

System.out.println(); //分割每个数组，便于观看

}

## 数组的基本操作

遍历数组

上文提到。遍历数组就是挨个读取数组里的每个元素。

使用for或者foreach遍历

替换数组元素

数组初始化后,可以通过.Arrays类的静态方法fill()对数组中的元素进行替换.

Arrays.fill(数组名字,要替换的值) //名字不用带方括号 值的类型必须和数组的类型相同

int a[] = new int[]{1,2,3,4,5,6};

Arrays.fill(a,0); //替换是全部替换

for(int i:a){

System.out.print(i); //输出结果为 888888

}

Arrays.fill( 数组名字 , 开始位置 (包括) , 结束位置(不包括) , 要替换的值) //选择性替换

int a[] = new int[]{1,2,3,4,5,6};

Arrays.fill(a,2,4,0); //替换是全部替换

for(int i:a){

System.out.print(i); //输出结果为 120056

}

索引位置不能大于数组长度,否则报错

数组排序

Arrays.sort(数组名字) 对数组进排序,整型数组从小到大,String类根据字典编排顺序

int a[] ={5,9,23,4,48}; //声明数组

Arrays.sort(a); //排序数组

for (int i = 0; i < a.length; i++) { //for循环输出

System.out.print(a[i]+”,”); //输出结果为 4,5,9,23,48

}

复制数组

Arrays.copyOf(数组名字 , 要复制的长度) 复制这个数组的一定长度的值 可以大于被复制的数组长度

Arrays.copyOfRange(数组名字 , 起始索引 , 结束索引) 起始索引必须小于被复制的数组长度,结束索引可以大于被复制数组的长度

int a[] = {1,2,3,4,5,6,7,8,9};

int one[] = Arrays.copyOf(a,5); //复制a数组中的前五个值到one数组 输出结果为 12345

int tow[] = Arrays.copyOfRange(a,2,15); //复制a数组中的索引2到15的值到tow数组 结果为3456789000000

//注意:索引从0开始 超过长度的就用0代替,char类型和引用类型就用null代替

数组查询

Arrays.binarySearch(数组名字 , 查询的值)

Arrays.binarySearch(数组名字 , 开始的索引 , 结束的索引 , 查询的值)

查询数组中有没有这个值,有则返回这个值的索引,没有则返回这个值应该在的索引位置的负数-1

int a[] = new int[]{1,2,3,4,6,7,8,9,};

Arrays.sort(a); //必须排序后才查询

int b = Arrays.binarySearch(a,0,3,5); //在索引0到3找5

//如果规定的范围索引小于值应该在的索引(排序后从大到小),则返回范围结束索引的负数-1

System.out.println(b); //返回的就是结束索引3的负数-1 输出结果-4

int c = Arrays.binarySearch(a,5); //5应该在索引4,没有找到就转为负数后-1 输出结果 -5

int d = Arrays.binarySearch(a,12); //查询的值在数组范围外(数组1-9,查12),就返回数组最后一个值的索引负数-1

//输出结果为-9

所以判断数组中有没有这个值,可以直接判断返回值是否 >=0

## 数组排序算法

冒泡排序

比较相邻的两个元素的大小(通常从索引0开始)

大的往后排,小的往前排(可以转换方向:前大后小)

循环判断,一次遍历后,最大(小)的值就在最下面,再次遍历,最大(小)的值就在倒数第二,以此类推

int a[] = new int[]{}; //假设排序a数组

for(int i=0;i<a.length;i++){ //遍历数组

for (int j = 0; j < a.length-1-i; j++) { //排序次数是数组的长度-已经排序的次数(i)-1

if(a[j]>a[j+1]) { //如果a[j]>a[j+1] 也就是相邻比较

int x=a[j]; //利用一个x变量实现轮换

a[j]=a[j+1];

a[j+1]=x;

}

}

}

直接选择排序

顾名思义,直接选择最大(小)的值,放在最后,然后再选择最大(小)的值,放在倒数第二,以此类推

实现此方法,需要一个数组遍历判断大小,选出最大值放后面,然后逐层递减

int a[] = new int[]{}; //创建一个int类型数组

int x; //用一个int变量记录每次选中的最大值的索引

for(int i=1 ; i<a.length ; i++){ //遍历数组,也就是排序的次数

x=0; //初始化x,每次大循环清空

for(int j=1 ; j<=a.length-i ; j++){ //每选中一个最大值,for循环少一次

if(a[j]>a[x]){ //for在循环 当数组中 j(索引的值} 比 x(索引的值) 大时

x=j; //遇到大的,就记住这个索引

} //最后,这个x就记住数组中最大值的索引

}

int y = a[a.length-i]; //进行变量轮换

a[a.length-i] = a[x]; //每次大的for循环,都会选出一个最大值,这个最大值放在最后面

a[x] = y; //所以就是a(数组长度) – i(已经循环的次数)

}

反转排序

把整个数组反过来(虽然没什么用)

int a[] = new int[]{}; //创建一个int类型数组

int b=a.length; //获取数组长度

for(int i=0 ; i<b/2 ; i++){ //由于只是换位置,循环次数只是数组长度的一半

int x = a[b-i-1]; //变量轮换 数组长度-1就是最后一个元素的索引(第一个索引是0)

a[b-i--1] = a[i];

a[i] = x;

}

# 类与对象

## 面向对象

对象

对象就是一个小盒子，里面装着各种属性和行为

类

一个大盒子，里面装着小盒子

封装

可是小盒子想装进大盒子，就要拿胶带封住

继承

每个大盒子都堆放在一起，当A大盒子中有一个a小盒子中有一个属性或行为，可以应用在B大盒子里的b小盒子时。

他们可以共享。

甚至盒子也可以共享

多态

大盒子里的a小盒子，有一个属性或方法跟同在大盒子里的b小盒子相似，就可以使用a小盒子里的属性跟方法

## 类

成员变量

每个变量都有生命周期

for、switch里创建的变量，生命周期只有在循环进行时有效，循环结束就没了。

每个方法(包括主方法main)中创建的变量，生命周期只有方法执行时有效，方法结束后就没了。

而成员变量，存活在程序运行时，程序结束才没了。

定义方法如[2-1主类结构](#_主类结构)下的成员方法

在方法外定义，并带上权限修饰符。

权限修饰符 变量类型 变量名字 = 值 ; //创建时可以不赋值

public int a; //创建一个公有的int变量a,不赋值

成员方法

定义在类中,和成员变量一样

变量是一个值,方法则是一系列操作.

权限修饰符 返回值类型 方法名 (参数类型 参数名){ //不用返回值时填void 参数可以填多个,用逗号隔开

方法语句 //参数是方法中用到的变量,可以说方法是一个机器,参数是倒入的材料,没有材料的机器无法正常工作

return 返回值; //当材料加工好时,可以选择拿出来,或者直接在机器里输出

}

pubilic int number(int a,int b){ //定义一个方法,名字是number 返回值是int类型 参数是int a和int b

int c=a+b; //方法体是a+b然后赋值给c

return c; //返回c

}

成员变量尽量不要和方法内的局部变量同名

权限修饰符

|  |  |  |  |
| --- | --- | --- | --- |
| 可以访问的位置 | Private | Protected | Public |
| 本类 | 可见 | 可见 | 可见 |
| 本包以下所有类 | 不可见 | 可见 | 可见 |
| 其他包的类 | 不可见 | 不可见 | 可见 |

优先读取类的权限,如果类的权限过高,类中的成员方法权限低,则会照类的权限读取

包也是如此

privte a; //包的权限是最高

class b{

public void c{ //方法的权限最低,可还是按照包的权限来读取

方法体

}

局部变量

在方法中定义的变量,就是局部变量.

使用时必须赋值,否则报错.

有效范围看大括号

{ //a的有效范围

int a;

{ //b的有效范围

int b;

} //b的有效范围

} //a的有效范围

this

行参：形式上的参数

实参：实际的参数

当类中的成员变量想在成员方法里使用，就要用到this

this会隐藏，当变量名字相同时，用this来区分哪个是成员变量

public int b; //成员变量

pubilc viod a(int b){ //成员方法

this.b=b; //左边的b是本类中的成员变量 右边的b是形参

} //这个方法就是给成员变量b赋值 值为使用这个方法时输入的b值(小括号里的)

构造方法

与类名相同的方法，作用是为成员变量初始化

每当实例化(new)一个类中对象时,都会调用该类的构造方法

权限修饰符 类名一致的方法名 (参数列表) [异常] {方法体}

public class note { //创建一个类

private int a; //

private String b; //创建两个成员变量

public note(){ //构造方法,赋值

a=2;

b=“这是二”;

}

public void modify() { //创建一个方法重新赋值

a=3;

b=“这是三”;

}

public static void main(String[] args) { //main方法

note n = new note(); //实例化对象

System.out.println(n.a+n.b); //输出值 输出结果 2这是二

n.modify(); //调用第二次赋值方法

System.out.println(n.a+n.b); //输出值 输出结果 3这是三

note n1 = new note(); //重新实例化对象

System.out.println(n1.a+n1.b); //输出值 输出结果 2这是二

}

}

初学时，没有写构造方法也可以实例化对象，因为系统会自动生成一个无参的构造方法，方法里什么都不做

自己定义了一个构造方法之后，程序就不会自动生成无参的构造方法

想要实例化对象时不给予参数，就要另外加个无参构造方法

建议每次自行定义构造方法后，添加一个无参构造方法，方法可以不执行语句

public class Test{ //Test类

int a; //成员变量a

public Test(){ //定义一个无参构造方法

System.out.println(“无参构造方法”)；

}

public Test(int a){ //定义一个有参构造方法

System.out.println(“有参构造方法”);

this.a=a;

}

public static void main(String []args){

Test test = new Test(); //实例化对象,不给参数,所以调用的是无参构造方法

Test test1 = new Test(1); //实例化对象,给int类型参数,调用的是有参构造方法

} //构造方法可以创建多个,具体看下面的方法重载,这个实例也用了方法重载

}

静态变量,常量和方法

静态(static),静态成员属于类所有,区别于个别对象,可以在本类或其他类中使用

类名 + “.” 调用 静态变量是共享的,被调用后赋值会一直保留,下次另一个对象调用时,值不会被初始化

public class Test{ //在权限修饰符和类型中间加static

final static int a=1; //定义静态常量

static int b=2; //定义静态变量

public static void c(){ //定义静态方法

方法体;

}

public static void main(String[] args){ //主方法

System.out.println(Test.a+Test.b); //输出结果 12

Test.c;

}

}

## 方法

主方法

类的入口点,定义程序执行顺序

public static void (String []args){ //主方法,也称main方法

}

方法重载

Overload

类中的一组方法，相同的方法名字，不同的参数列表,也就是说一个对象有多个同名方法,根据给定参数不同调用不同方法

不同于普通方法：传入参数的个数、类型、顺序

调用方法时，先找 方法名 方法名一样就找对应的参数类型

决定方法重载的要素 方法名 参数类型

public class Test{

public void a{ //创建方法,没有传递参数

System.out.println(“没有参数”);

}

public void a(int i){ //创建方法,名字相同,需要传递int类型参数

System.out.println(“参数为”+i);

}

public void a(String s){ //创建方法,名字相同,需要传递String类型参数

System.out.println(“参数为”+s)

}

public static void main(String []args){

Test t =new Test(); //创建对象

t.a(); //调用a方法，不给参数

t.a(1); //调用a方法 给int参数

t.a(“参数”); //调用a方法 给String参数

}

}

方法重载为程序实现了灵活性,JDK1.5版本出现了一个新的写法:

动态参数列表

public class Test{

public void a(int…a){ //创建方法,需要传递int类型动态列表参数

System.out.println(“参数为”);

for(int i=0 , i<=a.length , i++){ //遍历所有的参数

System.out.print(a[i]);

}

}

public static void main(String []args){

Test t =new Test(); //创建对象

t.a(1,2,3,4,5,6,7); //调用a方法 给int参数 运行结果为 参数为1234567

t.a(); //当a方法没有空参数的选项时,可以调用动态参数列表 运行结果为 参数为

} //因为动态参数的个数为0~n个 空参也就是0个参数 与空参方法冲突报错

}

方法可以调用多个参数,重载也可以,这时就要考虑参数的调用顺序,调用顺序决定调用的方法

public class Test{

public void a(int a,String b){ //创建方法,需要传递int和String类型的参数

System.out.println("参数为”+a+b);

}

public void a(String a,int b){

System.out.println(“参数为”+a+b); //创建方法,需要传递String和int类型的参数

}

public static void main(String []args){

Test t =new Test(); //创建对象

t.a(1,”参数”); //调用a方法 先给int再给String 运行结果为 参数为1参数

t.a(“参数”,1); //调用a方法 先给String再给int 运行结果为 参数为参数1

} //因为动态参数的个数为0~n个 空参也就是0个参数 与空参方法冲突报错

}

动态参数列表也要考虑参数顺序

因为动态是0~n个参数,所以用动态参数列表是不能在后面加参数

public void a(int…a,Stirng b){ //错误写法,只能在前面加,后面不能加 (String a,int…b)

}

方法递归

方法的自我调用

public class a {

public static void main(String[] args){

b; //引用对象，本类的静态对象不用 .

}

public static b(){ //静态方法

System.out.println(“递归进行”); //提示文字

b; //方法递归

System.out.println(“递归结束”); //提示文字

} //这时方法会不断地进行,直到栈内存溢出

} //递归的用处很多，像循环结构一样，给定一个满足条件终止运行与·· 即可

代码块

和普通的方法一样，区别为

无修饰符 无参数 无返回值 无名字

在类中定义，不在方法中

{

System.out.println(“abc”);

}

系统自行调用,在调用构造方法之前调用,也就是new一个对象是调用

创建对象时,做了什么事: new一个对象 >> 执行代码块 >> 执行构造方法

代码块没有方法重载的概念,因为没有参数,无法区别,但可以定义多个块,按顺序执行

public class Test{

public int a;

{

System.out.println(“第一个代码块”); //代码块

}

{System.out.println(“第二个代码块”);} //节省空间

{System.out.println(“第三个代码块”);} //三个代码块

public Test(){ //无参构造方法

System.out.println(“无参构造方法”);

{System.out.println(“无参构造方法”);} //构造方法里的代码块

}

public Test(int a){ //有参构造方法

this.a=a;

{System.out.println(“有参构造方法”);} //有参构造方法里的代码块

}

public static void main(String []args) { //主方法

Test t = new Test(); //创建对象 不传递参数 调用无参构造方法

Test t1 = new Test(1); //创建对象 传递参数 调用有参构造方法

System.out.println(t.a); //输出a

} //运行结果都会显示一二三代码块,无参就运行无参构造方法里的代码,有参就调用有参构造方法的代码

## 对象

对象是类的一个引用

创建对象

在栈内存中开辟一个空间，然后引用存储区的方法区中的类

相对于把类复制一份放到栈内存中,名字就是对象的名字

Scanner s= new Scanner(System.in); //类名.对象名 = new 类名();

使用对象

实例化一个对象后,可以使用 类名.类成员 来调用,类成员可以是方法,也可以是变量或属性.

public class Test{ //创建类

static int a = 0; //静态成员变量

int b = 0; //成员变量

public static void main(String []args){ //主方法

Test t1 = new Test(); //第一个实例化对象

Test t2 = new Test(); //第二个实例化对象

t2.b=1; //给成员变量赋值

System.out.println(t1.b+t2.b); //输出两个对象的a值 t1的b是0 t2的b是1

t2.a=2; //给静态成员变量赋值

System.out.println(t1.a+t2.a); //输出两个对象的b值 t1的a是2 t2的a是2

} //静态变量在实例化对象后被赋值,会一直保持不变,全员共享该值

}

对象的引用

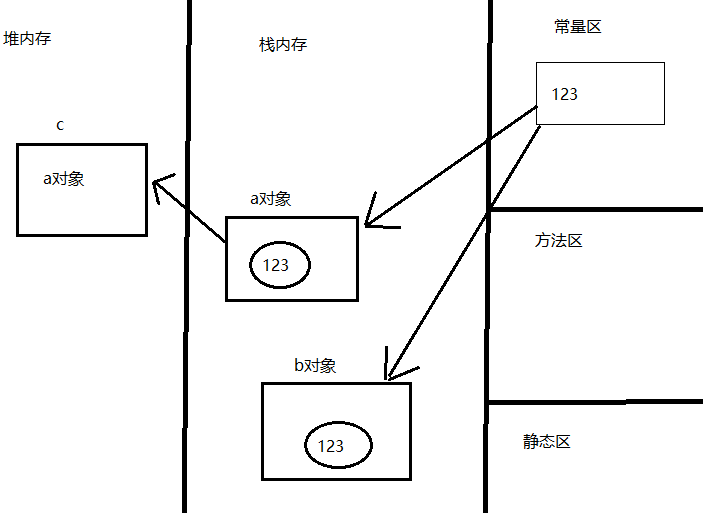
引用只是引用对象的地址,而不是存放一个对象

Test t; //引用对象

Test t = new Test(); //实例化一个对象

对象的比较

比较对象的内容

String a = new String(“123”);

String b = new String(“123”); //a和b赋值相同的内容

String c = a; //c引用a的内容

boolean judge = (b==c); //比较bc的地址是否相等 false

boolean judge2 = b.equals(c) //比较bc的值是否相等 true

//创建对象时，从常量区拿123放到栈内存的空间里，a和b各取一个

//c对象的创建在堆内存中，程序自行开空间，c中装的是a对象的地址

//a和b是两个互不相干的对象,虽然值相同,但地址不同

//所以比较地址时返回false,比较值时返回true

对象销毁

对象超过作用范围,会被销毁

对象赋值null,会被销毁

{ //对象存活范围

Book book = new Book();

} //对象存活范围

Book book = new Bokk();

book=null; //赋值null,销毁

# 包装类

Number的子类

包装类可以将基本数据类型封装成一个类，使基本数据类型可以使用方法

上面说到的String类，有很多方法可以使用，包装类也可以使基本数据类型有很多方法使用

八个基本数据类型，分别有八个包装类

需要注意的是，方法用于对象(对象.方法名)，常量是用于类的(类名.常量)

## Integer、Byte、Short、long(整数类型)

整数类型唯一的区别就是最大值和最小值的上限，所以包装类的方法都一样，以下用Integer举例

若要用其他数据类型，只需类名即可

装和拆

上面说到，包装类就是把数据类型包装成对象，也可以说是构造方法

Integer a = new Integer(1); //这个时候a就变成了一个对象,对象里装一个int类型的值 装箱

int b = a.intValue(); //把a对象的值赋给变量b 拆箱

Integer c = new Integer(“25”); //可以使用String类型变量来创建对象

//必须是数值型String变量,否则抛出NumberFormatException(数字格式异常)异常

可是在Java 9后，java提供了自动包装的方法

之前的方法会被弃用提示警告,并且Integer会被画上横线,但仍然可以使用

The constructor Integer(String) is deprecated since version 9

这个构造函数 整数（字符串）已弃用，因为版本9

Integer a=5; //直接将5赋值给a,而a是一个Integer类的属性,和声明变量有点像

常量

用于获取该数据类型的常量

|  |  |  |  |
| --- | --- | --- | --- |
| MIN\_VALUE | 返回该数据类型的最大值 | SIZE | 返回比特长度（多少位） |
| MAX\_VALUE | 返回该数据类型的最小值 | TYPE | 返回类型 |

int a = Integer.MIN\_VALUE; //-2147483648 这里的int是用来获取返回值的

int b = Integer.MAX\_VALUE; //2147483647

int c = Integer.SIZE; //32

int类型的最大值和最小值，位长

常用方法

下列方法用于装好箱的包装对象

byteValue(); intValue(); shortValue(); longValue();

这四个方法，可以将整数类型的包装对象转为任意类型的返回值

Integer a = 5； //创建一个int对象a

byte b = a.byteValue(); //将a对象转换成byte类型的值返回给b

toString 将int对象转换成String类型返回值

String c = a.toString(); //将a对象转换成String类型的值返回给c

另外,toString还可以转换进制

String c2 = a.toBinaryString(); //二进制

String c16 = a.toHexString(); //十六进制

String c8 =a.toOctalString(); //八进制

equals() 对比对象是否相等,上文提到过对象的比较

valueOf(String) 返回该String值的整数包装类对象

parseInt(String) 将String转为int类型

## Boolean(布尔类型)

装和拆

Boolean a = new Boolean(true); //装

boolean a1 = a.booleanValue(); //拆

同样，Java9版本后给出了自动装箱的方法

Boolean b = false ;

手动装箱仍然可以使用，但建议使用自动装箱

另外，用String创建对象的话，String的值在忽略大小写的情况下等于true，则分配true，否则分配false

Boolean c = new Boolean(“ture”); //这时c的值为true

Boolean d = new Boolean(“ok”); //false

常量

TRUE 对应基值的true的Boolean对象

FALSE 对应基值的false的Boolean对象

TYPE 基本类型boolean的Class对象

常用方法

booleanValue() //将Boolean对象的值以对应的boolean值返回

equals() //比较，当参数不是null且boolean值相同才返回true

parseBoolean(String) //将字符串参数解析为boolean值

toString //返回该boolean值代表的String对象

valueOf(String) //返回一个用指定的字符串表示值的boolean值

## Character(字符类型)

装和拆

Cheracter a = new Cheracter(‘a’); //装

Cheracter a1 = a.cherValue(); //拆

自动装箱

Cheracter b = ‘b’;

常量

CONNECTOR\_PUNCTUATION 返回byte值，表示Unicode规范中的常规类别

UNASSIGNED

TITLECASE\_LETTER

常用方法

char i = a.cherValue(); //拆箱,返回char值

int i= a.compareTo(b); //比较a对象和b对象char值是否相等,相等返回0,不相等返回的是b与a的Unicode表的排名差

boolean i = a.equals(b); //比较对象

char i = Character.toUpperCase(a); //转大写,不是字母就不转

char i = Character.toLowerCase(a); //小写

boolean i=Character.isUpperCase(a); //判断是否为大写字符,大写返回true,不是大写或其他字符返回false

boolean i= Character.isLowerCase(a); //判断小写,同上

boolean i= Character.isLetter(a); //是否是一个字母

boolean i= Character.isDigit(a); //是否是一个数字字符

boolean i= Character.isWhitespace(a); //是否是一个空白字符

String i = a.toString(); //转字符串，字符串的长度仅为1

## Double、Float(浮点类型)

装和拆

Double a = new Double(3.14); //装

Double b = 3.14; //自动装

double a1 = a.doubleValue(); //拆

常量

int i = Double.MAX\_EXPONENT; //有限double变量的最大指数

int i = Double.MIN\_EXPONENT; //有限double变量的最小指数

double i = Double.NEGATIVE\_INFINITY; //double负无穷大的常量

double i = Double.POSITIVE\_INFINITY; //double正无穷大的常量

常用方法

byte i = a.byteValue(); //强制转换为byte类型变量

int i = a.intValue(); //转换为int

double i=a.doubleValue(); //转为double

long i = a..longValue(); //转为long00

boolean i = a.equals(b); //比较对象

int i = a.compareTo(b) //比较值大小,相等返回0,a小于b返回负值,a大于b返回正值

Syting i=a.toString(); //转String

## Number

作为包装类的父类,Number类的方法完全被子类们继承

也就是说Number类的方法,在上述子类中都包含有,总结一下包装类的通用方法(所有基本类型都可以用)

byteValue(); //返回byte值

intValue(); //返回int值

floatValue(); //返回float值

shortValue(); //返回short值

longValue(); //返回long值

doubleValue(); //返回double值

基本类型包装成对象之后,可以进行更多的操作,也是进一步说明了面向对象的编程思想

万物皆可对象

# 数字处理类

顾名思义,本类就是用来处理数字

## 数字格式化

DecimalForamt作为数字格式化类,主要针对double和folat,且只是十进制格式化类

格式化的意思,并不是删除,而是将这个数字(double)为一个新的形式(String)显示

首先是定义模板,模板的概念在正则表达式中提过

正则表达式是定义一个判断的模板,然后判断这个字符串是否对应模板

DecimalFormat a = new DecimalFormat(“0000”); //定义模板对象

DecimalFormat b = new DecimalFormat(); //先创建对象

b.applyPattern(“0000”); //再定义模板

定义模板时，需要知道这些符号是什么意思，一个符号代表一个数字或字符

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 数字 | 代表数字，没有数字就填0 | # | 数字 | 代表数字，没有数字就不填 |
| .（点） | 数字 | 代表小数点 | , | 数字 | 分组分隔符 |
| % | 前缀或后缀 | 将数字乘以100加个百分号 | - | 数字 | 代表负号 |
| \u2030 | 前缀或后缀 | 将数字乘以1000加个千分号 | ` | 上逗号 | 转义符,用于输出有意义的符号 |

DecimalFormat a = new DecimalFormat("0,0,0,0,0"); //定义模板，每个数字用逗号分隔

int i = 123456789; //创建变量，用来格式化

System.out.println(a.format(i)); //使用format方法进行格式化 输出结果为1,2,3,4,5,6,7,8,9

格式化后,值的类型变成字符串

DecimalFormat a = new DecimalFormat(); //创建对象

a.applyPattern("000000.00kg"); //定义模板

double i = 11111.111; //创建变量

System.out.println(a.format(i)); //使用format方法格式化 输出结果为0011111.11kg

灵活使用0和#，逗号就像金额的分隔符一样，小数点前后的位数要记清楚

另外，可以通过setGroupingSize(int)来对值进行分组，需要注意，分组对小数点后的数字无效

还有setGroupingUsed(boolean)来设置是否允许分组

DecimalFormat a = new DecimalFormat(); //创建对象

a.setGroupingSize(2); //定义模板，使用方法，每个两个数字分隔一次

double i = 121212.1212; //创建变量

System.out.println(a.format(i)); //输出结果为12,12,12.121

a.setGroupingUsed(false); //不允许分组

System.out.println(a.format(i)); //输出结果为121212.121

## 数字运算

Math类在Java.lang包里 ,不需要引用

Math类全是静态方法,不用实例化对象

也就是说,直接使用 Math.方法 就可以使用了

常用的常量有π(PI),e(E)

System.out.println(Math.PI); //π就是圆周率输出结果 3.141592653589793

System.out.println(Math.E); //e就是无理数输出结果 2.718281828459045

最大(小)值、绝对值

参数什么类型,返回值什么类型

Math.max(a,b); //返回两个数中最大值

Math.min(a,b); //返回两个数中最小值

Math.abs(a); //返回a的绝对值

三角函数

以下方法返回值皆为double，建议参数也为double值，因为三角函数本来就是求近似值

Math.toDegrees(double); //弧度转角度

Math.toRadians(double); //角度转弧度

Math.sins(double); //正弦值,括号里输入的是角度数,可以用π代表,90°就是Math.PI/2

Math.asin(double); //反正弦值

Math.cos(double); //余弦值

Math.acos(double); //反余弦值

Math.tan(double); //正切值

Math.atan(double); //反正切值

Math.sinh(double); //双曲正弦值

Math.cosh(double); //双曲余弦值

Math.tanh(double); //双曲正切值

Math.atan2(double y, double x); //计算定点与极坐标的弧度

指函数

以下参数为double值,返回值可以是其他数字类型,不过可能会丢失小数点导致精度不足

Math.pow(double a, double b); //a的b次方

Math.exp(double a); //自然常数e的a次幂

Math.sqrt(double a); //a的平方根

Math.cbrt(double a); //a的立方根

Math.log(double a); //自然对数

Math.log10(double a); //取底数为10的对数

取整函数

Math.ceil(double a); //返回大于等于参数的最小整数 假如参数为2.1,返回值为3,向上取整

Math.floor(double a); //返回小于等于参数的最大整数 假如参数为2.9,返回值为2,向下取整

Math.rint(double a); //返回最接近的整数,如果两边的距离相等,取偶数

//比如参数为2.5,距离2和3相等,取2偶数

Math.round(float a); //四舍五入,只适用于十分位小数 假如参数为2.51,返回值为3

//原理是将2.51+0.5后向下取整 2.5+0.5=3 向下取整为3

// 2.4+0.5=2.9 向下取整为2

// 2.51+0.5=3.01 向下取整为3 不会判断百分位的那个小数

## 随机数

Math.random()方法

这个方法只返回0~1之间的随机数(包括0)

但在这个方法上加一些运算,就可以返回随意范围的随机数

假如条件是a到b之间的随机整数 大于等于a 小于b

int i = (int)a+ (int)(Math.random()\*(b-a));

//用int接收返回值 加a用来增加最小值(转int类型) 乘以随机范围,得到的随机数就是这个范围(转int类型)

//最小值+范围就是 a~b的随机数 如果在(b-a)中+1 返回的的随机数既包括a,也包括b

其实,rendom方法也可以用于char类型

把char类型的值看做Unicode值，也可以用随机数返回一个给定范围的字符

char i = (char)(a+Math.random()\*(b-a));

Random类

通过时间类生成随机数,如果运行太快,随机数可能会相同

因为这是个类,所以要实例化才能使用,实例化前导包

import java.util.Random; //导包

Random r = new Random(); //创建对象

int a1 = r.nextInt(); //一个int类型的随机数

System.out.println(a1); // 838401523

int a2 = r.nextInt(50); //0~50的随机数

System.out.println(a2); //18

long a3 = r.nextLong(); //一个long类型的随机数

System.out.println(a3); //-4108228843719486312

boolean a4 = r.nextBoolean(); //一个boolaen类型的随机值

System.out.println(a4); //true

float a5 = r.nextFloat(); //一个folat类型的随机数

System.out.println(a5); // 0.34101087

double a6 = r.nextDouble(); //一个double类型的随机数

System.out.println(a6); // 0.34604344761171424

## 大数字运算

int和long类型都有上限，满足不了大数字的运算

BigInteger类和BigDecimal类分别满足了大整数和大小数的管理和运算

它们支持任意精度的值(任意大小)

BigInteger

和其他的数字类一样,先创建对象

BigInteger big = new BigInteger(“99”); //直接用字符串代表数字,适用String

BigInteger big2 = BigInteger.valueOf(999L); //这种方法适用long类型数值

String s = "99"; //当然,也可以放入变量

BigInteger big = new BigInteger(s);

long l = 456L;

BigInteger big2 = BigInteger.valueOf(l);

然后就是各种运算方法,使用上面创建的对象

运算所使用的数也必须是大数字,不能是基本类型

System.out.println("加法"+big.add(big2)); //加法101

System.out.println("减法"+big.subtract(big2)); //减法97

System.out.println("乘法"+big.multiply(big2)); //乘法198

System.out.println("除法"+big.divide(big2)); //除法49

System.out.println("取余"+big.remainder(big2)); //取余1

//数组取商余

System.out.println("数组取商"+big.divideAndRemainder(big2)[0]); //数组取商49

System.out.println("数组取余"+big.divideAndRemainder(big2)[1]); //数组去余1

//开方和相反数

System.out.println("开二次方"+big.pow(2)); //开二次方9801开二次方用的是自己乘自己,可以不用大数字代表次方

System.out.println("相反数"+big.negate()); //相反数-99 相反数不用参数

//位移

System.out.println("左移"+big.shiftLeft(2)); //左移396 值为负数时右移

System.out.println("右移"+big.shiftRight(2)); //右移24 值为负数时左移

//逻辑操作

System.out.println("与操作"+big.and(big2)); //big和big2做与操作 与操作2

System.out.println("或操作"+big.or(big2)); //big和big2做或操作 或操作99

//比较

System.out.println("比较"+big.compareTo(big2)); //比较1

//-1代表big<big1 0代表相等 1代表big>big2 99>2返回1

System.out.println("最小值"+big.min(big2)); //最小值2

System.out.println("最大值"+big.max(big2)); //最大值99

System.out.println("判断相等"+big.equals(big2)); //判断相等false

//其他

System.out.println("公约数"+big.gcd(big2)); //公约数1

因为继承与Number,包装类的值类型转换在这里也可以用

byteValue(); //返回byte值

intValue(); //返回int值

floatValue(); //返回float值

shortValue(); //返回short值

longValue(); //返回long值

doubleValue(); //返回double值

//如果想不丢失精度的情况下

intValueExact(); //准确返回int类型值,如果装不下,抛出ArithmeticException(算术异常)异常

longValueExact(); //准确返回long类型值,同上

BigDecimal

还是创建对象,传递参数

BigDecimal d1= new BigDecimal("2.5"); //String类型参数

BigDecimal d2 = new BigDecimal(3.5); //double类型参数

BigDecimal d3 = BigDecimal.valueOf(4.5); //使用方法传递

和上面的BigInteger一样,可以使用变量

运算方法,其他和BigInteger差不多唯独除法需要传递多个值

System.out.println("加法"+d1.add(d2)); //加法6.0

System.out.println("减法"+d1.subtract(d2)); //减法-1.0

System.out.println("乘法"+d1.multiply(d2)); //乘法8.75

System.out.println("除数"+d1.divide(d2,5,BigDecimal.ROUND\_HALF\_UP));

//d1除以d2,保留五个小数,后面四舍五入 除数0.71429

//第一个是除数 第一个是保留的小数点后位数 第三个是对最后面那个商值进行处理



# 接口、继承、多态

## 继承

继承面向对象的一个重要属性，它使程序更有弹性

继承父类

子类可以继承父类的所有方法和属性

使用extends关键字继承

public class One{ //创建一个类

piblic void a(){

System.out.println(“我是父类123”)

}

}

public class Tow extends One{ //再创建一个类,使用extends继承上面那个类

public static void main(String[] args) { //主方法

Inherit i = new Inherit(); //创建对象

i.a(); //调用方法 输出结果为 我是父类123

//可以看到这个类只有一个主方法,输出的是继承于One类的方法

//可以说把父类(One)的所有方法复制一份给子类(Tow)

}

}

重写方法

当子类想修改父类的方法时，可以重写方法

public class One{ //创建一个类

public void a(){ //类中有个方法 a

System.out.println(“我是父类123”)

}

}

public class Tow extends One{ //再创建一个类,使用extends继承上面那个类

public void a(){ //直接重写一个方法 a

System.out.println(“我是子类456”)

}

public static void main(String[] args) { //主方法

Inherit i = new Inherit(); //创建对象

i.a(); //调用方法 输出结果为 我是子类456

//如果像上次一样,本来应该输出父类方法,可是子类进行了方法重写,所以输出的就是子类的代码

//构造方法,属性也可以进行重写

//当重写方法时,如果父类的方法权限为中级,那么子类的重写方法不能是高级

//也就是说,重写方法的权限只能 不变 或 越来越公开 ,不能越来越收缩权限范围

}

}

高级重写

当重写时,子类的方法会覆盖父类的方法

如果想保留父类的方法,就要用到 super()

super()的作用就是调用父类的方法,有点类似this关键字

只不过this关键字调用的是本类成员属性方法 super()调用的是父类属性方法

public class One{ //创建一个类

public void a(){ //类中有个方法 a

System.out.println(“我是父类123”)

}

}

public class Tow extends One{ //再创建一个类,使用extends继承上面那个类

public void a(){ //直接重写一个方法 a

super.a(); //super调用父类a方法

System.out.println(“我是子类456”) //外加子类代码

}

public static void main(String[] args) { //主方法

Inherit i = new Inherit(); //创建对象

i.a(); //调用方法 输出结果为 我是父类123

// 我是子类456

}

}

另外，如果父类的方法需要参数，使用super调用时也要传参

使用父类方法时也要传参，除非重写

## Object类

用法

Object是所有类的父类，每个类都源于Object类

所以可以使用某些不是本类的方法，如equals(),toString()

public class a{ ---> public class a extends Object{

//继承Object类不用写出来

} }

方法

Object类的方法有

getClass(); //获取当前对象对应的类实例

getname(); //结合gerClass方法获取当前类名

class a{} //先定义一个类

public class Test\_Object { //再定义一个类

public static void main(String[] args) { //主方法

a i =new a(); //创建对象

System.out.println(i.getClass().getName()); //获取a类的包名.类名 输出结果 core.a

}

}

toString(); //上面说过，包装类的类型转换，把基本类型转为String

equals(); //上面说过，对象的比较

由于继承于Object类，所以有些方法可以进行重写

有些方法不能重写，因为设置了final(常量)

## 对象类型转换

向上转型

子类型转父类型

简单来说，就是父类引用指向子类对象

//定义一个a类

class a{

public void test(){

System.out.println("我是父类"); //说明这是父类

}

}

//定义一个b类，继承a类

class b extends a{

public void test() {

System.out.println("我是子类"); //说明这是子类

}

public void open() {

System.out.println("打开子类"); //子类的第二个方法

}

}

//定义一个c类，放置主方法

class c{

public static void main(String[] args) {

a i = new b(); //父类的引用指向子类对象，有点像类型转换，将a类型的i转为b类型

i.test(); //调用两个类都存在的方法test 输出结果为 我是子类

i.open(); //调用只存在于b(子类)的方法 报错：没有为类型a定义open方法

//也就说，向上转型的代价就是丢失父类没有的方法 而子类中重写的方法会生效

//用父类的名字，用子类的代码

//这样做的好处就是，如果有多个子类继承这个父类，只需要向上转型即可同时使用父类所有方法

}

}

向下转型

父类型还原子类型

向上转型后，子类对象引用父类类型，父类类型中没有子类方法

向下转型后，可以调用子类特有的方法，同上

b i2 =(b)i; //创建一个b类型的对象，接收强制转换后的i

//b类型的i2 = (强制转换为b类型) i

因为向下转型有很多不确定性，不确定这个子类是否继承于这个父类，如果不属于，就会抛出异常

java.lang.ClassCastException

//类型转换异常

这时就要用到instanceof

instanceof

判断某个类是否继承于某个父类，返回boolean类型值

boolean b = i instanceof b;

//布尔值 = 子类对象 instanceof 父类名称

如果的确继承这个父类，就可以向下转型实现还原对象类型

上下转型的作用

上面说过继承，继承后方法会被重写，重写的方法就成为这个子类的区别

比如人类 男人继承于人类，重写了方法

女人继承于人类，重写了方法

这时男人和女人的区别就是方法不同(重写)，但也有相同的方法

相同 吃饭 睡觉 看书

不同 胡子 喉结

people p = new man(); //向上转型，将男人看做一个人

p.eat(); //调用男人吃饭方法，实际上是调用人的吃饭方法

//如果这时男人方法里重写了吃饭，就会运行重写的方法

man m = (man) p ; //强制转换为男人，这时就可以使用胡子，喉结方法

不需要面对子类型，通过提高扩展性，或者使用父类的功能即可完成操作，就是使用向上转型。

## 多态

同一个变量，同一个方法，不同的效果

public class Test\_Object { //存放主方法的类

public static void main(String[] args) {

people p = new man(); //实例化对象，向上转型，指向男人

p.eat(); //男人吃饭

p = new woman(); //重新实例化，指向女人

p.eat(); //女人吃饭

}

}

class people{ //定义父类

public void eat(){ //父类有一个吃饭方法

System.out.println("吃饭");

}

}

class man extends people{ //定义男人类

public void eat(){

System.out.println("男人吃饭"); //男人吃饭

}

}

class woman extends people{ //定义女人类

public void eat(){

System.out.println("女人吃饭"); //女人吃饭

}

}

通过重新赋值，可以使一个对象输出多种效果，这就是多态。

## 抽象类

抽象类，可以被叫做框架类，主要的作用就是构建程序框架

这种类主要作为父类存在，被子类继承后，子类必须复制这个框架的所有抽象方法

声明一个抽象方法或抽象类，就需要用到abstract关键字修饰

public abstract class Animal { //定义一个抽象类

public abstract void sleep(); //抽象类中有一个抽象方法

public abstract void sleep(int a); //抽象方法可以重载 这时他们的参数列表不同

}

class a extends Animal{ //a继承于抽象类

public void sleep() { //继承之后必须重写抽象方法

System.out.println("我是a"); //因为抽象方法只是一个框架，没有大括号，没有实际运行的代码

}

}

class b extends Animal{ //b继承于抽象类

public void sleep() { //同上

System.out.println("我是b");

}

}

由此可见，抽象类的作用就是创建框架，然后子类继承时就必须使用这个框架来重写方法

需要注意的是

抽象类不能被实例化，也就是说，不能new，但可以创建构造方法

抽象类不能使用static关键字修饰为静态，静态的用法就是不用实例化直接调用，而抽象类不能实例化

抽象方法可以重载

有抽象方法的类一定是抽象类，抽象类不一定有抽象方法

继承了抽象类的类，必须重写抽象方法

另外，抽象类中可以定义普通方法，变量

抽象方法和普通方法的区别就是 需不需要重写

抽象方法必须重写 普通方法可以选择重写或者直接继承父类的方法

## 接口

可以看做是一个纯粹的抽象类

接口的作用就是辅助抽象类搭建结构，例如，我需要实现一个生物的抽象类

public abstract class Biology { //首先定义一个生物的抽象类

public abstract void life(); //抽象类中定义一个抽象方法，生命 每个生物都有生命

}

这时，生物又区分为动物和植物

class Animal extends Biology{ //动物，继承生物抽象类

public void life() {

}

}

class Botany extends Biology{ //植物，继承生物抽象类

public void life() {

}

}

如果我想在生物里写一个光合作用的抽象方法，发现动物是不会光合作用的

这时植物类不能修改为抽象类，就要用到接口，创建一个光合作用的接口，在植物类中使用这个接口即可

接口的创建，用到interface关键字修饰

public interface Breathing { //创建一个接口 呼吸

public void photosynthesis(); //接口方法 光合作用

} //接口的方法都是final(常量)和static(静态)

//接口的方法省略abstract关键字，且所有方法都是抽象方法

接口创建后，使用接口要使用implements关键字

class Botany extends Biology implements Breathing{ //调用接口Breathing

public void life() {

}

public void photosynthesis() { //要重写接口方法

}

}

父类只能继承一个，接口可以调用多个，用逗号隔开

class a implements b,c,d{

}

另外，接口修饰符只有

public 公有

private 私有

abstract 抽象

default 默认

static 静态

strickfp 精确浮点

# 类的高级特性

## Java类包

类名冲突

由于Java的命名规约，每个类都要见名知意

可是，有些类的名字相同却执行方法不同，如sql的date类(数据库的日期格式)和util的date(数字格式化的日期格式)类

由此衍生出包这个概念，包由类组成，SQL包中装的都是对数据库管理的类

类路径

还是sql.date类和util.date类，当一个方法需要用到这两个类时，不可能new一个date就完事

Data d = new Data();

这时程序无法识别是哪个data类

这时就要用到完整的类路径，就比如同名的人不同姓，就只能用姓氏来区分

java.util.Date d1 = new java.util.Date(); //util的Date类

java.sql.Date d2 = new java.sql.Date(123); //sql的Date类，有参构造方法要传参123

创建导入包

在编译器中导入包，New-packag即可创建包

Package关键字代表的是本类属于某个包

package senior; //在类的第一行，说明本类属于senior包

注意，java包的命名规则是全部为小写字母

创建好之后，同包的类不需要导入，不同的包就要导入

package senior; //第一行说明本类属于senior包

import array.\*; //导入array包中所有类 \*代表所有类

导入静态变量

首先在core包下创建一个类Im，类中定义一个静态变量static\_int

package core; //本类属于core包

public class Im { //类名Im

public static int static\_int =123; //类中只有一个静态方法static\_int

}

接着在senior中导入这个静态变量

package senior; //本类属于senior包

import static core.Im.static\_int;; //导入一个静态变量

## final

final用来修饰变量，使变量值不能再被修改，称为常量，参考上面的[常量介绍](#_变量和常量)

这里深入了解一下final的定义

被final定义的方法通常需要用大写字母命名，同时必要时用下划线连接

被final定义的变量，方法甚至是类都不能改变，final的定义的对象也不能改变，可是有些对象的值本身就是改变的

比如定义一个随机数常量对象，这个常量就不能指向其他对象，可常量运行时输出的是不同的数字(随机数)

package senior; //包名

import java.util.Random; //导包

public class Final { //类名

static Random r= new Random(); //使用随机数

public static final int a1 = r.nextInt(20); //定义静态常量 随机数

public final int a2 = r.nextInt(20); //定义常量 随机数

public static void main(String[] args) { //主方法

Final f=new Final(); //实例化类

System.out.println(f.a1); //输出静态常量 3

System.out.println(f.a2); //输出常量 8

Final f2 = new Final(); //第二次实例化类

System.out.println(f2.a1); //再次输出静态常量 3

System.out.println(f2.a2); //再次输出常量 16

}

}  
由此可见，常量如果加上静态修饰符，常量就会被放在静态区，且常量规定不能更改，所以静态常量是不能改变的

final方法

常量方法，不能修改的方法

class f{ //父类f

public final void f1() { //常量方法f1

}

}

class i extends f{ //子类i继承父类f

public final void f1(){ //重写方法f1 报错，不能重写final类型的方法

}

}

final类

常量类，不允许被继承和修改

final class f{ //常量类

}

class i extends f{ //继承常量类，报错，不允许常量类有子类

}

## 内部类

内部类，就是类中类，在类中再定义一个类

而在内部类又分为成员、局部、静态、匿名

成员和局部就像前面说过的变量，成员属于类所有，局部属于方法所有

成员内部类

在类的内部创建内部类，级别同成员变量、方法，可以使用本类中所有的属性

public class CyClass { //外部类 CyClass

class Cy{ //成员内部类 Cy

public void cy() { //内部类的方法 cy

System.out.println("123");

}

}

创建好内部类之后，调用内部类的方法如下

//方法一 在外部类中实例化内部类，然后在方法中使用该对象

//外部类对象.内部类对象.内部类方法

Cy c = new Cy(); //在CyClass下实例化Cy

public static void main(String[] args) { //主方法

CyClass c1 = new CyClass(); //实例化CyClass

c1.c.cy();

}

//方法二 在方法中实例化外部类，再创建一个内部类型的 外部类对象.new 实例化对象

public static void main(String[] args) { //主方法

CyClass c = new CyClass(); //实例化CyClass

CyClass.Cy n = c.new Cy(); //外部类.内部类 名字= 外部类对象.new 内部类();

}

}

局部内部类

在外部类方法体中定义的类叫局部内部类，局部内部类不能有访问修饰符，且只能访问方法体中的final类型的局部变量

public class A { //外部类

public void method(){ //外部类方法

int aa=100; //外部类变量

class B { //局部内部类 这里不能有访问修饰符

public void method2(){ //局部内部类方法

System.out.println("这是局部内部类的方法体");

// System.out.println(aa);注意，这里不能访问aa，除非把aa声明成final

}

}

B b=new B(); //在这里new局部内部类的对象，并调用方法

b.method2(); //局部内部类只属于方法，不再属于外部类，只能在方法里调用

}

public static void main(String args[]){

A a=new A(); //实例化外部类

a.method(); //调用外部类的方法，方法里是B类的method2方法的调用(多层调用)

}

}

可是，可以通过接口的向上转型，并在接口中定义方法，在内部类重写方法，达到外部类主方法调用局部类

//第一步：定义接口，接口添加方法，这个方法是需要在外部类被调用的

interface Out{

public void syso();

}

//第二步：创建外部类，在外部类中创建方法，方法内创建内部类，内部类中重写接口方法，方法结束时返回接口对象

public class JbClass { //创建外部类

public Out j(){ //在外部类中创建方法

class Jb implements Out{ //方法内创建内部类

@Override

public void syso() { //内部类中重写接口方法

System.out.println("接口123");

}

} //注意，装内部类的方法需要一个返回值，返回Out类型

return new Jb(); //返回一个实例化的内部类 这就相当于 Out ? = new Jb();

//问号的意思是，在主方法中接收该对象的名称

}

public static void main(String[] args) {

JbClass jbclass=new JbClass(); //实例化外部类

Out o=jbclass.j(); //jbclass的j方法返回一个Out类型的对象，用o接收

//实际上这是一个向上转型，因为jbclass.j()的返回值就是new Jb();

//这就是上面的Out o = new Jb();

o.syso(); //调用被重写的方法，输出结果为 接口123

}

}

匿名内部类

换言之，就是没有名字，或者隐藏名字的内部类

直接在返回值中创建类方法

//1.定义接口

interface Nmjk{

void nmff();

}

//2.创建外部类

public class NmClass {

//3.创建外部类方法 返回值为接口类型

public Nmjk Nm(){

//4.方法内返回值里写创建类方法，不写类名，直接大括号里写方法

return new Nmjk() {

public void nmff() { //匿名类方法

System.out.println("123");

}

}; //类方法要用引号结束

}

//5.主方法内创建对象

public static void main(String[] args) {

NmClass n = new NmClass(); //外部类对象

Nmjk j = n.Nm(); //接口类型的j = 外部类对象.外部类方法(装匿名类的方法)

j.nmff(); //调用接口方法，被重写后的结果 输出123

}

}

静态内部类

|  |  |  |
| --- | --- | --- |
| 普通内部类 | 可以调用 | 外部类的非静态成员 |
| 静态内部类 | 不能调用 |
| 普通内部类 | 不能声明 | 静态变量、方法（mian） |
| 静态内部类 | 可以声明 |

和静态属性一样，不用类名.属性来调用

public class JtClass { //外部类

int a = 10; //外部类的成员变量

static int b = 20; //外部类的静态成员变量

static class Jt{ //静态 内部类

int c; //内部类的成员变量

Jt(){ //内部类的构造方法

this.c = a; //报错，调用了非静态变量

this.c = b; //将外部类的静态的b赋值给内部类静态的c

}

}

}

//另外，主方法是静态方法，所以可以写在静态内部类中

public class JtClass {

static class Jt{

public static void main(String[] args) {

System.out.println("123");

}

}

}

//值得注意的是，每多一个类，编译时都会多产生一个class文件

//也就是说明每一个类都可以独立运行

//所以，静态内部类的另一个好处就是，，可以设置主方法(静态)，可以单独运行

内部类继承

内部类继承，必须要创建一个带外部类对对象的构造方法

//首先定义一个内部类

class A1 {

class Cy{ //内部类

public void cy() {

System.out.println("123456");

}

}

}

//继承内部类

public class A extends A1.Cy{

//构造方法必须定义一个参数为外部类的对象

public A(A1 c) {

//引用外部类

c.super();

}

public static void main(String[] args) {

//创建外部类对象(A1)

A1 c=new A1();

//创建本类(A)的对象，调用带参构造方法

A i = new A(c);

//A类继承于内部类，这时就复制了内部类的方法，直接调用cy方法

//前面只是通过构造方法将引用从外部类拉到内部类

i.cy();

}

}

注意，局部内部类属于方法所有，不能被继承

# 异常处理

区分异常

异常就是程序运行时出现的错误，它会中断程序的运行

package d11;

public class Test { //类

public static void main(String[] args) { //主方法

System.out.println(3/0); //输出3/0的结果

}

}

异常：Exception in thread "main" java.lang.ArithmeticException: / by zero

at d11.Test.main(Test.java:5)

出现异常，java.lang.算术异常，在d11包Tset类main方法的第五行

异常又分为可捕捉异常(Exception)和不可捕捉异常(Error)

如果将上面的输出语句后的分号去掉，运行则会抛出不可捕捉异常

Exception in thread "main" java.lang.Error: 无法解析的编译问题：

语法错误，将“;”插入到完整 BlockStatements 中

at d11.Test.main(Test.java:5)

可捕捉异常通常出现在代码语法正确的情况下

不可捕捉的异常不能用代码纠正，只能重写

捕捉异常

try catch finally语句

try { //执行的代码，其中可能有异常。一旦发现异常，则立即跳到catch执行。否则不会执行catch里面的内容 }

catch { //除非try里面执行代码发生了异常，否则这里的代码不会执行 }

finally { //不管什么情况都会执行，包括try catch 里面用了return ,可以理解为只要执行了try或者catch，就一定会执行 finally }

try {

System.out.println(3/0);

} catch (Exception e) {

System.out.println(“0”)

} finally {

System.out.println("1");

}

//运行结果就是，如果try代码块出现异常，就会输出 0 1

//如果try运行没有异常，则输出 1

重点说一下catch

catch (Exception e) { //语法为:catch( 异常类型 对象名 ){ 异常处理代码 }

// Exception作为所有异常类的父类，可以捕捉所有异常

//另外，要捕捉特定的异常，需要改写异常类型，如(ArithmeticException e)只捕捉算术异常

//捕捉异常就像 if…else 语句一样，第一个判断如果包含了全部可能，则后面的判断永远都执行不到

//也就是说，catch括号中，要么只有一个catch(Exception)，要么Exception排最后

System.out.println(“0”)

}

接下来是finally

无论try中有没有异常，都会执行finally语句，也就是说只要执行了try语句，就一定会执行finally语句除非

* finally语句内有异常
* 前面的代码实现了Systrm.exit()
* 程序所在线程死亡(中断)
* 关闭CPU

Java常见异常

1、java.lang.NullpointerException(空指针异常)

原因：这个异常经常遇到，异常的原因是程序中有空指针，即程序中调用了未经初始化的对象或者是不存在的对象。

经常出现在创建对象，调用数组这些代码中，比如对象未经初始化，或者图片创建时的路径错误等等。对数组代码

中出现空指针，是把数组的初始化和数组元素的初始化搞混淆了。数组的初始化是对数组分配空间，而数组元素的

初始化，是给数组中的元素赋初始值

2、 java.lang.ClassNotFoundException（指定的类不存在）

原因：当试图将一个String类型数据转换为指定的数字类型，但该字符串不满足数值型数据的要求时，就抛出这个异

常。例如将String类型的数据"123456"转换为数值型数据时，是可以转换的的。但是如果String类型的数据中包含了

非数字型的字符，如123\*56，此时转换为数值型时就会出现异常。系统就会捕捉到这个异常，并进行处理

3、java.lang.ClassNotFoundExceptio（指定的类不存在）

原因：是因为类的名称和路径不正确，通常都是程序试图通过字符串来加载某个类的时候可能会引发异常。例如：

调用Class.forName()、或者调用ClassLoad的finaSystemClass()、或者是LoadClass()时出现异常

4、java.lang.IndexOutOfBoundsException（数组下标越界异常）

原因：查看程序中调用的数组或者字符串的下标值是不是超出了数组的范围，一般来说，显示调用数组不太容易出这

样的错，但隐式调用就有可能出错了，还有一种情况，是程序中定义的数组的长度是通过某些特定方法决定的，不是

事先声明的，这个时候可以先查看一下数组的length，以免出现这个异常

5、java.lang.IllegalArgumentException（方法的参数错误）

例如g.setColor(int red,int green,int blue)这个方法中的三个值，如果有超过２５５的会出现这个异常，如果程

序中存在这个异常，就要去检查一下方法调用中的参数传递或参数值是不是有错

6、java.lang.IllegalAccessException（没有访问权限）

当程序要调用一个类，但当前的方法即没有对该类的访问权限便会出现这个异常。如果程序中用了Package的情况下

有可能出现这个异常

7、java.lang.ArithmeticException（数学运算异常）

当数学运算中出现了除以零这样的运算就会出这样的异常。

8、java.lang.ClassCastException（数据类型转换异常）

当试图将对某个对象强制执行向下转换，但该对象又不可转换或又不可转换为其子类的实例时将出现该异常

9、 java.lang.FileNotFoundException（文件未找到异常）

当程序打开一个不存在的文件来进行读写时将会引发该异常。该异常由FileInputStream,FileOutputStream,

RandomAccessFile的构造器声明抛出，即使被操作的文件存在，但是由于某些原因不可访问，比如打开一个

只有只读权限的文件并向其中写入数据，以上构造方法依然会引发异常

10、java.lang.ArrayStoreException（数组存储异常）

当试图将类型为不兼容类型的对象存入一个Object[]数组时将引发异常

11、java.lang.NoSuchMethodException（方法不存在异常）

当程序试图通过反射来创建对象，访问(修改或读取)某个方法，但是该方法不存在就会引发异常。

12、 java.lang.EOFException（文件已结束异常）

当程序在输入的过程中遇到文件或流的结尾时，引发异常。因此该异常用于检查是否达到文件或流的结尾

13、java.lang.InstantiationException（实例化异常）

当试图通过Class的newInstance()方法创建某个类的实例,但程序无法通过该构造器来创建该对象时引发。

Class对象表示一个抽象类，接口，数组类，基本类型 。该Class表示的类没有对应的构造器。

14、java.lang.InterruptedException（被中止异常）

当某个线程处于长时间的等待、休眠或其他暂停状态，而此时其他的线程通过Thread的interrupt方法终止该线程时抛出该异常。

15、java.lang.CloneNotSupportedException （不支持克隆异常）

当没有实现Cloneable接口或者不支持克隆方法时,调用其clone()方法则抛出该异常

16、java.lang.OutOfMemoryException （内存不足错误）

当可用内存不足以让Java虚拟机分配给一个对象时抛出该错误

17、java.lang.NoClassDefFoundException （未找到类定义错误）

当Java虚拟机或者类装载器试图实例化某个类，而找不到该类的定义时抛出该错误

自定义异常

首先创建一个异常类，继承自异常类(Exception)

然后加一个有参构造方法，构造方法内调用父类构造方法，父类的构造方法是输出异常信息

public class ZDY extends Exception{

public ZDY(String a) {

super (a);

}

}

然后定义一个方法，这个方法负责判断哪种情况下抛出哪种异常

static void sum(int a) throws ZDY { //定义一个方法，传递一个int参数，然后声明这个方法可能会产生的异常

if(a>50) { //当参数满足某一个条件

throw new ZDY("大于50"); //抛出异常，并设置异常信息，上面的是String类型值

}

}

设置方法之后，只需要调用这个方法

try {

sum(51); //给这个方法输入一个数

}catch (ZDY e) { //设置可能产生的异常

System.out.println(e); //输出异常信息 输出结果为d11.ZDY: 大于50

}

值得注意的两个新语法，在创建自定义异常时用到

throws //用于声明异常，指定抛出的异常类型

//在方法后使用throws 异常1，异常2，异常3

throw //用于抛出异常对象，交由异常处理

//在判断中执行，throw new 异常(传递参数)

运行时异常

RuntimeException继承于Exception

大部分异常都是运行时异常，也就是程序运行时才会发现的异常

|  |  |
| --- | --- |
| NullPointerException | 空指针异常 |
| ArrayindexoutBoundsException | 数组下标越界异常 |
| ArithmeticException | 算术异常 |
| ArrayStoreException | 数组中包含不兼容的值异常 |
| SecurityException | 安全性异常 |
| negativeArraySizeException | 数组长度为负异常 |
| IllegalArgumentException | 非法参数异常 |

异常使用原则

* 不要过多的使用异常，这样会增加系统的负担。
* 在方法中使用try-catch捕获异常时，要对异常作出处理。
* try-catch语句块的范围不要太大，这样不利于对异常的分析。
* 一个方法被覆盖时，覆盖它的方法必须抛出相同的异常或子异常。
* 捕捉到的异常记得处理

# Swing

## Swing概述

Swing是一个用户界面开发工具包，完全由java语言编写，不依赖于操作系统简单来说，就是开发界面

继承关系为

然后就是各种组件

## 窗体

JFrame

JFrame作为一个容器，可以承载各种组件

首先，创建窗体有两种方法

//方法一，创建一个类，继承JFrame类，在构造方法中，设置窗体属性，然后在主方法中调用对象

public class Frame extends JFrame{

public Frame() {

setTitle("标题");

}

public static void main(String[] args) {

new Frame();

}

}

//方法二，在主方法中创建对象，设置属性

public class Frame{

public static void main(String[] args) {

JFrame f = new JFrame("标题");

}

}

通常，方法一适用于比较复杂的窗体，方法二适合调试窗体

上面的代码，创建的时候设置了窗体标题，设置标题语法：

//在窗体继承类方法下

setTitle("标题");

//在创建窗体对象下

JFrame f = new JFrame("标题");

设置了标题，看看其他属性的设置，在继承类方法下，不用 对象. 方法来调用，创建对象时就要

就是方法一不用.对象，方法二需要.对象，以下的代码使用方法一，如果使用方法二记得在前面加 对象.

setVisible(true); //窗体可见

setDefaultCloseOperation(DISPOSE\_ON\_CLOSE); //关闭方式

/\*EXIT\_ON\_CLOSE 隐藏窗口并关闭

\* DO\_NOTHING\_ON\_CLOSE 不允许关闭窗口，点关闭没反应

\* HIDE\_ON\_CLOSE 隐藏窗口不关闭程序

\* DISPOSE\_ON\_CLOSE 释放窗口资源，没资源自动关闭，存在一定延迟

\* \*/

setSize(300, 200); //窗口大小，单位像素(宽，高)

setLocation(500, 500); //窗口显示的坐标，计算左上角的第一个像素点

setBounds(x, y, width, height); //同时设置大小和坐标

setResizable(false); //是否可以调整大小

getX(); //获取窗体左上角的X坐标

getY(); //获取窗体左上角的Y坐标

setTitle(String); //设置窗体标题

JDialog

和上面的创建方法一样，也是两种创建方法

* 主方法中创建对象，设置属性
* 类继承JDialog，在构造方法中设置属性，主方法中调用对象

标题，位置，大小等属性同JFrame，参考上方语法

唯一不同的是，JDialog属于小窗口，可以选择父窗口

设置父窗口后，可以通过父窗口(JFrame)中的事件打开，打开后阻塞父窗口

阻塞：打开对话框后，不能与父窗口进行交互，只能对话框关闭后才能点击父窗口

class d1 extends JDialog{ //对话框类

public d1(JFrame j) { //构造方法

super(j, "答案", true); //设置父窗体，以及阻塞效果，true阻塞，false不阻塞

Container c= getContentPane(); //获取对话框的容器

c.add(new JLabel("一二三")); //在容器里加一段文字

setSize(200, 150); //设置大小

setLocationRelativeTo(null); //设置居中显示，必须设置在大小之后，不然就会以左上角第一个点作为居中点

//会发现，少了一个setVisible(true)代码

//因为在这里设置显示，显示不出大小

}

}

提前说一下按钮的点击事件，首先创建一个父窗体

public class Frame extends JDialog{ //窗体类，继承JFrame

public static void main(String[] args) { //主方法

JFrame j = new JFrame(); //创建Frame对象

j.setTitle("算术"); //标题

j.setVisible(true); //设置可见

j.setDefaultCloseOperation(DO\_NOTHING\_ON\_CLOSE); //关闭方式

j.setResizable(false); //可否改变大小

j.setSize(500, 300); //窗体大小

j.setLocationRelativeTo(null); //居中显示

Container c= j.getContentPane(); //然后就是获取这个窗体的容器

c.setLayout(new FlowLayout()); //容器布局为流布局

c.setBackground(Color.white); //背景颜色为白色

c.add(new JLabel("1+1=几")); //添加标签

JButton b1 = new JButton("0"); //创建按钮

c.add(b1); //放进容器

b1.addActionListener(new ActionListener() { //按钮点击事件

@Override

public void actionPerformed(ActionEvent e) { //一个匿名内部类

d1 f = new d1(j); //实例化对话框，new的同调用构造方法

f.setVisible(true); //设置对话框显示

}

});

}

}

## 容器

设置完窗体的属性，然后到设置组件的属性

组件被装在容器中，首先在窗体中获取容器

Container c=getContentPane(); //获取容器

然后就是容器属性

因为前面在类的构造方法中创建窗体，也就是方法一，现在也可以继续在构造方法中继续获取，也可以创建另一个方法获取

public void con(Frame f) { //创建方法，传入一个窗体

Container c = f.getContentPane(); //在方法中获取容器

}

接下来的属性都基于con方法

c.setBackground(Color.white); //容器背景颜色

c.add(组件); //添加组件

//创建组件有两种方法

//1.创建后不修改属性，直接new一个组件

c.add(new JLabel("一段文字")); //添加一个标签

//2.实例化一个组件对象，然后放进容器内

JButton b = new JButton(); //实例化一个按钮

c.add(b); //添加这个按钮

c.remove(b); //删除组件

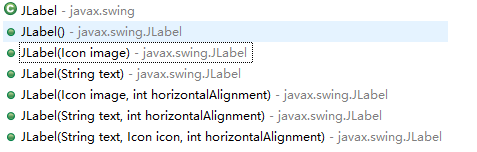
c.validate(); //验证容器中组件，等于刷新

f.setContentPane(c); //重新载入容器也可以达到刷新功能

## 标签和图

标签，可以显示一行只读文本，图像或者带图像的文本，不能产生任何类型事件

在创建标签时就可以定义的属性：

JLabel l1 = new JLabel("内容")；

JLabel l2 = new JLabel(图片)；

JLabel l3 = new JLabel(图片，对齐方式)；

JLabel l4 = new JLabel("内容"，对齐方式)；

JLabel l5 = new JLabel("内容"，图片，对齐方式)；

图可以自行绘制，也可以插入图片

图片的导入和使用：

//方法一，创建图片对象，指向图片路径(项目为起点，精确到文件名)

Icon i = new ImageIcon("src/d12/123.png"); //创建对象

//方法二，使用URL获取

URL u = Test.class.getResource("123.png"); //创建URL对象，后面接类名.class.getResource(文件名)

//u就是图片的绝对路径，盘符下的全部路径 file:/G:/eclipse\_java/Test/bin/d12/123.png(绝对路径)

Icon i = new ImageIcon(u); //创建对象

//创建之后，就要放进标签内

a.setIcon(i); //将图片放在标签内

如果要绘制图标，就要调用Icon接口

public class Test\_Icon implements Icon { //创建类，调用接口，接口有三个属性，分别是

private int w; //图标的宽，成员变量长

private int h; //图标的高，成员变量宽

public Test\_Icon(int w,int h) { //构造方法，在调用本类时传两个参数

this.w=w; //将传来的参数赋值给成员变量，this关键字

this.h=h;

}

public void paintIcon(Component c, Graphics g, int x, int y) { //第三个属性，图标的类型

//参数分别是 容器c 图标g 坐标值x和y

g.fillOval(x, y, w, h); //这里选择了一个实心圆，然后设置坐标x和y及大小

}

public int getIconWidth() { //获取图标长

return w;

}

public int getIconHeight() { //获取图标宽

return h;

}

public static void main(String[] args) { //主方法

Test\_Icon t=new Test\_Icon(15, 15); //实例化本类

JFrame j= new JFrame(); //创建窗体

Container c= j.getContentPane(); //获取窗体容器

JLabel l = new JLabel("图标",t,SwingConstants.CENTER)； //创建标签(文字，图标，位置)

c.add(l); //在容器中添加标签

j.setVisible(true); //窗体可见

}

}

## 布局管理

绝对布局

硬性指定组件的大小和位置，这个布局不属于布局管理器，强制指定位置，不使用任何辅助调整

首先，使用绝对布局就要取消原本的布局(默认边界布局)

setLayout(null); //设置布局为空

任何然后就是指定大小和位置

setBounds(200, 200, 500, 300);

//x为200，y为200，宽500，高300

对窗体使用这个方法，设置的是窗体在整个电脑屏幕的位置和大小

对组件使用这个方法，设置的是组件在窗体的位置和大小

组件之间有覆盖效果，下面就是右图的代码

public class Layout extends JFrame{ //创建类继承JFrame

public Layout() { //构造方法

setTitle("绝对布局"); //标题

setLayout(null); //取消布局管理

setBounds(200, 200, 500, 300); //设置窗体的大小显示位置

Container c = getContentPane(); //获取窗体容器

JButton b1 = new JButton("按钮一"); //创建按钮对象

JButton b2 = new JButton("按钮二");

b1.setBounds(10, 30, 80, 50); //设置按钮位置和大小

b2.setBounds(60, 70, 110, 70);

c.add(b1); //将按钮装进容器

c.add(b2);

setVisible(true); //窗体可见

setDefaultCloseOperation(EXIT\_ON\_CLOSE); /窗体关闭方式

}

public static void main(String[] args) { //主方法

new Layout(); //调用本类对象

}

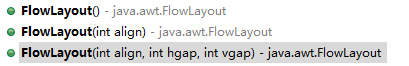
}

流布局

FlowLayout

一堆组件，从左到右摆，第一排放满了，就放在第二行，以此类推

组件在每一行都默认居中排列，也可以设置对齐方式，使用流布局的方法

setLayout(new FlowLayout());

//左图有三个构造方法，分别有三个属性

//1.align(0左对齐，1居中对齐，2右对齐)

//2.hgap(组件之间的水平间隔)

//3.vgap(组件之间的垂直间隔)

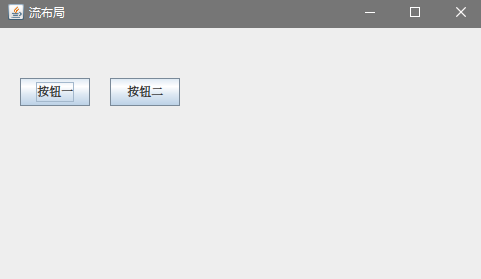
实例代码

public class Layout2 extends JFrame{ //创建类

public Layout2() { //构造方法

setTitle("流布局"); //标题

setLayout(new FlowLayout(0,20,50)); //流布局，左对齐，左右间距20(水平)，上下间距50(垂直)

 setBounds(200, 200, 500, 300); //窗体大小

Container c = getContentPane(); //获取容器

JButton b1 = new JButton("按钮一"); //创建按钮

JButton b2 = new JButton("按钮二");

c.add(b1); //添加按钮

c.add(b2);

setVisible(true); //窗体可见

setDefaultCloseOperation(EXIT\_ON\_CLOSE); //关闭方式

}

public static void main(String[] args) {

new Layout2(); //调用对象

}

}

边界布局

BorderLayout

边界布局是默认布局，边界布局分东西南北中五个区域，而每个区域的组件都会覆盖

边界布局的主要代码为

c.add(b1 ,BorderLayout.CENTER);

//代表的是b1这个组件在中区域

//区域： CENTER 中部

// EAST 东部

// WEST 西部

// SOUTH 南部

// NORTH 北部

实例代码

public class Layout3 extends JFrame{ //类

public Layout3() { //构造方法

setTitle("边界布局"); //标题

setBounds(200, 200, 500, 300); //窗体大小

Container c = getContentPane(); //窗体容器

c.setLayout(new BorderLayout()) //由于默认边界布局，可以不用设置

JButton b1 = new JButton("按钮一"), //创建五个按钮

b2 = new JButton("按钮二"),

b3 = new JButton("按钮三"),

b4 = new JButton("按钮四"),

b5 = new JButton("按钮五");

c.add(b1 ,BorderLayout.CENTER); //放进容器 中部

c.add(b2 ,BorderLayout.EAST); //东部

c.add(b3 ,BorderLayout.WEST); //西

c.add(b4 ,BorderLayout.SOUTH); //南

c.add(b5 ,BorderLayout.NORTH); //北

setVisible(true); //窗体可见

setDefaultCloseOperation(EXIT\_ON\_CLOSE); //关闭方式

}

public static void main(String[] args) { //主方法

new Layout3(); //调用对象

}

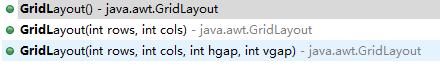
}

网格布局

GridLayout

网格布局，将容器变成表格，通过设置行列划分，每一个组件都会填满单元格

三个构造方法：

setLayout(new GridLayout());

//不传参数，默认一行n列，列由组件数量决定

setLayout(new GridLayout(2,3));

//设置两行三列

setLayout(new GridLayout(2,3,10,20));

//设置两行三列，水平(左右)间距10，垂直(上下)间距20

同样是上一节(边界布局)的五个按钮，分别使用无参，两参(行列)，四参(行列和间距)构造方法的效果



实例代码

public class Layout4 extends JFrame{

public Layout4() {

setTitle("网格布局");

setBounds(200, 200, 500, 300);

setLayout(new GridLayout(2,3,10,20));

Container c = getContentPane();

JButton b1 = new JButton("按钮一"),

b2 = new JButton("按钮二"),

b3 = new JButton("按钮三"),

b4 = new JButton("按钮四"),

b5 = new JButton("按钮五");

c.add(b1 );

c.add(b2 );

c.add(b3 );

c.add(b4 );

c.add(b5 );

setVisible(true);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

}

public static void main(String[] args) {

new Layout4();

}

}

需要注意的是，组件会随着窗口的改变而改变，这也是和流布局不同的地方

网格组布局

GridBagLayout

最复杂的一个布局，基于网格布局，添加了很多属性

使用方法：

//第一步和第二步很简单，就像前面的布局一样

setLayout(new GridLayout()); //设置网格布局

setLayout(new GridBagLayout()); //设置网格组布局

//第三步就不一样了，要创建一个约束对象，给这个约束对象创建多个属性

GridBagConstraints g = new GridBagConstraints();

//第四步，在添加组件的时候使用这个对象

c.add(组件,g);

实例代码，首先使用网格组布局

public class Layout5{ //类

JFrame f = new JFrame(); //窗体

Container c= f.getContentPane(); //容器

public void frame() { //设置窗体属性的方法

c.setLayout(new GridBagLayout()); //设置窗体布局管理：网格组布局

f.setSize(800, 600); //窗体大小

f.setLocationRelativeTo(null); //不设置相对位置，居中

f.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); //关闭方式

}

public static void main(String[] args) { //主方法

Layout5 d = new Layout5(); //实例化本类

d.frame(); //调用窗体属性方法

d.f.setVisible(true); //设置窗体可见

}

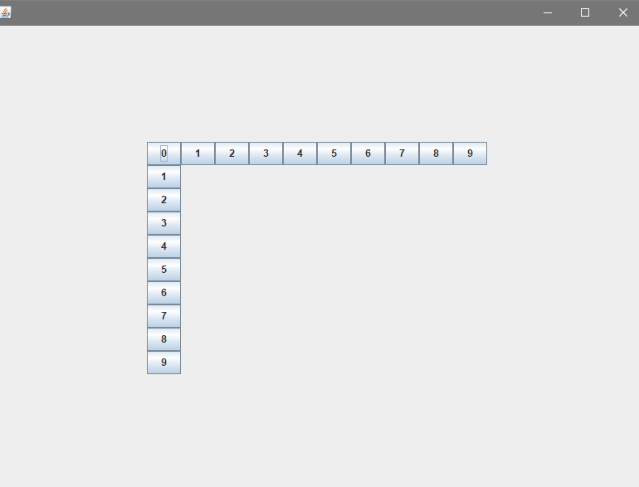
}

这里的gridx设置的是横坐标，gridy设置的是纵坐标

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |

gridx和gridy

为了方便观看布局，使用for循环制作一个坐标

public void test() {

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

GridBagConstraints g1 = new GridBagConstraints();

g1.gridx=i; //从零开始

g1.gridy=0; //左右

c.add(new JButton(i+""),g1);

GridBagConstraints g2 = new GridBagConstraints();

g2.gridx=0; //上下

g2.gridy=i; //从零开始

c.add(new JButton(i+""),g2);

}

}

d.test(); //在主方法中添加这个方法

gridwidth和gridheight

设置占用行数(gridheight或高度)和列数(gridwidth或长度)

public void test2() { //方法

GridBagConstraints g1 = new GridBagConstraints(); //约束对象

g1.gridx=1;

g1.gridy=1; //坐标(1,1)

g1.gridwidth = 2;

g1.gridheight = 1; //占两列一行

c.add(new JButton("两列一行"),g1); //添加

GridBagConstraints g2 = new GridBagConstraints(); //约束对象

g2.gridx=3;

g2.gridy=1; //坐标(3,1)

g2.gridwidth = 1;

g2.gridheight = 2; //占两行一列

c.add(new JButton("两行一列"),g2); //添加

GridBagConstraints g3 = new GridBagConstraints(); //约束对象

g3.gridx=5;

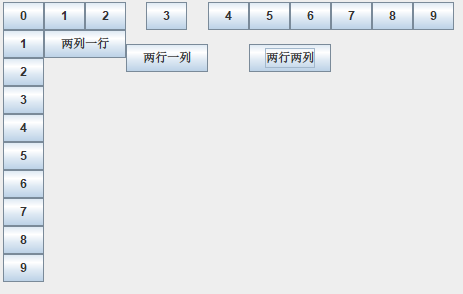
g3.gridy=1; //坐标(5,1)

g3.gridwidth = 2;

g3.gridheight = 2; //占两行两列

c.add(new JButton("两行两列"),g3); //添加

}



fill

填充方式在所属的区域内，可以设置

无填充(NONE)

水平填充(HORIZONTAL)

垂直填充(VERTICAL)

全部填充(BOTH)

public void test3() { //方法

GridBagConstraints g1 = new GridBagConstraints(); //约束对象

g1.gridx=1;

g1.gridy=3; //坐标(1,3)

g1.gridwidth = 2;

g1.gridheight = 2; //占两列两行

g1.fill = GridBagConstraints.NONE; //无填充

c.add(new JButton("无填充"),g1);

GridBagConstraints g2 = new GridBagConstraints(); //约束对象

g2.gridx=3;

g2.gridy=3; //坐标(3,3)

g2.gridwidth = 2;

g2.gridheight = 2; //占两行两列

g2.fill = GridBagConstraints.HORIZONTAL; //水平填充

c.add(new JButton("左右填充"),g2);

GridBagConstraints g3 = new GridBagConstraints(); //约束对象

g3.gridx=5;

g3.gridy=3; //坐标(5,3)

g3.gridwidth = 2;

g3.gridheight = 2; //占两行两列

g3.fill = GridBagConstraints.VERTICAL; //垂直填充

c.add(new JButton("上下填充"),g3);

GridBagConstraints g4 = new GridBagConstraints(); //约束对象

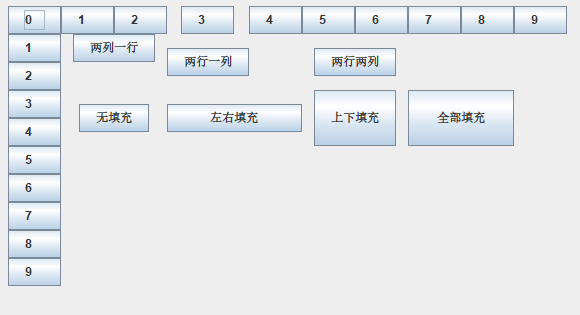
g4.gridx=7;

g4.gridy=3; //坐标(7,3)

g4.gridwidth = 2;

g4.gridheight = 2; //占两行两列

g4.fill = GridBagConstraints.BOTH; //全部填充

 c.add(new JButton("全部填充"),g4);

}

anchor

在显示区域的位置，组件在一个较大的位置中，可以设置所处的方位

|  |  |  |
| --- | --- | --- |
| 西北  NORTHWEST | 北  NODTH | 东北  NORTHEAST |
| 西方  WEST | 中  CENTER | 东  EAST |
| 西南  SOUTHWEST | 南方  SOUTH | 东南  SOUTHEAST |

public void test4() { //方法

GridBagConstraints g1 = new GridBagConstraints(); //约束对象

g1.gridx=1;

g1.gridy=5; //坐标(1,5)

g1.gridwidth = 2;

g1.gridheight = 2; //占2x2空间

g1.anchor = GridBagConstraints.EAST; //设置方位

c.add(new JButton("东"),g1);

GridBagConstraints g2 = new GridBagConstraints();

g2.gridx=3;

g2.gridy=5; //坐标(3,5)

g2.gridwidth = 2;

g2.gridheight = 2; //占2x2空间

g2.anchor = GridBagConstraints.NORTHEAST; //东北

c.add(new JButton("东北"),g2);

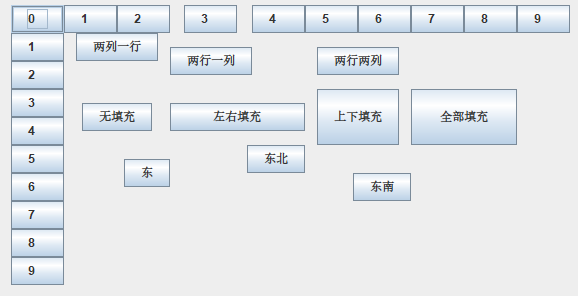
GridBagConstraints g3 = new GridBagConstraints();

g3.gridx=5;

g3.gridy=5;//坐标(5,5)

g3.gridwidth = 2;

g3.gridheight = 2;//占2x2空间

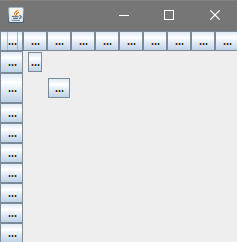
 g3.anchor = GridBagConstraints.SOUTHEAST;

c.add(new JButton("东南"),g3);

}

insets

设置组件四周距单元格边缘最小距离，这是一个类

使用的是构造方法，设置上下左右

g1.insets = new Insets(5,1,5,1); //依次是(上,左,下,右)

//最小距离在窗体极小时才特别明显

public void test5() {

GridBagConstraints g1 = new GridBagConstraints();

g1.gridx=1;

g1.gridy=1; //坐标(1,1)

g1.insets = new Insets(1,5,1,5);

c.add(new JButton("a"),g1);

GridBagConstraints g2 = new GridBagConstraints();

g2.gridx=2;

g2.gridy=2; //坐标(2,2)

g2.insets = new Insets(5,1,5,1);

c.add(new JButton("a"),g2);

}

ipadx和ipady

修改组件首选大小(初始大小)，正数加负数减

public void test6() {

GridBagConstraints g1 = new GridBagConstraints();

g1.gridx=1;

g1.gridy=1;//坐标(1,1)

g1.ipadx = 10;

g1.ipady = 5;//增加大小

c.add(new JButton("a"),g1);

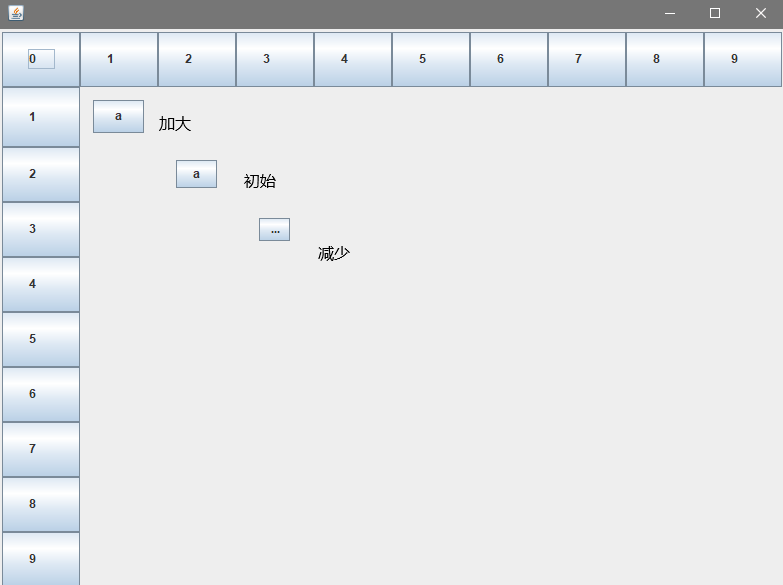
GridBagConstraints g2 = new GridBagConstraints();

g2.gridx=2;

g2.gridy=2;//坐标(2,2)

//不设置，默认

c.add(new JButton("a"),g2);

 GridBagConstraints g3 = new GridBagConstraints();

g3.gridx=3;

g3.gridy=3;//坐标(3,3)

g3.ipadx = -10;

g3.ipady = -5;//减少大小

c.add(new JButton("a"),g3);

}

weightx和weighty

没设置weightx和weighty时，窗体的容器总是有很多的额外空间

设置了weightx和weighty之后，容器的额外空间会分配给所设置的每行每列，也就是每行所占容器的占比，建议使用百分比取值

public void test7() {

GridBagConstraints g1 = new GridBagConstraints();

g1.gridx=1;

g1.gridy=1;//坐标(1,1)

g1.weightx =20;

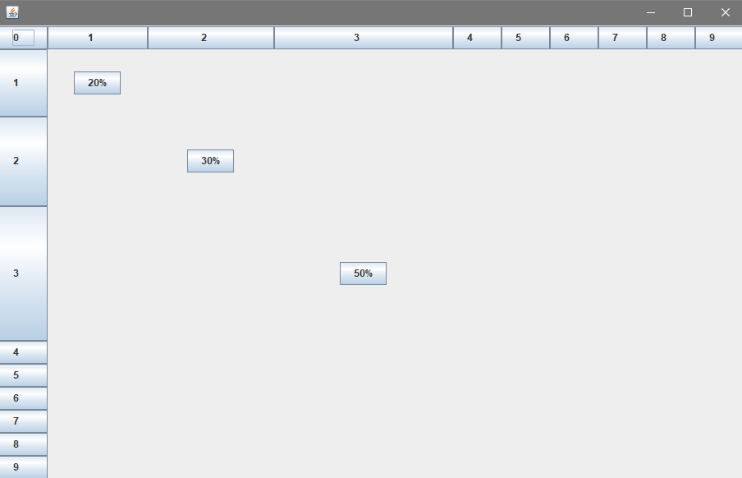
g1.weighty = 20;

c.add(new JButton("20%"),g1);

GridBagConstraints g2 = new GridBagConstraints();

g2.gridx=2;

g2.gridy=2;//坐标(2,2)

 g2.weightx = 30;

g2.weighty = 30;

c.add(new JButton("30%"),g2);

GridBagConstraints g3 = new GridBagConstraints();

g3.gridx=3;

g3.gridy=3;//坐标(3,3)

g3.weightx =50;

g3.weighty =50;

c.add(new JButton("50%"),g3);

}

同一行列设置了weightx\y属性，则会取最大值

## 面板

JPanel

特点：组件的级别，容器的作用

藏身于容器中的小容器，上面提到容器有多种布局，面板则可以当做组件一样添加进容器中

而面板也可以设置布局管理，面板也可以添加组件，从而实现了一个容器多个布局

public class Panel extends JFrame{

public Panel() {

setTitle("标题"); //标题

setSize(700, 550); //窗体大小

setLocationRelativeTo(null); //居中

setDefaultCloseOperation(EXIT\_ON\_CLOSE); //关闭方式

Container c = getContentPane(); //获取容器

c.setLayout(new GridLayout(2,2,5,5)); //容器是两行两列的网格

JPanel p1 = new JPanel(new GridLayout(2,1,3,3)); //网格布局两行一列

JPanel p2 = new JPanel(new GridLayout(1,2,3,3)); //网格布局一行两列

JPanel p3 = new JPanel(new BorderLayout()); //边界布局

JPanel p4 = new JPanel(new FlowLayout()); //流布局

p1.add(new JButton()); //网格布局的按钮插入

p1.add(new JButton());

p2.add(new JButton());

p2.add(new JButton());

p3.add(new JButton("a"),BorderLayout.CENTER); //边界布局的按钮插入

p3.add(new JButton("a"),BorderLayout.SOUTH);

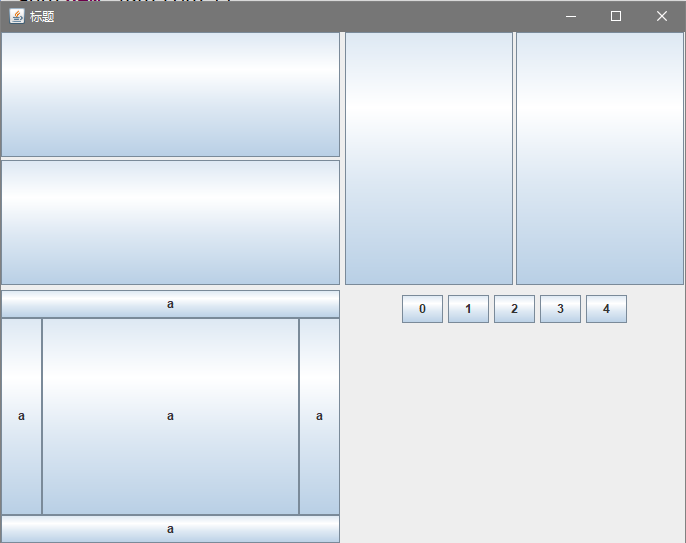
p3.add(new JButton("a"),BorderLayout.EAST);

p3.add(new JButton("a"),BorderLayout.WEST);

p3.add(new JButton("a"),BorderLayout.NORTH);

for (int i = 0; i < 5; i++) { //流布局的按钮插入，五个

p4.add(new JButton(i+""));

}

c.add(p1); //添加组件

c.add(p2);

c.add(p3);

c.add(p4);

setVisible(true); //可见

}

public static void main(String[] args) { //主方法

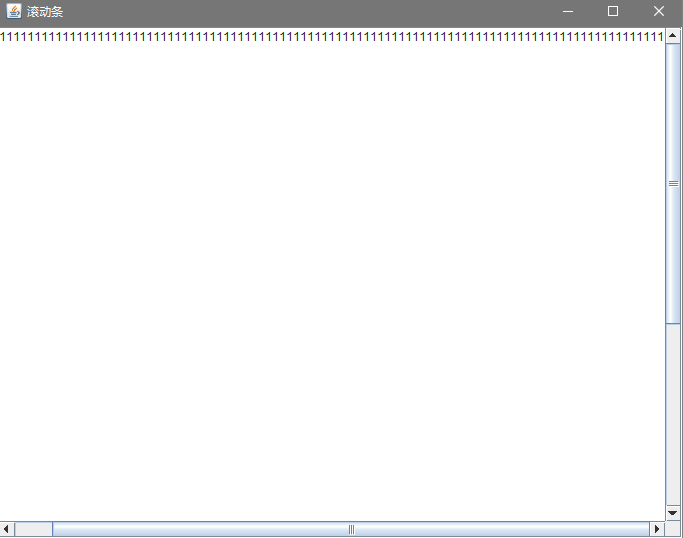
new Panel();

}

}

JScrollPane

和Jpanel一样，唯一的差别的就是

* 只能放一个组件
* 内容超出窗体时显示滚动条
* 不能使用布局管理器
* 可以放JPanel，JPanel里放多个组件

public class ScrollPane extends JFrame{

public ScrollPane() {

setTitle("滚动条"); //标题

setSize(700, 550); //大小

setLocationRelativeTo(null); //居中

setDefaultCloseOperation(EXIT\_ON\_CLOSE); //关闭方式

Container c =getContentPane(); //获取容器

JTextArea t= new JTextArea(); //创建文本域

JScrollPane s= new JScrollPane(t); //创建滚动面板，放文本域进去

c.add(s); //添加面板进去

setVisible(true);//显示窗体

}

public static void main(String[] args) { //主方法

new ScrollPane(); //调用对象

}

}

## 按钮组件

提交按钮

JButton构造方法有

Icon i = new ImageIcon("src/d12/123.png"); //图片对象

JButton b1= new JButton("文字"); //只带文字

JButton b2= new JButton(i); //只带图片

JButton b3= new JButton("文字加图片",i); //文字加图片

JButton b4= new JButton(); //空按钮

常用方法有

setText("文字"); //按钮文字

setIcon(); //按钮图标

setEnabled(boolean); //设置可用

setToolTipText("提示文字"); //鼠标悬停文字

setBackground(Color.GREEN); //背景颜色

setBorderPainted(false); //不设置边框

setBorder(BorderFactory.createLineBorder(Color.BLUE)); //边框颜色

addActionListener(new ActionListener() { //点击事件

public void actionPerformed(ActionEvent e) {

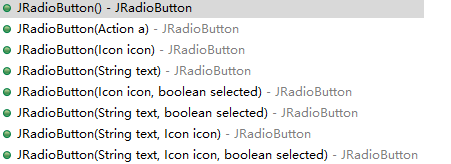
}

});

单选按钮

多选一，不能多选，选中时会清空其他单选按钮的选中状态

构造方法：文字，图片，初始状态(有没有被选中)

JRadioButton j1 = new JRadioButton("男");

JRadioButton j2 = new JRadioButton("女");

c.add(j1);

c.add(j2);

在容器中放入单选按钮，这时还不是单选

需要添加一个按钮组(ButtonGroup)

告诉程序这几个单选按钮为一个组

ButtonGroup b = new ButtonGroup(); //创建一个按钮组

b.add(j1);

b.add(j2); //加进去

这时的单选按钮只能选中一个，单选按钮的获取选中状态方法如下

getText(); //返回按钮文字

isSelected(); //返回的是布尔值，true为选中状态

setSelected(true); //设置默认选中状态。跟创建对象时使用的初始状态是一样的

b.clearSelection(); //按钮组的方法，清空现在组内所有单选按钮的选中状态

实例代码

Container c =getContentPane(); //容器

c.setLayout(new FlowLayout()); //流布局

ButtonGroup b = new ButtonGroup(); //按钮组



JRadioButton j1 = new JRadioButton("男");

JRadioButton j2 = new JRadioButton("女"); //两个单选按钮

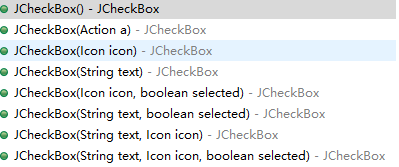
c.add(j1);

c.add(j2); //按钮放进容器

b.add(j1);

b.add(j2); //按钮放进组，告诉程序这两个是一组单选按钮

多选按钮

JCheckBox也叫复选框，可以进行多选

构造方法：文字，图片，初始选中状态

JCheckBox j1 = new JCheckBox("1",true);

JCheckBox j2 = new JCheckBox("2");

JCheckBox j3 = new JCheckBox("3");

c.add(j1);

c.add(j2);

c.add(j3);

获取复选框的选中状态也是一样的

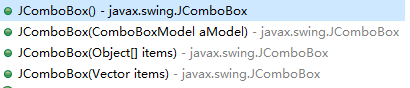
isSelected(); //获取选中状态

setSelected(true); //设置初始状态

## 列表组件

下拉列表

下拉框，点击下拉按钮显示一系列的选择，当然只能选一个

可以当做另一种单选按钮

构造方法：

下拉列表框模型(ComboBoxModel)

数组(Object[])

向量(Vector)

从传的参数来看，这些参数都是定义下拉列表的内容

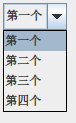
下面就是用数组定义下拉框列表，再用add方法外加选项

刚开始定义时，发现了一个问题

JComboBox b = new JComboBox<E>();

编译器给出的是带尖括号的E，这个E是一个枚举，后面会讲

现在先删了枚举，使用原始方法定义，删了有黄线警告

这里省略窗体设置以及属性，使用的是流布局，只展示下拉框代码

String[] a = {"第一个","第二个","第三个","第四个"}; //数组

JComboBox b = new JComboBox(a); //创建下拉框，放入数组

c.add(b); //放入下拉框

然后使用add方法添加第五个，添加选项的代码

b.addItem("第五个"); //使用addItem方法添加选项

注意，这行添加代码尽量放在下拉框放进容器前面

换言之，下拉框完全设置好再放入容器，体现了程序的可读性

接下来就是列表框模型

列表模型，将选项设置成一个模型(方便后期修改)，然后给下拉框设置这个模型

这里分两种，由于列表模型是一个接口，可以自定义模型，也可以直接给数值生成模型

//方法一

String[] a = {"第一个","第二个","第三个","第四个"}; //数值

ComboBoxModel m = new DefaultComboBoxModel(a); //创建模型，使用继承接口的类传入数组(生成模型)

JComboBox b = new JComboBox(); //创建下拉框

b.setModel(m); //给下拉框设置模型

//方法二，创建类调用接口

class test implements ComboBoxModel<String>{ //创建类，调用接口

String[] s = {"1","2","3","4","5"}; //创建数组

String n = null; //设置一个变量保存添加值

public int getSize() { //设置选项的数量

return s.length; //返回数组长度

}

public String getElementAt(int index) { //传入索引，返回值

return s[index]; //返回数组对应的索引选项

}

//从前两个方法可以看出模型的结构，下拉框接收到这个模型，会循环根据getSize的值循环填上值(getElementAt)

public void setSelectedItem(Object anItem) { //添加选项

n=(String)anItem; //将选项放进变量中，类型转换

}

public Object getSelectedItem() { //查看添加的值

return n; //返回变量保存的值

}

public void addListDataListener(ListDataListener l) {

}

public void removeListDataListener(ListDataListener l) { //忽略，接口必需方法

}

}

//然后在下拉框中实例化类

JComboBox b = new JComboBox(new test());

//或者使用向上转型

ComboBoxModel t = new test();

//这时可以使用刚才接口继承的方法setSelectedItem添加选项(仔细看上面)

JComboBox b = new JComboBox(t);

列表做好了，就看看常用方法，就拿刚才的下拉框b做例子

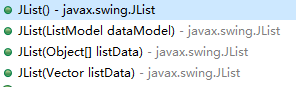
b.getSelectedIndex(); //获取当前选中值的索引

b.getSelectedItem(); //获取当前选中的值

b.setEditable(true); //设置可编辑，如果输入其他选择，索引值为-1 true可编辑 false不可编辑

列表框

与下拉列表不同的是，列表框显示全部选项，并可以选中多个

构造方法

列表框模型(ListModel)

数组(Object[])

向量(Vector)

和下拉框一样

String[] a = {"第一个","第二个","第三个","第四个"}; //数组

JList l = new JList(a); //利用数组创建框

l.setListData(a); //在创建空白框时，使用setListData添加选项，可以放数组，下面的setModel才是加模型

接着上面来说一下模型第三方法

String[] a = {"第一个","第二个","第三个","第四个"}; //数组

DefaultListModel m = new DefaultListModel(); //创建模型，注意，这里是继承接口的子类

for (String string : a) { //遍历数组

m.addElement(string); //使用addElement方法挨个添加进组件

}

JList l = new JList(); //创建列表框

l.setModel(m); //给列表框添加模型，

框做好了，常用方法有(列表框l)

l.setSelectionMode(); //设置选择方式，单选，多选，邻选

ListSelectionModel.SINGLE\_SELECTION //单选

ListSelectionModel.SINGLE\_INTERVAL\_SELECTION //邻选 只能选相邻的选项，不能隔着选

ListSelectionModel.MULTIPLE\_INTERVAL\_SELECTION //多选，随意选择

获取选中的选项，使用java.util.List类型来获取值，再通过for循环输出

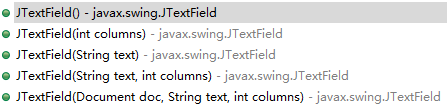
java.util.List a = l.getSelectedValuesList(); //由于得到的值可能是多个选项，所以使用List获取值

for (Object x : a) { //遍历，使用Object类型(所有类型的父类)接收并遍历

System.out.println(x); //输出

}

## 文本组件

文本框

单行文字，显示或编辑单行文本

构造方法有：

默认文字、长度

JTextField t = new JTextField("候选字", 30);

//方法

t.setText("123"); //默认文字

t.setColumns(20); //长度 单位字符

t.setFont(new Font("楷体",Font.BOLD,20)); //设置字体样式 实例化一个字体样式对象(字体，属性，大小)

t.requestFocus(); //在监听事件内，使用这个方法，将焦点和光标返回到文本框内

t.getText(); //获取文本框内文本

实例

JTextField t = new JTextField("候选字", 50); //创建文本框

t.setText("123"); //设置默认文字

t.setColumns(20); //设置长度，单位字符

t.setFont(new Font("楷体",Font.BOLD,15)); //设置字体样式

JButton b= new JButton("提交"); //创建按钮

b.addActionListener(new ActionListener() { //按钮点击事件

public void actionPerformed(ActionEvent e) {

System.out.println(t.getText()); //输出文本框内容

t.requestFocus(); //点击按钮后焦点在按钮上，通过这个方法返回到文本框内，光标也返回到文本框内

}

});

c.add(t);

c.add(b); //添加组件

密码框

同文本框，唯一不同的是输入内容可以更换成字符(回显字符)

构造方法：长度，默认字符

JPasswordField p=new JPasswordField();

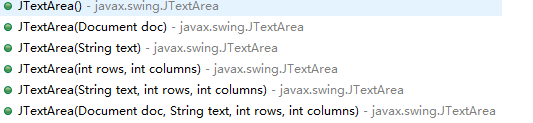
常用方法跟文本框一样，但可以通过setEchoChar方法设置回显字符

p.setEchoChar('0'); //这个值是char类型，记得使用单引号

这时在密码框中输入的所有字符都显示0

然后在监听事件里获取输出密码框内容

String s= new String(p.getPassword()); //密码框内容是字符数组，通过String的方法转换成字符串

文本域

可以输入多行文字的框

构造方法：默认文本、文本域的长宽

## 事件监听

事件监听分为动作和焦点

动作包含点击 点按 松开 滑动等，当某个组件被点击后，监听器铺捉到动作，就会执行相对于的方法

焦点分为窗口焦点和组件焦点，切换窗口就是切换焦点，输入框未选中时没有焦点，选中后获得焦点并显示光标

动作事件适用于JButton JList JTextField等

焦点事件适用于Component及其派生类

动作事件监听

为b1按钮添加点击事件，传统写法

b1.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

// TODO 自动生成的方法存根

}

});

当然，也可以这样写

public class OnClick extends JFrame{ //主类

JLabel jl = new JLabel(); //显示文字的标签，用于显示点击事件，放外面的原因是要使用内部类

public OnClick() { //构造方法

//设置窗口属性

setTitle("点击事件");

setSize(700, 550);

setLocationRelativeTo(null);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

//创建窗口容器

Container c =getContentPane();

c.setLayout(new FlowLayout());

//创建一个按钮

JButton b2= new JButton("按钮");

b2.addActionListener(new b2action()); //实例化内部类

//添加组件到窗口，显示窗口可见

c.add(b2);

c.add(jl);

setVisible(true);

}

class b2action implements ActionListener{ //内部类，实现ActionListener接口，并重写接口方法

public void actionPerformed(ActionEvent e) {

jl.setText("点击了按钮");

}

}

焦点事件监听

为文本框添加焦点监听

失去焦点时执行focusLost方法

获得焦点时执行focusGained方法，注意，两个方法不要冲突

public class OnClick extends JFrame{

JLabel jl = new JLabel();

public OnClick() {

//设置窗口属性

setTitle("点击事件");

setSize(700, 550);

setLocationRelativeTo(null);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

//创建窗口容器

Container c =getContentPane();

c.setLayout(new FlowLayout());

//创建一个文本框

JTextField jt = new JTextField("",10);

// 创建按钮点击事件

jt.addFocusListener(new FocusListener() { //创建文本框焦点事件

//失去焦点时调用

public void focusLost(FocusEvent e) {

JOptionPane.showMessageDialog(null, "文本框失去焦点"); //弹出一个提示框

}

//获取焦点时调用

public void focusGained(FocusEvent e) {

}

});

c.add(jt);

setVisible(true);

}

public static void main(String[] args) {

new OnClick();

}

}

# 集合类

## 集合类概述

位于java.util包下

集合类似于数组，数组存放的是基本数据类型，集合存放的则是引用数据类型，数组长度创建的时候就固定了，集合则可以变化

常见的集合有List（列表），Set（集合），Map集合（键值对），

不同的集合有不同的存储对象方式

## Collection接口

Collection里的东西称为元素

List和Set继承了Collection接口，所以Collection中的方法在List和Set集合中都是通用的

isEmpty(); //判断集合是否为空

add(); //添加元素

size(); //返回集合元素个数

remove(); //移除元素

Collection collection = new ArrayList(); //创建一个集合(数组列表)

System.out.println("是否为空" + collection.isEmpty()); // true

collection.add("第一个字符串");

collection.add("第二个字符串");

collection.add("待移除字符串");

System.out.println("元素个数" + collection.size()); //3

collection.remove("待移除字符串");

System.out.println("是否为空" + collection.isEmpty()); //false

System.out.println("元素个数" + collection.size()); //2

和数组一样,集合也有遍历,只不过要用到迭代器Iterator

Iterator iterator = collection.iterator();

while(iterator.hasNext()){ //hasNext判断是否有下一个元素,没有就退出循环

System.out.println(iterator.next()); //next返回当前元素,返回的类型是Object

}

以上方法对所有的List和Set集合都通用

## List集合

List集合包括List接口和List接口的所有类

元素允许重复，顺序就是插入元素时的顺序，可以通过索引（下标）来寻找元素并访问

List有两种储存方法ArrayList（数组列表），LinkedList（链表）

List有独自的方法,如上面所说,可以通过索引找到元素

list.set(0,null); // 将指定索引位置的元素改成指定元素

list.get(0); //获取指定索引的元素

ArrayList

查找:类似于数组,每个元素都有下标,便于随机索引查找

增删:插入元素时,需要改变后续元素索引,不便于插入元素

LinkedList

查找:元素之间直接链接,没有索引,需要程序挨个往下数,不利于随机索引查找

增删:插入元素时,直接改变指定索引前后的指向

List list = new ArrayList(); //实例化数组集合

list.add("添加的第一个元素"); //增

list.remove("添加的第一个元素"); //删

list.add("添加的第二个元素");

System.out.println(list.get(0)); //查

list.set(0,null); //改

## Set集合