Java基础目录

[1. java编译运行： 1](#_Toc8153)

[2. 一个整数的二进制表达 1](#_Toc7866)

[3. Java Scanner 1](#_Toc15113)

[4. switch 1](#_Toc14712)

[5. 数组 1](#_Toc10635)

[6. 赋值数组 System.arraycopy() 1](#_Toc9954)

[7. 增强型for循环 2](#_Toc22302)

[8. 二维数组 2](#_Toc23253)

[9. Java的随机数 2](#_Toc10973)

[10. Arrays是针对数组的工具类，可以进行 排序 查找 赋值填充等功能 2](#_Toc8171)

[11. 类类型传参 3](#_Toc27967)

[12. 类属性（静态属性） static 4](#_Toc11257)

[13. 属性初始化 5](#_Toc14099)

[14. 单例模式 5](#_Toc19871)

[15. 枚举类型 6](#_Toc31073)

[16. 对象转型 7](#_Toc8987)

[17. 重写 隐藏 7](#_Toc27994)

[18. Super 7](#_Toc3334)

[19. Object类 8](#_Toc29883)

[20. Final 9](#_Toc25505)

[21. 内部类 9](#_Toc30286)

[22. 默认方法 11](#_Toc27610)

[23. 数字与字符串的转换 12](#_Toc24431)

[24. 数学方法 12](#_Toc12597)

[25. 格式化输出 13](#_Toc31423)

[26. 字符串转换为字符数组 13](#_Toc21563)

[27. 字符串比较 14](#_Toc3436)

[28. StringBuffer是可变长的字符串 14](#_Toc6427)

[29. Date 14](#_Toc30645)

1. **java编译运行：**

.java 不可运行 -> 编译 -> .class

编译：java 使用 javac filename.java生成filename.class

运行：java filename

1. **一个整数的二进制表达**

**int** a = 1125;  
System.***out***.println(Integer.*toBinaryString*(a));

1. **Java Scanner**

**import** java.util.Scanner;

Scanner s = **new** Scanner(System.***in***);

**int** a = s.nextInt(); *//读入整数*String str = s.nextLine(); *//读入回车*str = s.nextLine(); *//读入一行字符串 包含空格***float** f = s.nextFloat(); *//读入浮点数*

1. **switch**

**switch**(op){  
 **case** 1:  
 System.***out***.println(1);  
 **break**;  
 **case** 2:  
 System.***out***.println(2);  
 **break**;  
 **case** 3:  
 System.***out***.println(3);  
 **break**;  
 **case** 4:  
 System.***out***.println(4);  
 **break**;  
 **case** 5:  
 System.***out***.println(5);  
 **break**;  
}

1. **数组**

**int** []a; *//声明数组*a = **new int**[5]; *//创建数组***int** []b = **new int**[10];

System.***out***.println(a.**length**); *//打印数组的长度 .length属性*

1. **赋值数组 System.arraycopy()**

**int** a [] = **new int**[]{18,62,68,82,65,9};  
**int** b[] = **new int**[3];*//分配了长度是3的空间，但是没有赋值  
//System.arraycopy(src, srcPos, dest, destPos, length)  
//src: 源数组  
//srcPos: 从源数组复制数据的起始位置  
//dest: 目标数组  
//destPos: 复制到目标数组的启始位置  
//length: 复制的长度*System.*arraycopy*(a, 0, b, 0, 3);

1. **增强型for循环**

**int** values[] = {1, 2, 3, 4, 5};  
**for**(**int** each : values){  
 System.***out***.println(each);  
}

1. **二维数组**

*//初始化二维数组，***int**[][] a = **new int**[2][3]; *//有两个一维数组，每个一维数组的长度是3*a[1][2] = 5; *//可以直接访问一维数组，因为已经分配了空间  
//只分配了二维数组***int**[][] b = **new int**[2][]; *//有两个一维数组，每个一维数组的长度暂未分配*b[0] =**new int**[3]; *//必须事先分配长度，才可以访问*b[0][2] = 5;  
*//指定内容的同时，分配空间***int**[][] c = **new int**[][]{  
 {1,2,4},  
 {4,5},  
 {6,7,8,9}  
};

1. **Java的随机数**

*//1. Random类*Random random = **new** Random();  
System.***out***.println(random.nextInt());  
System.***out***.println(random.nextFloat());  
*//2. Math.random()方法：默认产生大于等于0.0且小于1.0之间的随机double型随机数*System.***out***.println(Math.*random*());

1. **Arrays是针对数组的工具类，可以进行 排序 查找 赋值填充等功能**
2. 数组复制 *copyOfRange(int[] original, int from, int to)*

**int** a[] = **new int**[] {0, 1, 2, 3, 4};  
**int** b[] = Arrays.*copyOfRange*(a, 0, 3); *// 第一个参数表示源数组  
// 第二个参数表示开始位置(取得到)  
// 第三个参数表示结束位置(取不到)***for**(**int** it : b){  
 System.***out***.println(it);  
}*//输出: 0 1 2*

1. 转换为字符串 Arrays.toString()

**int** a[] = **new int**[] {0, 1, 2, 3, 4};  
String str = Arrays.*toString*(a);  
System.***out***.println(str);  
*//输出: [0, 1, 2, 3, 4]*

1. 排序 Arrays.sort()

**int** a[] = **new int**[]{18, 62, 68, 82, 65, 9};  
Arrays.*sort*(a); *//排序*System.***out***.println(Arrays.*toString*(a));

1. 搜索 （二分查找 需要先排序才能搜索）

如果数组中有多个相同的元素，查找结果是不确定的

**int** a[] = **new int**[]{18, 62, 68, 82, 65, 9};  
Arrays.*sort*(a); *//排序*System.***out***.println(Arrays.*binarySearch*(a, 18)); *//从a中找18*

1. 判断是否相同 Arrays.equals(a, b);

**int** a[] = **new int**[] { 18, 62, 68, 82, 65, 9 };  
**int** b[] = **new int**[] { 18, 62, 68, 82, 65, 8 };  
System.***out***.println(Arrays.*equals*(a, b)); *//false*

1. 填充 ： 使用同一个值，填充整个数组 Arrays.fill()

**int** a[] = **new int**[5];  
Arrays.*fill*(a, 2);  
System.***out***.println(Arrays.*toString*(a)); *//[2, 2, 2, 2, 2]*

1. **类类型传参**

**public class** Hero {  
 String **name**; *// 姓名* **float hp**; *// 血量* **float armor**; *// 护甲* **int moveSpeed**; *// 移动速度* **public** Hero(String name, **float** hp) {  
 **this**.**name** = name;  
 **this**.**hp** = hp;  
 }  
 *// 攻击一个英雄，并让他掉damage点血* **public void** attack(Hero hero, **int** damage) {  
 hero.**hp** = hero.**hp** - damage;  
 }  
 **public static void** main(String[] args) {  
 Hero teemo = **new** Hero(**"提莫"**, 383);  
 Hero garen = **new** Hero(**"盖伦"**, 616);  
 garen.attack(teemo, 100);  
 System.***out***.println(teemo.**hp**); *//283* }  
}

因为teemo和hero虽然是不同的引用但是指向了同一个对象，所以再attack()中的操作可以改变对象的hp属性

相比之下

**public class** Hero {  
 String **name**; *//姓名* **float hp**; *//血量* **float armor**; *//护甲* **int moveSpeed**; *//移动速度* **public** Hero(){ }  
 **public** Hero(String name,**float** hp){  
 **this**.**name** = name;  
 **this**.**hp** = hp;  
 }  
 *//复活* **public void** revive(Hero h){  
 h = **new** Hero(**"提莫"**,383);  
 }  
 **public static void** main(String[] args) {  
 Hero teemo = **new** Hero(**"提莫"**,383);  
 *//受到400伤害，挂了* teemo.**hp** = teemo.**hp** - 400;  
 teemo.revive(teemo);  
 System.***out***.println(teemo.**hp**); *// 输出多少？ -17* }  
}

因为revive()中的h重新指向一个新的对象，所以原本teemo指向的对象的hp属性没有变化。

1. **类属性（静态属性） static**

类属性：静态属性 所有对象都共享一个值

对象属性：实例属性，非静态属性

访问：1.对象.类属性 2.类.类属性

1. **属性初始化**

对象属性初始化有3种  
1. 声明该属性的时候初始化  
2. 构造方法中初始化  
3. 初始化块

执行顺序： 声明初始化 -> 初始化块 -> 构造方法中初始化

初始化块：

**public class** Hero {  
 **public** String **name**;  
 **float hp**;  
 **static float** *maxHP*;  
 *//初始化块* {  
 **hp** = 200;  
 }  
 *//静态初始化块* **static**{  
 *maxHP* = 400;  
 }  
}

1. **单例模式**

面试:什么是单例模式？

1.构造方法私有化

2.静态属性指向实例

3.public static的getInstance方法，返回第二步的静态属性

**饿汉式**是立即加载的方式，无论是否会用到这个对象，都会加载。  
如果在构造方法里写了性能消耗较大，占时较久的代码，比如建立与数据库的连接，那么就会在启动的时候感觉稍微有些卡顿。

**public class** Hero{  
 **private** Hero(){}  
 **private static** Hero *h* = **new** Hero();  
 **public static** Hero getInstance(){  
 **return** *h*;  
 }  
}

**懒汉式**，是延迟加载的方式，只有使用的时候才会加载。 并且有[线程安全](https://how2j.cn/k/thread/thread-synchronized/355.html" \l "step793)的考量(鉴于同学们学习的进度，暂时不对线程的章节做展开)。  
使用懒汉式，在启动的时候，会感觉到比饿汉式略快，因为并没有做对象的实例化。 但是在第一次调用的时候，会进行实例化操作，感觉上就略慢。

**public class** Hero{  
 **private** Hero(){}  
 **private static** Hero *h*;  
 **public static** Hero getInstance(){  
 **if**(*h* == **null**) *h* = **new** Hero();  
 **return** *h*;  
 }  
}

看业务需求，如果业务上允许有比较充分的启动和初始化时间，就使用饿汉式，否则就使用懒汉式。

1. **枚举类型**

枚举enum是一种特殊的类，使用枚举可以很方便的定义常量

**public enum** Season {  
 ***SPIRNG***,***SUMMER***,***AUTUMN***,***WINTER***}

**public class** Main{  
 **public static void** main(String args[]){  
 Season s = Season.***AUTUMN***;  
 **switch** (s){  
 **case *SPIRNG***:  
 System.***out***.println(**"Spring"**);  
 **break**;  
 **case *SUMMER***:  
 System.***out***.println(**"Summer"**);  
 **break**;  
 **case *AUTUMN***:  
 System.***out***.println(**"Autumn"**);  
 **break**;  
 **case *WINTER***:  
 System.***out***.println(**"Winter"**);  
 **break**;  
 }  
 }  
}

枚举：  **Season.values()**

**public class** Main{  
 **public static void** main(String args[]){  
 **for**(Season it : Season.*values*()){  
 System.***out***.println(it);  
 }  
 }  
}

1. **对象转型**

引用类型和对象类型

Hero h = new Hero();

引用类型：h的类型 为Hero

对象类型：右边对象的类型 也为Hero

转型：引用类型和对象类型不一致时，需要类型转换

1. 子类转父类（向上转型） √
2. 父类转子类（向下转型） 有的时候行，有的时候不行，所以必须进行强制转换。
3. 无继承关系转换 ×

**Instanceof**

instanceof Hero 判断一个引用所指向的对象，是否是Hero类型，或者Hero的子类

**public class** Main{  
 **public static void** main(String args[]){  
 ADHero ad = **new** ADHero();  
 Hero h = ad;  
 System.***out***.println(h **instanceof** Hero); *//true* System.***out***.println(h **instanceof** ADHero); *//true* }  
}

1. **重写 隐藏**

重写：针对对象方法

隐藏：针对类方法 static

Hero h =new ADHero();

h.battleWin(); 会调用父类的类方法

1. **Super**

无参：

**public class** Hero {  
 **public** String **name**;  
 **protected float hp**;  
 Hero(){}  
}

**public class** ADHero **extends** Hero{  
 ADHero(){}  
}

**ADHero实例化时，会先调用父类的构造方法（默认调用无参），再调用子类的构造方法。**

带参数的构造方法：需要显式的用super调用

**public class** Hero {  
 **public** String **name**;  
 **protected float hp**;  
 Hero(){}  
 Hero(String name){  
 **this**.**name** = name;  
 }  
}

**public class** ADHero **extends** Hero{  
 ADHero(){}  
 ADHero(String name){  
 **super**(name);  
 }  
}

访问父类属性：

**public class** ADHero **extends** Hero{  
 ADHero(){}  
 ADHero(String name){  
 **super**(name);  
 }  
 **public** String getName(){  
 **return super**.**name**;  
 }  
}

也可以通过super调用被重写的父类的方法。

1. **Object类**

Object类是所有类的父类

1. toString() 方法 返回当前对象的字符串表达

**public class** Main{  
 **public static void** main(String args[]){  
 ADHero ad = **new** ADHero();  
 System.***out***.println(ad.toString()); *//output: ADHero@6d6f6e28* }  
}

1. finalize()

**public** **void** finalize(){

    System.out.println("这个英雄正在被回收");

}

1. equals() 用于判断两个对象的内容是否相同
2. ==

这不是Object的方法，但是用于判断两个对象是否相同  
更准确的讲，用于判断两个引用，是否指向了同一个对象

1. hashCode() 6. 线程同步相关方法 7. getClass()
2. **Final**
3. final修饰类 表示不能被继承
4. final修饰方法 表示不能被重写
5. final修饰基本类型变量 表示该变量只有一次被赋值的机会
6. final修饰引用 表示该引用只有1次指向对象的机会
7. 常量 直接访问，不会变化的值
8. **内部类**

内部类分为四种：

1. 非静态内部类

语法：new 外部类().new 内部类()

非静态内部类，可以直接访问外部类的private属性

**public class** Hero {  
 **public** String **name**;  
 **protected float hp**;  
 **class** BattleScore{  
 **int kill**;  
 **int die**;  
 **int assist**;  
  
 **public void** legendary(){  
 **if**(**kill** >= 8) System.***out***.println(**name** + **"超神"**);  
 **else** System.***out***.println(**name** + **"尚未超神"**);  
 }  
 }  
 Hero(){}  
 Hero(String name){  
 **this**.**name** = name;  
 }  
  
 **public static void** main(String args[]){  
 Hero garen = **new** Hero(**"garen"**);  
 BattleScore score = garen.**new** BattleScore(); //创建非静态内部类  
 score.**kill** = 9;  
 score.legendary();  
 }  
}

1. 静态内部类

语法：new 外部类.静态内部类();

因为没有一个外部类的实例，所以在静态内部类里面**不可以访问外部类的实例属性和方法**

**public class** Hero {  
 **public** String **name**;  
 **protected float hp**;  
  
 **private static void** battleWin(){  
 System.***out***.println(**"battle win"**);  
 }  
 *//敌方的水晶* **static class** EnemyCrystal{  
 **int hp**=5000;  
 *//如果水晶的血量为0，则宣布胜利* **public void** checkIfVictory(){  
 **if**(**hp**==0){  
 Hero.*battleWin*();  
 *//静态内部类不能直接访问外部类的对象属性  
 //System.out.println(name + " win this game"); name无法访问* }  
 }  
 }  
  
 **public static void** main(String[] args) {  
 *//实例化静态内部类* Hero.EnemyCrystal crystal = **new** Hero.EnemyCrystal();  
 crystal.checkIfVictory();  
 }  
  
}

1. 匿名类

**public abstract class** Hero {  
 **public** String **name**;  
 **protected float hp**;  
  
 **public abstract void** attack();  
  
 **public static void** main(String args[]){  
 Hero h = **new** Hero(){  
 **public void** attack(){  
 System.***out***.println(**"hero attack"**);  
 }  
 };  
 h.attack();  
 System.***out***.println(h);  
 }  
}

1. 本地类 (可以理解为有名字的匿名类)

public abstract class Hero {

String name; //姓名

float hp; //血量

float armor; //护甲

int moveSpeed; //移动速度

public abstract void attack();

public static void main(String[] args) {

//与匿名类的区别在于，本地类有了自定义的类名

class SomeHero extends Hero{

public void attack() {

System.out.println( name+ " 新的进攻手段");

}

}

SomeHero h =new SomeHero();

h.name ="地卜师";

h.attack();

}

}

1. **默认方法**
2. 什么是默认方法

默认方法是JDK8新特性，指的是接口也提供具体方法了，而不像以前，只提供抽象方法

**public interface** mortal {  
 **default void** die(){  
 System.***out***.println(**"die"**);  
 }  
}

1. 为什么会有默认方法

假设没有默认方法这种机制，那么如果要为Mortal增加一个新的方法revive，那么所有实现了Mortal接口的类，都需要做改动。

但是引入了默认方法后，原来的类，不需要做任何改动，并且还能得到这个默认方法。

通过这种手段，就能够很好的扩展新的类，并且做到不影响原来的类。

1. **数字与字符串的转换**
2. 数字 -> 字符串
   1. 使用String类的静态方法valueOf
   2. 先把基本类型装箱 为对象，然后调用对象的toString

**public class** Main{  
 **public static void** main(String args[]){  
 **int** i = 5;  
 String str = String.*valueOf*(i);  
  
 Integer it = i;  
 String str2 = it.toString();  
 }  
}

1. 字符串 -> 数字

调用Integer的静态方parseInt

**public class** Main{  
 **public static void** main(String args[]){  
 String str = **"999"**;  
 **int** i = Integer.*parseInt*(str);  
 System.***out***.println(i);  
 }  
}

1. **数学方法**

**public class** Main {  
 **public static void** main(String[] args) {  
 **float** f1 = 5.4f;  
 **float** f2 = 5.5f;  
 System.***out***.println(Math.*round*(f1));*//5.4四舍五入即5* System.***out***.println(Math.*round*(f2)); *//5.5四舍五入即6* System.***out***.println(Math.*random*()); *//得到一个0-1之间的随机浮点数（取不到1）* System.***out***.println((**int**)( Math.*random*()\*10)); *//得到一个0-10之间的随机整数 （取不到10）* System.***out***.println(Math.*sqrt*(9)); *//开方* System.***out***.println(Math.*pow*(2,4)); *//次方（2的4次方）* System.***out***.println(Math.***PI***); *//π* System.***out***.println(Math.***E***); *//自然常数* }  
}

1. **格式化输出**

%s 表示字符串  
%d 表示数字  
%n 表示换行

**public class** Main{  
 **public static void** main(String args[]){  
 String name = **"garen"**;  
 **int** kill = 8;  
 String title = **"legendary"**;  
  
 String sentence = **"%s %d %s %n"**;  
 System.***out***.printf(sentence, name, kill, title);  
 }  
}

1. **字符串转换为字符数组**

**public class** Main{  
 **public static void** main(String args[]){  
 String str = **"abc123"**;  
 **char** s[] = str.toCharArray();  
 }  
}

1. **字符串比较**

**public class** Main {  
 **public static void** main(String[] args) {  
 String str1 = **"the light"**;  
 String str2 = **new** String(str1);  
 *//==用于判断是否是同一个字符串对象* System.***out***.println( str1 == str2); *//false  
 //equals()用于比较内容* System.***out***.println(str1.equals(str2)); *//true* }  
}

1. **StringBuffer是可变长的字符串**

追加 删除 插入 反转

**public class** Main {  
 **public static void** main(String[] args) {  
 String str1 = **"let there "**;  
 StringBuffer sb = **new** StringBuffer(str1); *//根据str1创建一个StringBuffer对象* sb.append(**"be light"**); *//在最后追加* System.***out***.println(sb);  
 sb.delete(4, 10);*//删除4-10之间的字符* System.***out***.println(sb);  
 sb.insert(4, **"there "**);*//在4这个位置插入 there* System.***out***.println(sb);  
 sb.reverse(); *//反转* System.***out***.println(sb);  
 }  
}

1. **Date**

**import** java.util.Date;  
  
**public class** Main {  
 **public static void** main(String[] args) {  
 Date d1 = **new** Date();  
 System.***out***.println(d1); *//Sun Jan 26 20:06:44 CST 2020* Date d2 = **new** Date(5000);  
 System.***out***.println(d2); *//Thu Jan 01 08:00:05 CST 1970*

}  
}