

## Java期末考试复习大纲

### 期末考试题型

### 章节知识点归纳

#### 第一章 Java历史特点和开发环境

Java程序的编译和运行过程

Java跨平台和Java虚拟机

Java的程序规范

#### 第二章 Java语言编程基础

标识符命名规范，掌握常用关键字

数据类型

补充：包装类

运算符

false&&true||的短路原则

区分=、==和equals函数

控制结构

#### 第三章 类和对象

类的基本概念和语句

类的修饰符、方法和属性的修饰符

包的作用

java.util.\*

java.io.\*

java.lang.\*

Java.swing

#### 第四章 继承和多态

类的继承

super关键字

继承中构造方法的关系

方法重写（override）

final关键字

类的多态

对象的向上转型

abstract类

接口

Object类

包装类以及字符串的转换

#### 第五章 数组与字符串

数组的定义，默认值，异常

数组的操作

foreach语句遍历数组

String StringBuilder StringBuffer的创建与使用

String类常用方法

#### 第六章 异常处理

异常类的分类，异常类的层次结构

异常的捕获

自定义异常类

#### 第七章 文件和流

File类的概念和使用方法

1、构造方法

2、File类创建和删除功能

3、File类的判断功能

4、File类的获取功能和修改名字功能

5、File 类的其他获取功能

字节流与字符流

使用FileInputStream类

- 使用FileOutputStream类
- FileWriter类 与 BufferedWriter类使用
- FileReader类 与 BufferedReader类使用
- FileOutputStream类 以及 ObjectOutputStream类的使用
- FileInputStream类 以及 ObjectInputStream类的使用
- Date类
- RandomAccess类
  - 简介
  - 创建对象
  - 字节数据读写操作
  - 文件指针操作
- Scanner类

第八章 集合和泛型

- 泛型
- 接口Collection和Map区别
- List与Set的区别
- HashSet与ArrayList
- HashMap
- 迭代器

第九章 线程

- 线程创建方式1
- 线程创建方式2 调用runable接口
- 匿名内部类创建线程
  - 例题
  - 补充
- 同步安全:
- 线程状态

# Java期末考试复习大纲

---

## 期末考试题型

---

选择题（20小题，每题1.5分，共30分）

程序阅读题（每题4分，4小题，共16分）

程序填空题（4题，每空2分，共24分）

程序设计题（3题，共30分）

## 章节知识点归纳

---

### 第一章 Java历史特点和开发环境

#### Java程序的编译和运行过程

Java程序由编译器进行编译，产生了一种中间代码，称为Java字节码（Java bytecodes）。字节码是Java虚拟机（Java virtual machine, JVM）的代码，是平台无关的中性代码，不能在各计算机上运行，必须在JVM上运行。Java解释器是JVM的实现，它把字节码转换为底层平台的机器码，使Java程序最终得以运行。

Myprogram.java→编译器（compiler）→Myprogram.class→解释器→（0010011110....） My program

在任何平台上，通过该平台的Java编译器把Java程序编译成字节码，读字节码可以在任何平台的JVM运行

Java api是一个很大的Java类库集合，这些类以包（package）的形式组织，它们提供了丰富的功能。

Java的简单性：去掉指针，取消重继承和运算符重载，内存管理移向Java内嵌的自动内存回收机制。

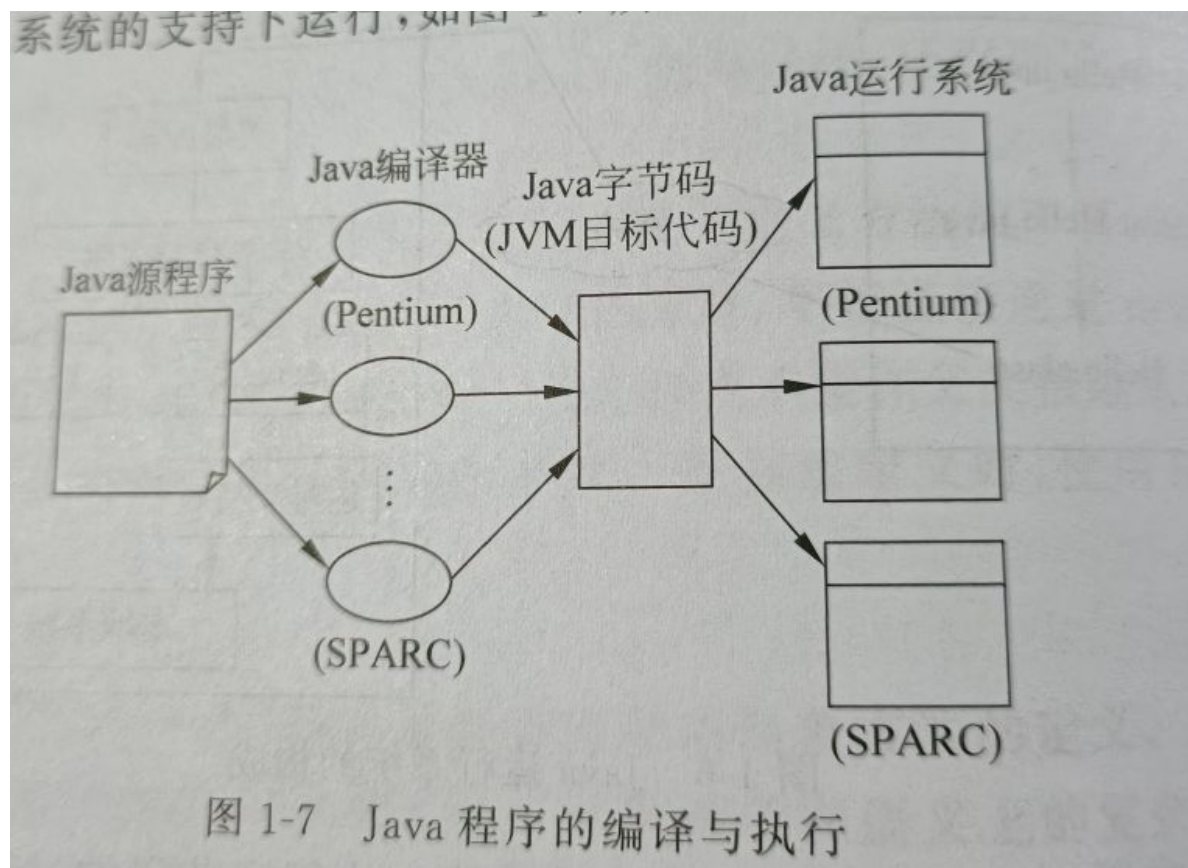
面向对象：Java的对象有模块化性质和信息隐藏能力，满足面向对象的封装要求。Java支持面向对象的继承性。同时，Java通过抽象类和内部接口支持面向对象的多态性要求，即一个对外接口，多种内部实现。

Java具有支持分布式计算的特征，分布的两层含义：1.数据分布2.操作分布

## Java跨平台和Java虚拟机

Java运行系统运行的是字节码即.class文件。执行字节码的三步：1.代码的装入2.代码的验证3.代码的执行

Java虚拟机JVM是一个能运行Java字节码的操作平台，JVM规范说明，包括指令系统、字节码格式等。有了这样的虚拟机规范，才使Java应用达到平台无关。



## Java的程序规范

源程序结构包括三部分

- 1.包声明语句package定义了该源程序中类所存放的包，一个源程序只能有一个或没有
- 2.引入类的声明语句import，引入jdk的标准类或其他已有类。程序中该语句可以没有或有很多条

3.类和接口的定义public class , interface一个源程序只能有一个public class 可以有任意数目的class 和interface

java的程序开发工具一般选择Sun的JDK为主要工具，JDK需要程序编程软件配合使用，一般采用两种方式

1.用普通文本编辑器后，使用命令行对其进行编译

javac program.class编译

java program运行

## 第二章 Java语言编程基础

### 标识符命名规范，掌握常用关键字

标识符命名规则：

- 1.标识符是以字母，\_，或\$开始的一个字符序列
- 2.数字不能作为标识符的第一个字符
- 3.标识符不能是java的关键字，但可以将关键字作为标识符的一部分
- 4.标识符大小写敏感，且无长度限定

常用关键字：

abstract	assert	boolean	break	byte
case	catch	char	class	const
continue	default	do	double	else
enum	extends	final	finally	float
for	goto	if	implements	import
instanceof	int	interface	long	native
new	package	private	protected	public
return	strictfp	short	static	super
switch	synchronized	this	throw	throws
transient	try	void	volatile	while

### 数据类型

l（基本类型8种，大小，对应的包装类，初始值），基本类型和引用类型的区别

逻辑型：boolean（java中的boolean不对应0和1，true与false在机器中只占1位）

文本型：char

整型：byte，short，int，long

浮点型：double，float

char: 16位 Character

转义字符: java转义字符[北街风的博客-CSDN 博客](https://blog.csdn.net/qg_45047809/article/details/112553953)java转义符  
[https://blog.csdn.net/qg\\_45047809/article/details/112553953](https://blog.csdn.net/qg_45047809/article/details/112553953)

byte: 8位  $2^7-1$  到  $2^8-1$

short: 16位  $2^{15}$  到  $2^{16}-1$

int: 32位  $2^{31}$  到  $2^{32}-1$  Interger

long: 64位  $2^{63}$  到  $2^{64}-1$

八进制以0为前导, 十六进制整数以0x为前导, long在数值后加L

float 32位

double 64位

自动转换: 当小数据转换成大数据时, 系统会自动转换

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int i = 1;
6      char a = 'a'; /* ascii 值是 99 */
7      float sum;
8
9      sum = i + a;
10     printf("%f\n", sum );
11
12 }
```

复制

解释: a首先被转换为整数, 但是由于最后的值是 float 型的, 所以会应用常用的算术转换, 编译器会把 i 和 a 转换为浮点型, 并把它们相加得到一个浮点数。

强制转换: 大数据传小数据时, 转换后可能会导致溢出或精度下降

```
int sum = 17, count = 5; double avg; avg = (double) sum / count;
```

无论是自动类型转换还是强制类型转换, 都只是为了本次运算而进行的临时性转换, 转换的结果也会保存到临时的内存空间, 不会改变数据本来的类型或者值。

```
// java八种基本数据类型之"double"在作为类的成员变量时,其初始值为:0.0;取值范围为:[4.9E-324,1.7976931348623157E308];对应的封装类为:Double
// java八种基本数据类型之"float"在作为类的成员变量时,其初始值为:0.0;取值范围为:[1.4E-45,3.4028235E38];对应的封装类为:Float
// java八种基本数据类型之"long"在作为类的成员变量时,其初始值为:0;取值范围为:[-9223372036854775808,9223372036854775807];对应的封装类为:Long
// java八种基本数据类型之"int"在作为类的成员变量时,其初始值为:0;取值范围为:[-2147483648,2147483647];对应的封装类为:Integer
// java八种基本数据类型之"char"在作为类的成员变量时,其初始值为:'';取值范围为:['','和汉字的口字很相似,无法打印出来'];转换成数字后为:[0,65535];对应的封装类为:Character
// java八种基本数据类型之"short"在作为类的成员变量时,其初始值为:0;取值范围为:[-32768,32767];对应的封装类为:Short
// java八种基本数据类型之"byte"在作为类的成员变量时,其初始值为:0;取值范围为:[-128,127];对应的封装类为:Byte
// 八种基本数据类型按容量大小升序排列结果为:byte<short<char<int<long<float<double
```

## 基本类型和引用类型的区别

基本类型: 八种

引用类型: Array,Class,Interface

### 1. 数据的内存存储的方式不同

基本类型存在栈中,引用类型在栈中存的是堆地址,具体数据存于堆中

### 2. 使用方式不同

基本类型可以使用所有的运算符进行处理

引用类型只可以使用运算符==和!=,但是可以使用方法和属性

### 3. 作为方法参数时的效果不同

基本类型:实参不会受到方法内部处理的影响

引用类型:实参会受到方法内部处理的影响

原因与数据存储的方式不同有关

### 4. 作为有finally块的方法返回值时,效果不同

原因与第3点相同

-----  
版权声明: 本文为CSDN博主「亦昕跑的码」的原创文章, 遵循CC 4.0 BY-SA版权协议, 转载  
请附上原文出处链接及本声明。

原文链接: [https://blog.csdn.net/yiXin\\_Chen/article/details/120028172](https://blog.csdn.net/yiXin_Chen/article/details/120028172)

## 补充: 包装类

Integer tg = 100;—>底层自动装箱 Integer tg =new Integer(100);

Java为每个基本数据类型提供了一个对应的包装类, 使其具有

包装类的一些用途:

1.集合不允许存放基本数据类型, 故常用包装类

2.包含了每种基本类型的相关属性, 如最大值、最小值、所占位等

3.作为基本数据类型对应的类类型, 提供了一系列实用的对象操作

包装类初始值为null

```

public class TestInteger {
    public static void main(String[] args) {
        int i = 128;
        Integer i2 = 128;
        Integer i3 = new Integer(128);
        System.out.println(i == i2); //Integer会自动拆箱为 int, 所以为 true
        System.out.println(i == i3); //true, 理由同上
        Integer i4 = 127; //编译时被翻译成: Integer i4 = Integer.valueOf(127);
        Integer i5 = 127;
        System.out.println(i4 == i5); //true
        Integer i6 = 128;
        Integer i7 = 128;
        System.out.println(i6 == i7); //false
        Integer i8 = new Integer(127);
        System.out.println(i5 == i8); //false
        Integer i9 = new Integer(128);
        Integer i10 = new Integer(123);
        System.out.println(i9 == i10); //false
    }
}

```

为什么i4和i5比是true，而i6和i7比是false呢？关键就是看valueOf()函数了，这个函数对于-128到127之间的数，会进行缓存，Integer i4 = 127时，会将127进行缓存，下次再写Integer i5 = 127时，就会直接从缓存中取，就不会new了。所以i4和i5比是true，而i6和i7比是false。  
@澄静英逸13

而对于后边的i5和i8，以及i9，i10，因为对象不一样，false

总结：

- 1.integer和new integer不会相等，不经历拆箱过程，new出来的对象放到堆中，非new的则在常量池，内存地址不同，所以false
- 2.两个都是非new出来的integer，如果数在-128——127之间，则true否则false，integer i2=128，相当于integer i2=integer.valueOf(128);而valueOf函数会对-128到127的数会缓存
- 3.int 和 integer对比都为true 会自动拆箱

包装类有内置函数可供使用

## 运算符

赋值运算符=

算术运算符：+ - \* / % +=, -=, \*=, %=, ++,

关系运算符：== != < > =

逻辑运算符: && || !

位运算符：& |

## false&&和true || 的短路原则

前错&&不运行后面

## 区分=、==和equals函数

=是赋值

带你区别equals和== - 知乎

<https://zhuanlan.zhihu.com/p/58126578>

==是对数值的一种比较

equals在重写前调用==（不能用于基本数据类型）

在一些库中重写了equals（String,Date,ArrayList),这样就不再是去比较堆内存中的存放地址了



```
Test.java MainC.java String.class Object.class
970      * @return {@code true} if the given object represents a {@code String}
971      *         equivalent to this string, {@code false} otherwise
972      *
973      * @see #compareTo(String)
974      * @see #equalsIgnoreCase(String)
975      */
976  public boolean equals(Object anObject) {
977      if (this == anObject) {
978          return true;
979      }
980      if (anObject instanceof String) {
981          String anotherString = (String)anObject;
982          int n = value.length;
983          if (n == anotherString.value.length) {
984              char v1[] = value;
985              char v2[] = anotherString.value;
986              int i = 0;
987              while (n-- != 0) {
988                  if (v1[i] != v2[i])
989                      return false;
990                  i++;
991              }
992              return true;
993          }
994      }
995      return false;
996  }
997
```

知乎 @冷都男

String类中的equals是逐一对比两者

String s1="HELLO"

String s2=new String ("HELLO")

String s3=s2

s1==s2 false

s1==s3 false

s2==s3 true

s1.equals (s2) true

s1.equals (s3) true

s2.equals (s2、3) true

## 控制结构

I 控制结构 (if、switch、for、while、do while、break、continue)

(3条消息)Java之控制结构. 弗兰克的博客-CSDN/博客java控制结构

<https://blog.csdn.net/z2768557792/article/details/122604161>

不多做介绍

## 第三章 类和对象

### 类的基本概念和语句



类的基本概念和语句：类、成员变量（属性）、成员函数（方法）、对象、new、构造函数及其定义要领、关键字this的概念和用法

类名必须是合法的标识符

类体中描述属性的变量称它们为成员变量，它是出现在类体的内部，方法的外部，成员变量的定义语法格式：

```
public class Pet { //类声明， Pet为类名
    //属性描述--使用成员变量的描述 name, strength, mood
    String name;
    int strength;
    int mood;

    • public Pet(String name, int strength, int mood) {
    •     this.name = name;
    •     this.strength = strength;
    •     this.mood = mood;
    • }
    //行为描述--使用成员方法实现
    • public void introduce() {
    •     System.out.print("我的名字是"+this.name+"，我的体力值时" + this.strength +
    •     "，我
    •         的心情值为" + this.mood);
    • }

}
-
```

方法：

```
public void eat(){
    System.out.println("我要吃饭!!!");
}

public String getNation(String nation){
    System.out.println("当前地点为" + nation);
}
```

返回值

如果方法有返回值，则必须在方法声明时，指定返回值的类型。同时，方法中，需要使用return关键字来返回指定类型的变量或常量：return 数据。

如果方法没返回值，则方法声明时，使用void来表示。通常，没返回值的方法中，就不需要使用return。如果使用的话，只能使用return;表示结束此方法的意思。

权限修饰符 返回值类型 方法名(形参列表){

方法体

}

(3条消息) 超详解Java类的概念(很全) Hon\_csu的博客-CSDN博客java类的概念  
<https://blog.csdn.net/shiqiuke/article/details/109262072>

## 方法的重载

### 4.1 重载的概念

在同一个类中，允许存在一个以上的同名方法，只要它们的参数个数或者参数类型不同即可。

重载实例：

```
// 举例一：Arrays类中重载的sort() / binarySearch(); PrintStream中的println()
// 举例二：
// 如下的4个方法构成了重载
public void getSum(int i,int j){
    System.out.println("1");
}

public void getSum(double d1,double d2){
    System.out.println("2");
}

public void getSum(String s ,int i){
    System.out.println("3");
}

public void getSum(int i,String s){
    System.out.println("4");
}
```

不构成重载：

```
// 如下的3个方法不能与上述4个方法构成重载
public int getSum(int i,int j){
    return 0;
}

public void getSum(int m,int n){
}

private void getSum(int i,int j){
}
```

构造函数：

java构造函数的三种类型总结 - 百度文库

<https://wenku.baidu.com/view/6d234d396f175f0e7cd184254b35eefdc8d31586.html>

java类的结构（属性、方法、构造函数） - JavaShuo

<http://www.javashuo.com/article/p-gjojtlji-ba.html>

#### 1.无参构造函数

```
public class MyClass{
    private String name ="Jerry";
    private int id=1;
    public MyClass(){

    }
}
```

无参构造器方法体里对类成员变量初始化

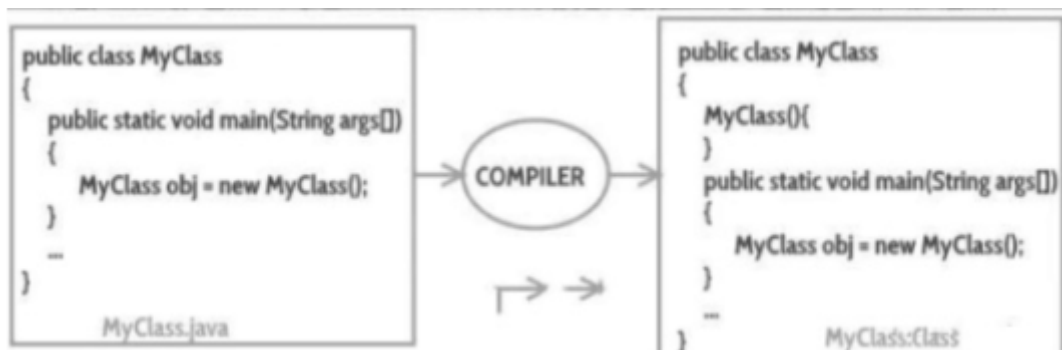
```
public class MyClass{
    private String name;
    private int id;
    public MyClass(){
        this.name="Jerry";
        this.id=1;
    }
}
```

## 2.有参构造函数

```
class Test {
    String name;
    Test(){
        name="韦小宝";
    }
    Test(String myName){
        this.name=myName;
    }
    public static void main(String[] args){
        Test t1=new Test();
        System.out.println(t1.name);
        Test t2=new Test("康熙");
        System.out.println(t2.name);
    }
}
```

## 3.默认构造函数

与无参相似，方法体为空。



this关键字

1.关键字this是一个变量，一个引用，保存当前对象的内存地址，指向自身。

所以严格意义来讲，this代表的就是“当前对象”。

this储存在堆内存中对象的内部

2.关键字this可以使用在实例方法中，谁调用这个实例方法，this就是谁。

所以this代表的是当前对象。

3.关键字this也可以使用在构造器中，大部分情况下可以省略，但是用来区分局部变量和实例变量的时候不能省略。

4.this();这种语法只能出现在构造器的第一行，表示当前构造器调用本类其他的构造器，目的是代码复用。

<https://blog.csdn.net/HXL521666/article/details/119063023>

```
public class Person{
    String name;
    int age;
    public Person(){
        this("Jack",32); //调用本类其他构造器
    }
    public Person(String name,int age){
        this.name = name; //用this来区分局部变量和实例变量
        this.age = age;
    }
}
```

## 类的修饰符、方法和属性的修饰符

类的修饰符、方法和属性的修饰符：static、final、public、protected、private、default(指不写修饰符)

类修饰符：

1.public 一个java文件可以写多个类，但是只能有一个类用public进行修饰（主类），且这个类必须和文件名保持一致，保证包中的类都可以访问

2.abstract 用abstract修饰符修饰类表示该类是一个抽象类。所谓抽象类是指没有实例的抽象概念类（没有具体对象）

**一个抽象类中可以没有抽象方法，但是有抽象方法的类必须是一个抽象类**

3.final（最终类修饰符）如果一个类被final进行修饰表示该类是最终类，不能被继承

4.类缺省（没有public）访问控制符：如果一个类没有访问控制符，说明它具有缺省的访问控制符特性。表示该类只能被本包中的其他类来进行访问。这一访问特性又称包访问性。

属性修饰符：

## 权限修饰符

	public	protected	默认	private
同一类中	√	√	√	√
同一包子类、其他类	√	√	√	
不同包子类	√	√		
不同包其他类	√			

## 包的作用

包：包的作用、package、import、常用几个包的熟悉（java.lang包、java.util包、java.io包）

Java.applet 提供创建applet小程序所需要的类

Java.awt 包含用于创建用户界面和绘制图形图像的所有类

Java.io 提供与输入输出相关的类

Java.beans 包含与开发javaBeans相关的类

Java.lang 提供java语言程序设计的基础类

Java.net 提供实现网络操作相关的类

Java.nio 为输入输出提供缓冲区的类

Java.text 提供处理文本、日期、数字和消息的类和接口

Java.util 提供处理日期、时间、随机数生成等各种使用工具的类

Javax.net 提供用于网络应用程序的类、网络应用扩展类

Java.swing 提供一组与AWT功能相同的纯java的组件类

原文链接：[https://blog.csdn.net/dream\\_go123/article/details/119583382](https://blog.csdn.net/dream_go123/article/details/119583382)

### java.util.\*

包含**集合框架**、**遗留的 collection 类**、事件模型、日期和时间设施、国际化和各种实用工具类（字符串标记生成器、随机数生成器和位数组、日期Date类、**堆栈Stack类**、向量Vector类等）。集合类、时间处理模式、日期时间工具等各类常用工具包

### java.io.\*

Java的核心库java.io提供了全面的IO接口。包括：文件读写、标准设备输出等。Java中IO是以流为基础进行输入输出的，所有数据被串行化写入输出流，或者从输入流读入。

### java.lang.\*

提供利用 Java 编程语言进行程序设计的基础类。最重要的类是 Object（它是类层次结构的根）和 Class（它的实例表示正在运行的应用程序中的类）。

该包提供了Java语言进行程序设计的基础类，它是默认导入的包。该包里面的Runnable接口和 Object、Math、String、StringBuffer、System、Thread以及Throwable类需要重点掌握，因为它们应用很广。

Java.lang包是java语言体系中其他所有类库的基础，已经内嵌到java虚拟机中，而且以对象的形式创建好了，所以，我们在使用java.lang包时不需要再使用import将其导入，可以直接使用java.lang包中的所有类以及直接引用某个类中的成员、变量和操作方法。

## 第四章 继承和多态

### 类的继承

只要创建了一个类，就隐式继承Object父类，如果指定父类则继承于父类，而父类继承于Object类  
extends继承

class 子类名 extends 父类名 {}

```
class Father{

    public void feature(){

        System.out.println("父亲的特征 ");

    }

}

class Son extends Father {

}
```

如果Son没有实现自己的方法的话，那么默认就是用的是父类的 feature方法。如果子类实现了自己的 feature方法，那么就相当于是重写了父类的 feature方法，这也是我们上面提到的重写了。

Java只支持单继承与多层继承（是多层）

### super关键字

this 代表本类对应的引用

Super 代表父亲存储空间的标识（可以理解为父亲引用）可以操作父亲 的成员

A:调用成员变量

·this.成员变量 调用本类的成员变量

·super.成员变量 调用父类的成员变量

B:调用构造方法

·this (...) 调用本类的成员变量

·Super (...) 调用父类的成员变量

C:调用成员方法

·this.成员方法 调用本类的成员方法

·super.成员方法 调用父类的成员方法

## 继承中构造方法的关系

A: 子类中所有的构造方法默认都会访问父类中空参数的构造方法

B: 理由:

因为子类会继承父类中的数据, 可能还会使用父类的数据

所以, 子类初始化之前, 一定要先完成父类数据的初始化

**注意:** 子类每一个构造方法的第一条语句默认都是: super()

如果父类没有无参构造方法, 那么子类的构造方法会出现什么现象?

报错

解决方法:

A: 在父类中加一个无参构造方法

B: 通过使用super关键字去显示的调用父类的带参构造方法

C: 子类通过this去调用本类的其他构造方法

子类中一定要有一个去访问父类的构造方法, 否则父类数据就没有初始化。

**注意事项:**

this(...)或者super(...)必须出现在第一条语句上。

否则就可能对父类的数据进行了多次初始化

原文链接: [https://blog.csdn.net/weixin\\_30041067/article/details/113413209](https://blog.csdn.net/weixin_30041067/article/details/113413209)

## 方法重写 (override)

方法重写 (又叫方法的覆盖): 子类中出现了和父类中方法声明一模一样的方法

(区别于方法重载, 方法重载仅仅是方法名一样)

子类重写的应用: 当子类需要父类的功能, 而功能主体子类又有自己特有的内容时, 这样既沿袭了父类的功能, 又定义了子类特有的功能

**方法重载和方法重写区别:** 重载发生在同一个类内部的两个或多个方法。重写发生在父类与子类之间。

super.方法名(参数名); //避免重复代码

## final关键字

**final 关键字**声明类可以把类定义为不能继承的, 即最终类;

或者用于修饰方法, 该方法不能被子类重写:

Final修饰变量的初始化时机

·被final修饰的变量只能赋值一次

·在构造方法完毕前 (非静态的常量)

## 类的多态



多态指的是同一个行为具有不同表现形式。是指一个类实例（对象）的相同方法在不同情形下具有不同表现形式。封装和继承是多态的基础，也就是说，多态只是一种表现形式

多态实现的三种充要条件

·继承

·重写父类方法

·父类引用指向子类对象

```
public class Fruit {
    int num;
    public void eat(){
        System.out.println("eat Fruit");
    }
}
public class Apple extends Fruit{
    @Override
    public void eat() {
        super.num = 10;
        System.out.println("eat " + num + " Apple");
    }
    public static void main(String[] args) {
        Fruit fruit = new Apple();
        fruit.eat();
    }
}
```

Fruit fruit = new Apple();, Fruit类型的对象指向了apple对象的引用，这就是多态：父类引用子类对象，因为Apple 继承于Fruit，并且重写了eat方法，所以能够表现出来多种状态的形式。

## 对象的向上转型

概念：一个子类的对象b赋值给其父类的对象a，父类对象a就是子类对象b的上转型对象

```
A a; //A是父类    a是父类对象
B b=new B(); //B是子类    b是子类对象
a=b;    //子类对象赋值给父类对象    a在上面    所以称之为子类对象的上转型对象
```

## abstract类

```
abstract class A{
}
}
```

对于抽象类不能直接实例化对象，即不能使用new运算符创建该类的对象，只能先创建其子类，由子类创建对象

抽象类可以声明对象，作为子类对象的上转型对象

和普通的类相比，抽象类里可以有抽象方法。也可以没有抽象方法。

对于abstract方法，值允许声明，不允许实现

抽象方法不允许使用static final修饰

含有抽象方法的类一定是抽象类

原文链接：[https://blog.csdn.net/weixin\\_43908626/article/details/105639546](https://blog.csdn.net/weixin_43908626/article/details/105639546)

## 抽象类的实现

```
abstract class A{//定义一个抽象类

    public void fun(){//普通方法
        System.out.println("存在方法体的方法");
    }

    public abstract void print();//抽象方法，没有方法体，有abstract关键字做修饰

}
```

## 抽象类的使用

```
abstract class A{//定义一个抽象类

    public void fun(){//普通方法
        System.out.println("存在方法体的方法");
    }

    public abstract void print();//抽象方法，没有方法体，有abstract关键字做修饰

}

public class TestDemo {

    public static void main(String[] args) {
        A a = new A();
    }

}
```

### 运行:

```
1 | Exception in thread "main" java.lang.Error: Unresolved compilation problem:
2 |     Cannot instantiate the type A
3 |
4 |     at com.wz.abstractdemo.TestDemo.main(TestDemo.java:14)
```

从上可知，A是抽象的，无法直接进行实例化操作。为什么不能直接实例化呢？当一个类实例化之后，就意味着这个对象可以调用类中的属性或者放过了，但在抽象类里存在抽象方法，而抽象方法没有方法体，没有方法体就无法进行调用。既然无法进行方法调用的话，又怎么去产生实例化对象呢。

### 抽象类的使用原则如下

- (1) 抽象方法必须为public或者protected（因为如果为private，则不能被子类继承，子类便无法实现该方法），缺省情况下默认为public；
- (2) 抽象类不能直接实例化，需要依靠子类采用向上转型的方式处理；
- (3) 抽象类必须有子类，使用extends继承，一个子类只能继承一个抽象类；

(4) 子类（如果不是抽象类）则必须覆写抽象类之中的全部抽象方法（如果子类没有实现父类的抽象方法，则必须将子类也定义为为abstract类。）；

例子：

```
abstract class A{//定义一个抽象类

    public void fun(){//普通方法
        System.out.println("存在方法体的方法");
    }

    public abstract void print();//抽象方法，没有方法体，有abstract关键字做修饰
}
//单继承
class B extends A{//B类是抽象类的子类，是一个普通类

    @Override
    public void print() { //强制要求覆写
        System.out.println("Hello world !");
    }

}
public class TestDemo {

    public static void main(String[] args) {
        A a = new B();//向上造型

        a.fun();//被子类所覆写的过的方法
    }

}
```

运行结果：

存在方法体的方法

版权声明：本文为CSDN博主「LoveDestiny」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：[https://blog.csdn.net/Liveor\\_Die/article/details/77882174](https://blog.csdn.net/Liveor_Die/article/details/77882174)

原文链接：[https://blog.csdn.net/Liveor\\_Die/article/details/77882174](https://blog.csdn.net/Liveor_Die/article/details/77882174)

## 接口

接口的存在也是为了弥补类无法多继承的缺点，假设一个情况，父类--Animal 子类--Dog、Cat、People、Sheep、Tiger、Lion。假设在 Animal 中都存在 eat () 这个公有的方法。但是 Tiger 和 Lion、People 还拥有 Hunt 的方法，但是hunt 不存在在 Dog / Cat中，所以需要对 Tiger Lion People 上面新建一个 接口。用于特定标识他们公有的。

- 使用interface来定义一个接口

```
public interface InterfaceName extends fatherInterface{

}
```

实现接口

通过关键字implements声明自己实现一个或多个接口

```
class A implements B,C{
    .....
}
```

1. 如果实现接口的类不是抽象类，那么这个类必须实现该接口的所有方法
2. 接口可以通过继承产生新的接口

对于接口，里面的组成只有抽象方法和全局常量，所以很多时候为了书写简单，可以不用写public abstract 或者public static final。并且，接口中的访问权限只有一种：public，即：定义接口方法和全局常量的时候就算没有写上public，那么最终的访问权限也是public，注意不是default。

[https://blog.csdn.net/Liveor\\_Die/article/details/77882174](https://blog.csdn.net/Liveor_Die/article/details/77882174)

```
public interface Award {
    //接口当中能定义的变量,默认都是public static final
    int DOUBLE_FIRE=0;//双倍火力
    int LIFE=1;//生命加1

    int getType();
}
```

(见shootgame)

## Object类

Object是所有Java的祖先类，位于java.lang包，理解自定义类中重写Object的toString () 方法、equals () 方法和hashCode()方法的作用

重写Object类的toString和equals方法\_地瓜 我是土豆的博客-CSDN博客

[https://blog.csdn.net/weixin\\_45812481/article/details/102807629](https://blog.csdn.net/weixin_45812481/article/details/102807629)

```
public class UserTest04 {
    public static void main(String[] args) {
        User u1=new User("张飞",41);
        User u2=new User("张飞",41);
        System.out.println(u1);
        System.out.println(u2);
        System.out.println(u1.equals(u2));
    }
}

class User{
    String name;
    int age;

    public User(String name, int age) {    //带参构造器
        super();
        this.name = name;
        this.age = age;
    }
}
```

```

    public String toString() {    //重写toString方法
        return name+age+"岁";    //返回值是个字符串，当前对象的姓名加年龄
    }

    public boolean equals(Object obj) {    //重写equals方法
        if(obj instanceof User) {        //obj判断是否是User的对象，向下转型
            User u=(User)obj;
            return (this.name).equals(u.name) && this.age == u.age;    //分别比较姓名和年龄
        }
        return false;
    }
}

```

再来个例子吧= =（因为我们是手考，实在记不住重写equals和toString（被idea惯坏的人））

```

public class Student {
    private String name;
    private String age;

    public Student() {}

    public Student(String name,String age) {
        this.setName(name);
        this.setAge(age);
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getAge() {
        return age;
    }

    public void setAge(String age) {
        this.age = age;
    }

    //重写toString方法
    public String toString() {
        return name+","+age;
    }

    //重写equals方法
    public boolean equals(Object obj) {

        if (obj == this)
            return true;
    }
}

```

```

//如果传进来的obj是调用equals方法的对象本身，return true

if (obj == null)
    return false;
//如果传进来的obj是null，return false

//判断传进来的obj是否是Student对象，防止出现类型转换的异常
if (obj instanceof Student) {
    //instanceof关键字 对比左边的对象是否属于右边的对象
    //结果为 boolean 类型，如果左右两边为同类型返回 true，否则返回 false
    Student s = (Student) obj;
    boolean flag = this.name.equals(s.name) && this.age == s.age;
    return flag;
}
return false;
}

}

public class object {
    public static void main(String[] args) {
        Student s = new Student("学生", "18");
        //public String toString() 返回对象的字符串表达形式
        //在使用对象直接输出的时候，默认输出的是一个对象在堆内存上的地址值；
        //如若要输出该对象的内容，则要重写toString()方法
        System.out.println(s.toString());
        Student ss = new Student("学生", "18");
        //public boolean equals(Object obj) 比较对象是否相等，默认比较地址，重写可以比
较内容
        System.out.println(s.equals(ss));
    }
}

```

可以参考一下

子类重写Object类中的toString、equals、hashCode方法\_钱甫新的博客-CSDN博客  
[https://blog.csdn.net/Mr\\_Qian\\_lves/article/details/110424332](https://blog.csdn.net/Mr_Qian_lves/article/details/110424332)

## 包装类以及字符串的转换

包装类详见第二章第三节

字符串转换

数字转字符串

```

package test;

public class lianxi {
    public static void main(String[] args) {
        int i = 3;
        String str = String.valueOf(i); //方法1
        Integer i1 = i;
        String str2 = i1.toString(); //方法2
    }
}

```

字符串转数字

```

package test;

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

```

实例

```

public class test {

    public static void main(String[] args) {
        float i = 3.14f;
        //方法1
        String str = String.valueOf(i);
        System.out.println(str);
        //方法2
        Float f = i;
        String str2 = f.toString();
        System.out.println(str2);
        //字符串转换为浮点数
        float i2 = Float.parseFloat(str);
        System.out.println(i2);
        //问题
        String str3 = "3.1a4";
        float i3 = Float.parseFloat(str3);
        System.out.println(i3); //这里会报错 java.lang.NumberFormatException

    }
}

```

## 第五章 数组与字符串

### 数组的定义，默认值，异常

Java中数组的定义

(1)适用于不使用初始化的数组。当数组特别长时，不初始化，值都是默认值。



数据类型[] 数组名 = new 数据类型[数组长度]

例如

```
int[ ] a= new int[3];
```

创建一个长度为3的int类型的数组

(2)适合直接初始化数组。

数据类型[] 数组名 = {1,2,3}

数组的长度由大括号内元素的个数决定

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

(3)匿名数组适用于直接向方法传输参数。

语法

数组类型 [] 数组名=new 数组类型 [] {数组0, 数组1, 数组2, .....};

```
int[ ] arr=new int[] {0,1,2,3}
```

数组初始值均为0;

二维数组的创建

```
int [][] number = new int[5][5];

int [][] nuber = {{1,2,3},{4,5,6},{7,8}};
//二维数组就是在一个大的数组内,还包含了二级数组,可以理解成嵌套的意思
//若要输出二维数组,也可以采用多个for循环的方式
int [][] nuber = {{1,2,3},{4,5,6},{7,8,9}};
    for(int i =0;i<nuber.length;i++){
        for (int j = 0;j<nuber.length;j++){
            system.out.println(nuber[i][j]);
        }
    }
```

数组中常见的异常:

1.数组角标越界异常: ArrayIndexOutOfBoundsException

2.空指针异常: NullPointerException

## 数组的操作

数组的操作 (给数组元素赋值, 打印数组元素的值、数组元素排序, 数组中找一个数等)、使用Arrays类的常用方法进行数组操作

```
// 变量声明
int num;
// 变量初始化
```

```

num = 10;
// 变量声明 + 初始化
int id = 1001;

// 数组声明
int[] ids;
// 静态初始化:数组的初始化和数组元素的赋值操作同时进行
ids = new int[]{1001,1002,1003,1004};
// 动态初始化:数组的初始化和数组元素的赋值操作分开进行
String[] names = new String[5];

// 错误的写法:
// int[] arr1 = new int[];
// int[5] arr2 = new int[5];
// int[] arr3 = new int[3]{1,2,3};

//也是正确的写法:
int[] arr4 = {1,2,3,4,5}; // 类型推断

// 总结: 数组一旦初始化完成, 其长度就确定了。

```

Java数组详解*我是波哩个波的博客*CSDN博客Java数组

<https://blog.csdn.net/ligonglanyuan/article/details/120200169>

数组的赋值:

```

// 创建一个长度为5的数组,用来存储学生的姓名
String[] names = new String[5];

// 给数组的指定位置赋值
names[0] = "张三";
names[1] = "李四";
names[2] = "王五";
names[3] = "马六";
names[4] = "田七";
// 数组长度5, 下标从0开始, 到4结束, 使用下标5会抛出异常。
// names[5] = "钱八";

// 调用数组的指定位置的元素
System.out.println("下标2的学生的姓名是:"+names[2]);

```

获取数组的长度

```

int[] ids = new int[]{1001,1002,1003,1004};
String[] names = new String[5];

System.out.println(names.length); // 5
System.out.println(ids.length); // 4

```

数组的遍历

```
String[] names = {"张三", "李四", "王五", "马六", "田七"};
/*System.out.println(names[0]);
System.out.println(names[1]);
System.out.println(names[2]);
System.out.println(names[3]);
System.out.println(names[4]);*/

for(int i = 0; i < names.length; i++){
    System.out.println(names[i]);
}
```

## Java数组排序

最简单的就是使用Arrays.sort

```
int[] arr1 = {21,16,34,5,20};
Arrays.sort(arr1);
for(int i=0;i<arr1.length;i++){
    System.out.print(arr1[i]+" ");
}
```

## 其他排序方式

冒泡排序、选择排序、反转排序

见

如何将Java数组进行排序？ 列举五种常见的方法

<https://m.w3cschool.cn/article/60055624.html>

```
public class TestArray {
    public static void main(String[] args) {
        //定义一个数组，用静态初始化完成数组元素的初始化
        int[] arr = new int[]{39,80,33,77,2};
        //键盘录入要查找的数据，用一个变量接收。
        Scanner sc = new Scanner(System.in);

        System.out.println("请输入您要查找的数据：");
        int num = sc.nextInt();
        //定义一个索引变量，初始值为-1
        //假设要查找的数据，在数组中是不存在的
        int index = -1;
        //遍历数组，获取到数组中的每一个元素
        for (int i = 0; i < arr.length; i++) {
            //如果值相同，就把该值对应的索引赋值给索引变量，并结束循环
            if (num == arr[i]){
                index = arr[i];
                break;
            }
        }
        System.out.println(index);
    }
}
```

## Arrays

```

package com.ilzy.blog.array;
import java.util.Arrays;
public class ArrayTest6 {
    public static void main(String[] args) {
        int[] array1 = {1,2,3,4,7878,353,324568,6234};
        System.out.println(array1);    // 直接打印是数组对象的hash值。
        Arrays.sort(array1);    //Arrays类自带的排序方法
        System.out.println(Arrays.toString(array1));    //Arrays类自带的打印数组方法。
        // printArray(array1);    //自己写的打印方法
        //Arrays.fill(array1,0);    //用0填充数组，即将指定的值赋给数组内每个元素。
        Arrays.fill(array1,1,4,0);    //用0填充数组中下标1到下标4之间的元素。    原则是左闭
        // 右开区间，即包含左不包含右。
        System.out.println(Arrays.toString(array1));
    }
    // 打印数组元素方法
    public static void printArray(int[] array1){
        System.out.print("[");
        for (int i = 0; i < array1.length; i++) {
            System.out.print(array1[i]);
            if (i<array1.length-1){
                System.out.print(", ");
            }
        }
        System.out.print("]");
    }
}

```

## foreach语句遍历数组

[foreach](#)的语句格式：

```

for(元素类型t 元素变量x : 遍历对象obj){
    引用了x的java语句;
}

```

foreach比for的好处和弊端

好处:相对于for来说方便了对容器的遍历

弊端:没有索引,不能操作元素中的元素

```

// 声明并初始化int数组
int[] numbers = { 43, 32, 53, 54, 75, 7, 10 };
System.out.println("----for each----");
// for-each语句
for (int item : numbers) {
    System.out.println("Count is:" + item);
}

```

## String StringBuilder StringBuffer的创建与使用

1.String、Stringbuffer、StringBuilder的异同

String：不可变的字符序列；底层使用char[]存储

StringBuffer：可变的字符序列;线程安全的，效率低；底层使用char[]存储

在实际开发当中,我们经常会使用到字符串连接的操作,如果用String来操作,则使用"+"号来完成字符串的连接操作.

注意:使用String连接字符串,代码性能会非常低,应为String的内容不可变.

解决这个问题的方法是使用StringBuffer.

```
package com.dream;
/**
 *2022年3月5日
 *@author Dream
 *@description
 *StringBuffer类解决String的性能问题
 */
public class StringBufferDemo {

    public static void main(String[] args) {
        // TODO StringBuffer类解决String的性能问题
        String a="a";
        String b="b";
        String c=a+b+1;//创建了四个对象
        System.out.println(c);

        String a1="a"+1+2+3+"b";//a123b:常量相加没有性能问题,创建一个对象
        System.out.println(a1);

        //StringBuffer目的是用来解决字符串相加时带来的性能问题(常量与变量)
        //StringBuffer的内部实现采用字符数组,默认数组的长度是16,超过数组大小时,动态扩充的算法是
        //所以当我们预知要添加的数据长度时,建议使用带初始化容量的构造方法,来避免动态扩充的次数,从而提高效率
        //synchronized线程安全的,影响性能
        StringBuffer sb=new StringBuffer();
        sb.append(a).append(b).append(1);
        System.out.println(sb.toString());
        //转换成StringBuffer类
        System.out.println(new StringBuffer(a1));
        String a2="1234DemoD567890";
        CharSequence cs=a2.subSequence(2, 8);
        System.out.println(new StringBuffer(cs));

        StringBuffer sb1=new StringBuffer(a2);
        //删除
        System.out.println(sb1.delete(0, 5));
        //查询
        System.out.println(sb1.indexOf("D"));
        //插入
        System.out.println(sb1.insert(0, "true"));
        //替换
        System.out.println(sb1.replace(1, 5, "fangfafaf"));
        //截取
        System.out.println(sb1.substring(5));
        //反转
        System.out.println(sb1.reverse());

    }

}
```

StringBuffer的常用方法:

StringBuffer append(xxx):提供了很多的append()方法, 用于进行字符串拼接  
StringBuffer delete(int start, int end):删除指定位置的内容  
StringBuffer replace(int start, int end, String str):把[start,end)位置替换为str  
StringBuffer insert(int offset, xxx):在指定位置插入xxx  
StringBuffer reverse():把当前字符序列逆转

StringBuilder: 可变的字符序列; 线程不安全, 效率高; 底层使用char[]存储, 适合在单线程中使用

```
package com.dream;
/**
 *2022年3月5日
 *@author Dream
 *@description
 *
 */
public class StringBuilder {

    public static void main(String[] args) {

        StringBuilder sb=new StringBuilder();

        /**
         * 字符串相加操作
         * 1.多个常量相加没有性能问题,在编译期优化
         * 2.变量与常量相加,会产生多个垃圾对象
         */
        /**
         * String a="a"+1;
         * String b=a+"b";
         */

        String c=null;
        for(int i=0;i<5;i++) {
            c+=i;
            //每次循环产生一个StringBuilder对象实现拼接,性能低,最好是手动创建
            //StringBuilder来拼接
        }
        System.out.println(c);
        //字符串相加,在变异后,会使用StringBuilder来优化代码,实现拼接

    }

}
```

- 因为StringBuilder 相比StringBuffer 有速度优势, 所以大多数情况下建议使用StringBuilder 类。但当对应用程序有线程安全要求时, 就必须使用StringBuffer 类。

## String类常用方法

String类型是我们在开发中常见的数据类型，也是我们使用最多的数据类型，String类提供了很多操作字符串的方法，但是我们只要掌握十多个即可，我们可以通过官方api文档学习(不建议使用中文)，你要习惯去阅读英文的文档。

### 1、public char charAt(int index)

[根据索引取得指定位置上的字符]

```
package com.baidu.demo;
public class Hello {
    public static void main(String [] args) {
        String str="Hello world";
        System.out.println(str.charAt(0));
    }
}
```

H

字符串的索引是0开始计算。

### 2、public boolean endsWith(String suffix)

[判断字符串是否以指定的内容结束，如果是返回true，否则返回false]

```
package com.baidu.demo;
public class Hello {
    public static void main(String [] args) {
        String str="Hello world";
        System.out.println(str.endsWith("ld"));
    }
}
```

true

该方法区别大小写

### 3、public byte[] getBytes()



[将一个字符串转换成一个byte类型的数组返回]

```
package com.baidu.demo;
public class Hello {
    public static void main(String [] args) {
        String str="Hello world";
        byte [] bts=str.getBytes();
        System.out.println("字节数组的长度是: "+bts.length);
    }
}
```

字节数组的长度是: 11

其中空格也算

#### 4、public int indexOf(String str)

[查询出指定内容在字符串中第一次出现的位置，如果没有指定的值返回-1]

```
package com.baidu.demo;
public class Hello {
    public static void main(String [] args) {
        String str="Hello world";
        System.out.println(str.indexOf("l"));
    }
}
```

2

#### 5、public String intern()

[让字符串入池，转移到String 常量池]

```

package com.baidu.demo;
public class Hello {
    public static void main(String [] args) {
        String str1="Hello world";
        String str2=new String("Hello world");
        String str3=str2.intern();
        System.out.println(str1==str3);
    }
}

```

true

## 6、public boolean isEmpty()

[判断字符串的长度，如果长度为0返回true否则返回false]

```

package com.baidu.demo;
public class Hello {
    public static void main(String [] args) {
        String str="";
        System.out.println(str.length());
        System.out.println(str.isEmpty());
    }
}

```

0  
true

## 7、public int lastIndexOf (String str)

[返回指定内容在字符串中最后一次出现的位置，如果没有出现返回-1]

```
package com.baidu.demo;
public class Hello {
    public static void main(String [] args) {
        String str="Hello world";
        System.out.println(str.lastIndexOf("o"));
    }
}
```

7

## 8、public boolean matches(String regex)

[使用指定正则匹配字符串，如果匹配上则返回true，否则返回false]

```
package com.baidu.demo;
public class Hello {
    public static void main(String [] args) {
        String str="Hello world";
        System.out.println(str.matches("Hello world"));
    }
}
```

<terminated> Hello [Java App  
true

参数还可以是正则表达式，正则的内容我们后面会花两节课时间讲解。

## 9、public String[] split(String regex)

[把字符串按照指定的字符串拆分，拆分之后保存到一个字符串数组返回]

```
package com.baidu.demo;
public class Hello {
    public static void main(String [] args) {
        String str="Hello world";
        String [] strs=str.split("");
        for(String temp:strs) {
            System.out.print(temp+" ");
        }
    }
}
```

```
<terminated> Hello [Java Application] C:\Program File
Hello world
```

```
package com.baidu.demo;
public class Hello {
    public static void main(String [] args) {
        String str="Hello world";
        String [] strs=str.split("o");
```

```
        for(String temp:strs) {
            System.out.print(temp+" ");
        }
    }
}
```

```
<terminated> Hello [Java
Hell w rld
```

## 10、 public String substring (int beginIndex)

[截取字符串，从指定的索引截取到最后，参数: beginIndex开始的索引下标]

```
package com.baidu.demo;
public class Hello {
    public static void main(String [] args) {
        String str="Hello world";
        System.out.println(str.substring(1));
    }
}
```

```
<terminated> Hello [Java Ap
ello world
```

该方法切割的时候包括了开始的索引下标。

## 11、public String substring(int beginIndex, int endIndex)

[截取字符串，从开始索引|截取到结束的索引]

```
package com.baidu.demo;
public class Hello {
    public static void main(String [] args) {
        String str="Hello world";
        System.out.println(str.substring(1,3));//输出多少? e1 ell
    }
}
```



包括开始的索引不包括结束的索引。

## 12、public String toLowerCase()

[将所有的字母转换成小写]

```
package com.baidu.demo;
public class Hello {
    public static void main(String [] args) {
        String str="Hello world";
        System.out.println(str.toLowerCase());
    }
}
```

```
}  
  
hello world  
  
package com.baidu.demo;  
public class Hello {  
    public static void main(String [] args) {  
        String str="Hello world";  
        System.out.println(str.toLowerCase());  
        System.out.println(str);  
    }  
}  
}
```

```
<terminated> Hello [Java Application] C:\  
hello world  
Hello world
```

[返回上一页](#)

使用该方法的时候不影响原字符串的内容，你可以使用这种方式验证一下其他方法。

在学习java的过程当中有遇见任何问题，可以加入我的Java交流学习群697888503 多多交流问题，互帮互助，群里有不错的学习教程和电子书，面试题等。

### 13、public String toUpperCase()

[将所有的字母转换成大写]

```
package com.baidu.demo;  
public class Hello {  
    public static void main(String [] args) {  
        String str="Hello world";  
        System.out.println(str.toUpperCase());  
        System.out.println(str);  
    }  
}  
}
```

```
<terminated> Hello [Java Application] C:\  
HELLO WORLD  
Hello world
```


该方法对原字符串还是没有影响。

## 14、 public String trim()

[删除字符串两端的空格，返回删除空格后的字符串]

```
package com.baidu.demo;
public class Hello {
    public static void main(String [] args) {
        String str=" Hello world ";
        System.out.println(str.length());
        System.out.println(str.trim().length());
        System.out.println(str.length());
    }
}
```

```
}
}
```



```
16
11
16
```

### 总结:

1、要掌握String类常用的方法

2、如果我们要分析一个方法(根据api文档或者是别人写的方法)，要从方法的返回值和方法的参数入手。

摘自 14个String 类的常用方法\_ITPUB博客

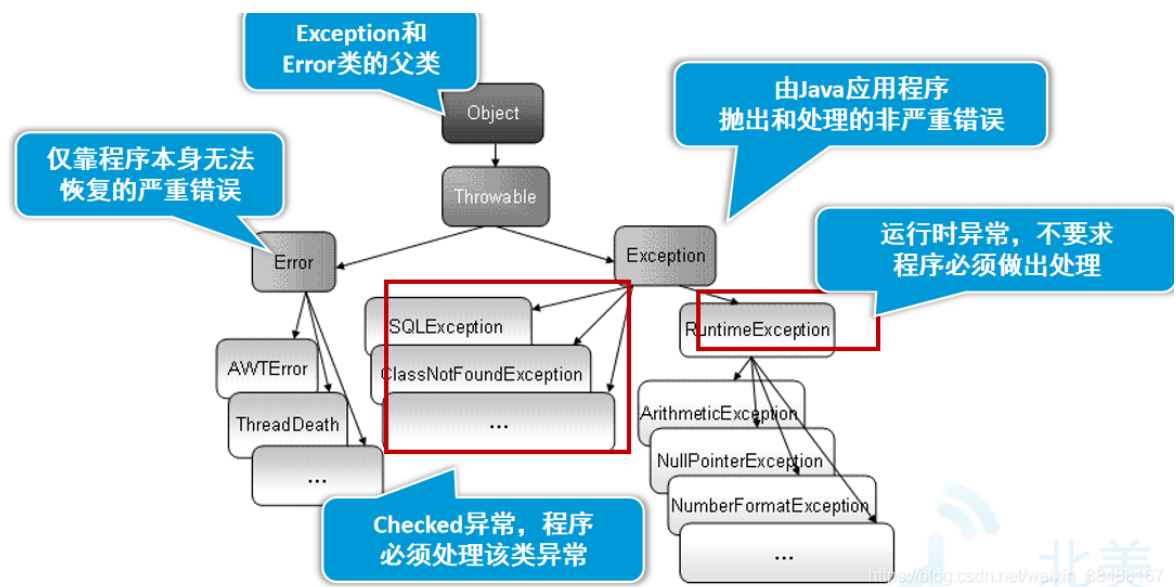
<http://blog.itpub.net/69985551/viewspace-2743586/>

## 第六章 异常处理

### 异常类的分类，异常类的层次结构

异常类分两大类型：Error类代表了编译和系统的错误，不允许捕获；Exception类代表了标准Java库方法所激发的异常。Exception类还包含运行异常类Runtime\_Exception和非运行异常类Non\_RuntimeException这两个直接的子类





异常类型	
ArithmeticException	算数异常
NullPointerException	空指针异常
ArrtIdexOutOfBound	数组越界异常
InsexOutBoundException	索引越界异常
ClassCastException	类型转换异常
InputFormatException	输入类型不匹配
IOException	输入输出异常
SQLException	SQL异常
IllegalArgumentException	非法参数异常

## 异常的捕获

用好try catch很重要，在处理[多线程](#)、避免线程阻塞中很有用，同时养成良好的异常处理习惯，也是作为一个程序员的必备素养。

异常的捕获语句（try、catch、finally）、打印异常信息的三个常用方法toString()、getMessage()、printStackTrace()

在java中可以通过try-catch语句来捕获异常，还是以开头的为例

```
/**
 * @ Author: zhangyu
 * @ Date: 2020/7/27
 * @ Description:
 */
public class Test {
    public static void main(String[] args) {
```

```

try { //将需要捕获异常的代码段放在try-catch中，若捕获到catch括号中的算数异常，则执行catch后面的代码段
    int num=1/0;
    System.out.println(num);

} catch (ArithmeticException e){
    System.out.println("除数不能为零");
}

}

} //输出：除数不能为0

```

finally:

```

try {
    // 可能会发生异常的程序代码
} catch (Type1 id1){
    // 捕获并处置try抛出的异常类型Type1
} catch (Type2 id2){
    //捕获并处置try抛出的异常类型Type2
}finally {
    // 无论是否发生异常，都将执行的语句块
}

```

#### try、catch、finally语句块的执行顺序:

- 1)当try没有捕获到异常时: try语句块中的语句逐一被执行，程序将跳过catch语句块，执行finally语句块和其后的语句;
- 2)当try捕获到异常，catch语句块里没有处理此异常的情况: 此异常将会抛给JVM处理，finally语句块里的语句还是会被执行，但finally语句块后的语句不会被执行;
- 3)当try捕获到异常，catch语句块里有处理此异常的情况: 在try语句块中是按照顺序来执行的，当执行到某一条语句出现异常时，程序将跳到catch语句块，并与catch语句块逐一匹配，找到与之对应的处理程序，其他的catch语句块将不会被执行，而try语句块中，出现异常之后的语句也不会被执行，catch语句块执行完后，执行finally语句块里的语句，最后执行finally语句块后的语句

1. try 块：用于捕获异常。其后可接零个或多个catch块，如果没有catch块，则必须跟一个finally块。
2. catch 块：用于处理try捕获到的异常。
3. finally 块：无论是否捕获或处理异常，finally块里的语句都会被执行。当在try块或catch块中遇到return语句时，finally语句块将在方法返回之前被执行。在以下4种特殊情况下，finally块不会被执行：
  - 1) 在finally语句块中发生了异常。
  - 2) 在前面的代码中用了System.exit()退出程序。
  - 3) 程序所在的线程死亡。
  - 4) 关闭CPU。

getMessage(): String

## 输出异常的描述信息

printStackTrace()

将异常栈打印到输出流中，此为一类方法，默认打印到console控制台，也可以显式指定输出流。

- ```
/**
 * String getMessage() : 返回此 throwable 的详细消息字符串。
 * String toString() : 返回此 throwable 的简短描述。
 * void printStackTrace(): 将此 throwable 及其追踪输出至标准错误流。（即 调用此方法
会把完整的异常信息打印到控制台）
 * @author 郑清
 */
public class Demo {
    public static void main(String[] args) {
        test(6,0);
    }
    public static void test(int a,int b){
        try{
            System.out.println(a/b);
        }catch(Exception e){
            //catch里面是出现异常的处理方式    下面err是以红色打印信息
            System.err.println(e.getMessage());//打印异常原因
            //==》 一般给用户看
            System.err.println(e.toString());//打印异常名称以及异常原因    ==》 很少使用
            e.printStackTrace();//打印异常原因+异常名称+出现异常的位置    ==》 给程序员debug的时候看
        }
        System.out.println("===try-catch结束===");
    }
}
```

##### 抛出异常

"throws + 异常类型"写在方法的声明处。指明此方法执行时，可能会抛出的异常类型。一旦当方法体执行时，出现异常，仍会在异常代码处生成一个异常类的对象，此对象满足throws后异常类型时，就会被抛出。异常代码后续的代码，就不再执行！

try-catch-finally:真正的将异常给处理掉了。

throws的方式只是将异常抛给了方法的调用者。 并没有真正将异常处理掉。

子类重写的方法抛出的异常类型不大于父类被重写的方法抛出的异常类型（子类重写的方法也可以不抛出异常）

```
```java
public class SuperClass {
    public void method() throws IOException {
```

```

    }
}

class SubClass extends SuperClass{
    //报错，子类重写的方法抛出的异常类型不大于父类被重写的方法抛出的异常类型
    // public void method() throws Exception{
    //
    // }

    public void method() throws FileNotFoundException{

    }
}

```

开发中如何选择使用try-catch-finally 还是使用throws？如果父类中被重写的方法没有throws方式处理异常，则子类重写的方法也不能使用throws，意味着如果

子类重写的方法中有异常，必须使用try-catch-finally方式处理。执行的方法a中，先后又调用了另外的几个方法，这几个方法是递进关系执行的。我们建议这几个方法使用throws

的方式进行处理。而执行的方法a可以考虑使用try-catch-finally方式进行处理。

代码示例：

```

public class ErrorThrows {
    public static void method1() throws IOException {
        File file = new File("a.txt");
        FileInputStream fileInputStream = new FileInputStream(file);

        int data = fileInputStream.read();
        while(data != -1){
            System.out.println((char)data);
            data = fileInputStream.read();
        }
        fileInputStream.close();
    }

    public static void method2() throws IOException {
        method1();
    }

    public static void method3() throws IOException {
        method1();
    }

    public static void main(String[] args) {
        try {
            method3();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

**问题来了，什么时候抛出异常，什么时候捕获异常**

答案来自

在Java的异常处理机制中，什么时候应该抛出异常，什么时候捕获异常？ - 知乎

<https://www.zhihu.com/question/25530011>

第一、传递一个危险信号，需要让调用方知道。第二、本方法没有能力处理的异常，调用方有能力处理。第三、抛出是框架层面的选择。

Java异常处理原则之一：延迟捕获

意思是，当异常发生时，不应立即捕获，而是应该考虑当前作用域是否有有能力处理这一异常的能力，如果没有，则应将该异常继续向上抛出，交由更上层的作用域来处理。

一个例子：

某方法String readFile(String filename)，会去尝试读出指定文件的内容并返回，其使用FileInputStream来读取指定文件，而FileInputStream的构造方法会抛出FileNotFoundException，这是一个Checked Exception。

那么readFile方法是应该捕获这个异常，还是抛出这个异常呢？

很显然应该抛出。因为readFile这个方法可能会在不同的场景下，被不同的代码调用，在这些场景中，出现“文件未找到”的情况时的处理逻辑可能是不同的，例如某场景下要发出告警信息，另一场景下可能会尝试从另一个文件中读取，第三个场景下可能需要将错误信息提示给用户。在这种情况下，在readFile方法内的作用域中，是处理不了这个异常的，需要抛出，交由上层的，具备了处理这个异常的能力的作用域来处理。

作者：二大王

链接：<https://www.zhihu.com/question/25530011/answer/154635437>

来源：知乎

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

## 自定义异常类

自定义异常类，有如下步骤：

- 继承于现有的异常结构：RuntimeException、Exception
- 提供全局常量：serialVersionUID
- 提供重载的构造器

自定义异常类：

```
public class MyExceptionClass extends Exception{

    static final long serialVersionUID = -5641210210148784L;

    public MyExceptionClass() {
    }

    public MyExceptionClass(String message) {
        super(message);
    }
}
```

手动抛出上述自定义的异常类对象：

```
public class MyExceptionTest {
```

```

static void method1() throws MyExceptionClass {
    Scanner scanner = new Scanner(System.in);

    System.out.println("请输入大于0的数据: ");
    double next = scanner.nextDouble();
    if(next > 0){
        System.out.println("您输入的数据为: "+next);
    }else {
        throw new MyExceptionClass("您输入的数据不满足要求!");
    }
}

public static void main(String[] args) {
    try {
        method1();
    } catch (MyExceptionClass myExceptionClass) {
        myExceptionClass.printStackTrace();
    }
}
}

```

## 第七章 文件和流

这里强烈建议去上手练习，不然真的难搞

Java 的 IO 编程 · 语雀

[https://www.yuque.com/gorit/learnjava/java\\_se\\_08](https://www.yuque.com/gorit/learnjava/java_se_08)

### File类的概念和使用方法

Java中将电脑中的文件或文件夹封装为一个File类，我们可以使用File类对文件和文件夹进行操作；

可以使用File类

java开发之File类详细使用方法介绍java脚本之家

<https://www.jb51.net/article/180831.htm>

- 1、创建一个文件/文件夹
- 2、删除文件/文件夹
- 3、判断文件/文件夹是否存在
- 4、对文件夹进行遍历
- 5、获取文件的大小

#### 1、构造方法

File(String pathname) 通过将给定路径名字符串转换为抽象路径名来创建一个新 File 实例。

File(String parent,String child) 根据指定的父路径和文件路径创建一个新File对象实例

File(File parent,String child) 根据指定的父路径对象和文件路径创建一个新的File对象实例

```

public static void main(String[] args) {
    //File(String pathname) 将指定路径名转换成一个File对象
    File file = new File("D:\\1.txt");
    System.out.println(file);
}

```

```
//File(String parent,String child) 根据指定的父路径和文件路径创建File对象
File file1 = new File("D:\\a","1.txt");
System.out.println(file1);
//File(File parent,String child) 根据指定的父路径对象和文件路径创建File对象
File parent = new File("D:\\a");
File file2 = new File(parent, "1.txt");
System.out.println(file2);
File file3 = new File(new File("D:\\a"),"1.txt");
System.out.println(file3);

}
```

## 2、File类创建和删除功能

boolean createNewFile();指定路径不存在该文件时创建文件，返回true 否则false

boolean mkdir () 当指定的单击文件夹不存在时创建文件夹并返回true 否则false

boolean mkdirs () 但指定的多级文件夹在某一级文件夹不存在时，创建多级文件夹并返回true 否则false

boolean delete () 删除文件或者删除单级文件夹

删除文件夹，这个文件夹下面不能有其他的文件和文件夹

代码实现过程

```
public static void main(String[] args) throws IOException {
    File file = new File("D:\\a\\1.txt");
    File file1 = new File("1.txt");
    boolean flag = file1.createNewFile();
    System.out.println(flag);

    File file2 = new File("b");
    boolean flag2 = file2.mkdir();
    System.out.println(flag);

    File file3 = new File("c\\d\\e");
    boolean d = file1.mkdir();
    boolean c = file1.mkdirs();
    System.out.println(d);
    System.out.println(c);
    File file4 = new File("c.txt");
    System.out.println(file4.mkdir());

    File file5 = new File("b");
    System.out.println(file2.delete());
}
//结果: false false false false true false
```

## 3、File类的判断功能

boolean exists() 判断指定路径的文件或文件夹是否为空

boolean isAbsolute() 判断当前路径是否是绝对路径

boolean isDirectory() 判断当前的目录是否存在

boolean isFile() 判断当前的目录是否是一个文件

boolean isHidden() 判断当前路径是否是一隐藏文件

```
public static void main(String[] args) throws IOException {
    // method();
    // method2();
    // method3();
    // method4();
}
//判断文件是否存在
public static void method() throws IOException {
    File file = new File("a.txt");
    file.createNewFile();
    boolean flag = file.exists();
    System.out.println(flag);
}
//判断当前路径是否为绝对路径
public static void method2() throws IOException{
    File file = new File("D:\\a\\1.txt");
    boolean flag = file.isAbsolute();
    System.out.println(flag);
}
//判断当前是文件夹还是文件
public static void method3() throws IOException{
    File file = new File("1.txt");
    File file1 = new File("b");
    file1.mkdir();
    boolean flag = file.isDirectory();
    boolean flag2 = file1.isFile();
    System.out.println(flag);
    System.out.println(flag2);
}
//判断当前路径是否为隐藏文件
public static void method4() throws IOException{
    File file = new File("D:\\a\\1.txt");
    System.out.println(file.isHidden());
}
```

#### 4、File类的获取功能和修改名字功能

File getAbsolutePath() 获取文件的绝对路径，返回File对象

String getAbsolutePath() 获取文件的绝对路径，返回路径的字符串

String getParent() 获取当前路径的父级路径，以字符串形式返回该父级路径

String getName() 获取文件或文件夹的名称

String getPath() 获取File对象中封装的路径

long lastModified() 以毫秒值返回最后修改时间

long length() 返回文件的字节数

boolean renameTo(File dest) 将当前File对象所指向的路径修改为指定File所指向的路径

```
public static void main(String[] args) throws IOException {
    // method();
    // method2();
    // method3();
}
```



```

// method4();
}
public static void method(){
    File file = new File("D:\\a\\1.txt");
    File file1 = new File("1.txt");
    //以File对象返回的形式返回当前File对象所指向的绝对路径
    System.out.println(file1.getAbsoluteFile());
    //返回File对象所指向的绝对路径
    System.out.println(file1.getAbsolutePath());
}

public static void method2() throws IOException {
    File file = new File("a.txt");
    File file1 = new File("b", "c.txt");
    System.out.println(file1.createNewFile()); //false

    File parent = new File("b");
    File file2 = new File(parent, "c.txt");
    if (!parent.exists()){
        parent.mkdirs();
    }
    System.out.println(file2.createNewFile()); //false
    System.out.println(file2.getParent()); //b
    System.out.println(file2.getParentFile()); //b
}

public static void method3() throws IOException{
    File file = new File("1.txt");
    File file1 = new File("D:\\a\\1.txt");
    File file2 = new File("b");

    System.out.println(file.getName()); //1.txt
    System.out.println(file1.getName()); //1.txt
    System.out.println(file2.getName()); //b

    System.out.println(file.getPath()); //1.txt
    System.out.println(file1.getPath()); //D:\a\1.txt
    System.out.println(file2.getPath()); //b

    System.out.println(file.lastModified()); //1556090648120
    Date date = new Date(1556085068524L);
    System.out.println(date.toLocaleString()); //2019-4-24 13:51:08

    System.out.println(file.length()); //18
    System.out.println(file2.length()); //0
}

public static void method4() throws IOException{
    File file = new File("a.txt");
    File file1 = new File("e.txt");
    System.out.println(file.renameTo(file1));
}

```

## 5、File 类的其他获取功能

String[] list(); 以字符串的形式返回当前路径下所有的文件和文件夹的名称

File[] listFile 以File对象的形式返回当前路径下的所有文件和文件夹名称

```
public static void main(String[] args) {
    //method();
    // method2();
    //method3();
}

public static void method(){
    File file = new File("b");
    File file1 = new File("D:\\QQ\\1916247350");
    File file2 = new File("e.txt");
    String[] files = file1.list();
    for (int i=0;i<files.length;i++){
        System.out.println(files[i]);
    }
}

public static void method2(){
    File file = new File("b");
    File file1 = new File("D:\\QQ\\1916247350");
    File file2 = new File("e.txt");

    File[] files = file1.listFiles();
    for (File file3 : files) {
        System.out.println(file3.getName());
    }
}

public static void method3(){
    File[] files = File.listRoots();
    for (File file : files) {
        System.out.println(file);
    }
}
```

## 字节流与字符流

流是指一连串流动的数据信号，通过FIFO(先进先出)的方式接收和发送数据

数据流又分为输入流和输出流

输入输出流又分为字节流和字符流

字节流：以字节为基本单位，在 java.io包中，大部分操作继承InputStream（输入字节流）类和 OutputStream（输出字节流）类

字符流：两个字节为基本单位，专门处理字符串和文本，对于字符流进行操作的类主要是Reader（读取流）类和 Writer（写入流）类。

### 使用FileInputStream类

- 继承于InputStream类，这是一个文件输入流，进行文件读操作的最基本的类
- 作用是将文件中的数据输入到内存中，我们可以用它来读文件操作
- 由于字节流的缘故，因此无法读取中文字符

```
import java.io.*;

public class FileStreamDemo {
    public static void main(String[] args) {
        try {
```

处理

```
File file=new File("src\\aa.txt");
FileInputStream f1=new FileInputStream(file);//这里需要进行抛出异常

for (int i = 0; i < file.length(); i++) {
    char ch=(char)(f1.read());//循环读取字符
    System.out.print(ch+" ");
}
System.out.println();//换行操作
f1.close();//关闭文件
} catch (Exception e) {
    // TODO: handle exception
    System.out.println("文件打开失败");
}

}
```

I a m f o n d o f p r o g r a m i n g i %

这里我在文件中的感叹号打的是中文的字符，所以字节流是无法读取的

这里我们使用了一个新的知识点，try catch的一个异常捕获的知识点，这样操作的好处，即使报错了，程序也可以正常的运行下去，并显示报错信息

### 使用FileOutputStream类

FileOutputStream类称为文件输出流，继承于OutputStream类，是文件的基本读写的一个类，它的作用和上面读文件恰恰相反，将内存中的数据输出到文件中，所以我们可以用这个类来进行写文件的操作

我们先按aa.txt文件的内容清空，然后实现下边的代码进行写操作

代码示例：

```
import java.io.*;

public class FileOutputDemo {
    public static void main(String[] args) throws FileNotFoundException {
        File file=new File("src\\aa.txt");
        FileOutputStream f1=new FileOutputStream(file);//(file,true)，这里有true的
        话，代表可以在文件后面追加内容
        String str="I love coding";
        byte[] buff=str.getBytes();//将字符串转换为字节数组
        try {
            f1.write(buff);//把字节数组的内容写进去文件
        } catch (Exception e) {
            // TODO: handle exception
        }finally {
            try {
                f1.close();
            } catch (IOException e) {
                // TODO Auto-generated catch block
            }
        }
    }
}
```

```

        e.printStackTrace();
    }
}
}
}

```

1 I love coding

## FileWriter类与BufferedWriter类使用

```

import java.io.*;

public class FileWriterDemo {
    public static void main(String[] args) {
        String[] str= {"春眠不觉晓,", "处处闻啼鸟,", "夜来风雨声,", "花落知多少,"};
        File file=new File("src\\cc.txt");//我们在该类的位置创建一个新文件
        FileWriter f=null;//创建文件写入对象
        BufferedWriter f1=null;//创建字符流写入对象

        try {
            //这里把文件写入对象和字符流写入对象分开写了
            f=new FileWriter("src\\cc.txt");//创建一个名为cc.txt的文件
            f1=new BufferedWriter(f);
            //通过循环遍历上面的String 数组中的元素
            for (int i = 0; i < str.length; i++) {
                f1.write(str[i]);//把String中的字符写入文件
                f1.newLine();//换行操作
            }
        } catch (Exception e) {
            // TODO: handle exception
        } finally { //如果没有catch 异常，程序最终会执行到这里
            try {
                f1.close();
                f.close();//关闭文件
            } catch (Exception e2) {
                // TODO: handle exception
            }
        }
    }
}

```

## FileReader类与BufferedReader类使用

```

import java.io.*;

public class FileReaderDemo {
    public static void main(String[] args) {

```

```

File file=new File("src\\cc.txt");
FileReader f=null;//文件读取对象
BufferedReader f1=null;//字符流对象
try {
    f=new FileReader(file);
    f1=new BufferedReader(f);
    //循环打印cc文件中的每行数据
    String str=null;
    while((str=f1.readLine())!=null) {
        System.out.println(str);
    }

} catch (Exception e) {
    // TODO: handle exception
}finally {
    try {
        f1.close();
        f.close();
    } catch (Exception e2) {
        // TODO: handle exception
    }
}
}
}

```

## FileOutputStream类 以及 ObjectOutputStream类的使用

序列化是把Java对象存在一个硬盘，网络，以便传输  
也就是把我们的数据永久的存放到计算机当中

#####

```

import java.io.*;
import java.util.*;

/**
 * 序列化操作
 * */
public class x1 {
    public static void main(String[] args) {
        //数据完成持久化的操作
        List<String> list=new ArrayList<String>();
        list.add("aaa");
        list.add("bbb");
        list.add("ccc");
        FileOutputStream f1=null;
        ObjectOutputStream f2=null;
        try {
            //第一行省略了前面的 File file = new File(~)的操作，直接创建一个文件
            f1=new FileOutputStream(new File("src\\dd.txt"));
            f2=new ObjectOutputStream(f1);
            f2.writeObject(list);
        } catch (Exception e) {
            // TODO: handle exception
        }finally {
            try {

```

```

        f2.close();
        f1.close();
    } catch (Exception e2) {
        // TODO: handle exception
    }
}
}

```

## FileInputStream类 以及 ObjectInputStream类的使用

```

//反序列化
import java.io.*;
import java.util.*;

//反序列化操作
public class FXL {
    public static void main(String[] args) {
        FileInputStream f=null;
        ObjectInputStream f1=null;
        List list=null;
        try {
            f=new FileInputStream("src\\dd.txt");//对应我们前面往输入dd.txt 文件的内
容
            f1=new ObjectInputStream(f);
            list=(List<String>)f1.readObject();
        } catch (Exception e) {
            // TODO: handle exception
        }finally {
            try {
                f1.close();
                f.close();
            } catch (Exception e2) {
                // TODO: handle exception
            }
        }
        //这种方法就不必循环打印数据了，可以一步到位
        System.out.println(list);
    }
}

```

参考自 Java文件操作（超详细+代码示例）Gorit的博客-CSDN博客java 文件操作  
<https://blog.csdn.net/caidewei121/article/details/89426032>

## Date类

```
import java.util.Date;

public class DateDemo {
    public static void main(String[] args) {
        //public Date(); 分配一个Date对象，并初始化以便他代表他被分配的时间，精确到毫秒
        Date d1=new Date();
        System.out.println(d1);
    }
}
//Thu Aug 05 16:04:07 CST 2021
```

对日期格式化，创建SimpleDateFormat对象,在构造方法中指定日期的格式  
然后调用SimpleDateFormat对象的format(Date)可以把Date日期转换为字符串

```
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
String ss = sdf.format(date);
System.out.println( ss );
```

结果

2020-07-20 21:12:01

把字符串转换为DATA对象，先根据日期字符串的格式创建SimpleDateFormat对象，调用parse(String)  
可以把字符串转换为日期

```
String text = "2066年5月1日 8:25:28";
SimpleDateFormat another = new SimpleDateFormat("yyyy年M月d日 H:mm:ss");
Date date2 = another.parse(text);
System.out.println( date2 );
```

## RandomAccess类

### 简介

Java提供了一个可以对文件随机访问的操作，访问包括读和写操作。该类名为RandomAccessFile。该类的读写是基于指针的操作。

RandomAccessFile在对文件进行随机访问操作时有两个模式，分别为只读模式（只读取文件数据），和读写模式（对文件数据进行读写）。

### 创建对象

在创建RandomAccessFile时，其提供的构造方法要求我们传入访问模式：

- — RandomAccessFile(File file, String mode)
- — RandomAccessFile(String filename, String mode)

其中构造方法的第一个参数是需要访问的文件，而第二个参数则是访问模式：

- — “r”：表示对该文件的访问是只读的。
- — “rw”：表示对该文件的访问是读写模式的。

### 字节数据读写操作

#### write(int d)方法

· RandomAccessFile提供了一个可以向文件中写出字节的方法：

— void write(int d) 该方法会根据当前指针所在位置处写入一个字节，是将参数int的“低8位”写出。

```
public void testWrite() throws Exception {  
    RandomAccessFile raf = new RandomAccessFile("raf.dat", "rw");  
    // 写出一个字节,写的是int值的低8位  
    raf.write(1);  
    raf.close();  
}
```

#### **read()方法**

· RandomAccessFile提供了一个可以向文件中读取字节的方法：

— int read()

该方法会从文件中读取一个byte(8位)填充到int的低八位，高24位为0，返回值范围正数：0 ~ 255，如果返回-1表示读取到了文件末尾！每次读取后自动移动文件指针，准备下次读取。

```
public void testRead() throws Exception {  
    RandomAccessFile raf = new RandomAccessFile("raf.dat", "r");  
    // 读取一个字节  
    int d = raf.read();  
    System.out.println(d);  
    raf.close();  
}
```

#### **write(byte[] d)方法**

· RandomAccessFile提供了一个可以向文件中写出一组字节的方法：

— void write(byte[] d) 该方法会根据当前指针所在位置处连续写出给定数组中的所有字节。

· 与该方法相似的还有一个常用方法：

— void write(byte[] d, int offset, int len)

该方法会根据当前指针所在位置处连续写出给定数组中的部分字节，这个部分是从数组的offset处开始，连续len个字节。

```
public void testWriteByteArray() throws Exception {  
    RandomAccessFile raf = new RandomAccessFile("raf.dat", "rw");  
    // 将字符串按照默认编码转换为字节  
    byte[] buf = "helloWorld".getBytes();  
    // 将字节数组中所有字节一次性写出  
    raf.write(buf);  
    raf.close();  
}
```

#### **read(byte[] b)方法**

· RandomAccessFile提供了一个可以向文件中批量读取字节的方法：

— int read(byte[] b)



该方法会从指针位置处尝试最多读取给定数组的总长度的字节量，并从给定的字节数组第一个位置开始，将读取到的字节顺序存放至数组中，返回值为实际读取到的字节量。

```
public void testReadByteArray() throws Exception {
    RandomAccessFile raf = new RandomAccessFile("raf.dat", "r");
    // 创建10字节数组
    byte[] buf = new byte[10];
    // 尝试读取10个字节存入数组，返回值为读取的字节量
    int len = raf.read(buf);
    System.out.println("读取到了:" + len + "个元素");
    System.out.println(new String(buf));
    raf.close();
}
```

#### **close()方法**

· RandomAccessFile在对文件访问的操作全部结束后，要调用close()方法来释放与其关联的所有系统资源。

— void close()

```
RandomAccessFile raf = new RandomAccessFile(file,"rw");
```

.....//读写操作

```
raf.close();//访问完毕后要关闭以释放系统资源。
```

#### **文件指针操作**

RandomAccessFile的读写操作都是基于指针的，也就是说总是在指针当前所指向的位置进行读写操作。

#### **getFilePointer方法**

— long getFilePointer() 该方法用于获取当前RandomAccessFile的指针位置。

```
RandomAccessFile raf = new RandomAccessFile(file," rw" );
```

```
System.out.println(raf.getFilePointer());//0
```

```
raf.write( 'A' );
```

```
System.out.println(raf.getFilePointer());//1
```

```
raf.writeInt(3);
```

```
System.out.println(raf.getFilePointer());//5
```

```
raf.close();
```

#### **seek()方法**

· RandomAccessFile提供了一个方法用于移动指针位置。

— void seek(long pos) 该方法用于移动当前RandomAccessFile的指针到指定位置。

```

RandomAccessFile raf = new RandomAccessFile(file," rw" );
System.out.println(raf.getFilePointer());//0
raf.write( 'A' );//指针位置1
raf.writeInt(3);//指针位置5
//将指针移动到文件开始处(第一个字节的位置)
raf.seek(0);
System.out.println(raf.getFilePointer());//0
raf.close();

```

#### skipBytes()方法

· RandomAccessFile提供了一个方法可以尝试跳过输入的n个字节以丢弃跳过的字节。

— int skipBytes(int n)

该方法可能跳过一些较少数量的字节（可能包括零）。这可能由任意数量的条件引起；在跳过n个字节之前已到达文件的末尾只是其中的一种可能。该方法不抛出EOFException。返回跳过的实际字节数。如果n为负数，则不跳过任何字节。

```

public void testPointer() throws Exception {
    RandomAccessFile raf = new RandomAccessFile("raf.dat", "r");
    // 输出指针位置,默认从0开始(文件的第一个字节位置)
    System.out.println("指针位置:" + raf.getFilePointer());
    // 读取world,需要将hello这5个字节跳
    raf.skipBytes(5);
    System.out.println("指针位置:" + raf.getFilePointer());
    // 读取world这5个字节
    byte[] buf = new byte[5];
    raf.read(buf);
    System.out.println(new String(buf));
    System.out.println("指针位置:" + raf.getFilePointer());
    // 将游标移动到文件开始
    raf.seek(0);
    System.out.println("指针位置:" + raf.getFilePointer());
    raf.close();
}

```

## Scanner类

使用hasNext判断是否有字符串的输入

```

import java.util.Scanner;
//next() 不能得到空白的字符串
public class Deom01 {
    public static void main(String[] args) {

        //创建一个扫描器对象，用于接收键盘数据
        Scanner scanner=new Scanner(System.in);
        System.out.println("使用next方法输入：");
        //判断用户有没有输入字符串
        if(scanner.hasNext()){
            String str=scanner.next();//使用next方法接收

```

```

        System.out.println("输出的内容为: "+str);
    }

    //凡是属于IO流的类，不关闭会一直占用空间
    scanner.e();
}
}

```

使用nextline判断是否有字符串的输入

```

import java.util.Scanner;
//nextLine()方法以回车作为结束符
public class Deom02 {
    public static void main(String[] args) {

        Scanner scanner=new Scanner(System.in);
        System.out.println("使用nextline方法接收: ");

        //判断是否输入字符串
        if(scanner.hasNextLine()){
            String str=scanner.nextLine();//使用nextLine方法接收
            System.out.println("输出的内容为: "+str);
        }
        scanner.close();
    }
}

```

## 第八章 集合和泛型

### 泛型

假定我们有这样一个需求：写一个排序方法，能够对整型数组、字符串数组甚至其他任何类型的数组进行排序，该如何实现？

答案是可以使用 **Java 泛型**。

你可以写一个泛型方法，该方法在调用时可以接收不同类型的参数。根据传递给泛型方法的参数类型，编译器适当地处理每一个方法调用。

下面是定义泛型方法的规则：

- 所有泛型方法声明都有一个类型参数声明部分（由尖括号分隔），该类型参数声明部分在方法返回类型之前（在下面例子中的）。
- 每一个类型参数声明部分包含一个或多个类型参数，参数间用逗号隔开。一个泛型参数，也被称为一个类型变量，是用于指定一个泛型类型名称的标识符。
- 类型参数能被用来声明返回值类型，并且能作为泛型方法得到的实际参数类型的占位符。
- 泛型方法体的声明和其他方法一样。注意类型参数只能代表引用型类型，不能是原始类型（像 **int**、**double**、**char**等）。
- **java 中泛型标记符：**
  - **E** - Element (在集合中使用，因为集合中存放的是元素)

- **T** - Type (Java 类)
- **K** - Key (键)
- **V** - Value (值)
- **N** - Number (数值类型)
- **?** - 表示不确定的 java 类型

```
public class GenericMethodTest
{
    // 泛型方法 printArray
    public static < E > void printArray( E[] inputArray )
    {
        // 输出数组元素
        for ( E element : inputArray ){
            System.out.printf( "%s ", element );
        }
        System.out.println();
    }

    public static void main( String args[] )
    {
        // 创建不同类型数组: Integer, Double 和 Character
        Integer[] intArray = { 1, 2, 3, 4, 5 };
        Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4 };
        Character[] charArray = { 'H', 'E', 'L', 'L', 'O' };

        System.out.println( "整型数组元素为:" );
        printArray( intArray ); // 传递一个整型数组

        System.out.println( "\n双精度型数组元素为:" );
        printArray( doubleArray ); // 传递一个双精度型数组

        System.out.println( "\n字符型数组元素为:" );
        printArray( charArray ); // 传递一个字符型数组
    }
}

//整型数组元素为:
1 2 3 4 5

双精度型数组元素为:
1.1 2.2 3.3 4.4

字符型数组元素为:
H E L L O
```

定义一个泛型类:

```
public class Box<T> {

    private T t;

    public void add(T t) {
        this.t = t;
    }
}
```

```

public T get() {
    return t;
}

public static void main(String[] args) {
    Box<Integer> integerBox = new Box<Integer>();
    Box<String> stringBox = new Box<String>();

    integerBox.add(new Integer(10));
    stringBox.add(new String("菜鸟教程"));

    System.out.printf("整型值为 :%d\n\n", integerBox.get());
    System.out.printf("字符串为 :%s\n", stringBox.get());
}
}
//整型值为 :10

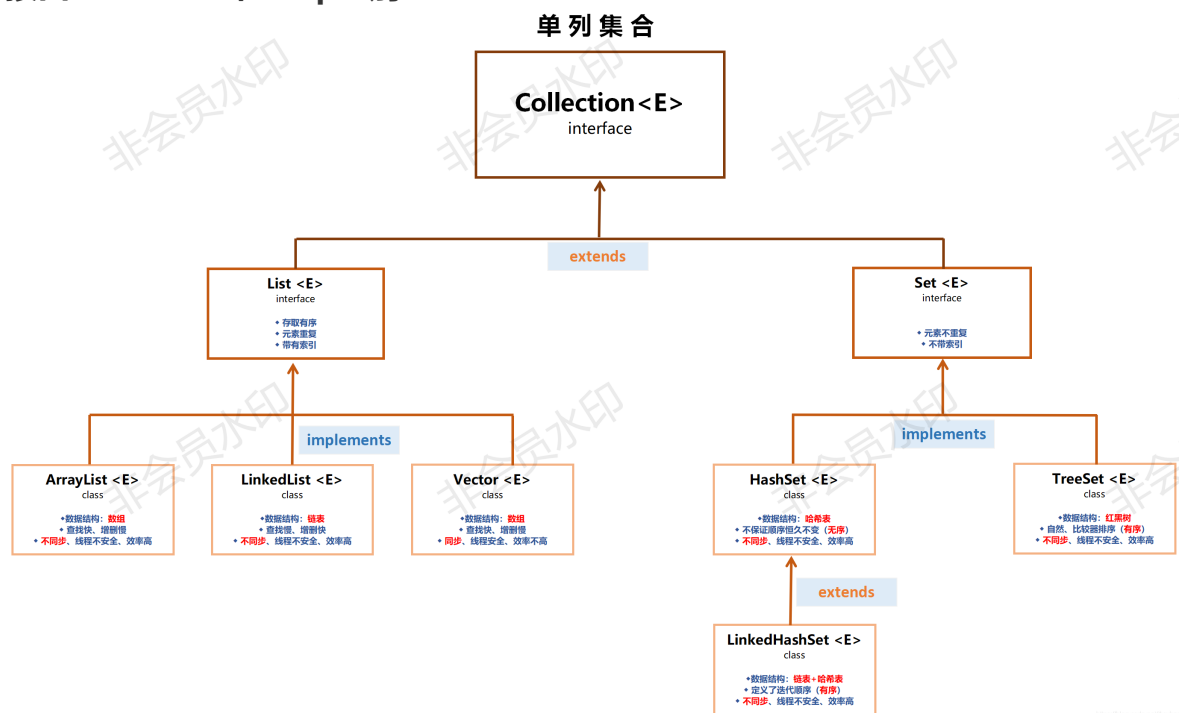
字符串为 :菜鸟教程

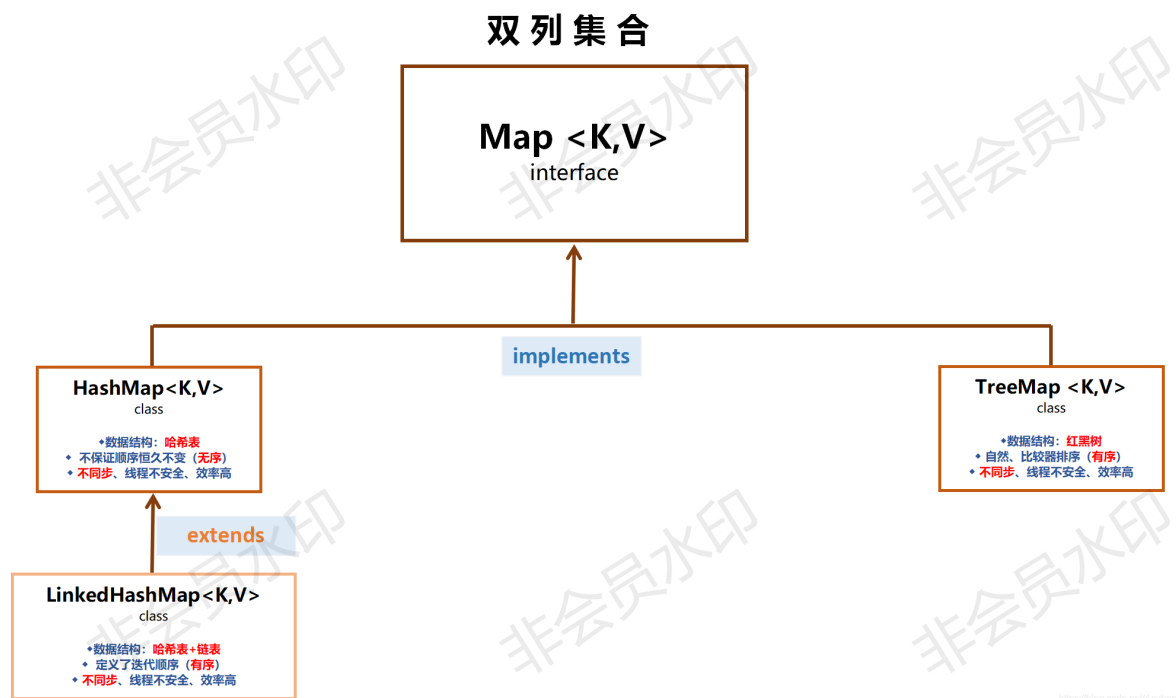
```

Java 泛型 | 菜鸟教程

<https://www.runoob.com/java/java-generics.html>

## 接口Collection和Map区别





**Collection : 单列集合。**

Map : 双列集合, 键和值——映射, 针对键有效, 跟值无关

## List与Set的区别

- 1 List允许存储重复的对象, 而Set不允许
- 2 List可以插入多个null元素, 而set只允许插入一个null元素
- 3 List是一个有序的容器, 可以保持每个元素的插入顺序, 而set是无序容器, 不能保持存储顺序。

Java List集合全解SuperSunQ的博客-CSDN博客java list

原文链接: <https://blog.csdn.net/a1781842634/article/details/116516967>

List接口的特点

List集合类中元素有序(添加顺序和取出顺序一致), 且可重复

List集合中的每个元素都有其对应的顺序索引, 即支持索引

List容器中的元素都对应一个整数型的序号记载其在容器中的位置, 额可以根据序号存取容器中的元素

JDK API中List接口的实现类非常多, 常用的有ArrayList、Set、Vector

List接口方法

```
public class ListMethod {
    @SuppressWarnings({"all"})
    public static void main(String[] args) {
        List list = new ArrayList();
        list.add("张三丰");
        list.add("贾宝玉");
        // void add(int index, Object ele):在index位置插入ele元素
        //在index = 1的位置插入一个对象
        list.add(1, "韩顺平");
        System.out.println("list=" + list);
        // boolean addAll(int index, Collection eles):从index位置开始将eles中的所有元素添加进来
        List list2 = new ArrayList();
        list2.add("jack");
    }
}
```

```

        list2.add("tom");
        list.addAll(1, list2);
        System.out.println("list=" + list);
//      Object get(int index):获取指定index位置的元素
//      说过
//      int indexOf(Object obj):返回obj在集合中首次出现的位置
        System.out.println(list.indexOf("tom"));//2
//      int lastIndexOf(Object obj):返回obj在当前集合中末次出现的位置
        list.add("韩顺平");
        System.out.println("list=" + list);
        System.out.println(list.lastIndexOf("韩顺平"));
//      Object remove(int index):移除指定index位置的元素，并返回此元素
        list.remove(0);
        System.out.println("list=" + list);
//      Object set(int index, Object ele):设置指定index位置的元素为ele，相当于替换。
        list.set(1, "玛丽");
        System.out.println("list=" + list);
//      List subList(int fromIndex, int toIndex):返回从fromIndex到toIndex位置的子集合
        // 注意返回的子集合 fromIndex <= subList < toIndex
        List returnlist = list.subList(0, 2);
        System.out.println("returnlist=" + returnlist);
    }
}

```

## HashSet与ArrayList

```

// 引入 HashSet 类
import java.util.HashSet;

public class RunoobTest {
    public static void main(String[] args) {
        HashSet<String> sites = new HashSet<String>();
        sites.add("Google");
        sites.add("Runoob");
        sites.add("Taobao");
        sites.add("Zhihu");
        sites.add("Runoob"); // 重复的元素不会被添加
        System.out.println(sites);
        System.out.println(sites.contains("Taobao"))
        sites.remove("Taobao"); // 删除元素，删除成功返回 true，否则为 false
        sites.clear();//删除集合中所有元素
        sites.size();//计算元素量
    }
}
//[Google, Runoob, Zhihu, Taobao]
//true

```

```

public class T {
    public static void main(String[] args) {
        List list = new ArrayList();
    }
}

```

```

//add添加元素
list.add("jack");
list.add(10);
list.add(true);
System.out.println("list = " + list);
//remove 删除指定元素
//list.remove(0); //删除第一个元素
list.remove(true);
//contains查找元素是否存在
System.out.println(list.contains("jack"));
//size:获取元素个数
System.out.println(list.size());
//isEmpty判断是否为空
System.out.println(list.isEmpty());
//clear 清空
list.clear();
System.out.println("list = " + list);

//addAll添加多个元素
ArrayList list2 = new ArrayList();
list2.add("hlm");
list2.add("sgyy");
list.addAll(list2);
System.out.println("list = " + list);
//containsAll:判断多个元素是否存在
System.out.println(list.containsAll(list2));
//removeAll 删除多个元素
list.add("lz");
list.removeAll(list2);
System.out.println("list = " + list);
}

}

```

## HashMap

```

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map.Entry;

public class HashMapTest {

    public static void main(String[] args) {
        HashMap<String, String> map = new HashMap<>();
        map.put("zhang", "31");//存放键值对

        System.out.println(map.containsKey("zhang"));//键中是否包含这个数据
        System.out.println(map.containsKey("daniu"));
        System.out.println("=====");

        System.out.println(map.get("zhang"));//通过键拿值
        System.out.println(map.get("daniu"));
        System.out.println("=====");
    }
}

```



```

System.out.println(map.isEmpty());">//判空
System.out.println(map.size());
System.out.println("=====");

System.out.println(map.remove("zhang")); ">//从键值中删除
System.out.println(map.containsKey("zhang"));
System.out.println(map.get("zhang")); ">//获取值
System.out.println(map.isEmpty());
System.out.println(map.size());
System.out.println("=====");

map.put("zhang", "31");
System.out.println(map.get("zhang"));
map.put("zhang", "32");
System.out.println(map.get("zhang"));
System.out.println("=====");

map.put("zhang", "31");
map.put("cheng", "32");
map.put("yun", "33");

for (String key : map.keySet()) {
    System.out.println(key);
}
System.out.println("=====");

for (String values : map.values()) {
    System.out.println(values);
}
System.out.println("=====");

map.clear();
map.put("A", "1");
map.put("B", "2");
map.put("C", "3");
map.put("D", "1");
map.put("E", "2");
map.put("F", "3");
map.put("G", "1");
map.put("H", "2");
map.put("I", "3");
for (Entry<String, String> entry : map.entrySet()) {
    String key = entry.getKey();
    String value = entry.getValue();
    System.out.println(key + "," + value);
}
System.out.println("=====");

// you can not remove item in map when you use the iterator of map
// for(Entry<String,String> entry : map.entrySet()){
// if(!entry.getValue().equals("1")){
// map.remove(entry.getKey());
// }
// }

// if you want to remove items, collect them first, then remove them by
// this way.
List<String> removeKeys = new ArrayList<String>();

```

```

        for (Entry<String, String> entry : map.entrySet()) {
            if (!entry.getValue().equals("1")) {
                removeKeys.add(entry.getKey());
            }
        }
        for (String removeKey : removeKeys) {
            map.remove(removeKey);
        }
        for (Entry<String, String> entry : map.entrySet()) {
            String key = entry.getKey();
            String value = entry.getValue();
            System.out.println(key + "," + value);
        }
        System.out.println("=====");
    }
}

```

## 迭代器

Iterator对象称为迭代器，主要用于遍历Collection集合中的元素

所有实现了Collection接口的集合类都有一个iterator()方法，用于返回一个实现了Iterator接口的对象，即可以返回一个迭代。

Iterator结构如图所示

Iterator仅用于遍历集合，Iterator本身不存放对象

```

public class CollectionIterator {
    @SuppressWarnings({"all"})
    public static void main(String[] args) {
        Collection col = new ArrayList();
        col.add(new Book("三国演义", "罗贯中", 10.1));
        col.add(new Book("小李飞刀", "古龙", 5.1));
        col.add(new Book("三红楼梦", "曹雪芹", 34.6));
        //      System.out.println("col : " + col);
        // 遍历集合：
        //1、 先得到col对应的迭代器
        Iterator iterator = col.iterator();
        //2、使用while循环遍历
        while(iterator.hasNext()) {
            //返回下一个元素， 类型是Object
            Object obj = iterator.next();
            System.out.println("obj=" + obj);
        }
        //快捷键，快速生成while => itit
    }
}

class Book {
    private String name;
    private String author;
}

```

```

private double price;

public Book(String name, String author, double price) {
    this.name = name;
    this.author = author;
    this.price = price;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getAuthor() {
    return author;
}

public void setAuthor(String author) {
    this.author = author;
}

public double getPrice() {
    return price;
}

public void setPrice(double price) {
    this.price = price;
}

@Override
public String toString() {
    return "Book{" +
        "name='" + name + '\'' +
        ", author='" + author + '\'' +
        ", price=" + price +
        '}';
}
}

```

## 第九章 线程

##

### 线程创建方式1

```

public class Test01 {
    public static void main(String[] args) {
        Thread thread1 = new MyThread1();
        Thread thread2 = new MyThread2();
        thread1.start();
        thread2.start();
    }
}

```

```

}
class MyThread1 extends Thread{
    public MyThread1() {

    }
    public MyThread1(String name) {
        super(name);
    }
    public void run(){
        for(int i=0;i<1000;i++){
            System.out.println("你好");
        }
    }
}
class MyThread2 extends Thread{
    public MyThread2() {

    }
    public MyThread2(String name) {
        super(name);
    }
    public void run(){
        for(int i=0;i<1000;i++){
            System.out.println("再见");
        }
    }
}

```

## 线程创建方式2 调用Runnable接口

```

public class Test02 {

    public static void main(String[] args) {
        Runnable1 runnable1 = new Runnable1();
        Thread thread1 = new Thread(runnable1);
        Runnable2 runnable2 = new Runnable2();
        Thread thread2 = new Thread(runnable2);

        thread1.start();
        thread2.start();

    }
}
class Runnable1 implements Runnable{

    @Override
    public void run() {
        for(int i=0; i<1000;i++){
            System.out.println("你好");
        }
    }
}
class Runnable2 implements Runnable{

    @Override
    public void run() {
        for(int i=0; i<1000;i++){

```

```

        System.out.println("再见");
    }
}
}

```

## 匿名内部类创建线程

```

public class Test03 {
    public static void main(String[] args) {
        Thread thread1 = new Thread(){
            @Override
            public void run() {
                for(int i =0;i<1000;i++){
                    System.out.println("你好");
                }
            }
        };
        Thread thread2 = new Thread(){
            @Override
            public void run() {
                for(int i =0;i<1000;i++){
                    System.out.println("再见");
                }
            }
        };
        thread1.start();
        thread2.start();
    }
}

```

1、匿名内部类就是没有名字的内部类。这是Java为了方便我们编写程序而设计的一个机制。因为有时候有的内部类只需要创建一个它的对象就可以了，以后不会再用到这个类，这时候使用匿名内部类就比较合适，而且也免去了给它取名字的烦恼。

2、如果满足下面的一些条件，使用匿名内部类是比较合适的：

- 只用到类的一个实例。
  - 类在定义后马上用到。
  - 类非常小（SUN推荐是在4行代码以下）
  - 给类命名并不会导致你的代码更容易被理解。
- 在使用匿名内部类时，要记住以下几个原则：
- 匿名内部类不能有构造方法。
  - 匿名内部类不能定义任何静态成员、方法和类。
  - 匿名内部类不能是public,protected,private,static。
  - 只能创建匿名内部类的一个实例。
  - 一个匿名内部类一定是在new的后面，用其隐含实现一个接口或实现一个类。
  - 因匿名内部类为局部内部类，所以局部内部类的所有限制都对其生效。

3、用匿名内部类创建多线程有两种方式，和创建一个线程是一样的。

第①种继承Thread：

```

new Thread(){
    public void run(){
        //do something
    };
}.start();

```

第②种实现 Runnable接口：

```
new Thread(new Runnable() {  
    public void run() {  
        //do something  
    };  
}) { }.start();
```

---

版权声明：本文为CSDN博主「黎明静悄悄啊」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：[https://blog.csdn.net/qg\\_41258326/article/details/103421486](https://blog.csdn.net/qg_41258326/article/details/103421486)

## 例题

```
public class Test04 {  
  
    public static void main(String[] args) {  
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");  
        while(true){  
            try {  
                Thread.sleep(1000);  
            } catch (InterruptedException e) {  
                e.printStackTrace();  
            }  
            System.out.println(sdf.format(new Date()));  
        }  
    }  
}
```

这里了解一下printStackTrace方法

当try语句中出现异常是时，会执行catch中的语句，java运行时系统会自动将catch括号中的Exception e 初始化，也就是实例化Exception类型的对象。e是此对象引用名称。然后e（引用）会自动调用Exception类中指定的方法，也就出现了e.printStackTrace();。

printStackTrace()方法的意思是：在命令行打印异常信息在程序中出错的位置及原因

## 补充

实现多线程的方式1:

- 1.自定义类继承Thread类
- 2.重写run方法
- 3.创建自定义类对象
- 4.调用对象的start方法

Thread的构造器:

new Thread(): 无参构造器, 线程名默认  
new Thread(String name): 指定线程名  
new Thread(Runnable run): 指定Runnable对象, 但是线程名默认  
new Thread(Runnable run, String name): 指定Runnable对象, 指定线程名

Thread的API:

Thread currentThread(): 可以获得当前正在执行的线程对象

String getName(): 获得线程的名字

static sleep(long): 让线程暂停执行对应时间, 在sleep时间结束之前, cpu都不会选择到这个线程执行

线程原理: 画图

实现多线程的方式2: 常用

- 1.自定义类来实现Runnable接口
- 2.实现run方法
- 3.创建这个实现类对象
- 4.将这个实现类对象作为Thread参数传递给Thread对象
- 5.调用线程的start方法

Runnable和Thread的区别?

- 1.Runnable没有类继承的局限性
  - 一个类的父类只能有一个(亲爹)
  - 一个类的父接口可以是无限个
- 2.Runnable降低了代码的耦合性(解耦)
  - 完成了启动线程和线程任务的分离

实现多线程的简单写法:

匿名内部类

JDK8 的新特性: Lambda表达式(函数式编程)

就是简化匿名内部类的

## 同步安全:

同步: 多个线程之间共享资源

线程执行过程中,可以随时交替执行另外的线程

就可能会产生问题

解决: 保证一个线程在卖票的时候, 另外的窗口不能执行卖票代码

解决方案:

- 1.同步代码块 synchronized() {}
  - 使用的是对象锁, 可以是任意对象, 同一时刻只能有一个线程获得对象锁
  - 当同步代码块执行结束, 将对象锁归还
  - 问题: 在一个线程执行同步代码块时, 其他线程只能等待

可见

同步代码块Synchronized笔记*假娃不是java的博客*CSDN博客synchronized代码块

<https://blog.csdn.net/xiaojiaxins/article/details/119774985>

2.同步方法

public synchronized 其他修饰词 void method() {}

使用的锁, 就是this对象

public synchronized static void method() {}

锁静态方法, 使用的锁, 就是类的class对象

3.Lock锁

Lock 实现提供了比使用 synchronized 方法和语句可获得的更广泛的锁定操作

void lock() 获取锁。

void unlock() 释放锁。

- 1.创建一个锁对象 ReentrantLock
- 2.调用lock方法加锁
- 3.调用unlock方法释放锁

# 线程状态

线程状态图

