

## 0.第一个程序

```
public class HelloWorld {  
    /* 第一个Java程序  
     * 它将打印字符串 Hello world  
     */  
    public static void main(String []args) {  
        System.out.println("Hello world"); // 打印 Hello world  
    }  
}
```

在cmd中执行编译、运行

```
C : > javac HelloWorld.java  
C : > java HelloWorld  
Hello world
```

编写 Java 程序时，应注意以下几点：

- **大小写敏感**：Java 是大小写敏感的，这就意味着标识符 Hello 与 hello 是不同的。
- **类名**：对于所有的类来说，类名的首字母应该大写。如果类名由若干单词组成，那么每个单词的首字母应该大写，例如 **MyFirstJavaClass**。
- **方法名**：所有的方法名都应该以小写字母开头。如果方法名含有若干单词，则后面的每个单词首字母大写。
- **源文件名**：源文件名必须和类名相同。当保存文件的时候，你应该使用类名作为文件名保存（切记 Java 是大小写敏感的），文件名的后缀为 **.java**。（如果文件名和类名不相同则会导致编译错误）。
- **主方法入口**：所有的 Java 程序由 **public static void main(String []args)** 方法开始执行。

## 1.基本数据类型

四类八种数据类型

### 1、整型

byte、short、int、long

### 2、浮点型

float、double

### 3、字符型

char

### 4、布尔型

boolean

需要注意 float 类型在书写时是要带上后缀 f，例如 1.2f，否则默认为 double 类型。long 类型也加上后缀 L，例如 1000000L。

此外，由于浮点数类型存储方式原因，在比较两个浮点数是否相同时，应该考虑到它的存储误差。比较他们的值是最好的使用做差，限制在允许误差范围内认为两个值相等。

基本类型（与其对应的非基本类型，称为复合类型）：

类型名称	关键字	占用内存	取值范围	默认值
字节型	byte	1 字节	-128~127(-2^7~2^7-1)	0
短整型	short	2 字节	-32768~32767	0
整型	int	4 字节	-2147483648~2147483647	0
长整型	long	8 字节	-9223372036854775808L~9223372036854775807L	0L
单精度浮点型	float	4 字节	+/-3.4E+38F (6~7 个有效位)	0.0f
双精度浮点型	double	8 字节	+/-1.8E+308 (15 个有效位)	0.0d
字符型	char	2 字节	ISO 单一字符集	
布尔型	boolean	1 字节	true 或 false	false

查看基本类型的情况

```
// byte
System.out.println("基本类型: byte 二进制位数: " + Byte.SIZE);
System.out.println("包装类: java.lang.Byte");
System.out.println("最小值: Byte.MIN_VALUE=" + Byte.MIN_VALUE);
System.out.println("最大值: Byte.MAX_VALUE=" + Byte.MAX_VALUE);
```

类型转换：

## 1.1 包装类

Java中的基本类型功能简单，不具备对象的特性，为了使基本类型具备对象的特性，所以出现了包装类，就可以像操作对象一样操作基本类型数据。

基本类型	包装类型
byte	Byte
int	Integer
short	Short
long	Long
float	Float
double	Double
boolean	Boolean
char	Character

包装类主要是两种方法，一种是本类型和其它类型之间进行转换，另一种是字符串和本类型以及基本类型之间的转换

```
// 包装类
public class Wrapper {
    public static void main(String[] args) {
        // 装箱：将基本类型转换成包装类
        int a = 1;
        Integer v1 = new Integer(a);    // 手动装箱
        Integer v2 = a;                 // 自动装箱

        // 拆箱：将包装类型转成基本类型
        int b = v1.intValue();          // 手动拆箱
        int c = v2;                     // 自动拆箱

        // 包装类用法一：本类型与其他类型转换
        Integer v3 = 3;
        System.out.println("v3-->" + v3);
        float f1 = v3.floatValue();
        System.out.println("f1-->" + f1);
        byte b1 = v3.byteValue();
        System.out.println("b1-->" + b1);
        String str1 = v3.toString();
        System.out.println("str1-->" + str1);
        System.out.println();

        // 包装类用法二：字符串和本类型以及基本类型之间的转换
        String str2 = "122";
        int i1 = Integer.parseInt(str1);
        System.out.println("i1-->" + i1);
        float f2 = Float.parseFloat(str2);
        System.out.println("f2-->" + f2);
        float f3 = 3.1f;
        String str_f2 = String.valueOf(f3);
        System.out.println("str_f2-->" + str_f2);

    }
}
```

```
}
```

## 2.运算符

运算符基本与C语言的运算符相同，这里不过多介绍。

- 算术运算符
- 关系运算符
- 位运算符
- 逻辑运算符
- 赋值运算符
- 条件运算符 (?:)
- instanceof 运算符

该运算符用于操作对象实例，检查该对象是否是一个特定类型（类类型或接口类型）。

1. 左侧对象是右侧对象子类，则返回为真

```
String name = "James";  
boolean result = name instanceof String; // 由于 name 是 String 类型，所以返回真
```

2. 左侧对象兼容右侧对象，则返回为真

```
class Vehicle {}  
  
public class Car extends Vehicle {  
    public static void main(String[] args){  
        Vehicle a = new Car();  
        boolean result = a instanceof Car;  
        System.out.println( result);  
    }  
}
```

### 2.1 比较==与equals区别

- ==

对于基本数据类型，比较它们的值是否相同。对于复合数据类型，则比较它们的存放地址是否相同。

- equals()方法

1. Object类中定义的equals方法，实现方式是用 == 运算。因此，Object类的equals方法是比较对象的内存地址是否一致。
2. 由于Java中所有类都继承自Object这个基类，因此继承它的类可能会复写 equals 方法，所以可能不在是比较其在内存中存储的地址。

```

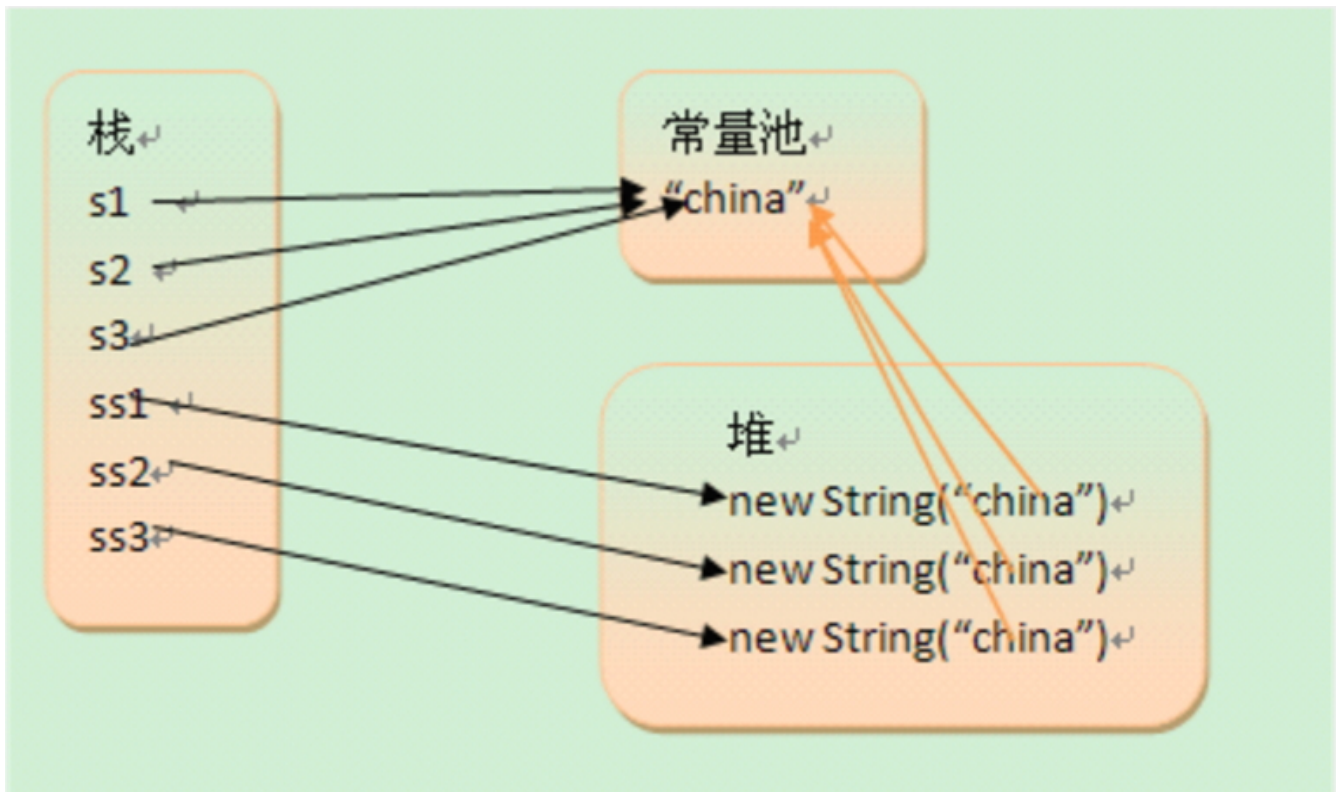
public class EqualCompare {
    public static void main(String[] args) {
        // 基本数据类型
        int a1 = 1;
        Integer a2 = 1;
        int b1 = 1;
        Integer b2 = 1;

        System.out.println(a1 == b1);
        System.out.println(a2.equals(b2));
        System.out.println();

        String s1 = "123"; // 存放在常量池
        String s11 = "123";
        String s2 = new String("123"); // 存放在堆中
        System.out.println(s1 == s11);
        System.out.println(s1 == s2);
        System.out.println(s1.equals(s2));
    }
}

```

附加材料：[Java内存分配之堆、栈和常量池](#)



### 3. 类

- 类：类是一个模板，它描述一类对象的行为和状态。
- 对象：对象是类的一个实例

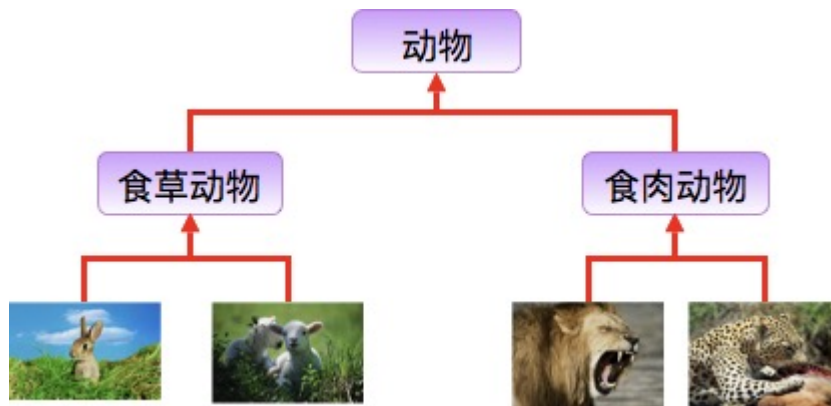
### 3.1 访问控制修饰符

- **private** : 在同一类内可见。使用对象：变量、方法。 **注意：不能修饰类（外部类）**
- **public** : 对所有类可见。使用对象：类、接口、变量、方法
- **protected** : 对同一包内的类和所有子类可见。使用对象：变量、方法。 **注意：不能修饰类（外部类）。**
- 使用修饰符时，与 `protected` 修饰符效果相同

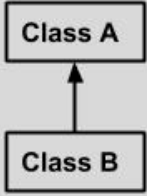
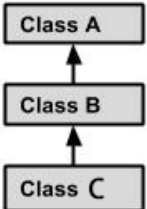
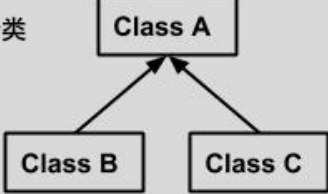
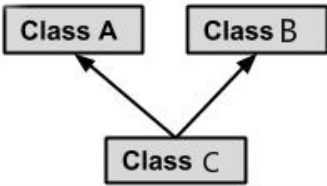
### 3.2 继承 extends

继承就是子类继承父类的特征和行为，使得子类对象（实例）具有父类的实例域和方法，或子类从父类继承方法，使得子类具有父类相同的行为。

- 生活中的继承



- 类和类之间存在一些共有的属性，那么可以将这些共有的属性抽象成为一个基类。然后这些不同的类可以继承这个基类，在修改共有的属性时，只需要在基类中修改即可。
- 继承类型

单继承	 <pre> graph BT     B[Class B] --&gt; A[Class A] </pre>	<pre> public class A {     ..... } public class B extends A {     ..... } </pre>
多重继承	 <pre> graph BT     C[Class C] --&gt; B[Class B]     B --&gt; A[Class A] </pre>	<pre> public class A { ..... } public class B extends A { ..... } public class C extends B { ..... } </pre>
不同类继承同一个类	 <pre> graph BT     B[Class B] --&gt; A[Class A]     C[Class C] --&gt; A </pre>	<pre> public class A { ..... } public class B extends A { ..... } public class C extends A { ..... } </pre>
多继承 (不支持)	 <pre> graph BT     C[Class C] --&gt; A[Class A]     C --&gt; B[Class B] </pre>	<pre> public class A { ..... } public class B { ..... } public class C extends A,B {     ..... } // Java 不支持多继承 </pre>

### 3.2.1 this/super

- this表示当前对象

用法:

- 作为对象, 它表示当前对象的一个引用, 因此可以通过 `this.` 来明确调用当前类的某个方法或字段
- 作为方法, 它表示调用构造方法
- 在匿名类或内部类时, 可以用 `类名.this.` 来明确调用它的外部类某个方法或字段

- super 表示当前对象的父对象。不支持通过 `super.super` 调用父对象的父对象

用法:

- 作为对象, 它表示父类对象的一个引用, 因此可以通过 `super.` 来明确调用父类类的某个方法或字段
- 作为方法, 它表示调用父类构造方法
- 在匿名类或内部类时, 可以用 `类名.super.` 来明确调用它的外部类的父类的某个方法或字段

### 3.2.2 封装

在面向对象程序设计方法中, 封装 (英语: Encapsulation) 是指一种将抽象性函式接口的实现细节部分包装、隐藏起来的方法。

封装可以被认为是一个保护屏障, 防止该类的代码和数据被外部类定义的代码随机访问。

要访问该类的代码和数据, 必须通过严格的接口控制。

封装最主要的功能在于我们能修改自己的实现代码, 而不用修改那些调用我们代码的程序片段。

适当的封装可以让程式码更容易理解与维护，也加强了程式码的安全性。

### 封装的优点

- 1. 良好的封装能够减少耦合。
- 2. 类内部的结构可以自由修改。
- 3. 可以对成员变量进行更精确的控制。
- 4. 隐藏信息，实现细节。

## 3.3 抽象类

- abstract 关键字

如果一个类拥有抽象方法, 这个类必须是抽象类 抽象类未必有抽象方法 子类继承一个抽象类,如果子类不希望也成为抽象类,就必须实现父类中所有的抽象方法

### 抽象类总结规定

- 1. 抽象类不能被实例化(初学者很容易犯的错)，如果被实例化，就会报错，编译无法通过。只有抽象类的非抽象子类可以创建对象。
- 2. 抽象类中不一定包含抽象方法，但是有抽象方法的类必定是抽象类。
- 3. 抽象类中的抽象方法只是声明，不包含方法体，就是不给出方法的具体实现也就是方法的具体功能。
- 4. 构造方法，类方法（用 static 修饰的方法）不能声明为抽象方法。
- 5. 抽象类的子类必须给出抽象类中的抽象方法的具体实现，除非该子类也是抽象类。

## 3.4 接口 interface

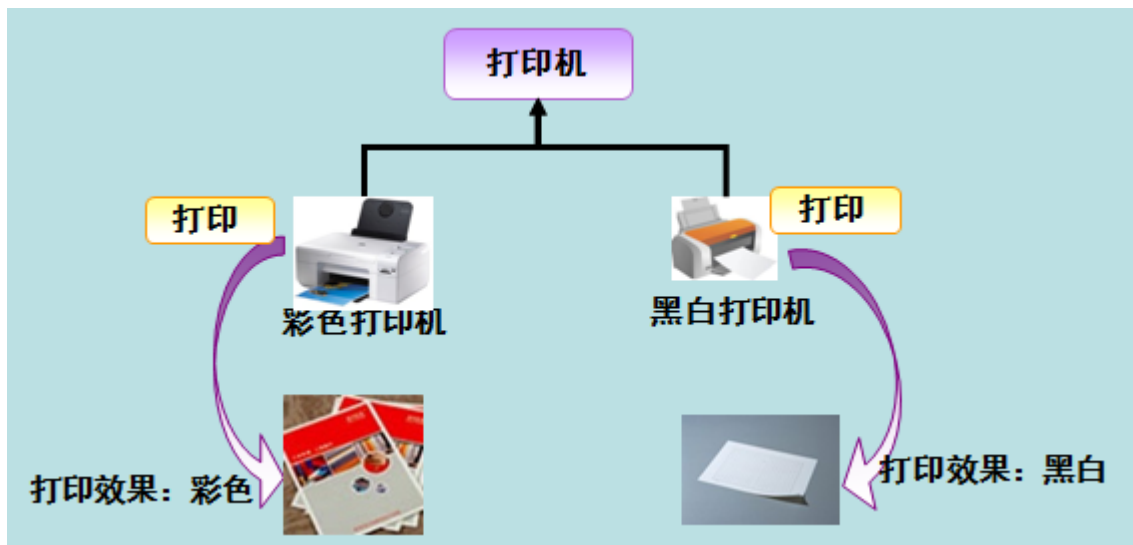
刚刚的抽象类中存在抽象方法，但它任然是一个类。而java只支持单继承，不支持多继承。如果要继承多个类，那么有一个选择，它就是接口。

- 接口并不是类，编写接口的方式和类很相似，但是它们属于不同的概念。类描述对象的属性和方法。接口则包含类要实现的方法。
- 除非实现接口的类是抽象类，否则该类要定义接口中的所有方法。
- 接口无法被实例化，但是可以被实现。一个实现接口的类，必须实现接口内所描述的所有方法，否则就必须声明为抽象类。另外，在 Java 中，接口类型可用来声明一个变量，他们可以成为一个空指针，或是被绑定在一个以此接口实现的对象。

## 3.5 多态

多态是同一个行为具有多个不同表现形式或形态的能力。多态就是同一个接口，使用不同的实例而执行不同操作，如图所示：





多态性是对象多种表现形式的体现。

### 多态存在的三个必要条件

- 继承
- 重写
- 父类引用指向子类对象

### 3.5.1 重写 (Override)

子类与父类方法名，这时发生了方法重写。

当子类对象调用重写的方法时，调用的是子类的方法，而不是父类中被重写的方法。

#### 方法的重写规则

- 参数列表必须完全与被重写方法的相同。
- 返回类型与被重写方法的返回类型可以不相同，但是必须是父类返回值的派生类（java5 及更早版本返回类型要一样，java7 及更高版本可以不同）。
- 访问权限不能比父类中被重写的方法的访问权限更低。例如：如果父类的一个方法被声明为 public，那么在子类中重写该方法就不能声明为 protected。
- 父类的成员方法只能被它的子类重写。
- 声明为 final 的方法不能被重写。
- 声明为 static 的方法不能被重写，但是能够被再次声明。
- 子类和父类在同一个包中，那么子类可以重写父类所有方法，除了声明为 private 和 final 的方法。
- 子类和父类不在同一个包中，那么子类只能够重写父类的声明为 public 和 protected 的非 final 方法。
- 重写的方法能够抛出任何非强制异常，无论被重写的方法是否抛出异常。但是，重写的方法不能抛出新的强制性异常，或者比被重写方法声明的更广泛的强制性异常，反之则可以。
- 构造方法不能被重写。
- 如果不能继承一个方法，则不能重写这个方法。

### 3.5.2 重载 (Overload)

重载(overloading) 是在一个类里面，方法名字相同，而参数不同。返回类型可以相同也可以不同。

每个重载的方法（或者构造函数）都必须有一个独一无二的参数类型列表。

**重载规则：**

- 被重载的方法必须改变参数列表(参数个数或类型不一样);
- 被重载的方法可以改变返回类型;
- 被重载的方法可以改变访问修饰符;
- 被重载的方法可以声明新的或更广的检查异常;
- 方法能够在同一个类中或者在一个子类中被重载。
- 无法以返回值类型作为重载函数的区分标准。

## 3.6 静态代码块(static) 和 final关键字

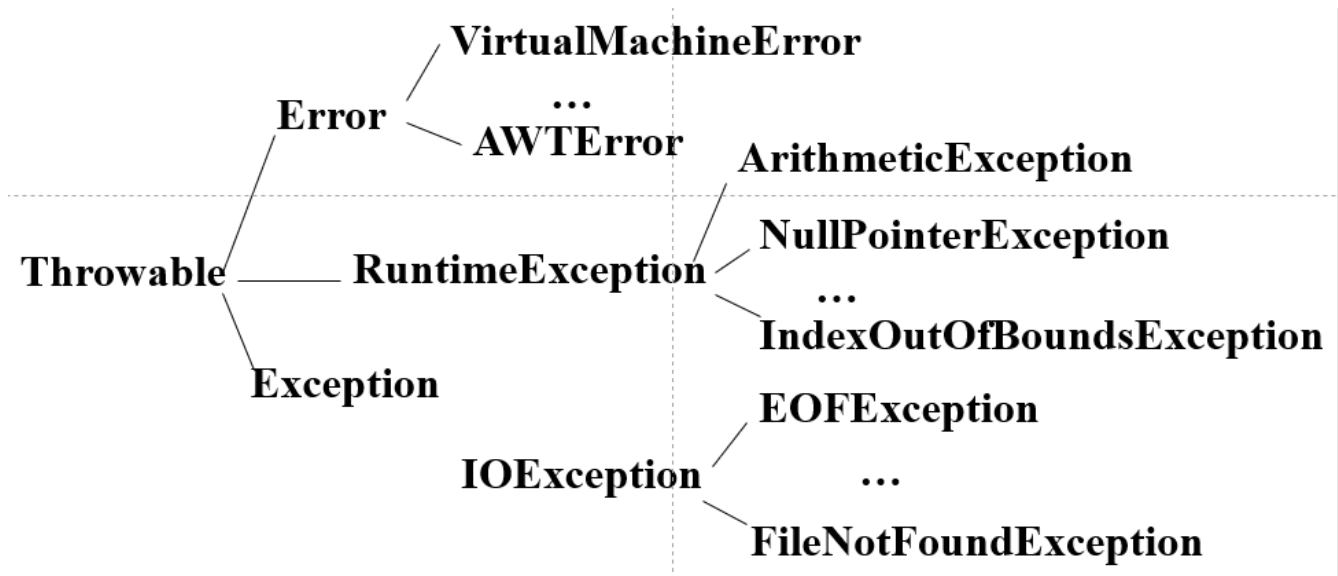
- static
  - 属性:静态属性 全类共有 可以用类名直接访问
  - 方法:类名调用 静态方法中只能访问类的静态成员,不能出现this
    - 静态方法只能被子类的静态方法覆盖,而且没有多态 (只根据引用类型,调用相应的静态方法)
  - 初始代码块:静态初始代码块在类加载的时候执行
    - 类加载:当JVM第一次使用一个类时,读入这个类所对应的.class文件,并保存起来
    - 类加载的时机: 1)创建对象 2)加载子类,需要先加载父类 3)访问静态成员 4)Class.forName("类名")
    - 如果只是声明一个类的引用,不需要类加载
  - 类加载的步骤:
    - 1)如果需要,先加载父类
    - 2)按顺序初始化静态属性,或执行静态初始代码块
- final 变量:常量 一旦赋值,不可改变 final修饰属性的时候,该属性就没有默认值,就必须手动赋值 方法:final方法不能被子类覆盖

---

## 4.异常

异常(Exception) 是在程序运行时打断正常程序流程的不正常的情况。比如:

- 试图打开的文件不存在
- 网络链接中断
- 操作符越界
- 要加载类文件不存在



Exception 类是 Throwable 类的子类。除了Exception类外，Throwable还有一个子类Error。

Java 程序通常不捕获错误。错误一般发生在严重故障时，它们在Java程序处理的范畴之外。Error 用来指示运行时环境发生的错误。例如，JVM 内存溢出。一般地，程序不会从错误中恢复。

异常类有两个主要的子类：IOException 类和 RuntimeException 类。

## 4.1 try-catch

```

public class Main {
    public static void main(String[] args) {
        try {
            double num = 1 / 0;
        } catch (ArithmeticException e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
        System.out.println("这里被执行了");
    }
}

```

- finally一般执行收尾工作

```

try{
    插卡
    输入密码和金额2000
    余额 -= 2000
    吐钱 2000
}
catch(吐钱异常 e){
    余额 += 2000    处理异常
}
finally{
    退卡
}

```

## 5.泛型

泛型提供了**编译时**类型安全检测机制，该机制允许程序员在编译时检测到非法的类型。

泛型的本质是参数化类型，也就是说所操作的数据类型被指定为一个参数。

- 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
- 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的等）。

泛型: JDK5 集合:约束集合中元素的类型

- 泛型没有多态, 赋值语句前后泛型必须一致 方法参数的泛型定义: 通配符 ? : 任何类型 ? extends A: A或A的任何子类 A:可以是类 也可以是接口 ? super A:A或A的父类
- 泛型方法: 泛型定义在修饰符之后,返回值类型之前 T为A或A的子类 <T extends A & B> T为A或A的子类, 同时实现B接口

### 5.1 泛型方法

```
public class Main {
    private static double max(double a, double b) {
        return a > b ? a : b;
    }

    private static <T extends Comparable> T max_T(T a, T b) {
        // 等于返回0, 大于返回1, 小于返回-1
        if (a.compareTo(b) > 0)
            return a;
        else
            return b;
    }

    public static void main(String[] args) {
        Integer i1 = 1;
        Integer i2 = 2;
        Integer i3 = max_T(i1, i2);
        System.out.println(i3);
    }
}
```

### 5.2 泛型类

```

package code5_2;

public class Shape<T> {
    private T temp;

    Shape() {

    }

    Shape(T temp) {
        this.temp = temp;
    }

    public void setTemp(T temp) {
        this.temp = temp;
    }

    public T getTemp() {
        return this.temp;
    }

}

```

## 5.3 泛型通配符

```

public class GenericTest {

    public static void main(String[] args) {
        List<String> name = new ArrayList<String>();
        List<Integer> age = new ArrayList<Integer>();
        List<Number> number = new ArrayList<Number>();

        name.add("icon");
        age.add(18);
        number.add(314);

        //getUperNumber(name);//1
        getUperNumber(age);//2
        getUperNumber(number);//3
    }

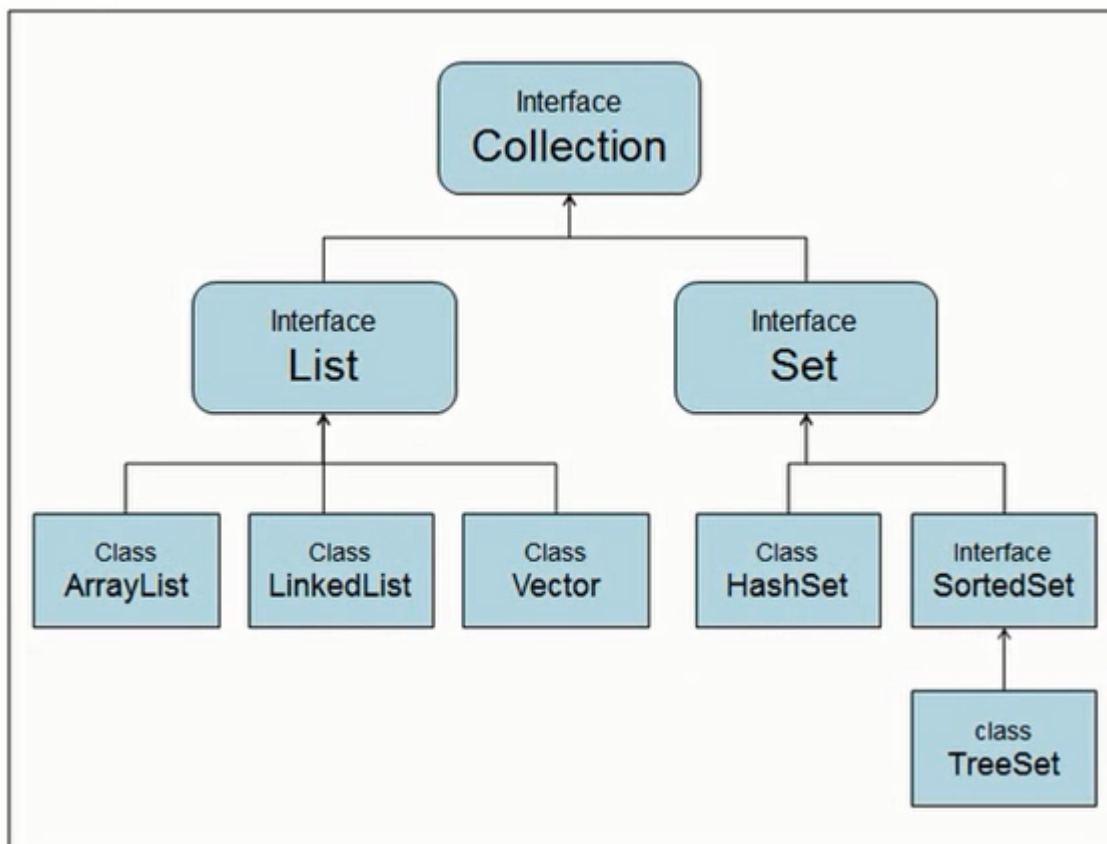
    public static void getData(List<?> data) {
        System.out.println("data :" + data.get(0));
    }

    public static void getUperNumber(List<? extends Number> data) {
        System.out.println("data :" + data.get(0));
    }
}

```

}

## 6. 集合框架



集合：存储多个对象

Collection:元素是Object

常用方法:

add(Object) 把元素添加到集合中

addAll(Collection c):把c集合中的所有元素添加到集合中

clear():清空集合

contains(Object o):判断o元素在集合中是否存在

remove(Object o):从集合中删除元素o

size():返回集合的长度

toArray():将集合转化为数组

遍历

1. 迭代器遍历 Collection 可以通过迭代器,删除集合中的元素

2. for-each Collection JDK5

List: 元素按顺序存储(下标) 元素可以重复

常用方法:

add(int pos, Object o):将元素插入到指定位置

remove(int pos):删除指定位置的元素

get(int pos):获得指定位置的元素  
indexOf(Object o):获得o元素的首下标  
lastIndexOf(Object o):获得o元素的尾下标  
set(int pos,Object o):将元素o设置到pos位置,覆盖原有的元素  
subList(int start,int end):获得某段位置的子集合

遍历:

- 1.下标遍历 List
- 2.迭代器遍历 Collection 可以通过迭代器,删除集合中的元素
- 3.for-each Collection JDK5

实现类:

- 1.ArrayList 数组实现 查询快 增删慢 线程不安全 并发效率高 JDK1.2
- 2.Vector 数组实现 线程安全 并发效率低 JDK1.0
- 3.LinkedList 链表实现 查询慢 增删快

排序:

Collections.sort(List) : 要求集合中的元素实现Comparable接口

Collections.sort(List,Comparator):Comparator比较器, 实现排序逻辑 集合中的元素不需要实现Comparable接口

list.sort(Comparator):直接对list调用sort方法排序,只能传入Comparator JDK8

Set:元素无顺序 元素内容不可重复

常用方法:等同于Collection接口

遍历:

- 1.迭代器遍历 Collection 可以通过迭代器,删除集合中的元素
- 2.for-each Collection JDK5
- 3.forEach() 方法 需要实现java.util.function.Consumer接口 JDK8

实现类:

1. HashSet : 保证内容重复对象只有一个
  - 1)覆盖hashCode()方法,保证相同对象返回相同的int , 尽可能保证不同对象返回不同的int
  - 2)覆盖equals()方法,保证相同对象返回true
2. LinkedHashSet HashSet的子类, 维护元素添加到Set中的顺序
3. TreeSet SortedSet(Set的子接口)的实现类 自动实现对元素的排序 依照排序规则判断重复对象

Queue:队列 FIFO

常用方法:

add() : 添加元素  
offer(): 添加元素 优先使用  
remove():删除元素  
poll():删除元素 优先使用  
element():获取队列的头元素  
peek():获取队列的头元素 优先使用

实现类:

LinkedList ConcurrentLinkedQueue

Map : 元素是 key-value key无顺序,不可重复 value 可重复

常用方法:

get(Object key) :通过key查找对应的value  
put(Object key,Object value):将key-value添加到Map中 如果key已存在,新的value覆盖旧的value  
remove(Object key):删除key所对应的key-value对  
size():长度

`containsKey(Object key)`: 判断key是否存在  
`containsValue(Object value)`: 判断value是否存在

遍历:

1. `keySet():Set` 遍历Map中所有的key
2. `values():Collection` 遍历Map中所有的value
3. `entrySet():Set` Set中的元素为Map.Entry对象. 代表了一个键值对 遍历Map中的所有的键值对
4. `forEach()`: 实现BiConsumer接口 直接遍历Map JDK8

实现类:

HashMap : 依靠哈希算法保证key不重复 1.2 线程不安全 并发效率高 允许用null作为key或value

Hashtable: 1.0 线程安全 并发效率低 不允许用null作为key或value

LinkedHashMap : HashMap的子类 维护元素添加到集合中的顺序

TreeMap SortedMap (Map的子接口)的实现类 自动对key排序

Properties Hashtable的子类 key和value都是String 通常用于配置文件的处理

---

## 7.线程

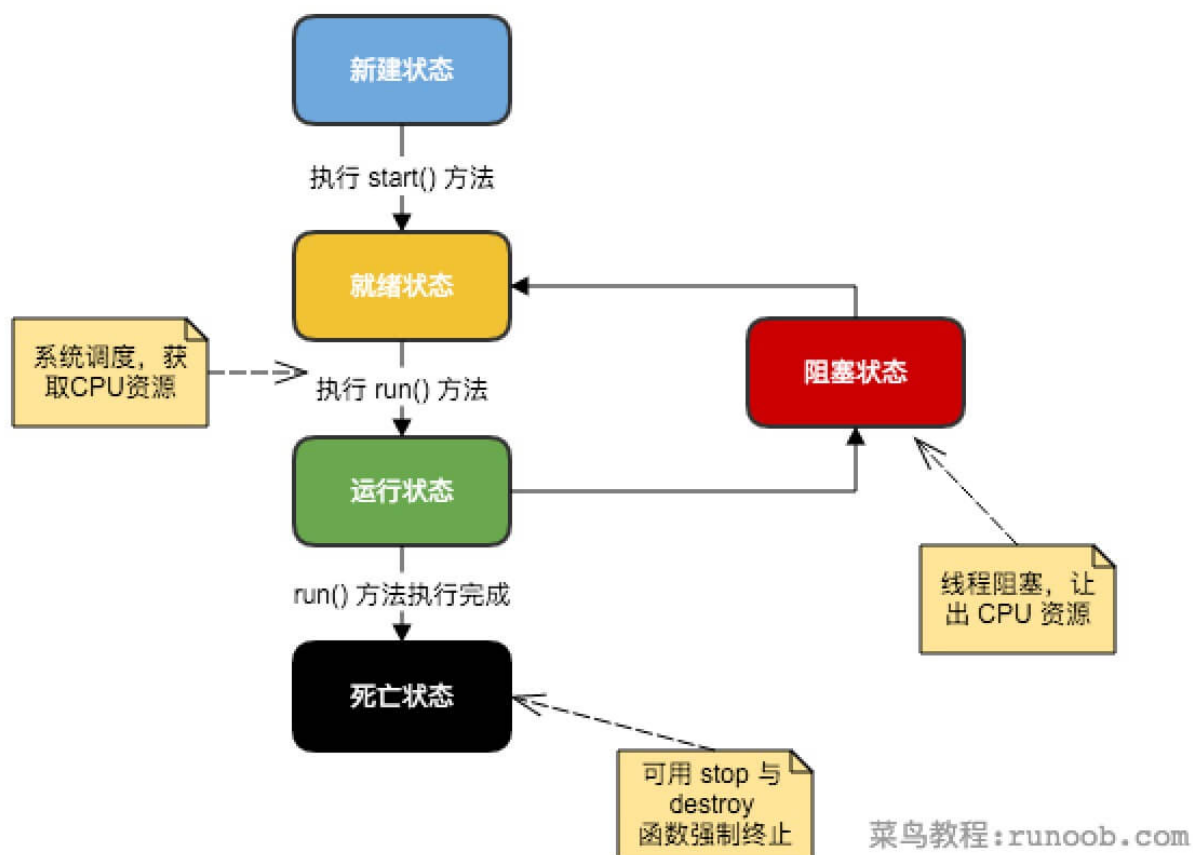
首先定义一个术语:

- 进程: 一个进程包括由操作系统分配的内存空间, 包含一个或多个线程。一个线程不能独立的存在, 它必须是进程的一部分。一个进程一直运行, 直到所有的非守护线程都结束运行后才能结束。
- 一条线程指的是进程中一个单一顺序的控制流, 一个进程中可以并发多个线程, 每条线程并行执行不同的任务。

Java进程中有一主线程负责main方法的执行, 而在main方法的执行中创建的线程, 就称为程序中的其他线程(相对主线程而言的)。

- 线程的生命周期





- **新建**:当一个Thread类或其子类的对象被声明并创建时，新生的线程对象处于新建状态。此时它已经有了相应的内存空间和其它资源。
- **就绪**:线程创建之后就具备了运行的条件，一旦调度机制把CPU时间片分配给线程，线程开始运行了（运行run()方法）。
- **死亡**:run方法结束。此时，调度机制将释放掉分配给线程的内存。
- **阻塞**:线程能够运行，但有某个条件阻止它的运行。此时，调度机制将忽略该线程，不会给线程分配CPU时间片。

一个线程进入阻塞状态，可能有如下原因：

- 调用sleep(int milliseconds)使线程进入休眠状态。
- 线程要执行一段同步代码，由于无法获得相关的同步锁而陷入阻塞状态，只有等获得了同步锁，才能进入就绪状态。
- 线程试图在某个对象上调用其同步控制方法，但是对象锁不可用。
- 通过调用wait()使线程挂起。直到线程得到notify()或notifyAll()消息，线程才会进入就绪状态。
- 线程在等待某个输入/输出完成。

## 7.1 创建线程

三种方式介绍常用的两种

```
public class MyThread extends Thread {
    @Override
    public void run() {
```

```

        System.out.println("Thread 被执行");
    }
}

public class MyRunnable implements Runnable {
    @Override
    public void run() {
        System.out.println("runnable thread 被执行");
    }
}

public class Main {
    public static void main(String[] args) {
        MyThread myThread = new MyThread();
        myThread.run();
        Thread thread = new Thread(new MyRunnable());
        thread.run();
    }
}

```

## 7.2线程同步

**原因：**线程同步是为了保证在多个线程访问同一个对象的时候不会出现访问冲突的一种机制。

**实现：**通过给对象（或类）加锁来完成线程的同步。synchronized关键字可以作为方法的修饰符，也可作为方法内的语句，也就是平时说的同步方法和同步语句块。

- 同步方法

```
public synchronized void save() {}
```

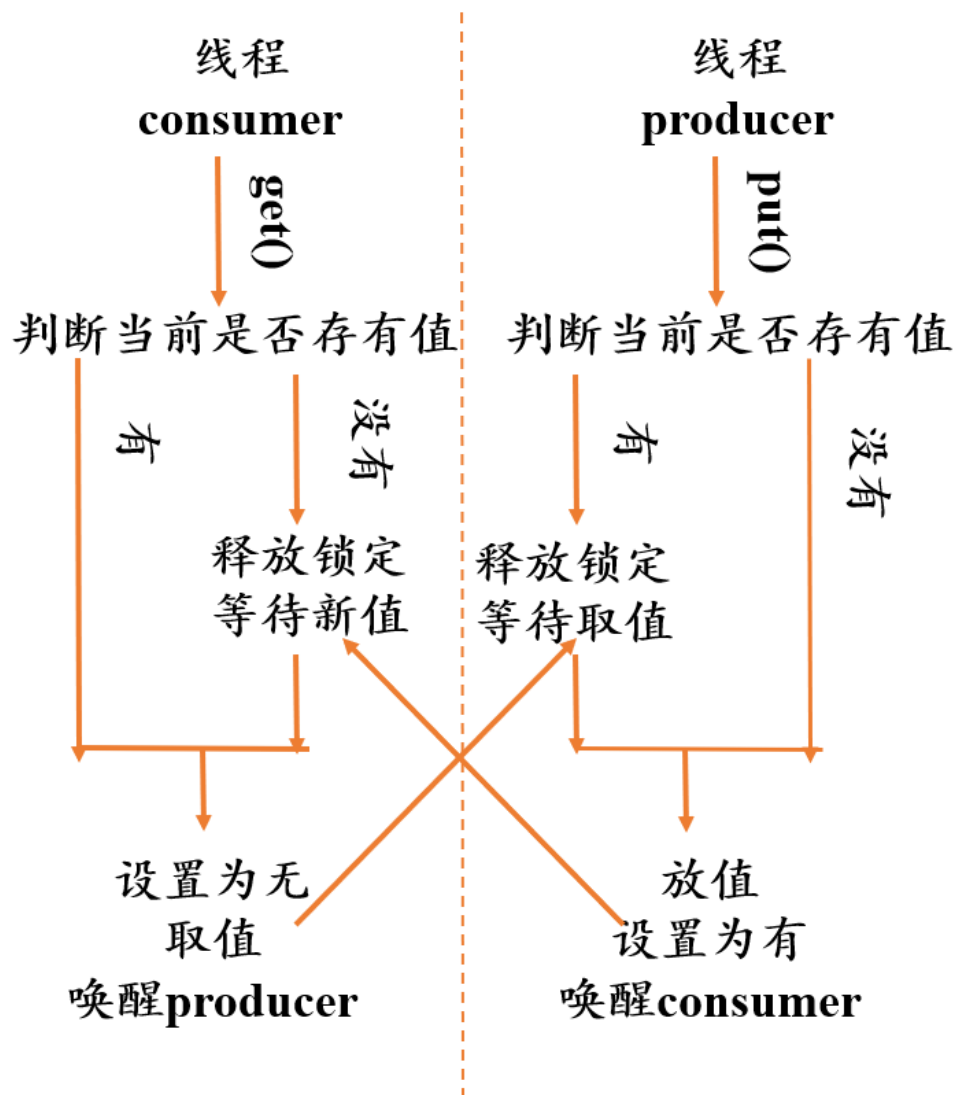
- 同步代码块

```
synchronized(object){}
```

## 7.3 线程通讯

- wait()/notify机制

生产者消费者问题：



- 使用wait()方法使本线程等待，暂时让出CPU的使用权，并允许其它线程使用这个同步方法。
- 其它线程如果在使用这个同步方法时不需要等待，那么它用完这个同步方法的同时，应当执行notifyAll()方法通知所有的由于使用wait()方法而处于等待的线程结束等待。

```
public class Producer implements Runnable {
    private int contents;
    private boolean available = false;
    private Thread producer;
    private Thread consumer;

    public synchronized int get() {
        while (available == false) {
            try {
                wait(); //释放锁，等候producer放值
            } catch (InterruptedException e) {
            }
        }
        available = false;
        System.out.println("Consumer gets:" + contents);
    }
}
```

```

        notifyAll();
        return contents;
    }

    public synchronized void put(int value) {
        while (available == true) {
            try {
                wait(); //释放锁, 等候consumer取值
            } catch (InterruptedException e) {
            }
        }
        contents = value;
        available = true;
        System.out.println("Producer produces:" + contents);
        notifyAll();
    }

    @Override
    public void run() {
        if (Thread.currentThread() == producer) {
            for (int i = 0; i <= 9; i++) put(i);
        } else {
            for (int i = 0; i <= 9; i++) get();
        }
    }
}

Producer(){
    producer = new Thread(this);
    consumer = new Thread(this);
    producer.start();
    consumer.start();
}

public static void main(String[] args) {
    new Producer();
}
}

```

## Thread 方法

下表列出了Thread类的一些重要方法：

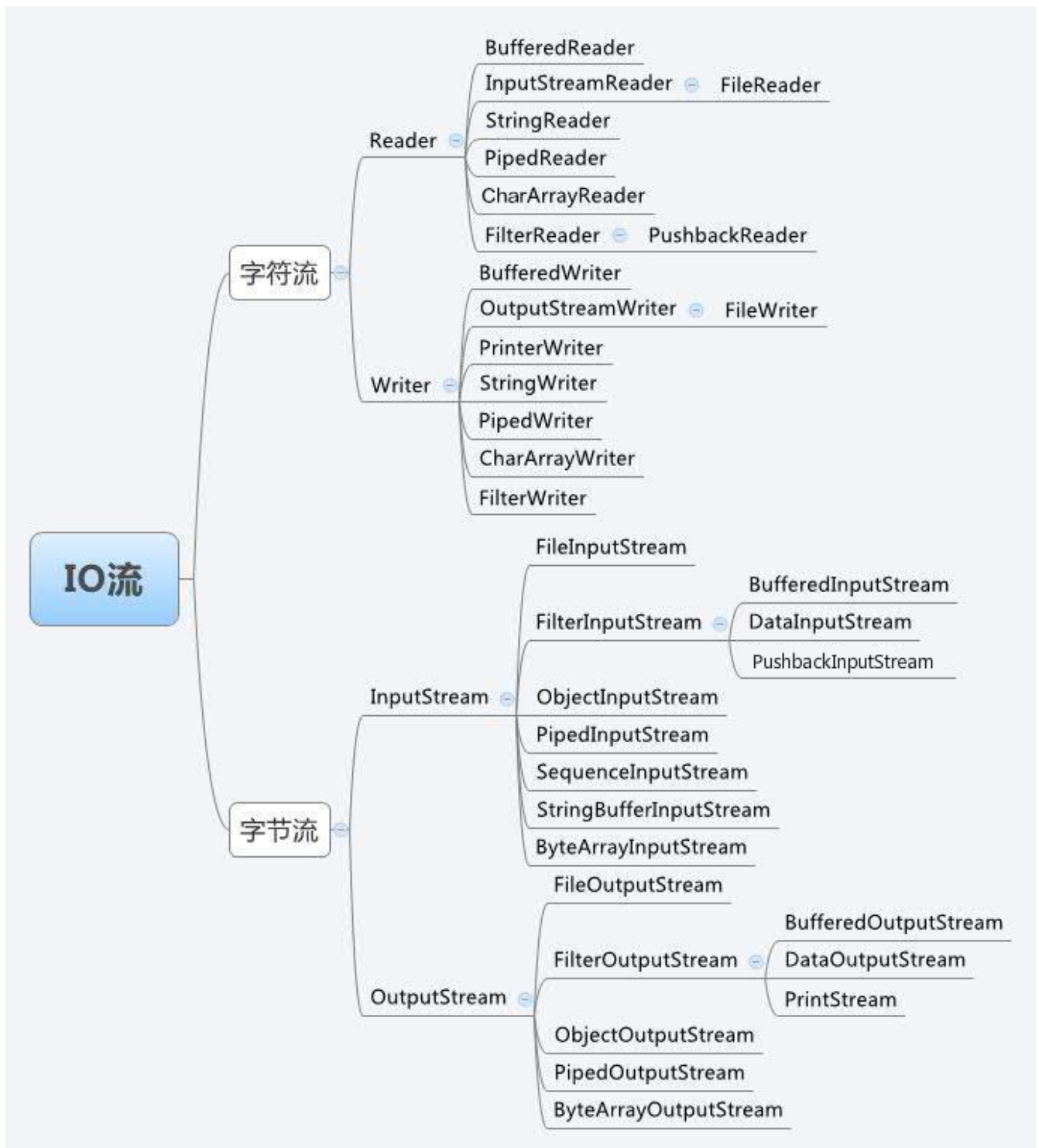
序号	方法描述
1	<b>public void start()</b> 使该线程开始执行；Java 虚拟机调用该线程的 run 方法。
2	<b>public void run()</b> 如果该线程是使用独立的 Runnable 运行对象构造的，则调用该 Runnable 对象的 run 方法；否则，该方法不执行任何操作并返回。
3	<b>public final void setName(String name)</b> 改变线程名称，使之与参数 name 相同。
4	<b>public final void setPriority(int priority)</b> 更改线程的优先级。
5	<b>public final void setDaemon(boolean on)</b> 将该线程标记为守护线程或用户线程。
6	<b>public final void join(long millisec)</b> 等待该线程终止的时间最长为 millis 毫秒。
7	<b>public void interrupt()</b> 中断线程。
8	<b>public final boolean isAlive()</b> 测试线程是否处于活动状态。

测试线程是否处于活动状态。上述方法是被Thread对象调用的。下面的方法是Thread类的静态方法。

序号	方法描述
1	<b>public static void yield()</b> 暂停当前正在执行的线程对象，并执行其他线程。
2	<b>public static void sleep(long millisec)</b> 在指定的毫秒数内让当前正在执行的线程休眠（暂停执行），此操作受到系统计时器和调度程序精度和准确性的影响。
3	<b>public static boolean holdsLock(Object x)</b> 当且仅当当前线程在指定的对象上保持监视器锁时，才返回 true。
4	<b>public static Thread currentThread()</b> 返回对当前正在执行的线程对象的引用。
5	<b>public static void dumpStack()</b> 将当前线程的堆栈跟踪打印至标准错误流。

## 8.文件IO

Java.io 包几乎包含了所有操作输入、输出需要的类。所有这些流类代表了输入源和输出目标。一个流可以理解为一个数据的序列。输入流表示从一个源读取数据，输出流表示向一个目标写数据。



- 流：个流可以理解为一个数据的序列。
- 缓存流：从磁盘读入到内存并将数据序列分块返回。

## 8.1 File类

Java文件类以抽象的方式代表文件名和目录路径名。该类主要用于文件和目录的创建、文件的查找和文件的删除等。File对象代表磁盘中实际存在的文件和目录。

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

```

public class Main {
    public static void main(String[] args) {
        File file = new
File("C:\\Users\\Administrator\\Desktop\\JavaTrain\\mangzhong.mp3");
        if(!file.isDirectory())
            file = new File(file.getParent());
        File[] listFiles = file.listFiles();
        System.out.println(listFiles.length);
        for (File f:listFiles) {
            System.out.println(f.getName());
        }
    }
}

```

[参考](#)

## 8.2 FileInputStream FileOutputStream

- 该流用于从文件读取数据

```

public class Main {
    public static void main(String[] args) throws IOException {
        File file = new File("C:\\Users\\Administrator\\Desktop\\JavaTrain\\cont.txt");
        File out = new File("C:\\Users\\Administrator\\Desktop\\JavaTrain\\img\\cont.txt");
        FileInputStream inputStream = new FileInputStream(file);
        FileOutputStream outputStream = new FileOutputStream(out);

        int size = inputStream.available();
        for (int i = 0; i < size; i++) {
            byte temp = (byte) inputStream.read();
            outputStream.write(temp);
            System.out.println((char) temp);
        }
        inputStream.close();
        outputStream.close();
    }
}

```

## 8.3 InputStreamReader OutputStreamWriter

```

public class Main {
    public static void main(String[] args) throws IOException {
        File file = new File("C:\\Users\\Administrator\\Desktop\\JavaTrain\\cont.txt");
        File out = new File("C:\\Users\\Administrator\\Desktop\\JavaTrain\\img\\cont.txt");
        FileInputStream inputStream = new FileInputStream(file);
        FileOutputStream outputStream = new FileOutputStream(out);
        InputStreamReader inputStreamReader = new InputStreamReader(inputStream, "UTF-8");
        OutputStreamWriter outputStreamWriter = new OutputStreamWriter(outputStream, "UTF-8");
    }
}

```

```

while (inputStreamReader.ready()){
    char temp = (char)inputStreamReader.read();
    outputStreamWriter.write(temp);
    System.out.println(temp);
}
outputStreamWriter.append("123456");
inputStreamReader.close(); // 注意顺序

inputStream.close();
outputStreamWriter.close();
outputStream.close();
}
}

```

## 9.网络编程

### 报文格式

The diagram shows an HTTP request message with the following components and annotations:

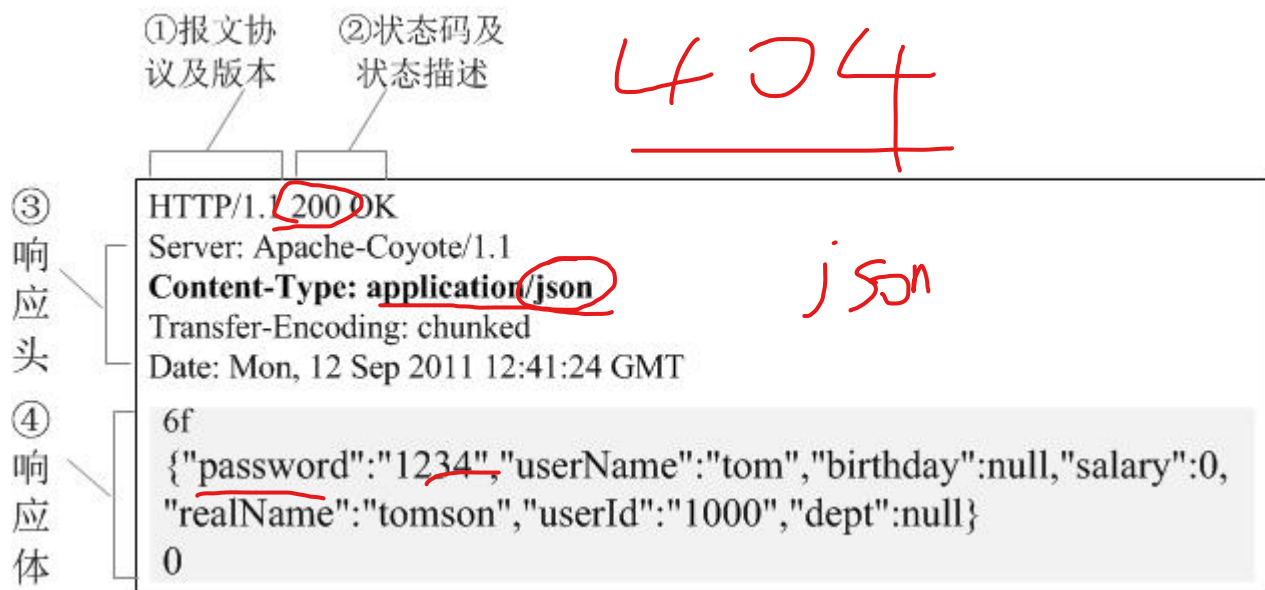
- ① 请求方法 (Request Method):** POST
- ② 请求URL (Request URL):** /chapter17/user.html
- ③ HTTP协议及版本 (HTTP Protocol and Version):** HTTP/1.1
- ④ 报文头 (Message Header):**
  - Accept: image/jpeg, application/x-ms-application, ..., \*/\*
  - Referer: http://localhost:8088/chapter17/user/register.html?code=100&time=123123
  - Accept-Language: zh-CN
  - User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1;
  - Content-Type: application/x-www-form-urlencoded
  - Host: localhost:8088
  - Content-Length: 112
  - Connection: Keep-Alive
  - Cache-Control: no-cache
  - Cookie: JSESSIONID=24DF2688E37EE4F66D9669D2542AC17B
- ⑤ 报文体 (Message Body):** [name=tom&password=1234&realName=tomson]

图 15-4 HTTP 请求报文

1 2 3 4 5

POST





- GET请求

User-Agent: 产生请求的浏览器类型。 Mozilla/4.0 (compatible; MSIE 6.0; windows NT 5.1;SV1) Accept: 客户端可识别的内容类型列表。 \*/\* Host: 请求的主机名, 允许多个域名同处一个IP地址, 即虚拟主机。 connection:Keep-Alive

GET /sn/index.php?sn=123&n=asa HTTP/1.1  
 Accept: \*/\*  
 Accept-Language: zh-cn  
 host: localhost

Content-Type: application/x-www-form-urlencoded  
 Content-Length: 12  
 Connection:close

```

public class GET {
    public String send(String url, String param) throws FileNotFoundException {
        String result = "";
        BufferedReader in = null;
        File out = new File("C:\\Users\\Administrator\\Desktop\\JavaTrain\\img\\cont.mp3");
        FileOutputStream outputStream = new FileOutputStream(out);

        try {
            String urlNameString = url + "?" + param;
            URL realUrl = new URL(urlNameString);
            // 打开和URL之间的连接
            URLConnection connection = realUrl.openConnection();
            // 设置通用的请求属性
            connection.setRequestProperty("accept", "*/*");
            connection.setRequestProperty("connection", "Keep-Alive");
            connection.setRequestProperty("user-agent",
                "Mozilla/4.0 (compatible; MSIE 6.0; windows NT 5.1;SV1)");
  
```

```

// 建立实际的连接
connection.connect();
// 获取所有响应头字段
Map<String, List<String>> map = connection.getHeaderFields();
// 遍历所有的响应头字段
for (String key : map.keySet()) {
    System.out.println(key + "--->" + map.get(key));
}
// 定义 BufferedReader输入流来读取URL的响应
InputStream inputStream = connection.getInputStream();
int size = inputStream.available();
for (int i = 0; i < size; i++) {
    outputStream.write(inputStream.read());
}
outputStream.close();
inputStream.close();

//      in = new BufferedReader(new InputStreamReader(
//          connection.getInputStream()));
//      String line;
//      while ((line = in.readLine()) != null) {
//          result += line;
//      }
} catch (Exception e) {
    System.out.println("发送GET请求出现异常! " + e);
    e.printStackTrace();
}
// 使用finally块来关闭输入流
finally {
    try {
        if (in != null) {
            in.close();
        }
    } catch (Exception e2) {
        e2.printStackTrace();
    }
}
return result;
}
}

```

- POST请求

```
POST /sn/index.php HTTP/1.1
Accept: */*
Accept-Language: zh-cn
host: localhost

Content-Type: application/x-www-form-urlencoded
Content-Length: 12
Connection:close
sn=123&n=asa
```

## 网络MP3播放器Demo

---

### 1.使用第三方解决方案 ([jl1.0.jar](#))播放MP3音乐

```
public class PlayLocalMP3 {
    public static void main(String[] args) throws FileNotFoundException, JavaLayerException
    {
        File file = new
File("C:\\Users\\Administrator\\Desktop\\JavaTrain\\mangzhong.mp3");
        FileInputStream fileInputStream= new FileInputStream(file);
        Player player = new Player(fileInputStream);
        player.play();
    }
}
```

## 参考

---

- [菜鸟教程](#)
- [卢瑞峰java第一次培训](#)