

第一章and第二章 JAVA基础

1. Java程序基本结构

Java语言的源程序是一个或多个以.java为扩展名的文件

1.1 三大基本部分：

- 一个包声明package语句(可选);
package database;
- 任意数量的引入import语句(可选);
import java.applet.Applet;
- 类和接口声明。
class Hello{ ... }
interface DataCollect{ ...}

1.1.1包声明： package语句

- 包是类和接口的集合，即为类库；
- 用类库管理类，方便对类和接口管理，减少类名、接口名之间的重名问题；
- Java的类都包含在类库中，package语句为类、接口(或者说是字节码文件)来指定所属的类库(包)。
- 在一个源程序中，只能有一个包声明语句，且是程序的第一条语句。

1.1.2引入语句： import语句




- 源程序中可以有任意条import引入语句；
- 当源程序在编译时，会将需要的在引入语句中的类引入到程序中。
- import语句在包语句后，所有类或接口之前。
- import语句有两种形式：
1: **import 包名.类名;** 2: **import 包名.*;**

1.1.3类和接口声明

- 类和接口是程序的基本组成单元；
- 类是由成员变量和成员方法等组成，表示了对象的基本属性和行为；
- 接口表现了对对象所具有的行为规范。
- 源程序中至少有一个类或接口创建。

1.2注释

三种注解：

-  image-20220908153601801
-  image-20220908153512776
-  image-20220908153541394

注释不可以嵌套

1.2关键词

2. 基本数据类型

image-20220913222825471

2.0 补充

2.0.1 判断数据类型

```
System.out.println(s instanceof String);
```

判断s是否是某种特定的数据类型

2.0.2 输入数据—Scanner

```
import java.util.Scanner;

Scanner in = new Scanner(System.in);
String name = in.nextLine();
Double a = in.nextDouble();
int b = in.nextInt();
```

注意，in.nextInt 和 in.nextDouble会留下一个回车号，再次使用in.nextLine时会无法读入数据。可以用一个单独的in.nextLine();吃掉回车号或者干脆使用in.next()。

2.1 整数类型

1字节-2字节-4字节-8字节

在JAVA中直接给出一个整型默认为int

- 将一个byte或short取值范围内的数值赋值给其，系统会将其视为相应的类型。byte/short n = 100;
- 将一个超过int型范围的整数赋值给long型，必须要在数字后面加一个L或l。如果没有超过，可以省略。

2.2 浮点数类型

float 4字节 double 8字节

Java中使用浮点数默认为double，需要为float时后缀需要F或f（不可省略）

2.3 字符类型

char 2字节

Java采用Unicode字符集编码

2.4 布尔类型

boolean 只有真和假 true false

Java中0和1不能代表真假。

2.5 引用数据类型*

引用类型(**数组、class或interface**)声明变量时, 是不会为变量(即对象)分配存储空间。它们声明的变量不是数据本身, 而是数据的引用(reference), 需用new运算符来为引用类型的变量分配存储空间;

- 引用:类似C/C++中的指针, 但又不同于C/C++中的指针, 它的引用必须由Java的虚拟机创建和管理。Java语言本身不支持指针;
- **引用类型变量的值是一个数据的引用(即地址)**。它是对占有由多个贮存单元构成的贮存空间的引用。引用类型的变量通过点“.”运算符访问它的成员。

3 变量与常量

3.1 标识符

取名的规则:

- 必须由字母、下划线或美元符开头的; (**数字不能开头**)
- 并由字母、数字、下划线和美元符组成的;
- 不能与关键字同名;

好的取名习惯:

- 类名、接口名用名词, 大小写混用, 第一个字母大写
`class WorldTool`
- 方法名用动词, 大小写混用, 第一个字母小写
`depositAccount()`
- 变量名用名词或形容词等, 大小写混用, 第一个字母小写
`currentCustomer`
- 常量符号用全部大写, 并用下划线将词分隔
`PERSON_COUNT`

3.2 变量

变量的分类及作用域


依变量创建所在处可分为:

- 成员变量; (全局)
- 方法的变量(包含参数); (局部)
- 语句块的变量; (局部)
- 异常处理的变量。 (局部)

3.3 变量的类型转换

隐式转换

- 转换的两种类型相互兼容
- 目标类型的取值范围比原类型大

image-20220913153119983

强制类型转换

```
int n;    byte b = ( byte ) n;
```

3.4 常量

4 Java中的运算符

4.1 算数运算符

4.2 赋值运算符


4.3 关系运算符

4.4 逻辑运算符

与 或非 短路与 短路或

4.5 位运算符

用于进行二进制的运算

image-20220913164125522

5 程序的结构

5.1 顺序结构

5.2 选择结构

5.2.1 if- else

```
if () {  
    #语句块  
} else {  
  
} else {  
  
}
```

5.2.2 三目运算符

判断条件? 表达式一: 表达式二;

5.2.3 switch

```
switch(/*表达式*/){  
    case 常量值1 :  
        语句块1  
        break;  
    default:  
  
}
```

5.3 循环结构

5.3.1 while

5.3.2 do-while

5.3.3 for

5.3.4 循环嵌套

基本等于C

5.4 循环中断

5.4.1 break

用法一：基本和C相同

用法二：带标记的break

在较深的循环嵌套中，可以使用**带标记的break语句**（有点像goto），标记须在break所在循环的外层循环之前，否则与普通break相同。

```
public class sd {
    public static void main(String[] args) {
        label:          //创建标签 注意后面为引号不是分号
        for (int i = 1; i < 10; i++){
            for(int j = 1; j < 10; j++){
                System.out.println(i + "," + j);
                if(j % 2 == 0)
                    break label; //跳出
            }
        }
    }
}
```

5.4.2 continue

用于结束本次循环

第三章 数组与方法

1. 数组

1.1 基本属性与定义

基本属性

- 在Java中，数组是引用类型，本身也会占用一个内存地址。
- 数组元素的**下标不能越界**。
- 数组是一个对象，数组声明不能创建对象本身，而创建一个引用。数组元素由new语句或数组初始化软件动态分配
- 数组有一个length属性，数组的长度，数组名.length 能访问。

声明

```
int[] a;
int a[];
//两种均可，但面向对象的本质决定了第一种更合适
```

1.2 数组的初始化

1.2.1 静态初始化

```
//形式一
int[] array;
array = new int[]{1,2,3,4,5}
//形式二
int[] array = new int[]{1,2,3,4,5}
//或简写(仅能在同时声明与初始化)
int[] array = {1,2,3,4,5}
```

1.2.2 动态初始化

```
int[] array = new int[10];
```

初始化时指定数组的长度，系统为数组元素分配初始值（一般为0或者null或false）

1.3 数组的常用操作

1.3.1 访问数组

1.3.2 数组遍历

有一些代码可敲

1.4 数组的内存原理

- 数组本身是一个引用变量，被储存在栈（Stack）中。
- 数组初始化后，数组对象被储存在堆（heap）内存中。
- 数组变量储存了数组对象的首地址
- 数组初始化完成后，空间分配结束，数组无法变长。可改变一个数组变量所引用的数组造成数组变长的假象，
- 复制数组时，“=”仅能是两个数组变量指向同一个数组对象。

1.5 二维数组

1.5.1 声明与初始化

```
int[][] array; int array[][];
//动态初始化
array = new int[3][2];
array[0] = {1, 2};
array[1] = {3, 4};
array[2] = {5, 6};
//静态初始化
array = new int[][]{
    {1},
    {2,3},
    {4}
};

int[][] array = {
    {1},
    {2,3},
    {4}
}
```

1.5.2 锯齿数组

每一行数组长度不同

1.6 字符串

字符串是一串字符组成的数据，并用 "" 包括起来。字符串常量是String类型的对象。

Java编译器在对字符串数据与其它类型数据使用"+"运算符连接操作编译时，总是首先将其它类型数据转换为字符串类型，然后再进行字符串连接。

- 例: "Age: "+18 ==> "Age: 18"

1.6.1 字符串与基本数据的转化

- 字符串转换为整型

Integer类的静态方法 parseInt() //返回值为数字int

```
public static int parseInt(String s, int radix);  
//radix 为字符串s进制，默认为10，该函数将字符串 s 转换为 十进制数字  
System.out.println(Integer.parseInt("1111",2)); //15  
System.out.println(Integer.parseInt("101",8)); //65  
System.out.println(Integer.parseInt("1111",10)); //1111  
System.out.println(Integer.parseInt("1A",16)); //26
```

也有double类等

- 整形转换为字符串

String类的静态方法 valueOf()

```
public static String valueOf(int n);  
//将基本数据类型转换为String
```

2. 方法

```
修饰符 返回值类型 方法名(参数类型 形参1, 参数类型 参数2){  
    方法体;  
    return (返回值);  
}
```

2.1 返回值

方法可以有或没有返回值

- 有返回值可作为语句 或 一个值
- 无返回值仅能作为语句

2.2 参数传递

方法在被调用时，其参数的数据传递是**值传递**，即实际参数传值给形式参数。

注：在Java和C等语言中，只有值传递，没有引用传递，即使是引用变量，也只是将变量的地址赋给形参，本质还是值传递。C++中有引用传递。

1) 形式参数是简单类型

在方法调用时，实际参数将其存储单元的数据赋值给形式参数

2) 形式参数是引用类型

在方法中，引用类型的参数没有发生引用的改变，则形式参数对引用中的变量值改变自然会影响到实际参数引用中变量的值

如 **数组** 书P82页

2.3 方法重载

在同一个类中创建的具有相同方法名，但是参数不同的方法。

- 参数数量不同
- 参数数量相同，但是对应的类型不同

注意：

```
double max(int num1, double num2);
double max(double num1, int num2);
//只是顺序不同，可能会导致编译错误，如传入两个int类型，就会出现错误
```

2.4 方法递归

第四章 面向对象（上）

1 面向对象的概念

面向对象程序（OOP）依照现实世界的实体的特点，把复杂的现实的事物按它们所共有的状态和行为抽象并封装。

- 封装性
- 继承性
- 多态性

2 类与对象

2.1 类的声明

```
[public] class <clsName> extends <supCls> implements <intf>
```

- class是表示创建类的关键字；
- 是Java合法标识符；
- [public]是可选项，表示该类是public类；类的可选项还有abstract、final等等；
- extends 则是继承性表示，该类继承了类

- implements 则是对接口实现表示，该类实现了接口

类中一般含有 1) 成员变量、2) 成员方法、3) 类的构造器

2.2 对象的创建

```
类名 对象名;           //对象的声明
对象名 = new 类名();    //对象实例化、创建
Person p = new Person();
Person p 声明了一个Person类型的引用变量
```

- 对象创建p1,p2有不同的储存空间
- 对象创建时会默认对成员变量进行初始化
- 当一个对象不被任何一个变量引用时，成为垃圾，等待垃圾回收机制。

2.3 类的封装性

在定义类时，将类的属性私有化，外界将不能随意访问，主要通过private关键字来修饰是私有属性

private 关键字

成员变量的定义还有许多可加的修饰符，用于声明成员变量的访问控制权限和使用限制。
访问控制权限的修饰符有public、protected、private等；
使用限制的修饰符有final、abstract、static、transient、volatile等。

private修饰的变量变为私有属性，只能在所在类中被访问，外界不能访问。

```
class Person{
    private String name;    //声明姓名私有属性
    //外界将不能访问name,设置name可通过类中方法

    public void setName(String str){
        name = str;
    }    //设置属性方法
    public String getName(){
        return name;
    }    //获取属性方法
}
```

2.4 类的构造方法

每一类都有自己的构造方法，或者称为类的构造器。构造方法是用来创建一个类的实例的。

- 构造方法是用类名作构造方法名；
- 构造方法具有参数和语句体，但没有返回类型的声明。（与成员变量区别）
- 构造方法不是类的成员方法，所以不能用对象调用它。
- 构造方法的调用是由new运算符实现；

```
class Person{
    private String name;
    public Person(String str){
        name = str;
    }
}
```

构造方法的重载

参数列表不同，调用不同的构造方法为不同属性赋值

Person p = new **Person**();

Person()是类自带的零参数构造方法，当在类中重新定义一个非空参数的构造方法时，系统将不会提供空参数构造方法。使用前需自行定义。

2.5 this关键字

this表示当前对象。

- 调用类中的属性
this将明确调用类的成员变量，不会与局部变量名起冲突
- 调用成员方法
- this调用构造方法
 - 在某个构造方法中使用 this(实参列表)；调用对应的另一个构造方法。
 - this () 必须位于首行，且只能调用一次
 - 必须留出口，否则无限循环。

2.6 static关键词

static表示静态的，可修饰成员变量、成员方法以及代码块。

- 一旦将成员设为static，数据或方法就不会同那个类的任何对象实例联系在一起
- 即使从未创建那个类的一个对象，仍能调用一个static方法，或访问一些static数据

2.6.1 静态变量

- 有static修饰的成员，称为**类成员**或静态成员；
 - 用类名直接访问：
 - 用对象名来访问，该类的所有对象都共享类成员变量。
- 无static修饰的成员，**实例成员**
 - 通过创建实例才能访问和使用
 - 不能用类名直接访问

static修饰变量只能为成员变量，不能为局部变量

2.6.2 静态方法

无需创建实例就可以通过类名调用

限制

- 它们仅能调用其他的static方法。
- 它们只能访问static数据。

- 它们不能以任何方式引用this或super

2.6.3 代码块

1) 普通代码块

在方法名后或方法体内

2) 构造代码块

- 直接定义在类中
- 实例化对象时被引用，且在构造方法之前被引用
- 可将每个构造方法中重复的部分拿出来放在这里

3) 静态代码块

- static修饰的代码块 最早执行的代码块
- 也是对象实例化之前被引用，只执行一次

3 垃圾回收

4 单例模式

设计模式之一——单例模式

一个类在程序运行期间有且仅有一个实例，并且自行实例化向整个系统提供这个实例。

```
class Single {  
    //静态私有对象  
    private static Single INSTANCE = new Single();  
    //私有化构造方法  
    private Single(){  
    }  
    //返回静态私有对象  
    public static Single getInstance(){  
        return INSTANCE;  
    }  
}  
  
public class TestSingle {  
    public static void main(String[] args) {  
        Single s1 = Single.getInstance();  
        Single s2 = Single.getInstance();  
        System.out.println(s1 == s2);  
    }  
}
```

5 内部类

三大共有特性：

- 内部类与外部类编译后生成的两个类**相互独立**
- **内部类是外部类一个成员**，内部类可访问外部类变量（包括私有），外部类不能直接访问内部类成员。
- 内部类可为静态，可用protected和 private修饰，外部类只能用public和默认访问权限。

5.1 成员内部类

- 内部类可以访问外部类中所有成员
- 外部类访问内部类
 - 外部类中访问内部类：需要在外部类中创建内部类对象，使用该对象来实现访问。
 - 外部类外，需要借助外部类对象创建内部类对象

```
外部类名.内部类名 引用变量 = new 外部类名().new 内部类名()
```

需要注意：内部类中不能定义静态变量、静态方法和静态内部类。

外部类被加载时，成员内部类是非静态的，Java编译器不会初始化内部类中静态成员，与编译原则相违背。

5.2 静态内部类

外部类对象与内部类对象之间无联系——静态内部类。

静态内部类可以**拥有实例成员和静态成员**

- 静态内部类可**直接访问外部类的静态成员**，访问实例成员需要通过外部类的对象。
- 外部类外访问静态内部类成员，不需要创建外部类对象，仅需创建内部类对象。

```
外部类名.内部类名 引用变量 = new 外部类名.内部类名()
```

5.3 方法内部类

方法内部类是指在成员方法中定义的类，与局部变量类似，仅能在方法内实例化，不可以外部实例化。作用域仅为方法内。

方法内部类可访问外部类成员。

5.4 匿名内部类

没有名称的内部类，创建匿名内部类时需要立即创建一个对象，该类定义随即失效。

```

class A{
    public void say(){
        System.out.println("sdcs");
    }
}
public class Test{
    public static void main(String[] args){
        A obj = new A(){
            system.out.println("匿名内部类");
        }
        obj.say();
    }
}

```

匿名内部类不能加访问修饰符

5.4.1 特殊内部类（待填写）

第五章 面向对象（下）

1 类的继承

1.1 概念

类继承另一个类，这个类除了创建自己的成员外，还能够继承或扩展另一个类的成员。（子类继承超类）

```

class 子类名 extends 超类名{
    属性和方法;
}

```

- Java支持单继承，不允许多重继承。一个子类只能有一个超类，但一个超类可以有多个子类。
- Java支持多层继承。

```

class A{}
class B extends A{}
class C extends B{}

```

- 虽然子类可以继承超类所有成员，但是因为超类中成员的访问控制，子类无法访问某些受限成员。
- 在超类中，由private修饰的访问权限的成员变量和方法，虽然被子类继承，但是子类不能访问

1.2 重写父类方法—方法覆盖

- 子类与超类的**方法名**、**返回值类型**和**参数列表**相同。
- 不同于方法重载

1.3 super关键词

补充：null、this、supper

每个类中都有的三个关键词

1. null

null表示变量的值为“空”，用于表示对象或数组还没有相应的**实例引用**。

例子，在参数传递的过程中 drawPoint (null) 该方法在定义时有参数

2. this

- 表示对类的实例访问，它也表示了对象对该实例引用访问。
- 在类中可以来指向成员变量，以示区别于非成员变量；
- 在构造器中，使用this()形式对另一个构造器的调用；
- 在类的创建中，需要表示对自身的实例访问时，用this表示

1) 访问超类变量（被覆盖）

2) 访问超类方法（被覆盖）

3) 调用父类构造方法

实例化子类对象时，首先会调用父类的**不含参**的构造方法（默认或编写），然后调用子类的。

子类不能继承超类的构造器，但能在在构造器中通过super()调用超类的构造器；

- super（参数）直接调用父类构造方法。

必须放在子类构造方法第一句

2 final 关键字

final可用来修饰类、方法和变量。表示最终的意思

2.1 修饰类

成为最终类，不能再被其他的类继承

2.2 修饰方法

最终方法——子类不能重写此方法

2.3 修饰变量

final修饰的变量，称为常量，只能被赋值一次。

空白final——先声明、等到使用前再赋值完成初始化。

- 空白final 具有最大的灵活性：
 - 位于类内部的一个final 字段现在对每个对象都可以有所不同，同时依然保持其“不变”的本质

3 抽象类

抽象方法：没有方法体的方法，由abstract修饰

抽象类：拥有**抽象方法**，且用abstract关键字声明。

3.1 使用原则

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abstract类。

3.2 使用限制

1. 抽象类中有构造方法。

抽象类中不只有抽象方法，还有一些属性，子类对象实例化的时候，依然满足先执行父类构造，再执行子类构造的顺序。

2. 构造方法、类方法(static)、私有方法(private)不可作为抽象，但抽象类中可以有类方法，并被全局调用。

4 Object类

当一个类被定义后，如果没有指定继承的父类，那么默认父类就是 Object 类。

Object中有很多方法

4.1 equals() 方法

两种比较方法，分别是 == 运算符和 equals() 方法，== 运算符是比较两个引用变量是否指向同一个实例，equals() 方法是比较两个对象的内容是否相等，通常字符串的比较只是关心内容是否相等。

其使用格式如下：

```
boolean result = obj.equals(Object o);
```

其中，obj 表示要进行比较的一个对象，o 表示另一个对象。

5 接口

接口是由**全局常量和公共的抽象方法**所组成。接口是**解决Java无法使用多继承的一种手段**，但是接口在实际中更多的作用是**制定标准的**。

5.1 特点

- 接口指明了一个类必须要做什么和不能做什么，相当于类的蓝图。
- 抽象方法只能存在于抽象类或者接口中，但抽象类中却能存在非抽象方法，即有方法体的方法。接口是百分之百的抽象类。
- 抽象类内部可能包含非final的变量，但是在接口中存在的变量一定是final，public,static的。

5.2 重点

- 接口**不能实例化**
- 一个类可以实现不止一个接口。
- 一个接口可以**继承**于另一个接口，或者另一些接口，**接口也可以继承，并且可以多继承**。
- 类实现某个接口，需要实现**所有方法**

- 接口中所有的方法都是抽象的和public的，所有的属性都是public,static,final的。
- 接口用来弥补类无法实现多继承的局限。
- 接口也可以用来实现解耦。

```
interface 接口名{
    全局常量声明;
    抽象方法声明;
}
// 全局常量声明时可省略public static final
// 方法声明时可省略 public abstract

class A implements 接口名{

}
```

利用**implements**实现多个接口

```
interface Person{
    void say();
}
interface Parent{
    void work();
}

class Child implements Parent,Person{
    public void work(){

    }
    public void say(){

    }
}
```

6 多态性*

Java的多态性体现在两个方面

1. 编译时的多态——方法的重载

系统在编译程序时，面对两个同名的方法，根据参数列表的不同来区分，编译出来是两个方法。

例如，在以下两个同名方法在编译过程中会当作两个方法来编译。

```
int max(int a, int c, int b);
int max(int a, int b);
```

2. 运行时的多态——父类对象引用子类实例

当一个父类对象引用子类的实例时：

- 该对象仍然只能够调用超类中定义的方法和变量
- 对于覆盖或继承的方法，Java运行时系统根据调用该方法实例的类型来决定选择哪个方法调用。
- 对子类的一个实例，如果子类覆盖了超类的方法，则运行时系统调用子类的方法。
- 如果子类继承了超类的方法(未覆盖)，则运行时系统调用超类的方法。


```

class Person {
    private String name;
    public String getName() {
        System.out.println("i am super");
        return name;
    }

    public void setName(String name) {
        System.out.println("i am super");
        this.name = name;
    }
    public void say(){
        System.out.println("super say");
    }
}

class Man extends Person{
    private String name;
    public String getName() {
        System.out.println("i am man");
        return name;
    }

    public void setName(String name) {
        System.out.println("i am man");
        this.name = name;
    }
    public void drink(){
        System.out.println("men drink wine");
    }
}

```

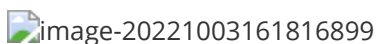
main 函数中

```

public class hello {
    public static void main(String[] args) {
        Person p1 = new Person();
        p1.setName("father");
        String p1_name = p1.getName();
        System.out.println(p1_name);
        p1.say();
        System.out.println("-----");
        Person p2 = new Man();
        p2.setName("son");
        String p2_name = p2.getName();
        System.out.println(p2_name);
        p2.say();
        //p2.drink(); 如果加上会在这一行报错
    }
}

```

结果：



其中，p1引用父类Person对象，作为对比组。p2为父类对象引用子类man。

- p2能够使用父类中被继承未被重写的方法 如 say

- 父类中被重写的方法则使用子类的，如getName, setName
- 不能使用父类中没有、子类中才有的方法，如drink。如果使用会报错。

第六章 包、访问控制

6.1 包的含义与使用

在Java语言中，对象以类的形式体现，通过把类存放在包中，实现类的反复调用，包实际上是一组类组成的集合，也称之为类库。

- 对类进行管理
 - 不同包里有相同的类不会发生冲突
 - 相同功能的类放在同一个包里
- 规定了类的使用范围
 - 同一个包里的类可以相互访问，不同包里的类不能直接相互访问

包的使用主要涉及到两个语句

6.1.1 package语句

package语句作为Java源文件第一条语句，指明该文件中定义的类所在的包，若无该语句则称为无名包。

```
package pkgName1[.pkgName2[.pkgName3...]]
```

pkgName n 表示包的目录层次，对应于系统文件的目录结构

6.1.2 import语句

允许用户使用Java提供或用户已创建的类。

- 两种形式：
 - 直接指明所要引入的类。import src.Point;
 - 使用“星号”引入语句，指明引入包中多个类。import src.*;
- 没有引入语句，使用时需显示其包
 - src.Point p

6.2 访问控制

对类的成员访问的四个范围：

- private 同一类中
- protected 同一包中
- 缺省 (default) 不同的包中的子类
- public 不同包中的非子类。

四种修饰符

- public: 可访问性最大修饰符，由public修饰的成员，则可以被任何范围中所访问。
- protected: 允许类中、子类(包括在或不在同一包中)和它所在包中的类所访问。
- 缺省: 可以被类自身和同一个包中的类访问。

- private：限制最强的修饰符。私有成员只能在它自身的类中访问。

第七章

第八章 多线程

8.1 多线程机制

线程是从一个大进程里分出的小的、独立的进程。

- 作为基本的执行单元，线程的划分比进程小，因此，支持多线程的系统要比只支持多进程的系统并发程度高。
- 进程把内存空间作为自己的资源之一，每个进程均有自己的内存单元。线程却共享内存单元，通过共享的内存空间来交换信息，从而有利于提高执行效率

Java中线程的组成：

- 虚拟的CPU，封装在Java.lang.Thread类中。
- CPU所执行的代码，传递给Thread类。
- CPU所处理的数据，传递给Thread类。

8.2 多线程实现方法

8.2.1 生成Thread子类

1. 生成Thread类的子类。 `class MyThread extends Thread`
2. 在子类中覆盖run()方法。 `public void run()`
3. 生成子类的对象，并且调用start()方法启动新线程。 `MyThread thread = new MyThread(); thread.start();`

8.2.2 生成一个实现Runnable接口

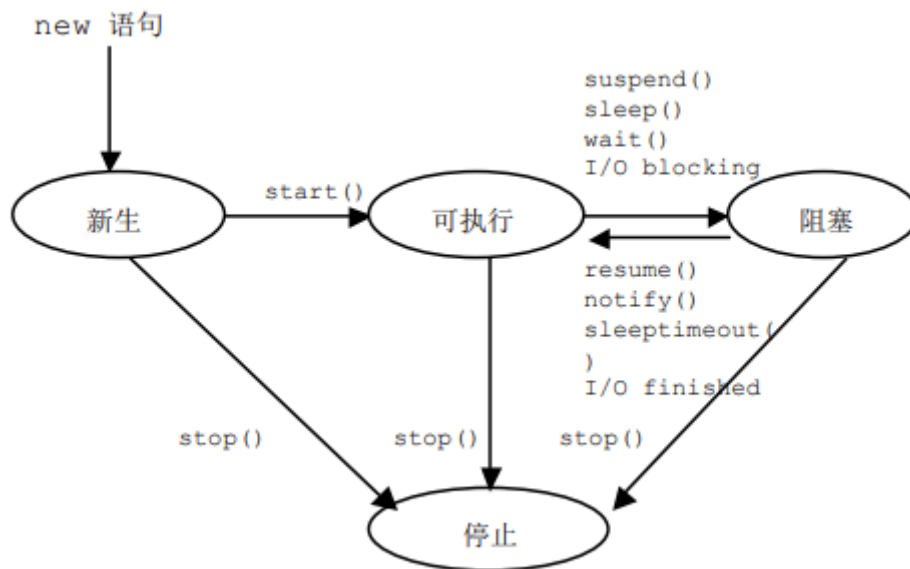
1. 程序中某个类声明实现Runnable接口，并且在这个类中实现run()方法。
2. 生成这个类的对象。
3. 用Thread(Runnable target)构造器生成Thread对象，其中target是声明实现了Runnable接口的对象，并且用start()方法启动线程。

8.3 线程状态

在Thread类中定义了三个优先级常量：MIN_PRIORITY, MAX_PRIORITY和NORM_PRIORITY，其值分别为1, 10, 5。

- 如果应用程序没有为线程分配优先级，则Java系统为其赋值为NORM_PRIORITY。
- 可以通过Thread类的setPriority(int a)方法来修改系统自动设置的线程优先级

控制线程方法



- `start()`: 用于调用`run()`方法使线程开始执行。
- `stop()`: 立即停止线程执行，其内部状态清零，放弃占用资源。
- `wait()`: 使线程处于等待状态。线程等待某个条件调用`wait()`方法。
- `notify()`: 使线程脱离阻塞状态。在条件变量所在的对象中调用`notify()`方法即可使线程脱离阻塞状态。
- `sleep()`: 调整线程执行时间，参数指定睡眠时间。
- `yield()`: 暂停调度线程并将其放在等待队列末尾，等待下一轮执行，使同优先级的其它线程有机会执行

8.4 线程同步

同步锁:

方法同步: `synchronized` 方法名

代码同步: `synchronized(this){ }`

第九章 输入输出流

9.1 I/O基本原理

流: 指在计算机的输入与输出之间运动的数据序列。

流序列中的数据既可以是未经加工的原始的二进制数据，也可以是经一定编码处理后符合某种格式规定的特定数据。

分类:

1. 输入输出流
 - 根据数据流向
2. 字节、字符流
 - 字节流: 每次处理二进制数据
 - 字符流: 每次处理一个字符

在最底层，所有的输入/输出都是字节形式的。基于字符的流只为处理字符提供方便有效的方法。

3. 节点流、过滤器

- 节点流：直接从指定的位置（如磁盘文件或内存区域）读或写。
- 过滤器：非字节流，往往是以其它输入流作为它的输入源，经过过滤或处理后再以新的输入流的形式提供给用户。

9.2 文件以及文件I/O

9.2.1 File类

- 文件路径和属性
 - `getPath()`和`getAbsolutePath()`方法返回File对象的路径和绝对路径。
 - `getName()`方法返回File对象的文件名或目录名。
 - `getParent()`返回File对象的父目录。
- 表示文件的属性或状态：
 - `canWrite()`, `canRead()`, `isDirectory()`, `isAbsolute()`, `exists()`, `isFile()`都返回 `boolean`型数据，分别表示文件是否写保护，是否读保护，是目录还是文件，是否使用绝对路径，是否存在
- 创建目录和删除文件
 - `mkdir()`和`mkdirs()`用于创建目录。创建目录的位置完全取决于File对象的路径。
 - `delete()`用于删除文件或目录，删除目录时，应该保证所删目录是一个空目录，否则删除操作失败。
- 文件更名
 - `renameTo()`方法不但可以给文件更名，而且可以给目录更名。
 - `equals()`判断两个File对象是否相等，程序用它来判断用户给定的原文件名和新文件名是否相等，如果相等则不能进行更名操作。
- 目录清单：
 - `list()`方法产生目录清单，它只返回指定目录中包含的文件名或子目录名，没有文件长度、修改时间、文件属性等信息。
 - `lastModified()`返回文件最后一次被修改的时间，其值是相对于 1970年1月1日的时间毫秒数，为便于阅读，必须变成 `java.util.Date`对象。

9.2.3 RandomAccessFile类

`RandomAccessFile`类和输入输出流类具有读写文件的功能。它有两个构造器：

- `RandomAccessFile(String name, String mode)` ▪
- `RandomAccessFile(File file, String mode)` ▪

其中：`name` 是一个String对象，表示被访问的文件名。`file` 是一个File对象，表示被访问的文件。
`Mode` 用字符串表示被访问文件的读写模式：“r”表示文件以只读方式打开，“rw”表示文件以读写方式打开（没有该文件则创建、有则覆盖）。

第十章

第十一章
