|  |  |
| --- | --- |
| **目录** |  |
| **1** | **目录** |
| **2** | **堆栈内存** |
| **3** | **二维数组** |
| **4** | **函数（方法）** |
| **5** | **设计模式** |
| **6** | **类和对象** |
| **7** | **匿名对象** |
| **8** | **构造函数** |
| **9** | **Static** |
| **10** | **主函数** |
| **11** | **封装** |
| **12** | **继承** |
| **13** | **final** |
| **14** | **抽象类** |
| **15** | **模板设计模式** |
| **16** | **接口** |
| **17** | **多态** |
| **18** | **object** |
| **19** | **内部类** |
| **20** | **异常** |
| **21** | **多线程概述** |
| **22** | **多线程的创建方式1** |
| **23** | **多线程的创建方式2** |
| **24** | **多线程的安全问题** |
| **25** | **线程间的通讯** |
| **26** | **停止线程** |
| **27** | **集合框架** |
| **28** | **泛型** |
| **29** | **File类** |
| **30** | **I/O输入输出流** |
| **31** | **装饰设计模式** |
| **32** | **GUI程序** |

**堆栈内存**

**栈（操作系统）：由操作系统自动分配释放 ，存放函数的[参数值](https://baike.baidu.com/item/%E5%8F%82%E6%95%B0%E5%80%BC" \t "https://baike.baidu.com/item/%E5%A0%86%E6%A0%88/_blank)，[局部变量](https://baike.baidu.com/item/%E5%B1%80%E9%83%A8%E5%8F%98%E9%87%8F" \t "https://baike.baidu.com/item/%E5%A0%86%E6%A0%88/_blank)的值等。其操作方式类似于数据结构中的栈。**

**栈使用的是[一级缓存](https://baike.baidu.com/item/%E4%B8%80%E7%BA%A7%E7%BC%93%E5%AD%98" \t "https://baike.baidu.com/item/%E5%A0%86%E6%A0%88/_blank)， 他们通常都是被调用时处于存储空间中，调用完毕立即释放。**

**堆（操作系统）： 一般由程序员分配释放， 若程序员不释放，程序结束时可能由OS回收，分配方式倒是类似于链表。**

**堆则是存放在[二级缓存](https://baike.baidu.com/item/%E4%BA%8C%E7%BA%A7%E7%BC%93%E5%AD%98" \t "https://baike.baidu.com/item/%E5%A0%86%E6%A0%88/_blank)中，生命周期由虚拟机的垃圾回收算法来决定（并不是一旦成为孤儿对象就能被回收）。所以调用这些对象的速度要相对来得低一些。**

**二维数组**

**实例解说：**

**Int [][]arr=new[3][4];**

**定义了一个名为arr的二维数组，二维数组中有三个一维数组，每个一维数组中可以存放四个元素；**

**书写方式：**

1. **int [][]arr=new [x][x]; 直接定义一维数组的个数以及所有一维数组的长度**
2. **int[][]arr={{},{},{},...}; 边声明边赋值**
3. **int[][]arr=new arr[x][];**

**Arr[0]=new int [y]; ..... 给了固定的一维数组个数，但每个一维数组的长度不是固定的**

1. **System.out.println(arr.length); 打印二维数组的长度，即一维数组的个数**

**5、System.out.println(arr[x].length); 打印二维数组中的某个一维数组的长度**

**函数**

**概述：**

**函数就是方法，函数是定义在类中的具有特定功能的独立小程序**

**函数格式：**

**修饰符 返回类型 名称(形式参数列表...)**

**{**

**执行语句;**

**Return 返回值; (如果返回类型为void 则不用返回)**

**}**

**形式参数：是一个变量，用于存储调用函数是传递的实际参数**

**实际参数：传递给实际参数具体的值**

**如何定义一个函数？**

1. **既然函数是一个具有特定功能的小程序，那么就要先明确函数的运算结果是什么 (这是在明确该功能的返回值类型);**
2. **明确在定义功能的过程中是否需要有位置内容参与运算 (这是在明确函数的参数列表);**

**函数重载：**

**概念：在同一个类中，允许一个以上的同名函数，只要参数列表不同即可，这样的函数称为重载；**

**特点：与返回类型无关，只看参数列表**

**好处：方便了阅读，优化了程序设计**

**什么时候使用重载？**

**当定义的功能相同，但参加运算的未知内容不同时可以使用重载，虚拟机会跟据不同的参数区分多个同名方法。**

**设计模式：解决某一类问题最行之有效的方法；**

**Java中有二十三中设计模式，单例设计就是其中一种，设计模式偏重思想；**

**单例设计模式：解决一个类中只能有一个对象。**

**思路：**

**想要保证对象唯一：**

1. **为了避免其他程序过多建立该类对象。先禁止其他程序建立该类对象**
2. **为了其他程序可以访问到该类对象，可以在该类中自定义一个对象**
3. **为了方便其他程序对该类的访问，可以对外提供访问方式**

**用代码体现：**

1. **私有构造函数**
2. **在类中创建一个本类对象，私有并静态**
3. **提供一个静态方法获取到该对象**

**例：**

**写法一：先初始化对象**

**称为：饿汉式 开发时比较常用**

**Class student**

**{**

**Private name;**

**Private student(){}**

**Private static student s=new student();**

**Public static student get()**

**{**

**Return s;**

**}**

**}**

**写法二：调用方法时在初始化对象,也叫作对象的延时加载。**

**称为：懒汉式**

**Class student**

**{**

**private name;**

**Private student(){}**

**Private static student s=null;**

**Public student get()**

**{**

**If(s==null) 此部分在多线程执行过程中可能会出错**

**{**

**S=new student();**

**Return s;**

**}**

**}**

**}**

**解决方法：1**

**Class student**

**{**

**private name;**

**Private student(){}**

**Private static student s=null;**

**Public synchronized student get()**

**{**

**If(s==null)**

**{**

**S=new student();**

**Return s;**

**}**

**}**

**}**

**解决方法：2**

**Class student**

**{**

**private name;**

**Private student(){}**

**Private static student s=null;**

**Public student get()**

**{**

**if(s==null) { 此方法可以提高效率**

**synchronized (){**

**If(s==null)**

**{**

**S=new student();**

**}**

**}**

**}**

**Return s;**

**}**

**}**

**效果，无论在其他类创建多少个对象，操作的都是在该类中自定义的对象，即一个对象的多个引用；**

**类和对象**

**面向对象理解：**

1. **面向对象是相对面向过程而言**
2. **面向对象和面向过程都是一种思想**
3. **面向过程强调的是功能行为**
4. **面向对象是将功能 封装进对象，强调了具备功能的对象**
5. **面向对象是基于面向过程的**

**面向对象三个特征：封装 继承 多态**

**以后开发：找对象使用，没有对象就创建一个对象使用**

**过程：找对象，建立对象，使用对象，维护对象关系**

**类和对象的关系**

**1、类是对生活中事物的描述，对象是生活中实实在在存在的个体**

**匿名对象**

**New xx();没有名字的对象为匿名对象;**

**使用方式一：当对对象的方法只调用一次时，可以使用匿名对象，这样比较简便**

**情形：**

**Public class haha{**

**Public void lala(){**

**执行语句**

**}**

**}**

**Class hehe{**

**Public static void main(string []args){**

**New haha().lala();**

**}**

**}**

**使用方式二：匿名对象可以作为实际参数进行传递**

**构造函数**

**特点：**

1. **函数名与类名相同**
2. **不用定义返回值类型**
3. **不可以有return语句**

**作用：给对象进行初始化**

**注意：**

1. **默认构造函数的特点**
2. **多个构造函数是以重载的方式存在的**

**对象一建立。就会调用与之对应的构造函数，当一个类中没有定义构造函数时，系统会自动给它加上一个空参数的构造函数**

**构造函数和一般函数的不同：**

1. **构造函数在对象建立时运行，一般函数在调用时运行**
2. **构造函数只在建立对象时运行一次，一般函数可以多次调用**

**什么时候定义构造函数？**

**分析事物时，该事物本身具备一些特性，可以定义在构造函数中**

**构造代码块**

**构造代码块和构造函数的区别：构造代码块是给所有对象进行统一初始化，构造函数是给对应的对象进行初始化、**

**构造代码块优先于构造函数执行**

**写法**

**Class haha{**

**{**

**System.out.println();**

**}**

**}**

**Static 静态**

**优先于对象存在**

**被所有对象所共享**

**特点：**

1. **用于修饰成员（成员变量，成员函数），不能修饰局部**
2. **当成员被修饰后，可以直接通过类名调用（类变量）**

**生命周期：**

**随着类的加载而加载，随着类的消失而消失，生命周期最长**

**类变量和实例变量的区别**

1. **存放位置**

**类变量随着类的加载而存在于方法区中**

**实例变量随着对象的建立而存在于堆内存中**

**什么时候使用静态呢？**

**静态可以修饰静态变量和静态函数**

**当对象中存在共享数据时，该数据可以用静态修饰**

**当功能内部没有访问到非静态的成员时，可以用静态修饰**

**注意事项：**

**静态方法只能访问静态成员，非静态可以访问静态**

**静态中不能出现this，super关键字，因为静态优先于对象存在**

**静态有利有弊：**

**利处：对对象的共享数据进行单独空间的存放，没有必要将其在每个对象中存储一份，节省内存。**

**弊端：生命周期过长，访问出现局限性**

**主函数**

**public static void main(string [] args){}**

**主函数：是一个特殊的函数，作为程序的入口，可以被jvm调用**

**public:代表该函数的访问权限是最大的**

**static: 代表着它随着类的加载就已经存在了**

**void:该函数的返回类型不是确定的**

**main:方法名，可以被jvm识别**

**string[]args:函数的参数列表**

**封装**

**封装：是指隐藏对象的属性和实现细节，仅对外提供公共访问方式**

**好处：**

1. **将变化隔离**
2. **便于使用**
3. **提高重用性**
4. **提高安全性**

**封装原则：**

**将不需要对外提供的内容都隐藏起来**

**把属性都隐藏起来，对外提供公共访问方式**

**例：**

**class haha{**

**private string name;//私有**

**Public void setName(name){**

**this.name=name;**

**}**

**public string getName(){**

**return name;**

**}**

**}**

**this关键字**

**this代表所在函数所属对象的引用。**

**也就是哪个对象调用this所在函数，this就代表哪个对象**

**继承**

**关键字：**

**extends 继承**

**Super 代表父类对象**

**Java 中只支持单继承，不支持多继承，因为多继承可能会带来安全隐患，当多个父类中定义了相同的功能，当功能内容不同时，子类对象不确定要运行哪一个，但是Java保留这种机制，用另一种形式体现，即多实现。**

**Java支持多层继承：即父继承爷，子继承父**

**好处：**

1. **提高了代码的复用性**
2. **让类与之间有了关系，有了这个关系，才有了多态的特性**

**注意：不要为了获取其他类的功能而继承，从而简化代码，类与类之间必须是有所属关系才可以继承，即is a 关系**

**子父类中的函数：**

**当子类和父类具有相同的方法时，子类调用方法，会运行子类方法，此现象为覆盖（即重写），提高了代码的扩展性**

**当子类继承父类，衍袭了父类的功能，子类虽然具备该功能，但功能内容不同，没有必要定义新功能，而是使用覆盖特殊，保留父类的功能定义，并重写该功能。**

**例如：**

**class fu{**

**Public void speak(){**

**System.out.println(“中文”);**

**}**

**}**

**class zi extends fu(){**

**Public void speak(){//重写父类方法**

**Super.speak();//引用父类功能**

**System.out.println(“英语”);**

**}**

**}**

**覆盖（重写）：**

1. **子类覆盖父类，必须保证子类权限大于等于父类权限，否则编译失败**
2. **静态只能覆盖静态**

**子父类中的构造函数**

**不存在覆盖**

**在对子类对象进行初始化时，父类的构造函数也会运行，那是应为子类所有的构造函数第一行都有一条隐式语句super();**

**例如：**

**class fu{**

**Fu(){**

**System.out.println();**

**}**

**}**

**class zi{**

**Fu(){**

**Super.fu(); //隐式语句，根据需要，有时需要自己手动定义**

**System.out.println();**

**}**

**}**

**子类对象一定要访问父类的构造函数**

**因为父类中的数据子类可以直接获取，所以子类对象在建立时，先要查看父类是如何对这些数据进行初始化的**

**final 最终 修饰符**

**1、可以修饰 变量、函数、类**

**2、被final修饰的类不能被继承**

**3、被final修饰的方法不能被复写**

**4、被final修饰的变量为常量，只能赋值一次，既可以修饰成员变量，又可以修饰局部变量**

**书写规范，被final修饰的变量每个字母都大写，如果有多个单词组成，用 \_ 连接。**

**抽象类**

**当多个类中出现相同功能，但功能主体不同，也可以进行向上抽取，这时只抽取功能定义，不抽取功能主体**

**例如：**

**abstract class student{**

**public abstract void study();//没有方法体，以分号结束**

**}**

**Class baseStudent extends student{**

**Public void study(){**

**}**

**}**

**Class chongciStudent extends student{**

**Public void study(){**

**}**

**}**

**抽象类特点：**

1. **抽象方法一定要定义在抽象类中**
2. **抽象方法和抽象类都要使用关键字abstract进行修饰**
3. **抽象类不可以创建对象，因为抽象方法没有方法体，调用没有意义**
4. **抽象类中的方法要被使用，必须由子类重写，如果没有全部重写，那么子类还是一个抽象类**
5. **抽象类只能修饰类和方法**

**模板设计模式**

**当一段代码优化后就可以解决这一类的问题，这就是模板设计模式**

**什么是模板方法呢？**

**当定义功能时，功能的一部分时确定的，而有一部分不是确定的，并且确定的部分在使用不确定的部分，这时就将不确定的部分暴露出去，有子类完成。**

**例如：**

**获取一个程序运行的时间**

**abstract class a{**

**Public final getTime(){**

**Int strat=System.currentTimeMillis();**

**Haha(); 不确定的运行代码**

**Int end=System.currentTimeMillis();**

**System.out.println(“运行时间：”+(strat-end));**

**}**

**Public abstract void haha();**

**}**

**Class b extends{**

**Haha(){**

**执行代码;**

**}**

**}**

**接口**

**初期理解：**

**可以认为是一个特殊的抽象类，当抽象类中的方法全是抽象的，可以定义为接口**

**Class用于定义类，**

**Interface用于定义接口**

**接口中的成员都是public的，变量为静态常量，它有固定写法**

**例如：**

**Interface inter{**

**Public static final int a=4; 可以不写修饰符，系统会自动加上，但一般建议规范写法**

**Public void haha();**

**}**

**接口不可以创建对象，因为接口中有抽象方法，它需要被子类实现，而不是继承，子类对接口中的方法覆盖后才可以实例化，不然子类还是一个抽象类。**

**实现关键字：implement**

**一个类可以实现多个接口**

**例如**

**Class a implement 接口1，接口2...{**

**}**

**接口的特点：**

1. **接口是对外暴露的规则**
2. **接口是程序的功能扩展**
3. **接口可以用来多实现**
4. **接口之间可以继承，并且可以多继承**

**5、一个类可以实现多个接口**

**多态**

**可以理解为事物存在的多种体现形态**

**多态的体现**

**父类的引用指向了自己的子类对象**

**父类的引用也可以接受自己的子类对象**

**多态的前提**

**类与类之间必须存在关系，要么继承，要么实现**

**存在覆盖**

**多态的好处**

**多态的出现大大提高了程序的扩展性**

**多态的弊端**

**只能使用父类的引用访问父类中的成员**

**多态的应用**

**例子：**

**abstract class Animal{**

**public abstract void eat(); //抽象方法**

**}**

**class Cat extends Animal{**

**public void eat(){ //重写父类方法**

**代码块;**

**}**

**public void CatchMouse(){ //子类特有的方法**

**代码块;**

**}**

**}**

**class Dog extends Animal{**

**public void eat(){ //重写父类方法**

**代码块;**

**}**

**public void kanHome(){ //子类特有的方法**

**代码块;**

**}**

**}**

**class haha{**

**public void function(Animal a){**

**a.eat();**

**}**

**public void main(string [] args){**

**Animal cat=new Cat();//父类的引用指向子类对象，类型提升，也称向上转型。**

**}**

**}**

**如果想要调用子类特有的方法，该怎么办？**

**强制将父类的引用转成子类类型，即向下转型。**

**例：**

**Animal a=new Cat();**

**Cat c=(Cat)a;**

**c.CatchMouse();**

**错误示例：**

**Animal a=new Animal();**

**Cat c=(Cat)a; //将父类对象转化为子类类型，不可以出现这种情况**

**关键字 instanceof**

**用于判断父类的引用指向哪一个子类对象,一般在子类对象有限的情况下可以使用此关键字，如果子类对象过多，会降低程序性能。**

**例：**

**public void function(Animal a){**

**a.eat();**

**if(a instanceof Cat){**

**Cat c=(Cat)a;**

**}else if(a instanceof Dog){**

**Dog c=(Dog)a;**

**}**

**}**

**object 类**

**所有对象的直接父类或间接父类，简称上帝。**

**该类中定义的是所有的类都具备的功能**

**内部类**

**将一个类定义在另一个类里边，对里边的那个类就称为内部类(内置类、嵌套类)**

**访问特点**

1. **内部类可以直接访问外部类的成员，包括私有成员,之所以可以直接访问外部类，是因为内部类中持有一个外部类的引用 格式：外部类名.this**
2. **外部类要访问内部类中的成员，必须要建立内部类的对象**

**示例：**

**class Out{ //外部类**

**int x=1;**

**public void print(){**

**Inner p=new Inner();//创建内部类对象，用于访问内部类**

**p.show();**

**}**

**class Inner{ //内部类**

**int x=2;**

**public void show(){**

**int x=3;**

**System.out.println(3); //打印结果为3**

**System.out.println(this.x); //打印结果为2**

**System.out.println(Out.this.x); //打印结果为1**

**}**

**}**

**}**

**class User{**

**public void main (string [] args){**

1. **创建外部类对象，调用外部类访问内部类的方法**

**Out o=new Out();**

**o.print();**

**2、直接访问内部类**

**Out.Inner p=new Out.new Inner();**

**p.show();**

**}**

**}**

**注意：**

**当内部类中有了静态成员，该内部类必须是静态的。**

**当外部类的静态成员访问内部类时，该内部类必须是静态的**

**解释：**

1.内部类是外部类的一个成员，可以等同于外部类的一个属性或者方法。  
2.非静态的内部类是外部类对象的一个成员，必须先有外部类的对象才有这个内部类，它是属于对象的。  
3.内部类的静态成员不属于对象的，也就是类在加载的时候就存在了，那么这就发生冲突了，也就是说外部类还没有new对象前，这个内部类是不存在的，而这个内部类的静态成员就存在了，这就好比没有鸡就先有鸡蛋一样。

**什么时候定义内部类？**

**当描述事物时，事物内部还有事物，该事物用内部类描述，因为内部事物在使用外部事物的内容**

**内部类可以定义在外部类的成员部分，也可也局部。**

**例如：**

**class a{**

**public void show(){**

**class b{**

**public void print(){**

**}**

**}**

**}**

**}**

**当内部类定义在外部类局部中时，还可以直接访问外部成员，但不能访问局部成员，除非是被final修饰的。并且不能为静态，包括方法**

**匿名内部类**

**其实就是内部类的简写**

**定义内部类的前提：**

**该内部类必须是继承一个类或者实现接口**

**例如：**

**----》简化前代码：**

**abstract class a{**

**public void show();**

**}**

**class b{**

**int x=1;**

**class c extends a{**

**public void show(){**

**System.out.println(x);**

**}**

**}**

**public void function(){**

**new c().show(); //调用内部类中的show()方法**

**}**

**}**

**----》简化后**

**abstract class A{**

**public void show();**

**}**

**class b{**

**int x=1;**

**new A(){**

**public void show(){**

**System.out.println(x) //此部分相当于子类对象**

**}**

**}.show();**

**}**

**或者 A a=new A(){**

**public void show(){**

**System.out.println(x);**

**}**

**}**

**a.show();**

**}**

**异常**

**异常就是程序运行时出现的不正常情况**

**异常由来：对不正常情况封装成对象。**

**问题划分**

1. **对于严重的问题，Java通过Error类进行描述。一般不写针对性的代码进行处理**
2. **对于非严重的问题，Java通过Exception类进行描述。一般有针对性的代码进行处理**

**异常的处理：**

**try{**

**可能会发生异常的代码**

**}catch(Exception){ //catch块可以有多个**

**对异常的处理**

**}finally{**

**一定会执行的语句**

**}**

**获取异常信息**

**string getMessage();**

**关键字：throws 在函数上声明异常，让调用者进行处理，不处理则编译失败**

**对多异常的处理：**

**建议声明更具体的异常，这样处理的可以更具体，声明几个异常，就有几个catch块**

**自定义异常**

**程序运行时可能会发生一些特有的问题，而Java中没有对这些问题进行封装，这时候需要我们自定义异常**

**注意：**

**1、当在函数内部出现了异常，要么给出对应的处理方式，要么抛出，给调用者进行处理**

**2、一般情况下，在函数内出现异常，需要在函数上进行处理**

**如何自定义异常信息？**

1. **需要继承Exception类**
2. **父类中已经把异常信息的操作完成了，只需要使用super语句调用即可**

**例如：**

**//负数异常类**

**class Fushu extends Exception{**

**/\***

**private string message;**

**public Fushu(message){**

**this.message=message;**

**}**

**public string m(){**

**return m;**

**}**

**原本类型**

**\*/**

**Fushu(String message){**

**super(message);**

**}**

**}**

**class haha{**

**public void cacl(int a,int b)throws Fushu{//声明异常**

**if(a<0){**

**throw new Fushu("出现了负数");//抛出一个异常对象**

**}**

**}**

**public static void main(String []args){**

**try{//捕捉异常**

**new haha().cacl(-1,2);**

**}catch(Fushu e){**

**//处理异常**

**System.*out*.println(e.toString());**

**}finally{**

**System.*out*.println("程序结束");**

**}**

**}**

**finally中一般定义的是关闭资源的方法，因为程序最终结束时需要释放资源。**

**finally不被执行的唯一情况，System.exit(0);退出虚拟机**

**继承Exception的原因：**

**异常类中有一个体系，异常类和异常对象都被抛出，具备可抛性，是Throwable这个体系中独有的特点**

**throws和throw的区别**

1. **throws使用在函数使用在函数上，throw使用在函数内部**
2. **throws后边跟异常类，可跟多个，用逗号隔开，throw后边跟异常对象。**

**RuntimeException 运行时异常**

**是一个特殊的子类异常**

**当在函数内抛出异常时，函数上不用声明**

**在函数上声明，调用者可以不用进行处理**

**之所以不用在函数上声明，是因为不需要调用者处理，当该异常发生时，希望程序停掉，因为在运行时出现了无法继续运算的情况，希望程序停掉后，对代码进行处理。**

**自定义异常也可以继承RuntimeException异常，具备它的特性**

**多线程**

**进程：**

**狭义定义：进程是正在运行的程序的实例（an instance of a computer program that is being executed）。**

**广义定义：进程是一个具有一定独立功能的程序关于某个数据集合的一次运行活动。它是[操作系统](https://baike.baidu.com/item/%E6%93%8D%E4%BD%9C%E7%B3%BB%E7%BB%9F/192" \t "https://baike.baidu.com/item/%E8%BF%9B%E7%A8%8B/_blank)动态执行的[基本单元](https://baike.baidu.com/item/%E5%9F%BA%E6%9C%AC%E5%8D%95%E5%85%83" \t "https://baike.baidu.com/item/%E8%BF%9B%E7%A8%8B/_blank)，在传统的[操作系统](https://baike.baidu.com/item/%E6%93%8D%E4%BD%9C%E7%B3%BB%E7%BB%9F" \t "https://baike.baidu.com/item/%E8%BF%9B%E7%A8%8B/_blank)中，进程既是基本的[分配单元](https://baike.baidu.com/item/%E5%88%86%E9%85%8D%E5%8D%95%E5%85%83" \t "https://baike.baidu.com/item/%E8%BF%9B%E7%A8%8B/_blank)，也是基本的执行单元。**

**进程的概念主要有两点：**

**第一，进程是一个实体。**

**第二，进程是一个“执行中的程序”。**

**线程概述：**

**线程，有时被称为轻量级进程(Lightweight Process，LWP），是程序执行流的最小单元。**

**线程是进程中的一个实体，是被系统独立调度和分派的基本单位，线程自己不拥有系统资源，只拥有一点儿在运行中必不可少的资源，但它可与同属一个进程的其它线程共享进程所拥有的全部资源**

**一个线程可以创建和撤消另一个线程，同一进程中的多个线程之间可以并发执行。由于线程之间的相互制约，致使线程在运行中呈现出间断性。**

**线程也有[就绪](https://baike.baidu.com/item/%E5%B0%B1%E7%BB%AA/10932509" \t "https://baike.baidu.com/item/%E7%BA%BF%E7%A8%8B/_blank)、[阻塞](https://baike.baidu.com/item/%E9%98%BB%E5%A1%9E/9032404" \t "https://baike.baidu.com/item/%E7%BA%BF%E7%A8%8B/_blank)和[运行](https://baike.baidu.com/item/%E8%BF%90%E8%A1%8C/5480697" \t "https://baike.baidu.com/item/%E7%BA%BF%E7%A8%8B/_blank)三种基本状态。**

**就绪状态：是指线程具备运行的所有条件，逻辑上可以运行，在等待处理机；**

**运行状态：是指线程占有处理机正在运行；**

**阻塞状态：是指线程在等待一个事件（如某个信号量），逻辑上不可执行。**

**每一个程序都至少有一个线程，若程序只有一个线程，那就是程序本身。**

**Java vm启动的时候会有一个进程Java.exe**

**该进程中至少有一个线程负责Java程序的执行，而且这个线程存在于main方法中，该线程称为主线程**

**扩展：其实更细节说明jvm,jvm启动不知一个线程，还有负责垃圾回收机制的线程**

**如何在自定义的代码中自定义一个线程呢？**

**Java中已经提供了对线程这类事物的描述，就是Thread类**

**创建线程的第一种方式：继承Thread类**

**步骤：**

1. **继承Thread类。**
2. **复写run()方法**

**目的：将自定义的代码存储在run()方法中让线程运行**

1. **创建线程，调用线程的star方法**

**目的：启动线程，调用run()方法**

**例如：**

**class haha extends Thread{**

**public void run{**

**System.out.println(...);**

**}**

**}**

**class demo1{**

**public static void main(string []args ){**

**haha p=new haha(); //创建一个线程**

**p.star(); //启动线程**

**}**

**}**

**明确一点，在某一时刻，只能有一个程序运行(多核除外)，CPU执行到谁，谁就运行，在这当中，CPU在做着快速的切换，达到了看似一起运行的效果**

**多线程的特性：随机性，执行到谁，谁运行，运行时间CPU说的算。**

**每个线程都有自己默认的名称：Thread—编号编号从零开始**

**static Thread.currentThread(); 获取线程对象**

**getName(); 获取线程的名称**

**设置线程名称：**

1. **setName();**
2. **构造函数**

**线程的第二种创建方式：实现Runnable接口**

**步骤：**

1. **定义类实现Runnable接口**
2. **覆盖Runnable接口中的Run()方法**
3. **通过Thread类建立线程对象**
4. **将Runnable接口的子类对象作为实际参数传递给Thread类的构造函数**
5. **调用Thread对象的star()方法开启线程并调用Runnable子类的run()方法**

**补充：死锁状态**

**其实就是同步中嵌套同步，其中也可能出现和谐状态**

**例如：**

**class A implements Runnable{**

**@Override**

**public void run() {**

**while(true){**

**synchronized(MyLock.*A*){**

**System.*out*.println("I am A");**

**synchronized(MyLock.*B*){**

**System.*out*.println("I am B");**

**}**

**}**

**}**

**}**

**}**

**class B implements Runnable{**

**@Override**

**public void run(){**

**while(true){**

**synchronized(MyLock.*B*){**

**System.*out*.println("I am B");**

**synchronized(MyLock.*A*){**

**System.*out*.println("I am A");**

**}**

**}**

**}**

**}**

**}**

**class MyLock{**

**static Object *A*=new Object();**

**static Object *B*=new Object();**

**}**

**public class DealLock {**

**public static void main(String [] args){**

**A a=new A();**

**B b=new B();**

**Thread t1=new Thread(a);**

**Thread t2=new Thread(b);**

**t1.start();**

**t2.start();**

**}**

**}**

**注意：多线程存在安全问题;**

**原因：**

**当多条语句操作同一个线程的共享数据时，一个线程对多条语句只执行了一部分，另一个线程参与进来执行导致共享数据错误；**

**例如：**

**class X2 implements Runnable{**

**private int a=10; //总票数**

**public void run() {**

**while(true){**

**if (a>0) {**

**//冻结程序，让程序做一个短暂停留**

**try{Thread.*sleep*(10);}catch(Exception e){} System.*out*.println(Thread.*currentThread*().getName()+"---"+(a--));**

**}**

**}**

**}**

**}**

**public class XianCheng1 {**

**public static void main(String[] args) {**

**X2 x=new X2();//创建Runnable的子类对象**

**Thread t=new Thread(x); //创建线程 一号售票窗口**

**Thread t1=new Thread(x); //二号售票窗口**

**Thread t2=new Thread(x); //二号售票窗口**

**Thread t3=new Thread(x); //二号售票窗口**

**t.start();**

**t1.start();**

**t2.start();**

**t3.start();**

**}**

**}**

**该打印结果出现了-1，-2.属于错误数据；**

**解决方法：**

**对操作同一个线程共享数据的语句，在执行过程中不让其他的线程加入**

**Java对多线程的安全问题提供了专业的解决方法，同步代码块**

**synchronized(对象){**

**需要被同步的代码块**

**}**

**同步可以解决多线程的安全问题（个人理解：相当于等待空位的出现）**

**使用同步的前提：**

1. **必须保证有两个或两个以上的线程**
2. **必须是多个线程使用同一个锁**
3. **必须保证同步中只能有一个线程在运行**

**好处：解决了多线程安全问题；**

**弊端：多个线程需要判断锁，较为消耗资源**

**同步函数的锁是this**

**程序是否存在安全问题？如果有，如何解决？**

**如何找问题？**

1. **明确哪些代码是多线程运行代码**
2. **明确共享数据**
3. **明确多线程运行代码中哪些语句是操作共享数据的**

**如何解决问题？**

1. **同步代码块**
2. **同步函数**

**静态的同步函数的锁是什么？**

**因为静态中不允许有this，所以，锁不能是this**

**静态进内存时，内存中没有本类对象，但是，一定有该类对应的字节码对象**

**类名.Class 该对象的类型是Class**

**线程间的通讯**

**概念：多个线程操作同一个资源，但是操作动作不同**

**等待唤醒机制**

**wait();**

**notify();**

**notifyAll();**

**都使用在同步中，因为要对监视器（锁）进行操作，而只有同步中有锁，所以要使用在同步在**

**这些方法都定义在object类中，为什么？**

**因为这些方法在操作的过程中，都要标识它们所持有的锁，只有同一个锁上等待的线程可以被同一个锁上的notify()唤醒**

**也就是说等待和唤醒必须是同一个锁**

**而同步的锁可以是任意对象，所以定义在object类中**

**JDK1.5升级版新增Lock**

**将同步synchronized替换成了Lock**

**将object类中的wait(),notity(),notityAll(),替换成了Condition对象**

**该对象可以通过Lock锁获取**

**可以进行本方只唤醒对方的操作**

**private Lock lock=new ReentrantLock();//创建一个锁**

**//创建Condition对象替代objet类中的wait(),notity()等方法**

**private Condition condition\_p=lock.newCondition();**

**private Condition condition\_c=lock.newCondition();**

**condition\_p.await();//等待**

**condition\_p.signal();唤醒一个**

**condition\_p.signalAll();唤醒所有**

**停止线程**

**stop();方法已经过时，不可用**

**解决方法**

**可以让run()方法结束**

**开启多线程运行，运行代码通常都是在循环结构中**

**只要控制住循环，就可以结束run()方法，也就是线程结束**

**特殊情况：**

**当线程处于了冻结状态，即等待中，**

**就不会读取到让方法结束的标记，那么线程就不会结束**

**当没有指定的方式让冻结的线程恢复到运行状态，这时需要对冻结进行清除**

**强制让线程恢复到运行状态中，这样就可以操作标记，结束线程**

**Thread类提供了该方法interrupt();**

**集合框架**

**概述：**

1. **为什么出现集合类？**

**面向对象语言对事物的体现都是以对象的体现，所以为了方便对多个对象的操作，就对对象进行存储，集合就是存储对象最常用的一种方式**

1. **为什么会出现这么多集合容器呢？**

**因为每个容器的存储方式都不同，这种存储方式叫做：数据结构**

**集合框架大致体系结构**

**Collection接口**

TreeSet

HashSet

**Vector**

**LinkedList**

**ArrayList**

List

set

**Collection**

**|----List元素是有序的，元素可以重复，因为该集合体系中有索引**

**|---ArrayList:底层数据结构使用的是数组结构**

**特点：查询速度快，但是增删稍慢，线程不同步**

**|---LinkedList:底层使用的是链表数据结构**

**特点：增删快，查询稍慢**

**|---Vector：底层数据结构为数组结构，被ArrayList的出现替代了**

**特点：线程同步**

**|----Set元素是无序的（存入和取出的顺序不一定一致），元素不可以重复**

**|---HashSet:底层数据结构是哈希表**

**|---TreeSet:底层数据结构是二叉树**

**ArrayList集合**

**1、添加元素**

**add(Object obj);**

**2、删除元素**

**remove();移除指定元素，参数可以是元素本身，也可以是下标**

**clear();清空所有**

1. **判断**

**contains(obj);是否包含**

**isEmpty();是否为空**

**迭代器**

**其实就是元素取出的方式，实在内部实现的**

**在迭代过程中，不能使用集合的方法对元素进行操作，会引发异常，而使用迭代器的方法操作不会引发异常**

**通过对外提供获取方式（iterator()）进行使用**

**Iterator**

**Iterator i=arr.iterator();//该迭代器是个普通迭代器。内部方法较少，不能对集合进行添加修改操作**

**List集合特有的迭代器ListIterator ,是Iterator的子接口，具有对集合操作的一系列方法**

**Vector集合**

**枚举就是Vector特有的取出方式**

**枚举和迭代器其实是一样的，因为枚举的名称和方法都过长，所以被迭代器取代了**

**Vector v=new Vector();**

**Enumeration e=v.elements();**

**e.hasMoreElements();判断是否还有元素**

**e.hasNext();一次读取一条**

**LinkedList集合**

**特有方法：**

**addFirst();**

**addLast();**

**getFirst();**

**getLast();**

**RemoveFirst();可以获取元素，然后删除**

**RemoveLast();**

**获取元素时，如果集合中没有元素，会发生NoSuchElementExecption**

**JDK1.6版本之后出现了替代方法**

**offerFirst();**

**offerLast();**

**peekFirst();**

**peekLast();**

**获取元素时，如果集合中没有元素，则返回null**

**HashSet集合**

**是如何保证数据的唯一的呢？**

**通过元素的两个方法，hashCode和equals来完成**

**如果元素的hashCode值相同，才会判断equals是否为true**

**如果hashCode的值不同，则不会调用元素的equals.**

**TreeSet集合**

**可以对Set集合中的元素进行自然排序**

**保证数据唯一性的依据是compareTo();返回0代表相等**

**由于TreeSet会对元素进行自然排序，所以存储的元素必须具备可比较性**

**实现Comparable接口，会让其具备比较性**

**重写compareTo()时，要考虑在主要条件相等时，比较次要条件**

**排序方式1：**

**让元素具备比较性，实现Comparable接口覆盖compareTo();方法**

**排序方式2：**

**让集合在初始化时，具备比较性**

**即：创建一个类实现Comparator接口，重写compare()方法，然后将该类作为参数传递给TreeSet集合**

**注意：两种比较方式都存在时，以方式2为主**

**字符串对象本身具备比较性，因为它实现了Comparable**

**Integer对象也具备比较性**

**Map集合**

**|---HashTable:底层数据结构哈希表数据结构,不允许存入null键null值，该集合是线程同步的，效率比较低**

**|---HashMap:底层数据结构是哈希表数据结构，允许使用null键和null值，该集合不是同步的，效率高**

**|---TreeMap:底层数据结构是二叉树结构，可以用于给Map集合中的键进行排序**

**以键值对的形式存储数据**

**Map集合使用put()方法向集合中添加元素，如果已有相同的键，那么它返回原来键的值，并将原来键的值覆盖掉**

**Map集合的两种取出方式：**

**1：通过集合的keySet();方法，获取Map集合的所有键，并返回set集合，通过键值，可以获取对应的值**

**2：将集合的映射关系取出，并存储到set集合中**

**Set<Map.Entry<string,string>> entry=map.entrySet();**

**可以直接获取键和值**

**集合框架的工具类**

**当中都是静态方法**

**Collections**

**常用方法：**

**Sort();,可以有一个参数或者两个参数，，第一个为要排序的集合，第二个参数为Comparable**

**Max();集合中最大值**

**binarySerach();二分查找法，可以找到某一个新元素的插入位置，并且保证原有的顺序。**

**Collections.reverserOrder();//实现反转元素，作为参数传给集合**

**Collections.Synlist();将集合变为同步的**

**Collections.shuffle();将集合中的内容重新进行一个随机排序。**

**Arrays**

**常用方法：**

**atList();将数组变成集合。**

**方法好处：**

**可以用集合的思想操作数组**

**注意：将数组变为集合，不可以使用集合的增删方法，因为数组的长度是固定的。**

**泛型**

**JDK1.5版本后出现的新特性，用于解决安全问题，是一个安全机制**

**好处：**

**\*将运行时期出现的ClassCastExecption转移到了编译时期，方便程序员解决问题，让运行事情减少，安全**

**\*避免了拆箱装箱的频繁操作**

**经常应用在集合中**

**什么时候使用泛型类？**

**当类中要操作的引用数据类型不确定的时候，早期定义object来进行扩展，现在定义泛型来扩展**

**泛型类上定义的泛型，在整个类中有效，如果被方法使用，那么泛型类的对象明确要操作的具体类型后，所有要操作类型的方法就已经固定了**

**为了让不同的方法操作不同的类型，而且类型还不确定，那么可以将泛型定义在方法上**

**例如：**

**public <T> void sayHi(T t){}**

**特殊之处：**

**静态方法不可以访问类上的泛型，要想定义，那么只能在方法上定义，例如**

**public static <T> void sayHi(T t){}**

**泛型限定**

**？通配符，可以理解为占位符**

**？ extends E:可以接受E类型或者其子类，上限**

**？ super E:可以接受E类型或者其父类型，下限**

**File类**

**概述：**

**用来将文件或者文件夹封装成对象**

**方便对文件与文件夹进行操作**

**File对象可以作为参数传递给流的构造函数**

**File 类的常用方法：**

**创建：**

**boolean createNewFile()如果文件存在则创建**

**mkdir() / mkdirs()创建文件夹，后者支持多级目录**

**判断：**

**boolean exists()**

**boolean isFile()**

**boolean isDirectory()**

**boolean isHidden()是否为隐藏文件**

**boolean isAbstract()是否为绝对路径**

**获取：**

**String getPath() 获得相对路径**

**String getAbsolutePath()**

**String getName()**

**String getParent()获取的是绝对路径的父目录，如果文件不存在返回空**

**renameTo()功能类似于剪切**

**String[] list()获取目录下的所有目录名和文件名，调用他的对象必须是目录对象，该目录还必须存在**

**File.listRoots()获取系统盘符**

**File[] listFiles();//获取目录下的文件夹**

**删除：**

**boolean delete() 返回值boolean 正在使用的文件无法删除**

**void deleteOnExit(); 在退出程序时删除**

**long length()返回字节**

**long lastModified()最后一次修改的时间**

**创建方式：**

**File f=new File(“haha.txt”);**

**File f=new File(“c://abc”,”haha.txt”);**

**File f=new File(“c://abc”);**

**File f1=new File(f,”haha.txt”);**

**补充：File.separator 相当于\\ 可用于解决跨平台识别问题**

**文件过滤：**

**应用场景：在一个目录下获取指定要求的文件**

**实现代码：**

**File dir=new File(“C://abc”);**

**dir.list(new FilenameFilter(){**

**public boolean accpet(File dir,String name){**

**return false;//此处为条件，返回值为false则会将文件过滤掉**

**}**

**});**

**I/O输入输出流**

**概述：**

**IO流用来处理设备之间的数据传输**

**Java对数据的操作是通过流的方式**

**Java用于操作流的对象都在IO包中**

**流按操作数据分为两种：字符流与字节流**

**流按照流向分为：输入流；输出流**

**常用的基类：**

1. **字节流的抽象基类：**

**InputStream、OutputStream**

1. **字符流的抽象基类：**

**Reader、Writer**

**注：由这些基类派生出来的子类一般以父类名为后缀名，前边为功能的具体体现，比如FileReader.**

**FileReader 与 FileWriter**

**分别为字符输入输出流**

**用于操作文本文件，流出现就要指定要操作的文件,需要进行异常处理**

**FileWriter fw=new FileWriter(“demo.txt”);//会创建一个文件，如果文件已经存在，则会发生覆盖（覆盖无非就是删除原来的，在创建一个新的）**

**要想不被覆盖，则需多加一个参数**

**FileWriter fw=new FileWriter(true,”demo.txt”);**

**FileReader fr=new FileReader(“demo.txt”);**

**要保证该文件是存在的，如果不存在会发生FileNotFoundExecption异常**

**读取方式1**

**read();一次读取一个字符，如果读到流的末尾，返回-1;**

**读取方式2**

**read(char[]);将内容读取到字符数组中，返回int类型，即读到的字符个数**

**字符缓冲区**

**缓冲区的出现是为了提高流的操作效率**

**对应类：**

**BufferWriter BufferReader**

**缓冲区要结合流才能使用**

**使用方法：**

**只需要将需要被提高效率的流对象作为参数传递给缓冲区的构造函数即可**

**BufferWriter bw=new BufferWriter(fw);**

**bw.writer(“abc”);**

**bw.flush();//只要用到字符缓冲区，就得刷新**

**bw.close();**

**//fw.close(); 可以省略，因为缓冲区就是基于对流的操作，关闭缓冲区时，在其内部就已经将流关闭了。**

**BufferReader中对数据的读取提供了一个特有方法，readLine();一次读取一行数据。**

**BufferReader类的子类LineNumberReader**

**实现了打印行号的功能**

**LineNumberReader ln=new LineNumberReader(fr);**

**ln.getLineNumber();**

**也可以设置行号的开始数字**

**ln.setLineNumber();**

**InputStream与OutputStream**

**一般用于操作二进制数据**

**FileInputStream FileOutputStream**

**BufferInputStream BufferOutputStream**

**读取键盘录入：**

**system.out:对应的是标准输出设备，控制台**

**system.in:对应的是标准输入设备，键盘**

**system.setOut(PrintStream ps);改变标准输出设备**

**system.setIn(PrintStream ps);改变标准输入设备**

**system.getProperties()获取系统信息**

**InputStream in=system.in;**

**in.read();//是一个阻塞方法，如果没有读到数据，就会处于等待状态**

**OutputStream os=system.out;**

**注：\r\n 相当于enter键 13 49**

**转换流：**

**InputStream in=system.in;**

**字节通向字符的桥梁**

**//将in转换为字符流**

**InputStreamReader isr=new InputStreamReader(in);**

**//为提高效率，可以加入缓冲区**

**BufferReader br=new BufferReader(isr);**

**简写方式：**

**BufferReader br=new BufferReader(new InputStreamReader(in));**

**字符通向字节的桥梁**

**OutputStream os=system.out;**

**OutputStreamWriter osw=new OutputStreamWriter(os);**

**BufferWriter bw=new BufferWriter(osw);**

**转换流可以指定字符编码。**

**一般涉及到编码问题时，通常使用转换流**

**编码：将字符转换为字节**

**解码：将字节转换为字符**

**注意：“联通”这个词是个特例，它的编码格式符合UTF-8编码格式，用gbk保存后，读取时如果用utf-8会出现乱码**

**编码编错无法解决**

**解码解错，将其重新编码，再用别的编码解读**

**流操作的基本规律：**

**1：明确源和目的**

**源：输入流 InputStream Reader**

**目的： 输出流 OutputStream Writer**

**2:操作的数据是否是纯文本**

**是：字符流**

**不是：字节流**

**3：当体系明确后，再明确要使用哪个具体对象**

**通过设备来进行区分**

**源设备：内存，硬盘，键盘**

**目的设备：内存，硬盘，控制台**

**例如：**

**需求：将键盘录入的数据存储到文件中去**

**分析：**

**源：键盘录入：对应对象system.in;**

**是不是纯文本？**

**是：Reder,单是system.in不是字节流吗？那么为了操作方便我们应该使用转换流InputStreamReader**

**是否需要提高效率？**

**是：加入缓冲技术**

**目的：硬盘**

**是不是纯文本？**

**是，Writer ,是一个文件，使用FileWriter**

**Properties**

**它是hashtable的子类**

**也就是说它具备map集合的特点，而且它里边存储的键值都是字符串。**

**是集合中和IO技术相结合的集合容器**

**该对象的特点：可以用于键值对形式配置文件**

**1、将文本数据中存储的信息存储到集合中（文件中存储的形式应该为name=value）**

**FileInputStream fis=new FileInputStream(“demo,txt”);**

**Properties prop=new Properties();**

**prop.load(fis);**

**prop.list(输出流);列出集合数据**

1. **更改集合中数据的值，并将其文件中的数据也修改掉**

**FileOutputStream fos=new FileOutputStream(“demo.txt”);**

**Properties prop=new Properties();**

**prop.setProperty(name,value)**

**prop.store(fos，“注释信息，不支持中文”);写回数据**

**打印流**

**该流提供了打印方法，可以将各种数据类型数据都原样打印**

**可以指定字符编码，当打印方法为println，可以在构造中加参数PrintWriter(in,true);此时可以不用刷新**

**字节打印流**

**PrintStream**

**构造函数可以接受的参数类型**

1. **file对象**
2. **字符串路径**
3. **字节输出流（OutputStream）**

**字符打印流**

**PrintWriter**

**构造函数可以接受的参数类型**

**1、file对象**

**2、字符串路径**

**3、字节输出流（OutputStream）**

**4、字符输出流**

**序列流**

**用于将多个流变成一个流**

**使用步骤：**

1. **创建一个Vector集合，用于存储字节输入流（如果用其他流。需要重写Enumeration中的方法）**
2. **将Vector中的元素存储进Enumeration中**
3. **将Enumeration作为参数传递给序列流**

**Vector<FileInputStream> v=new Vector<FileInputStream>**

**Enumeration<FileInputStream> en=v.elements();**

**SequenceInputStream sis =new SequnceInputStream(en);此流为InputStream的子类**

**操作对象的流**

**ObjectInputStream ObjectOutputStream**

**被操作的对象需要实现Serializable（标记接口，没有方法的接口，称为标记接口），通过实现Serializable接口，使其具有被序列化的功能**

**ObjectOutputStream oos=new ObjectOutputStream(OutputStream os);**

**oos.writerObject(Object obj);**

**ObjectInputStream ojs=new ObjectInputStream(InputStream in);**

**Object obj=ojs.readerObject();**

**序列号：static final long serialVersionUID=42L;**

**通过该id号识别被序列的对象，序列时会根据类的成员进行计算，也可以自己给指定了固定值（此时你的类被修改后，使用的还是原来的序列对象），**

**注意：静态成员不能被序列化，当非静态成员被transizent修饰后，也不可被序列化**

**管道流**

**PipedInputStream 和PipedOutputStream**

**输入输出可以直接进行连接，通过结合线程使用**

**使用时，这两个流要进行连接**

**可以将输出流作为参数传给输入流**

**也可以使用connect（）进行连接**

**随机访问流**

**RandomAccessFile**

**该类不是IO体系中的子类**

**而是直接继承子object**

**但它是io包中的成员，因为它具备读和写数据的能力**

**内部封装了一个数组，而数组通过指针对数组的元素进行操作。**

**可以通过getFilePointer获取指针的位置，同时也可以通过seek改变指针的位置，seek指定的是字节数**

**它能完成读写的功能，其原理就是内部封装了输入输出流**

**它只能操作文件，而且还有 只读r（不会自动创建文件） 读写rw 等模式**

**应用：可以进行多线程下载，一个线程负责指定段的数据写入，不会造成数据顺序错误**

**操作基本数据类型的流的对象**

**DataInputStream 与 DataOutputStream**

**装饰设计模式**

**思想：**

**当想要对已有的对象进行功能增强时，可以定义一个类，将以有的对象传入，基于已有的功能，提供加强功能。这个自定义的类为装饰类**

**装饰类通常会通过构造方法接受被装饰的对象。**

**装饰模式与继承：**

**装饰模式比继承要灵活，当有大量已有对象需要进行增强时，使用继承会造成体系臃肿。而装饰设计模式避免了继承体系的臃肿，而且降低了类与类之间的关系。**

**因为装饰类增强的是已有对象的功能，具备的功能和已有对象是相同的，所以装饰类和被装饰类通常都是属于同一个体系中的（即同一个级别）。**

**GUI程序**

**Graphical User Interface(图形用户接口)**

**用图形的方式，来显示计算机操作的界面，这样更方便更直观**

**补充：CLI**

**Command line User Interface(命令行接口)**

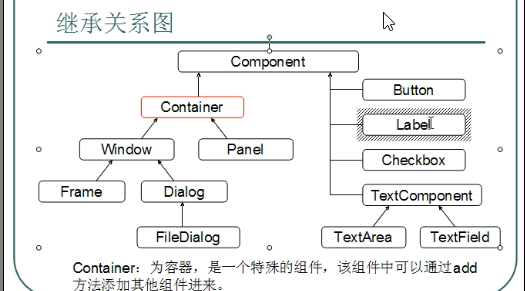
**就是常见的DOS命令行操作**

**Awt 与Swing**

**Java.Awt：Abstract Window Toolkit(抽象窗口工具包)，需要调用本地系统方法实现功能，属于重量级控件**

**javax.Swing:在AWT的基础上，建立的一套图形界面系统，其中提供了更多的组件，而且完全由Java实现，增强了抑制性，属于轻量级控件**

**体系结构：**

****

**常用布局管理器：**

**FlowLayout(流式布局管理器)**

**从左到右的顺序排列，**

**是Panel的默认布局管理器**

**BorderLayout(边界布局管理器)**

**东南西北中**

**Frame默认的布局管理器**

**GridLayout(网格布局管理器)**

**规则的矩阵**

**GardLayout(卡片布局管理器)**

**选项卡**

**GridBayLayout(网格包布局管理器);**

**非规则的矩阵**

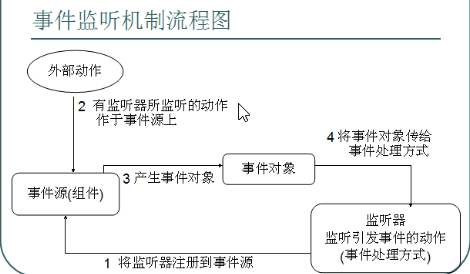
**窗体对象：**

**Frame**

**方法：**

**setSize();设置窗体大小**

**setLocation();初始位置**

****

**事件监听机制的特点：**

1. **事件源**

**就是awt包或者swing包中的那些图形界面组件**

1. **事件**

**每一个事件源都有自己特有的对应事件和共性事件**

1. **监听器**

**将可以触发某一个事件的动作（不止一个）都已经封装到了监听器中**

1. **处理事件**

**更多控件的创建和使用，请参考ＡＰＩ文档**

**网络编程**

**网络通讯要素：**

**IP地址：**

网络中的设备的标识

不易记忆，可用主机名

本地回环地址：127.0.0.1 主机名：localhost

**端口号：**

用于标进识程的逻辑地址，不同进程的表示（也可以说不同应用）

有效端口：0~65535，其中0~1024系统使用或保留端口

**传输协议：**

通讯的规则

常见的有：

**TCP**

建立连接，形成传输数据的通道。

在连接中进行大量数据传输

通过三次握手完成连接，是可靠协议

必须建立连接，效率低

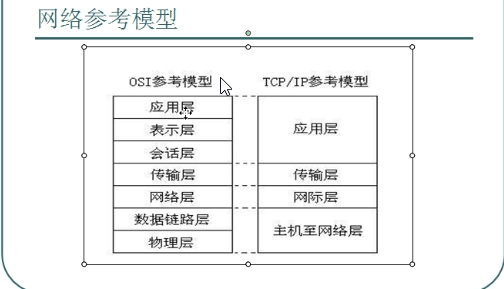
**UDP**

将数据及源和目的封装在数据包中，不需要建立连接

每个数据包的大小限制在64k内

因无连接，是不可靠协议

不需要建立连接，速度快

****

**IP地址封装**

**InetAddress address=InetAddress.getByName(“name”);**

**Socket套接字**

socket就是为网络服务提供的一种机制。

通讯的两端都有socket

网络通信，其实就是socket之间的通信

数据在两个socket间通过IO传输

**UDP传输**

DatagramSocket（UDP服务） 与 DatagramPacket（数据包）

建立发送端、接收端，他们都是DatagramSocket对象

建立数据包

调用socket的发送方法

关闭socket

发送端和接收端是两个独立的运行程序

数据包有四个参数，第一个为字节数组，即你要传输的数据，第二个为字节长度，第三个是你要目的地的主机名称，第四个是端口号

接收端与发送端应明确一个相等的端口，不然接收不到数据

发送数据send() 接收数据 receive()接收是一个阻塞方法

**TCP传输**

Socket（客户端） 和 ServerSocket（服务端）

建立客户端和服务器端

建立连接后，通过Socket中的IO流进行数据的传输

关闭socket

同样，客户端与服务器端是两个独立运行的应用程序

客户端在建立时，就可以去连接指定的主机，因为socket是面向连接的，所以建立socket服务时，就要有服务端存在，并连接成功，形成通路后，就可以通过该通达进行数据的传输

创建socket时，就要指定要连接的主机和端口

服务器端通过accept()方法获取接收过来的socket 对象，并进行交互，如果没有连接，就会进入等待状态，是一个阻塞方法