# Java

## 说在前面的话JAVA那点事

Ctrl+Shift+O作用是缺少的Import语句被加入，多余的Import语句被删除

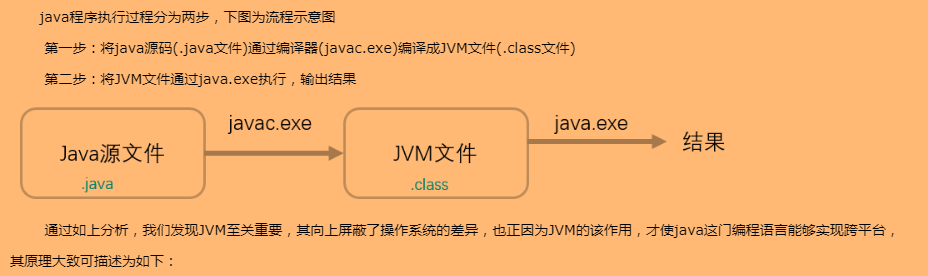
解释执行的运行环境，跨平台运行，解释型语言每次执行都要进行一次编译，运行效率较低，不能脱离解释器独立运行。但其跨平台容易，只需提供特定的平台解释器即可。可以方便进行程序的移植

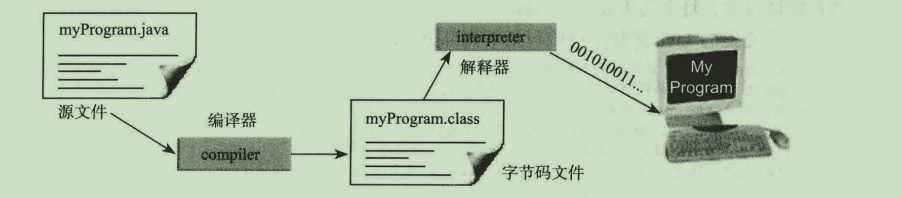
而编译型语言是一次编译成机器码，脱离开发环境独立运行，效率较高。

Java语言编写的程序先经过编译，但不会生成特定平台的机器码，而是生成与平台无关的字节码，即.class文件，这种文件是不可执行的，

必须通过Java解释器来解释执行

而解释执行字节码文件的是JVM.JVM是可执行Java字节码文件的虚拟计算机





IDE集成开发环境

Java程序的跨平台主要指字节码文件(.class文件)可以在任何具有Java虚拟机的计算机或电子设备上运行

Java程序的两种运行方式，通过Java虚拟机直接运行的Java Application,另一种是通过浏览器运行的Java Applet

1.Java三兄弟

Java SE(Java Platform Standard Edition)

标准版，是基础包，适用于客户端桌面单机系统应用和Applet小程序

Java EE(Java Platform Enterprise Edition)

企业版，在Java SE的基础上构建

Java ME(Java Platform Micro Edition)

微缩版，适用于小型设备和智能卡，移动设备等

## 前言：名字

方法名和变量名首字母小写，其后单词的首字母大写

类名和接口名一样方法所有单词的首字母大写

常量名全部大写，单词与单词之间用下划线分割如：

SIZE\_NAM

类的书写：

[public][abstract|final] class ClassName{

}

公有类要求该类所在的文件名与此类名相同，一个程序文件中最多包含一个公有类，缺省public是表示普通类，abstract表示抽象类。Final表示最终类

类中的成员：变量成员，方法成员

变量成员咋子写：作用域是整个类

[public|protected|private][static][final] [transient][volatile] type variableName

方法成员咋子写：

[public|protected|private][static] returnType methodName([paramList])

Public void getName(){

}

## 构造方法P78

通过构造方法，类的设计者可以确保每个对象都会被初始化。创建对象时，如果其类具有构造方法，Java就会在操作对象之前调用相应的构造方法，从而保证了初始化的进行。

1.不接受任何参数的构造方法叫做默认构造方法

2.构造方法可以有带形式参数，以便指定如何创建对象

PS：构造方法必须和类具备相同的名字

构造方法没有返回类型，甚至连void都没有

构造方法的作用是初始化对象

## Java修饰符

1.访问控制修饰符

Default 默认，什么不写的情况下，同一个包内可见

Private 对象，变量，方法，不能修饰类 用一个类中可见

Public 类，接口，方法，变量，对所有的类可见

Protected 不能修饰类，对同一个包内的类和所有子类可见

使用默认访问修饰符声明的变量和方法，对于接口变量都隐式声明为public static final ,接口的方法默认为public

接口及接口的成员变量和成员方法不能声明为 protected

2.非访问修饰符

static 修饰符，用来修饰类方法和类变量。

final 修饰符，用来修饰类、方法和变量，final 修饰的类不能够被继承，修饰的方法不能被继承类重新定义，修饰的变量为常量，是不可修改的。

abstract 修饰符，用来创建抽象类和抽象方法。

synchronized 和 volatile 修饰符，主要用于线程的编程。

一个类不能同时被 abstract 和 final 修饰

抽象方法不能被声明成 final 和 static。

final 修饰符通常和 static 修饰符一起使用来创建类常量。

## Java基本数据类型

内置数据类型

1.byte 8位数据类型，有符号，默认为0，-127~127

2.short 16位，有符号

3.int 32位，有符号

4.long 64位，有符号

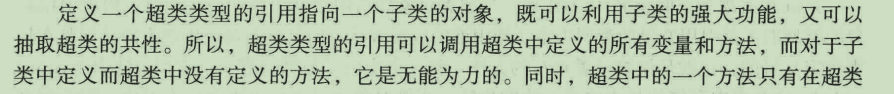
5.float 32位，单精度的浮点型

6.double 64

7.boolean ture或者false

8.char类型，单一的16位 Unicode字符

## 向上转型（即子类型转化为超类）upcasting

Animal a=new Cat();

一般来说子类在功能上 较超类更强大，属性也更丰富，因为子类是对超类的一个改进和扩充

超类中定义的且子类中没有重写的方法，才可以被超类类型的引用调用，而如果子类重写了该方法，那么超类类型的引用将调用子类中的这个方法，这就是动态连接。

## 重写和重载

方法重载overload是在一个类中，方法名相同，但是参数列表不同，即参数的类型，个数和不同参数类型的顺序不同

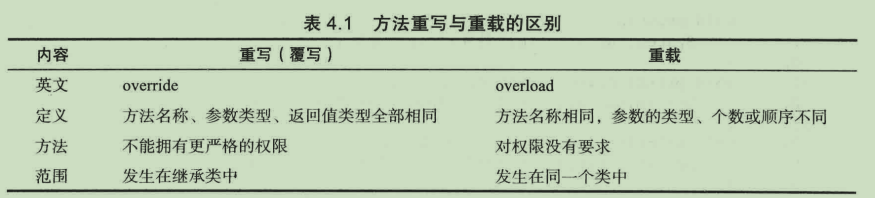
构造方法也可以重载的

Java通过参数列表决定实际调用的重载方法

方法重写override是在继承类中，要求参数列表相同，返回值类型相同

重写的方法不能拥有比超类方法更加严格的访问权限

重载继承类



## Abstract和final关键字

类必须为抽象类的情况，即前面必须加abstract关键字的类如：

abstract class a1

1.类中包含一个或多个明确声明的抽象方法

2.类中任何的一个超类包含一个或多个没有实现的抽象方法

3.类实现了一个接口或者继承了一个抽象类而没有实现该类所拥有的抽象方法

抽象类的实现依据子类的继承和实现

每一个抽象类的子类都可以对抽象方法进行不同的实现（多态性）

Final关键字修饰的final类不能被继承，即不能有子类

Final方法，如果一个类不允许其子类覆盖某个方法，在方法前加上final关键词，上锁，也可以提高程序的执行效率

Final常量

## this和super

this变量出现在方法成员的内部，指向当前对象

## 绑定（binding）

将一个方法调用和一个方法主体连接到一起的机制

静态绑定：即在程序运行之前进行绑定，也称为早期绑定

动态绑定：在程序运行期间绑定个，称为后期绑定

Java中只有private、static和final修饰的方法以及构造方法是静态绑定。

PS:由于动态绑定需要在运行时确定执行哪个版本的方法实现或者变量，比起静态绑定起来要耗时。所以在不影响整体设计，我们可以考虑将方法或者变量使用private，static或者final进行修饰。

## 接口（interface）纯的抽象类

Java中的一个类只能有一个超类，但是可以有多个接口，多个类也可以实现相同的接口。



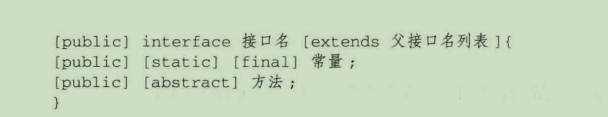
接口包含基本数据类型的成员变量，默认为static和final

接口中只有常量和方法原型

如：int CONST=1全称为final int CONST=1或static int CONST=1

1.接口中的方法默认都是public或abstract类型的，只有方法声明而没有方法体

2.接口中只包括public,static,final类型的成员变量

定义接口的基本类型

接口的访问权限可设为public或缺省值

缺省权限下，只有与该接口定义在同一个包中的类才可以访问这个接口

Public interface app {

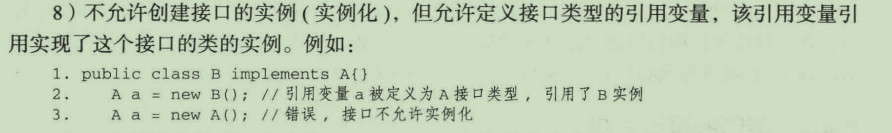
final float PI=3.13;

float getArea(float r);

}

定义接口时必须有interface关键字和接口名字

接口中没有构造方法，不能被直接实例化



3.接口必须通过类来实现它的抽象方法

Public class A implements A{

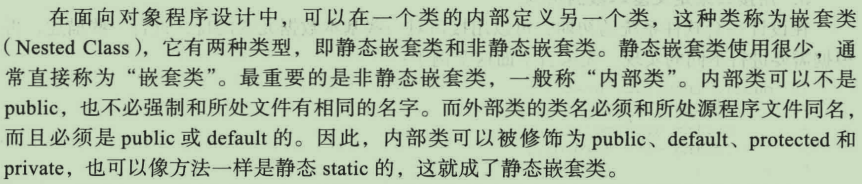
}

4.当类实现了某个Java接口时，它必须实现接口中的所有抽象方法，否则这个类必须声明为抽象

5.public class A extends b implements c,d{

}一个类只能继承一个直接的超类，但可以实现多个接口，间接的实现了多重继承

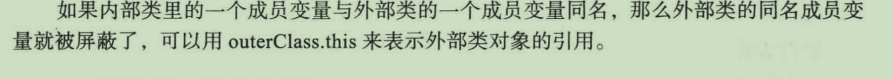
## 内部类

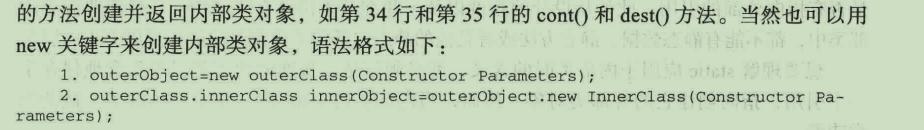


内部类(inner class)

说说内部类的好：1.封装性，咋就实现了，。。看不见我

2.一个内部对象可以访问创建它的外部类对象的内容，甚至私有变量





Outer out=new Outer();

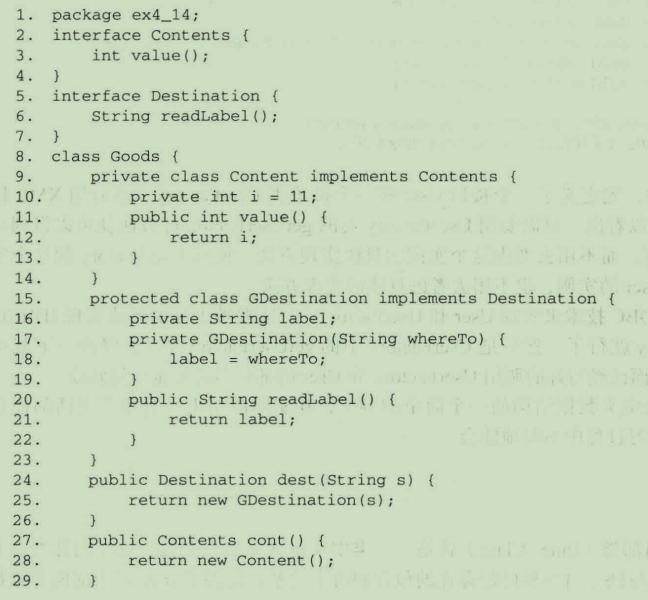
Outer.Inner in =out.new Inner();

其他内部类-静态嵌套类(Nested Class)

静态内部类没有指向外部的引用，与外部类对象没有联系,在任何非静态类中，都不能有静态数据，静态方法或其他静态内类

静态嵌套类只能访问外部类的静态成员

Outer.Inner in = new Outer.Inner();



接口类型的引用变量



在main方法里，直接引用了Contents c和Destination d进行操作，用户对内部类是透明的，即隐藏了内部类的操作，即封装性

2.局部内部类

Java内部类可以定义在方法甚至代码块之内，但不能有访问控制符

可以访问当前句块内的常量和外部类所有的成员

## 8.静态staticP47

只有用new创建对象时，数据才分配存储空间，其方法才可以供外界调用

1.静态方法只能访问静态成员，非静态既可以又可以

2.静态方法中无this和super关键字

3.最最重要的是主函数是静态的

Public static void main (){

}

4.被static修饰过的成员被所有的对象所共享

5.其可以被类名.静态成员直接调用

6.static修饰的为共享数据，对象中的存储的是特有数据

## 9.Java的常见类们

### Math类

Math的方法都被定义为static形式，通过Math类可在主函数中直接调用，如：

Math.sin(Math.PI)

Integer x=4

x.byteValue()返回byte原生数据类型

x.doubleValue()

Math.ceil(2.3) Math.floor(-3.2) Math.round(3.4)四舍五入

Math.random() 随机数的默认范围0-1

### Character类

Character.isLetter(‘c’)返回true或者false

Character.toLowerCase(‘a’)如果有的话返回小写字母，否则返回字符本身

### String类

获取字符串的长度

String site = “dcdcdcdcd”

Int len site.length();

连接字符串 string1.concat(string2)

或者直接相加 +

Char a=String1.charAt(3);返回索引处的字符串

### Java StringBuffer 和 StringBuilder 类

与String类不同的是，两者可以被多次的修改，并且不产生新的未使用的对象

StringBuffer与StringBuilder相比，线程安全的

但是StringBuilder有速度优势，大多使用StringBuilder类，但在程序要求线程安全的情况下，则使用StringBuffer类

**1 public StringBuffer append(String s)**

**将指定的字符串追加到此字符序列。**

**2 public StringBuffer reverse()**

**将此字符序列用其反转形式取代。**

**3 public delete(int start, int end)**

**移除此序列的子字符串中的字符。**

**4 public insert(int offset, int i)**

**将 int 参数的字符串表示形式插入此序列中。**

**5 replace(int start, int end, String str)**

**使用给定 String 中的字符替换此序列的子字符串中的字符。**

### 日期格式

import java.util.Date;

Date date = new Date();

System.out.println(date.toString());

格式化日期

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

1.使用 SimpleDateFormat 格式化日期

需要日期和时间的格式化编码

Date dNow = new Date( );

SimpleDateFormat ft = new SimpleDateFormat ("yyyy-MM-dd hh:mm:ss");

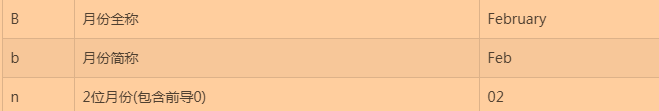
System.out.println("当前时间为: " + ft.format(dNow));

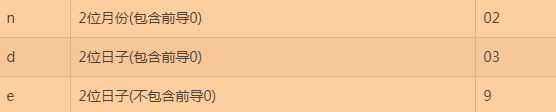
2.System.out.printf("%1$s %2$tB %2$td, %2$tY", "Due date:", date);

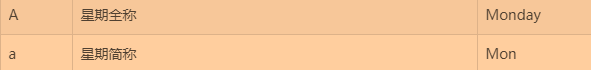
S代表字符串，第一参数为Due date

利用一个格式化字符串指出要被格式化的参数的索引。









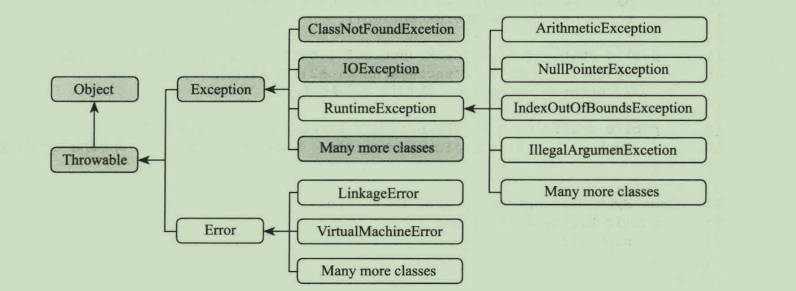
## 10.异常处理148

Exception(异常)指能够恢复正常的非正常事件

Error (错误)及其子类属于系统错误，描述系统内部的错误，由Jvm抛出，一旦发生，程序员没法处理

P151Unchecked Exception(不受检异常)不要求程序员捕获和处理异常，大多数情况下反映程序的逻辑错误，一般包括RuntimeException和Error及其子类。Java编译器不去检查，不需要进行显示捕获

受检异常（Checked Exception）编译器强制程序员检测和处理的异常

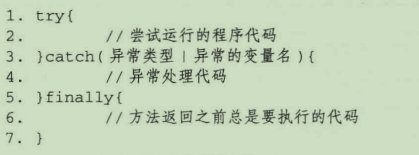


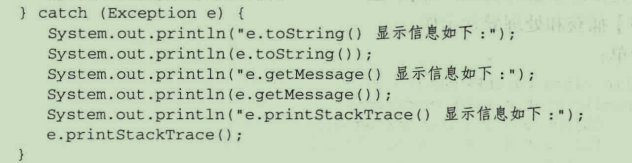
### 异常处理方法

1.使用try-catch-finally语句块结构在程序代码中捕获和处理异常

Catch块负责处理一种类型的异常，可以有多个catch语句，但匹配上一个异常后，后面的catch不再匹配

Finally:是程序必须执行的部分，不管try语块是否发生异常，总在方法返回前执行





e.getMessage() 获取异常的信息、

e.toString() 给出异常的类型和信息

e.printStackTrace() 打印出异常的类型，信息，并显示更深的调用信息

2.把异常对象通过向调用层抛出，一层层向上直至交给Java虚拟机处理，这种产生异常和转交异常的过程称为抛出异常

发生异常时逐层上抛因为必须保存一个完整的异常链信息，这会消耗大量的资源



Public void catch TrowException (int str) throws ArrayIndexOutOfBoundsException

如果在方法中产生了异常，方法会在抛出异常的地方退出

如果不想终止方法，就需要在特定的区域用try来捕获异常

============================================

### !!!throw和throws的区别

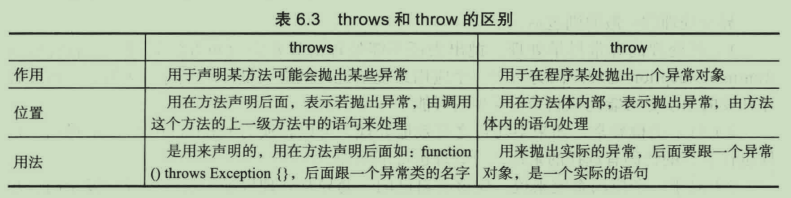
Throw是一种控制程序流程的特殊方法，无相应的catch,可以终止当前方法继续执行

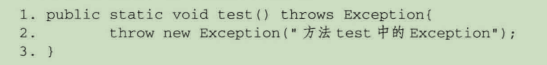
**throw则是抛出了异常，执行throw则一定抛出了某种异常 。抛出的异常，由方法体内的语句处理**

Throws用于对方法进行声明，如catch处理，否则编译时会产生错误

**抛出异常，由该方法的调用者来处理，用来声明一个方法可能产生的所有异常，不做任何处理而是将异常往上传，谁调用我我就抛给谁。  
 throws表示出现异常的一种可能性，并不一定会发生这些异常**

总结：throws可以单独使用，throw不可以，必须搭配try catch，或者throws，若程序执行到throw exception 语句，则后面的语句不会再执行。





### 自定义异常类

自定义的异常类必须是Exception的子类，注意的是自定义异常类之间也可以有继承关系，同时也需要为自定义异常类设计构造方法，方便自定义异常对象

## 11.对象容纳P166

### 数组

数组是一种数据结构，用来存储相同类型（对象，基本类型）值得集合

声明---》初始化

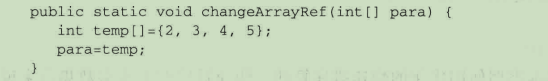
数组容纳的是对对象的引用，而基本数据类型容纳的是具体的数值

调用方法时使用数组作为参便可以改变数组中的值

动态初始化：

注意数组使用时要进行初始化，即将a初始化为一个真正的数组需要使用new关键字为数组开辟足够的空间

Int[ ] a = new int[100]



如上图，该方法企图改变a指向数组的应用，实际只改变临时变量para，并为对a造成影响

dataType[] arrayRefVar; // 首选的方法

double[ ] myList 首选的方法

*double myList[ ]*

数组变量的声明，和创建数组可以用一条语句完成，如下所示：

dataType[] arrayRefVar = new dataType[arraySize];

double[] myList = new double[size];

For-Each 循环

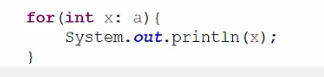
JDK 1.5 引进了一种新的循环类型，被称为 For-Each 循环或者加强型循环，它能在不使用下标的情况下遍历数组。

for (type element: array)

{

System.out.println(element);

}



多维数组

String str[ ][ ] = new String[3][4];

多维数组的属性 a.length第一维数组的长度 a[0].length数组中第一个元素引用的数组长度

多维数组的多态初始化

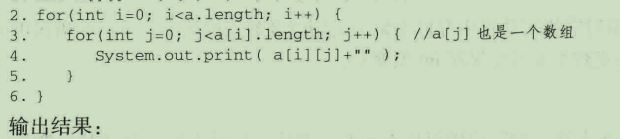
type[ ][ ] typeName = new type[typeLength1][typeLength2];

int a[ ][ ] = new int[2][3];

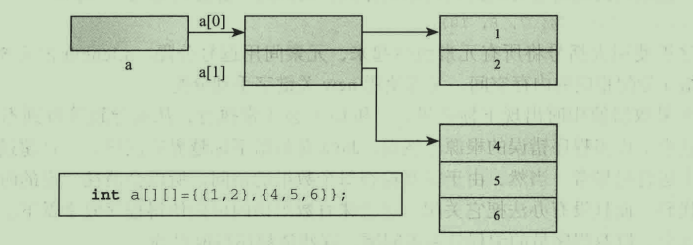
2.String s[ ][ ] = new String[2][ ];

s[0] = new String[2];

s[1] = new String[3];



Java中没有多维数组，只有一维数组。对于多维数组，其每个元素实际上都是一个低维数组的引用。直到其指向一维数组为止



如上图所示，整型数组a是一个不规则的数组，a[0]和a[1]分别为一个一维数组的引用。

值传递和引用传递

Int [ ] a

Int [ ] b=a

New的对象放在heap堆中

基本类型，类的实例（类的指针/实例）局部变量放在stack栈内存

数组工具类 java.util P170

Arrays.sort(a) //升序排列

Arrays.binarySearch(a,4) //索引值

Arrays.fill(a,100)//将数组a都填充为100

对象数组(即指向对象的数组元素<数组元素为对象类型>)：

class Apple{ }

Apple[ ] a= new Apple[3]

类型[ ]数组名 = new 类型[] {new构造方法，new构造方法。。。

} 如：Apple[] a = new Apple[ ]{new Apple( ),new Apple( )…}

当未指定数组中的元素值时，new操作符将数组元素自动初始化为0（针对数值类型）”\u0000”(针对字符类型)，null(针对字符串类型，对象类型)，false(针对布尔类型)

### 对象比较(Comparable和Compartor)

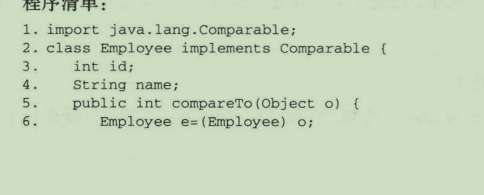
（当数组元素为对象类型时）判断两个接口的大小

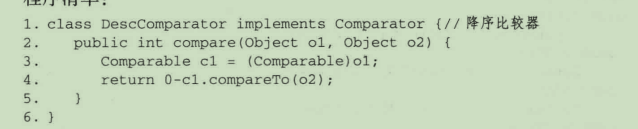
1.Comparable接口

Java,lang.Comparable接口中只有一个方法

Int compareTo(Object o)

对象o为被比较对象，大于，小于，等于指定对象o正整数，0，负整数







2.Comparator接口

Java.util.Comparator接口中有一个compare方法,与Comparable相比，Comparator拥有更优良的模块性，可以看做一种算法的实现，将算法和数据分离

int compare (Object o1,Object o2)

### 简简单单的枚举enumeration

为定义一组值得对象，包括0个或多个成员，枚举对象的值会自动获取一个数字值，从零开始递增

使用enum关键字，

Enum Season {SPRING,SUMMER,FALL,WINTER};

Season s = Season.SPRING

Season类型的变量只能保存此类声明的某个枚举量，或者null值

所有的枚举类型都是Enum类的子类

比较常用方法的有 toString（）返回枚举常量名，values( )返回一个包含所有枚举值的数组

Season.SPRING.toString( ) 返回字符串”SPRING”

Season[ ] a = Season.values( );

### 重点：容器（只保存对象）

数组的局限性：当数组初始化后，数组的容量酒确定

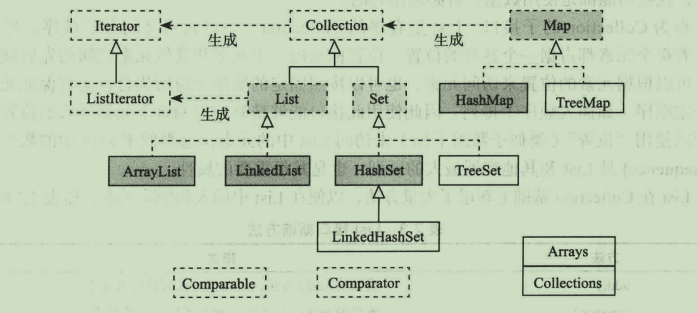
需要一种容量动态变化的对象保存保存方法

Java容器机制，只能保存对象，不能保存基本类型

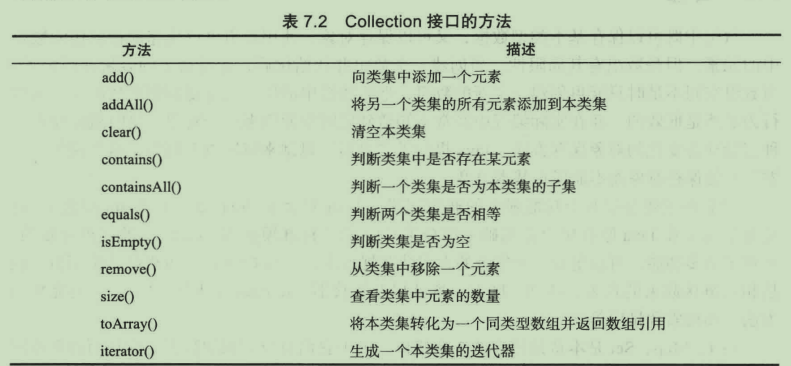
List,set,map是本章的三个核心接口

Collection类集：一组单独元素的集合

List ，Set Map键值对

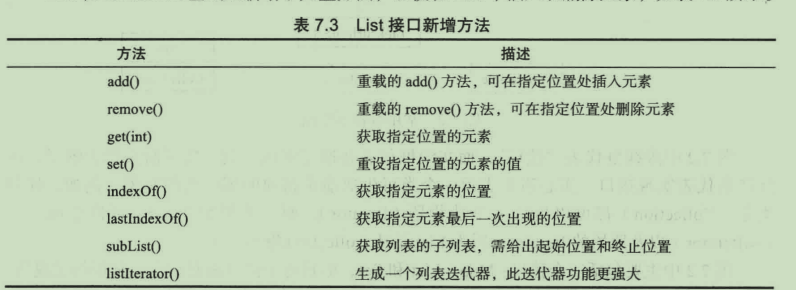


上图中虚线框表示接口，实线框代表普通类，实线箭头表示继承，虚线箭头代表实现接口



### （1）List（列表）

保存的对象有顺序，按照元素的索引位置检索对象，允许重复元素

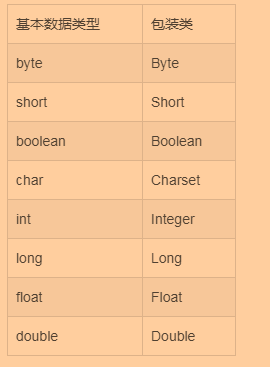


实现List接口的常见类 ArrayList(线性表)，LinkedList(链表)

ArrayList相当于动态数组，具有数组快速存取的优点，容量可动态变化，但插入和删除其中的元素较慢，因为维护的是一段连续的存储空间

ArrayList a = new ArrayList ( );

或者List a = new ArrayList( );//使用接口向上转型 八种基本类型



list.add(3);

//即自动转换为list.add(Integer.valueOf(3));

这种变换就叫做自动装箱(autoBoxing)

1. 还有
2. Integer n = 1000;

n++;

List a = new ArrayList( );//使用接口向上转型的好处

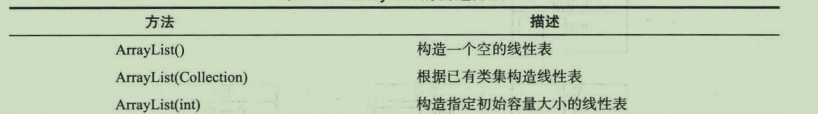
List b = new LinkedList( );

[1]改变集合类型时其他的代码不用变化

[2]ArrayList和LinkedList 的大部分方法都是接口List 定义的

[3]但是，实现类的特有方法时应使用具体的类定义

ArrayList的构造方法



对于不指定参数的做法（即无参数的构造方法），默认容量为10

问题：当不断向ArrayList中添加元素时，某时刻会超过表的容量，此时会再申请一个容量为原来数组二倍的新数组

ArrayList a = new ArrayList ( );

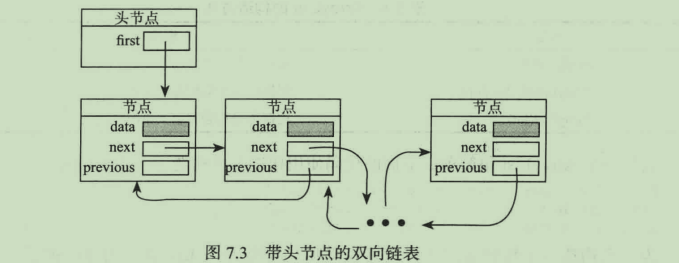
自动拆箱

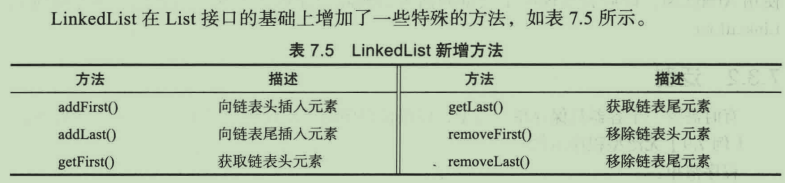
int n = list.get(i);

即为int n = list.get(i).intValue();

2.LinkedList链表

插入与删除较快，内部维护了一个带头节点的双向链表，方便进行双向访问





从链表中执行删除操作的代价小

在ArrayList中进行随机访问（即get()）以及循环访问是最高效的，但这些操作对于LinkedList开销较大。

进行插入和删除的操作LinkedList的效率更高

最好的方法 是

先选择为ArrayList,以后若发现大量的插入和删除操作降低性能，再考虑换成LinkedList

### （2）难点：泛型（generics）

<https://www.cnblogs.com/coprince/p/8603492.html>

标题：java 泛型详解-绝对是对泛型方法讲解最详细的，没有之一

需要一个容器只保存单一类型,除了异常类，枚举类，或者匿名类外

泛型允许编译器实施附加的类型约束，类型错误可以在编译时被捕获，而不是在运行时当做ClassCastException显示出来，可以提高程序的可靠性。消除强制类型转换

使用<>指定形式类型参数（类似于形参和实参），Java泛型不支持基本类型。如：

List<String> a = new ArrayList<String> ( );

类的泛型 class A <x1,x2> {x1 a,X2b}

泛型类的实例化：A<String,Integer> a = new A<String,Interger>

代表的是列表a只能保存String和String子类类型的元素，也就是只能保存可以强制类型转换成String类型的元素

原来：String s = (String) a.get(0)

现在：List<string> a = new ArrayList<String>( );//使用泛型后

String s = a.get(0);//无需强制类型转化直接为String类型

泛型方法，原因：该方法的多个参数之间声明一个类型约束

容器的泛型。。。。。\*&\*

#### 泛型类和泛型方法

[1]泛型类，是在实例化类的时候指明泛型的具体类型，（即传入泛型实参），从而根据泛型实参起到相应的限制作用。如果不传入实参也可以，从而能表示任意的类型。

[2]泛型方法比较复杂：只有声明<T>的方法才是泛型方法，注意泛型类中使用了泛型的成员方法不是泛型方法，

[3]泛型方法是在调用时指明泛型的参数类型

[4]在泛型类中声明泛型方法：

泛型类为T类型，泛型方法为E类型，两者可以相同，也可以不同

泛型接口：

与泛型类基本相同，如下：

Public interface Generator<T> {

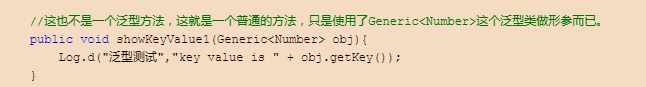
Public T next();

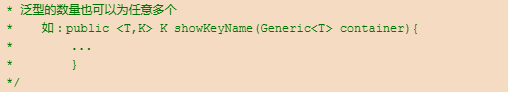
}

声明类的时候，将泛型的声明也一起加入到类中，即：

Class FruitGenerator<T> implements Genertor<T> {

}





2.静态方法与泛型

[1]静态方法无法访问类上定义的泛型，所以，如果静态方法操作的引用数据类型不确定的时候，必须要将泛型定义在方法上。

[2]即静态方法使用泛型，必须将静态方法也定义为泛型方法

如下：public static <T> void show (T t){ }

而public static void show (T t) { } 静态无法使用类泛型

E — Element，常用在java Collection里，如：List<E>,Iterator<E>,Set<E>

K,V — Key，Value，代表Map的键值对

N — Number，数字

T — Type，类型，如String，Integer等等

### （3）Set (集合)

不允许重复，元素无顺序，靠元素值索引对象

Set最大的特点是不允许保存相同值的元素，没有List的顺序概念

即{1,2}和{2,1}无差别，Set中的元素靠元素的值区别彼此

Set的遍历问题，因为无顺序，不能采取List的方法，

### 用Iterator（迭代器）

<增强代码的复用性和健壮性>

[1]Iterator功能简单，而且只能单向移动

调用方法Iterator( )要求容器返回一个Iterator类型的对象

第一次调用Iterator的方法时，返回序列的第一个元素

[2]使用next();方法获取序列的下一个元素，每次调用成功迭代器向后移动一个元素

[3]使用hasNext()的方法检查序列中是否还有元素

[4]使用remove（）的方法将迭代器新返回的元素删除

List<String> a = new ArrayList<String>( );

For(Iterator<String> it = a.iterator( );it.hasNext( ); ){

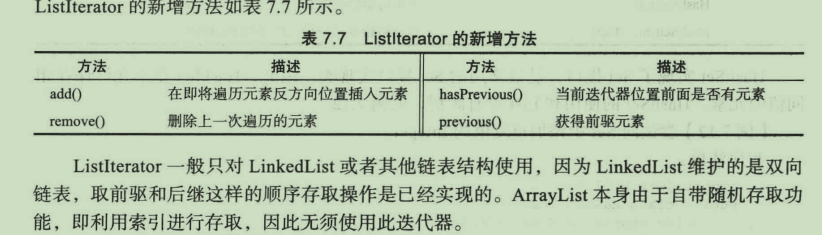
…

String s = it.next( );//取出元素

}

好处：当实现其他Iterable接口的类，只需改变集合类型的声明语句而后面的遍历代码无需任何修改

Set<String> a = new HashSet<String>( );



ListIterator的新增方法

2.for-each遍历

更为简洁地遍历数组和Collection的方法

Set<String> set = new HashSet<String>( );

…

For(String s :set) { System.out.println(s);//for(变量类型 变量名：集合 )

}相当于把迭代器包装了一下，代码更加简洁

只能遍历两种类型的对象【1】数组【2】实现java,lang.Iterable接口的类的实例，Iterable接口声明的方法只有一个就是iterator()的方法

该方法返回实现接口的对象。List和Set的实现类都可以使用for-each遍历

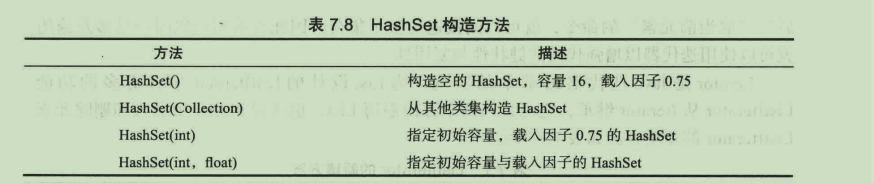
3.散列表（hash table）

目的：为了保证向Set中添加的元素不重复

快速查找特定对象的数据结构，即为hash table。散列表对每个对象计算一个整数，称为散列码（hash code）

HashSet(散列集，哈希集)正是实现这种数据结构的Set实现类

内部维护的是一个链表数组，常数查找时间



查找元素a的过程：

[1]调用a.hashCode();方法获得哈希码

[2]由哈希码计算出a在数组中的位置，即插入哪一个链表

[3]调用a.equals()方法和此链表中的每一个元素比较

[4]如果有相同的元素，a被找到，插入表头，结束。否则说明为找到

对于不同的对象，没有重写equals()与 hashCode()方法，则比较的是对象的地址，重写后则比较的是内容

重写时，对于一个合格的equals()方法必须在重写后满足

Public Boolean equals(Object o){

Employee e = (Employee) o;

If (this.id==e.id) return turn;

Else return flase;

}

[1]自反性：，对于任意x x.equals(x)总返回ture

[2]对称性：对任意x和y. x.equals(y)总返回ture,y.equals(x)也返回ture

[3]传递性：对任意的x,y,z 如果x.equals(y)为true,且y.equals(z)也为true,那么x.equals(z)总返回true

[4]持续性：对任意x,y 多次调用x.equals(y)的结果不会改变

[5]对任意非空x,x.equals(null)总是返回false

同时有 a.equals(b)时有a.hashCode()=b.hashCode()

1.内容相等的对象，散列码一定相等

2.对象的散列码不等，两个对象一定不相等

即在equals（）中进行比较的属性（如id）,一定要在hashCode()方法中进行计算

4.TreeSet 实现红黑树（自平衡排序二叉树）数据结构后得到的顺序Set.其中每个节点的值，大于等于他的左子树所有节点的值，小于等于它的所有右节点的值。

与HashSet相比，其优势在于TreeSet中所有的元素总是根据指定的排序规则而保持有序状态。

5.LinkedHashSet是HashSet的子类，在HashSet的基础上将每一个元素用链表串联起来。可以以按元素的插入顺序遍历

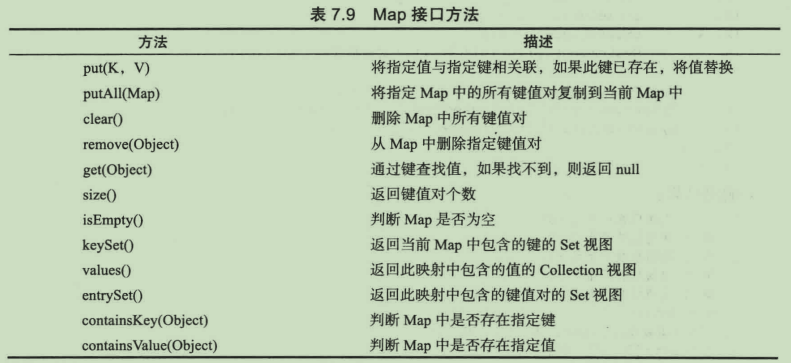
### （4）Map（映射）

键值对，有键对象和值对象，键是唯一的，值可以重复

Map没有继承Collection类

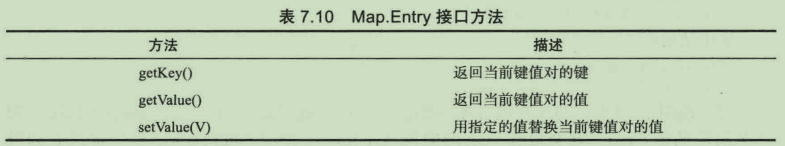
HashSet的add()方法调用其内部HashMap的put()方法

Public interface Map<K,V>



Map遍历，1.不能构造迭代器 2.没有像List那样的位置概念，无法使用索引

Map.Entry实际上将Map中的每个键值对看作一个对象，所以Map的映射集成为Map.Entry数据类型的集合



2.HashMap散列映射

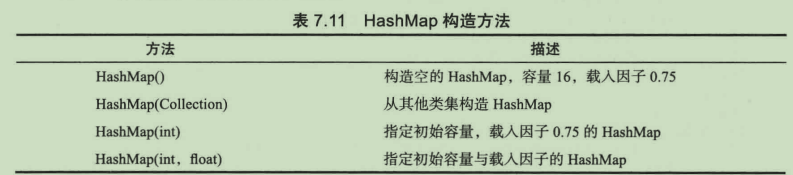
Map必须保持键的单一性

实际上，HashSet的实现完全基于HashMap来进行的，在HashSet的内部维持着一个HashMap，其中键就是要存入的对象，值是一个常量PRESENT,从而确保要储存的是 键

HashSet的所有方法都是调用它的内部HashMap的方法实现的

综上，HashSet与HashMap在本质上没有区别，只不过HashSet仅对HashMap的部分操作，HashMap多出了对值的操作功能

Public class HashMap<K,V>extends AbstracctMap<K,V>implements Map<K,V>,



HashMap的底层是一个数组，数组中的每个元素又是一个链表

向HashMap添加键值对时，首先调用k.hashCode( )方法获取其散列码，从而计算出其在数组中的位置，即插入到哪一个链表中，定位到某个具体链表后，接下来调用k.equals( )方法和链表中每个元素的键

比较，如果有相同的键，k被视为已经添加过，此时更新值v,否则将键值对插入链表头

HashMap的对象对应的类需要实现equals()：为了保证拥有相同键的元素无法插入。hashCode()方法：为了实现高效的查找速度

其他的Map类：

TreeMap同TreeSet类似，基于红黑树的数据结构的实现，得到的结果是排序的

LinkedHashMap是HashMap的子类，类似于LinkedHashSet

LinkedHashMap的遍历速度只与实际数据有关，与容量无关

HashMap的遍历速度与容量相关

用的最多的还是HashMap,在Map中插入删除和定位元素，选择HashMap.其存入的键值对在取出的时候是随意的

如果按照自定义顺序或自然顺序遍历键。

### （5）Collection容器的工具类

1.sort()排序，可以根据元素的自然顺序对指定List按升序排列

Collections.sort(list)

Collections中还有一个Collections.shuffle( )混排方法

2.binarySearch( )二分查找

前提，(有序)在排序好的List中查找指定元素并返回元素的位置。

对于没找到显示为负数

Collections.sort(list)

Int i = Collections.binarySearch(list, new Double(34.7));

3.reverse( )和fill( )填充

Collections.reverse(list)

For(Iterator<Double> it = list.iterator( );it.hasNext( );) {

System.out.print(it.next( )+” ,”)}

Fill将List中所有元素替换为指定元素

Collections.fill(list, new Double(1234) );

4.copy( )复制

Collections.copy(目标list,源list)

即将源list复制到目标list：目标至少与源list一样长。如果目标更长，则目标的剩余元素不受影响。

5.min( ),max( )

Collections.min( );

### （6）容器的选择

选择哪种容器更高效地解决问题

1.List的选择

在ArrayList中进行随机访问（即调用get()和set()方法是高效的

但同样的操作对于LinkedList开销较大，LinkedList只支持顺序存储

LinkedList按索引存取实际上用若干次顺序存储实现

而对于插入和删除

LinkedList比ArrayList的效率要高

通常的做法应该是先选择使用ArrayList作为默认List，如果以后发现大量的插入和删除操作，再考虑换成LinkedList

2.Set和Map的选择

HashSet 在增，删，查，改，操作这些数据的处理效率上是无与伦比的，而且与元素的多少关系不大。

TreeSet的元素按顺序排列，根据平衡二叉树，其查找的速度较快的（相对于List来说）。但是比不上HashSet

大概为log(n)级别因此如果想使用TreeSet,可以先构造HashSet，再从HashSet构造TreeSet

LinkedHashSet在要求有高效的存取性能，同时要求数据有序的情况下，比较适用

TreeSet相对于List,有较优的添加速度，但存取速度并不高

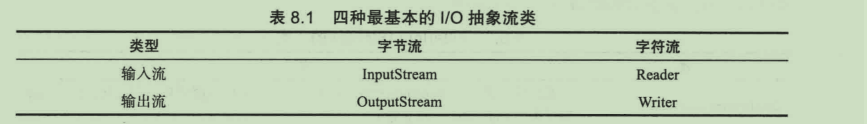
有时可以不把他作为Set使用，而作为创建顺序列表的一种途径

红黑树的本质决定了其内部元素的顺序排列的

一旦创建并赋值了一个TreeSet,就可以调用它的toArray( )方法产生包含其自身所有元素的一个有序数组，然后可用方法Arrays.binarySearch( )快速查找排好序数组中的内容。当然是在HashSet不适用的时候，因为HashSet的速度最快

P197-----》未看

## 12.I/O输入和输出流

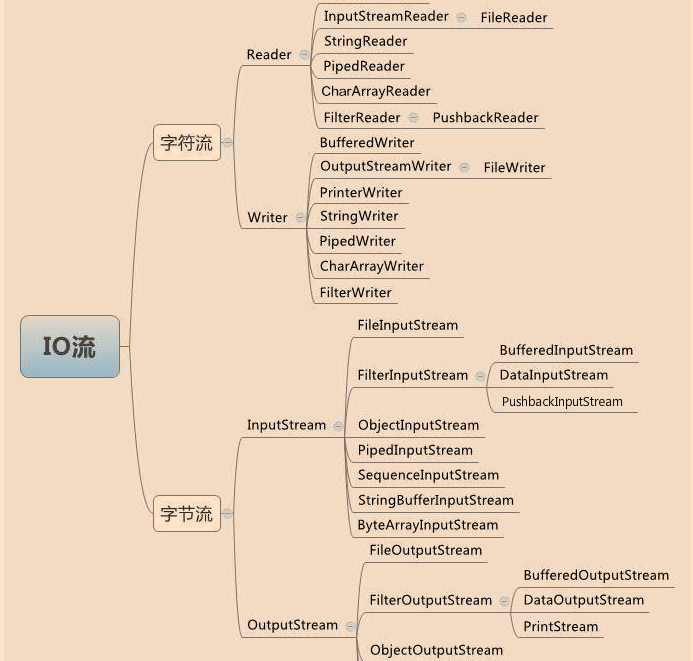


### 1.字节流及其方法

字节流是最基本的流，字符流是以字符为处理单位的流



### 2.输入/输出框架大纲



### **3.InputStream和OutputStream类及其子类中的方法read( )**

输入流为read( )方法

[1]public int read( )

从输入流读取下一个字节数据。返回0到255范围内的int字节值

如果因为到达结尾而没有可用的字节，则返回值为-1

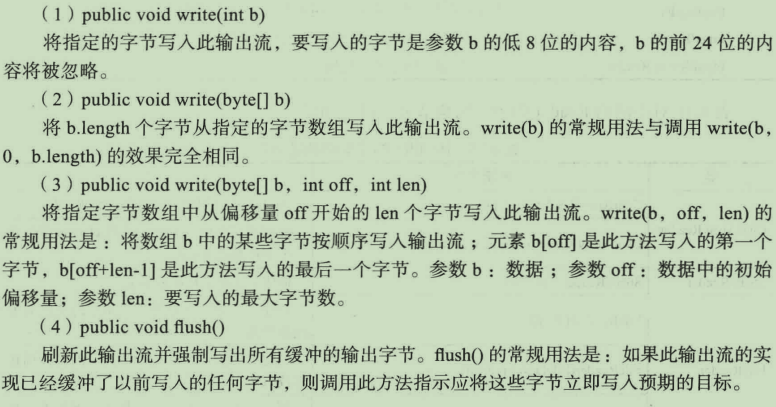
[2]public int read(byte[] b)

读取一定量的字节数据到缓冲区数组b中

[3]public int read（byte[ ]b,int off,int len）

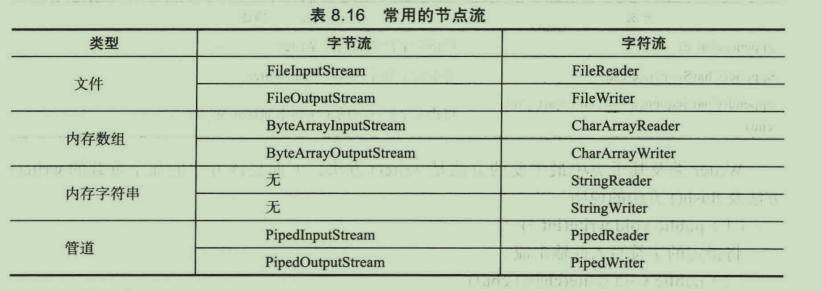
将输入流中最多len个字节数据读入数组,参数off,写入数据的初始偏移量

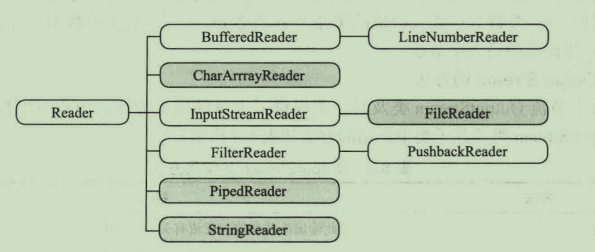
输出流为write( )方法

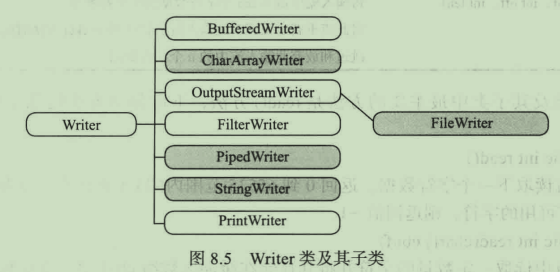


### 4.节点流和处理流

节点流可以从一个特定的数据源读取数据，数据源包括文件，内存和管道。主要的节点流包括：文件I/O流，内存的流，管道的流

FileInputStream和FileOutputStream用于从文件中读取，写入字节流。



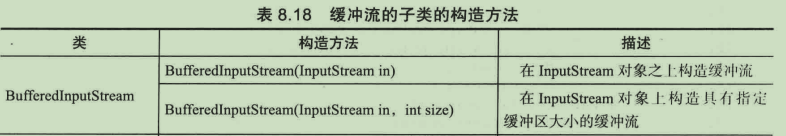


2.处理流是对一个已存在的流的连接和封装，通过调用所封装流的功能来实现数据的读写，修改或管理流中的数据，并提供额外功能，处理流的构造方法总要带一个其他的流对象作为参数



[1]缓冲流，没有缓冲的I/O,直接读写效率低，对硬盘损害大

带缓冲的输入流从一个类似于缓冲区的内存区中读取数据



[2]打印流（PrintStream/PrintWriter）

打印字节流和打印字符流

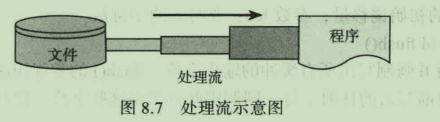
[3]数据流

提供可以存取与机器无关的java基本的数据类型的方法，以支持基本的数据类型的输入/输出

[4]对象流

使任意一个对象能够像数据流一样进行输入和输出。

注意：在对象流中进行输入和输出的对象需要是可序列化的类型



## One.重难点复习（抽象和接口）

1.不能直接创建new的抽象类对象

2.必须用一个子类来继承抽象父类

3.子类必须覆盖重写父类的所有抽象方法，否则子类也为抽象类

4.抽象方法所在的类一定是抽象类，抽象类中不一定有抽象方法，

5.抽象类中，可以有构造方法，是供子类创建对象时，初始化父类成员使用的

## 12.图形的用户话界面（GUI）P115

图形用户化界面道德实现所需的基本组件，全部在java,awt包中

抽象窗口工具包（Abstract Windowing Toolkit,AWT）设计目标为：

希望构建一个通用的GUI

之后推出的：Swing的功能更加强大，使用更加方便，但是Swing的框架在AWT之上，Swing并没有完全替代AWT

Eclipse组织编写的SWT的另一个不错的开源GUI编程框架

## 8.P202