# 1.IO流

## 1.1.合并流

### 1.1.1.SequenceInputStream

字节流---可以合并多个字节输入流，将这多个字节输入流进行合并的时候，需要将这个多个流放到一个Vector集合，利用Vector集合产生一个Enumeration对象，利用Enumeration对象来构造合并流对象---合并流只有输入没有输出

#### 1.1.1.1.构造函数

①SequenceInputStream(Enumeration<? extends InputStream> e);

②SequenceInputStream(InputStream s1,InputStream s2);

按照参数顺序来读取参数对象。

#### 1.1.1.2.重要方法

①read();

②read(byte[] b,int off,int len);

#### 1.1.1.3.代码示例

public static void main(String[] args) throws Exception {

// 创建一个输出流，将读取的数据写出

FileOutputStream out = new FileOutputStream("E:\\test.txt");

// 创建一个输入流指向a.txt

FileInputStream in1 = new FileInputStream("E:\\a.txt");

FileInputStream in2 = new FileInputStream("E:\\b.txt");

FileInputStream in3 = new FileInputStream("E:\\c.txt");

// 创建一个Vector对象来存储这三个输入流

Vector<InputStream> v = new Vector<>();

v.add(in1);

v.add(in2);

v.add(in3);

// 获取Enumeration对象

Enumeration<InputStream> e = v.elements();

// 创建合并流对象

SequenceInputStream sis = new SequenceInputStream(e);

// 利用合并流来读取数据

byte[] bs = new byte[4];

int len = -1;

while ((len = sis.read(bs)) != -1) {

out.write(bs, 0, len);

}

}

// 关流

out.close();

sis.close();

## 1.2.序列化/反序列化流

### 1.2.1.定义

序列化：将对象进行完整存储。对应的类：ObjectOutputStream

持久化：将对象保存到硬盘上

反序列化：将对象还原回来。对应的类：ObjectInputStream

注意：

1. 如果一个对象要想被序列化，这个对象对应的类必须实现一个接口---Serializable---这个接口中没有任何的方法和属性，仅仅用来标识这个类产生的对象可以被序列化

2. 被static/transient修饰的属性不会被序列化

3. 每一个类在序列化的时候都会有一个版本号。如果没有手动指定版本号，那么在编译的时候会根据当前类中的方法和属性自动计算一个版本号。对象在反序列化的时候会比较对象中的版本号和当前类中的版本号是否一致。如果手动指定了版本号，就不再自动计算。---版本号serialVersionUID---默认用private static final long修饰

4. 集合/数组不能被序列化

### 1.2.3.代表序列化的类

#### 1.2.3.1.ObjectOutputStream

①构造函数

1)ObjectOutputStream();

2)ObjectOutputStream(OutputStream out);

②重要方法

1)writeObject();

代码示例：

// 准备一个对象

Person p = new Person();

p.setName("雪花");

p.setAge(15);

p.setHeight(150);

p.setGender('女');

// 静态属性没有被序列化出去，为什么？

// p.hobby = "睡觉~~~";

// 创建一个序列化流

ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("E:\\p.data"));

// 将对象序列化出去

oos.writeObject(p);

// 关流

oos.close();

#### 1.2.3.2.ObjectInputStream

①构造函数

1)ObjectInputStream();

2)ObjectInputStream(InputStream in);

②重要方法

1)readObject();

代码示例

// 创建反序列化流对象

ObjectInputStream ois = new ObjectInputStream(new FileInputStream("E:\\p.data"));

// 将对象反序列化回来

Person p = (Person) ois.readObject();

// 关流

ois.close();

System.out.println(p.getName());

System.out.println(p.getAge());

// 如果一个属性没有被序列化出去，那么在反序列化的时候会给一个默认值

System.out.println(p.getHeight());

# 2.静态导入

## 2.1.定义

import static 包名.类名.方法名; ---只能够导入静态方法

import static java.util.Arrays.sort

//表示只导入sort方法。

# 4.可变参数

用...来定义可变参数，此时在传入参数的时候，参数的个数可以随意。可变参数本质上是一个数组。在参数列表里只能定义一个可变参数，而且必须在参数列表的末尾。

public static int sum(int...arr){

int sum = 0;

for(int i = 0;i<arr.length;i++){

sum+=arr[i];

}

return sum;

}

# 5.枚举

适用于一些取值相对固定并且能够一一列举的场景（星期、月份、季节）。用enum来定义一个枚举类。枚举常量必须定义在首行，而且之间用 , 隔开。

枚举类中允许定义一切的方法和属性。---可以定义抽象方法

所有的枚举类的顶级父类---java.lang.Enum

switch---需要一个表达式，表达式的值是byte/short/char/int, JDK1.7允许使用String，JDK1.5开始支持枚举常量

## 5.1.定义

当取值情况相对固定并且能够一一列举出来的现象，可以使用枚举来定义。

使用关键字enum来定义一个枚举类。将一个类声明为枚举，那么默认就将此类的构造函数私有化，枚举常量必须放在枚举类的首行，每个枚举常量之间是用“,”隔开，最后一个枚举常量以“;”结尾。

枚举本质上是一个类，所以可以在其中定义一切的属性和方法，包括抽象方法。

Java中枚举的顶级父类是java.long.Enum。

示例：

enum Level {

// public static final Level A = new Level(90);

A {

@Override

public void getLevel() {

System.out.println("优秀");

}

},

B(80) {

@Override

public void getLevel() {

System.out.println("良好 ");

}

},

C(70) {

@Override

public void getLevel() {

System.out.println("中等");

}

},

D(60) {

@Override

public void getLevel() {

System.out.println("及格");

}

},

E(0) {

@Override

public void getLevel() {

System.out.println("挂科");

}

};

private Level() {

}

private Level(double score) {

}

private double score;

public static final int count = 10;

public double getScore() {

return this.score;

}

public void setScore(double score) {

this.score = score;

}

public abstract void getLevel();

}

switch - case byte/short/char/int，可以使用，因为都可以自动提升为int。1.5可以使用枚举常量，1.7可以使用String。

示例：

public static void main(String[] args) {

Level l = Level.A;

switch (l) {

case A:

System.out.println("很好~~~");

break;

case B:

System.out.println("也行~~~");

break;

case C:

System.out.println("凑合~~~");

default:

break;

}

}