# 星期一

## JAVA基础

### 1、JVM、JRE和JDK的区别

JVM(Java Virtual Machine):java虚拟机，用于保证java的跨平台的特性。

java语言是跨平台，jvm不是跨平台的。

JRE(Java Runtime Environment):java的运行环境,包括jvm+java的核心类库。

JDK(Java Development Kit):java的开发工具,包括jre+开发工具

### 2、环境变量path和classpath的作用是什么？

(1)path是配置Windows可执行文件的搜索路径，即扩展名为.exe的程序文件所在的目录，

用于指定DOS窗口命令的路径。

(2)Classpath是配置class文件所在的目录，用于指定类搜索路径，JVM就是通过它来寻找该类的class类文件的。

### 3、变量有什么用？为什么要定义变量？什么时候用？

答：变量的作用：用来存储数据。

为什么要定义变量：用来不断的存放同一类型的常量，并可以重复使用

### 4、&和&&的区别?

答：（1）&&会出现短路，如果可以通过第一个表达式判断出整个表达式的结果，则后面表达式的不运算；

只能操作boolean类型数据；

（2）&不会出现短路，将整个表达式都运算。既可以操作boolean数据还可以操作数。

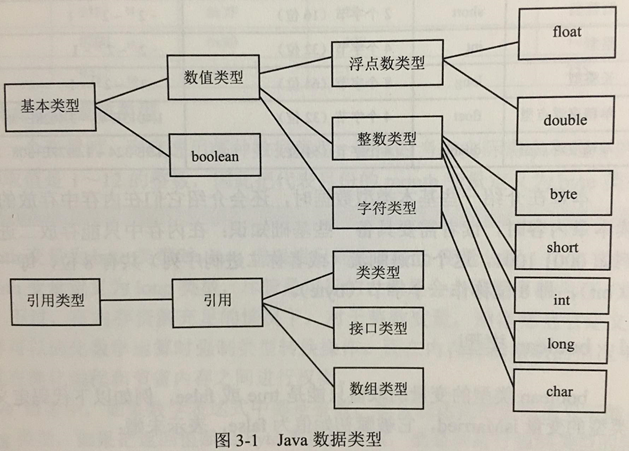
### 5、标示符命名规则：

由数字(0-9)，大小写英文字母，以及\_和$组成。

不能以数字开头。

不能使用关键字来自定义命名。

### 6、数据类型：



(1)基本数据类型(4类8种)：

整数类型：byte(8bits)、short(16bits)、int(32bits)、long(64bits)

浮点数类型：float单精度（32bits）、double双精度（64bits）

字符类型：char( unicode字符,16bits)

布尔类型：boolean(ture false)[1字节(8位)]

对应的类类型：Integer、Float、Boolean、Character、Double、Short、Byte、Long

(2)引用数据类型：类、接口、数组

### 7、类型转换

（1）布尔型和其它基本数据类型之间不能相互转换；

（2）byte型可以转换为short、int、、long、float和double；

（3）short可转换为int、long、float和double；

（4）char可转换为int、long、float和double；

（5）int可转换为long、float和double；

（6）long可转换为float和double；

（7）float可转换为double；

精度从高到低 double >float> long> int> short(char)> byte

(1)自动类型转换 低精度 转换 高精度

(2)强制类型转换 高精度 转换 低精度(精度会下降)

### 8、java语言的三种技术架构

J2EE：企业版

是为开发企业环境下的应用程序提供的一套解决方案。

该技术体系中包含的技术如 Servlet、Jsp等，主要针对于Web应用程序开发。

J2SE：标准版

是为开发普通桌面和商务应用程序提供的解决方案。

该技术体系是其他两者的基础，可以完成一些桌面应用程序的开发。

比如Java版的扫雷。

J2ME：小型版

是为开发电子消费产品和嵌入式设备提供的解决方案。

该技术体系主要应用于小型电子消费类产品，如手机中的应用程序等。

### 9、java的跨平台性：

通过Java语言编写的应用程序在不同的系统平台上都可以运行。

跨平台的原因：

只要在需要运行java应用程序的操作系统上，先安装一个Java虚拟机(JVM Java Virtual Machine)即可。

由JVM来负责Java程序在该系统中的运行。

### 10、有符号数据的表示法(次重点)

原码，反码(原码取反)，补码(反码+1)。

### 11、方法

定义：方法就是定义在类中的具有特定功能的一段独立小程序。

特点：

定义方法可以将功能代码进行封装

便于对该功能进行复用

方法只有被调用才会被执行

方法的出现提高了代码的复用性

对于方法没有具体返回值的情况，返回值类型用关键字void表示，

那么该方法中的return语句如果在最后一行可以省略不写。

方法的应用两个明确：

明确要定义的功能最后的结果是什么？

明确在定义该功能的过程中，是否需要未知内容参与运算

### 12、重载：

概念：在同一个类中，允许存在一个以上的同名函数，只要它们的参数个数或者参数类型不同即可。

特点：与返回值类型无关，只看参数列表(参数类型以及参数个数)。

好处：方便于阅读，优化了程序设计。

### 13、数组：

概念：同一种数据类型的集合。

好处：可以自动给数组中的元素从0开始编号，方便操作这些元素。

### 14、内存结构：

栈内存：用于存储局部变量，当数据使用完，所占空间会自动释放。

堆内存：数组和对象，通过new建立的实例都存放在堆内存中。

方法区：静态成员、构造函数、常量池、线程池

本地方法区：window系统占用

寄存器：寄存器是中央处理器内的组成部分。寄存器是有限存贮容量的高速存贮部件，它们可用来暂存指令、数据和地址。在中央处理器的控制部件中，包含的寄存器有指令寄存器(IR)和程序计数器(PC)。在中央处理器的算术及逻辑部件中，寄存器有累加器(ACC)。

## 面向对象

### 1、面向对象思想：

(1)概述：

面向对象是相对于面向过程而言的，面向过程强调是功能，面向对象强调是将功能封装进对象，

强调具备功能的对象；

(2)思想特点：

A:是符合人们思考习惯的一种思想；

B:将复杂的事情简单化了；

C:将程序员从执行者变成了指挥者；

比如我要达到某种结果，我就寻找能帮我达到该结果的功能的对象，如我要洗衣服，至于怎么洗我不管。

(3)特征：

封装:隐藏对象的属性和实现细节，仅对外提供公共访问方式

继承:多个类中存在相同属性和行为时，将这些内容抽取到单独一个类中，那么多个类无需再定义

这些属性和行为，只要继承那个类即可。

多态:一个对象在程序不同运行时刻代表的多种状态，父类或者接口的引用指向子类对象

### 2、类和对象：

类：对现实世界中某类事物的描述,是抽象的，概念上的定义。

对象：事物具体存在的个体。

### 3：成员变量和局部变量的区别(重点)

(1)作用域

成员变量：针对整个类有效。

局部变量：只在某个范围内有效。(一般指的就是方法,语句体内)

(2)存储位置

成员变量：随着对象的创建而存在，随着对象的消失而消失，存储在堆内存中。

局部变量：在方法被调用，或者语句被执行的时候存在，存储在栈内存中。

当方法调用完，或者语句结束后，就自动释放。

(3)初始值

成员变量：有默认初始值。

局部变量：没有默认初始值，使用前必须赋值。

### 4、匿名对象

(1)匿名对象就是没有名字的对象。是对象的一种简写形式。

(2)应用场景

A:只调用一次类中的方法。

B:可以作为实际参数在方法传递中使用

### 5、封装：

指隐藏对象的属性和实现细节，仅对外提供公共访问方式；比如电脑机箱、笔记本等

好处：将变化隔离；方便使用；提高复用性；提高安全性

### 6、关键字private：封装在代码中的体现

(1)私有的意思，权限修饰符

(2)用来修饰成员变量和成员函数

(3)用private修饰的成员只在本类中有效

(4)私有是封装的一种体现

### 7、构造方法：

(1)特点：方法名与类名相同;没有返回类型;没有返回值

(2)作用：构造函数是用于创建对象，并对其进行初始化赋值，对象一建立就自动调用相对应的构造函数，

(3)构造方法的注意事项:

A:如果一个自定义类没有构造方法，系统会默认给出一个无参构造方法。

B:如果一个自定义类提供了构造方法，那么，系统将不再给出无参构造方法。

这个时候，你可以不使用无参构造方法。

如果你想使用，那么，就必须手动给出无参构造方法。

建议：一般情况下，我们的自定义类都要手动给出无参构造方法。

(4)构造方法和成员方法的区别

A:格式区别

构造方法和类名相同，并且没有返回类型，也没有返回值。

普通成员方法可以任意起名，必须有返回类型，可以没有返回值。

B:作用区别

构造方法用于创建对象，并进行初始化值。

普通成员方法是用于完成特定功能的。

C:调用区别

构造方法是在创建对象时被调用的，一个对象建立，只调用一次相应构造函数

普通成员方法是由创建好的对象调用，可以调用多次

### 8、构造代码块：{}

(1)作用：给对象进行初始化，对象一建立就执行，而且优先于构造函数执行

(2)构造代码块和构造函数的区别：

构造代码块是给所有不同对象的共性进行统一初始化

构造函数是给对应的对象进行初始化

### 9、this关键字

(1)this关键字代表本类对象的一个引用，谁调用this所在的方法，this就代表谁

(2)this的使用场景

A:用于区分同名成员变量和局部变量；

B:在定义方法时，该方法内部要用到调用该方法的对象时，因为此时对象还没建立，故this代表此对象

B:构造方法间调用 \*\*这个时候，this(参数)必须作为第一条语句存在。

### 10、Person p = new Person();在内存中做了哪些事情。

(1)将Person.class文件加载进内存中。

(2)如果p定义在主方法中，那么，就会在栈空间开辟一个变量空间p。

(3)在堆内存给对象分配空间。

(4)对对象中的成员进行默认初始化。

(5)对对象中的成员进行显示初始化。

(6)调用构造代码块对对象进行初始化。(如果没有就不执行)

(7)调用构造方法对对象进行初始化。对象初始化完毕。

(8)将对象的内存地址赋值给p变量，让p变量指向该对象。

### 11、static关键字：

(1)静态的意思，用来修饰成员变量和成员函数

(2)静态的特点:

随着类的加载而加载；

优先于对象存在；

对所有对象共享；

可以被类名直接调用；

(3)静态的注意事项

A:静态方法只能访问静态成员

为什么：因为静态的内容是随着类的加载而加载，它是先进内存的。

B:静态方法中不能使用this,super关键字

C:主方法是静态的

D:主方法描述：public static void main(String[] args)

public:公共的意思，是最大权限修饰符。

static:由于jvm调用main方法的时候，没有创建对象。

只能通过类名调用。所以，main必须用static修饰。

void:由于main方法是被jvm调用，不需要返回值。用void修饰。

main:main是主要的意思，所以jvm采用了这个名字。是程序的入口。

String[]:字符串数组

args:数组名

在运行的时候，通过java命令给args数组赋值。

格式：java MainTest hello world itcast

(4)静态变量和成员变量的区别

A：调用方式

静态变量也称为类变量，可以直接通过类名调用。也可以通过对象名调用。

这个变量属于类。

成员变量也称为实例变量，只能通过对象名调用。这个变量属于对象。

B：存储位置

静态变量存储在方法区中的静态区。

成员变量存储在堆内存。

C：生命周期

静态变量随着类的加载而存在，随着类的消失而消失。生命周期长。

成员变量随着对象的创建而存在，随着对象的消失而消失。

D：与对象的相关性

静态变量是所有对象共享的数据。

成员变量是每个对象所特有的数据。

(5)静态的优点和弊端

优点：

对对象的共享数据进行单独空间的存储，节省内存，没有必要每个对象都存储一份

可直接被类名调用

弊端：

生命周期过长，随着类的消失而消失

访问出现权限，即静态虽好但只能访问静态

(6)什么时候使用静态呢？

A:当所有对象共享某个数据的时候，就把这个成员变量定义为静态修饰的。

B:当某个方法没有访问该类中的非静态成员，就可以把这个方法定义为静态修饰。

静态的生命周期比较长，所以一般不推荐使用。

(7)静态代码块

A:它只执行一次，它比main还先执行。

B:执行顺序：静态代码块-->构造代码块-->构造方法

### 12、制作API文档(次重点)

API(全拼):Application Program Interface 应用程序编程接口。

(1)类中的内容需要用文档注释。

(2)使用JDK\bin目录下的javadoc工具。

格式:javadoc -d 目录 -author -version ArrayTool.java

### 13、单例设计模式：

(1)设计模式：解决某类问题行之有效的方法，是一种思想，是规律的总结

(2)用来保证某个类在内存中只有一个对象

(3)保证唯一性的思想及步骤

\*\*为了避免其他程序建立该类对象，先禁止其他程序建立该类对象，即将构造函数私有化

\*\*为了其他程序访问到该类对象，须在本类中创建一个该类私有对象

\*\*为了方便其他程序访问到该类对象，可对外提供一个公共访问方式

比如API中的Runtime类就是单例设计模式。

(4)单例设计模式的两种方式

A:饿汉式 当类加载的时候，就创建对象。空间换时间

class Student{

private Student(){}

private static final Student s = new Student();

public static Student getInstance(){

return s;

}

B:懒汉式 当使用的使用，才去创建对象。时间换空间

class Student{

private Student(){}

private static final Student s = null;

public static Student getInstance(){

if(s==null) {

//线程1就进来了，线程2就进来了。

s = new Student();

}

return s;

}

饿汉式和懒汉式的区别：

饿汉式是类一加载进内存就创建好了对象；

懒汉式则是类才加载进内存的时候，对象还没有存在，只有调用了getInstance()方法时，

对象才开始创建。

懒汉式是延迟加载，如果多个线程同时操作懒汉式时就有可能出现线程安全问题，解决线程安全问题

可以加同步来解决。但是加了同步之后，每一次都要比较锁，效率就变慢了，

所以可以加双重判断来提高程序效率。

注：开发常用饿汉式，因为饿汉式简单安全。懒汉式多线程的时候容易发生问题

### 14、Math类的使用(重点)

(1)数学操作类:该类没有构造函数，方法均为静态的

(2)掌握内容

A:成员变量

Math.E：比任何其他值都更接近e（即自然对数的底数）的double值。

Math.PI：比任何其他值都更接近pi（即圆的周长与直径之比）的double值。

B:成员方法

abs(double a) 返回 double 值的绝对值。返回绝对值

ceil(double a) 返回最小的（最接近负无穷大）double 值，该值大于等于参数，并等于某个整数。

double floor(double a) 返回最大的（最接近正无穷大）double 值，该值小于等于参数，并等于某个整数。

max：返回两个值中较大的那个

min：返回两个值中较小的那个

long round(double a) 返回最接近参数的 long。

int round(float a) 返回最接近参数的 int。

double random() 返回带正号的 double 值，该值大于等于 0.0 且小于 1.0。

double pow(double a, double b) 返回第一个参数的第二个参数次幂的值。

double sqrt(double a) 返回正确舍入的 double 值的正平方根。

### 15、Random类的使用(重点)

(1)产生随机数的类

(2)掌握内容

A:构造方法

Random() 创建一个新的随机数生成器。

Random(long seed) 使用单个 long 种子创建一个新的随机数生成器。

B:成员方法

int nextInt() 返回下一个伪随机数，它是此随机数生成器的序列中均匀分布的 int 值。

int nextInt(int n) 返回一个伪随机数，它是取自此随机数生成器序列的、

在 0（包括）和指定值（不包括）之间均匀分布的 int 值。

### 16、Scanner类的使用

(1)可以获取从键盘的输入数据

(2)掌握内容

构造方法：

Scanner(InputStream source) 构造一个新的 Scanner，它生成的值是从指定的输入流扫描的。

如：Scanner sc = new Scanner(System.in);

方法摘要

sc.nextInt();获取整型数据

sc.nextLine();获取字符串数据

### 17、继承(重点)

(1)把很多类的相同特征和行为进行抽取，用一个类来描述。让多个类和这个类产生一个关系。

这样的话，多个类就可以省略很多代码。这个关系就是继承。java中用extends关键字表示。

(2)继承的体系结构

A:多个具体的对象，不断的向上抽取共享的内容，最终形成了一个体系。这个体系叫做继承体系。

B:继承体系的学习和使用原则

学习顶层的内容。因为他是整个体系的共性内容。

创建子类使用。也就是使用底层的具体对象。

(3)继承的特点:

A:java中只能单继承，没有多继承。

B:java可以有多重(层)继承。

(4)继承的好处：

继承的出现提高了代码的复用性。

继承的出现让类与类之间产生了关系，提供了多态的前提。

(5)子父类中的成员关系

A:成员变量

在子类方法中使用一个变量时：

首先，在方法的局部变量中找这个变量，有则使用。

否则，在本类中找成员变量，有则使用。

否则，在父类中找成员变量，有则使用。

否则，报错。

B:成员方法

用子类对象使用一个方法时。

首先，在子类中找这个方法，有则使用。

否则，在父类中找这个方法，有则使用。

否则，报错。

重写和重载的区别？

重载：在同一类中。方法名相同，参数列表不同。重载可以改变返回类型。

重写：在不同类中(子父类中)。

方法声明相同(返回类型，方法名，参数列表均相同)。

重写需要注意：

子类方法的访问权限要大于等于父类方法的访问权限。

静态只能重写静态。但是这种情况一般不会出现。

构造方法

子类的实例化过程

子类创建对象时，会先去创建父类的对象。

默认是去调用父类的无参构造方法。

\*\*\*子类构造方法中，第一行默认是super()

为什么子类中第一行会默认有super()

因为他继承父类的成员使用，使用前这些成员必须初始化，

而他们是父类的成员，所以，必须通过父类进行初始化。

所以，会先创建一个父类的对象。

当父类没有无参构造方法时

必须使用this或者super调用其他的构造方法。

(6)this和super的区别

this:代表本类对象的引用。

super:代表父类的存储空间。

### 18、final关键字(重点)

(1)最终的意思，可以用于修饰类，方法，变量。

(2)final修饰

修饰的类不能被继承。

修饰的方法不能被重写。

修饰的变量是一个常量。只能被赋值一次。

内部类只能访问被final修饰的局部变量。

### 19、抽象类(重点)

(1)多个类有相同的方法声明，但是方法体不一样。这个时候，我们考虑把方法声明进行抽取。

让子类继承后，自己去实现方法体。没有方法体的方法，我们需要用抽象标志下。

抽象的关键字是：abstract。

(2)抽象类：

该方法称为抽象方法，包含抽象方法的类就是抽象类。

(3)抽象类的特点：

A:抽象类和抽象方法都要用abstract进行修饰

B:抽象类不能被实例化

C:抽象类中不一定有抽象方法，但是，有抽象方法的类一定是抽象类。

(4)抽象类中数据的特点

A:成员变量

抽象类中可以有变量，也可以有常量。

B:成员方法

抽象类中可以有抽象方法，也可以有非抽象方法。

C:构造方法

抽象类是一个类，所以，它有构造方法。

虽然本身不能实例化。但是可以给子类实例化使用。

(5)抽象类中的问题

A:抽象类中是否有构造方法？能不能被实例化？如果不能，为什么有构造方法？

抽象类有构造方法。

抽象类不能被实例化。

抽象类中的构造方法供子类实例化调用。

B:抽象关键字abstract不可以和哪些关键字共存？

\*\*private：私有内容子类继承不到，所以，不能重写。

但是abstract修饰的方法，要求被重写。两者冲突。

\*\*final：final修饰的方法不能被重写。

而abstract修饰的方法，要求被重写。两者冲突。

\*\*static：假如一个抽象方法能通过static修饰，那么这个方法，就可以直接通过类名调用。

而抽象方法是没有方法体的，这样的调用无意义。所以，不能用static修饰。

C:抽象类中可不可以没有抽象方法？如果可以，这样的类有什么用吗？

抽象类可以没有抽象方法。

抽象类中没有抽象方法的作用，只是为了不让别的类建立该抽象类对象。这个在awt中有体现。

### 20、接口interface

(1)当一个类中的方法都是抽象的时候，java提供了另一种表示方式，叫接口。

用interface关键字表示。类与接口关系用implements表示。

(2)接口的成员特点

A:成员变量

是常量，默认修饰 public static final

B:成员方法

都是抽象的，默认修饰 public abstract

(3)关系

A:类与类的关系

是继承关系。类与类只能单继承，可以多重继承。

B:类和接口的关系

是实现关系。类可以多实现接口。

类在继承一个类的同时，可以实现多个接口。

C:接口和接口的关系

是继承关系。接口可以多继承接口。

(4)接口的特点

A:是对外暴露的规则

B:是功能的扩展

C:接口的出现降低耦合性。

耦合(类与类之间的关系)

内聚(类完成功能的能力)

编程规范：低耦合，高内聚。

D:接口可以多实现。如：CPU和主板、笔记本的USB插口、插座

(5)接口和抽象类的区别

A：抽象类只能被单继承

接口可以多实现,接口的出现避免了多继承的局限性。

B：抽象类中的数据特点：

成员变量：可以是变量，也可以是常量

成员方法：可以是抽象方法，也可以是非抽象方法

构造方法：有构造方法

接口中的数据特点：

成员变量：是常量。默认修饰 public static final

成员方法：都是抽象方法。都有默认修饰 public abstract

构造方法：没有构造方法

C：抽象类中定义的是继承体系中的共性功能。

接口中定义的是继承体系中的扩展功能。

D：抽象类被继承是"is a"关系:xx是yy的一种

接口被实现是"like a"关系:xx像yy的一种

### 21、多态

(1)同一个对象，在程序不同时刻的多种运行状态。举例：动物，狗是狗，狗是动物。水(气态，液态，固态)

(2)多态前提

A:存在着继承或者实现关系

B:有方法的重写

C:父类(接口)引用指向子类(实现)对象

(3)多态的好处和弊端：

好处：多态的存在提高了程序的扩展性和后期可维护性

弊端：只能访问父类中已有的功能，运行的是后期子类的功能内容。不能使用子类中定义的特有功能。

(4)多态中对象调用成员的特点

Fu f = new Zi();

A:成员变量

编译看左边，运行看左边

B:成员方法

编译看左边，运行看右边

C:静态方法

编译看左边，运行看左边

(5)多态的思想

指挥同一批对象做事情。举例：带兵打仗，下课等。

### 22、instanceof关键字

A:用于判断某个对象是否是某种类型。

B:格式

对象名 instanceof 子类(实现)名

### 23、Object类

(1)是所有类的根类，超类。

java中提供的类以及我们自定义的类都直接或者间接的继承自Object类。

(2)Object类中的方法

A:void finalize()

当垃圾回收器确定不存在对该对象的更多引用时，由对象的垃圾回收器调用此方法。

B:Class getClass()

获取对象的字节码文件的描述类，后面再讲反射的时候还会在说这个类。

String name = s.getClass().getName();

C:int hashCode()

获取对象的哈希值。其实就是对象的内存地址值十进制表示

D:String toString()

返回对象的字符串表示。

表示格式：

getClass().getName()+"@"+Integer.toHexString(hashCode());

一般我们输出对象名的时候，其实底层调用的就是该对象的toString()方法。

这种返回没有意义，所以，我们会重写这个方法，显示类的成员变量信息。

E:boolean equals(Object obj)

用于比较两个对象的地址值是否相同。

我们获取对象后，比较它的地址值意义不大。所以也会对这个方法进行重写。

重写要完成什么功能，是根据需求定的。

(3)==和equals的用法：

A:==怎么用？

\*\*可以用于比较基本数据类型，比较的就是基本数据类型的值是否相等。

\*\*可以用于比较引用数据类型，比较的是对象的地址值是否相等。

B:equals怎么用？

equals只能用于比较引用数据类型的。

\*\*Object提供的equals是用于比较对象地址值是否相同。

\*\*自定义类中，如果重写了equals方法，那么就是按照你自己的需求来比较的。

### 24、package关键字

(1)包：其实就是文件夹。用于区分不同包下相同的类名。

(2)好处：

A：对类文件进行分类管理。

B：给类提供了多层命名空间

aaa.Demo

bbb.Demo

C：写在程序文件的第一行。

D：包也是一种封装形式。

### 25、import关键字

(1)导入包的关键字

(2)格式：

import 包名;

(3)注意：

A:一个程序文件中只有一个package，可以有多个import。

B:用来导包中的类，不导入包中的包。

C:通常写import mypack.Demo，明确自己使用的类。

(4)关键字的顺序

类，包，导包这些关键的顺序。

包 -- > 到包 -- > 类

### 26、不同修饰符可以修饰哪些内容

本类中 同一个包中 ，不同包中的子类中 ，不同包中

private OK

默认 OK Ok

protected OK Ok OK

public OK Ok OK Ok

一般格式：

成员变量：

权限修饰符+static/final+数据类型+成员变量名

public static final int NUM = 10;

成员方法：

权限修饰符+static/final/abstract+返回类型+方法名

### 27、内部类(次重点)

(1)把一个类定义在某个类中的，这个类就被称为内部类，内置类，嵌套类。

(2)访问特点：

A:内部类可以直接访问外部类中的成员，因为内部类持有外部类的引用，

格式为：外部类名.this

B:外部类要想访问内部类的成员，必须创建对象访问。

(3)内部类的访问格式：

A:当内部类定义在外部类的成员位置，而且非私有，则可以在其他外部类中直接建立内部类对象

格式：外部类名.内部类名 变量名 = new 外部类对象.内部类对象

如：Outer.Inner in = new Outer().new Inner()

B:当内部类在外部类成员位置，且被static修饰时

\*\*外部其他类可直接访问静态内部类的非静态成员

格式：new 外部类名.内部类名().内部类成员

如：new Outer.Inner().function();

\*\*外部其他类可直接访问静态内部类的静态成员

格式：new 外部类名.内部类名.内部类成员

如：new Outer.Inner.function();

(4)什么使用时候内部类呢？

假如有A类和B类，A类想直接访问B类的成员，B类访问A类成员的时候，

需要创建A类对象进行访问，这个时候，就可以把A类定义为B类的内部类。

(5)内部类的位置

A:成员位置

\*\*可以被private修饰(Body，Heart)

\*\*可以被static修饰。(它访问的外部类的成员必须是静态的)

B:局部位置

\*\*可以直接访问外部类中的成员，因为还持有外部类的持用

也可以直接访问局部成员，但是局部成员要用final修饰。

注意：局部内部类不能用private和static修饰

(6)通过class文件我们就可以区分是否带有内部类，以及内部类的位置

Outer$Inner:成员内部类

Outer$1Inner:局部内部类

### 28、匿名内部类(局部内部类的简写) (重点)

(1)前提：继承一个类或者实现一个接口

(注意不要弄混匿名内部类的前提和多态的前提)

(2)格式：

new 父类名或者接口名(){

重写父类方法或者实现接口中的方法。

也可以自定义其他方法。

};

(3)什么时候定义匿名内部类？

匿名内部类只是为了简化书写，匿名内部类有局限，通常定义匿名内部类时，该类方法不超过3个

(4)匿名内部类的好处和弊端：

好处：简化代码书写

弊端：

不能直接调用自己的特有方法

不能执行强转换动作

如果该类里面方法较多，不允许使用匿名内部类

### 29、模板设计模式：

在定义功能时，功能的一部分是确定的，有一部分是不确定的，而且确定的部分在使用不确定的部分，

可将不确定的部分暴露出去，由该类的子类去完成。

如：求一段程序的运行时间例子。

### 30、异常

(1)程序运行过程中的不正常现象就叫异常。

(2)导致程序运行不正常的现象有很多，所以，就有很多的异常对象。

而这些异常对象存在着共性的内容，所以，可以不断的进行抽取。最终形成了异常的体系结构。

异常体系的根类是:Throwable

Throwable：

|--Error:重大的问题，我们处理不了。也不需要编写代码处理。比如说内存溢出。

|--Exception:一般性的错误，是需要我们编写代码进行处理的。

|--RuntimeException:运行时异常，这个我们也不需要处理。

其实就是为了让他在运行时出问题，然后我们回来修改代码。

(3)异常的分类

异常有两种：

编译时被检测异常：

该异常在编译时，如果没有处理(没有抛也没有try)，编译失败。

该异常被标识，代表这可以被处理。

运行时异常(编译时不检测)

在编译时，不需要处理，编译器不检查。

该异常的发生，建议不处理，让程序停止。需要对代码进行修正。

(4)异常体系的特点：

异常体系中的所有类及其子类对象都具备可抛性。也就是说可以被throw和throws关键字所操作。

(5)main方法是如何处理异常的。

A:在main里面编写代码进行处理

B:交给jvm自己进行处理。采用的是jvm的默认处理方式。

其实就是相当于调用了异常对象的printStackTrace()方法。

(6)Throwable类的学习

getMessage():获取异常信息，返回字符串。

toString():获取异常类名和异常信息，返回字符串。

printStackTrace():获取异常类名和异常信息，以及异常出现在程序中的位置。返回值void。

(7)异常的处理·

A:try...catch...finally

基本格式：

try{

可能出现异常的代码

}

catch(异常对象){

异常处理代码

}

finally{

释放资源

}

变形格式：

try...catch

try...catch...catch...

try...catch...catch...finally

\*\*多个异常同时被捕获的时候，记住一个原则：

先逮小的，再逮大的。

\*\*finally:永远被执行，除非退出jvm。System.exit(0);

面试题2个。

final,finally,finalize区别。

final是最终的意思。它可以用于修饰类，成员变量，成员方法。

它修饰的类不能被继承，它修饰的变量时常量，它修饰的方法不能被重写。

finally:是异常处理里面的关键字。

它其中的代码永远被执行。特殊情况：在执行它之前jvm退出。System.exit(0);

finalize:是Object类中的一个方法。

它是于垃圾回收器调用的方式。

假如catch中有return语句， finally里中的代码会执行吗？是在return前，还是在return后呢？

会，在return前执行finally里面的代码。

(8)Exception和RuntimeException的区别

A:Exception:一般性的错误，是需要我们编写代码进行处理的。

B:RuntimeException:运行时异常，这个我们也不需要处理。

其实就是为了让他在运行时出问题，然后我们回来修改代码。

在用throws抛出一个的时候，如果这个异常是属于RuntimeException的体系的时候，

我们在调用的地方可以不用处理。(RuntimeException和RuntimeException的子类)

在用throws抛出一个的时候，如果这个异常是属于Exception的体系的时候，

我们在调用的地方必须进行处理或者继续抛出。

(9)自定义异常

定义类继承Exception或者RuntimeException

1,为了让该自定义类具备可抛性。

2，让该类具备操作异常的共性方法。

class MyExcepiton extends Exception{

MyExcepiton(){}

MyExcepiton(String message){

super(message);

}

}

class MyException extends RuntimeException{

MyExcepiton(){}

MyExcepiton(String message){

super(message);

}

}

(10)throws和throw的区别

A：有throws的时候可以没有throw。

有throw的时候，如果throw抛的异常是Exception体系，那么必须有throws在方法上声明。

B：throws用于方法的声明上，其后跟的是异常类名，后面可以跟多个异常类，之间用逗号隔开

throw用于方法体中，其后跟的是一个异常对象名

## 多线程

### 1、进程和线程：

进程：正在进行的程序。每一个进程执行都有一个执行顺序，该顺序是一个执行路径，或者叫一个控制单元。

线程：进程内部的一条执行路径或者一个控制单元。

两者的区别：

一个进程至少有一个线程

进程在执行过程中拥有独立的内存单元，而多个线程共享内存；

jvm多线程的启动是多线程吗？

java的虚拟机jvm启动的是单线程，就有发生内存泄露的可能，而我们使用java程序没出现这样的问题，

也就是jvm启动至少有两个线程，一个执行java程序，一个执行垃圾回收。所以是多线程。

### 2、多线程的优势

解决了多部分同时运行的问题，提高效率

### 3、线程的弊端

线程太多会导致效率的降低，因为线程的执行依靠的是CPU的来回切换。

### 4、什么叫多线程

一个进程中有多个线程，称为多线程。

### 5、实现多线程的方法

实现多线程可以通过继承Thread类和实现Runnable接口。

(1)继承Thread

定义一个类继承Thread类

复写Thread类中的public void run()方法，将线程的任务代码封装到run方法中

直接创建Thread的子类对象，创建线程

调用start()方法，开启线程(调用线程的任务run方法)

//另外可以通过Thread的getName()获取线程的名称。

(2)实现Runnable接口；

定义一个类，实现Runnable接口；

覆盖接口的public void run()的方法，将线程的任务代码封装到run方法中；

创建Runnable接口的子类对象

将Runnabl接口的子类对象作为参数传递给Thread类的构造函数，创建Thread类对象

（原因：线程的任务都封装在Runnable接口子类对象的run方法中。

所以要在线程对象创建时就必须明确要运行的任务）。

调用start()方法，启动线程。

两种方法区别：

(1)实现Runnable接口避免了单继承的局限性

(2)继承Thread类线程代码存放在Thread子类的run方法中

实现Runnable接口线程代码存放在接口的子类的run方法中；

在定义线程时，建议使用实现Runnable接口，因为几乎所有多线程都可以使用这种方式实现

### 6、创建线程是为什么要复写run方法？

Thread类用于描述线程。Thread类定义了一个功能，用于存储线程要运行的代码，该存储功能就是run方法。

### 7、start()和run方法有什么区别？

调用start方法方可启动线程，而run方法只是thread的一个普通方法，调用run方法不能实现多线程；

Start()方法:

start方法用来启动线程,实现了多线程运行,这时无需等待run方法体代码执行完毕而直接继续执行下面的

代码。通过调用Thread类的start()方法来启动一个线程,这时此线程处于就绪(可运行)状态，并没有运行，

一旦得到cpu时间片(执行权),就开始执行run()方法,这里方法run()称为线程体，

它包含了要执行的这个线程的内容，Run方法运行结束,此线程随即终止。

Run()方法:

run()方法只是Thread类的一个普通方法,如果直接调用Run方法,程序中依然只有主线程这一个线程,

其程序执行路径还是只有一条，还是要等待run方法体执行完毕后才可继续执行下面的代码，

这样就没有达到多线程的目的。

### 8、线程的几种状态：

1、新建：new一个Thread对象或者其子类对象就是创建一个线程，当一个线程对象被创建，但是没有开启，这个时候，

只是对象线程对象开辟了内存空间和初始化数据。

2、就绪：新建的对象调用start方法，就开启了线程，线程就到了就绪状态。

在这个状态的线程对象，具有执行资格，没有执行权。

3、运行：当线程对象获取到了CPU的资源。

在这个状态的线程对象，既有执行资格，也有执行权。

4、阻塞：运行过程中的线程由于某些原因(比如wait,sleep)，释放了执行资格和执行权。

当然，他们可以回到运行状态。只不过，不是直接回到。

而是先回到就绪状态。

5、死亡：当线程对象调用的run方法结束，或者直接调用stop方法，就让线程对象死亡，在内存中变成了垃圾。

### 9、sleep()和wait()的区别：

(1)这两个方法来自不同的类，sleep()来自Thread类，和wait()来自Object类。

(2)sleep是Thread的静态类方法，谁调用的谁去睡觉，即使在a线程里调用了b的sleep方法，实际上还是a去睡觉，

要让b线程睡觉要在b的代码中调用sleep。而wait()是Object类的非静态方法

(3)sleep()释放资源不释放锁，而wait()释放资源释放锁；

(4)使用范围：wait,notify和notifyAll只能在同步控制方法或者同步控制块里面使用,而sleep可以在任何地方使用

### 10、多线程安全问题：

(1)原因:当程序的多条语句在操作线程共享数据时(如买票例子中的票就是共享资源)，由于线程的随机性导致

一个线程对多条语句，执行了一部分还没执行完，另一个线程抢夺到cpu执行权参与进来执行，

此时就导致共享数据发生错误。比如买票例子中打印重票和错票的情况。

(2)解决方法:对多条操作共享数据的语句进行同步，一个线程在执行过程中其他线程不可以参与进来

### 11、Java中多线程同步是什么？

同步是用来解决多线程的安全问题的，在多线程中，同步能控制对共享数据的访问。如果没有同步，当一个线程在

修改一个共享数据时，而另外一个线程正在使用或者更新同一个共享数据，这样容易导致程序出现错误的结果。

### 12、什么是锁?锁的作用是什么?

锁就是对象

锁的作用是保证线程同步，解决线程安全问题。

持有锁的线程可以在同步中执行，没有锁的线程即使获得cpu执行权，也进不去。

### 13、同步的前提:

(1)必须保证有两个以上线程

(2)必须是多个线程使用同一个锁，即多条语句在操作线程共享数据

(3)必须保证同步中只有一个线程在运行

### 14、同步的好处和弊端

好处：同步解决了多线程的安全问题

弊端：多线程都需要判断锁，比较消耗资源

### 15、同步的两种表现形式：

(1)同步代码块:

可以指定需要获取哪个对象的同步锁,使用synchronized的代码块同样需要锁,但他的锁可以是任意对象

考虑到安全问题，一般还是使用同一个对象，相对来说效率较高。

注意：

\*\*虽然同步代码快的锁可以使任何对象，但是在进行多线程通信使用同步代码快时，

必须保证同步代码快的锁的对象和，否则会报错。

\*\*同步函数的锁是this，也要保证同步函数的锁的对象和调用wait、notify和notifyAll的对象是

同一个对象，也就是都是this锁代表的对象。

格式：

synchronized(对象){

需同步的代码;

}

(2)同步函数

同步方法是指进入该方法时需要获取this对象的同步锁，在方法上使用synchronized关键字，

使用this对象作为锁，也就是使用了当前对象，因为锁住了方法，所以相对于代码块来说效率相对较低。

注:静态同步函数的锁是该方法所在的类的字节码文件对象，即类名.class文件

格式：

修饰词 synchronized 返回值类型 函数名(参数列表){

需同步的代码;

}

在jdk1.5后，用lock锁取代了synchronized，个人理解也就是对同步代码块做了修改，

并没有提供对同步方法的修改，主要还是效率问题吧。

### 16、多线程的单例设计模式：保证某个类中内存中只有一个对象

(1)饿汉式:

class Single{

private Single(){}//将构造函数私有化，不让别的类建立该类对象

private static final Single s=new Single();//自己建立一个对象

public static Single getInstance(){//提供一个公共访问方式

return s;

}

}

(2)懒汉式：

饿汉式和懒汉式的区别：

饿汉式是类一加载进内存就创建好了对象；

懒汉式则是类加载进内存的时候，对象还没有存在，只有调用了getInstance()方法时，对象才开始创建。

懒汉式是延迟加载，如果多个线程同时操作懒汉式时就有可能出现线程安全问题，解决线程安全问题

可以加同步来解决。但是加了同步之后，每一次都要比较锁，效率就变慢了，

所以可以加双重判断来提高程序效率。

如将上述懒汉式的Instance函数改成同步：

public static Single getInstance(){

if(s==null){

synchronized(Single.class){

if(s==null)

s=new Single();

}

}

return s;

}

### 17、死锁

两个线程对两个同步对象具有循环依赖时，就会发生死锁。即同步嵌套同步，而锁却不同。

### 18、wait()、sleep()、notify()、notifyAll()

wait():使一个线程处于等待状态，并且释放所持有的对象的lock。

sleep():使一个正在运行的线程处于睡眠状态，是一个静态方法，调用此方法要捕捉InterruptedException异常。

notify():唤醒一个处于等待状态的线程，注意的是在调用此方法的时候，并不能确切的唤醒某一个等待状态的线程，

而是由JVM确定唤醒哪个线程(一般是最先开始等待的线程)，而且不是按优先级。

Allnotity():唤醒所有处入等待状态的线程，注意并不是给所有唤醒线程一个对象的锁，而是让它们竞争。

为什么wait()、notify()、notifyAll()这些用来操作线程的方法定义在Object类中？

(1)这些方法只存在于同步中；

(2)使用这些方法时必须要指定所属的锁，即被哪个锁调用这些方法；

(3)而锁可以是任意对象，所以任意对象调用的方法就定义在Object中。

### 19、多线程间通讯：

多线程间通讯就是多个线程在操作同一资源,但是操作的动作不同.

(1)为什么要通信

多线程并发执行的时候, 如果需要指定线程等待或者唤醒指定线程, 那么就需要通信.比如生产者消费者的问题，

生产一个消费一个,生产的时候需要负责消费的进程等待,生产一个后完成后需要唤醒负责消费的线程,

同时让自己处于等待，消费的时候负责消费的线程被唤醒，消费完生产的产品后又将等待的生产线程唤醒，

然后使自己线程处于等待。这样来回通信，以达到生产一个消费一个的目的。

(2)怎么通信

在同步代码块中, 使用锁对象的wait()方法可以让当前线程等待, 直到有其他线程唤醒为止.

使用锁对象的notify()方法可以唤醒一个等待的线程，或者notifyAll唤醒所有等待的线程.

多线程间通信用sleep很难实现，睡眠时间很难把握。

### 20、Lock和Condition

实现提供比synchronized方法和语句可获得的更广泛的锁的操作，可支持多个相关的Condition对象

Lock是个接口

锁是控制多个线程对共享数据进行访问的工具。

JDK1.5中提供了多线程升级的解决方案：

将同步synchonized替换成了显示的Lock操作，将Object中的wait、notify、notifyAll替换成了Condition对象。

该对象可以Lock锁进行获取

Lock的方法摘要：

void lock() 获取锁。

Condition newCondition() 返回绑定到此 Lock 实例的新 Condition 实例。

void unlock() 释放锁。

Condition方法摘要：

void await() 造成当前线程在接到信号或被中断之前一直处于等待状态。

void signal() 唤醒一个等待线程。

void signalAll() 唤醒所有等待线程。

### 21、停止线程：

stop方法已经过时，如何停止线程？

停止线程的方法只有一种，就是run方法结束。如何让run方法结束呢？

开启多线程运行，运行代码通常是循环体，只要控制住循环，就可以让run方法结束，也就是结束线程。

特殊情况：当线程属于冻结状态，就不会读取循环控制标记，则线程就不会结束。

为解决该特殊情况，可引入Thread类中的Interrupt方法结束线程的冻结状态；

当没有指定的方式让冻结线程恢复到运行状态时，需要对冻结进行清除，强制让线程恢复到运行状态

interrupt: void interrupt() 中断线程:

中断状态将被清除，它还将收到一个 InterruptedException

### 22、守护线程(后台线程)

setDaemon(boolean on):将该线程标记为守护线程或者用户线程。

当主线程结束，守护线程自动结束，比如圣斗士星矢里面的守护雅典娜，

在多线程里面主线程就是雅典娜，守护线程就是圣斗士，主线程结束了，

守护线程则自动结束。

当正在运行的线程都是守护线程时，java虚拟机jvm退出；所以该方法必须在启动线程前调用；

守护线程的特点：

守护线程开启后和前台线程共同抢夺cpu的执行权，开启、运行两者都没区别，

但结束时有区别，当所有前台线程都结束后，守护线程会自动结束。

### 23、多线程join方法：

void join() 等待该线程终止。

void join(long millis) 等待该线程终止的时间最长为 millis 毫秒。

throws InterruptedException

特点：当A线程执行到B线程的join方法时，A就会等待B线程都执行完，A才会执行

作用: join可以用来临时加入线程执行；

### 24、多线程优先级：yield()方法

yield():暂停当前正在执行的线程对象，并执行其他线程

setPriority(int newPriority):更改线程优先级

int getPriority() 返回线程的优先级。

String toString() 返回该线程的字符串表示形式，包括线程名称、优先级和线程组

(1)MAX\_PRIORITY:最高优先级(10级)

(1)Min\_PRIORITY:最低优先级(1级)

(1)Morm\_PRIORITY:默认优先级(5级)

### 25、什么是ThreadLocal类,怎么使用它？

ThreadLocal类提供了线程局部 (thread-local) 变量。是一个线程级别的局部变量，并非“本地线程”。

ThreadLocal 为每个使用该变量的线程,提供了一个独立的变量副本，每个线程修改副本时不影响其它线程对象的副本

下面是线程局部变量(ThreadLocal variables)的关键点：

一个线程局部变量(ThreadLocal variables)为每个线程方便地提供了一个单独的变量。

ThreadLocal 实例通常作为静态的私有的(private static)字段出现在一个类中，这个类用来关联一个线程。

当多个线程访问 ThreadLocal 实例时，每个线程维护 ThreadLocal 提供的独立的变量副本。

常用的使用可在 DAO 模式中见到，当 DAO 类作为一个单例类时，

数据库链接(connection)被每一个线程独立的维护，互不影响。(基于线程的单例)

### 26、什么时候抛出InvalidMonitorStateException异常?为什么？

调用 wait ()/notify ()/notifyAll ()中的任何一个方法时，如果当前线程没有获得该对象的锁，

那么就会抛出 IllegalMonitorStateException 的异常

也就是说程序在没有执行对象的任何同步块或者同步方法时，

仍然尝试调用 wait ()/notify ()/notifyAll ()时。由于该异常是 RuntimeExcpetion 的子类，

所以该异常不一定要捕获(尽管你可以捕获只要你愿意

作为 RuntimeException，此类异常不会在 wait (),notify (),notifyAll ()的方法签名提及。

### 27、在静态方法上使用同步时会发生什么事？

同步静态方法时会获取该类的“Class”对象，所以当一个线程进入同步的静态方法中时，

线程监视器获取类本身的对象锁，其它线程不能进入这个类的任何静态同步方法。

它不像实例方法，因为多个线程可以同时访问不同实例同步实例方法。

### 28、当一个同步方法已经执行，线程能够调用对象上的非同步实例方法吗？

可以，一个非同步方法总是可以被调用而不会有任何问题。

实际上，Java 没有为非同步方法做任何检查，锁对象仅仅在同步方法或者同步代码块中检查。

如果一个方法没有声明为同步，即使你在使用共享数据Java照样会调用，而不会做检查是否安全，

所以在这种情况下要特别小心。一个方法是否声明为同步取决于临界区访问(critial section access)，

如果方法不访问临界区(共享资源或者数据结构)就没必要声明为同步的。

### 29、在一个对象上两个线程可以调用两个不同的同步实例方法吗？

不能，因为一个对象已经同步了实例方法，线程获取了对象的对象锁。

所以只有执行完该方法释放对象锁后才能执行其它同步方法。

30、什么是线程饿死，什么是活锁？

线程饿死和活锁虽然不像死锁一样是常见的问题，但是对于并发编程的设计者来说就像一次邂逅一样。

当所有线程阻塞，或者由于需要的资源无效而不能处理，不存在非阻塞线程使资源可用。

JavaAPI 中线程活锁可能发生在以下情形：

当所有线程在程序中执行 Object.wait (0)，参数为 0 的 wait 方法。

程序将发生活锁直到在相应的对象上有线程调用 Object.notify ()或者 Object.notifyAll ()。

当所有线程卡在无限循环中。

四、集合框架

1：String类：字符串(重点)

(1)多个字符组成的一个序列，叫字符串。

生活中很多数据的描述都采用的是字符串的。而且我们还会对其进行操作。

所以，java就提供了这样的一个类供我们使用。

(2)创建字符串对象

复制代码

1 A:String():无参构造

2 \*\*举例：

3 String s = new String();

4 s = "hello";

5 sop(s);

6 B:String(byte[] bys):传一个字节数组作为参数 \*\*\*\*\*

7 \*\*举例

8 byte[] bys = {97,98,99,100,101};

9 String s = new String(bys);

10 sop(s);

11 C:String(byte[] bys,int index,int length):把字节数组的一部分转换成一个字符串 \*\*\*\*\*

12 \*\*举例

13 byte[] bys = {97,98,99,100,101};

14 String s = new String(bys,1,2);

15 sop(s);

16 D:String(char[] chs):传一个字符数组作为参数 \*\*\*\*\*

17 \*\*举例

18 char[] chs = {'a','b','c','d','e'};

19 String s = new String(chs);

20 sop(s);

21 E:String(char[] chs,int index,int length):把字符数组的一部分转换成一个字符串 \*\*\*\*\*

22 \*\*举例

23 char[] chs = {'a','b','c','d','e'};

24 String s = new String(chs,1,2);

25 sop(s);

26 F:String(String str):把一个字符串传递过来作为参数

27 char[] chs = {'a','b','c','d','e'};

28 String ss = new String(s);

29 sop(ss);

30 G:直接把字符串常量赋值给字符串引用对象(最常用) \*\*\*\*\*

31 \*\*举例

32 String s = "hello";

33 sop(s);

复制代码

(3)面试题

A:请问String s = new String("hello");创建了几个对象。

两个。一个"hello"字符串对象，在方法区的常量池；一个s对象，在栈内存。

B:请写出下面的结果

String s1 = new String("abc");

Strign s2 = new String("abc");

String s3 = "abc";

String s4 = "abc";

sop(s1==s2); //false

sop(s1==s3); //false

sop(s3==s4); //true

C:字符串对象一旦被创建就不能被改变。

指的是字符串常量值不改变。

(4)字符串中各种功能的方法

A:判断

\*\*\*\* boolean equals(Object anObject):判断两个字符串的内容是否相同，复写了Object的方法

\*\*\*\* boolean equalsIgnoreCase(String anotherString):判断两个字符串的内容是否相同，

不区分大小写

\*\*\*\* boolean contains(String s):判断一个字符串中是否包含另一个字符串

注意：判断字符串是否包含特殊字符.直接表示为str.contains(".")

boolean endsWith(String suffix):测试此字符串是否以指定的后缀结束

boolean startsWith(String suffix):测试此字符串是否以指定的前缀开始

boolean isEmpty():测试字符串是否为空

B:获取

\*\*\*\*\* int length():返回此字符串的长度

\*\*\*\*\* char charAt(int index):返回指定索引处的 char值

\*\*\*\*\* int indexOf(int ch):返回指定字符在此字符串中第一次出现处的索引。

int indexOf(int ch, int fromIndex):返回在此字符串中第一次出现指定字符处的索引，

从指定的索引开始搜索。

int indexOf(String str):返回指定子字符串在此字符串中第一次出现处的索引。

int indexOf(String str, int fromIndex):返回指定子字符串在此字符串中第一次

出现处的索引，从指定的索引开始。

\*\*\* int lastIndexOf(int ch)：返回指定字符在此字符串中最后一次出现处的索引。

int lastIndexOf(int ch, int fromIndex)

返回指定字符在此字符串中最后一次出现处的索引,从指定的索引处开始进行反向搜索。

int lastIndexOf(String str)

返回指定子字符串在此字符串中最右边出现处的索引。

int lastIndexOf(String str, int fromIndex)

返回指定子字符串在此字符串中最后一次出现处的索引，从指定的索引开始反向搜索。

\*\*\*\*\* String substring(int beginIndex) (注意：该方法substring的String是小写！！！)

返回一个新的字符串，它是此字符串的一个子字符串。

String substring(int beginIndex, int endIndex) (注意该方法的String是小写！！！)

返回一个新字符串，它是此字符串的一个子字符串,包含头不包含尾。

C:转换

\*\*\*\*\* byte[] getBytes()：(很常用！)从字符串到字节数组的方法

void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)

将字符从此字符串复制到目标字符数组。

\*\*\*\*\* char[] toCharArray()：(很常用！)从字符串到字符数组的方法

\*\*\*\* static String copyValueOf(char[] data)

返回指定数组中表示该字符序列的 String。

static String copyValueOf(char[] data, int offset, int count)

返回指定数组中表示该字符序列的 String。

\*\*\*\*\* static String valueOf(数据类型):把该数据类型的数据转换成字符串。

\*\*\* String toLowerCase()：把字符串转换成小写

String toUpperCase()：把字符串转换成大写

\*\*\* 字符串的连接

String concat(String str):将指定字符串连接到此字符串的结尾。

D:替换

String replace(char oldChar, char newChar):用新字符替换旧字符(替换所有)

String replace(String target, String replacement):用新的子串换旧串

E:分割

String[] split(String regex)：根据指定的字符串把一个字符串分割成一个字符串数组

F:

String trim():去除字符串的前后空格

G:

int compareTo(String anotherString)

按字典顺序比较两个字符串。

int compareToIgnoreCase(String str)

按字典顺序比较两个字符串，不考虑大小写。

(5)练习

1:模拟登录,给三次机会,并提示还有几次.

默认的用户名和密码为admin。 区分大小写。

自己从键盘输入用户名和密码。

2:给定一个字符串统计,统计大写字母,小写字母,数字出现的个数.

\*\*\*注意:不包括特殊字符

从键盘输入一个不包含特殊字符的字符串(只有26个字母和0-9组成)。

3:给定一个字符串,把它变成首字母大写,其他字母小写的字符串.

从键盘输入一个字符串，全部26个字母组成的。

4:子串在整串中出现的次数。

也就是说:获取一个字符串中,指定的字串在该字符串中出现的次数.

例如:

"nbasdnbafllgnbahjnbakqqqqlnba" 在这个字符串中，多有个nba.

5:对字符串中字符进行自然顺序排序。

"basckd"-->"abcdks"

先留做思考内容：

6:两个字符串的最大相同子串。

两个字符串的最大相同子串。

比如:

"sadabcdfghjkl"

werabcdtyu"

2：StringBuffer

(1)字符串的缓冲区，是一个容器。

(2)它和String的区别

它是缓冲区可变长度的。

(3)构造方法

StringBuffer() 构造一个其中不带字符的字符串缓冲区，初始容量为 16 个字符。

StringBuffer(int num) 构造一个不带字符，但具有指定初始容量的字符串缓冲区。

StringBuffer(String str) 构造一个字符串缓冲区，并将其内容初始化为指定的字符串内容。

(4)常用方法

A:增加数据

\*\*append ：添加各种类型的数据

\*\*insert : 在容器指定位置插入各种类型的数据。

B:删除数据

\*\*deleteCharAt : 删除指定位置的字符

\*\*delete 还可以用于清空StringBuffer的缓冲区

C:替换

\*\*replace

D:获取

\*\*charAt

E:长度和容量

\*\*length() 元素的个数

\*\*capacity 元素的理论值

F:获取元素的位置

\*\*indexOf

\*\*lastIndexOf

G:截取

\*\*substring(int start)

\*\*substring(int start,int end)

H:反转

\*\*reverse

(5)字符串和StringBuffer的转换

String-->StringBuffer通过构造:

如:StringBuffer sb = new StringBuffer(String str)

StringBuffer--String通过toString方法

如:StringBuffer sb = new StringBuffer();

sb.toString();

3：StringBuilder

和StringBuffer的功能是一样的，但是有区别：

StringBuffer(JDK1.0)是线程安全的。

StringBuilder(JDK1.5)不保证线程安全。

一般来说，我们写的程序都是单线程的，所以，用StringBuilder，效率高。

JDK版本的升级原则：

A:提高效率

B:提高安全性

C:简化书写

4：基本数据类型的对象包装类

(1)为了更方便的操作每个基本数据类型，java对其提供了很多的属性和方法供我们使用。

(2)用途：

\*\*将基本数据类型封装成对象的好处在于可以在对象中定义更多的功能操作该数据。

\*\*常用的操作之一：用于基本数据类型与字符串之间的转换。

A:方便操作

B:用于和字符串进行相互转换

(3)基本数据类型和对象类型的对应

byte Byte

short Short

int Integer

long Long

float Float

double Double

boolean Boolean

char Character

(4)构造方法

字段摘要：

static int MAX\_VALUE 值为 2^31-1 的常量，它表示 int 类型能够表示的最大值

static int MIN\_VALUE 值为 -2^31 的常量，它表示 int 类型能够表示的最小值

static Class<Integer> TYPE 表示基本类型int的Class 实例

Integer(int value) 构造一个新分配的Integer对象，它表示指定的int值。

Inreger(String s) 注意：s必须是纯数字的字符串。否则会有异常NumberFormatException

(5)几个常用的方法

Integer.toBinaryString();

以二进制（基数 2）无符号整数形式返回一个整数参数的字符串表示形式。

Integer.toOctalString();

以八进制（基数 8）无符号整数形式返回一个整数参数的字符串表示形式。

Integer.toHexString();

以十六进制（基数 16）无符号整数形式返回一个整数参数的字符串表示形式。

static int Integer.parseInt(String s) 将字符串参数作为有符号的十进制整数进行解析,

字符串必须是int型范围内的数字字符串

static int Integer.parseInt(String s,int basic)

使用第二个参数指定的基数,将字符串参数解析为有符号的整数.

字符串必须是int型范围内的数字字符串

short shortValue() 以short类型返回该Integer的值。

int intValue() 以int类型返回该Integer的值。

static Integer valueOf(int num) 返回一个表示指定的 int 值的 Integer 实例。

static Integer valueOf(String s) 返回保存指定的String的值的Integer对象。

static Integer valueOf(String s, int radix)

返回一个Integer对象，该对象中保存了用第二个参数提供的基数进行

解析时从指定的String中提取的值。

(6)类型转换

复制代码

1 int -- Integer

2 int num = 20;

3 A:Integer i = new Integer(num);

4 B:Integer i = Integer.valueOf(num);

5 Integer -- int

6 Integer i = new Integer(20);

7 A:int num = i.intValue();

8

9 int -- String

10 int num = 20;

11 A:String s = String.valueOf(num);

12 B:String s = ""+num;

13 C:String s = Integer.toString(num);

14 String -- int

15 String s = "20";

16 A:int num = Integer.parseInt(s);

17 B:Integer i = new Integer(s);或者Integer i = Integer.valueOf(s);

18 int num = i.intValue();

复制代码

6、集合框架：

(1)为什么出现集合类？

面向对象对事物的体现都是以对象的形式，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.

(2)数组和集合都是容器，两者有何不同？

\*\*数组长度固定，而集合长度是可变的

\*\*数组值可以存储对象，还可以存储基本数据类型;而集合只能存储对象

\*\*数组存储数据类型是固定的，而集合存储的数据类型不固定

(3)集合类的特点：

集合只能存储对象

集合的长度是可变的

集合可以存储不同类型的对象

(4)集合类框架(重要！！！要分清几种容器间的区别)：

\*\*Collection:顶层接口

|--->List:列表，元素是有序的(元素带角标索引)，可以有重复元素,可以有null元素。

|--->ArrayList(JDK1.2):底层的数据结构是数组数据结构，特点是查询速度快(因为带角标)，

但是增删速度稍慢,因为当元素多时，增删一个元素则所有元素的角标都得改变

线程不同步。默认长度是10，当超过长度时，按50%延长集合长度。

|--->LinkedList(JDK1.2):底层数据结构式链表数据结构(即后面一个元素记录前一个)，

特点：查询速度慢，因为每个元素只知道前面一个元素，但增删速度快

因为元素再多，增删一个，只要让其前后的元素重新相连即可

线程是不同步的。

|--->Vector(JDK1.0):底层数据结构是数组数据结构.特点是查询和增删速度都很慢。

默认长度是10，当超过长度时,按100%延长集合长度。

线程同步。

(Vector功能跟ArrayList功能一模一样，已被ArrayList替代)

\*\*List使用注意！

|--->ArrayList:

(1)当往ArrayList里面存入元素没什么要求时，即只要求有序就行时；

(2)当往ArrayList里面存入元素要求不重复时，比如存入学生对象，当同名同姓时

视为同一个人，则不往里面存储。则定义学生对象时，需复写equals方法

public boolean equals(Object obj)

{

if(!(obj instanceof Student))

return false;

Student stu = (Student)obj;

return this.name.equals(stu.name)&&this.age==stu.age;

}

则往ArrayList集合通过add存入学生对象时，集合底层自己会调用学生类的equals方法，

判断重复学生则不存入。

注：对于List集合，无论是add、contains、还是remove方法，判断元素是否相同，

都是通过复写equals方法来判断！

|--->LinkedList

(1)LinkLedist的特有方法：

boolean offerFirst(E e) 在此列表的开头插入指定的元素。

boolean offerLast(E e) 在此列表末尾插入指定的元素。

E peekFirst() 获取但不移除此列表的第一个元素；如果此列表为空，则返回 null。

E peekLast() 获取但不移除此列表的最后一个元素；如果此列表为空，则返回 null。

E pollFirst() 获取并移除此列表的第一个元素；如果此列表为空，则返回 null。

E pollLast() 获取并移除此列表的最后一个元素；如果此列表为空，则返回 null。

(2)通过LinkLedist的特有方法，可以实现某些数据特殊方式的存取，比如堆栈和队列。

一般情况下，使用哪种List接口下的实现类呢？

如果要求增删快，考虑使用LinkedList

如果要求查询快，考虑使用ArrayList

如果要求线程安全，考虑使用Vector。

|--->Set:集合，元素是无序的(因为没有索引)，元素不可以重复。可以有null元素。

|--->HashSet(JDK1.2):底层数据结构是哈希表、存取速度快、元素唯一、线程不同步。

保证性元素唯一的原理:

先判断元素的hashCode值是否相同，再判断两元素的equals方法是否为true

(往HashSet里面存的自定义元素要复写hashCode和equals方法，

以保证元素的唯一性！)

|--->TreeSet:底层数据结构式二叉树。可以对Set集合中的元素进行排序。元素有序、线程不同步。

保证元素唯一性的依据：compareTo方法return 0

TreeSet排序的第一种方式:让元素自身具备比较性，比如八种基本数据类型或则字符串，

实现Compareble接口,覆盖compareTo方法，

此方式是元素的自然顺序

TreeSet排序的第一种方式:当元素自身不具备比较性(比如存储学生对象时)或者具备的

比较性不是我们所需要的比较性时(比如想字符串的长度排序),

此时就需要让集合自身具备自定义的比较性。

那如何让集合自身具备比较性呢？可在集合初始化时，

就让集合具备比较方式。即定义一个类，

实现Comparator接口，覆盖compare方法。

\*\*Set集合使用注意事项：

(1)HashSet:

通过new的方式往HashSet里面存的元素的hashCode都不同，但通常我们定义对象，

比如学生对象时，虽然是new的两个学生对象，但是当他们name和age一样时，我们认为是

同一个对象，所以为了保证元素的唯一性，我们通常在往HashSet集合里面存储元素时，

在定义对象的类中通常复写hashCode和equals方法。

复制代码

1 public int hashCode()

2 {

3 return name.hashCode()+age\*39;

4 }

5 public boolean equals(Object obj)

6 {

7 if(!(obj instanceof Student))

8 return false;

9 Student stu = (Student)obj;

10 return this.name.equals(stu.name)&&this.age==stu.age;

11 }

复制代码

HashSet是如何保证元素唯一性的呢？

\*\*如果两元素的hashCode值不同，则不会调用equals方法

\*\*如果两元素的hashCode值相同，则继续判断equals是否返回true；

\*\*hashCode和equals方法虽然定义在自定义对象类里面，但不是我们手动调用

而是往HashSet集合里面存储元素的时候，集合底层自己调用hashCode和equals

它自己拿对象去判断，自己判断两元素是否是同一个元素。

(2)TreeSet:

TreeSet要求往里面存的元素具备比较性，否则会报错。

TreeSet排序的第一种方式:让元素自身具备比较性

定义对象类，实现Compareble接口,复写compareTo方法，此方式是元素的自然顺序

复制代码

1 class Student implements Comparable

2 {

3 private String name;

4 private int age;

5 public Student(String name,int age)

6 {

7 this.name=name;

8 this.age=age;

9 }

10 public String getName()

11 {

12 return name;

13 }

14 public int getAge()

15 {

16 return age;

17 }

18 public int compareTo(Object obj)

19 {

20 if(!(obj instanceof Student))

21 throw new RuntimeException("不是学生对象！");

22 Student stu = (Student)obj;

23 int num = this.age-stu.age;

24 if(num==0)

25 return this.name.compareTo(stu.name);

26 return num;

27 }

28 }

复制代码

TreeSet排序的第一种方式:让集合具备比较性

当元素自身不具备比较性(比如存储学生对象时)或者具备的

比较性不是我们所需要的比较性时(比如想字符串的长度排序),

此时就需要让集合自身具备自定义的比较性。

那如何让集合自身具备比较性呢？可在集合初始化时，

就让集合具备比较方式。即定义一个类，

实现Comparator接口，覆盖compare方法。

复制代码

1 class StringLengthComparator implements Comparator

2 {

3 public int compare(Object obj1,Object obj2)

4 {

5 String s1 = (String)obj1;

6 String s2 = (String)obj2;

7 int num = new Integer(s1.length()).compareTo(new Integer(s2.length()));

8 if(num==0)

9 return s1.compareTo(s2);

10 return num;

11 }

12 }

13 class TreeSetTest

14 {

15 public static void main(String[] args)

16 {

17 TreeSet ts = new TreeSet(new StringLengthComparator());

18 ts.add("addfg");

19 ts.add("dfg");

20 ts.add("agtuug");

21 ts.add("vgjkg");

22 sop(ts);

23 }

24 }

复制代码

基本数据类型或字符串对象均实现了Comparable接口，故同种类型基本数据间具备比较性，即自然顺序。

\*\*Map:顶层接口,该集合存储的是键值对,而且键是唯一的,Map和Set很像,Set集合底层就是使用了Map集合。

Map集合没有迭代器，要取出元素必须先将Map集合转换成Set集合才能遍历元素

|--->HashTable(JDK1.0):

底层是哈希表数据结构；

不可以使用null键和null值；

用作键的对象必须实现hashCode和equals方法来保证键的唯一性

线程同步，效率低

|--->HashMap(JDK1.2):

底层是哈希表数据结构；

允许使用null键和null值；

线程不同步，效率高；

保证元素唯一性的:

原理：先判断元素的hashCode值是否相同，再判断两元素的equals方法是否为true

(往HashSet里面存的自定义元素要复写hashCode和equals方法，

以保证元素的唯一性！)

复制代码

1 class Student {

2 private String name;

3 private int age;

4 public Student(String name, int age) {

5 super();

6 this.name = name;

7 this.age = age;

8 }

9 public int getAge() {

10 return age;

11 }

12 public void setAge(int age) {

13 this.age = age;

14 }

15 public String getName() {

16 return name;

17 }

18 public void setName(String name) {

19 this.name = name;

20 }

21

22 @Override

23 public int hashCode(){

24 return name.hashCode()+age\*34;

25 }

26 @Override

27 public boolean equals(Object obj){

28

29 if(!(obj instanceof Student))

30 return false;

31 Student stu = (Student)obj;

32 return this.name.equals(stu.name)&&this.age==stu.age;

33 }

34 public class HashMapDemo1 {

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

36 Map<Student , String> hmap = new HashMap<Student , String>();

37 hmap.put(new Student("001",20), "beijing");

38 hmap.put(new Student("002",25), "hebei");

39 hmap.put(new Student("003",50), "hainan");

40 hmap.put(new Student("001",20), "beijing");

41

42 System.out.println(hmap.size());

43 Set<Student> keySet = hmap.keySet();

44 Iterator<Student> it = keySet.iterator();

45 while(it.hasNext()){

46 Student stu = it.next();

47 String addr = hmap.get(stu);

48 System.out.println(stu.getName()+".."+stu.getAge()+"::"+addr);

49 }

50 }

51 }

52 |--->TreeMap(JDK1.0):

53 底层是二叉树结构；

54 允许使用null键和null值；

55 线程不同步；

56 可以给Map集合中的键进行排序.

57 TreeMap排序的第一种方式:让元素自身具备比较性，比如八种基本数据类型或则字符串，

58 实现Compareble接口,覆盖compareTo方法，

59 此方式是元素的自然顺序

60 TreeMap排序的第一种方式:当元素自身不具备比较性(比如存储学生对象时)或者具备的

61 比较性不是我们所需要的比较性时(比如想字符串的长度排序),

62 此时就需要让集合自身具备自定义的比较性。

63 那如何让集合自身具备比较性呢？可在集合初始化时，

64 就让集合具备比较方式。即定义一个类，

65 实现Comparator接口，覆盖compare方法。

66 class Student implements Comparable<Student>{

67 private String name;

68 private int age;

69 public Student(String name, int age) {

70 super();

71 this.name = name;

72 this.age = age;

73 }

74 public int getAge() {

75 return age;

76 }

77 public void setAge(int age) {

78 this.age = age;

79 }

80 public String getName() {

81 return name;

82 }

83 public void setName(String name) {

84 this.name = name;

85 }

86 @Override

87 public int compareTo(Student stu) {

88 int num = new Integer(this.age).compareTo(new Integer(stu.age));

89 if(num==0)

90 return this.name.compareTo(stu.name);

91 return num;

92 }

93 }

94

95 public class HashMapDemo1 {

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

97

98 Map<Student , String> tmap = new TreeMap<Student , String>();

99 tmap.put(new Student("001",20), "beijing");

100 tmap.put(new Student("002",25), "hebei");

101 tmap.put(new Student("003",50), "hainan");

102 tmap.put(new Student("001",20), "beijing");

103

104 System.out.println(tmap.size());

105 Set<Student> keySet1 = tmap.keySet();

106 Iterator<Student> it1 = keySet1.iterator();

107 while(it1.hasNext()){

108 Student stu = it1.next();

109 String addr = tmap.get(stu);

110 System.out.println(stu.getName()+".."+stu.getAge()+"::"+addr);

111 }

112 }

113 }

复制代码

\*\*Iterator：对collection进行迭代的迭代器.迭代器取代了Enumeration。

迭代器和枚举的区别：

迭代器允许调用者利用定义良好的语义在迭代期间从迭代器所指向的collection移除元素

方法名称得到了改进，简化书写

\*\*LisIterator：系列表迭代器，允许程序员按任一方向遍历列表、迭代期间修改列表

\*\*Comparable：此接口强行对实现它的每个类的对象进行整体自然排序。使元素具备比较性

\*\*Comparator：强行对某个对象collection进行整体排序的比较函数，使集合具备比较性

\*\*Collections：此类完全由在 collection 上进行操作或返回 collection 的静态方法组成。

\*\*Arrays：此类包含用来操作数组(比如排序和搜索)的各种静态方法

7、集合类各容器方法：

\*\*接口Collection方法摘要(没有构造方法)

a)添加：

i. boolean add(E e)

j. boolean addAll(Collection c)

b)删除：

i. void clear():清空容器

j. boolean remove(Objec object):

k. boolean removeAll(Collection c):

c)判断：

i. boolean contains(Object object):判断是否包含此元素

j. boolean containsAll(Collection c):判断是否包含一堆元素

k. boolean equals(Object object):比较此collection与指定对象是否相等

m. boolean isEmpty():判断是否集合为空

d)获取：

h. Iterator iterator():取出

i. int hashCode():返回此collection的哈希值

j. int size():返回此collection中元素的个数

k. boolean retainAll(Collection c):取交集

m. Object toArray():返回此collection中所有元素的数组

n. T[] toArray(T[] a):返回包含此collection中所有元素的数值。

\*\*\*\*\*List集合子类及其方法

(1)List接口是Collection接口的一个子接口。

(2)List接口中的元素有如下特点(对角标的操作都是特有方法，因为有序)：

A:元素有序(存储顺序和取出顺序一致)

B:元素可以重复

(3)List接口中的特有方法

A:add(int index,Object obj):在指定位置加入元素

B:remove(int index):移除指定位置的元素

C:set(int index,Object obj):修改指定位置的元素

D:get(int index):获取指定位置的元素

E:indexOf(Object obj):获取指定元素的位置

F:subList(int start,int end):从一个大的List中截取一个小的List

G:listIterator():返回一个List接口特有的迭代器

(1)、ArrayList：

|--->构造方法摘要：(少用，不是重点)

ArrayList()：构造一个初始容量为 10 的空列表。

ArrayList(Collection<? extends E> c)： 构造一个包含指定 collection 的元素的列表，

ArrayList(int initialCapacity)： 构造一个具有指定初始容量的空列表。

|--->方法摘要：

|--->添加：

boolean add(E e)： 将指定的元素添加到此列表的尾部。

void add(int index, E element)： 将指定的元素插入此列表中的指定位置。

boolean addAll(Collection<? extends E> c):按照指定 collection 的迭代器所返回的元素顺序，

将该 collection 中的所有元素添加到此列表的尾部

boolean addAll(int index, Collection<? extends E> c)： 从指定的位置开始，将指定 collection

中的所有元素插入到此列表中。

|--->删除：

void clear(): 移除此列表中的所有元素。

E remove(int index): 移除此列表中指定位置上的元素。

boolean remove(Object o): 移除此列表中首次出现的指定元素（如果存在）。

protected void removeRange(int fromIndex, int toIndex):

移除列表中索引在 fromIndex（包括）和 toIndex（不包括）之间的所有元素。

boolean removeAll(Collection<?> c): 从列表中移除指定 collection 中包含的其所有元素

|--->获取：

E get(int index): 返回此列表中指定位置上的元素。

int indexOf(Object o): 返回此列表中首次出现的指定元素的索引，或如果此列表不包含元素，则返回 -1。

int lastIndexOf(Object o) 返回此列表中最后一次出现的指定元素的索引，或如果此列表不包含索引，则返回 -1。

public List<E> subList(int fromIndex,int toIndex): 返回列表中指定的 fromIndex（包括 ） 和 toIndex（不包括）之间的部分视图。

Iterator<E> iterator(): 返回按适当顺序在列表的元素上进行迭代的迭代器。

ListIterator<E> listIterator(int index):返回列表中元素的列表迭代器(按适当顺序),从列表的指定位置开始。

|--->修改：(特有方法！！)

E set(int index, E element): 用指定的元素替代此列表中指定位置上的元素。

(2)LinkedList:

|--->构造方法摘要：

LinkedList(): 构造一个空列表。

LinkedList(Collection<? extends E> c): 构造一个包含指定 collection 中的元素的列表，

这些元素按其 collection 的迭代器返回的顺序排列。

|--->方法摘要:(特有的)

|--->添加

void addFirst(E e): 将指定元素插入此列表的开头。

void addLast(E e): 将指定元素添加到此列表的结尾。

|--->获取元素，但不删除元素

E get(int index): 返回此列表中指定位置处的元素。

E getFirst(): 返回此列表的第一个元素。

E getLast(): 返回此列表的最后一个元素。

|--->获取元素且删除元素

E remove()： 获取并移除此列表的头（第一个元素）。

E remove(int index)： 移除此列表中指定位置处的元素。

boolean remove(Object o)： 从此列表中移除首次出现的指定元素（如果存在）。

E removeFirst()： 移除并返回此列表的第一个元素。

E removeLast(): 移除并返回此列表的最后一个元素。

|--->修改

E set(int index, E element) 将此列表中指定位置的元素替换为指定的元素。

(3)Vector

|--->构造方法摘要：

Vector(): 构造一个空向量，使其内部数据数组的大小为 10，其标准容量增量为零。

Vector(Collection<? extends E> c): 构造一个包含指定 collection 中的元素的向量，

这些元素按其 collection 的迭代器返回元素的顺序排列。

|--->方法摘要：

|--->添加：

boolean add(E e): 将指定元素添加到此向量的末尾。

void add(int index, E element): 在此向量的指定位置插入指定的元素。

boolean addAll(Collection<? extends E> c):

将指定 Collection 中的所有元素添加到此向量的末尾，

按照指定 collection 的迭代器所返回的顺序添加这些元素。

boolean addAll(int index, Collection<? extends E> c): 在指定位置将指定 Collection 中的所有元素插入到此向量中。

|--->获取:

Enumeration<E> elements(): 返回此向量的组件的枚举。

Vector特有的取出方式:

枚举和迭代器很像，其实枚举和迭代器是一样的，只是因为枚举的名称和方法的名称

名字都过长，所以枚举被迭代器取代了。

|--->枚举Enumeration的方法摘要：

boolean hasMoreElements(): 测试此枚举是否包含更多的元素。

E nextElement(): 如果此枚举对象至少还有一个可提供的元素，

则返回此枚举的下一个元素。

\*\*\*\*\*Set集合子类及其方法

(1)HashSet:它不保证set的迭代顺序;特别是它不保证该顺序恒久不变.此类允许使用null元素。

|--->构造方法：

HashSet() 构造一个新的空 set，其底层 HashMap 实例的默认初始容量是 16，加载因子是 0.75。

HashSet(Collection<? extends E> c) 构造一个包含指定 collection 中的元素的新 set。

|--->方法摘要：

boolean add(E e) 如果此 set 中尚未包含指定元素，则添加指定元素。

void clear() 从此 set 中移除所有元素。

Object clone() 返回此 HashSet 实例的浅表副本：并没有复制这些元素本身。

boolean contains(Object o) 如果此 set 包含指定元素，则返回 true。

boolean isEmpty() 如果此 set 不包含任何元素，则返回 true。

Iterator<E> iterator() 返回对此 set 中元素进行迭代的迭代器。

boolean remove(Object o) 如果指定元素存在于此 set 中，则将其移除。

int size() 返回此 set 中的元素的数量（set 的容量）。

(2)TreeSet:使用元素的自然顺序对元素进行排序，或者根据创建 set 时提供的 Comparator 进行排序.

|--->构造方法：

TreeSet() 构造一个新的空 set，该set根据其元素的自然顺序进行排序。

TreeSet(Collection<? extends E> c)

构造一个包含指定 collection 元素的新 TreeSet，它按照其元素的自然顺序进行排序。

TreeSet(Comparator<? super E> comparator) 构造一个新的空 TreeSet，它根据指定比较器进行排序。

|--->方法摘要：

添加：

boolean add(E e) 将指定的元素添加到此 set（如果该元素尚未存在于 set 中）。

boolean addAll(Collection<? extends E> c) 将指定 collection 中的所有元素添加到此 set 中。

删除：

void clear() 移除此 set 中的所有元素。

boolean remove(Object o) 将指定的元素从 set 中移除（如果该元素存在于此 set 中）。

E pollFirst() 获取并移除第一个（最低）元素；如果此 set 为空，则返回 null。

E pollLast() 获取并移除最后一个（最高）元素；如果此 set 为空，则返回 null。

获取：

Iterator<E> iterator() 返回在此 set 中的元素上按升序进行迭代的迭代器。

E first() 返回此 set 中当前第一个（最低）元素。

E last() 返回此 set 中当前最后一个（最高）元素。

int size() 返回 set 中的元素数（set 的容量）。

判断：

boolean isEmpty() 如果此 set 不包含任何元素，则返回 true。

boolean contains(Object o) 如果此 set 包含指定的元素，则返回 true。

\*\*Map:将键映射到值的对象。Map集合没有迭代器！Map集合特点：该集合存储键值对。而且键是唯一的。

|--->方法摘要：

|--->添加：

V put(K key, V value) 将指定的值与此映射中的指定键关联（可选操作）。

void putAll(Map<? extends K,? extends V> m) 从指定映射中将所有映射关系复制到此映射中

|--->删除：

void clear() 从此映射中移除所有映射关系（可选操作）。

V remove(Object key) 如果存在一个键的映射关系，则将其从此映射中移除（可选操作）。

|--->判断

boolean containsKey(Object key) 如果此映射包含指定键的映射关系，则返回 true。

boolean containsValue(Object value) 如果此映射将一个或多个键映射到指定值，则返回 true。

boolean isEmpty() 如果此映射未包含键-值映射关系，则返回 true。

|--->获取

int size() 返回此映射中的键-值映射关系数。

Collection<V> values() 返回此映射中包含的值的 Collection 视图。

重点：Map集合没有迭代器，以下是Map的两种取出方式：

第一种：Set<K> keySet()

返回此映射中包含的键的Set视图，将Map集合中所有的键存入Set集合，然后再通过Set集合的

迭代器取出所有的键，再根据get方法获取每个键的值；

第二种：Set<Map.Entry<K,V>> entrySet()

返回此映射中包含的映射关系的Set视图，将Map集合中的映射关系存入到Set集合中，

这个映射关系的数据类型是Map.entry,再通过Map.Entry类的方法再要取出关系里面的键和值

Map.Entry的方法摘要：

boolean equals(Object o) 比较指定对象与此项的相等性。

K getKey() 返回与此项对应的键。

V getValue() 返回与此项对应的值。

int hashCode() 返回此映射项的哈希码值。

V setValue(V value) 用指定的值替换与此项对应的值(特有！！！)。

8、Map集合和Collection集合的区别？

1，

Map中一次存储是键值对。

Collection中一次存储是单个元素。

2，

Map的存储使用的put方法。

Collection存储使用的是add方法。

3，

Map集合没有迭代器，Map的取出，是将Map转成Set，在使用迭代器取出。

Collection取出，使用就是迭代器。

4，

如果对象很多，必须使用集合存储。

如果元素存在着映射关系，可以优先考虑使用Map存储或者用数组，

如果没有映射关系，可以使用Collection存储。

8、迭代器：Iterator(Map集合没有迭代器)

(1)迭代器就是取出集合元素的方式

(2)迭代器的作用

因为每个集合中元素的取出方式都不一样，于是就把元素的取出方式进行抽取，并定义在集合内部，

这样取出方式就可以直接访问集合内部的元素；

而每个容器的数据结构不同，所以取出动作的细节也不一样，但是有共性内容：判断和取出。

那么就将共性内容进行抽取，从而形成了接口Iterater

(3)获取迭代器的方法：

Iterator<E> iterator() 返回在此 collection 的元素上进行迭代的迭代器。

Iterator<E> iterator() 返回在此 set 中的元素上进行迭代的迭代器。

(3)迭代器方法：

boolean hasNext() 如果仍有元素可以迭代,则返回 true。

E next() 返回迭代的下一个元素。

void remove() 从迭代器指向的collection中移除迭代器返回的最后一个元素（可选操作）。

9、列表迭代器：ListIterator

(1)List集合特有的迭代器ListIterator是Iterator的子接口，在迭代时，不可以通过集合对象的

方法操作集合中的元素，因为会发生ConcurrentModificationException(当方法检测到对象的并发修改，

但不允许这种修改时，抛出此异常)

(2)Iterator方法有限，只能对元素进行判断、取出和删除的操作

ListIterator可以对元素进行添加和修改动作等。

(3)获取列表迭代器方法：

ListIterator<E> listIterator() 返回此列表元素的列表迭代器（按适当顺序）。

ListIterator<E> listIterator(int index)

返回此列表中的元素的列表迭代器（按适当顺序），从列表中指定位置开始。

(4)列表迭代器方法：

void add(E e) 将指定的元素插入列表（可选操作）。

boolean hasPrevious() 如果以逆向遍历列表，列表迭代器有多个元素，则返回 true。

int nextIndex() 返回对 next 的后续调用所返回元素的索引。

E previous() 返回列表中的前一个元素。

int previousIndex() 返回对 previous 的后续调用所返回元素的索引。

void set(E e) 用指定元素替换 next 或 previous 返回的最后一个元素（可选操作）。

10、堆栈和队列

堆栈：先进后出，比如杯子里的水

队列：先进先出，比如水管的水

11、集合类各种容器的使用注意细节：

(1)迭代器：

\*\*迭代器的next方法是自动向下取元素，要避免出现NoSuchElementException。

也就是在迭代循环中调用一次next方法一次就要hasNext判断一次，比如语句

sop(it.next()+"..."+it.next())会发生上述异常。

\*\*迭代器的next方法返回值类型是Object，所以要记得类型转换,应用泛型后就不用强转

(2)List集合：

\*\*List集合里面的元素因为是带角标，所以List集合里面的元素都是有序的，

另外List集合可以包含重复元素，也可以包含null。

\*\*List集合有迭代器Iterator，还有一个特有迭代器列表ListIterator

\*\*List集合中判断元素是否相同都是用equals方法，无论contains、remove都依赖equals方法

比如往ArrayList集合里面存放学生，同名同年龄视为同一个人，此时就需要在学生类复写Object类

里面的equals方法(非常重要！！！要注意！！)

(3)Set集合：

\*\*Set接口里面存放的是元素是无序的，不可以有重复元素，可以包含null

\*\*Set集合只有一种取出方式，就是迭代器Iterator

\*\*Set集合功能和Collection是一致的，没有特殊方法

|--->HashSet:

\*\*集合里面存放的元素是无序的，唯一的

\*\*底层数据结构是哈希表，哈希表结构的数据都是无序的，哈希表结构的操作效率都高效

\*\*线程不同步

\*\*保证元素唯一性的原理是：通过复写hashCode和equals方法

\*\*\*\*如果两元素的hashCode值相同，则继续判断两元素equals是否为真

\*\*\*\*如果两元素的hashCode值不同，则不会调用equals方法。

\*\*当我们往HashSet集合存放自定义的元素时(比如学生对象)，通常都要复写hashCode和equals方法，

而且hashCode和equals方法不通过我们调用，HashSet集合底层内部自己调用，自己拿元素去比较

|--->TreeSet

\*\*TreeSet集合可以对存放的元素进行排序，弥补了Set集合元素无序的缺点，且元素是唯一的

\*\*底层数据结构是二叉树，二叉树结构都是有序的

\*\*线程不同步

\*\*TreeSet集合要求往集合里存放的元素自身具备比较性，否则会报错

\*\*TreeSet集合保证元素唯一性的依据是：通过compareTo或者compare方法中的来保证元素的唯一性。

TreeSet排序的第一种方式:让元素自身具备比较性，

定义元素类实现Compareble接口，覆盖compare方法，

此方式是元素的自然顺序。

TreeSet排序的第二种方式:让集合具备比较性

当元素自身不具备比较性或者具备的比较性不是

我们所需要的比较性时，此时就需要让集合具备自定义的比较性。

那如何让集合自身具备比较性呢？

可在集合初始化时，就让集合具备比较方式。

即定义一个类，实现Comparator接口，覆盖compare方法。

注：

\*\*判断元素唯一时，当主要条件一样时，判断次要条件

\*\*两种排序方式都在时，以比较器为主！！！

(4)Map集合：

|--Hashtable

底层是哈希表结构

线程安全的，并且键和值不能为null。

|--HashMap

底层是哈希表结构

线程不安全的，键和值可以为null。

|--LinkedHashMap

底层是链表和哈希表

线程不安全

|--TreeMap

底层是二叉树

线程不安全的

12、如果你想将一组对象按一定顺序存取，在不考虑并发访问的情况下会使用\_\_\_\_C\_\_\_\_\_ ,

反之则会使用\_\_\_\_A\_\_\_\_\_；如果你想存储一组无序但唯一的对象，你会使用\_\_\_B\_\_\_\_\_\_ ;

如果你想按关键字对对象进行存取，在不考虑并发访问的情况下会使用\_\_\_D\_\_\_\_\_\_ ,反之则会使用\_\_\_\_\_E\_\_\_\_。

A. Vector

B. HashSet

C. ArrayList

D. HashMap

E. Hashtable

13、泛型：

(1)为什么会出现泛型？

因为集合存放的数据类型不固定，故往集合里面存放元素时，存在安全隐患，

如果在定义集合时，可以想定义数组一样指定数据类型，那么就可以解决该类安全问题。

JDK1.5后出现了泛型，用于解决集合框架的安全问题。

泛型是一个类型安全机制。

(2)泛型定义格式：通过<>来定义要操作的引用数据类型

ArrayList<String> al = new ArrayList<String>;

(3)泛型的好处：

\*\*将运行时期出现的ClassCastException(类型转换异常)问题转移到编译时期；

\*\*避免了强制转换的麻烦

(4)什么时候定义泛型？

泛型在集合框架中很常见，只要见到<>就要定义泛型。其实<>就是用来接收类型的。

当使用集合时，将集合中要存储的数据类型作为参数传递到<>中即可

(5)泛型的形式

\*\*泛型类：即自定义泛型类

A：当类中要操作的引用数据类型不确定时，早起定义Object来完成扩展，现在定义泛型来完成

B：局限性：泛型类定义的泛型，在整个类中有效，如果该泛型类的方法被调用，

当泛型类的对象明确要操作的类型后，所有要操作的类型就被固定。

\*\*泛型方法：泛型放在返回值前面，修饰符的后面

A:为了避免泛型类的局限性，让不同方法可以操作不同的类型，而且类型还不确定，

则可以将泛型定义在方法上

B:特殊之处：静态方法不可以反问类上定义的泛型

如果静态方法操作的应用数据类型不确定，可以讲泛型定义在静态方法上

\*\*泛型接口：

当泛型定义在接口上时，则子类中要指定实现接口类型，同时还可以子类也可以定义为泛型类

(6)泛型的高级应用：？通配符

\*\*当指定两种泛型的集合，则迭代时也要定义两种泛型的迭代器，麻烦，此时可通过将迭代器的泛型

改为？，如Iterator<?> it=al.iterator();

\*\*两种泛型限定

向上限定： ? extends E ;E可以接收E类型或者E的子类

向下限定： ? super E ;E可以接收E类型或者E的父类

14、高级for循环

(1)JDK1.5新特性，代替迭代器使用时的不爽，简化书写，底层原理是迭代器凡是支持迭代器的都支持高级for循环

高级for循环，只用于集合和数组的遍历，集合只能用Collection不能用Map集合

只能把Map集合转化成Set集合，才能用for循环。

(2)格式

for(数据类型 变量名:被遍历的集合(Collection)或者数组)

{

}

(3)局限性：

必须要有遍历的目标

对集合或者数组进行遍历时，只能获取集合元素，不能对集合元素进行操作

迭代器除了遍历，还可以进行remove操作集合中的元素

列表迭代器还可以在遍历过程中进行增删改查的操作

(4)传统for循环和高级for循环的区别

高级for循环有一个局限性，就是必须要有遍历的目标(集合或者数组)

遍历数组时建议使用传统for循环，因为可以定义角标，比如打印100次helloworld时用传统for循环方便

15、可变参数

(1)数组的可变参数

格式：

int... arr

(3)方法的可变参数

格式：

public static void show(String str，int... arr)

{

}

注意：可变参数一定要放在参数列表的最后面

16、静态导入：

\*\*import static java.util.Arrays.\* 导入的是Arrays这个类中所有的静态方法

\*\*当类名重名时，需要制定具体的报名

\*\*当方法重名时，需要制定具体所属的对象或者类

17、Collections类：

(1)此类完全由在 collection 上进行操作或返回 collection 的静态方法组成。

(2)静态方法摘要：

static <T> boolean addAll(Collection<? super T> c, T... elements)

将所有指定元素添加到指定 collection 中。

static <T> void fill(List<? super T> list, T obj)

使用指定元素替换指定列表中的所有元素。

static <T> boolean replaceAll(List<T> list, T oldVal, T newVal)

使用另一个值替换列表中出现的所有某一指定值。

static void reverse(List<?> list)

反转指定列表中元素的顺序。

static <T> Comparator<T> reverseOrder()

返回一个比较器，它强行逆转实现了 Comparable 接口的对象 collection 的自然顺序

static <T> Comparator<T> reverseOrder(Comparator<T> cmp)

返回一个比较器，它强行逆转指定比较器的顺序。

(3)Collections类特牛的方法：

集合有一个共同的缺点，那就是线程不安全，被多线程操作时，容易出现问题，虽然可以自己加锁

但是麻烦。Collections提供特牛的方法，就是给它一个不同步的集合，它返回一个同步的安全的集合

static <T> Collection<T> synchronizedCollection(Collection<T> c)

返回指定 collection 支持的同步（线程安全的）collection。

static <T> List<T> synchronizedList(List<T> list)

返回指定列表支持的同步（线程安全的）列表。

static <K,V> Map<K,V> synchronizedMap(Map<K,V> m)

返回由指定映射支持的同步（线程安全的）映射。

static <T> Set<T> synchronizedSet(Set<T> s)

返回指定 set 支持的同步（线程安全的）set。

static <K,V> SortedMap<K,V> synchronizedSortedMap(SortedMap<K,V> m)

返回指定有序映射支持的同步（线程安全的）有序映射。

static <T> SortedSet<T> synchronizedSortedSet(SortedSet<T> s)

返回指定有序 set 支持的同步（线程安全的）有序 set。

18、Arrays类：

此类包含用来操作数组（比如排序和搜索）的各种方法。里面都是静态方法。

如果指定数组引用为 null，则此类中的方法都会抛出 NullPointerException。

(1)静态方法摘要：

static <T> List<T> asList(T... a)

返回一个受指定数组支持的固定大小的列表。

注意：

A:该方法将一个数组变成集合后，不可以使用集合的增删方法，因为数组的长度是固定的！

如果增删，则发生UnsupportedOprationException(不支持操作异常)

B:如果数组中的元素都是基本数据类型，则该数组变成集合时，会将该数组作为集合的一个

元素出入集合

C:如果数组中的元素都是对象，如String，那么数组变成集合后，数组中的元素就直接转成

集合中的元素

19、数组变集合以及集合变数组的对比：

(1)数组变集合：

方法：static <T> List<T> asList(T... a) 返回一个受指定数组支持的固定大小的列表。

好处：可以使用集合的思想和方法操作数组中的元素，数组是一个对象，但是数组中的功能很少

(2)集合变数组：

方法：Collction中的toArray方法

好处：可以限定对集合元素的操作，防止对集合的元素进行增删，因为数组长度是固定的。

20、Collections类和Arrays类的使用。(重点)

A:Collections

排序

二分查找

发转

B:Arrays

把数组变成字符串输出

排序

二分查找

21、System：

(1)描述系统信息的类

(2)该类没有构造方法，该类的方法和属性都是静态的

(3)字段摘要：

static InputStream in “标准”输入流。

static PrintStream out “标准”输出流。

(4)方法摘要：

static void exit(int status) 终止当前正在运行的 Java 虚拟机。

static void gc() 运行垃圾回收器。

static Properties getProperties() 确定当前的系统属性

static String getProperty(String key) 获取指定键指示的系统属性。

static String getProperty(String key, String def) 获取用指定键描述的系统属性。

static void setIn(InputStream in) 重新分配“标准”输入流。

static void setOut(PrintStream out) 重新分配“标准”输出流。

static void setProperties(Properties props) 将系统属性设置为 Properties 参数。

static String setProperty(String key, String value) 设置指定键指示的系统属性。

22、Runtime:

(1)每个 Java 应用程序都有一个 Runtime 类实例，使应用程序能够与其运行的环境相连接。

可以通过 getRuntime 方法获取当前运行时。 应用程序不能创建自己的 Runtime 类实例。

(2)该类没有构造函数，也就是它不能直接创建对象，但是它里里面的方法又不是静态的

，故它一定有一个方法返回本类对象

(3)故该类是单例设计模式，保证在内存中只有一个对象

(4)方法摘要：

Process exec(String command) 在单独的进程中执行指定的字符串命令

void gc() 运行垃圾回收器。

static Runtime getRuntime() 返回与当前 Java 应用程序相关的运行时对象

void exit(int status) 通过启动虚拟机的关闭序列，终止当前正在运行的 Java 虚拟机

23、Date:

(1)Date接口表示特定的瞬间，精确到毫秒

(2)构造方法

Date() 分配 Date 对象并初始化此对象，以表示分配它的时间（精确到毫秒）。

Date(long date) 分配Date对象并初始化此对象，以表示自从标准基准时间（称为“历元（epoch）”，

即1970年1月1日00:00:00GMT）以来的指定毫秒数。

(3)方法摘要：

int compareTo(Date anotherDate) 比较两个日期的顺序。

boolean equals(Object obj) 比较两个日期的相等性。

24、Calendar：

(1)直接已知子类： GregorianCalendar

(2)构造方法：

protected Calendar() 构造一个带有默认时区和语言环境的 Calendar。

protected Calendar(TimeZone zone, Locale aLocale) 构造一个带有指定时区和语言环境的 Calendar。

(3)方法摘要：

static Calendar getInstance() 使用默认时区和语言环境获得一个日历。

四、jdk1.5的新特性

(1)静态导入：

\*\*import语句可以导入一个类或某个包中的所有类

\*\*import static语句导入一个类中的某个静态方法或所有静态方法

静态导入后，静态方法前面就不用写类名.方法的方式类调用

\*\*语法举例：

import static java.lang.Math.sin;//导入一个静态方法

import static java.lang.Math.\*; //导入一个类中的所有静态方法

\*\*静态导入使用注意：

当类名重复时，需要制定具体的包名；

当方法重名时，需要制定具体所属的对象或者类

(2)可变参数：

\*\*可变参数的特点：

\*可变参数只能出现在参数列表的最后；

\*...位于变量类型和变量名之间，前后有无空格都可以;

\*调用可变参数的方法时，编译器为该可变参数隐含创建一个数组，

在方法体中以数组的形式访问可变参数。

\*\*可变参数举例：

\*变量类型... 变量名 如 int... arr 表示可变参数数组

\*public static void show(String str , int... arr){}

(3)增强for循环：

\*\*语法：

for ( type 变量名：集合变量名 ) { … }

\*\*注意事项：

迭代变量必须在( )中定义！

集合变量可以是数组或实现了Iterable接口的集合类

\*\*举例：

public static int add(int x,int ...args) {

int sum = x;

for(int arg:args) {

sum += arg;

}

return sum;

}

\*\*增强for循环代替了迭代器使用的不爽，简化书写

\*\*增强for循环局限性：

对集合或者数组进行遍历时，只能取元素，不能对集合进行操作

(4)基本数据类型的自动装箱和拆箱

\*\*基本数据类型

byte ---> Byte

short ---> Short

int ---> Integer

long ---> Long

float ---> Float

double ---> Double

char ---> Character

boolean ---> Boolean

\*\*例子：

\*\*装箱：自动把一个基本数据类型的数据装箱成一个该类型数据的对象引用

Integer i = 3;(jdk1.5之前这样写是不行的，编译报错)

\*\*拆箱：自动把一个基本数据类型的对象引用拆箱成一个基本数据类型的数据，再参与运算

Integer i = 12;

sop(i+4);

\*\*享元模式：

Integer num1 = 12;

Integer num2 = 12;

System.out.println(num1 == num2);//打印true

Integer num5 = Integer.valueOf(12);

Integer num6 = Integer.valueOf(12);

System.out.println(num5 == num6);//打印true

Integer num3 = 129;

Integer num4 = 129;

System.out.println(num3 == num4);//打印false

为什么前面的返回true而后面的运算返回false呢？

对于基本数据类型的整数，装箱成Integer对象时，如果该数值在一个字节内,(-128~127)，

一旦装箱成Integer对象后，就把它缓存到磁里面，当下次，又把该数值封装成Integer对象时

会先看磁里面有没有该对象，有就直接拿出来用，这样就节省了内存空间。因为比较小的整数，

用的频率比较高，就没必要每个对象都分配一个内存空间。

这就是享元模式！比如26个英文字母，10个阿拉伯数字

(5)枚举

\*\*为什么要有枚举？

问题：要定义星期几或性别的变量，该怎么定义？假设用1-7分别表示星期一到星期日，

但有人可能会写成int weekday = 0;或即使使用常量方式也无法阻止意外。

枚举就是要让某个类型的变量的取值只能为若干个固定值中的一个，否则，编译器就会报错。

枚举可以让编译器在编译时就可以控制源程序中填写的非法值，

普通变量的方式在开发阶段无法实现这一目标。

\*\*用普通类如何实现枚举的功能？定义一个Weekday类来模拟实现：

步骤：

\*私有化构造方法

\*每个元素分别用一个公有的静态成员变量表示(public static final)

\*可以有若干公有方法或抽象方法。采用抽象方法定义nextDay就将大量的if.else语句

转移成了一个个独立的类。

\*\*枚举的应用：

举例：定义一个Weekday的枚举。

扩展：枚举类的values,valueOf,name,toString,ordinal等方法

（记住，讲课时要先于自定义方法前介绍，讲课更流畅）

总结：枚举是一种特殊的类，其中的每个元素都是该类的一个实例对象。

例如可以调用WeekDay.SUN.getClass().getName和WeekDay.class.getName()。

\*\*枚举的高级应用：

\*\*枚举就相当于一个类，其中也可以定义构造方法、成员变量、普通方法和抽象方法。

\*\*枚举元素必须位于枚举体中的最开始部分，枚举元素列表的后要有分号与其他成员分隔。

把枚举中的成员方法或变量等放在枚举元素的前面，编译器报告错误。

\*\*带构造方法的枚举

构造方法必须定义成私有的

如果有多个构造方法，该如何选择哪个构造方法？

枚举元素MON和MON()的效果一样，都是调用默认的构造方法。

\*\*带方法的枚举

定义枚举TrafficLamp

实现普通的next方法

实现抽象的next方法：每个元素分别是由枚举类的子类来生成的实例对象，

这些子类采用类似内部类的方式进行定义。增加上表示时间的构造方法

\*\*枚举只有一个成员时，就可以作为一种单例的实现方式。

(6)泛型：

\*\*泛型是提供给javac编译器使用的，可以限定集合中的输入类型，让编译器挡住源程序中的非法输入，

编译器编译带类型说明的集合时会去除掉“类型”信息，使程序运行效率不受影响，

对于参数化的泛型类型，getClass()方法的返回值和原始类型完全一样。

由于编译生成的字节码会去掉泛型的类型信息，只要能跳过编译器，

就可以往某个泛型集合中加入其它类型的数据，例如，用反射得到集合，再调用其add方法即可。

\*\*ArrayList<E>类定义和ArrayList<Integer>类引用中涉及如下术语：

整个称为ArrayList<E>泛型类型

ArrayList<E>中的E称为类型变量或类型参数

整个ArrayList<Integer>称为参数化的类型

ArrayList<Integer>中的Integer称为类型参数的实例或实际类型参数

ArrayList<Integer>中的<>念着typeof

ArrayList称为原始类型

\*\*参数化类型与原始类型的兼容性：

参数化类型可以引用一个原始类型的对象，编译报告警告，

例如，  
Collection<String> c = new Vector();//可不可以，不就是编译器一句话的事吗？

原始类型可以引用一个参数化类型的对象，编译报告警告，

例如，  
Collection c = new Vector<String>();//原来的方法接受一个集合参数，新的类型也要能传进去

\*\*参数化类型不考虑类型参数的继承关系：

Vector<String> v = new Vector<Object>(); //错误!///不写<Object>没错，写了就是明知故犯

Vector<Object> v = new Vector<String>(); //也错误!

编译器不允许创建泛型变量的数组。即在创建数组实例时，

数组的元素不能使用参数化的类型，

例如，下面语句有错误：

Vector<Integer> vectorList[] = new Vector<Integer>[10];

\*\*泛型限定：

\*\*限定通配符的上边界：

正确：Vector<? extends Number> x = new Vector<Integer>();

错误：Vector<? extends Number> x = new Vector<String>();

\*\*限定通配符的下边界：

正确：Vector<? super Integer> x = new Vector<Number>();

错误：Vector<? super Integer> x = new Vector<Byte>();

\*\*提示：

限定通配符总是包括自己。

?只能用作引用，不能用它去给其他变量赋值

Vector<? extends Number> y = new Vector<Integer>();

Vector<Number> x = y;

上面的代码错误，原理与Vector<Object > x11 = new Vector<String>();相似，

只能通过强制类型转换方式来赋值。

五、IO流

1、IO流概述

(1)用来处理设备(硬盘，控制台，内存)间的数据。

(2)java中对数据的操作都是通过流的方式。

(3)java用于操作流的类都在io包中。

(4)按照流操作的数据的类型不同：分为字节流和字符流。字符流是为了方便中文的操作而来的。

(5)按照流的流向不同分为：输入流，输出流

2、IO流常用基类：

(1)字节流

输出字节流：OutputStream：字节写入流抽象类

|--->FileOutputStream：

字节写入流

|--->BufferedOutputStream：

字节写入流缓冲区

|--->PrintStream：

打印流

输入字节流：InputStream：字节读取流抽象类

|--->FileInputStream：

字节读取流

|--->BufferedInputStream：

字节读取流缓冲区

(2)字符流

输出字符流：Writer：字符写入流的抽象

|--->FileWriter：

字符写入流

|--->BufferedWriter：

字符写入流缓冲区

|--->OutputStreamWriter：

字符通向字节的转换流(涉及键盘录入时用)

|--->OutputStreamWriter：

打印流，可处理各种类型的数据

输入字符流：Reader: 字符读取流的抽象类

|--->FileReader：

字符读取流

|--->LineNumberReader：

跟踪行号的缓冲字符读取流

|--->BufferedReader：

字符读取流缓冲区

|--->InputStreamReader：

字节通向字符的转换流(涉及键盘录入时用)

(3)IO流常用基类方法摘要：

\*\*字节写入流：OutputStream：

void close() 关闭此输出流并释放与此流有关的所有系统资源。

void flush()刷新此输出流并强制写出所有缓冲的输出字节。

abstract void write(int b) 将指定的字节写入此输出流。

void write(byte[] b) 将 b.length 个字节从指定的 byte 数组写入此输出流。

void write(byte[] b, int off, int len)

将指定 byte 数组中从偏移量 off 开始的 len 个字节写入此输出流。

\*\*字节读取流：InputStream：

void close() 关闭此输入流并释放与该流关联的所有系统资源。

int available() (特有方法！！)

返回此输入流下一个方法调用可以不受阻塞地从此输入流读取（或跳过）的估计字节数。

abstract int read() 从输入流中读取数据的下一个字节。

int read(byte[] b) 从输入流中读取一定数量的字节，并将其存储在缓冲区数组 b 中。

int read(byte[] b, int off, int len) 将输入流中最多 len 个数据字节读入 byte 数组。

long skip(long n) 跳过和丢弃此输入流中数据的 n 个字节。

\*\*字符写入流：Writer：

abstract void close() 关闭此流，但要先刷新它。

abstract void flush() 刷新该流的缓冲。

void write(int c) 写入单个字符。

void write(char[] cbuf) 写入字符数组。

abstract void write(char[] cbuf, int off, int len) 写入字符数组的某一部分。

void write(String str) 写入字符串。

void write(String str, int off, int len) 写入字符串的某一部分。

\*\*字符读取流：Reader：

abstract void close() 关闭该流并释放与之关联的所有资源。

int read() 读取单个字符。

int read(char[] cbuf) 将字符读入数组

abstract int read(char[] cbuf, int off, int len) 将字符读入数组的某一部分。

long skip(long n) 跳过字符。

3、IO流常用字节流基类的子类：

\*\*写入流：

(1)FileOutputStream：

\*\*构造方法：

FileOutputStream(String name)

创建一个向具有指定名称的文件中写入数据的输出文件流。

FileOutputStream(String name, boolean append)

创建一个向具有指定 name 的文件中写入数据的输出文件流。

FileOutputStream(File file)

创建一个向指定 File 对象表示的文件中写入数据的文件输出流。

FileOutputStream(File file, boolean append)

创建一个向指定 File 对象表示的文件中写入数据的文件输出流。

\*\*方法摘要：

public void flush()

void close() 关闭此文件输出流并释放与此流有关的所有系统资源。

void write(int b) 将指定字节写入此文件输出流。

void write(byte[] b, int off, int len)

将指定 byte 数组中从偏移量 off 开始的 len 个字节写入此文件输出流。

void write(int b) 将指定字节写入此文件输出流。

(2)BufferedOutputStream：

\*\*构造方法：

BufferedOutputStream(OutputStream out)

创建一个新的缓冲输出流，以将数据写入指定的底层输出流。

BufferedOutputStream(OutputStream out, int size)

创建一个新的缓冲输出流，以将具有指定缓冲区大小的数据写入指定的底层输出流。

\*\*方法摘要：

void flush() 刷新此缓冲的输出流。

void write(byte[] b, int off, int len)

将指定 byte 数组中从偏移量 off 开始的 len 个字节写入此缓冲的输出流。

void write(int b) 将指定的字节写入此缓冲的输出流。

(3)PrintStream：打印流，可将各种类型的数据原样打印，有自动刷新功能

\*\*构造方法：

PrintStream(String fileName)

创建具有指定文件名称且不带自动行刷新的新打印流。

PrintStream(File file)

创建具有指定文件且不带自动行刷新的新打印流。

PrintStream(OutputStream out)

创建新的打印流。

PrintStream(OutputStream out, boolean autoFlush) (当autoFlush为true时具有自动刷新功能)

创建新的打印流。

\*\*方法摘要：

PrintStream append(char c)

将指定字符添加到此输出流。

void close()

关闭流。

void flush()

刷新该流的缓冲。

void print(各种类型的数据：)

打印各种类型的数据

void println(各种类型的数据：)：自动换行

打印各种类型的数据

void write(byte[] buf, int off, int len)

将 len 字节从指定的初始偏移量为 off 的 byte 数组写入此流。

void write(int b)

将指定的字节写入此流。

\*\*读取流：

(1)FileInputStream：

\*\*构造方法：

FileInputStream(String name)

通过打开一个到实际文件的连接来创建一个 FileInputStream，

该文件通过文件系统中的路径名 name 指定。

FileInputStream(File file)

通过打开一个到实际文件的连接来创建一个 FileInputStream，

该文件通过文件系统中的 File 对象 file 指定。

\*\*方法摘要：

int available() (字节读取流特有方法！！！)

返回下一次对此输入流调用的方法可以不受阻塞地从此输入流读取（或跳过）的估计剩余字节数。

int read()

从此输入流中读取一个数据字节。

int read(byte[] b)

从此输入流中将最多 b.length 个字节的数据读入一个 byte 数组中。

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

从此输入流中将最多 len 个字节的数据读入一个 byte 数组中。

long skip(long n)

从输入流中跳过并丢弃 n 个字节的数据。

(2)BufferedInputStream:

\*\*构造方法：

BufferedInputStream(InputStream in)

创建一个 BufferedInputStream 并保存其参数，即输入流 in，以便将来使用。

BufferedInputStream(InputStream in, int size)

创建具有指定缓冲区大小的 BufferedInputStream 并保存其参数，即输入流 in，以便将来使用。

\*\*方法摘要：

int available() (字节读取流特有方法！！！)

返回可以从此输入流读取（或跳过）、且不受此输入流接下来的方法调用阻塞的估计字节数。

int read()

参见 InputStream 的 read 方法的常规协定。

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

从此字节输入流中给定偏移量处开始将各字节读取到指定的 byte 数组中。

long skip(long n)

参见 InputStream 的 skip 方法的常规协定。

4、字符流常用基类的子类

\*\*写入流：

(1)FileWriter:

\*\*构造方法：

FileWriter(String fileName)

根据给定的文件名构造一个 FileWriter 对象。

FileWriter(String fileName, boolean append)

根据给定的文件名以及指示是否附加写入数据的 boolean 值来构造 FileWriter 对象。

FileWriter(File file)

根据给定的 File 对象构造一个 FileWriter 对象。

FileWriter(File file, boolean append)

根据给定的 File 对象构造一个 FileWriter 对象。

FileWriter(FileDescriptor fd)

构造与某个文件描述符相关联的 FileWriter 对象。

\*\*方法摘要：跟Writer一样

abstract void close() 关闭此流，但要先刷新它。

abstract void flush() 刷新该流的缓冲。

void write(int c) 写入单个字符。

void write(char[] cbuf) 写入字符数组。

abstract void write(char[] cbuf, int off, int len) 写入字符数组的某一部分。

void write(String str) 写入字符串。

void write(String str, int off, int len) 写入字符串的某一部分。

(2)BufferedWriter:

\*\*构造方法：

BufferedWriter(Writer out)

创建一个使用默认大小输出缓冲区的缓冲字符输出流。

BufferedWriter(Writer out, int sz)

创建一个使用给定大小输出缓冲区的新缓冲字符输出流。

\*\*方法摘要：

void close()

关闭此流，但要先刷新它。

void flush()

刷新该流的缓冲。

void newLine()

写入一个行分隔符。

void write(char[] cbuf, int off, int len)

写入字符数组的某一部分。

void write(int c)

写入单个字符。

void write(String s, int off, int len)

写入字符串的某一部分。

(3)OutputStreamWriter：字节通向字符的转换流

\*\*构造方法：

OutputStreamWriter(OutputStream out)

创建使用默认字符编码的 OutputStreamWriter。

\*\*方法摘要：

void write(char[] cbuf, int off, int len)

写入字符数组的某一部分。

void write(int c)

写入单个字符。

void write(String str, int off, int len)

写入字符串的某一部分。

(4)PrintWriter:

\*\*构造方法：

PrintWriter(String fileName)

创建具有指定文件名称且不带自动行刷新的新 PrintWriter。

PrintWriter(File file)

使用指定文件创建不具有自动行刷新的新 PrintWriter。

PrintWriter(Writer out)

创建不带自动行刷新的新 PrintWriter。

PrintWriter(Writer out, boolean autoFlush)

创建新 PrintWriter。

PrintWriter(OutputStream out)

根据现有的 OutputStream 创建不带自动行刷新的新 PrintWriter。

PrintWriter(OutputStream out, boolean autoFlush)

通过现有的 OutputStream 创建新的 PrintWriter。

\*\*方法摘要：

PrintWriter append(char c)

将指定字符添加到此 writer。

void print(各种类型的数据：)

打印各种类型的数据

void println(各种类型的数据：)：自动换行

打印各种类型的数据

void write(char[] buf)

写入字符数组。

void write(char[] buf, int off, int len)

写入字符数组的某一部分。

void write(int c)

写入单个字符。

void write(String s)

写入字符串。

void write(String s, int off, int len)

写入字符串的某一部分。

\*\*读取流：

(1)FileReader:

\*\*构造方法：

FileReader(String fileName)

在给定从中读取数据的文件名的情况下创建一个新 FileReader。

FileReader(File file)

在给定从中读取数据的 File 的情况下创建一个新 FileReader。

FileReader(FileDescriptor fd)

在给定从中读取数据的 FileDescriptor 的情况下创建一个新 FileReader。

\*\*方法摘要：和Reader基类方法一致：

abstract void close() 关闭该流并释放与之关联的所有资源。

int read() 读取单个字符。

int read(char[] cbuf) 将字符读入数组

abstract int read(char[] cbuf, int off, int len) 将字符读入数组的某一部分。

long skip(long n) 跳过字符。

(2)BufferedReader:

\*\*构造方法：

BufferedReader(Reader in)

创建一个使用默认大小输入缓冲区的缓冲字符输入流。

\*\*方法摘要：

int read()

读取单个字符。

int read(char[] cbuf, int off, int len)

将字符读入数组的某一部分。

String readLine()

读取一个文本行。

(3)InputStreamReader：字符通向字节的桥梁：

\*\*构造方法：

InputStreamReader(InputStream in)

创建一个使用默认字符集的 InputStreamReader。

\*\*方法摘要:

int read() 读取单个字符。

int read(char[] cbuf) 将字符读入数组

abstract int read(char[] cbuf, int off, int len) 将字符读入数组的某一部分。

long skip(long n) 跳过字符。

(4)LineNumberReader：

\*\*构造方法：

LineNumberReader(Reader in)

使用默认输入缓冲区的大小创建新的行编号 reader。

\*\*方法摘要：

int read()

读取单个字符。

int read(char[] cbuf, int off, int len)

将字符读入数组中的某一部分。

String readLine()

读取文本行。

long skip(long n)

跳过字符。

int getLineNumber()

获得当前行号。

void setLineNumber(int lineNumber)

设置当前行号。

6、IO流常见需求：

\*\*\*\*字符流：

(1)需求1：在硬盘上创建一个文件并写入信息

用字符写入流：FileWriter

FileWriter fw = new FileWriter("g:\\filewriter.txt");

fw.write("输入信息");

fw.write("也可以写入字符数组".toCharArray());

fw.flush();

fw.close();

(2)需求2：在原有文件上续写数据

FileWriter fw = new FileWriter("g:\\filewriter.txt",true);

fw.write("还可以续写信息");

fw.write("也可以写入字符数组".toCharArray());

fw.flush();

fw.close();

(3)需求3：读取硬盘上的文本文件，并将数据打印在控制台

FileReader fr = new FileReader("g:\\filewriter.txt");

\*\*第一种读取方法：一个一个字节的读

int ch = 0;

ch = fr.read();

sop((char)ch);

fr.close();

\*\*第二种读取方法：利用数组来提高效率

char[] buf = new char[1024];

int len = 0;

while((len = fr.read(buf))!=-1)

{

sop(new String(buf,0,len));

}

fr.close();

(4)需求4:拷贝文本文件

利用缓冲区提高数据读写效率

(无缓冲区就相当于一滴一滴的喝水，有缓冲区就相当于一杯一杯的喝水)

BufferedReader bufr = new BufferedReader(new FileReader("g:\\filewriter.txt"));

BufferedWriter bufw = new BufferedWriter(new FileWriter("d:\\copyfilewriter.txt"));

String line = null;

while((line = bufr.readLine())!=null)

{

burw.write(line);

bufw.newLine();

bufw.flush();

}

bufr.close();

bufw.close();

\*\*\*\*字节流：字节流写入时没有刷新

(1)需求1：在硬盘上创建一个文件并写入信息(字节流写入时没有刷新)

FileOutputStream fos = new FileOutputStream("g:\\filestream.txt");

fos.write(97);//写入一个字节,int：97代表写入char：a

fos.write("也可以写入字节数组".getBytes());//通常使用此种方式写入，直观！

fos.close();

(2)需求2：在硬盘已有文件上续写数据(字节流写入时没有刷新)

FileOutputStream fos = new FileOutputStream("g:\\filestream.txt",true);

fos.write("创建字节写入流时，传进去一个true参数就可以继续写入信息".getBytes());

fos.close();

(3)需求3：读取硬盘上的文件

FileInputStream fis = new FileInputStream("g:\\filestream.txt");

\*\*第一种读法：一个字节一个字节的读(此种读法慢)

int ch = 0;

while((ch = fis.read())!=-1)

{

sop((char)ch);

}

\*\*第一种读法：利用字节数组读(此种读法效率有一定提高)

byte[] buf = new byte[1024];

int len = 0;

while((len = fis.read())!=-1)

{

sop(new String(buf,0,len));

}

(4)需求4:拷贝字节文件，如图片或者MP3或者电影

\*\*第一种拷贝：不带缓冲区(慢，还是效率问题)

复制代码

1 FileInputStream fis = new FileInputStream("g:\\1.mp3");

2 FileOutputStream fos = new FileOutputStream("g:\\copy1.mp3");

3 byte[] buf = new byte[1024];

4 int len = 0;

5 while((len = fis.read(buf))!=-1)

6 {

7 fos.(buf,0,len);//字节流写入无需刷新

8 }

9 fis.close();

10 fos.close();

11 \*\*第二种拷贝：带缓冲区，高效

12 BufferedInputStream bufi = new BufferedInputStream(new FileInputStream("g:\\1.mp3"));

13 BufferedOutputStream bufo = new BufferedOutputStream(new FileOutputStream("g:\\copy1.mp3"));

14 int ch = 0;

15 while((ch = bufi.read())!=-1)

16 {

17 bufo.write(ch);

18 }

19 bufi.close();

20 bufo.close();

21 \*\*\*\*转换流：

22 (1)需求1：读取一个键盘录入

23 InputStream in = System.in;//创建一个键盘录入流，流不关则可以一直录入

24 int by1 = in.read();//一次读一个字节

25 int by2 = in.read();//一次读一个字节

26 sop(by1);//假设键盘录入的是abcd,则打印a

27 sop(by2);//假设键盘录入的是abcd,则打印b

28 in.close();

29 (2)需求2：键盘录入一行数据打印一行数据，如果录入的是over则结束录入

30 InputStream in = System.in;

31 StringBuilder sb = new StringBuilder();

32 while(true)

33 {

34 int ch = in.read();

35 if(ch=='\r')

36 continue;

37 if(ch=='\n')

38 {

39 String line = sb.toString();

40 if("over".equals(line))

41 break;

42 sop(line.toUpperCase());//输出大写

43 sb.delete(0.sb.length());//清除上一行录入的数据

44

45 }

46 else

47 sb.append((char)ch);

48 }

49 in.close();

复制代码

(3)需求3：发现需求2中其实就是读一行的原理，故引入字节通向字符的桥梁：InputStreamReader

为提高效率加入缓冲区：

复制代码

BufferedReader bufr = new BufferedReader(new InputStreamReader(System.in));

String line = null;

while((line = bufr.readLine())!=null)

{

if("over".equals(line))

break;

sop(line.toUpperCase());//输出大写

}

复制代码

bufr.close();

(4)需求4：键盘录入数据并打印到控制台

复制代码

1 BufferedReader bufr = new BufferedReader(new InputStreamReader(System.in));

2 BufferedWriter bufw = new BufferedWriter(new OntputStreamWriter(System.out));

3 String line = null;

4 while((line = bufr.readLine())!=null)

5 {

6 if("over".equals(line))

7 break;

8 bufw.write(line.toUpperCase());

9 bufw.newLine();

10 bufw.flush();

11 }

12 bufr.close();

13 bufw.close();

复制代码

(5)需求5:将键盘录入的数据存储到硬盘文件

则只需将(4)中的

复制代码

1 BufferedWriter bufw = new BufferedWriter(new OntputStreamWriter(System.out));

2 改为：

3 BufferedWriter bufw = new BufferedWriter(new OntputStreamWriter(new FileWriter("g:\\demo.txt")));

4 即：

5 BufferedReader bufr = new BufferedReader(new InputStreamReader(System.in));

6 BufferedWriter bufw = new BufferedWriter(new OntputStreamWriter(new FileWriter("g:\\demo.txt")));

7 String line = null;

8 while((line = bufr.readLine())!=null)

9 {

10 if("over".equals(line))

11 break;

12 bufw.write(line.toUpperCase());

13 bufw.newLine();

14 bufw.flush();

15 }

16 bufr.close();

17 bufw.close();

复制代码

(6)需求6：将硬盘文件的数据打印到控制台

则只需将(4)中的

复制代码

1 BufferedReader bufr = new BufferedReader(new InputStreamReader(System.in));

2 改为：

3 BufferedReader bufr = new BufferedReader(new InputStreamReader(new FileReader("g:\\demo.txt")));

4 即：

5 BufferedReader bufr = new BufferedReader(new InputStreamReader(new FileReader("g:\\demo.txt")));

6 BufferedWriter bufw = new BufferedWriter(new OntputStreamWriter(System.out));

7 String line = null;

8 while((line = bufr.readLine())!=null)

9 {

10 if("over".equals(line))

11 break;

12 bufw.write(line.toUpperCase());

13 bufw.newLine();

14 bufw.flush();

15 }

16 bufr.close();

17 bufw.close();

复制代码

7、流操作的规律：

\*\*\*\*流操作的难点：流对象很多，不知道具体用哪个

\*\*\*\*规律：

(1)第一步：先明确源和目的

源：

文本：用Reader

字节：用InputStream

目的：

文本：用Writer

字节：用OutputStream

(2)第二步：明确是不是纯文本

是：用字符流；

不是：用字节流

(3)第三步：明确流体系后，通过设备来明确具体使用哪个流对象

源设备：

键盘：System.in

硬盘：文件流File

内存：数组流ArrayStream

目的设备：

键盘：System.out

硬盘：文件流File

内存：数组流ArrayStream

8、File类

构造方法：

File(String pathname)

通过将给定路径名字符串转换为抽象路径名来创建一个新 File 实例。

File(String parent, String child)

根据 parent 路径名字符串和 child 路径名字符串创建一个新 File 实例。

File(File parent, String child)

根据 parent 抽象路径名和 child 路径名字符串创建一个新 File 实例。

方法摘要：

(1)创建：

boolean createNewFile()

当且仅当不存在具有此抽象路径名指定名称的文件时，不可分地创建一个新的空文件。

boolean mkdir()

创建一级文件夹

boolean mkdirs()

创建多级文件夹

(判断)：

boolean canExecute()

测试应用程序是否可以执行此抽象路径名表示的文件。

boolean canRead()

测试应用程序是否可以读取此抽象路径名表示的文件。

boolean canWrite()

测试应用程序是否可以修改此抽象路径名表示的文件。

int compareTo(File pathname)

按字母顺序比较两个抽象路径名。

boolean isAbsolute()

测试此抽象路径名是否为绝对路径名。

boolean isDirectory()

测试此抽象路径名表示的文件是否是一个目录。

boolean isFile()

测试此抽象路径名表示的文件是否是一个标准文件。

boolean isHidden()

测试此抽象路径名指定的文件是否是一个隐藏文件。

boolean exists()

测试此抽象路径名表示的文件或目录是否存在。

(3)获取：

String getParent()

返回此抽象路径名父目录的路径名字符串；如果此路径名没有指定父目录，则返回 null。

File getParentFile()

返回此抽象路径名父目录的抽象路径名；如果此路径名没有指定父目录，则返回 null。

String getName()

返回由此抽象路径名表示的文件或目录的名称。

String getPath()

将此抽象路径名转换为一个路径名字符串。

String getAbsolutePath()

返回此抽象路径名的绝对路径名字符串。

File getAbsoluteFile()

返回此抽象路径名的绝对路径名形式。

(4)删除：

boolean delete()

删除此抽象路径名表示的文件或目录。

oid deleteOnExit()

在虚拟机终止时，请求删除此抽象路径名表示的文件或目录。

(5)获取全部：(非常重要！！！)

String[] list()

返回一个字符串数组，这些字符串指定此抽象路径名表示的目录中的文件和目录。

String[] list(FilenameFilter filter)

返回一个字符串数组，这些字符串指定此抽象路径名表示的目录中满足指定过滤器的文件和目录。

File[] listFiles()

返回一个抽象路径名数组，这些路径名表示此抽象路径名表示的目录中的文件。

File[] listFiles(FileFilter filter)

返回抽象路径名数组，这些路径名表示此抽象路径名表示的目录中满足指定过滤器的文件和目录。

\*\*\*\*FilenameFilter接口只有一个方法：

boolean accept(File dir, String name)

测试指定文件是否应该包含在某一文件列表中。

\*\*\*\*FileFilter接口只有一个方法：

boolean accept(File dir, String name)

测试指定文件是否应该包含在某一文件列表中。

8、File类常见需求：

(1)文件名过滤:列出给定目录的所有.java文件

复制代码

public void showFileName(File file)

{

String[] filenames = file.list(new FilenameFilter()//匿名内部类

{

public boolean accept(File dir,String name)//复写唯一方法

{

return name.endsWith(".java");//列出所有.java文件

}

});

}

复制代码

(2)列出指定目录下的所有文件和文件夹(递归)

复制代码

1 \*\*示例1：不带层次递归：

2 public static void showDir(File dir)

3 {

4 File[] files = dir.listFile();

5 for(int i = 0;i<files.length;i++)

6 {

7 if(files[i].isDirectory&&!files[i].isHidden())

8 showDir(files[i]);

9 else

10 sop(files[i]);

11 }

12 }

13 \*\*示例2：带层次递归：

14 public static void showDir(File dir,int level)

15 {

16 sop(getLevel(level)+C);//进来先打印层次和目录

17 level++;

18 File[] files = dir.listFile();

19 for(int i = 0;i<files.length;i++)

20 {

21 if(files[i].isDirectory&&!files[i].isHidden())

22 showDir(files[i]);

23 else

24 sop(getLevel(level)+files[i]);//是文件就打印层次和目录

25 }

26 }

27 public static String getLevel(int level)

28 {

29 sop("|--");

30 StringBuilder sb = new StringBuilder();

31 for(int i=0;i<level;i++)

32 {

33 sb.inset(0."| ")

34 }

35 return sb.toString();

36 }

37 (3)需求：删除带内容的目录：

38 public static void removeDir(File dir)

39 {

40 File[] files = file.listFile();

41 for(int i = 0;i<files.length;i++)

42 {

43 if(files[i].isDirectory&&!files[i].isHidden())

44 removeDir(files[i]);//如果是文件夹则继续调用函数

45 else//如果是文件则删除。注意删除的时候打印删除的结果，防止误删或者重删的情况

46 sop(files[i].toString()+"::"+files[i].delete());

47 }

48 sop(dir+"::"+dir.delete());

49 }

复制代码

(4)需求：将制定目录下的java文件的绝对路径存储到文本文件中。

思路：

\*\*对指定目录进行递归

\*\*获取递归过程中所有java文件的路径

\*\*将这些路径存储到集合中

\*\*将集合中的数据写入文件中

//对指定目录进行递归并将所以Java文件存储到集合中

复制代码

public static void getFileName(File file,ArrayList<File> arraylist){

File[] files = file.listFiles();

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

if(files[i].isDirectory()&&!files[i].isHidden()){

getFileName(files[i],arraylist);

}else{

if(files[i].getName().endsWith(".java")){

arraylist.add(files[i]);

}

}

}

}

//将集合中所有数据存储到新文件中

public static void saveFileToNewDir(ArrayList<File> arraylist,File newDir){

BufferedWriter bufw = null;

try {

bufw = new BufferedWriter(new FileWriter(newDir));

for (File file : arraylist) {

String fileAbsolutePath = file.getAbsolutePath();

bufw.write(fileAbsolutePath);

bufw.newLine();

bufw.flush();

}

} catch (Exception e) {

System.out.println("文件写入失败");

}finally{

try {

if(bufw!=null)

bufw.close();

} catch (Exception e2) {

System.out.println("文件写入流关闭失败");

}

}

}

复制代码

9、Properties

(1)Properties是HashTable的子类，具备Map集合的特点，里面存储的是键值对

(2)Properties是IO流合集合相结合的集合容器

(3)Properties的特点是可以用于存储键值对形式的配置文件

(4)构造方法：

Properties()

创建一个无默认值的空属性列表。

Properties(Properties defaults)

创建一个带有指定默认值的空属性列表。

(5)方法摘要：

Object setProperty(String key, String value)

调用 Hashtable 的方法 put。

String getProperty(String key)

用指定的键在此属性列表中搜索属性。

void load(InputStream inStream)

从输入流中读取属性列表（键和元素对）。

void load(Reader reader)

按简单的面向行的格式从输入字符流中读取属性列表（键和元素对）。

void list(PrintStream out)

将属性列表输出到指定的输出流。

void list(PrintWriter out)

将属性列表输出到指定的输出流。

void store(OutputStream out, String comments)

以适合使用 load(InputStream) 方法加载到 Properties 表中的格式，

将此 Properties 表中的属性列表（键和元素对）写入输出流。

void store(Writer writer, String comments)

以适合使用 load(Reader) 方法的格式，将此 Properties 表中的

属性列表（键和元素对）写入输出字符。

Set<String> stringPropertyNames()

返回此属性列表中的键集，其中该键及其对应值是字符串，如果在主属性列表中

未找到同名的键，则还包括默认属性列表中不同的键

复制代码

1 (6)Properties代码示例：

2 public static void show()

3 {

4 Properties prop = new Properties();

5 prop.setProperty("张三","26");

6 prop.setProperty("李四","30");

7 prop.setProperty("王五","35");

8 sop(prop);

9 String value = prop.getProperty("张三");

10

11 Set<String> keys = prop.stringPropertyName();

12 for(String key : values)

13 {

14 sop(key+":"+prop.getPropety(key));

15 }

16 }

复制代码

(7)需求：记录应用程序的使用次数，如果使用次数已到，则提示用户注册。

思路：

\*\*第一次使用时建立一个配置文件用于记录使用次数

\*\*每次使用都加载该配置文件，并先判断已使用次数

\*\*每次使用完使用次数加1，写入配置文件

复制代码

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

2 Properties prop = new Properties();//定义Properties，用来和IO流结合

3 File file = new File("library\\time.ini");//配置文件

4 if(!file.exists())

5 file.createNewFile();//如果文件不存在则创建文件(用于第一次使用时创建文件)

6 FileInputStream fis = new FileInputStream(file);//定义字节读取流，读取配置文件中记录的使用次数

7 prop.load(fis);//载入流，以获取文件中配置的键值对

8 int count = 0;//定义使用次数

9 String countValue = prop.getProperty("time");//通过键获取值

10 if(countValue!=null){//第一次时countValue为null

11 count = Integer.parseInt(countValue);//将字符串次数变成数字次数

12 if(count>3){

13 System.out.println("您使用次数已到，继续使用请注册！");

14 return;

15 }

16 }

17 count++;//如果使用次数未到则次数加1

18 prop.setProperty("time", count+"");//配置新的键值对

19 FileWriter fos = new FileWriter(file);

20 prop.store(fos, "这是应用程序使用次数的配置文件");//将新的键值对写入文件

21 fis.close();

22 fos.close();

23 }

复制代码

10、IO中的其他流：

(1)打印流：

\*\*PrintWriter:字符打印流

\*\*\*\*构造方法：

PrintWriter(String fileName)

创建具有指定文件名称且不带自动行刷新的新 PrintWriter。

PrintWriter(File file)

使用指定文件创建不具有自动行刷新的新 PrintWriter。

PrintWriter(Writer out)

创建不带自动行刷新的新 PrintWriter。

PrintWriter(Writer out, boolean autoFlush)

自动刷新

PrintWriter(OutputStream out)

根据现有的 OutputStream 创建不带自动行刷新的新 PrintWriter。

PrintWriter(OutputStream out, boolean autoFlush)

自动刷新

\*\*\*\*方法摘要：

PrintWriter append(char c)

将指定字符添加到此 writer。

void close()

关闭该流并释放与之关联的所有系统资源。

void flush()

刷新该流的缓冲。

void print(Object obj)

打印对象。

void print(String s)

打印字符串。

void println()

通过写入行分隔符字符串终止当前行。

\*\*PrintStream:字节打印流

\*\*\*\*构造方法：

PrintStream(String fileName)

创建具有指定文件名称且不带自动行刷新的新打印流。

PrintStream(File file)

创建具有指定文件且不带自动行刷新的新打印流。

PrintStream(OutputStream out)

创建新的打印流。

PrintStream(OutputStream out, boolean autoFlush)

创建新的打印流。

\*\*\*\*方法摘要：

PrintWriter append(char c)

将指定字符添加到此 writer。

void close()

关闭该流并释放与之关联的所有系统资源。

void flush()

刷新该流的缓冲。

void print(Object obj)

打印对象。

void print(String s)

打印字符串。

void println()

通过写入行分隔符字符串终止当前行。

(2)对象系列化：

\*\*对象实体化：找一个介质，能长期的存储对象。

\*\*对象的属性在Java程序中，都是存在于对内存中，随着对象的消失而消失，

而ObjectOutputStream可以将对象实体化

\*\*Serializable接口没有一个方法，也就是说其是一个标记接口。比如盖章的猪肉才是安全的。

\*\*只有实现Serializable接口的子类才能被ObjectOutputStream系列化写入流，当某个

类实现该接口后，会被Java自动分配UID号，以便编译器识别，区分不同对象。

\*\*用ObjectOutputStream系列化的对象存储到文件后，该文件是乱码，也就是不可读的

的用ObjectInputStream读取该类对象的属性。

\*\*由于对象是有Java给对象分配相应的UID号，而UID号是根据对象的属性不同而分配的。

当一个类对象被系列化到文件后，如果该类改动了对象的属性，比如将某个成员变量变成私有

则该对象再用ObjectInputStream读取时会报异常，也就是说该系列化到文件的对象不能再被使用了

那么，要想继续使用属性被改动后的对象，我们可以自定义给对象分配UID号，让UID号不随对象的属性

变化而变化。

自定义对象分配UID方法如下：

public static final long serialVersion UID = 43L;

\*\*注意：

静态不能被系列化，因为静态成员变量实在内存的方法区，而ObjectOutputStream只能

对对内存里面的数据进行系列化

被transient修饰的非静态成员变量也不能被系列化

被系列化的对象存储到文件中，该文件是不可读的，所以该文件的扩展名一般

不写成.txt，通常后缀名写.object

\*\*ObjectOutputStream

\*\*ObjectInputStream

(3)管道流：

PipedInputStream

PipedOutputStream

(4)随机访问文件：RandomAccess(重要！！！)

\*\*自身具备读写方法(很牛逼！又可以读又可以写)

\*\*通过skipByte(int x)和seek(int x)来达到随机访问文件

\*\*该类不是IO体系子类，而是直接继承Object，但它是IO包中的成员，因为它具备读写方法

\*\*该类内部封装了数组，而且通过指针对数组的元素进行操作，可以通过getFilePoint获取指针位置

同时可以通过seek改变指针位置

\*\*该类完成读写的原理是内部封装了字节输入输出流

\*\*通过该类的构造看出，该类只能操作文件，而且操作的文件只能有固定模式：

"r":只读

"rw":读写

"rws":

"red":

\*\*构造方法：

RandomAccessFile(File file, String mode)

创建从中读取和向其中写入（可选）的随机访问文件流，该文件由 File 参数指定。

RandomAccessFile(String name, String mode)

创建从中读取和向其中写入（可选）的随机访问文件流，该文件具有指定名称。

\*\*方法摘要：

void write(byte[] b)

将 b.length 个字节从指定 byte 数组写入到此文件，并从当前文件指针开始。

void write(byte[] b, int off, int len)

将 len 个字节从指定 byte 数组写入到此文件，并从偏移量 off 处开始。

void write(int b)

向此文件写入指定的字节。

int read()

从此文件中读取一个数据字节。

int read(byte[] b)

将最多 b.length 个数据字节从此文件读入 byte 数组。

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

将最多 len 个数据字节从此文件读入 byte 数组。

String readLine()

从此文件读取文本的下一行。

long getFilePointer()

返回此文件中的当前偏移量。

long length()

返回此文件的长度。

void seek(long pos)

设置到此文件开头测量到的文件指针偏移量，在该位置发生下一个读取或写入操作。

(4)操作基本数据类型的流对象：DateStream

(5)操作字节数组流：

ByteArrayInputStream

ByteArrayOutputStream

11、IO流转换流的字符编码

(1)字符流的出现为了方便操作字符，更重要的是加入了编码转换

(2)通过子类转换流来完成

InputStreamReander

OutputStreamWriter

(3)在两个子类对象进行构造的时候可以加入编码表

(4)编码表：

将各个国家的文字用二进制数字表示并一一对应，形成一张表，这就是编码表

(5)常见的编码表：

\*\*ASCII：美国标准信息交换码，用一个字节的七位表示

\*\*ISO8859-1：拉丁码表，欧洲码表，用一个字节的八位表示

\*\*GB2312：中文编码表，用两个字节表示

\*\*GBK：中文编码表升级，融合录入更多的中文字符，用两个字节表示，为避免和老美重复

两字节的最高位都是1，即汉字都是用负数表示

\*\*Unicode：国际标准码，融合了多种文字，所有文字都用两个字节表示

\*\*UTF-8：用一个字节到三个字节表示。

注：Unicode能识别中文，UTF-8也能识别中文，但两种编码表示一个汉字所用的字节数不同

Unicode用两个字节，UTF-8用三个字节，故涉及到编码转换。

(6)在流中涉及编码表的转换只有转换流：

InputStreamReander

OutputStreamWriter

(7)代码示例：

复制代码

1 public static void write() throws IOException

2 {

3 OutputStreamWriter osw1 = new OutputStreamWriter(new FileOutputStream("gbk.txt"),"GBK");

4 osw1.write("你好");

5 osw1.close();

6

7 OutputStreamWriter osw2 = new OutputStreamWriter(new FileOutputStream("utf-8.txt"),"UTF-8");

8 osw2.write("你好");

9 osw2.close();

10 }

11 public static void read() throws IOException

12 {

13 InputStreamReader isr = new InputStreamReader(new FileInputStream("gbk.txt"),"GBK");

14 byte[] buf = new byte[1024];

15 int len = isr.read(buf);

16 sop(new String(buf,0,len));

17 }

18 (8)

复制代码

编码解码

编码：字符串变成字节数组：String-->getBytes()-->byte[]()

解码：字节数组变成字符串：byte[]-->new String(byte[],0,len)-->String

(9)代码示例：

public static void main(String[] args)

{

//编码解码1：默认编码

String str1 = "你好";

byte[] buf1 = str1.getBytes();//默认解码：Unicode，四个字节

//编码解码2：指定编码

String str2 = "你好";

byte[] buf2 = str2.getBytes("UTF-8");//指定解码：UTF-8,六个字节

//编码解码3：编码正确解码错误

String str3 = "你好";

byte[] buf3 = str3.getBytes("GBK");//指定编码：GBK,四个字节

String str3 = new String(buf3,"ISO8859-1");//错误解码

//编码解码4：错误编码正确解码

String str4 = "你好";

byte[] buf4 = str4.getBytes("ISO8859-1");//错误编码

String str4 = new String(buf4,"GBK");//正确解码，读不出来

//编码解码5：编码对了，但是解码错误了，怎么办呢？

//此时可以将错误的解码再错编回去，载用正确编码解码

String str5 = "你好";

byte[] buf5 = str5.getBytes("GBK");//正确编码

String str6 = new String(buf5,"ISO8859-1");//错误解码，读不出来

byte[] buf6 = str6.getBytes("ISO8859-1");//再错误编码

String str7 = new String(buf6,"GBK");//再正确解码，这样就可以读出来了

}

六、网络编程：

1、网络编程概述

(1)网络模型

OSI参考模型

TCP/IP参考模型

(2)网络通讯要素

IP地址

端口号

传输协议

(3)网络通讯前提：

\*\*找到对方IP

\*\*数据要发送到指定端口。为了标示不同的应用程序，所以给这些网络应用程序都用数字进行标示

。这个表示就叫端口。

\*\*定义通信规则。这个规则称为通信协议，国际组织定义了通用协议TCP/IP

(4)计算机网络：

是指将地理位置不同的具有独立功能的多台计算机及其外部设备，

通过通信线路连接起来，在网络操作系统，网络管理软件及网络通信协议的管理和协调下，

实现资源共享和信息传递的计算机系统。

(5)IP地址：

IP地址 = 网络号码+主机地址

A类IP地址:第一段号码为网络号码，剩下的三段号码为本地计算机的号码

B类IP地址:前二段号码为网络号码，剩下的二段号码为本地计算机的号码

C类IP地址:前三段号码为网络号码，剩下的一段号码为本地计算机的号码

特殊地址:

127.0.0.1 回环地址,可用于测试本机的网络是否有问题. ping 127.0.0.1

ipconfig:查看本机IP地址

xxx.xxx.xxx.0 网络地址

xxx.xxx.xxx.255 广播地址

A类 1.0.0.1---127.255.255.254 10.X.X.X是私有地址(私有地址就是在互联网上不使用，而被用在局域网络中的地址) (2)127.X.X.X是保留地址，用做循环测试用的。

B类 128.0.0.1---191.255.255.254 172.16.0.0---172.31.255.255是私有地址。169.254.X.X是保留地址。

C类 192.0.0.1---223.255.255.254 192.168.X.X是私有地址

D类 224.0.0.1---239.255.255.254

E类 240.0.0.1---247.255.255.254

(6)各种网络分类方式

A:按网络覆盖范围划分

　　局域网(几米至10公里以内) 　　城域网(10~100公里) 　　广域网(几百公里到几千公里) 　　国际互联网

B:按网络拓扑结构划分

　　总线型网络 　　星形网络 　　环型网络 　　树状网络 　　混合型网络

C:按传输介质划分

　　有线网 　　无线网

D:按网络使用性质划分

　　公用网 　　专用网

(7)虚拟专用网络（Virtual Private Network ，简称VPN)指的是在公用网络上建立专用网络的技术。

其之所以称为虚拟网，主要是因为整个VPN网络的任意两个节点之间的连接并没有传统专网

所需的端到端的物理链路，而是架构在公用网络服务商所提供的网络平台，如Internet、

ATM(异步传输模式〉、Frame Relay （帧中继）等之上的逻辑网络，

用户数据在逻辑链路中传输。它涵盖了跨共享网络或公共网络的封装、

加密和身份验证链接的专用网络的扩展。VPN主要采用了隧道技术、加解密技术、

密钥管理技术和使用者与设备身份认证技术。

(8)网络模型：

\*\*\*\*OSI模型

应用层

表示层

会话层

传输层

网络层

数据连接层

物理层

\*\*\*\*TCP/IP模型

应用层

传输层

网际层

主机至网络层

2、TCP和UDP

(1)UDP和TCP的区别：

UDP

将数据及源和目的封装成数据包中，不需要建立连接

每个数据报的大小在限制在64k内

因无连接，是不可靠协议

不需要建立连接，速度快

TCP

建立连接，形成传输数据的通道。

在连接中进行大数据量传输

通过三次握手完成连接，是可靠协议

必须建立连接，效率会稍低

注：三次握手：

第一次：我问你在么？

第二次：你回答在。

第三次：我反馈哦我知道你在。

3、Socket(UDP传输)

\*\*Socket就是为网络服务提供的一种机制。

\*\*通信的两端都有Socket。

\*\*网络通信其实就是Socket间的通信。

\*\*数据在两个Socket间通过IO传输。

\*\*玩Socket主要就是记住流程，代码查文档就行

(1)UDP传输：DatagramSocket与DatagramPacket

\*\*发送端：

建立DatagramSocket服务；

提供数据，并将数据封装到字节数组中；

创建DatagramPacket数据包，并把数据封装到包中，同时指定IP和接收端口

通过Socket服务，利用send方法将数据包发送出去；

关闭DatagramSocket和DatagramPacket服务。

\*\*接收端：

建立DatagramSocket服务，并监听一个端口；

定义一个字节数组和一个数据包，同时将数组封装进数据包；

通过DatagramPacket的receive方法，将接收的数据存入定义好的数据包；

通过DatagramPacke关闭t的方法，获取发送数据包中的信息；

关闭DatagramSocket和DatagramPacket服务。

DatagramSocket与DatagramPacket方法摘要：

\*\*\*\*\*DatagramSocket

构造方法：

DatagramSocket()

构造数据报套接字并将其绑定到本地主机上任何可用的端口。

DatagramSocket(int port)

创建数据报套接字并将其绑定到本地主机上的指定端口。

DatagramSocket(int port, InetAddress laddr)

创建数据报套接字，将其绑定到指定的本地地址。

方法摘要:

void close()

关闭此数据报套接字。

InetAddress getInetAddress()

返回此套接字连接的地址。

InetAddress getLocalAddress()

获取套接字绑定的本地地址。

int getPort()

返回此套接字的端口。

void receive(DatagramPacket p)

从此套接字接收数据报包。

void send(DatagramPacket p)

从此套接字发送数据报包。

\*\*\*\*DatagramPacket

构造方法：

DatagramPacket(byte[] buf, int length)

构造 DatagramPacket，用来接收长度为 length 的数据包。

DatagramPacket(byte[] buf, int length, InetAddress address, int port)

构造数据报包，用来将长度为 length 的包发送到指定主机上的指定端口号。

InetAddress getAddress()

返回某台机器的 IP 地址，此数据报将要发往该机器或者是从该机器接收到的。

byte[] getData()

返回数据缓冲区。

int getLength()

返回将要发送或接收到的数据的长度。

int getPort()

返回某台远程主机的端口号，此数据报将要发往该主机或者是从该主机接收到的。

代码示例：

\*\*\*\*发送端：

复制代码

1 \*\*\*\*发送端

2 class UDPSend

3 {

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

5 {

6 DatagramSocket ds = new DatagramSocket();

7 byte[] buf = "这是UDP发送端".getBytes();

8 DatagramPacket dp = new DatagramPacket(

9 buf,buf.length,InetAddress.getByName("192.168.1.253"),10000);

10 ds.send(dp);

11 ds.close();

12 }

13 }

14 \*\*\*\*接收端

15 class UDPRece

16 {

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

18 {

19 DatagramSocket ds = new DatagramSocket(10000);

20 byte[] buf = new byte[1024];

21 DatagramPacket dp = new DatagramPacket(buf,buf.length);

22 ds.receive(dp);//将发送端发送的数据包接收到接收端的数据包中

23 String ip = dp.getAddress().getHosyAddress();//获取发送端的ip

24 String data = new String(dp.getData(),0,dp.getLength());//获取数据

25 int port = dp.getPort();//获取发送端的端口号

26 sop(ip+":"+data+":"+port);

27 ds.close();

28 }

29 }

30 需求1：UDP键盘录入数据，并发送给接收端

31 发送端：

32 class UDPSend

33 {

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

35 {

36

37 DatagramSocket ds = new DatagramSocket();

38 BufferedReader bufr = new BufferedReader(new InputStreamReader(System.in));

39 String line = null;

40 while((line = bufr.readLine())!=null)

41 {

42 if("886".equals(line))

43 break;

44 byte[] buf = line.getBytes();

45 DatagramPacket dp = new DatagramPacket(

46 buf,buf.length,InetAddress.getByName("192.168.1.253"),10000);

47 ds.send(dp);

48 }

49 ds.close();

50 }

51

52 }

53 接收端：

54 class UDPRece

55 {

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

57 {

58 DatagramSocket ds = new DatagramSocket(10000);

59 while(true)

60 {

61 byte[] buf = new byte[1024];

62 DatagramPacket dp = new DatagramPacket(buf,buf.length);

63 ds.receive(dp);//将发送端发送的数据包接收到接收端的数据包中

64 String ip = dp.getAddress().getHosyAddress();//获取发送端的ip

65 String data = new String(dp.getData(),0,dp.getLength());//获取数据

66 int port = dp.getPort();//获取发送端的端口号

67 sop(ip+":"+data+":"+port);

68 ds.close();

69 }

70 }

71

72 }

复制代码

需求2：编写简单的聊天工具

思路：

使用多线程技术

复制代码

1 发送端：

2 class UDPSend implements Runnable

3 {

4 private DatagramSocket ds;

5 public UDPSend(){}

6 public UDPSend(DatagramSocket ds)

7 {

8 this.ds=ds;

9 }

10 public void run()

11 {

12 try

13 {

14 BufferedReader bufr = new BufferedReader(

15 new InputStreamReader(System.in));

16 String line = null;

17 while((line = bufr.readLine())!=null)

18 {

19 if("886".equals(line))

20 break;

21 byte[] buff = line.getBytes();

22 DatagramPacket dp = new DatagramPacket(

23 buf,buf.length,InetAddress.getByName("192.168.1.253"),10000);

24 ds.send(dp);

25 }

26 }

27 catch(Exception e)

28 {

29 throw new RuntimeException("发送失败");

30 }

31 }

32 }

33 接收端：

34 class UDPRece implements Runnable

35 {

36 private DatagramSocket ds;

37 public UDPSend(){}

38 public UDPSend(DatagramSocket ds)

39 {

40 this.ds=ds;

41 }

42 public void run()

43 {

44 try

45 {

46 while(true)

47 {

48 byte[] buf = new byte[1024];

49 DatagramPacket dp = new DatagramPacket(buf,buf.length);

50 ds.receive(dp);//将发送端发送的数据包接收到接收端的数据包中

51 String ip = dp.getAddress().getHosyAddress();//获取发送端的ip

52 String data = new String(dp.getData(),0,dp.getLength());//获取数据

53 int port = dp.getPort();//获取发送端的端口号

54 sop(ip+":"+data+":"+port);

55 }

56 }

57 catch(Exception e)

58 {

59 throw new RuntimeException("接收失败");

60 }

61 }

62 }

63 测试类：

64 class UDPTest

65 {

66 public static void main(String[] args)

67 {

68 DatagramSocket sendSocket = new DatagramSocket();

69 DatagramSocket receSocket = new DatagramSocket(10000);

70

71 new Thread(new UDPSend(sendSocket)).start();

72 new Thread(new UDPRece(receSocket)).start();

73 }

74 }

复制代码

(2)TCP传输

Socket和ServerSocket

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的应用程序。

\*\*\*\*Socket

\*\*构造方法：

Socket()

通过系统默认类型的 SocketImpl 创建未连接套接字

Socket(InetAddress address, int port)

创建一个流套接字并将其连接到指定 IP 地址的指定端口号。

Socket(String host, int port)

创建一个流套接字并将其连接到指定主机上的指定端口号。

\*\*方法摘要：

void close()

关闭此套接字。

InetAddress getInetAddress()

返回套接字连接的地址。

InputStream getInputStream()

返回此套接字的输入流。

OutputStream getOutputStream()

返回此套接字的输出流。

int getPort()

返回此套接字连接到的远程端口。

void shutdownInput()

此套接字的输入流置于“流的末尾”。

void shutdownOutput()

禁用此套接字的输出流。

String toString()

将此套接字转换为 String。

\*\*\*\*ServerSocket

\*\*构造方法：

ServerSocket()

创建非绑定服务器套接字。

ServerSocket(int port)

创建绑定到特定端口的服务器套接字。

方法摘要：

Socket accept()

侦听并接受到此套接字的连接。

void close()

关闭此套接字。

InetAddress getInetAddress()

返回此服务器套接字的本地地址。

\*\*\*\*TCP传输流程：

\*\*客户端：

建立Socket服务，并制定要连接的主机和端口；

获取Socket流中的输出流OutputStream，将数据写入流中，通过网络发送给服务端；

获取Socket流中的输出流InputStream，获取服务端的反馈信息；

关闭资源。

\*\*服务端：

建立ServerSocket服务，并监听一个端口；

通过ServerSocket服务的accept方法，获取Socket服务对象；

使用客户端对象的读取流获取客户端发送过来的数据；

通过客户端对象的写入流反馈信息给客户端；

关闭资源；

\*\*\*\*代码示例：

View Code