Java API第一天

* 什么是API

由Java提供(Oracle, SUN), 现成的程序组件（类）。

API封装了开发时候常用的功能！

字符串操作

数据管理（集合）

IO，文件访问

线程管理

...

API 的手册

http://doc.tedu.cn

* String API

String 对象的特点

String对象的数据不可改变！

String类型对象封装了一个字符串数组

任何的操作也不能改变这个字符数组的内容

String s = "123";

String ss = s;

s = s+"abc";

System.out.println(s);//123abc

System.out.println(ss);//123

说明: 在如上代码中改变的是字符串引用变量，但是字符串没有变！好处是字符串可以“作为”基本类型使用！

原理:

字符串常量的重用现象

Java中的字符串常量是尽可能重复使用的！好处是节省资源（内存）

字符串字面量（直接量）在内容一样时候重用同一个String对象。 String s1 = "123abc"; String s2 = "123abc"; //s1 s2 是字符串类型的引用变量 //"123abc" 是直接量（字面量）

字符串常量也参与重用！

字面量、常量的运算结果是字符串，也重用同一个字符串 String s4 = "123abc"; String s5 = 123 + "abc";//1+"23abc" System.out.println(s4==s5);//true

字符串变量，变量的运算结果 和 新创建的字符串对象不参与重用！！ String name = in.nextLine();//Tom String s1 = "Tom and Jerry"; String s2 = name + " and Jerry"; System.out.println(s1==s2);//false String s3 = new String("Tom and Jerry"); System.out.println(s3==s1);//false

原理:

经典题目：

String s1 = "1"+"23"+"abc";

String s2 = "1"+23+"abc";

String s3 = '1'+23+"abc";

System.out.print(s1==s2);

System.out.print(s1==s3);

如上代码的执行结果：

A.truetrue B.truefalse C.falsetrue D.falsefalse

字符串中的字符

字符串中封装了一个字符数组，字符串中的字符就是char类型的数据。

char 类型是整数， 是一个字符的Unicode编码。

16位无符号整数， 占用2个字节

案例：

String s = "Tom and Jerry";

// 0123456789012

char c = s.charAt(4);

System.out.println(c);//a

System.out.println((int)c);//97

1)indexOf 方法

找出一个字符在字符串中的位置：

indexOf() 1. 如果有重复，找出左侧第一个位置 2. 如果没有找到，返回-1

2)lastIndexOf

//反序查找：从右到左查找，返回字符的位置

String url = "http://tedu.cn/index.html";

int i = url.lastIndexOf("/");//14

System.out.println(i);//14

3)substring 方法

// 从字符串中截取一部分作为子字符串

url.substring(起始位置)

//从起始位置开始到最后截取为子字符串

String url = "http://tedu.cn/index.html";

String filename = url.substring(15);

// filename = index.html

// 包括起始不包括结束位置

String str = url.substring(7, 14);

String str = url.substring(7, 7+8);

4)trim

//去除字符串两端的空白，返还新的字符串

String str = " \t Tom \n \r";

String s = str.trim();

5)startsWith /endsWith

//检测一个字符串是否以指定字符串开头或结尾

String str = "Hello World!";

boolean b = str.startsWith("Hello");//true

b = str.startsWith("World");//false

b = str.endsWith("World");//false

b = str.endsWith("!");//true

案例:

String name = "demo.JPG";

if(name.toLowerCase().endsWith(".jpg")){

System.out.println("图片文件");}

6)toUpperCase/toLowerCase

// 转换大小写

Integer.parseInt();

//字符串转换为数字

7)StringBuilder

//Java 提供的用于计算字符串的API， 其运算性能好：

案例：

String s = "A";

s = s + "1";

s = s + "1";

s = s + "1";

System.out.println(s);

性能比较：快

StringBuilder API:

//追加

buf.append("李洪鹤老师...");

//插入

buf.insert(0, "那一年");

//替换

buf.replace(4, 4+2, "某人");

//删除

buf.delete(6, 6+2);

String s = buf.toString();

System.out.println(s);

Java API第二天

* String 和 StringBuilder

String是不变字符串： 对象不可改变，对象中的字符数组中的数据不可改变。

StringBuilder是可变字符串：对象封装的字符数组中的数据可以改变。

StringBuilder类型的操作性能好于Srting，字符串操作建议使用StringBuilder。 字符串显示使用String。

运行期间字符串连接计算利用StringBuilder的append完成。

案例：

String s = "123";

String ss = s + "abc";

//ss = new StringBuilder(s).append("abc").toString();

案例：

String s = "123"+"456"+"abc";

String ss = "123";

String str = ss + "456"+"abc";

//在一个表达式中出现连续的字符串连接，Java会自动的优化为一个StringBuilder对象

//String str = new StringBuilder(ss).append("456").append("abc").toString();

在工作中一个表达式中的连续字符串连接不需要优化为StringBuilder

在反复进行字符串连接时候建议使用StringBuilder

* 正则表达式

用于声明字符串的规则表达式。

经常用于检测一个字符串是否符合特定规则。

语法：

字符集合：

[abcd] abcd四个字符选一个

[a-f] a到f字符选一个

[^abcd] 除了abcd字符的其他字符

缩写版

. 任意字符

\d 数字

\D 非数字

\w 单词字符

\W 非单词字符

\s 空白

\S 非空白

数量词

X{n} n个X

X{n,m} n到m个

X{n, } 最少n个

\\? {0,1}

\\\* {0,}

\\+ {1,}

\\^开始

\\$结束

分组

(序列1|序列2)

**常见案例：**

**邮政编码的规则 \d{6}**

**身份证的规则 d{17}[\dXx]**

**用户名的规则: 8到11个单词字符 \w{8,11}**

**检查一个文件是否为jpeg照片文件： .+\.jpe?g**

**检查一个文件是否为照片文件： .+\.(jpe?g|png|gif)**

**Excel能够支持的文件名规则 ： .+\.xls[xbm]?**

**手机号的规则： (0086|\+86)?\s\*1\d{10}**

* String 正则API

（1）matches

用于检测一个字符串整体是否符合指定的正则规则。

案例：

**//String reg=".+\\.jpe?g";**

**String reg=".+\\.(jpe?g|png|gif)";**

**//正则表达式： .+\.(jpe?g|png|gif)**

**//Java 字符串: .+\\.(jpe?g|png|gif)**

**String file = "她.png";**

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

**boolean b = file.matches(reg);**

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

（2）split

按照一定规则将字符串劈开

**String str=**

**"1, 李洪鹤, 110, lihh@tedu.cn, 北京海淀区";**

**String[] data=str.split(",\\s\*");**

**// "1", "李洪鹤" ...**

**System.out.println(data[1]);**

**replaceAll**

（3）replace 替换

String str = "那一年，我去你家，你说我去！"

String reg = "我去";

String s = str.replaceAll(reg, "XXX");

System.out.println(s);

* Object 类

在Java类继承结构中，java.lang.Object类位于顶端；

如果定义一个Java类时没有使用extends关键字声明其父类，则其父类默认为 java.lang.Object 类；

Object类型的引用变量可以指向任何类型对象。

Java 设计者将子类中必须有的方法都定义在Object类中

toString

equals

toString方法

* 关于Object 的toString方法：

返回该对象的字符串表示。通常，toString 方法会返回一个“以文本方式表示”此对象的字符串。结果应是一个简明但易于读懂的信息表达式。

建议所有子类都重写此方法。

Object默认的toString方法返回值是:类名@散列码

这个默认方法是没有实际意义的结果，所以建议重写此方法。

由于很多Java API会自动调用toStirng方法，所以重写toString方法可以简化软件的开发调试。重写时候一般返回对象关键数据。

案例:重写toString方法：

public class Demo11 {

public static void main(String[] args) {

Foo foo = new Foo();

//调用Object提供的默认toString方法

//返回一个字符串，意义不大

String str = foo.toString();

System.out.println(str);

//调用重写的toString()方法

Goo goo = new Goo();

String s = goo.toString();

System.out.println(s);

//toString的用途：println 等

//很多API会自动的调用toString

//可以简化软件的开发和测试

System.out.println(goo);

}

}

class Foo /\*extends Object\*/{

}

class Goo{

//重新继承与Object类的toString();

public String toString(){

return "Hello World!";

}

}

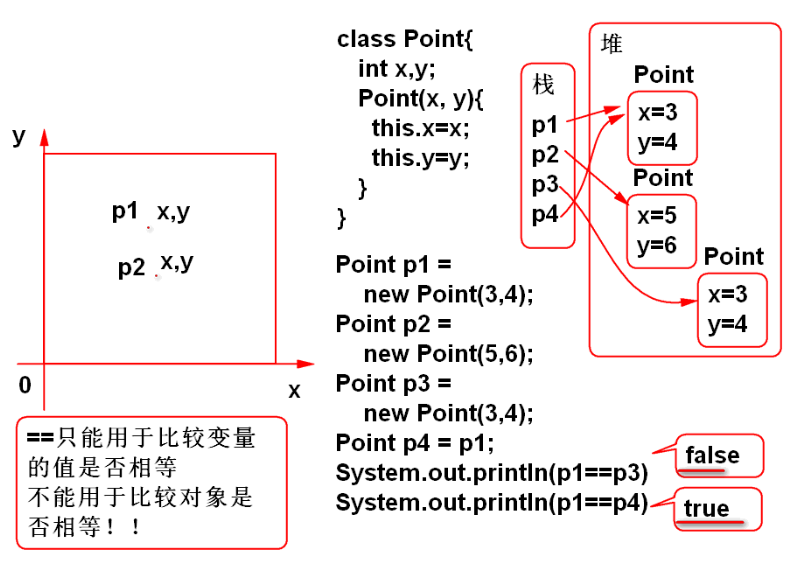
//重写toString方法后可以简化调试程序

//在程序中使用 System.out.println(goo);

//即可输出对象的数据值

* equals

== 不能比较两个对象是否相等 图示：↓



Java 在Object提供了equals方法，用于比较两个对象是否相等

equals默认的方法还是采用 == 比较，默认方法方法不能用于比较两个对象是否相等。

Java建议重写equals方法实现对象的相等比较

如何重写？按照对象的关键属性比较两个对象是否相等。

重写equase模板：

public boolean equals(Object obj){

//当两个对象的x和y都相等时候则相等

if(obj==null) return false;

if(this==obj) return true;

//使用if语句保护，避免造型异常

if(obj **instanceof** PointX){

//为了读取x y属性必须造型为子类型

PointX other=(PointX)obj;

return this.x == other.x &&

this.y == other.y;

}

//方法一定返回一个boolean值！

return false;}

/\*\*

\* 测试 重写 equals 方法

\*/

public class Demo13 {

public static void main(String[] args) {

Point p1 = new Point(3, 4);

Point p2 = new Point(5, 6);

Point p3 = new Point(3, 4);

Point p4 = p1;

//p1和p3引用的对象逻辑上是相等的！

//但是==比较不能反映这个“相等”结果

//== 只能用于比较两个变量相等

//不能用于比较两个对象是否相等

System.out.println(p1==p3);//false

System.out.println(p1==p4);//true

//默认的equals方法也不能对象相等

//必须重新才行！！！

class PointX{

int x, y;

public PointX(int x, int y) {

this.x = x;

this.y = y;

}

public String toString() {

return x+","+y;

}

//重写equals方法比较两个对象是否相等

public boolean equals(Object obj){

//当两个对象的x和y都相等时候则相等

if(obj==null) return false;

if(this==obj) return true;

//使用if语句保护，避免造型异常

if(obj instanceof PointX){

//为了读取x y属性必须造型为子类型

PointX other=(PointX)obj;

return this.x == other.x &&

this.y == other.y;

}

//方法一定返回一个boolean值！

return false;

}

}

Java API 第三天

* 包装类

1)将基本类型包装对象。

包装类包装的是基本类型数据

基本类型计算性能好

包装类型计算性能差

每个基本类型对应一个包装类型

**byte -> Byte**

**short -> Short**

**int -> Integer**

**long -> Long**

**float -> Float**

**double -> Double**

**boolean -> Boolean**

**char -> Charcater**

包装类用于将基本类型包装为对象

包装类中声明的每种类型的 “极值”

int max = Integer.MAX\_VALUE;

int min = Integer.MIN\_VALUE;

System.out.println(max);

System.out.println(min);

2)每种类型的极值都定义对应的包装类中。

包装类中提供了 字符串类转换为基本类型的方法

Integer.parseInt()

Double.parseDouble()

Boolean.parseBoolean()

3):自动拆包和自动包装

Java5 为了方便程序员，提供了自动包装和自动拆包

注意: 包装类型计算性能没有基本类型好，要尽量避免大量使用包装类进行计算。

案例：

Integer i = 5;//自动包装

int j = i;//自动拆包

* 计算机里的时间原理

计算机中的时间是一个毫秒数

从1970年元旦开始累积

1970年以前是负数

Date 类型

(1):默认的Date类型对象中封装当前系统时间毫秒

getTime 获取时间毫秒数

//long now = date.getTime();

修改date中的毫秒数

(2):Date 重写了toString 方法

修改date中的毫秒数

//0毫秒代表 GTM 时间的1970年元旦

date.setTime(-1000L\*60\*60\*24);

System.out.println(date);

(3):SimpleDateFormat 用于将Java系统时候 换算为人类习惯的 年月日 时分秒

(4):SimpleDateFormat fmt=

new SimpleDateFormat();

Date date = new Date();

//将系统时间换算为人类习惯的时间

String str = fmt.format(date);

System.out.println(str);

(5):自定义时间格式化显示

SimpleDateFormat fmt =

new SimpleDateFormat(

"yyyy年M月d日 HH:mm:ss");

Date date = new Date();

System.out.println(date);

System.out.println(fmt.format(date));

(6):将字符串解析为 系统时间（毫秒数）

String str = "1980-5-6";

String pattern = "yyyy-M-d";

SimpleDateFormat fmt=

new SimpleDateFormat(pattern);

//将字符串时间解析为计算机时间

Date date=fmt.parse(str);

System.out.println(date);

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

Calenda

1)历法是替代 Date 类型的API，用于替换Date类型的那些过时的API方法

Calendar c1 = new GregorianCalendar();

Calendar c2 = Calendar.getInstance();

//默认的Calendar里面封装是当前系统时间

System.out.println(c1);

System.out.println(c2);

//将Calender转换为Date类型

Date date = c1.getTime();

SimpleDateFormat fmt =

new SimpleDateFormat();

System.out.println(fmt.format(date));

//创建指定年月日的Calender

Calendar cal =

new GregorianCalendar(

2017, Calendar.MARCH, 6);

//输出结果：

date = cal.getTime();

System.out.println(fmt.format(date));

2)设置时间的分量

//默认的Calender是当前时间

Calendar cal=Calendar.getInstance();

System.out.println(cal.getTime());

//利用set方法修改时间分量：

//修改年份分量

cal.set(Calendar.YEAR, 2000);

//修改月份分量

cal.set(Calendar.MONTH, Calendar.AUGUST);

//修改日期分量

cal.set(Calendar.DATE, 1);

//检查修改的结果：

System.out.println(cal.getTime());

//超量更改时间：32天是一个月多一天，也就是顺延到下个月第一天

cal.set(Calendar.DATE, 32);

System.out.println(cal.getTime());

3）获取时间分量

Calendar cal = Calendar.getInstance();

//获取年份分量

int year=cal.get(Calendar.YEAR);

System.out.println(year);

//获取月份分量

int month = cal.get(Calendar.MONTH);

System.out.println(month);

//获取日期分量

int d = cal.get(Calendar.DATE);

System.out.println(d);

//获取星期

int w = cal.get(Calendar.DAY\_OF\_WEEK);;

System.out.println(w);

4）输出每个月的最大日期

Calendar cal=Calendar.getInstance();

for(int i=Calendar.JANUARY; i<=Calendar.DECEMBER; i++){

cal.set(Calendar.MONTH, i);

cal.set(Calendar.DATE,1);

int max=cal.getActualMaximum(

Calendar.DAY\_OF\_MONTH);

System.out.println(max);

}

5）日期相对增加方法 add

Calendar cal=Calendar.getInstance();

print(cal);

//将cal中的当前时间的月份分量上

//增加5个月的时间。

cal.add(Calendar.MONTH, -5);

print(cal);

cal.add(Calendar.DATE, 5);

print(cal);

JAVA API 第四天

* 集合框架

**1. Collection**

java提供了一种可以存储一组数据的数据结构，其提供了丰富的方法，在实际开发中往往比数组使用的广泛。

这种数据结构成为集合:Collection。

Collection是一个接口，其定义了集合的相关功能方法。使用时候必须使用具体实现类

Collection col=new ArrayList();

col.add("Tom");

col.add("Jerry");

System.out.println(col);

1.1. List和Set

Collection派生出了两个子接口，一个是List另一个则是Set。

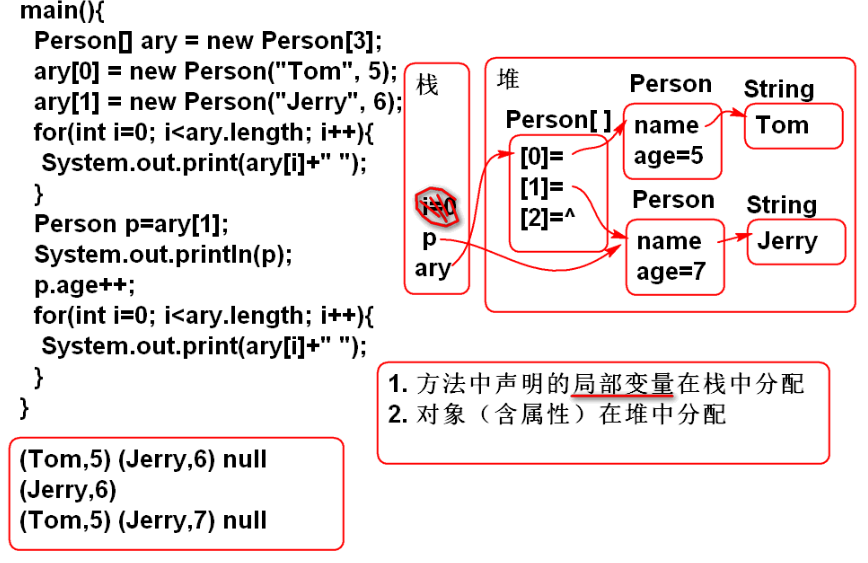
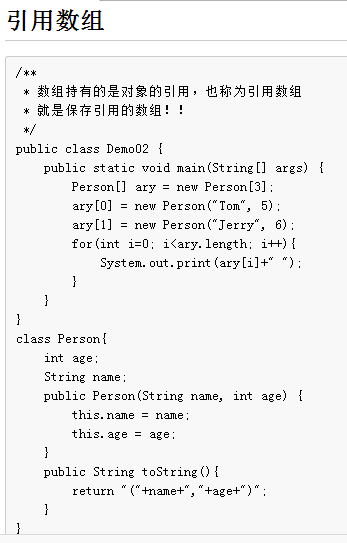
List:称为可重复集，顾名思义，该集合中是允许存放重复元素的，那么何为重复元素？重复元素指的并非是同一个元素，而是指equals方法比较为true的元素。

Set:称为不可重复集，所以，该集合中是不能将相同的元素存入集合两次，同List，这里相同指的也是两个元素的equals比较结果为true。

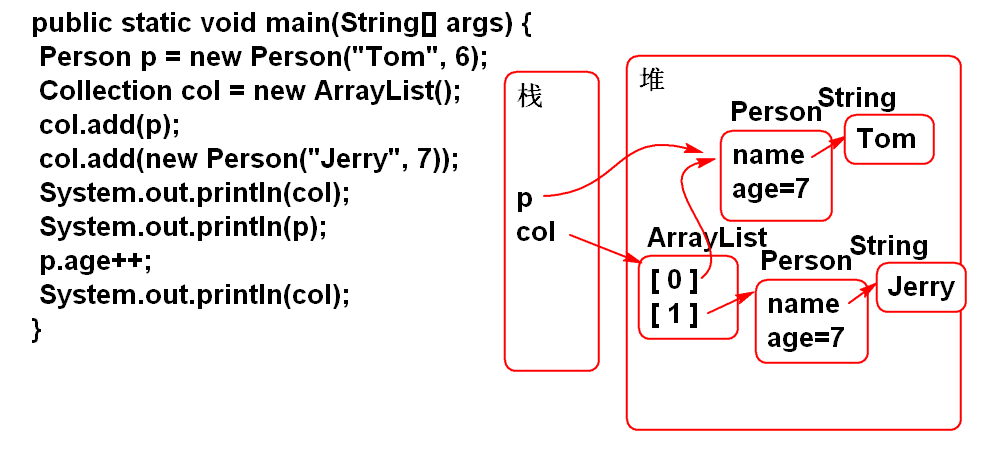
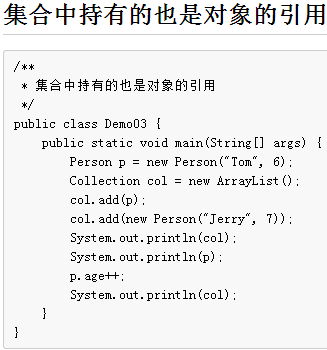
1.2. 集合持有对象的引用

集合中存储的都是引用类型的元素，那么引用类型变量实际上存储的是对象的“地址”，所以实际上集合只存储了元素对象在堆中的地址。而并不是将对象本身存入了集合中。

* 图示：引用类型



* 图示：集合类型



1.3. add()方法

Collection定义了一个add方法用于向集合中添加新元素。 该方法定义为:

boolean add(E e)

该方法会将给定的元素添加进集合，若添加成功则返回true,否则返回false

public static void main(String[] args) {

//HashSet实现了Collection接口

Collection col=new HashSet();

boolean b = col.add("Tom");

System.out.println(b);//true

b = col.add("Jerry");

System.out.println(b);

//重复添加 Tom 结果是false，添加失败

b = col.add("Tom");

System.out.println(b);//false

//Set类型的集合不能存在重复的元素

System.out.println(col);

1.4. contains方法

boolean contains(Object o)

该方法会用于判断给定的元素是否被包含在集合中。若包含则返回true,否则返回false。

这里需要注意的是，集合在判断元素是否被包含在集合中是使用元素的equals的比较结果。

(o==null ? e==null : o.equals(e)) 其中e是集合中的元素。

1.5. size,clear,isEmpty方法

size方法用于获取当前集合中的元素总数。该方法定义为:int size()

clear方法用于清空集合。该方法定义为:void clear()

isEmpty方法用于判断当前集合中是否不包含元素。该方法定义为:boolean isEmpty()

* addAll与containsAll方法

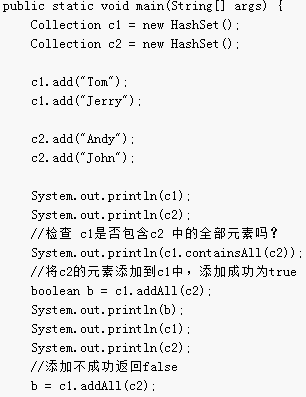
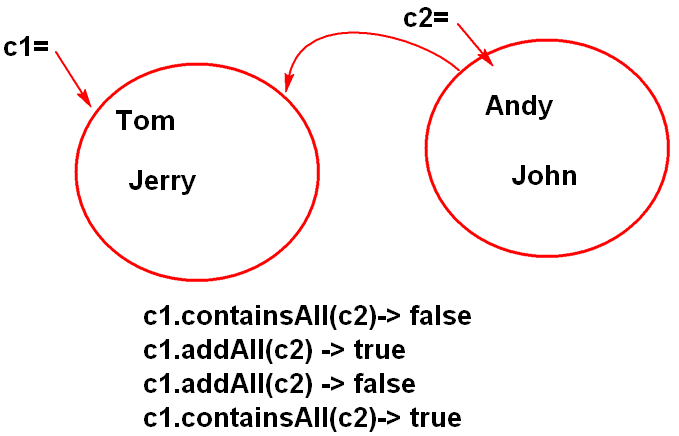
集合也提供了批处理操作:

addAll方法用于将给定集合中的所有元素添加到当前集合中，其方法定义为:

boolean addAll(Collection c)

containsAll方法用于判断当前集合是否包含给定集合中的所有元素，若包含则返回true。其方法定义为:

boolean containsAll(Collection c)

* Iterator 迭代器

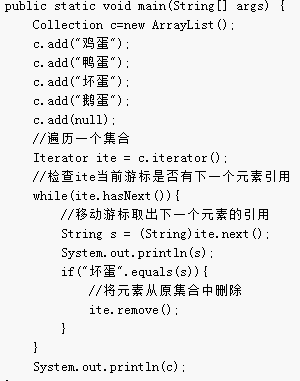
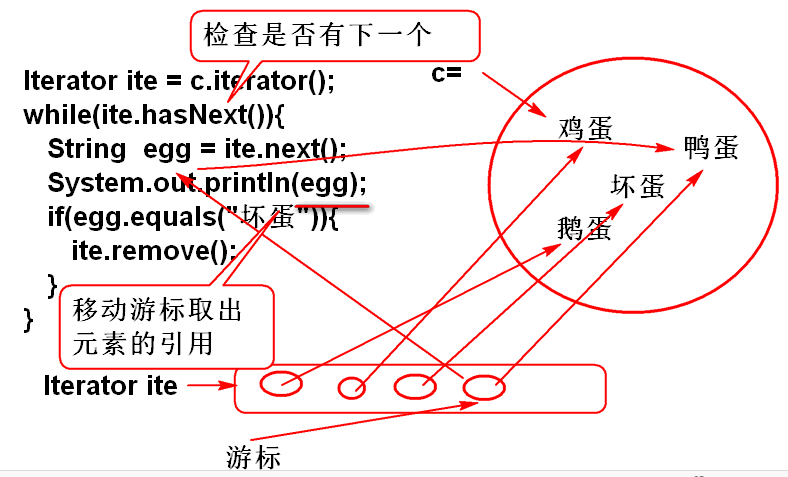
Collection提供了一个遍历集合的通用方式，迭代器(Iterator)。

获取迭代器的方式是使用Collection定义的方法:

Iterator iterator()

迭代器Iterator是一个接口，集合在覆盖Collection的iterator()方法时提供了迭代器的实现。

Iterator提供了统一的遍历集合元素的方式。

1、hasNext与next方法

迭代器用于遍历集合的两个主要方法:

boolean hasNext():判断集合是否还有元素可以遍历。

E next():返回迭代的下一个元素

遍历集合应遵循“先问后取”的方式，也就是说，应当在确定hasNext()方法的返回值为true的情况下再通过next()方法取元素。

由此可以看出，使用迭代器遍历集合是通过boolean值驱动的，所以它更适合使用while循环来遍历。

2.remove方法

迭代器还提供了一个方法:void remove()。该方法用于删除迭代器当次从集合中获取的元素。若我们在迭代过程中想删除集合元素时，我们就需要通过该方法来进行。这里需要注意，在使用迭代器遍历集合时是不能通过集合自身提供的remove方法删除元素的，否则迭代器在迭代时会抛出异常。

3. 增强for循环

Java5.0之后推出了一个新的特性，增强for循环,也称为新循环。该循环不通用于传统循环的工作，其只用于便利集合或数组。 语法:

for(元素类型 e : 集合或数组){

循环体

}

for(String str: col){

System.out.println(str);

新循环并非新的语法，而是在编译过程中，编译器会将新循环转换为迭代器模式。所以新循环本质上是迭代器。

* 泛型机制

1. 泛型在集合中的应用

泛型是Java SE 5.0引入的特性，泛型的本质是参数化类型。在类、接口和方法的定义过程中，所操作的数据类型被传入的参数指定。

Java泛型机制广泛的应用在集合框架中。所有的集合类型都带有泛型参数，这样在创建集合时可以指定放入集合中的对象类型。Java编译器可以据此进行类型检查，这样可以减少代码在运行时出现错误的可能性。

再声明ArrayList时，类名的右侧有一个<E>。"<>"表示泛型，而其中可以使用数字字母下划线(数字不能是第一个字符)来表示泛型的名字。(通常我们使用一个大写字母来表示，当然这个不是规定。)这时，在类中声明的方法的参数，返回值类型可以被定义为泛型。这样在创建对象时可以将类型作为参数传递，此时，类定义所有的E将被替换成传入的参数。

* 集合操作——线性表

1. List

List接口是Collection的子接口，用于定义线性表数据结构；可以将List理解为存放对象的数组，只不过其元素个数可以动态的增加或减少。并且List是可重复集，这个我们在以前的章节已经描述。

1.1. ArrayList和LinkedList

List接口的两个常见实现类为ArrayList和LinkedList，分别用动态数组和链表的方式实现了List接口。

可以认为ArrayList和LinkedList的方法在逻辑上完全一样，只是在性能上有一定的差别，ArrayList更适合于随机访问而LinkedList更适合于插入和删除；在性能要求不是特别苛刻的情形下可以忽略这个差别。

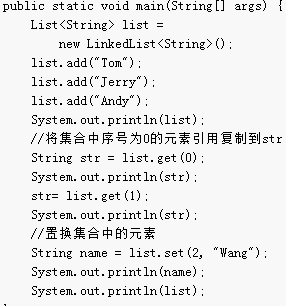
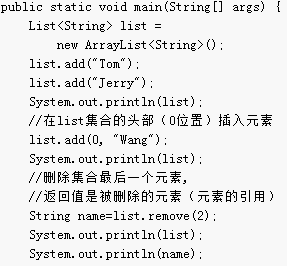
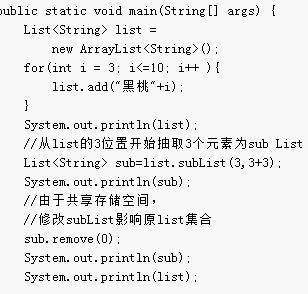
1.2. get与set方法

List除了继承Collection定义的方法外，还根据其线性表的数据结构定义了一系列方法，其中最常用的就是基于下标的get和set方法。

E get(int index):获取集合中指定下标对应的元素，下标从0开始。

E set(int index, E elment):将给定的元素存入给定位置，并将原位置的元素返回。

1.2get与set方法示例： / 1.3插入和删除示例： / 1.4subList示例：

1.3. 插入和删除

List根据下标的操作还支持插入与删除操作:

void add(int index,E element):

将给定的元素插入到指定位置，原位置及后续元素都顺序向后移动。

E remove(int index):

删除给定位置的元素，并将被删除的元素返回。

1.4. subList方法

List的subList方法用于获取子List

需要注意的是，subList获取的List与原List占有相同的存储空间，对子List的操作会影响的原List

List<E> subList(int fromIndex, int toIndex);

fromIndex和toIndex是截取子List的首尾下标（前包括，后不包括) 。

1.5. List转换为数组

List的toArray方法用于将集合转换为数组。但实际上该方法是在Collection中定义的，所以所有的集合都具备这个功能。

其有两个方法:

Object[] toArray()

<T>T[] toArray(T[] a)

其中第二个方法是比较常用的，我们可以传入一个指定类型的数组，该数组的元素类型应与集合的元素类型一致。返回值则是转换后的数组，该数组会保存集合中所有的元素。

1.6. 数组转换为List

Arrays类中提供了一个静态方法asList，使用该方法我们可以将一个数组转换为对应的List集合。

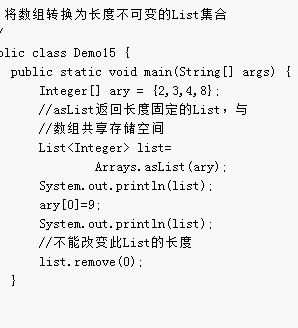
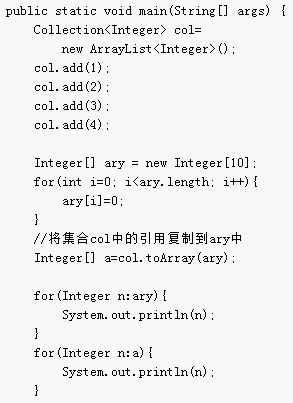
其方法定义为:

static <T>List<T> asList<T… a>

返回的List的集合元素类型由传入的数组的元素类型决定。

需要注意的是，返回的集合我们不能对其增删元素，否则会抛出异常。并且对集合的元素进行的修改会影响数组对应的元素。

**1.5. List转换为数组 1.6. 数组转换为List**



* List排序

1. Collections.sort方法实现排序

Collections是集合的工具类，它提供了很多便于我们操作集合的方法，其中就有用于集合排序的sort方法。该方法的定义为:

void sort(List<T> list)

其作用是对集合元素进行自然排序(按照元素的由小至大的顺序)

2. Comparable

通过上一节我们知道了如何对集合元素进行自然排序，但是要想对元素进行自然排序那么就必须要有一个必要条件，就是元素的大小。集合中存入的都是引用类型，是以对象的形式存在于内存中，那么对象是如何进行的大小比较呢？实际上，若想对某个集合的元素进行自然排序，该集合的元素有一个要求，就是这些元素必须是Comparable的子类。

Comparable是一个接口，用于定义其子类是可以比较的。因为该接口有一个抽象方法:

int compareTo(T t)

所有子类都需要重写该方法来定义对象间的比较规则。该方法要求返回一个整数，这个整数不关心具体的值，而是关注取值范围。

当返回值>0时，表示当前对象比参数给定的对象大。

当返回值<0时，表示当前对象比参数给定的对象小。

当返回值=0时，表示当前对象和参数给定的对象相等。

那么Collections的sort在进行排序时就会根据集合中元素的compareTo方法的返回值来判断大小从而进行自然排序。

3. comparator

一旦Java类实现了Comparable，其比较逻辑就已经确定；如果希望在排序的操作中临时指定比较规则，可以采用Comparator接口回调的方式。

该接口要求实现类必须重写其定义的方法:

int compare(T o1,T o2)

该方法的返回值要求，若o1>o2则返回值应>0,若o1<o2则返回值应<0，若o1==o2则返回值应为0

JAVA API 第五天

* 队列和栈

1. Queue

队列（Queue）是常用的数据结构，可以将队列看成特殊的线性表，队列限制了对线性表的访问方式：只能从线性表的一端添加（offer）元素，从另一端取出（poll）元素。

队列遵循先进先出（FIFO First Input First Output ）的原则。

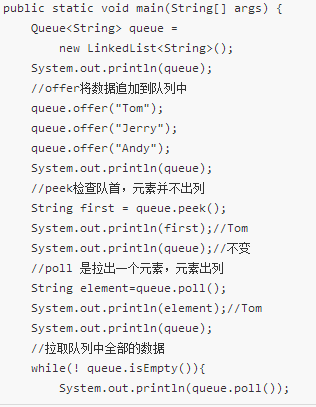
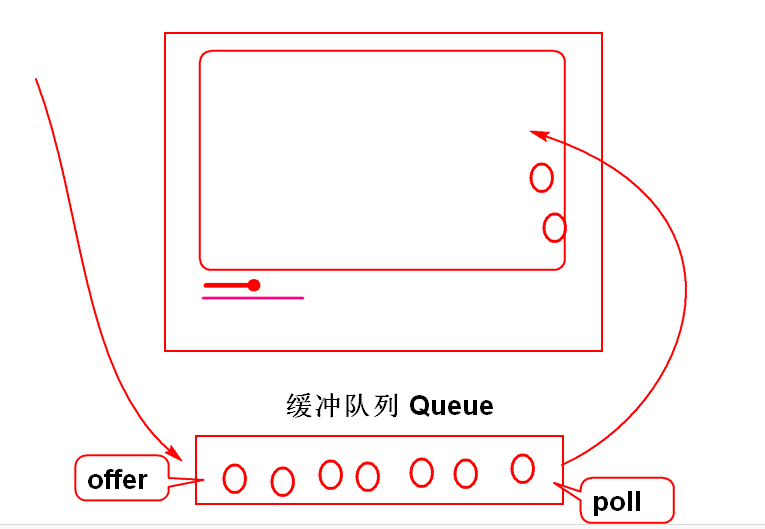
JDK中提供了Queue接口，同时使得LinkedList实现了该接口（选择LinkedList实现Queue的原因在于Queue经常要进行插入和删除的操作，而LinkedList在这方面效率较高）。

Queue提供了操作队列的相关方法,其主要方法如下:

boolean offer(E e):将元素追加到队列末尾,若添加成功则返回true。

E poll():从队首删除并返回该元素。

E peek():返回队首元素，但是不删除。



2.Deque

Deque是Queue的子接口，定义了所谓“双端队列”即从队列的两端分别可以入队（offer）和出队（poll），LinkedList实现了该接口。

如果将Deque限制为只能从一端入队和出队，则可实现“栈”（Stack）的数据结构，对于栈而言，入栈称之为push，出栈称之为pop。

栈遵循先进后出（FILO First Input Last Output ）的原则。

Deque提供了操作栈的相关方法,其主要方法如下:

void push(E e):将给定元素"压入"栈中。存入的元素会在栈首。即:栈的第一个元素

E pop():将栈首元素删除并返回。

* 查询表

1.1. Map 接口

java提供了一组可以以键值对(key-value)的形式存储数据的数据结构，这种数据结构称为Map。我们可以把Map看成一个多行两列的表格，其中第一列存放key,第二列存放value。

而每一行就相当于一组key-value对，表示一组数据。

Map对存入的元素有一个要求，就是key不能重复，所谓不能重复指的是在Map中不能包含两个equals为true的key。

Map对于key,value的类型没有严格要求，只要是引用类型均可。但是为了保证在使用时不会造成数据混乱，通常我们会使用泛型去约束key与value的类型。

1.2. put方法

Map在保存数据时实际上是存入了两部分信息的 ，key与value。如何向Map中存入数据。Map提供了一个方法:

V put(K k,V v)

该方法的作用是将key-value对存入Map中，因为Map中不允许出现重复的key,所以若当次存入的key已经在Map中存在，则是替换value操作，而返回值则为被替换的元素。若此key不存在，那么返回值为null。

1.3. get方法

我们学会了如何向Map中存入数据，那么我们再来看看如何获取数据。Map中获取数据的方式是给定Key获取对应的Value。

Map提供了一个方法:

V get(Object key)

该方法的作用就是根据给定的key去查找Map中对应的value并返回，若当前Map中不包含给定的key,那么返回值为null。

1.4. containsKey方法

Map中的containsKey方法用于检测当前Map中是否包含给定的key。其方法定义如下:

boolean containsKey(Object key)

若当前Map中包含给定的key(这里检查是否包含是根据key的equals比较结果为依据的。)则返回true。

* HashMap

1. hash表原理

HashMap是Map的一个常用的子类实现。其实使用散列算法实现的。

2. hashcode方法

HashMap的存取是依赖于key的hashcode方法的返回值的，而hashcode方法实际上是在Object中定义的。其定义如下:

int hashCode()

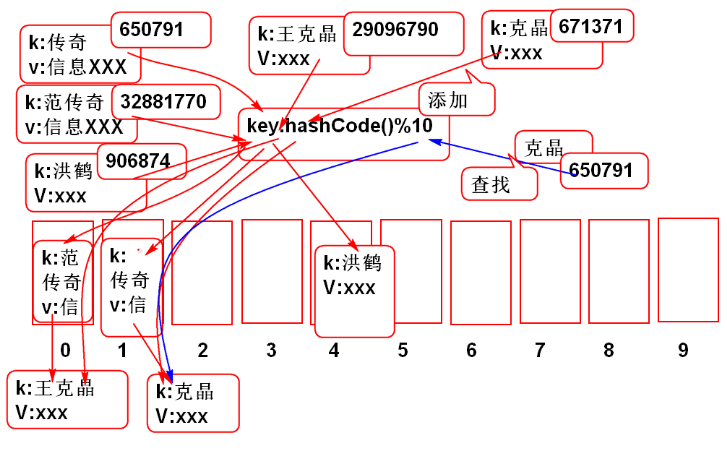
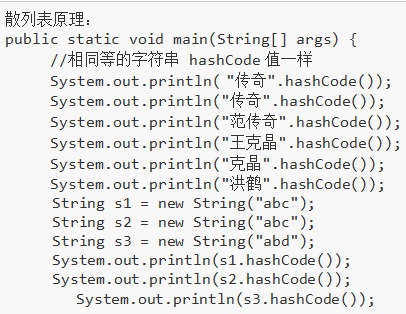
重写一个类的hashcode()方法有以下注意事项:

1、若一个类重写了equals方法，那么就应当重写hashcode()方法。

2、若两个对象的equals方法比较为true,那么它们应当具有相同的hashcode值。

3、对于同一个对象而言，在内容没有发生改变的情况下，多次调用hashCode()方法应当总是返回相同的值。

4、对于两个对象equals比较为false的，并不要求其hashcode值一定不同，但是应尽量保证不同，这样可以提高散列表性能。



3. 装载因子及HashMap优化

在散列表中有一下名词需要了解:

Capacity：容量, hash表里bucket(桶)的数量, 也就是散列数组大小.

Initial capacity：初始容量, 创建hash表的时 初始bucket的数量, 默认构建容量是16. 也可以使用特定容量.

Size ： 大小, 当前散列表中存储数据的数量.

Load factor：加载因子, 默认值0.75(就是75%), 向散列表增加数据时如果 size/capacity 的值大于Load factor则发生扩容并且重新散列(rehash).

那么当加载因子较小时候散列查找性能会提高, 同时也浪费了散列桶空间容量. 0.75是性能和空间相对平衡结果. 在创建散列表时候指定合理容量, 从而可以减少rehash提高性能。

* 有序Map

1. LinkedHashMap实现有序的Map

Map 接口的哈希表和链表实现，具有可预知的迭代顺序。此实现与 HashMap 的不同之处在于，LinkedHashMap维护着一个双向循环链表。此链表定义了迭代顺序，该迭代顺序通常就是将存放元素的顺序。

需要注意的是，如果在Map中重新存入以有的key，那么key的位置会不会发生改变，只是将value值替换。

**PM2.5**

收集PM2.5信息，方便查找某个区域的PM2.5最大值 PM2.5数据：

"海淀:180,顾家庄:100,香山:78,驻马店:120,石家庄:125,海淀:156,香山:298,海淀:356"

参考代码：

public class Demo11 {

public static void main(String[] args) {

//"海淀:180,顾家庄:100,香山:78,驻马店:120,石家庄:125,海淀:156,香山:298,海淀:356"

//1. 解析字符串得到每个地点和其PM2.5值

//2. 创建散列表 map

//3. 遍历输入数据

//4. 如果有地点，取出比较在存入

//5. 如果没有地点，直接存入

String pm25="海淀:180,顾家庄:100,香山:78,驻马店:120,石家庄:125,"

+ "海淀:156,香山:298,海淀:356";

String[] data = pm25.split("[,:]");

System.out.println(Arrays.toString(data));

Map<String, Integer> map=

new LinkedHashMap<String, Integer>();

for(int i=0; i<data.length; i+=2){

String loc=data[i];

String str=data[i+1];

//System.out.println(loc);

int val = Integer.parseInt(str);

if(map.containsKey(loc)){

int v = map.get(loc);

if(val>v){

map.put(loc, val);

}

}else{

map.put(loc, val);

}

}

//查询

int val = map.get("海淀");

System.out.println(val);

//输出全部结果？

}

}

制作人：王鑫

批阅：郭宏达

QQ：65991040

微信号：18599995521

欢迎大家联系我共同交流学习，帮助我指出错误，提出建议！