第10天 常用API

今日内容介绍

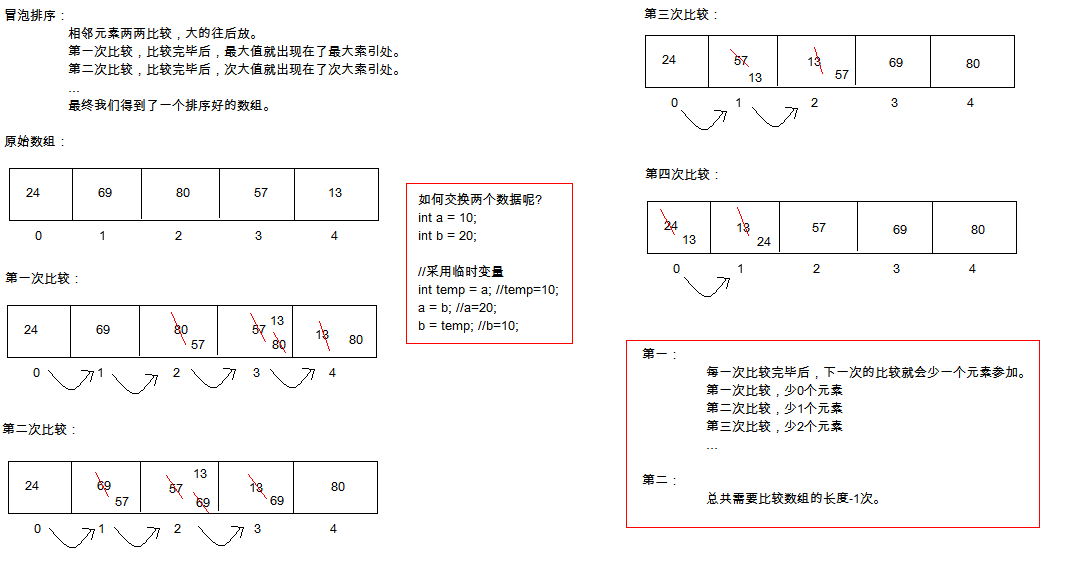
* 数组排序和Arrays工具类
* 包装类
* Date类和SimpleDateFormat类

# 数组冒泡排序和Arrays工具类

## 数组的冒泡排序

冒泡排序:就是相邻的两个元素进行两两比较,把元素值大的元素依次向后排.

### 数组排序之冒泡排序原理图解



### 数组排序之冒泡排序代码实现

#### 案例代码一:

**package** com.itheima;

/\*

\* 数组排序：(冒泡排序)

\*/

**public** **class** ArrayDemo {

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

//定义一个int类型的数组

**int**[] arr = {24,69,80,57,13};

/\*

//第一次比较

//arr.length-1是为了防止索引越界

//arr.length-1-0是为了减少比较的次数

for(int x=0; x<arr.length-1-0; x++) {

//ArrayIndexOutOfBoundsException

if(arr[x] > arr[x+1]) {

//交换数据

int temp = arr[x];

arr[x] = arr[x+1];

arr[x+1] = temp;

}

}

//调用遍历方法

System.out.println("第一次比较完毕：");

printArray(arr);

//第二次比较

//arr.length-1是为了防止索引越界

//arr.length-1-1是为了减少比较的次数

for(int x=0; x<arr.length-1-1; x++) {

//ArrayIndexOutOfBoundsException

if(arr[x] > arr[x+1]) {

//交换数据

int temp = arr[x];

arr[x] = arr[x+1];

arr[x+1] = temp;

}

}

//调用遍历方法

System.out.println("第二次比较完毕：");

printArray(arr);

//第三次比较

//arr.length-1是为了防止索引越界

//arr.length-1-2是为了减少比较的次数

for(int x=0; x<arr.length-1-2; x++) {

//ArrayIndexOutOfBoundsException

if(arr[x] > arr[x+1]) {

//交换数据

int temp = arr[x];

arr[x] = arr[x+1];

arr[x+1] = temp;

}

}

//调用遍历方法

System.out.println("第三次比较完毕：");

printArray(arr);

//第四次比较

//arr.length-1是为了防止索引越界

//arr.length-1-3是为了减少比较的次数

for(int x=0; x<arr.length-1-3; x++) {

//ArrayIndexOutOfBoundsException

if(arr[x] > arr[x+1]) {

//交换数据

int temp = arr[x];

arr[x] = arr[x+1];

arr[x+1] = temp;

}

}

//调用遍历方法

System.out.println("第四次比较完毕：");

printArray(arr);

\*/

//用循环改进

/\*

for(int y=0; y<4; y++) {

for(int x=0; x<arr.length-1-y; x++) {

if(arr[x] > arr[x+1]) {

//交换数据

int temp = arr[x];

arr[x] = arr[x+1];

arr[x+1] = temp;

}

}

}

\*/

/\*

//循环做的次数不能写固定的值，用arr.length-1改进即可

for(int y=0; y<arr.length-1; y++) {

for(int x=0; x<arr.length-1-y; x++) {

if(arr[x] > arr[x+1]) {

//交换数据

int temp = arr[x];

arr[x] = arr[x+1];

arr[x+1] = temp;

}

}

}

printArray(arr);

\*/

//如果我有多个数组要排序，每个数组写这样的一段代码，太麻烦，怎么办呢

//用方法改进

*sort*(arr);

System.***out***.println("排序后：");

*printArray*(arr);

}

/\*

\* 两个明确：

\* 返回值类型：void

\* 参数列表：int[] arr

\*/

**public** **static** **void** sort(**int**[] arr) {

**for**(**int** x=0; x<arr.length-1; x++) {

**for**(**int** y=0; y<arr.length-1-x; y++) {

**if**(arr[y] > arr[y+1]) {

**int** temp = arr[y];

arr[y] = arr[y+1];

arr[y+1] = temp;

}

}

}

}

/\*

\* 数组遍历

\*/

**public** **static** **void** printArray(**int**[] arr) {

System.***out***.print("[");

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

**if**(x==arr.length-1) {

System.***out***.print(arr[x]);

}**else** {

System.***out***.print(arr[x]+", ");

}

}

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

}

}

## Arrays工具类

### Arrays类的概述和使用

Arrays:提供了对数组操作的各种方法。

public static String toString(int[] a):把数组转成字符串

public static void sort(int[] a):对数组进行升序排序

#### 案例代码二

**package** com.itheima;

**import** java.util.Arrays;

/\*

\* Arrays:提供了对数组操作的各种方法。

\* public static String toString(int[] a):把数组转成字符串

\* public static void sort(int[] a):对数组进行升序排序

\*/

**public** **class** ArraysDemo {

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

//定义一个数组

**int**[] arr = {24,69,80,57,13};

//public static String toString(int[] a):把数组转成字符串

System.***out***.println("排序前："+Arrays.*toString*(arr));

//public static void sort(int[] a):对数组进行升序排序

Arrays.*sort*(arr);

System.***out***.println("排序后："+Arrays.*toString*(arr));

}

}

### Arrays类中构造方法的问题

Arrays类中真的没有构造方法吗?

一个类中没有构造方法，系统将提供一个无参构造方法。

而我们在帮助文档中没有看到Arrays类的构造方法，这是为什么呢?

Arrays类中有构造方法，只不过构造方法被private修饰，外界是无法使用的。因为外界无法使用，所以帮助文档中就看不到。

通过查看源码，我们找到了如下的内容：

private Arrays() {}

Arrays类的这种设计是常用的工具类的设计思想：

构造方法私有。

成员都用static修饰。

Math，Collections等

# 包装类

## 基本类型包装类的概述

需求：我要判断一个数据是否在int范围内?

要想判断一个数据是否在int范围内，首先我们得知道int范围，在前面我们讲解基本数据类型的时候说过了：

-2147483648 到 2147483647

为了对基本数据类型进行更多更方便的操作，Java就针对每一种基本数据类型提供了一个对应的引用类型。

基本类型包装类：

Byte byte

Short short

Integer int

Long long

Float float

Double double

Character char

Boolean boolean

基本数据类型包装类最常见的用法就是用于和字符串之间进行相互转换。

## Integer类的概述和构造方法

Integer:Integer类在对象中包装了一个基本类型 int 的值。

构造方法：

Integer(int value)

Integer(String s)

注意：这个字符串必须由数字字符组成

### 案例代码三

**package** com.itheima\_02;

/\*

\* Integer:Integer类在对象中包装了一个基本类型 int 的值。

\*

\* 构造方法：

\* Integer(int value)

\* Integer(String s)

\* 注意：这个字符串必须由数字字符组成

\*/

**public** **class** IntegerDemo {

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

//Integer(int value)

**int** value = 100;

Integer i = **new** Integer(value);

System.***out***.println(i); //100

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

//Integer(String s)

String s = "100";

//NumberFormatException:数据格式化异常

//String s = "abc";

Integer ii = **new** Integer(s);

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

}

}

## int类型和String类型的相互转换

int类型和String类型的相互转换

int -- String

String类中：public static String valueOf(int i)

String -- int

Integer类中：public static int parseInt(String s)

### 案例代码四

**package** com.itheima\_03;

/\*

\* int类型和String类型的相互转换

\*

\* int -- String

\* String类中：public static String valueOf(int i)

\*

\* String -- int

\* Integer类中：public static int parseInt(String s)

\*/

**public** **class** IntegerDemo {

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

//int -- String

**int** number = 100;

//方式1

String s1 = "" + number;

System.***out***.println(s1);

//方式2

//public static String valueOf(int i)

String s2 = String.*valueOf*(number);

System.***out***.println(s2);

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

//String -- int

String s = "100";

//方式1

//String -- Integer -- int

Integer i = **new** Integer(s);

//public int intValue()

**int** x = i.intValue();

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

//方式2

//public static int parseInt(String s)

**int** y = Integer.*parseInt*(s);

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

}

}

## Integer的练习之把字符串中的数据排序

需求:

我有如下一个字符串:”91 27 46 38 50”

请写代码实现最终输出结果是：”27 38 46 50 91”

提示：这里需要参考String类中的方法

public String[] split(String regex)

### 案例代码五

**package** com.itheima\_04;

**import** java.util.Arrays;

/\*

\* 我有如下一个字符串:”91 27 46 38 50”

\* 请写代码实现最终输出结果是：”27 38 46 50 91”

\* 提示：这里需要参考String类中的方法

\* public String[] split(String regex)

\*

\* 分析：

\* A:定义一个字符串对象

\* B:把字符串中的数字数据存储到一个int类型的数组中

\* C:对int数组进行排序

\* D:把排序后的数组中的元素进行拼接得到一个字符串

\* E:输出字符串

\*/

**public** **class** IntegerDemo {

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

//定义一个字符串对象

String s = "91 27 46 38 50";

//把字符串中的数字数据存储到一个int类型的数组中

//public String[] split(String regex)

String[] strArray = s.split(" ");

/\*

for(int x=0; x<strArray.length; x++) {

System.out.println(strArray[x]);

}

\*/

//定义一个int类型的数组

**int**[] arr = **new** **int**[strArray.length];

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

arr[x] = Integer.*parseInt*(strArray[x]);

}

//对int数组进行排序

Arrays.*sort*(arr);

//把排序后的数组中的元素进行拼接得到一个字符串

StringBuilder sb = **new** StringBuilder();

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

**if**(x==arr.length-1) {

sb.append(arr[x]);

}**else** {

sb.append(arr[x]).append(" ");

}

}

String result = sb.toString();

//输出字符串

System.***out***.println("result:"+result);

}

}

## JDK5的新特性自动装箱和拆箱

JDK5新特性：

自动装箱：把基本数据类型转换为对应的包装类类型

public static Integer valueOf(int i)

自动拆箱：把包装类类型转换为对应的基本数据类型

public int intValue()

Java程序的运行：

编写java文件 -- 编译生成class文件 -- 执行

注意：在使用包装类类型的新特性的时候，如果做操作，最好先判断是否为null。

开发中的原则：

只要是对象，在使用前就必须进行不为null的判断。

### 案例代码六:

**package** com.itheima\_05;

/\*

\* JDK5新特性：

\* 自动装箱：把基本数据类型转换为对应的包装类类型

\* public static Integer valueOf(int i)

\* 自动拆箱：把包装类类型转换为对应的基本数据类型

\* public int intValue()

\*

\* Java程序的运行：

\* 编写java文件 -- 编译生成class文件 -- 执行

\*

\* 注意：在使用包装类类型的新特性的时候，如果做操作，最好先判断是否为null。

\*

\* 开发中的原则：

\* 只要是对象，在使用前就必须进行不为null的判断。

\*/

**public** **class** IntegerDemo {

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

//创建一个包装类类型的对象

//Integer i = new Integer(100);

Integer ii = 100; //自动装箱

//public static Integer valueOf(int i)

ii += 200; //ii = ii + 200; 自动拆箱,自动装箱

//public int intValue()

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

/\*

Integer ii = Integer.valueOf(100);

ii = Integer.valueOf(ii.intValue() + 200);

System.out.println(ii);

\*/

Integer iii = **null**;

**if**(iii != **null**) {

iii += 300; //NullPointerException

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

}

}

}

# Date类和SimpleDateFormat类

## Date类的概述和构造方法

Date:Date表示特定的瞬间，精确到毫秒。

构造方法：

Date():根据当前时间创建的日期对象

Date(long date):根据给定的毫秒值创建对象，从1970 年 1 月 1 日 00:00:00

### 案例代码七:

**package** com.itheima;

**import** java.util.Date;

/\*

\* Date:Date表示特定的瞬间，精确到毫秒。

\*

\* 构造方法：

\* Date():根据当前时间创建的日期对象

\* Date(long date):根据给定的毫秒值创建对象，从1970 年 1 月 1 日 00:00:00

\*/

**public** **class** DateDemo {

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

//Date()

Date d = **new** Date();

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

//Date(long date)

**long** date = 1000 \* 60 \* 60;

Date dd = **new** Date(date);

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

}

}

## Date类的成员方法getTime()和setTime()

**成员方法:**

public long getTime():获取的是毫秒值。从1970年1月1日 00:00:00开始的。

public void setTime(long time):设置时间，给的是毫秒值。

从Date得到一个毫秒值：

getTime()

从一个毫秒值得到一个Date对象

构造方法

setTime(long time)

### 案例代码八:

**package** com.itheima;

**import** java.util.Date;

/\*

\* public long getTime():获取的是毫秒值。从1970年1月1日 00:00:00开始的。

\* public void setTime(long time):设置时间，给的是毫秒值。

\*

\* 从Date得到一个毫秒值：

\* getTime()

\*

\* 从一个毫秒值得到一个Date对象

\* 构造方法

\* setTime(long time)

\*/

**public** **class** DateDemo2 {

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

//创建对象

Date d = **new** Date();

//public long getTime()

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

//public void setTime(long time)

d.setTime(1000\*60\*60);

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

}

}

## SimpleDateFormat类的概述和使用

SimpleDateFormat:是一个以与语言环境有关的方式来格式化和解析日期的具体类。

它允许进行格式化（日期 -> 文本）、解析（文本 -> 日期）

格式化（日期 -> 文本）: Date -- String

public final String format(Date date)

解析（文本 -> 日期）: String -- Date

public Date parse(String source)

package com.itheima\_01;

### 案例代码九:

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

/\*

\* SimpleDateFormat:是一个以与语言环境有关的方式来格式化和解析日期的具体类。

\* 它允许进行格式化（日期 -> 文本）、解析（文本 -> 日期）

\*

\* 格式化（日期 -> 文本）: Date -- String

\* public final String format(Date date)

\*

\* 解析（文本 -> 日期）: String -- Date

\* public Date parse(String source)

\*/

public class SimpleDateFormatDemo {

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

/\*

//Date -- String

Date d = new Date();

//SimpleDateFormat(): 用默认的模式

//SimpleDateFormat sdf = new SimpleDateFormat();

//SimpleDateFormat(String pattern):用给定的模式

SimpleDateFormat sdf = new SimpleDateFormat("yyyy年MM月dd日 HH:mm:ss");

String s = sdf.format(d);

//现在虽然实现了把日期格式化成一个字符串，但是不是我们想要的格式

//我们想要的格式是：xxxx年xx月xx日 xx:xx:xx

System.out.println(s);

\*/

//String -- Date

String str = "2080-08-08 12:23:45";

//把一个字符串解析为日期的时候，请注意模式字符串和给定的日期字符串的格式要匹配

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

Date d = sdf.parse(str);

System.out.println(d);

}

}

## Date的练习之日期工具类的定义和使用

### 制作工具类DateUtil

#### 案例代码十:

package com.itheima;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

/\*

\* 工具类

\*

\* 构造方法私有

\* 成员方法静态

\*/

public class DateUtil {

private DateUtil() {}

/\*

\* 把日期转换为指定格式的字符串

\* 两个明确：

\* 返回值类型：String

\* 参数列表：Date date, String format

\*/

public static String dateToString(Date date, String format) {

SimpleDateFormat sdf = new SimpleDateFormat(format);

String s = sdf.format(date);

return s;

}

/\*

\* 把指定格式的字符串解析为日期

\* 两个明确：

\* 返回值类型：Date

\* 参数列表：String s, String format

\*/

public static Date stringToDate(String s,String format) throws ParseException {

SimpleDateFormat sdf = new SimpleDateFormat(format);

Date d= sdf.parse(s);

return d;

}

}

### 测试工具类

#### 案例代码十一:

package com.itheima;

import java.text.ParseException;

import java.util.Date;

public class DateUtilTest {

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

Date d = new Date();

String s = DateUtil.dateToString(d, "yyyy年MM月dd日 HH:mm:ss");

System.out.println(s);

String s2 = DateUtil.dateToString(d, "yyyy年MM月dd日");

System.out.println(s2);

String s3 = DateUtil.dateToString(d, "HH:mm:ss");

System.out.println(s3);

String str = "2080-08-08 12:23:34";

Date dd = DateUtil.stringToDate(str, "yyyy-MM-dd HH:mm:ss");

System.out.println(dd);

}

}