java.lang.Object

.clone 类要实现一个特殊接口 Cloneable

.equals 比较两个对象是否相等

默认实现是判断两个对象的地址是否相等

对象1 == 对象2

当更常用的是重写equals方法来判断对象的内容是否相等

Studnet s1 = new Student("张三")

Student s2 = new Student("张三")

s1.equals(s2);

.toString 用字符串表示对象

.getClass 获取对象的类型

.finallize 在对象被垃圾回收时调用

包装类

.valueOf(字符串） 转换为基本数字类型

可以直接将包装类赋值给基本类型，也可以反过来赋值，中间的转换方法不需要显式地调用，称为自动拆箱和装箱

# 1. 时间的处理

## 毫秒值

当前时间 - （1970-1-1 00:00:00 GMT (格林尼治时间)） = 毫秒值

System.**[currentTimeMillis](mk:@MSITStore:C:\\Users\\adminstator\\Desktop\\JAVA%20API1.6中文文档.CHM::/java/lang/../../java/lang/System.html" \l "currentTimeMillis())**()

示例代码：

System.out.println(System.currentTimeMillis());

long start = System.currentTimeMillis();

for(int i = 0; i<1000000000; i++) {

}

long end = System.currentTimeMillis();

System.out.println(end - start);

## java.util.Date (年月日，时分秒，毫秒)

使用方法

Date d = new Date(); // 也是代表当前时间

// 2. 日期对象 alt+/

Date d = new Date(); // 当前时间

// 1) 年 必须+1900 才是真正年份

System.out.println(d.getYear() + 1900);

// 2) 月 月份是从0开始

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

// 3) 天 ， 时分秒

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

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

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

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

// 是一个数字，表示星期 0-周日 1-周一 ....

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

// set 设置时间

Date d1 = new Date();

d1.setYear(116);

d1.setMonth(2);

d1.setDate(21);

// toString 格式

// 星期 月 日 时:分:秒 时区 年

// Mon Mar 21 09:42:26 CST 2016

System.out.println(d1);

Thread.sleep(1000); // 让程序暂停1秒

Date d2 = new Date();

d2.setYear(116);

d2.setMonth(2);

d2.setDate(21);

System.out.println(d2);

// 比较日期大小 (新的时间更大)

/\* 方法返回一个int 如果是 0 表示两个时间相等，

如果返回正数，表示大于对方

如果返回负数, 表示小于对方

\*/

System.out.println(d1.compareTo(d2));

// getTime 是返回日期对象对应的毫秒值

System.out.println(d2.getTime()-d1.getTime());

## java.util.Calendar 日历类

Calendar c = Calendar.getInstance();

// getInstance方法内部会创建Calendar 的子类对象返回

// 1. 创建 Calendar 对象

Calendar c = Calendar.getInstance(); // 代表当前时间

System.out.println("对象的类型是：" + c.getClass());

// 1) 加14小时 (当前时间10点，变为第二天的0点)

// 参数1：要操作时间的哪一部分， 参数2：修改的值

// HOUR\_OF\_DAY 0-23 小时

c.add(Calendar.HOUR\_OF\_DAY, 14);

// 2) 获取时间

Date d = c.getTime();

System.out.println(d);

// 2) 要减 ， 值给负数即可

c.add(Calendar.HOUR\_OF\_DAY, -1);

Date d2 = c.getTime();

System.out.println(d2);

// 3） 天份+11天

c.add(Calendar.DATE, 11);

Date d3 = c.getTime();

System.out.println(d3);

// 4) 获取日期的某一组成部分

System.out.println("获取年:" + c.get(Calendar.YEAR));

System.out.println("获取月:" + c.get(Calendar.MONTH));// 从0开始

System.out.println("获取日:" + c.get(Calendar.DATE));

System.out.println("获取小时(24):" + c.get(Calendar.HOUR\_OF\_DAY));

System.out.println("获取小时(12):" + c.get(Calendar.HOUR));

System.out.println("获取上下午:" + c.get(Calendar.AM\_PM)); // 0 表示上午，1表示下午

System.out.println("获取分钟:" + c.get(Calendar.MINUTE));

System.out.println("获取秒:" + c.get(Calendar.SECOND));

// 5) 修改分钟部分为20分

c.set(Calendar.MINUTE, 20);

System.out.println("获取分钟:" + c.get(Calendar.MINUTE));

# 数值的精确运算

double 以及 float 无法精确运算， 例如：2.0 - 1.1

进行精确计算，需要使用java.math.BigDecimal类

// 构造方法的参数类型必须是字符串

BigDecimal d1 = new BigDecimal("2.0");

BigDecimal d2 = new BigDecimal("1.1");

/\*

.add 加法

.subtract 减法

.multiply 乘法

.divide 除法

\*/

BigDecimal d3 = d1.subtract(d2);

System.out.println(d3);

BigDecimal d4 = new BigDecimal("20.0");

BigDecimal d5 = new BigDecimal("3.0");

// 参数2：小数点后保留位数, 参数3：舍入模式（常见的是四舍五入）

// 3.3333333333333335

BigDecimal d6 = d4.divide(d5, 4, RoundingMode.HALF\_UP);

System.out.println(d6);

## java.lang.String

1） 创建字符串的两种方式：

String str = "字符串值"; // 推荐

String str = new String("字符串值"); // 不推荐

1. 常用方法

// 1) charAt(下标从0开始) 返回字符串中某一字符

String s1 = "abcdefg";

System.out.println(s1.charAt(3));

// 2) 对象1.compareTo(对象2) 0表示相等， 正数:表示前一个大于后一个, 负数：与正数相反

// 对于某一个字符，是按照unicode 码表来决定他们的大小顺序

String s2 = "A";

String s3 = "a";

System.out.println(s2.compareTo(s3));

// 如果有多个字符, 首先比较首字母，首字母相等再比较第二个字母，依次类推...

String s4 = "aaac";

String s5 = "aaab";

System.out.println(s4.compareTo(s5));

System.out.println("110".compareTo("20"));

// 如果前面字符都一样，但接下来没有可比字符了，这时比较长度，长度大的字符串就大

System.out.println("比较长度的例子：" + "a".compareTo("ab"));

// 3) concat 不会改动字符串原有内容，结果是返回了一个新的字符串

String s6 = "aaa";

String s7 = "bbb";

String s8 = s6.concat(s7); // 将s7字符串拼接至s6后方

System.out.println(s8);

// String s9 = s6 + s7; // 结果一样，但底层实现不一样

// 4) contains 判断字符串中是否包含了某个子串

String str10 = "aaa89ccc89";

System.out.println(str10.contains("89"));

// 十有xx

// 5) indexOf 获取某一个子串在整个字符串中的下标, 从0开始

// 如果子串不存在，那么下标是-1

// 从前往后找， 只匹配一次

System.out.println("下标值："+ str10.indexOf("89"));

// aaa.bbb.zip 如果查找一个文件的后缀, 用lastIndexOf 从后往前找

String s11 = "aaa.bbb.zip";

System.out.println("最后一个小数点的下标：" + s11.lastIndexOf("."));

// 6）.length() 字符串的长度 (中文也是一个字符)

System.out.println("长度是:" + "332ff中文".length());

// 7) 看字符串是否以某个子串开始

System.out.println("Hello, abc".startsWith("Ha"));

// 8) .substring 获取字符串的一个子串

String s12 = "abcdefg";

String s13 = s12.substring(2); // 从c开始截取之后的部分

System.out.println("截取的结果："+ s13);

// 参数1： 开始位置（包含该位置字符）

// 参数2： 结束位置 （不包含该位置字符）

String s14 = s12.substring(2, 5);

System.out.println("截取的结果："+ s14);

// 9) 转大小写

System.out.println("abCD".toLowerCase()); // 转小写

System.out.println("abCD".toUpperCase()); // 转大写

// 10）截取掉首尾的空白字符

System.out.println(" 123 456 ".trim());

## 字符串的比较

题目：

String s1 = "aaa";

String s2 = "aaa";

String s3 = new String("aaa");

String s4 = "aaa" + "bbb";

String s5 = "aaabbb";

String s6 = "bbb";

String s7 = s1 + s6;

s1 == s2 true

s1 == s3 false

s4 == s5 true

s5 == s7 false

思考一个问题：

"hello"

"hello"

"hello"

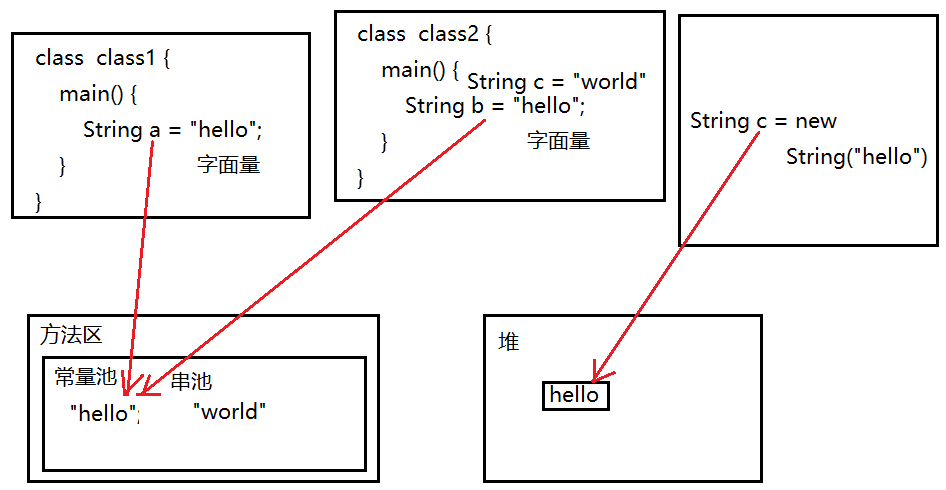
"hello"

"hello"

相同的字符串内容不应该存储多次，java这里对字符串类型的对象做了优化处理

对于字符串的字面量，只会存储一次，不会创建多个对象，存储于常量池中

对于new String，每次会分配新的内存，创建新的对象，存储于堆中



字符串字面量做加法:

String a = "aaa" + "bbb"; 在编译阶段就已经连接起来了 "aaabbb"

String b = "aaabbb";

a == b true

字符串的变量相加：

String a = "aaa";

String b = "bbb";

String c = "aaabbb";

String d = a + b;

c == d

StringBuffer 字符串的创建器

与String的区别:

String是字符串内容一旦创建好，不能改动

" aabb ".trim() => "aabb" (新的字符串, 旧的字符串内容不变)

StringBuffer 字符内容可以随时改动

String a = "aaa";

String b = "aaa";

String c = new String("aaa");

String d = new String("aaa");

System.out.println( a == b);

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

System.out.println( a == d);

System.out.println( c == d);

// 验证字面量加法

String e = "aaa" + "bbb";

String f = "aaabbb";

System.out.println(e == f);

// 验证变量相加

String a1 = "aaa";

String b1 = "bbb";

String c1 = "aaabbb";

String d1 = a1 + b1; // 在编译阶段被替换掉了

System.out.println(c1 == d1);

// 只要字符串加法中出现了一个变量，就会被转换为StringBuffer对象做加法, 最后会产生新的字符串对象

// 变量相加的底层实现

// StringBuffer tmp = new StringBuffer();

// tmp.append(a1);

// tmp.append(b1);

// String d1 = tmp.toString(); // 生了新的字符串对象(堆)

## 正则表达式

regular expresion

是一种验证规则，用来判断字符串是否符合某种特定格式:

年龄 "12"

手机号码 "15811112222" 11位数字

邮箱地址 ["aaa@sina.com"](mailto:\"aaa@sina.com\")

### 常见正则规则：

[0-9] 用来匹配一个数字 简写方式: \d

[a-z] 用来匹配任意一个小写英文字母

[A-Z] 用来匹配任意一个大写英文字母

[\u4e00-\u952f] 用来匹配任意一个汉字

[135] 用来匹配1或3或5

[1ab] 用来匹配1或a或b

[0-9a-zA-Z] 匹配数字+大小写英文

[0-9a-zA-Z\_] 单词字符 简写方式: \w

取反

[^0-9] 用来匹配一个非数字 简写方式: \D

[^a-z] 除了小写的a-z以外其它字符

[^0-9a-zA-Z\_] 非单词字符 简写方式: \W

空白字符

空格 \t 简写方式: \s

量词

{m, n} m表示最少出现的次数, n 表示最多出现的次数

{k} 恰好出现k次

{m,} 最少出现m次

{,n} 最多出现n次

{0,} 可以出现0到多次 简写方式: \*

{1,} 可以出现1到多次 简写方式: +

{0,1} 可以出现0到1次 简写方式: ?

正则的小数点

. 表示任意一个字符

\. 表示真正的小数点

验证邮箱地址： [xxxx@xxxx.xx](mailto:xxxx@xxxx.xx)

\w+@\w+\.\w+ // 简单写法，后面改进

### 原始字符串.matches(正则)

返回一个布尔值，表示原始字符串与正则是否匹配

System.out.println("a".matches("[0-9]")); // false

System.out.println("1".matches("[0-9]")); // true

// 正则表达式中的 \ 在java中需要写为 \\

System.out.println("1".matches("\\d")); // true

System.out.println("2".matches("[135]")); // false

System.out.println("3".matches("[135]")); // true

System.out.println("丈".matches("[\u4e00-\u4e0f]")); // true

System.out.println("严".matches("[\u4e00-\u4e0f]")); // false

// 验证单词字符

System.out.println("@".matches("[\\w]")); // false

System.out.println("\_".matches("[\\w]")); // true

System.out.println("a".matches("[\\w]")); // true

// 验证取反

System.out.println("2".matches("[^0-9]")); // false

System.out.println("y".matches("[^0-9]")); // true

// 空白字符

System.out.println(" ".matches("\\s")); // true

System.out.println("\t".matches("\\s")); // true

// {m,n}量词

System.out.println("147328".matches("[0-9]{11,11}")); // false

System.out.println("15811112222".matches("[0-9]{11}")); // true

// 验证邮箱

System.out.println("aaa443@cc".matches("\\w+@\\w+\\.\\w+")); // false

System.out.println("aaa443@cc.com".matches("\\w+@\\w+\\.\\w+")); // true

// abc 出现1到多次

// () 叫做分组，把括号内的内容作为一个整体看待

String str1 = "abcabc";

System.out.println(str1.matches("(abc)+"));

// a出现1到多次, b 出现1到多次, c 出现1到多次

// 错误的正则，+只影响了c, c可以出现1到多次，ab仅能出现一次

// String str1 = "abccccc";

// System.out.println(str1.matches("abc+"));

// abc 或 def 出现1到多次, | 代表组内的或者关系

System.out.println("abcabc".matches("(abc|def)+"));

### 原始字符串.split("正则")

根据正则表达式，将原始字符串按正则表达式切分

"aaa,bbb,ccc".split(","); => String[]{"aaa","bbb","ccc"}

String str = "aaa,bbb,ccc";

String[] result = str.split(",");

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

System.out.println(result[i]);

}

// 按空白字符拆分

String str2 = "aaa bbb ccc"; // \t

String[] r2 = str2.split("\\s");

System.out.println(Arrays.toString(r2));

// 有些特殊字符需要转义 ， 例如 |

String str3 = "aaa|bbb|ccc";

String[] r3 = str3.split("\\|");

System.out.println(Arrays.toString(r3));

### "原始字符串".replaceAll(正则, "替换的新内容")

将原始字符串中，所有与正则表达式匹配的地方，都进行替换操作, 返回一个新字符串

// 1. 基本用法

String str = "bbb,ccc,ddd,eee"; // 将所有,替换为空格

String r1 = str.replaceAll(",", " ");

System.out.println(r1);

System.out.println(str.replaceFirst(",", " ")); //只替换第一个匹配结果

// 2. 贪婪模式， +,\*,? 他们默认都会尽可能多地匹配内容 (了解)

// 因此 .+ 匹配的内容是 : aa</p><p>bbb</p><p>ffff

String str1 = "<p>aa</p><p>bbb</p><p>ffff</p>";

// 替换为 bbb bbb bbb

System.out.println(str1.replaceAll("<p>.+</p>", "bbb"));

// 3. 非贪婪模式

// +? 尽可能少地匹配内容

// \*? 尽可能少地匹配内容

// ?? 尽可能少地匹配内容

System.out.println(str1.replaceAll("<p>.+?</p>", "bbb"));

// 3. 替换中的分组 (了解) 例1：

// 最终是希望 结果为 aabbbffff

String str2 = "<p>aa</p><p>bbb</p><p>ffff</p>";

// 要替换为第一组的内容 ： $1

System.out.println(str2.replaceAll("<p>(.+?)</p>", "<a>$1</a>"));

// 替换中的分组 例2：

String str3 = "mary xing"; // => xing mary

System.out.println(str3.replaceAll("(.+?)\\s(.+)", "$2 $1"));