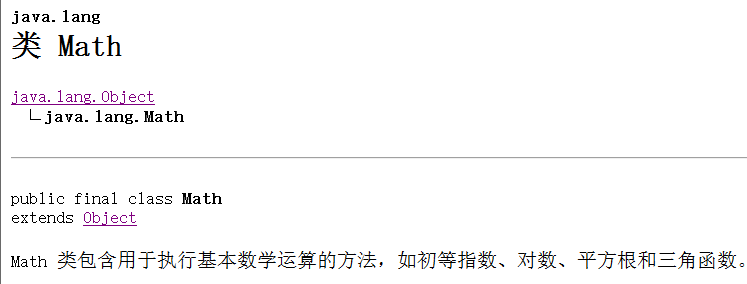
# day12

## Math类(了解)

1. Math类 : JDK提供的对于数学进行基本运算的类



1. Math类在API中, 没有给出任何构造, 原因 : Math类中的唯一构造方法私有, 不能实例化对象, 因此, Math类中的所有成员和方法, 都使用static静态修饰
2. Math类中静态成员常量

public static final double E; 表示自然对数--->2.718...

public static final double PI; 表示圆周率--->3.14...

1. Math类中常用的方法功能:
2. abs(int i) : 重载方法, 获取到参数i的绝对值(绝对值都是正数)
3. ceil(double d) : 将参数数据向上取整, 返回值类型double
4. floor(double d) : 将参数数据向下取整, 返回值类型double
5. max(double d, double d1) : 获取到两数中的较大值
6. min(double d, double d1) : 获取到两数中的较小值
7. random() : 生成[0.0, 1.0)之间的随机小数
8. round(double d) : 根据提供的参数的第一个小数位数, 决定数据是向上取整或者是向下取整(第一位小数数值 >= 5, 向上取整; 否则向下取整), 返回一个long类型结果

代码

|  |
| --- |
| package com.ujiuye.otherclass;  public class MathDemo {  public static void main(String[] args) {  // 1. Math类中静态成员常量  System.out.println(Math.E);  System.out.println(Math.PI);    // 1)abs(int i) : 重载方法, 获取到参数i的绝对值(绝对值都是正数)  System.out.println(Math.abs(-3));// 3    // 2)ceil(double d) : 将参数数据向上取整, 返回值类型double  System.out.println(Math.ceil(3.01));// 4.0    // 3)floor(double d) : 将参数数据向下取整, 返回值类型double  System.out.println(Math.floor(3.99));// 3.0    // 4)max(double d, double d1) : 获取到两数中的较大值  // 5)min(double d, double d1) : 获取到两数中的较小值  System.out.println(Math.max(3.14, 3.14));// 3.14  System.out.println(Math.min(2.13, -9.0));// -9.0    // 6)random() : 生成[0.0, 1.0)之间的随机小数  System.out.println(Math.random());    /\*7)round(double d) : 根据提供的参数的第一个小数位数, 决定数据是向上取整或者是向下取整(第一位小数数值 >= 5, 向上取整; 否则向下取整), 返回一个long类型结果\*/  System.out.println(Math.round(6.77)); // 7  System.out.println(Math.round(7.51)); // 8  System.out.println(Math.round(9.49)); // 9  }  } |

## System类(了解)

1. System类 : 来自于java.lang包,表示系统资源的类型, 也封装了一些数组复制功能
2. System类中, 构造方法私有化, 因此不能实例化对象, 所有成员和方法, 都使用static静态修饰
3. System类中的成员常量:
4. System.in : 标准输入流资源, 主要提供数据的输入源, 默认通过键盘录入

Scanner sc = new Scanner(System.in);

1. System.out : 标准的输出流资源(打印流资源), 主要就是将数据进行输出打印, 默认打印在控制台上

System.out.println(“abc”);

1. System.err : 标准的错误流, err全称error,表示错误; Java代码中的异常信息都是使用标准错误流打印, 打印出的数据结果是红色的突出显示
2. System类中获取当前时间方法功能 :

static long currentTimeMillis() : 表示获取到当前系统时间所对应的毫秒值结果, 返回值类型long类型

1. 毫秒值 : 时间的计算单位, 1000毫秒 = 1秒
2. Java中时间原点(对应毫秒值为0), 1970年1月1日 00:00:00
3. 可以利用此方法, 计算代码的运行时间

代码

|  |
| --- |
| package com.ujiuye.otherclass;  public class SystemDemo {  public static void main(String[] args) {  long beginTime = System.currentTimeMillis();    for(int i = 0; i <= 1000000; i++) {  int w = i + 5;  }    long endTime = System.currentTimeMillis();  System.out.println(endTime-beginTime);  }  } |

## StringBuilder

### StringBuilder概述

1. StringBuilder类 : 来自于java.lang包 , 表示可变字符序列(可变字符串), 称为字符串生成器或者字符串缓冲区
2. String与StringBuilder之间的比较
3. String类型表示字符串, 字符串是常量, 因为字符串底层使用一个private final char[] value表示字符序列, final不可变, 同时也没有提供任何修改value数组中数据的方式
4. StringBuilder 类型表示可变字符序列(字符串), 底层也是使用一个字符数组char[] value存储字符序列, 但是对于value , 可以进行修改
5. StringBuilder使用场景 :
6. String类型因为表示的都是字符串常量, 因此在进行大量字符串数据拼接时候, 性能会比较差, 会额外在内存中开辟无用字符串空间,为了解决字符串拼接浪费内存问题, 引入StringBuilder可变字符序列的学习, 可以一定程度解决字符串拼接浪费内存问题
7. StringingBuilder主要通过append和insert两个方法向缓冲区中添加字符数据, 如果缓冲区容量不够, 代码底层自动进行容量的扩容, 支持存储和追加字符序列

### StringBuilder构造方法

1. StringBuilder() : 创建出一个字符串缓冲区, 默认初始容量大小为16字符
2. StringBuilder(int count) : 创建出一个字符串缓冲区,初始容量大小为参数count
3. StringBuilder(String str) : 创建出一个字符串缓冲区, 容量为str.length() + 16

代码

|  |
| --- |
| package com.ujiuye.stringbuilder;  public class Demo01\_StringBuilder构造 {  public static void main(String[] args) {  // "a" "b" "ab" "c" "abc" "d" "abcd" "e" "abcde"  // 需要9块常量池空间, 比较浪费内存, 因此StringBuilder就是为了解决字符串拼接浪费内存问题  String s = "a" + "b" + "c" + "d" + "e";    StringBuilder sb = new StringBuilder();  System.out.println(sb);    StringBuilder sb1 = new StringBuilder(56);    StringBuilder sb2 = new StringBuilder("abc");  System.out.println(sb2);// abc  }  } |

### StringBuilder添加功能

在 StringBuilder 上的主要操作是 append 和 insert 方法，可重载这些方法，以接受任意类型的数据。每个方法都能有效地将给定的数据转换成字符串，然后将该字符串的字符追加或插入到字符串生成器中。append 方法始终将这些字符添加到生成器的末端；而 insert 方法则在指定的点添加字符。

1. append (Object obj) : 可以将参数obj转换成字符串, 以末尾添加的方式追加到StringBuilder的字符串缓冲区中,方法返回正在操作的 StringBuilder类型对象本身
2. insert(int index, Object obj) : 将参数obj添加到指定的index索引位置上, 方法返回正在操作的 StringBuilder类型对象本身

注意 : insert方法, 第一个参数index最大可以为当前StringBuilder中存储的字符个数(StringBuilder的长度), 如此功能就会与append一致, 表示末尾追加

如果 sb 引用 StringBuilder 的实例，则 sb.append(x) 和 sb.insert(sb.length(), x) 具有相同的效果

1. length() : 表示获取StringBuilder中的存储的字符的个数(长度),返回int类型结果
2. [capacity](mk:@MSITStore:C:\\Users\\96243\\Desktop\\JDK_API_1_6_zh_CN.CHM::/java/lang/../../java/lang/StringBuilder.html" \l "capacity())(): 表示当前StringBuilder的容量大小

代码

|  |
| --- |
| package com.ujiuye.stringbuilder;  public class Demo02\_StringBuilder添加功能 {  public static void main(String[] args) {  //1. 创建出一个StringBuilder字符串缓冲区  StringBuilder sb = new StringBuilder();  // 1) append : 末尾添加数据  sb.append(true);  sb.append(3.14).append('z').append(new Object());  System.out.println(sb);    // 2) insert : 指定索引位置添加数据  sb.insert(3, "添加");  System.out.println(sb);//tru添加e3.14zjava.lang.Object@15db9742    // 3) length : 获取缓冲区中的字符个数  System.out.println(sb.length());// 36  // 相当于末尾追加  sb.insert(sb.length(), "end");  System.out.println(sb);    // 4) 获取缓冲区的容量  System.out.println(sb.capacity());// 70  }  } |

### StringBuilder删除功能

1. deleteCharAt(int index) : 删除指定index索引位置上的对应的字符, 返回值为当前正在操作的StringBuilder类型对象本身
2. delete(int beginIndex, int endIndex) : 删除从beginIndex索引开始到endIndex-1索引之间的字符序列, 返回当前正在操作的StringBuilder类型对象本身

代码

|  |
| --- |
| package com.ujiuye.stringbuilder;  public class Demo03\_StringBuilder删除功能 {  public static void main(String[] args) {  StringBuilder sb = new StringBuilder("helloworld");  //1.deleteCharAt(int index) : 删除指定index索引位置上的对应的字符 sb.deleteCharAt(3);  System.out.println(sb);//heloworld    // 2.delete(int beginIndex, int endIndex) : 删除从beginIndex索引开始到endIndex-1索引位置结束, 删除中间的字符序列  sb.delete(1, 5);// elow  System.out.println(sb);//horld  }  } |

### StringBuilder替换和反转功能

1. replace(int beginIndex, int endIndex, String str) : 从beginIndex索引位置开始到endIndex-1索引之间的字符序列, 使用字符串str进行替换,返回当前正在操作的StringBuilder类型对象本身
2. reverse() : 将StringBuilder中的字符内容进行逆序排序,返回当前正在操作的StringBuilder类型对象本身

代码

|  |
| --- |
| package com.ujiuye.stringbuilder;  public class Demo04\_StringBuilder替换和反转 {  public static void main(String[] args) {  StringBuilder sb = new StringBuilder("helloworld");  //1.replace(int beginIndex, int endIndex, String str) : 从beginIndex索引位置开始到endIndex-1索引位置结束, 中间字符序列使用字符串str进行替换  sb.replace(1, 4, "替换");  System.out.println(sb); // h替换oworld    // 2.reverse() : 将StringBuilder中的字符内容进行逆序排序,返回当前正在操作的StringBuilder类型对象本身  sb.reverse();  System.out.println(sb);  }  } |

### StringBuilder和StringBuffer之间的比较

从JDK1.0版本开始, 一直都是使用StringBuffer类型, 作为字符串缓冲区, 解决字符串大量拼接浪费内存问题; StringBuffer是线程安全的(每一个方法都添加保证数据安全的验证, 因此方法执行时间就会长, 性能就会比较低), 在JDK1.5版本时,推出了一个与StringBuffer等价类StringBuilder, 两个类唯一的不同就是StringBuilder线程不安全的, 使用在单线程场景下;

比较两个类型:

a : 出现的版本不同

1. StringBuffer出现于JDK1.0版本
2. StringBuilder出现于JDK1.5版本

b : 线程安全性不同

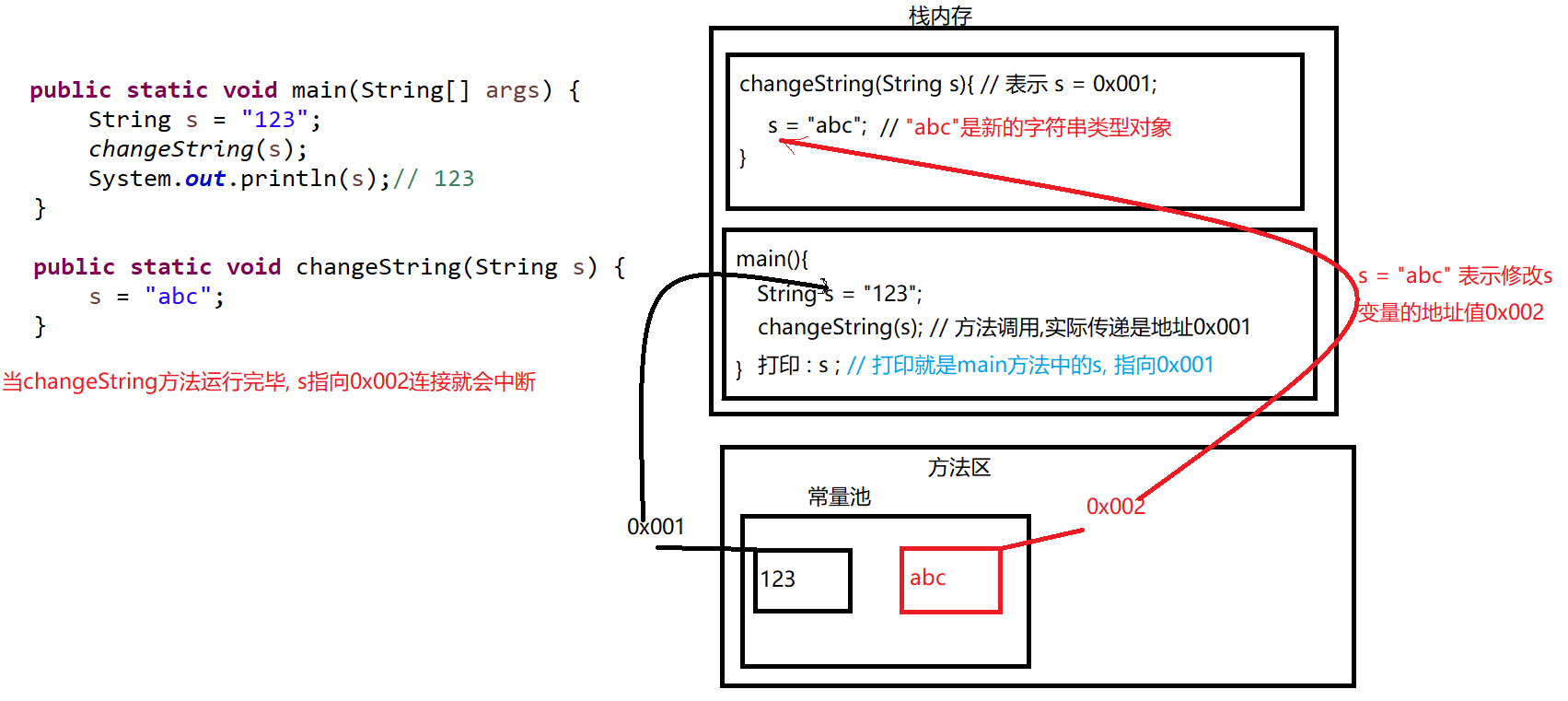
1. StringBuffer线程安全的
2. StringBuilder线程不安全

c : 运行效率不同

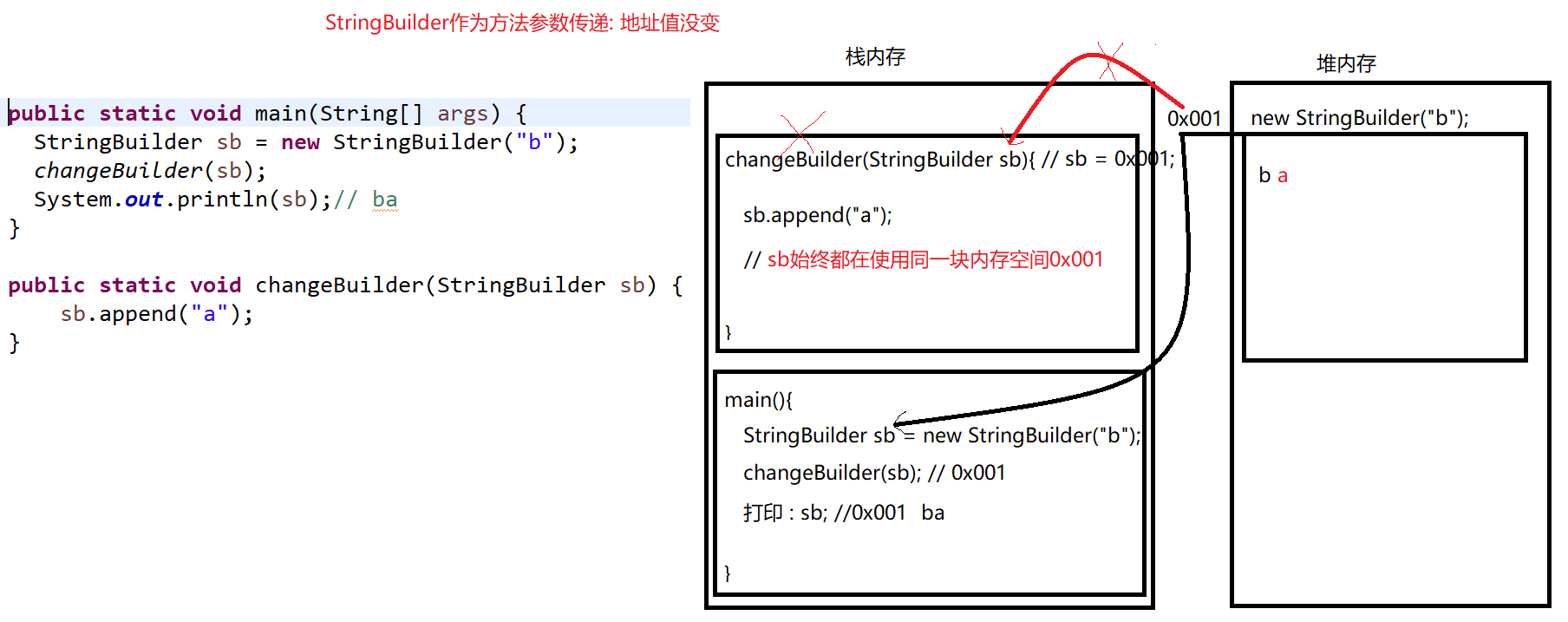
1. StringBuffer执行效率比较低
2. StringBuilder执行效率比较高

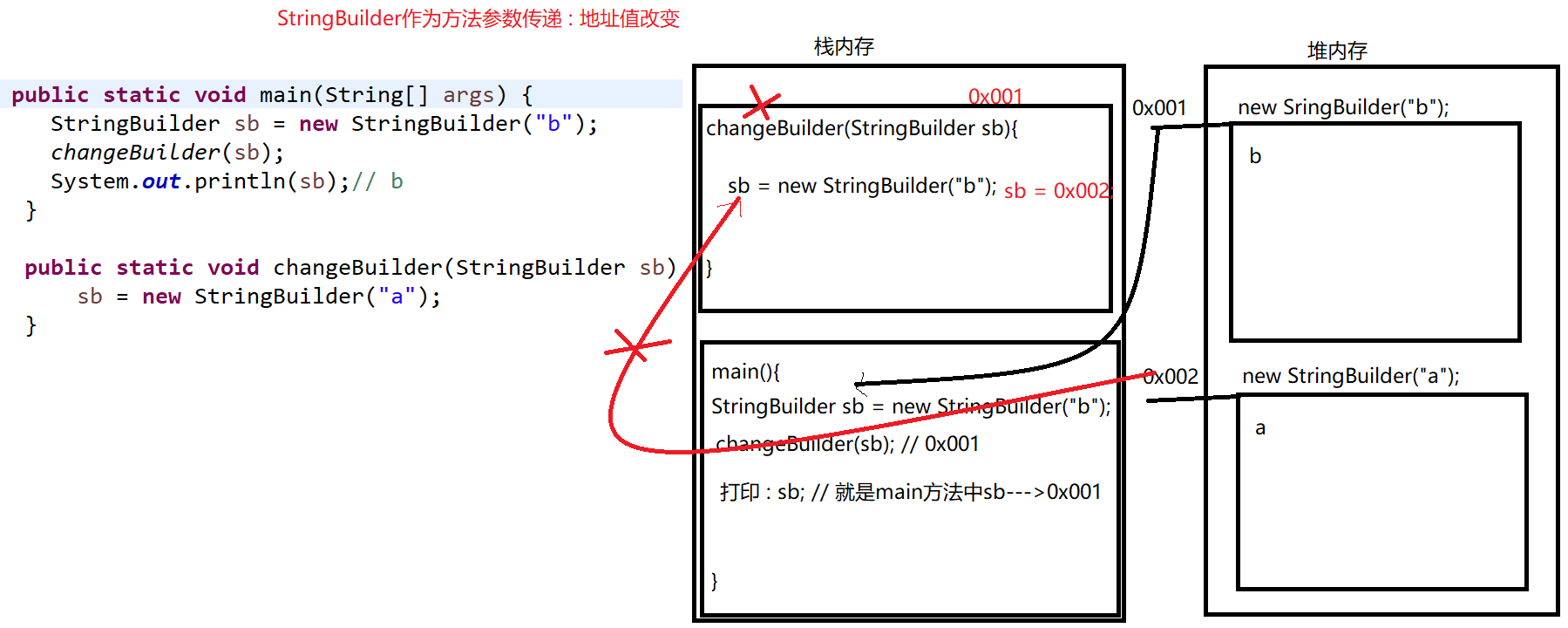
### String与StringBuilder作为方法参数

1. String作为方法参数传递



2. StringBuilder作为方法参数传递





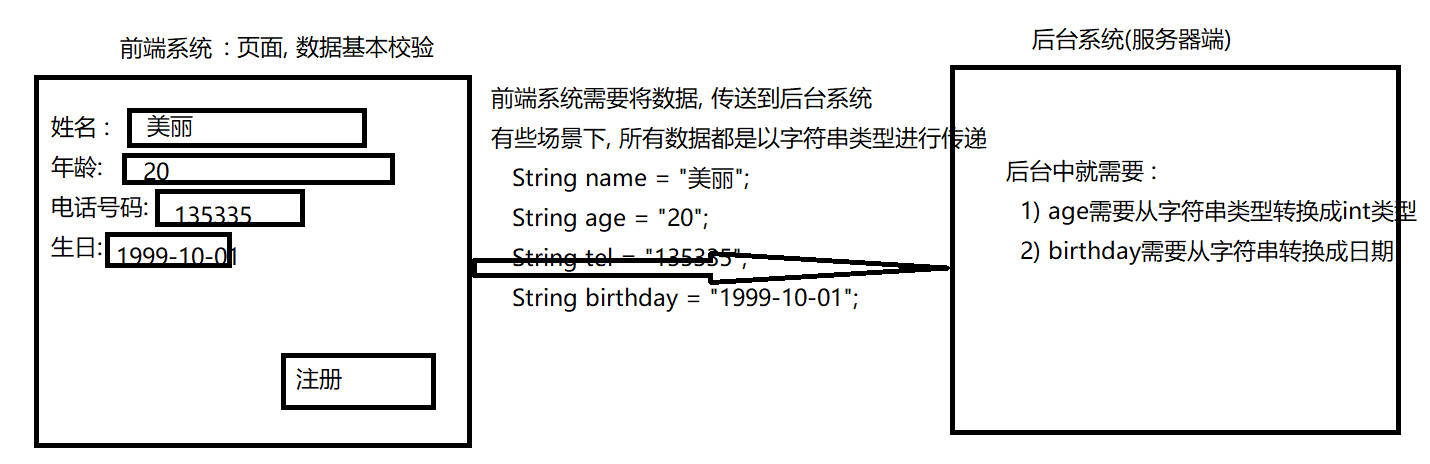
代码

|  |
| --- |
| package com.ujiuye.stringbuilder;  public class Demo05\_String和Builder作为参数 {  public static void main(String[] args) {    String s = "123";  changeString(s);  System.out.println(s);// 123    StringBuilder sb = new StringBuilder("b");  changeBuilder(sb);  System.out.println(sb);// b  }    public static void changeBuilder(StringBuilder sb) {// 0x001  sb = new StringBuilder("a");  }    /\*public static void changeBuilder(StringBuilder sb) {  sb.append("a");  }\*/    public static void changeString(String s) {// s = 0x001  s = s + "1";// s = 0x002  }  } |

## 基本数据类型包装类

### 基本数据类型包装类的概述

1. 基本数据类型一共4类8种, 每一种基本类型都表示最简单,最直接的数据; 并且基本数据类型没有任何方法封装, 只能表示数据;
2. 就是因为基本数据类型中没有任何方法功能, 无法对基本数据类型进行封装和操作, 实际开发中, 也会涉及到基本数据类型与其他数据类型进行转换, 例如 : 基本数据类型与字符串之间的转换, 因此需要为每一种基本数据类型定义出其对应的包装类, 有了包装类之后,可以在包装类中封装对基本数据类型很多操作和转换功能

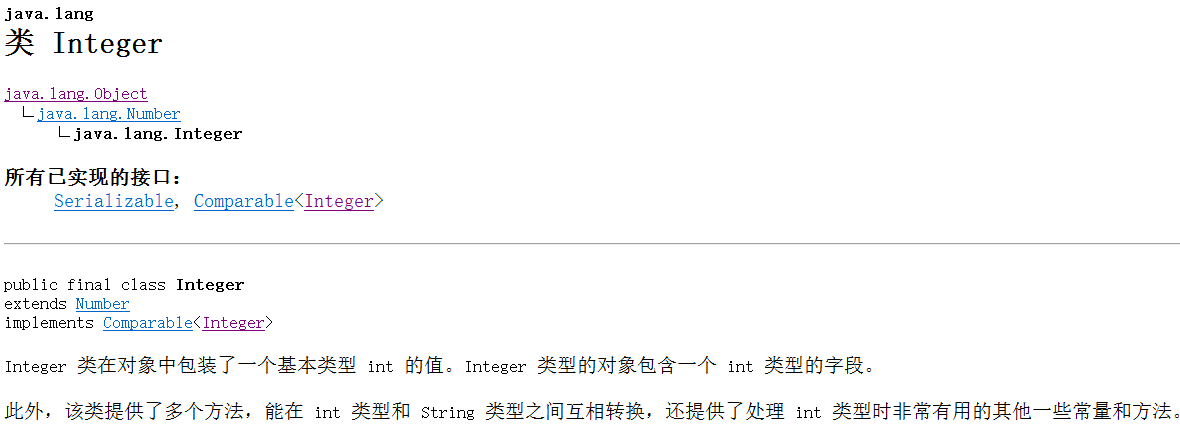


1. 4类8种基本数据类型对应包装类

|  |  |
| --- | --- |
| 基本数据类型 | 包装类 |
| byte | Byte |
| short | Short |
| int | Integer |
| long | Long |
| float | Float |
| double | Double |
| char | Character |
| boolean | Boolean |

### Integer类型

基本数据类型对应的包装类 : 设计思路和使用上非常相似, 因此讲解最常用的Integer类型, 其他包装类举一反三



#### Integer类型构造方法

1. Integer(int i) : 将int类型整数包装成Integer类型
2. Integer(String s) : 将s字符串所表示的数值转换成Integer类型

代码

|  |
| --- |
| package com.ujiuye.integer;  public class Demo01\_Integer构造 {  public static void main(String[] args) {  Integer i = new Integer(15);  System.out.println(i);    Integer i2 = new Integer("11");  System.out.println(i2);    // 并不是所有的字符串都可以转换成数值, 如果不能转换, 报错 : NumberFormatException 数字格式化异常  Integer i3 = new Integer("飞机");  System.out.println(i3);  }  } |

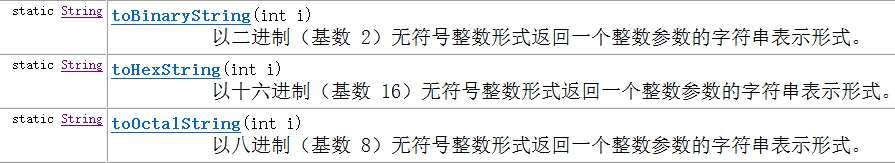
#### Integer类型中常用方法

1. xxxValue() : xxx表示基本数据类型, 方法功能就是将Integer类型转换成其他基本数据类型

举例 : byteValue() : 将Integer类型所表示的数据转换成基本数据类型byte

1. static int parseInt(String s): 将字符串s表示的数值转换成int类型整数
2. 十进制与其他进制之间转换

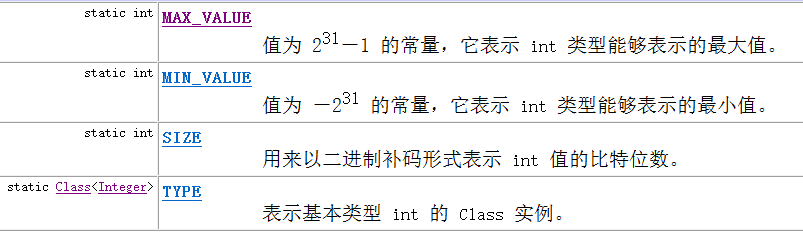
100100100二进制 16进制数带有字母, 因此这些方法返回值类型String类型表示



代码

|  |
| --- |
| package com.ujiuye.integer;  public class Demo02\_Integer常用方法 {  public static void main(String[] args) {  Integer i = new Integer(26);  // 1. byteValue() : 将Integer类型所所表示的数据转换成基本数据类型byte  byte b = i.byteValue();  System.out.println(b);    // 2. static parseInt(String s): 将字符串s表示的数值转换成int类型整数  int num = Integer.parseInt("-99");  System.out.println(num + 1);    // 3. 将int类型转换成String类型 -99 + ""  String s = String.valueOf(-99);  System.out.println(s + 1);// -991    // 4. 十进制转换成其他进制  System.out.println(Integer.toBinaryString(5));// 101  System.out.println(Integer.toOctalString(10));// 12  System.out.println(Integer.toHexString(26));//1a  }  } |

#### Integer类型中的成员常量(了解)



代码

|  |
| --- |
| package com.ujiuye.integer;  public class Demo03\_Integer成员常量 {  public static void main(String[] args) {    System.out.println(Integer.MAX\_VALUE);// 2147483647  System.out.println(Integer.MIN\_VALUE);// -2147483648    // int类型在内存中占有4字节, 1字节就是8个连续比特位(1个比特位就是1个2进制位)  System.out.println(Integer.SIZE);// 32    System.out.println(Integer.TYPE);// int  }  } |

#### 自动装箱和自动拆箱(JDK1.5特性)

1. 装箱 : 将基本数据类型封装成一个引用数据类型包装类, 称为装箱

基本数据类型----> 引用数据类型

1. 拆箱 : 将引用数据类型包装类中封装的基本数据类型获取到

引用数据类型----> 基本数据类型

1. 自动装箱 : 可以使用基本数据类型直接给对应引用数据类型进行赋值
2. 自动拆箱 : 引用数据类型包装类可以直接给基本数据类型进行赋值; 引用类型可以直接与基本类型进行运算

代码

|  |
| --- |
| package com.ujiuye.integer;  public class Demo04\_自动拆装箱 {  public static void main(String[] args) {  // 1.自动装箱  Integer i = 10;    // 2. 自动拆箱  int num = i;    // 分析:  // 1) i对应的Integer类型数据, 进行自动拆箱, 变成int类型 10 + 5 = 15---> int  // 2) 将1)中计算出的结果int类型的15, 赋值给引用类型i , 自动装箱  i = i + 5;  }  } |

## 正则表达式

### 正则表达式的概述

1. 正则表达式 : 本质就是一个字符串, 但是这个字符串可以表示一类字符串需要遵守的规则, 普通字符串只能表示本身, 正则表达式可以表示一类字符串
2. 正则表达式的作用: 可以进行字符串的规则验证

举例 : 验证客户的电话号码 : 13566778899

要求 : 电话号码中全数字; 第一位必须是1; 第二位必须是3-9之间的数值; 剩下可以是任意数字; 长度必须为11位;

举例 : 验证QQ号码注册 : 2345678

要求 : 全数字; 第一位不能是0; 长度5-15位之间

1. 正则表达式优势和弊端:

a : 优势, 可以使用简单表达式验证出复杂字符串规则

b : 弊端, 写出正确的正则表达式比较难, 后期维护困难

### 字符类

1. 字符类 : 表示每次匹配一个字符规则, 使用[]表示, 每次可以匹配中括号中一个符合条件的字符
2. 常用字符类匹配:

[abc] : 匹配a或者b或者c中任意一个字符

[^abc] : ^表示取反, 匹配1个字符, 但是这个字符不能是a或者b或者c

[a-zA-Z0-9] : 如果有数据是连续, 可以使用-来表示区间, 表示匹配一个小写字母或者一个大写字母或者一个数字

注意 : 正则表达式匹配, 使用String类型中方法功能 :



代码

|  |
| --- |
| package com.ujiuye.regex;  public class Demo01\_字符类 {  public static void main(String[] args) {  // 1.定义出一个正则表达式  String regex = "[abc]";  // 2. 定义出一个需要验证字符串  String s = "a";    System.out.println(s.matches(regex));// true  System.out.println("b".matches(regex));// true  System.out.println("ab".matches(regex));// false  System.out.println("1".matches(regex));// false    System.out.println("------------");    String regex2 = "[^abc]";  System.out.println("b".matches(regex2));// false  System.out.println(" ".matches(regex2));// true    System.out.println("+++++++++++++");  String regex3 = "[a-zA-Z0-9]";  System.out.println("X".matches(regex3));// true  System.out.println("?".matches(regex3));// false  System.out.println("-1".matches(regex3));// false  }  } |