

# 1. 纲要

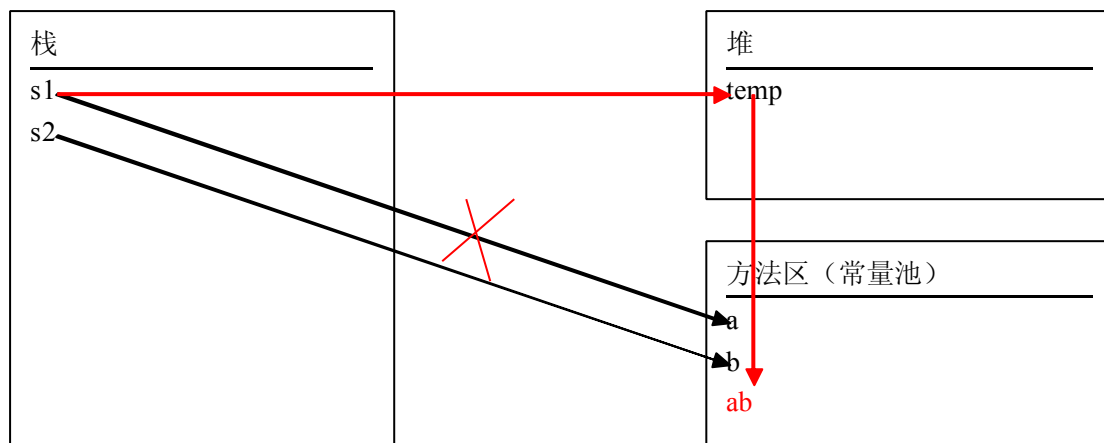
- a) String
- b) StringBuffer
- c) 基础类型对应的 8 个包装类
- d) 日期相关类
- e) 数字相关类
- f) Random
- g) Enum

## 2. 内容

### 2.1、String

**2.1.1、String 类是不可变类**，也就是说 String 对象声明后，将不可修改

```
public class StringTest01 {  
  
    public static void main(String[] args) {  
  
        String s1 = "a";  
        String s2 = "b";  
        s1=s1 + s2;//ab  
        //new String("a");  
        System.out.println(s1);  
    }  
}
```



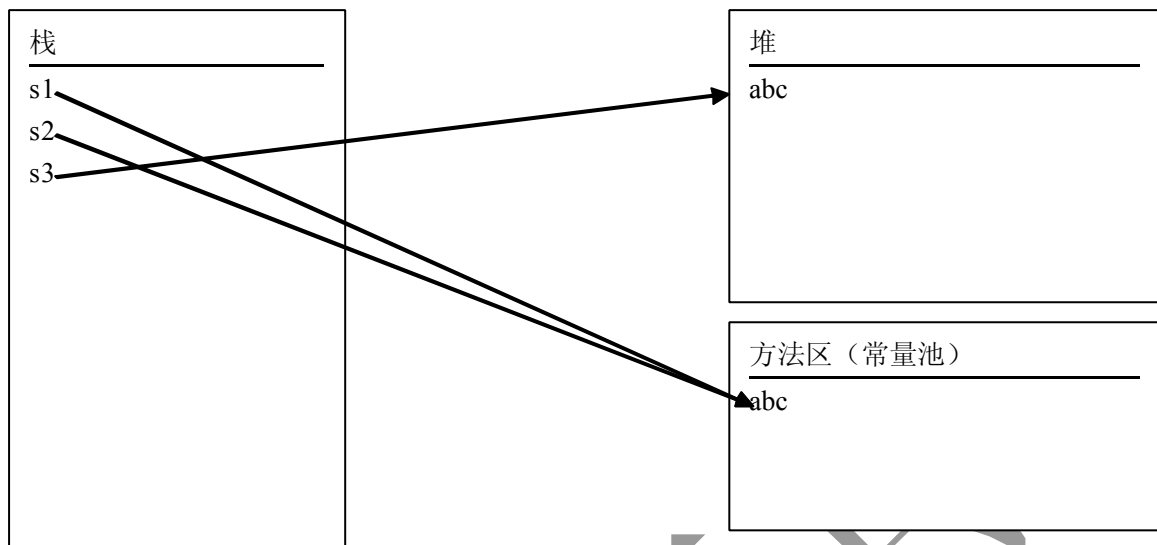
从以上内存图，大家可以看到，String 对象赋值后不能再修改，这就是不可变对象，如果对字符串修改，那么将会创建新的对象

**注意：**只要采用双引号赋值字符串，那么在编译期将会放到方法区中的字符串的常量池里，如果是运行时对字符串相加或相减会放到堆中（放之前会先验证方法区中是否含有相同的字符串常量，如果存在，把地址返回，如果不存在，先将字符串常量放到池中，然后再返回该对象的地址）

### 2.1.2、String s1 = “abc”和 String s2 = new String(“abc”)

```
public class StringTest02 {  
  
    public static void main(String[] args) {  
  
        String s1 = "abc";  
  
        String s2 = "abc";  
  
        String s3 = new String("abc");  
  
        System.out.println("s1==s2, " + (s1==s2));  
        System.out.println("s2==s3, " + (s2==s3));  
        System.out.println("s2 equals s3, " + (s2.equals(s3)));  
  
    }  
}
```

```
D:\share\JavaProjects\j2se\chapter06>java StringTest02  
s1==s2, true  
s2==s3, false  
s2 equals s3, true  
D:\share\JavaProjects\j2se\chapter06>
```

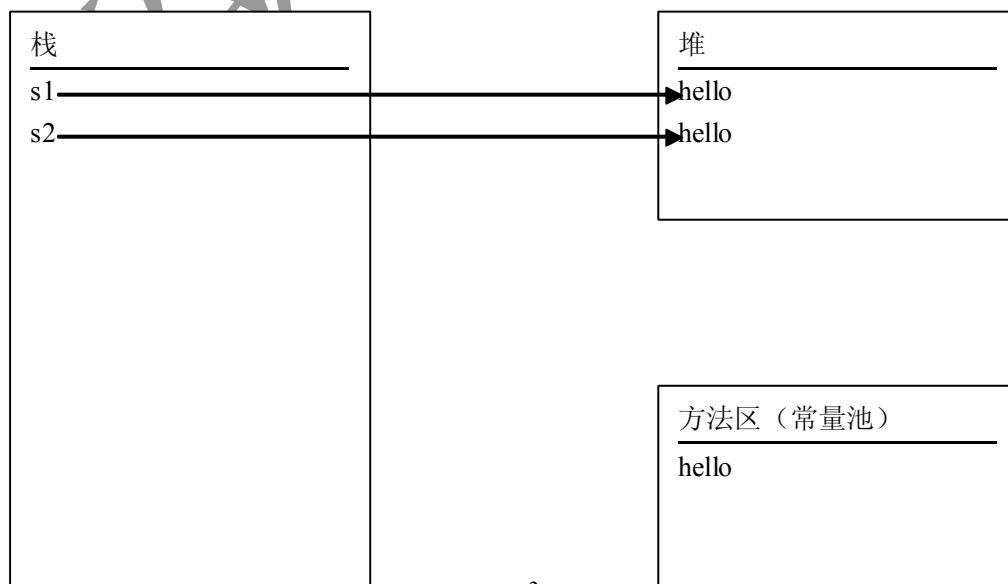


- 如果是采用双引号引起来的字符串常量，首先会到常量池中去查找，如果存在就不再分配，如果不存在就分配，常量池中的数据是在编译期赋值的，也就是生成 class 文件时就把它放到常量池里了，所以 s1 和 s2 都指向常量池中的同一个字符串“abc”
- 关于 s3，s3 采用的是 new 的方式，在 new 的时候存在双引号，所以他会到常量区中查找“abc”，而常量区中存在“abc”，所以常量区中将不再放置字符串，而 new 关键字会在堆中分配内存，所以在堆中会创建一个对象 abc，s3 会指向 abc
- 如果比较 s2 和 s3 的值必须采用 equals，String 已经对 equals 方法进行了覆盖

### 2.1.3、String 面试题分析

```
String s1 = new String("hello");  
String s2 = new String("hello");
```

以上代码创建了几个对象？



通过以上分析，大家会看到创建了 3 个对象，堆区中 2 个，常量池中 1 个

通过以上分析，使用 String 时，不建议使用 new 关键字，因为使用 new 会创建两个对象

记住：堆区中是运行期分配的，常量池中是编译器分配的

## 2.1.4、String 常用方法简介

1. endsWith：判断字符串是否以指定的后缀结束
2. startsWith，判断字符串是否以指定的前缀开始
3. equals，字符串相等比较，不忽略大小写
4. equalsIgnoreCase，字符串相等比较，忽略大小写
5. indexOf，取得指定字符在字符串的位置
6. lastIndexOf，返回最后一次字符串出现的位置
7. length，取得字符串的长度
8. replaceAll，替换字符串中指定的内容
9. split，根据指定的表达式拆分字符串
10. substring，截子串
11. trim，去前尾空格
12. valueOf，将其他类型转换成字符串

## 2.1.5、使用 String 时的注意事项

因为 String 是不可变对象，如果多个字符串进行拼接，将会形成多个对象，这样可能会造成内存溢出，会给垃圾回收带来工作量，如下面的应用最好不要用 String

【代码示例】

```
public class StringTest04 {  
  
    public static void main(String[] args) {  
        String s = "";  
        for (int i=0; i<100; i++) {  
            //以下语句会生成大量的对象  
            //因为 String 是不可变对象  
            //存在大量的对象相加或相减一般不建议使用 String  
            //建议使用 StringBuffer 或 StringBuilder  
            s+=i;// s = s+i;  
        }  
    }  
}
```

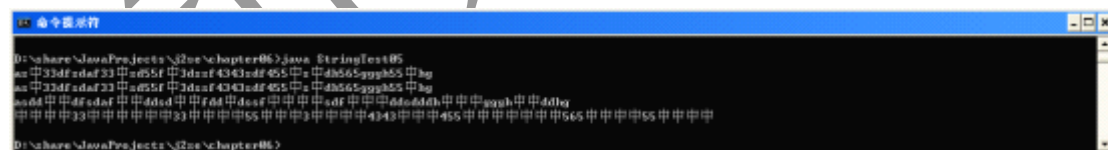
## 2.1.5、正则表达式初步

正则表达式，主要可以用来做字符串处理，可以描述特定的字符模式，如：“a{2}”表示由两个字符“a”构成的字符串，等同于普通字符串“aa”，如“\d”代表任意一个数字 0~9

public boolean matches(String regex)，返回此字符串是否匹配给定的正则表达式。

public String replaceAll(String regex, String replacement)使用给定的 replacement 字符串替换此字符串匹配给定的正则表达式的每个子字符串。

```
public class StringTest05 {  
  
    public static void main(String[] args) {  
        String s1 = "asdd33dfsdf33ddsd55fdd3dssf4343sdf455ddsdddh565gggh55ddhg";  
  
        //将 dd 替换为"中"  
        System.out.println(s1.replaceAll("dd", "中"));  
  
        //将 dd 替换为"中"  
        System.out.println(s1.replaceAll("d{2}", "中"));  
  
        //将数字替换为"中"  
        System.out.println(s1.replaceAll("\\d", "中"));  
  
        //将非数字替换为"中"  
        System.out.println(s1.replaceAll("\\D", "中"));  
  
    }  
}
```



## 2.2、StringBuffer 和 StringBuilder

### 2.2.1、StringBuffer

StringBuffer 称为字符串缓冲区，它的工作原理是：预先申请一块内存，存放字符序列，如果字符序列满了，会重新改变缓存区的大小，以容纳更多的字符序列。StringBuffer 是可变对象，这个是 String 最大的不同

```
public class StringBufferTest01 {
```

```
public static void main(String[] args) {  
    StringBuffer sbStr = new StringBuffer();  
    for (int i=0; i<100; i++) {  
        //sbStr.append(i);  
        //sbStr.append(",");  
  
        //方法链的编程风格  
        sbStr.append(i).append(",");  
  
        //拼串去除逗号  
        //sbStr.append(i);  
        //if (i != 99) {  
        //    sbStr.append(",");  
        //}  
    }  
  
    //可以输出  
    System.out.println(sbStr);  
    System.out.println("");  
    System.out.println(sbStr.toString());  
    System.out.println("");  
    //去除逗号  
    System.out.println(sbStr.toString().substring(0,sbStr.toString().length()-1));  
    System.out.println("");  
    System.out.println(sbStr.substring(0, sbStr.length()-1));  
}  
}
```

## 2.2.2、StringBuilder

用法同 `StringBuffer`，`StringBuilder` 和 `StringBuffer` 的区别是 `StringBuffer` 中所有的方法都是同步的，是线程安全的，但速度慢，`StringBuilder` 的速度快，但不是线程安全的

## 2.3、基本类型对应的包装类

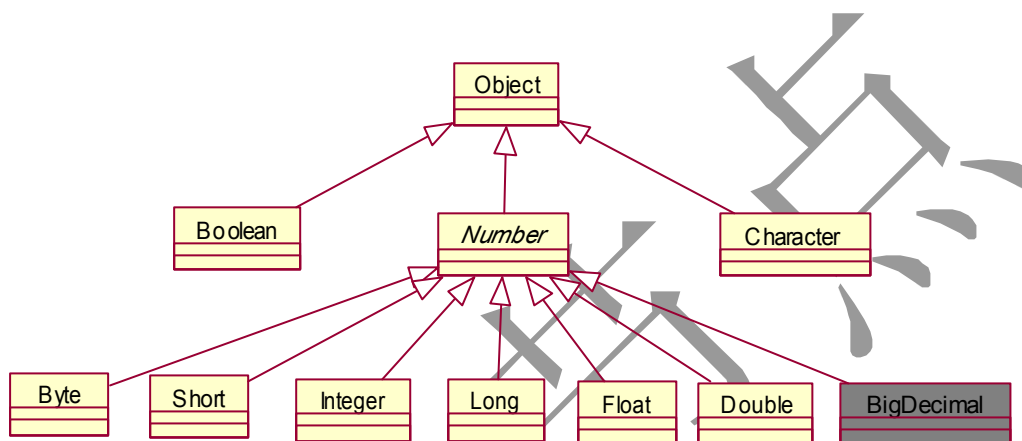
### 2.3.1、包装类概述

基本类型的包装类主要提供了更多的实用操作，这样更容易处理基本类型。所有的包装类都是 `final` 的，所以不能创建其子类，包装类都是不可变对象

基本类型	包装类
byte	Byte

short	Short
char	Character
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean

### 2.3.2、类层次结构



除了 boolean 和 Character 外，其它的包装类都有 valueOf()和 parseXXX 方法，并且还具有 byteVaue(),shortVaue(),intValue(),longValue(),floatValue()和 doubleValue()方法，这些方法是最常用的方法

```
public class IntegerTest01 {  
  
    public static void main(String[] args) {  
        int i1 = 100;  
        Integer i2 = new Integer(i1);  
        double i3 = i2.doubleValue();  
  
        String s = "123";  
  
        int i4 = Integer.parseInt(s);  
  
        Integer i5 = new Integer(s);  
  
        Integer i6 = Integer.valueOf(s);  
    }  
}
```

## 2.3.4、JDK5.0 的新特性

在 JDK5.0 以前，包装类和基本类型做运算时，必须将包装类转换成基本类型才可以，而 JDK5.0 提供 Auto-boxing/unboxing（自动装箱和拆箱）

- 自动将基础类型转换为对象
- 自动将对象转换为基础类型

```
public class IntegerTest01 {  
  
    public static void main(String[] args) {  
  
        //jdk1.5 以前版本，必须按如下方式赋值  
        Integer i1 = new Integer(100);  
  
        //jdk1.5 及以后版本支持  
        //自动装箱  
        Integer i2 = 100;  
  
        //jdk1.5 及以后版本支持  
        //自动拆箱  
        int i3 = i2;  
  
        //jdk1.5 以前版本，必须按如下方式赋值  
        int i4 = i2.intValue();  
  
    }  
}
```

## 2.4、日期类

常用日期类：

```
java.util.Date  
java.text.SimpleDateFormat  
java.util.Calendar
```

```
import java.util.Date;  
import java.text.SimpleDateFormat;  
import java.util.Calendar;  
  
public class DateTest01 {  
  
    public static void main(String[] args) throws Exception{  
  
        //取得今天的日期  
        Date today = new Date();  
        System.out.println(today);  
  
    }  
}
```



```
//格式化日期
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
System.out.println(sdf.format(today));

Calendar c = Calendar.getInstance();
System.out.println(c.get(Calendar.DAY_OF_MONTH));
//取得 2000-10-01 为星期几
Date d = new SimpleDateFormat("yyyy-MM-dd").parse("2000-10-01");
c.setTime(d);
System.out.println(c.get(Calendar.DAY_OF_WEEK));

    }
}
```

## 2.5、数字类

Java.text.DecimalFormat 和 java.math.BigDecimal

【示例代码】，DecimalFormat

```
import java.text.DecimalFormat;

public class DecimalTest01 {

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

        //加入千分位，保留两位小数
        DecimalFormat df = new DecimalFormat("###,###.##");
        System.out.println(df.format(1234.23452));

        //加入千分位保留 4 位小数，不够补零
        System.out.println(new DecimalFormat("###,###.0000").format(12345.12));

    }
}
```

【示例代码】，BigDecimal 可以精确计算，特别是财务数据

```
import java.math.BigDecimal;

public class BigDecimalTest01 {

    public static void main(String[] args) throws Exception{
        BigDecimal v1 = new BigDecimal(10);
        BigDecimal v2 = new BigDecimal(20);
        //相加运算
        BigDecimal v3 = v1.add(v2);
    }
}
```

```
        System.out.println(v3);  
    }  
}
```

## 2.6、Random

Random 位于 java.util 包下，可以产生随机数

### 2.6.1、生成 5 个 0~100 之间的整数随机数

```
import java.util.Random;  
  
public class RandomTest01 {  
  
    public static void main(String[] args) throws Exception{  
        Random r = new Random();  
        for (int i=0; i<5; i++) {  
            System.out.println(r.nextInt(100));  
        }  
    }  
}
```

## 2.7、Enum

### 2.7.1、为什么使用枚举

//以下返回 1 或 0 存在问题  
//在编译器就容易把程序错了，如：1 和 111 没有什么区别，编译器认为两者是一样的  
//不会报错，错误发现的越早越好，最好在编译器把所有的错误都消除掉

```
public class EnumTest01 {  
  
    public static void main(String[] args) throws Exception{  
        int ret = method1(10, 2);  
        if (ret == 1) {  
            System.out.println("成功！");  
        }  
        if (ret == 0) {  
            System.out.println("失败！");  
        }  
    }  
}
```

```
}

//正确返回 1，失败返回：0
private static int method1(int value1, int value2) {
    try {
        int v = value1/value2;
        return 1;
    } catch (Exception e) {
        return 0;
    }
}
```

## 2.7.1、改进示例一

//此种方式比第一种方案好一些  
//有一个统一的约定，成功用 1 表示，失败采用 0 标识  
//但是也存在问题，如果不准许约定也会产生问题  
//如果成功我们可以返回 SUCCESS,但也可以返回 100，因为返回值为 int，  
//并没有强制约束要返回 1 或 0

```
public class EnumTest02 {

    private static final int SUCCESS = 1;

    private static final int FAILURE = 0;

    public static void main(String[] args) throws Exception {
        int ret = method1(10, 2);
        if (ret == SUCCESS) {
            System.out.println("成功！");
        }
        if (ret == FAILURE) {
            System.out.println("失败！");
        }
    }
}
```

```
//正确返回 1，失败返回：0
private static int method1(int value1, int value2) {
    try {
        int v = value1/value2;
        return SUCCESS;
    } catch (Exception e) {
        return FAILURE;
    }
}
```

```
        return FAILURE;
    }
}
}
```

### 2.7.1、采用枚举改进

//使用枚举类型，能够限定取值的范围  
//使程序在编译时就会及早的返现错误  
//这样程序会更加健壮

```
public class EnumTest03 {

    public static void main(String[] args) throws Exception{
        Result r = method1(10, 2);
        if (r == Result.SUCCESS) {
            System.out.println("成功！");
        }
        if (r == Result.FAILURE) {
            System.out.println("失败！");
        }
    }

    //正确返回 SUCCESS，失败返回：FAILURE
    private static Result method1(int value1, int value2) {
        try {
            int v = value1/value2;
            return Result.SUCCESS;
        } catch (Exception e) {
            return Result.FAILURE;
        }
    }
}

enum Result {
    SUCCESS,FAILURE
}
```

### 2.8、补充 public static void main(String[] args)，数组如何传值

```
public class MainTest01 {
```

```
public static void main(String[] args){
    for (int i=0; i<args.length; i++) {
        System.out.println(args[i]);
    }
    if (args.length !=0) {
        if (args[0].equals("A")) {
            System.out.println("添加用户");
        } else if (args[0].equals("U")) {
            System.out.println("更新用户");
        }
    } else {
        System.out.println("请输入参数 A:添加, U: 修改");
    }
}
```

java MainTest01 A