# String类与内存

## 相关内存概念

1. 字符串常量池（编译期）

JVM为了减少字符串对象的重复创建，其维护了一个特殊的内存，这段内存被成为字符串常量池或者字符串字面量池。

1. 工作原理

字符串常量池实现的前提条件就是Java中String对象是不可变的，这样可以安全保证多个变量共享同一个对象。如果Java中的String对象可变的话，一个引用操作改变了对象的值，那么其他的变量也会受到影响，显然这样是不合理的。

1. 堆栈的概念（运行期）

Java中所有由类实例化的对象和数组都存放在堆内存中，无论是成员变量，局部变量，还是类变量，它们指向的对象都存储在堆内存中。而栈内存用来存储局部变量和方法调用。

1. 寄存器概念

Java中运行时数据区有一个程序寄存器（又称程序计数器），该寄存器为线程私有。Java中的程序计数器用来记录当前线程中正在执行的指令。如果当前正在执行的方法是本地方法，那么此刻程序计数器的值为undefined。

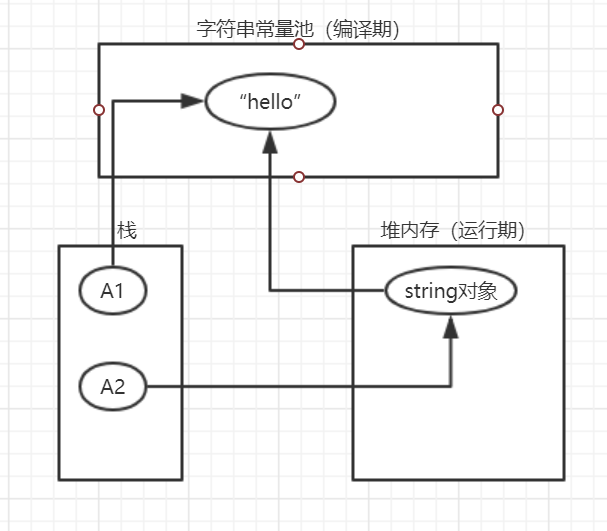
## 对象的生成

* 1. new String(“hello”) 生成了几个对象？两个

在编译期，“heelo”在字符串常量池生成了一个string对象，假设引用为A1。

在运行期，在堆中生成了一个string对象，再将指针指向了“hello”，假设引用为A2。

此时A1 不等于 A2



2.2 代码实例

String str1 = "hello"; //str1直接指向“hello”

String str2 = “he” + “llo” //此时str2 也是指向“hello”

String str3="he"+ new String("llo"); //当String str3 = “he”+”llo”时，判断为true

System.out.println(str1==str3); //判断为false

String sg1 = “he”

String sg2 = “llo”

System.out.println((sg1+sg2) == str1) //判断为false

执行过程：

1. 在编译期，在常量池中生成“hello”，并将引用地址直接赋值给str1（在栈中的地址值）
2. 在编译期，在常量池中生成了“he”和“llo”两个string，执行一下和为“hello”，再将引用地址赋值给str2（所以str1和str2在栈中的地址值是相等的）
3. 在运行期，在堆中生成一个string对象，并将该对象的地址赋值给str3。该对象中也有一个指针值，指向常量池“llo”。执行一下，该对象中的指针值变为指向“hello”，所以str3最终指向了“hello”，但是str3本身的地址值和str1 str2是不相等的。
4. 在编译期，sg1指向“he”，sg2指向“llo”
5. 在运行期，sg1+sg2生成一个新对象在堆中，sg1+sg2指向堆中的对象，对象中的值再指向常量池的“hello”

**总结：在编译期不生成堆中对象，只会进行常量池中的操作。编译期中没有解决的运算会放到运行期中解决，直接生成对象。**

# 二、值传递与引用传递

public class SendValue{

public String str="6";

public static void main(String[] args) {

SendValue sv=new SendValue();

sv.change(sv.str);

System.out.println(sv.str);

}

public void change(String str) {

str="10";

}

} 输出：6

1. 值传递和引用传递的区别
2. 值传递：(形式参数类型是基本数据类型)：方法调用时，实际参数把它的值传递给对应的形式参数，**形式参数只是用实际参数的值初始化自己的存储单元内容**，是两个不同的存储单元，所以方法执行中形式参数值的改变不影响实际参数的值。

**Stirng也可以当做基本类型，方法内部本身有自己的存储单元。**

1. 引用传递：(形式参数类型是引用数据类型参数)：也称为**传地址**。方法调用时，实际参数是对象(或数组)，这时实际参数与形式参数指向同一个地址，在方法执行中，对形式参数的操作实际上就是对实际参数的操作，这个结果在方法结束后被保留了下来，所以方法执行中形式参数的改变将会影响实际参数。

**最形象的是传入数组的引用，在方法内部改变数组的值，数组的实际值也会被改变。**

# String类的常用方法

1. 构造方法 – 改变编码方式

public String(byte[] bytes, Charset charset);//将某个字符数组转换成特定编码

实例：new String("ISO8959-1".getBytes("ISO8959-1")，“GB2312”)

1. 与byte字节数组相关

(1) String(byte[] bytes, String charsetName)

通过使用指定的 charset 解码指定的 byte 数组，构造一个新的String。

(2) getBytes(String charsetName)

使用指定的字符集将此String编码为 byte 序列，并将结果存储到一个新的 byte 数组中。

1. 子字符串

(1) public String[] split(String str) 按照str来分割字符串

比如： str = “12,32,23,232”

String[] strArr = str.split(“,”);

(2) public String substring(int beginIndex)

返回一个子字符串，起始位置为index

1. public String substring(int beginIndex , int endIndex)

返回一个子字符串，起始位置为beginIndex，长度为（endIndex - beginIndex）

4. replace方法

public static void main (String[] args) {

String classFile = "com.jd.". replaceAll(".", "/") + "MyClass.class";

System.out.println(classFile);

}

(1) 打印结果：///////MyClass.class

(2) 原理：由于replaceAll方法的第一个参数是一个正则表达式，而"."在正则表达式中表示任何字符，所以会把前面字符串的所有字符都替换成"/"。如果想替换的只是"."，那么久要写成\\.

(3) API：public String replaceAll(String regex，String replacement)

原理：replaceAll当中第一个参数是正则表达式

5. 其他

代码String str=”123456a”；int i=Integer.parseInt(str);会报异常的是（）

java.lang.NumberFormatException

**非纯数字的字符串转化为Integer对象会报数字格式异常**。

# 四、String、StringBuffer、StringBuilder的区别

java中String、StringBuffer、StringBuilder是编程中经常使用的字符串类。

1. 可变与不可变

(1) String类中使用字符数组保存字符串，如下就是，因为有“final”修饰符，所以可以知道**string对象是不可变的**。

private final char value[];

String 为不可变对象,一旦被创建,就不能修改它的值。对于已经存在的String对象的修改都是重新创建一个新的对象,然后把新的值保存进去.

(2) StringBuilder与StringBuffer都继承自AbstractStringBuilder类，在AbstractStringBuilder中也是**使用字符数组保存字符串**，如下就是，可知**这两种对象都是可变的**。

char[] value;

StringBuffer:是一个可变对象,当对他进行修改的时候不会像String那样重新建立对象 , 它只能通过构造函数来建立, 如： StringBuffer sb = new StringBuffer();

不能通过赋值符号对他进行付值. ， 如 sb = "welcome to here!";//error

对象被建立以后,在内存中就会分配内存空间,并初始保存一个null.向StringBuffer中赋值的时候可以通过它的append方法. sb.append("hello");

2. 是否多线程安全

**(1) String中的对象是不可变的，也就可以理解为常量， 显然线程安全** 。

AbstractStringBuilder是StringBuilder与StringBuffer的公共父类，定义了一些字符串的基本操作，如expandCapacity、append、insert、indexOf等公共方法。

(2) **StringBuffer对方法加了同步锁或者对调用的方法加了同步锁，所以是 线程安全的** 。看如下源码：

public synchronized StringBuffer reverse() {

super .reverse();

return this ;

}

public int indexOf(String str) {

//存在 public synchronized int indexOf(String str, int fromIndex) 方法

return indexOf(str, 0);

}

(2) **StringBuilder并没有对方法进行加同步锁，所以是 非线程安全的** 。

3. StringBuilder与StringBuffer共同点

(1) StringBuilder与StringBuffer有公共父类AbstractStringBuilder( 抽象类 )。

(2) 抽象类与接口的其中一个区别是：抽象类中可以定义一些子类的公共方法，子类只需要增加新的功能，不需要重复写已经存在的方法；而接口中只是对方法的申明和常量的定义。

(3) StringBuilder、StringBuffer的方法都会调用AbstractStringBuilder中的公共方法，如super.append(...)。只是StringBuffer会在方法上加synchronized关键字，进行同步。

(4) 最后，如果程序不是多线程的，那么使用StringBuilder效率高于StringBuffer。

效率比较String < StringBuffer < StringBuilder，但是在String S1 =“This is only a”+“simple”+“test”时，String效率最高。（因为直接在编译期就就执行成功了）