# String

## 实现接口

### Java.io.Serialzable

这个序列号接口没有任何方法和域，仅用于标识序列化的语意

### Comparable<String>

这个接口只有一个方法compareTo（T o），用于比较两个实例对象的大小

### CharSequence

这个接口是一个只读的字符序列。包括length(), charAt(int index), subSequence(int start, int end)这几个API接口，值得一提的是，StringBuffer和StringBuild也是实现了改接口。

## 成员变量

### Private final char value[]

这是一个字符数组，并且是final类型，他用于存储字符串内容, 从fianl这个关键字中我们可以看出，String的内容一旦被初始化了是不能被更改的

### Private int hash;

缓存字符串的hash Code，默认值为 0；

### Public static final Comparator<String> CASE\_INSENSITIVE\_ORDER = CaseInsesitiveComparator()

这是String静态内部类对象，用于在比较两个字符串的时候忽略大小写

### 序列化id，serialVersionUID

因为 String 实现了 Serializable 接口，所以支持序列化和反序列化支持。Java 的序列化机制是通过在运行时判断类的 serialVersionUID 来验证版本一致性的。在进行反序列化时，JVM 会把传来的字节流中的 serialVersionUID 与本地相应实体（类）的 serialVersionUID 进行比较，如果相同就认为是一致的，可以进行反序列化，否则就会出现序列化版本不一致的异常 (InvalidCastException)。

## 构造方法

### New String()

默认构造器，this.value = new char[0]，char数组长度为0，实际就是空字符串

### New String(String original)

This.value = original.value;this.hash=original.hash由于字符串和里面的char数组是不可变的，这个构造函数实际上就是将某个字符串的引用指向另一个字符串。

### New String(char value[])

This.value = Arrays.copyof(value,value.length);将字符数组复制一份并复制给对象的value。复制是为了避免外界改变value从而改变此String对象。

### New String(byte[] bytes, charset charset)

使用字节数组和指定编码集构造字符串对象

### New String(StringBuilder sb)/new String(StringBulder sb)

synchronized(buffer) {

this.value = Arrays.copyOf(buffer.getValue(), buffer.length());

}

其中第一个构造方法需要通过synchronized关键字同步，然后将其value复制。

### String(char[] value, Boolean share)

this.value = value;

这是一个default修饰的构造方法，所以只能在同包下的其他类中使用，主要是Integer等包装类。

## 常用方法

**String replace(char oldChar, char newChar)**

**String concat(String str)**

**substring**

**trim()**

**String[] split(String regex)**

**intern()**

**length()**

**charAt(int index)**

**toLowerCase()/toUpperCase()**

**boolean matches(String regex)**

**boolean contains(CharSequence s)**

**boolean contentEquals(Char Sequencecs)**

## 总结

### 字符串String为什么是不可变对象

String类用final修饰的，所以它不能被继承；成员变量字符数组也是被final修饰的，所以成员变量所指向的地址是不能改变的；同时该成员变量是private，外部无法无法访问该数组，也就无法修改这个数组；另外通过字符数组构造的String对象的构造器中是拷贝一份新的数组而不是直接引用该数组；基于以上原因，字符串是不可变对象。

### String为什么要设计成不可变对象

安全性，因为String对象是不可变的，所以不用担心改变引用对象的值，而造成的多线程安全问题，或者hashmap键值对等问题。

高效性，因为String对象是不可变的，变量所指向的地址中的值是不会改变的，所以可以用作常量池， 当大量使用字符串特别是字符串内容一样的情况下，不需要单独new一个字符串对象，而是共同指向常量池中同一个地址，这样就可以节省内存和初始化对象的开销，提高性能。

### String字符串+连接

(1).String中使用 + 字符串连接符进行字符串连接时，连接操作最开始时如果都是字符串常量，编译后将尽可能多的直接将字符串常量连接起来，形成新的字符串常量参与后续连接（通过反编译工具jd-gui也可以方便的直接看出）；

(2).接下来的字符串连接是从左向右依次进行，对于不同的字符串，首先以最左边的字符串为参数创建StringBuilder对象，然后依次对右边进行append操作，最后将StringBuilder对象通过toString()方法转换成String对象（注意：中间的多个字符串常量不会自动拼接）。

也就是说String c = "xx" + "yy " + a + "zz" + "mm" + b; 实质上的实现过程是： String c = new StringBuilder("xxyy ").append(a).append("zz").append("mm").append(b).toString();

（3）。对于两个常量字符串相加，编译器会自动优化成一个字符串常量，所以常量池中就只会存在合并后的字符串，如：

String s = "zz"+"mm"，此时常量池中有zzmm字符串，并没有zz和mm字符串**。**

（4）.对应两个字符串变量相加，相加的变量指向的是堆内存地址而不是常量池中的地址，如：

String a = “a”;

String b = “b”;

String ab = a + b;

此时ab指向的是堆内存中的地址。

# Hashmap

Hashmap底层结构是数组+链表形式，此外链表长度如果超过8会自动转换成红黑树。

## 构造方法

### public HashMap(int initialCapacity, float loadFactor)

传入初始容量和负载因子构造一个HashMap对象，初始容量会经过一个方法转换成超过该初始容量的一个最小的2的幂次方的数（如果这个数超过2的30次方，则使用2的30次方）并赋值给成员变量threshold。

其他构造函数类似，只不过使用默认值赋值给相应的成员变量，比如初始容量阈值默认为16，负载因子默认为0.75.

## 成员变量

**threshold** 阈值，默认值为16，如果构造方法中有传初始容量，则会将大于等于这容量的最小2次幂赋值给该变量。

为什么值要设置为2的幂次方？

因为如果不是2的幂次方，则数组长度-1的值换算成二进制后，一定有为0的位，而map在put的时候需要通过key的hashcode与该值进行求与计算对应数组索引，这样无论key的hashcode值为多少，这些为0的位永远都是0，也就是说永远也无法得到这些位置为1的索引。这些索引对应位置就永远不会存放元素，浪费空间的同时加大了碰撞几率。

**loadFactor** 负载因子，默认为0.75，当map容量超过负载因子和阈值的积，会进行重散列。负载因子0.75是对空间和时间效率的一个平衡选择，建议大家不要修改，除非在时间和空间比较特殊的情况下，如果内存空间很多而又对时间效率要求很高，可以降低负载因子Load factor的值；相反，如果内存空间紧张而对时间效率要求不高，可以增加负载因子loadFactor的值，这个值可以大于1。

**transient Node<K,V>[] table**

**transient int modCount;**

标志位，用于标识并发问题，主要用于迭代的快速失败

**transient int size;**

## 内部类

### static class Node<K,V> implements Map.Entry<K,V>

## 添加元素

**put方法**