public final class **String**

extends [Object](mk:@MSITStore:F:\my\mygit\enhancingProgram\JAVA\%5bJava参考文档%5dJDK_API_1_6_zh_CN.CHM::/java/lang/Object.html)

implements [Serializable](mk:@MSITStore:F:\my\mygit\enhancingProgram\JAVA\%5bJava参考文档%5dJDK_API_1_6_zh_CN.CHM::/java/io/Serializable.html), [Comparable](mk:@MSITStore:F:\my\mygit\enhancingProgram\JAVA\%5bJava参考文档%5dJDK_API_1_6_zh_CN.CHM::/java/lang/Comparable.html)<[String](mk:@MSITStore:F:\my\mygit\enhancingProgram\JAVA\%5bJava参考文档%5dJDK_API_1_6_zh_CN.CHM::/java/lang/String.html)>, [CharSequence](mk:@MSITStore:F:\my\mygit\enhancingProgram\JAVA\%5bJava参考文档%5dJDK_API_1_6_zh_CN.CHM::/java/lang/CharSequence.html)

1. final修饰的String类为最终类，不能被继承
2. String类实现了序列化接口，可比较接口和[CharSequence](mk:@MSITStore:F:\my\mygit\enhancingProgram\JAVA\%5bJava参考文档%5dJDK_API_1_6_zh_CN.CHM::/java/lang/CharSequence.html)接口
3. Java中所有的字符串字面值都是这个类的实例，如声明：

String str="abc" #则str就是一个string类的实例

上面的定义方法等效于：

char[] charArr={'a', 'b', 'c'}

String str2=new String(charArr)

1. 字符串是常量，他们的值在创建后不能修改
2. 字符串转换时通过toString方法实现的，该方法是由Object定义的，因此java中的所有类都可以继承并重写此方法

* 字段

|  |  |
| --- | --- |
| 字段类型 名定义 | 说明 |
| private final char value[]; | 真正存储字符串各个字符的类，字符串的底层实现为字符数组 |
| private int hash; |  |
| private static final long serialVersionUID = -6849794470754667710L; | 固定值：-6849794470754667710L |
| private static final ObjectStreamField[] serialPersistentFields = new ObjectStreamField[0]; |  |

方法、参数和说明以及示例

public String()

public String() {

this.value="".value();

}

#创建了一个新的对象

public String(String original)

public String(String original){

this.value=original.value; #这些private变量可以在类的内部被访问

this.hash=original.hash;

}

#创建了一个新的对象，处理了String的核心value和hash

public String(char[] value)

public String(char value[]) {

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

}

将字符数组的值赋值给字符串的底层char[],由此可以看出字符串的底层实现是一个字符数组

public String(char value[], int offset, int count)

拷贝字符数组的指定部分，初始化String对象

offset+count在数组范围之内

public String(char value[], int offset, int count) {

if (offset < 0) {

throw new StringIndexOutOfBoundsException(offset);

}

if (count <= 0) {

if (count < 0) {

throw new StringIndexOutOfBoundsException(count);

}

if (offset <= value.length) {

this.value = "".value;

return;

}

}

// Note: offset or count might be near -1>>>1.

if (offset > value.length - count) {

throw new StringIndexOutOfBoundsException(offset + count);

}

this.value = Arrays.copyOfRange(value, offset, offset+count);

}

public String(int[] codePoints, int offset, int count)

通过数字数组的指定部分构建字符串对象

将指定数组转换为它的字符表示

最后新建的字符串对象字符的长度可能大于count，因为某些数字可能对应多个字符

public String(int[] codePoints, int offset, int count) {

if (offset < 0) {

throw new StringIndexOutOfBoundsException(offset);

}

if (count <= 0) {

if (count < 0) {

throw new StringIndexOutOfBoundsException(count);

}

if (offset <= codePoints.length) {

this.value = "".value;

return;

}

}

// Note: offset or count might be near -1>>>1.

if (offset > codePoints.length - count) {

throw new StringIndexOutOfBoundsException(offset + count);

}

final int end = offset + count;

// Pass 1: Compute precise size of char[]

int n = count;

for (int i = offset; i < end; i++) {

int c = codePoints[i];

if (Character.isBmpCodePoint(c))

continue;

else if (Character.isValidCodePoint(c))

n++;

else throw new IllegalArgumentException(Integer.toString(c));

}

// Pass 2: Allocate and fill in char[]

final char[] v = new char[n];

for (int i = offset, j = 0; i < end; i++, j++) {

int c = codePoints[i];

if (Character.isBmpCodePoint(c))

v[j] = (char)c;

else

Character.toSurrogates(c, v, j++);

}

this.value = v;

}

private static void checkBounds(byte[] bytes, int offset, int length)

内部在由字节数组新建字符串对象是调用，判断字节数组的范围是否越界

其实在将字符数组和数字数组转换为字符串对象的时候在代码的前一部分也有范围验证，也是代码重复，也应该合并到一起

private static void checkBounds(byte[] bytes, int offset, int length) {

if (length < 0)

throw new StringIndexOutOfBoundsException(length);

if (offset < 0)

throw new StringIndexOutOfBoundsException(offset);

if (offset > bytes.length - length)

throw new StringIndexOutOfBoundsException(offset + length);

}

public String(byte bytes[], int offset, int length, String charsetName)

由字节数组构建新的字符串对象，调用了上面的checkBounds方法检查字节数组的范围

传入的第三个参数为字符编码的名字

public String(byte bytes[], int offset, int length, String charsetName)

throws UnsupportedEncodingException {

if (charsetName == null)

throw new NullPointerException("charsetName");

checkBounds(bytes, offset, length);

this.value = StringCoding.decode(charsetName, bytes, offset, length);

}

public String(byte bytes[], int offset, int length, Charset charset)

由字节数组构建新的字符串对象，与上面的方法几乎一样，只是传递的第三参数是字符编码对象

public String(byte bytes[], int offset, int length, Charset charset) {

if (charset == null)

throw new NullPointerException("charset");

checkBounds(bytes, offset, length);

this.value = StringCoding.decode(charset, bytes, offset, length);

}

public String(byte bytes[], String charsetName)

由字节数组构建新的字符串对象，是上面的方法的特殊形式，拷贝到是整个字节数组

public String(byte bytes[], String charsetName)

throws UnsupportedEncodingException {

this(bytes, 0, bytes.length, charsetName);

}

public String(byte bytes[], Charset charset)

由字节数组构建新的字符串对象，是上面几种方法的特殊形式，只是传递的参数为字符编码对象

public String(byte bytes[], Charset charset) {

this(bytes, 0, bytes.length, charset);

}

public String(byte bytes[], int offset, int length)

由字节数组构建字符串对象

public String(byte bytes[], int offset, int length) {

checkBounds(bytes, offset, length);

this.value = StringCoding.decode(bytes, offset, length);

}

public String(byte bytes[])

public String(byte bytes[]) {

this(bytes, 0, bytes.length);

}

public String(StringBuffer buffer)

由StringBuffer构建新的字符串对象

正是因为使用了synchronized关键字所有才是线程安全的

public String(StringBuffer buffer) {

synchronized(buffer) {

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

}

}

public String(StringBuilder builder)

由StringBuilder构建新的字符串对象

因为没有使用synchronized关键字，并且可拷贝的操作是非原子操作，所以是非线程安全的

public String(StringBuilder builder) {

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

}

String(char[] value, boolean share)

由字符数组构建新的字符串对象，貌似好像很少使用

这个构造方法是包内方法

String(char[] value, boolean share) {

// assert share : "unshared not supported";

this.value = value;

}

字符串类自身的非构造方法：

public int length()

public int length() {

return value.length;

}