

包装类:

针对八种基本数据类型相应的引用类型—包装类，有了类的特点，就可以调用类中的方法。

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

1. jdk5 前是手动装箱和拆箱，手动装箱 `int->Integer`。
2. jdk5 后，就可以自动装箱和自动拆箱。
3. 自动装箱底层调用 `valueOf` 方法

String

1. String 对象用于保存字符串，也就是一组字符序列。
2. "jack" 字符串常量，双引号括起的字符序列。
3. 字符串的字符使用 Unicode 字符编码，一个字符(不区分字母还是汉字)占两个字节。
4. String 类有很多构造器，构造器的重载。

常用的有 `String s1 = new String(); //`

`String s2 = new String(String original);`

`String s3 = new String(char[] a);`

`String s4 = new String(char[] a, int startIndex, int count)`

`String s5 = new String(byte[] b)`

5. String 类实现了接口 `Serializable` 【String 可以串行化:可以在网络传输】 接口 `Comparable` [String 对象可以比较大小]。
6. String 是 `final` 类，不能被其他的类继承。
7. String 有属性 `private final char value[]`; 用于存放字符串内容。
8. 一定要注意: `value` 是一个 `final` 类型， 不可以修改(需要功力): 即 `value` 不能指

向新的地址，但是单个字符内容是可以变化。

创造 String 对象的两种方式

1) 方式一：直接赋值 String s = "hspedu";
2) 方式二：调用构造器 String s = new String("hspedu");

方式一：直接赋值 String s = "hsp";
方式二：调用构造器 String s2 = new String("hsp");

- 1. 方式一：先从常量池查看是否有"hsp" 数据空间，如果有，直接指向；如果没有则重新创建，然后指向。s最终指向的是常量池的空间地址**
- 2. 方式二：先在堆中创建空间，里面维护了value属性，指向常量池的hsp空间。如果常量池没有"hsp"，重新创建，如果有，直接通过value指向。最终指向的是堆中的空间地址。**
- 3. 画出两种方式的内存分布图**

1. String 是一个 final 类，代表不可变的字符序列。
2. 字符串是不可变的，一个字符串对象一旦被分配，其内容是不可变的。

String 的常用方法：

1. equals 比较内容是否相同，区分大小写。
2. equalsIgnoreCase 忽略大小写的判断内容是否相等。
3. length 获取字符的个数，字符串的长度。
4. indexOf 获取字符在字符串对象中第一次出现的索引，索引从 0 开始，如果找不到，返回-1。
5. lastIndexOf 获取字符在字符串中最后一次出现的索引，索引从 0 开始，如果找不到，返回-1。
6. substring 截取指定范围的子串。
7. toUpperCase 转换成大写。
8. toLowerCase 转换成小写。
9. concat 拼接字符串。
10. replace 替换字符串中的字符。
11. split 分割字符串，对于某些分割字符，我们需要 转义比如 | [\\](#)等。
12. toCharArray 转换成字符数组。
13. compareTo 比较两个字符串的大小，如果前者大，则返回正数，后者大，则返回负数，如果相等，返回 0。

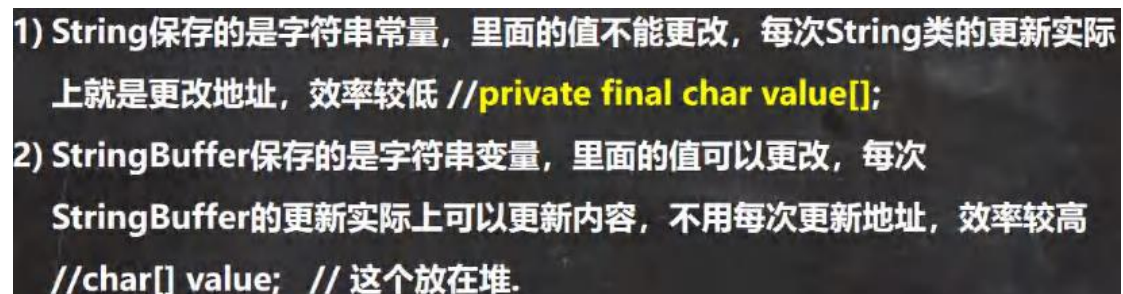
StringBuffer 类

代表可变的字符序列，可以对字符串内容进行增删。

很多方法与 String 相同，但是 StringBuffer 是可变长度的。

1. StringBuffer 的直接父类 是 AbstractStringBuilder。
2. StringBuffer 实现了 Serializable, 即 StringBuffer 的对象可以串行化。
3. 在父类中 AbstractStringBuilder 有属性 char[] value, 不是 final, 该 value 数组存放 字符串内容，引出存放在堆中的。
4. StringBuffer 是一个 final 类，不能被继承。
5. 因为 StringBuffer 字符内容是存在 char[] value, 所有在变化(增加/删除)，不用每次都更换地址(即不是每次创建新对象)， 所以效率高于 String。

String 与 StringBuffer



1) String保存的是字符串常量，里面的值不能更改，每次String类的更新实际上就是更改地址，效率较低 `//private final char value[];`

2) StringBuffer保存的是字符串变量，里面的值可以更改，每次StringBuffer的更新实际上可以更新内容，不用每次更新地址，效率较高 `//char[] value; // 这个放在堆.`

String→StringBuffer

方式 1 使用构造器

注意： 返回的才是 StringBuffer 对象，对 str 本身没有影响

```
StringBuffer stringBuffer = new StringBuffer(str);
```

方式 2 使用的是 append 方法

StringBuffer →String

方式 1 使用 StringBuffer 提供的 toString 方法

方式 2: 使用构造器

StringBuffer 常见方法:

增: append

删: delete

改: replace

查: indexOf

插: insert

长度: length

StringBulider 类

StringBuilder01.java

1) 一个可变的字符序列。此类提供一个与 StringBuffer 兼容的 API, 但不保证同步(StringBuilder 不是线程安全)。该类被设计用作 StringBuffer 的一个简易替换, 用在字符串缓冲区被单个线程使用的时候。如果可能, 建议优先采用该类, 因为在大多数实现中, 它比 StringBuffer 要快 [后面测]。

2) 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。

1. StringBuilder 继承 AbstractStringBuilder 类
2. 实现了 Serializable, 说明 StringBuilder 对象是可以串行化(对象可以网络传输, 可以保存到文件)
3. StringBuilder 是 final 类, 不能被继承
4. StringBuilder 对象字符序列仍然是存放在其父类 AbstractStringBuilder 的 char[] value; 因此, 字符序列是堆中
5. StringBuilder 的方法, 没有做互斥的处理, 即没有 synchronized 关键字, 因此在单线程的情况下使用常用方法与 StringBuffer 相同

1) StringBuilder 和 StringBuffer 非常类似, 均代表可变的字符序列, 而且方法也一样

2) String: 不可变字符序列, 效率低, 但是复用率高。

3) StringBuffer: 可变字符序列、效率较高(增删)、线程安全, 看源码

4) StringBuilder: 可变字符序列、效率最高、线程不安全

Math 类

1. abs 绝对值
2. pow 求幂
3. ceil 向上取整, 返回>=该参数的最小整数(转成 double)
4. floor 向下取整, 返回<=该参数的最大整数(转成 double)
5. round 四舍五入
6. sqrt 求开方
7. random 求随机数
8. max, min 返回最大值和最小值

Arrays 类

ArraysMethod01.java

Arrays里面包含了一系列静态方法，用于管理或操作数组(比如排序和搜索)。

1) toString 返回数组的字符串形式

Arrays.toString(arr)

2) sort 排序（自然排序和定制排序） Integer arr[] = {1, -1, 7, 0, 89};

ArraysSortCustom.java

ArraysMethod02.java

3) binarySearch 通过二分搜索法进行查找，要求必须排好序

int index = Arrays.binarySearch(arr, 3);

ArraysMethod02.java

4) copyOf 数组元素的复制

Integer[] newArr = Arrays.copyOf(arr, arr.length);

5) fill 数组元素的填充

Integer[] num = new Integer[]{9,3,2};

Arrays.fill(num, 99);

6) equals 比较两个数组元素内容是否完全一致

boolean equals = Arrays.equals(arr, arr2);

7) asList 将一组值，转换成list

List<Integer> asList = Arrays.asList(2,3,4,5,6,1);

System.out.println("asList=" + asList);

System 类

1. exit(0) 表示程序退出，0 表示一个状态，正常的状态。

2. arraycopy：复制数组元素，比较适合底层调用

3. currentTimeMillens:返回当前时间距离 1970-1-1 的毫秒数

BigInteger 和 BigDecimal 类

应用场景：

1) BigInteger适合保存比较大的整型

2) BigDecimal适合保存精度更高的浮点型（小数）

常见方法：

加：add

减: subtract

乘: multiply

除: divide

BigDecimal 类在调用 divide 方法时, 指定精度. BigDecimal.ROUND_CEILING, 如果有无限循环小数, 就会保留 分子 的精度

日期类:

第一代日期类

Date, 精确到毫秒

```
Date d1 = new Date();
```

1. 获取当前系统时间
2. 这里的 Date 类是在 java.util 包
3. 默认输出的日期格式是国外的方式, 因此通常需要对格式进行转换

格式转换:

1. 创建 SimpleDateFormat 对象, 可以指定相应的格式
2. 这里的格式使用的字母是规定好, 不能乱写

```
SimpleDateFormat sdf = new SimpleDateFormat("yyyy 年 MM 月 dd 日 hh:mm:ss E");
```

第二代日期类

Calender

1. Calendar 是一个抽象类, 并且构造器是 private

```
Calendar c = Calendar.getInstance();
```

2. 可以通过 getInstance() 来获取实例
3. 提供大量的方法和字段提供给程序员
4. Calendar 没有提供对应的格式化的类, 因此需要程序员自己组合来输出(灵活)
5. 如果我们需要按照 24 小时进制来获取时间, Calendar.HOUR == 改成=>

```
Calendar.HOUR_OF_DAY
```

第三代日期类

1) `LocalDate`(日期/年月日)、`LocalTime`(时间/时分秒)、`LocalDateTime`(日期时间/年月日时分秒) JDK8加入

`LocalDate`只包含日期，可以获取日期字段

`LocalTime`只包含时间，可以获取时间字段

`LocalDateTime`包含日期+时间，可以获取日期和时间字段