

Java - P17

Standard

5.9 - Primitives

Number.XXXValue(), 用于将 Number 对象转换为 XXX 数据类型并返回

如 byteValue() :: Number → byte, (short)

intValue() :: Number → abstract int (long / float / double)

XXX.valueOf(), 用于将传入的参数转换为指定的 XXX 数据类型

如对于 Integer.valueOf(), 返回类型为 Integer

其定义的不同参数类型传入方式有 内置

static Integer valueOf(int i) : int 类型 int 的值

static Integer valueOf(String s) : 解析用的字符串 s

static Integer valueOf(String s, int radix) : 用于解析的基数 radix

如 Integer.valueOf("11", 16) → 17

parseXXX(), 用于将传入的字符串描述为指定的 XXX 数据类型

与 valueOf() 的定义类似, 不同的是只接受字符串 String

另外 valueOf() 返回的是指定类型的包装类

而 parseXXX() 返回的是指定类型的内置类型

如 static double parseDouble(String s)

XXX.toString(), 用于将 XXX 数据类型的对象转换为字符串表示

如 String Double.toString()

random(), 返回一个 [0, 1] 上的随机数, 返回类型为 double

定义为 static double Math.random()

ceil(), 返回 $\lceil a \rceil$, 定义为 static double Math.ceil(double a)

floor(), 返回 $\lfloor a \rfloor$, 定义为 static double Math.floor(double a)

round(), 返回 a 的四舍五入的整数, 定义为 static int Math.round(float a)

static long Math.round(double a)

其算法为 Math.round(a) \leftrightarrow Math.floor(a + 0.5)

	-1.6	-1.5	-1.4	1.4	1.5	1.6
ceil()	-1	-1	-1	2	2	2
floor()	-2	-2	-2	1	1	1
round()	-2	-1	-1	2	2	2

Java - P18

Character类，在Java中，与Number类似，为内置数据类型char提供包装类Character

Character类在对象中包装一个内置类型char的值，用于对单个字符操作

部分情况下Java编译器会自动创建一个Character对象

如对需要Character类型参数的方法传入一个char类型的参数

装箱：内置数据类型 → 包装类

拆箱：包装类 → 内置数据类型

Character类提供了一系列方法来操作字符

static Character valueOf(char c)

用于代替new Character(char c)作为构造Character类对象的方式

static boolean isDigit / isLetter (char c)

判断是否为数字/字母

static boolean isWhitespace (char c)

判断是否为空白字符

static boolean isLowerCase / isUpperCase (char c)

判断是否为小写/大写字母

static char toLowerCase / toUpperCase (char c)

将传入的字符串转换为小写/大写字符串

转义字符 (escape character)，在Java中，前缀为反斜杠(\)的字符表示转义字符

\t : 在该处插入一个tab键

\b : 在该处插入一个后退键(backspace)

\n : 在该处换行(newline, Line Feed)，向下移动一行，但不移动左右

\r : 在该处插入一个回车(carriage Return)，移动到当前行最左边

\f : 在该处插入换页符

\' : 在该处插入一个单引号

\\" : 在该处插入一个双引号

\\" : 在该处插入一个反斜杠

回车+换行 Linux中为\n；Windows中为\r\n，Mac中为\r

硬回车(Enter)：在换行同时也分隔段落，html中为<p>...</p>

软回车(shift+Enter)：换行但是不换段落，html中为

软回车不是法定段落标记，前后行间距大幅缩小

Java - P19

Java - Session 3

String 类

在 Java 中，提供了 String 类用于创建和操作字符串

在代码中遇到字符串常量时，编译器会使用该值创建一个 String 对象

如 `String foo = "foo";`

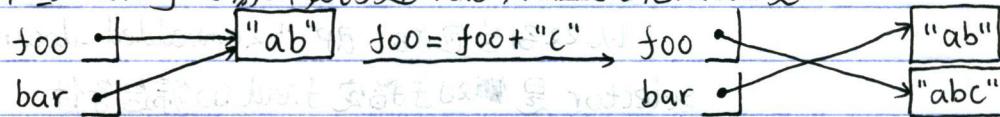
String 类也可以通过关键字和构造方法创建，在 Java 中提供了不同参数类型的构造

如 `String foo = new String("foo");`

注意与 String 在 Python 中是不可变对象类似

Java 中的 String 类是不可改变的

即当 String 对象被创建以后，其值就无法改变了



访问器方法 (accessor method)，在 Java 中用于获取对象信息的方法

如 `int length()` 用于返回 String 对象包含的字符数

注意，在 Java 中用于获取对象尺寸的 `length()` 方法，`length` 属性，`size()` 方法的不同

`length()` 针对 String 对象，是 String 类的访问器方法

如 `String foo = "foo"; foo.length() → 3`

`length` 属性针对 Java 中的数组，是数组对象的属性

如 `char[] foo = {'f', 'o', 'o'}; foo.length → 3`

`size()` 针对 Java 中的泛型集合，是泛型集合的访问器方法

如 `ArrayList<Character> foo = new ArrayList<Character>();`

`foo.add('f');` `foo.add('o');` `foo.add('o');`

`foo.size() → 3`

在 Java 中提供了两种连接字符串的方式

`String concat (String str)` 方法将传入的参数 str 连接在原字符串之后

使用加法运算符 (+) 连接字符串

如 `"foo" + "bar" → "foobar"`

在 Java 中提供了 String 类方法用于字符串比较，(大小敏感 / 忽略大小写)

`public int compareTo (String anotherString) / public int compareToIgnoreCase (String`

以字典序 (lexicographical) 比较两个字符串，返回一个整型值

正数表示 `this` 字典序靠后，负数表示 `this` 靠前，0 表示字符串相同

`public boolean equals (Object anObject) / public boolean equalsIgnoreCase (String`

判断两个字符串是否为相同的字符串序列 (same sequence of character)

Java - P20

String - equals()

String 类 在 Java 中，对于运算符(==)在 String 类的实现与 C++ 和 Python 不同
在 Java 中 String.equals(other) 与 C++ 和 Python 中的运算符(==)行为一致
但 Java 中的运算符(==)对于 String 类 是判断是否指向内存同一地址

```
String str1 = "abc";  
String str2 = "a" + "b" + "c";  
String str3 = new String("abc");  
String str4 = "ab"; str4 = str4 + "c";  
String str5 = new String("abc");  
str1 == str2 → true
```

在编译期间生成了字面常量和符号引用

在运行期间字面常量存储在运行时常量池

当通过字面常量绑定 String 对象时引用时

JVM 执行引擎会先在运行时常量池查找是否存在相同字面常量
如果存在，直接将引用指向已存在的字面常量

如果不存在，则在运行时常量池分配新空间存储字面常量，再使引用指向

于是 str1 已经在常量池中创建 "abc"，于是 str2 不需要再创建

```
str1 == str3 → false
```

通过 new 关键字生成的对象是在堆区(cheap)进行的

所以与常量池是不同的

```
str1 == str4 → false
```

str4 先创建 StringBuilder/StringBuffer 对象

通过 append() 方法连接得到 "abc"

最后调用 toString() 转换得到地址得到地址指向 str4

所以与常量池是不同的

```
str3 == str5 → false
```

注意在堆区进行对象生成的过程不会检测对象是否已经存在

即通过关键字 new 创建的对象一定是不同的对象

格式化字符串，在 Java 中，可以在 printf() 和 format() 方法中使用格式化字符串

基本语法为 System.out.printf(.., arg1, arg2, ...)

```
String str = String.format(.., arg1, arg2, ...)
```

静态方法 format() 返回一个 String 对象而非 PrintStream 对象

用于创建可复用的格式化字符串

Java - P21

829 - 1000

格式化字符串

在 Java 中，对传入参数为整型值的字符串格式化

基本语法为 `%[<index>$][<specifier>][<width>]<type>`

`<index>` 表示使用第 `<index>` 个传入的参数进行格式化

由于在 Java 中，对于 `System.out.println()` 和 `String.format()` 方法
传入的第一个参数 (`args[0]`) 为格式化的字符串

于是 `<index>` 从 1 开始编号

`<width>` 表示由整型值参数转化的字符串至少包含多少位字符

`<specifier>` 表示用于描述如何转化的标识符

`'0'`: 字符串之前的空白以 '0' 填充

`'-'`: 在最大宽度内左对齐，不可以与 '0' 同时使用

`'#'`: 只适用于八进制和十六进制

`'0'`: 在八进制前添加 "0"，在十六进制前添加 "0x"

`+' '`: 正值结果前添加 '+', 负值结果前添加 '-'，0 之前也添加 '+'

`'-'`: 非负值结果前添加 '0', 负值结果前添加 '-'

'+' 和 '-' 一般只适用于十进制，当对象为 BigInteger 可适用于八进制与十六进制

`'()'`: 非负值保持默认，负值结果以括号表示（与会计表示类似）

`', '`: 每 3 位数字之间以 ',' 分隔，只适用于十进制

`<type>` 表示整型值的进制转换方式

`'d'`: 以十进制表示，`'0'`: 以八进制表示

`'x' / 'X'`: 以十六进制表示，字母以小写 / 大写表示

在 Java 中，对传入参数为浮点数的字符串格式化

基本语法为 `%[<index>$][<specifier>][<width>][.<precision>]<type>`

`<precision>` 表示控制小数点后面向的位数

`<type>` 表示浮点数的转换方式

`'e' / 'E'`: 格式化为以计算机科学记数法的十进制表示

其中的字母以小写 / 大写表示

`'g' / 'G'`: 格式化为十进制普通表示方法

根据具体情况，自动选择用普通方式 / 科学计数法表示

`'a' / 'A'`: 格式化为带有效位数和指数的十六进制浮点数

`<specifier>` 与整型值的标识符类似

`, '`: 在浮点数中只适用于 `'e' / 'E' / 'f' / 'g' / 'G'`

`'()'`: 在浮点数中只适用于 `'e' / 'E' / 'f' / 'g' / 'G'`

String类

`char c = charAt (int index)`

返回指定索引index位置的字符

`byte[] getBytes ()`使用平台默认字符集将字符串转换为字节序列(sequence of bytes)
并返回一个新的byte数组`int indexOf (int ch) / int indexOf (int ch, int fromIndex)``int indexOf (String str) / int indexOf (String str, int fromIndex)``int lastIndexOf (int ch) / int lastIndexOf (int ch, int fromIndex)``int lastIndexOf (String str) / int lastIndexOf (String str, int fromIndex)`

返回指定字符/字符串在此字符串中第一次/最后一次出现位置的索引

根据传入的fromIndex，从指定索引处正向/反向搜索

`boolean matches (String regex)`

判断字符串是否匹配给定的正则表达式

`String replace (char oldChar, char newChar)`

返回用newChar替换此字符串中所有oldChar形成的新字符串

`String replaceAll (String regex, String replacement)`

使用replacement替换此字符串中所有匹配给定正则表达式的子字符串

并返回新形成的新字符串`String[] split (String regex)`

根据给定的正则表达式的匹配拆分此字符串

并返回生成的字符串数组

`String toLowerCase () / String toUpperCase ()`

使用默认语言环境的规则将此字符串中的所有字符转换为小写/大写

`String trim ()`：返回忽略此字符串所有前后和后缀空格的新字符串

Java - P23

String Builder类，在Java中，String类是不可变的(immutable)，
当需要修改字符串时，Java提供了StringBuffer类和StringBuilder类。

在Java中String类的内部封装了一个char数组，且由final修饰。

```
public final class String extends Object implements Comparable<String>, CharSequence {  
    private final char value[];  
    ...}
```

实际上String类的concat也是拼接字符串后重新创建一个新的String类对象。

与String Builder类相比，String Buffer类是线程安全的(Synchronized)。

但是注意String Buffer类的线程安全仅保证JVM不抛出异常地顺利执行
而不保证逻辑正确和调用顺序正确。

所以字符串的线程安全实际上需要的是锁的加锁和解锁。

并且相比String Builder类，String Buffer类会产生性能损耗。

所以建议在所有情形下选择使用String Builder类。

在String Builder类的使用中，主要的方法为append()。

```
public StringBuilder append(CharSequence s);
```

有不同的实现以传入不同类型的参数。

内置数据类型：boolean / char / int / long / float / double

字符串序列：char[] / CharSequence / String / String Buffer

对象类型：Object

当传入参数为对象(Object)类型时

通过String.valueOf(Object)将传入参数转换为字符串。

当Object参数为null时，String.valueOf(Object)返回"null"

否则调用Object.toString()方法并返回结果。

在Java中，通过加号拼接字符串实际上是通过String Builder类实现的。

即加号连接的String运算隐式地改为String Builder类的方法调用。

如 "score :" + 77 等价于 new StringBuilder("score :").append(77).toString()

```
new StringBuilder().append("score :").append(77).toString()
```

所以在Java中，可是对于不同类型的对象直接采用加号连接的String运算
而不用显式地先将变量转换为字符串再进行连接。

Java - P24

Part 9 - ~~数组与类~~

数组

在 Java 中，数组用来存储固定大小的同类型元素

必须先声明数组变量，才能在程序中使用数组

声明数组的基本语法为

<datatype>[] <arrayRefVar>;

或 <datatype> <arrayRefVar>[];

注意，虽然从效果上看两种语法是一致的

但是首选的声明风格为 <datatype>[] <arrayRefVar>;

由于 <datatype> <arrayRefVar>[]; 是来自于 C/C++ 的风格

提供这种形式是为了 C/C++ 程序员可以快速理解 Java

在 Java 中，使用 new 关键字来创建数组对象

基本语法结构为（声明与创建）：

<datatype>[] <arrayRefVar> = new <datatype>["<arraySize>"]

<datatype>["<arraySize>"]：用于创建一个新的数组对象

<datatype> 指定了数组元素的数据类型

<arraySize> 指定了数组的大小

在 Java 中，数组元素通过索引访问

数组索引从 0 开始，到 arrayRefVar.length - 1 结束

由于数组元素的类型和数组大小都是确定的

通常使用 for 循环 或增强型 for 循环 (for-each 循环) 处理数组

for-each 循环在不使用下标的前提下遍历数组

for (<datatype> element : array) {

...
}

数组也可以作为函数的传入参数或返回值

在 Java 中，多维数组可以看作数组的数组

声明的基本语法为：

<types>[][] <arrayName> = new <type>["<row>"]["<column>"]

<type>：可以是基本数据类型和复合数据类型

<row>, <column> 分别为行数和列数，必须是正整数

Java - P25

2019 - 2020

数组

在 Java 中，多维数组的低维部分可以有不同的尺寸
即可以从高维开始，分别为每一个维度分配空间
如 `String[][] s = new String[2][];`
`s[0] = new String[3];`
`s[1] = new String[4];`
`s.length → 2, s[0].length → 3, s[1].length → 4`

多维数组引用与一维数组类似，采用索引访问

分别对每个维度指定索引

如果有任何索引超过了定义的范围

build 时抛出错误 `java.lang.ArrayIndexOutOfBoundsException`

数组也可以通过字面常量进行初始化

如 `int[] i = {1, 2, 3, 4};`

`i.length → 4`

另外也可以通过这种方式定义数组的字面常量

如 `int[]{1, 2, 3, 4}.length → 4`

Arrays 类

在 Java 中，`Arrays` 类提供了静态的用于操作数组的方法

`public static int binarySearch(array, key)`

`public static int binarySearch(array, fromIndex, toIndex, key)`

根据传入参数的不同类型有不同的实现 `(array, key)`

内置数据类型：`byte[], byte / char[], char / int[], int`

`long / short[], short / long[], long / float[], float`

`double[], double`

对象类型：`Object[], Object`

`fromIndex`：定义二分搜索的起点，`] 定义二分搜索的范围`

`toIndex`：定义二分搜索的终点

另外提供了带有类型变量的版本

`public static <T> int binarySearch(T[] a, T key, Comparator<? super T> c)`

其中 `c` 指定了用于进行 `sort(T[], Comparator)` 的 `comparator`

注意这个 `sort(T[], Comparator)` 应在 `binarySearch` 之前完成

传入 `null` 时使用元素的自然排序 (natural ordering)

Java - P26

Arrays 类

public static boolean equals (array1, array2)

判断传入的两个 array 是否包含相同数量的元素且对应位置的元素均相等

two arrays considered equal: both arrays contain same number of elements

all corresponding pairs of elements are equal

根据传入参数数组的不同类型有不同的实现

内置数据类型: boolean[] / byte[] / char[] / int[]

short[] / long[] / float[] / double[]

对象类型: Object[]

当传入的 array1, array2 是 Object[] 类型时

判断条件为

即两个对象类型数组中对应位置的元素 e1 和 e2

e1, e2 are equal : (e1 == null ? e2 == null : e1.equals(e2))

或者说, 两个数组包含相同的元素且元素有相同的排序

contain the same elements in the same order

特别地有, 当数组引用都是 null 时, 也视为相同

array references considered equal if both are null

在 Java 中, 还提供了一个深度比较版本的函数

public static boolean deepEquals (Object[], a1, Object[] a2)

与 equals (Object[], Object[]) 相比, 用于比较任意深度的嵌套数组

appropriate for use with nested array of arbitrary depth

对于两个数组引用, deeply equal 当且仅当

或者两个 array reference 均为 null

或者所有对应位置的元素均为 deeply equal

all corresponding pairs of elements are deeply equal

对于两个可能为空 (possibly null) 的元素 e1, e2, deeply equal 当且仅当 满足其一

e1 和 e2 均为对象引用数组 (array of object reference type)

且 Arrays. deepEquals (e1, e2) return TRUE

e1 和 e2 为有相同基类的数组 (array of same primitive type) return TRUE

且 Arrays. equals (e1, e2) 的合理重载 (appropriate overloading)

e1 == e2 / e1.equals (e2) return TRUE

于是可知这个定义允许在任何深度出现 null

如果任何数组直接 / 间接包含其自身, 则此方法的行 为未定义 (behavior undefined)

contain itself either directly or indirectly through one or more levels of array

Java - P27

Arrays类

public static void sort(array)

public static void sort(array, int fromIndex, int toIndex)

根据传入参数 array 的不同类型有不同的实现

内置数据类型：byte[], char[], int[], short[], long[], float[], double[]

对象类型：Object[]

对于对象类型数组 Object[]

依据其元素的自然排序 (natural ordering of element)

且所有元素都必须实现 Comparable 接口

all elements in array must implement the Comparable interface

且所有的元素都必须是相互可比的 (mutually comparable)

即对于任意元素 e₁, e₂, e₁.compareTo(e₂) 必须不能抛出 ClassCastException

另外，sort 函数是保证稳定的 (guaranteed to be stable)

equal elements will not be reordered as a result of sort

fromIndex : index of first element (inclusive) to be sorted

toIndex : index of last element (exclusive) to be sorted

在 Java 中，还提供了一个传入参数类型为泛型的 sort 函数实现

public static <T> void sort(T[] a, Comparator<? super T> c)

public static <T> void sort(T[] a, int fromIndex, int toIndex, Comparator<? super T> c)

注意 所有的元素都必须是相互可比的 (mutually comparable)

即对于任意元素 e₁, e₂, c.compare(e₁, e₂) 必须不能抛出 ClassCastException

Date类

在 Java 中，java.util 包中提供 3 Date 类，用于封装日期和时间

public Date()，返回以当前日期和时间初始化的 Date 类对象

allocate Date object and initialize to represent the time the object allocated
measured to the nearest millisecond (毫秒)

public Date(long date)

standard time base

allocate Date object and initialize to represent specified time since

the epoch : "January 1st, 1970, 00:00:00 GMT"

date : specified number of milliseconds since "the epoch"

Java - P28

109 - 10.06

Date类

在Java中，可以通过`toString()`方法打印当前的日期和时间

```
public String toString()
    convert Date object to String of specific form
    <day> <mon> <dd> <hh>:<mm>:<ss> <zzz> <yyyy>
    <day> ::= Sun | Mon | Tue | Wed | Thu | Fri | Sat
    <mon> ::= Jan | Feb | Mar | Apr | May | Jun | Jul | Aug | Sep | Oct | Nov | Dec
    <dd> ::= 00 | 01 | ... | 31, day of month as two decimal digit
    <hh> ::= 00 | 01 | ... | 23, hour of day as two decimal digit
    <mm> ::= 00 | 01 | ... | 59, minute within hour as two decimal digit
    <ss> ::= 00 | 01 | ... | 61, second within minute as two decimal digit
    <zzz> ::= the time zone and may reflect daylight saving time (夏令时)
    standard time zone abbreviation include recognized by parse()
    if time zone not available, the <zzz> is empty
    <yyyy> : year as four decimal digit
```

```
System.out.println(new Date().toString());
→ Sun Jul 07 15:54:47 MDT 2019
```

在Java中，提供了不同的方式来比较两个日期

```
public long getTime()
    return number of millisecond since January 1, 1970, 00:00:00 GMT
```

```
new Date().getTime() → 1562536754760
```

可以使用`getTime()`获取两个时间的毫秒数并比较大小

```
public boolean before(Date when)
```

```
public boolean after(Date when)
```

```
public boolean equals(Object obj)
```

注意对于`equals()`，返回`true`当且仅当参数不为`null`且为`Date`类型对象且`Date objects equal if and only if getTime() return same long value`

```
public int compareTo(Date anotherDate)
```

由`Comparable`接口定义，`Date`类实现了这个接口

返回一个整型值，`=0`表示相等，`<0`表示早于，`>0`表示晚于

Java - P29

Java Date and Time API

格式化日期 在Java中，提供了SimpleDateFormat类用于格式化日期
以语言环境敏感的方式来格式化和分析日期的类
允许用户选择自定义日期时间格式来运行
`new SimpleDateFormat("yyyy-MM-dd G EEE HH:mm:ss:SSS, z XXX")
→ 2019-07-11 AD Thu 14:53:59:274, MDT -06:00

时间模式字符串(date and time pattern string), 用于指定时间格式

注意所有的ASCII字母都保留为模式字母

添加一般文本需要使用单引号('')

Letter	Date/Time component	Presentation
G	Era designator	Text
y	Year	Year
Y	Week year	Year
M	month in year (context sensitive)	Month] digit ≥ 3 : Text
L	month in year (standalone form)	Month] digit < 3 : Number
w	week in year	Number
W	week in month	Number
E	day name in week	Text] digit ≥ 4 : full form
u	day number in week (Monday: 1)	Number digit < 4 : short /
a	am/pm marker	Text abbreviated form
H	hour in day [0, 23]	Number
k	hour in day [1, 24]	Number
K	hour in am/pm [0, 11]	Number
h	hour in am/pm [1, 12]	Number
m	minute in hour	Number
s	second in minute	Number
S	millisecond	Number
z	time zone	General time zone
Z	time zone	RFC 822 time zone
X	time zone	ISO 8601 time zone
D	day in year	Number
d	day in month	Number
F	day of week in month	Number

Java - P30

时间戳

2019-07-20 15:01:05

格式化日期，在Java中，可以在printf()和format()方法中利用格式化字符串格式化日期

基本语法为 %[<index>\$]t [<specifier>]

specifier 用于指定输出日期时间的格式

'c'：输出时间日期的完整信息，等价于 "%ta %tb %td %tT %tz %ty"

printf("%tc", new Date())

→ Sat-Jul 20 15:01:05 MDT 2019

'Y'：输出 4 位数形式的年份，如 "2019"

'y'：输出 2 位数形式的年份，如 "19"

'm'：输出以数字表示的月份，如 "07"

'd'：输出以数字表示的日期，如 "20"

'H'：输出以 24 小时制表示的小时，如 "15"

'M'：输出小时中的分钟数，如 "01"

'S'：输出分钟中的秒数，如 "05"

'L'：输出 3-digit 表示的毫秒数

'N'：输出 9-digit 表示的纳秒数

'I'：输出以 12 小时制表示的小时，与 'H' 类似，但当 0~9 时不加前边的 0

'l'：与 'I' 类似，但当 0~9 时不加前边的 0

'p'：输出 am / pm

'Z'：输出以数字表示的时区，如 "-06:00"

'z'：输出以字符串表示的时区，如 "MDT"

's'：输出从 epoch 到 时间的秒数

'Q'：输出从 epoch 到 时间的毫秒数

'B'：输出以字符串表示的月份全称

'b'：与 'B' 类似，但输出为 3-digit 的简称

'A'：输出以字符串表示的星期全称

'a'：与 'A' 类似，但输出为 3-digit 的简称

'j'：输出 3-digit 表示的年中的天数

'R'：24 小时制表示的时间，等价于 "%tH:%tm"

'r'：12 小时制表示的时间，等价于 "%tI:%tm:%ts %Tp"

'T'：24 小时制表示的时间，等价于 "%tH:%tm:%ts"

'D'：输出日期，等价于 "%tm/%td/%ty"

'F'：输出日期，等价于 "%tY-%tm-%td"