2019-12-10 课程

公告

今天我们学习 Java 开发中最常用的两种数据类型字符串和数组。

- 字符串
  - 字符串的定义、创建
  - 字符串常用操作：长度、子串、拼接等
  - String / StringBuffer /StringBuilder
  - JVM 对字符串的优化
  - 应用练习：使用 TDD 编写
  - 常见面试问题
- 数组
  - 数组定义及声明
  - 数组初始化
  - 数组遍历
  - 数组拷贝
  - 数组填充
  - 数组排序
  - 多维数组和不规则数组
  - 应用练习
  - 常见面试问题
- 课后练习

字符串

Java 字符串就是 Unicode 字符序列。Java 没有内置的字符类型，而是在标准 Java 类库中提供了一个预定义类 `String` 。每个用双引号括起来的字符串都是 String 类的一个实例。可以通过直接赋值或者 new 操作符来创建字符串。

```
String str1 = "";
String str2 = "hello, this is a string.";
String str3 = new String("create string with new");
```

String 类没有提供用于修改字符串的方法，所以我们将 String 类对象称为不可变字符串，它被声明为 final class，所有的属性也被定义为 final 的。但是我们可以修改字符串变量，让它指向另外一个字符串。

为了提高内存利用率，JVM 有一个字符串常量池，每次使用双引号定义字符串，JVM 会先到该常量池中来检测是否已经存在，存在则直接该对象的引用；否则在常量池中创建一个新增并返回该值的引用。

使用 new 创建字符串（`new String("字符串");`）时，会直接在堆中创建该字符串并返回其引用。从 Java 6 开始，String 类提供了 `intern()` 方法，调用该方法时，JVM 去字符串常量池检测是否已存在该字符串，如果已经存在则直接返回引用；如果不存在则在常量池中添加并返回其引用。

Java 6 中，字符串常量池存在 PermGen 里，也就是(“永久代”)，而这个空间是有限的，基本不会被 FullGC 之外的垃圾收集机制扫描。如果使用不当，经常会发生 OOM。在后续版本中，将字符串常量池放在了堆中，而且默认缓存大小也在不断扩大。在 Java 8 中永久代 PermGen 也被元数据区 MetaSpace 替代。

下边我们通过一些示例来验证一下：

```
String str1 = "hello";
String str2 = "hello";
String str3 = "hello" + "world";
String str4 = str2 + "world";
String str5 = new String("hello");
String str6 = new String("hello");
String str7 = str6.intern();
String str8 = new String("hello").intern();

System.out.println("str1 = str2, " + (str1 == str2));
System.out.println("str3 = str4, " + (str3 == str4));
System.out.println("str1 = str5, " + (str1 == str5));
System.out.println("str5 = str6, " + (str5 == str6));
System.out.println("str1 = str7, " + (str1 == str7));
System.out.println("str1 = str8, " + (str1 == str8));

String str9 = "hello";
str9 += "world";
System.out.println("str3 = str9, " + (str3 == str9));
```

字符串操作

- 长度
  - `int length()` 返回采用 UTF-16 编码表示的给定字符串所需要的代码单元数量。也即是 String 类内部 char 数组的长度。（char 数据类型是一个采用 UTF-16 编码表示 Unicode 代码点的代码单元）
  - `int codePointCount(int beginIndex, int endIndex)` 表示字符串的实际长度，及代码点数。

```
    String str =
"hello,\uD835\uDD5D\uD835\uDD60\uD835\uDD60\uD835\uDD5C";
    System.out.println(str);
    System.out.println("length is: " + str.length());
    System.out.println("code point count is: " +
str.codePointCount(0, str.length()));
```

- 子串
  - `String substring(int beginIndex)`
  - `String substring(int beginIndex, int endIndex)`

```
    String str = "hello, world!";

    System.out.println(str.substring(1));

    System.out.println(str.substring(0, 1));
    System.out.println(str.substring(0, str.length() - 1));
    System.out.println(str.substring(0, str.length() + 1));
```

- 拼接

  可以直接使用 `+` 和 `+=` 运算符来进行字符串的拼接，例如：

  - String str = "hello" + " world!";
  - String str = "hello"; str += " world!";
  - String str = "hello"; str = str + " world!";

- 格式化

  为了让字符串拼接更简洁直观，我们可以使用字符串格式化方法 `String.format`

  - `%s` 字符串
  - `%c` 字符类型
  - `%b` 布尔类型
  - `%d` 整数类型（十进制数）
  - `%x` 整数类型（十六进制数）
  - `%o` 整数类型（八进制数）
  - `%f` 浮点类型
  - `%a` 浮点类型（十六进制数）
  - `%%` 百分比类型

  - `%n` 换行

    示例：

    ```
    System.out.printf("hello, %s %n", "world");
    System.out.printf("大写a：%c %n", 'A');
    System.out.printf("100 > 50：%b %n", 100 > 50);
    System.out.printf("100除以2：%d %n", 100 / 2);
    System.out.printf("100的16进制数是：%x %n", 100);
    System.out.printf("100的8进制数是：%o %n", 100);
    System.out.printf("100元打8.5折扣是：%f 元%n", 50 * 0.85);
    System.out.printf("上述价格的16进制数是：%a %n", 50 * 0.85);
    System.out.printf("上面的折扣是%d%% %n", 85);
    ```

- 相等判断

  - `equals` 判断是否相等。
  - `equalsIgnoreCase` 不区分大小写判断是否相等。

- 前缀判断

  ```
  "hello".startsWith("h")
  ```

- 后缀判断

  ```
  "hello".endsWith("o")
  ```

- 包含判断

  ```
  "hello".contains("ll")
  ```

- 查找

  - `indexOf` 从前边查找

- `lastIndexOf` 从后边开始找

  示例：

```
String str = "hello, world!";
System.out.println(str.indexOf("e"));
System.out.println(str.indexOf('e'));
System.out.println(str.indexOf(101));

System.out.println(str.indexOf("e", 2));

System.out.println(str.indexOf("l"));
System.out.println(str.lastIndexOf("l"));
System.out.println(str.lastIndexOf("l", 9));
```

- 查找替换
  - replace
  - replaceAll

    示例：

```
System.out.println("hello, world!".replace('o', 'A'));
System.out.println("hello, world!".replace("o", "OOO"));
System.out.println("hello, world!".replaceAll("o", "OOO"));
```

- 去空格

  System.out.println(" hello, world! ".trim());

- 大小写转换
  - System.out.println("Hello, world!".toUpperCase());
  - System.out.println("Hello, world!".toLowerCase());

- 空串和 Null 串
  - 空串是一个长度为0且内容为空的 String 对象。
  - String 存放 null，表示没有任何对象与该变量关联。

String / StringBuilder / StringBuffer

String 在拼接过程中或操作不当时，可能会产生大量的中间对象。而 StringBuffer 就是为了解决这个问题而提供的一个类，StringBuffer 是线程安全的，如果没有线程安全的需要则使用 StringBuilder（Java 1.5 中新增）。

`StringBuffer` 和 `StringBuilder` 都继承自 AbstractStringBuilder 类，而 StringBuffer 类的所有方法都使用关键字 `synchronized` 来保证线程安全。它们的底层都是通过可修改的 char 数组（Java 9 以后改为 byte 数组实现）来实现修改。以下内容没有特别说明则均基于 Java 8。

StringBuffer，StringBuilder 在创建时，如果未指定容量，默认容量为 16。如果容量可预估，则最好在创建时指定合适的大小，这样可以避免多次扩容。扩容会产生多重开销：抛弃原有数组、创建新的数组、进行 arraycopy。

StringBuffer，StringBuilder 常用方法：

- `append` 在字符串结尾追加
- `length` 返回当前长度
- `setLength` 设置字符串长度

示例：

```
String str1 = "hello" + " world" + "!";
System.out.println(str1);

StringBuffer strB1 = new StringBuffer();
strB1.append("hello");
strB1.append(" world");
strB1.append("!");
System.out.println(strB1.toString());

StringBuilder strB2 = new StringBuilder();
strB2.append("hello");
strB2.append(" world");
strB2.append("!");
System.out.println(strB2.toString());

System.out.println("strB2 length is " + strB2.length());

strB2.setLength(strB2.length() - 1);
System.out.println(strB2.toString());

strB2.setLength(strB2.length() + 10);
System.out.println(strB2.toString());
```

JVM 对字符串的优化

现代 JVM 的实现是很智能的，编译时 JVM 对 String 操作进行一些优化以提高程序的运行效率。

示例1

```
String str = "hello" + ", " + "world!";
System.out.println(str);
```

JVM 优化后

```
String str = "hello, world!";
System.out.println(str);
```

示例2

```
String str1 = "hello";
String str2 = str1 + ", world!";
System.out.println(str2);
```

JVM 优化后

```
String str1 = "hello";
StringBuilder str2 = new StringBuilder();
str2.append(str1);
str2.append(", world!");
System.out.println(str2.toString());
```

示例3

```
long start = System.currentTimeMillis();
String str = "";
for (int i = 0; i < 50000; i++) {
    str += i;
}
System.out.println(str.length());
System.out.println("耗时: " + (System.currentTimeMillis() - start) + "ms");
```

JVM 优化后

```
long start = System.currentTimeMillis();
String str = "";
for (int i = 0; i < 50000; i++) {
    StringBuilder tmp = new StringBuilder();
    tmp.append(str);
    tmp.append(i);
    str = tmp.toString();
}
System.out.println(str.length());
System.out.println("耗时: " + (System.currentTimeMillis() - start) + "ms");
```

for 循环经过优化后，虽然节省了空间，但是 StringBuilder 是在 for 循环内，每次都会创建。性能并不会提升，反而可能会下降。按下边实现方式改写代码后性能提升好几个数量级。

```
long start = System.currentTimeMillis();
StringBuilder str = new StringBuilder();
for (int i = 0; i < 50000; i++) {
    str.append(i);
}
System.out.println(str.length());
System.out.println("耗时: " + (System.currentTimeMillis() - start) + "ms");
```

说明：可以通过 `javap -c 编译生成的class文件` 反编译出字节码，然后来分析

应用练习

去掉字符串开头/结尾/中间的空格（不使用 trim 方法）

```
public String trimAll(String str) {
    StringBuilder tmp = new StringBuilder();
    for (int i = 0; i < str.length(); i++) {
        char c = str.charAt(i);
```

```
            if (c == ' ') {
                continue;
            }
            tmp.append(c);
        }
        return tmp.toString();
    }
```

反转字符串，比如输入 123，反转结果 321

```
    public String reverse(String str) {
        StringBuilder tmp = new StringBuilder();
        char[] chars = str.toCharArray();
        for (int i = chars.length - 1; i >= 0; i--) {
            char c = chars[i];
            tmp.append(c);
        }
        return tmp.toString();
    }
```

常见面试问题

== 和 equals 的区别

1. == 对于基本类型来说是值比较；而对于引用类型比较的是引用，是否是指向同一个对象的引用。
2. equals 默认是引用比较，而 Integer、String 等包装类都重写了 equals 方法，改为了值比较。

所以对象都可以看作是继承自 Object，我们来看一下 Object 的 equals 实现，如果自定义类未覆写 equals，调用对象实例的 equals 方法默认是引用比较。

```
    public boolean equals(Object obj) {
        return (this == obj);
    }
```

我们再来看一下 Integer 类的 equals 方法

```
    public boolean equals(Object obj) {
        if (obj instanceof Integer) {
            return value == ((Integer)obj).intValue();
        }
        return false;
    }
```

下列代码的执行结果

```
    String s1 = "hello" + ", world!";
    String s2 = "hello";
    s2 += ", world!";
    String s3 = "hello, world!";
    String s4 = s2.intern();
```

```
System.out.println(s1 == s2);
System.out.println(s1 == s3);
System.out.println(s1 == s4);
System.out.println(s2 == s3);
System.out.println(s2 == s4);
```

执行结果：

```
false
true
true
false
false
```

String / StringBuffer / StringBuilder 的区别

- String 为不可变字符串；StringBuffer 和 StringBuilder 为字符串可变对象。
- String 的 substring 等修改操作每次都会产生一个新的 String 对象；字符串拼接性能 String 低于 StringBuffer，而 StringBuffer 低于 StringBuilder。
- StringBuffer 是线程安全的，StringBuilder 而线程不安全的。二者都是继承自 AbstractStringBuilder，它们的唯一区别是 StringBuffer 的所有方法都使用了 synchronized 修饰符来保证线程安全。

String 对象的 intern 的作用

String 对象的 intern 方法用于字符串的显示排重。调用此方法时，JVM 去字符串常量池查找池中是否已经存在该字符串，如果已存在则直接返回它的引用；如果不存在则在池中先创建然后返回其引用。

String 不可变性的优点

- 字符串不可变，因此可以通过字符串常量池来实现，共享对象，从而节省空间，提高性能。
- 多线程安全，因为字符串不可变，所以当字符串被多个线程共享时不会存在线程安全问题。
- 适合做缓存的 Key，因为字符串不可变，因此它的哈希值也就不变；创建时它的哈希值就被缓存了，不需要重新计算，速度更快。

String 是否可以被继承

String 不能被继承。因为 String 被声明为 final，所以不能被继承。

数组

数组是一种数据结构，用来存放同一类型的值的集合。通过整数下标来访问数组中的值，数组下标从 `0` 开始；当下标越界，不在范围之内时，程序会报错 `java.lang.ArrayIndexOutOfBoundsException` 。

数组是一种引用类型，只能用来存储固定大小的同类型数据。在 Java 中很多集合的内部都是使用数组来实现的，比如 ArrayList 和 HashMap 等。

数组的常用排序算法：冒泡算法、选择排序。

声明数组变量时，必须指明数组类型，类型后边紧跟 `[]` 或者将 `[]` 放在数组变量之后，数组类型可以是基本数据类型或者引用类型。例如：

```
int[] arr1;
int arr2[];
String[] arr3;
String arr4[];
```

数组初始化

数组有以下几种初始化方式：

1. `int[] arr1 = new int[3];` / `String[] arrS1 = new String[3];`

   使用 new 操作符创建的数组时，基本数据类型每个值会初始化为二进制的0；而引用类型会初始化为null。

2. `int[] arr2 = new int[] {1, 2, 3};` / `String[] arrS2 = new String[]{"1", "2", "3"};`

   使用 new 操作符创建数组时，直接使用大括号方法赋值，数组的长度为大括号内元素的个数，不能在[] 内指定长度，否则编译器会报错。

3. `int[] arr3 = { 1, 2, 3};` / `String[] arrS3 = {"1", "2", "3"};`

   使用第 2 种方式创建数组时，可以省略前边的 new 操作符而直接使用大括号。

数组遍历

Java 种对数组的遍历主要有以下三种方式：

1. for 循环使用数组下标
2. for each 循环
3. Java 8 中新增的 Lambda 表达式

示例：分别使用上述三种方式遍历数组 `Integer[] arr = {1, 2, 3, 4, 5};`

```
Integer[] arr = {1, 2, 3, 4, 5};

System.out.println("1. for循环使用数组下标");
for (int index = 0; index < arr.length; index++) {
    System.out.println(arr[index]);
}

System.out.println("2. for each循环");
for (int numb : arr) {
    System.out.println(numb);
}

System.out.println("3. Java 8新增的 Lambda 表达式");
Arrays.asList(arr).forEach(numb -> System.out.println(numb));
```

特别说明： 将 `Integer[] arr = {1, 2, 3, 4, 5};` 改为 `int[] arr = {1, 2, 3, 4, 5};`，然后再执行，看看使用 Java 8 新增的 Lambda 表达式方式时的输出结果有什么不一样。

数组拷贝

Java 中数组拷贝常用的两种方式：

1. 使用工具类 Arrays.copyOf 或 Arrays.copyOfRange
2. 使用底层方法 System.arraycopy

示例：

```
Integer[] arr1 = {1, 2, 3, 4, 5};
Integer[] arr2 = {6, 7, 8, 9, 10};

// 拷贝数组 arr1 的前 3 个元素
Integer[] arr3 = Arrays.copyOf(arr1, 3);
System.out.println(Arrays.toString(arr3));

// 拷贝数组 arr1 的第 1 位到第 3 位的元素（不包括第 3 位）
Integer[] arr4 = Arrays.copyOfRange(arr1, 1, 3);
System.out.println(Arrays.toString(arr4));

// 拷贝数组 arr2 的后 3 位到 arr1 到后 3 位
System.arraycopy(arr2, 2, arr1, 2, 3);
System.out.println(Arrays.toString(arr1));
```

数组填充

使用工具类提供的方法 `Arrays.fill` 可以对数组全部或指定范围内的元素赋值为指定的值。

示例：定义一个大小为 10 的 int 数组，并将数组全部初始化为指定的值 5，打印数组；然后将数组后 3 为赋值为 3，再次打印数组。

```
int arr[] = new int[10];
Arrays.fill(arr, 5);
System.out.println(Arrays.toString(arr));

Arrays.fill(arr, arr.length - 3, arr.length, 3);
System.out.println(Arrays.toString(arr));
```

数组排序

可以使用 Arrays.sort 方法对数组进行排序。

示例：定义一个大小为 100 的 int 数组，随机给每一位赋值一个 `0 ~ 100` 之间的数值，然后对该数组进行排序并打印排序结果。

```
int arr[] = new int[100];
for (int index = 0; index < arr.length; index++) {
    arr[index] = new Random().nextInt(100);
}
System.out.println("排序前：" + Arrays.toString(arr));
Arrays.sort(arr);
System.out.println("排序后：" + Arrays.toString(arr));
```

多维数组和不规则数组

前边介绍出现的数组，都是一维数组，Java 实际上没有多维数组，只有一维数组。多维数组可以被理解为“数组的数组”。多维数组的同一个维可以有不同的长度，因此也可以称为不规则数组。声明时使用多个 [] 标识来声明。

示例：定义一个二维数组，第一维表示用户，第二维表示用户的具体信息（1. 编码, 2. 姓名, 3.性别, 4.年龄）。定义赋值并打印。

```
String[][] users = new String[3][];
users[0] = new String[4];
users[0][0] = "001";
users[0][1] = "张三";
users[0][2] = "女";
users[0][3] = "25";

users[1] = new String[4];
users[1][0] = "002";
users[1][1] = "李四";
users[1][2] = "男";
users[1][3] = "30";

System.out.println(Arrays.toString(users));

for (String[] user : users) {
    System.out.println(Arrays.toString(user));
}
```

应用练习

给定一个整数数组 nums 和一个目标值 target，请你在该数组中找出和为目标值的那 两个 整数，并返回他们的数组下标。

你可以假设每种输入只会对应一个答案。但是，你不能重复利用这个数组中同样的元素。

示例:

```
给定 nums = [2, 7, 11, 15], target = 9

因为 nums[0] + nums[1] = 2 + 7 = 9
所以返回 [0, 1]
```

来源：https://leetcode-cn.com/problems/two-sum/

解答代码：

```
public int[] twoSum(int[] nums, int target) {
    for (int start = 0; start < nums.length - 1; start++) {
        for (int secondNumStart = start + 1; secondNumStart < nums.length;
secondNumStart++) {
            if (nums[start] + nums[secondNumStart] == target) {
                return new int[] {start, secondNumStart};
            }
        }
```

```
        }
        return null;
    }
```

这个解答代码仅仅实现了功能，算法不是最优的，尝试进行优化。

常见面试问题

下列代码的执行结果

```
int[] n = new int[3] {1, 2, 3};
System.out.println(n[1]);
```

解答：编译时报错，数组声明时如果使用大括号赋值，不能在 [] 内指定数组的长度，数组的长度为大括号内元素的个数。

下列代码的执行结果

```
private static void plus1ForEvenIndex(int[] arr) {
    for (int index = 0; index < arr.length; index++) {
        if ((index + 1) % 2 == 0) {
            arr[index] = arr[index] + 1;
        }
    }
}

public static void main(String[] args) {
    int[] n = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    plus1ForEvenIndex(n);
    System.out.println(Arrays.toString(n));
}
```

执行结果：[1, 3, 3, 5, 5, 7, 7, 9, 9, 11]。Java 方法参数是值传递，方法得到是参数的拷贝，所以方法内代码不会改变所传参数的值。但对于引用类型，虽然不能改变引用类型的引用，却可以修改引用类型指向的对象的值。

下列代码的执行结果

```
int[] n = new int[3];
String[] s = new String[3];
System.out.println(n[1]);
System.out.println(s[1]);
```

执行结果：0 和 null。数组声明时未进行赋值时，对于基本数据类型，将每一个元素赋值为二进制0；而对于引用类型，则将每一个元素赋值为 null。

数组的遍历方式有哪些

- for 循环使用数组下标
- for each 循环
- Java 8 新增的 Lambda 表达式

举例说明 Arrays 工具类的常用方法

- copyOf
- copyOfRange
- fill
- sort
- toString
- equals
- asList
- binarySearch

每个方法的作用，以及用法大家自己来补充。

课后练习

1. 自己编写练习一下课程中的示例和练习。
2. 编写数据验证工具类，并编写相应单元测试用例，具体要求如下：
    - GitHub 上创建项目 java-common-tools
    - 创建 maven 项目（使用自己熟悉的IDE），并提交到 GitHub 上
    - 创建数据验证工具类 com.zeroten.common.util.CheckUtils。（src/main/java目录下）
    - 为工具类编写如下数据验证方法（使用静态方法）：
        - public static boolean isAnyEmpty(String... strings)

          `String... strings` 为可变参数，你可以认为它是 String[] strings。如果 strings 中有任意一个字符串为空，则返回 false，否则返回 true。
        - public static boolean isEmpty(Object[] arr)

          判断引用类型数组是否为空，为空或 null 则返回 true，否则返回 false。
        - public static boolean equals(String str1, String str2)

          判断 str1 字符串是否相等，相等则返回 true，否则返回 false。当其中一个是 null 时返回 false。
        - public static boolean equals(Integer n1, Integer n2)

          判断 n1 和 n2 的值是否相等，相等则返回 true，否则返回 false。当其中一个是 null 时返回 false。
    - 编写测试类 com.zeroten.common.util.CheckUtilsTest（src/main/test目录下）对数据验证工具类进行测试，设计的测试用例尽量把各种输入情况都考虑到。
    - 将作业提交 GitHub 并写清楚提交说明。
3. 注册 LeanCloud 账号，并试着创建应用，安装它的客户端工具。

   后边 web 部分会用到 LeanCloud 来托管咱们自己写的应用，提前注册熟悉一下，网址：https://www.leancloud.cn/