

前言

Java 语言经过 20 多年的发展，变得更加成熟、易用，而且多年来一直是排名第一的语言，在很多领域仍然是不可替代的。目前，市场上讲解 Java 的书虽然很多，但大多是循规蹈矩地讲授 Java 知识点，讲授方式枯燥无趣，初学者不容易看进去。2020 年，我与赵大羽老师合作出版了一本以漫画形式介绍 Python 的图书，市场反馈非常好，鉴于此，这里我们再次合作，联手为广大读者奉献一本漫画形式的 Java 书。

本书内容

本书共 20 章，适合零基础的初学者学习 Java，具体内容如下。

第 1 章介绍了 Java 的历史和特点、Java 三大平台和 Java 虚拟机。

第 2 章主要介绍了 Java 开发环境的搭建，包括 JDK 的下载、安装以及配置过程。

第 3 章介绍了 Java 开发工具，使用常用的文本编辑工具编写 HelloWorld 程序，并通过 JDK 编译和运行 Java 程序。

第 4 章主要介绍了 Java 最基本的语法。

第 5 章介绍了 Java 的基本运算符。

第 6 章主要介绍了 Java 的数据类型。

第 7 章介绍了 Java 的数组。

第 8 章主要介绍了 Java 的字符串数据类型。

第 9 章主要介绍了 Java 的判断语句。

第 10 章主要介绍了 Java 的循环语句。

第 11 章主要介绍了 Java 中面向对象编程的基础内容。

第 12 章主要介绍了 Java 中面向对象编程的进阶内容。

第 13 章重点介绍了 Java 函数式编程。



第 14 章介绍了 Java 的异常处理机制。

第 15 章主要介绍了 Java 文件管理和 I/O 流技术。

第 16 章主要介绍了 Java 访问互联网资源的相关内容。

第 17 章介绍了 Java 中的集合，其中包括常用接口 `Collection`、`Set`、`List` 和 `Map`。

第 18 章介绍了 Java 中的图形用户界面编程技术 `Swing`。

第 19 章介绍了 Java 线程技术。

第 20 章介绍了基于 Java 实现的网络爬虫项目。

知识图谱



源代码

为了方便读者学习，我们给广大读者提供了书中实例的配套代码，大家可以到图灵社区本书主页免费注册下载。

免费视频

另外，为了帮助读者学习，我们按照本书讲授的知识点录制了讲解视频。读者可以结合讲解视频来学习本书。视频使用方式，大家可以通过如下二维码免费观看。



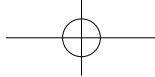
视频讲解

致谢

在此感谢人民邮电出版社图灵公司的王军花编辑在本书写作过程中给予我们的指导与鞭策。感谢赵大羽老师手绘了书中全部的漫画。感谢赵静仪为漫画提供的新鲜灵感和创意。感谢我们智捷团队的赵志荣、关锦华参与本书的部分编写工作。感谢我们的家人对我们的理解、关心和照顾，使我们能够投入全部精力来专心编写本书。

由于时间仓促，书中难免存在不妥之处，请读者原谅，并提出宝贵意见。

2021 年 10 月于鹤城
关东升



人民邮电出版社

目 录

第 1 章 Java 咖啡真好喝 / 1

1.1 Java 语言与（爪哇）咖啡 / 2

1.2 Java 语言的历史 / 3

1.3 Java 语言的特点 / 3

1.4 Java 平台 / 5

1.4.1 Java SE / 5

1.4.2 Java EE / 5

1.4.3 Java ME / 5

1.5 Java 虚拟机 / 6

同步练习题 / 7

第 2 章 磨刀不误砍柴工

Java 环境搭建 / 8

2.1 下载和安装 JDK / 9

2.2 配置 JDK / 10

2.3 测试环境 / 14

同步练习题 / 15

第 3 章 光说不练假把式

第一个 Java 应用程序 / 16

3.1 使用文本编辑工具编写 HelloWorld 程序 / 17

3.1.1 用记事本编写 Java 应用程序 / 17

3.1.2 编译和运行 Java 应用程序 / 18

3.2 专业项目开发工具 IntelliJ IDEA / 20

3.2.1 下载和安装 IntelliJ IDEA / 21

3.2.2 配置 IntelliJ IDEA / 22

3.2.3 使用 IntelliJ IDEA 编写和运行
Java 应用程序 / 23

3.3 解释代码 / 27

同步练习题 / 30

第 4 章 万丈高楼平地起

Java 语法基础 / 31

4.1 关键字和保留字 / 32

4.2 标识符 / 33

4.3 Java 分隔符 / 33

4.3.1 分号 / 34

4.3.2 大括号 / 34

4.3.3 空白 / 35

4.4 注释 / 35

4.4.1 单行注释 / 35

4.4.2 多行注释 / 36

4.4.3 文档注释 / 36

4.5 变量 / 37

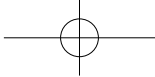
4.6 常量 / 38

4.7 Java 源代码的组织方式 / 39

4.7.1 Java 源代码文件 / 39

4.7.2 包 / 40

同步练习题 / 42



第5章 掐指一算

运算符 / 43

5.1 算术运算符 / 44

5.1.1 一元运算符 / 44

5.1.2 二元运算符 / 45

5.2 关系运算符 46

5.3 逻辑运算符 48

5.4 位运算符 / 50

5.5 赋值运算符 / 52

5.6 运算符的优先级 / 53

同步练习题 / 54

第6章 一大波数据向你走来

Java 数据类型 / 56

6.1 基本数据类型 57

6.1.1 整数类型 / 58

6.1.2 浮点类型 / 59

6.1.3 字符类型 / 60

6.1.4 布尔类型 / 62

6.2 数值类型数据的转换 / 63

6.2.1 自动类型转换 / 63

6.2.2 强制类型转换 / 64

6.3 引用数据类型 / 66

同步练习题 / 68

第7章 把数据集中管理起来

数组 / 69

7.1 声明数组 / 70

7.2 数组初始化 / 70

7.3 访问数组元素 / 72

7.4 多维数组 / 73

7.4.1 二维数组的声明 / 73

7.4.2 二维数组的初始化 / 73

同步练习题 / 76

第8章 字符串，“好吃”吗？

字符串 / 77

8.1 字符串表示形式 / 78

8.2 不可变字符串 / 79

8.2.1 字符串拼接 / 81

8.2.2 字符串查找 / 82

8.2.3 字符串比较 / 85

8.2.4 字符串截取 / 87

8.3 可变字符串 / 88

8.3.1 字符串追加 / 89

8.3.2 字符串插入、删除和替换 / 90

同步练习题 / 92

第9章 让程序学会思考

判断语句 / 93

9.1 if 语句 / 94

9.1.1 if 结构 / 94

9.1.2 if-else 结构 / 96

9.1.3 if-else-if 结构 / 97

9.2 switch 语句 / 98

同步练习题 / 101

第10章 别转圈了，我都懵了

循环语句 / 102

10.1 循环语句 / 103

10.1.1 while 语句 / 103

10.1.2 do-while 语句 / 104

10.2 for 语句 / 105

 10.2.1 Java 语言风格的 for 循环 / 105

 10.2.2 C 语言风格的 for 循环 / 106

10.3 跳转语句 / 108

 10.3.1 break 语句 / 108

 10.3.2 continue 语句 / 110

同步练习题 / 112

第 11 章 我喜欢“对象”

Java 面向对象编程基础 / 113

11.1 面向对象的概念 / 114

11.2 定义类 / 115

 11.2.1 成员变量 / 116

 11.2.2 成员方法 / 116

11.3 方法重载 / 118

11.4 对象 / 120

 11.4.1 创建对象 / 120

 11.4.2 空对象 / 121

 11.4.3 对象销毁 / 121

11.5 构造方法 / 122

 11.5.1 默认构造方法 / 123

 11.5.2 构造方法重载 / 124

11.6 类的封装性 / 126

 11.6.1 私有级别 / 126

 11.6.2 默认级别 / 128

 11.6.3 公有级别 / 129

 11.6.4 保护级别 / 130

11.7 类变量和类方法 / 133

11.8 静态代码块 / 135

同步练习题 / 137

第 12 章 那些“烧脑”的面向对象知识

Java 面向对象编程进阶 / 138

12.1 类的继承性 / 139

12.2 抽象类 / 143

12.3 接口 / 145

12.4 多态性 / 146

 12.4.1 对象类型检查 / 147

 12.4.2 对象类型转换 / 149

12.5 内部类 / 151

同步练习题 / 155

第 13 章 我不喜欢“抽象的”函数

Java 函数式编程 / 156

13.1 函数式接口 / 157

13.2 Lambda 表达式 / 160

同步练习题 / 163

第 14 章 一次郊游引发的意外

异常处理 / 164

14.1 理解 Java 中的异常 / 165

 14.1.1 分析异常栈跟踪信息 / 167

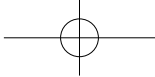
 14.1.2 好大一棵“异常树” / 168

14.2 捕获异常 / 169

14.3 释放资源 / 171

14.4 声明方法抛出异常 / 174

同步练习题 / 177



第 15 章 上级来文件了

访问文件 / 179

15.1 文件管理 / 180

15.2 I/O 流 / 184

15.3 字节流 / 184

15.3.1 字节输入流类的继承树 / 184

15.3.2 字节输出流类的继承树 / 185

15.3.3 实践一下：文件复制工具 / 186

15.4 字符流 / 190

15.4.1 字符输入流类的继承树 / 191

15.4.2 字符输出流类的继承树 / 191

15.4.3 实践一下：使用字符流重构文件复制工具 / 192

15.5 中介流 / 194

同步练习题 / 198

第 16 章 我要上网

Java 互联网编程 / 200

16.1 网络基础知识 / 201

16.1.1 HTTP/HTTPS 协议 / 201

16.1.2 什么是 URL / 202

16.2 Java 访问互联网资源的相关类 / 202

16.2.1 URL 类 / 202

16.2.2 HttpURLConnection 类 / 204

16.2.3 实践一下：图片下载器 / 206

同步练习题 / 212

第 17 章 快到碗里来

集合 / 213

17.1 Java 集合继承树 / 214

17.2 List / 214

17.2.1 List 的常用方法 / 215

17.2.2 遍历 List / 218

17.3 Set / 220

17.3.1 Set 的常用方法 / 220

17.3.2 遍历 Set / 222

17.4 Map / 223

17.4.1 Map 的常用方法 / 223

17.4.2 遍历 Map / 225

17.5 泛型 / 227

同步练习题 / 232

第 18 章 我讨厌那个黑乎乎的窗口

图形用户界面编程 / 233

18.1 Swing 技术概述 / 234

18.1.1 Swing 容器类继承树 / 234

18.1.2 Swing 的组件类继承树 / 234

18.2 你的第一个 Java GUI 程序 / 235

18.3 添加更多组件 / 236

18.4 事件处理机制 / 238

18.4.1 事件处理三要素 / 238

18.4.2 事件处理流程 / 239

18.4.3 实践一下：事件处理示例 / 241

18.4.4 使用匿名内部类实现事件监听器 / 242

18.4.5 使用 Lambda 表达式实现事件监听器 / 243

18.5 布局管理 / 245

18.5.1 流式布局 / 245

18.5.2 边界布局 / 247

18.5.3 网格布局 / 248

18.5.4 使用可视化设计工具 / 250

18.6 常用的 Swing 组件 / 253

- 18.6.1 文本输入组件 / 253
- 18.6.2 收音机按钮 / 255
- 18.6.3 复选框 / 259
- 18.6.4 列表 / 261

同步练习题 / 263

第 19 章 我要同时做几件事情
多线程编程 / 264

19.1 线程的相关概念 / 265

- 19.1.1 进程 / 265
- 19.1.2 线程 / 265
- 19.1.3 主线程 / 266

19.2 子线程 / 266

- 19.2.1 创建子线程 / 267
- 19.2.2 线程执行对象 / 267
- 19.2.3 使用匿名内部类实现线程执行对象 / 270
- 19.2.4 使用 Lambda 表达式实现线程执行对象 / 270
- 19.2.5 简化！再简化！ / 271

19.3 线程的状态 / 273

19.4 线程管理 / 274

- 19.4.1 死循环与“劳模”线程 / 274
- 19.4.2 “软着陆”停止线程 / 275

- 19.4.3 我等你，不见不散——等待其他线程结束 / 277

19.5 线程的安全 / 279

- 19.5.1 “吃苹果”问题 / 280
- 19.5.2 线程同步 / 283

同步练习题 / 286

第 20 章 Java 项目实战
网站图片爬虫 / 287

20.1 爬虫如何修炼 / 288

20.2 青铜级爬虫：爬取数据 / 288

20.3 白银级爬虫：解析数据 / 291

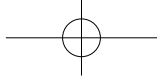
- 20.3.1 使用正则表达式 / 292
- 20.3.2 下载和安装 jsoup 库 / 296
- 20.3.3 jsoup 库的基本用法——常用 API / 297

20.4 黄金级爬虫：存储数据 / 300

20.5 铂金级爬虫：爬虫工作计划 / 305

20.6 钻石级爬虫：最终修炼完成 / 308

同步练习题答案 / 312



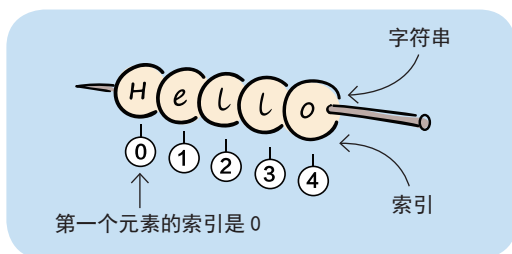
第 8 章 字符串，“好吃”吗？

字符串

“字符串”是由字符组成的一串字符序列，其中的每一个字符都是通过索引来提取的。按字符串从左到右的顺序，索引从 0 开始依次递增。

本章中我们介绍的内容如下：

- 1 字符串表示形式
- 2 不可变字符串
- 3 可变字符串



视频讲解



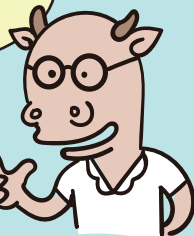


8.1 字符串表示形式



牛哥，每一种数据类型都有一种表示形式，例如 123 表示的是整数类型数据，那么字符串如何表示呢？

在 Java 中，我们使用双引号"" 将字符串括起来。

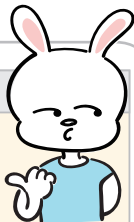


在下面的示例中，s1 和 s2 变量都表示 "Hello World" 字符串：

```
public class HelloWorld {  
  
    public static void main(String[] args) {  
  
        String s1 = "Hello World";  
        String s2 = "\u0048\u0065\u006c\u006c\u006f\u0020\u0057\u006f\u0072\u006c\u0064";  
        System.out.println("s1 = " + s1);  
        System.out.println("s2 = " + s2);  
  
    }  
  
}
```

输出结果
看这里。

```
s1 = Hello World  
s2 = Hello World
```



可以看到，无论采用普通的字符，还是采用 Unicode 编码表示的字符串，输出结果都是相同的，例如，通过 `System.out.println` 方法输出到控制台时，我们也会看到 "Hello World" 字符串。

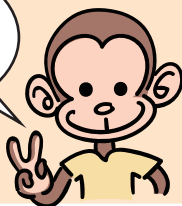
8.2 不可变字符串

很多计算机语言都提供了两种字符串, 即不可变字符串和可变字符串, 它们的区别在于当字符串进行拼接等操作时, 不可变字符串会创建新的字符串对象, 而可变字符串不会创建新对象。在 Java 中, 不可变字符串类是 `String`, 它也是 Java 中非常重要的类, 属于 `java.lang` 包。

小贴士

`java.lang` 包中有很多 Java 基础类, 包括 `Object`、`Class`、`String` 和 `Math` 等。在使用 `java.lang` 包中的类时, 不需要引入 (`import`) 包, 因为它是由解释器自动引入的。当然, 如果你引入该包, 程序也不会有编译错误。

创建不可变字符串对象的方式有下面两种。



- 直接使用字符串常量创建。
- 使用 `new` 关键字创建, `new` 关键字会为创建的对象开辟内存空间。

小贴士

在 Java 中, 使用 `new` 关键字创建对象时, 首先开辟内存空间, 此时对象还没有初始化, 而初始化对象的过程是通过构造方法完成的。当开辟完内存空间后, Java 虚拟机紧接着会调用构造方法实现对象的初始化。例如在 `new String()` 语句中, 首先开辟内存空间创建对象, 然后 `String()` 调用构造方法初始化对象。

为了初始化对象, Java 中的类一般都有多个构造方法。不可变字符串类 `String` 常用的构造方法如下。



- `String()`: 使用空字符串创建并初始化一个新的 `String` 对象。
- `String(String original)`: 使用另外一个字符串创建并初始化一个新的 `String` 对象。
- `String(StringBuffer buffer)`: 使用可变字符串对象 `StringBuffer` 创建并初始化一个新的 `String` 对象。
- `String(StringBuilder builder)`: 使用可变字符串对象 `StringBuilder` 创建并初始化一个新的 `String` 对象。
- `String(char[] value)`: 通过字符数组创建并初始化一个新的 `String` 对象。
- `String(char[] value, int offset, int count)`: 通过字符数组的子数组创建并初始化一个新的 `String` 对象; `offset` 参数是子数组第一个字符的索引, `count` 参数指定子数组的长度。



创建不可变字符串对象的示例代码如下：

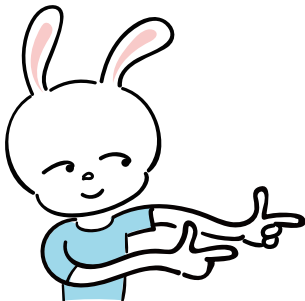
```
public class HelloWorld {  
  
    public static void main(String[] args) {  
  
        // 创建空字符串对象  
        String s1 = new String();  
        // 通过常量创建字符串 Hello World 对象  
        String s2 = "Hello World";  
        // 通过 new 创建字符串 Hello World 对象  
        String s3 = new String("Hello World");  
  
        char chars[] = {'a', 'b', 'c', 'd', 'e'};  
        // 通过字符数组创建字符串对象  
        String s4 = new String(chars);  
        // 通过子字符数组创建字符串对象  
        String s5 = new String(chars, 1, 3);  
        System.out.println("s4 = " + s4);  
        System.out.println("s5 = " + s5);  
        System.out.println("s5 字符串的长度 = " + s5.length());  
    }  
}
```

开辟内存空间创建对象

调用构造方法初始化对象

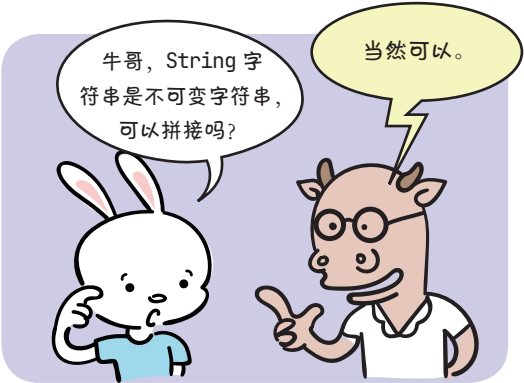
获得字符串长度的方法是 length()

输出结果看这里。



```
s4 = abcde  
s5 = bcd  
s5 字符串的长度 = 3
```

8.2.1 字符串拼接



String 字符串虽然是不可变字符串，但也可以进行拼接，只是会产生一个新的对象。String 字符串拼接可以使用 + 运算符或 String 的 concat(String str) 方法。+ 运算符的优势是可以连接任何类型数据并将其拼接成为字符串，而 concat 方法只能拼接 String 字符串类型的数据。

字符串拼接示例如下：

```
public class HelloWorld {
    public static void main(String[] args) {
        String s1 = "Hello";
        // 使用 + 运算符连接
        String s2 = s1 + "World";
        System.out.printf("s1= %s\n", s2);
        String s3 = "Hello";
        s3 += " World";
        System.out.printf("s3= %s\n", s3);

        String s4 = "Hello";
        // 使用 concat 方法连接
        s4 = s4.concat(" ").concat("World");
        System.out.printf("s4= %s\n", s4);

        int age = 18;
        String s5 = " 她的年龄是 " + age + " 岁。";
        System.out.printf("s5= %s\n", s5);

        java.util.Date now = new java.util.Date();
        // 对象拼接自动调用 toString 方法
        String s6 = " 今天是: " + now;
        System.out.printf("s6= %s\n", s6);
    }
}
```

s2 是新创建的字符串对象

字符串连接也支持 += 赋值运算符

采用 concat 方法进行拼接，并且可以连续调用该方法

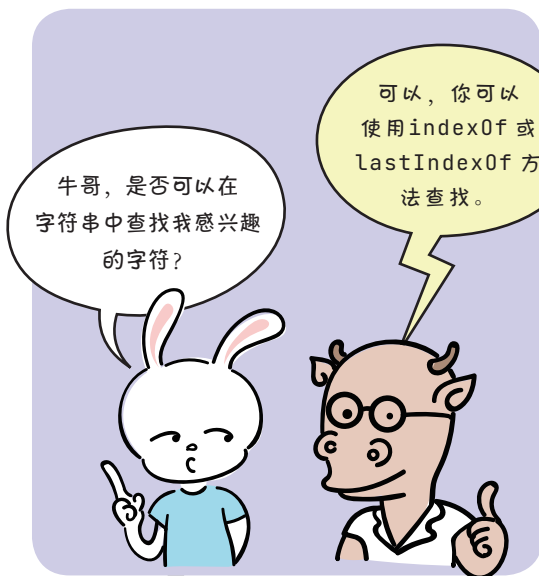
字符串与对象可以进行拼接。在 Java 中，所有对象都有 toString 方法，用于将对象转换为字符串

```
s1= HelloWorld
s3= Hello World
s4= Hello World
s5= 她的年龄是 18 岁。
s6= 今天是: Tue Dec 29 16:53:28 CST 2020
```



8.2.2 字符串查找

`indexOf` 和 `lastIndexOf` 方法用于查找字符或子字符串，其返回值是查找的字符或子字符串所在的位置，-1 表示没有找到。`indexOf` 方法是从左往右查找，它的几个重载方法如下。



- `int indexOf(int ch)`: 从左往右查找字符 `ch`，返回第一次找到字符 `ch` 所在处的索引。
- `int indexOf(int ch, int fromIndex)`: 从指定的索引 (`fromIndex`) 开始，从左往右查找字符 `ch`，返回第一次找到字符 `ch` 所在处的索引。
- `int indexOf(String str)`: 从左往右查找子字符串 `str`，返回第一次找到子字符串所在处的索引。
- `int indexOf(String str, int fromIndex)`: 从指定的索引开始从左往右查找字符串 `str`，返回第一次找到字符串所在处的索引。

`lastIndexOf` 方法也有类似的 4 个方法，只不过它是从右往左查找的，参数一样，具体内容不再赘述。示例代码如下：

```
public class HelloWorld {
    public static void main(String[] args) {
```

```
String s = "Hello!";  
// 获得字符串长度  
int len = s.length(); // 返回 6  
// 获得索引位置 1 的字符  
char ch = s.charAt(1); // 返回字符 e  
// 从左往右查找字符 l  
int pos = s.indexOf('l'); // 返回 2  
// 从右往左查找字符 l  
pos = s.lastIndexOf('l'); // 返回 3  
pos = s.indexOf("lo"); // 返回 3  
pos = s.lastIndexOf("lo"); // 返回 3  
pos = s.indexOf('l', 4); // 返回 -1
```

返回字符串 s 的长度

返回索引位置 1 的字符，字符串 s 的索引如下图所示。

索引	0	1	2	3	4	5
	H	e	l	l	o	!

返回字符串 s 的长度

```
}  
}
```

牛哥，字符串作为一个类，应该有很多方法吧？你之前介绍了几个方法，其他方法怎么用呢？

你可以查询 API 文档。

Java 有很多类，每个类又有很多方法和变量。查看 Java API 文档，就能够知道这些类、方法和变量如何使用。Java 官方提供的在线 API 文档页面如下页图所示。



为了在 API 文档中找到感兴趣的主体，可以通过主题框搜索感兴趣的主体。

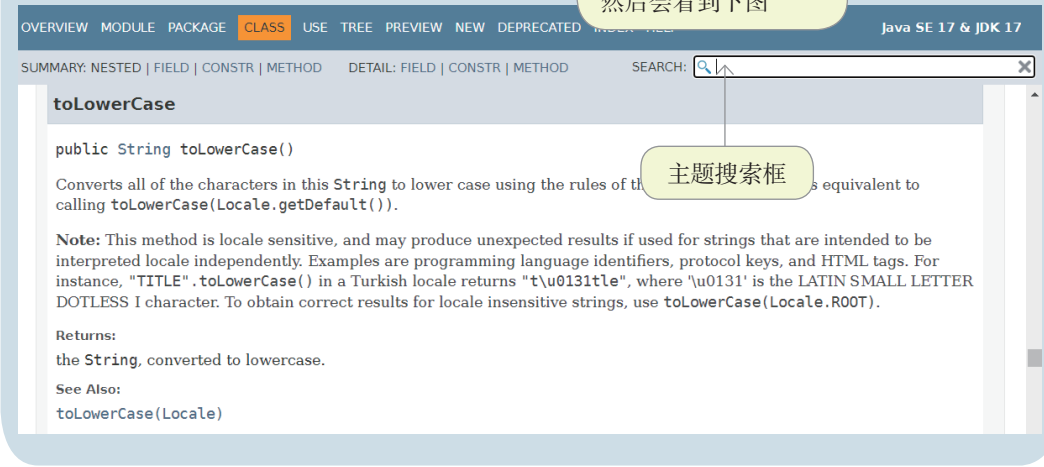


下图所示的是查找 toLowerCase 方法的过程。



第一步，在主题搜索框中输入 toLowerCase 关键字

第二步，在下拉列表中选择搜索到的内容，然后会看到下图



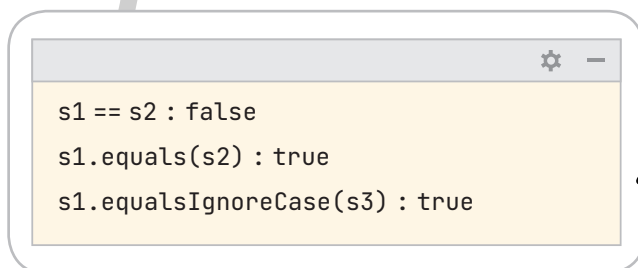
8.2.3 字符串比较



比较字符串相等的示例代码如下：

```
public class HelloWorld {
    public static void main(String[] args) {
        String s1 = new String("Hello");
        String s2 = "Hello";
        // 比较字符串是否是相同的引用
        System.out.println("s1 == s2 : " + (s1 == s2));
        // 比较字符串内容是否相等
        System.out.println("s1.equals(s2) : " + (s1.equals(s2)));

        String s3 = "HELlo";
        // 忽略大小写，比较字符串内容是否相等
        System.out.println("s1.equalsIgnoreCase(s3) : " + (s1.equalsIgnoreCase(s3)));
    }
}
```



字符串比较是常见操作,其中包括比较相等、比较前缀和后缀等。

1 比较相等

String 提供的比较字符串相等的方法如下。

- `boolean equals(Object anObject)`: 比较两个字符串中的内容是否相等。
- `boolean equalsIgnoreCase(String anotherString)`: 类似 `equals` 方法,只是忽略大小写。





2 比较前缀和后缀

比较前缀和后缀的方法如下。

- `boolean endsWith(String suffix)`: 测试此字符串是否以指定的后缀结束。
- `boolean startsWith(String prefix)`: 测试此字符串是否以指定的前缀开始。

字符串比较前缀和后缀的示例代码如下:

```
public class HelloWorld {  
    public static void main(String[] args) {  
  
        // 判断文件夹中的文件名  
        String[] docFolder = {"java.docx",  
                               " JavaBean.docx",  
                               "Objecitive-C.xlsx",  
                               "Swift.docx "};  
  
        int wordDocCount = 0;  
        // 查找文件夹中 Word 文档的个数  
        for (String doc : docFolder) {  
            // 去掉前后空格  
            doc = doc.trim();  
            // 比较后缀是否有 .docx 字符串  
            if (doc.endsWith(".docx")) {  
                wordDocCount++;  
            }  
        }  
  
        System.out.println(" 文件夹中 Word 文档的个数是:  " + wordDocCount);  
  
        int javaDocCount = 0;  
        // 查找文件夹中 Java 相关文档的个数  
        for (String doc : docFolder) {
```

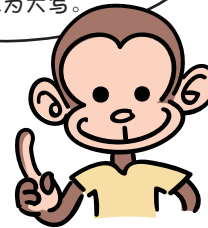
`trim` 方法可以去除字符串前后空白（包括空格、换行、回车和制表符等）

```

// 将字符串的字母全部转成小写
doc = doc.toLowerCase();
// 比较前缀是否有 java 字符串
if (doc.startsWith("java")) {
    javaDocCount++;
}
}
System.out.println(" 文件夹中 Java 相关文档的个数是: " + javaDocCount);
}
}

```

示例中的 toLowerCase 方法用于将字符串中的字母全部转化为小写。类似的方法还有 toUpperCase 方法, 它可将字符串中的字母全部转化为大写。



文件夹中 Word 文档的个数是: 3
文件夹中 Java 相关文档的个数是: 2

8.2.4 字符串截取

在Java中, 字符串截取方法是substring, 它有几个重载方法, 如右边所示。



- `String substring(int beginIndex)`: 从指定索引 `beginIndex` 开始到字符串结束处的字符。
- `String substring(int beginIndex, int endIndex)`: 从指定索引 `beginIndex` 开始截取直到索引 `endIndex - 1` 处的字符, 注意包括索引为 `beginIndex` 处的字符, 但不包括索引为 `endIndex` 处的字符。

字符串截取方法的示例代码如下：

```
public class HelloWorld {
    public static void main(String[] args) {

        String s = "Hello!";

        String subStr1 = s.substring(1); // 返回字符串 ello!
        String subStr2 = s.substring(2, 4); // 返回字符串 ll

        System.out.printf("subStr1 = %s\n", subStr1);
        System.out.printf("subStr2 = %s\n", subStr2);

    }
}
```

从索引 1 位置开始直到字符串结尾

输出结果
看这里。

```
subStr1 = ello!
subStr2 = ll
```

8.3 可变字符串

牛哥，既然有不可变字符串，那么是否也有可变字符串呢？

当然有。可变字符串在追加、删除、修改、插入和拼接等操作时不会产生新的对象。

Java 提供了两个可变字符串类：StringBuffer 和 StringBuilder。StringBuffer 是线程安全的，它的方法支持线程同步。当多个线程同时调用该方法时，这些线程按照一定的顺序串行执行。但在单线程环境下效率会比较低。StringBuilder 是 StringBuffer 的单线程版本，是在 Java 5 之后发布的，它不是线程安全的，但它的执行效率很高。

StringBuffer 和 StringBuilder 具有完全相同的 API，即构造方法和普通方法等内容一样。它们常用的构造方法如下。

- `StringBuilder()`：创建字符串内容是空的 `StringBuilder` 对象。
- `StringBuilder(String str)`：构造一个字符串生成器，并初始化为指定的字符串内容。

8.3.1 字符串追加

字符串追加方法是 `append`。它有很多重载方法, 可以追加任何类型的数据, 它的返回值还是可变字符串本身。

可变字符串追加的示例代码如下:

```
public class HelloWorld {  
    public static void main(String[] args) {  
  
        // 创建可变字符串对象  
        StringBuilder sbuilder1 = new StringBuilder("Hello");  
        sbuilder1.append(" ").append("World");  
        sbuilder1.append('.');  
        System.out.printf("sbuilder1 = %s\n", sbuilder1);  
  
        // 创建可变字符串对象  
        StringBuilder sbuilder2 = new StringBuilder();  
        Object obj = null;  
        // 连续多次追加数据  
        sbuilder2.append(false).append('\t').append(obj);  
        System.out.printf("sbuilder2 = %s\n", sbuilder2);  
  
        // 创建可变字符串对象  
        StringBuilder sbuilder3 = new StringBuilder();  
        for (int i = 0; i < 10; i++) {  
            sbuilder3.append(i);           // 追加整数  
        }  
        System.out.printf("sbuilder3 = %s\n", sbuilder3);  
    }  
}
```

由于 `append` 方法的返回值还是 `StringBuilder` 对象本身, 所以我们可以连续调用 `append` 方法追加数据

追加布尔值 `false`, 它会转换为 `false` 字符串

追加空对象 `null`, `null` 会转换为 `"null"` 字符串

追加一个制表符

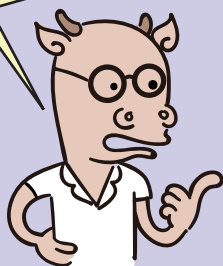
输出结果
看这里。

```
sbuilder1 = Hello World.  
sbuilder2 = false null  
sbuilder3 = 0123456789
```



8.3.2 字符串插入、删除和替换

可变字符串 `StringBuilder` 和 `StringBuffer` 都有实现插入、删除和替换等操作的常用方法，如右侧所示。



- `insert(int offset, String str)`: 在字符串中索引为 `offset` 的字符位置之前插入 `str`。`insert` 有很多重载方法，可以插入任何类型数据。
- `delete(int start, int end)`: 在字符串中删除子字符串，要删除的子字符串从指定索引 `start` 开始直到索引 `end - 1` 处的字符。`start` 和 `end` 两个参数与 `substring(int beginIndex, int endIndex)` 方法中两个参数的含义一样。
- `replace(int start, int end, String str)`: 用 `str` 替换子字符串中从指定索引 `start` 开始直到索引 `end - 1` 处的字符。`start` 和 `end` 同 `delete(int start, int end)` 方法中的两个参数。

示例代码如下：

```
public class HelloWorld {  
    public static void main(String[] args) {  
        // 原始不可变字符串  
        String str1 = "Hello!";  
        // 通过不可变字符串创建可变字符串对象  
        StringBuilder mstr = new StringBuilder(str1);  
  
        // 插入字符串  
        mstr.insert(6, " Java");  
        System.out.println(mstr);  
  
        // 具有追加效果的插入字符串  
        mstr.insert(mstr.length(), " Python");  
        System.out.println(mstr);  
    }  
}
```

在索引 6 位置之前插入字符串 " Java"。原始字符串索引如下图所示。

0	1	2	3	4	5
H	e	l	l	o	!

```
// 追加字符串
mstr.append(" and C++");
System.out.println(mstr);

// 删除字符串
mstr.delete(11, 26); ←
System.out.println(mstr);
mstr.replace(6, 11, "C++"); ←
System.out.println(mstr);

}
}
```

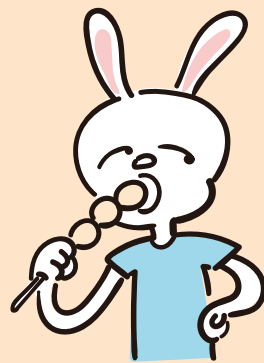
替代 6 到 11 之间字符串（即 Java）为 C++

删除字符串之前的字符串如下图所示。

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
H	e	l	l	o	!		J	a	v	a		P	y	t	h	o	n		a	n	d		c	+	+

输出结果
看这里。

Hello! Java
Hello! Java Python
Hello! Java Python and C++
Hello! Java
Hello!C++



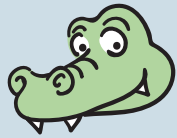


小结



本章主要介绍了 Java 中的字符串数据类型，读者需要重点掌握字符串表示方式以及不可变字符串的相关操作，如字符串拼接、查找、比较和截取等操作。读者还需要了解可变字符串的相关操作，如字符串的追加、插入、删除和替换等操作。

实践一下



你是哪儿的人？

我们的身份证号码是 18 位的数字，其中前两位代表省（自治区或直辖市），例如：山西是 14、辽宁是 21。请编写一个 Java 应用程序，输入某人的身份证号码，判断他是否为辽宁人。

同步练习题

1. 选择题

下列选项中哪些是可变字符串？（ ）

- A. String B. StringBuilder C. StringBuffer D. string

2. 选择题

下列选项中哪些是不可变字符串？（ ）

- A. String B. StringBuilder C. StringBuffer D. string

3. 选择题

下列选项中哪个正确表示 Java 字符串？（ ）

- A. "a" B. 'a' C. " 你好 " D. 'Hello'

4. 判断对错

String 对象在进行拼接时，会创建新的 String 对象。