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

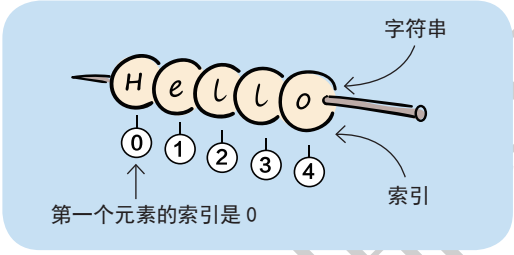
关东升

目 录

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。

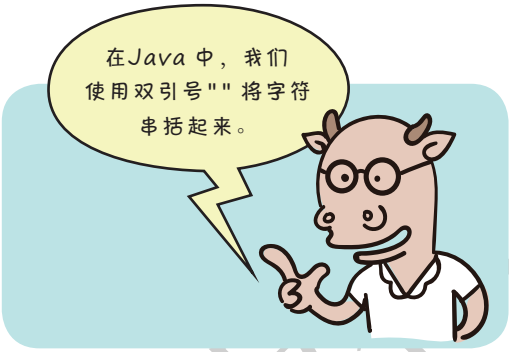


本章中我们介绍的内容如下：

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



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);

    }

}
```

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。

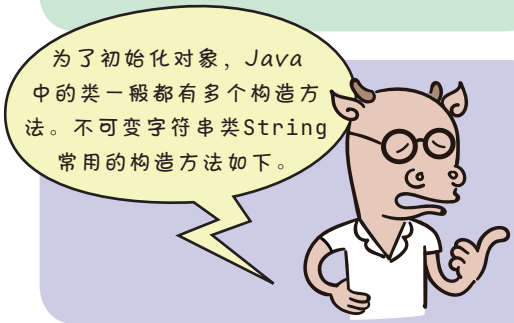
小贴士

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。

- 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。



- 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。

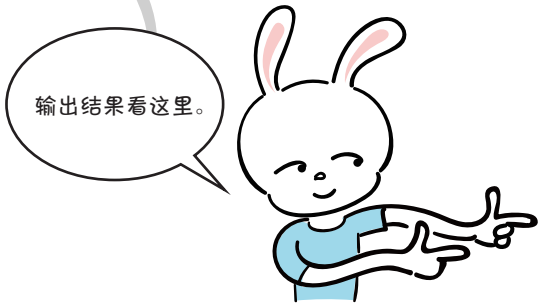
创建不可变字符串对象的示例代码如下：

```
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

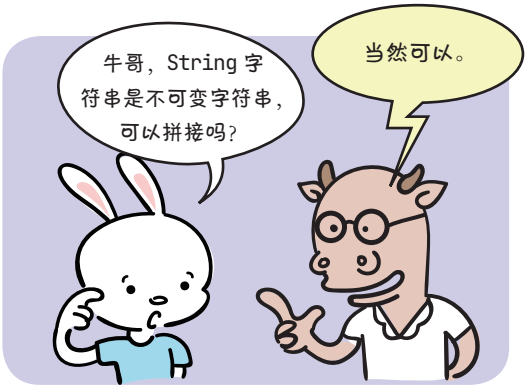
调用构造方法初始化对象

获得字符串长度的方法是 length()




```
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

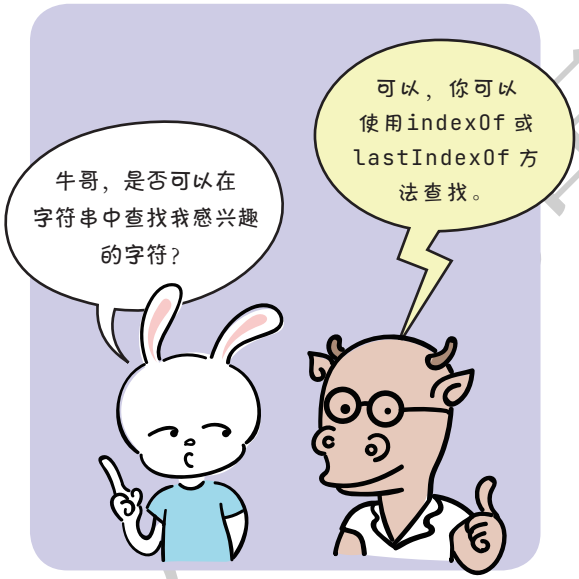
采用 concat 方法进行拼接，并且可以连续调用该方法

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

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题，可以通过主题框搜索感兴趣的主题。

8.2.3　字符串比较

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。

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)));
    }
}
```

```
s1 == s2 : false
s1.equals(s2) : true
s1.equalsIgnoreCase(s3) : true
```

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。

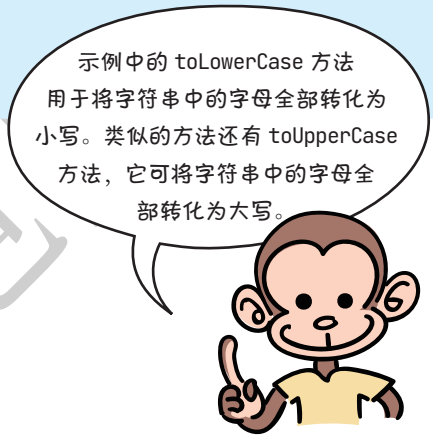
字符串比较前缀和后缀的示例代码如下：

```
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",
                "Objecitve-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) {
```

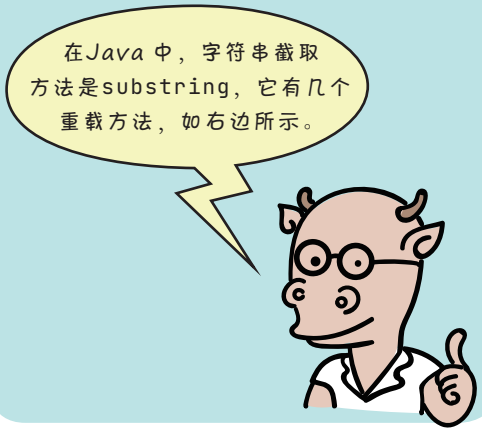
trim 方法可以去除字符串前后空白（包括空格、换行、回车和制表符等）

```
            // 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);

    }
}
```

```
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

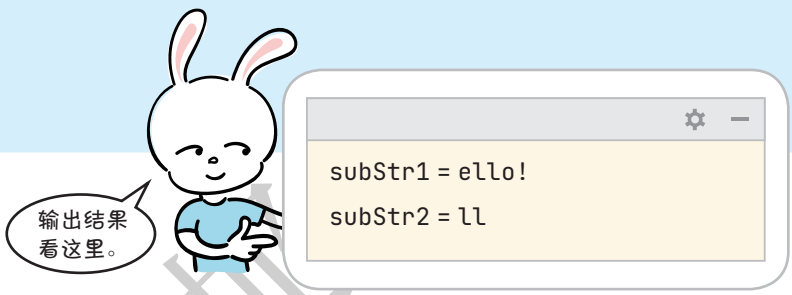


- `String substring(int beginIndex)`：从指定索引 `beginIndex` 开始到字符串结束处的字符。
- `String substring(int beginIndex, int endIndex)`：从指定索引 `beginIndex` 开始截取直到索引 `endIndex - 1` 处的字符，注意包括索引为 `beginIndex` 处的字符，但不包括索引为 `endIndex` 处的字符。

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



8.3 可变字符串

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：

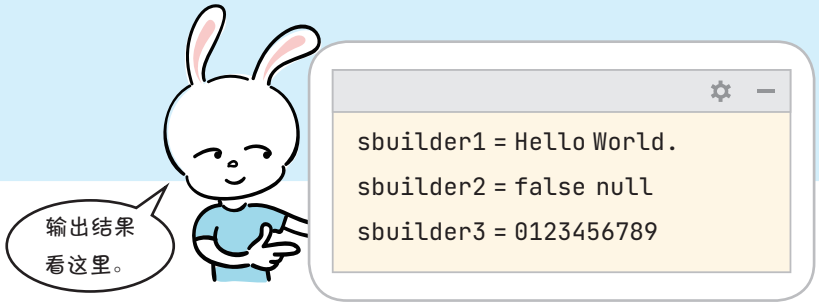
```
public class HelloWorld {
    public static void main(String[] args) {

        // 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



8.3.2 字符串插入、删除和替换

- 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。

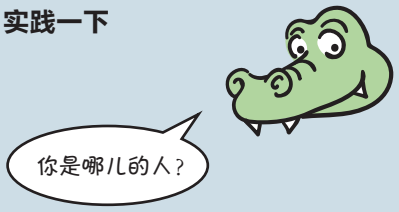
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
H	e	l	l	o	!		J	a	v	a		P	y	t	h	o	n		a	n	d		c	+	+


```
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



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。