

JC2002 Java 程序设计

第 11 天：字符串和集合（人工智能、计算机科学与技术）

11 月 15 日星期三/11 月 16 日星期四

JC2002 Java 程序设计

第 11 天，第 1 课时：字符串和基本字符串操作

参考文献和学习对象

- 今天的课程主要基于 **Java：如何编程**、第 14 章和 **Java 简要介绍**
- 今天的课程结束后，您应该能够
 - 在 Java 程序中使用字符串、字符串构建器和字符串操作
 - 使用基本的正则表达式 (RegExp) 操作
 - 在 Java 程序中使用列表和迭代器实现集合

弦乐入门

- 在许多 Java 程序中，字符串和字符串操作是必不可少的
 - 字符串类的实例代表一个字符串，即一个字符序列
 - 字符串类提供了几种创建和操作字符串的方法：在前面的课程中，我们已经使用了一些基本的字符串操作。
- 字符串对象是不可变的
 - 字符串对象的内容在创建后无法更改；这就是为什么许多字符串方法实际上是创建了一个被操作的原始字符串副本的原因

字符串构造函数 (1)

- 类中初始化字符串对象的构造函数。
各种不同的方式：
 - 在没有参数的情况下，会创建一个空字符串；但是，由于字符串是不可变的，空字符串通常没有价值：

```
s0 = ""
```

字符串 `s0 = 新字符串()`；

- 您可以使用常量字符串作为参数：

```
s1 = "你好"
```

字符串 `s1 = 新字符串 ("hello")` ；

- 您可以使用字符串对象作为参数来创建副本：

字符串 `s2 = 新字符串 (s1)` ；

```
s2 = "hello"
```

字符串构造函数 (2)

- 字符串类还提供了可接受字符或字节的构造函数
数组作为参数：

```
char[] charArray = {'b','i','r','t','h','','d','a','y'};
```

```
String s3 = new String(charArray);
```

字符串 `s4 = new String(charArray, 6, 3)` ;

`s3 = "出生日期"`

`s4 = "日"`

要访问的字符数
(计数

访问数组中字符的起始位
置 (偏移量

将字符串初始化为字面量

- 在 Java 中，也可以通过为 *字符串* 赋值来创建 *字符串* 对象。
字面意思，不含关键字 new：

字符串 `s = "hello"`；

- 需要注意的是，关键字 new *总是* 创建一个新的字符串对象，而通过字面量创建的字符串将引用现有对象，如果字符串字面量库中已经存在类似字符串的话。
 - 由于字符串对象是不可变的，因此两者之间的区别实际上是无关紧要

字符串字面量示例 (1)

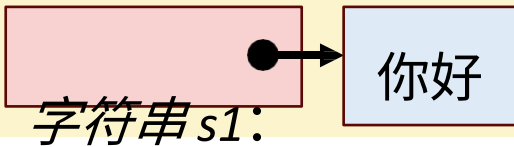
```
1 public class StringLiteralExample {  
2     public static void main(String[] args){  
3         字符串 s1 = "hello";  
4         字符串 s2 = 新字符串 ("hello") ; 字  
5         符串 s3 = "hello";  
6  
7         System.out.println("s1 和 s2 是否相同? " + (s1 == s2));  
8  
9         System.out.println("s1 和 s3 是否相同? " + (s1 == s3));  
10        System.out.println("s2 和 s3 相同吗? " + (s2 == s3));  
$ java StringLiteralExample  
}
```

内存

字符串字面量示例 (2)

```
1 public class StringLiteralExample {
2     public static void main(String[] args){
3         字符串 s1 = "hello";
4         字符串 s2 = 新字符串 ("hello") ; 字
5         符串 s3 = "hello";
6
7         System.out.println("s1 和 s2 是否相同? " + (s1 == s2));
8
9         System.out.println("s1 和 s3 是否相同? " + (s1 == s3));
10        System.out.println("s2 和 s3 相同吗? " + (s2 == s3));
11    }
12 }
$ java StringLiteralExample
```

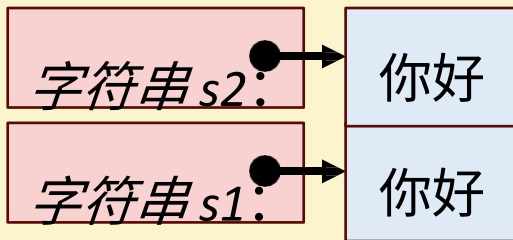
内存



字符串字面量示例 (3)

```
1 public class StringLiteralExample {
2     public static void main(String[] args){
3         字符串 s1 = "hello";
4         字符串 s2 = 新字符串 ("hello") ; 字
5         符串 s3 = "hello";
6
7         System.out.println("s1 和 s2 是否相同? " + (s1 == s2));
8
9         System.out.println("s1 和 s3 是否相同? " + (s1 == s3));
10        System.out.println("s2 和 s3 相同吗? " + (s2 == s3));
$ java StringLiteralExample
}
```

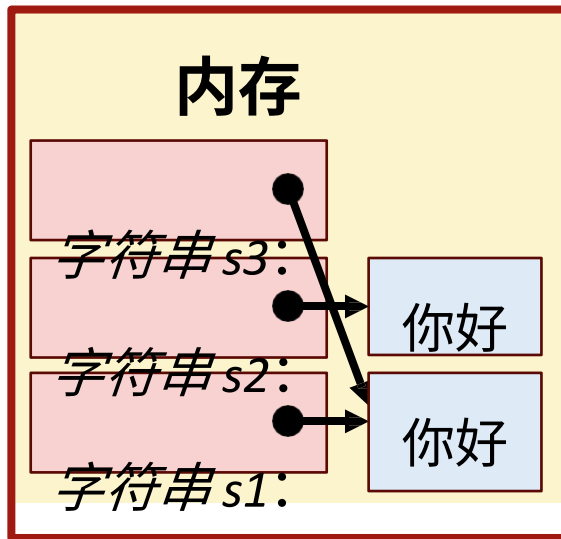
内存



字符串字面量示例 (3)

```
1 public class StringLiteralExample {
2     public static void main(String[] args){
3         字符串 s1 = "hello";
4         字符串 s2 = 新字符串 ("hello") ; 字
5         符串 s3 = "hello";
6
7         System.out.println("s1 和 s2 是否相同? " + (s1 == s2));
8
9         System.out.println("s1 和 s3 是否相同? " + (s1 == s3));
10        System.out.println("s2 和 s3 相同吗? " + (s2 == s3));
11    }
12 }
```

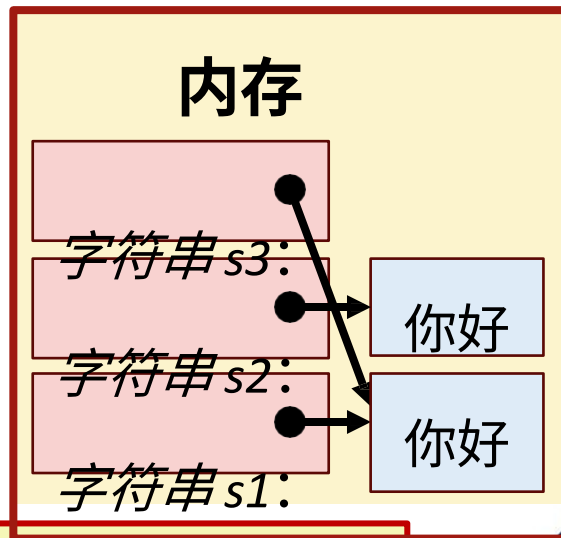
\$ java StringLiteralExample



字符串字面量示例 (4)

```
1 公共类 StringLiteralExample {  
2      public static void main(String[] args){  
3          字符串 s1 = "hello";  
4          字符串 s2 = 新字符串 ("hello") ;  
5          字符串 s3 = "hello";  
6          System.out.println ("s1 和 s2 是否相同? " + (s1 == s2));  
7          System.out.println ("s1 和 s3 是否相同? " + (s1 == s3));
```

```
$ java StringLiteralExample  
s1 和 s2 是否相同? 是  
s1 和 s3 是否相同? 是  
s2
```



请注意，比较 (==) 适用于指针，而不是它们指向的数据！

基本字符串方法

- 字符串的基本方法包括以下几种：
 - **int length()**: 返回长度（字符串中的字符数）
 - **charAt(pos)**: 返回参数 pos 中指定位置的字符（注意第一个字符位于 0 位置）。
 - **void getChars(begin,end,dest,destBeg)**: 复制来自于字符串到字符数组
 - int beg: 复制开始的索引（位置
 - int end: 最后一个要复制的字符旁边的索引（位置

- `char[] dest`: 复制字符的字符数组
- `int destBeg`: 开始复制的 `dest` 中的索引（位置）。

基本字符串方法示例 (1)

```
1  公共类 BasicStringExample {  
2  
3      public static void main(String[] args){  
4          String str = "hello world!";  
5          for(int i=str.length()-1; i >= 0; i--){  
6              System.out.printf("%c", str.charAt(i))  
7              ;  
8          }  
9  
10         System.out.println();  
11         char[] charArr = new char[5];  
12         str.getChars(6,11,charArr,0);  
        System.out.println(charArr);  
    }
```


基本字符串方法示例 (2)

```
1  公共类 BasicStringExample {  
2      public static void main(String[] args){  
3  
4          String str = "hello world!";  
5          for(int i=str.length()-1; i >= 0; i--){  
6              System.out.printf("%c", str.charAt(i))  
7          }  
8          ;  
9      }  
10  
11      System.out.println();  
12      char[] charArr = new char[5];  
13      str.getChars(6,11,charArr,0);  
14      System.out.println(charArr);  
15  }
```

```
$ java BasicStringExample  
!dlrow olleh
```

反向循环字符并逐个打印。注意，
最后一个字符位于索引位置
length()-1。

基本字符串方法示例 (3)

```
1  公共类 BasicStringExample {  
2      public static void main(String[] args){  
3          String str = "hello world!";  
4          for(int i=str.length()-1; i >= 0; i--){  
5              System.out.printf("%c", str.charAt(i))  
6              ;  
7          }  
8      }  
9      System.out.println();  
10     char[] charArr = new char[5];  
11     str.getChars(6,11,charArr,0);  
12     System.out.println(charArr);  
}
```

```
$ java BasicStringExample  
世界  
$
```

将字符串中索引 6 至
索引 10 位置的字符复
制到字符数组 charArr
中。

比较字符串

- 请注意，Java 比较运算符 `==` 会比较引用、而不是字符串的内容
- 对于比较两个字符串的内容，方法 `equals()` 和可使用 `compareTo()`
 - 方法 `equals()` 如果参数字符串包含相同的该对象的字符序列
 - 方法 `equalsIgnoreCase()` 与 `equals()` 类似，但忽略大小写
 - 方法 `compareTo()` 返回一个负整数，如果该字符串按词典顺序（字母顺序）排在参数字符串之前；如果字符串相等，则返回 0；如果该字

字符串按词典顺序排在参数字符串之后，则返回正整数。

字符串比较示例

```
1  公共类 BasicStringComparisonExample {
2      public static void main(String[] args){
3          字符串 s1 = "albert";
4          字符串 s2 = "Albert";
5          字符串 s3 = "Bertha";
6          System.out.printf("%s equals %s: %b\n", s1, s2, s1.equals(s2));
7          System.out.printf("%s equalsIgnoreCase %s: %b\n", s1, s2,
8              s1.equalsIgnoreCase(s2));
9          System.out.printf("%s compareTo %s: %d\n", s1, s2, s2.compareTo(s3));
10         System.out.printf("%s compareTo %s: %d\n", s1, s3, s3.compareTo(s2));
11     }
12 }
```

S:\java\BasicStringComparisonExample

比较字符串区域（子串）

- 对于比较字符串 区域 而非完整字符串，方法可使用 **regionMatches()**
 - 如果指定区域中的子字符串相等，则返回 true
- 两个版本，有四个或五个参数：
 - 方法 `regionMatches(off1,str2,off2,len)` 如果从本字符串中 `off1` 位置开始的长度为 `len` 的子字符串与参数字符串 `str2` 中 `off2` 位置开始的长度为 `len` 的子字符串相等，则返回 true
 - 在区域匹配 (`ignoreCase,off1,str2,off2,len`) 方法中，还

有一个布尔类型的第一参数，用于确定是否忽略大小写。

从字符串中提取子串

- 提取子串时，可使用方法 **substring()**
 - 返回一个新的字符串对象。
字符串对象
- 两个版本，有一个或两个参数：
 - 方法 `substring(start)` 返回从以字符串最后一个字符为起点和终点的位置索引
 - 方法 `substring(start,end)` 返回从位置索引 `start` 开始到位置索引 `end`（不包括）的子串。

使用 regionMatches() 和 substring()

```
1  公共类 RegionMatchesExample {  
2      public static void main(String[] args){  
3          字符串 s1 = "Hello World!";  
4          字符串 s2 = "世界早安! ";  
5          System.out.println ("区域匹配: " + " ) 。  
6              s1.regionMatches(6,s2,13,6));  
7          System.out.println("Regions matching with case ignored: " +  
8              s1.regionMatches(true,6,s2,13,6));  
9              "s1 从 6 到末尾的子串: "    s1.substring(6))  
10             "s2 从 13 到 17 的子串: "    s2.substring(13,18))  
11  
12
```


有问题或意见？

JC2002 Java 程序设计

第 11 天，第 2 课时：高级字符串和字符运算

字符串标记化

- 将长字符串分解成较小的片段或*标记*，例如从句子中提取单个单词
- 这一过程称为*标记化*
- String 类的 **split()** 方法可将一个字符串分解为多个部分（标记）。
 - 标记之间由分隔符分隔，分隔符通常是空白字符，如*空格*、*制表符*、*换行符*和*回车符*。

- 其他字符也可用作分隔符来分隔标记
- `split()` 的参数包括定界 *正则表达式* 和可选的代币数量最大限制

标记化示例 1

```
1  import java.util.Scanner;
2
3  公共类 TokenizingExample1 {
4      public static void main(String[] args){
5
6          Scanner scanner = new Scanner(System.in);
7          System.out.println("Enter a sentence and press Enter");
8          String sentence = scanner.nextLine();
9          String[] tokens = sentence.split(" ");
10         System.out.printf("Number of tokens: %d\n", tokens.length);
11         System.out.println("The tokens are:");
12         for (String token : tokens) {
13             System.out.println(token);
14         }
15     }
16 }
```

分隔符为空格 " "

标记化示例 2

```
1 import java.util.Scanner;
2
3 公共类 TokenizingExample2 {
4     public static void main(String[] args){
5
6         Scanner scanner = new Scanner(System.in);
7         System.out.println ("输入您的电子邮件地址: ");
8         String sentence = scanner.nextLine();
9         String[] tokens = sentence.split("@",2);
10        System.out.printf("Your user name is: %s\n", tokens[0]);
11        System.out.printf("Your URL is: %s\n", tokens[1]);
12    }
13 }
```

分隔符为"@”，最多提取 2 个标记符

```
$ java TokenizingExample2
```

```
输入您的电子邮件地址: jamesbond007@abdn.ac.uk
```

```
您的用户名是: jamesbond007 您的 URL
```


串联字符串

- String 的方法 **concat()** 可用来将两个 String 对象连接成一个新的 String 对象，其中包含两个字符串中的字符
 - 语法：s1.concat(s2) 将连接字符串对象 s1 和 s2 使 s2 出现在 s1 之后
- 在 Java 中，用于字符串对象的加法运算符 + 被定义为执行连接
 - 假设 s1、s2 和 s3 都是字符串对象：

s1+s2 等于 s1.concat(s2)

$s_1 + s_2 + s_3$ 等于 $s_1.concat(s_2).concat(s_3)$

连接示例

```
1 public class ConcatenationExample {  
2     public static void main(String[] args){  
3         字符串 s1 = "好"; 字符串 s2  
4         = "早"; 字符串 s3 = "世界! "  
5  
6         ;  
7  
8         System.out.println(s1+s2+s3);  
9         System.out.println(s1.concat(s2).concat(s3));  
10  
11        int age = 18;  
  
        System.out.println("Michael is " + age + " years old");  
    }  
}
```

```
$ java ConcatenationExample 世  
界早安!  
世界早安  
迈克尔 18 岁
```

请注意，如果数据类型不是字符串对象，Java 会在
使用操作符 + 时自动将其转换为字符串表示法，但在

使用 `StringBuilder` 生成可修改字符串

- 在经常执行字符串连接或其他字符串修改的程序中，使用类 **`StringBuilder`** 而不是类 `String` 通常会更有效率。
 - `StringBuilder` 是 `String` 的 "可修改" 版本：它提供了 **`append()`**、**`insert()`** 和 **`delete()`** 等方法来修改它所包含的字符串的内容。
 - 修改 `StringBuilder` 对象时，它不会返回一个新的对象，但会更改原始对象的内容

StringBuilder 构造函数

- StringBuilder 类提供了几个不同的构造函数：
 - **StringBuilder()**: 构造一个不含字符和初始容量为 16 个字符
 - **字符串生成器 (字符串序列 seq)**: 构造一个字符串生成器，其中包含与 CharSequence 对象 seq 相同的字符
 - **StringBuilder(int 容量)**: 构造一个没有字符的字符串生成器，初始容量由容量参数指定

字符串参数 str

StringBuilder 方法

- 以下是一些最基本的 StringBuilder 类：
 - **length()**、**setLength(int 长度)**：返回长度（字符计数），并分别设置长度
 - **capacity()** 和 **ensureCapacity()**：分别返回当前容量，如果低于指定的最小容量，则增加容量
 - **charAt()**、**setCharAt()**、**getChars()**：获取和设置指定位置的字符
 - **append()**、**insert()**、**delete()**、**deleteCharAt()**：通过追加、插入

和删除内容来修改字符串生成器；这些方法有多个重载版本，以支持不同的数据类型

字符串生成器示例 (1)

```
1  公共类 StringBuilderExample1 {
2      public static void main(String[] args){
3          StringBuilder sb = new StringBuilder("Good morning world!");
4          System.out.printf("Buffer = %s | length = %d | capacity = %d%n",
5              sb.toString(), sb.length(), sb.capacity());
6          sb.ensureCapacity(75);
7          System.out.printf ("新容量 = %d%n", sb.capacity()) ;
8          sb.setLength(10);
9          System.out.printf("New buffer = %s | new length = %d%n",
10             sb.toString(), sb.length());
11     }
```

字符串生成器示例 (2)

```
1  公共类 StringBuilderExample1 {  
2      public static void main(String[] args){  
3          StringBuilder sb = new StringBuilder("Good morning world!");  
4          System.out.printf("Buffer = %s | length = %d | capacity = %d%n",  
5              sb.toString(), sb.length(), sb.capacity());  
6          sb.ensureCapacity(75);  
7          System.out.printf("新容量 = %d%n", sb.capacity());  
8          sb.setLength(10);  
9          System.out.printf("New buffer = %s | new length = %d%n",  
10             sb.toString(), sb.length());  
11     }  
12 }
```


字符串生成器示例 (3)

```
1  公共类 StringBuilderExample1 {
2      public static void main(String[] args){
3          StringBuilder sb = new StringBuilder("Good morning world!");
4          System.out.printf("Buffer = %s | length = %d | capacity = %d%n",
5              sb.toString(), sb.length(), sb.capacity());
6          sb.ensureCapacity(75);
7          System.out.printf ("新容量 = %d%n", sb.capacity());
8          sb.setLength(10);
9          System.out.printf ("New buffer = %s | new length = %d%n",
10             sb.toString(), sb.length());
11     }
```


字符串生成器示例 (4)

```
1  公共类 StringBuilderExample1 {
2      public static void main(String[] args){
3          StringBuilder sb = new StringBuilder("Good morning world!");
4          System.out.printf("Buffer = %s | length = %d | capacity = %d%n",
5              sb.toString(),sb.length(),sb.capacity());
6          sb.ensureCapacity(75);
7          System.out.printf ("新容量 = %d%n", sb.capacity()) ;
8          sb.setLength(10);
9          System.out.printf ("New buffer = %s | new length = %d%n",
10             sb.toString(), sb.length());
11     }
```


字符串生成器示例 2 (1)

```
1  公共类 StringBuilderExample2 {
2      public static void main(String[] args){
3          StringBuilder sb = new StringBuilder("Good morning world!");
4          System.out.printf ("缓冲区 = %s\n", sb.toString());
5          char[] charArray = new char[7];
6          sb.getChars(5, 12, charArray, 0);
7          System.out.printf("Char array = ");
8          for (char character : charArray) {System.out.print(character);}
9          sb.setCharAt (5, 'M') ;
10         sb.setCharAt (13, 'W') ;
11         System.out.printf("%nNew Buffer = %s\n", sb.toString());
12         sb.deleteCharAt(sb.length()-1);
13         sb.append("再来一次! ");
```

```
14     System.out.printf("New Buffer = %s%n", sb.toString());
15 }
16 }
```

字符串生成器示例 2 (2)

```
1  公共类 StringBuilderExample2 {  
2      public static void main(String[] args){  
3          StringBuilder sb = new StringBuilder("Good morning world!");  
4          System.out.printf ("缓冲区 = %s\n", sb.toString());  
5          char[] charArray = new char[7];  
6          sb.getChars(5, 12, charArray, 0);  
7          System.out.printf("Char array = ");  
8          for (char character : charArray) {System.out.print(character);}  
9          sb.setCharAt (5, 'M') ;  
10         sb.setCharAt (13, 'W') ;  
11         System.out.print("%nNew Buffer = %s\n"  
12         sb.deleteCharAt (sb.length()-1);  
13         sb.append("再来一次! ");
```

```
14     System.out.printf("New Buffer = %s%n", sb.toString());
15 }
16 }
```

```
$ java StringBuilderExample2
Buffer = 早上好, 世界!
```

字符串生成器示例 2 (3)

```
1  公共类 StringBuilderExample2 {
2      public static void main(String[] args){
3          StringBuilder sb = new StringBuilder("Good morning world!");
4          System.out.printf ("缓冲区 = %s%n", sb.toString()) ;
5          char[] charArray = new char[7];
6          sb.getChars(5, 12, charArray, 0);
7          System.out.printf("Char array = ");
8          for (char character : charArray) {System.out.print(character);}
9          sb.setCharAt (5, 'M') ;
10         sb.setCharAt (13, 'W') ;
11         System.out.print("%nNew Buffer = %s%n"
12         sb.deleteCharAt(-1);
13         sb.append("再来一次! ");
```

```
14      System.out.printf("New Buffer = %s\n", sb.toString());
15  }
16 }
```

```
$ java StringBuilderExample2
Buffer = 早安世界! Char array
= morning
```


字符串生成器示例 2 (4)

```
1  公共类 StringBuilderExample2 {
2      public static void main(String[] args){
3          StringBuilder sb = new StringBuilder("Good morning world!");
4          System.out.printf ("缓冲区 = %s%n", sb.toString());
5          char[] charArray = new char[7];
6          sb.getChars(5, 12, charArray, 0);
7          System.out.printf("Char array = ");
8          for (char character : charArray) {System.out.print(character);}
9          sb.setCharAt (5, 'M') ;
10         sb.setCharAt (13, 'W') ;
11         System.out.printf ("%nNew Buffer = %s%n" sb.toString());
12         sb.deleteCharAt (-1);
13         sb.append("再来一次! ");
```

```
14     System.out.printf("New Buffer = %s%n", sb.toString());
15 }
16 }
```

```
$ java StringBuilderExample2
```

```
Buffer = 早安世界! Char array
```

```
= morning
```

```
新缓冲区 = 早安世界
```

字符串生成器示例 2 (5)

```
1  公共类 StringBuilderExample2 {  
2      public static void main(String[] args){  
3          StringBuilder sb = new StringBuilder("Good morning world!");  
4          System.out.printf ("缓冲区 = %s%n", sb.toString());  
5          char[] charArray = new char[7];  
6          sb.getChars(5, 12, charArray, 0);  
7          System.out.printf("Char array = ");  
8          for (char character : charArray) {System.out.print(character);}  
9          sb.setCharAt (5, 'M') ;  
10         sb.setCharAt (13, 'W') ;  
11         System.out.print ("%nNew Buffer = %s%n"  
12         sb.deleteCharAt (sb.length() -1);  
13         sb.append("再来一次! ");
```

```
14     System.out.printf("New Buffer = %s\n", sb.toString());
15 }
16 }
```

```
$ java StringBuilderExample2
```

```
Buffer = 早安世界! Char array
```

```
= morning
```

```
新缓冲区 = 早安世界
```

```
新 Buffer = 早上好, 世界又回来了!
```

类 StringBuffer

- 使用 StringBuilder 类创建的字符串构建器 *不是线程安全的*：如果多个线程需要访问相同的动态（即可修改）字符串内容，请使用 **StringBuffer** 类而不是 StringBuilder 类。
 - StringBuilder 和 StringBuffer 这两个类提供了相同的功能，但只有 StringBuffer 是线程安全的（即同步）。
 - 如果不需要从多个线程访问同一个字符串生成器，那么 StringBuilder 类比 StringBuffer 更快、更有效。

封装类

- 在某些情况下，需要将基元类型值作为对象处理 (即参照类型值)
 - Java 为布尔、**字符**、**双倍**、浮点、字节、短和**整数**等原始类型分别提供了封装类
 - 将基元类型转换为包装类对象的推荐方法是静态方法 `valueOf()`:

```
int iPrim = 1; Integer i = Integer.valueOf(iPrim);
```

- 封装类也可以通过字面形式 (*自动框选*) 直接初始化:

字符 `c = 'A'`; 整数 `i = 5`;

字符类的方法

- 字符类方法可用于测试和操作单字值
 - 每个方法至少需要一个字符作为输入参数
 - *测试*字符的方法包括：**isDefined()**、**isDigit()**、**isJavaIdentifierStart()**、**isJavaIdentifierPart()**、**isLetter()**、**isLetterOrDigit()**、**isLowerCase()**和**isUpperCase()**
 - 操作字符的方法示例包括：**toUpperCase()** 返回字符的大写版本，**toLowerCase()** 返回字符的小写版本。

字符示例 (1)

```
1  公共类 CharacterExample {
2      public static void main(String[] args){
3          char c = 'a';
4          字符 c1 = 'A';
5          字符 c2 = Character.valueOf(c);
6          System.out.printf("c1 = %c | c2 = %s\n", c1, c2.toString());
7          System.out.printf("c1 和 c2 是否相等? %b\n", c1.equals(c2));
8          System.out.printf("c1 和 c2 在忽略大小写时是否相等? %b\n",
9                          c1.toString().equalsIgnoreCase(c2.toString()));
10         System.out.printf("'%c' is digit?%b\n", c1, Character.isDigit(c1));
11         System.out.printf("'%c' is letter?%b\n", c1, Character.isLetter(c1));
12         System.out.printf("'%c' is uppercase?%b\n", c1, Character.isUpperCase(c1));
13         System.out.printf("'%c' 是数字吗? %b\n", c2, Character.isUpperCase(c2));
14         System.out.printf("'%c' in uppercase is %c\n", c1, Character.toUpperCase(c1));
```

```
15     System.out.printf("'%c' in uppercase is %c%n",c2,Character.toUpperCase(c2));  
16 }  
17 }
```

字符示例 (2)

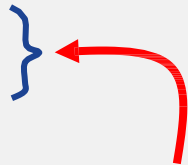
```
1  公共类 CharacterExample {  
2      public static void main(String[] args){  
3          char c = 'a'; 字符 c1  
4          = 'A';  
5          字符 c2 = Character.valueOf(c);  
6          System.out.printf("c1 = %c | c2 = %s\n", c1, c2.toString());  
7          System.out.printf("c1 和 c2 是否相等? %b\n", c1.equals(c2));  
8          System.out.printf("c1 和 c2 在忽略大小写时相等吗? %b\n",  
9              c1.toString().equalsIgnoreCase(c2.toString()));  
10         System.out.printf("'%c' 是数字? %b\n", c1, Character.isDigit(c1));  
11         System.out.printf("'%c' is letter? %b\n", c1, Character.isLetter(c1));  
12         System.out.printf("'%c' is uppercase? %b\n", c1, Character.isUpperCase(c1));  
13         System.out.printf("'%c' 是数字? %b\n", c2, Character.isUpperCase(c2));  
14         System.out.printf("'%c' 为大写字母是 %c\n", c1, Character.toUpperCase(c1));  
15         System.out.printf("'%c' 为大写字母是 %c\n", c2, Character.toUpperCase(c2));  
16  
17     }  
}
```

不同的初始化

}
}

字符示例 (3)

```
1 公共类 CharacterExample {
2
3      public static void main(String[] args){
4
5          char c = 'a'; 字符 c1
6
7          = 'A';
8          字符 c2 = Character.valueOf(c);
9          System.out.printf("c1 = %c | c2 = %s\n", c1, c2.toString());
10
11         System.out.printf("c1 和 c2 是否相等? %b\n", c1.equals(c2));
12
13         System.out.printf("c1 和 c2 在忽略大小写时相等吗? %b\n",
14                             c1.toString().equalsIgnoreCase(c2.toString()));
15
16         System.out.printf("'c' 是数字? %b\n", c1, Character.isDigit(c1));
17         System.out.printf("'c' is letter? %b\n", c1, Character.isLetter(c1));
18         System.out.printf("'c' is uppercase? %b\n", c1, Character.isUpperCase(c1));
19         System.out.printf("'c' is lowercase? %b\n", c1, Character.isLowerCase(c1));
20         System.out.printf("'c' 为大写字母是 %c\n", c1, Character.toUpperCase(c1));
21         System.out.printf("'c' 为小写字母是 %c\n", c1, Character.toLowerCase(c1));
22     }
```



不同的比较；注意

`equalsIgnoreCase()` 是

为字符串定义的，

字符示例 (4)

```
1 公共类 CharacterExample {
2      public static void main(String[] args){
3
4          char c = 'a'; 字符 c1
5
6          = 'A';
7          字符 c2 = Character.valueOf(c);
8          System.out.printf("c1 = %c | c2 = %s\n", c1, c2.toString());
9          System.out.printf("c1 和 c2 是否相等? %b\n", c1.equals(c2));
10
11         System.out.printf("c1 和 c2 在忽略大小写时相等吗? %b\n",
12                             c1.toString().equalsIgnoreCase(c2.toString()));
13
14         System.out.printf("'c' 是数字? %b\n", c1, Character.isDigit(c1));
15         System.out.printf("'c' is letter? %b\n", c1, Character.isLetter(c1));
16         System.out.print("'c' is uppercase? %b\n", Character.isUpperCase(c1));
17
18         System.out.printf("'c' 是数字? %b\n", c1, Character.isDigit(c1));
19         System.out.printf("'c' 是大写字母? %b\n", c1, Character.isUpperCase(c1));
20         System.out.printf("'c' 为大写字母是 %c\n", c1, Character.toUpperCase(c1));
21         System.out.printf("'c' 为大写字母是 %c\n", c2, Character.toUpperCase(c2));
```

基本测试

```
}  
}
```

...

A' 数字? 假 'A' 字母

? 真 'A' 大写? 真

a' 大写?

字符示例 (5)

```
1 公共类 CharacterExample {
2      public static void main(String[] args){
3
4          char c = 'a'; 字符 c1
5
6          = 'A';
7          字符 c2 = Character.valueOf(c);
8          System.out.printf("c1 = %c | c2 = %s\n", c1, c2.toString());
9          System.out.printf("c1 和 c2 是否相等? %b\n", c1.equals(c2));
10
11         System.out.printf("c1 和 c2 在忽略大小写时相等吗? %b\n",
12                             c1.toString().equalsIgnoreCase(c2.toString()));
13
14         System.out.printf("'%c' 是数字? %b\n", c1, Character.isDigit(c1));
15         System.out.printf("'%c' is letter? %b\n", c1, Character.isLetter(c1));
16         System.out.printf("'%c' is uppercase? %b\n", c1, Character.isUpperCase(c1));
17         System.out.printf("'%c' 是数字? %b\n", c2, Character.isDigit(c2));
18         System.out.printf("c2 为大写字母是 %b\n", c2, Character.isUpperCase(c2));
19         System.out.printf("c1 为大写字母是 %c\n", c1, Character.toUpperCase(c1));
20         System.out.printf("'%c' 为大写字母是 %c\n", c2, Character.toUpperCase(c2));
21     }
```

基本特征
操控

```
}  
}
```

```
....
```

大写的'A'是 A 大写的'a'是

A

\$

有问题或意见？

JC2002 Java 程序设计

第 11 天，第 3 课：正则表达式

正则表达式 (regex)

- 正则表达式 (regex) 是描述搜索的字符串用于匹配其他字符串中字符的模式
 - 此类表达式可用于验证输入，确保数据符合特定格式
- 正则表达式可用于执行所有类型的文本搜索和文本替换操作
- 例如，大型复杂的正则表达式可用于验证程序的语法
 - 如果程序代码与正则表达式不匹配，则编译器知道代码中存在语法错误

正则表达式字符

- 一个 regex 由 *字面字符*和*元字符*组成
 - 字面字符是具有字面意义的常规字符：例如
例如，字符 "b" 是与字符 "b" 匹配的字面字符
 - 元字符是在 regex 中具有特殊含义的字符：例如，元字符 "." （点）可与任何字符匹配。
 - 某些元字符前面有 *转义序列*（反斜杠）
例如，元字符 "\d" 与任何数字匹配
 - 反斜杠还用于区分字面字符和元字符：例如， "*" 是一个元字符，而 "*" 是一个与字符 "*" （星号）匹配的字面字符。

一些常见的元字符

元角色	说明
-	匹配任何字符（换行符除外）
^	匹配字符串中的起始位置
\$	匹配字符串的结束位置
*	与前一个元素匹配 0 次或更多次
?	与前一个元素匹配 0 次或 1 次
+	与前一个元素匹配一次或多次

I

匹配以"|"分隔的任意模式

使用元字符的示例

Regex	匹配和不匹配示例
博。	匹配 "box "和 "boy", 但不匹配 "but "或 "bo"
猫	匹配 "猫", 但不匹配 "一只猫"
帽子\$	匹配 "hat "和 "chat", 但不匹配 "hatch"
c*at	与 "at"、"cat "和 "ccat "匹配, 但不与 "chat "匹配
c?	匹配 "at "和 "cat", 但不匹配 "ccat"
c?	匹配 "cat "和 "ccat", 但不匹配 "at"

猫|狗

与 "猫 "和 "狗 "匹配，但与 "牛 "不匹配

一些常见的角色类别

- 字符类是继字面匹配之后最基本的 regex 概念
 - 字符类由与特定类型字符（如数字或空格）匹配的元字符定义
- 一些常用的角色类别示例：

人物	匹配	人物	匹配
<code>\d</code>	任何数字	<code>\D</code>	任何非数字
<code>\w</code>	任何单词字符	<code>\W</code>	任何非字字符

\s	任何空白字符	\S	任何非空格字符
\b	词语界限		

在 regex 中使用括号

- 括号 `[]` 用于匹配任何单个字符，这些字符是包含在括号内
 - 例如，`[abc]` 匹配 `"a"`、`"b"` 和 `"c"`，但不匹配 `"d"`。
- 在括号内，元字符 `"^"` 用于匹配字符不包含在括号内的
 - 例如，`[^ab]` 匹配 `"c"` 和 `"z"`，但不匹配 `"a"` 或 `"b"`。
- 在括号内，`"-"` 用于定义匹配一系列字符

- 例如，`[a-d]`匹配 `"a"`、`"b"`、`"c"`和 `"d"`，但不匹配 `"e"`。

量词

- Regex *数量词*用于指定要匹配的序列长度

量词	说明
n{x}	匹配任何包含 x 次字符 "n" (x 为数字) 的字符串
n{x,y}	匹配任何包含至少 x 个序列的字符串 但不超过字符 "n" 的 y 倍
n{x,}	匹配任何包含至少 x 个序列的字符串 乘以 'n' 字符

用于执行 regex 操作的字符串方法

- 字符串类提供了几种执行 regex 的方法运营
 - 方法 **matches()** 将包含 regex 的字符串对象作为输入参数，只有当整个字符串与 regex 匹配时才返回 true。
 - 方法 **split()** 使用 regex 表达式作为输入，为标记化字符串
 - 方法 **replaceAll()** 使用 regex 输入参数查找匹配的子串，并用替换参数替换它们
 - 方法 **replaceFirst()** 类似于 **replaceAll()**，但会替换只匹配第一个子字符串

- 请注意，字符串方法 `replace()` 不支持 regex!

使用字符串方法的 Regex 示例 (1)

```
1  import java.util.Scanner;
2  公共类 StringRegexExample {
3      public static void main(String[] args){
4          Scanner scanner = new Scanner(System.in);
5          System.out.println ("想结束时输入'stop'! ");
6          做 {
7              System.out.print("Enter the email: ");
8              Sinput = scanner.nextLine();
9              if(input.matches("[a-z]+@[a-z]+\\.\\.[a-z]{2,3}")){
10                 System.out.println ("您的电子邮件有效! ");
11             }
12             else if(input.matches("stop|Stop|STOP")) {
13                 打破;
14             }
15             否则 {
```

```
16         System.out.println ("您的电子邮件无效! ") ;
17     }
18     } while(true);
19 }
20 }
```

使用字符串方法的 Regex 示例 (2)

```
1  import java.util.Scanner  
;  
2  公共类 StringRegex  
3      public static void mai  
4          扫描仪 scanner = new  
5          System.out.println("  
6          做 {  
7          19  
8          20 }  
9  
10  
11  
12  
13  
14  
15  
16  
17
```

为简单起见，我们假设电子邮件格式为
username@domain.xxx，用户名和 URL 中只
允许使用小写字母

S
y
s
t
e
m
.
o
u
t
.
p
r
i
n
t
(
"
E
n
t
e
r

t
h
e

```
email: "); Sinput = scanner.nextLine();  
if(input.matches("[a-z]+@[a-z]+\\. [a-z]{2,3}")) {  
    System.out.println ("您的电子邮件有效! ");  
}  
else if(input.matches("stop|Stop|STOP")) {  
    break;  
}  
否则 {  
    System.out.println ("您的电子邮件无效! ");  
}  
} while(true);  
}
```

使用字符串方法的 Regex 示例 (3)

```
1  import java.util.Scanner;
2  公共类 StringRegexExample {
3      public static void main(String[] args){
4          Scanner scanner = new Scanner(System.in);
5          System.out.println ("想结束时输入'stop'! ");
6          做 {
7              System.out.print("Enter the email: ");
8              Sinput = scanner.nextLine();
9              if(input.matches("[a-z]+@[a-z]+\\. [a-z]{2,3}")) {
10                 System.out.println ("您的电子邮件有效! ");
11             }
12             else if(input.matches("stop|Stop|STOP")) {
13                 打破;
14             }
15         }
16     }
17 }
18
19
20 }
```

匹配一个

否则

或更多小写字母

```
{
    System.out.println
}
while
(true
);
}
```

ln("您的电子邮件无效! ");

使用字符串方法的 Regex 示例 (4)

```
1  import java.util.Scanner;
2  公共类 StringRegexExample {
3      public static void main(String[] args){
4          Scanner scanner = new Scanner(System.in);
5          System.out.println ("想结束时输入'stop'! ");
6          做 {
7              System.out.print("Enter the email: ");
8              Sinput = scanner.nextLine();
9              if(input.matches("[a-z]+@[a-z]+\.[a-z]{2,3}")) {
10                 System.out.println ("您的电子邮件有效! ");
11             }
12             else if(input.matches("stop|Stop|STOP")) {
13                 打破;
14             }
```

匹配 '@'

否则 {


```
16         System.out.println ("您的电子邮件无效! ") ;
17     }
18 } while(true);
19 }
20 }
```

使用字符串方法的 Regex 示例 (5)

```
1  import java.util.Scanner;
2  公共类 StringRegexExample {
3      public static void main(String[] args){
4          Scanner scanner = new Scanner(System.in);
5          System.out.println ("想结束时输入'stop'! ");
6          做 {
7              System.out.print("Enter the email: ");
8              Sinput = scanner.nextLine();
9              if(input.matches("[a-z]+@[a-z]+\\. [a-z]{2,3}")) {
10                  System.out.println ("您的电子邮件有效! ") ;
11              }
12              else if(input.matches("stop|Stop|STOP")) {
13                  打破;
14              }
15          }
16      }
17  }
18  }
19
20 }
```

匹配一个

```
否      }  
则      } while(true);  
{  
    s  
    y  
    s  
    t  
    e  
    m  
    .  
    o  
    u  
    t  
    .  
    p  
    r  
    i  
    n  
    t  
    l  
    n  
    (  
    "  
    Y  
    o  
    u  
    r  
    e
```

或更多小写字母

邮件无效! ") ;

使用字符串方法的 Regex 示例 (6)

```
1 import java.util.Scanner;
2 公共类 StringRegexExample {
3     public static void main(String[] args){
4         Scanner scanner = new Scanner(System.in);
5         System.out.println ("想结束时输入'stop'! ");
6         做 {
7             System.out.print("Enter the email: ");
8             Sinput = scanner.nextLine();
9             if(input.matches("[a-z]+@[a-z]+\\.\\. [a-z]{2,3}")) {
10                 System.out.println ("您的电子邮件有效! ");
11             }
12             else if(input.matches("stop|Stop|STOP")) {
13                 打破;
14             }
15             否则 {
16                 System.out.print
17             }
18         }
19     } while(true);
20 }
```

匹配'.'： 请注意，在 Java 字符串中，regex 字符'\\.'必须写成'\\.\\.'， 因为 Java 编译器在编译 regex 之前会将反斜杠假定为转义字符！

使用字符串方法的 Regex 示例 (7)

```
1  import java.util.Scanner;
2  公共类 StringRegexExample {
3      public static void main(String[] args){
4          Scanner scanner = new Scanner(System.in);
5          System.out.println ("想结束时输入'stop'! ");
6          做 {
7              System.out.print("Enter the email: ");
8              Sinput = scanner.nextLine();
9              if(input.matches("[a-z]+@[a-z]+\\. [a-z]{2,3}")) {
10                 System.out.println ("您的电子邮件有效! ");
11             }
12             else if(input.matches("stop|Stop|STOP")) {
13                 打破;
14             }
15             否则 {
```



匹配两个
到三个小写字母

```
16         System.out.println("Your email is not valid! ");
17     }
18 } while(true);
19 }
20 }
```

使用字符串方法的 Regex 示例 (8)

```
1  import java.util.Scanner;
2  公共类 StringRegexExample {
3      public static void main(String[] args){
4          Scanner scanner = new Scanner(System.in);
5          System.out.println ("想结束时输入'stop'! ");
6          做 {
7              System.out.print("Enter the email: ");
8              Sinput = scanner.nextLine();
9              if(input.matches("[a-z]+@[a-z]+\\. [a-z]{2,3}")){
10                 System.out.println ("您的电子邮件有效! ");
11             }
12             else if(input.matches("stop|Stop|STOP")) {
13                 打破;
14             }
15             否则 {
```



```
16         System.out.println ("您的电子邮件无效! ") ;  
17     }  
18     } while(true);  
19 }  
20 }
```

接受不同的 "停止" 写法

使用字符串方法的 Regex 示例 (9)

```
1  import java.util.Scanner;
2  公共类 StringRegexExample {
3      public static void main(String[] args){
4          Scanner scanner = new Scanner(System.in);
5          System.out.println ("想结束时输入'stop'! ");
6          做 {
7              System.out.print("Enter your email: ");
8              Sinput = scanner.nextLine();
9              if(input.matches("[a-z]+@[a-z]+\\. [a-z]{2,3}")
10                 System.out.println ("您的电子邮件有效! ")
11             }
12             else if(input.matches("stop|Stop|STOP")
13                 打破;
14             18
15             19
16             20 }
```

否则

"Your email"

```
    }  
  }  
  while  
  (true  
  );  
}
```

```
$ java StringRegexExample
```

要结束时，请输入

"stop" (停止) !

输入电子邮件：

teacher@school.

edu 您的电子邮件

有效!

输入电子邮件：

james.smith@comp

any.com 您的电子邮

件无效!

输入电子邮件：停止

\$

1 无效! ") ;

类模式和匹配器

- Java 没有任何内置的 `regex` 类，但我们可以导入 **`java.util.regex`** 包，使用以下类处理正则表达式：
 - **类模式**：定义模式（用于搜索）
 - **类匹配器**：用于搜索模式
 - **类 `PatternSyntaxException`**：定义当 `regex` 字符串中出现语法错误时抛出的异常

使用模式和匹配器

- 模式对象由静态方法 **Pattern.compile()** 创建
 - 第一个参数是一个 regex 字符串，指定了要搜索的模式
 - 第二个参数（可选）指定指示如何执行搜索的标志，例如，标志 **Pattern.CASE_INSENSITIVE** 指示忽略大小写。
- Pattern 对象的方法 **matcher()** 用于搜索输入参数所给字符串中的模式；该方法返回一个包含结果信息的 Matcher 对象。
- 方法 **find()** 返回 true。

则为假，未找到则为假

模式和匹配器示例 (1)

```
1  import java.util.regex.Matcher;
2  import java.util.regex.Pattern;
3
4  公共类 PatternMatcherExample {
5      public static void main(String[] args){
6          Pattern pattern = Pattern.compile("[0-3]\\d/[0-1]\\d/\\d\\d\\d");
7          String text = "John Smith was born on 14/05/1973.\n" +
8                        "他的妻子简生于1976年12月9日" +n
9                        "他们有一个儿子, 生于 1997 年 10 月 31 日" + "他们有一个儿子, 生于 1997 年 10 月 31"
10                       "日"。
11                       "一个女儿, 2001年2月1日出生";
12          匹配器 matcher = pattern.matcher(text);
13          while(matcher.find()) {
14              System.out.println("Date found: " + matcher.group());
15          }
16      }
```

模式和匹配器示例 (2)

```
1  import java.util.regex.Matcher;
2  import java.util.regex.Pattern;
3
4  公共类 PatternMatcherExample {
5      public static void main(String[] args){
6          Pattern pattern = Pattern.compile("[0-3]\\d/[0-1]\\d/\\d\\d\\d");
7          String text = "John Smith was born on 14/05/1973.\n" +
8                      "他的妻子简生于1976年12月9日" +n
9                      "他们有一个儿子, 生于 1997 年 10 月 31 日" + "他们有一个儿子, 生于 1997 年 10
10                     月 31 日"。
11                     "和一个女儿, 01
12
13         匹配器 matcher = pattern.matcher(text);
14         while (matcher.find()) {
```

编译日期格式为

dd/mm/yyyy 的模式匹配器

; (注意, 该 regex 仅对日

期进行强验证)

```
13         System.out.println("Date found: " + mat
14     }
15 }
16 }
```


模式和匹配器示例 (3)

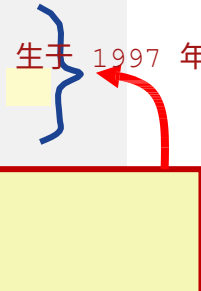
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16



尝试在文本中找到指定的模式

模式和匹配器示例 (4)

```
1  import java.util.regex.Matcher;
2  import java.util.regex.Pattern;
3
4  公共类 PatternMatcherExample {
5      public static void main(String[] args){
6          Pattern pattern = Pattern.compile("[0-3]\\d/[0-1]\\d/\\d\\d\\d");
7          String text = "John Smith was born on 14/05/1973.\n" +
8              "他的妻子简生于1976年12月9日" + \n
9              "他们有一个儿子, 生于 1997 年 10 月 31 日" + "他们有一个儿子, 生于 1997 年 10
              月 31 日"。
10             "一个女儿, 2001年2月1日出生";
11      匹配器 matcher = pattern.matcher(text);
12      while(matcher.find()) {
```



The diagram illustrates the matching process. A red bracket groups the date strings in the text: "14/05/1973", "1976年12月9日", "1997 年 10 月 31 日", and "2001年2月1日". A red arrow points from this bracket to a yellow box, which represents the matched text.

```
13      System.out.println("Date found: " + matcher.group());
14  }
15  }
16 }
```

循环查看所有匹配的子字符串
在输入字符串中找到

模式和匹配器示例 (5)

```
1  import java.util.regex.Matcher;
2  import java.util.regex.Pattern;
3
4  公共类 PatternMatcherExample {
5      public static void main(String[] args){
6          Pattern pattern = Pattern.compile("[0-3]\\d/[0-1]\\d/\\d\\d\\d");
7          String text = "John Smith was born on 14/05/1973.\n" +
8                      "他的妻子简生于1976年12月9日" +\n
9                      "他们有一个儿子, 生于 1997 年 10 月 31 日" + "他们有一个儿子, 生于 1997 年 10\n
10                     "月 31 日"。
11                     "一个女儿, 2001年2月1日出生"
11      匹配器 matcher = pattern.matcher(text);
12      while(matcher.find()) {
```

```
13     System.out.println("Date found: " + matcher.group());
14 }
15 }
16 }
```

```
$ java PatternMatcherExample
```

```
找到日期: 1973 年 5 月 14 日
```

```
发现日期: 9/12/1976 发
```

```
现日期: 31/10/1997 发现
```

```
日 期 :
```

```
01/02/200131/10/1997
```

```
发现日期: 01/02/2001
```

```
$
```

有问题或意见？

JC2002 Java 程序设计

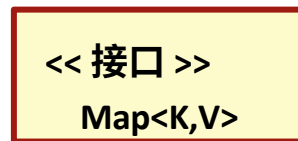
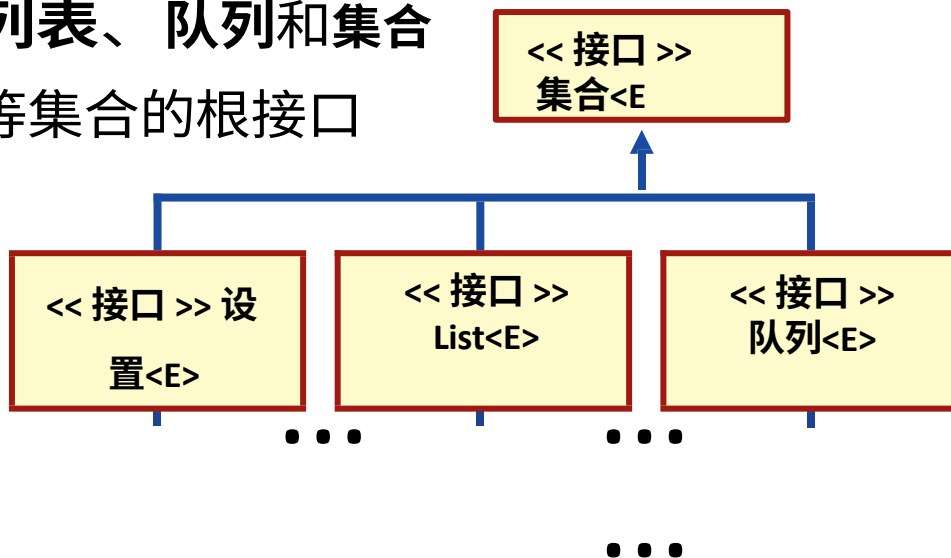
第 11 天，第 4 课：收藏的基础知识

收藏品

- 任何一组单独的物体，作为一个整体，被称为物体集合
 - 例如，动物园可以定义为动物的集合体
- 在 Java 中，有一个单独的框架（集合框架）用于处理定义了集合的数据结构
 - 集合的主要类和接口包含在 **java.util.Collection** 和 **java.util.Map** 包中。

收集界面

- 列表、队列和集合等集合的根接口



- 关联键集合

- 先入先出的收集模式等待队伍
到值，不能包含重复的键，不是从 Collection 派生的！

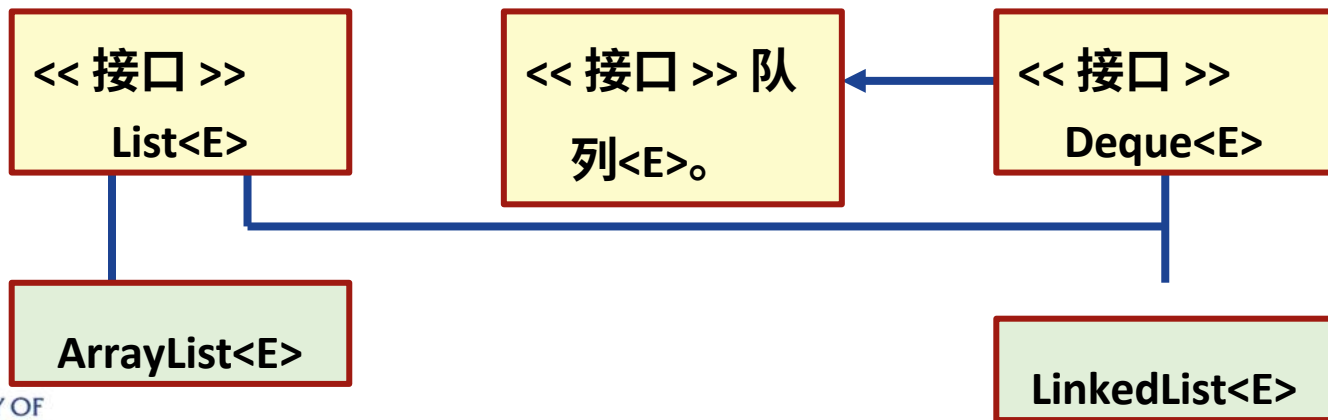
集合与数组

- 与数组不同，集合 *可以*
 - 存储同质和异质数据类型
 - 扩大规模
 - 执行速度比数组慢
- 与数组不同，集合 *不能*:
 - 存储原始类型 (*int*、*char* 等)
- 集合已定义了许多支持方法，并且更好地

在内存空间使用方面

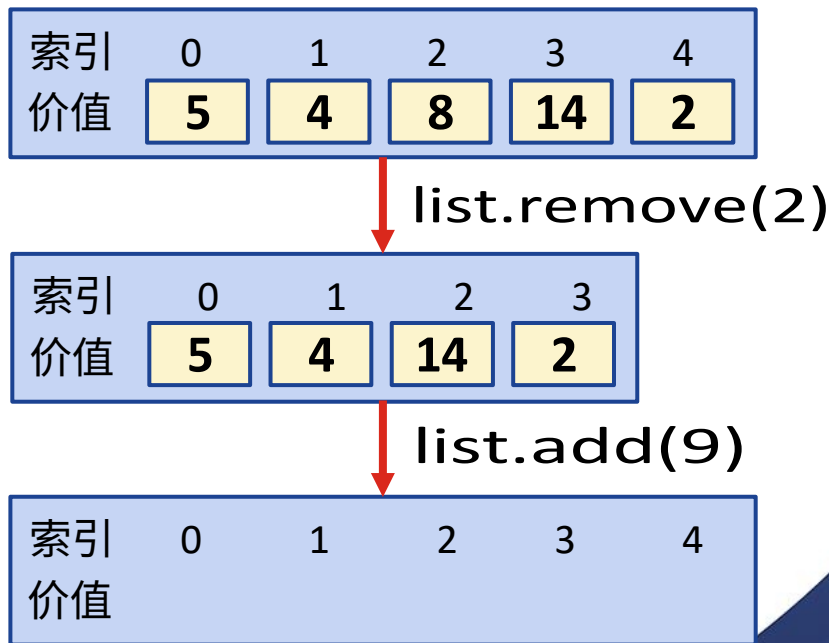
列表

- 有两种类型的列表类： **ArrayList** 和 **LinkedList**
- 对 List 进行迭代时，会保留元素的顺序



数组列表

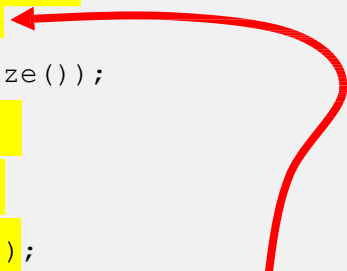
- 通常，列表实现的最佳选择是
- 通过调用 **list.get(index)**
- 通过调用 **list.remove(index)**
- 通过调用 **list.add(data)**



5	4	14	2	9
---	---	----	---	---

数组列表示例

```
1  import java.util.* ;
2
3  公共类 ArrayListExample {
4      public static void main ( String[] args) {
5          List<String> nameList = new ArrayList<String>();
6          System.out.println("initial size: " + nameList.size());
7          nameList.add("Bob");      nameList.add("Cecilia");
8          nameList.add("Alice");    nameList.add("Daniel");
9          System.out.println("new size: " + nameList.size());
10         nameList.remove(1);
11         System.out.println("new size: " + nameList.size());
12         nameList.add("Edward");
13     }
14 }
15
```




```
16     System.out.println("Final name list:");
17     for(int i=0; i<nameList.size(); i++) {
18         System.out.print(nameList.get(i) + " ");
    }
    System.out.println("");
}
```

我们引用 `ArrayList` 实现 `List` 接口是为了增加灵活性：如果我们后来发现 **LinkedList** 更适合我们，我们可以很容易地改变它。

数组列表示例：输出

```
1  import java.util.* ;
2  公共类 ArrayListExample {
3      public static void main ( String[] args) {
4          List<String> nameList = new ArrayList<String>();
5          System.out.println("initial size: " + nameList.size());
6          nameList.add("Bob");          nameList.add("Cecilia");
7          nameList.add("Alice");        nameList.add("Daniel");
8          System.out.println("new size: " + nameList.size())
9          nameList.remove(1);
10         System.out.println("new size: " + nameList.size())
11         nameList.add("Edward");
```

```
18 }      System.out.println("Final name list:");
          for(int i=0; i<nameList.size(); i++) {
              System.out.print(nameList.get(i) + " ")
              ;
          }
          System.out.println("");
      }
```

```
$ java ArrayListExample
```

初始大小: 0

新尺寸: 4

新尺寸: 3

最终名单:

鲍勃-爱丽丝 丹尼尔-爱德华

```
$
```

迭代器

- 迭代器类对象可用于循环浏览集合
 - "迭代"是循环的专业术语
 - 方法 **iterator()** 可用于为任何系列
 - 迭代器的 **hasNext()** 和 **next()** 方法可用于循环浏览集合
- 注意，如果集合是使用其方法之一进行修改
 - 然后，使用迭代器会抛出 `ConcurrentModificationException`（并发修改异

常

- 有助于避免两个线程同时修改集合

迭代器示例

```
1  import java.util.* ;
2  公共类 IteratorExample {
3      public static void main ( String[] args) {
4          ArrayList<String> nameList = new ArrayList<String>();
5          nameList.add("Bob");      nameList.add("Cecilia");
6          nameList.add("Alice"); nameList.add("Daniel");
7          Iterator<String> iterator = nameList.iterator();
8          while(iterator.hasNext()) {
9              System.out.println(iterator.next());
10             // nameList.remove(0);
11             // iterator.remove();
12         }
13     }
14 }
```

这行不通

这样可以

```
15 }      System.out.println ("大小: " + nameList.size()) ;
```

迭代器示例：输出 (1)

```
1  import java.util.* ;
2  公共类 IteratorExample {
3      public static void main ( String[] args) {
4          ArrayList<String> nameList = new ArrayList<String>();
5          nameList.add("Bob");      nameList.add("Cecilia");
6          nameList.add("Alice"); nameList.add("Daniel");
7          Iterator<String> iterator = nameList.iterator();
8          while(iterator.hasNext()) {
9              System.out.println(iterator.next());
10             // nameList.remove(0)
11         } ;
12         // iterator.remove();
```

```
$ java IteratorExample
Bob
塞西莉亚
```



```
13  
14     } System.out.println ("大小: " + nameList.size())  
15 } ;
```

爱丽丝-丹尼

尔 尺寸: 4

\$

迭代器示例：输出 (2)

```
1  import java.util.* ;
2  公共类 IteratorExample {
3      public static void main ( String[] args) {
4          ArrayList<String> nameList = new ArrayList<String>();
5          nameList.add("Bob");      nameList.add("Cecilia");
6          nameList.add("Alice"); nameList.add("Daniel");
7          Iterator<String> iterator = nameList.iterator();
8          while(iterator.hasNext()) {
9              System.out.println(iterator.next());
10             // nameList.remove(0)
11             ;
12         }
13         iterator.remove();
```

```
$ java IteratorExample
Bob
塞西莉亚
```

```
13  
14     } System.out.println ("大小: " + nameList.size())  
15 } ;
```

爱丽丝-丹尼

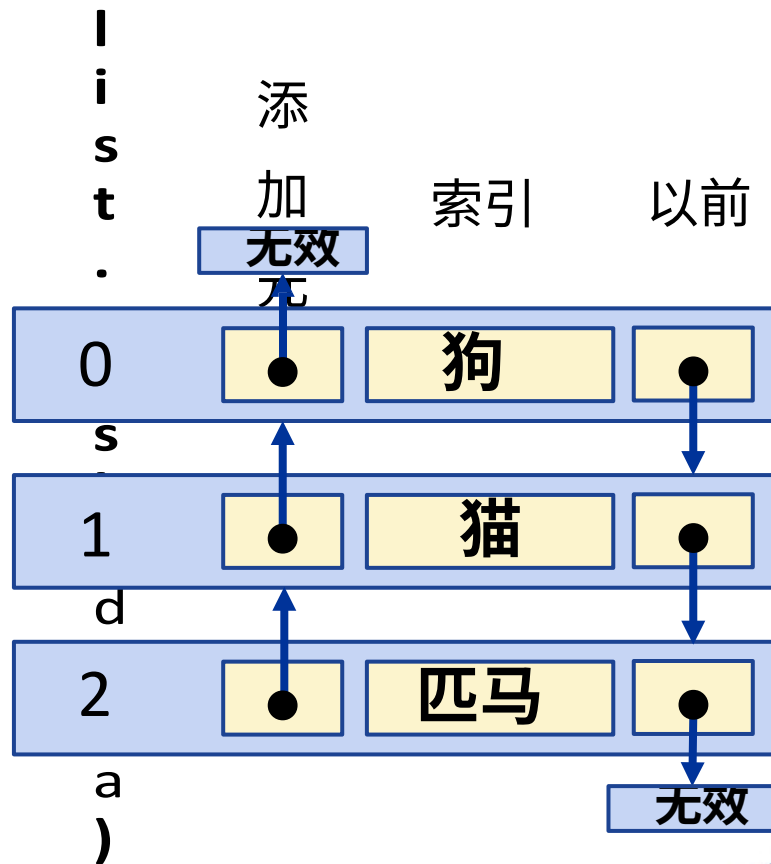
尔 尺寸: 0

\$

关联列表

- 始于头，终于尾
- 元素与下一个元素相连
- 高效插入和删除
但不利于记忆
- 通过调用
list.get(index)
- 通过调用

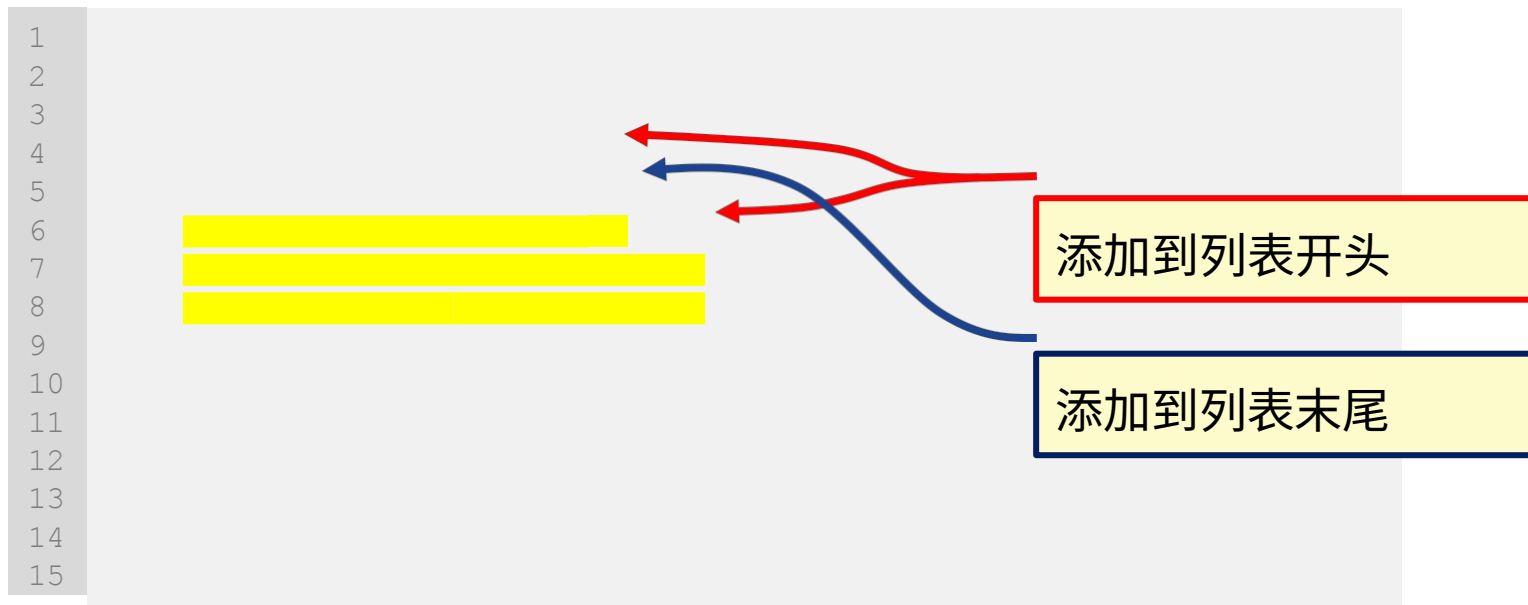
list.add(data) 或



价值

下一个

LinkedList 示例



链接列表表示例：输出

```
1  import java.util.* ;
2  公共类 LinkedListExample {
3      public static void main ( String[] args) {
4          LinkedList<String> nameList = new LinkedList<String>();
5          nameList.push ("Bob") ;
6          nameList.add("Daniel");
7          nameList.addFirst ("Alice") ;
8          nameList.add (2, "Cecilia") ;
9          System.out.println("Name list:"
10         for(int i=0; i<nameList.size()
11             System.out.print(nameList.get(i) " ")
12     }
```

```
13  
14     } System.out.println("");  
15 }
```

```
$ java LinkedListExample
```

名单:

爱丽丝-鲍勃-塞西莉亚-丹尼尔

```
$
```


ArrayList 与 LinkedList

- 基本上，两种类型的列表都可以做同样的事情：它们的主要区别在于数据的内部表示法
 - 在 ArrayList 中，查找某个索引处的元素更快，因为元素在内存中按固定顺序排列
 - 在 LinkedList 中，添加和删除元素的速度更快，因为无需在内存中移动大块数据
- 选择取决于应用

- 如果经常需要添加和删除元素，则 LinkedList 更好
- 如果您只需要修改元素而不需要添加或删除元素，那么 ArrayList 更好。

收藏中的一些重要方法

方法	说明
排序()	对列表元素排序
二进制搜索()	使用高效的二进制搜索算法定位列表中的元素
反转()	逆转列表元素
洗牌()	随机调整列表元素的顺序
填充()	设置 List 中的每个元素，使其指向特定对象

复制()

将参考资料从一个列表复制到另一个列表

纸牌示例：纸牌类

```
1  import java.util.* ;
2  类 卡 {
3      public enum Face {Ace, Two, Three, Four, Five, Six, Seven、
4                          8、9、10、J、Q、K}
5      公共枚举 Suit {小熊、方块、黑桃、红心}。
6      私有最终 Face face;
7      私人最终西装;
8      public Card(Face face, Suit suit) {
9          this.face = face; this.suit =
10     }
11     public Face getFace() { return face
12     public Suit getSuit() { return
```

```
16 } public String toString() {  
    return String.format("%s of %s", face, suit)  
}
```

\$ java LinkedListExample

名单:

爱丽丝-鲍勃-塞西莉亚-丹尼尔

\$

纸牌示例：使用列表

```
18 public class DeckOfCards {
19     private List<Card> cards;
20     public DeckOfCards() {
21         Card deck[] = new Card[52];
22         int count = 0;
23
24         for(Card.Suit suit: for(Card.Suit suit:
25             Card.Suit.values()) { for(Card.Face face:
26                 for(Card.Face.values()) {
27                     deck[count++] = 新卡 (面值、花色);
28                 }
29             }
30         }
31         cards = Arrays.asList(deck);
32         Collections.shuffle(cards);
33     }
34     public void printCards() {
35         for(int i=0; i<52; i++) {
36             System.out.printf("%-19s%s", card
37                 ((i+1) % 4 == 0) ? "\n" : "");
38         }
39     }
40 }
```

将数组转换为列表

```
37 public static void main ( String[] args) {
38     DeckOfCards deck = new DeckOfCards();
39     deck.printCards();
40 }
```

```
41 }
```

```
}
```


纸牌示例：输出

```
$ java DeckOfCards
```

```
国王 的 黑桃      钻石皇后      方块八      红心六
五 的 心          红心杰克      梅花J        黑桃三
五 的 黑桃        黑桃 A       红心八号      黑桃四
国王 的 钻石      红心四        黑桃皇后      梅花九
杰克 的 黑桃      红心三        钻石杰克      梅花四
四个 的 钻石      俱乐部二人组 梅花 A       红心 A
梅花十      钻石王牌      黑桃六        黑桃八
梅花六      梅花七        方块五        梅花八
黑桃九      红心七号      黑桃二        两颗钻石
梅花王      方块九        红桃皇后      钻石三
红心九      黑桃七        黑桃十        梅花三
红心二号    钻石六        方块七        钻石十
红心国王    红心皇后      梅花五        红心十
$
```

sort() 和 binarySearch() 示例

```
1  import java.util.* ;
2  公共类 BinarySearchExample {
3      public static void main ( String[] args) {
4          String[] names = {"Bob", "Alice", "Edward", "Cecilia", "David", "Frank"};
5          List<String> list = new ArrayList<>(Arrays.asList(names));
6          Collections.sort(list);
7          for(String name : list) {
8              System.out.println(name);
9          }
10         int index = Collections.binarySearch(list, "Edward");
11         System.out.printf("Index of Edward is %d\n", index);
12     }
13 }
```

sort() 和 binarySearch() 的输出结果

```
1  import java.util.* ;
2  公共类 BinarySearchExample {
3      public static void main ( String[] args) {
4          String[] names = {"Bob", "Alice", "Edward", "Cecilia", "David", "Frank"};
5          List<String> list = new ArrayList<>(Arrays.asList(names));
6          Collections.sort(list);
7          for(String name : list) {
8              System.out.println(name);
9          }
10         int index = Collections.binarySearch(list, "David");
11         System.out.printf("Index of David is %d\n", index);
12     }
13 }
```

```
$ java BinarySearchExample
Alice
鲍勃-塞
西莉亚-
戴维-爱
德华-弗
兰克
```


摘要

- 在许多应用中，字符串操作如搜索子串、连接字符串和修改字符串内容是必不可少的。
 - 在 Java 中，字符串对象是不可变的：创建后其内容不能更改
 - 类 `StringBuilder` 可用于创建动态字符串缓冲区
- 正则表达式通常用于定义字符串模式，以特定格式搜索和验证字符串内容
- Java 集合框架可用于处理数据结构

持有一组项目，如列表、集合和队列

有问题或意见？