**07\_List\_and\_Collections\_笔记**

**一、List接口**

**1. List接口的特点 ﻿**

* 继承关系：List是Collection的子接口
* 元素特性：
  + 有序性：元素按插入顺序排列，每个元素对应整数型序号
  + 可重复性：允许存储相同元素
* 与数组区别：List大小可动态扩展，而数组大小固定需要手动复制扩容

**2. List接口的实现类 ﻿**

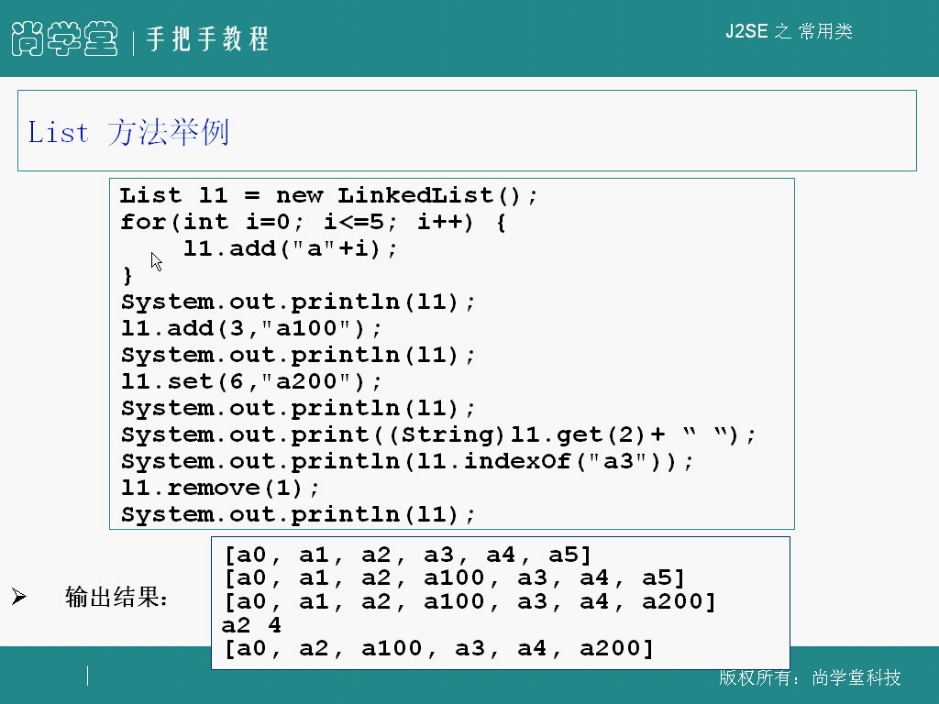
* ArrayList：底层基于数组实现
* LinkedList：底层基于链表实现

**3. List接口的方法**



* 元素访问：
  + Object get(int index)：获取指定位置元素，返回Object类型
  + Object set(int index, Object element)：设置指定位置元素，返回旧元素
* 元素操作：
  + void add(int index, Object element)：在指定位置插入元素，后续元素后移
  + Object remove(int index)：移除指定位置元素
* 元素查找：
  + int indexOf(Object o)：返回元素首次出现位置（使用equals方法比较）
  + int lastIndexOf(Object o)：返回元素最后出现位置

**二、List方法举例**



* 初始化示例：
* 输出结果：[a0,a1,a2,a3,a4,a5]
* 插入操作：
* 结果：[a0,a1,a2,a100,a3,a4,a5]（a3及后续元素后移）
* 修改操作：
* 结果：[a0,a1,a2,a100,a3,a4,a200]
* 元素获取：
* 输出：a2
* 元素查找：
* 输出：4（查找时使用equals方法比较）
* 元素移除：
* 最终结果：[a0,a2,a100,a3,a4,a200]

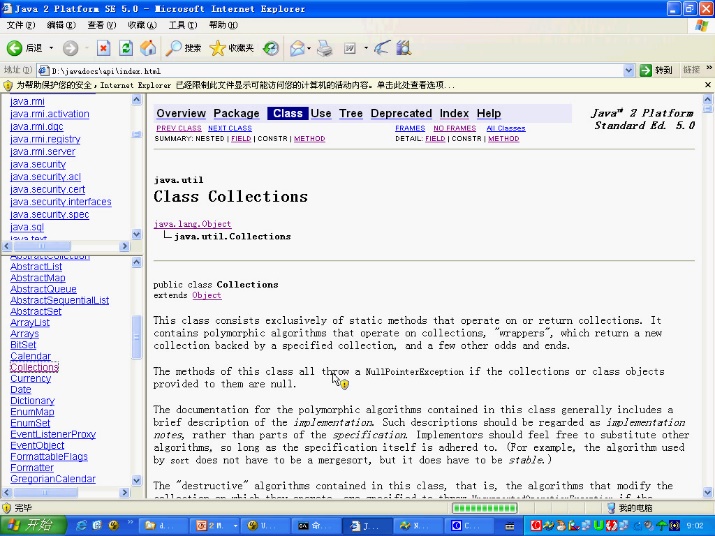
**三、List常用算法**

**1. List常用算法介绍**

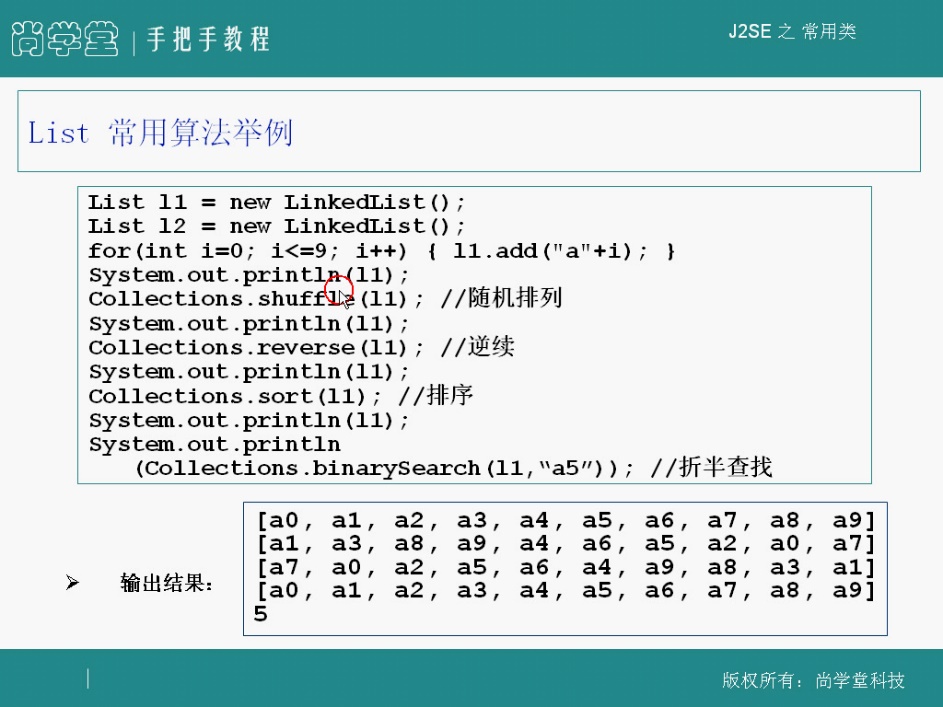


* 核心类：java.util.Collections（注意是复数形式，区别于Collection接口）
* 方法特性：所有方法均为静态方法，专门操作List容器
* 排序算法：
  + sort：对List元素进行稳定排序（默认升序）
  + shuffle：实现元素的随机排列（每次结果可能不同）
  + reverse：将元素顺序完全倒置
* 填充操作：
  + fill：用指定对象替换List中所有元素
* 拷贝操作：
  + copy：将源List内容完整复制到目标List（需保证目标List容量足够）
* 查找算法：
  + binarySearch：对已排序List使用二分查找（时间复杂度

）

* + 
* 实现注意：算法实现细节可能变化（如sort不强制要求归并排序，但必须保证稳定性）
* 空指针约束：所有方法对null参数抛出NullPointerException
* 修改规则：会修改原容器的方法称为"destructive algorithms"

**2. List常用算法举例 ﻿**



* 基础操作：
  + 初始化：List l1 = new LinkedList()创建链表实现的List
  + 填充数据：循环添加元素"a0"到"a9"（保持插入顺序）
* 随机排列：
  + shuffle效果：原有序列表变为随机排列（如[a1,a3,a8,...]）
  + 注意事项：每次运行结果不同是正常现象
* 逆序操作：
  + reverse原理：链表逆序只需调整指针，效率高于数组的拷贝操作
  + 排序组合：实现降序排列=先sort升序再reverse
* 二分查找：
  + 前提条件：必须在已排序列表上使用（示例返回元素索引5）
  + 效率优势：相比线性查找O(n)，二分法仅需

﻿

* 实现细节：
  + 数据结构选择：ArrayList逆序需要元素拷贝，LinkedList只需指针调整
  + 默认排序：自然顺序（可通过Comparator自定义）
  + 封装优势：直接调用标准库算法，无需自行实现底层数据结构

**五、知识小结**

|  |  |  |  |
| --- | --- | --- | --- |
| 知识点 | 核心内容 | 考试重点/易混淆点 | 难度系数 |
| List接口特性 | 有序、可重复的容器，元素通过整数型序号定位，类似数组但可动态扩容 | ArrayList（数组实现）与LinkedList（链表实现）的性能差异 | ⭐⭐ |
| List常用方法 | get(index)、set(index,element)（返回旧值）、add(index,element)（后移元素）、remove(index) | set返回旧元素 vs add导致元素位移；indexOf依赖equals方法 | ⭐⭐⭐ |
| Collections工具类 | 提供静态方法：sort（排序）、shuffle（随机排列）、reverse（逆序）、binarySearch（二分查找） | 默认升序排序，降序需先sort后reverse；binarySearch需预先排序 | ⭐⭐ |
| 泛型与历史版本 | JDK1.4使用Object类型返回值，导致类型转换问题；泛型（后续讲解）解决此问题 | 理解Object返回值的设计动机与泛型的必要性 | ⭐⭐⭐⭐ |
| 链表与数组性能对比 | LinkedList逆序效率高于ArrayList（仅调整指针 vs 数组元素拷贝） | 底层数据结构差异对算法效率的影响 | ⭐⭐⭐ |
| 示例代码分析 | add(3,"a100")插入元素导致后续位移；set(6,"a200")替换元素并返回旧值 | 操作后容器状态变化的跟踪与调试 | ⭐⭐ |

该部分主要讲述了Java中的Collections类。Collections类提供了一系列静态方法，用于对List容器进行常用算法操作，包括排序、随机排列、逆序排列、复制、二分查找等。这些方法可以直接通过类名调用，不需要创建对象。

此外，还介绍了List的顺序性以及不同类型的List实现方式。最后，提到了从高到低排序和二分查找的方法。

* **分段总结**

**List容器**

1.List容器类似于数组，但大小可以改变。

2.List容器中的元素具有整数型序号，记录它们在容器中的位置。

**ArrayList和LinkedList**

1.ArrayList底层是数组实现，LinkedList底层是链表实现。

2.ArrayList可以随机访问元素，但插入和删除操作可能较慢。

3.LinkedList插入和删除操作较快，但随机访问元素较慢。

**List的相关方法**

1.get(index):获取指定位置上的元素。

2.set(index, element):设置指定位置上的元素。

3.add(index, element):在指定位置上添加新元素。

4.remove(index):去除指定位置上的元素。

**List的常用算法**

1.sort:排序某个List对象。

2.shuffle:进行随机排列。

3.reverse:进行逆序排列。

4.fill:用一个特定对象替换List中所有元素。

5.copy:将某个List的内容复制到另一个List中。

6.binarySearch:使用二分法查找List中的元素。

**Collections类**

1.Collections类提供了基于List容器的常用算法。

2.sort:排序某个List对象。 3.shuffle:进行随机排列。

4.reverse:进行逆序排列。

5.fill:用一个特定对象替换List中所有元素。

6.copy:将某个List的内容复制到另一个List中。

7.binarySearch:使用二分法查找List中的元素。