

## 第25课

# 会扩容的生物队列

用 ArrayList 编辑动态波次

### 两种容器都按顺序保存同类型元素

#### 数组

创建后长度固定

#### ArrayList

自动管理容量

#### size()

当前元素数

波次编辑器要随时加怪、换怪、删怪。会扩容的列表给你自由，也要求你重新检查索引。你会用测试证明失败操作没有改坏队伍。

前置背包：数组、循环、方法和对象。数组像固定快捷栏，ArrayList 像会扩建的仓库，但管理员编号仍从0开始。

**阅读方式：先猜结果，再运行核对；每次只改一个条件。**

本章使用 JDK 21，无需预览功能。故事与插画借用方块世界，练习运行在普通 Java 控制台。

## 开始前：先预测，再让程序回答

本章使用 JDK 21，无需预览功能。故事借用方块世界，示例运行在普通 Java 控制台；数值和规则若是课程模型，都会明确说明。

每个普通示例都是完整独立文件，按框上标注的文件名保存。各示例放在不同文件夹，避免辅助类重名。完整项目分段时，按顺序接在同一个文件中。

### 终端命令：中文占位符换成实际文件名与类名

```
javac -encoding UTF-8 示例文件名.java  
java 示例类名
```

- 预测：先写预期输出，或画清引用和数据的变化。
- 验证：运行后逐行核对，再只改一个条件。
- 解释：说清结果来自 Java 规则、课程合同还是输入数据。

“故意错误”用于认识 Bug，不能与普通示例混用。文件实验使用专门临时目录；不要拿真实存档测试覆盖和删除。

# 01 数组固定而列表会扩展

怪物波次数量在编辑时不断变化。数组需要预先决定长度，ArrayList 可以按需添加或删除。

## 两种容器都按顺序保存同类型元素

### 数组

创建后长度固定

### ArrayList

自动管理容量

### size()

当前元素数

ArrayList 会自动管理内部容量，但仍受内存限制。课程中的“会扩容”表示无需手动重建更大的数组。

## 验收对照表

| 比较   | 数组           | ArrayList      |
|------|--------------|----------------|
| 长度   | 创建后固定        | 可动态变化          |
| 读取   | items[i]     | list.get(i)    |
| 修改   | items[i] = x | list.set(i, x) |
| 添加   | 需另建数组        | list.add(x)    |
| 长度查询 | length       | size()         |

## 02 泛型限制元素类型

泛型把错误挡在编译阶段。ArrayList<String> 表示列表元素必须是 String。

### 尖括号写明队列只收哪种物品

#### List<String>

元素类型在编译时检查

#### ArrayList<>

创建可变实现

#### 泛型

不能添加不兼容数值

不要写裸的 ArrayList。缺少泛型会失去类型检查，取出元素时还要危险地强转。

#### L25Example02.java • 完整独立示例

```
import java.util.*;
public class L25Example02 {
    public static void main(String[] args) {
        List<String> wave = new ArrayList<>();
        wave.add("Zombie");
        System.out.println(wave);
    }
}
```

#### 运行核对

[Zombie]

故障实验：String 列表不能添加 int 数值。下面故意保留错误，请先预测，再尝试修复。

#### BadGeneric.java • 故意错误示例

```
import java.util.*;
public class BadGeneric {
    public static void main(String[] args) {
        List<String> wave = new ArrayList<>();
        wave.add(20);
    }
}
```

## 03 add get 与 size

add 把元素放到末尾，size 返回当前元素数量。列表有3个元素时，合法索引为0、1、2。

### 添加到末尾 用索引读取 用 size 查询数量

**add(x)**

末尾添加

**add(i,x)**

插入并移动后续项

**get(i)**

读取合法索引

在1号位置插入 Creeper 后，原来的1号及后续元素整体向右移动，索引都会变化。

### L25Example03.java • 完整独立示例

```
import java.util.*;
public class L25Example03 {
    public static void main(String[] args) {
        List<String> wave = new ArrayList<>();
        wave.add("Zombie"); wave.add("Skeleton");
        wave.add(1, "Creeper");
        System.out.println(wave);
        System.out.println(wave.size());
        System.out.println(wave.get(2));
    }
}
```

### 运行核对

```
[Zombie, Creeper, Skeleton]
3
Skeleton
```

## 04 set 与按索引 remove

set 替换指定元素，列表大小不变；remove(index) 删除元素，大小减1。

### 替换不改大小 删除会让后面元素左移

#### set(i,x)

替换，大小不变

#### remove(i)

删除，大小减1

#### 右侧元素

索引向左移动

删除元素后，右侧索引全部减1。若你保存了旧索引，继续使用前要重新确认它指向谁。

### L25Example04.java • 完整独立示例

```
import java.util.*;
public class L25Example04 {
    public static void main(String[] args) {
        List<String> wave = new ArrayList<>(List.of("Zombie", "Skeleton"));
        System.out.println(wave.set(1, "Spider"));
        System.out.println(wave.remove(0));
        System.out.println(wave);
    }
}
```

### 运行核对

```
Skeleton
Zombie
[Spider]
```

## 05 删除索引还是删除对象

`remove(1)` 通常表示删除1号位置, `remove("Spider")` 表示删除首次匹配的对象。整数列表需要特别小心。

### remove 重载根据参数类型选择版本

#### `remove(int)`

参数是索引

#### `remove(Object)`

删除首个匹配值

#### `Integer.valueOf`

明确选择对象删除

看到 `remove` 先问传入的是 `int` 索引还是某个对象。泛型是 `Integer` 时, 两种版本都可能看起来像数字。

### L25Example05.java • 完整独立示例

```
import java.util.*;
public class L25Example05 {
    public static void main(String[] args) {
        List<Integer> ids = new ArrayList<>(List.of(10, 20, 30));
        System.out.println(ids.remove(1));
        System.out.println(ids.remove(Integer.valueOf(10)));
        System.out.println(ids);
    }
}
```

### 运行核对

```
20
true
[30]
```

## 06 增强 for 读取队列

只想依次读取元素时，增强 for 最合适。遍历期间直接 add 或 remove 可能触发 ConcurrentModificationException。

### 只读遍历简洁清楚

#### 增强 for

适合顺序读取

#### 结构修改

会使迭代失效

#### 受控删除

用 removeIf 等方案

增强 for 期间把列表结构当成只读。需要删除时使用 removeIf、Iterator，或倒序索引循环。

### L25Example06.java • 完整独立示例

```
import java.util.*;
public class L25Example06 {
    public static void main(String[] args) {
        List<String> wave = new ArrayList<>(List.of("Zombie", "Spider"));
        for (String mob : wave) System.out.println(mob);
    }
}
```

### 运行核对

Zombie  
Spider

故障实验：这组数据在继续迭代时抛 ConcurrentModificationException；不要依靠这个异常保证每次违规都能被发现。下面故意保留错误，请先预测，再尝试修复。

### BadIteration.java • 故意错误示例

```
import java.util.*;
public class BadIteration {
    public static void main(String[] args) {
        List<String> wave = new ArrayList<>(List.of("A", "B", "C"));
        for (String mob : wave) wave.remove(mob);
    }
}
```

## 07 索引循环支持受控修改

替换指定位置时普通 for 很合适。若正向删除多个元素，删除后不要跳过刚移动过来的新元素。

### 需要位置时使用普通 for

#### 倒序循环

从最后一个索引开始

#### 删除当前项

不改变较小索引

#### 不跳项

相邻匹配也可删净

从末尾删除不会改变尚未访问的较小索引，因此不会跳过元素。

### L25Example07.java • 完整独立示例

```
import java.util.*;
public class L25Example07 {
    public static void main(String[] args) {
        List<String> wave = new ArrayList<>(List.of("Zombie", "Zombie", "Spider"));
        for (int i = wave.size() - 1; i >= 0; i--)
            if ("Zombie".equals(wave.get(i))) wave.remove(i);
        System.out.println(wave);
    }
}
```

### 运行核对

[Spider]

## 08 removelf 安全过滤

按条件删除全部匹配时，removelf 比手写索引循环更直接。

### 把删除条件交给列表统一执行

#### removelf

按条件删除匹配项

#### 常量.equals

安全处理 null

#### 可变列表

删除的必要前提

若列表允许 null，条件里先检查 `mob != null`，或把常量放在 `equals` 左边。

### L25Example08.java • 完整独立示例

```
import java.util.*;
public class L25Example08 {
    public static void main(String[] args) {
        List<String> wave = new ArrayList<>();
        wave.add(null); wave.add("Zombie"); wave.add("Spider");
        wave.removeIf("Zombie"::equals);
        System.out.println(wave);
    }
}
```

### 运行核对

[null, Spider]

常量字符串的 `equals` 可安全接收 null。removelf 会修改列表，要求调用者传入支持删除的容器；List.of 与 Stream.toList 的结果不能用于这类修改。

# 09 空列表与索引边界

空列表可以正常存在，但 `get(0)` 会抛 `IndexOutOfBoundsException`。

## size 为0时没有0号元素

### 空列表

可以存在但不能 `get(0)`

### 合法索引

$0 \leq i < \text{size}()$

### size()

不是最后一个索引

合法条件始终是 `index >= 0 && index < list.size()`。不要写成小于等于 `size`。

## L25Example09.java • 完整独立示例

```
import java.util.*;
public class L25Example09 {
    static String safeGet(List<String> wave, int i) {
        return wave != null && i >= 0 && i < wave.size() ? wave.get(i) : null;
    }
    public static void main(String[] args) {
        List<String> wave = new ArrayList<>();
        System.out.println(safeGet(wave, 0));
        wave.add("Spider");
        System.out.println(safeGet(wave, 0));
        System.out.println(safeGet(wave, 1));
    }
}
```

## 运行核对

```
null
Spider
null
```

## 验收对照表

| size | 合法索引 | 说明                     |
|------|------|------------------------|
| 0    | 无    | <code>get(0)</code> 越界 |
| 1    | 0    | 只有第一项                  |
| 3    | 0到2  | 最后是size减1              |

## 10 方法优先接收 List 合同

若方法只需要 List 的通用操作，参数类型可写 `List<String>`，调用者仍能传入 `ArrayList`。

### 调用者可选择不同列表实现

#### 参数 List

依赖通用合同

#### 对象 ArrayList

提供可变实现

#### 返回 false

非法请求不修改

参数写 `List` 表示方法不依赖 `ArrayList` 的特殊实现。创建对象时仍可使用 `new ArrayList`。

## 10 方法优先接收 List 合同 (续)

### L25Example10.java • 完整独立示例

```
import java.util.*;
public class L25Example10 {
    static boolean addMob(List<String> wave, String mob) {
        if (wave == null || mob == null || mob.isBlank()) return false;
        wave.add(mob.strip());
        return true;
    }
    static boolean replace(List<String> wave, int index, String mob) {
        if (wave == null || index < 0 || index >= wave.size()
            || mob == null || mob.isBlank()) return false;
        wave.set(index, mob.strip());
        return true;
    }
    static int removeAll(List<String> wave, String mob) {
        if (wave == null || mob == null || mob.isBlank()) return 0;
        String target = mob.strip();
        int before = wave.size();
        wave.removeIf(target::equals);
        return before - wave.size();
    }
    static void show(List<String> wave) {
        if (wave == null) { System.out.println("没有队列"); return; }
        for (int i = 0; i < wave.size(); i++)
            System.out.println(i + ": " + wave.get(i));
    }
    public static void main(String[] args) {
        List<String> wave = new ArrayList<>();
        System.out.println(addMob(wave, " Spider "));
        System.out.println(replace(wave, 1, "Zombie"));
        System.out.println(wave);
    }
}
```

### 运行核对

```
true
false
[Spider]
```

# 11 三类测试动态队列

动态容器改变频繁，测试要同时检查返回值、大小和元素顺序。

## 添加 替换 删除都验证状态

### 前快照

复制元素序列

### 执行操作

一次只测试一项

### 后比较

返回值、大小、顺序

失败操作前复制一份列表，操作后比较内容。返回 false 但数据变了，仍然是严重 Bug。

## 验收对照表

| 场景 | 操作       | 预期           |
|----|----------|--------------|
| 正常 | 添加Zombie | size增加1      |
| 正常 | 替换索引1    | size不变 元素改变  |
| 边界 | 删除最后一个   | 成功且顺序正确      |
| 错误 | 索引等于size | 返回false 数据不变 |
| 错误 | 添加空名字    | 返回false 数据不变 |

## 12 挑战 蓝图动态波次编辑器

挑战 程序叫 DynamicWave。它接收 List<String> 参数，主程序用 ArrayList 保存波次。

### 添加 替换 删除和显示组成编辑流程

#### addMob

只负责添加

#### replace

只负责替换

#### removeAll

按名字删除全部

addMob 只添加，replace 只替换，removeAll 只按名过滤，show 只显示。职责清楚才容易定位 Bug。

### 验收顺序

- 依次加入 Zombie、Skeleton、Zombie。
- 把1号元素替换为 Spider，大小仍为3。
- 删除全部 Zombie，应删除2只。
- 最后显示0号位置为 Spider。
- 非法索引和空名字返回 false，列表内容与顺序不变。

# 13 挑战 实现动态波次

新建 DynamicWave.java, 使用 Java 21 编译运行。

### 泛型列表通过方法安全编辑

#### 添加三项

Zombie、Skeleton、Zombie

#### 替换中间

改为 Spider

#### 过滤僵尸

剩下一个 Spider

把 wave 改为空列表运行；添加空字符串；删除不存在的女巫；确认失败操作不改变列表。

## 13 挑战 实现动态波次（续）

### DynamicWave.java • 完整独立示例

```
import java.util.*;
public class DynamicWave {
    static boolean addMob(List<String> wave, String mob) {
        if (wave == null || mob == null || mob.isBlank()) return false;
        wave.add(mob.strip());
        return true;
    }
    static boolean replace(List<String> wave, int index, String mob) {
        if (wave == null || index < 0 || index >= wave.size()
            || mob == null || mob.isBlank()) return false;
        wave.set(index, mob.strip());
        return true;
    }
    static int removeAll(List<String> wave, String mob) {
        if (wave == null || mob == null || mob.isBlank()) return 0;
        String target = mob.strip();
        int before = wave.size();
        wave.removeIf(target::equals);
        return before - wave.size();
    }
    static void show(List<String> wave) {
        if (wave == null) { System.out.println("没有队列"); return; }
        for (int i = 0; i < wave.size(); i++)
            System.out.println(i + ": " + wave.get(i));
    }
    public static void main(String[] args) {
        List<String> wave = new ArrayList<>();
        addMob(wave, "Zombie"); addMob(wave, "Skeleton");
        addMob(wave, "Zombie");
        System.out.println("替换=" + replace(wave, 1, "Spider"));
        System.out.println("删除=" + removeAll(wave, "Zombie"));
        System.out.println("拒绝空名=" + !addMob(wave, " "));
        show(wave);
    }
}
```

### 运行核对

替换=true  
删除=2  
拒绝空名=true  
0: Spider

List 合同并不承诺允许增删。本项目要求传入可变列表，例如 ArrayList；非法索引与空名返回失败，不修改数据。

# 14 拓展任务与单元复盘

第21至25课完成了继承、多态、接口、值对象和动态集合。作业同时提交代码、测试表和一次故障记录。

### 三档任务继续扩建数据仓库

#### 类型设计

继承、接口、值对象

#### 容器操作

泛型、索引、增删

#### 交付证据

代码与测试记录

第26课用 HashMap 管理玩家计分。名字当键，分数当值，不再为第100位玩家创建第100个变量。

## 三档作业

- 木镐：用 ArrayList 管理任务清单，支持添加、完成和显示。
- 铁镐：写 removeAllByName，删除全部同名生物并返回删除数量。
- 钻石镐：用 Collections.shuffle 随机打乱波次，并保留一份原顺序副本用于比较。

### 验收对照表

| 自评       | 能                        | 还要练                      |
|----------|--------------------------|--------------------------|
| 我会使用泛型列表 | <input type="checkbox"/> | <input type="checkbox"/> |
| 我能安全增删元素 | <input type="checkbox"/> | <input type="checkbox"/> |
| 我会测试列表边界 | <input type="checkbox"/> | <input type="checkbox"/> |

# 自己的验收记录

漂亮的输出是起点，能解释边界才算掌握。把正常、边界与错误各选一个案例，运行前写预期，运行后留下证据。

## 为判断留下依据

| 类别 | 你选择的操作        | 核对重点          |
|----|---------------|---------------|
| 正常 | 一个常用输入或一次成功动作 | 结果是否逐行一致      |
| 边界 | 范围端点与相邻值      | 有没有多一步、少一项    |
| 错误 | 非法数量、缺失对象或坏数据 | 是否明确拒绝，状态是否保持 |

- Bug 记录：触发条件、实际现象、原因、最小修复。
- 图示记录：为每条箭头说明起点、终点和含义。
- 改编记录：只改一个规则，同时修改预期和测试。

技术核对：Java SE 21 语言规范与标准库。类、接口、类型与表达式规则见语言规范；集合、Random、Files 与 Stream 的行为见对应 API。  
<https://docs.oracle.com/javase/specs/jls/se21/html/index.html>  
<https://docs.oracle.com/en/java/javase/21/docs/api/index.html>