

第26课

计分板与 HashMap

按名字查分，安全更新计分板

玩家名字连接自己的分数

key

玩家名，唯一查找标识

value

分数，可重复

entry

一条键值映射

服务器里来了新玩家，计分板无需再添一个变量。以名字为键，读出分数，再用明确规则处理缺失、扣分、排序与整数上限。

前置背包：变量、方法、循环、ArrayList 和泛型。今天让第100位玩家加入时也不必创建 player100Score。

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 键和值组成查找表

Map 像服务器计分板。玩家名字是键，分数是值；程序通过名字快速找到对应分数。

同一个键只保留一份当前值。两个不同玩家可以恰好拥有相同分数。

玩家名字连接自己的分数

key

玩家名，唯一查找标识

value

分数，可重复

entry

一条键值映射

先问谁负责查找，再问要保存什么。这里用玩家名查分数，所以名字当键，分数当值。

验收对照表

概念	计分板例子	规则
键 key	玩家名 Alex	在同一 Map 中唯一
值 value	分数 12	允许重复
映射 entry	Alex 对应 12	一组键和值

02 创建 HashMap 并放入数据

声明时写出键和值的类型。尖括号中的 String, Integer 表示只能使用字符串键和整数值。

put 添加或替换 get 按键查询

Map<String,Integer>

明确键值类型

put

同名键替换旧值

size

当前记录数量

需要 import java.util.HashMap; 和 import java.util.Map;。变量写 Map，创建对象写 HashMap。

L26Example02.java • 完整独立示例

```
import java.util.*;
public class L26Example02 {
    public static void main(String[] args) {
        Map<String, Integer> scores = new HashMap<>();
        scores.put("Alex", 12); scores.put("Steve", 12);
        scores.put("Alex", 20);
        System.out.println(scores.get("Alex"));
        System.out.println(scores.size());
    }
}
```

运行核对

```
20
2
```

故障实验：值类型是 Integer，字符串 "12" 不是整数12。下面故意保留错误，请先预测，再尝试修复。

BadMapType.java • 故意错误示例

```
import java.util.*;
public class BadMapType {
    public static void main(String[] args) {
        Map<String, Integer> scores = new HashMap<>();
        scores.put("Alex", "12");
    }
}
```

03 安全累加分数

同名 put 会替换旧值，记录数量不增加。新键的 get 返回 null，直接拆箱相加会失败。先看简单计数，再给大数、null 与非法旧记录建立明确合同。

getOrDefault 和 merge 都能处理新玩家

不存在

getOrDefault 给默认0

已存在 null

默认值不会代替它

安全加法

大数先转 long 检查

本页小数值示例不含 null；getOrDefault 只处理缺失键。已有 null、负分或累加溢出要按第7节合同处理。

L26Example03.java • 完整独立示例

```
import java.util.*;
public class L26Example03 {
    public static void main(String[] args) {
        Map<String, Integer> scores = new HashMap<>();
        scores.put("Alex", scores.getOrDefault("Alex", 0) + 5);
        scores.merge("Alex", 3, Integer::sum);
        System.out.println(scores.get("Alex"));
    }
}
```

运行核对

8

本例保证记录没有 null 且小整数相加不溢出。getOrDefault 只为“键不存在”给默认值，不会替换已存在的 null。大数累加要用第7节的 long 校验，不能直接照抄加法。

04 不存在的键要明确处理

get 对不存在的键返回 null。计分板可以选择显示0，也可以明确告诉用户找不到玩家。

containsKey 区分没有玩家和分数为0

get

缺失键返回 null

containsKey

判断键是否存在

0分玩家

存在与缺失不同

查询方法到底返回0、null还是提示文字，应在方法说明里写清楚。调用者不该靠猜。

L26Example04.java • 完整独立示例

```
import java.util.*;
public class L26Example04 {
    public static void main(String[] args) {
        Map<String, Integer> scores = new HashMap<>();
        scores.put("Alex", 0);
        System.out.println(scores.get("Mia"));
        System.out.println(scores.containsKey("Mia"));
        System.out.println(scores.containsKey("Alex"));
        System.out.println(scores.containsKey("Mia"));
    }
}
```

运行核对

```
null
0
true
false
```

05 entrySet 一次拿到键和值

只遍历 keySet 再调用 get 也能工作，但 entrySet 直接提供完整记录，表达更清楚。

遍历每条计分记录

entrySet

同时拿到键与值

HashMap

不承诺遍历顺序

TreeMap 副本

按键生成稳定展示

HashMap 的遍历顺序没有承诺。测试内容时不要把加入顺序误当成固定输出顺序。

L26Example05.java • 完整独立示例

```
import java.util.*;
public class L26Example05 {
    public static void main(String[] args) {
        Map<String, Integer> scores = new HashMap<>(Map.of("Steve", 8, "Alex", 12));
        for (var entry : new TreeMap<>(scores).entrySet())
            System.out.println(entry.getKey() + "=" + entry.getValue());
    }
}
```

运行核对

Alex=12
Steve=8

06 删除玩家并理解顺序

`remove(key)` 删除整条记录，并返回旧值；键不存在时返回 `null`。`clear()` 会清空整张表。

remove 返回旧值 HashMap 不排队

`remove(key)`

删除并返回旧值

`clear()`

清空全部记录

展示顺序

需明确选择实现

需要按加入顺序遍历可了解 `LinkedHashMap`；需要按键排序可了解 `TreeMap`。本课 挑战 不依赖 `HashMap` 顺序。

L26Example06.java • 完整独立示例

```
import java.util.*;
public class L26Example06 {
    public static void main(String[] args) {
        Map<String, Integer> scores = new HashMap<>(Map.of("Alex", 12));
        System.out.println(scores.remove("Alex"));
        System.out.println(scores.remove("Alex"));
        scores.clear();
        System.out.println(scores.size());
    }
}
```

运行核对

```
12
null
0
```

07 方法参数面向 Map 合同

方法若只需要 Map 的通用操作，参数应写 `Map<K,V>`，而不是锁死为 `HashMap`。

调用者可选择不同 Map 实现

名字检查

拒绝空名，去首尾空白

long 计算

防止 int 相加溢出

更新最后

拒绝时数据不变

空 Map、空名字和扣成负数都要提前决定规则。示例拒绝空输入，并把最低分限制为0。

07 方法参数面向 Map 合同 (续)

L26Example07.java • 完整独立示例

```
import java.util.*;
public class L26Example07 {
    static boolean addScore(Map<String, Integer> scores,
                           String player, int delta) {
        if (scores == null || player == null || player.isBlank()) return false;
        String key = player.strip();
        Integer old = scores.get(key);
        if (scores.containsKey(key) && (old == null || old < 0)) return false;
        int current = old == null ? 0 : old;
        long next = (long) current + delta;
        if (next > Integer.MAX_VALUE) return false;
        scores.put(key, (int) Math.max(0L, next));
        return true;
    }
    static int getScore(Map<String, Integer> scores, String player) {
        if (scores == null || player == null || player.isBlank()) return 0;
        String key = player.strip();
        Integer value = scores.get(key);
        if (!scores.containsKey(key)) return 0;
        if (value == null || value < 0)
            throw new IllegalStateException("计分记录损坏: " + key);
        return value;
    }
    public static void main(String[] args) {
        Map<String, Integer> scores = new HashMap<>();
        System.out.println(addScore(scores, " Alex ", 5));
        System.out.println(addScore(scores, "Alex", -20));
        System.out.println(getScore(scores, "Alex"));
        scores.put("Alex", Integer.MAX_VALUE);
        System.out.println(addScore(scores, "Alex", 1));
        System.out.println(getScore(scores, "Alex"));
    }
}
```

运行核对

```
true
true
0
false
2147483647
```

先把旧分数提升为 long 再加 delta，越过 int 上限就拒绝且不写入。低于0夹紧到0是计分规则；传入已含 null 或负值的损坏记录则明确拒绝。

08 排行榜先复制再排序

HashMap 适合按名字查分，不适合表达第1名、第2名。制作排行榜时，把记录复制到列表再排序。

Map 负责查找 List 负责有序展示

Map.entry

复制不可修改值条目

按分数降序

确定排名

同分按名字

输出可重复

排序前用 Map.entry 创建独立的不可修改条目，且名字、分数不可变。同分按名字排序；只复制 entrySet 的引用不是数值快照。

L26Example08.java • 完整独立示例

```
import java.util.*;
public class L26Example08 {
    public static void main(String[] args) {
        Map<String, Integer> scores = new HashMap<>(Map.of("Steve", 8, "Alex", 8));
        List<Map.Entry<String, Integer>> ranking = new ArrayList<>();
        scores.forEach((name, score) -> ranking.add(Map.entry(name, score)));
        ranking.sort(Map.Entry.<String, Integer>comparingByValue().reversed()
            .thenComparing(Map.Entry.comparingByKey()));
        scores.put("Alex", 99);
        for (var entry : ranking)
            System.out.println(entry.getKey() + "=" + entry.getValue());
    }
}
```

运行核对

Alex=8
Steve=8

new ArrayList<>(scores.entrySet()) 只复制条目引用，并非独立数值快照。本例用 Map.entry 建立不可修改条目，且 String、Integer 都不可变；同分按名字排序，输出可重复。前提是计分板不含 null。

09 三类测试守住计分板

测试加分方法时，不只看一行输出。还要检查失败操作是否偷偷改变了 Map。

正常 边界 错误输入都要留下证据

正常加分

旧值增加

最低0分

扣分夹紧

失败检查

与操作前快照相等

失败操作最重要的证据常常是数据保持原样。先拍快照，再比较内容。

验收对照表

场景	操作	预期
正常	Alex加5	分数增加5
边界	现有0分玩家扣1	仍为0
错误	空名字加5	返回false 数据不变
错误	null Map	返回false 不抛异常
查询	不存在玩家	返回0

09 三类测试守住计分板（续）

L26Example09.java • 完整独立示例

```
import java.util.*;
public class L26Example09 {
    static boolean addScore(Map<String, Integer> scores,
                           String player, int delta) {
        if (scores == null || player == null || player.isBlank()) return false;
        String key = player.strip();
        Integer old = scores.get(key);
        if (scores.containsKey(key) && (old == null || old < 0)) return false;
        int current = old == null ? 0 : old;
        long next = (long) current + delta;
        if (next > Integer.MAX_VALUE) return false;
        scores.put(key, (int) Math.max(0L, next));
        return true;
    }
    static int getScore(Map<String, Integer> scores, String player) {
        if (scores == null || player == null || player.isBlank()) return 0;
        String key = player.strip();
        Integer value = scores.get(key);
        if (!scores.containsKey(key)) return 0;
        if (value == null || value < 0)
            throw new IllegalStateException("计分记录损坏: " + key);
        return value;
    }
    public static void main(String[] args) {
        Map<String, Integer> scores = new HashMap<>(Map.of("Alex", 12));
        Map<String, Integer> before = new HashMap<>(scores);
        boolean ok = addScore(scores, " ", 5);
        System.out.println(!ok && scores.equals(before));
        System.out.println(getScore(scores, "Mia"));
    }
}
```

运行核对

```
true
0
```

10 挑战 蓝图玩家计分板

挑战 程序叫 ScoreBoard。它要支持新玩家、老玩家加减分、安全查询和删除，同时不依赖 HashMap 的遍历顺序。

添加 扣分 查询 删除形成完整流程

添加玩家

不存在时从0起步

查询缺失

本课返回0

删除玩家

映射完整移除

挑战 直接按名字查询结果，不用整张 Map 的遍历顺序当验收依据。

验收顺序

- Alex 从12加5，查询得到17。
- Steve 从8扣20，最低分夹紧为0。
- 查询缺失 Mia 返回0，但 containsKey 可辨认缺失。
- 空名字操作返回 false，计分板不变。
- 删除 Steve 返回旧分数0，最后剩1名玩家。

11 计分板程序

新建 ScoreBoard.java, 使用 Java 21 编译运行。

Map 合同封装安全的计分操作

Alex

12加5得到17

Steve

8扣20得到0

删除 Steve

最后剩1名玩家

把Alex连续扣20分，确认不会低于0；再把 Map 换成 TreeMap，方法无需修改。

11 计分板程序 (续)

ScoreBoard.java • 完整独立示例

```
import java.util.*;
public class ScoreBoard {
    static boolean addScore(Map<String, Integer> scores,
                           String player, int delta) {
        if (scores == null || player == null || player.isBlank()) return false;
        String key = player.strip();
        Integer old = scores.get(key);
        if (scores.containsKey(key) && (old == null || old < 0)) return false;
        int current = old == null ? 0 : old;
        long next = (long) current + delta;
        if (next > Integer.MAX_VALUE) return false;
        scores.put(key, (int) Math.max(0L, next));
        return true;
    }
    static int getScore(Map<String, Integer> scores, String player) {
        if (scores == null || player == null || player.isBlank()) return 0;
        String key = player.strip();
        Integer value = scores.get(key);
        if (!scores.containsKey(key)) return 0;
        if (value == null || value < 0)
            throw new IllegalStateException("计分记录损坏: " + key);
        return value;
    }
    public static void main(String[] args) {
        Map<String, Integer> scores = new HashMap<>();
        addScore(scores, "Alex", 12); addScore(scores, "Steve", 8);
        addScore(scores, "Alex", 5); addScore(scores, "Steve", -20);
        System.out.println("Alex=" + getScore(scores, "Alex"));
        System.out.println("Steve=" + getScore(scores, "Steve"));
        System.out.println("Mia=" + getScore(scores, "Mia"));
        System.out.println("空名=" + addScore(scores, " ", 5));
        System.out.println("删除=" + scores.remove("Steve"));
        System.out.println("玩家数=" + scores.size());
    }
}
```

运行核对

Alex=17
Steve=0
Mia=0
空名=false
删除=0
玩家数=1

12 拓展任务与通关复盘

作业同时提交代码、测试表和一次故障记录。解释为什么选择某个字段当键。

三档任务扩建服务器统计

选择键

说明查找目的

定义失败

缺失、空名、溢出

验证顺序

不能依赖 HashMap 次序

第27课用 Set 记录不重复成就，再用 Random 和概率区间设计一张可测试的战利品表。

三档作业

- 木镐：建立物品价格表，支持新增、改价和查询。
- 铁镐：统计一组掉落物的出现次数。
- 钻石镐：把计分记录复制到列表，输出分数最高的前三名。

验收对照表

自评	能	还要练
我会设计键和值	☐	☐
我能安全查询缺失键	☐	☐
我不依赖HashMap顺序	☐	☐

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>