

第30课

方块数据流水线

用 Lambda 与 Stream 生成挖矿简报

参数箭头表达式写出临时规则

Lambda

把行为作为值传递

Predicate

一个参数，返回真假

BiFunction

两个参数，返回结果

矿车带来一批记录：筛选、转换、排序、统计，各走一段管道。用清楚的行为规则组织数据，再手算答案，让简报的每个数字有证据。

前置背包：List、Map、record、方法引用和循环。今天让挖矿数据乘上矿车，依次经过筛选、加工、排序和统计站。

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 Lambda 是一小段行为

Lambda 不是新型方法，而是把一段可传递的行为写得更短。ore -> ore.length() >= 2 表示接收 ore，再回答条件是否成立。

参数箭头表达式写出临时规则

Lambda

把行为作为值传递

Predicate

一个参数，返回真假

BiFunction

两个参数，返回结果

把箭头读成“给我左边参数，我执行右边规则”。先写完整方法，再缩成 Lambda，通常更容易理解。

L30Example01.java • 完整独立示例

```
public class L30Example01 {  
    public static void main(String[] args) {  
        java.util.function.Predicate<String> longName = name -> name.length() >= 2;  
        java.util.function.BiFunction<Integer, Integer, Integer> sum = (x, y) -> x + y;  
        System.out.println(longName.test("钻石"));  
        System.out.println(sum.apply(3, 4));  
    }  
}
```

运行核对

```
true  
7
```

Lambda 需要目标函数式接口：它只有一个抽象方法。这里 Predicate 接受一个参数返回真假，BiFunction 接受两个参数返回结果；default 等方法不计入这一个抽象方法。

02 Stream 管道分三段

Stream 描述数据处理方式：来源提供元素，中间操作搭建步骤，终止操作请求结果。它不会像 List 那样保存一份可反复读取的数据。

来源经过中间操作最后由终止操作启动



终止操作请求计算结果。实现可能省略不影响答案的步骤，不要依赖 filter 或 map 里的副作用做日志或计数。

L30Example02.java • 完整独立示例

```
import java.util.*;
public class L30Example02 {
    public static void main(String[] args) {
        List<String> ores = List.of("煤", "钻石", "铁矿石");
        var pipeline = ores.stream().filter(name -> name.length() >= 2);
        System.out.println(pipeline.toList());
    }
}
```

运行核对

[钻石, 铁矿石]

中间操作是惰性的；终止操作请求结果。实现还可以优化、跳过不影响结果的步骤，不要用 filter 或 map 中的打印、计数副作用判断是否一定逐项执行。

验收对照表

阶段	例子	作用
来源	ores.stream	提供元素
中间操作	filter map sorted	描述处理
终止操作	toList sum count	产生结果

03 filter 只保留匹配元素

filter 表达保留条件：满足条件的元素进入结果。筛选不删除原列表成员，也不应在判断条件时偷偷修改来源。

Predicate 返回 true 的记录继续前进

filter

按真假选择元素

新结果

用终止操作接住

原 List

不会因此被删除

filter 返回新的流，不会从原 List 删除元素。要保存结果，必须接住终止操作的返回值。

L30Example03.java • 完整独立示例

```
import java.util.*;
record MinedBlock(String ore, int amount, int value) {
    MinedBlock {
        if (ore == null || ore.isBlank() || amount < 0 || amount > 1_000_000
            || value < 0 || value > 1_000_000)
            throw new IllegalArgumentException("检查矿物名、数量与单价");
        ore = ore.strip();
    }
}
public class L30Example03 {
    public static void main(String[] args) {
        List<MinedBlock> blocks = List.of(new MinedBlock("煤", 2, 1),
            new MinedBlock("钻石", 3, 10));
        var rare = blocks.stream().filter(b -> b.value() >= 5).toList();
        System.out.println(rare.size());
        System.out.println(blocks.size());
    }
}
```

运行核对

1
2

04 map 把元素变成另一种形态

map 像工作台配方。每个输入元素被转换一次，数量通常不变，但类型可以改变。

记录可变成名称 标签或数值

map

把元素转换成新形态

返回类型

可以与来源不同

HashMap

是键值容器，概念不同

Stream 的 map 是转换操作，与 HashMap 不是同一概念。一个是动词，一个是键值容器。

L30Example04.java • 完整独立示例

```
import java.util.*;
public class L30Example04 {
    public static void main(String[] args) {
        List<String> ores = List.of("diamond", "iron");
        System.out.println(ores.stream().map(s -> s.toUpperCase(Locale.ROOT)).toList());
        System.out.println(ores.stream().map(String::length).toList());
    }
}
```

运行核对

```
[DIAMOND, IRON]
[7, 4]
```

Locale.ROOT 让英文标识符的大写不依赖电脑语言。String.length 统计 UTF-16 代码单元，不等同于所有文字的可见字符数；本例英文没有这个差异。

05 sorted 控制报告顺序

自然顺序适合字符串和数字。复杂记录要明确按哪个字段排，分数相同时还可增加第二规则。

Comparator 描述比较规则

排序字段

明确按单价还是小计

第二规则

同分按矿物名

原序列

sorted 不修改来源

Stream sorted 不改变原列表。需要新的有序结果时正合适，也更容易比较前后数据。

L30Example05.java • 完整独立示例

```
import java.util.*;
record MinedBlock(String ore, int amount, int value) {
    MinedBlock {
        if (ore == null || ore.isBlank() || amount < 0 || amount > 1_000_000
            || value < 0 || value > 1_000_000)
            throw new IllegalArgumentException("检查矿物名、数量与单价");
        ore = ore.strip();
    }
}
public class L30Example05 {
    public static void main(String[] args) {
        List<MinedBlock> blocks = List.of(new MinedBlock("B", 1, 5),
            new MinedBlock("A", 1, 5), new MinedBlock("C", 1, 10));
        var names = blocks.stream()
            .sorted(Comparator.comparingInt(MinedBlock::value).reversed()
                .thenComparing(MinedBlock::ore))
            .map(MinedBlock::ore).toList();
        System.out.println(names);
        System.out.println(blocks.get(0).ore());
    }
}
```

运行核对

[C, A, B]
B

06 数值流负责求和与平均

对象流要先映射成数值流才能直接求和。数量可用 `mapToInt`，可能很大的价值使用 `mapToLong`；每次都先确定取值范围。

mapToInt 把对象转换成整数流

long 乘法

先转换再计算

sum

空数值流为0

average

可能没有结果，用 `orElse`

数值流提供求和与平均。本章价值用 `mapToLong`，并在乘法前转换；空流 `sum` 为0，`average` 和 `max` 用 `Optional` 表达可能没有结果。

乘法之前把操作数转成 `long`。先做 `int` 乘法再转 `long`，错误已经发生。单条与记录总数都有上限时，才能证明累加也不溢出。

06 数值流负责求和与平均（续）

L30Example06.java • 完整独立示例

```
import java.util.*;
record MinedBlock(String ore, int amount, int value) {
    MinedBlock {
        if (ore == null || ore.isBlank() || amount < 0 || amount > 1_000_000
            || value < 0 || value > 1_000_000)
            throw new IllegalArgumentException("检查矿物名、数量与单价");
        ore = ore.strip();
    }
}
public class L30Example06 {
    public static void main(String[] args) {
        List<MinedBlock> blocks = List.of(new MinedBlock("钻石", 3, 10),
            new MinedBlock("铁", 7, 3));
        long total = blocks.stream().mapToLong(b -> (long) b.amount() * b.value()).sum();
        System.out.println(total);
        System.out.println(blocks.stream().mapToInt(MinedBlock::amount).average().orElse(0));
        System.out.println(List.<MinedBlock>of().stream()
            .mapToLong(MinedBlock::amount).sum());
    }
}
```

运行核对

```
51
5.0
0
```

07 distinct limit 与 skip

先排序再 limit 可取前三名；先 limit 再排序只会排序原列表的前三个。管道顺序就是算法顺序。

去重和截取塑造最终报告

distinct

按相等规则去重

sorted 与 limit

先后顺序改变答案

skip

跳过指定数量

交换 sorted 与 limit 后运行同一数据，记录输出差异。不要把管道步骤当成随便排列的装饰。

L30Example07.java • 完整独立示例

```
import java.util.*;
public class L30Example07 {
    public static void main(String[] args) {
        List<Integer> values = List.of(3, 1, 3, 2);
        System.out.println(values.stream().distinct().sorted().limit(2).toList());
        System.out.println(values.stream().limit(2).sorted().toList());
        System.out.println(values.stream().distinct().sorted().skip(1).toList());
    }
}
```

运行核对

```
[1, 2]
[1, 3]
[2, 3]
```

08 toList 接住最终结果

Java 21 的 `Stream.toList` 返回不可修改列表。可以读取，但对它 `add` 或 `remove` 会失败。

管道结果通常是新的不可修改列表

toList

返回不可修改列表

可变副本

`new ArrayList<>(result)`

元素状态

容器只读不等于深度不可变

不要假设所有返回 `List` 的方法都允许修改。阅读方法合同，必要时创建明确的可变副本。

故障实验：`toList` 返回不可修改列表，`add` 抛 `UnsupportedOperationException`。下面故意保留错误，请先预测，再尝试修复。

不可修改列表不代表元素也不可变。若元素是可变对象，它们的字段仍可能改变；本章 `MinedBlock` 只含 `String` 与 `int`，因此值本身保持不变。

L30Example08.java • 完整独立示例

```
import java.util.*;
public class L30Example08 {
    public static void main(String[] args) {
        List<String> result = List.of("煤", "钻石").stream().toList();
        List<String> editable = new ArrayList<>(result);
        editable.add("铁");
        System.out.println(result);
        System.out.println(editable);
    }
}
```

运行核对

[煤, 钻石]
[煤, 钻石, 铁]

08 toList 接住最终结果（续）

BadToList.java • 故意错误示例

```
import java.util.*;
public class BadToList {
    public static void main(String[] args) {
        public static void main(String[] args) {
            var result = List.of("煤").stream().toList();
            result.add("铁");
        }
    }
}
```

09 Stream 只能消费一次

Stream 像一次性矿车路线。到达终点后不能倒回起点再跑一次。

终止操作结束后流已经关闭

一次性 Stream

终止后不能再消费

保留来源

List 可反复建立新流

再次报告

从 stream() 重新开始

通常保存 List 等数据来源，而不是把 Stream 长期放进字段。每次报告都从来源建立新管道。

故障实验：流已消费，再次终止操作抛 `IllegalStateException`。下面故意保留错误，请先预测，再尝试修复。

BadStreamReuse.java • 故意错误示例

```
import java.util.*;
public class BadStreamReuse {
    public static void main(String[] args) {
        var stream = List.of("煤", "铁").stream();
        stream.count();
        stream.toList();
    }
}
```

L30Example09.java • 完整独立示例

```
import java.util.*;
public class L30Example09 {
    public static void main(String[] args) {
        List<String> ores = List.of("煤", "铁");
        System.out.println(ores.stream().count());
        System.out.println(ores.stream().toList());
    }
}
```

运行核对

2
[煤, 铁]

10 避免修改外部集合

在 `forEach` 中偷偷修改外部 `List`，数据流向就分成两条。返回结果比隐藏副作用更容易读和测试。

管道步骤应根据输入返回结果

避免隐藏副作用

返回结果表达收集

不要修改来源

遍历合同要求不干扰

`forEach`

适合确实要执行的动作

让终止操作明确返回最终数据。只有真正需要执行动作时才使用 `forEach`。

L30Example10.java • 完整独立示例

```
import java.util.*;
public class L30Example10 {
    public static void main(String[] args) {
        List<String> ores = List.of("煤", "钻石", "铁矿石");
        List<String> rare = ores.stream().filter(s -> s.length() >= 2).toList();
        System.out.println(rare);
    }
}
```

运行核对

[钻石, 铁矿石]

顺序流的 `forEach` 向独立可变容器添加并非语法错误，但副作用使重构与并行更难。用返回值表达收集结果，别在遍历时修改来源集合。

11 复杂 Lambda 提取为方法

Lambda 超过一两行后，提取成有名字的方法。主流程更清楚，规则也能单独测试。

名字说明规则 测试可以单独运行

命名方法

解释复杂选择规则

方法引用

把已命名行为放进管道

独立测试

不用整份报告也能验证

管道显示处理顺序，命名方法解释每条复杂规则。两层信息各做一件事。

L30Example11.java • 完整独立示例

```
import java.util.*;
record MinedBlock(String ore, int amount, int value) {
    MinedBlock {
        if (ore == null || ore.isBlank() || amount < 0 || amount > 1_000_000
            || value < 0 || value > 1_000_000)
            throw new IllegalArgumentException("检查矿物名、数量与单价");
        ore = ore.strip();
    }
}
public class L30Example11 {
    static boolean isRare(MinedBlock b) {
        return b != null && b.value() >= 5;
    }
    public static void main(String[] args) {
        List<MinedBlock> blocks = List.of(new MinedBlock("煤", 2, 1),
            new MinedBlock("钻石", 3, 10));
        System.out.println(blocks.stream().filter(L30Example11::isRare)
            .map(MinedBlock::ore).toList());
    }
}
```

运行核对

[钻石]

12 挑战 蓝图挖矿统计简报

挑战 程序叫 MiningReport。每条记录保存矿物名、数量和单块价值，Stream 生成总方块、稀有列表、总价值和最高价值矿物。

筛选 转换 排序和聚合组成报告

总数量
14+7+4+3+20=48

总价值
14+21+20+30+0=85

最高小计
钻石这条记录为30

先手算总数量和总价值，再运行程序。统计程序最怕输出看起来合理，却没人知道正确答案。

验收条件

- 总方块数量为48。
- 稀有矿物按单块价值从高到低排列。
- 总价值按数量乘单价求和。
- 最高价值矿物按整条记录的总价值选择。
- 空列表时最高价值显示无。

验收对照表

矿物	数量	单价	小计
煤炭	14	1	14
铁矿石	7	3	21
金矿石	4	5	20
钻石	3	10	30
圆石	20	0	0

13 挖矿统计程序

新建 MiningReport.java, 使用 Java 21 编译运行。

Stream 管道生成四项可验证结果

先清理来源

拒绝 null 列表, 过滤空记录



再筛选稀有

单价至少为5



最后稳定排序

单价降序, 再按名字

空列表得到总数0、稀有列表空、最高值“无”。加入同单价和同小计的记录, 检查名字作为第二排序规则是否稳定。

MiningReport.java • 第1/2段, 顺序接在同一文件

```
import java.util.*;
record MinedBlock(String ore, int amount, int value) {
    MinedBlock {
        if (ore == null || ore.isBlank() || amount < 0 || amount > 1_000_000
            || value < 0 || value > 1_000_000)
            throw new IllegalArgumentException("检查矿物名、数量与单价");
        ore = ore.strip();
    }
}
public class MiningReport {
    static boolean isRare(MinedBlock block) {
        return block != null && block.value() >= 5;
    }
    static long totalValue(MinedBlock block) {
        return (long) block.amount() * block.value();
    }
    static List<MinedBlock> clean(List<MinedBlock> blocks) {
        if (blocks == null || blocks.size() > 1_000_000)
            throw new IllegalArgumentException("记录表不能为null且最多100万条");
        return blocks.stream().filter(Objects::nonNull).toList();
    }
    static long totalBlocks(List<MinedBlock> blocks) {
        return clean(blocks).stream().mapToLong(MinedBlock::amount).sum();
    }
    static long valueOfAll(List<MinedBlock> blocks) {
        return clean(blocks).stream().mapToLong(MiningReport::totalValue).sum();
    }
    static List<String> rareLabels(List<MinedBlock> blocks) {
```

13 挖矿统计程序（续）

MiningReport.java • 第2/2段，顺序接在同一文件

```
return clean(blocks).stream().filter(MiningReport::isRare)
    .sorted(Comparator.comparingInt(MinedBlock::value).reversed()
        .thenComparing(MinedBlock::ore))
    .map(block -> block.ore() + "x" + block.amount()).toList();
}
static String bestOre(List<MinedBlock> blocks) {
    return clean(blocks).stream()
        .sorted(Comparator.comparingLong(MiningReport::totalValue).reversed()
            .thenComparing(MinedBlock::ore))
        .map(MinedBlock::ore).findFirst().orElse("无");
}
public static void main(String[] args) {
    List<MinedBlock> blocks = List.of(
        new MinedBlock("煤炭", 14, 1), new MinedBlock("铁矿石", 7, 3),
        new MinedBlock("金矿石", 4, 5), new MinedBlock("钻石", 3, 10),
        new MinedBlock("圆石", 20, 0));
    System.out.println("总方块=" + totalBlocks(blocks));
    System.out.println("稀有=" + rareLabels(blocks));
    System.out.println("总价值=" + valueOfAll(blocks));
    System.out.println("最高价值=" + bestOre(blocks));
}
}
```

运行核对

总方块=48
稀有=[钻石x3, 金矿石x4]
总价值=85
最高价值=钻石

这里“稀有”指单价 ≥ 5 ，“最高价值”按每条记录的数量 $\times$ 单价比较，不是按单价，也不是同矿物合并后的总额。记录最多100万条，每条数量与单价最多100万，总价值不超过 10^{18} ，long可容纳。空列表总数0、稀有列表空、最高值为“无”；空记录被过滤，null来源列表被拒绝。

14 拓展任务与单元复盘

第26至30课完成了键值统计、集合去重、随机模拟、异常恢复、文件存档和数据流水线。

三档任务继续扩建数据流水线

管道解释

说清每一步的输入输出

空结果

总数0，最高为无

边界规则

数量、单价与总记录数

第31课穿过Paper传送门，创建真正的服务器插件，并让第一条欢迎消息在服务器加载时出现。

三档作业

- 木镐：筛选整数列表中的偶数并求和。
- 铁镐：按名称长度排序物品，长度相同再按字母顺序。
- 钻石镐：从玩家战绩生成前五名、平均分和未达标名单。

验收对照表

自评	能	还要练
我会读Stream管道	☐	☐
我能选择filter和map	☐	☐
我会处理空结果	☐	☐

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>