

第28课

给程序穿防爆甲

读懂异常，让失败也有清楚的结果

类型 消息和调用位置组成故障线索

异常类型

说明哪类操作失败

消息

给出具体输入线索

源文件行号

定位自己的代码

用户输入“很多”，程序不必立刻结束。你会读懂异常线索，捕获能够恢复的失败，在写入库存前检查全部条件，并给出可行动的提示。

前置背包：方法、条件判断、Map 和字符串转换。今天玩家输入很多颗钻石时，程序不会像苦力怕一样原地爆炸。

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 异常报告失败的操作

异常不是程序员被扣分的通知，而是运行时发现操作无法正常完成后送出的故障报告。

堆栈跟踪会列出方法调用路径。先看异常类型和消息，再找第一条指向自己源文件的行。

类型 消息和调用位置组成故障线索

异常类型

说明哪类操作失败

消息

给出具体输入线索

源文件行号

定位自己的代码

报错信息是线索。只保留异常类型、消息和自己代码中的行号，就能开始定位。

故障实验：运行时抛 `NumberFormatException`；找到指向 `BadNumber.java` 的堆栈行。下面故意保留错误，请先预测，再尝试修复。

BadNumber.java • 故意错误示例

```
public class BadNumber {  
    public static void main(String[] args) {  
        int amount = Integer.parseInt("很多");  
        System.out.println(amount);  
    }  
}
```

02 try catch 处理可恢复错误

字符串转整数可能失败。若用户可以重新输入，就捕获 `NumberFormatException` 并返回可理解的结果。

危险操作放进 try 对应异常交给 catch

try

仅放可能失败的操作

catch

处理能够恢复的错误

失败返回-1

不与合法正数混淆

try 里只放真正可能失败的操作。范围太大时，catch 很难说明到底哪一步出错。

L28Example02.java • 完整独立示例

```
public class L28Example02 {
    static int parseAmount(String text) {
        if (text == null) return -1;
        try {
            int amount = Integer.parseInt(text.strip());
            return amount > 0 ? amount : -1;
        } catch (NumberFormatException error) { return -1; }
    }
    public static void main(String[] args) {
        System.out.println(parseAmount("12"));
        System.out.println(parseAmount("很多"));
    }
}
```

运行核对

12
-1

这里返回 -1 代表失败，合法值为正整数。别把合法0与“解析失败”混为一谈；若0也合法，可以设计返回结果对象，分别携带成功标志与数值。

03 捕获具体异常

捕获 Exception 虽然省事，却可能把程序错误误装成用户输入问题。优先捕获你能明确处理的具体类型。

不同故障需要不同修理方案

具体异常

NumberFormatException 在前

父类异常

IllegalArgumentException 在后

不可恢复

继续传播，别假装成功

catch 应该恢复、转换或补充上下文。若无法正确处理，就让异常继续传播。

L28Example03.java • 完整独立示例

```
public class L28Example03 {
    public static void main(String[] args) {
        for (String text : new String[]{"很多", "80"}) {
            try {
                int amount = Integer.parseInt(text);
                if (amount < 1 || amount > 64)
                    throw new IllegalArgumentException(
                        "单次数量必须在1到64之间: " + amount);
                System.out.println("数量合法=" + amount);
            } catch (NumberFormatException error) {
                System.out.println("数量必须是整数: " + text);
            } catch (IllegalArgumentException error) {
                System.out.println(error.getMessage());
            }
        }
    }
}
```

运行核对

数量必须是整数: 很多

单次数量必须在1到64之间: 80

NumberFormatException 是 IllegalArgumentException 的子类，必须先捕获较具体的它。顺序反过来，后面的 catch 无法到达，编译器会报错。

04 普通验证不必抛异常

玩家输入空名字是常见业务情况，用 if 可以直接表达验证规则。异常用于不能正常完成的操作；选择方式取决于调用合同，而非把异常当作普通分支循环。

可预期选择用 if 处理 真正失败再交给异常

常见验证

空名先用 if

清楚合同

输入怎样才合法

无副作用

通过检查才操作

正常流程中经常出现的情况用 if；库或操作真正无法完成时再考虑异常。

L28Example04.java • 完整独立示例

```
public class L28Example04 {  
    static boolean validName(String name) { return name != null && !name.isBlank(); }  
    public static void main(String[] args) {  
        System.out.println(validName(" "));  
        System.out.println(validName(" Alex "));  
    }  
}
```

运行核对

```
false  
true
```

05 throw 拒绝坏参数

方法若要求单次数量为1到64，就在入口处检查。throw 把错误交给调用者，而不是继续保存坏数据。

方法发现合同被破坏就立即报告

入口检查

参数不合法就拒绝

throw

交给调用者处理

消息

包含范围与收到的值

只写“错误”没有帮助。写出参数名、收到的值和合法范围，修复会快很多。

L28Example05.java • 完整独立示例

```
public class L28Example05 {
    static void checkAmount(int amount) {
        if (amount < 1 || amount > 64)
            throw new IllegalArgumentException("数量应为1到64: " + amount);
    }
    public static void main(String[] args) {
        try { checkAmount(-1); }
        catch (IllegalArgumentException e) { System.out.println(e.getMessage()); }
        checkAmount(64);
        System.out.println("64通过");
    }
}
```

运行核对

数量应为1到64: -1
64通过

06 finally 与资源清理

收尾要沿着控制路径执行。finally 适合普通收尾；需要关闭的资源优先使用 try-with-resources，避免成功与失败分支各写一遍。

正常返回或抛出异常时，沿控制路径收尾

finally

控制路径上的收尾

自动资源管理

try-with-resources

不要 return

避免覆盖原异常或结果

finally 在正常和异常的控制路径上收尾，但进程终止等情况没有保证。不要在其中 return，以免遮蔽原异常或返回值。

L28Example06.java • 完整独立示例

```
import java.util.*;
public class L28Example06 {
    public static void main(String[] args) {
        try { Integer.parseInt("很多"); }
        catch (NumberFormatException e) { System.out.println("需要整数"); }
        finally { System.out.println("本次尝试结束"); }
        try (Scanner scanner = new Scanner("3")) {
            System.out.println(scanner.nextInt());
        }
    }
}
```

运行核对

需要整数
本次尝试结束
3

finally 在正常返回和异常退出的控制路径上执行；JVM 被终止、进程崩溃等情况下没有保证。别在 finally 中 return。这里关闭的是字符串 Scanner；关闭包装 System.in 的 Scanner 会一起关闭输入流。

07 受检与非受检异常

受检异常会被编译器要求捕获或在方法签名中声明。非受检异常通常表示参数或程序状态不符合合同。

编译器要求处理部分可恢复故障

IOException

编译器要求处理或声明

RuntimeException

非受检异常

检查方式

不同不代表重要性不同

两类异常是编译检查方式不同，不代表一种重要、另一种不重要。

L28Example07.java • 完整独立示例

```
public class L28Example07 {
    static void read() throws java.io.IOException {
        throw new java.io.IOException("演示读取失败");
    }
    public static void main(String[] args) {
        try { read(); }
        catch (java.io.IOException e) { System.out.println(e.getMessage()); }
    }
}
```

运行核对

演示读取失败

验收对照表

类别	例子	常见处理
受检异常	IOException	捕获或 throws
非受检异常	IllegalArgumentException	修正参数合同
非受检异常	NumberFormatException	用户输入可捕获

08 自定义异常表达领域故障

当多个背包规则都可能失败时，自定义异常能让调用者明确知道这是库存领域的问题。

InventoryException 说明背包操作失败

InventoryException

表达一类领域失败

extends Exception

这里设计为受检异常

调用者

明确选择如何恢复

只有调用者需要统一区分和处理一类领域失败时，才创建自定义异常。单个简单参数错误可直接用标准类型。

L28Example08.java • 完整独立示例

```
class InventoryException extends Exception {
    InventoryException(String message) { super(message); }
}
public class L28Example08 {
    static int withdraw(int stock, int amount) throws InventoryException {
        if (stock < 0 || amount <= 0 || amount > stock)
            throw new InventoryException("库存不足");
        return stock - amount;
    }
    public static void main(String[] args) {
        try { withdraw(0, 1); }
        catch (InventoryException e) { System.out.println(e.getMessage()); }
    }
}
```

运行核对

库存不足

09 错误消息要能行动

好消息让使用者知道发生了什么和下一步怎么改，也让程序员能找到对应操作。

输入 操作和合法范围都写清楚

发生了什么

添加操作失败

哪个输入

物品名或数量

下一步

提供合法范围与修正建议

面向学生的提示使用清楚中文；调试日志可保留异常类型和原因。两者不要混成一大串堆栈。

验收对照表

消息	问题	改进
错误	没有上下文	数量必须是整数：很多
失败了	没有操作名	添加钻石失败：物品名为空
范围不对	没有合法范围	数量必须在1到64之间：80

10 异常路径也要测试

只测试正常输入等于只给盔甲涂颜色。真正的防爆甲要在错误路径上承受冲击。

成功 捕获 抛出和状态不变都要验收

正常操作

数量增加

解析失败

中文数量不能转整数

状态比较

任何失败都不改库存

失败前复制库存，失败后比较 Map。提示正确但数据被改坏，仍然没有通关。

L28Example10.java • 第1/2段，顺序接在同一文件

```
import java.util.*;

public class L28Example10 {
    static void addItem(Map<String, Integer> inventory,
                        String item, int amount) {
        if (inventory == null)
            throw new IllegalArgumentException("库存不能为null");
        if (item == null || item.isBlank())
            throw new IllegalArgumentException("物品名不能为空");
        if (amount < 1 || amount > 64)
            throw new IllegalArgumentException("单次数量必须在1到64之间: " + amount);
        String key = item.strip();
        Integer old = inventory.get(key);
        if (inventory.containsKey(key) && (old == null || old < 0))
            throw new IllegalArgumentException("原库存记录不合法: " + key);
        long total = (long) (old == null ? 0 : old) + amount;
        if (total > Integer.MAX_VALUE)
            throw new IllegalArgumentException("累计库存超出整数上限: " + key);
        inventory.put(key, (int) total);
    }

    static String process(Map<String, Integer> inventory,
                         String item, String amountText) {
        if (item == null || item.isBlank()) return "物品名不能为空";
        if (amountText == null || amountText.isBlank()) return "数量不能为空";
        String key = item.strip();
        try {
            int amount = Integer.parseInt(amountText.strip());
            addItem(inventory, key, amount);
            return key + " 添加成功 数量=" + inventory.get(key);
        }
    }
}
```

10 异常路径也要测试（续）

L28Example10.java • 第2/2段，顺序接在同一文件

```
        } catch (NumberFormatException error) {
            return "数量必须是整数: " + amountText;
        } catch (IllegalArgumentException error) {
            return error.getMessage();
        }
    }
    public static void main(String[] args) {
        Map<String, Integer> inventory = new HashMap<>(Map.of("钻石", 3));
        Map<String, Integer> before = new HashMap<>(inventory);
        System.out.println(process(inventory, "钻石", "-2"));
        System.out.println(inventory.equals(before));
    }
}
```

运行核对

单次数量必须在1到64之间: -2
true

验收对照表

输入	路径	预期
钻石 3	成功	数量增加3
钻石 很多	解析失败	提示整数 数据不变
钻石 -2	范围失败	提示1到64 数据不变
空名字 5	验证失败	提示名称 数据不变
钻石 64	边界成功	允许添加

11 挑战 蓝图安全背包管理员

挑战 程序叫 SafeInventory。每次操作返回一条友好消息，只有通过全部检查才修改库存。

输入处理：先检查，后修改



process 负责把输入变成操作结果，addItem 负责维护库存合同。异常在能补充上下文的位置处理。

验收顺序

- 钻石3添加成功，累计数量为3。
- 数量文本“很多”提示必须是整数，库存不变。
- 数量-2提示单次范围1到64，库存不变。
- 空名字在解析之前被拒绝，库存不变。
- 再添加钻石64，累计为67；区分单次上限与累计库存。

12 安全背包程序

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

具体异常得到具体提示

单次1到64

每个请求的数量范围

累计库存

可超过64, 受 int 上限约束

友好结果

成功或明确拒绝

把数量改为 null、空串、超大整数和65；每次比较操作前后的库存，确认拒绝请求后状态没有改变。

SafeInventory.java • 第1/2段, 顺序接在同一文件

```
import java.util.*;
public class SafeInventory {
    static void addItem(Map<String, Integer> inventory,
                        String item, int amount) {
        if (inventory == null)
            throw new IllegalArgumentException("库存不能为null");
        if (item == null || item.isBlank())
            throw new IllegalArgumentException("物品名不能为空");
        if (amount < 1 || amount > 64)
            throw new IllegalArgumentException("单次数量必须在1到64之间: " + amount);
        String key = item.strip();
        Integer old = inventory.get(key);
        if (inventory.containsKey(key) && (old == null || old < 0))
            throw new IllegalArgumentException("原库存记录不合法: " + key);
        long total = (long) (old == null ? 0 : old) + amount;
        if (total > Integer.MAX_VALUE)
            throw new IllegalArgumentException("累计库存超出整数上限: " + key);
        inventory.put(key, (int) total);
    }
    static String process(Map<String, Integer> inventory,
                        String item, String amountText) {
        if (item == null || item.isBlank()) return "物品名不能为空";
        if (amountText == null || amountText.isBlank()) return "数量不能为空";
        String key = item.strip();
        try {
            int amount = Integer.parseInt(amountText.strip());
            addItem(inventory, key, amount);
            return key + " 添加成功 数量=" + inventory.get(key);
        }
    }
}
```

12 安全背包程序（续）

SafeInventory.java • 第2/2段，顺序接在同一文件

```
} catch (NumberFormatException error) {
    return "数量必须是整数: " + amountText;
} catch (IllegalArgumentException error) {
    return error.getMessage();
}
}

public static void main(String[] args) {
    Map<String, Integer> inventory = new HashMap<>();
    System.out.println(process(inventory, "钻石", "3"));
    System.out.println(process(inventory, "钻石", "很多"));
    System.out.println(process(inventory, "钻石", "-2"));
    System.out.println(process(inventory, " ", "5"));
    System.out.println(process(inventory, "钻石", "64"));
    System.out.println("最终库存=" + inventory.get("钻石"));
}
}
```

运行核对

钻石 添加成功 数量=3
数量必须是整数: 很多
单次数量必须在1到64之间: -2
物品名不能为空
钻石 添加成功 数量=67
最终库存=67

1到64是每次操作数量上限，累计库存可超过64；这不是一个游戏物品格的堆叠规则。空数量提前提示，累计超出 int 也会拒绝；所有写入都在检查之后。

13 拓展任务与通关复盘

作业同时提交成功路径和至少三个失败路径。每个 catch 都写一句为什么能正确恢复。

三档任务继续强化错误处理

成功路径

至少一次真实添加

失败路径

解析、范围、空名

catch 理由

解释为何能够恢复

第29课让数据穿越程序重启。我们会用 Path、Files 和 UTF-8 保存并加载计分板。

三档作业

- 木镐：安全读取年龄，拒绝非整数和负数。
- 铁镐：解析x y z坐标，任何一项非法时保留旧位置。
- 钻石镐：记录每种错误出现次数，并在最后输出诊断报告。

验收对照表

自评	能	还要练
我会读异常线索	☐	☐
我会捕获具体异常	☐	☐
我能测试失败后不变	☐	☐

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>