

第39课

服务器急诊室

靠日志、边界与测试找到真正的 Bug



调试靠证据。读出堆栈中的自己的位置，复现赛外计分、晚结束和重复计时三个 Bug；用24项纯 Java 检查和十个游戏场景分别验证规则与接线。

继续第38课OreRush项目。测试使用Java21；游戏内检查沿用普通Paper1.21.11。故障片段只放在练习副本，不要覆盖可用版本。

阅读方式：先猜结果，再运行核对；每次只改一个条件。

环境：JDK 21、普通 Paper 1.21.11、同版本客户端。服务器插件由 Paper 加载；只有明确标注的规则测试使用 main。

开始前：先认清运行位置

前30课打好了普通 Java 程序的基础。这一阶段把代码接到 Paper 服务器：Minecraft 客户端负责进入世界，服务器加载插件，事件和命令把玩家动作交给你的方法。

本书固定使用 JDK 21 与 Paper 1.21.11，客户端版本也保持一致。下载页面上的最新版本可能已经变了，按版本号选择。本书采用普通 plugin.yml 与 Bukkit API；本阶段没有声称支持 Folia。

图框中的短例是局部片段，先读本节的位置说明，不要各自保存成插件。完整文件会明确给出文件名；同一文件分成多段时，按编号连接，包名、import 与外层类只出现一次。

- 预测：先写成功条件和预期消息。
- 运行：构建成功后，在专用本机测试服安装并重启。
- 核对：普通玩家、管理身份与控制台分别走对应路径。

场景插画帮助理解故事；图解解释真实规则，不作为 IDE 或游戏实测截图。做 Java 修改后重新构建、停服替换新 JAR，再启动；不要把旧 JAR 的表现当作新代码结果。

01 红色堆栈是一张定位图

服务器抛出一长串红字，阿方从最下面开始猜。侦探递来放大镜：先认异常类型和原因，再找到自己项目的第一处位置，沿调用路径追就不会在日志里迷路。

调试先重现，再缩小条件，最后修一处并重测。测试有两层：纯Java检查规则，真实服务器检查事件、命令与调度是否接对。编译通过不能代替游戏内验收。

红色堆栈是一张定位图

异常类型

说明哪类失败

自己的源文件

找到项目内第一处位置

复现步骤

同样输入才能重测

行号不是永久坐标：教材示意行号用于练习，不保证与你文件的当前行号相同。加了import或注释，后面的行号就会变化。

教学示意堆栈 并非本次实测日志 • 位置见本节说明

```
java.lang.NullPointerException: ...
    at de.teacherma.orerush.OreRushPlugin.onCommand(OreRushPlugin.java:58)
    at org.bukkit.command.PluginCommand.execute(PluginCommand.java:...)

```

写一张故障记录 • 位置见本节说明

步骤：输入/orerush score
 预期：未报名提示
 实际：红字异常
 环境：Java21 Paper1.21.11 插件版本与完整异常另附

02 找不到玩家就先停止

查询在线玩家可能得到null，离线不等于“找到了一个隐形Player”。在取得结果后立刻检查，再读取名字或发消息。并非所有null都应该强行换成默认对象，有时明确告诉操作者失败更合适。

找不到玩家就先停止

查询对象

在线查找可能为null

检查失败

说明离线并停止

成功对象

之后才调用方法

区分预期失败和程序缺陷：玩家离线是可预期情况；错误类型转换可能是代码缺陷。不要用一个catch(Exception)把所有问题盖住，再告诉玩家“操作成功”。

可加入管理命令的安全片段 • 位置见本节说明

```
Player target = getServer().getPlayerExact(name);
if (target == null) {
    sender.sendMessage("玩家不在线，请检查名字");
    return true;
}
target.sendMessage("老师邀请你参加测试");
```

保留异常而不是装作成功 • 位置见本节说明

```
catch (IOException ex) {
    getLogger().log(Level.WARNING, "读取失败", ex);
}
```

03 边界修复一 结束后仍然加分

坏版本只检查玩家是否报名。比赛结束后名单还在时，继续挖矿就会涨分。把“比赛中”加入加分方法的条件，即使调用者忘记检查，核心规则仍能挡住赛外加分。

停下来想一想：规则类的重复开始测试通过，就能证明服务器只创建一个定时任务吗？还要观察什么？

边界修复一 结束后仍然加分

WAITING

即使已报名也不加分

RUNNING

已报名正分才写入

ENDING

入口已关闭

想通这一步：不能。还要在游戏中重复发送开始命令，观察倒计时速度，确认停止后无残留消息、再次开始正常。

故障实验：下面故意保留错误，仅在练习副本中观察。不要把它合并进可用插件。

坏版本 忘记状态 • 故意错误，先预测再修复

```
if (scores.containsKey(id)) scores.merge(id, points, Integer::sum);
```

修复后的入口 • 位置见本节说明

```
if (state == State.RUNNING && scores.containsKey(id) && points > 0)
    scores.computeIfPresent(id, (key, old) -> old + points);
```

能揭露问题的检查 • 位置见本节说明

```
r.add(alex, 10); // 在WAITING状态
check(r.snapshot().get(alex) == 0, "waiting cannot score");
```

04 边界修复二 多活一秒的比赛

若先判断remaining等于0，之后才减数，2秒的比赛会等到第三次回调才结束。用一个短到能数清的例子：开始2，第一次变1，第二次变0并结算。不用真的等一分钟。



前置条件：tick开头还要检查state是否RUNNING。否则结束后继续减数，会把剩余时间一路挖到负数地下室。

故障实验：下面故意保留错误，仅在练习副本中观察。不要把它合并进可用插件。

坏版本 先判断后减数 • 故意错误，先预测再修复

```
if (remaining == 0) return true;
remaining--;
return false;
```

归零的那一次就结束 • 位置见本节说明

```
if (--remaining > 0) return false;
state = State.ENDING;
return true;
```

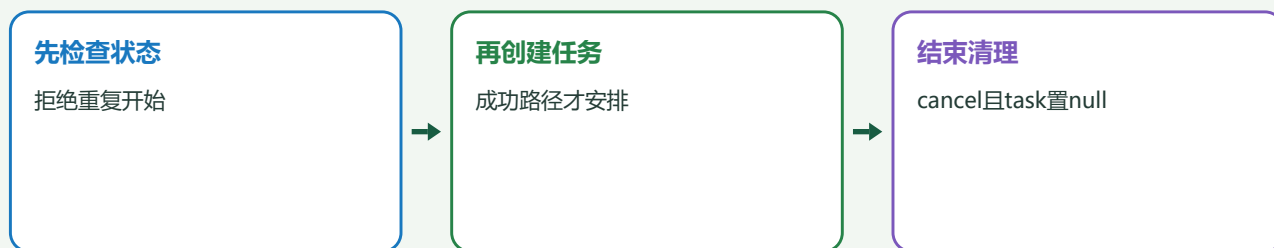
核对表

回调次数	修复后剩余	是否结束
0	2	否
1	1	否
2	0	是
3	0	否，不重复结束

05 边界修复三 两个裁判同时数秒

坏版本每收到一次start就创建任务。连按两次，秒数掉得像吃了急迫药水。先让GameRound.start验证状态，只有成功才进入调度器；停止时既cancel，也要清空任务字段。

边界修复三 两个裁判同时数秒



沿箭头进入下一步前，必须满足前一步条件；被拒绝的分支提前返回。

两层都要测：纯Java检查能证明重复start被拒绝，但不能证明Paper里真的只创建了一个任务。游戏内还要验证倒计时速度、stop后的安静和再次开始。

不把创建任务放在检查之前 • 位置见本节说明

```

if (!round.start(duration)) {
    sender.sendMessage("无法开始");
    return true;
}
// 下面才创建任务
  
```

清理要完整 • 完整文件

```

if (task != null) task.cancel();
task = null;
round.reset();
names.clear();
  
```

06 自动检查程序上半部

在OreRush项目新建checks/RoundChecks.java。本页接下一页是一份完整文件，不放进插件JAR。check不通过时抛AssertionError，运行立即失败；不会像默认关闭的Java assert那样悄悄跳过。

自动检查程序上半部

空局与报名

拒绝无人开始和重复报名

开始边界

时长与状态都校验

计分边界

赛外、外人和负分拒绝

时间由测试控制：2秒比赛只需调用两次tick，不用sleep。测试不访问世界、网络或Player，所以快而且容易重复。

RoundChecks第1部分 • 顺序接在同一源文件

```
import de.teacherma.orerush.GameRound;
import java.util.UUID;

public class RoundChecks {
    private static int checks;
    private static void check(boolean condition, String name) {
        checks++;
        if (!condition) throw new AssertionError(name);
    }
    public static void main(String[] args) {
        UUID alex = new UUID(0, 1), steve = new UUID(0, 2);
        GameRound r = new GameRound();
        check(!r.start(60), "empty cannot start");
        check(r.join(alex), "first join");
        check(!r.join(alex), "duplicate join");
        r.add(alex, 10);
        check(r.snapshot().get(alex) == 0, "waiting cannot score");
        check(!r.start(0), "zero duration rejected");
        check(!r.start(3601), "too long rejected");
        check(r.start(2), "start two seconds");
        check(!r.start(2), "duplicate start rejected");
        check(!r.join(steve), "late join rejected");
        r.add(alex, 10); r.add(steve, 10); r.add(alex, -5);
        check(r.snapshot().get(alex) == 10, "valid points only");
        check(!r.snapshot().containsKey(steve), "outsider cannot score");
    }
}
```

07 自动检查程序下半部

继续上一页。归零后再尝试加分，再重置，检查旧成绩快照是否仍然存在。最后覆盖深板岩矿石、普通石头和null输入。成功时输出PASS 24 checks。

自动检查程序下半部

时间与结束

第二次tick归零

重置与快照

旧快照仍保持成绩

材料规则

深板岩、普通石头及null

运行命令说明：两条命令分别执行；长命令在书中可能换行显示，输入时保持为一条。也可在IDE直接运行RoundChecks的main。测试失败不是通关，要定位并修复。

RoundChecks第2部分 • 顺序接在同一源文件

```

    check(!r.tick() && r.remaining() == 1, "first tick");
    check(r.tick(), "second tick ends");
    check(r.state() == GameRound.State.ENDING, "closed before result");
    r.add(alex, 10);
    check(r.snapshot().get(alex) == 10, "ending cannot score");
    check(!r.tick(), "cannot end twice");
    var result = r.snapshot();
    r.reset();
    check(r.state() == GameRound.State.WAITING, "reset state");
    check(r.snapshot().isEmpty(), "reset enrollment");
    check(result.get(alex) == 10, "snapshot survives reset");
    check(GameRound.points("DEEPSLATE_DIAMOND_ORE") == 10, "deep diamond");
    check(GameRound.points("GOLD_ORE") == 5, "gold");
    check(GameRound.points("DEEPSLATE_IRON_ORE") == 3, "deep iron");
    check(GameRound.points("STONE") == 0, "ordinary block");
    check(GameRound.points(null) == 0, "missing material");
    System.out.println("PASS " + checks + " checks");
}
}

```

先新建 checks/sources.txt，写入下面两条相对于项目根目录的源文件路径，每行一条。然后从项目根目录运行短命令；也可以在 IDE 直接运行 RoundChecks.main。

checks/sources.txt • 完整内容

```

src/main/java/de/teacherma/orerush/GameRound.java
checks/RoundChecks.java

```

07 自动检查程序下半部（续）

Windows项目根目录运行 • 位置见本节说明

```
javac -encoding UTF-8 -d checks/out @checks/sources.txt  
java -cp checks/out RoundChecks
```

运行核对

PASS 24 checks

08 十个场景把游戏也测一遍

自动检查的是数字规则，游戏里还多了命令权限、模式、世界和其他插件。按表操作，记录“实际结果”和日志位置；发现问题时只改一个条件再试，证据才不会像末影人一样瞬移。

十个场景把游戏也测一遍

规则检查

数字和状态能独立验证

游戏验收

权限事件调度要对接

两份证据

不能互相替代

注意开关：第33课的保护插件可能先挡住破坏。测试正常计时时要在允许破坏的位置；测试取消事件时再启用相应保护。

核对表

编号与动作	预期
1 无人报名start	拒绝
2 同人重复join	第二次拒绝
3 两人报名start	只启动一局
4 立刻再start	拒绝，无第二任务
5 未报名者挖矿	不计分
6 创造模式或异世界挖矿	不计分
7 保护取消挖矿	不计分
8 自然结束后再挖	不计分
9 计时中stop	无后续倒计时广播
10 再报名开始后停服重启	旧任务和本局数据不恢复

09 优化前先找真正的耗时

玩家说“卡”，可能是网络、世界生成，也可能是插件。先记录时间、人数、操作和复现步骤，再测量。不要看到一个for循环就宣布它有罪：循环十个分数，与每块矿石都写文件不是同一回事。

优化前先找真正的耗时

先重现

记录操作、人数与时刻



再测量

识别真实耗时路径



后改进

复测同一个场景

看哪两个指标：TPS是每秒完成多少tick；MSPT是平均或某种统计口径下每tick耗时。目标20TPS对应约50毫秒预算，但一次采样或平均数不能代表所有卡顿。

只测一段纯计算 • 位置见本节说明

```
long started = System.nanoTime();
var result = round.snapshot();
long elapsedNs = System.nanoTime() - started;
// 单次结果受JVM预热等影响，不是正式性能结论。
```

有用但不过量的日志 • 位置见本节说明

```
getLogger().info("OreRush started, players=" + round.snapshot().size());
```

- 不要在每次挖矿里下载网页、遍历所有区块或保存整个文件。
- 在测试服可以按Paper官方Profiling指南使用性能分析器，先看耗时调用再优化。
- 分享日志或分析报告前检查玩家名、地址和路径；未成年学生不要自行上传完整服务器资料。

10 拓展挑战与自己的验收记录

留下源代码、实际运行结果和一条问题记录。先用自己的话解释，再运行验证；结果与预测不同的地方，往往最值得讨论。

拓展任务与通关复盘

木镐

解释自己的堆栈

铁镐

修复三个边界Bug

钻石镐

制作只读自检命令

下一站：第40课把通过验收的作品整理成别人也能安装使用的毕业包。
官方延伸阅读：docs.papermc.io/paper/dev/reading-stacktraces/

- 木镐：解释一段自己项目的堆栈，指出异常、方法和源文件位置。
- 铁镐：在练习副本复现并修复本课三个Bug，提交修复前后测试结果。
- 钻石镐：制作只读自检命令，显示版本、当前状态和报名人数；不要输出私人信息。
- 验收要求：保留修复前复现步骤、定位依据和修复后结果；换一个同桌按记录操作，仍应得到一致结论。

核对表

我能独立完成	已掌握	再练一次
能用证据复现错误	☐	☐
会运行24项规则检查	☐	☐
能区分单元规则与游戏验收	☐	☐

把证据留给下一次冒险

把“能编译”“数字规则通过”和“游戏实际表现正确”分成三栏。没有运行的场景写待测；发现 Bug 时记录触发条件、预期、实际、最小修复与复测结果。

你的验证记录

类别	你选择的操作	留下的证据
正常	本课的一次成功路径	版本与实际输出
边界	结束前一刻、最大值或满背包	与合同逐项比较
拒绝	错误参数、无权限或不相关事件	反馈清楚，状态未误改

- 解释本课每条箭头为什么从这里指向那里；并列卡片没有先后含义。
- 先改一个规则，再同时修改说明和预期，用同一输入复测。

技术核对：本课以 Paper 1.21.11 API 和 Java SE 21 为准。参考文档可能切换到较新版本，查阅时注意版本号。

<https://jd.papermc.io/paper/1.21.11/>
<https://docs.papermc.io/paper/dev/>
<https://docs.oracle.com/en/java/javase/21/docs/api/>