

第38课

小项目 矿石冲刺赛

把命令、事件与计时组合成一场比赛



把前七课的零件连成一局矿石冲刺赛。规则类管状态与分数，插件接命令、事件和计时；比赛结束先关计分入口，再读取成绩和清理任务。

前置背包：第31-37课。独立项目名OreRush，包名de.teacherma.orerush，继续使用Java21和普通Paper1.21.11。本课适合分两次课完成。

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 先约好比赛规则

比赛还没开始，阿方已经挖出一袋煤，催裁判加分。裁判翻出状态牌：集合时只能报名，比赛时才记分，结算后再挥镐也不能修改这一局成绩。

先把规则写成状态变化：WAITING 等报名，RUNNING 接受有效计分，ENDING 冻结并结算。纯Java类管理规则，Paper主类把命令、事件和任务接进来，两边各有可检查的责任。

先约好比赛规则

玩家报名查分

join与score

管理员启停

start与stop

计分合同

指定世界生存模式且事件未取消

练习范围：这是老师监督的MVP最小可玩版本，不含自动矿区重置、跨服、防刷分和实物奖励。只能用于可丢弃的练习测试世界；不要拿重要生存存档试搞。

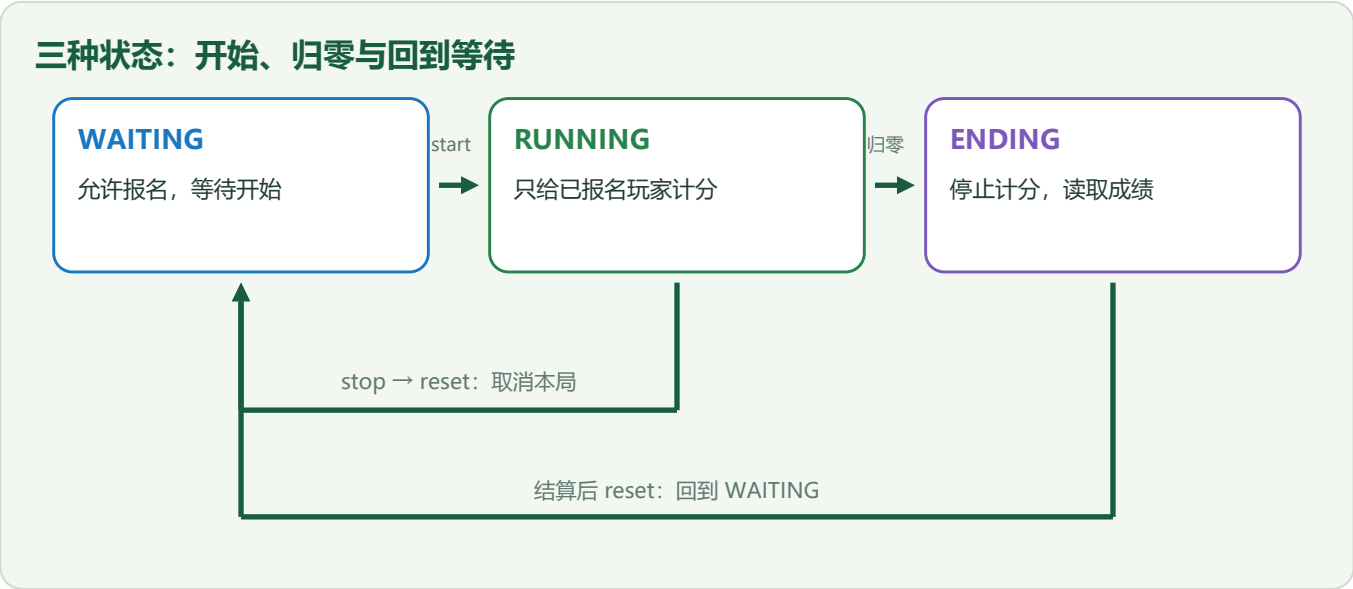
玩家与老师的四条命令 • 位置见本节说明

```
/orrush join      # 玩家报名
/orrush score     # 查询本局分数
/orrush start     # 老师开始
/orrush stop      # 老师取消并清空报名
```

- 三类矿石都包括普通矿石与深板岩矿石。按获准破坏的方块计分，不按掉落物数量。
- 只计算指定世界、生存模式、已报名玩家的事件。
- 一局结束后清空报名和分数；下一局必须重新报名。

02 状态决定现在能做什么

WAITING像集合区，允许报名；RUNNING像比赛中的矿洞，允许计分；ENDING先锁住计分，再整理成绩。
状态不是写在墙上的标语，而是每次动作前都会被程序检查的值。



补上返回箭头：结算完成后reset回到WAITING。画图时给ENDING画一条回集合区的箭头，下一场比赛才有入口。

只有比赛中才能加分 • 位置见本节说明

```
if (state != State.RUNNING) return;
```

核对表

动作	WAITING	RUNNING	ENDING
报名	允许	拒绝	拒绝
开始	有人才允许	拒绝	拒绝
加分	拒绝	已报名才允许	拒绝
reset	清空	取消并清空	结算后清空

03 报名名单和分数共享玩家键

UUID做键，0分表示已报名但还没有得分。没有这个键表示未报名，两者不能混淆。names另存报名时的名字，只用于结果展示；程序不会长期抓住一个Player对象不放。

报名名单和分数共享玩家键

同一个UUID

名字与分数共用身份键

初始0分

报名成功才创建

名字快照

只用于成绩展示

离线的处理：离线不会删除本局成绩；回来后仍用同一UUID继续。默认练习人数较少，至少一人报名即可开始，不强制所有报名者在线；规则要告诉玩家。

避免重复报名清零 · 位置见本节说明

```
if (state != State.WAITING || scores.containsKey(id)) return false;
scores.put(id, 0);
return true;
```

区分未报名与零分 · 位置见本节说明

```
Integer score = round.snapshot().get(id);
String message = score == null ? "你尚未报名" : "本局得分：" + score;
```

04 先过滤事件 再计算矿石分值

一块石头也会产生破坏事件，但不必给它发钻石奖牌。监听器先筛选比赛状态、世界和生存模式；规则类再检查报名。材料名称交给纯Java方法，得到0、3、5或10。

停下来想一想：结束时应先广播排行榜，还是先让状态离开 RUNNING？如果结算期间又收到挖矿事件呢？

先过滤事件 再计算矿石分值



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。

想通这一步：先关闭计分，再生成并广播结果。这样结算期间的新事件不会改动本局分数，也能防止重复结算。

统一分值入口 • 位置见本节说明

```
int points = GameRound.points(event.getBlock().getType().name());
```

两类深板岩矿石也要认 • 位置见本节说明

```
GameRound.points("DEEPSLATE_DIAMOND_ORE"); // 10
GameRound.points("DEEPSLATE_IRON_ORE");    // 3
GameRound.points("STONE");                  // 0
```

05 一场比赛只领一张任务票

开始成功后才创建重复任务。GameRound.start拒绝重复开始，因此连按两次start也不会冒出两个裁判，同时把秒数减两遍。第一次回调等20tick，再减1。

一场比赛只领一张任务票

开始成功

状态检查通过才安排

每20tick

回调一次更新

任务唯一

拒绝再次start

游戏节拍：理想20TPS下60次间隔约为60秒。服务器卡顿时现实时间会更长。这个版本采用tick倒计时，不承诺精确墙上时钟计时。

先检查再创建任务 · 位置见本节说明

```
if (!round.start(duration)) {  
    sender.sendMessage("现在不能开始");  
    return true;  
}  
// 通过后才调用runTaskTimer
```

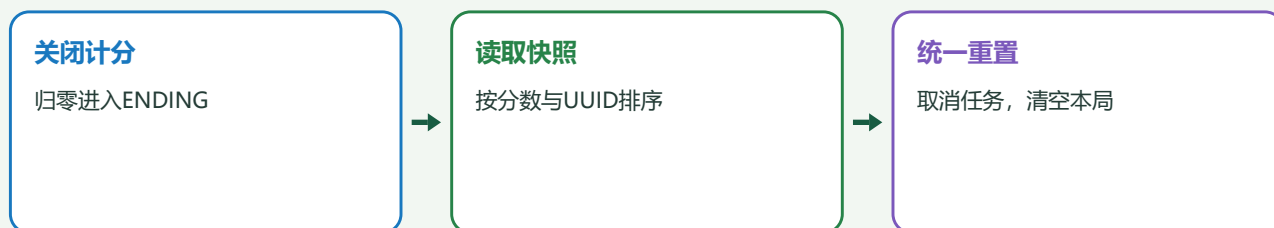
回调中的顺序 · 位置见本节说明

```
if (round.tick()) { finish(); return; }  
int left = round.remaining();
```

06 先关计分 再宣布成绩

最后一次tick先把状态变为ENDING，才调用finish。结算时取消任务，读取成绩快照，按分数从高到低显示，最后清理。否则裁判念到第二个人时，第一个人还可能偷偷再挖一块。

先关计分 再宣布成绩



不要顺手发奖励：这个版本没有领奖代码。要发实物奖励，还需处理重复结算、断线和满背包，不能只在广播后塞一行发钻石。

状态在规则类里改变 • 位置见本节说明

```

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

成绩快照不怕原表清空 • 位置见本节说明

```

var result = round.snapshot();
round.reset();
// result仍保留刚才的成绩，不能修改。
  
```

- 同分者分数相同，展示顺序用UUID字符串稳定排列，不宣称先显示的人赢了。
- 本局结果只广播，不保存历史；stop属于取消，不颁奖。
- 插件停用也取消任务并清空内存状态。

07 准备两个类和两个资源文件

创建Gradle项目OreRush，使用第31课的build.gradle.kts。Java文件放src/main/java/de/teacherma/orerush。以下两份yaml都放src/main/resources。第一次启动后修改plugins/OreRush/config.yml，开服再启动生效。

准备两个类和两个资源文件

GameRound.java

数字规则与状态

OreRushPlugin.java

命令事件与调度

两个yaml

身份权限与玩法配置

老师课前准备：在world世界、远离出生保护的地方准备少量三类矿石，给学生合适的镐并切到生存模式。本课不自动生成矿场，也不限制世界内的具体矿区。

完整plugin.yml • 完整文件

```
name: OreRush
version: 1.0.0
main: de.teacherma.orerush.OreRushPlugin
api-version: '1.21.11'
author: TeacherMa
website: https://www.teacher-ma.de
commands:
  orerush:
    description: 报名和管理矿石冲刺
    usage: /orerush <join|score|start|stop>
permissions:
  blockacademy.orerush.admin:
    default: op
```

完整config.yml • 完整文件

```
world: "world"
duration-seconds: 60
```

08 完整比赛规则类

把下面完整保存为GameRound.java。这个类不导入Paper，只处理状态、UUID和数字；第39课会直接测试它，不必每次都启动游戏。

完整比赛规则类

报名合同

只在WAITING且不重复

开始合同

有人、合法时长、空闲

计分合同

比赛中已报名且正分

完整GameRound.java • 本段1/2，顺序连接

```
package de.teacherma.orerush;
import java.util.HashMap;
import java.util.Map;
import java.util.UUID;
public final class GameRound {
    public enum State { WAITING, RUNNING, ENDING }
    private State state = State.WAITING;
    private int remaining;
    private final Map<UUID, Integer> scores = new HashMap<>();
    public State state() { return state; }
    public int remaining() { return remaining; }
    public boolean join(UUID id) {
        if (state != State.WAITING || scores.containsKey(id)) return false;
        scores.put(id, 0); return true;
    }
    public boolean start(int seconds) {
        if (state != State.WAITING || scores.isEmpty() || seconds < 1 || seconds > 3600)
            return false;
        scores.replaceAll((id, old) -> 0);
        remaining = seconds; state = State.RUNNING; return true;
    }
    public void add(UUID id, int points) {
        if (state == State.RUNNING && scores.containsKey(id) && points > 0)
            scores.computeIfPresent(id, (key, old) -> old + points);
    }
}
```

08 完整比赛规则类（续）

完整GameRound.java • 本段2/2, 顺序连接

```
public boolean tick() {
    if (state != State.RUNNING) return false;
    if (--remaining > 0) return false;
    state = State.ENDING; return true;
}
public Map<UUID, Integer> snapshot() { return Map.copyOf(scores); }
public void reset() {
    state = State.WAITING; remaining = 0; scores.clear();
}
public static int points(String material) {
    if (material == null) return 0;
    return switch (material) {
        case "DIAMOND_ORE", "DEEPSLATE_DIAMOND_ORE" -> 10;
        case "GOLD_ORE", "DEEPSLATE_GOLD_ORE" -> 5;
        case "IRON_ORE", "DEEPSLATE_IRON_ORE" -> 3;
        default -> 0;
    };
}
}
```

09 比赛主类第一部分

接下来四页按顺序合成OreRushPlugin.java。本页导入依赖、声明共享状态并读取配置；如果YAML本身读不通，就停用插件并保留日志。

比赛主类第一部分

插件字段

一份规则与一个任务

候选配置

类型、时长与世界验证

登记入口

配置合格才接命令事件

主类第1部分 · 本段1/2，顺序连接

```
package de.teacherma.orerush;

import java.io.File;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import java.util.Objects;
import java.util.UUID;
import net.kyori.adventure.text.Component;
import org.bukkit.GameMode;
import org.bukkit.command.Command;
import org.bukkit.command.CommandSender;
import org.bukkit.configuration.InvalidConfigurationException;
import org.bukkit.configuration.file.YamlConfiguration;
import org.bukkit.entity.Player;
import org.bukkit.event.EventHandler;
import org.bukkit.event.EventPriority;
import org.bukkit.event.Listener;
import org.bukkit.event.block.BlockBreakEvent;
import org.bukkit.plugin.java.JavaPlugin;
import org.bukkit.scheduler.BukkitTask;

public final class OreRushPlugin extends JavaPlugin implements Listener {
    private final GameRound round = new GameRound();
    private final Map<UUID, String> names = new HashMap<>();
```

09 比赛主类第一部分（续）

主类第1部分 • 本段2/2, 顺序连接

```
private BukkitTask task;
private String worldName;
private int duration;
@Override
public void onEnable() {
    saveDefaultConfig();
    var config = new YamlConfiguration();
    try { config.load(new File(getDataFolder(), "config.yml")); }
    catch (IOException | InvalidConfigurationException ex) {
        getLogger().log(java.util.logging.Level.SEVERE, "配置读取失败", ex);
        getServer().getPluginManager().disablePlugin(this); return;
    }
}
```

10 比赛主类第二部分

继续同一个文件。检查配置的类型和范围，注册命令及事件。join和score必须由玩家执行；报名还要确认玩家所在世界和生存模式。

比赛主类第二部分

join

玩家在指定世界报名

score

查询自己的分数

start与stop

管理权限保护

失败不偷偷修正：世界不存在或时长超范围时让老师修正配置。本课不把错误世界自动换成别的世界，也不悄悄开始比赛。

主类第2部分 · 顺序接在同一源文件

```
worldName = config.getString("world", "");
duration = config.getInt("duration-seconds", 0);
if (!config.isString("world") || !config.isInt("duration-seconds")
    || getServer().getWorld(worldName) == null || duration < 1 || duration > 3600) {
    getLogger().severe("配置无效: world需已加载, duration-seconds需为1到3600的整数");
    getServer().getPluginManager().disablePlugin(this); return;
}
Objects.requireNonNull(getCommand("orerush")).setExecutor(this);
getServer().getPluginManager().registerEvents(this, this);
}
@Override
public boolean onCommand(CommandSender sender, Command command,
    String label, String[] args) {
    if (args.length != 1) { sender.sendMessage("/orerush join|score|start|stop"); return true; }
    String action = args[0];
    if (action.equals("join") || action.equals("score")) {
        if (!(sender instanceof Player p)) {
            sender.sendMessage("报名和查分请在游戏中操作"); return true;
        }
        UUID id = p.getUniqueId();
        if (action.equals("score")) {
            Integer score = round.snapshot().get(id);
            p.sendMessage(score == null ? "你尚未报名" : "本局得分: " + score);
        } else if (!p.getWorld().getName().equals(worldName)
            || p.getGameMode() != GameMode.SURVIVAL) {
            p.sendMessage("请在比赛世界使用生存模式报名");
        } else if (round.join(id)) {
            names.put(id, p.getName()); p.sendMessage("报名成功, 等待老师开始");
        } else p.sendMessage("已经报名, 或者比赛正在进行");
        return true;
    }
}
```

11 比赛主类第三部分

本段处理老师的start和stop。start被拒绝时不创建任何新任务；stop取消本局并清空报名。定时回调只做一次减数和少量消息判断。

比赛主类第三部分

开始

成功后创建一个任务

倒计时

首次与间隔都20tick

事件

只把合格材料送入规则

为什么用同步任务：计时回调与普通方块事件在本课的Paper主线程上顺序执行。不要把round的修改随意搬到异步线程，否则HashMap和状态就可能出现竞争。

主类第3部分 · 顺序接在同一源文件

```

    if (!action.equals("start") && !action.equals("stop")) {
        sender.sendMessage("参数只能是 join、score、start、stop"); return true;
    }
    if (!sender.hasPermission("blockacademy.orerush.admin")) {
        sender.sendMessage("只有老师可以开始或停止比赛"); return true;
    }
    if (action.equals("stop")) {
        reset(); announce("本局已取消，报名已清空"); return true;
    }
    if (getServer().getWorld(worldName) == null || !round.start(duration)) {
        sender.sendMessage("无法开始：检查世界、报名人数和比赛状态"); return true;
    }
    announce("矿石冲刺开始，共 " + duration + " 秒!");
    task = getServer().getScheduler().runTaskTimer(this, () -> {
        if (round.tick()) { finish(); return; }
        int left = round.remaining();
        if (left % 10 == 0 || left <= 5) announce("剩余 " + left + " 秒");
    }, 20L, 20L);
    return true;
}

```

12 比赛主类第四部分

补齐方块事件、成绩排序和清理方法，最后关闭主类。finish只由归零的tick调用，所有收尾路径都能取消任务；Map快照用于读分，原表仍由规则类管理。

比赛主类第四部分

结算

先读独立成绩快照

广播

同分也有确定顺序

清理

任务、报名、名字统一收尾

构建并安装：运行.\gradlew.bat clean build，得到build/libs/OreRush-1.0.0.jar。停服后安装到plugins并重启；不要同时放入本插件的两个版本。

主类第4部分 • 顺序接在同一源文件

```
@EventHandler(priority = EventPriority.MONITOR, ignoreCancelled = true)
public void onBreak(BlockBreakEvent event) {
    if (round.state() != GameRound.State.RUNNING) return;
    if (!event.getBlock().getWorld().getName().equals(worldName)) return;
    if (event.getPlayer().getGameMode() != GameMode.SURVIVAL) return;
    round.add(event.getPlayer().getUniqueId(),
        GameRound.points(event.getBlock().getType().name()));
}
private void finish() {
    if (task != null) task.cancel();
    task = null;
    announce("时间到，结算本局得分！");
    round.snapshot().entrySet().stream().sorted((a, b) -> {
        int order = Integer.compare(b.getValue(), a.getValue());
        return order != 0 ? order : a.getKey().toString().compareTo(b.getKey().toString());
    }).forEach(entry -> announce(names.getOrDefault(entry.getKey(), "未知玩家")
        + ": " + entry.getValue() + " 分"));
    reset();
}
private void announce(String text) { getServer().broadcast(Component.text(text)); }
private void reset() {
    if (task != null) task.cancel();
    task = null; round.reset(); names.clear();
}
@Override
public void onDisable() { reset(); }
```

13 从报名走完整整一局

邀请两位同学分别报名，让老师开始。Alex挖一块钻石，Steve挖两块铁矿，结束时应该分别是10分和6分。
再做故障路线：未报名、重复开始、结束后继续挖，看看分数是否被挡住。

Alex钻石

10分

Steve铁矿两块

6分

结束后再挖

不再写入本局

公平性边界：本课没有区分天然矿与玩家摆放的矿石，也没有完整反作弊。只在老师准备并监督的测试场玩；公开服务器还需矿区隔离、方块来源和奖励审计。

核对表

操作	预期
无人报名时start	拒绝
比赛中再start或join	拒绝，不叠加任务
生存模式挖两类钻石矿	每块10分
未报名者或其他世界挖矿	不计分
破坏被保护插件取消	不计分
自然结束后再挖	不计分，需重新报名
比赛中stop或停服	任务清理，成绩不保留

14 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

完整完成一次对局

铁镐

修改分值并重测

钻石镐

增加只读状态查询

下一站：第39课用日志与自动检查把边界Bug从矿洞里请出来。
官方延伸阅读：docs.papermc.io/paper/dev/scheduler/

- 木镐：为三种矿石及它们的深板岩版本列测试表，确认普通石头为0分。
- 铁镐：完成双人一局、重复start、stop和重启测试，记录十条验收结果。
- 钻石镐：设计队伍或连击，只选一个新增功能；写清何时加分、何时清理，再实现。
- 验收要求：完成双人一整局，记录报名、重复开始、未报名挖矿、取消事件、结束后挖矿和第二局重置。

核对表

我能独立完成	已掌握	再练一次
能解释三个比赛状态	☐	☐
会连接命令事件与调度	☐	☐
能走通正常与失败路径	☐	☐

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/>