

第34课

每个玩家的数据

用 UUID 为每位玩家保管独立冷却



大家共用一张 Map，每位玩家拥有自己的 UUID 标签。把冷却写成截止时间，用可控制的测试时钟验证边界，再接到真正的冲刺命令。

前置背包：HashMap、命令、权限和简单算术。新项目BlockAcademyL34沿用第31课构建脚本，使用Java21与Paper1.21.11。

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 玩家名字会变 身份键要稳

两位玩家轮流冲刺，第二位刚出发就被告知“你还在冷却”。原来程序把全村人塞进同一只秒表。每位玩家用自己的 UUID 找自己的记录，冷却才不会串门。

Map 的键是玩家身份，值是下一次允许行动的时刻。比较当前时间和这个时刻即可，不必每秒给每人都建任务。测试时把时间作为参数传入，就能不用真的等三十秒。

玩家名字会变 身份键要稳

UUID

服务器提供的身份键

显示名

给读者和玩家看

数据记录

按键找到自己的状态

短名字与长编号：名字方便人阅读，UUID方便程序关联数据。本课直接使用服务器提供的UUID，不用玩家名字自己拼一个。

读取身份键 • 位置见本节说明

```
UUID id = player.getUniqueId();  
String displayName = player.getName();
```

合适的字段类型 • 位置见本节说明

```
private final Map<UUID, Integer> mined = new HashMap<>();
```

02 每位玩家有自己的格子

若只写一个全局次数变量，Alex挖一次，Steve查看时也会增加。Map用UUID分隔记录：大家共用一只箱子，每人拥有自己的标签。

每位玩家有自己的格子

Alex的记录

只更新Alex

Steve的记录

不跟着增加

缺失记录

getOrDefault提供初值

在线玩家不是数据文件：长期Map存UUID和普通数据，避免长期保存Player对象。需要操作在线玩家时，再通过UUID寻找并检查是否在线。

累计开采次数 • 位置见本节说明

```
UUID id = player.getUniqueId();
mined.merge(id, 1, Integer::sum);
int count = mined.getOrDefault(id, 0);
```

新玩家默认零 • 位置见本节说明

```
UUID other = UUID.randomUUID(); // 只用于本地练习
System.out.println(mined.getOrDefault(other, 0));
```

运行核对

0

03 记录可以再次行动的时刻

假设当前时间为1000毫秒，冷却持续30000毫秒，那么可再次使用的时间是31000。每位玩家都存自己的 readyAt，比较当前时间就知道是否还在冷却。

停下来想一想：离冷却结束还差1毫秒，应该允许吗？记录在玩家退出时直接删掉，会带来什么绕过方式？

记录可以再次行动的时刻

现在1000ms

开始30秒冷却

结束前30999ms

还剩1ms，拒绝

截止31000ms

剩余0，允许

想通这一步：还没到允许时刻就应拒绝。退出即删除会让重连跳过冷却，所以清理策略要保留尚未结束的记录。

时间模型：本例把系统时刻作为短时冷却依据；调整系统时钟可能改变显示的剩余时间。只计经过多久的进阶功能可研究 System.nanoTime，但它的起点与单位不同，不能直接和毫秒存档混算。

计算截止时间 · 位置见本节说明

```
long now = System.currentTimeMillis();
long readyAt = now + 30_000L;
boolean cooling = now < readyAt;
```

统一时间单位 · 位置见本节说明

```
long durationMs = 30_000L; // 30秒，不是30毫秒
long threeSeconds = 3L * 1000L;
```

04 剩一毫秒也还没结束

玩家不喜欢看到“再等0秒”，却仍然不能行动。判断冷却使用毫秒，显示给玩家时向上取整。只要还有1毫秒，界面就显示1秒。

剩一毫秒也还没结束

逻辑判断

使用剩余毫秒

玩家显示

向上取整为秒

边界证据

1ms也显示1秒

亲手预测：先遮住右列，算完再对答案。直接除以1000会截掉小数，正是“0秒却不能用”的来源。

剩余时间不小于零 · 位置见本节说明

```
long remainingMs = Math.max(0L, readyAt - now);
```

整数向上取整 · 位置见本节说明

```
long seconds = (remainingMs + 999L) / 1000L;
```

核对表

剩余毫秒	显示秒数
0	0
1	1
999	1
1000	1
1001	2

05 冷却规则独立成一个类

新建 src/main/java/de/teacherma/lesson34/Cooldowns.java。方法接收now，不在内部偷偷读取时钟，这样测试时可以直接把时间推到边界。

冷却规则独立成一个类

remaining

计算剩余时间

start

记录截止时刻

prune与clear

按策略清理记录

参数约定：id使用真实玩家UUID，durationMs是短时间的非负毫秒数，now在一次操作中保持相同。这个类用于主线程上的练习短时冷却，不做跨重启持久化。

完整 Cooldowns 类 • 完整文件

```
package de.teacherma.lesson34;

import java.util.HashMap;
import java.util.Map;
import java.util.UUID;

public final class Cooldowns {
    private final Map<UUID, Long> readyAt = new HashMap<>();
    public long remaining(UUID id, long now) {
        return Math.max(0L, readyAt.getOrDefault(id, now) - now);
    }
    public void start(UUID id, long now, long durationMs) {
        if (durationMs < 0) throw new IllegalArgumentException("冷却不能为负");
        readyAt.put(id, now + durationMs);
    }
    public long seconds(UUID id, long now) {
        long ms = remaining(id, now);
        return (ms + 999L) / 1000L;
    }
    public void prune(long now) {
        readyAt.entrySet().removeIf(entry -> entry.getValue() <= now);
    }
    public void clear() { readyAt.clear(); }
}
```

06 让假时钟替你等三十秒

不必真的站在屏幕前数秒。新建普通测试类，在main里调用下面代码。所有数字都是测试时钟；这样边界测试运行很快，也容易重现。

让假时钟替你等三十秒

控制测试时间

不用真的等30秒

两个UUID

证明状态独立

两个边界

结束前1ms与恰好结束

关键证据：最后一行是Steve。Alex还在冷却，Steve却可以行动，证明状态按玩家分开了。再测试负冷却参数应抛出异常。

两个玩家的边界测试 • 位置见本节说明

```
Cooldowns c = new Cooldowns();
UUID alex = UUID.randomUUID();
UUID steve = UUID.randomUUID();
c.start(alex, 1000L, 30_000L);
System.out.println(c.remaining(alex, 1000L));
System.out.println(c.remaining(alex, 30999L));
System.out.println(c.seconds(alex, 30999L));
System.out.println(c.remaining(alex, 31000L));
System.out.println(c.remaining(steve, 2000L));
```

运行核对

```
30000
1
1
0
0
```

07 冲刺方向是一支箭头

getDirection 返回玩家视线方向。把Y设为0，得到水平箭头，再归一化并乘以0.9，最后给一个0.2的轻微向上速度。玩家几乎竖直看天时，水平箭头可能接近零，要先拒绝。

冲刺方向是一支箭头



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。

速度不是距离：这不是传送固定0.9格。地形、碰撞、阻力和服务器规则会影响实际移动。到平坦测试区尝试，旁边留出空间。

防止归一化零向量 · 位置见本节说明

```

Vector direction = player.getLocation().getDirection().setY(0);
if (direction.lengthSquared() < 0.0001) {
    player.sendMessage(Component.text("请平视前方再冲刺"));
    return true;
}
  
```

给玩家一次速度 · 位置见本节说明

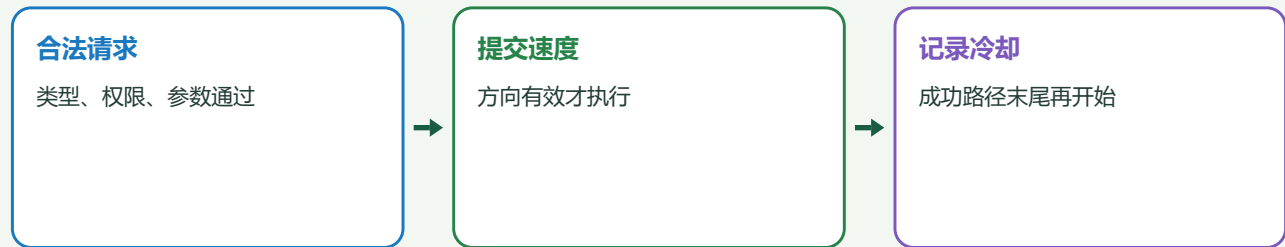
```

player.setVelocity(direction.normalize().multiply(0.9).setY(0.2));
  
```

08 只有成功执行后才开始冷却

把扣冷却放得太早，玩家抬头输错一次也要等30秒，像自动售货机先吞钱再检查有没有饮料。按顺序验证，真正调用setVelocity后才记录冷却。

只有成功执行后才开始冷却



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。

成功的含义：这里表示通过本插件检查并提交速度，不保证最终移动距离。其他插件可能拦截或修改速度；复杂技能需要进一步定义成功规则。

使用同一份现在时间 • 位置见本节说明

```

long now = System.currentTimeMillis();
if (cooldowns.remaining(id, now) > 0) {
    player.sendMessage(Component.text(
        "再等 " + cooldowns.seconds(id, now) + " 秒"));
    return true;
}
  
```

成功路径末尾 • 位置见本节说明

```

player.setVelocity(direction.normalize().multiply(0.9).setY(0.2));
cooldowns.start(id, now, 30_000L);
  
```

09 退出游戏不等于冷却消失

如果玩家一退出就删除他的冷却，重新登录立刻又能冲刺。本课保留尚未过期记录，下次有人执行命令时清理过期项；停用插件时统一清空。

退出游戏不等于冷却消失

退出重进

未结束记录保留

记录过期

有人执行命令时清理

服务器重启

本章内存冷却清空

小班规模：prune会扫描当前记录，适合练习人数。大型服务器可以定时分批清理；学习第35拓展再设计。内存Map不是永久存档。

懒清理 • 位置见本节说明

```
long now = System.currentTimeMillis();
cooldowns.prune(now);
```

停用时清空 • 位置见本节说明

```
@Override
public void onDisable() {
    cooldowns.clear();
}
```

核对表

动作	本课行为
玩家退出再进	未到期冷却继续
插件停用或服务器重启	内存数据清空
记录过期后有人执行命令	删除过期项

10 冲刺项目需要两份 Java 文件

src/main/java/de/teacherma/lesson34 下放 Cooldowns.java 和 DashPlugin.java。它们在同一个包里，所以主类能直接使用 Cooldowns。资源文件只声明主类，不用登记普通辅助类。

冲刺项目需要两份 Java 文件

Cooldowns.java

只管数字与时间

DashPlugin.java

连接命令和玩家

plugin.yml

只登记插件主类

准备构建：settings 中的项目名设为BlockAcademyL34；build.gradle.kts沿用第31课。不要把两个public类粘进同一个Java文件。

完整 plugin.yml • 完整文件

```
name: BlockAcademyL34
version: 1.0.0
main: de.teacherma.lesson34.DashPlugin
api-version: '1.21.11'
author: TeacherMa
website: https://www.teacher-ma.de
commands:
  dash:
    description: 向前冲刺
    usage: /dash
permissions:
  blockacademy.dash:
    default: true
```

11 冲刺主类上半部

新建 DashPlugin.java，先复制本页，再接下一页。主类持有一个Cooldowns对象，这个对象内部按UUID给每名玩家保管记录。

冲刺主类上半部

一个插件字段

持有一份Cooldowns

Map多个键

每名玩家一条记录

一个请求

查询与更新用同一个now

字段不是每人一个：cooldowns字段属于插件；Map中的键值记录属于各玩家。既不用为每人new一个插件，也不用使用全局统一倒计时。

DashPlugin 第1部分 • 顺序接在同一源文件

```
package de.teacherma.lesson34;

import java.util.Objects;
import net.kyori.adventure.text.Component;
import org.bukkit.command.Command;
import org.bukkit.command.CommandSender;
import org.bukkit.entity.Player;
import org.bukkit.plugin.java.JavaPlugin;
import org.bukkit.util.Vector;

public final class DashPlugin extends JavaPlugin {
    private final Cooldowns cooldowns = new Cooldowns();
    @Override
    public void onEnable() {
        Objects.requireNonNull(getCommand("dash")).setExecutor(this);
    }
    @Override
    public boolean onCommand(CommandSender sender, Command command,
        String label, String[] args) {
        if (!(sender instanceof Player player)) {
            sender.sendMessage("冲刺需要游戏中的身体");
            return true;
        }
        if (!player.hasPermission("blockacademy.dash")) {
            player.sendMessage(Component.text("你还没有冲刺权限"));
            return true;
        }
        if (args.length != 0) {
            player.sendMessage(Component.text("用法: /dash"));
            return true;
        }
    }
}
```

12 冲刺主类下半部

继续上一页。查询、提示、方向验证、执行和记录都发生在一次命令调用中。未通过检查的分支提前返回，方便从上到下阅读。

冲刺主类下半部

查询

剩余时间决定分支

方向

零向量不执行

动作完成

记录冷却并反馈

开始游戏验收：构建并重启测试服。执行 /dash，再立即执行一次，第二次应只显示剩余秒数；30秒后才再次冲刺。

DashPlugin 第2部分 • 顺序接在同一源文件

```

    long now = System.currentTimeMillis();
    cooldowns.prune(now);
    var id = player.getUniqueId();
    if (cooldowns.remaining(id, now) > 0) {
        player.sendMessage(Component.text(
            "再等 " + cooldowns.seconds(id, now) + " 秒"));
        return true;
    }
    Vector direction = player.getLocation().getDirection().setY(0);
    if (direction.lengthSquared() < 0.0001) {
        player.sendMessage(Component.text("请平视前方再冲刺"));
        return true;
    }
    player.setVelocity(direction.normalize().multiply(0.9).setY(0.2));
    cooldowns.start(id, now, 30_000L);
    player.sendMessage(Component.text("冲刺！下次需要等30秒"));
    return true;
}
@Override
public void onDisable() { cooldowns.clear(); }
}

```

13 两名玩家才能揭穿共享冷却

单人测试看不出“所有人共用同一个截止时间”的错误。请两名同学分别连接测试服，按下表走一遍，再检查退出重进和时间边界。

两名玩家才能揭穿共享冷却

Alex先用

Alex进入冷却

Steve马上用

Steve仍可使用

Alex再用

只收到剩余时间提示

你的核对清单：让学生解释 readyAt、now和remaining的单位，再解释为什么不能靠退出清除记录。把“数据何时消失”写进功能说明。

核对表

操作	预期
Alex使用后Steve立即使用	两人都成功
Alex立刻再次使用	提示剩余秒数
冷却中退出并立即重进	仍受剩余冷却限制
近乎竖直看天后使用	提示平视，不新增冷却
带额外参数 /dash now	显示用法，不新增冷却
控制台输入 dash	清楚提示，无类型转换错误
停服重启后使用	内存冷却已清空

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

独立记录开采数

铁镐

测试1ms与0ms

钻石镐

玩家和技能两层独立键

下一站：第35课用调度器让倒计时自动更新。
官方延伸阅读：jd.papermc.io/paper/1.21.11/org/bukkit/entity/Player.html

- 木镐：用Map记录每位玩家获准开采的次数，未出现过的玩家显示0。
- 铁镐：把冷却设为10秒，验证剩余1毫秒显示1秒，截止时显示0秒。
- 钻石镐：为冲刺、治疗、工具包各设独立冷却，设计玩家与技能两层键，并测互不影响。
- 验收要求：用两个UUID和可控制的测试时间，验证互不影响、结束前1毫秒、恰好结束、退出后重新加入。

核对表

我能独立完成	已掌握	再练一次
用UUID关联玩家记录	☐	☐
能算剩余时间并取整	☐	☐
能解释退出与重启策略	☐	☐

可独立运行的规则验证

在项目内新建 checks/L34RuleCheck.java，完整保存下面代码。新建 checks/sources.txt，写入下面两条路径，然后从项目根目录执行命令。这是普通 Java 测试，不是插件主类。

L34RuleCheck.java • 完整文件

```
import de.teacherma.lesson34.Cooldowns;
import java.util.UUID;

public class L34RuleCheck {
    public static void main(String[] args) {
        Cooldowns c = new Cooldowns();
        UUID alex = new UUID(0, 1), steve = new UUID(0, 2);
        c.start(alex, 1000L, 30_000L);
        System.out.println(c.remaining(alex, 1000L));
        System.out.println(c.remaining(alex, 30999L));
        System.out.println(c.seconds(alex, 30999L));
        System.out.println(c.remaining(alex, 31000L));
        System.out.println(c.remaining(steve, 2000L));
    }
}
```

运行核对

```
30000
1
1
0
0
```

checks/sources.txt • 完整内容

```
src/main/java/de/teacherma/lesson34/Cooldowns.java
checks/L34RuleCheck.java
```

项目根目录终端命令 • 分别执行两行

```
javac -encoding UTF-8 -d checks/out @checks/sources.txt
java -cp checks/out L34RuleCheck
```

纯规则通过只证明数字合同，权限、玩家动作、事件登记和服务器加载仍按前面的游戏验收表检查。

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/>