

第35课

让任务按时间发生

把稍后执行交给调度器



服务器需要继续工作，不能陪倒计时一起睡觉。用 tick 安排延迟和重复任务，保存任务句柄，保证重复开始、手动取消和停服都有明确结果。

前置背包：Lambda、命令和对象字段。新项目BlockAcademyL35继续使用普通Paper1.21.11、Java21与第31课构建脚本。

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 服务器用 tick 数拍子

计时员想等一秒，竟站在服务器主线程门口睡觉，身后的玩家动作排起长队。调度器可以约好稍后再做，让这条忙碌走廊继续通行。

tick 是服务器工作的节拍。正常20 TPS时，20 tick约一秒；卡顿时现实时间会更长。重复任务要保存任务句柄，结束、取消和停用时都能叫停。

服务器用 tick 数拍子

tick
服务器工作节拍

20TPS
20tick约1秒

低TPS
现实等待更长

不能拿毫秒冒充tick：第34课的30000是毫秒；调度器的delay和period是tick。20000tick理想情况下约1000秒，不是20秒。

明确单位 • 位置见本节说明

```
long delayTicks = 5L * 20L;  
long durationTicks = 60L * 20L;
```

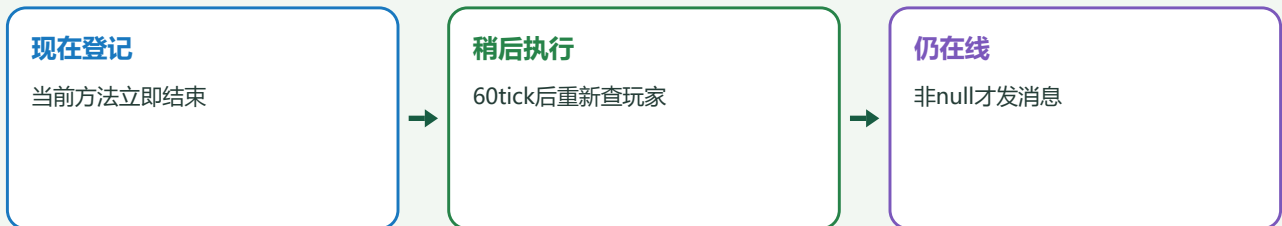
核对表

目标	tick数
3秒	60
10秒	200
60秒	1200

02 稍后再做 不要原地睡觉

`runTaskLater` 把任务交给服务器，等指定tick后执行，当前方法可以先结束。`Thread.sleep`会堵住当前线程；放在服务器主线程里，会让其他玩家一起陪你罚站。

稍后再做 不要原地睡觉



玩家可能已经走了：延迟执行时重新按UUID查找并检查null。代码例子需要import `java.util.UUID`与`org.bukkit.entity.Player`。

三秒后给在线玩家发消息 • 位置见本节说明

```

UUID id = player.getUniqueId();
getServer().getScheduler().runTaskLater(this, () -> {
    Player online = getServer().getPlayer(id);
    if (online != null) {
        online.sendMessage(Component.text("三秒提示到达!"));
    }
}, 60L);
  
```

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。

危险的主线程等待 • 故意错误，先预测再修复

```
Thread.sleep(3000); // 不要在命令或同步事件中这样等
```

03 重复任务需要自己的状态

runTaskTimer 接收首次延迟和重复间隔。练习活动立即广播60秒，但第一次减数要等20tick；如果延迟写0并马上减数，开场就少一秒。

重复任务需要自己的状态



三个位置：this是插件所有者；第一个20L是首次延迟；第二个20L是以后每次的间隔。示例需要import org.bukkit.scheduler.BukkitRunnable。

十秒倒计时小实验 • 位置见本节说明

```
new BukkitRunnable() {  
    int seconds = 10;  
    @Override  
    public void run() {  
        seconds--;  
        getServer().broadcast(Component.text("剩余 " + seconds));  
        if (seconds <= 0) cancel();  
    }  
}.runTaskTimer(this, 20L, 20L);
```

04 拿好任务票据 才能叫停

调度器返回 BukkitTask，相当于任务票据。插件把它存在字段active里。active为null表示空闲；有票据时再收到start，就告诉操作者已经在计时。

停下来想一想：活动已经运行时再执行开始命令，会不会多一个倒计时？停止后怎样证明任务真的取消？

拿好任务票据 才能叫停

active为空

可以开始

active非空

拒绝第二次启动

统一停止

cancel后清空字段

想通这一步：实现应拒绝重复启动。停止后继续观察超过一个间隔，不能再有计时消息；再开始一局应只有一份倒计时。

统一取消方法 • 位置见本节说明

```
private void stopTimer() {  
    if (active != null) active.cancel();  
    active = null;  
}
```

防止重复启动 • 位置见本节说明

```
if (active != null) sender.sendMessage("活动已经在计时");  
else startTimer();
```

05 世界操作留在服务器主线程

本课全部使用同步调度，广播和世界操作在服务器主线程执行。异步任务适合某些耗时计算或I/O，但许多Bukkit世界API不能从异步线程访问；异步还会带来共享数据竞争。

世界操作留在服务器主线程

同步短回调

操作玩家与世界

耗时I/O

不能堵住每秒回调

普通Paper

本章不宣称支持Folia

不要套错调度器：Folia使用不同的线程与区域模型。本课的BukkitScheduler示例不能直接当作Folia支持方案。服务器卡顿时，tick倒计时也会变慢。

保持每次回调很短 • 位置见本节说明

```
// 每次只更新一个计数并决定是否广播
seconds--;
boolean announce = seconds % 10 == 0 || seconds <= 5;
```

核对表

工作	本课处理原则
发送活动广播	同步任务
修改方块或玩家状态	同步任务
大量文件或网络I/O	不放进本课每秒回调
普通Paper与Folia	本课只适用普通Paper

06 活动计时器的蓝图

挑战提供 `/eventtimer start` 和 `/eventtimer stop`。控制台也可以操作，因为全服活动不需要操作者有背包。默认仅op有管理权限，普通玩家只能听广播。

活动计时器的蓝图

start

管理员启动一局

stop

取消当前任务

自然结束

广播一次并回到空闲

项目路径：主类位于src/main/java/de/teacherma/lesson35/EventTimerPlugin.java。settings中的项目名设为BlockAcademyL35，构建脚本沿用第31课。

完整 plugin.yml • 完整文件

```
name: BlockAcademyL35
version: 1.0.0
main: de.teacherma.lesson35.EventTimerPlugin
api-version: '1.21.11'
author: TeacherMa
website: https://www.teacher-ma.de
commands:
  eventtimer:
    description: 管理课堂活动计时器
    usage: /eventtimer <start|stop>
permissions:
  blockacademy.timer:
    default: op
```

- 开始时广播60秒；之后每10秒广播一次，最后5秒每秒广播。
- 重复start不创建第二个任务；stop只取消当前任务。
- 自然结束只广播一次；结束后可以重新开始。

07 活动计时程序上半部

下面两页合起来是完整 EventTimerPlugin.java。先查权限与参数，再分派start和stop；字段active保存共同状态。控制台输入eventtimer start，不带斜杠。

活动计时程序上半部

入口检查

权限与一个参数

开始分支

先确认没有活动

停止分支

取消且如实反馈

主类第1部分 · 本段1/2，顺序连接

```
package de.teacherma.lesson35;

import java.util.Locale;
import java.util.Objects;
import net.kyori.adventure.text.Component;
import org.bukkit.command.Command;
import org.bukkit.command.CommandSender;
import org.bukkit.plugin.java.JavaPlugin;
import org.bukkit.scheduler.BukkitRunnable;
import org.bukkit.scheduler.BukkitTask;

public final class EventTimerPlugin extends JavaPlugin {
    private BukkitTask active;
    @Override
    public void onEnable() {
        Objects.requireNonNull(getCommand("eventtimer")).setExecutor(this);
    }
    @Override
    public boolean onCommand(CommandSender sender, Command command,
                             String label, String[] args) {
        if (!sender.hasPermission("blockacademy.timer")) {
            sender.sendMessage("只有活动管理员可以操作计时器");
            return true;
        }
        if (args.length != 1) {
```

07 活动计时程序上半部（续）

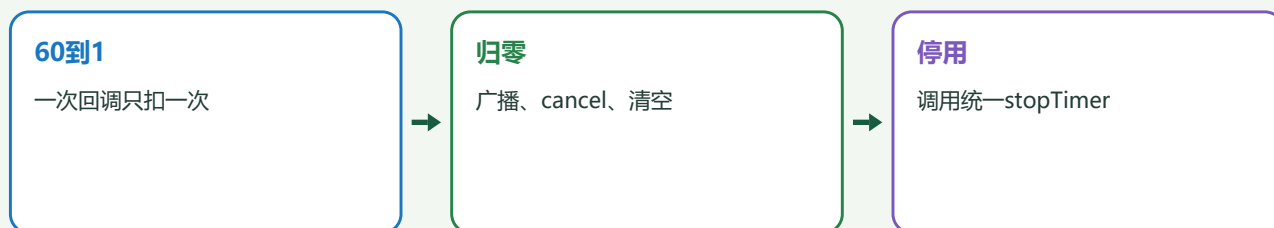
主类第1部分 • 本段2/2, 顺序连接

```
        sender.sendMessage("用法: /eventtimer start 或 stop");
        return true;
    }
    switch (args[0].toLowerCase(Locale.ROOT)) {
        case "start" -> {
            if (active != null) sender.sendMessage("活动已经在计时");
            else startTimer();
        }
        case "stop" -> {
            if (active == null) sender.sendMessage("当前没有活动");
            else {
                stopTimer();
                getServer().broadcast(Component.text("活动已取消"));
            }
        }
        default -> sender.sendMessage("参数只能是 start 或 stop");
    }
    return true;
}
```

08 活动计时程序下半部

继续上一页。自然结束时在回调里cancel，随后清空active；手动停止和插件停用都调用统一的stopTimer。三种结束路径最终回到空闲。

活动计时程序下半部



运行节奏：在理想20TPS下，第一次回调在约1秒后，归零在约60秒后。每秒只做一次短计算，不创建60个线程，也不调用sleep。

主类第2部分 • 顺序接在同一源文件

```

private void startTimer() {
    getServer().broadcast(Component.text("活动开始，剩余60秒"));
    active = new BukkitRunnable() {
        private int seconds = 60;
        @Override
        public void run() {
            seconds--;
            if (seconds <= 0) {
                getServer().broadcast(Component.text("时间到，活动结束! "));
                cancel();
                active = null;
                return;
            }
            if (seconds % 10 == 0 || seconds <= 5) {
                getServer().broadcast(Component.text("剩余 " + seconds + " 秒"));
            }
        }
    }.runTaskTimer(this, 20L, 20L);
}

private void stopTimer() {
    if (active != null) active.cancel();
    active = null;
}

@Override
public void onDisable() { stopTimer(); }
}
  
```

09 检查开头 结尾 和再次启动

计时器最容易在边界出错：一启动就扣秒、结束广播两遍、停止后还在说话。用下表逐项验收，把日志中的顺序和游戏里的消息对起来。

检查开头 结尾 和再次启动

开头

不能立即少一秒

结尾

不能重复广播

再开始

只有一份新倒计时

现实时间比较：可用秒表观察，但不要把低TPS下的变慢判成减数公式错误。若活动必须严格按现实时间截止，应结合截止时间与每次回调重新计算。

核对表

测试	预期
start后立即再start	只存在一个倒计时
运行中stop	取消消息一次，之后安静
空闲时stop	提示当前没有活动
让60秒完整结束	50、40、30、20、10、5到1、结束
结束后再次start	重新从60秒开始
计时中stop服务器	插件清理任务
普通玩家start	权限提示，无任务



马老师教编程
www.teacher-ma.de

12 / 14

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

延迟提示兼顾离线

铁镐

解释10秒倒计时

钻石镐

昼夜提醒可启停

下一站：第36课继续把服务器规则变成可配置的功能。
官方延伸阅读：docs.papermc.io/paper/dev/scheduler/

- 木镐：用runTaskLater在3秒后提示玩家，玩家退出时不报错。
- 铁镐：制作10秒倒计时，说明首次延迟、间隔、结束条件分别在哪。
- 钻石镐：做一个可启停的昼夜提醒器，保证重复start不叠加，并在停用时取消。
- 验收要求：记录开始、剩余1、结束、手动取消、再次开始和停服六个时刻。明确本课采用tick计时。

核对表

我能独立完成	已掌握	再练一次
能区分毫秒和tick	☐	☐
会保存并取消任务	☐	☐
理解同步任务的边界	☐	☐

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/>