**以下内容必须打前置后才能进行对mod的编写和执行！没有前置一切免谈。**

**以下内容必须打前置后才能进行对mod的编写和执行！没有前置一切免谈。**

**以下内容必须打前置后才能进行对mod的编写和执行！没有前置一切免谈。**

**好了避免完瞎子了，我们开始正文**

**前言：受够了天书？那就自己写一本**

**如果你点开了这篇教程，大概率和曾经的我一样：揣着一肚子绝妙的、涩涩的任务构想，却在踏出第一步时就撞上了一堵名为“教程”的墙。**

**国内相关的教程？几乎是片空白。于是我们只能把目光投向国外，找到那些翻译过来的指南。然后呢？然后你就看到了一篇篇堪称“后现代主义”的文档——它们热情地把一个完整的世界拆解成一堆孤零零的代码块，然后告诉你：“看，这就是活塞，这是传送门，现在你可以去玩《我的世界》了。”**

**我反正是什么都没看懂。**

**那些所谓的“教程”，读起来就像一本充满了专业术语却不教你如何造句的词典。它告诉你 setCheckpoint 是核心，却不解释它在实际应用中如何像一个“传送陷阱”一样运作；它罗列了几十个 condition，却不教你如何将它们组合成“必须在裸体且戴着手铐的情况下跳舞”的有趣指令。这种感觉，就像是有人递给你一堆乐高零件，却不给你图纸，还催着你拼个航母出来。**

**这太荒谬了。**

**在经历了无数次对着报错代码抓耳挠腮、通读大量晦涩的原始文档、以及把游戏当成代码实验室反复试错之后，我终于摸索出了一套能让正常人理解的逻辑和方法。我发现，任务制作的本质并非是堆砌代码，而是像导演一样，去设计场景、安排情节、引导玩家的情绪。**

**为了让后来者不必再重复我那段痛苦的“开荒”经历，我决定动手写下这篇教程。它不是对官方文档的生硬翻译，而是我个人经验的总结和提炼，是我踩过的每一个坑、解决的每一个BUG的结晶。在这个过程中，我也借助了AI的强大整合与归纳能力，力求将复杂零散的知识点，梳理成一套清晰、易懂、并且可立即上手实操的完整体系。**

**本教程的目标只有一个：让你从一个对着代码满头问号的“玩家”，转变为一个能够随心所欲创造自己世界的“导演”。**

**现在，忘掉那些天书吧。让我们从零开始，一步一步地，把你的想法变成现实。**

**↑哈哈这也是ai写的。我是懒狗，我只想看到更多的mission**

**作者无所谓是谁，别把教程拿去当引流的盈利道具就行。虽然可能有人这么干。  
好了废话少说，故事开始。**

**作者：佚名———致谢某个愿意分享游戏给我的大佬。**

**这个作品属于互联网。**

目录

[自定义任务制作完全教程 - 序章 & 第一章 3](#_Toc208479702)

[第二章：舞台搭建 - 定义 Zones（区域） 7](#_Toc208479703)

[第三章：事件核心 - Checkpoints 与 Condition 对象 11](#_Toc208479704)

[第四章：逻辑与判断 - 精通 condition 字符串—“你要满足我的”条件“ 16](#_Toc208479705)

[第五章：导演之手 - 触发强制命令 21](#_Toc208479706)

[第六章：流程控制 - 构建动态的任务结构—非线性结构的诞生 31](#_Toc208479707)

[第七章：调试、润色与发布 37](#_Toc208479708)

## 自定义任务制作完全教程 - 序章 & 第一章

**序言：你将学到什么？**

欢迎来到自定义任务制作的世界！通过这个系列教程，你将从一个对代码一无所知的新手，成长为能够独立设计、编写和调试复杂任务的创作者。你将能够创造出引导玩家完成特定挑战、触发特殊事件、甚至讲述一个简单故事的互动体验。

本教程将循序渐进，确保你掌握每一个核心概念：

* **第一章：基础入门** - 准备工具，理解最基本的文件结构和语法规则。
* **第二章：舞台搭建** - 学习如何使用 Zones 在游戏世界中划定触发事件的“地点”。
* **第三章：事件核心** - 深入 Checkpoints 和 Condition，学习如何设置任务目标和文本提示。
* **第四章：逻辑与判断** - 掌握 condition 字符串的编写，实现对玩家状态、动作、位置的精确检测。
* **第五章：导演之手** - 学习使用 oncomplete 等命令，实现传送、换装、物品操作等强制效果。
* **第六章：流程控制** - 探索事件跳转、分支、循环和随机事件，让你的任务变得动态和不可预测。
* **第七章：调试与发布** - 学会如何排查错误，并完善你的任务。

现在，让我们开始第一章的学习。

**第一章：基础入门 - 构建你的第一个任务文件**

在这一章，我们将完成所有前期准备工作，了解任务文件的基本构成和必须遵守的“黄金法则”。这是后续一切学习的基石。

**1.1 必备工具与基本概念**

1. **文本编辑器**:  
   虽然理论上任何文本编辑器（如 Windows 自带的记事本）都可以，但我强烈推荐使用 **Notepad++** 或 **Visual Studio Code**。
   * **为什么？** 因为它们支持“语法高亮”。当你编写 JSON 文件时，它们会自动为不同的代码部分（如关键字、字符串、数字）标上不同颜色，这能让你一眼就看出拼写错误或格式问题，极大降低出错率。

**笔者注：你用txt编辑一点毛病也没有，最后改个后缀就行。有问题丢给ai检测，没问题就跑，就这么简单。**

1. **JSON 简介**:  
   我们的任务文件使用一种名为 **JSON** (JavaScript Object Notation) 的格式。别被这个名字吓到，它其实非常简单，本质上是一种存储和组织数据的方式。你可以把它想象成一个信息登记表。
   * **核心是“键值对”**: "键": "值"。例如: "姓名": "张三"。
   * **语法规则**:
     + 所有的“键”（Key）都必须用英文双引号 " 包起来。
     + “值”（Value）的类型有很多：
       - **字符串 (String)**: 文本，必须用双引号 " 包起来。例如: "前往公园"。
       - **数字 (Number)**: 不需要引号。例如: 5, 3.14。
       - **布尔值 (Boolean)**: 只有 true 和 false 两种，代表“是”与“否”，不需要引号。
       - **数组 (Array)**: 一个值的列表，用方括号 [] 包起来，值之间用逗号 , 分隔。例如: [ "苹果", "香蕉", "橘子" ]。
       - **对象 (Object)**: 一组键值对的集合，用花括号 {} 包起来。我们的整个任务文件就是一个巨大的对象。

**笔者注：这部分内容就好像大学教程里面开头的发展历史一样，属于你不知道也不影响编写的东西，看不懂算了。**

1. **游戏内调试工具 (你的挚友)**:
   * **F9**: 获取当前状态。这是你最重要的工具！它会实时显示你当前的位置坐标 (x, y, z)、朝向、以及你正在执行的动作等关键信息。制作任务时，你需要频繁地使用它来获取坐标。
   * **Ctrl+F9**: 显示游戏内所有官方任务、服装 (Cosplay) 和道具的 ID 列表。当你需要让玩家穿上某件特定衣服时，就要从这里查找它的准确 ID。
   * **Alt+F9**: 重新加载所有自定义任务。当你在游戏外修改并保存了你的 .json 文件后，切回游戏按下此组合键，你的修改就会立刻生效，无需重启游戏。
   * **↑我不知道上面除了F9之外的是不是真的**

**笔者注：关键内容！！！！记住这个F9，别到处问别人当前状态怎么获取。**

**1.2 创建你的第一个任务文件**

1. 打开你的游戏安装目录。
2. 找到名为 CustomMissions 的文件夹。如果不存在，请手动创建一个。
3. 在该文件夹内，右键新建一个文本文档。
4. 将其重命名为 我的第一个任务.json。**确保文件后缀是 .json**，而不是 .txt。

**1.3 任务文件的顶层结构**

现在，用你选择的文本编辑器打开这个空白的 .json 文件，然后将下面的基本结构复制进去。这是所有任务都必须具备的“骨架”。

例如：

{

"title": "我的第一个任务",

"listmission": true,

"addtitleinlist": true,

"addtitleinpanel": true,

"zones": [

],

"checkpoints": [

]

}

让我们逐一解析这些顶层“键”的含义：

* "title": 你的任务标题。它会显示在任务列表和进度面板中，也是某些情况下默认的提示文本。
* "listmission": 设为 true，你的任务就会和官方任务一起出现在任务列表中。设为 false 则会隐藏。
* "addtitleinlist": 设为 true，会自动将你的 title 添加到任务列表的描述区域。
* "addtitleinpanel": 设为 true，会自动将你的 title 添加到游戏界面右侧的任务进度面板中。
* "zones": 一个数组（目前为空 []）。这里将定义任务中所有需要用到的“地点”或“触发区域”。我们将在第二章详细讲解。
* "checkpoints": 一个数组（目前为空 []）。这里将定义任务的所有具体步骤、逻辑和事件。这是任务的“心脏”，我们将在后续章节中花费大量时间来学习它。

**1.4 黄金法则与常见错误**

在你开始填充内容之前，请牢记以下两条会让你免于痛苦的黄金法则：

**法则一：JSON 文件中绝对不能有注释！**

在很多编程语言中，你可以用 // 或 /\* ... \*/ 来写一些注释方便自己理解。但在 JSON 中，这是**绝对禁止**的。任何注释都会导致游戏无法读取文件。

**错误示范**:

例如：

{

// 这是我的任务标题

"title": "我的第一个任务"

}

游戏加载这个文件时会直接失败，而你可能完全不知道为什么。请务必保持你的 .json 文件纯净。

**法则二：小心处理逗号！**

逗号用于分隔数组中的元素或对象中的键值对。最常见的错误有两个：

1. **忘记加逗号**: 在两个并列的元素之间缺少逗号。
2. **画蛇添足**: 在最后一个元素的后面多加了一个逗号。

**错误示范 (结尾多了逗号)**:

例如：

{

"title": "我的第一个任务",

"listmission": true, <-- 这里多了一个逗号

}

**正确示范**:

例如：

{

"title": "我的第一个任务",

"listmission": true

}

一个对象或数组的最后一个元素后面，永远不要加逗号。

**第一章总结**

恭喜你！你已经成功创建了你的第一个任务文件，并了解了它的基本结构和最重要的语法规则。你现在拥有一个可以随时启动游戏进行测试的、虽然是空白的但格式完全正确的任务“骨架”。

**在下一章，我们将开始为这个骨架赋予血肉，深入探讨任务的“舞台”——zones（区域）。你将学会如何利用 F9 获取坐标，并在游戏世界中精确地划定出触发任务的第一个光圈。**

## 第二章：舞台搭建 - 定义 Zones（区域）

如果说 Checkpoints (检查点) 是任务剧本中的“情节”，那么 Zones (区域) 就是这些情节上演的“舞台”。Zone 的作用是在游戏的广阔地图上定义一个具体的、有形的范围。当玩家进入这个范围时，与之关联的事件才有可能被触发。

**2.1 什么是 Zone？**

Zone 是一个逻辑上的区域定义。在我们的 .json 文件中，它位于顶层的 "zones" 数组内。一个任务可以包含任意多个 Zone，每个 Zone 都需要有一个独一无二的 id，以便在后续的 Checkpoints 中被引用。

**基本结构**:  
让我们回到第一章创建的骨架，在 "zones" 的方括号 [] 中添加我们的第一个 Zone。

例如：

{

"title": "我的第一个任务",

...

"zones": [

{

"id": "start\_point\_zone",

"areas": [

{

"type": "sphere",

"stage": "Park",

"x": 66.72,

"y": 50.04,

"z": 30.93,

"r": 2

}

]

}

],

"checkpoints": [

...

]

}

这个结构看起来有点复杂，别担心，我们来逐层拆解。

* "zones": [ ... ]: 这是一个数组，意味着你可以定义多个 Zone。每个 Zone 都是一个被花括号 {} 包裹的对象，彼此之间用逗号 , 分隔。
* { "id": "start\_point\_zone", ... }: 这是我们定义的第一个 Zone。
  + "id": **极为重要**。这是该 Zone 的唯一名称或标识符。后续在 Checkpoints 中，我们需要通过这个 id 来告诉游戏：“当玩家进入名为 start\_point\_zone 的这个区域时，请执行某某操作。”
    - **命名建议**: 使用清晰、有描述性的英文名称，例如 fountain\_area (喷泉区域), mission\_start\_zone (任务开始区域)。避免使用 1, 2, 3 这种难以记忆和辨认的 id。
* "areas": [ ... ]: 每个 Zone 内部都包含一个 "areas" 数组。这允许你用多个不同的形状组合成一个复杂的区域。但在大多数情况下，一个 Zone 里只包含一个 area 就足够了。

**2.2 Area 的构成 - 区域的形状与位置**

Area 定义了区域的具体物理属性：它在哪里，是什么形状，有多大。

* "type": 区域的几何形状。
  + "sphere" (球体): **最常用、性能最好**的选择。它由一个中心点和一个半径定义，非常适合设定大多数圆形触发区域。
  + "cylinder" (圆柱): 由一个底面中心点、半径和高度定义。适用于需要限定垂直范围的场景，比如电梯。
  + "cuboid" (长方体): 由两条平行的底边中心点坐标和宽度、高度定义。计算最复杂，性能开销最大，请尽量避免使用，除非必须定义矩形区域。
* "stage": 区域所在的地图或场景。**这个值必须与游戏内的地图名称完全一致**，否则区域不会生效。
  + **当前可用地图列表 (stage 值)**:
    - Park: 中央公园
    - StationFront: 市中心
    - ShoppingMall: 购物中心
    - FashionShop: 服装店
    - Convenience: 便利店
    - Mansion: 公寓楼
    - Apart: 你的家
    - Residence: 住宅区 (离开家后的第一个区域)
* **坐标与尺寸参数 (以最常用的 sphere 为例)**:
  + "x", "y", "z": 球心的三维坐标。
  + "r": 球体的半径，单位大致为米。"r": 2 大约可以覆盖公园**你传送进去的那条路的应该有1/2到3/4的区域，直径4很大了。**

**2.3 实战：如何获取坐标并设定第一个 Zone**

现在，理论讲完了，我们来动手实践。我们的目标是：**在公园的中央喷泉附近设置一个任务起始点。**

1. **启动游戏**：加载你包含空白任务文件的存档。
2. **前往目标地点**：操作你的角色，跑到公园的中央喷泉。
3. **打开控制台**：按下键盘上的 **F9** 键。屏幕上会出现一个半透明的控制台窗口。
4. **寻找坐标信息**：在控制台输出的信息中，找到类似下面这样的一行：  
   Player position: (x: 75.57, y: 52.09, z: 83.63)
   * 这串数字就是你角色当前位置的精确坐标！
5. **记录坐标**：将这三个数值（x, y, z）复制或记录下来。
6. **修改你的 .json 文件**：回到你的文本编辑器，将刚才记录的坐标填入你的 Zone 定义中。我们再给它起一个更有意义的 id，比如 fountain\_zone。

例如：

"zones": [

{

"id": "fountain\_zone",

"areas": [

{

"type": "sphere",

"stage": "Park",

"x": 75.57,

"y": 52.09,

"z": 83.63,

"r": 3

}

]

}

]

* + 我将半径 "r" 设置为 3，这样触发范围会大一些，玩家更容易进入。
  + **↑这是ai写的，太他妈大了。0.5半径就差不多站个人大小，1完全够看，而且只要有高亮也很明显。**

**2.4 区域的可视化与隐藏**

默认情况下，游戏会在你定义的 Zone 区域地面上显示一个发光的圆环，方便你调试和确认位置是否正确。但在正式发布的任务中，你可能希望隐藏它，以增加沉浸感。

* "outlinehidden": 在 area 对象中添加这个键，并将其值设为 true，即可隐藏光环。
* "compasshidden": 将其值设为 true，可以隐藏屏幕上方白色罗盘（雷达）上的目标点指示。

**示例：一个完全隐藏的区域**

例如：

{

"id": "hidden\_trap\_zone",

"areas": [

{

"type": "sphere",

"stage": "StationFront",

"x": -227.00,

"y": 0.17,

"z": -62.82,

"r": 1,

"outlinehidden": true,

"compasshidden": true

}

]

}

这个 Zone 在游戏中将是完全不可见的，非常适合制作陷阱或者需要玩家自行探索的任务。

**第二章总结**

你已经成功地在游戏世界中定义了一个具体的、可触发事件的区域！这是将你的任务构想与游戏世界连接起来的关键一步。你学会了：

* Zone 的基本结构和 id 的重要性。
* 如何使用 F9 获取坐标，并定义一个 Area 的位置、形状和大小。
* 如何控制 Zone 在游戏中的可见性。

**在下一章，我们将正式进入任务的核心——Checkpoints（检查点）。你将学习如何将刚刚创建的 fountain\_zone 与一个具体的任务目标关联起来，让玩家在进入这个区域时，看到第一条任务指示。**

## 第三章：事件核心 - Checkpoints 与 Condition 对象

Checkpoints 是一个数组，它包含了任务的所有步骤。游戏会像阅读剧本一样，按照你在数组中定义的顺序，一个接一个地执行这些检查点。每个检查点都定义了一个具体的目标、触发条件以及完成后要执行的动作。

**3.1 Checkpoint 的基本构成**

**一个故事只有一个checkpoints。**

一个最基础的检查点（Checkpoint）由四个关键部分组成：id、zone、condition 和 nextcheckpoint。让我们在之前文件的基础上，添加第一个检查点。

例如：

{

...

"zones": [

{

"id": "fountain\_zone",

"areas": [ ... ]

}

],

"checkpoints": [

{

"id": "mission\_start",

"zone": "fountain\_zone",

"condition": {

"description": "欢迎来到中央喷泉。站在这里不要动，任务即将开始...",

"duration": 5

},

"nextcheckpoint": {

"selectortype": "SpecificId",

"id": "second\_step"

}

}

]

}

现在，我们来详细解析这个检查点的每一个“键”：

* "id": 检查点的唯一标识符。与 Zone 的 id 类似，它用于在任务流程中被引用，尤其是在进行事件跳转时。同样，建议使用有意义的英文名，如 mission\_start, first\_task, punishment\_branch。
* "zone": **连接舞台与剧情的桥梁**。这个值必须是你在 "zones" 数组中定义过的某个 Zone 的 id。在这个例子中，"zone": "fountain\_zone" 意味着，**玩家必须首先进入我们第二章定义的喷泉区域，这个检查点的逻辑才会开始被检测**。如果玩家不在这个区域内，mission\_start 检查点就如同不存在一样。
* "condition": 一个至关重要的**对象** {}。它描述了玩家在进入指定 zone 后，**需要做什么**才能完成这个检查点。我们稍后会详细展开。
* "nextcheckpoint": 一个**对象** {}，用于告诉游戏，当这个检查点成功完成后，接下来应该去哪个检查点。
  + "selectortype": 选择下一个检查点的方式。最常用的是 "SpecificId"，表示“明确指定下一个 ID”。
  + "id": 在 selectortype 为 "SpecificId" 时使用，其值是下一个检查点的 id。在这个例子中，任务完成后会跳转到名为 second\_step 的检查点。
  + **如果省略 nextcheckpoint 会怎样？** 游戏会默认执行 checkpoints 数组中的下一个检查点。虽然方便，但为了逻辑清晰和避免潜在的BUG，**强烈建议为每个检查点明确指定 nextcheckpoint**。（当然，省事也能用。）

**3.2 深入 Condition 对象 - 任务目标的描述**

Condition 对象是检查点的“灵魂”，它定义了玩家需要达成的目标和看到的指示。让我们聚焦于 condition 内部的键值对。

例如：

"condition": {

"description": "欢迎来到中央喷泉。站在这里不要动，任务即将开始...",

"duration": 5,

"condition": "Action\_None",

"hidepanel": false,

"rp": 50

}

* "description": 显示在游戏界面右侧任务进度面板上的提示文本。这是你与玩家沟通的主要方式，告诉他们当前需要做什么。
* "duration": 持续时间，单位为秒。玩家需要**持续满足** "condition" 字符串中定义的条件达到这个秒数，才算完成检查点。在这个例子中，玩家需要在喷泉区域内什么都不做 (Action\_None) 持续 5 秒。
* "condition": **一个字符串**，定义了需要被检测的玩家状态或动作。这是任务逻辑的核心，我们将在第四章专门用一整章来讲解它的各种写法和组合。目前，你只需要知道 "Action\_None" 代表“玩家站立且没有任何动作”。
* "hidepanel": 一个布尔值 (true 或 false)。
  + false (默认值): 任务进度面板正常显示。
  + true: 隐藏整个任务进度面板。
  + "IfNotCondition": 一个特殊值，表示仅当玩家**不满足** "condition" 条件时才隐藏面板。这可以用于创造一种“当你做对时，提示才会出现”的效果。
* "hideprogress": 一个布尔值 (true 或 false)。设为 true 时，会隐藏进度条，但保留任务面板的文本。
* "reset": 一个布尔值 (true 或 false)，默认为 true。
  + true: 如果玩家中途离开 zone 或者不再满足 condition 条件，duration 的计时会**重置为0**。
  + false: 即使玩家中途离开或中断，计时器也会**暂停**，当玩家回来并重新满足条件时，会从暂停处继续计时。
* "rp": 一次性 RP 奖励。当该检查点完成时，给予玩家指定数量的 RP 点数。

**笔者注：这一段贼他妈关键认真看**

**3.3 实践：构建一个完整的双步骤任务**

现在，让我们利用所学知识，扩展示例，创建一个包含两个步骤的简单任务。

**任务目标**：

1. 引导玩家前往公园喷泉 (fountain\_zone)。
2. 要求玩家在喷泉区域站立不动 5 秒。
3. 完成后，引导玩家前往公园的另一个地点——公共厕所 (toilets\_zone)。
4. 要求玩家进入厕所区域即可完成任务。

**第一步：定义两个 Zones**  
(假设你已经用 F9 获取了公园厕所区域的坐标)

例如：

"zones": [

{

"id": "fountain\_zone",

"areas": [

{

"type": "sphere", "stage": "Park",

"x": 75.57, "y": 52.09, "z": 83.63, "r": 3

}

]

},

{

"id": "toilets\_zone",

"areas": [

{

"type": "sphere", "stage": "Park",

"x": 123.96, "y": 49.21, "z": 123.17, "r": 4

}

]

}

]

**注意**：两个 Zone 对象 {} 之间用逗号 , 分隔。

**第二步：定义两个 Checkpoints**

例如：

"checkpoints": [

{

"id": "step\_one\_goto\_fountain",

"zone": "fountain\_zone",

"condition": {

"description": "请前往公园中央喷泉，并在光圈内站定5秒。",

"duration": 5,

"condition": "Action\_None"

},

"nextcheckpoint": {

"selectortype": "SpecificId",

"id": "step\_two\_goto\_toilets"

}

},

{

"id": "step\_two\_goto\_toilets",

"zone": "toilets\_zone",

"condition": {

"description": "干得不错！现在请前往公园的公共厕所。【任务完成】",

"duration": 1,

"rp": 200

}

}

]

**注意**：两个 Checkpoint 对象 {} 之间也用逗号 , 分隔。在第二个检查点中，我们没有设置 condition 字符串，这意味着玩家只要进入 toilets\_zone 区域，条件就立即满足，等待 1 秒后任务完成并获得 200 RP。

**第三章总结**

你已经掌握了创建任务核心逻辑的技能！现在，你不仅能设置“舞台”，还能编写“剧本”，引导玩家完成一系列简单的步骤。你学会了：

* Checkpoint 的完整结构及其四大组成部分。
* 如何通过 "zone" 键将检查点与一个具体的地理位置绑定。
* Condition 对象的内部构成，包括设置任务描述、持续时间和奖励。
* 如何使用 nextcheckpoint 将多个步骤串联成一个完整的任务流。

**在下一章，我们将深入 condition 字符串的奇妙世界。你将学习如何从简单的“站着不动”扩展到检测玩家是否裸体、是否戴着手铐、是否在使用特定道具、是否处于特定姿势等各种复杂的组合条件，让你的任务逻辑实现质的飞跃。**

## 第四章：逻辑与判断 - 精通 condition 字符串—“你要满足我的”条件“

"condition" 字符串是任务逻辑的核心判断依据。它是一个表达式，游戏会不断地检查玩家的当前状态是否满足这个表达式。只有满足了，duration 的计时才会开始。

**笔者注：（所有我主动写的注释都是加粗）以下内容存在歧义，这是ai他吗的局限性不是我的**

**我要先声明：一个事件{}中，在我看到的最简单结构里必然包含两个，zone和condition。这里的condition不是条件而是你可以理解为一个执行内容，执行内容是写这个事件你要干什么**

{

"zone":"cuffshidden",

"condition":

{

"description":"蕾丝内裤！我喜欢，你的腿真漂亮，就该多给我看看",

"duration":5,

"condition":"[Action\_HandcuffsAtMap,AdultToy\_TimerHandcuff]",

"hideprogress":true,

"oncomplete":[

{"type":"unequipCosplay", "parts":["m\_cosplay\_general\_leg\_stocking"]}

]

}

},

**看这个案例，在{}的一级一句中只有两个，zone和condition。一切执行对象比如说话description，等待时间duration，检测是否进行oncomplete的condition和是否隐藏进程的hideprogress都是二级语句，是一级语句condition下的执行内容。所有语句都要有condition，但condition并不只是检测状态（二级语句），也有构成一级语句的执行功能。**

继续

**4.1 单一条件：基础状态检测**

首先，让我们来认识一些最常用的单一条件关键词。这些关键词涵盖了玩家的身体、装备、动作等多个方面。

**身体状态 (Exposed 相关)**

* Naked: 玩家处于完全裸体状态（内衣也脱掉）。
* FrontOpen1: 前部打开一次（例如，解开上衣扣子，露出胸部）。
* FrontOpen2: 前部打开两次（例如，完全敞开上衣，露出上半身）。
* BackOpen: 后部打开（露出臀部）。
* Blindfolded: 被蒙眼（通过玩具眼罩或特定服装实现）。

**装备与道具 (Item / Toy 相关)**

* NoHandcuffs: 未佩戴任何手铐。
* NormalHandcuffs: 佩戴着普通手铐。
* KeyedHandcuffs: 佩戴着需要钥匙的手铐。
* TimedHandcuffs: 佩戴着计时手铐。
* VibrationOff / VibrationLow / VibrationHigh / VibrationRandom: 振动器处于关闭/低档/高档/随机档。
* PistonOff / PistonLow / PistonMedium / PistonHigh / PistonRandom: 活塞处于对应档位。

**动作与姿势 (Action 相关)**

* Action\_None: 站立不动。
* Crouching: 蹲下。
* Sitting: 坐下。
* Peeing: 小便。
* Action\_HipShake: 正在执行“摇臀”动作。
* Action\_UseDildoFloorPussy1: 正在使用地面假阳具进行特定自慰动作。
  + **如何获取动作名？** 具体的动作名称 (Action\_...) 需要在游戏中执行相应动作时，通过按 F9 打开控制台来查看。

**环境与位置**

* InOpenToilet: 位于一间门开着的厕所隔间内。
* NearNPC: 附近有 NPC。
* Watched: 处于 NPC 的视野中。
* IsDayTime: 当前是白天。

**示例：一个要求特定状态的任务**  
假设我们想让玩家在公园厕所里，打开上衣，并且蹲下来。

例如：

{

"id": "toilet\_task",

"zone": "toilets\_zone",

"condition": {

"description": "很好，现在进入隔间，打开你的上衣，然后蹲下来。",

"duration": 3,

"condition": "[FrontOpen2, Crouching, InOpenToilet]"

},

...

}

**4.2 逻辑运算符：组合你的条件**

当单一条件无法满足你的设计需求时，就需要使用逻辑运算符将多个条件组合起来，形成更复杂的判断逻辑。

* **与 (AND)**: 使用方括号 [ ]，并用逗号 , 分隔所有条件。
  + **含义**: 括号内的所有条件**必须同时满足**。
  + **示例**: "[Naked, BackOpen]"
  + **解读**: 玩家必须既是 Naked (裸体) **并且** 臀部也是 BackOpen (暴露) 的。
* **或 (OR)**: 使用圆括号 ( )，并用逗号 , 分隔所有条件。
  + **含义**: 括号内的条件中**至少有一个满足**即可。
  + **示例**: "(VibrationLow, VibrationHigh, VibrationRandom)"
  + **解读**: 玩家的振动器处于低档、高档、或随机档中的**任意一种**状态都可以。
* **非 (NOT)**: 在任意条件或条件组前加上感叹号 !。
  + **含义**: 对该条件或条件组的结果取反。
  + **示例**: "!Dashing"
  + **解读**: 玩家**没有**处于 Dashing (疾跑) 状态。
  + **示例**: "[Naked, !NearNPC]"
  + **解读**: 玩家必须是裸体，**并且**附近**没有** NPC。

**4.3 实战：构建复杂的逻辑判断**

现在，让我们来挑战一些更复杂的任务设计。

**任务目标1：让玩家在裸体状态下，使用振动器自慰（不限档位），同时不能被附近的NPC看到。**

* **分析**:
  1. 需要裸体: Naked
  2. 振动器开启 (低、高、随机档皆可): (VibrationLow, VibrationHigh, VibrationRandom)
  3. 不能被NPC看到: !Watched
  4. 这三个条件需要**同时**满足。
* **condition 字符串编写**:  
  "[Naked, (VibrationLow, VibrationHigh, VibrationRandom), !Watched]"

**任务目标2：要求玩家使用地面假阳具进行阴道或肛门自慰（二选一），并且必须戴着手铐（不限类型）。**

* **分析**:
  1. 阴道自慰动作: Action\_UseDildoFloorPussy1
  2. 肛门自慰动作: Action\_UseDildoFloorAnal1
  3. 以上两个动作**满足其一**即可: (Action\_UseDildoFloorPussy1, Action\_UseDildoFloorAnal1)
  4. 戴着普通手铐: NormalHandcuffs
  5. 戴着计时手铐: TimedHandcuffs
  6. 戴着钥匙手铐: KeyedHandcuffs
  7. 以上三种手铐**满足其一**即可: (NormalHandcuffs, TimedHandcuffs, KeyedHandcuffs)
  8. 动作条件和手铐条件需要**同时**满足。
* **condition 字符串编写**:  
  "[(Action\_UseDildoFloorPussy1, Action\_UseDildoFloorAnal1), (NormalHandcuffs, TimedHandcuffs, KeyedHandcuffs)]"

**4.4 数值比较：超越布尔判断**

某些状态（如物品数量、任务完成次数）不是简单的“是/否”，而是有具体数值的。你可以对这些状态进行比较。

* **支持比较的状态**:
  + Item\_<物品名>: 物品栏中某物品的数量 (例如 Item\_HandcuffKey)
  + Ecstasy: 快感值 (范围 0 到 1)
  + Detection: 暴露风险值 (范围 0 到 1)
  + Rank: 等级 (整数 0~7)

**例如："condition": "[Ecstasy>=0.8, Peeing]",**

* **比较运算符**:
  + == (等于)
  + != (不等于)
  + > (大于)
  + < (小于)
  + >= (大于等于)
  + <= (小于等于)
* **语法**: **被检查的状态必须在运算符的左侧**。
  + **正确**: "Item\_Dildo>=1"
  + **错误**: "1<=Item\_Dildo"
* **示例：要求玩家至少拥有两把手铐钥匙。**
  + "condition": "Item\_HandcuffKey>=2"

**第四章总结**

你已经解锁了自定义任务中最强大、最灵活的工具！通过自由组合条件和逻辑运算符，你现在可以设计出各种富有挑战性和趣味性的任务目标。你学会了：

* 识别和使用涵盖身体、装备、动作、环境的各种单一条件。
* 熟练运用 AND [ ], OR ( ), NOT ! 来构建复杂的逻辑表达式。
* 对带有数值的状态（如物品数量）进行比较判断。

**在下一章，我们将学习如何在玩家满足这些精心设计的条件后，执行我们想要的“导演剪辑”——通过 oncomplete 等命令，实现强制传送、换装、掉落物品等酷炫的自动化效果，将任务的互动性提升到一个全新的高度。**

## 第五章：导演之手 - 触发强制命令

在 condition 对象中**内部**，除了定义“需要满足什么条件”之外，还可以定义“当条件满足/不满足时，要发生什么”。这些强制执行的脚本命令，能让你完全掌控玩家的角色状态、位置甚至物品栏，极大地丰富了任务的可能性。

**笔者：这里面所有的oncomplete都是基于二级对象的！搞不明白二级对象说的啥看第四章开头吐槽！**

**5.1 触发的时机：oncomplete vs. onviolatecondition**

有两个关键的“钩子”（Hooks）可以让你挂载这些强制命令，它们都位于 condition 对象内部，与 "description"、"duration" 等键平级。

* "oncomplete": [ ... ]
  + **触发时机**: 当 "condition" 字符串中的条件**被满足**，并且 duration 的**计时走完**的那一刻，这个数组中的所有命令会按顺序执行。
  + **用途**: 最常见的用途，用于在玩家完成一个步骤后，给予奖励、改变状态、传送至下一地点等。
* "onviolatecondition": [ ... ]
  + **触发时机**: 当玩家的状态\*\*从“满足条件”变为“不满足条件”\*\*的那一刻，立即执行。
  + **用途**: 通常用于实现“惩罚”机制。例如，任务要求玩家保持蹲下，如果玩家中途站起来（违背了 Crouching 条件），就可以触发此处的命令。

**当然，你可以不给任何条件（比如给一个极容易触发的condition），那么就会直接执行。**

**但不给condition（一级condition）99%概率不行，（二级condition）很可能不行，笔者没试过。**

这两个键的值都是一个**数组 []**，意味着你可以放入一个或多个命令对象 {}。

**说人话就是你可以在这个执行里面同时做很多事情**

比如：  
{

"id": "second\_point",

"zone": "second\_zone",

"travelcondition": {

"condition": "VibrationRandom",

"onviolatecondition": [

{

"type": "setCheckpoint",

"selectortype": "SpecificId",

"id": "start\_punishment\_entry"

}

]

},

//这一段代码就是不满足随机时，强制执行事件跳转。

//我们可以通过这种方式实现，我们在某个事件之前一定要保持某个状态否则任务失败。

//如本案例，↓下方的代码只有当前面的任务完成时，走到second\_zone，并且中间不触犯travelcondition的情况下，下面的condition才会执行。

"condition": {

"description": "真乖，小可爱。奖励你更刺激的活塞运动！现在，带上这盏小灯继续前进吧。",

"duration": 1.5,

"oncomplete": [

{

"type": "setPiston",

"level": "High"

},

{

"type": "equipCosplay",

"parts": [

"m\_cosplay\_general\_genital\_plug\_light"

]

}

]

},

"nextcheckpoint": {

"selectortype": "SpecificId",

"id": "end\_point"

}

},

**笔者：这里的nextcheckpoint属于这个{}事件中的一句话！看是不是最简单的方法就是看这段话结束有没有逗号，只有{}大的一个事件内部的最后一句话没有逗号。而事件和事件之间是因为它们分属于最大的{}里面<你可以看代码的第一行，有一个{,这就说明后续所有代码都属于这个内容里面的>**

**<比如一个最简单的{}事件中，zone语句后要逗号，而最后一句condition说完后面不用逗号。>**

**当然，所有代码都在一个最大的{}里面，所以你可以看到很多任务的源文件里面最后的代码后面一堆}}]]之类的东西。那是因为对于它们对应的层级，这就是最后一句话。**

**5.2 常用命令详解**

每个命令都是一个包含 "type" 键的对象，"type" 决定了这个命令的功能，其他键则是该功能所需的参数。

**1. 传送与位移 (setPlayerPosition & setStage)**

* **setPlayerPosition**: 强制将玩家传送到指定坐标。

例如：

{

"type": "setPlayerPosition",

"x": -266.41,

"y": 32.77,

"z": -19.56,

"rx": 0.0, "ry": 0.0, "rz": 0.0, "rw": 1.0

}

* + "x", "y", "z": 目标位置坐标（通过 F9 获取）。
  + "rx", "ry", "rz", "rw": 四元数旋转值，决定了传送后的**面朝方向**。如果你不关心朝向，可以省略它们，或者直接照抄一个已知坐标的数值。
* **setStage**: 切换地图。常用于跨地图传送。

例如：

{

"type": "setStage",

"stage": "Park"

}

* + 通常与 setPlayerPosition 配合使用，先切换地图，再设定新地图内的坐标。

例如：

"oncomplete": [

{ "type": "setStage", "stage": "StationFront" },

{ "type": "setPlayerPosition", "x": -232.42, "y": 0.29, "z": -11.59 }

]

**笔者注：rxryrz的用途---一个强制位移配合手铐配合方向，可以做成陷阱传送阵，强制位移到某个点然后拷上.**

**如例5**

**"checkpoints": [**

**{**

**"id": "start\_surprise",**

**"zone": "first",**

**"condition": {**

**"description": "这里有一个惊喜？试试看，等待一会儿...",**

**"duration": 3,**

**"condition": "[Action\_None]",**

**//站着什么都不做触发以下内容**

**"oncomplete": [**

**{**

**"type": "setPlayerPosition",**

**"x": -232.42,**

**"y": 0.29,**

**"z": -11.59,**

**"rx": 0,**

**"ry": 0.68956,**

**"rz": 0,**

**"rw": 0.72423**

**},**

**//传送**

**{**

**"type": "lockHandcuffs",**

**"handcuffstype": "TimerHandcuff",**

**"duration": 1,**

**"attachtoobject": true**

**},**

**//拷上，锁的时间是1s，attachobject是指锁在东西上。（疑似这个如果通过代码实现可以不需要作用目标就直接锁上了。）**

**{**

**"type": "dropItem",**

**"itemtype": "Coat",**

**"stage": "StationFront",**

**"x": -232.42,**

**"y": 0.29,**

**"z": -12.91**

**}**

**]**

**},**

**//爆衣（移除衣服并找个地方放置。请注意，这里必须要给衣服放一个地方，否则红色报错事件失效）**

**"nextcheckpoint": {**

**"selectortype": "SpecificId",**

**"id": "second\_point"**

**}**

**},**

**{**

**"id": "second\_point",**

**"zone": "second",**

**"condition": {**

**"condition": "Naked",**

**"oncomplete": [**

**{**

**"type": "unequipAllCosplay"**

**},**

**{**

**"type": "equipCosplay",**

**"parts": [**

**"m\_cosplay\_general\_head\_kuchikase"**

**]**

**}**

**],**

**//内衣给你拽掉（指移除所有衣物相关的装扮），戴个口塞**

**"description": "怎么样？小母狗，很惊喜吧",**

**"duration": 3**

**},**

**这个事件中，就出现了传送+手铐的组合。**

**2. 装备与换装 (equipCosplay, equipAdultToy, unequipAllCosplay)**

* **equipCosplay**: 穿戴指定的服装/饰品。

例如：

{

"type": "equipCosplay",

"parts": [

"m\_cosplay\_general\_head\_kuchikase",

"m\_cosplay\_general\_genital\_plug\_light"

]

}

* + "parts": 一个**字符串数组**，包含所有要穿戴物品的 ID。ID 可以通过 Ctrl+F9 查询。
* **unequipAllCosplay**: 脱掉所有服装和饰品，让玩家变为裸体（保留内衣）。
* **equipAdultToy**: 装备成人玩具。

例如：

{

"type": "equipAdultToy",

"parts": ["Vibrator", "TitRotor", "KuriRotor"]

}

同样是可以用oncomplete或者onviolatecomplete执行

**3. 物品操作 (dropItem)**

* **dropItem**: 强制玩家掉落指定的物品。

例如：

{

"type": "dropItem",

"itemtype": "Coat",

"stage": "StationFront",

"x": -232.42,

"y": 0.29,

"z": -12.91

}

* + "itemtype": 物品类型，可选值有 Coat (外套), HandcuffKey (手铐钥匙), VibeRemocon (振动器遥控器), DildoFloor, DildoWall。
  + **【黄金法则】**: **必须**提供物品掉落的 stage 和 x, y, z 坐标，否则游戏会报错，整个命令序列都会失败！

**4. 状态与效果控制 (setVibrator, lockHandcuffs)**

* **setVibrator / setPiston**: 设置振动器/活塞的强度。

例如：

{

"type": "setVibrator",

"level": "High"

}

* + "level": 可选值 Off, Low, Medium (仅活塞), High, Random。
* **lockHandcuffs**: 给玩家上锁。

例如：

{

"type": "lockHandcuffs",

"handcuffstype": "TimerHandcuff",

"duration": 60,

"attachtoobject": true

}

* + "handcuffstype": 手铐类型，如 Handcuff (普通), KeyHandcuff (钥匙), TimerHandcuff (计时)。
  + "duration": 配合 TimerHandcuff 使用，设置锁定时间（秒）。
  + "attachtoobject": 设为 true 可以实现将玩家的手铐锁在某个物体上（例如墙壁），即使代码中没有明确的物体目标。

**5.3 实战：制作一个“陷阱传送阵”**

让我们结合所学，制作一个经典的任务桥段：玩家踏入一个看似无害的光圈，结果被传送到一个密室，并被缴械和束缚。

**任务目标**:

1. 在市中心设置一个小的触发区域。
2. 玩家进入并站立 3 秒后触发陷阱。
3. 触发后，将玩家传送到一个偏僻的坐标。
4. 传送后，强制玩家面朝特定方向。
5. 强制给玩家戴上60秒的计时手铐。
6. 强制玩家脱掉外套并掉落在地上。

**JSON 代码实现**:

例如：

// --- 在 "zones" 中定义陷阱区域 ---

{

"id": "trap\_zone",

"areas": [{"type": "sphere", "stage": "StationFront", "x": -227.0, "y": 0.17, "z": -62.82, "r": 0.5}]

},

// --- 在 "checkpoints" 中定义陷阱事件 ---

{

"id": "trap\_entry",

"zone": "trap\_zone",

"condition": {

"description": "这里似乎有什么奇怪的东西... 靠近看看？",

"duration": 3,

"condition": "Action\_None",

"oncomplete": [

{

"type": "setPlayerPosition",

"x": -232.42, "y": 0.29, "z": -11.59,

"rx": 0, "ry": 0.68956, "rz": 0, "rw": 0.72423

},

**//这里的所有坐标代码均ai生成，未尝试，谨慎使用，你可以在游戏里自己用F9获取坐标后替换测试。**

{

"type": "lockHandcuffs",

"handcuffstype": "TimerHandcuff",

"duration": 60,

"attachtoobject": true

},

{

"type": "dropItem",

"itemtype": "Coat",

"stage": "StationFront",

"x": -232.42, "y": 0.29, "z": -12.91

},

{

"type": "unequipAllCosplay"

},

{

"type": "equipCosplay",

"parts": ["m\_cosplay\_general\_head\_kuchikase"]

}

]

},

"nextcheckpoint": { "selectortype": "SpecificId", "id": "after\_trap\_message" }

}

在这个例子中，当玩家在 trap\_zone 站定3秒后，oncomplete 中的5个命令会依次执行：传送并转向 -> 上手铐 -> 掉落外套 -> 脱光所有衣服 -> 戴上口塞。一个完整的陷阱事件就此完成。

**第五章总结**

你现在已经拥有了作为任务“导演”的全部能力！你可以随心所欲地在关键时刻改变游戏世界和玩家状态，创造出充满惊喜、惩罚和戏剧性效果的桥段。你学会了：

* oncomplete 和 onviolatecondition 的触发时机和用途。
* 掌握了传送、换装、物品操作、状态控制等一系列强大的强制命令。
* 通过组合这些命令，能够制作出像“陷阱传送阵”这样复杂的多步骤自动化事件。

**在下一章，我们将探讨任务制作的终极话题：流程控制。你将学习如何打破线性的任务结构，通过 setCheckpoint、travelcondition 和随机跳转，创造出带有惩罚分支、循环任务和随机元素的高级动态任务。**

## 第六章：流程控制 - 构建动态的任务结构—非线性结构的诞生

到目前为止，我们主要依赖 nextcheckpoint 来实现从一个步骤到下一个步骤的线性流程。这一章，我们将学习三种打破这种线性结构的高级技巧：setCheckpoint（强制跳转）、travelcondition（旅途监督）和 RandomId（随机分支）。

**6.1 强制跳转 (setCheckpoint)**

setCheckpoint 是一个可以放在 oncomplete 或 onviolatecondition 数组中的**命令**。它就像一个“传送门”，可以无视 nextcheckpoint 的设定，立即将任务流程强行跳转到任何一个你指定的 Checkpoint。这是实现惩罚分支和非线性叙事的关键工具。

**基本语法**:

例如：

{

"type": "setCheckpoint",

"selectortype": "SpecificId",

"id": "target\_checkpoint\_id"

}

**实战：创建一个惩罚分支**

**任务目标**：要求玩家在 A 点保持裸体状态 10 秒。如果玩家中途穿上衣服（违背了 Naked 条件），则立即将他们传送到一个“惩罚房间”(punishment\_room\_checkpoint)，而不是继续正常流程。

例如：

{

"id": "stay\_naked\_task",

"zone": "zone\_A",

"condition": {

"description": "请在这里保持裸体10秒。",

"duration": 10,

"condition": "Naked",

"onviolatecondition": [

{

"type": "setCheckpoint",

"selectortype": "SpecificId",

"id": "punishment\_entry"

}

]

},

"nextcheckpoint": { "selectortype": "SpecificId", "id": "task\_success" }

},

{

"id": "punishment\_entry",

"zone": "punishment\_zone",

"condition": {

"description": "不听话的孩子需要接受惩罚... 在这里戴上计时手铐1分钟。",

// ... 此处省略惩罚的具体逻辑

},

// 惩罚结束后，可以再跳转回原来的任务，或者直接结束

"nextcheckpoint": { "selectortype": "SpecificId", "id": "stay\_naked\_task" }

},

{

"id": "task\_success",

// ... 正常完成任务后的流程

}

在这个例子中，stay\_naked\_task 有两条出路：

1. **成功**: 玩家保持裸体 10 秒，oncomplete 未触发，流程按照 nextcheckpoint 正常走向 task\_success。
2. **失败**: 玩家中途穿上衣服，Naked 条件被违背，onviolatecondition 立即触发，setCheckpoint 命令将流程强行扭转至 punishment\_entry，从而进入惩罚分支。

**6.2 旅途监督 (travelcondition)**

travelcondition 是一个非常精妙的机制。它**不**在玩家**到达**某个区域时检测，而是在玩家**从上一个检查点前往这一个检查点的途中**持续检测。如果玩家在路上违反了 travelcondition 设定的条件，就会立即触发惩罚，通常也是通过 setCheckpoint 实现跳转。

它与 Checkpoint 内的 condition 是平级的。

**实战：护送任务**

**任务目标**：玩家从 A 点出发，需要前往 B 点。在从 A 到 B 的整个过程中，玩家必须全程保持振动器处于随机档。一旦中途关闭或切换模式，任务失败并跳转到惩罚。

例如：

// --- 假设玩家已在 A 点完成了上一个检查点，并正前往 B 点 ---

{

"id": "goto\_point\_B",

"zone": "zone\_B",

// 这个 travelcondition 会在玩家离开 A 点后立刻开始生效

"travelcondition": {

"condition": "VibrationRandom",

"onviolatecondition": [

{

"type": "setCheckpoint",

"selectortype": "SpecificId",

"id": "punishment\_for\_travel\_fail"

}

]

},

// 这是玩家成功到达 B 点后，才会被检测的 condition

"condition": {

"description": "很好，你成功到达了。奖励你更刺激的活塞运动！",

"oncomplete": [

{ "type": "setPiston", "level": "High" }

]

},

"nextcheckpoint": { "selectortype": "SpecificId", "id": "next\_main\_task" }

}

**travelcondition 与 condition 的区别总结**：

* condition: 描述了**在目标地点需要做什么**。
* travelcondition: 描述了**在前往目标地点的路上需要保持什么状态**。

这使得你可以设计出“不能跑动”、“不能被发现”、“必须保持特定姿势移动”等更具挑战性的任务环节。

**6.3 随机分支 (RandomId)**

想让你的任务每次玩都有点不一样吗？可以在 nextcheckpoint 中使用 "RandomId" 来实现。

**语法**:

例如：

"nextcheckpoint": {

"selectortype": "RandomId",

"ids": [

"checkpoint\_A",

"checkpoint\_B",

"checkpoint\_C",

"checkpoint\_A"

]

}

* "selectortype": 设为 "RandomId"。
* "ids": 一个包含多个检查点 ID 的数组。当流程走到这里时，游戏会从这个数组中**随机挑选一个** ID 作为下一个检查点。

**实战：随机任务生成器**

**任务目标**：玩家完成一个前置任务后，系统会从三个不同的后续任务中随机分配一个。

例如：

{

"id": "mission\_hub",

"zone": "hub\_zone",

"condition": {

"description": "准备好接受你的随机任务了吗？",

"duration": 3

},

"nextcheckpoint": {

"selectortype": "RandomId",

"ids": [

"random\_task\_1", // 任务1的起始点

"random\_task\_2", // 任务2的起始点

"random\_task\_3" // 任务3的起始点

]

}

}

**小技巧：控制概率**  
如果你想让某个任务出现的概率更高，只需在 ids 数组中多次放入它的 ID 即可。例如：  
"ids": ["common\_task", "common\_task", "rare\_task"]  
在这个设置中，common\_task 被选中的概率是 rare\_task 的两倍 (2/3 vs 1/3)。

**实战：循环等待与随机事件**  
你可以结合 RandomId 和 setCheckpoint 制作一个“等待”状态，每隔一段时间就有几率触发一个随机事件。

例如：

{

"id": "wait\_loop",

"zone": "anywhere\_zone", // 一个覆盖全图的 zone

"condition": { "hidepanel": true, "duration": 10 }, // 安静地等待10秒

"nextcheckpoint": {

"selectortype": "RandomId",

"ids": [

"random\_event\_A", // 25% 概率触发事件A

"wait\_loop", // 75% 概率继续等待

"wait\_loop",

"wait\_loop"

]

}

}

这个结构创造了一个循环：大部分时间它会跳转回自身，实现持续等待，但有 25% 的几率会跳出循环，去执行 random\_event\_A。

**第六章总结**

恭喜你，你已经掌握了任务制作的最高阶技巧！通过灵活运用流程控制工具，你现在可以：

* 使用 setCheckpoint 创建带有失败惩罚的复杂分支，增加任务的挑战性。
* 利用 travelcondition 设计护送、潜行等对过程有要求的移动任务。
* 通过 RandomId 让任务充满不确定性，提高重复可玩性。

结合之前章节学到的所有知识，你已经完全有能力创作出结构复杂、互动性强、体验丰富的顶级自定义任务了。

**最后一章，我们将简单讨论一下如何排查错误，以及一些让你的任务体验更好的收尾工作和建议。**

好的，教程即将进入尾声。你已经学会了所有核心的制作技术，从搭建舞台到编写剧本，再到导演复杂的非线性流程。现在，让我们来谈谈最后也是同样重要的一步：如何打磨你的作品，修复那些恼人的错误，并最终发布一个让玩家满意的任务。

## 第七章：调试、润色与发布

一个伟大的作品不仅需要巧妙的设计，还需要精心的打磨。在这一章，我们不学习新的代码，而是学习一套“工匠”的方法论，确保你的任务不仅能运行，而且能运行得很好。

**7.1 错误排查（Debugging）：我的任务为什么不动了？**

当你按下 Alt+F9 重新加载任务，却发现任务没有出现，或者玩到一半卡住时，不要慌张。99% 的问题都源于一些常见的、微小的错误。以下是一个系统性的排查清单：

**第一步：检查 JSON 语法**

这是最常见也是最基础的错误来源。

1. **括号匹配**：确保每一个 { 都有一个对应的 }，每一个 [ 都有一个对应的 ]。现代文本编辑器（如 VS Code）通常有括号高亮匹配功能，可以帮你快速检查。
2. **逗号问题**：
   * 检查数组或对象中，并列的元素之间是否都用 , 分隔了。
   * 检查数组或对象的**最后一个元素后面，是否多加了逗号**。这是新手最常犯的错误。
3. **引号闭合**：确保所有的“键”和“字符串值”都用英文双引号 " 包围，并且成对出现。
4. **禁止注释**：再次确认你的 .json 文件中没有任何 // 或 /\* \*/ 形式的注释。

**工具推荐**：如果你不确定语法是否有错，可以使用在线的 “JSON Validator” 或 “JSON Linter” 工具。将你的代码粘贴进去，它会自动帮你找出语法错误的位置。

**第二步：逻辑流程检查**

如果语法没问题，但任务卡在某一步不动了，那很可能是逻辑出了问题。

1. **ID 拼写错误**：
   * 检查 Checkpoint 的 zone 值，是否与 "zones" 数组中定义的某个 id 完全一致（大小写敏感！）。
   * 检查 nextcheckpoint 或 setCheckpoint 中的 id，是否与目标 Checkpoint 的 id 完全一致。一个字母的差异就会导致流程中断。
2. **无法达成的条件**：
   * 你设置的 condition 字符串是否过于苛刻，或者存在逻辑矛盾？例如："[Naked, FrontClosed]" (既要裸体又要前部关闭)，这是永远无法满足的。
   * 检查 stage (地图) 名称是否正确。如果你在 Park 地图上设置了一个 zone，但 Checkpoint 却要求玩家在 StationFront 完成，那这个检查点永远不会被触发。
3. **强制命令失败**：
   * 最常见的是 dropItem 命令。**你是否忘记为它指定 stage 和 x, y, z 坐标？** 缺少坐标会导致该命令失败，并且 oncomplete 数组中**后续的所有命令都不会被执行**，这可能会导致你的任务流程卡住。

**第三步：利用游戏内信息**

* **观察任务面板**：游戏右侧的任务面板是你最好的朋友。它会显示 description，告诉你当前任务进行到了哪一步。如果面板上显示的还是上一步的提示，说明你当前这一步的触发条件（进入 zone 或满足 condition）没有达成。
* **F9 控制台**：密切关注控制台。虽然它不一定会直接告诉你“你的 JSON 第 X 行错了”，但它会提供你当前的状态信息。比如，你以为自己处于 Crouching 状态，但控制台显示你的动作是 Action\_SomeOtherThing，你就知道为什么条件没有满足了。

**7.2 任务润色：从“能玩”到“好玩”**

当你的任务逻辑全部跑通后，花点时间从玩家的角度来优化体验。

1. **清晰的描述 (description)**：
   * 你的任务指示是否清晰明确？避免使用模糊的词语。告诉玩家“去公园”，不如告诉他们“去公园的中央喷泉”。
   * 如果任务有隐藏元素（例如 outlinehidden: true 的区域），务必在描述中给出足够清晰的线索，否则玩家会因为找不到目标而感到沮丧。
2. **合理的节奏 (duration)**：
   * duration 的设置是否合理？要求玩家保持一个高难度动作 30 秒可能会很无聊。对于简单的确认性步骤，"duration": 1 就足够了。
3. **视觉引导**：
   * 在需要玩家前往下一个目标点时，默认显示的光圈 (outlinehidden: false) 是一个很好的引导。在任务的关键节点，可以考虑暂时关闭 outlinehidden，让玩家明确知道下一步该去哪里。
4. **给予反馈**：
   * 当玩家完成一个复杂步骤后，可以增加一个没有实际逻辑、只有 description 的中间检查点，例如：“干得漂亮！你成功了。现在准备迎接下一个挑战。” 这种及时的文本反馈能极大地提升玩家的成就感。
   * 善用 rp 奖励，让玩家觉得自己的努力得到了回报。

**7.3 最终检查与发布**

在你准备将任务分享给其他人之前，请做最后一次完整的测试。

1. **从头到尾完整玩一遍**：模拟一个完全不了解你任务设计的玩家，看看整个流程是否顺畅，指示是否清晰。
2. **测试所有分支**：如果你的任务有成功/失败分支，或者随机事件，请想办法（例如暂时修改代码）把每个分支都测试一遍，确保它们都能正常工作并正确地回到主线或结束。
3. **整理文件**：删除所有用于测试的临时代码，确保最终发布的 .json 文件是干净、完整的。

**教程终章总结**

恭喜你完成了整个自定义任务制作教程的学习！

从一个空白的 .json 文件开始，你一步步学会了：

* **搭建舞台 (Zones)**：在世界中定义触发点。
* **编写剧本 (Checkpoints)**：设定任务目标与步骤。
* **设定谜题 (condition 字符串)**：用逻辑判断检测玩家状态。
* **执导场景 (强制命令)**：在关键时刻改变游戏进程。
* **编排流程 (流程控制)**：创造动态、非线性的任务结构。
* **打磨作品 (调试与润色)**：修复错误并优化玩家体验。

你现在所掌握的知识，已经足够你将脑海中任何有趣的、复杂的、甚至是疯狂的任务构想变为现实。真正的学习才刚刚开始，不断地实践、尝试新的组合、从其他优秀的任务中汲取灵感，你将成为一名出色的任务创作者。

**现在，去创造属于你自己的故事吧！**

**笔者最后的说明：**

**这是一种塞雷卡精神—那种露出知识的精神。**

**作者是自学，所以很可能有错误。**

**欢迎各位大佬指正。**

**当然你要问我怎么做，当然是偷写好的代码喂给ai然后输入自己的需求啊。**

**这个教程的本质是给你看懂ai那个代码报错的时候你应该怎么办，至少不能完全看不懂。**

**Ai很他妈蠢。**