









正在收集工作区信息# 攀爬系统架构分析

## 整体架构

这个攀爬系统采用了状态模式和能力模式的组合架构，具有清晰的层次结构。以下是系统的主要组成部分：

## 1. 核心控制层

### 玩家控制器

- \*\*CSPlayerController\*\*: 继承自MonoBehaviour，负责输入处理和相机控制

- 接收输入并更新角色动作状态(Jump、Walk、Roll等)

- 支持新旧两套输入系统(`ENABLE\_INPUT\_SYSTEM`和Legacy Input)

- 处理相机旋转和角色朝向

### 能力调度器

- \*\*AbilityScheduler\*\*: 能力系统的调度中心

- 维护角色所有能力（`CharAbilities`数组）

- 处理能力的优先级和执行顺序

- 提供事件通知机制（`OnAbilityStarted`、`OnAbilityStopped`等）

## 2. 能力系统层

### 抽象能力基类

- \*\*AbstractAbility\*\*: 所有能力的基类

- 定义能力生命周期：`ReadyToRun()`、`OnStartAbility()`、`UpdateAbility()`等

- 与动画系统和输入系统的接口

### 具体能力类

- \*\*ClimbAbility\*\*: 攀爬能力的实现

- 检测可攀爬表面

- 管理攀爬状态机

- \*\*DropToClimb\*\*: 下降到可攀爬表面的能力

## 3. 攀爬状态机层

### 状态上下文

- \*\*ClimbStateContext\*\*: 攀爬状态机的上下文和管理器

- 包含所有攀爬状态的实例

- 提供状态转换机制

- 存储共享数据（动画器、碰撞信息、参照点等）

### 状态基类

- \*\*ClimbStateBase\*\*: 所有攀爬状态的抽象基类

- 定义状态接口：`EnterState()`、`ExitState()`

- 定义可能的状态转换：`Idle()`、`ClimbUp()`、`Jump()`等

### 具体状态类

- \*\*ClimbIdleState\*\*: 攀爬静止状态

- \*\*ClimbUpState\*\*: 向上攀爬状态

- \*\*ClimbJumpState\*\*: 攀爬跳跃状态

- \*\*ClimbDropState\*\*: 攀爬下降状态

- \*\*CornerOutState\*\*: 外角转向状态

- \*\*CornerInState\*\*: 内角转向状态

- \*\*ClimbShimmyState\*\*: 攀爬横向移动状态

## 4. 组件系统层

### 移动组件

- \*\*IMover\*\* 接口: 定义移动功能

- \*\*Mover\*\*: 基于CharacterController的实现

- \*\*RigidbodyMover\*\*: 基于Rigidbody的实现

- \*\*ICapsule\*\* 接口: 定义胶囊体操作

### 动画控制

- 利用Animator系统控制角色动画

- 使用动画事件和状态机同步游戏逻辑

### IK系统

- \*\*ClimbIK\*\*: 处理攀爬时的手脚位置

- 创建手部IK目标点

- 根据攀爬表面动态调整手脚位置

- 平滑过渡不同攀爬状态

### 调试工具

- \*\*CastDebug\*\*: 提供可视化调试工具

- 绘制射线、球体和胶囊体

- 显示文本标签和辅助信息

## 5. 项目设置工具

### 编辑器工具

- \*\*SetupProjectSettings\*\*: 编辑器扩展，用于配置项目设置

- 设置输入绑定

- 配置层碰撞矩阵

- 添加必要的标签

## 数据流向

1. \*\*输入处理\*\*：CSPlayerController → AbilityScheduler → CharacterActions

2. \*\*能力执行\*\*：AbilityScheduler → ClimbAbility → ClimbStateContext

3. \*\*状态转换\*\*：ClimbStateContext.SetState() → CurrentClimbState.ExitState() → newState.EnterState()

4. \*\*动画控制\*\*：ClimbStateBase → Animator → OnAnimatorMove/IK → Mover

5. \*\*物理交互\*\*：Mover → Physics.Raycast/SphereCast → 环境交互

## 扩展性设计

- 采用接口和抽象类实现松耦合

- 状态模式使攀爬行为可单独修改

- 通过能力系统可轻松添加新功能

- 事件系统允许观察者监听状态变化

这种架构设计让攀爬系统具有很高的灵活性和可扩展性，同时保持了代码的可维护性。