Hook 系统设计文档

一、模块介绍

Hook 是 StepFlow 工作流引擎中的扩展点系统。它提供了对事件组件、数据存储、上流通知等模块的触发点。

二、触发点列表

分类	触发时机	Hook 方法
工作流开始	引擎 run 时	`on_workflow_start(run_id)`
节点进入	Task 进入执行	`on_node_enter(run_id, state_id, input_data)`
节点成功	Task 执行完成	`on_node_success(run_id, state_id, output_data)`
节点失败	Task 执行报错	`on_node_fail(run_id, state_id, error)`
工作流结束	执行完成或失败	`on_workflow_end(run_id, result)`
控制信号	发送 Cancel/Terminate	`on_control_signal(run_id, signal_type, reason)`

三、Hook 接口要求

- 全部方法都应支持 `async def`
- 需要是完全异步和并可缓慢错误不影响主体逻辑
- 支持自定义 Hook 插件类型

四、Composite Hook 模式

支持以类似于 Observer 模式的多 Hook 分发:

```
hook = CompositeHook([PrintHook(), EventBusHook(), DBHook()])
```

五、实现类型

1. `PrintHook`

输出行为日志

2. `EventBusHook`

将事件扔入到 EventBus:

- 输出给 WebSocket
- 通知 UI 应用

3. `DBHook`

将事件表单按类型写入到相关表:

- `workflow_events`
- `workflow_executions`
- `activity_tasks`

六、扩展 Hook

支持在 pytest/生产环境下动态指定 Hook 类型

七、接口协议

- 全部 Hook 接口需要支持:
    - 异步执行
    - 已执行日志/存储/通知等
    - 可以是纯类型培系

八、设计原则

- **分层分责**: Engine 不直接知道 EventBus/数据库
- **引擎简单化**: 不需要为各种事件操作缓慢
- **应用分布式设计**: Hook 不得影响正常执行

九、系统集成规则

- Engine 通过 Hook 写入 DB/EventBus
- Worker/前端通过 subscribe 监听事件并对应

十、应用场景

- 日志导出
- Web UI 追踪
- 执行相关性进程/配置模块规范化

十一、测试策略

- 单元测试: 验证 Hook 写 DB/日志/扔 Event 行为
- 联合测试: Engine + Hook + UI 全链进程
- 高并发测试: Hook 在 1000+ 次进程下表现

十二、未条件扩展

- Hook 打包成 plugin 模型支持 DI
- 支持 WebHook / Kafka / Redis 扔事件
- 动态设置 Hook 扩展配置

若需添加具体示例和源码描述可再扩展依赖模块涉及设计规则。