alphaflow-nodes 开发任务清单

> 本文是针对 `alphaflow-nodes/src/` 目录下各个文件夹和文件的**整体开发任务**的**详细清单**。每个文件目前是空的，需要按此 checklist 分步实现功能。

总体说明

- 该 `alphaflow-nodes` crate 将存放所有"节点"相关逻辑，包括：
  1. **核心 Trait & Struct**：`node_type.rs`（定义 `NodeType`, `NodeExecutionContext`, `NodeError` 等）
  2. **节点注册**：`registry.rs`（维护 `NodeRegistry`）
  3. **公共工具**：`shared/`（比如 HTTP、AI 工具函数，验证逻辑等）
  4. **各节点文件夹**（`http/`, `openai/`, `file_upload/`, `slack/` 等），每个节点至少包含：
     - `mod.rs`：节点注册函数
     - `xxx_params.rs`：参数 struct + validate
     - `xxx_handler.rs`：实现 `NodeType`

按照下文的勾选项，逐步完成对应的开发。

---

1. node_type.rs

> 用于定义**节点基础接口**和必要的上下文/错误类型

- [ ] **定义 `NodeError`**
  - 使用 `thiserror` crate（或手动实现），包含 `InvalidConfig(String)` / `ExecutionFailed(String)` 等
  - 例如：

```rust
#[derive(thiserror::Error, Debug)]
pub enum NodeError {
  #[error("Invalid config: {0}")]
  InvalidConfig(String),
  #[error("Execution failed: {0}")]
  ExecutionFailed(String),
  // ...
}
```

- [ ] **定义 `NodeExecutionContext`**
  - 包含 `parameters: serde_json::Value`、`input_data: serde_json::Value`(可选)、以及其他可能的引用
  - 如：

```rust
pub struct NodeExecutionContext {
  pub parameters: Value,
```

```
    pub input_data: Value,
    // ...
}
```

- [ ] **定义 `NodeOutput`**
  - 用于节点执行成功后的产物：

```
pub struct NodeOutput {
    pub data: Value,
    // maybe metadata, status, etc.
}
```

- [ ] **定义 `NodeType` trait**
  - 提供 `fn name(&self) -> &str;`, `fn display_name(&self) -> &str;`
  - 提供 `async fn execute(...) -> Result<NodeOutput, NodeError>`
  - 可以在这里也加 `fn description(&self)->NodeDescription` 若要做 UI 集成

示例结构完成后，确保能 `cargo check` 通过。

---

2. registry.rs

> 用于统一管理并查找节点实例

- [ ] **定义 `NodeRegistry` struct**
  - 内含 `HashMap<String, Arc<dyn NodeType>>`
- [ ] **`register(...)` 方法**
  - 接受 `Arc<dyn NodeType>`，以 `node_type.name()` 作为 key 存入 map
- [ ] **`get(...)` 方法**
  - 通过节点名称字符串取到对应的 `Arc<dyn NodeType>`
- [ ] **(可选) 在 `#[cfg(test)]` 中写测试**
  - 伪造一个 “mock node” 来测试能否注册 + get

---

3. lib.rs

> 作为本 crate 的顶层入口

- [ ] **`pub mod node_type;`**
- [ ] **`pub mod registry;`**
- [ ] **`pub mod shared;`**
- [ ] **各节点文件夹**：`pub mod http; pub mod openai; ...`
- [ ] **编写 `register_all_nodes`**
  - 形如：

```rust
pub fn register_all_nodes(registry: &mut NodeRegistry) {
  http::register_node(registry);
  openai::register_node(registry);
  // ...
}
```

- 保证后续由 orchestrator 调用即可一次性加载所有节点

---

4. shared/

> 放置公共工具函数, 避免各节点重复逻辑

4.1 shared/mod.rs

- [ ] **`pub mod http_utils; pub mod ai_utils; pub mod validation; ...`**
- [ ] **(可选) `pub use xx`** 做 re-export

4.2 shared/http_utils.rs

- [ ] **定义通用 HTTP 请求函数**
  - 例如 `pub async fn http_request(client: &Client, method: &str, url: &str, body: Option<String>) -> Result<String, NodeError>`
  - 处理 `reqwest`、返回字符串或报错

4.3 shared/ai_utils.rs

- [ ] **定义 `call_openai_completion(...)`**
  - 作为 OpenAI 相关节点的基础函数
  - 传入 API key、model、prompt 等, 返回文本/JSON

4.4 shared/validation.rs

- [ ] **通用校验函数**
  - eg. `ensure_in_range(value, min, max) -> Result<(), NodeError>`
  - eg. `parse_params<T: Deserialize>(value: Value)->Result<T, NodeError>`
- [ ] **(可选) parse错误统一 => `NodeError::InvalidConfig`**

4.5 shared/error_utils.rs (可选)

- [ ] **若需要** `From<reqwest::Error>` => `NodeError`, etc.

---

5. http/

以 `http` 节点为例

5.1 http/mod.rs

- [ ] `pub mod http_handler; pub mod http_params;`
- [ ] `pub fn register_node(registry: &mut NodeRegistry)`
  - `registry.register(Arc::new(http_handler::HttpHandler::new()))`

5.2 `http_params.rs`

- [ ] **定义** `HttpParams` **struct +** `#[derive(Deserialize)]`
- [ ] `fn validate(&self)->Result<(),NodeError>`
  - 检查 url、method 等合规

5.3 `http_handler.rs`

- [ ] `pub struct HttpHandler;`
- [ ] `impl NodeType for HttpHandler`:
  - `fn name(&self)-> &str { "http" }`
  - `async fn execute(...) { parse HttpParams -> validate -> call shared::http_utils::http_request(...)? }`

---

6. openai/

- [ ] `mod.rs`:
  - `pub mod openai_handler; pub mod openai_params;`
  - `pub fn register_node(registry: &mut NodeRegistry) {...}`
- [ ] `openai_params.rs`:
  - `#[derive(Deserialize)] struct OpenAiParams { prompt, model, temperature... }`
  - `fn validate(&self) -> Result<(), NodeError>`
- [ ] `openai_handler.rs`:
  - `pub struct OpenAiHandler { ... }`
  - `impl NodeType` => parse params, validate, call `shared::ai_utils::call_openai_completion(...)`

---

7. file_upload/, slack/ 等其它节点

> 同理，先留空或只写 `mod.rs` + `register_node`, 具体逻辑可后续补充

- [ ] file_upload
  - `file_upload_params.rs`, `file_upload_handler.rs`
- [ ] slack
  - `slack_params.rs`, `slack_handler.rs`
- [ ] ... (更多)

---

8. 最终测试 & Demo

1. **(可选)在本 crate** 做简单集成测试:
   - `#[cfg(test)] mod tests { ... }`

- `let mut registry = NodeRegistry::new(); alphaflow_nodes::register_all_nodes(&mut registry); let node = registry.get("http").unwrap(); ...`

2. **(可选) 在 engine/app**
   - `engine` crate 中使用 `alphaflow_nodes::register_all_nodes(...)`，对 `"http"`, `"openai"` 等节点做实际运行或工作流demo

Checklist 一览

```
- [ ] **node_type.rs**
  - [ ] 定义 `NodeType`, `NodeExecutionContext`, `NodeOutput`,
`NodeError`
- [ ] **registry.rs**
  - [ ] `NodeRegistry` with `register/get`
- [ ] **lib.rs**
  - [ ] 导出 mod & `register_all_nodes`
- [ ] **shared/**
  - [ ] `mod.rs` (导出http_utils, ai_utils, etc.)
  - [ ] `http_utils.rs`
  - [ ] `ai_utils.rs`
  - [ ] `validation.rs`
  - [ ] `error_utils.rs` (可选)
- [ ] **http/**
  - [ ] `mod.rs` (register_node)
  - [ ] `http_params.rs` (Deserialize + validate)
  - [ ] `http_handler.rs` (impl NodeType)
- [ ] **openai/**
  - [ ] `mod.rs` (register_node)
  - [ ] `openai_params.rs` (Deserialize + validate)
  - [ ] `openai_handler.rs` (impl NodeType)
- [ ] **file_upload/** & **slack/**
  - [ ] 空文件夹，后续同样结构
```