n8n-Style Workflow Implementation Checklist (Rust)

本清单列出了实现 n8n 风格的 Workflow 所需的各个模块和功能，建议按顺序逐个实现、测试、并集成。每个模块的目标、关键功能和实现要点均已标出。

---

1. 基础环境配置

- [ ] **设置 Rust 项目**
  - 初始化 Cargo 项目（`cargo new`）
  - 配置 `Cargo.toml`，添加依赖：
    - `serde` 与 `serde_json`（用于 JSON 序列化与反序列化）
    - 其他工具库（如 `lazy_static`、`tokio`、`warp` 等，根据需要）

---

2. 全局常量（Constants）

- [ ] 定义全局常量：
  - `MANUAL_CHAT_TRIGGER_LANGCHAIN_NODE_TYPE`
  - `NODES_WITH_RENAMABLE_CONTENT`
  - `STARTING_NODE_TYPES`
- [ ] 根据业务需求设置具体值，确保后续各模块引用一致。

---

3. 错误处理（ApplicationError）

- [ ] 实现自定义错误类型 `ApplicationError`
  - 实现 `std::error::Error` trait
  - 支持附加标签和额外数据（例如 context 信息）

---

4. 全局状态管理（GlobalState）

- [ ] 实现一个全局状态模块
  - 提供 `getGlobalState()` 方法
  - 返回默认设置，如默认时区、全局配置参数等

---

5. 数据接口与数据结构（Interfaces）

- [ ] 定义所有工作流相关的数据结构：
  - **节点**：`INode`、`INodes`
  - **连接**：`IConnections`、`IConnection`、`INodeConnection`
  - **节点执行数据**：`INodeExecutionData`
  - **节点参数**：`INodeParameters`、`NodeParameterValueType`
  - **节点类型**：`INodeType`、`INodeTypes`
  - **静态数据**：`IDataObject`、`IObservableObject`

    - **其他**：`IPinData`、`IWorkflowSettings`、`IConnectedNode`、`INodeOutputConfiguration`
- [ ] 根据需要选择 Rust 结构体（或 trait）来实现这些接口

---

6. 节点工具类（NodeHelpers）

- [ ] 实现辅助函数，用于：
    - 获取节点默认参数：`getNodeParameters(...)`
    - 获取节点输出数据：`getNodeOutputs(...)`
    - 其它节点数据格式转换、校验和处理

---

7. 可观察对象（ObservableObject）

- [ ] 实现一个轻量级的观察者包装器：
    - 提供类似 `ObservableObject.create(...)` 的方法
    - 自动设置标志（如 `__dataChanged`）以便后续判断数据是否发生变化

---

8. 表达式解析（Expression）

- [ ] 实现表达式解析器，用于：
    - 解析节点参数中的表达式
    - 求值与变量替换
    - 支持引用其他节点数据的表达式
- [ ] 使其能与 Workflow 上下文结合使用

---

9. Workflow 核心类

- [ ] 实现 Workflow 结构体，包含以下字段：
    - `id`, `name`
    - `nodes: INodes`（以节点名称为键存储节点数据）
    - `connectionsBySourceNode: IConnections`（直接传入的连接数据）
    - `connectionsByDestinationNode: IConnections`（通过 __getConnectionsByDestination 计算得到）
    - `nodeTypes: INodeTypes`
    - `expression: Expression`
    - `active: boolean`
    - `settings: IWorkflowSettings`
    - `timezone: string`
    - `staticData: IDataObject`
    - `testStaticData?: IDataObject`
    - `pinData?: IPinData`
- [ ] 在构造函数中完成：
    - 节点集合初始化（遍历参数中的 nodes，将默认值填入）
    - 连接关系转换（调用 `__getConnectionsByDestination`）

- 初始化静态数据（使用 ObservableObject）
- 获取全局设置（如时区）
- 创建 Expression 实例

---

10. Workflow 方法实现

10.1 静态数据操作

- [ ] `overrideStaticData(staticData?: IDataObject)`
- [ ] `getStaticData(type: string, node?: INode): IDataObject`

10.2 节点查询与管理

- [ ] `getTriggerNodes()`
- [ ] `getPollNodes()`
- [ ] `queryNodes(checkFunction: (nodeType: INodeType) => boolean): INode[]`
- [ ] `getNode(nodeName: string): INode | null`
- [ ] `getPinDataOfNode(nodeName: string): INodeExecutionData[] | undefined`

10.3 节点重命名相关

- [ ] `renameNodeInParameterValue(...)`：递归替换表达式中节点名称
- [ ] `renameNode(currentName: string, newName: string)`：更新节点、参数、连接关系中所有旧名称为新名称

10.4 节点遍历与依赖关系

- [ ] `getHighestNode(nodeName: string, nodeConnectionIndex?: number, checkedNodes?: string[]): string[]`
- [ ] `getChildNodes(nodeName: string, type?: NodeConnectionType | 'ALL' | 'ALL_NON_MAIN', depth?: number): string[]`
- [ ] `getParentNodes(nodeName: string, type?: NodeConnectionType | 'ALL' | 'ALL_NON_MAIN', depth?: number): string[]`
- [ ] `getConnectedNodes(...)`：遍历连接关系
- [ ] `searchNodesBFS(...)`：使用 BFS 遍历节点依赖关系
- [ ] `getParentMainInputNode(node: INode): INode`
- [ ] `getNodeConnectionIndexes(...)`

10.5 工作流起始节点确定

- [ ] `__getStartNode(nodeNames: string[]): INode | undefined`
- [ ] `getStartNode(destinationNode?: string): INode | undefined`

---

11. 测试

- [ ] 为每个模块编写单元测试（例如：节点重命名、连接关系转换、节点遍历）
- [ ] 为 Workflow 类整体功能编写集成测试（加载 JSON、修改节点、查询依赖关系等）

12. 文档与注释

- [ ] 为每个函数和模块编写详细的注释
- [ ] 撰写 README.md，说明如何构建、运行、测试各模块

总结

按照以上清单逐步实现基础模块和核心逻辑后，你将完成一个基于 Rust 的 n8n 风格 Workflow 系统。每个模块实现完毕后，记得编写测试用例验证正确性，再整合到 Workflow 类中，确保整体功能正常。

Happy coding!