StepFlow Gateway SQLite 数据库设计

概述

设计一个完整的 SQLite 数据库结构来支持：

1. OpenAPI 文档模板存储
2. 解析后的 API 文档存储
3. 前端展示支持
4. HTTP 执行支持
5. 工作流节点和调度任务引用

SQLite 特性适配

主要调整

- 使用 `TEXT` 替代 `UUID` 类型（存储为字符串）
- 使用 `TEXT` 替代 `JSONB` 类型（存储为 JSON 字符串）
- 使用 `TEXT` 替代 `INET` 类型（存储 IP 地址字符串）
- 使用 `INTEGER` 替代 `BIGINT` 类型
- 移除 PostgreSQL 特有的扩展和功能
- 简化索引和约束语法

数据库表设计

1. OpenAPI 模板表 (openapi_templates)

```sql
CREATE TABLE openapi_templates (
    id TEXT PRIMARY KEY, -- UUID 存储为字符串
    name TEXT NOT NULL,
    description TEXT,
    version TEXT NOT NULL,
    category TEXT,
    tags TEXT, -- JSON 数组字符串，如 '["petstore", "demo"]'
    template_content TEXT NOT NULL,
    content_type TEXT NOT NULL DEFAULT 'yaml',
    is_public INTEGER DEFAULT 0, -- 0=false, 1=true
    created_by TEXT,
    created_at TEXT NOT NULL, -- ISO 8601 格式
    updated_at TEXT NOT NULL,
    status TEXT DEFAULT 'active'
);

-- 索引
CREATE UNIQUE INDEX uk_template_name_version ON openapi_templates(name, version);
CREATE INDEX idx_template_category ON openapi_templates(category);
CREATE INDEX idx_template_status ON openapi_templates(status);
CREATE INDEX idx_template_created_at ON openapi_templates(created_at);
```

2. API 文档实例表 (api_documents)

```
CREATE TABLE api_documents (
    id TEXT PRIMARY KEY,
    template_id TEXT,
    name TEXT NOT NULL,
    description TEXT,
    version TEXT NOT NULL,
    base_url TEXT NOT NULL,
    server_url TEXT,
    parsed_spec TEXT NOT NULL, -- JSON 字符串
    original_content TEXT NOT NULL,
    content_type TEXT NOT NULL,
    auth_config TEXT, -- JSON 字符串
    rate_limit_config TEXT, -- JSON 字符串
    timeout_config TEXT, -- JSON 字符串
    status TEXT DEFAULT 'active',
    health_status TEXT DEFAULT 'unknown',
    last_health_check TEXT,
    total_endpoints INTEGER DEFAULT 0,
    active_endpoints INTEGER DEFAULT 0,
    created_by TEXT,
    created_at TEXT NOT NULL,
    updated_at TEXT NOT NULL,
    FOREIGN KEY (template_id) REFERENCES openapi_templates(id) ON DELETE
SET NULL
);

-- 索引
CREATE UNIQUE INDEX uk_api_name_version ON api_documents(name, version);
CREATE INDEX idx_api_template_id ON api_documents(template_id);
CREATE INDEX idx_api_status ON api_documents(status);
CREATE INDEX idx_api_health_status ON api_documents(health_status);
CREATE INDEX idx_api_created_at ON api_documents(created_at);
```

3. API 端点表 (api_endpoints)

```
CREATE TABLE api_endpoints (
    id TEXT PRIMARY KEY,
    api_document_id TEXT NOT NULL,
    path TEXT NOT NULL,
    method TEXT NOT NULL,
    operation_id TEXT,
    summary TEXT,
    description TEXT,
    tags TEXT, -- JSON 数组字符串
    parameters TEXT, -- JSON 字符串
    request_body_schema TEXT, -- JSON 字符串
    response_schemas TEXT, -- JSON 字符串
    auth_required INTEGER DEFAULT 0,
    rate_limit_enabled INTEGER DEFAULT 0,
    timeout_ms INTEGER DEFAULT 30000,
    status TEXT DEFAULT 'active',
    is_deprecated INTEGER DEFAULT 0,
    call_count INTEGER DEFAULT 0,
    success_count INTEGER DEFAULT 0,
    error_count INTEGER DEFAULT 0,
    avg_response_time_ms INTEGER,
    created_at TEXT NOT NULL,
    updated_at TEXT NOT NULL,
    FOREIGN KEY (api_document_id) REFERENCES api_documents(id) ON DELETE
CASCADE
);

-- 索引
CREATE UNIQUE INDEX uk_endpoint_path_method ON
api_endpoints(api_document_id, path, method);
CREATE INDEX idx_endpoint_api_document_id ON
api_endpoints(api_document_id);
CREATE INDEX idx_endpoint_method ON api_endpoints(method);
CREATE INDEX idx_endpoint_status ON api_endpoints(status);
CREATE INDEX idx_endpoint_operation_id ON api_endpoints(operation_id);
```

4. 资源引用表 (resource_references)

```
CREATE TABLE resource_references (
    id TEXT PRIMARY KEY,
    resource_type TEXT NOT NULL,
    resource_id TEXT NOT NULL,
    api_endpoint_id TEXT NOT NULL,
    reference_config TEXT, -- JSON 字符串
    display_name TEXT,
    description TEXT,
    status TEXT DEFAULT 'active',
    last_used_at TEXT,
    usage_count INTEGER DEFAULT 0,
    created_by TEXT,
    created_at TEXT NOT NULL,
    updated_at TEXT NOT NULL,
    FOREIGN KEY (api_endpoint_id) REFERENCES api_endpoints(id) ON DELETE
CASCADE
);

-- 索引
CREATE INDEX idx_ref_resource_type_id ON
```

```
resource_references(resource_type, resource_id);
CREATE INDEX idx_ref_api_endpoint_id ON
resource_references(api_endpoint_id);
CREATE INDEX idx_ref_status ON resource_references(status);
CREATE INDEX idx_ref_last_used ON resource_references(last_used_at);
```

5. API 调用日志表 (api_call_logs)

```
CREATE TABLE api_call_logs (
    id TEXT PRIMARY KEY,
    api_endpoint_id TEXT NOT NULL,
    resource_reference_id TEXT,
    request_method TEXT NOT NULL,
    request_url TEXT NOT NULL,
    request_headers TEXT, -- JSON 字符串
    request_body TEXT,
    request_params TEXT, -- JSON 字符串
    response_status_code INTEGER,
    response_headers TEXT, -- JSON 字符串
    response_body TEXT,
    response_time_ms INTEGER,
    request_size_bytes INTEGER,
    response_size_bytes INTEGER,
    error_message TEXT,
    error_type TEXT,
    client_ip TEXT,
    user_agent TEXT,
    created_at TEXT NOT NULL,
    FOREIGN KEY (api_endpoint_id) REFERENCES api_endpoints(id) ON DELETE
CASCADE,
    FOREIGN KEY (resource_reference_id) REFERENCES
resource_references(id) ON DELETE SET NULL
);

-- 索引
CREATE INDEX idx_log_api_endpoint_id ON api_call_logs(api_endpoint_id);
CREATE INDEX idx_log_resource_reference_id ON
api_call_logs(resource_reference_id);
CREATE INDEX idx_log_status_code ON api_call_logs(response_status_code);
CREATE INDEX idx_log_created_at ON api_call_logs(created_at);
CREATE INDEX idx_log_error_type ON api_call_logs(error_type);
```

6. API 健康检查表 (api_health_checks)

```
CREATE TABLE api_health_checks (
    id TEXT PRIMARY KEY,
    api_document_id TEXT NOT NULL,
    check_type TEXT NOT NULL,
    status TEXT NOT NULL,
    response_time_ms INTEGER,
    error_message TEXT,
    details TEXT, -- JSON 字符串
    checked_at TEXT NOT NULL,
    FOREIGN KEY (api_document_id) REFERENCES api_documents(id) ON DELETE
CASCADE
);

-- 索引
CREATE INDEX idx_health_api_document_id ON
api_health_checks(api_document_id);
CREATE INDEX idx_health_status ON api_health_checks(status);
CREATE INDEX idx_health_checked_at ON api_health_checks(checked_at);
```

视图设计

1. API 端点资源视图 (api_endpoint_resources)

```
CREATE VIEW api_endpoint_resources AS
SELECT
    e.id as endpoint_id,
    e.path,
    e.method,
    e.operation_id,
    e.summary,
    e.description,
    e.tags,
    d.id as api_document_id,
    d.name as api_name,
    d.version as api_version,
    d.base_url,
    e.status as endpoint_status,
    d.status as api_status,
    e.call_count,
    e.success_count,
    e.error_count,
    e.avg_response_time_ms,
    COUNT(r.id) as reference_count
FROM api_endpoints e
JOIN api_documents d ON e.api_document_id = d.id
LEFT JOIN resource_references r ON e.id = r.api_endpoint_id AND r.status
= 'active'
GROUP BY e.id, d.id;
```

2. 资源引用统计视图 (resource_reference_stats)

```
CREATE VIEW resource_reference_stats AS
SELECT
    r.resource_type,
    r.resource_id,
    COUNT(r.id) as total_references,
    SUM(CASE WHEN r.status = 'active' THEN 1 ELSE 0 END) as
active_references,
    SUM(CASE WHEN r.status = 'error' THEN 1 ELSE 0 END) as
error_references,
    SUM(r.usage_count) as total_usage,
    MAX(r.last_used_at) as last_used
FROM resource_references r
GROUP BY r.resource_type, r.resource_id;
```

约束和检查

检查约束（SQLite 3.38+ 支持）

```
-- 确保HTTP方法有效
CREATE TABLE api_endpoints (
    -- ... 其他字段 ...
    method TEXT NOT NULL CHECK (method IN ('GET', 'POST', 'PUT',
'DELETE', 'PATCH', 'HEAD', 'OPTIONS')),
    -- ... 其他字段 ...
);

-- 确保状态值有效
CREATE TABLE api_documents (
    -- ... 其他字段 ...
    status TEXT DEFAULT 'active' CHECK (status IN ('active', 'inactive',
'error')),
    -- ... 其他字段 ...
);

CREATE TABLE api_endpoints (
    -- ... 其他字段 ...
    status TEXT DEFAULT 'active' CHECK (status IN ('active', 'inactive',
'deprecated')),
    -- ... 其他字段 ...
);

CREATE TABLE resource_references (
    -- ... 其他字段 ...
    status TEXT DEFAULT 'active' CHECK (status IN ('active', 'inactive',
'error')),
    -- ... 其他字段 ...
);
```

触发器

更新时间戳触发器

```sql
-- 创建触发器函数（SQLite 使用触发器直接实现）
CREATE TRIGGER update_openapi_templates_updated_at
    BEFORE UPDATE ON openapi_templates
    FOR EACH ROW
BEGIN
    UPDATE openapi_templates SET updated_at = datetime('now') WHERE id =
NEW.id;
END;

CREATE TRIGGER update_api_documents_updated_at
    BEFORE UPDATE ON api_documents
    FOR EACH ROW
BEGIN
    UPDATE api_documents SET updated_at = datetime('now') WHERE id =
NEW.id;
END;

CREATE TRIGGER update_api_endpoints_updated_at
    BEFORE UPDATE ON api_endpoints
    FOR EACH ROW
BEGIN
    UPDATE api_endpoints SET updated_at = datetime('now') WHERE id =
NEW.id;
END;

CREATE TRIGGER update_resource_references_updated_at
    BEFORE UPDATE ON resource_references
    FOR EACH ROW
BEGIN
    UPDATE resource_references SET updated_at = datetime('now') WHERE id
= NEW.id;
END;
```

统计更新触发器

```sql
-- 当API调用日志插入时，更新端点的统计信息
CREATE TRIGGER update_endpoint_stats_trigger
    AFTER INSERT ON api_call_logs
    FOR EACH ROW
BEGIN
    UPDATE api_endpoints
    SET
        call_count = call_count + 1,
        success_count = CASE WHEN NEW.response_status_code BETWEEN 200
AND 299
```

```sql
                                THEN success_count + 1 ELSE success_count
END,
        error_count = CASE WHEN NEW.response_status_code >= 400
                          THEN error_count + 1 ELSE error_count END,
        avg_response_time_ms = CASE
            WHEN avg_response_time_ms IS NULL THEN NEW.response_time_ms
            ELSE (avg_response_time_ms + NEW.response_time_ms) / 2
        END
    WHERE id = NEW.api_endpoint_id;
END;

-- 端点计数更新触发器
CREATE TRIGGER update_document_endpoint_count_insert
    AFTER INSERT ON api_endpoints
    FOR EACH ROW
BEGIN
    UPDATE api_documents
    SET
        total_endpoints = total_endpoints + 1,
        active_endpoints = active_endpoints + CASE WHEN NEW.status =
'active' THEN 1 ELSE 0 END
    WHERE id = NEW.api_document_id;
END;

CREATE TRIGGER update_document_endpoint_count_update
    AFTER UPDATE ON api_endpoints
    FOR EACH ROW
BEGIN
    UPDATE api_documents
    SET
        active_endpoints = active_endpoints +
            CASE WHEN NEW.status = 'active' THEN 1 ELSE 0 END -
            CASE WHEN OLD.status = 'active' THEN 1 ELSE 0 END
    WHERE id = NEW.api_document_id;
END;

CREATE TRIGGER update_document_endpoint_count_delete
    AFTER DELETE ON api_endpoints
    FOR EACH ROW
BEGIN
    UPDATE api_documents
    SET
        total_endpoints = total_endpoints - 1,
        active_endpoints = active_endpoints - CASE WHEN OLD.status =
'active' THEN 1 ELSE 0 END
    WHERE id = OLD.api_document_id;
END;
```

使用场景

1. 前端展示

- 查询 `api_endpoint_resources` 视图获取所有可用的API端点
- 根据标签、状态等条件过滤
- 显示调用统计和健康状态

2. HTTP执行

- 从 `api_endpoints` 表获取端点配置
- 从 `api_documents` 表获取基础URL和认证配置
- 记录调用日志到 `api_call_logs` 表

3. 工作流节点引用

- 在 `resource_references` 表中创建引用记录
- `resource_type` 设为 'workflow_node'
- `resource_id` 为工作流节点ID
- 通过 `reference_config` 存储参数映射

4. 调度任务引用

- 在 `resource_references` 表中创建引用记录
- `resource_type` 设为 'scheduled_task'
- `resource_id` 为调度任务ID
- 通过 `reference_config` 存储调度配置

性能优化建议

1. 索引优化

```
-- 复合索引
CREATE INDEX idx_endpoints_doc_method_status ON
api_endpoints(api_document_id, method, status);
CREATE INDEX idx_refs_type_status_endpoint ON
resource_references(resource_type, status, api_endpoint_id);
CREATE INDEX idx_logs_endpoint_time ON api_call_logs(api_endpoint_id,
created_at DESC);
```

2. 查询优化

- 使用 `EXPLAIN QUERY PLAN` 分析查询性能
- 避免在 JSON 字段上进行复杂查询
- 考虑将频繁查询的 JSON 数据提取到独立字段

3. 数据清理

```
-- 定期清理旧的调用日志
DELETE FROM api_call_logs WHERE created_at < datetime('now', '-30
days');
```

```
-- 清理过期的健康检查记录
DELETE FROM api_health_checks WHERE checked_at < datetime('now', '-7
days');
```

扩展建议

1. 缓存表

```
CREATE TABLE api_endpoint_cache (
    endpoint_id TEXT PRIMARY KEY,
    cache_data TEXT, -- JSON 字符串
    cached_at TEXT NOT NULL,
    expires_at TEXT NOT NULL,
    FOREIGN KEY (endpoint_id) REFERENCES api_endpoints(id) ON DELETE
CASCADE
);
```

2. 审计表

```
CREATE TABLE api_audit_logs (
    id TEXT PRIMARY KEY,
    table_name TEXT NOT NULL,
    record_id TEXT NOT NULL,
    operation TEXT NOT NULL, -- 'INSERT', 'UPDATE', 'DELETE'
    old_data TEXT, -- JSON 字符串
    new_data TEXT, -- JSON 字符串
    changed_by TEXT,
    changed_at TEXT NOT NULL
);
```

这个 SQLite 数据库设计保持了完整的功能，同时适配了 SQLite 的语法和限制。