


# StepFlow Gateway 开发计划

---

## 项目概述

StepFlow Gateway 是一个动态 API 网关，作为 AI 驱动平台的核心组件，用于自动生成和部署简单的工具。

## 当前状态

-  OpenAPI 文档解析
-  DTO 生成
-  基础验证
-  API 注册和管理（部分完成）
-  动态代理功能
-  前端集成支持

## 开发阶段规划

第一阶段：核心功能完善（优先级：高）

### 1.1 API 注册和管理系统

目标：实现完整的 API 生命周期管理

需要实现的功能：

```
// 数据结构
struct ApiRegistry {
    apis: HashMap<String, RegisteredApi>,
}

struct RegisteredApi {
    id: String,
    name: String,
    version: String,
    spec: OpenApi30Spec,
    base_url: String,
    created_at: DateTime<Utc>,
    updated_at: DateTime<Utc>,
}

// API 端点
POST /v1/apis/register
{
    "name": "string",
    "version": "string",
    "spec": "OpenAPI spec content",
    "base_url": "string"
}
```

```

GET /v1/apis
Response: {
  "apis": [
    {
      "id": "string",
      "name": "string",
      "version": "string",
      "base_url": "string",
      "endpoints_count": "number",
      "created_at": "datetime"
    }
  ]
}

GET /v1/apis/{api_id}
DELETE /v1/apis/{api_id}

```

#### 实施步骤：

1. 设计数据模型和存储结构
2. 实现 API 注册逻辑
3. 添加 API 列表和详情端点
4. 实现 API 删除功能
5. 添加数据持久化

## 1.2 动态代理核心

**目标：**实现请求的动态路由和转发

#### 核心逻辑：

```

// 代理处理器
async fn proxy_request(
  Path(api_id): Path<String>,
  Path(path): Path<String>,
  method: Method,
  headers: HeaderMap,
  body: Bytes,
  Query(query): Query<HashMap<String, String>>,
  State(state): State<AppState>,
) -> Result<Response, ServiceError> {
  // 1. 查找注册的 API
  let api = state.api_registry.get(&api_id)?;

  // 2. 匹配路径和操作
  let operation = match_operation(&api.spec, &method, &path)?;

  // 3. 验证参数
  validate_parameters(&operation, &query, &body)?;

```

```

// 4. 构建目标请求
let target_url = build_target_url(&api.base_url, &path, &query);

// 5. 转发请求
let response = forward_request(&target_url, method, headers,
body).await?;

// 6. 验证响应
validate_response(&operation, &response)?;

Ok(response)
}

```

#### 实施步骤：

1. 实现路径匹配算法
2. 添加参数验证逻辑
3. 实现请求转发功能
4. 添加响应验证
5. 集成错误处理

#### 第二阶段：验证和错误处理（优先级：中）

##### 2.1 参数验证系统

#### 功能：

- 路径参数验证
- 查询参数验证
- 请求体验证
- 响应验证

#### 实现要点：

```

// 验证器
trait ParameterValidator {
    fn validate_path_params(&self, params: &HashMap<String, String>) ->
Result<(), ValidationError>;
    fn validate_query_params(&self, params: &HashMap<String, String>) ->
Result<(), ValidationError>;
    fn validate_request_body(&self, body: &[u8]) -> Result<(),
ValidationError>;
    fn validate_response(&self, response: &Response) -> Result<(),
ValidationError>;
}

```

##### 2.2 统一错误处理

错误类型：

```
#[derive(Debug, Serialize)]
pub enum ServiceError {
    ApiNotFound(String),
    InvalidPath(String),
    ValidationError(String),
    ProxyError(String),
    InternalError(String),
}
```

第三阶段：前端集成（优先级：中）

### 3.1 动态表单生成

端点设计：

```
GET /v1/apis/{api_id}/forms/{operation_id}
Response: {
  "form_schema": {
    "fields": [
      {
        "name": "string",
        "type": "string",
        "required": "boolean",
        "description": "string",
        "default": "any"
      }
    ]
  }
}

POST /v1/apis/{api_id}/forms/{operation_id}
{
  "form_data": "object"
}
```

### 3.2 API 文档界面

功能：

- 动态 Swagger UI 集成
- 交互式 API 测试
- 实时文档更新

第四阶段：高级功能（优先级：低）

## 4.1 监控和日志

- 请求日志记录
- 性能监控
- 错误追踪

## 4.2 缓存和优化

- 响应缓存
- 连接池管理
- 负载均衡

## 技术栈和依赖

### 后端 (Rust)

```
[dependencies]
axum = "0.7"
tokio = { version = "1.0", features = ["full"] }
serde = { version = "1.0", features = ["derive"] }
serde_json = "1.0"
serde_yaml = "0.9"
request = { version = "0.11", features = ["json"] }
uuid = { version = "1.0", features = ["v4"] }
chrono = { version = "0.4", features = ["serde"] }
tracing = "0.1"
tracing-subscriber = "0.3"
```

### 前端集成

- Swagger UI
- React/Vue 组件库
- WebSocket 实时更新

## 开发优先级建议

### 立即开始 (本周)

1. 完善 API 注册和管理功能
2. 实现基础的动态代理
3. 添加基本的错误处理

### 下周计划

1. 完善参数验证系统
2. 实现响应验证
3. 添加日志记录

## 下个月计划

1. 开发动态表单生成
2. 集成 Swagger UI
3. 添加监控功能

## 测试策略

### 单元测试

- API 注册逻辑测试
- 参数验证测试
- 代理转发测试

### 集成测试

- 端到端 API 调用测试
- 错误处理测试
- 性能测试

### 前端测试

- 表单生成测试
- UI 交互测试
- 响应式设计测试

## 部署和运维

### 开发环境




- Docker Compose 本地开发
- 热重载支持
- 调试工具集成

### 生产环境




- Kubernetes 部署
- 健康检查
- 监控告警

## 成功指标

### 功能指标

-  支持 100+ 并发 API 注册
-  请求转发延迟 < 100ms
-  99.9% 可用性

### 开发指标

-  代码覆盖率 > 80%
-  API 文档完整性 100%
-  错误处理覆盖率 100%

## 风险评估

### 技术风险

- OpenAPI 规范复杂性
- 性能瓶颈
- 内存泄漏

### 缓解措施

- 渐进式功能开发
- 性能测试和优化
- 代码审查和测试

## 下一步行动

1. **立即开始**：完善 API 注册功能
2. **本周完成**：基础代理功能
3. **下周目标**：参数验证系统
4. **持续改进**：根据测试反馈优化

---

最后更新：2024年12月