# StepFlow Gateway 认证系统设计

## 🎯 认证系统概述

StepFlow Gateway 需要支持多种认证方式，包括：

1. **API 端点认证** - 目标 API 的认证配置
2. **Gateway 认证** - Gateway 自身的用户认证
3. **动态认证** - 运行时动态获取认证信息
4. **认证缓存** - 提高性能的认证信息缓存

## 📊 认证相关数据库表设计

### 1. API 认证配置表 (api_auth_configs)

```
CREATE TABLE api_auth_configs (
    id TEXT PRIMARY KEY,
    api_document_id TEXT NOT NULL,
    auth_type TEXT NOT NULL, -- 'none', 'basic', 'bearer', 'api_key',
'oauth2', 'custom'
    auth_config TEXT NOT NULL, -- JSON 格式的认证配置
    is_required INTEGER DEFAULT 1, -- 是否必需认证
    is_global INTEGER DEFAULT 0, -- 是否全局配置
    priority INTEGER DEFAULT 0, -- 优先级
    status TEXT DEFAULT 'active',
    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 INDEX idx_auth_api_document_id ON
api_auth_configs(api_document_id);
CREATE INDEX idx_auth_type ON api_auth_configs(auth_type);
CREATE INDEX idx_auth_status ON api_auth_configs(status);
```

### 2. 认证凭据表 (auth_credentials)

```
CREATE TABLE auth_credentials (
    id TEXT PRIMARY KEY,
    auth_config_id TEXT NOT NULL,
    credential_type TEXT NOT NULL, -- 'static', 'dynamic', 'template'
    credential_key TEXT NOT NULL, -- 凭据标识
    credential_value TEXT, -- 凭据值（加密存储）
    credential_template TEXT, -- 动态凭据模板
```

```
    is_encrypted INTEGER DEFAULT 1, -- 是否加密存储
    expires_at TEXT, -- 过期时间
    refresh_before_expiry INTEGER DEFAULT 3600, -- 过期前刷新时间（秒）
    last_refreshed_at TEXT, -- 最后刷新时间
    status TEXT DEFAULT 'active',
    created_at TEXT NOT NULL,
    updated_at TEXT NOT NULL,
    FOREIGN KEY (auth_config_id) REFERENCES api_auth_configs(id) ON
DELETE CASCADE
);

-- 索引
CREATE INDEX idx_cred_auth_config_id ON
auth_credentials(auth_config_id);
CREATE INDEX idx_cred_type ON auth_credentials(credential_type);
CREATE INDEX idx_cred_expires_at ON auth_credentials(expires_at);
CREATE INDEX idx_cred_status ON auth_credentials(status);
```

3. 认证缓存表 (auth_cache)

```
CREATE TABLE auth_cache (
    id TEXT PRIMARY KEY,
    auth_config_id TEXT NOT NULL,
    cache_key TEXT NOT NULL, -- 缓存键
    cache_value TEXT NOT NULL, -- 缓存值（加密存储）
    cache_type TEXT NOT NULL, -- 'token', 'session', 'credential'
    expires_at TEXT NOT NULL, -- 缓存过期时间
    created_at TEXT NOT NULL,
    FOREIGN KEY (auth_config_id) REFERENCES api_auth_configs(id) ON
DELETE CASCADE
);

-- 索引
CREATE INDEX idx_cache_auth_config_id ON auth_cache(auth_config_id);
CREATE INDEX idx_cache_key ON auth_cache(cache_key);
CREATE INDEX idx_cache_expires_at ON auth_cache(expires_at);
```

4. 认证日志表 (auth_logs)

```
CREATE TABLE auth_logs (
    id TEXT PRIMARY KEY,
    auth_config_id TEXT NOT NULL,
    request_id TEXT, -- 关联的请求ID
    auth_type TEXT NOT NULL, -- 认证类型
    auth_status TEXT NOT NULL, -- 'success', 'failed', 'expired',
'refreshed'
    auth_method TEXT NOT NULL, -- 'static', 'dynamic', 'cached'
    error_message TEXT, -- 错误信息
```

```
    response_time_ms INTEGER, -- 认证响应时间
    client_ip TEXT, -- 客户端IP
    user_agent TEXT, -- 用户代理
    created_at TEXT NOT NULL,
    FOREIGN KEY (auth_config_id) REFERENCES api_auth_configs(id) ON
DELETE CASCADE
);

-- 索引
CREATE INDEX idx_auth_log_config_id ON auth_logs(auth_config_id);
CREATE INDEX idx_auth_log_status ON auth_logs(auth_status);
CREATE INDEX idx_auth_log_created_at ON auth_logs(created_at);
```

5. Gateway 用户表 (gateway_users)

```
CREATE TABLE gateway_users (
    id TEXT PRIMARY KEY,
    username TEXT UNIQUE NOT NULL,
    email TEXT UNIQUE,
    password_hash TEXT NOT NULL, -- 加密的密码
    salt TEXT NOT NULL, -- 密码盐值
    role TEXT NOT NULL, -- 'admin', 'user', 'api_user'
    permissions TEXT, -- JSON 格式的权限配置
    is_active INTEGER DEFAULT 1,
    last_login_at TEXT,
    created_at TEXT NOT NULL,
    updated_at TEXT NOT NULL
);

-- 索引
CREATE INDEX idx_user_username ON gateway_users(username);
CREATE INDEX idx_user_email ON gateway_users(email);
CREATE INDEX idx_user_role ON gateway_users(role);
CREATE INDEX idx_user_status ON gateway_users(is_active);
```

6. Gateway 会话表 (gateway_sessions)

```
CREATE TABLE gateway_sessions (
    id TEXT PRIMARY KEY,
    user_id TEXT NOT NULL,
    session_token TEXT UNIQUE NOT NULL, -- 会话令牌
    refresh_token TEXT, -- 刷新令牌
    expires_at TEXT NOT NULL, -- 过期时间
    client_info TEXT, -- JSON 格式的客户端信息
    is_active INTEGER DEFAULT 1,
    created_at TEXT NOT NULL,
    FOREIGN KEY (user_id) REFERENCES gateway_users(id) ON DELETE CASCADE
);
```

```sql
-- 索引
CREATE INDEX idx_session_user_id ON gateway_sessions(user_id);
CREATE INDEX idx_session_token ON gateway_sessions(session_token);
CREATE INDEX idx_session_expires_at ON gateway_sessions(expires_at);
```

## 🔐 认证配置示例

1. Basic 认证配置

```json
{
  "auth_type": "basic",
  "auth_config": {
    "username": "api_user",
    "password": "encrypted_password",
    "encoding": "base64"
  }
}
```

2. Bearer Token 认证配置

```json
{
  "auth_type": "bearer",
  "auth_config": {
    "token": "encrypted_token",
    "prefix": "Bearer",
    "header_name": "Authorization"
  }
}
```

3. API Key 认证配置

```json
{
  "auth_type": "api_key",
  "auth_config": {
    "key_name": "X-API-Key",
    "key_value": "encrypted_api_key",
    "location": "header" // "header", "query", "cookie"
  }
}
```

4. OAuth2 认证配置

```
{
  "auth_type": "oauth2",
  "auth_config": {
    "grant_type": "client_credentials",
    "token_url": "https://auth.example.com/oauth/token",
    "client_id": "encrypted_client_id",
    "client_secret": "encrypted_client_secret",
    "scope": "read write",
    "token_type": "Bearer"
  }
}
```

5. 动态认证配置

```
{
  "auth_type": "dynamic",
  "auth_config": {
    "provider": "vault",
    "path": "secret/api-credentials",
    "key": "api_key",
    "refresh_interval": 3600
  }
}
```

## 🔄 认证流程设计

1. API 调用认证流程

```
1. 接收 API 调用请求
    ↓
2. 查找对应的 API 端点
    ↓
3. 检查是否需要认证
    ↓
4. 获取认证配置
    ↓
5. 检查认证缓存
    ↓
6. 执行认证逻辑
    ↓
7. 更新认证缓存
    ↓
8. 转发请求到目标 API
    ↓
9. 记录认证日志
```

2. 动态认证刷新流程

```
1. 检查认证凭据是否即将过期
     ↓
2. 调用认证提供者获取新凭据
     ↓
3. 更新认证缓存
     ↓
4. 记录刷新日志
     ↓
5. 返回新凭据
```

3. Gateway 用户认证流程

```
1. 用户登录请求
     ↓
2. 验证用户名和密码
     ↓
3. 生成会话令牌
     ↓
4. 存储会话信息
     ↓
5. 返回认证响应
     ↓
6. 后续请求验证会话
```

## 安全考虑

1. 数据加密

```python
# 敏感数据加密示例
import cryptography.fernet

def encrypt_sensitive_data(data: str, key: bytes) -> str:
    """加密敏感数据"""
    f = cryptography.fernet.Fernet(key)
    return f.encrypt(data.encode()).decode()

def decrypt_sensitive_data(encrypted_data: str, key: bytes) -> str:
    """解密敏感数据"""
    f = cryptography.fernet.Fernet(key)
    return f.decrypt(encrypted_data.encode()).decode()
```

2. 密码安全

```python
import bcrypt

def hash_password(password: str) -> tuple[str, str]:
    """哈希密码"""
    salt = bcrypt.gensalt()
    password_hash = bcrypt.hashpw(password.encode(), salt)
    return password_hash.decode(), salt.decode()

def verify_password(password: str, password_hash: str, salt: str) ->
bool:
    """验证密码"""
    return bcrypt.checkpw(password.encode(), password_hash.encode())
```

3. 令牌安全

```python
import secrets
import jwt

def generate_session_token(user_id: str, secret: str) -> str:
    """生成会话令牌"""
    payload = {
        'user_id': user_id,
        'exp': datetime.utcnow() + timedelta(hours=24),
        'iat': datetime.utcnow()
    }
    return jwt.encode(payload, secret, algorithm='HS256')

def verify_session_token(token: str, secret: str) -> dict:
    """验证会话令牌"""
    try:
        return jwt.decode(token, secret, algorithms=['HS256'])
    except jwt.ExpiredSignatureError:
        raise ValueError("Token expired")
    except jwt.InvalidTokenError:
        raise ValueError("Invalid token")
```

## 📊 认证监控和审计

1. 认证统计视图

```sql
CREATE VIEW auth_statistics AS
SELECT
    ac.auth_type,
    COUNT(al.id) as total_attempts,
    SUM(CASE WHEN al.auth_status = 'success' THEN 1 ELSE 0 END) as
success_count,
    SUM(CASE WHEN al.auth_status = 'failed' THEN 1 ELSE 0 END) as
```

```
    failure_count,
    AVG(al.response_time_ms) as avg_response_time,
    MAX(al.created_at) as last_attempt
FROM api_auth_configs ac
LEFT JOIN auth_logs al ON ac.id = al.auth_config_id
GROUP BY ac.auth_type;
```

2. 认证失败监控

```
CREATE VIEW auth_failures AS
SELECT
    al.auth_type,
    al.error_message,
    COUNT(*) as failure_count,
    MAX(al.created_at) as last_failure
FROM auth_logs al
WHERE al.auth_status = 'failed'
GROUP BY al.auth_type, al.error_message
ORDER BY failure_count DESC;
```

## 🔧 实现建议

1. 认证管理器

```
class AuthenticationManager:
    """认证管理器"""

    def __init__(self, db_connection):
        self.db = db_connection
        self.cache = {}

    def get_auth_config(self, api_document_id: str) -> dict:
        """获取认证配置"""
        pass

    def authenticate_request(self, request_data: dict, auth_config:
dict) -> dict:
        """认证请求"""
        pass

    def refresh_credentials(self, auth_config_id: str) -> dict:
        """刷新认证凭据"""
        pass

    def cache_auth_info(self, key: str, value: dict, expires_in: int):
        """缓存认证信息"""
        pass
```

2. 认证提供者接口

```
from abc import ABC, abstractmethod

class AuthProvider(ABC):
    """认证提供者接口"""

    @abstractmethod
    def authenticate(self, config: dict) -> dict:
        """执行认证"""
        pass

    @abstractmethod
    def refresh(self, config: dict) -> dict:
        """刷新认证"""
        pass

    @abstractmethod
    def validate(self, credentials: dict) -> bool:
        """验证凭据"""
        pass
```

3. 具体认证提供者

```
class BasicAuthProvider(AuthProvider):
    """Basic 认证提供者"""

    def authenticate(self, config: dict) -> dict:
        username = config['username']
        password = config['password']
        credentials = f"{username}:{password}"
        encoded = base64.b64encode(credentials.encode()).decode()
        return {'Authorization': f"Basic {encoded}"}

class BearerAuthProvider(AuthProvider):
    """Bearer Token 认证提供者"""

    def authenticate(self, config: dict) -> dict:
        token = config['token']
        prefix = config.get('prefix', 'Bearer')
        return {'Authorization': f"{prefix} {token}"}

class OAuth2AuthProvider(AuthProvider):
    """OAuth2 认证提供者"""

    def authenticate(self, config: dict) -> dict:
        # 实现 OAuth2 认证逻辑
        pass
```

```python
    def refresh(self, config: dict) -> dict:
        # 实现 OAuth2 刷新逻辑
        pass
```

## 📋 认证系统检查清单

### 基础功能 ✅

- [ ] API 认证配置管理
- [ ] 多种认证方式支持
- [ ] 认证凭据安全存储
- [ ] 认证缓存机制

### 高级功能 ✅

- [ ] 动态认证凭据
- [ ] 认证凭据自动刷新
- [ ] Gateway 用户认证
- [ ] 会话管理

### 安全功能 ✅

- [ ] 敏感数据加密
- [ ] 密码安全哈希
- [ ] 令牌安全生成
- [ ] 认证日志审计

### 监控功能 ✅

- [ ] 认证统计
- [ ] 失败监控
- [ ] 性能监控
- [ ] 安全审计

---

这个认证系统设计提供了完整的认证解决方案，支持多种认证方式、安全存储、动态刷新和监控审计功能。