StepFlow Gateway HTTP 请求完整信息记录

概述

`api_call_logs` 表记录了每个 HTTP 请求的完整信息，包括请求和响应的所有细节。这为调试、监控、审计和性能分析提供了全面的数据支持。

完整的 HTTP 请求信息结构

数据库表结构

```
CREATE TABLE api_call_logs (
    id TEXT PRIMARY KEY,                        -- 日志记录唯一ID
    api_endpoint_id TEXT NOT NULL,              -- 关联的API端点ID
    resource_reference_id TEXT,                 -- 关联的资源引用ID（工作流/调度
任务）

    -- 请求信息
    request_method TEXT NOT NULL,               -- HTTP方法：GET, POST, PUT,
DELETE, PATCH
    request_url TEXT NOT NULL,                  -- 完整的请求URL
    request_headers TEXT,                       -- 请求头（JSON格式）
    request_body TEXT,                          -- 请求体内容
    request_params TEXT,                        -- 请求参数（JSON格式）

    -- 响应信息
    response_status_code INTEGER,               -- HTTP状态码
    response_headers TEXT,                      -- 响应头（JSON格式）
    response_body TEXT,                         -- 响应体内容

    -- 性能信息
    response_time_ms INTEGER,                   -- 响应时间（毫秒）
    request_size_bytes INTEGER,                 -- 请求大小（字节）
    response_size_bytes INTEGER,                -- 响应大小（字节）

    -- 错误信息
    error_message TEXT,                         -- 错误消息
    error_type TEXT,                            -- 错误类型

    -- 客户端信息
    client_ip TEXT,                             -- 客户端IP地址
    user_agent TEXT,                            -- 用户代理字符串

    -- 时间信息
    created_at TEXT NOT NULL,                   -- 请求时间戳

    -- 外键约束
    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
);
```

详细的字段说明

1. 请求信息 (Request Information)

`request_method`

- **类型**: TEXT
- **说明**: HTTP 请求方法
- **示例值**: `"GET"`, `"POST"`, `"PUT"`, `"DELETE"`, `"PATCH"`
- **用途**: 确定请求的操作类型

`request_url`

- **类型**: TEXT
- **说明**: 完整的请求 URL
- **示例值**: `"https://api.example.com/v1/users?limit=10&page=1"`
- **用途**: 记录实际请求的目标地址

`request_headers`

- **类型**: TEXT (JSON 字符串)
- **说明**: 所有请求头信息
- **示例值**:

```
{
  "Content-Type": "application/json",
  "Authorization": "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "User-Agent": "StepFlow-Gateway/1.0.0",
  "Accept": "application/json",
  "X-Request-ID": "req-12345-67890",
  "X-Forwarded-For": "192.168.1.100"
}
```

- **用途**: 记录认证信息、内容类型、追踪ID等

`request_body`

- **类型**: TEXT
- **说明**: 请求体内容（适用于 POST, PUT, PATCH 请求）
- **示例值**:

```
{
  "name": "John Doe",
  "email": "john@example.com",
  "age": 30,
  "preferences": {
    "theme": "dark",
    "language": "en"
  }
}
```

- **用途**: 记录请求的完整数据

`request_params`

- **类型**: TEXT (JSON 字符串)
- **说明**: 查询参数和路径参数
- **示例值**:

```
{
  "query": {
    "limit": "10",
    "page": "1",
    "sort": "name",
    "filter": "active"
  },
  "path": {
    "userId": "12345",
    "categoryId": "67890"
  }
}
```

- **用途**: 记录 URL 参数和路径变量

2. 响应信息 (Response Information)

`response_status_code`

- **类型**: INTEGER
- **说明**: HTTP 响应状态码
- **示例值**: `200`, `201`, `400`, `401`, `404`, `500`
- **用途**: 判断请求是否成功

`response_headers`

- **类型**: TEXT (JSON 字符串)
- **说明**: 所有响应头信息
- **示例值**:

```
{
  "Content-Type": "application/json; charset=utf-8",
  "Content-Length": "1024",
  "Cache-Control": "no-cache",
  "X-Rate-Limit-Remaining": "999",
  "X-Rate-Limit-Reset": "1640995200",
  "X-Request-ID": "req-12345-67890"
}
```

- **用途**: 记录响应元数据、限流信息等

`response_body`

- **类型**: TEXT
- **说明**: 响应体内容
- **示例值**:

```
{
  "success": true,
  "data": {
    "id": "12345",
    "name": "John Doe",
    "email": "john@example.com",
    "created_at": "2024-01-01T00:00:00Z"
  },
  "meta": {
    "total": 1,
    "page": 1,
    "limit": 10
  }
}
```

- **用途**: 记录完整的响应数据

3. 性能信息 (Performance Information)

`response_time_ms`

- **类型**: INTEGER
- **说明**: 请求响应时间（毫秒）
- **示例值**: `150`, `2500`, `5000`
- **用途**: 性能监控和优化

`request_size_bytes`

- **类型**: INTEGER

- **说明**: 请求大小（字节）
- **示例值**: `1024`, `5120`, `0`
- **用途**: 网络流量监控

`response_size_bytes`

- **类型**: INTEGER
- **说明**: 响应大小（字节）
- **示例值**: `2048`, `10240`, `500`
- **用途**: 网络流量监控

4. 错误信息 (Error Information)

`error_message`

- **类型**: TEXT
- **说明**: 详细的错误消息
- **示例值**:
- `"Connection timeout after 30 seconds"`
- `"Invalid JSON format in request body"`
- `"Authentication failed: Invalid token"`
- **用途**: 错误诊断和调试

`error_type`

- **类型**: TEXT
- **说明**: 错误类型分类
- **示例值**:
- `"timeout"` - 超时错误
- `"validation"` - 验证错误
- `"authentication"` - 认证错误
- `"authorization"` - 授权错误
- `"network"` - 网络错误
- `"server_error"` - 服务器错误
- **用途**: 错误分类和统计

5. 客户端信息 (Client Information)

`client_ip`

- **类型**: TEXT
- **说明**: 客户端 IP 地址
- **示例值**: `"192.168.1.100"`, `"10.0.0.50"`
- **用途**: 访问追踪和安全审计

`user_agent`

- **类型**: TEXT
- **说明**: 用户代理字符串
- **示例值**:
- "StepFlow-Gateway/1.0.0"
- "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36"
- **用途**: 客户端识别和统计

实际使用示例

1. 记录完整的 HTTP 请求

```
import json
import time
from datetime import datetime

def log_http_request(api_endpoint_id, request_data, response_data,
client_info=None):
    """记录完整的 HTTP 请求信息"""

    # 计算响应时间
    response_time = int((time.time() - request_data.get('start_time',
time.time())) * 1000)

    # 计算请求和响应大小
    request_size = len(request_data.get('body', '').encode('utf-8'))
    response_size = len(response_data.get('body', '').encode('utf-8'))

    # 准备日志数据
    log_data = {
        'id': str(uuid.uuid4()),
        'api_endpoint_id': api_endpoint_id,
        'resource_reference_id':
request_data.get('resource_reference_id'),

        # 请求信息
        'request_method': request_data['method'],
        'request_url': request_data['url'],
        'request_headers': json.dumps(request_data['headers']),
        'request_body': request_data.get('body', ''),
        'request_params': json.dumps({
            'query': request_data.get('query_params', {}),
            'path': request_data.get('path_params', {})
        }),

        # 响应信息
        'response_status_code': response_data['status_code'],
        'response_headers': json.dumps(response_data['headers']),
        'response_body': response_data.get('body', ''),

        # 性能信息
```

```
            'response_time_ms': response_time,
            'request_size_bytes': request_size,
            'response_size_bytes': response_size,

            # 错误信息
            'error_message': response_data.get('error_message'),
            'error_type': response_data.get('error_type'),

            # 客户端信息
            'client_ip': client_info.get('ip') if client_info else None,
            'user_agent': client_info.get('user_agent') if client_info else
None,

            # 时间信息
            'created_at': datetime.now().isoformat()
        }

        # 插入数据库
        cursor.execute('''
            INSERT INTO api_call_logs
            (id, api_endpoint_id, resource_reference_id, request_method,
request_url,
             request_headers, request_body, request_params,
response_status_code,
             response_headers, response_body, response_time_ms,
request_size_bytes,
             response_size_bytes, error_message, error_type, client_ip,
user_agent, created_at)
            VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
        ''', (
            log_data['id'], log_data['api_endpoint_id'],
log_data['resource_reference_id'],
            log_data['request_method'], log_data['request_url'],
log_data['request_headers'],
            log_data['request_body'], log_data['request_params'],
log_data['response_status_code'],
            log_data['response_headers'], log_data['response_body'],
log_data['response_time_ms'],
            log_data['request_size_bytes'], log_data['response_size_bytes'],
log_data['error_message'],
            log_data['error_type'], log_data['client_ip'],
log_data['user_agent'], log_data['created_at']
        ))

        conn.commit()
        return log_data['id']
```

2. 查询和分析请求日志

```
def analyze_request_logs(endpoint_id=None, time_range=None,
status_code=None):
    """分析请求日志"""

    query = '''
        SELECT
            request_method,
            request_url,
            response_status_code,
            response_time_ms,
            request_size_bytes,
            response_size_bytes,
            error_type,
            created_at
        FROM api_call_logs
        WHERE 1=1
    '''
    params = []

    if endpoint_id:
        query += ' AND api_endpoint_id = ?'
        params.append(endpoint_id)

    if time_range:
        query += ' AND created_at > datetime("now", "-{}
days")'.format(time_range)

    if status_code:
        query += ' AND response_status_code = ?'
        params.append(status_code)

    query += ' ORDER BY created_at DESC'

    cursor.execute(query, params)
    return cursor.fetchall()

def get_performance_stats(endpoint_id, days=7):
    """获取性能统计"""

    cursor.execute('''
        SELECT
            COUNT(*) as total_requests,
            AVG(response_time_ms) as avg_response_time,
            MAX(response_time_ms) as max_response_time,
            MIN(response_time_ms) as min_response_time,
            SUM(request_size_bytes) as total_request_size,
            SUM(response_size_bytes) as total_response_size,
            SUM(CASE WHEN response_status_code BETWEEN 200 AND 299 THEN
1 ELSE 0 END) as success_count,
            SUM(CASE WHEN response_status_code >= 400 THEN 1 ELSE 0 END)
as error_count
        FROM api_call_logs
```

```
        WHERE api_endpoint_id = ?
        AND created_at > datetime('now', '-{} days')
    '''.format(days), (endpoint_id,))

    return cursor.fetchone()
```

3. 错误分析和监控

```
def get_error_analysis(days=1):
    """获取错误分析"""

    cursor.execute('''
        SELECT
            error_type,
            COUNT(*) as error_count,
            AVG(response_time_ms) as avg_response_time,
            GROUP_CONCAT(DISTINCT response_status_code) as status_codes
        FROM api_call_logs
        WHERE error_type IS NOT NULL
        AND created_at > datetime('now', '-{} days')
        GROUP BY error_type
        ORDER BY error_count DESC
    '''.format(days))

    return cursor.fetchall()

def get_slow_requests(threshold_ms=1000, limit=10):
    """获取慢请求"""

    cursor.execute('''
        SELECT
            request_method,
            request_url,
            response_time_ms,
            response_status_code,
            created_at
        FROM api_call_logs
        WHERE response_time_ms > ?
        ORDER BY response_time_ms DESC
        LIMIT ?
    ''', (threshold_ms, limit))

    return cursor.fetchall()
```

数据安全和隐私

敏感信息处理

```
def sanitize_log_data(log_data):
    """清理敏感信息"""

    # 清理认证头
    if 'headers' in log_data:
        headers = json.loads(log_data['headers'])
        if 'Authorization' in headers:
            headers['Authorization'] = '***REDACTED***'
        if 'X-API-Key' in headers:
            headers['X-API-Key'] = '***REDACTED***'
        log_data['headers'] = json.dumps(headers)

    # 清理请求体中的敏感字段
    if log_data.get('body'):
        body = json.loads(log_data['body'])
        sensitive_fields = ['password', 'token', 'secret', 'key']
        for field in sensitive_fields:
            if field in body:
                body[field] = '***REDACTED***'
        log_data['body'] = json.dumps(body)

    return log_data
```

数据保留策略

```
def cleanup_old_logs(retention_days=30):
    """清理旧的日志数据"""

    cursor.execute('''
        DELETE FROM api_call_logs
        WHERE created_at < datetime('now', '-{} days')
    '''.format(retention_days))

    deleted_count = cursor.rowcount
    conn.commit()

    print(f"已清理 {deleted_count} 条旧日志记录")
    return deleted_count
```

这个设计确保了每个 HTTP 请求的完整信息都被记录下来，为调试、监控、性能分析和安全审计提供了全面的数据支持。