AsyncAPI 扩展计划

🎯 扩展目标

为 StepFlow Gateway 添加 AsyncAPI 支持，实现异步架构的 API 网关功能，支持消息队列、事件驱动架构和实时通信。

📋 AsyncAPI 核心概念

1. **异步 API 规范**

- **AsyncAPI 2.x**: 基于 OpenAPI 3.0 的异步扩展
- **消息驱动**: 发布/订阅模式
- **事件流**: 实时数据流处理
- **协议支持**: MQTT, AMQP, Kafka, WebSocket, Server-Sent Events

2. **核心组件**

- **Channels**: 消息通道（类似 OpenAPI 的 paths）
- **Messages**: 消息定义（类似 OpenAPI 的 operations）
- **Servers**: 服务器配置（消息代理）
- **Schemas**: 消息格式定义
- **Security**: 安全配置

🏗️ 架构设计

1. **模块扩展结构**

```
src/stepflow_gateway/
├── asyncapi/                    # AsyncAPI 模块
│   ├── __init__.py
│   ├── manager.py               # AsyncAPI 管理器
│   ├── parser.py                # AsyncAPI 解析器
│   ├── executor.py              # 异步执行器
│   ├── protocols/               # 协议支持
│   │   ├── __init__.py
│   │   ├── mqtt.py              # MQTT 协议
│   │   ├── amqp.py              # AMQP 协议
│   │   ├── kafka.py             # Kafka 协议
│   │   ├── websocket.py         # WebSocket 协议
│   │   └── sse.py               # Server-Sent Events
│   └── handlers/                # 消息处理器
│       ├── __init__.py
│       ├── publisher.py         # 发布者
│       ├── subscriber.py        # 订阅者
│       └── stream.py            # 流处理器
```

2. 数据库扩展

```
-- AsyncAPI 模板表
CREATE TABLE asyncapi_templates (
    id TEXT PRIMARY KEY,
    name TEXT NOT NULL,
    content TEXT NOT NULL,
    status TEXT DEFAULT 'active',
    created_at TEXT NOT NULL,
    updated_at TEXT NOT NULL
);

-- AsyncAPI 文档表
CREATE TABLE asyncapi_documents (
    id TEXT PRIMARY KEY,
    template_id TEXT NOT NULL,
    name TEXT NOT NULL,
    version TEXT,
    base_url TEXT,
    status TEXT DEFAULT 'active',
    created_at TEXT NOT NULL,
    updated_at TEXT NOT NULL,
    FOREIGN KEY (template_id) REFERENCES asyncapi_templates(id)
);

-- 消息通道表
CREATE TABLE message_channels (
    id TEXT PRIMARY KEY,
    asyncapi_document_id TEXT NOT NULL,
    channel_name TEXT NOT NULL,
    protocol TEXT NOT NULL,
    description TEXT,
    publish_operations TEXT,  -- JSON
    subscribe_operations TEXT, -- JSON
    parameters TEXT,          -- JSON
    status TEXT DEFAULT 'active',
    created_at TEXT NOT NULL,
    updated_at TEXT NOT NULL,
    FOREIGN KEY (asyncapi_document_id) REFERENCES asyncapi_documents(id)
);

-- 消息定义表
CREATE TABLE message_definitions (
    id TEXT PRIMARY KEY,
    channel_id TEXT NOT NULL,
    message_name TEXT NOT NULL,
    message_type TEXT NOT NULL, -- publish/subscribe
    payload_schema TEXT,        -- JSON
    headers_schema TEXT,        -- JSON
    description TEXT,
    status TEXT DEFAULT 'active',
```

```
    created_at TEXT NOT NULL,
    updated_at TEXT NOT NULL,
    FOREIGN KEY (channel_id) REFERENCES message_channels(id)
);

-- 消息代理配置表
CREATE TABLE message_brokers (
    id TEXT PRIMARY KEY,
    asyncapi_document_id TEXT NOT NULL,
    broker_name TEXT NOT NULL,
    protocol TEXT NOT NULL,
    url TEXT NOT NULL,
    security_scheme TEXT,      -- JSON
    description TEXT,
    status TEXT DEFAULT 'active',
    created_at TEXT NOT NULL,
    updated_at TEXT NOT NULL,
    FOREIGN KEY (asyncapi_document_id) REFERENCES asyncapi_documents(id)
);

-- 消息调用日志表
CREATE TABLE message_call_logs (
    id TEXT PRIMARY KEY,
    channel_id TEXT NOT NULL,
    message_id TEXT NOT NULL,
    operation_type TEXT NOT NULL, -- publish/subscribe
    payload TEXT,                 -- JSON
    headers TEXT,                 -- JSON
    broker_response TEXT,         -- JSON
    status TEXT NOT NULL,         -- success/error
    error_message TEXT,
    response_time_ms INTEGER,
    created_at TEXT NOT NULL,
    FOREIGN KEY (channel_id) REFERENCES message_channels(id),
    FOREIGN KEY (message_id) REFERENCES message_definitions(id)
);
```

🔧 核心功能实现

1. **AsyncAPI 解析器**

```
class AsyncApiParser:
    """AsyncAPI 文档解析器"""

    def parse_document(self, asyncapi_content: str) -> Dict[str, Any]:
        """解析 AsyncAPI 文档"""
        pass

    def extract_channels(self, doc: Dict[str, Any]) -> List[Dict[str,
Any]]:
```

```
        """提取消息通道"""
        pass

    def extract_messages(self, doc: Dict[str, Any]) -> List[Dict[str,
Any]]:
        """提取消息定义"""
        pass

    def extract_servers(self, doc: Dict[str, Any]) -> List[Dict[str,
Any]]:
        """提取服务器配置"""
        pass
```

2. **协议适配器**

```
class ProtocolAdapter:
    """协议适配器基类"""

    async def connect(self, config: Dict[str, Any]):
        """连接到消息代理"""
        pass

    async def publish(self, channel: str, message: Dict[str, Any]):
        """发布消息"""
        pass

    async def subscribe(self, channel: str, callback: Callable):
        """订阅消息"""
        pass

    async def disconnect(self):
        """断开连接"""
        pass

class MqttAdapter(ProtocolAdapter):
    """MQTT 协议适配器"""
    pass

class KafkaAdapter(ProtocolAdapter):
    """Kafka 协议适配器"""
    pass

class WebSocketAdapter(ProtocolAdapter):
    """WebSocket 协议适配器"""
    pass
```

3. **消息管理器**

```python
class AsyncApiManager:
    """AsyncAPI 管理器"""

    def register_asyncapi(self, name: str, asyncapi_content: str,
                          version: str = None, base_url: str = None) ->
Dict[str, Any]:
        """注册 AsyncAPI 文档"""
        pass

    async def publish_message(self, channel_id: str, message_data:
Dict[str, Any]) -> Dict[str, Any]:
        """发布消息"""
        pass

    async def subscribe_to_channel(self, channel_id: str, callback:
Callable) -> str:
        """订阅消息通道"""
        pass

    def list_channels(self, asyncapi_document_id: str = None) ->
List[Dict[str, Any]]:
        """列出消息通道"""
        pass

    def get_message_schema(self, message_id: str) -> Dict[str, Any]:
        """获取消息模式"""
        pass
```

🌐 Web API 扩展

1. **AsyncAPI 管理接口**

```python
# 注册 AsyncAPI 文档
@app.post("/asyncapis/register")
def register_asyncapi(req: AsyncApiRegisterRequest):
    pass

# 列出 AsyncAPI 文档
@app.get("/asyncapis")
def list_asyncapis():
    pass

# 获取消息通道
@app.get("/channels")
def list_channels(asyncapi_document_id: str = None):
    pass

# 发布消息
@app.post("/messages/publish")
```

```python
async def publish_message(req: PublishMessageRequest):
    pass

# 订阅消息
@app.post("/messages/subscribe")
async def subscribe_message(req: SubscribeMessageRequest):
    pass

# WebSocket 连接
@app.websocket("/ws/{channel_id}")
async def websocket_endpoint(websocket: WebSocket, channel_id: str):
    pass
```

2. 实时监控接口

```python
# 消息流监控
@app.get("/messages/stream")
async def message_stream():
    pass

# 连接状态
@app.get("/connections/status")
def get_connection_status():
    pass

# 消息统计
@app.get("/messages/statistics")
def get_message_statistics():
    pass
```

📊 监控和日志

1. 消息监控

- 消息吞吐量统计
- 延迟监控
- 错误率统计
- 连接状态监控

2. 日志记录

- 消息发布日志
- 消息接收日志
- 连接事件日志
- 错误日志

🔐 安全支持

1. **认证方式**

- MQTT: 用户名/密码, TLS 证书
- Kafka: SASL, SSL/TLS
- AMQP: PLAIN, AMQPLAIN, EXTERNAL
- WebSocket: JWT, API Key

2. **授权控制**

- 通道级权限控制
- 消息级权限控制
- 发布/订阅权限分离

🧪 测试策略

1. **单元测试**

- AsyncAPI 解析器测试
- 协议适配器测试
- 消息管理器测试

2. **集成测试**

- 端到端消息流测试
- 多协议集成测试
- 性能测试

3. **示例场景**

- IoT 设备数据流
- 实时通知系统
- 事件驱动架构
- 微服务通信

🚀 实施计划

阶段 1: 基础架构 (1-2 周)

- [ ] 创建 AsyncAPI 模块结构
- [ ] 实现数据库模式扩展
- [ ] 创建基础解析器

阶段 2: 协议支持 (2-3 周)

- [ ] 实现 MQTT 适配器
- [ ] 实现 WebSocket 适配器
- [ ] 实现基础消息管理

阶段 3: Web API (1-2 周)

- [ ] 实现 REST API 接口
- [ ] 实现 WebSocket 端点
- [ ] 添加监控接口

阶段 4: 高级功能 (2-3 周)

- [ ] 实现 Kafka 适配器
- [ ] 实现 AMQP 适配器
- [ ] 添加安全支持
- [ ] 完善监控和日志

阶段 5: 测试和优化 (1-2 周)

- [ ] 编写测试用例
- [ ] 性能优化
- [ ] 文档完善

📚 使用示例

1. **注册 AsyncAPI 文档**

```
# 注册 MQTT 设备 API
asyncapi_content = {
    "asyncapi": "2.5.0",
    "info": {
        "title": "IoT Device API",
        "version": "1.0.0"
    },
    "servers": {
        "production": {
            "url": "mqtt://broker.example.com",
            "protocol": "mqtt"
        }
    },
    "channels": {
        "device/{deviceId}/data": {
            "publish": {
                "message": {
                    "$ref": "#/components/messages/DeviceData"
                }
            }
        }
    },
    "components": {
        "messages": {
            "DeviceData": {
                "payload": {
                    "type": "object",
                    "properties": {
                        "temperature": {"type": "number"},
```

```
                        "humidity": {"type": "number"}
                    }
                }
            }
        }
    }
}

result = gateway.register_asyncapi("IoT API",
json.dumps(asyncapi_content))
```

2. **发布消息**

```
# 发布设备数据
result = await gateway.publish_message(
    channel_id="channel-123",
    message_data={
        "deviceId": "sensor-001",
        "temperature": 25.5,
        "humidity": 60.2
    }
)
```

3. **订阅消息**

```
# 订阅设备数据
async def handle_device_data(message):
    print(f"收到设备数据: {message}")

subscription_id = await gateway.subscribe_to_channel(
    channel_id="channel-123",
    callback=handle_device_data
)
```

🎯 预期收益

1. **功能扩展**

- 支持异步 API 规范
- 实现实时通信能力
- 支持事件驱动架构

2. **应用场景**

- IoT 设备管理
- 实时通知系统

- 微服务通信
- 数据流处理

3. **技术优势**

- 统一的 API 网关
- 多协议支持
- 完整的监控体系
- 安全可靠

这个扩展计划将为 StepFlow Gateway 添加强大的异步架构支持，使其能够处理现代分布式系统中的实时通信需求。