リスト1　Discord Botのメッセージ処理

```
intents = discord.Intents.default()
intents.message_content = True
client = discord.Client(intents=intents)
bot_id = "<@XXXXXXXXXXXXXXXXXX>"  # 使用するBotのユーザID

@client.event
async def on_message(message):
    if message.author.bot or bot_id not in message.content:
        return

    # ユーザID除去
    cleaned = re.sub(r"^<@¥d+>¥s*", "", message.content)
    user_input = cleaned.strip()

    if not user_input:
        return

    try:
        response = requests.post(f"http://localhost:{LLM_PORT}/interpret", json={"message": user_input}, timeout=10)
        if response.status_code == 200:
            result = response.json()
            reply = result["reply"]
            await message.channel.send(reply)
        else:
            await message.channel.send(f"⚠ LLMサーバーからの応答に失敗しました（{response.status_code}）")
    except Exception as e:
        await message.channel.send(f"✖ エラーが発生しました: {e}")


client.run(DISCORD_TOKEN)
```

リスト2　LLMサーバのAPIキー設定

```
from openai import OpenAI

OPENAI_API_KEY = "sk-proj-..."  # 取得したAPIキー
openai = OpenAI(api_key=OPENAI_API_KEY)
```

リスト3　action_mapの定義

```
action_map = {
    "turn_on_mcp": {"func": turn_on, "description": "照明の電源をONにする", "params": ["deviceId"]},
    "turn_off_mcp": {"func": turn_off, "description": "照明の電源をOFFにする", "params": ["deviceId"]},
    "turn_on_color_mcp": {"func": turn_on_color, "description": "照明の色を変更してONにする", "params": ["deviceId",
"color"]},
    "turn_on_brightness_mcp": {"func": turn_on_brightness, "description": "照明の明るさを変更してONにする", "params":
["deviceId", "brightness"]},
    "get_device_data_mcp": {"func": get_device_data, "description": "デバイスのデータを取得する", "params": ["deviceId"]},
    "air_conditioner_on_mcp": {"func": air_conditioner_on, "description": "エアコンをONにする", "params": ["deviceId",
"mode", "temperature"]},
    "air_conditioner_off_mcp": {"func": air_conditioner_off, "description": "エアコンをOFFにする", "params": ["deviceId"]},
    "add_schedule_mcp": {
        "func": add_schedule,
        "description": "スケジュール（予約）を追加する",
        "params": ["time", "action", "deviceId", "note", "mode", "temperature", "brightness", "color", "fan_speed",
"commandType"],
    },
}
```

リスト4　ツール一覧の生成処理

```
def build_tools_from_action_map(action_map):
    tools = []
    for name, info in action_map.items():
        params = info["params"]
        properties = {p: {"type": "string"} for p in params}
        tools.append(
            {
                "type": "function",
                "function": {
                    "name": name,
                    "description": info["description"],
                    "parameters": {"type": "object", "properties": properties, "required": params},
                },
            }
        )
    return tools
```

リスト5　MCPサーバへのツール実行要求

```
async def call_mcp_tool(tool_name, args):
    async with Client(MCP_BASE_URL) as client:
        result = await client.call_tool(tool_name, args)
        return result
```

リスト6　ツール実行ループとLLM応答生成

```
response = openai.chat.completions.create(model="gpt-4o", messages=[{"role": "user", "content": prompt}], tools=tools,
tool_choice="auto")  # モデルはfunction calling対応のものを指定
```

```
message = response.choices[0].message
messages = [{"role": "user", "content": prompt}]
for _ in range(3):
    if message.tool_calls:
        assistant_message = message.to_dict()
        messages.append(assistant_message)
        for tool_call in message.tool_calls:
            tool_name = tool_call.function.name
            args = json.loads(tool_call.function.arguments)
            # 中略
            mcp_response = requests.post(f"{MCP_BASE_URL}/use_mcp_tool", json={"tool_name": tool_name, "arguments": args})
            mcp_result = mcp_response.json().get("result")
            messages.append({"role": "tool", "content": str(mcp_result), "tool_call_id": tool_call.id})
        # 再度LLMにtool messageを渡して応答生成
        response = openai.chat.completions.create(model="gpt-4-0613", messages=messages, tools=tools, tool_choice="auto")
        message = response.choices[0].message
    else:
        # ツール呼び出しが不要な場合はそのまま返す
        return {"reply": message.content.strip()}
# tool_callsが続く場合はエラーを返す
return {"reply": "ツール呼び出しが3回を超えたため、処理を中断します。"}
```

リスト7　LLMに与えるシステム・プロンプト

```
PROMPT_TEMPLATE = """
以下の指示に従い、必要ならツールを使って自然な日本語で回答してください。
- 「〇時に」「明日」「◯分後」など時刻・日付・未来のタイミングが含まれる指示の場合は、予約用ツールを使ってください。
- 曖昧な表現（例: 落ち着いた暗さ、暖かめ、強めなど）は必ず具体的な数値・パラメータとして指定してください。
  - 数値やパラメータは文脈から最適な値を推論し、function callingで明示的に指定すること。
- timeは必ずISO8601形式（yyyy-mm-ddThh:mm）で指定してください
- colorは2700-6500の色温度で指定してください
- actionには「_mcp」のついたツール名を指定してください
- air_conditioner_onのmodeは 0/1(自動), 2(冷房), 3(乾燥), 4(ファン), 5(暖房)のように指定できます
- 室温, 湿度, 照度などはハブ2が取得できます
- 目標時刻に室温が指定温度になるようにする場合は、エアコンの強さ・部屋の温度変化を考慮し、必要に応じて目標時刻より前にエアコンON予約を入れてください。
  - 例: 「7時に室温20度」→暖房/冷房強めなら6:30、弱めなら6:00にON予約など。

デバイスリスト: {devices}

指示: {instruction}
現在時刻: {now}
"""
```

リスト8　SwitchBot API用の認証ヘッダ

```
def create_header():
    token = os.getenv("SWITCHBOT_TOKEN")
    secret = os.getenv("SWITCHBOT_SECRET")
    if token:
        token = token.strip()
    if secret:
        secret = secret.strip()
    nonce = uuid.uuid4()
    timestamp = int(round(time.time() * 1000))
    string_to_sign = f"{token}{timestamp}{nonce}"
    signature = base64.b64encode(hmac.new(secret.encode(), msg=string_to_sign.encode(),
digestmod=hashlib.sha256).digest()).decode()
    return {
        "Authorization": token,
        "Content-Type": "application/json",
        "charset": "utf8",
        "t": str(timestamp),
        "sign": signature,
        "nonce": str(nonce),
    }
```

リスト9　エアコン操作APIのラップ処理

```
def air_conditioner_on(deviceId, mode, temperature):
    print("エアコンON")
    url = f"https://api.switch-bot.com/v1.1/devices/{deviceId}/commands"
    parameter = f"{temperature},{mode},1,on"
    payload = {"command": "setAll", "parameter": parameter, "commandType": "command"}
    headers = create_header()
    response = requests.post(url, headers=headers, json=payload)
    return response.json() if response.status_code == 200 else response.text
```

リスト10　MCPサーバの実装

```
from fastmcp import FastMCP
from switchbot_api import air_conditioner_on
import json
import os

mcp = FastMCP("switchbot-mcp-server", stateless_http=True)
```

```
@mcp.tool
def air_conditioner_on_mcp(deviceId: str, mode: str, temperature: str):
    return air_conditioner_on(deviceId, mode, temperature)


if __name__ == "__main__":
    mcp.run(transport="streamable-http", port=int(os.getenv("MCP_PORT")), host="0.0.0.0")
```

リスト11　予約実行サーバの実装

```
for task in schedule:
    if task["time"][:16] == now[:16]:  # 分単位で一致
        action = task.get("action")
        note = task.get("note")
        func = action_map.get(action)
        if callable(func):
            print(f"実行: {note}")
            # 必要なパラメータを自動で抽出して関数に渡す
            params = action_map_dict.get(action, {}).get("params", [])
            args = [task.get(p) for p in params]
            func(*args)
            executed.append(task)  # 実行済みに追加
        else:
            print(f"[警告] 未定義のアクション: {action}")
    else:
        updated_schedule.append(task)
```

リスト12　システム起動時のサーバ・ログ例

```
mcp_server-1  | INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
llm_server-1  | INFO:     Uvicorn running on http://0.0.0.0:8001 (Press CTRL+C to quit)
llm_server-1  | 2025-11-11 02:46:24 INFO     discord.gateway Shard ID None has connected to Gateway (Session ID:
xxxxxxxxxxxxxx).
```