```
 1: import asyncio
 2: from typing import Any, Dict, List
 3: from fastmcp import Client
 4: from llama_index.llms.ollama import Ollama
 5: from llama_index.core.agent import FunctionAgent
 6: from llama_index.core.tools import FunctionTool
 7: import sys
 8:
 9: # Pydantic関連の非推奨警告を抑制
10: import warnings
11: from pydantic import PydanticDeprecatedSince20, PydanticDeprecatedSince211
12: warnings.filterwarnings("ignore", category=DeprecationWarning, module="pydantic")
13: warnings.filterwarnings("ignore", category=PydanticDeprecatedSince20)
14: warnings.filterwarnings("ignore", category=PydanticDeprecatedSince211)
15:
16: # ------------------------------------------------------------
17: # 設定（接続先とモデル）
18: # ------------------------------------------------------------
19:
20: OLLAMA_BASE_URL = "http://192.168.10.36:11434"    #<-Ollama用 アドレス・ポートは状況に合わせてくださi
21: MCP_SERVER = "http://192.168.10.35:8000/mcp"      #<-Ollama用 アドレス・ポートは状況に合わせてくださis
22: LLM_MODEL = "gpt-oss"                                          #<- LLM モデル
23: DEFAULT_QUERY = "東京の天気を教えて"                        #<- 試験用 問い合わせ
24:
25: # LLM に設定するプロンプト
26: SYSTEM_PROMPT = """You are a good weather forecaster.
27: The answer is to be able to briefly explain the main points of the weather forecast in Japanese.
28: Output the weather summary in the following format:
29: Example:
30: 今日の東京は晴れです。温度は20度で、少し涼しいでしょう。
31: """
32:
33: # ------------------------------------------------------------
34: # LLM 初期化（最小構成）
35: # ------------------------------------------------------------
36:
37: def init_llm() -> Ollama:
38:     return Ollama(
39:         model=LLM_MODEL,
40:         base_url=OLLAMA_BASE_URL,
41:         request_timeout=120.0,
42:         temperature=0.1,
43:         stream=False,     # <ーstream を OFF
44:     )
45:
46: llm = init_llm()
47:
48: # ------------------------------------------------------------
49: # MCP ツール呼び出し用のシンプルなラッパ
50: # ------------------------------------------------------------
51: # - MCP のツールを Python 関数として扱うための簡略化レイヤ
52: # - FunctionTool から async_fn として呼ばれる
53: # ------------------------------------------------------------
54:
55: class MCPWrapper:
56:     def __init__(self, name: str):
57:         self.name = name      # MCP ツール名を保持
58:
59:     async def __call__(self, **kwargs) -> str:
60:         # LlamaIndex 側のラップ形式対応（kwargs の中に kwargs が入ってくる）
61:         args = kwargs.get("kwargs", kwargs)
62:
63:         # MCP ツール実行
64:         result = await _mcp.call_tool(self.name, args)     #<- MCPサーバのツールを実行
65:
66:         # 最低限のレスポンスパース
67:         if hasattr(result, "content") and result.content:
68:             c = result.content[0]
69:             return getattr(c, "text", str(c))
70:
71:         return str(result)
72:
73: # ------------------------------------------------------------
74: # MCP サーバー側のツールを FunctionTool に変換
75: # ------------------------------------------------------------
76:
77: async def load_tools() -> List[FunctionTool]:
78:     tools = await _mcp.list_tools()               #<-- MCP サーバからTool list 取得
79:     out = []
80:
81:     for t in tools:
82:         wrapper = MCPWrapper(t.name)              #< - MCPで取得した Ｔｏｏｌの情報を llamaIndex で利用できるように修
正
83:         tool = FunctionTool.from_defaults(        #<- ツールの登録
84:             async_fn=wrapper,
85:             async_fn=wrapper,
86:             name=t.name,                          #<- ツール名
87:             description=t.description,
```

```
        )
        out.append(tool)

    return out

# ------------------------------------------------------------
# ReAct エージェント実行（構造は維持しつつ簡潔に）
# ------------------------------------------------------------
# - MCP 接続
# - ツール読み込み
# - ReAct Agent 作成
# - クエリ実行
# ------------------------------------------------------------

async def run_agent(query: str):
    global _mcp

    async with Client(MCP_SERVER) as client:
        _mcp = client

        # MCP → FunctionTool 変換
        tools = await load_tools()                #<- mcpサーバとの接続関連の処理も登録

        # ReAct Agent 初期化
        # LlamaIndex の FunctionAgentをReAct Agent として 利用
        agent = FunctionAgent(
            tools=tools,
            llm=llm,
            system_prompt=SYSTEM_PROMPT,
            max_iterations=10,
        )

        # ReAct Agent 実行
        result = await agent.run(query)

        # 出力
        print("¥n--- 応答 ---")
        print(str(result))

# ------------------------------------------------------------
# メイン処理
# ------------------------------------------------------------

if __name__ == "__main__":
    query = sys.argv[1] if len(sys.argv) > 1 else DEFAULT_QUERY

    try:
        asyncio.run(run_agent(query))
    except KeyboardInterrupt:
        print("中断されました")
    except Exception as e:
        # 最低限のエラー表示
        print(f"エラー: {e}")
```