2025.06.01. chatGPT 답변 전재

---

1. MCP란 무엇인가?

정의

- MCP는 LLM(대형 언어 모델)·AI 에이전트를 외부 도구(툴)와 표준화된 방식으로 연결하기 위한 "도구 호출 프로토콜"입니다.

- 즉, LLM은 자연어 요청(예: "현재 날씨 알려줘")을 MCP 형식의 JSON 메시지로 변환하여 서버에 보내고, 서버는 해당 도구(예: 날씨 조회 API)를 실행한 뒤 결과를 다시 JSON으로 응답하는 구조를 가집니다.

목표

- LLM이 여러 서드파티 도구(파일·DB·API 등)를 일관된 방식으로 호출(Chain, 멀티 호출, 병렬 호출)
- 도구별 입출력 스펙(specification)을 JSON 스키마 형태로 정의해, 런타임 시점에 자동 검증 및 문서화
- 새로운 도구를 추가하거나 수정해도, LLM 쪽 코드를 고치지 않고 JSON 툴 스펙만 갱신하면 서비스를 확장 가능

---

2. MCP서버의 내부 동작 원리

MCP 서버는 크게 다음 네 가지 주요 컴포넌트로 구성됩니다.

(1) Transport Layer (전송 인터페이스)

stdio또는 **HTTP/WS**방식으로 입력 메시지를 수신

예를 들어, stdio모드에서는 LLM(클라이언트)이 표준 입출력(stdio)을 통해 JSON 요청을 보내고, 서버가 표준 출력으로 결과를 반환

HTTP/WS 모드에서는 REST API 엔드포인트나 WebSocket으로 메시지를 주고받을 수 있음

(2) Input Parser (요청 파싱 & 검증)

들어온 JSON 메시지를 파싱하여, tool이름, parameters등을 추출

JSON 스키마(JSON Schema)를 기반으로 파라미터 타입·필수 여부 등을 검증

만약 잘못된 파라미터가 들어오면, 에러 메시지를 JSON 형식으로 되돌려주어 클라이언트(LLM)가 재시도할 수 있도록 함

(3) Tool Registry & Invocation Engine

서버 코드 내 @mcp.tool()데코레이터를 사용해 "어떤 함수가 어떤 이름으로 호출될지"를 등록

각 툴(함수)마다 name, description, parameters(JSON 스키마) 정보를 docstring으로 명시해 두

면, 런타임 시 자동으로 툴 스펙이 등록됨
클라이언트의 요청이 들어오면, 해당 이름의 함수를 호출하고, 반환값을 JSON으로 직렬화하여 반환
(4) Response Formatter (결과 반환 & 에러 핸들링)
함수(툴)가 정상적으로 실행되면, 결과값을 JSON { "status": "success", "data": ... }형태로 응답
함수 실행 중 예외(Exception)가 발생하면, { "status": "error", "message": "에러 메시지" }형태로 반환하여 LLM이 후속 조치를 취하도록 함
위 네 단계를 통해, **LLM <-> MCP 서버**간에 “무엇을 어떻게 실행할지”가 JSON 메시지 하나로 명확히 규격화됩니다.

---

3. MCP 서버 만드는 법 (Python + FastMCP 예제)

아래 예시는 Python 환경에서 FastMCP 라이브러리를 사용해 ‘간단한 날씨 조회 서버’를 만드는 과정을 설명합니다.

3.1. 사전 준비

Python 3.12 이상 설치
가상환경(Virtualenv) 생성

```
python3 -m venv venv
source venv/bin/activate
```

uv(ASGI 런처) 설치
→ MCP 서버 실행 시 ASGI 방식으로 빠르게 띄우기 위해 필요합니다.

```
pip install uvicorn
``` :contentReference[oaicite:3]{index=3}
```

**FastMCP 라이브러리 설치**

```
pip install fastmcp
``` :contentReference[oaicite:4]{index=4}
```

3.2. 프로젝트 구조

```
mcp_server_example/
├── main.py
├── requirements.txt
└── tools/
    └── weather_tool.py
```

- main.py: MCP 서버 인스턴스를 생성하고, 툴 모듈(tool)을 import 하는 진입점
- tools/weather_tool.py: 실제로 호출될 "날씨 조회 함수"를 정의

3.3. 도구 정의 (Tool)

- tools/weather_tool.py파일을 생성하고, 다음 내용을 작성합니다.

```
# tools/weather_tool.py
import mcp    # fastmcp에서 제공하는 mcp 데코레이터·유틸

@mcp.tool()
def get_weather(city: str) -> dict:
    """
    name: get_weather
    description: 특정 도시(city)의 현재 날씨 정보를 반환합니다.
    parameters:
      type: object
      properties:
        city:
          type: string
          description: 조회할 도시 이름 (예: "Seoul")
      required: [city]
    """
    # (예시) 실서비스라면 OpenWeatherMap API 등을 호출해야 하지만,
    # 여기서는 샘플 데이터를 하드코딩하여 반환합니다.
    sample_data = {
        "Seoul": {"temp": 24, "condition": "맑음"},
        "Busan": {"temp": 27, "condition": "구름 많음"},
        "London": {"temp": 15, "condition": "비"}
    }
    return sample_data.get(city, {"temp": None, "condition": "정보 없음"})
```

- @mcp.tool()데코레이터를 사용하면, **get_weather**함수가 "툴"로 등록됩니다.
- 함수 docstring 내부에 name, description, parameters등의 JSON 스펙을 명시해야, 클라이언트가 어떤 입력값을 주면 어떤 출력값을 받을지 알 수 있습니다.

3.4. 서버 엔트리포인트 작성

main.py파일을 생성하고, 다음과 같이 작성합니다.

```
# main.py
import mcp
```

```
from tools.weather_tool import get_weather

def main():
    # FastMCP 인스턴스 생성 (기본 transport="stdio")
    server = mcp.FastMCP()
    # 이미 @mcp.tool()로 등록된 함수는 모듈 import 시 자동으로 등록됨
    print("Starting MCP server on stdio...")
    server.run(transport="stdio")  # 표준 입출력 기반으로 서버 실행

if __name__ == "__main__":
    main()
```

- mcp.FastMCP()로 서버 객체를 만들면, 프로젝트 내에서 @mcp.tool()이 붙은 함수들을 모두 자동으로 스캔하여 등록합니다.

- server.run(transport="stdio")을 호출하면, **표준 입출력(stdio)**방식으로 MCP 메시지를 주고받을 수 있는 상태로 대기합니다.

3.5. 서버 실행

- 터미널(가상환경 활성화 상태)에서 다음 명령어로 서버를 시작합니다.

```
python main.py
```

또는 ASGI 런처(uvicorn)를 쓸 수도 있습니다. 이 경우 main:server.app형식으로 지정하면, HTTP/WS 모드로 띄울 수 있습니다.

```
uvicorn main:server.app --host 0.0.0.0 --port 8000
```

→ 이 방법을 쓰면, 클라이언트가 HTTP POST 방식으로 JSON 요청을 보낼 때도 동일하게 동작합니다.

3.6. 클라이언트(LLM) 연동

- **Claude Desktop**등 MCP 지원 클라이언트(예: Cursor, Supabase-MCP-Client)에서 새 MCP 서버를 등록해야 합니다.

  macOS 기준: ~/Library/Application Support/Claude/claude_desktop_config.json파일 편집

```
jsonc

{
  "mcpServers": {
```

```
    "weather": {
      "command": "python",
      "args": ["/full/path/to/mcp_server_example/main.py"]
    }
  }
}
```

- Windows의 경우 %APPDATA%\Claude\claude_desktop_config.json처럼 경로가 다를 수 있습니다.
- 클라이언트에서 “weather”라는 이름으로 서버를 등록하면, LLM이 { "tool": "get_weather", "parameters": { "city": "Seoul" } }형태의 JSON을 보내면, 서버가 응답을 JSON으로 반환합니다.

---

4. MCP 서버 구성 요소별 세부 설명

툴 스펙(JSON 스키마) 정의
- 각 함수(도구)에 대해 name, description, parameters(type, properties, required) 등을 docstring으로 작성
- 이를 바탕으로, LLM이 “도구 목록”을 조회하거나, 파라미터 검증(validate) 시 이 스펙을 사용할 수 있습니다.

Transport 방식 선택

stdio
- 개발 · 테스트 용도로 간편. LLM(클라이언트)이 Python 프로세스를 직접 실행해 표준 입출력으로 메시지를 주고받음
단, 프로덕션 환경에서는 CPU 자원을 많이 쓰는 작업이나 다수 사용자 동시 연결 시 부적합
- HTTP/WS(ASGI)
uvicorn등 ASGI 서버를 이용해 HTTP POST/GET, WebSocket으로 요청 가능
클라우드에 배포하여 여러 클라이언트가 동시에 호출할 수 있음
인증·HTTPS 설정 등을 추가로 구성해야 안정적

에러 핸들링
- 파라미터가 스키마와 맞지 않을 때: 서버는 { "status": "error", "message": "잘못된 파라미터" } 반환
- 내부 예외 발생 시: { "status": "error", "message": "툴 실행 중 예외 발생: …" }형태로 반환해, LLM이 재시도하거나 대체 플로우를 선택할 수 있도록 함

멀티툴 호출 & 체이닝

- 클라이언트 쪽에서 JSON 배열 형태로 여러 툴을 한 번에 호출하거나, 한 툴의 결과를 다음 툴의 파라미터로 넘기는 체이닝 패턴을 적용
- FastMCP 라이브러리는 기본적으로 “싱글 콜(single-call)”, “체인(chained-call)”, “병렬 호출(parallel-call)”을 지원하며, 코드 레벨에서 추가 로직 없이 JSON 명세만 수정해 주면 동작

---

5. 요약