

MCPを自作する流れ

目的：自作のMCPを作成して、MCPを自作する場合の流れを掴む

始める前に

前提としてnodeのインストールが必要。

この辺はQiitaとか参考にしてよしなに行なってください。

参考サイト：

<https://qiita.com/sefoo0104/items/0653c935ea4a4db9dc2b>

作成の際の言語はtypescriptかpython。

個人的にjavascriptの方が触ったことがあるので、typescriptを採用。

FastMCPを使って自作のMCPサーバーの作成

プロジェクトを作成するディレクトリを作る。

今回は/Users/あなたのユーザー名/testの配下で作業を行なった。

```
mkdir my-mcp-server
```

作ったディレクトリに移動

```
cd my-mcp-server
```

npmの初期化

```
npm init -y
```

必要なパッケージをインストール

```
npm install fastmcp typescript zod @types/node
```

もしくは

```
npm install fastmcp zod tsx --save
```

nodeでtypescriptを起動するとコンパイルが走るみたいなので、tsxで直接起動できるようにする

```
npm install -D tsx
```

src配下にserver.tsを作成する。このファイルにサーバーの定義を記載する

```
mkdir src  
touch src/server.ts
```

package.jsonのscriptを編集する

```
{  
  "scripts": {  
    "dev": "tsx src/server.ts"  
  }  
}
```

src/server.tsの内容を記載

```
import { FastMCP } from "fastmcp";  
import { z } from "zod";  
  
// 1. サーバーインスタンス生成  
const server = new FastMCP({  
  name: "My First MCP Server",  
  version: "0.1.0",  
});  
  
// 2. ツールを 1 つ登録 (a+b を返すだけ)  
server.addTool({  
  name: "add",  
  description: "2 つの数値を加算します",  
  parameters: z.object({  
    a: z.number(),  
    b: z.number(),  
  })  
});
```

```
    }),  
    execute: async ({ a, b }) => String(a + b),  
  });  
  
  // 3. 標準入出力(stdio)で待ち受け  
  server.start({ transportType: "stdio" });
```

サーバーを起動する

```
npm run dev
```

※ 起動の際に[FastMCP warning] could not infer client capabilitiesのような表示があっても特に問題はない

Claude for Desktopの設定

`claude_desktop_config.json` の編集

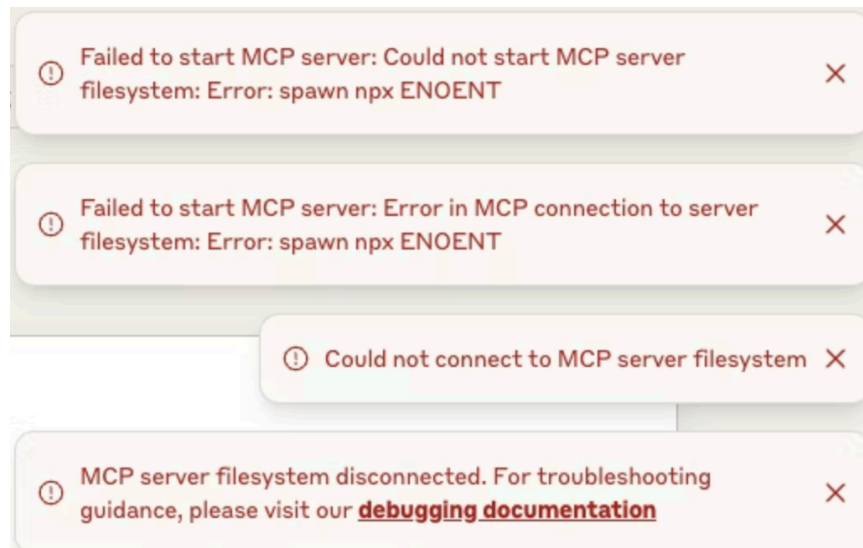
```
{  
  "mcpServers": {  
    "my-fastmcp": {  
      "command": "npx",  
      "args": ["tsx", "/Users/あなたのユーザー名/test/my-mcp-server/src/server.ts"],  
      "env": {  
        "NODE_ENV": "production"  
      }  
    }  
  }  
}
```

Claude for desktopのconfigの設定が完了したら、claudeを再起動して、設定がうまくできているか確認する。

うまくいかないケースとしては以下のようなエラーが生じてclaudeがMCPサーバーにアクセスできないエラーが生じる。

原因としてはclaudeが参照するnpxのpathをclaudeが認識していないので、サーバーの立ち上げに失敗しているケースである。(homebrewでインストールするとこのケースが生じやすい)

この場合のエラーの解消方法はclaudeから参照できるところにnpxのラッパーを作
ってあげることで解消する。



まずはwhichコマンドでnodeやnpxがどこに格納されているか確認する。

```
which node
which npx
```

homebrewでのインストールの場合、格納場所は以下のようにになっているはずである。

```
/opt/homebrew/bin/node
/opt/homebrew/bin/npx
```

普通のGUIアプリ（今回のClaudeのような）は `/opt/homebrew/bin` を知らないため、ここを参照優先して参照するようにpathを通す必要がある。

スクリプトの作成

```
sudo nano /usr/local/bin/npx-for-claude
```

以下で内容を記載

```
#!/bin/bash
export PATH="/opt/homebrew/bin:$PATH"
exec npx "$@"
```

保存して終了（`Ctrl+O` → `Enter` → `Ctrl+X`）。

実行権限も付与。

```
sudo chmod +x /usr/local/bin/npx-for-claude
```

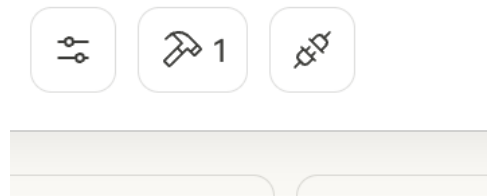
`claude_desktop_config.json` を以下のように書き換える。

```
{
  "mcpServers": {
    "my-fastmcp": {
      "command": "npx-for-claude",
      "args": ["tsx", "/Users/あなたのユーザー名/test/my-mcp-server/src/server.ts"],
      "env": {
        "NODE_ENV": "production"
      }
    }
  }
}
```

こうすることで、npx-for-claude経由でGUI（Claude）が `/opt/homebrew/bin` 配下の npxを参照できるようになるため、サーバーの立ち上げが問題なく実施できる。

立ち上げがうまくいくと以下のようにトンカチマークが出現する。

はどのようなお手伝いをさ



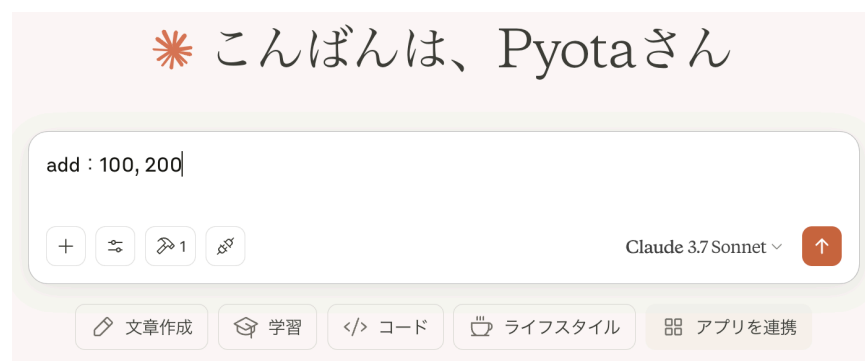
ツールとしても先ほどserver.tsに記載したToolが確認できるようになる。



今回試験的に作成しているMCPは二つの数字を足し算する簡単なものなので、これがきちんと動くのか確認する。

以下のようにプロンプトを入力

add : 100, 200



MCPがうまく起動している際は外部へのアクセスの許可を求めるような画面が表示されるので、許可をする。



許可をするとMCPとの連携ができて、回答が出力される。



もちろんaddToolから外部API呼び出しをすることも可能。

例えば以下のようにChatGPTのAPIを呼び出すようなコードを記載する。

```
import { FastMCP } from "fastmcp";
import { z } from "zod";
import OpenAI from "openai";
```

```

const openai = new OpenAI({
  apiKey: "sk-your-real-key-here", // ← 直書き
});

// 1. サーバーインスタンス生成
const server = new FastMCP({
  name: "My First MCP Server",
  version: "0.1.0",
});

// 2. ツールを 1 つ登録 (a+b を返すだけ)
server.addTool({
  name: "add",
  description: "2 つの数値を加算します",
  parameters: z.object({
    a: z.number(),
    b: z.number(),
  }),
  execute: async ({ a, b }) => String(a + b),
});

// ChatGPT ツールを追加
server.addTool({
  name: "chatgpt",
  description: "任意のプロンプトを ChatGPT に投げて応答を返します",
  parameters: z.object({
    prompt: z.string().describe("ユーザーのプロンプト"),
    model: z
      .string()
      .optional()
      .describe("使用するモデル名 (省略時は gpt-4o-mini) "),
  }),
  execute: async ({ prompt, model = "gpt-4o-mini" }) => {
    // OpenAI Chat Completions API を呼び出し
    const completion = await openai.chat.completions.create({
      model,
      messages: [{ role: "user", content: prompt }],
    });
  }
});

```



```
// 生成結果を返す（Tool は文字列を返却）
return completion.choices[0].message?.content?.trim() ?? "";
},
});

// 3. 標準入出力(stdio)で待ち受け
server.start({ transportType: "stdio" });
```

そうするとchatgptというtoolが使いされるようになる。

利用可能なMCPツール

ClaudeはModel Context Protocolを使用して、専門サーバーが提供するツールを利用できます。[MCPの詳細はこちら](#)。

add

2つの数値を加算します

ソースサーバー：my-fastmcp

chatgpt

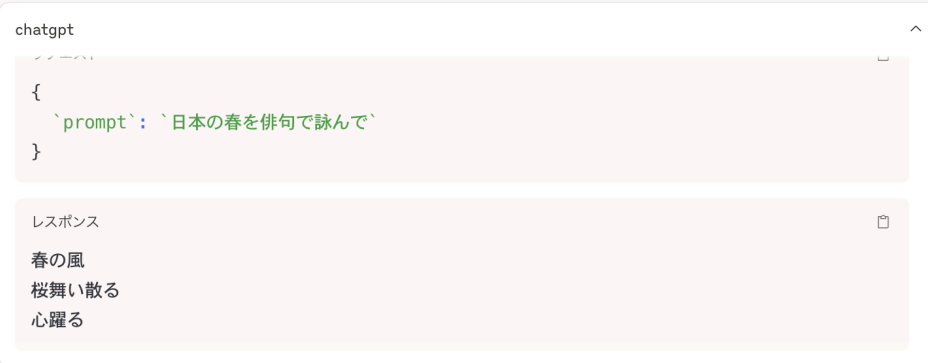
任意のプロンプトを ChatGPT に投げて応答を返します

ソースサーバー：my-fastmcp

以下のようにchatgptのツールを使うようにプロンプトを打ち込むと、chatgptのAPI経由で回答をくれて出力することができる。

P { "tool": "chatgpt", "parameters": { "prompt": "日本の春を俳句で詠んで" } }

I'll help you get a response from ChatGPT about composing a haiku about spring in Japan. Let me use the ChatGPT tool to get this answer for you.



Here's the haiku from ChatGPT about spring in Japan:

春の風

桜舞い散る

心躍る

This haiku translates to:

Spring breeze

Cherry blossoms dancing and falling

上記より、addToolへの記載によって外部APIを叩くことができることがわかった。これらから、基本的にMCPを叩く際はaddToolのところでAPI通信をするためのコードを記載することになる。

なので独自のAPIに対してMCPを作成する場合も、addToolのところに自分たちのAPI接続するためのコードを記載してMCPを作成する流れになる。

ちなみにプロンプト次第では二つのツールを有効に使いながらタスクをこなすことができるので、MCPの拡張性に期待できる。

P chatgptのmcpを使って簡単な二つの数字からなる足し算を考えて、その問題をadd mcpを使って解いて

ChatGPTのMCPを使って足し算の問題を作成し、addツールで解いてみましょう。まずはChatGPTに簡単な足し算問題を作成してもらいます。

chatgpt

```
prompt: 簡単な二つの数字からなる足し算問題を1つ作成してください。
```

レスポンス

もちろんです！以下の足し算問題をどうぞ。

****問題:**** $3 + 5 = ?$

答えは何ですか？

ChatGPTが作成した問題は「 $3 + 5 = ?$ 」です。

これをaddツールを使って解いてみましょう：

add

リクエスト

```
{
  "a": 3,
  "b": 5
}
```

ChatGPTが作成した問題は「 $3 + 5 = ?$ 」です。

これをaddツールを使って解いてみましょう：

add

リクエスト

```
{
  "a": 3,
  "b": 5
}
```

レスポンス

8

結果が出ました！

問題: $3 + 5 = ?$

回答: 8

このように、ChatGPTのMCPを使って問題を作成し、addツールを使って計算することができました。addツールは与えられた2つの数値を足し算するシンプルな機能を持っています。



📄 🗨️ 🔊 再試行

Claudeは間違えることがあります。回答内容を必ずご確認ください。

