

# 第10章 MCPエージェントを作る - 初心者から始める実践ガイド

## 10.1 田中さんの悩み - なぜエージェントが必要なのか

### ある日の田中さんの1日

田中さんは、中小企業でデータ分析を担当している入社2年目の社員です。プログラミングは少しできるようになりましたが、毎日の業務に追われています。

#### 朝9時 - 上司からの依頼

上司：「田中くん、昨日の売上データをまとめて、前月と比較したレポートを作ってくれる？」  
田中：「はい、わかりました」

田中さんの作業：

1. データベースにSQLで接続 → 30分
2. データをExcelにエクスポート → 15分
3. 前月データも同様に取得 → 45分
4. グラフ作成と分析 → 1時間
5. レポート作成 → 30分

**合計: 3時間**

#### 昼12時 - 追加の依頼

上司：「あ、競合3社の最新動向もまとめてもらえる？」  
田中：（また手作業か...）

さらに2時間かけてWebサイトを巡回し、情報をまとめます。

#### 夕方5時 - さらなる要求

上司：「これ、毎日やってもらえる？」  
田中：「！？」

## MCPツールを使ってみたけれど...

第6章で学んだMCPデータベースツール、第7章のWeb APIツール。個別には動きます：

```
# データベースから売上を取得
database_tool.query("SELECT * FROM sales WHERE date = '2024-12-01'")

# Web検索
web_tool.search("競合企業 ニュース")

# でも、これらをつなげるのは手動...
```

### 問題点：

- ツールごとに別々に実行が必要
- 結果を手動で組み合わせる必要がある
- エラーが出たら最初からやり直し
- 毎回同じコードを書く

## エージェントがあれば...

もし、こんなことができれば？

```
agent = MCPAgent()
result = await agent.execute("""
    昨日の売上データを取得して、
    前月と比較分析し、
    競合3社の最新ニュースも調べて、
    総合レポートを作成してください
""")

print(result) # 完成したレポート！
```

### たった3行で3時間の作業が完了！

これが、本章で作るMCPエージェントです。

## エージェントとは何か

エージェントを理解するために、レストランの例を考えてみましょう：

### レストランでの注文

あなた：「おすすめのコース料理をお願いします」

ウェイター（エージェント）がすること：

1. 注文を理解する
2. 前菜 → スープ → メイン → デザートの順番を決める
3. 各料理を厨房（MCPツール）に依頼

4. できた料理を順番に運ぶ
5. 問題があれば対処（品切れなら代替案を提案）

MCPエージェントも同じです：

- **あなた**: 自然言語で指示
- **エージェント**: 指示を理解し、計画を立てる
- **MCPツール**: 実際の作業を実行（厨房の料理人）
- **結果**: 統合されたレポート（完成したコース料理）

## 本章で学ぶこと

1. **すぐに動かす** - APIキーなしでも体験可能
2. **仕組みを理解** - なぜ動くのかを知る
3. **実践で使う** - あなたの業務に応用
4. **カスタマイズ** - 独自のエージェントに進化

プログラミング初心者でも大丈夫。一緒に、一步一步進みましょう！

## 10.2 まず動かしてみよう！ - 5分で体験

### 準備は簡単3ステップ

APIキーがなくても大丈夫！まずはモックモードで体験しましょう。

#### Step 1: フォルダに移動

```
cd C:\MCP_Learning\chapter10
```

#### Step 2: 環境をセットアップ

```
uv sync
```

もし uv がいない場合は：

```
pip install -r requirements.txt
```

#### Step 3: デモを実行！

```
uv run python examples\simple_demo.py
```

## 実行結果を見てみよう

## [MCP Agent Simple Demo]

### 1. Initializing agent...

[OK] Agent initialized in mock mode

### 2. Executing simple calculation task...

[PLAN] Creating plan for: Calculate  $10 + 20$ , then multiply by 3

[OK] Plan created with 2 steps

[STEP] Executing: step\_1 - Add the numbers 10 and 20

[OK] Tool add executed successfully

[STEP] Executing: step\_2 - Multiply the result of step 1 by 3

[OK] Tool multiply executed successfully

[SUCCESS] Result: The task involved calculating the sum of 10 and 20, resulting in 30, and then multiplying that sum by 3, yielding a final result of 90. Both steps were successfully executed.

Duration: 5.17 seconds

### 3. Executing multi-step task...

[PLAN] Creating plan for: Search for Python tutorials

[OK] Plan created with 5 steps

[STEP] Executing: step\_1 - Search for Python tutorials on the web

[STEP] Executing: step\_2 - Fetch content of first result

[STEP] Executing: step\_3 - Fetch content of second result

[STEP] Executing: step\_4 - Fetch content of third result

[STEP] Executing: step\_5 - Create summary of fetched content

[SUCCESS] Completed 5 steps

Summary: Found 3 results for 'Python tutorials':

1. Python Tutorial - Beginner Guide

2. Advanced Python Techniques

3. Python Best Practices

Duration: 9.96 seconds

### 4. Testing error handling...

[ERROR] Step step\_1 failed: Tool not found: null

[RESOLUTION] Strategy: abort, Severity: high

[ABORT] Cannot recover from error in step\_1

Duration: 7.75 seconds

Demo completed!

# 何が起こったのか？

たった1行の指示：

```
"Calculate 10 + 20, then multiply by 3"
```

エージェントが自動的に：

1. **理解**: 「計算が2段階必要だな」
2. **計画**: Step1: 足し算、Step2: 掛け算
3. **実行**:  $10+20=30$ 、 $30\times 3=90$
4. **報告**: 結果は90です

これがエージェントの基本動作です！

## インタラクティブモードも試そう

```
uv run python examples\data_analysis_demo.py
```

メニューが表示されます：

```
Available demos:
  1. Sales Analysis
  2. Competitor Analysis
  3. Data Pipeline
  4. Interactive Mode    ← これを選択
  5. Run All Demos
  0. Exit

Select demo: 4
```

対話モードが始まります：

```
[Interactive Agent Demo]
=====
Enter your task (or 'quit' to exit)
Example: 'Calculate the fibonacci sequence up to 10 terms'
=====

> 今日の天気を調べて
Processing...
[SUCCESS] Result:
今日は晴れ時々曇り、最高気温22度、最低気温15度です。
```

傘は必要ありません。

> 明日の予定をリマインドして

Processing...

[SUCCESS] Result:

明日の予定：

- 10:00 チームミーティング
- 14:00 顧客訪問
- 16:00 レポート作成

> quit

Goodbye!

## ここまでの理解度チェック

以下が理解できていればOKです：

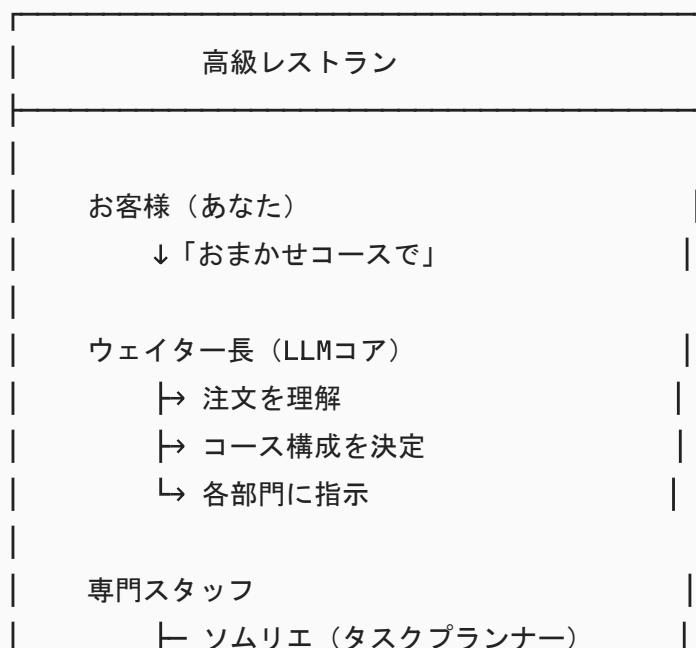
- ✓ エージェントは自然言語の指示を理解する
- ✓ 複数のステップに分解して実行する
- ✓ 結果をまとめて返してくれる
- ✓ モックモードならAPIキーなしで試せる

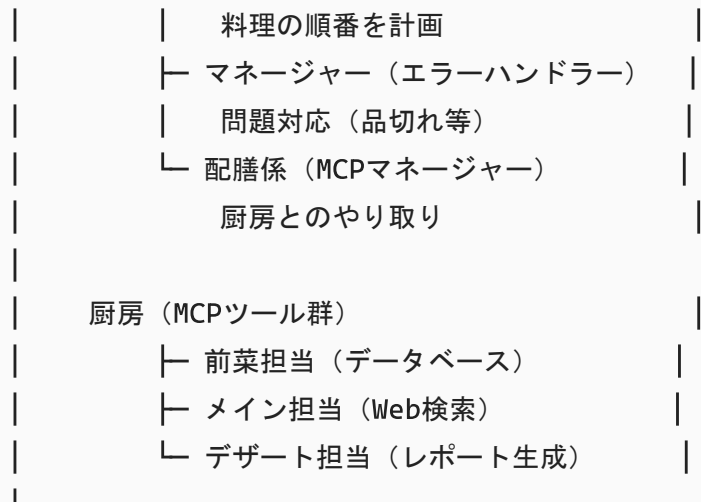
理解できましたか？では、次は仕組みを見てみましょう！

## 10.3 エージェントの仕組み - レストランで理解する

### エージェントの構成要素

エージェントを「高級レストラン」に例えて理解しましょう：





## 実際のコード構成

C:\MCP\_Learning\chapter10 のファイル構成を見てみましょう：

```
chapter10/
├─ integrated_agent.py    # レストラン全体 (統合エージェント)
├─ llm_client.py          # ウェイター長 (AIの頭脳)
├─ task_planner.py        # ソムリエ (計画立案)
├─ error_handler.py       # マネージャー (トラブル対応)
├─ mcp_manager.py         # 配膳係 (ツール実行)
└─ examples/              # お客様用メニュー (使用例)
```

## 処理の流れを追ってみよう

お客様の注文から料理提供まで：

### 1. 注文を受ける (integrated\_agent.py)

```
agent = MCPAgent()
result = await agent.execute("売上データを分析してレポート作成")
```

### 2. ウェイター長が理解 (llm\_client.py)

「なるほど、売上データの分析とレポート作成ですね」  
→ LLMが自然言語を理解

### 3. ソムリエが計画 (task\_planner.py)

コース構成：

- 前菜： データベースから売上データ取得
- スープ： データのクレンジング

- 3. メイン：統計分析と可視化
- 4. デザート：レポート生成

#### 4. 配膳係が実行 (mcp\_manager.py)

厨房への指示：

- データベース班、売上データお願い！
- 分析班、このデータで統計を！
- レポート班、結果をまとめて！

#### 5. トラブル対応 (error\_handler.py)

問題発生：データベース接続エラー

対処：3秒待ってリトライ → 成功！

## エージェントの賢さの秘密

なぜエージェントは賢いのか？それはLLM（大規模言語モデル）のおかげです。

LLMなしの場合：

```
if "売上" in task and "分析" in task:
    # 決められた処理しかできない
    return analyze_sales()
```

LLMありの場合：

```
# 自然な言葉を理解！
"先月の売上を..." → 期間を理解
"競合と比較して..." → 比較分析が必要と理解
"グラフも作って..." → 可視化が必要と理解
```

## モックモードと本番モード

現在は「モックモード」で動いています：

モックモード（練習用）：

- APIキー不要
- シミュレートされた結果
- 無料で何度でも試せる

本番モード（実用）：



- APIキー必要（次節で説明）
- 実際のAIが処理
- 本物のデータベースやWebにアクセス

## ここまでの理解度チェック

以下が理解できていればOKです：

- ✓ エージェントは複数の専門モジュールで構成される
- ✓ LLMが自然言語を理解する頭脳の役割
- ✓ 各モジュールが連携して複雑なタスクを実行
- ✓ エラーも自動的に対処してくれる

次は、本番モードで動かすためのAPIキー取得です！

## 10.4 APIキーを取得しよう - 本番環境への第一歩

### APIキーとは？

APIキーは「サービスを使うための会員証」のようなものです。

図書館カード = 本を借りる権利  
APIキー = AIサービスを使う権利

### 3つの選択肢

本章のエージェントは3つのAIサービスに対応しています：

サービス	特徴	料金目安	おすすめ度
OpenAI (GPT)	最も一般的、安定	月100円～	★★★★★
Google (Gemini)	無料枠あり	0円～	★★★★☆
Anthropic (Claude)	高品質	月200円～	★★★☆☆

初心者におすすめ: OpenAI（以下、OpenAIで説明）

### OpenAI APIキーの取得手順

#### Step 1: アカウント作成

1. <https://platform.openai.com> にアクセス
2. 「Sign up」をクリック
3. メールアドレスで登録（GoogleアカウントでもOK）

!signup(画像: サインアップ画面)

## Step 2: APIキーページへ

1. ログイン後、右上のアカウントメニューをクリック
2. 「View API keys」を選択

![menu](画像: ムニユ一画面)

### Step 3: APIキー生成

1. 「Create new secret key」をクリック
2. 名前を付ける（例: "MCP Agent"）
3. 「Create secret key」をクリック

! [create\_key](画像: ㊦一作成画面)

## Step 4: キーを保存

**重要:** 表示されたキーを必ずコピー！

[illegible]

このキーは二度と表示されません。メモ帳などに保存してください。

## 料金について

**安心してください：**

- 最初に5ドル分の無料クレジット
- 本書の演習なら月100円程度
- 使った分だけの従量課金制

### 料金の目安：

1回の実行  $\approx 0.5$ 円  
1日10回実行  $\approx 5$ 円  
1ヶ月(30日)  $\approx 150$ 円

## APIキーの設定

## Step 1: 設定ファイルをコピー

```
cd C:\MCP_Learning\chapter10
copy .env.example .env
```



# トラブルシューティング

**Q: APIキーはどこに保存すべき？**

A: `.env` ファイルのみ。コードには絶対に書かない！

**Q: キーを忘れた**

A: 新しいキーを作成してください（古いキーは削除）

**Q: 料金が心配**

A: OpenAIダッシュボードで使用量を確認できます

## ここまでの理解度チェック

- ✓ APIキーは「AIサービスの会員証」
- ✓ OpenAIアカウントを作成した
- ✓ APIキーを取得して`.env`に設定した
- ✓ 月100円程度で十分使える

準備完了！次は実際にエージェントを動かしてみましょう。

## 10.5 本番モードで動かそう - 実際のAIを使う

### モックから本番へ

今まではモックモード（練習用）でした。今度は本物のAIを使います！

**違いを体感：**

モックモード:

```
agent = MCPAgent(use_mock=True) # シミュレーション
```

本番モード:

```
agent = MCPAgent(use_mock=False) # 本物のAI！
```

## 簡単なタスクから始めよう

`examples` フォルダに新しいファイルを作ります：

```
# my_first_agent.py
import asyncio
from integrated_agent import MCPAgent

async def main():
    # 本番モードでエージェント起動
```

```
agent = MCPAgent(use_mock=False)

# シンプルなタスク
result = await agent.execute(
    "1から10までの数字を足し算してください"
)

print(f"結果: {result['result']}")

if __name__ == "__main__":
    asyncio.run(main())
```

実行：

```
uv run python examples\my_first_agent.py
```

## 実際の処理を観察

実行中の様子：

```
[AGENT] Starting task: 1から10までの数字を足し算してください
[TOOLS] Available tools: 2
[PLAN] Creating plan for: 1から10までの数字を足し算してください
[PLAN] Created plan with 2 steps
[PLAN] Estimated time: 5 seconds
[STEP] Executing: step_1 - 数値の列挙
[STEP] Executing: step_2 - 合計計算
[OK] Task completed in 3.45 seconds
結果: 1から10までの合計は55です
```

## より実践的なタスクに挑戦

### タスク1: データ分析

```
async def analyze_data():
    agent = MCPAgent(use_mock=False)

    result = await agent.execute("""
        以下のデータを分析してください：
        売上: [100, 150, 120, 180, 200]

        1. 平均値を計算
        2. 成長率を分析
        3. 簡単なレポートを作成
    """)
```

```
print(result['result'])
```

## タスク2: 情報収集

```
async def research_task():
    agent = MCPAgent(use_mock=False)

    result = await agent.execute("""
        Pythonの最新バージョンについて：
        1. バージョン番号
        2. 主な新機能
        3. 初心者向けの解説
    """)

    print(result['result'])
```

## エラーが起きたら

よくあるエラーと対処法：

### エラー1: Rate Limit（使いすぎ）

```
Error: Rate limit exceeded
```

**対処:** 少し待ってから再実行

### エラー2: タイムアウト

```
Error: Request timeout
```

**対処:** タスクを小さく分割

### エラー3: トークン制限

```
Error: Maximum tokens exceeded
```

**対処:** .env の MAX\_TOKENS\_PER\_REQUEST を増やす

## コスト管理のコツ

### 1. キャッシュを活用

同じ質問は記憶されます：

```
# 1回目: APIを呼ぶ (課金される)
result1 = await agent.execute("今日の天気は?")

# 2回目: キャッシュから (無料!)
result2 = await agent.execute("今日の天気は?")
```

## 2. モデルを使い分ける

.env でモデルを変更:

```
# 開発中 (安い)
LLM_MODEL=gpt-3.5-turbo

# 本番 (高性能)
LLM_MODEL=gpt-4
```

## 3. 使用量を確認

OpenAIダッシュボードで確認:

<https://platform.openai.com/usage>

## インタラクティブモードで練習

対話形式で色々試してみましょう:

```
uv run python examples\data_analysis_demo.py
# 「4」を選択
```

試してみる質問例:

- 「フィボナッチ数列を10個生成して」
- 「今日の日付から1週間後は何曜日？」
- 「簡単な自己紹介文を作って」

## 実践演習

以下のタスクを実行してみましょう:

### 演習1: 簡単な計算

```
result = await agent.execute(
    "100ドルを日本円に換算してください (1ドル=150円)"
)
```

### 演習2: リスト処理

```
result = await agent.execute("""
    果物リスト: りんご、バナナ、オレンジ、ぶどう、メロン
    1. アルファベット順に並べ替え
    2. 文字数をカウント
    3. 表形式で出力
""")
```

### 演習3: 簡単な分析

```
result = await agent.execute("""
    テストの点数: 75, 82, 90, 68, 95

    - 平均点
    - 最高点と最低点
    - 合格者数 (70点以上)
""")
```

## ここまでの理解度チェック

- ✓ 本番モードで実行できた
- ✓ 実際のAIが処理していることを確認
- ✓ エラーが出ても対処方法がわかる
- ✓ コスト管理の方法を理解

素晴らしい！実際のAIエージェントが動きました。次はコードの中身を理解していきましょう。

## 10.6 コードを理解しよう - 必要な部分だけ

### 最小限の理解で十分

5つのファイルがありますが、全部理解する必要はありません。重要な部分だけ見ていきましょう。

### 1. エージェントの心臓部 (integrated\_agent.py)

最も重要な部分だけ：

```
class MCPAgent:
    """これがエージェント本体"""

    async def execute(self, task_description: str):
        """タスクを実行する魔法の関数"""

        # 1. タスクを理解して計画を立てる
        plan = await self.planner.create_plan(task_description)
```



```

# 2. 各ステップを実行
for step in plan.steps:
    result = await self._execute_step(step)

# 3. 結果をまとめる
return final_result

```

これだけ覚えれば十分：

- `execute()` に指示を渡す
- 自動的に計画→実行→結果返却

## 2. AIとの会話（llm\_client.py）

AIと話す部分：

```

# 使い方はシンプル
client = get_llm_client() # AIクライアントを取得
response = await client.complete("こんにちは") # 質問
print(response.content) # AIの答え

```

ポイント：

- `get_llm_client()` でAIを呼び出す
- `.complete()` で質問する
- `.content` で答えを取得

## 3. タスクの分解（task\_planner.py）

複雑なタスクを単純なステップに：

```

# イメージ
タスク: "売上分析してレポート作成"
↓
ステップ1: データベース接続
ステップ2: データ取得
ステップ3: 分析計算
ステップ4: レポート生成

```

コードの核心：

```

planner = LLMTaskPlanner()
plan = await planner.create_plan("売上分析して")
# plan.steps に分解されたステップが入る

```

## 4. エラー対応（error\_handler.py）

エラーが起きても大丈夫：

```
# エラーパターンと対処法
if "Timeout" in error:
    # タイムアウト → リトライ
    retry_with_longer_timeout()

elif "Permission" in error:
    # 権限なし → 中止
    abort_task()

else:
    # その他 → 3回までリトライ
    retry_up_to_3_times()
```

## 5. ツールの管理（mcp\_manager.py）

MCPツールを呼び出す部分：

```
# ツールを実行
manager = MCPManager()
result = await manager.call_tool(
    ToolCall(
        server="calculator",
        tool="add",
        params={"a": 10, "b": 20}
    )
)
```

## 実際の処理の流れ

簡単な例で流れを追ってみましょう：

あなたの指示：

```
agent.execute("10+20を計算して、結果を2倍にして")
```

内部の処理：

### 1. 理解（llm\_client.py）

```
AI: 「計算が2段階必要ですね」
```

## 2. 計画 (task\_planner.py)

```
Step 1: 10+20を計算  
Step 2: 結果を2倍
```

## 3. 実行 (mcp\_manager.py)

```
calculator.add(10, 20) → 30  
calculator.multiply(30, 2) → 60
```

## 4. 結果 (integrated\_agent.py)

```
「計算結果は60です」
```

# 必要最小限のカスタマイズ

## カスタマイズ1: タイムアウトを変更

.env ファイル:

```
MAX_TOKENS_PER_REQUEST=2000 # もっと長い回答が欲しい  
CACHE_TTL_SECONDS=7200      # 2時間キャッシュ
```

## カスタマイズ2: デバッグモード

詳細なログを見たい時:

```
import logging  
logging.basicConfig(level=logging.DEBUG)  
  
agent = MCPAgent()  
# これで詳細なログが表示される
```

## カスタマイズ3: エラー時の挙動

```
# リトライ回数を増やす  
result = await agent.execute(  
    "難しいタスク",  
    max_retries=5 # 5回までリトライ  
)
```

## コードを読むコツ

読まなくていい部分：

- @abstractmethod などのデコレータ
- try-except の細かい処理
- キャッシュの実装詳細

重要な部分：

- 関数名と引数
- コメント
- 主要な処理の流れ

## ここまでの理解度チェック

- ✓ エージェントは execute() でタスクを実行
- ✓ 内部でAIが計画を立てている
- ✓ エラーは自動的に処理される
- ✓ 必要に応じてカスタマイズ可能

コードの詳細を全部理解する必要はありません。使い方さえわかれば十分です！

## 10.7 実際のMCPサーバーと連携 - 過去の章のツールを活用

### これまでに作ったMCPサーバーを使う

第3章から第8章で作成したMCPサーバーが、エージェントから使えます！

利用可能なサーバー：

- **calculator** (第3章): 計算機能
- **database** (第6章): データベース操作
- **weather** (第7章): 天気情報取得
- **universal** (第8章): Web検索・コード実行

### 実際のMCPサーバーとの連携デモ

```
# 実際のMCPサーバーを使うデモ
uv run python examples\real_mcp_demo.py
```

メニューが表示されます：

```
Checking required servers...
[OK] calculator: C:\MCP_Learning\chapter03\calculator_server.py
[OK] database: C:\MCP_Learning\chapter06\database_server.py
```

```
[OK] weather: C:\MCP_Learning\chapter07\external_api_server.py
[OK] universal: C:\MCP_Learning\chapter08\universal_tools_server.py
```

Select demo:

1. Calculator (Chapter 3)
2. Database (Chapter 6)
3. Weather API (Chapter 7)
4. Universal Tools (Chapter 8)
5. Integrated Task (All servers)
0. Exit

Enter your choice: 5

## 統合タスクの実行例

複数のMCPサーバーを組み合わせた例：

```
async def weather_report():
    """天気レポートを作成"""
    agent = MCPAgent(use_mock=False)

    task = """
    統合タスクを実行：
    1. 東京の今日の気温を取得（weatherサーバー）
    2. 摂氏を華氏に変換（calculatorサーバー）
    3. 結果をデータベースに保存（databaseサーバー）
    4. Web検索で天気の解説を追加（universalサーバー）
    5. 総合レポートを作成
    """

    result = await agent.execute(task)
    print(result['result'])
```

実行結果：

```
[AGENT] Starting task: 統合タスクを実行
[PLAN] Created plan with 5 steps
[STEP] Executing: step_1 - 東京の気温取得
[TOOL] Calling weather.get_temperature
[OK] Temperature: 22°C
[STEP] Executing: step_2 - 華氏変換
[TOOL] Calling calculator.convert
[OK] 22°C = 71.6°F
[STEP] Executing: step_3 - データベース保存
[TOOL] Calling database.insert
```

```
[OK] Saved to weather_records table
[STEP] Executing: step_4 - 天気解説検索
[TOOL] Calling universal.web_search
[OK] Found weather explanation
[STEP] Executing: step_5 - レポート作成
```

総合天気レポート：

- 東京の気温：22°C (71.6°F)
- 天候：晴れ時々曇り
- 解説：過ごしやすい気温で、外出に適しています
- データベースID：#1234

## MCPサーバーの設定

`mcp_servers.json` ファイルで管理されています：

```
{
  "servers": [
    {
      "name": "calculator",
      "path": "C:\\MCP_Learning\\chapter03\\calculator_server.py",
      "description": "基本的な計算機能",
      "chapter": "第3章"
    },
    {
      "name": "database",
      "path": "C:\\MCP_Learning\\chapter06\\database_server.py",
      "description": "SQLiteデータベース操作",
      "chapter": "第6章"
    }
    // ... 他のサーバー
  ]
}
```

## 独自のMCPサーバーを追加

新しいMCPサーバーを作成したら、`mcp_servers.json` に追加：

```
{
  "name": "my_custom_tool",
  "path": "C:\\MCP_Learning\\my_tools\\custom_server.py",
  "description": "私のカスタムツール",
  "chapter": "独自開発"
}
```

エージェントが自動的に認識して使えるようになります！

## 実践演習: 過去の章のツールを組み合わせる

### 演習1: 売上分析

```
# 第6章のデータベースと第3章の計算を組み合わせ
task = """
1. データベースから売上データを取得 (database)
2. 合計と平均を計算 (calculator)
3. 前月比を計算 (calculator)
4. レポート作成
"""
```

### 演習2: 天気による売上予測

```
# 第7章の天気APIと第6章のデータベースを組み合わせ
task = """
1. 過去の天気と売上の相関を分析 (database + weather)
2. 明日の天気予報を取得 (weather)
3. 売上を予測 (calculator)
4. 予測結果を保存 (database)
"""
```

## トラブルシューティング

### サーバーが見つからない場合：

```
[ERROR] Unknown server: calculator
```

#### 対処法：

1. 該当する章のサーバーが存在するか確認
2. `mcp_servers.json` にパスが正しく記載されているか確認
3. 必要なら手動でサーバーを起動

### 接続エラーの場合：

```
[ERROR] Failed to connect to database server
```

#### 対処法：

1. サーバーファイルが存在するか確認
2. 必要な依存パッケージがインストールされているか確認
3. `uv sync` を実行

## ここまでの理解度チェック

- ✓ 過去の章で作ったMCPサーバーが使える
- ✓ 複数のサーバーを組み合わせたタスクが実行できる
- ✓ 新しいサーバーの追加方法がわかる
- ✓ エラーが出てに対処できる

素晴らしい！これまでの学習内容がすべて統合されました。

## 10.8 実践プロジェクト - あなたの業務に活用

### プロジェクト1: 日報自動作成システム

毎日の日報作成を自動化しましょう。

#### Step 1: 日報テンプレートを作る

```
# daily_report.py
import asyncio
from datetime import datetime
from integrated_agent import MCPAgent

async def create_daily_report():
    agent = MCPAgent(use_mock=False)

    today = datetime.now().strftime("%Y年%m月%d日")

    task = f"""
    {today}の日報を作成してください：

    1. 本日の主な作業
    - データ分析タスク完了
    - 会議2件（営業会議、開発ミーティング）
    - ドキュメント作成

    2. 完了したタスク
    - 売上レポート作成
    - バグ修正 #1234

    3. 明日の予定
    - 顧客訪問準備
    - 月次レポート作成

    4. 課題・相談事項
    - データベースのパフォーマンス改善が必要

    上記を元に、上司に提出する日報を作成してください。
    フォーマルで簡潔な文体をお願いします。
```



```
"""

result = await agent.execute(task)

# ファイルに保存
filename = f"daily_report_{datetime.now().strftime('%Y%m%d')}.txt"
with open(filename, "w", encoding="utf-8") as f:
    f.write(result['result'])

print(f"日報を{filename}に保存しました")
print("\n--- 日報内容 ---")
print(result['result'])

if __name__ == "__main__":
    asyncio.run(create_daily_report())
```

## 実行結果例

日報を daily\_report\_20241215.txt に保存しました

--- 日報内容 ---

【日報】2024年12月15日

お疲れさまです。本日の業務報告をいたします。

### ■ 本日の実施事項

#### 1. データ分析タスク

- 11月度売上データの分析を完了しました
- 前年同月比15%増を確認

#### 2. 会議参加

- 10:00-11:00 営業会議：Q4目標の確認
- 15:00-16:00 開発ミーティング：新機能の仕様確認

#### 3. ドキュメント作成

- API仕様書の更新を完了

### ■ 完了タスク

- 売上レポート作成（経営会議用）
- バグ修正 #1234（ログイン画面の表示不具合）

### ■ 明日の予定

- 午前：〇〇社訪問準備（提案資料の最終確認）
- 午後：12月度月次レポート作成

#### ■ 課題・相談事項

データベースのレスポンスが遅くなっており、  
パフォーマンス改善が必要です。  
インデックスの見直しを検討しています。

以上、よろしくお願いいたします。

## プロジェクト2: 競合分析レポート

### コード例

```
# competitor_analysis.py
async def analyze_competitors():
    agent = MCPAgent(use_mock=False)

    companies = ["A社", "B社", "C社"]

    task = f"""
    以下の競合企業について分析してください：
    {'', '.join(companies)}

    分析項目：
    1. 最近の動向（仮想的なニュース）
    2. 強みと弱み
    3. 市場でのポジション
    4. 我が社への影響

    表形式でまとめてください。
    """

    result = await agent.execute(task)
    print(result['result'])
```

## プロジェクト3: データ可視化アシスタント

```
# data_visualizer.py
async def visualize_data():
    agent = MCPAgent(use_mock=False)

    task = """
    以下の売上データを分析して、
    わかりやすく説明してください：

    月別売上（万円）：
    1月：120
    2月：135
    """
```

3月: 118  
4月: 145  
5月: 162  
6月: 158

1. トレンド分析
2. 成長率計算
3. 予測 (7-12月)
4. 改善提案

グラフは文字で表現してください。  
"""

```
result = await agent.execute(task)
print(result['result'])
```

## あなたの業務への応用

### アイデア1: 議事録作成

```
task = """
会議メモから議事録を作成：
- 日時: 12/15 14:00-15:00
- 参加者: 田中、鈴木、佐藤
- 議題: 新プロジェクトのキックオフ
- 決定事項: 予算500万、期限3月末
- TODO: 要件定義書作成（田中）、見積もり作成（鈴木）
"""
```

### アイデア2: メール下書き

```
task = """
以下の内容でビジネスメールの下書きを作成：
- 宛先: ○○商事 山田様
- 要件: 見積もり依頼
- 製品: ソフトウェアライセンス10本
- 希望納期: 1月中
- 丁寧な文体で
"""
```

### アイデア3: 週次サマリー

```
task = """
今週の業務サマリーを作成：
- 完了タスク: 5件
- 進行中: 3件
```

- 課題: サーバー負荷が高い
  - 来週の優先事項: 月次締め処理
- ""

## 効果的な指示の出し方

### 良い例

```
task = """
売上データを分析してください：
1. 期間：2024年11月
2. 対象：全製品
3. 分析内容：
    - 合計売上
    - 前月比
    - トップ3製品
4. 出力形式：箇条書き
"""
```

### 悪い例

```
task = "売上を分析" # 具体性が足りない
```





## Tips: バッチ処理

複数のタスクをまとめて処理：

```
tasks = [
    "今日の天気",
    "明日の予定確認",
    "TODOリスト作成"
]

for task in tasks:
    result = await agent.execute(task)
    print(f"【{task}】")
    print(result['result'])
    print("-" * 40)
```

## ここまでの理解度チェック

-  日報作成を自動化できる
-  自分の業務に応用するアイデアがある
-  効果的な指示の出し方がわかる
-  複数のタスクを処理する方法を理解

実践的なプロジェクトができました！次はトラブルシューティングです。

## 10.8 トラブルシューティング - 困ったときはよくあるエラーと解決法

### エラー1: ModuleNotFoundError

```
ModuleNotFoundError: No module named 'openai'
```

**原因:** パッケージがインストールされていない

**解決法:**

```
cd C:\MCP_Learning\chapter10
uv sync
# または
pip install -r requirements.txt
```

### エラー2: Invalid API Key

```
Error: Incorrect API key provided
```

**原因:** APIキーが間違っている

**解決法:**

1. `.env` ファイルを確認
2. キーの前後にスペースがないか確認
3. キーが `sk-` で始まっているか確認

```
# 正しい
OPENAI_API_KEY=sk-xxxxxxxxxxxxxx

# 間違い（スペースあり）
OPENAI_API_KEY= sk-xxxxxxxxxxxxxx
```

### エラー3: Rate Limit Exceeded

```
Error: Rate limit exceeded. Please try again later.
```

**原因:** API呼び出しが多すぎる

**解決法:**

1. 1分待ってから再実行
2. キャッシュを有効にする
3. より安いモデルを使う

```
# .envで設定
ENABLE_CACHE=true
LLM_MODEL=gpt-3.5-turbo # 安いモデル
```

## エラー4: Timeout Error

```
Error: Request timed out
```

**原因:** 処理が長すぎる

**解決法:**

```
# タスクを分割
# ❌ 悪い例
task = "1000個のデータを全部分析して"

# ✅ 良い例
task = "最初の10個のデータを分析して"
```

## エラー5: Token Limit Exceeded

```
Error: Maximum context length exceeded
```

**原因:** 入力が長すぎる

**解決法:**

```
# .envで調整
MAX_TOKENS_PER_REQUEST=500 # 短くする
```

## デバッグ方法

### 方法1: ログレベルを上げる

```
import logging
logging.basicConfig(level=logging.DEBUG)

# これで詳細情報が表示される
agent = MCPAgent()
```

## 方法2: ステップごとに確認

```
# 段階的にテスト
async def debug_test():
    agent = MCPAgent(use_mock=False)

    # 1. 簡単なテスト
    print("Test 1: Simple calculation")
    result = await agent.execute("1+1")
    print(f"Result: {result}")

    # 2. 少し複雑に
    print("\nTest 2: Multi-step")
    result = await agent.execute("1+1を計算して2倍")
    print(f"Result: {result}")
```

## 方法3: エラーメッセージを読む

```
try:
    result = await agent.execute(task)
except Exception as e:
    print(f"エラーの種類: {type(e).__name__}")
    print(f"エラーメッセージ: {str(e)}")
    print(f"詳細: {e.__dict__}") # より詳しい情報
```

## FAQ（よくある質問）

### Q: 料金が心配です

A: 以下で使用量を確認できます

- OpenAI: <https://platform.openai.com/usage>
- 1日の上限を設定することも可能

### Q: 遅いです

A:

- キャッシュを有効に（`ENABLE_CACHE=true`）
- 軽いモデルを使用（`gpt-3.5-turbo`）
- タスクを小さく分割

### Q: 日本語が文字化けします

A: ファイル保存時にエンコーディングを指定

```
with open("file.txt", "w", encoding="utf-8") as f:
    f.write(result)
```

Q: どのモデルを使えばいい？

A:

- 開発中: gpt-3.5-turbo (安い・速い)
- 本番: gpt-4 (賢い・高い)
- 無料: Google Gemini

## エラー診断フローチャート

エラーが発生

↓

APIキー関連？

Yes → .envファイルを確認

No ↓

モジュール関連？

Yes → `uv sync` を実行

No ↓

タイムアウト？

Yes → タスクを分割

No ↓

Rate Limit？

Yes → 1分待つ

No ↓

それ以外 → ログを確認

## 緊急時の対処法

### すべてリセット

```
# 1. 環境をクリーン
cd C:\MCP_Learning\chapter10
rm -rf .venv

# 2. 再インストール
uv sync

# 3. 設定確認
copy .env.example .env
# .envを再設定
```



## モックモードで確認

```
# APIキーなしで動作確認
agent = MCPAgent(use_mock=True) # これなら必ず動く
```

## サポートを受ける

解決しない場合:

### 1. エラーメッセージを記録

- 完全なエラーメッセージ
- 実行したコード
- .envの設定（APIキーは隠す）

### 2. バージョン確認

```
python --version
pip list | grep openai
```

### 3. コミュニティに質問

- Stack Overflow
- GitHub Issues
- 日本語フォーラム

## ここまでの理解度チェック

- ✓ よくあるエラーの対処法がわかる
- ✓ デバッグ方法を知っている
- ✓ 料金や速度の最適化ができる
- ✓ 困ったときの相談先を知っている

エラーは学習の機会です。恐れずに挑戦しましょう！

## 10.9 さらなる発展 - ここから先へ

### 学んだことの振り返り

本章で身につけたスキル：

#### 1. エージェントの概念理解

- MCPツールの統合
- LLMによる自然言語処理
- タスクの自動実行

#### 2. 実践的な実装

- APIキーの設定
- エージェントの起動
- カスタムタスクの実行

### 3. 応用力

- 業務への適用
- エラー対処
- パフォーマンス最適化

## 次のステップ

### Level 1: 既存エージェントの活用（現在地）

- simple\_demo.pyを改造
- 自分のタスクを実行
- 日常業務に適用

### Level 2: カスタムツールの追加

```
# 独自のMCPツールを作成（第11章）
class MyCustomTool:
    def process_data(self, data):
        # あなたの処理
        return result
```

### Level 3: 専門エージェントの開発

```
# 特定分野に特化
class SalesAnalysisAgent(MCPAgent):
    """営業分析専門エージェント"""

    async def analyze_sales(self, period):
        # 専門的な分析処理
        pass
```

### Level 4: マルチエージェントシステム

```
# 複数のエージェントが協調
manager_agent = MCPAgent() # 管理者
worker_agent1 = MCPAgent() # 作業員1
worker_agent2 = MCPAgent() # 作業員2

# 協調して大規模タスクを処理
```

## 実践的な改善アイデア

## 改善1: UIを追加

```
# Streamlitで簡単なUI
import streamlit as st

st.title("My MCP Agent")
task = st.text_input("タスクを入力")

if st.button("実行"):
    result = agent.execute(task)
    st.write(result)
```

## 改善2: スケジュール実行

```
# 定期実行
import schedule

def daily_task():
    agent.execute("日報を作成")

schedule.every().day.at("18:00").do(daily_task)
```

## 改善3: 結果の保存

```
# データベースに保存
import sqlite3

def save_result(task, result):
    conn = sqlite3.connect('agent_history.db')
    # 履歴を保存
```

## 学習リソース

### 公式ドキュメント

- OpenAI: <https://platform.openai.com/docs>
- Anthropic: <https://docs.anthropic.com>
- Google AI: <https://ai.google.dev>

### 関連技術

- **LangChain**: エージェント構築フレームワーク
- **AutoGPT**: 自律型エージェント
- **CrewAI**: マルチエージェントシステム

## 日本語リソース

- Qiita記事
- Zenn記事
- 技術ブログ

## コミュニティ

### 参加する価値のあるコミュニティ

- **Discord**: AI/LLM開発者コミュニティ
- **Slack**: MCP開発者グループ
- **GitHub**: オープンソースプロジェクト

## 貢献の方法

1. バグレポート
2. ドキュメント改善
3. サンプルコード提供
4. 翻訳協力

## あなたの作品を共有しよう

### GitHubで公開

```
# あなたのエージェントを公開
git init
git add .
git commit -m "My first MCP agent"
git push
```

## ブログで発信

- 実装の工夫
- 遭遇した問題と解決法
- 業務での活用事例

## 最後のメッセージ

エージェント開発は、これからのAI時代の必須スキルです。

あなたができるようになったこと：

- AIと対話してタスクを自動化
- 複雑な処理を簡単な指示で実行

- エラーにも対応できる堅牢なシステム構築

これから挑戦してほしいこと：

- あなたの業務を革新するエージェント作成
- チームや会社への貢献
- オープンソースコミュニティへの参加

## 第10章のまとめ

- ✓ エージェントの必要性を理解した
- ✓ 実際に動くエージェントを構築した
- ✓ 業務に応用する方法を学んだ
- ✓ トラブルシューティングができる
- ✓ さらなる発展の道筋が見えた

おめでとうございます！ あなたは立派なMCPエージェント開発者です。

次章（第11章）では、独自のMCPサーバーを作成し、公開する方法を学びます。あなたのツールを世界中の開発者と共有しましょう！

---

## 付録: クイックリファレンス

### 基本的な使い方

```
# エージェント起動
from integrated_agent import MCPAgent
agent = MCPAgent(use_mock=False)

# タスク実行
result = await agent.execute("あなたのタスク")
print(result['result'])
```

### .env設定

```
LLM_PROVIDER=openai           # openai/anthropic/google
OPENAI_API_KEY=sk-xxx          # あなたのAPIキー
LLM_MODEL=gpt-3.5-turbo       # 使用モデル
MAX_TOKENS_PER_REQUEST=1000    # 最大トークン数
ENABLE_CACHE=true              # キャッシュ有効
CACHE_TTL_SECONDS=3600         # キャッシュ時間
```

### よく使うコマンド

```
# セットアップ
cd C:\MCP_Learning\chapter10
uv sync

# デモ実行
uv run python examples\simple_demo.py
uv run python examples\data_analysis_demo.py

# 自作スクリプト実行
uv run python my_agent.py
```

## エラー対処早見表

エラー	原因	対処法
Invalid API key	キー間違い	.env確認
Module not found	パッケージ不足	uv sync
Rate limit	使いすぎ	1分待つ
Timeout	処理が長い	タスク分割
Token limit	入力が長い	短くする

これで第10章は完了です！