第4章 クラス化/分割/マルチスレッド化/エラー・ハンドリング/ログ/コメント/API追加/API抽象クラス化

AIにリファクタリングをさせてみる

氏森 充

リスト1　リファクタリングの対象プログラム（server_0.py，一部抜粋）
基本的な通信や認証の処理が実装されている

```
import socket
import json
import random
import string

USER_CREDENTIALS = {
    "user1": "pass123",
    "user2": "password456"
}

SESSIONS = {}

def generate_session_id(length=16):        ← 認証処理
    return ''.join(random.choices(
   string.ascii_letters + string.digits, k=length))

def handle_client(conn, addr):
    while True:
        data = conn.recv(1024)             ← 通信処理
        if not data:
            break
```

本章では，特設 第3章で作成したAIコーディング・アシスタントに，実際のプログラムをリファクタリングさせてみます．その結果から，コーディングにおけるAI導入によるメリットとデメリットの両面を検証します．コード・レビューおよびリファクタリングに使用するAIとしては，ローカル実行が可能なLLM phi4：latestを選びました．

このモデルは，個人用PCでも動作するので，ローカル環境でのAI活用の現実性を検証する上でも有用です．

リファクタリングの準備

● リファクタリング対象プログラム

リファクタリングの題材には，第2章でAPI通信サーバ用に実装した初期コードを使用します（**リスト1**）．このコードには，基本的な通信処理および認証処理のみが実装されており，機能としては最小限の構成となっています．

本検証では，この初期コードをもとにAIコーディング・アシスタントによるコード・レビューを実施し，プログラム中に潜在する課題の抽出と改善提案を行います．

● 使用するプロンプト

レビュー時のプロンプトは非常にシンプルで，次の1文のみです．

プログラムをコードレビューしてください

本来，コード・レビューを行う際には，「セキュリティ面に注目する，保守性を評価する」といった目的に応じた観点を事前に明確に設定しておくことが推奨されます．しかし，今回の検証ではLLMによるレビュー能力そのものの動作を確認することを目的としているため，あえて特定の観点に絞らず，汎用的な短いプロンプトのみでレビューを依頼する方針としました．

このアプローチにより，LLMが自律的にどのような点に着目し，どのような指摘や提案を行うのかを確認できます．

● レビュー結果より改善対象を決める

レビュー結果を**リスト2**に示します．

さらに，改善点の追加を試みることとしました．プロンプトは「改善点として追加する項目はありますか？」です．結果を**リスト3**に示します．

リファクタリングによる改善の対象となる主な項目は次の通りとしました．

1. プログラムのクラス化
2. クラスごとの個別のファイルに分割
3. マルチスレッド化
4. エラー・ハンドリングを追加
5. ログ出力を追加
6. 次にコメントを追加
7. APIを追加
8. APIの抽象クラス化

本文では上記のリファクタリングのステップについて，誌面の都合上，ソースコードは一部抜粋して掲載しています．各ステップごとの全プログラムは本誌サポート・ページから入手できます．

https://interface.cqpub.co.jp/if2507u/