Coverity Connect MCP Server - シーケンス図集

Version: 1.0.0
作成日: 2025年7月21日
更新日: 2025年7月21日

📋 概要

本ドキュメントでは、Coverity Connect MCP Serverの主要な処理フローをシーケンス図で表現します。

🔄 1. システム初期化シーケンス

```
mermaid

sequenceDiagram
    participant User
    participant CLI as CLI (Click)
    participant Main as main.py
    participant Client as CoverityClient
    participant Env as Environment

    User->>CLI: coverity-mcp-server
    CLI->>Main: cli()
    Main->>Env: os.getenv('COVERITY_HOST')
    Env-->>Main: server configuration
    Main->>Main: initialize_client()
    Main->>Client: CoverityClient(host, port, ssl, username, password)
    Client->>Client: __init__()
    Client-->>Main: client instance
    Main->>Main: create_server()
    Main->>FastMCP: FastMCP("Coverity Connect MCP Server")
    FastMCP-->>Main: mcp instance
    Main->>Main: register tools and resources
    Main->>FastMCP: mcp.run()
    FastMCP-->>User: Server Ready
```

🔍 2. 欠陥検索処理シーケンス

```
mermaid
```

```
sequenceDiagram
    participant Claude as Claude Desktop
    participant MCP as FastMCP Server
    participant Main as main.py::search_defects
    participant Client as CoverityClient
    participant API as Coverity Connect API

    Claude->>MCP: MCP Request: search_defects(severity="High")
    MCP->>Main: @mcp.tool() search_defects()
    Main->>Main: initialize_client()
    Main->>Client: get_defects(filters={'severity': 'High'})
    Client->>Client: _get_session()

    alt Session not exists
        Client->>Client: create aiohttp.ClientSession
        Client->>Client: setup SSL context
        Client->>Client: setup BasicAuth
    end

    Client->>API: GET /api/viewContents/issues/v1?severity=High
    API-->>Client: HTTP 200 + JSON response
    Client->>Client: response.json()
    Client-->>Main: List[Dict] defects
    Main-->>MCP: defects or error
    MCP-->>Claude: JSON response
```

👥 3. ユーザー情報取得シーケンス

```
mermaid
```

```
sequenceDiagram
    participant Claude as Claude Desktop
    participant MCP as FastMCP Server
    participant Main as main.py::get_user_details
    participant Client as CoverityClient
    participant API as Coverity Connect API

    Claude->>MCP: MCP Request: get_user_details("developer1")
    MCP->>Main: @mcp.tool() get_user_details()
    Main->>Main: initialize_client()
    Main->>Client: get_user_details("developer1")

    Client->>API: GET /api/v2/users/developer1

    alt User found
        API-->>Client: HTTP 200 + user data
        Client-->>Main: user details
    else User not found
        API-->>Client: HTTP 404
        Client->>Client: get_users() # fallback search
        Client->>API: GET /api/v2/users
        API-->>Client: HTTP 200 + all users
        Client->>Client: filter by username
        Client-->>Main: user details or None
    end

    Main-->>MCP: user data or error
    MCP-->>Claude: JSON response
```

🏗 4. プロジェクトサマリー生成シーケンス

```
mermaid
```

```
sequenceDiagram
    participant Claude as Claude Desktop
    participant MCP as FastMCP Server
    participant Main as main.py::get_project_summary
    participant Client as CoverityClient
    participant API as Coverity Connect API

    Claude->>MCP: MCP Request: get_project_summary("WebApp")
    MCP->>Main: @mcp.tool() get_project_summary()
    Main->>Main: initialize_client()

    # Step 1: Get Project Details
    Main->>Client: get_project("WebApp")
    Client->>API: GET /api/viewContents/projects/v1
    API-->>Client: projects list
    Client->>Client: filter by project_id
    Client-->>Main: project details

    # Step 2: Get Streams
    Main->>Client: get_streams(project_id="WebApp")
    Client->>API: GET /api/viewContents/streams/v1?projectId=WebApp
    API-->>Client: streams list
    Client-->>Main: project streams

    # Step 3: Get Defects for each stream
    loop For each stream
        Main->>Client: get_defects(stream_id="main-stream", limit=1000)
        Client->>API: GET /api/viewContents/issues/v1?streamId=main-stream
        API-->>Client: defects list
        Client-->>Main: stream defects
        Main->>Main: aggregate severity and status counts
    end

    Main->>Main: build summary response
    Main-->>MCP: project summary
    MCP-->>Claude: comprehensive JSON response
```

🔐 5. 認証・セッション管理シーケンス

```
mermaid
```

```
sequenceDiagram
    participant Client as CoverityClient
    participant Session as aiohttp.ClientSession
    participant SSL as SSL Context
    participant API as Coverity Connect API

    Client->>Client: _get_session()

    alt Session not exists or closed
        Client->>SSL: ssl.create_default_context()
        SSL->>SSL: check_hostname = False
        SSL->>SSL: verify_mode = CERT_NONE
        SSL-->>Client: ssl_context

        Client->>Session: aiohttp.ClientSession()
        Note over Client,Session: auth=BasicAuth(username, password)<br/>timeout=30s<br/>headers=JSON
        Session-->>Client: session instance
        Client->>Client: self._session = session
    end

    Client->>Session: session.request(method, url, **kwargs)
    Session->>API: HTTP Request with Basic Auth

    alt Success
        API-->>Session: HTTP 200 + data
        Session-->>Client: response
    else Auth Error
        API-->>Session: HTTP 401
        Session-->>Client: AuthError
        Client->>Client: raise Exception("Authentication failed")
    else Not Found
        API-->>Session: HTTP 404
        Session-->>Client: response
        Client->>Client: logger.warning + return {}
    else Server Error
        API-->>Session: HTTP 500
        Session-->>Client: response
        Client->>Client: raise Exception(f"HTTP {status}")
    end
```

📊 6. リソースアクセスシーケンス

```
mermaid
```

```
sequenceDiagram
    participant Claude as Claude Desktop
    participant MCP as FastMCP Server
    participant Resource as @mcp.resource
    participant Main as main.py
    participant Client as CoverityClient
    participant API as Coverity Connect API

    Claude->>MCP: MCP Resource Request: coverity://projects/WebApp/config
    MCP->>Resource: get_project_config(project_id="WebApp")
    Resource->>Main: initialize_client()
    Resource->>Client: get_project("WebApp")
    Client->>API: GET /api/viewContents/projects/v1
    API-->>Client: projects data
    Client-->>Resource: project details

    Resource->>Resource: format config_info
    Note over Resource: Extract: projectKey, projectName,<br/>description, createdDate, lastModified, streams

    Resource-->>MCP: formatted project configuration
    MCP-->>Claude: configuration text response
```

🚨 7. エラーハンドリングシーケンス

```
mermaid
```

```
sequenceDiagram
    participant Claude as Claude Desktop
    participant MCP as FastMCP Server
    participant Main as main.py
    participant Client as CoverityClient
    participant API as Coverity Connect API

    Claude->>MCP: MCP Request: search_defects()
    MCP->>Main: @mcp.tool() search_defects()
    Main->>Main: initialize_client()

    alt Missing Environment Variables
        Main->>Main: os.getenv() returns None
        Main->>Main: raise ValueError("Missing required env vars")
        Main-->>MCP: [{"error": "Missing environment variables"}]
    else Network Connection Error
        Main->>Client: get_defects()
        Client->>API: HTTP Request
        API-->>Client: Connection Timeout
        Client->>Client: catch aiohttp.ClientError
        Client->>Client: logger.error()
        Client->>Client: raise Exception("Connection error")
        Client-->>Main: Exception
        Main->>Main: catch Exception
        Main->>Main: logger.error()
        Main-->>MCP: [{"error": "Connection error: ..."}]
    else Authentication Failure
        Client->>API: HTTP Request with Invalid Auth
        API-->>Client: HTTP 401
        Client->>Client: raise Exception("Authentication failed")
        Client-->>Main: Exception
        Main-->>MCP: [{"error": "Authentication failed"}]
    end

    MCP-->>Claude: Error response with details
```

🔄 8. セッション管理・クリーンアップシーケンス

```
mermaid
```

```
sequenceDiagram
    participant Main as main.py
    participant Client as CoverityClient
    participant Session as aiohttp.ClientSession
    participant Context as Context Manager

    Note over Main,Context: Using async context manager pattern

    Main->>Client: async with CoverityClient(...) as client:
    Client->>Context: __aenter__()
    Context-->>Main: client instance

    Main->>Client: client.get_projects()
    Client->>Session: session operations
    Session-->>Client: responses
    Client-->>Main: data

    Main->>Context: __aexit__() (automatic)
    Context->>Client: close()
    Client->>Session: session.close()
    Session->>Session: cleanup connections
    Session-->>Client: cleanup complete
    Client-->>Context: cleanup complete
    Context-->>Main: exit complete
```

🧪 9. 開発環境・Mock Serverシーケンス

```
mermaid
```

```
sequenceDiagram
    participant Claude as Claude Desktop
    participant MCP as MCP Server
    participant Client as CoverityClient
    participant Mock as Mock Server (localhost:5000)

    Note over Claude,Mock: Development Environment

    Claude->>MCP: search_defects(severity="High")
    MCP->>Client: get_defects(filters={'severity': 'High'})
    Client->>Mock: GET localhost:5000/api/viewContents/issues/v1

    Mock->>Mock: generate dummy data
    Note over Mock: Returns hardcoded defects:<br/>CID 1001-1003 with various severities

    Mock-->>Client: HTTP 200 + dummy JSON
    Client-->>MCP: dummy defects list
    MCP-->>Claude: test data response

    Note over Claude,Mock: Allows development without real Coverity Connect
```

📈 10. バッチ処理・大量データシーケンス

```
mermaid
```

```
sequenceDiagram
    participant Claude as Claude Desktop
    participant MCP as FastMCP Server
    participant Main as get_project_summary
    participant Client as CoverityClient
    participant API as Coverity Connect API

    Claude->>MCP: get_project_summary("LargeProject")
    MCP->>Main: get_project_summary()

    # Parallel processing for multiple streams
    par Stream 1
        Main->>Client: get_defects(stream_id="main", limit=1000)
        Client->>API: GET /issues/v1?streamId=main&rowCount=1000
        API-->>Client: 1000 defects
    and Stream 2
        Main->>Client: get_defects(stream_id="develop", limit=1000)
        Client->>API: GET /issues/v1?streamId=develop&rowCount=1000
        API-->>Client: 800 defects
    and Stream 3
        Main->>Client: get_defects(stream_id="release", limit=1000)
        Client->>API: GET /issues/v1?streamId=release&rowCount=1000
        API-->>Client: 200 defects
    end

    Main->>Main: aggregate all stream data
    Main->>Main: calculate statistics
    Note over Main: severity_counts, status_counts,<br/>total_defects per stream

    Main-->>MCP: comprehensive summary
    MCP-->>Claude: aggregated project statistics
```

これらのシーケンス図は、Coverity Connect MCP Serverの主要な処理フローを詳細に表現しています。システムの初期化から複雑なデータ集約処理まで、実装の動作を理解するための包括的なガイドとなります。