生成AI時代のMCPプロトコル実践ガイド

効率的な知識処理とAI連携のための統合プロトコル

序文

生成AIの急速な発展により、さまざまなクライアントツールとナレッジリポジトリを効率的に連携させる必要性が高まっています。本書では、Message Capacity Protocol (MCP)を軸に、AIツールと各種サービスの連携方法を実践的に解説します。単なる技術解説にとどまらず、実際のプロジェクトでの活用事例や、導入時の課題と解決策まで幅広くカバーしています。

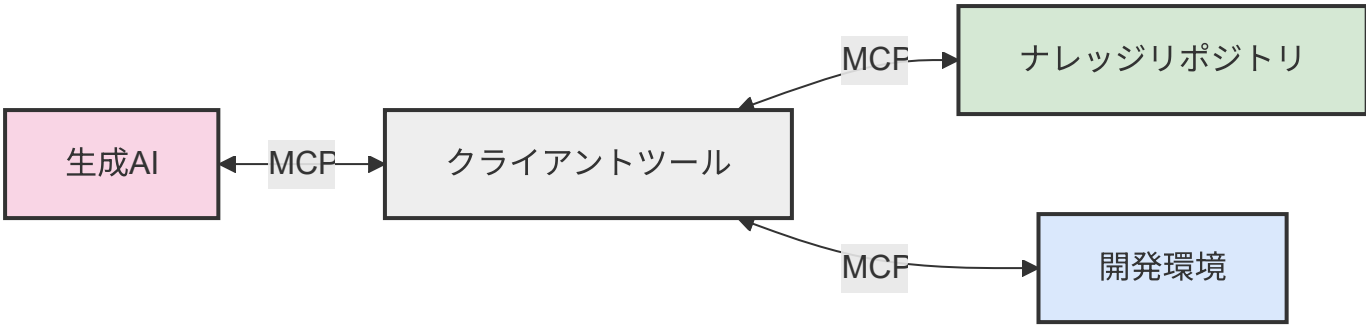
目次

- C.1 Python/FastAPI MCPサーバー 主要コンポーネント
- C.2 React/TypeScript MCPクライアント サンプル
- C.3 C# MCPコネクタ サンプル
- C.4 MCPプラグイン サンプル

第1章: MCPとは何か - 概念と基本原理

1.1 MCPの概念と目的

Message Capacity Protocol (MCP)は、生成AIと各種開発・知識管理ツールを効率的に連携させるための通信プロトコルです。従来の個別連携の限界を超え、統一されたインターフェースを通じて異なるAIモデルと多様なツール間のシームレスな連携を実現します。



MCPの主な目的は以下の通りです：

1. **知識アクセスの統一**: さまざまなソースの情報に一貫した方法でアクセス
2. **コンテキスト共有**: AIとツール間で作業コンテキストをスムーズに引き継ぎ
3. **機能拡張性**: 新しいツールやサービスの容易な統合
4. **セマンティック処理**: 単なるデータ交換を超えた意味理解と処理

> **ベテランの知恵袋**: *「MCPの真の価値は、単なるデータのやり取りではなく、コンテキストの保持にあります。プロジェクトの文脈を失わずにツール間を移動できることで、開発者の生産性は飛躍的に向上します。」* - 大規模AIプロジェクト責任者

1.2 MCPの基本アーキテクチャ

MCPは以下の主要コンポーネントで構成されています：

1.2.1 メッセージング層

MCPの核となるのはメッセージング層です。これはJSON-RPC形式の標準化されたメッセージを使用し、以下の特徴を持ちます：

```
{
  "jsonrpc": "2.0",
  "method": "query.knowledge",
  "params": {
    "context": "プロジェクト計画の最適化手法について",
    "sources": ["github://org/repo", "obsidian://vault/folder"],
    "limit": 10
  },
  "id": "req-12345"
}
```

このメッセージング層は、同期・非同期両方の通信モードをサポートし、WebSocketやHTTP/2などの複数のトランスポートプロトコル上で動作します。

1.2.2 コンテキスト管理

MCPの重要な特徴は、会話や作業のコンテキストを維持する能力です。これにより：

- ツール間を移動しても会話履歴が維持される
- ドキュメントの作業状態が保持される
- 異なるリポジトリからの情報が統合される

1.2.3 セキュリティレイヤー

MCPには強固なセキュリティ機能が組み込まれています：

- トークンベースの認証
- きめ細かなアクセス制御
- エンドツーエンドの暗号化オプション
- 監査ログ機能

```
# MCPクライアントでの認証例
from mcp_client import MCPClient

client = MCPClient()
client.authenticate(
    method="oauth",
    token="eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
    scope=["read:repos", "write:documents"]
)

# 認証後にリクエスト
response = client.query_knowledge(
    context="マイクロサービスアーキテクチャ設計パターン",
    sources=["github://org/architecture-docs"]
)
```

> **若手の疑問解決**: *「なぜMCPは新しいプロトコルを定義する必要があるの？既存のREST APIやGraphQLではダメなの？」*
>
> MCPが独自プロトコルを定義する理由は、AIとの対話に特化した双方向性とコンテキスト管理にあります。従来のAPIは基本的に要求と応答のパターンに限定されますが、MCPはストリーミング応答や、コンテキストを保持したまま複数のサービスを横断するセッション管理など、AIとの継続的な対話に必要な機能を提供します。

1.3 MCPの基本プロトコル仕様

MCPのコア仕様は以下の主要部分で構成されています：

1.3.1 リクエスト・レスポンス構造

基本的なリクエスト構造：

```
interface MCPRequest {
  version: string;       // MCPプロトコルバージョン
  requestType: string;   // リクエストタイプ (query, update, stream, etc.)
  resource: string;      // 対象リソース識別子
```

```
  context?: {             // オプションのコンテキスト情報
    sessionId?: string;   // セッションID
    history?: Message[]; // これまでの会話履歴
    metadata?: any;       // その他のメタデータ
  };
  params: any;            // リクエスト固有のパラメータ
  auth?: {                // 認証情報（必要な場合）
    token?: string;
    type?: string;
  };
}
```

基本的なレスポンス構造：

```
interface MCPResponse {
  version: string;        // MCPプロトコルバージョン
  status: number;         // ステータスコード
  requestId: string;      // 対応するリクエストID
  data?: any;             // レスポンスデータ
  error?: {               // エラー情報（失敗時）
    code: number;
    message: string;
    details?: any;
  };
  context?: {             // 更新されたコンテキスト情報
    sessionId?: string;
    metadata?: any;
  };
}
```

1.3.2 主要オペレーション

MCPでは、以下の主要オペレーションが定義されています：

オペレーション	説明	一般的な用途
`query`	情報検索リクエスト	ナレッジベースやリポジトリからの情報取得
`update`	情報更新リクエスト	ドキュメントやリソースの更新
`stream`	ストリーミング接続	リアルタイムレスポンスやイベント通知
`execute`	コマンド実行	特定のアクションやワークフロー実行
`connect`	サービス接続	新しいサービスへの接続とハンドシェイク

1.3.3 リソース指定方式

MCPでは、統一されたリソース指定方式を使用します：

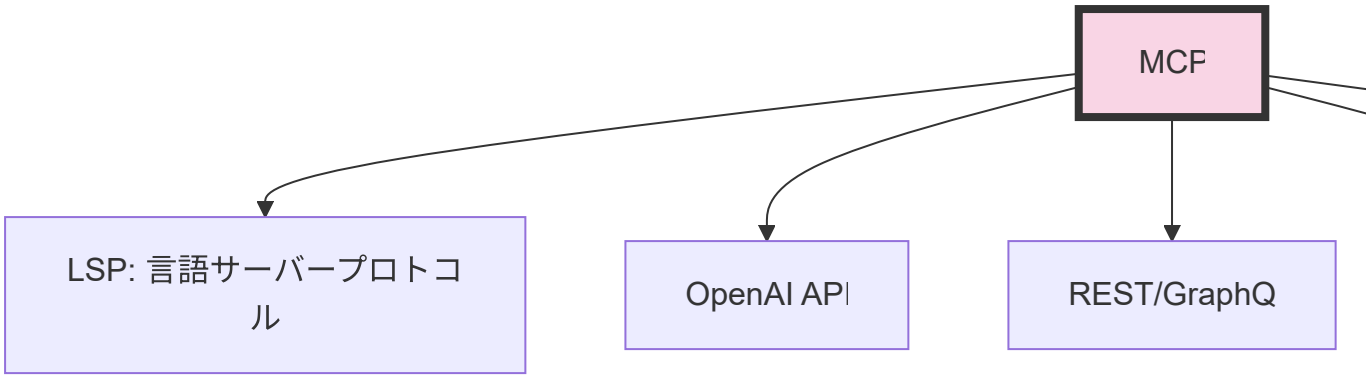
```
protocol://service/resource-type/identifier?parameters
```

例：

- `github://organization/repo/file.md` - GitHubの特定ファイル
- `obsidian://vault/folder/note.md` - Obsidianのノート
- `redmine://project/issues?status=open` - Redmineのオープン課題

1.4 実世界でのMCPの位置づけ

MCPは以下のような既存技術との関係の中で位置づけられます：



LSP（Language Server Protocol）が開発ツールと言語サービスを分離したように、MCPはAIと各種サービス間のインターフェースを標準化します。これにより、クライアントとサーバーの実装が分離され、相互運用性が向上します。

> **プロジェクト事例**: *大手製造業のエンジニアリング部門では、技術文書、CADデータ、過去の設計レビューなど異なるシステムに分散していた情報をMCPで統合しました。エンジニアは設計作業中にAIアシスタントを通じて、これらの情報に統一的にアクセスできるようになり、設計時間が約30%短縮されました。*

第2章: MCPのエコシステム - 対応ツールと連携サービス

2.1 MCPクライアントの全体像

MCPクライアントは、ユーザーがMCPサーバーと対話するためのインターフェースです。現在、以下のような主要なクライアントが存在します：

2.1.1 AI対話型クライアント

クライアント	特徴	MCPサポート状況
Claude Desktop	AIアシスタントとの高度な対話、ドキュメント処理	ネイティブサポート
ChatGPT Desktop	OpenAIモデルとの対話、プラグイン連携	部分的サポート（プラグイン経由）
Perplexity	研究・調査特化型対話、引用付き回答	実験的サポート

2.1.2 開発環境統合クライアント

クライアント	特徴	MCPサポート状況
VSCode	コーディング支援、ドキュメント連携	拡張機能経由でサポート
Cursor	AIベースコード生成と編集、ペアプログラミング	ネイティブサポート
Windsurf	ブラウザベース開発環境、複数AIモデル連携	フルサポート
JetBrains IDEs	各種プログラミング言語向けIDE	プラグイン経由でサポート準備中

2.1.3 ナレッジ管理クライアント

クライアント	特徴	MCPサポート状況
Obsidian	個人知識管理、ノート連携	コミュニティプラグイン経由
Notion	チーム協働、データベース機能	APIベースの限定サポート
Roam Research	双方向リンクベースの知識管理	実験的サポート

2.2 サーバー実装のバリエーション

MCPサーバーは様々な形で実装されており、以下のような主要なタイプがあります：

2.2.1 コード・プロジェクト管理系

サーバー	特徴	MCPサポート状況
GitHub	コード管理、課題追跡、CI/CD	公式APIとMCPアダプター
GitLab	統合開発ライフサイクル	拡張モジュール経由

サーバー	特徴	MCPサポート状況
GitBucket	軽量Gitサーバー	プラグイン対応
Gitea	自己ホスト型Git管理	コミュニティ開発中

2.2.2 プロジェクト管理・課題追跡系

サーバー	特徴	MCPサポート状況
Redmine	プロジェクト管理、チケット管理	プラグイン対応
Jira	課題追跡、アジャイル支援	アダプター開発中
Trello	カンバンボード型タスク管理	限定的サポート

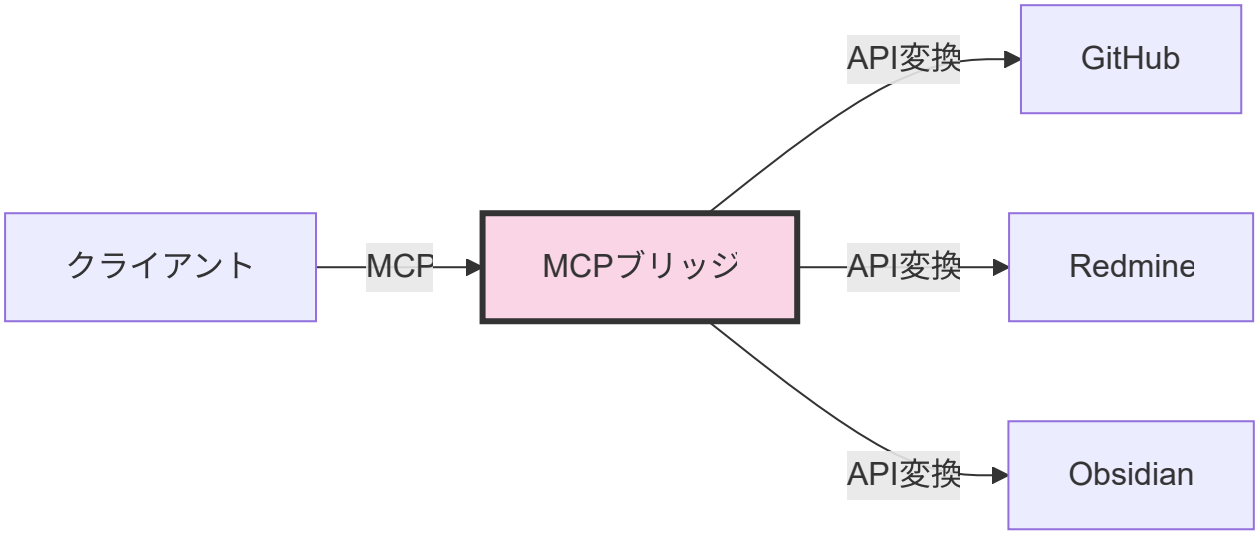
2.2.3 知識管理・ドキュメント系

サーバー	特徴	MCPサポート状況
Obsidian Publish	マークダウンベース知識共有	コミュニティプラグイン
Confluence	チーム知識管理	API経由の連携
MediaWiki	構造化ウィキプラットフォーム	拡張機能開発中
DokuWiki	シンプルなファイルベースWiki	実験的サポート

2.3 統合サービスとブリッジ

クライアントとサーバーを橋渡しする統合サービスとブリッジも存在します：

2.3.1 MCPブリッジサービス



主要なブリッジサービス：

- **MCPHub**: オープンソースのMCPゲートウェイ
- **Semantic Bridge**: 意味解析強化型MCPブリッジ
- **DevConnect**: 開発者向け統合MCPサービス

2.3.2 プロキシとエクステンダー

既存サービスをMCP対応にするためのプロキシも登場しています：

- **GitProxy**: GitサービスへのMCPインターフェース提供
- **WikiBridge**: Wiki系サービスへのMCP連携
- **LegacyLink**: レガシーシステムのMCP化支援

> **失敗から学ぶ**: *「当初、私たちは各サービスに個別のMCP対応を実装しようとして苦労しました。結局、中央のMCPブリッジを立てて、そこで変換と認証を一元管理する方式に切り替えたところ、統合コストが大幅に下がりました。新しいサービスを追加する際も、ブリッジに新しいコネクタを追加するだけで済むようになりました。」* - システム統合担当エンジニア

2.4 将来性のある新興サービス

現在は限定的なサポートですが、将来的に注目されるMCP対応サービス：

サービス	概要	MCPとの関連性
CodeInterpreter	コード実行環境とAIの統合	AIが書いたコードをその場で実行・検証
KnowledgeGraph	グラフベースの知識管理	セマンティックな情報連携
WebPilot	ウェブ情報とAIの統合	リアルタイムウェブ情報へのアクセス
LangStream	ストリーミングAI処理基盤	大規模データストリームの処理

> **ベテランの知恵袋**: *「MCPエコシステムを選ぶときは、単に現在のサポート状況だけでなく、コミュニティの活発さや開発スピードも重視しましょう。小さくても活発なコミュニティを持つプロジェクトの方が、大企業の不活発なプロジェクトより将来性があることが多いです。」*

MCPエコシステム選定チェックリスト

- [ ] 組織の主要ワークフローに必要なツールのMCPサポート状況確認
- [ ] 認証・セキュリティ要件との互換性確認
- [ ] 拡張性とカスタマイズの容易さの評価
- [ ] コミュニティまたはベンダーサポートの充実度確認
- [ ] 将来的な統合可能性の評価
- [ ] 導入・運用コストの見積もり
- [ ] 既存システムとの移行・統合計画の策定

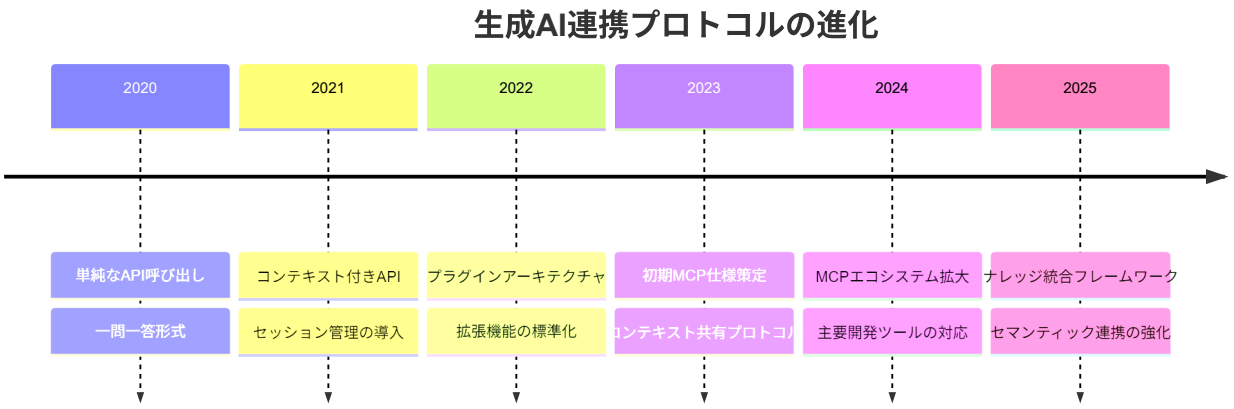
第3章: MCPの歴史と発展 - AI連携プロトコルの変遷

3.1 AI連携プロトコルの進化

MCPの理解を深めるには、その歴史的背景と発展過程を知ることが重要です。

3.1.1 前史：初期の連携方式（2020-2022）

初期の生成AIツールは、単純なAPI呼び出しによる連携が主流でした：



この時期の課題：

- コンテキスト維持の困難さ
- 認証の複雑さ
- ツール間連携の欠如

3.1.2 プラグインアーキテクチャの台頭（2022-2023）

ChatGPTプラグインなど、拡張可能なアーキテクチャの登場により、AIの機能拡張が容易になりました：

```
# 初期のプラグイン連携例（ChatGPT Plugin形式）
def get_weather(location: str, unit: str = "celsius") -> dict:
    """
    指定された場所の天気情報を取得する

    Args:
        location: 都市名や地域名
        unit: 温度単位（celsius/fahrenheit）

    Returns:
        天気情報を含む辞書
    """
    # 実際のAPI呼び出し
    # ...
```

この方式の限界：

- AIプラットフォーム固有の実装

- ツール間でのコンテキスト共有の欠如
- 認証の分断

3.1.3 MCPの誕生と初期標準化（2023-2024）

複数のAIプラットフォームとツールを統合する必要性から、MCPの初期仕様が策定されました：

- 複数のAIベンダーと開発ツールベンダーによる共同イニシアチブ
- 基本的なメッセージング構造の標準化
- リソース指定方式の統一
- 認証フレームワークの設計

3.2 現在のMCP仕様状況

現在のMCP仕様は以下のようなバージョンと特徴があります：

バージョン	主な特徴	採用状況
MCP 0.9	基本的なメッセージング構造とリソースアドレッシング	初期実装、実験的
MCP 1.0	安定したコア機能、認証フレームワーク	主要サービスでの採用開始
MCP 1.1	ストリーミング拡張、非同期処理の強化	新規実装で採用中
MCP 1.2 (開発中)	セマンティックコンテキスト、フェデレーション機能	草案段階

> **プロジェクト事例**: *大手ソフトウェア企業では、MCPを活用して社内の知識管理システム、ソースコード管理、プロジェクト管理ツールを統合しました。開発者はIDEから直接、関連ドキュメントや過去の設計判断理由にアクセスでき、コードの品質と開発スピードが向上しました。特に新メンバーの立ち上げ時間が40%短縮されたと報告されています。*

3.3 MCPの設計原則と哲学

MCPの設計には以下のような核となる原則があります：

3.3.1 コンテキスト中心設計

従来のプロトコルとは異なり、MCPはコンテキストを中心に設計されています：

- 会話や作業のコンテキストを明示的に表現
- コンテキストの保持と転送メカニズム
- 異なるサービス間でのコンテキスト共有

```
// MCPにおけるコンテキスト表現例
{
  "contextType": "conversation",
  "contextId": "conv-12345",
  "history": [
    {
      "role": "user",
      "content": "プロジェクトの設計ドキュメントを教えて"
```

```
      },
      {
        "role": "assistant",
        "content": "どのプロジェクトについてお探しですか？"
      },
      {
        "role": "user",
        "content": "WebAPIモダナイゼーションプロジェクト"
      }
    ],
    "metadata": {
      "projectId": "proj-modernization",
      "lastAccessed": "2023-10-15T14:30:00Z"
    }
  }
```

3.3.2 拡張性と互換性

MCPはシステムの長期的進化を考慮して設計されています：

- バージョニングメカニズム
- 後方互換性の維持
- 拡張可能なメッセージ構造
- オプション機能の発見メカニズム

3.3.3 セキュリティファースト

MCPはセキュリティを基本設計に組み込んでいます：

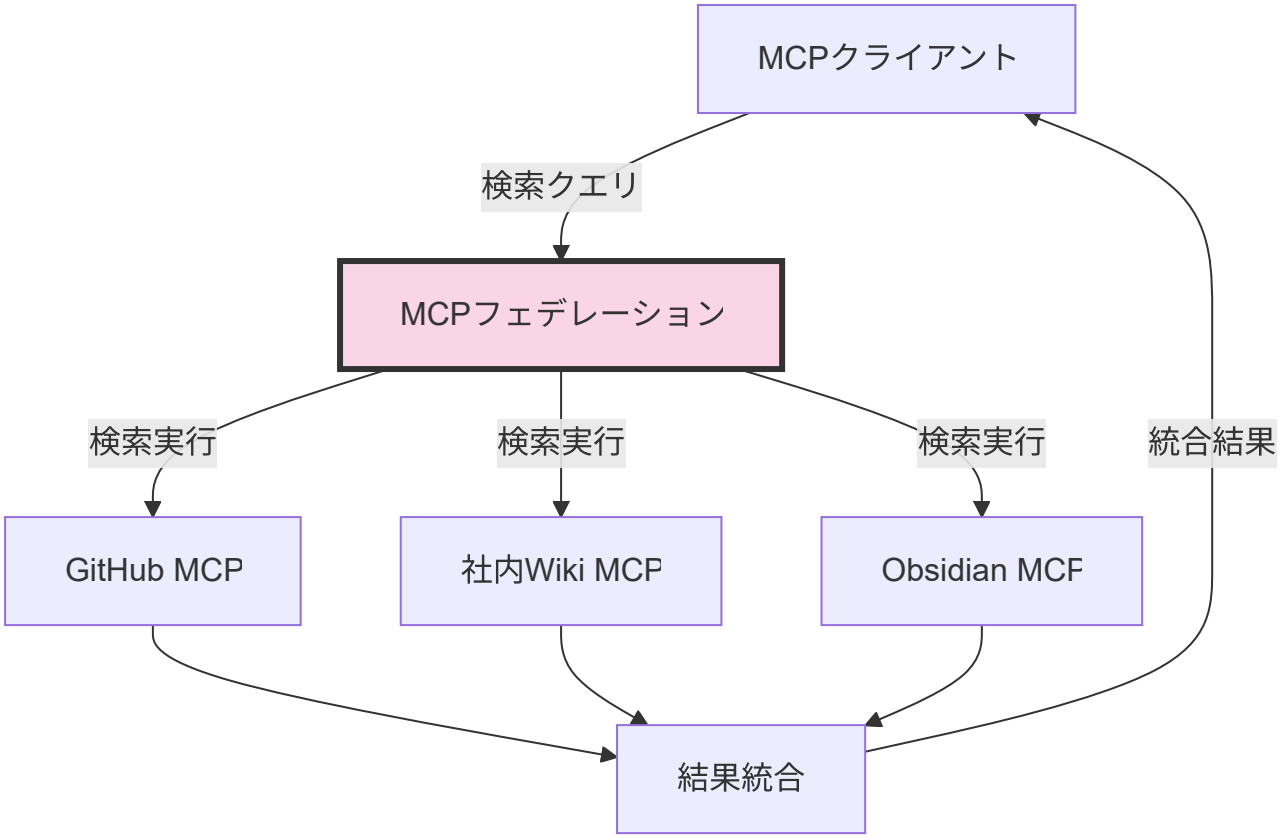
- ゼロトラストモデルの採用
- 最小権限の原則
- エンドツーエンド暗号化オプション
- 詳細な監査ログ機能

> **若手の疑問解決**: *「MCPは本当に新しいプロトコルなの？既存のものの組み合わせでは？」*
>
> MCPは確かに多くの既存技術（JSON-RPC、OAuth、WebSocketなど）の上に構築されていますが、その革新性は個々の技術ではなく、それらを統合してAIとツール間の対話に特化した統一的なフレームワークを提供している点にあります。車のパーツは既存でも、それを組み合わせた車としての設計が新しいのと同じです。

3.4 MCPの未来展望

今後数年間で予想されるMCPの発展方向：

3.4.1 フェデレーションとP2P対応



- MCP対応サービス間のP2P検索と情報共有
- 分散型認証と権限管理
- コンテンツの分散検索とマージ

3.4.2 セマンティック拡張

- 高度な意味理解によるコンテキスト処理
- 知識グラフとの統合
- マルチモーダルコンテンツ（テキスト、画像、コードなど）の統合処理

3.4.3 リアルタイム協調作業

- 複数AIと複数ユーザーの同時対話
- リアルタイム編集とフィードバック
- 共同作業コンテキストの管理

> **ベテランの知恵袋**: *「プロトコルの世界では、最初に100%完璧を目指すより、核となる部分を確実に定義し、実装経験から学びながら進化させていくアプローチが成功しています。MCPも同様で、基本機能の安定性を確保しつつ、実際の使用から学んでいくことが重要です。」* - プロトコル設計者

第3章まとめ

- MCPは単なる新技術ではなく、AIとツールの連携に関する考え方の進化を表している
- コンテキストを中心とした設計が、従来の単純なAPI連携と一線を画す特徴
- 現在のバージョン1.1が安定版として主要ツールで採用が広がっている
- フェデレーション、セマンティック処理、リアルタイム協調作業が今後の発展方向

第4章: Claude Desktopで始めるMCP活用

4.1 Claude Desktopの概要とMCP対応

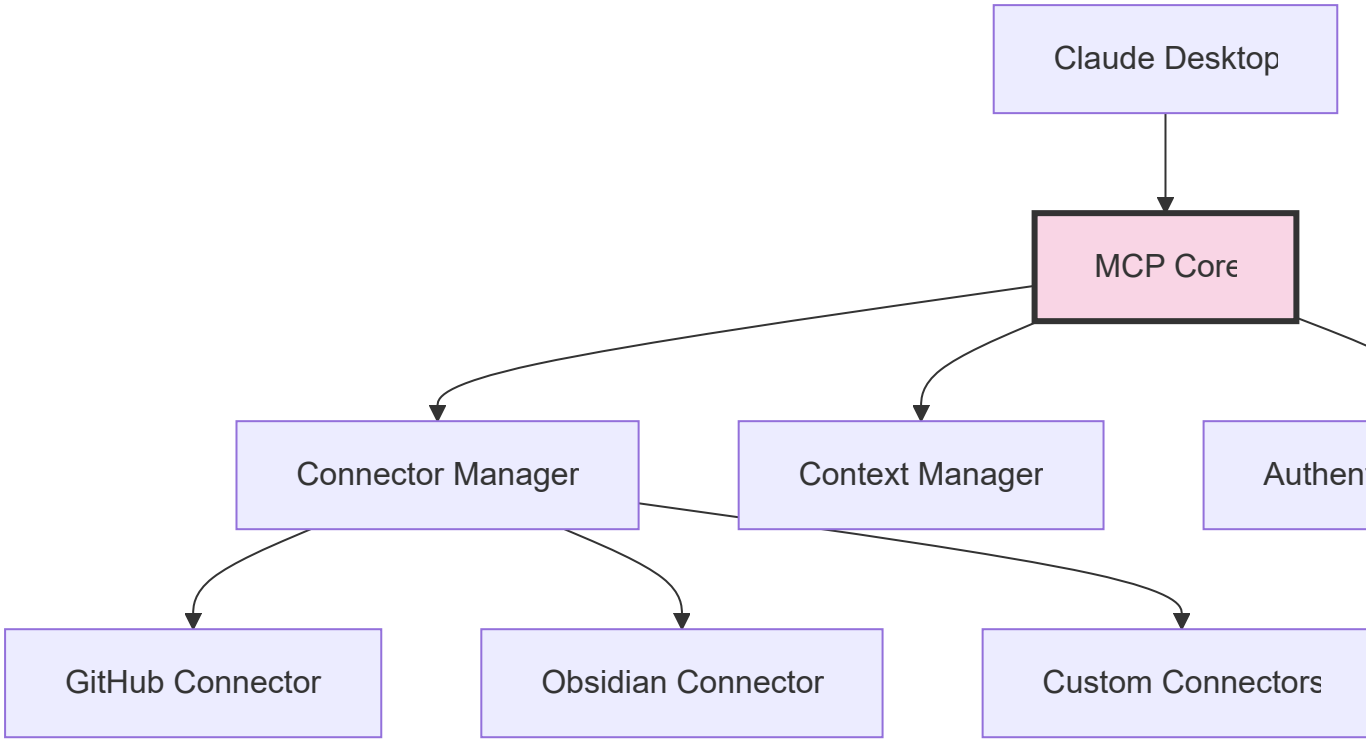
Claude Desktopは、Anthropicが提供するAIアシスタントのデスクトップクライアントで、MCPのネイティブサポートを備えています。その特徴は以下の通りです：

4.1.1 Claude Desktopの基本機能

- 高度な自然言語対話能力
- ドキュメント解析と要約
- コード生成と解析
- マルチモーダル入力（テキスト、画像、ファイル）

4.1.2 MCPサポート機能

Claude DesktopのMCP機能は、以下のコンポーネントで構成されています：



- **MCP Core**: 基本的なプロトコル処理エンジン
- **Connector Manager**: 各種サービスへの接続管理
- **Context Manager**: 会話・作業コンテキストの保持
- **Authentication Manager**: 認証情報の安全な管理

4.2 Claude Desktopのセットアップと基本設定

4.2.1 インストールと初期設定

1. Claude Desktop公式サイトからインストーラーをダウンロード
2. インストーラーを実行し、指示に従ってインストール
3. 初回起動時、Anthropicアカウントでログイン
4. MCPサポートを有効化（設定メニュー > 拡張機能 > MCP）

4.2.2 MCP接続の設定

```
// MCP設定ファイル例（~/.claude/mcp-config.json）
{
  "enabled": true,
  "connections": [
    {
      "name": "GitHub",
      "type": "github",
      "url": "https://api.github.com",
      "authType": "oauth",
      "repos": ["user/repo1", "org/repo2"]
    },
    {
      "name": "Obsidian",
      "type": "obsidian",
      "path": "/path/to/vault",
      "syncType": "local"
    }
  ],
  "defaultConnections": ["GitHub", "Obsidian"],
  "contextRetention": {
    "conversationHistory": 10,
    "maxDays": 30
  }
}
```

> **若手の疑問解決**: *「MCPの設定って複雑そうだけど、GUIで設定できないの？」*
>
> 最新版のClaude Desktopでは、「設定 > MCP接続」からグラフィカルインターフェースでMCP設定が可能です。設定はJSON形式でエクスポート/インポートもできるので、複数環境で同じ設定を使いたい場合に便利です。

4.2.3 認証設定

各サービスへの認証設定：

1. **GitHub認証**:
   - 設定メニュー > MCP接続 > GitHub > 認証設定
   - 「新規トークン作成」をクリック
   - GitHubの認証ページが開き、必要な権限を選択
   - 生成されたトークンが自動的にClaude Desktopに保存
2. **Obsidian認証**:
   - ローカルVaultの場合：パスを指定するだけ
   - Obsidian Publishの場合：APIキーを設定

4.3 Claude Desktopでの基本的なMCP活用

4.3.1 リポジトリ情報へのアクセス

Claude Desktopを通じてGitHubリポジトリの情報にアクセスする例：

```
GitHubリポジトリuser/projectの最新のIssueを教えてください
```

Claude DesktopはMCP経由でGitHubに接続し、最新のIssue情報を取得して回答します。

4.3.2 ナレッジベースとの連携

Obsidianの知識ベースと連携する例：

```
Obsidianのプロジェクト計画ノートの概要を教えて、
そしてそれをもとにタスクリストを作成して
```

MCPを通じてObsidianのノートにアクセスし、その内容に基づいた回答が得られます。

4.3.3 マルチソース情報統合

複数のソースからの情報を統合する例：

```
GitHubの最新コミットとObsidianのデザインドキュメントを
比較して、実装とデザインの違いを分析してください
```

> **プロジェクト事例**: *あるソフトウェア開発チームでは、Claude DesktopのMCP機能を活用して、プロジェクト文書（Obsidian）、コード（GitHub）、課題管理（Redmine）を統合的に参照できるようにしました。チームメンバーは「なぜこのコードがこのように実装されたのか」という質問に対して、設計文書と課題チケットを関連付けた回答を得られるようになり、コードレビューの効率が大幅に向上しました。*

4.4 高度なClaude Desktop活用テクニック

4.4.1 カスタムコネクタの作成

Claude Desktopでは、標準でサポートされていないサービス向けのカスタムコネクタを作成できます：

```
// カスタムMCPコネクタの例（~/.claude/connectors/custom-wiki.js）
module.exports = {
  name: "CustomWiki",
  version: "1.0.0",

  // 接続設定
  configure: async (config) => {
    // 設定の検証と保存
    if (!config.baseUrl) {
      throw new Error("baseUrl is required");
    }
    return { ...config };
  },

  // 認証処理
  authenticate: async (config, credentials) => {
    // 認証処理の実装
    const response = await fetch(`${config.baseUrl}/api/auth`, {
      method: "POST",
      headers: { "Content-Type": "application/json" },
```

```
      body: JSON.stringify(credentials)
    });

    if (!response.ok) {
      throw new Error("Authentication failed");
    }

    const authResult = await response.json();
    return { token: authResult.token };
  },

  // クエリ処理
  query: async (config, auth, params) => {
    const response = await fetch(`${config.baseUrl}/api/query`, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
        "Authorization": `Bearer ${auth.token}`
      },
      body: JSON.stringify(params)
    });

    if (!response.ok) {
      throw new Error(`Query failed: ${response.statusText}`);
    }

    return await response.json();
  }
};
```

4.4.2 プロンプトテンプレートとMCPの連携

Claude Desktopでは、プロンプトテンプレートとMCP機能を組み合わせることで、強力なワークフローを構築できます：

```
# GitHub PR レビューテンプレート

以下のGitHub PRをレビューしてください:
repo: {{repo}}
PR番号: {{pr_number}}

レビュー観点:
- コードの品質と可読性
- テストの十分性
- セキュリティ上の懸念点
- パフォーマンスへの影響

フィードバックは「良い点」「改善点」「質問」に分けて整理してください。
```

このテンプレートを使用すると、Claude DesktopはMCP経由でGitHubからPR情報を取得し、自動的にコードレビューを実施します。

4.4.3 ワークフロー自動化

Claude DesktopのMCP機能を使った自動化の例：

```
// Claude Desktop Automation API example
const claude = require('claude-desktop-api');

// GitHub IssueからObsidianのプロジェクトノートを自動生成
async function createProjectNoteFromIssue(issueUrl) {
  // MCPを通じてGitHub Issueの詳細を取得
  const issueDetails = await claude.mcp.query({
    resource: issueUrl,
    requestType: 'get'
  });

  // Issueの内容をもとにプロジェクト概要を生成
  const projectSummary = await claude.generate({
    prompt: `以下のIssueの内容から、プロジェクト概要ドキュメントを作成してください：

      タイトル: ${issueDetails.title}
      説明: ${issueDetails.body}
      ラベル: ${issueDetails.labels.join(', ')}

      以下の項目を含めてください：
      - プロジェクトの目的
      - 主要なタスク
      - 想定される課題
      - 必要なリソース`,
    maxTokens: 1000
  });

  // MCPを通じてObsidianにノートを作成
  await claude.mcp.query({
    resource: 'obsidian://vault/Projects',
    requestType: 'create',
    params: {
      filename: `Project-${issueDetails.number}.md`,
      content: projectSummary
    }
  });

  return {
    success: true,
    notePath: `Projects/Project-${issueDetails.number}.md`
  };
}
```

> **失敗から学ぶ**: *「最初、私たちはClaude Desktopの高度な機能を一度に全て活用しようとして混乱しました。後から学んだベストプラクティスは、まず基本的なクエリから始めて、徐々に複雑な機能を追加していくアプローチです。特に認証関連のトラブルは段階的に解決するのが効果的でした。」* - デベロッパーアドボケイト

Claude Desktop MCPセットアップチェックリスト

- [ ] Claude Desktopの最新バージョンをインストール
- [ ] MCPサポートを設定メニューから有効化

- [ ] 必要なサービスコネクタをインストール
- [ ] 各サービスの認証設定を完了
- [ ] テスト接続で動作確認
- [ ] 必要に応じてカスタムコネクタを設定
- [ ] プロンプトテンプレートの作成と検証
- [ ] バックアップと復元手順の確認

第5章: 開発環境と連携 - VSCode、Cursor、Windsurfの実践

5.1 VSCodeでのMCP活用

Visual Studio CodeはMCP対応拡張機能により、AIと各種サービスを統合的に活用できます。

5.1.1 VS Code MCP拡張機能のインストールと設定

1. VS Codeマーケットプレイスから「MCP Connector」拡張機能をインストール
2. 拡張機能の設定からMCPエンドポイントとサービス設定を構成

```
// settings.json
{
  "mcp.enabled": true,
  "mcp.endpoint": "https://mcp.example.com",
  "mcp.auth.method": "token",
  "mcp.connectors": {
    "github": {
      "enabled": true,
      "repositories": ["user/repo1", "org/repo2"]
    },
    "obsidian": {
      "enabled": true,
      "vault": "/path/to/vault"
    }
  }
}
```

5.1.2 VS CodeからのMCPクエリ

VS Codeでは、コマンドパレット（Ctrl+Shift+P）から「MCP: Query」コマンドを実行するか、専用のサイドパネルからMCPクエリを実行できます：

```
> MCP: Query
```

クエリ例：

```
GitHubリポジトリorg/projectのREADME.mdファイルとプロジェクトの構造を分析して、
このプロジェクトの概要と主要コンポーネントを説明してください
```

5.1.3 コーディング支援としてのMCP活用

VS Codeでは、MCP機能をコーディングに直接統合できます：

```
// MCP-Assisted Coding Example
function calculateTotalRevenue() {
  // MCPに質問: データベースからユーザー収益データを取得するクエリは？
  // MCP回答: 以下のようなクエリが適しています
  const query = `
    SELECT user_id, SUM(amount) as total_revenue
    FROM transactions
```

```
    WHERE status = 'completed'
    GROUP BY user_id
    ORDER BY total_revenue DESC
  `;

  // MCPに質問: このクエリの結果を処理するベストプラクティスは？
  // MCP回答: 以下のパターンが推奨されます
  const results = executeQuery(query);
  const processedData = results.map(row => ({
    userId: row.user_id,
    revenue: formatCurrency(row.total_revenue),
    tier: calculateUserTier(row.total_revenue)
  }));

  return processedData;
}
```

> **ベテランの知恵袋**: *「VS CodeのMCP機能は、キーボードショートカットを覚えることで生産性が大幅に向上します。私のお気に入りは Alt+M, Q でMCPクエリ、Alt+M, C でコードコンテキスト付きクエリ、Alt+M, D でドキュメント生成です。」*

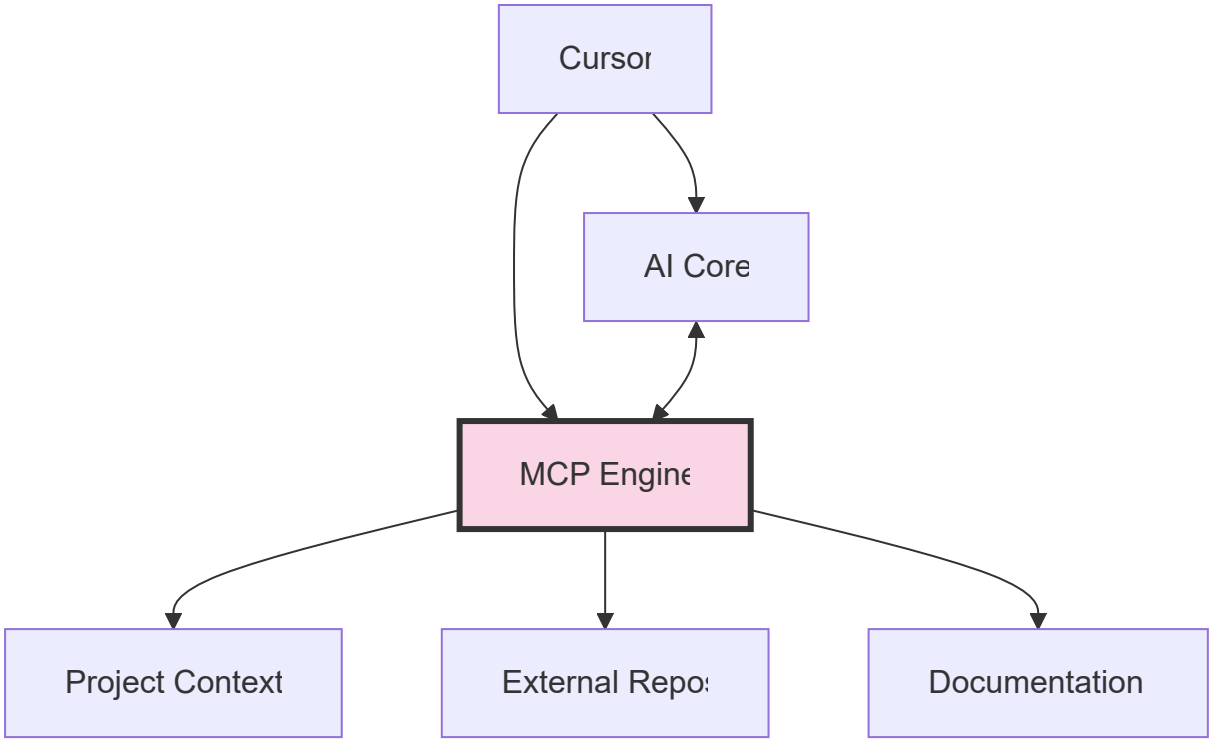
5.2 Cursorによる高度なAIコーディング連携

Cursorは、AIを中心に設計された新世代のコードエディタで、MCPのネイティブサポートにより強力な開発環境を提供します。

5.2.1 Cursorの基本機能とMCP連携

Cursorの主な特徴：

- AIを活用したコード生成と編集
- コンテキスト認識コード補完
- リアルタイムエラー解決と最適化提案
- MCPによる外部知識・コードリポジトリとの連携



5.2.2 Cursorの設定とMCP有効化

1. Cursor公式サイトからインストーラーをダウンロード
2. インストール後、設定メニューからMCP設定を開く
3. MCPエンドポイントと認証情報を設定

```
// ~/.cursor/config.json
{
  "mcp": {
    "enabled": true,
    "providers": [
      {
        "name": "Default MCP",
        "url": "https://mcp.example.com",
        "auth": {
          "type": "bearer",
          "token": "${MCP_TOKEN}"
        }
      }
    ],
    "contextSync": true,
    "historyRetention": "session"
  }
}
```

5.2.3 Cursorでのコーディング例

Cursorでは、MCPを活用した高度なコード生成と編集が可能です：

```
// コマンド入力（Ctrl+K）
GitHubからのusers/authenticationモジュールの実装パターンを参考に、
```

```
OAuthベースのユーザー認証システムをTypeScriptで実装してください。
セキュリティのベストプラクティスに従い、JWT使用を前提とします。
```

このコマンドに対し、CursorはMCP経由でGitHubリポジトリから関連コードパターンを検索し、それを参考にした最適な実装を提案します。

5.2.4 Cursorの高度な機能

Cursorには以下のような高度な機能があります：

- **コード診断**: 既存コードの問題点を分析し改善提案
- **リファクタリング支援**: 大規模コードのリファクタリングをAIとMCPで支援
- **ペアプログラミング**: AIとの対話的なコーディングセッション
- **テスト自動生成**: コードから適切なテストケースを自動生成

> **プロジェクト事例**: *あるフィンテックスタートアップでは、CursorのMCP連携機能を使って、社内のセキュリティガイドラインとGitHubのベストプラクティスを組み合わせたコード生成環境を構築しました。開発者は新機能実装時に「社内ガイドラインに準拠したAPI実装」のようなプロンプトを使うだけで、セキュリティチームの承認を得やすいコードを生成できるようになりました。これにより、セキュリティレビューのサイクルが平均2週間から3日に短縮されました。*

5.3 Windsurfによるブラウザベース開発環境

Windsurfは、ブラウザ上で動作する次世代のAI強化開発環境で、マルチAIモデル対応とMCP連携が特徴です。

5.3.1 Windsurfの概要

Windsurfの主な特徴：

- ブラウザベースのフルIDE機能
- 複数のAIモデルを状況に応じて選択・切替
- リアルタイムコラボレーション機能
- 拡張性の高いプラグインシステム
- 完全なMCPネイティブサポート

5.3.2 Windsurfのセットアップ

1. Windsurf公式サイトでアカウント作成
2. プロジェクトの作成またはインポート
3. 設定メニューからMCPを構成

```
// Windsurf MCP Configuration
windsurf.configureMCP({
  endpoints: [
    {
      name: "Primary MCP",
      url: "https://mcp.example.com",
      default: true
    },
    {
      name: "Backup MCP",
```

```
      url: "https://backup-mcp.example.com",
      fallback: true
    }
  ],

  services: {
    github: {
      enabled: true,
      repositories: ["user/repo1", "org/repo2"]
    },
    obsidian: { enabled: true },
    custom: {
      enabled: true,
      url: "https://custom-service.example.com",
      authMethod: "oauth"
    }
  },

  ai: {
    preferredModels: ["claude-3", "gpt-4"],
    contextStrategy: "adaptive"
  }
});
```

5.3.3 Windsurfの実践的活用

Windsurfでは、AIチャットパネルからMCPクエリを実行できます：

```
@github プロジェクトorg/repoの最近のPRで、パフォーマンス改善に関するものを要約して

@obsidian パフォーマンス最適化のベストプラクティスノートを参照して、
このコードに適用できる改善点を提案して
```

5.3.4 Windsurfのユニーク機能

Windsurfには他の開発環境にない特徴があります：

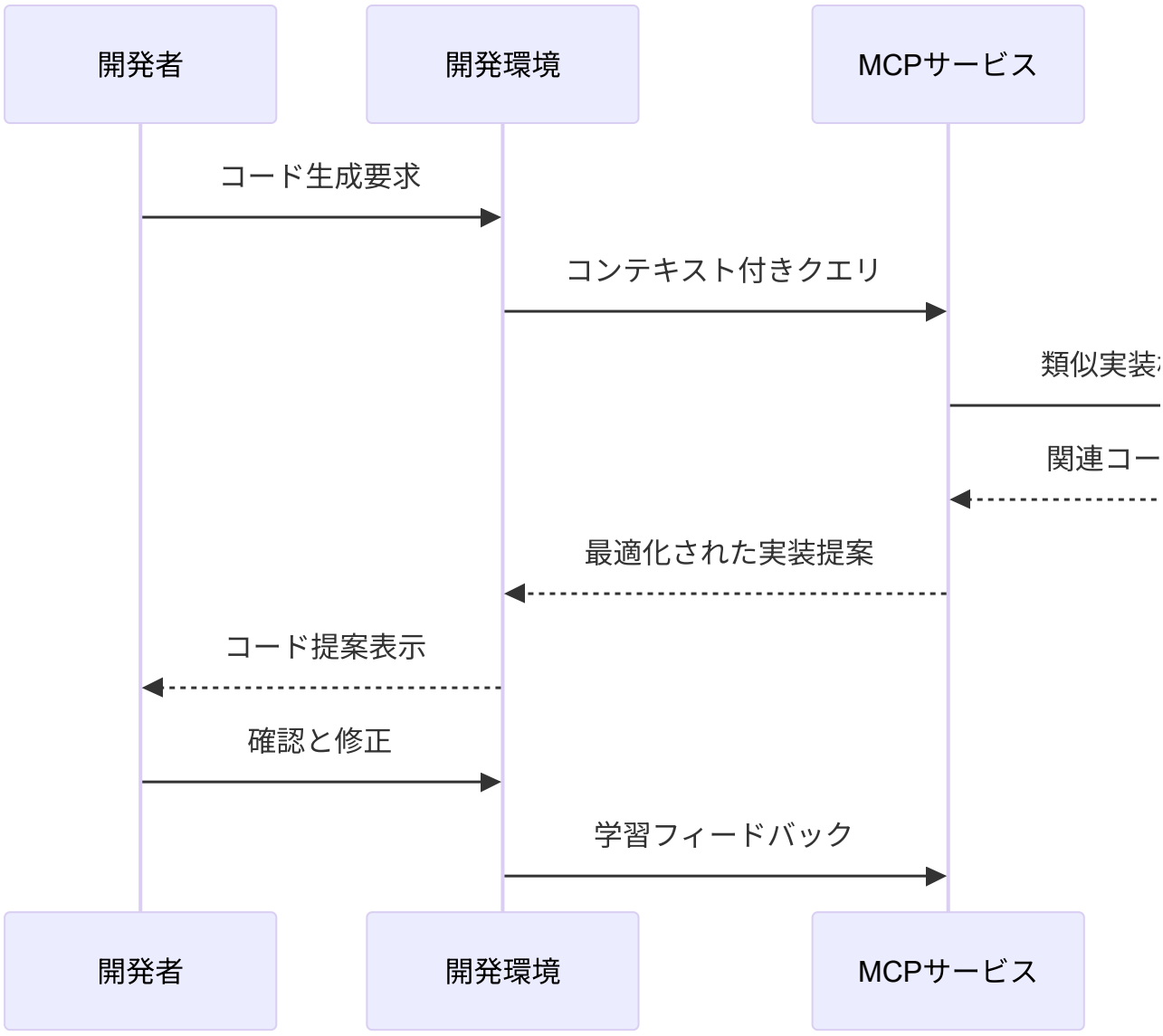
- **マルチAIスイッチング**: タスクに最適なAIモデルを自動選択
- **コラボレーティブMCP**: チーム全体でMCPコンテキストを共有
- **ブラウザネイティブ**: ローカル環境構築不要でどこからでもアクセス
- **ビジュアルフィードバック**: コード変更の影響をリアルタイムビジュアライズ

> **若手の疑問解決**: *「Windsurfって、ブラウザベースだとパフォーマンスや機能に制限がありそう...」*
>
> 最新のWebAssemblyとブラウザAPIの進化により、Windsurfのパフォーマンスはネイティブアプリに迫っています。特に日常的な開発タスクでは違いを感じにくいレベルです。オフライン作業やGPUアクセスなど一部の高度な機能には制限がありますが、代わりにどこからでもアクセスできる利便性とチームコラボレーション機能が強みです。

5.4 開発環境におけるMCP活用パターン

様々な開発環境に共通するMCP活用パターンを紹介します。

5.4.1 コード生成と改善



5.4.2 ドキュメント生成

開発環境からMCPを通じて最適なドキュメントを生成：

```
# 以下のコードのドキュメントをMCPで生成
def process_transaction(transaction_data, user_id=None, options=None):
    """
    トランザクションを処理し、結果を返す

    Parameters:
    -----------
    transaction_data : dict
        処理するトランザクションデータ
    user_id : str, optional
        関連するユーザーID
    options : dict, optional
        処理オプション

    Returns:
    --------
    dict
```

```
        処理結果と状態情報を含む辞書

    Raises:
    -------
    ValueError
        無効なトランザクションデータが提供された場合
    AuthError
        ユーザー認証に失敗した場合

    Notes:
    ------
    このメソッドは外部決済プロバイダーとの通信を行い、
    ネットワークの状態によっては処理に時間がかかる場合があります。
    高負荷環境では非同期バージョンのprocess_transaction_asyncの使用を検討してください。

    Examples:
    ---------
    >>> result = process_transaction({"amount": 100, "currency": "USD"},
user_id="user123")
    >>> print(result["status"])
    'completed'
    """
    # 実装...
```

5.4.3 知識統合クエリ

```
# VS Code、Cursor、Windsurfでの共通MCPクエリパターン

@github 現在のプロジェクトに関連するセキュリティ問題をチェック

@documentation このクラスの設計意図とユースケースを説明

@codebase この関数の呼び出し階層と依存関係を表示

@tickets このモジュールに関連する未解決のタスクを一覧表示
```

5.4.4 開発環境選択の判断基準

機能	VS Code	Cursor	Windsurf
MCP統合レベル	拡張機能経由	ネイティブ	ネイティブ
AI機能充実度	中（拡張次第）	高	非常に高い
カスタマイズ性	非常に高い	中	中〜高
チーム連携	限定的	中程度	強力
導入の容易さ	要拡張設定	容易	非常に容易
オフライン対応	完全対応	対応	限定的

> **失敗から学ぶ**: *「私たちのチームでは、最初からWindsurfの高度な機能に飛びついたものの、既存のワークフローとの統合に苦労しました。後から学んだのは、まずVS Codeの拡張からMCP機能を試*

> *し、徐々に移行するのが効果的だということです。ツールよりもプロトコルとワークフローの理解を優先すべきでした。」* - シニア開発マネージャー

開発環境MCP連携チェックリスト

- [ ] 開発チームの既存ワークフローとの互換性確認
- [ ] 必要なリポジトリとサービスへのアクセス権設定
- [ ] コード生成ポリシーとレビュー基準の策定
- [ ] MCP経由でのコード共有とプライバシーポリシーの確認
- [ ] チーム内トレーニングと導入計画の策定
- [ ] パイロットプロジェクトでの試験運用
- [ ] フィードバックループの確立と継続的改善

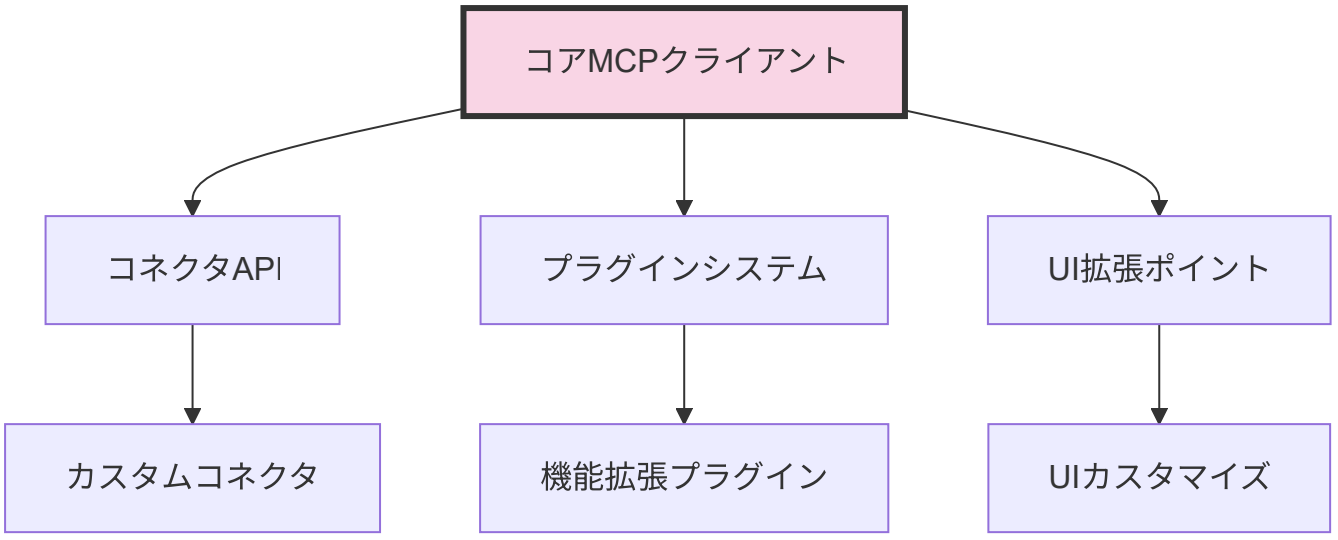
第6章: クライアントカスタマイズと拡張の実装

6.1 MCPクライアント拡張の基礎

MCPクライアントは様々な方法で拡張可能です。基本的な拡張方法を解説します。

6.1.1 拡張アーキテクチャの概要

MCPクライアント拡張は以下のレイヤーで構成されています：



1. **コネクタAPI**: 新しいサービスとの連携を追加
2. **プラグインシステム**: クライアントの機能を拡張
3. **UI拡張ポイント**: インターフェースをカスタマイズ

6.1.2 拡張の種類

MCPクライアントで実装できる主な拡張：

1. **サービスコネクタ**: 新しいMCP対応サービスとの連携
2. **プロセッサ**: メッセージの前処理や後処理
3. **UIコンポーネント**: カスタムパネルや視覚化
4. **コマンド**: 新しいユーザーコマンドの追加
5. **テンプレート**: 再利用可能なプロンプトや設定

6.2 カスタムMCPコネクタの開発

新しいサービスへの接続を実現するMCPコネクタの開発方法を解説します。

6.2.1 コネクタの基本構造

```
// カスタムMCPコネクタの基本構造
import { MCPConnector, ConnectorConfig, QueryRequest, QueryResponse } from 'mcp-core';

interface MyServiceConfig extends ConnectorConfig {
  baseUrl: string;
  apiKey?: string;
  options?: {
```

```
    timeout?: number;
    cacheResults?: boolean;
  };
}

class MyServiceConnector implements MCPConnector<MyServiceConfig> {
  // コネクタのメタデータ
  public readonly id = 'my-service';
  public readonly name = 'My Custom Service';
  public readonly version = '1.0.0';

  private config: MyServiceConfig;
  private authToken: string | null = null;

  // 初期化メソッド
  async initialize(config: MyServiceConfig): Promise<void> {
    // 設定の検証
    if (!config.baseUrl) {
      throw new Error('baseUrl is required');
    }

    this.config = {
      ...config,
      options: {
        timeout: config.options?.timeout || 30000,
        cacheResults: config.options?.cacheResults ?? true
      }
    };

    // 必要な初期化処理
    await this.setupConnection();
  }

  // 認証メソッド
  async authenticate(credentials: any): Promise<boolean> {
    try {
      // サービスへの認証リクエスト
      const response = await fetch(`${this.config.baseUrl}/auth`, {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json'
        },
        body: JSON.stringify(credentials)
      });

      if (!response.ok) {
        throw new Error(`Authentication failed: ${response.statusText}`);
      }

      const authData = await response.json();
      this.authToken = authData.token;
      return true;
    } catch (error) {
      console.error('Authentication error:', error);
      return false;
    }
  }
```

```
// クエリ実行メソッド
async query(request: QueryRequest): Promise<QueryResponse> {
  // 認証確認
  if (!this.authToken) {
    throw new Error('Not authenticated');
  }

  try {
    // サービスへのクエリリクエスト
    const response = await fetch(`${this.config.baseUrl}/query`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        'Authorization': `Bearer ${this.authToken}`
      },
      body: JSON.stringify(request.params)
    });

    if (!response.ok) {
      throw new Error(`Query failed: ${response.statusText}`);
    }

    const data = await response.json();

    return {
      status: 'success',
      data: data,
      metadata: {
        source: this.id,
        timestamp: new Date().toISOString()
      }
    };
  } catch (error) {
    console.error('Query error:', error);
    return {
      status: 'error',
      error: {
        message: error.message,
        code: 'QUERY_ERROR'
      },
      metadata: {
        source: this.id,
        timestamp: new Date().toISOString()
      }
    };
  }
}

// リソース処理メソッド
async resolveResource(resourceUri: string): Promise<any> {
  // リソースURIの解析と処理
  const parsedUri = new URL(resourceUri);
  // ...リソース固有の処理ロジック
}

// 後処理とクリーンアップ
```

```
  async dispose(): Promise<void> {
    // 接続終了処理
    this.authToken = null;
    // その他のクリーンアップ処理
  }

  // 内部ヘルパーメソッド
  private async setupConnection(): Promise<void> {
    // 初期接続設定
  }
}

// コネクタをエクスポート
export default MyServiceConnector;
```

6.2.2 コネクタのテストと検証

コネクタを開発する際のテスト手法：

```
// MCPコネクタのテスト例
import { TestConnectorFramework } from 'mcp-testing';
import MyServiceConnector from './my-service-connector';
import { createMockServer } from './test-utils';

describe('MyServiceConnector', () => {
  let connector: MyServiceConnector;
  let mockServer: any;

  beforeEach(async () => {
    // モックサーバーのセットアップ
    mockServer = createMockServer();
    await mockServer.start();

    // コネクタのインスタンス化
    connector = new MyServiceConnector();
    await connector.initialize({
      baseUrl: mockServer.url,
      apiKey: 'test-key'
    });
  });

  afterEach(async () => {
    // クリーンアップ
    await connector.dispose();
    await mockServer.stop();
  });

  test('認証が成功すること', async () => {
    // 認証のモック応答を設定
    mockServer.addMockResponse('/auth', {
      status: 200,
      body: { token: 'mock-token' }
    });

    const result = await connector.authenticate({
      username: 'testuser',
```

```
      password: 'testpass'
    });

    expect(result).toBe(true);
  });

  test('基本的なクエリが正常に動作すること', async () => {
    // 認証
    mockServer.addMockResponse('/auth', {
      status: 200,
      body: { token: 'mock-token' }
    });

    await connector.authenticate({
      username: 'testuser',
      password: 'testpass'
    });

    // クエリのモック応答を設定
    mockServer.addMockResponse('/query', {
      status: 200,
      body: {
        items: [
          { id: 1, name: 'Item 1' },
          { id: 2, name: 'Item 2' }
        ],
        total: 2
      }
    });

    const response = await connector.query({
      resource: 'my-service://items',
      params: { filter: 'all' }
    });

    expect(response.status).toBe('success');
    expect(response.data.items).toHaveLength(2);
    expect(response.data.total).toBe(2);
  });

  // その他のテストケース...
});
```

> **若手の疑問解決**: *「コネクタのエラー処理はどれくらい重要？」*
>
> エラー処理はMCPコネクタにおいて極めて重要です。サービスとの通信は様々な理由（ネットワーク問題、認証エラー、サービス側の制限など）で失敗する可能性があり、適切なエラーハンドリングがないとユーザー体験が大きく低下します。特に重要なのは：
>
> 1. 詳細なエラーメッセージ（デバッグ可能な情報）
> 2. リトライ戦略（一時的な問題に対応）
> 3. 代替アクション（エラー時の別の選択肢提示）
> 4. ユーザーフレンドリーなエラー表示

6.2.3 コネクタパッケージング

コネクタを配布可能な形にパッケージ化する方法：

```
// package.json
{
  "name": "mcp-connector-myservice",
  "version": "1.0.0",
  "description": "MCP connector for My Custom Service",
  "main": "dist/index.js",
  "types": "dist/index.d.ts",
  "scripts": {
    "build": "tsc",
    "test": "jest",
    "prepare": "npm run build"
  },
  "keywords": ["mcp", "connector", "myservice"],
  "author": "Your Name",
  "license": "MIT",
  "dependencies": {
    "mcp-core": "^1.0.0"
  },
  "devDependencies": {
    "typescript": "^4.9.5",
    "jest": "^29.5.0",
    "mcp-testing": "^1.0.0"
  },
  "mcpConnector": {
    "id": "my-service",
    "name": "My Service Connector",
    "supportedClients": ["claude-desktop", "vscode", "cursor"],
    "configSchema": {
      "type": "object",
      "properties": {
        "baseUrl": {
          "type": "string",
          "description": "API base URL"
        },
        "apiKey": {
          "type": "string",
          "description": "API key (optional)"
        }
      },
      "required": ["baseUrl"]
    }
  }
}
```

6.3 クライアント機能拡張の開発

MCPクライアントの機能を拡張するプラグインの開発方法を解説します。

6.3.1 Claude Desktop拡張

```
// Claude Desktop Extension Example
// ~/.claude/extensions/project-analyzer.js
```

```
module.exports = {
  id: 'project-analyzer',
  name: 'Project Analyzer',
  version: '1.0.0',

  // 初期化時に呼ばれる
  async initialize(claudeApi) {
    // Claude Desktop APIへの参照を保存
    this.claude = claudeApi;

    // コマンドの登録
    this.claude.commands.register('analyze-project', this.analyzeProject.bind(this));

    // サイドパネルの追加
    this.claude.ui.registerPanel({
      id: 'project-analysis',
      title: 'Project Analysis',
      icon: 'chart-bar',
      component: 'project-analysis-panel'
    });

    // コンテキストメニューの拡張
    this.claude.ui.registerContextMenu({
      id: 'repo-context',
      target: 'repository-item',
      items: [{
        id: 'analyze',
        label: 'Analyze Repository',
        command: 'analyze-project'
      }]
    });

    console.log('Project Analyzer extension initialized');
  },

  // カスタムコマンドの実装
  async analyzeProject(params) {
    const { repositoryUrl } = params;

    try {
      // 進捗通知
      this.claude.notifications.show({
        type: 'info',
        message: `Analyzing repository: ${repositoryUrl}`,
        duration: 3000
      });

      // MCPを使ってリポジトリ情報を取得
      const repoData = await this.claude.mcp.query({
        resource: repositoryUrl,
        requestType: 'analyze',
        params: {
          depth: 2,
          includeStats: true
        }
      });
```

```
      // 分析プロンプトの生成
      const analysisPrompt = `
        以下のリポジトリ情報を分析し、プロジェクトの概要、構造、主要コンポーネント、
        潜在的な問題点、改善提案をまとめてください：

        ${JSON.stringify(repoData, null, 2)}
      `;

      // Claudeに分析を依頼
      const analysis = await this.claude.generate({
        prompt: analysisPrompt,
        maxTokens: 2000
      });

      // 結果をパネルに表示
      this.claude.ui.updatePanel('project-analysis', {
        content: analysis,
        metadata: {
          repository: repositoryUrl,
          analyzedAt: new Date().toISOString()
        }
      });

      return {
        success: true,
        message: 'Repository analysis completed'
      };
    } catch (error) {
      console.error('Analysis error:', error);

      this.claude.notifications.show({
        type: 'error',
        message: `Analysis failed: ${error.message}`,
        duration: 5000
      });

      return {
        success: false,
        error: error.message
      };
    }
  },

  // クリーンアップ処理
  async dispose() {
    // 登録したリソースの解放
    console.log('Project Analyzer extension disposed');
  }
};
```

6.3.2 VS Code拡張

```
// VS Code MCP Extension Example
import * as vscode from 'vscode';
import { MCPClient } from 'mcp-client';
```

```typescript
export function activate(context: vscode.ExtensionContext) {
  // MCPクライアントの初期化
  const mcpClient = new MCPClient();

  // 設定から構成を読み込む
  const config = vscode.workspace.getConfiguration('mcp');
  const endpoint = config.get<string>('endpoint');
  const authToken = context.secrets.get('mcp.authToken');

  if (endpoint && authToken) {
    mcpClient.configure({
      endpoint,
      auth: {
        type: 'bearer',
        token: authToken
      }
    });

    vscode.window.showInformationMessage('MCP Client connected');
  }

  // コマンドの登録
  const queryCommand = vscode.commands.registerCommand('mcp.query', async () => {
    // エディタの選択テキストを取得
    const editor = vscode.window.activeTextEditor;
    const selection = editor?.selection;
    const selectedText = editor?.document.getText(selection);

    // クエリ入力ボックスを表示
    const query = await vscode.window.showInputBox({
      prompt: 'Enter MCP query',
      placeHolder: 'Search knowledge base or ask question...',
      value: selectedText
    });

    if (!query) return;

    try {
      // プログレス表示
      vscode.window.withProgress({
        location: vscode.ProgressLocation.Notification,
        title: 'Querying MCP...',
        cancellable: true
      }, async (progress, token) => {
        // 現在のファイルコンテキストを取得
        const fileContext = {
          path: editor?.document.uri.fsPath,
          language: editor?.document.languageId,
          selection: {
            text: selectedText,
            startLine: selection?.start.line,
            endLine: selection?.end.line
          }
        };

        // MCPクエリを実行
```

```
        const result = await mcpClient.query({
          query,
          context: {
            file: fileContext,
            workspace: vscode.workspace.name
          }
        });

        // 結果表示用のウェブビューパネルを作成
        const panel = vscode.window.createWebviewPanel(
          'mcpResult',
          'MCP Query Result',
          vscode.ViewColumn.Beside,
          {
            enableScripts: true
          }
        );

        // 結果を表示
        panel.webview.html = generateResultHtml(query, result);

        return result;
      });
    } catch (error) {
      vscode.window.showErrorMessage(`MCP query failed: ${error.message}`);
    }
  });

  // その他のコマンドや機能...

  context.subscriptions.push(queryCommand);
}

// 結果表示HTMLの生成
function generateResultHtml(query: string, result: any): string {
  return `
    <!DOCTYPE html>
    <html>
    <head>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
      <title>MCP Result</title>
      <style>
        body { font-family: system-ui; padding: 20px; }
        .query { font-weight: bold; margin-bottom: 10px; }
        .result { white-space: pre-wrap; }
        .source { font-size: 0.8em; color: #666; margin-top: 10px; }
      </style>
    </head>
    <body>
      <div class="query">Query: ${escapeHtml(query)}</div>
      <div class="result">${formatResult(result)}</div>
      <div class="source">Source: ${result.metadata?.source || 'Unknown'}</div>
    </body>
    </html>
  `;
}
```

```
// HTMLエスケープ
function escapeHtml(text: string): string {
  return text
    .replace(/&/g, '&amp;')
    .replace(/</g, '&lt;')
    .replace(/>/g, '&gt;')
    .replace(/"/g, '&quot;')
    .replace(/'/g, '&#039;');
}

// 結果のフォーマット
function formatResult(result: any): string {
  if (typeof result.data === 'string') {
    return escapeHtml(result.data);
  } else {
    return escapeHtml(JSON.stringify(result.data, null, 2));
  }
}

export function deactivate() {
  // クリーンアップ処理
}
```

> **プロジェクト事例**: *ある企業の研究開発部門では、VS Code向けにカスタムMCP拡張を開発し、社内の実験データベース、論文リポジトリ、特許データベースをシームレスに連携させました。研究者は実験コードを書きながら、関連論文や過去の実験結果に直接アクセスできるようになり、研究サイクルが加速しました。特にAIを活用した仮説生成と検証のサイクルが短縮され、実験設計の質が向上したと報告されています。*

6.4 UIカスタマイズと体験の最適化

MCPクライアントのUIをカスタマイズして最適なユーザー体験を実現する方法を解説します。

6.4.1 カスタムビューとパネル

Claude Desktopにカスタムビューを追加する例：

```
// Claude Desktop UI Extension
claude.ui.registerView({
  id: 'knowledge-graph',
  title: 'Knowledge Graph',
  icon: 'diagram-project',
  component: KnowledgeGraphComponent,
  location: 'sidebar',
  options: {
    width: 300,
    minWidth: 200,
    initialState: {
      expanded: true
    }
  }
});

// カスタムコンポーネントの定義
```

```
function KnowledgeGraphComponent(props) {
  const { claudeApi, state } = props;
  const [graphData, setGraphData] = useState(null);

  // データ読み込み
  useEffect(() => {
    async function loadGraphData() {
      const data = await claudeApi.mcp.query({
        resource: 'knowledge://current-context',
        requestType: 'graph',
        params: {
          depth: 2,
          relations: ['references', 'dependencies']
        }
      });

      setGraphData(data);
    }

    loadGraphData();
  }, [claudeApi, state.context]);

  // グラフ描画
  if (!graphData) {
    return <div>Loading knowledge graph ...</div>;
  }

  return (
    <div className="knowledge-graph-container">
      <GraphVisualization data={graphData} />
      <div className="graph-controls">
        <button onClick={() => claudeApi.mcp.refreshContext()}>
          Refresh
        </button>
        <select onChange={(e) => claudeApi.mcp.setGraphDepth(e.target.value)}>
          <option value="1">Depth 1</option>
          <option value="2" selected>Depth 2</option>
          <option value="3">Depth 3</option>
        </select>
      </div>
    </div>
  );
}
```

6.4.2 カスタムテーマとスタイル

```
/* Claude Desktop Custom Theme */
.claude-theme-custom {
  --primary-color: #3a7ca5;
  --secondary-color: #d9e5ec;
  --accent-color: #f9a826;
  --text-color: #2c3e50;
  --background-color: #f5f7fa;
  --panel-background: #ffffff;
  --border-color: #e1e5ea;
```

```
  /* フォントとタイポグラフィ */
  --font-main: 'Source Sans Pro', sans-serif;
  --font-code: 'Fira Code', monospace;
  --font-size-base: 15px;
  --line-height: 1.6;

  /* スペーシング */
  --spacing-unit: 8px;
  --padding-small: calc(var(--spacing-unit) * 1);
  --padding-medium: calc(var(--spacing-unit) * 2);
  --padding-large: calc(var(--spacing-unit) * 3);

  /* アニメーション */
  --transition-fast: 0.15s ease-in-out;
  --transition-normal: 0.25s ease-in-out;
}

/* MCP固有の要素スタイル */
.mcp-query-input {
  background-color: var(--panel-background);
  border: 1px solid var(--border-color);
  border-radius: 8px;
  padding: var(--padding-medium);
  font-family: var(--font-main);
  font-size: var(--font-size-base);
  transition: border-color var(--transition-fast);
}

.mcp-query-input:focus {
  border-color: var(--primary-color);
  outline: none;
  box-shadow: 0 0 0 2px rgba(58, 124, 165, 0.2);
}

.mcp-result-panel {
  background-color: var(--panel-background);
  border-radius: 8px;
  box-shadow: 0 2px 8px rgba(0, 0, 0, 0.05);
  overflow: hidden;
}

.mcp-result-panel .header {
  background-color: var(--secondary-color);
  padding: var(--padding-medium);
  font-weight: 600;
  display: flex;
  justify-content: space-between;
  align-items: center;
}

.mcp-result-panel .content {
  padding: var(--padding-medium);
  max-height: 400px;
  overflow-y: auto;
}
```

```css
.mcp-result-panel .source-label {
  font-size: 0.8em;
  color: #666;
  margin-top: var(--padding-small);
}

/* サービス固有のアイコンとバッジ */
.mcp-service-icon {
  width: 16px;
  height: 16px;
  display: inline-block;
  margin-right: 4px;
  vertical-align: middle;
}

.mcp-service-badge {
  display: inline-block;
  padding: 2px 6px;
  border-radius: 12px;
  font-size: 0.8em;
  font-weight: 600;
}

.mcp-service-github {
  background-color: #24292e;
  color: white;
}

.mcp-service-obsidian {
  background-color: #7e6bc4;
  color: white;
}

.mcp-service-redmine {
  background-color: #b41e3b;
  color: white;
}
```

6.4.3 カスタムコマンドとショートカット

```javascript
// Cursor Custom Commands
cursor.commands.register({
  id: 'mcp.searchCodeExamples',
  name: 'Search Code Examples',
  description: 'Search for related code examples via MCP',
  keybinding: 'Ctrl+Alt+E', // Windows/Linux
  macKeybinding: 'Cmd+Alt+E', // macOS

  execute: async (editor) => {
    // 現在のコードコンテキストを取得
    const selectedCode = editor.getSelectedText() || editor.getCurrentLine();
    const language = editor.getLanguage();

    // 検索クエリ入力を表示
    const query = await cursor.ui.showInputBox({
```

```
      title: 'Search Code Examples',
      prompt: 'Enter search terms or description',
      value: selectedCode.length > 80 ? selectedCode.substring(0, 80) + '...' :
selectedCode
    });

    if (!query) return;

    // プログレス表示
    cursor.ui.showProgress({
      title: 'Searching code examples...',
      cancellable: true
    });

    try {
      // MCPクエリを実行
      const examples = await cursor.mcp.query({
        resource: 'code-examples',
        params: {
          query: query,
          language: language,
          context: selectedCode,
          limit: 5
        }
      });

      // 結果が見つからない場合
      if (!examples || examples.length === 0) {
        cursor.ui.showMessage('No code examples found for your query.');
        return;
      }

      // 結果表示
      const selection = await cursor.ui.showQuickPick({
        title: 'Found Code Examples',
        items: examples.map((ex, i) => ({
          id: `example-${i}`,
          label: ex.title || `Example ${i+1}`,
          description: ex.description || truncate(ex.code, 60),
          detail: `Source: ${ex.source} | Language: ${ex.language}`,
          example: ex
        }))
      });

      if (!selection) return;

      // 選択された例の詳細表示
      cursor.ui.showCodePreview({
        title: selection.label,
        code: selection.example.code,
        language: selection.example.language,
        actions: [
          {
            id: 'insert',
            label: 'Insert',
            callback: () => editor.insertText(selection.example.code)
          },
```

```
          {
            id: 'adapt',
            label: 'Adapt to Context',
            callback: async () => {
              const adapted = await cursor.ai.generate({
                prompt: `Adapt this code example to my current
context:\n\nExample:\n${selection.example.code}\n\nMy current code:\n${selectedCode}`,
                maxTokens: 1000
              });

              editor.insertText(adapted);
            }
          }
        ]
      });
    } catch (error) {
      cursor.ui.showErrorMessage(`Failed to search code examples: ${error.message}`);
    } finally {
      cursor.ui.hideProgress();
    }
  }
});

// ヘルパー関数
function truncate(str, length) {
  if (!str) return '';
  return str.length > length ? str.substring(0, length) + '...' : str;
}
```

> **ベテランの知恵袋**: *「MCPクライアントのカスタマイズでは、チームの実際のワークフローに合わせた最適化が重要です。私の経験では、最も効果的なのは『頻度×煩雑さ』の高いタスクを特定し、それを1-2キーで呼び出せるショートカットに割り当てることです。日々の作業で数十回実行する3-4ステップの操作を自動化するだけで、チーム全体の生産性が驚くほど向上します。」* - シニアデベロッパーエクスペリエンス担当

6.4.4 インテリジェントなコンテキスト管理

MCPクライアントの大きな強みは、コンテキストの管理と転送です。これを最適化する方法：

```
// Windsurf Context Management Extension
windsurf.extensions.register({
  id: 'smart-context',
  name: 'Smart Context Manager',

  initialize() {
    // コンテキスト変更の監視
    windsurf.context.onChanged(this.handleContextChange.bind(this));

    // コマンドの登録
    windsurf.commands.register('smart-context.save', this.saveContext.bind(this));
    windsurf.commands.register('smart-context.restore',
this.restoreContext.bind(this));

    // キーバインディングの登録
    windsurf.keybindings.register({
```

```
    command: 'smart-context.save',
    key: 'Ctrl+Shift+S',
    mac: 'Cmd+Shift+S'
  });

  windsurf.keybindings.register({
    command: 'smart-context.restore',
    key: 'Ctrl+Shift+R',
    mac: 'Cmd+Shift+R'
  });

  // ツールバーボタンの追加
  windsurf.ui.toolbar.addButton({
    id: 'context-snapshot',
    icon: 'camera',
    tooltip: 'Save Context Snapshot',
    command: 'smart-context.save'
  });

  // コンテキストライブラリの初期化
  this.contextLibrary = new Map();
},

// コンテキスト変更の処理
async handleContextChange(newContext) {
  // コンテキストの分析
  const analysis = await windsurf.ai.analyze({
    content: newContext,
    task: 'extract-topics'
  });

  // 関連コンテキストの提案
  if (analysis.topics && analysis.topics.length > 0) {
    const relatedContexts = this.findRelatedContexts(analysis.topics);

    if (relatedContexts.length > 0) {
      windsurf.ui.showSuggestion({
        message: 'Related contexts available',
        items: relatedContexts.map(ctx => ({
          label: ctx.name,
          description: ctx.description,
          action: () => this.switchToContext(ctx.id)
        }))
      });
    }
  }
},

// コンテキストの保存
async saveContext() {
  const currentContext = windsurf.context.current;

  // コンテキスト名の入力を促す
  const name = await windsurf.ui.showInputBox({
    title: 'Save Context Snapshot',
    prompt: 'Enter a name for this context',
    value: `Context-${new Date().toISOString().slice(0, 10)}`
```

```
    });

    if (!name) return;

    // コンテキストの説明生成
    const description = await windsurf.ai.summarize({
      content: currentContext.content,
      maxLength: 100
    });

    // コンテキストの保存
    const contextId = `ctx-${Date.now()}`;
    this.contextLibrary.set(contextId, {
      id: contextId,
      name,
      description,
      content: currentContext.content,
      metadata: {
        savedAt: new Date().toISOString(),
        topics: currentContext.topics || []
      }
    });

    windsurf.ui.showMessage(`Context saved as "${name}"`);
  },

  // コンテキストの復元
  async restoreContext() {
    if (this.contextLibrary.size === 0) {
      windsurf.ui.showMessage('No saved contexts available');
      return;
    }

    // 保存されたコンテキストの一覧表示
    const contexts = Array.from(this.contextLibrary.values());
    const selection = await windsurf.ui.showQuickPick({
      title: 'Restore Context',
      items: contexts.map(ctx => ({
        id: ctx.id,
        label: ctx.name,
        description: ctx.description,
        detail: `Saved: ${new Date(ctx.metadata.savedAt).toLocaleString()}`
      }))
    });

    if (!selection) return;

    // 選択されたコンテキストを復元
    this.switchToContext(selection.id);
  },

  // コンテキスト切り替え
  switchToContext(contextId) {
    const context = this.contextLibrary.get(contextId);
    if (!context) return;

    windsurf.context.set(context.content);
```

```
    windsurf.ui.showMessage(`Switched to context: ${context.name}`);
  },

  // 関連コンテキストの検索
  findRelatedContexts(topics) {
    const relatedContexts = [];

    for (const context of this.contextLibrary.values()) {
      // トピックの重複度をスコア化
      const commonTopics = topics.filter(topic =>
        context.metadata.topics.includes(topic)
      );

      const score = commonTopics.length / topics.length;

      // スコアが閾値を超えた場合、関連コンテキストとして追加
      if (score > 0.3) {
        relatedContexts.push({
          ...context,
          relevanceScore: score
        });
      }
    }

    // 関連度スコアでソート
    return relatedContexts.sort((a, b) => b.relevanceScore - a.relevanceScore);
  }
});
```

6.5 拡張機能の配布とエコシステムへの参加

MCPクライアント拡張を共有し、コミュニティに貢献する方法について解説します。

6.5.1 拡張機能パッケージの構成

```
my-mcp-extension/
├── README.md               # 説明ドキュメント
├── package.json            # パッケージメタデータ
├── src/                    # ソースコード
│   ├── index.js            # メインエントリーポイント
│   ├── components/         # UIコンポーネント
│   ├── connectors/         # MCPコネクタ
│   └── utils/              # ユーティリティ関数
├── assets/                 # アイコンなどのリソース
├── config/                 # 設定スキーマ
└── examples/               # 使用例
```

典型的な `package.json` の例：

```
{
  "name": "mcp-extension-knowledge-graph",
  "version": "1.0.0",
  "description": "Knowledge Graph visualization for MCP clients",
```

```
  "main": "src/index.js",
  "author": "Your Name",
  "license": "MIT",
  "engines": {
    "claudedesktop": ">=1.5.0",
    "vscode": ">=1.60.0",
    "cursor": ">=0.9.0",
    "windsurf": ">=0.3.0"
  },
  "dependencies": {
    "d3": "^7.0.0",
    "lodash": "^4.17.21"
  },
  "mcpExtension": {
    "id": "knowledge-graph",
    "name": "Knowledge Graph",
    "description": "Visualize knowledge connections across repositories",
    "icon": "assets/icon.png",
    "category": "visualization",
    "capabilities": [
      "ui",
      "query",
      "graph"
    ],
    "permissions": [
      "mcp.query",
      "ui.panel"
    ],
    "supportedClients": [
      "claudedesktop",
      "vscode",
      "cursor"
    ]
  }
}
```

6.5.2 拡張機能の公開

各プラットフォームの拡張機能マーケットプレイス向けの公開方法：

1. **Claude Desktop Extensions Hub**:

```
# Claude Desktop CLI経由での公開
claude-cli extension publish --directory ./my-mcp-extension
```

2. **VS Code Marketplace**:

```
# VS Code Extension公開
vsce package
vsce publish
```

3. **Cursor Extension Hub**:

```
# Cursor向け拡張機能公開
cursor-ext publish
```

6.5.3 拡張機能開発のベストプラクティス

効果的なMCP拡張機能開発のためのベストプラクティス：

1. **ユーザー中心設計**
   - 実際のワークフローを観察して設計する
   - 早期からユーザーテストを行う
   - 段階的な導入を考慮する
2. **パフォーマンス最適化**
   - 大量のデータを扱う際にはページング処理を実装
   - UI更新の効率化（不必要な再レンダリング防止）
   - バックグラウンド処理の適切な利用
3. **エラー処理と回復性**
   - ネットワークエラーに対する堅牢な処理
   - わかりやすいエラーメッセージ
   - 自動リトライと回復メカニズム
4. **クロスプラットフォーム対応**
   - 特定プラットフォーム機能への依存を最小化
   - 条件付きコードで分岐処理
   - 共通抽象化レイヤーの活用

> **失敗から学ぶ**: *「私たちの最初のMCP拡張機能では、大量のデータをフロントエンドですべて処理しようとして、ブラウザがクラッシュする問題が頻発しました。後から実装したストリーミング処理とサーバーサイド集約により、同じデータ量でもスムーズに動作するようになりました。MCPの大きな強みはデータ量ですが、それを扱う戦略が非常に重要です。」* - エンタープライズ拡張機能開発者

第6章まとめ

- MCPクライアントのカスタマイズと拡張により、組織固有のニーズに合わせた環境構築が可能
- コネクタ開発はMCPエコシステムを新しいサービスへと拡大する基本単位
- UI拡張とコマンド追加により、チーム特有のワークフローを効率化できる
- インテリジェントなコンテキスト管理がAIとの協働作業の核心部分
- 拡張機能の共有と標準化がMCPエコシステム全体の成長を促進

クライアント拡張技術選定チェックリスト

- [ ] 対象ユーザーの主要ワークフローとニーズの分析
- [ ] 拡張範囲の定義（コネクタ、UI、コマンド、テーマなど）
- [ ] サポート対象クライアントの選定と互換性確認
- [ ] 開発言語とフレームワークの選択
- [ ] デバッグと検証環境の構築
- [ ] パフォーマンス要件と制約の確認
- [ ] クロスプラットフォーム戦略の確立
- [ ] 配布と更新方法の決定

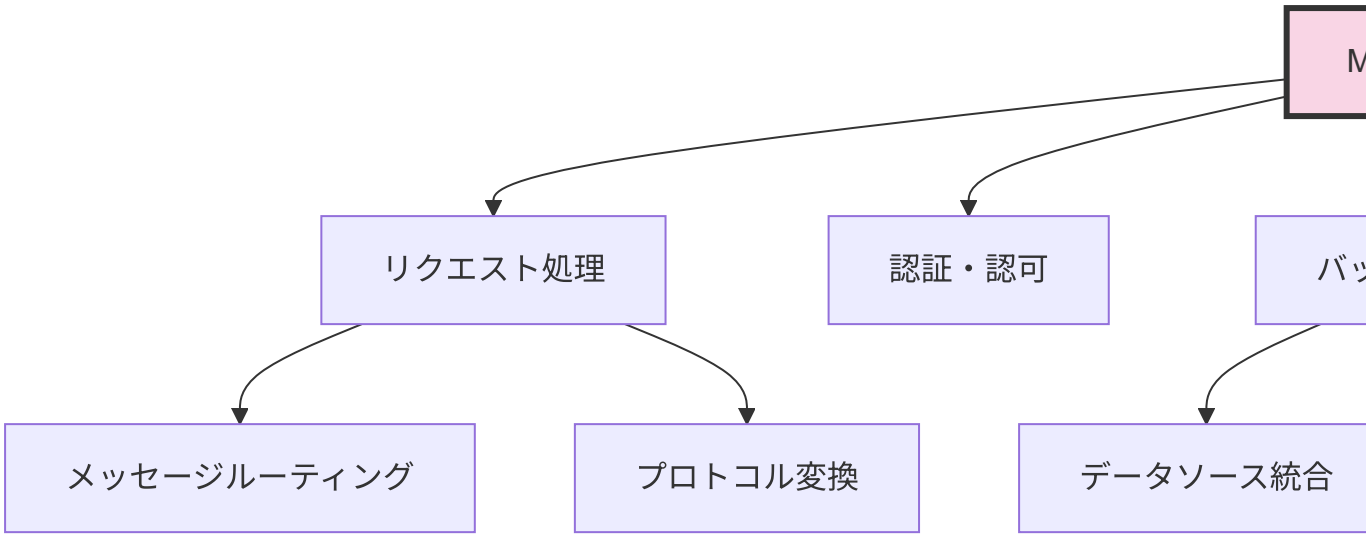
第7章: サーバー構築の基礎 - アーキテクチャと設計原則

7.1 MCPサーバーの概要と役割

MCPサーバーは、クライアントからのリクエストを処理し、様々なバックエンドシステムやデータソースとの統合を担うコンポーネントです。

7.1.1 MCPサーバーの基本機能

MCPサーバーが提供する基本機能は以下の通りです：



1. **リクエスト処理**: クライアントからのMCPメッセージを受信し解析
2. **認証・認可**: セキュアなアクセス制御とユーザー権限管理
3. **バックエンド連携**: 各種サービスやデータストアとの統合
4. **コンテキスト管理**: クエリコンテキストの保持と活用
5. **キャッシュ・最適化**: 応答時間短縮と負荷分散

7.1.2 MCPサーバーの種類

MCPサーバーには主に以下の3つの種類があります：

1. **専用MCPサーバー**
   - MCPプロトコル専用に実装されたサーバー
   - 最適化されたパフォーマンスと拡張性
   - 例: MCPHub, Enterprise MCP Server
2. **アダプターベースMCPサーバー**
   - 既存サービスの前にプロキシとして配置
   - 既存APIをMCPプロトコルに変換
   - 例: GitHub MCP Adapter, Obsidian MCP Bridge
3. **内蔵MCPサーバー**
   - 既存アプリケーションに組み込まれたMCP機能
   - ネイティブ統合によるパフォーマンス最適化
   - 例: MCPサポート付きのRedmine, GitBucket

> **若手の疑問解決**: *「MCPサーバーって結局APIサーバーと何が違うの？」*

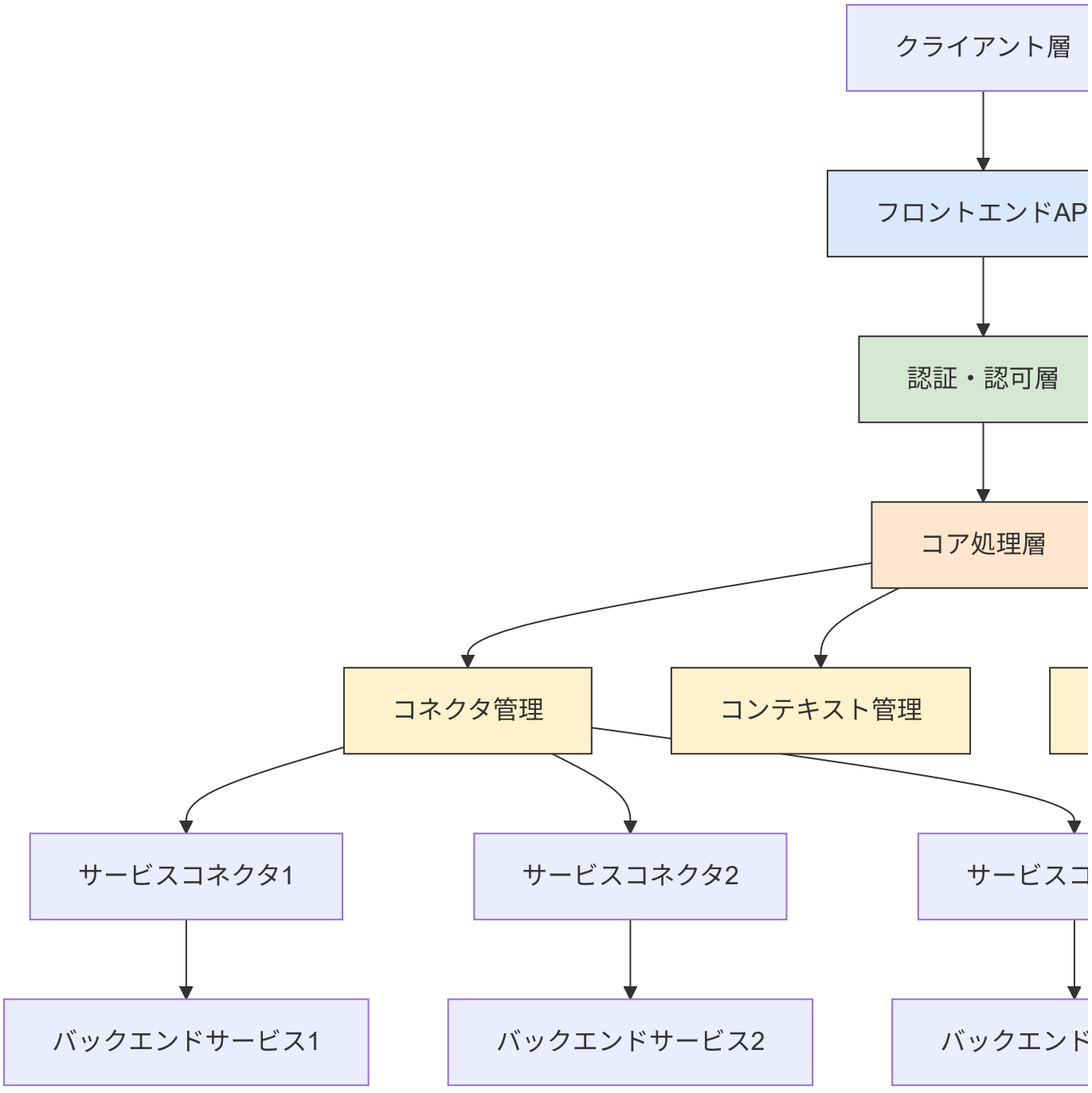
> MCPサーバーは通常のAPIサーバーの進化形と考えられます。主な違いは3つあります：
>
> 1. **コンテキスト管理**: 単なるリクエスト/レスポンスではなく、会話や作業の文脈を維持します
> 2. **セマンティック処理**: データレベルではなく、意味レベルでの処理が可能です
> 3. **統一インターフェース**: 複数の異なるサービスへの標準化されたアクセス方法を提供します
>
> 技術的には標準的なAPIサーバーをベースにしていることが多いですが、これらの機能が加わることで、より高度な統合体験を提供できます。

7.2 MCPサーバーアーキテクチャの設計

効果的なMCPサーバーを構築するための設計原則と参照アーキテクチャを解説します。

7.2.1 参照アーキテクチャ

標準的なMCPサーバーの参照アーキテクチャ：

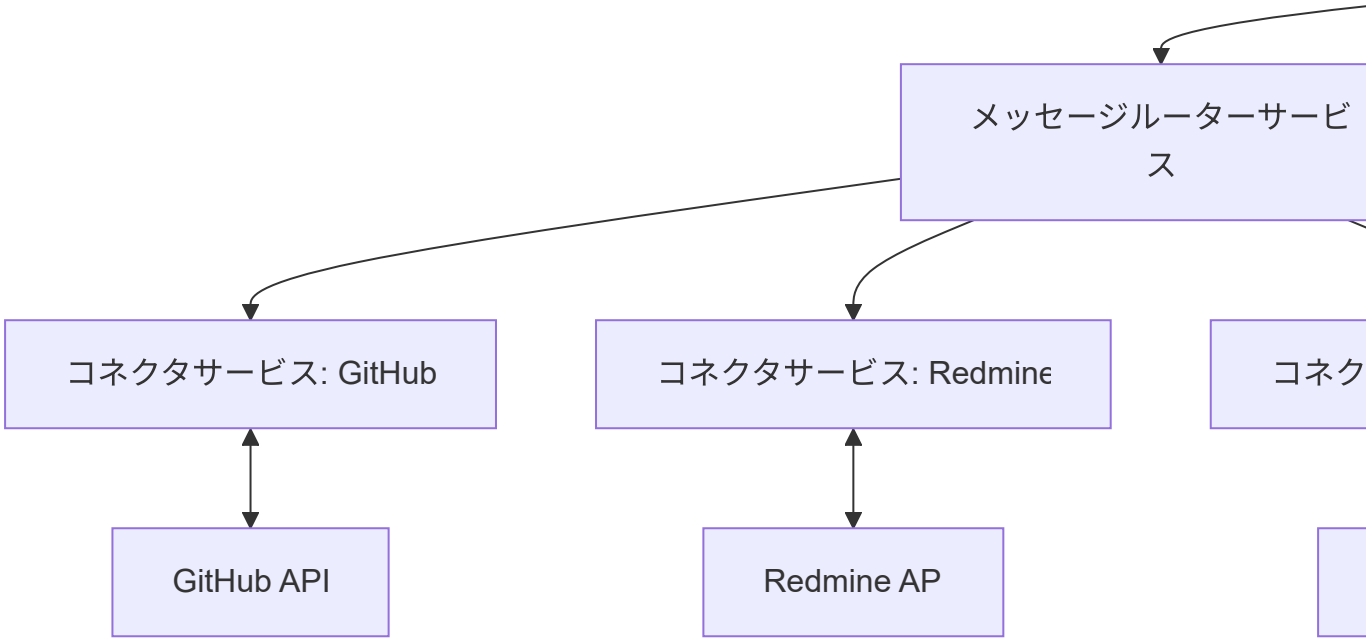


各レイヤーの役割：

1. **フロントエンドAPI層**
   - クライアントリクエストの受信
   - プロトコルバージョン管理
   - 初期バリデーション
2. **認証・認可層**
   - ユーザー認証
   - サービス認証
   - 権限チェック
3. **コア処理層**
   - メッセージルーティング
   - リクエスト変換
   - コンテキスト管理
4. **コネクタ層**
   - バックエンドサービス統合
   - プロトコル変換
   - レスポンス正規化

7.2.2 マイクロサービスベースの実装

大規模システム向けのマイクロサービスベースMCPアーキテクチャ：



このアーキテクチャの利点：

- 各コンポーネントの独立したスケーリング
- サービスごとの技術スタック最適化
- 段階的な展開と更新
- 耐障害性とフォールトトレランス

7.2.3 単一サーバー実装

小〜中規模向けの単一サーバー実装：

```
// Node.js MCPサーバーの基本構造例
const express = require('express');
const cors = require('cors');
const bodyParser = require('body-parser');
const jwt = require('jsonwebtoken');
const { v4: uuidv4 } = require('uuid');

// コネクタとコンテキスト管理の読み込み
const connectorManager = require('./connectors');
const contextManager = require('./context');

const app = express();
app.use(cors());
app.use(bodyParser.json());

// 認証ミドルウェア
const authenticate = (req, res, next) => {
  const authHeader = req.headers.authorization;
  if (!authHeader || !authHeader.startsWith('Bearer ')) {
    return res.status(401).json({ error: 'Unauthorized' });
  }

  const token = authHeader.split(' ')[1];
  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.user = decoded;
    next();
  } catch (err) {
    return res.status(401).json({ error: 'Invalid token' });
  }
};

// 認証エンドポイント
app.post('/auth', async (req, res) => {
  const { username, password } = req.body;

  try {
    // ユーザー認証ロジック
    const user = await authenticateUser(username, password);

    if (!user) {
      return res.status(401).json({ error: 'Invalid credentials' });
    }

    // JWTトークン生成
    const token = jwt.sign(
      { id: user.id, username: user.username, roles: user.roles },
      process.env.JWT_SECRET,
      { expiresIn: '8h' }
    );

    res.json({ token });
```

```
  } catch (err) {
    console.error('Authentication error:', err);
    res.status(500).json({ error: 'Authentication failed' });
  }
});

// MCP処理エンドポイント
app.post('/mcp', authenticate, async (req, res) => {
  try {
    const { version, requestType, resource, context, params } = req.body;

    // バージョン確認
    if (version !== '1.0' && version !== '1.1') {
      return res.status(400).json({
        error: 'Unsupported MCP version'
      });
    }

    // リクエストID生成
    const requestId = uuidv4();

    // コンテキスト管理
    let sessionContext = context?.sessionId ?
      await contextManager.getContext(context.sessionId) :
      await contextManager.createContext();

    // コンテキスト更新
    if (context?.metadata) {
      sessionContext = await contextManager.updateContext(
        sessionContext.id,
        context.metadata
      );
    }

    // リソース解析
    const resourceInfo = parseResource(resource);

    // コネクタ取得
    const connector = connectorManager.getConnector(resourceInfo.service);
    if (!connector) {
      return res.status(404).json({
        error: `Service not found: ${resourceInfo.service}`
      });
    }

    // アクセス権限確認
    const hasAccess = await checkAccess(req.user, resourceInfo);
    if (!hasAccess) {
      return res.status(403).json({
        error: 'Access denied'
      });
    }

    // リクエスト実行
    const result = await connector.executeRequest({
      requestType,
      resource: resourceInfo,
```

```
      params,
      context: sessionContext,
      user: req.user
    });

    // レスポンス形成
    res.json({
      version,
      status: 200,
      requestId,
      data: result.data,
      context: {
        sessionId: sessionContext.id,
        metadata: sessionContext.metadata
      }
    });
  } catch (err) {
    console.error('MCP request error:', err);

    res.status(err.status || 500).json({
      version: req.body.version,
      status: err.status || 500,
      requestId: uuidv4(),
      error: {
        code: err.code || 'INTERNAL_ERROR',
        message: err.message || 'Internal server error'
      }
    });
  }
});

// ストリーミングエンドポイント (WebSocket)
const wsServer = require('http').createServer(app);
const io = require('socket.io')(wsServer);

io.use((socket, next) => {
  // WebSocket認証処理
  const token = socket.handshake.auth.token;
  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    socket.user = decoded;
    next();
  } catch (err) {
    next(new Error('Authentication error'));
  }
});

io.on('connection', (socket) => {
  console.log('Client connected:', socket.id);

  socket.on('mcp.stream', async (request) => {
    try {
      // ストリーミングリクエスト処理
      const { resource, params, context } = request;

      // リソース解析とコネクタ取得
      const resourceInfo = parseResource(resource);
```

```
      const connector = connectorManager.getConnector(resourceInfo.service);

      if (!connector || !connector.supportsStreaming) {
        socket.emit('mcp.error', {
          message: 'Streaming not supported for this resource'
        });
        return;
      }

      // ストリーム開始
      const stream = await connector.createStream({
        resource: resourceInfo,
        params,
        context,
        user: socket.user
      });

      // イベントハンドラ設定
      stream.on('data', (data) => {
        socket.emit('mcp.data', data);
      });

      stream.on('error', (error) => {
        socket.emit('mcp.error', error);
      });

      stream.on('end', () => {
        socket.emit('mcp.streamEnd');
      });

      // クライアント切断時のクリーンアップ
      socket.on('disconnect', () => {
        stream.close();
      });
    } catch (err) {
      socket.emit('mcp.error', {
        message: err.message
      });
    }
  });
});

// ヘルパー関数
function parseResource(resourceUri) {
  try {
    const url = new URL(resourceUri);
    return {
      service: url.protocol.replace(':', ''),
      path: url.pathname,
      query: Object.fromEntries(url.searchParams),
      full: resourceUri
    };
  } catch (err) {
    throw new Error(`Invalid resource URI: ${resourceUri}`);
  }
}
```

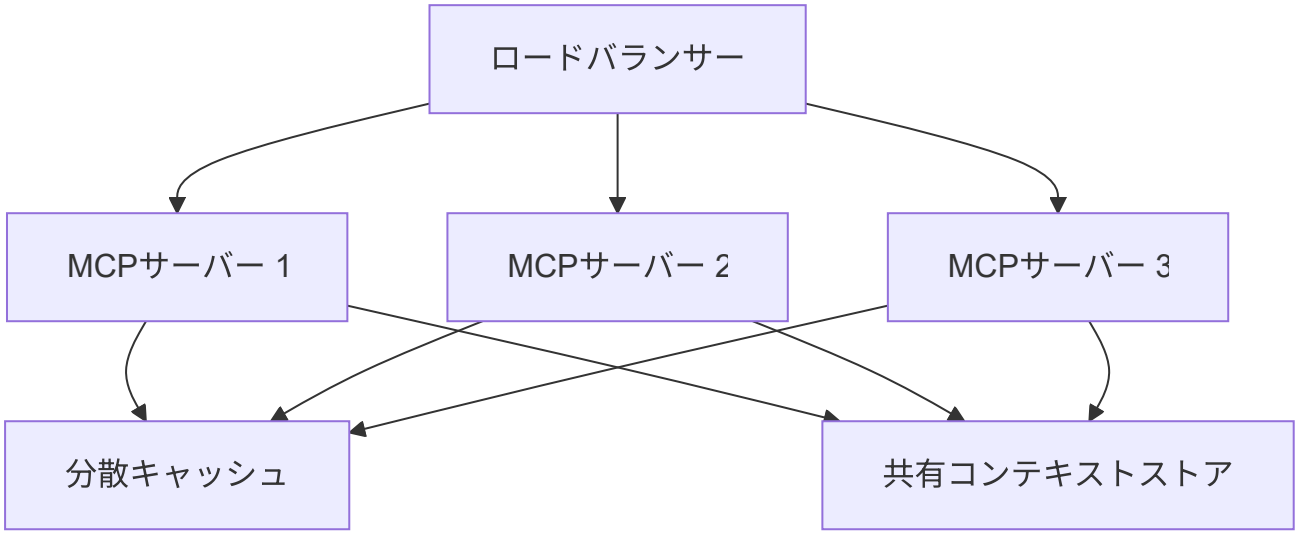
```
async function checkAccess(user, resource) {
  // アクセス制御ロジック
  // ...
  return true;
}

async function authenticateUser(username, password) {
  // ユーザー認証ロジック
  // ...
}

// サーバー起動
const PORT = process.env.PORT || 3000;
wsServer.listen(PORT, () => {
  console.log(`MCP Server running on port ${PORT}`);
});
```

> **プロジェクト事例**: *中規模の製造業企業では、製品設計情報、部品データベース、製造指示書など、異なるシステムに分散していた情報を統合するためにMCPサーバーを導入しました。設計者は3D CADモデルを操作しながら、過去の類似部品の設計理由や製造時の注意点などを直接参照できるようになり、設計ミスが30%減少し、設計から製造までのリードタイムが平均で2週間短縮されました。*

7.3 スケーラビリティと高可用性の確保

MCPサーバーを大規模環境でも安定して運用するための設計と実装方法を解説します。

7.3.1 水平スケーリングの実装



水平スケーリングの実装ポイント：

1. **ステートレス設計**
   - サーバー自体はステートレスに保ち、複数インスタンスで処理を分散
   - セッション情報は外部ストアで管理
2. **分散キャッシュの活用**

```
# Redisを使った分散キャッシュ実装例
import redis
```

```python
import json

class DistributedCache:
    def __init__(self, redis_url, ttl=3600):
        self.redis = redis.from_url(redis_url)
        self.default_ttl = ttl

    def get(self, key):
        value = self.redis.get(f"mcp:cache:{key}")
        if value:
            return json.loads(value)
        return None

    def set(self, key, value, ttl=None):
        if ttl is None:
            ttl = self.default_ttl

        self.redis.setex(
            f"mcp:cache:{key}",
            ttl,
            json.dumps(value)
        )

    def invalidate(self, key):
        self.redis.delete(f"mcp:cache:{key}")

    def invalidate_pattern(self, pattern):
        keys = self.redis.keys(f"mcp:cache:{pattern}")
        if keys:
            self.redis.delete(*keys)
```

3. **共有コンテキストストア**

```python
# MongoDBを使ったコンテキストストア実装例
import pymongo
from datetime import datetime

class ContextStore:
    def __init__(self, mongo_uri, db_name="mcp", collection="contexts"):
        self.client = pymongo.MongoClient(mongo_uri)
        self.db = self.client[db_name]
        self.collection = self.db[collection]

        # インデックス作成
        self.collection.create_index("sessionId", unique=True)
        self.collection.create_index("lastAccessed")

    def get_context(self, session_id):
        context = self.collection.find_one({"sessionId": session_id})
        if context:
            # 最終アクセス時間を更新
            self.collection.update_one(
                {"_id": context["_id"]},
                {"$set": {"lastAccessed": datetime.utcnow()}}
            )
            return context
```

```
        return None

    def create_context(self, initial_data=None):
        context = {
            "sessionId": generate_session_id(),
            "created": datetime.utcnow(),
            "lastAccessed": datetime.utcnow(),
            "data": initial_data or {}
        }

        result = self.collection.insert_one(context)
        context["_id"] = result.inserted_id
        return context

    def update_context(self, session_id, updates):
        result = self.collection.update_one(
            {"sessionId": session_id},
            {
                "$set": {
                    "data": updates,
                    "lastAccessed": datetime.utcnow()
                }
            }
        )

        if result.modified_count == 0:
            return None

        return self.get_context(session_id)

    def cleanup_old_contexts(self, max_age_hours=24):
        cutoff = datetime.utcnow() - timedelta(hours=max_age_hours)
        self.collection.delete_many({"lastAccessed": {"$lt": cutoff}})
```

7.3.2 負荷分散と利用制限

高トラフィック環境での安定性確保：

1. **アダプティブレート制限**

```
# 動的レート制限の実装例
import time
import redis

class AdaptiveRateLimiter:
    def __init__(self, redis_client, window_size=60, initial_limit=100):
        self.redis = redis_client
        self.window_size = window_size  # 秒単位のウィンドウサイズ
        self.initial_limit = initial_limit

    def check_limit(self, user_id, service=None):
        now = int(time.time())
        window_key = now // self.window_size
        rate_key = f"rate:{window_key}:{user_id}"
```

```
        if service:
            rate_key += f":{service}"

        # 現在の使用量を取得
        current_usage = int(self.redis.get(rate_key) or 0)

        # ユーザー別の制限を取得
        user_limit_key = f"limit:{user_id}"
        user_limit = int(self.redis.get(user_limit_key) or self.initial_limit)

        # サービス別の制限を確認
        if service:
            service_limit_key = f"service_limit:{service}"
            service_limit = int(self.redis.get(service_limit_key) or 0)
            if service_limit > 0:
                user_limit = min(user_limit, service_limit)

        # 制限を超えていないか確認
        if current_usage >= user_limit:
            return False

        # 使用量をインクリメント
        pipe = self.redis.pipeline()
        pipe.incr(rate_key, 1)
        pipe.expire(rate_key, self.window_size)
        pipe.execute()

        return True

    def adjust_limit(self, user_id, new_limit, duration=None):
        """ユーザーの制限を動的に調整"""
        user_limit_key = f"limit:{user_id}"

        self.redis.set(user_limit_key, new_limit)
        if duration:
            self.redis.expire(user_limit_key, duration)

    def get_usage_stats(self, user_id):
        """現在の使用状況を取得"""
        now = int(time.time())
        window_key = now // self.window_size
        rate_key = f"rate:{window_key}:{user_id}"

        current_usage = int(self.redis.get(rate_key) or 0)
        user_limit_key = f"limit:{user_id}"
        user_limit = int(self.redis.get(user_limit_key) or self.initial_limit)

        return {
            "usage": current_usage,
            "limit": user_limit,
            "remaining": max(0, user_limit - current_usage)
        }
```

2. **インテリジェントキャッシング**
   - 頻繁に要求されるリソースをキャッシュ

- コンテキスト依存度に基づいたキャッシュ戦略
- リソースタイプごとのキャッシュTTL最適化

3. **バックプレッシャー制御**

```
// Node.jsでのバックプレッシャー制御例
class RequestQueue {
  constructor(options = {}) {
    this.maxQueueSize = options.maxQueueSize || 1000;
    this.processingLimit = options.processingLimit || 50;
    this.queue = [];
    this.processing = 0;
  }

  enqueue(request) {
    return new Promise((resolve, reject) => {
      // キューが満杯の場合は拒否
      if (this.queue.length >= this.maxQueueSize) {
        return reject(new Error('Server too busy'));
      }

      // リクエストをキューに追加
      this.queue.push({
        request,
        resolve,
        reject
      });

      // キュー処理の開始
      this.processQueue();
    });
  }

  async processQueue() {
    // 処理中のリクエストが上限に達している場合は待機
    if (this.processing >= this.processingLimit) {
      return;
    }

    // キューからリクエストを取得
    const item = this.queue.shift();
    if (!item) {
      return;
    }

    try {
      this.processing++;

      // リクエスト処理
      const result = await processRequest(item.request);

      // 結果を返す
      item.resolve(result);
    } catch (error) {
      item.reject(error);
    } finally {
```

```
      this.processing--;

      // キューが空でなければ続けて処理
      if (this.queue.length > 0) {
        this.processQueue();
      }
    }
  }
}
```

7.3.3 フォールトトレランスとリカバリ

信頼性の高いMCPサーバーのための障害対策：

1. **サーキットブレーカー**

```
// Java実装のサーキットブレーカー
public class CircuitBreaker {
    private enum State {
        CLOSED, OPEN, HALF_OPEN
    }

    private State state = State.CLOSED;
    private int failureCount = 0;
    private long lastFailureTime = 0;

    private final int failureThreshold;
    private final long resetTimeout;

    public CircuitBreaker(int failureThreshold, long resetTimeoutMs) {
        this.failureThreshold = failureThreshold;
        this.resetTimeout = resetTimeoutMs;
    }

    public synchronized <T> T execute(Supplier<T> supplier) throws Exception {
        if (isOpen()) {
            throw new RuntimeException("Circuit breaker is open");
        }

        try {
            T result = supplier.get();
            reset();
            return result;
        } catch (Exception e) {
            recordFailure();
            throw e;
        }
    }

    private boolean isOpen() {
        if (state == State.OPEN) {
            // 半開状態への移行をチェック
            long elapsed = System.currentTimeMillis() - lastFailureTime;
            if (elapsed >= resetTimeout) {
                state = State.HALF_OPEN;
```

```
                return false;
            }
            return true;
        }
        return false;
    }

    private void recordFailure() {
        failureCount++;
        lastFailureTime = System.currentTimeMillis();

        if (state == State.HALF_OPEN || failureCount >= failureThreshold) {
            state = State.OPEN;
        }
    }

    private void reset() {
        failureCount = 0;
        state = State.CLOSED;
    }
}
```

2. **冗長性とフェイルオーバー**
   - アクティブ-アクティブまたはアクティブ-パッシブクラスター
   - サービスヘルスチェックと自動フェイルオーバー
   - スタンバイレプリカの維持
3. **グレースフルデグラデーション**

```
# Pythonでの段階的縮退実装例
class ServiceRegistry:
    def __init__(self):
        self.services = {}
        self.health_status = {}
        self.fallback_handlers = {}

    def register_service(self, service_id, handler, priority=0):
        if service_id not in self.services:
            self.services[service_id] = []

        self.services[service_id].append({
            "handler": handler,
            "priority": priority
        })

        # 優先度でソート
        self.services[service_id].sort(key=lambda x: x["priority"])

        # 初期ヘルスステータスを設定
        self.health_status[service_id] = True

    def register_fallback(self, service_id, fallback_handler):
        self.fallback_handlers[service_id] = fallback_handler

    def mark_unhealthy(self, service_id):
        self.health_status[service_id] = False
```

```
    def mark_healthy(self, service_id):
        self.health_status[service_id] = True

    async def execute(self, service_id, request):
        if service_id not in self.services:
            raise ValueError(f"Unknown service: {service_id}")

        # 利用可能なハンドラーを探す
        handlers = self.services[service_id]

        for handler_info in handlers:
            handler = handler_info["handler"]

            try:
                # ハンドラーを実行
                result = await handler(request)

                # 正常に実行できたらヘルス状態を更新
                self.mark_healthy(service_id)

                return result
            except Exception as e:
                # エラーログ
                logger.error(f"Service {service_id} handler failed: {e}")

                # ヘルス状態を更新
                self.mark_unhealthy(service_id)

                # 次のハンドラーを試す（縮退）
                continue

        # すべてのハンドラーが失敗した場合はフォールバックを試す
        if service_id in self.fallback_handlers:
            try:
                return await self.fallback_handlers[service_id](request)
            except Exception as e:
                logger.error(f"Fallback for {service_id} failed: {e}")

        # 最終的な失敗
        raise RuntimeError(f"All handlers for {service_id} failed")
```

> **ベテランの知恵袋**: *「MCPサーバーの運用で最も重要なのは、予期せぬ障害が発生してもユーザー体験を維持することです。私たちが学んだのは、『部分的な成功』の概念を取り入れることの重要性です。複数のデータソースからの情報取得が一部失敗しても、利用可能な情報だけでも返し、透明性をもってユーザーに伝えることで、完全な失敗よりもはるかに良い体験を提供できます。」* - インフラストラクチャアーキテクト

7.4 パフォーマンス最適化

MCPサーバーのパフォーマンスを向上させるための設計と実装テクニックを解説します。

7.4.1 クエリ最適化

複雑なMCPクエリの処理を最適化する方法：

1. **クエリ分析と実行計画**

```
# クエリ分析エンジンの例
class QueryAnalyzer:
    def __init__(self, statistics_db):
        self.stats_db = statistics_db
        self.analyzers = {
            "github": GithubQueryAnalyzer(),
            "redmine": RedmineQueryAnalyzer(),
            "obsidian": ObsidianQueryAnalyzer()
        }

    def analyze(self, query):
        # クエリ構造の解析
        resources = self.extract_resources(query)
        params = query.get("params", {})

        # リソースごとにコスト見積もり
        costs = {}
        data_sizes = {}

        for resource in resources:
            service = resource.split("://")[0]
            if service in self.analyzers:
                analyzer = self.analyzers[service]
                costs[resource] = analyzer.estimate_cost(resource, params)
                data_sizes[resource] = analyzer.estimate_data_size(resource,
params)

        # 実行計画の生成
        plan = self.generate_execution_plan(resources, costs, data_sizes)

        return {
            "original_query": query,
            "resources": resources,
            "costs": costs,
            "data_sizes": data_sizes,
            "execution_plan": plan
        }

    def generate_execution_plan(self, resources, costs, data_sizes):
        # 並列実行可能なリソースのグループ化
        parallel_groups = []
        current_group = []

        # コストでソート
        sorted_resources = sorted(resources, key=lambda r: costs.get(r,
float('inf')))

        for resource in sorted_resources:
            # 依存関係チェック
            if self.has_dependencies(resource, current_group):
                # 新しいグループを開始
                if current_group:
                    parallel_groups.append(current_group)
                current_group = [resource]
```

```
            else:
                # 現在のグループに追加
                current_group.append(resource)

        if current_group:
            parallel_groups.append(current_group)

        # 大きなデータを返すリソースは最後に処理
        final_plan = []
        large_resources = []

        for group in parallel_groups:
            small_resources = []

            for resource in group:
                if data_sizes.get(resource, 0) > 1000000:  # 1MB以上は大きいと判断
                    large_resources.append(resource)
                else:
                    small_resources.append(resource)

            if small_resources:
                final_plan.append(small_resources)

        if large_resources:
            final_plan.append(large_resources)

        return {
            "parallel_groups": final_plan,
            "estimated_total_cost": sum(costs.values())
        }
```

2. **パラレルクエリ実行**

```
// Node.jsでの並列クエリ実行
async function executeParallelQueries(queryPlan, params, context) {
  const results = {};

  // 各並列グループを順次実行
  for (const group of queryPlan.parallel_groups) {
    // グループ内のクエリを並列実行
    const groupPromises = group.map(async (resource) => {
      const service = resource.split('://')[0];
      const connector = connectorManager.getConnector(service);

      if (!connector) {
        throw new Error(`Connector not found for service: ${service}`);
      }

      const resourceResult = await connector.query({
        resource,
        params,
        context
      });

      return {
        resource,
```

```
        result: resourceResult
      };
    });

    // すべての結果を待機
    const groupResults = await Promise.all(groupPromises);

    // 結果をマージ
    for (const { resource, result } of groupResults) {
      results[resource] = result;
    }

    // コンテキストを更新
    context = updateContextWithResults(context, results);
  }

  return {
    results,
    finalContext: context
  };
}
```

3. **インクリメンタルレスポンス**

```
// WebSocketを使ったインクリメンタルレスポンス
function handleStreamingQuery(socket, query) {
  const queryAnalyzer = new QueryAnalyzer();
  const plan = queryAnalyzer.analyze(query);

  // 実行計画を通知
  socket.emit('mcp.plan', {
    executionPlan: plan.execution_plan
  });

  let context = query.context || {};

  // 各グループを順次実行
  plan.execution_plan.parallel_groups.forEach(async (group, groupIndex) => {
    // 各リソースの処理開始を通知
    socket.emit('mcp.progress', {
      phase: `Processing group ${groupIndex +
1}/${plan.execution_plan.parallel_groups.length}`,
      resources: group
    });

    // グループ内の各リソースを並列処理
    const promises = group.map(async (resource) => {
      try {
        const service = resource.split('://')[0];
        const connector = connectorManager.getConnector(service);

        const result = await connector.query({
          resource,
          params: query.params,
          context
        });
```

```
        // 個別結果を即時送信
        socket.emit('mcp.result', {
          resource,
          status: 'success',
          data: result
        });

        return { resource, result };
      } catch (error) {
        // エラーを送信
        socket.emit('mcp.result', {
          resource,
          status: 'error',
          error: {
            message: error.message,
            code: error.code
          }
        });

        return { resource, error };
      }
    });

    // グループの全結果を待機
    const results = await Promise.allSettled(promises);

    // 成功した結果でコンテキストを更新
    results.forEach(result => {
      if (result.status === 'fulfilled' && !result.value.error) {
        context = updateContext(context, result.value.resource,
result.value.result);
      }
    });

    // グループ完了を通知
    socket.emit('mcp.groupComplete', {
      groupIndex,
      nextGroup: groupIndex + 1 < plan.execution_plan.parallel_groups.length
    });
  });

  // すべての処理が完了したら最終通知
  socket.emit('mcp.complete', {
    finalContext: context
  });
}
```

7.4.2 メモリとリソース管理

大規模データ処理のための最適化テクニック：

1. **ストリーム処理**

```
// Node.jsでのストリーム処理例
const { Transform } = require('stream');

// 大きなJSONレスポンスを処理するトランスフォームストリーム
class JSONChunkTransformer extends Transform {
  constructor(options = {}) {
    super({ objectMode: true,  ...options });
    this.buffer = '';
    this.depth = 0;
    this.inString = false;
    this.escape = false;
  }

  _transform(chunk, encoding, callback) {
    // バッファにチャンクを追加
    const data = this.buffer + chunk.toString();
    let start = 0;

    // JSONの構造を解析
    for (let i = 0; i < data.length; i++) {
      const char = data[i];

      // 文字列内のエスケープ処理
      if (this.inString) {
        if (this.escape) {
          this.escape = false;
        } else if (char === '\\') {
          this.escape = true;
        } else if (char === '"') {
          this.inString = false;
        }
        continue;
      }

      // 文字列の開始
      if (char === '"') {
        this.inString = true;
        continue;
      }

      // オブジェクトや配列の深さを追跡
      if (char === '{' || char === '[') {
        this.depth++;
      } else if (char === '}' || char === ']') {
        this.depth--;

        // トップレベルのオブジェクトや配列が完了
        if (this.depth === 0) {
          // 完全なJSONオブジェクトを出力
          const json = data.substring(start, i + 1);
          try {
            const parsed = JSON.parse(json);
            this.push(parsed);
          } catch (err) {
            this.emit('error', new Error(`Invalid JSON: ${json}`));
          }
```

```
            // 次のオブジェクトのスタート位置を更新
            start = i + 1;
          }
        }
      }

      // 残りのデータをバッファに保存
      this.buffer = data.substring(start);
      callback();
    }

    _flush(callback) {
      // 残りのデータをフラッシュ
      if (this.buffer.trim().length > 0) {
        try {
          const parsed = JSON.parse(this.buffer);
          this.push(parsed);
        } catch (err) {
          this.emit('error', new Error(`Invalid JSON in flush: ${this.buffer}`));
        }
      }

      this.buffer = '';
      callback();
    }
}

// 使用例
function processLargeResponse(response) {
  return new Promise((resolve, reject) => {
    const results = [];

    response
      .pipe(new JSONChunkTransformer())
      .on('data', (chunk) => {
        // 各JSONオブジェクトを処理
        results.push(processChunk(chunk));
      })
      .on('error', (err) => {
        reject(err);
      })
      .on('end', () => {
        resolve(results);
      });
  });
}
```

2. **バッファ管理**

```
# Pythonでのバッファ管理実装
class AdaptiveBuffer:
    def __init__(self, max_size=10000, chunk_size=1000):
        self.buffer = []
        self.max_size = max_size
        self.chunk_size = chunk_size
```

```
        self.overflow_handler = None

    def add(self, item):
        self.buffer.append(item)

        # バッファサイズをチェック
        if len(self.buffer) >= self.max_size:
            self.flush()

    def register_overflow_handler(self, handler):
        """オーバーフロー時に呼び出されるハンドラーを登録"""
        self.overflow_handler = handler

    def flush(self):
        """バッファをフラッシュして処理"""
        if not self.buffer:
            return

        if self.overflow_handler:
            # チャンクに分割して処理
            for i in range(0, len(self.buffer), self.chunk_size):
                chunk = self.buffer[i:i+self.chunk_size]
                self.overflow_handler(chunk)

        # バッファをクリア
        self.buffer = []

    def get_all(self):
        """バッファ内のすべてのアイテムを取得"""
        return self.buffer.copy()

    def clear(self):
        """バッファをクリア"""
        self.buffer = []
```

3. **リソース制限モニタリング**

```
# リソース使用量モニタリングとスロットリング
import psutil
import time
from threading import Thread

class ResourceMonitor:
    def __init__(self, cpu_threshold=70, memory_threshold=80, check_interval=5):
        self.cpu_threshold = cpu_threshold
        self.memory_threshold = memory_threshold
        self.check_interval = check_interval
        self.throttling_enabled = False
        self.listeners = []
        self._running = False
        self._thread = None

    def start(self):
        """モニタリングを開始"""
        if self._running:
            return
```

```
        self._running = True
        self._thread = Thread(target=self._monitor_loop)
        self._thread.daemon = True
        self._thread.start()

    def stop(self):
        """モニタリングを停止"""
        self._running = False
        if self._thread:
            self._thread.join(timeout=1.0)
            self._thread = None

    def add_listener(self, listener):
        """リソース状態変更リスナーを追加"""
        self.listeners.append(listener)

    def _monitor_loop(self):
        """リソース監視ループ"""
        while self._running:
            cpu_percent = psutil.cpu_percent(interval=1)
            memory_percent = psutil.virtual_memory().percent

            # CPU使用率チェック
            cpu_throttle = cpu_percent > self.cpu_threshold

            # メモリ使用率チェック
            memory_throttle = memory_percent > self.memory_threshold

            # スロットリング状態の更新
            new_throttling_state = cpu_throttle or memory_throttle

            # 状態が変わった場合に通知
            if new_throttling_state != self.throttling_enabled:
                self.throttling_enabled = new_throttling_state
                self._notify_state_change()

            # 一定間隔で監視
            time.sleep(self.check_interval)

    def _notify_state_change(self):
        """リスナーに状態変更を通知"""
        for listener in self.listeners:
            try:
                listener(self.throttling_enabled)
            except Exception as e:
                print(f"Error notifying listener: {e}")

    def is_throttling(self):
        """現在のスロットリング状態を取得"""
        return self.throttling_enabled
```

> **失敗から学ぶ**: *「大規模なMCPサーバーを運用し始めた当初、特定の複雑なクエリでメモリ使用量が急増し、サーバーがクラッシュする問題に悩まされました。分析の結果、複数のデータソースからの大きな結果セットを一度にメモリに保持していたことが原因でした。ストリームベースの処理とイン*

> *クリメンタルレスポンスに切り替えることで、メモリ使用量を80%削減でき、より大きなデータセットも安定して処理できるようになりました。」 - パフォーマンスエンジニア*

MCPサーバー実装チェックリスト

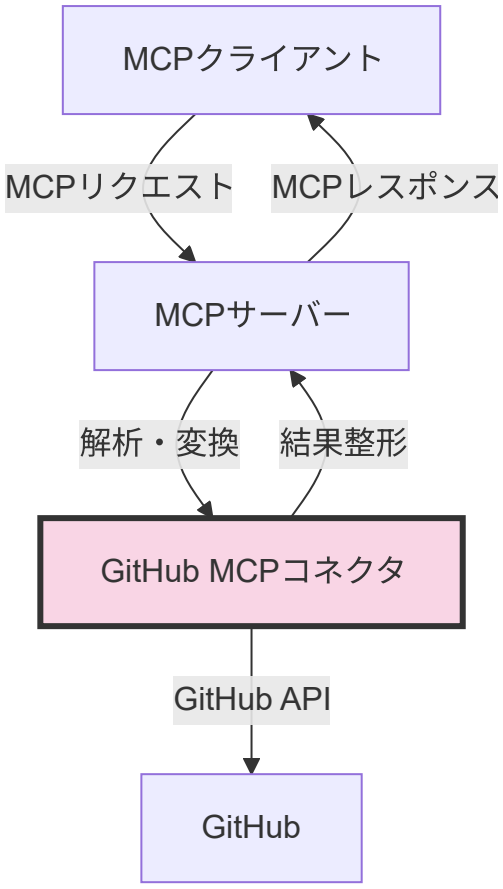
- [ ] 基本アーキテクチャの決定（単一サーバー、マイクロサービス）
- [ ] 必要なコネクタの特定と設計
- [ ] 認証・認可メカニズムの設計
- [ ] コンテキスト管理ストラテジーの確立
- [ ] スケーリング要件の分析と実装計画
- [ ] キャッシュ戦略の設計
- [ ] エラー処理とリカバリメカニズムの実装
- [ ] パフォーマンスボトルネックの特定と最適化
- [ ] 監視とロギングインフラの構築
- [ ] セキュリティレビューの実施

第8章: GitHubとの連携 - コード管理とAI連携

8.1 GitHub MCPコネクタの基本

GitHubのリポジトリやプロジェクト情報にMCPを通じてアクセスする方法を解説します。

8.1.1 GitHubコネクタのアーキテクチャ



GitHub MCPコネクタは以下の機能を提供します：

1. **リポジトリ情報アクセス**
   - コードファイルの取得
   - コミット履歴の検索と分析
   - ブランチ・タグ情報の取得
2. **Issue・PR管理**
   - Issue一覧取得と詳細アクセス
   - PRの状態確認とレビュー情報
   - コメントとディスカッションの追跡
3. **セマンティック検索**
   - コードベース内のコンセプト検索
   - 関連コード例の発見
   - 実装パターンのクエリ

8.1.2 GitHubコネクタの実装例

```python
# Python版GitHub MCPコネクタ
import requests
import base64
import json
from typing import Dict, Any, List, Optional

class GitHubConnector:
    """GitHub MCPコネクタの実装"""

    def __init__(self, config: Dict[str, Any]):
        self.api_base_url = config.get('api_base_url', 'https://api.github.com')
        self.default_headers = {
            'Accept': 'application/vnd.github.v3+json'
        }

        # トークン設定
        self.token = config.get('token')
        if self.token:
            self.default_headers['Authorization'] = f'token {self.token}'

    async def handle_request(self, request: Dict[str, Any]) -> Dict[str, Any]:
        """MCPリクエストを処理"""
        request_type = request.get('requestType', '')
        resource = request.get('resource', '')
        params = request.get('params', {})

        # リソースURIの解析
        resource_info = self._parse_resource_uri(resource)

        # リクエストタイプに基づいて処理を分岐
        if request_type == 'query':
            return await self._handle_query(resource_info, params)
        elif request_type == 'update':
            return await self._handle_update(resource_info, params)
        elif request_type == 'execute':
            return await self._handle_execute(resource_info, params)
        else:
            raise ValueError(f"Unsupported request type: {request_type}")

    def _parse_resource_uri(self, uri: str) -> Dict[str, Any]:
        """GitHub URIのパース
        例: github://owner/repo/file.md
            github://owner/repo/issues/123
        """
        if not uri.startswith('github://'):
            raise ValueError(f"Invalid GitHub resource URI: {uri}")

        # プロトコルを削除
        path = uri[9:]
        parts = path.split('/')

        if len(parts) < 2:
            raise ValueError(f"Invalid GitHub resource URI format: {uri}")

        result = {
            'owner': parts[0],
```

```
            'repo': parts[1],
            'type': 'repository',
            'path': None,
            'number': None
        }

        if len(parts) > 2:
            if parts[2] == 'issues':
                result['type'] = 'issue'
                if len(parts) > 3:
                    result['number'] = parts[3]
            elif parts[2] == 'pulls':
                result['type'] = 'pull'
                if len(parts) > 3:
                    result['number'] = parts[3]
            else:
                result['type'] = 'file'
                result['path'] = '/'.join(parts[2:])

        return result

    async def _handle_query(self, resource: Dict[str, Any], params: Dict[str, Any]) ->
Dict[str, Any]:
        """クエリ処理"""
        resource_type = resource['type']

        if resource_type == 'repository':
            return await self._query_repository(resource, params)
        elif resource_type == 'file':
            return await self._query_file(resource, params)
        elif resource_type == 'issue':
            return await self._query_issue(resource, params)
        elif resource_type == 'pull':
            return await self._query_pull(resource, params)
        else:
            raise ValueError(f"Unsupported resource type: {resource_type}")

    async def _query_repository(self, resource: Dict[str, Any], params: Dict[str, Any])
-> Dict[str, Any]:
        """リポジトリ情報を取得"""
        owner = resource['owner']
        repo = resource['repo']

        # リポジトリ基本情報の取得
        repo_info = await self._api_get(f'/repos/{owner}/{repo}')

        # 追加情報の取得（パラメータに応じて）
        if params.get('include_branches', False):
            branches = await self._api_get(f'/repos/{owner}/{repo}/branches')
            repo_info['branches'] = branches

        if params.get('include_tags', False):
            tags = await self._api_get(f'/repos/{owner}/{repo}/tags')
            repo_info['tags'] = tags

        if params.get('include_contributors', False):
            contributors = await self._api_get(f'/repos/{owner}/{repo}/contributors')
```

```
            repo_info['contributors'] = contributors

        return {
            'data': repo_info,
            'metadata': {
                'resource_type': 'repository',
                'owner': owner,
                'repo': repo
            }
        }

    async def _query_file(self, resource: Dict[str, Any], params: Dict[str, Any]) ->
Dict[str, Any]:
        """ファイル内容を取得"""
        owner = resource['owner']
        repo = resource['repo']
        path = resource['path']
        ref = params.get('ref', 'main')  # ブランチまたはコミットSHA

        # ファイル内容の取得
        content_response = await self._api_get(
            f'/repos/{owner}/{repo}/contents/{path}',
            {'ref': ref}
        )

        # ベース64でエンコードされたコンテンツをデコード
        if 'content' in content_response:
            content = base64.b64decode(content_response['content']).decode('utf-8')
        else:
            content = None

        # コミット履歴の取得（オプション）
        history = None
        if params.get('include_history', False):
            history = await self._api_get(
                f'/repos/{owner}/{repo}/commits',
                {'path': path, 'page': 1, 'per_page': 10}
            )

        return {
            'data': {
                'file_info': content_response,
                'content': content,
                'history': history
            },
            'metadata': {
                'resource_type': 'file',
                'owner': owner,
                'repo': repo,
                'path': path,
                'ref': ref
            }
        }

    async def _query_issue(self, resource: Dict[str, Any], params: Dict[str, Any]) ->
Dict[str, Any]:
        """Issue情報を取得"""
```

```
        owner = resource['owner']
        repo = resource['repo']
        issue_number = resource['number']

        if issue_number:
            # 特定のIssue詳細を取得
            issue = await self._api_get(f'/repos/{owner}/{repo}/issues/{issue_number}')

            # コメント取得（オプション）
            if params.get('include_comments', False):
                comments = await
self._api_get(f'/repos/{owner}/{repo}/issues/{issue_number}/comments')
                issue['comments_data'] = comments

            return {
                'data': issue,
                'metadata': {
                    'resource_type': 'issue',
                    'owner': owner,
                    'repo': repo,
                    'number': issue_number
                }
            }
        else:
            # Issueリストを取得
            query_params = {}

            # フィルターパラメータの追加
            for param in ['state', 'labels', 'sort', 'direction', 'since']:
                if param in params:
                    query_params[param] = params[param]

            # ページネーション
            if 'page' in params:
                query_params['page'] = params['page']
            if 'per_page' in params:
                query_params['per_page'] = params['per_page']

            issues = await self._api_get(f'/repos/{owner}/{repo}/issues', query_params)

            return {
                'data': issues,
                'metadata': {
                    'resource_type': 'issues',
                    'owner': owner,
                    'repo': repo,
                    'params': query_params
                }
            }

    async def _query_pull(self, resource: Dict[str, Any], params: Dict[str, Any]) ->
Dict[str, Any]:
        """PR情報を取得"""
        owner = resource['owner']
        repo = resource['repo']
        pr_number = resource['number']
```

```
        if pr_number:
            # 特定のPR詳細を取得
            pr = await self._api_get(f'/repos/{owner}/{repo}/pulls/{pr_number}')

            # 追加情報の取得（パラメータに応じて）
            if params.get('include_commits', False):
                commits = await
self._api_get(f'/repos/{owner}/{repo}/pulls/{pr_number}/commits')
                pr['commits_data'] = commits

            if params.get('include_files', False):
                files = await
self._api_get(f'/repos/{owner}/{repo}/pulls/{pr_number}/files')
                pr['files_data'] = files

            if params.get('include_comments', False):
                comments = await
self._api_get(f'/repos/{owner}/{repo}/pulls/{pr_number}/comments')
                pr['comments_data'] = comments

            return {
                'data': pr,
                'metadata': {
                    'resource_type': 'pull',
                    'owner': owner,
                    'repo': repo,
                    'number': pr_number
                }
            }
        else:
            # PRリストを取得
            query_params = {}

            # フィルターパラメータの追加
            for param in ['state', 'head', 'base', 'sort', 'direction']:
                if param in params:
                    query_params[param] = params[param]

            # ページネーション
            if 'page' in params:
                query_params['page'] = params['page']
            if 'per_page' in params:
                query_params['per_page'] = params['per_page']

            pulls = await self._api_get(f'/repos/{owner}/{repo}/pulls', query_params)

            return {
                'data': pulls,
                'metadata': {
                    'resource_type': 'pulls',
                    'owner': owner,
                    'repo': repo,
                    'params': query_params
                }
            }

    async def _handle_update(self, resource: Dict[str, Any], params: Dict[str, Any]) ->
```

```
Dict[str, Any]:
        """更新処理"""
        resource_type = resource['type']

        if resource_type == 'file':
            return await self._update_file(resource, params)
        elif resource_type == 'issue':
            return await self._update_issue(resource, params)
        elif resource_type == 'pull':
            return await self._update_pull(resource, params)
        else:
            raise ValueError(f"Update not supported for resource type:
{resource_type}")

    async def _update_file(self, resource: Dict[str, Any], params: Dict[str, Any]) ->
Dict[str, Any]:
        """ファイル内容を更新"""
        owner = resource['owner']
        repo = resource['repo']
        path = resource['path']

        # 現在のファイル情報を取得
        current_file = await self._api_get(
            f'/repos/{owner}/{repo}/contents/{path}',
            {'ref': params.get('branch', 'main')}
        )

        # ファイル更新リクエスト
        update_data = {
            'message': params.get('commit_message', f'Update {path}'),
            'content': base64.b64encode(params['content'].encode('utf-8')).decode('utf-
8'),
            'sha': current_file['sha']
        }

        # ブランチ指定
        if 'branch' in params:
            update_data['branch'] = params['branch']

        # 更新実行
        result = await self._api_put(f'/repos/{owner}/{repo}/contents/{path}',
update_data)

        return {
            'data': result,
            'metadata': {
                'resource_type': 'file',
                'action': 'update',
                'owner': owner,
                'repo': repo,
                'path': path
            }
        }

    async def _update_issue(self, resource: Dict[str, Any], params: Dict[str, Any]) ->
Dict[str, Any]:
        """Issue情報を更新"""
```

```
        owner = resource['owner']
        repo = resource['repo']
        issue_number = resource['number']

        if not issue_number:
            raise ValueError("Issue number is required for updates")

        # 更新データの準備
        update_data = {}
        for field in ['title', 'body', 'state', 'assignees', 'labels', 'milestone']:
            if field in params:
                update_data[field] = params[field]

        if not update_data:
            raise ValueError("No update parameters provided")

        # 更新実行
        result = await self._api_patch(f'/repos/{owner}/{repo}/issues/{issue_number}',
update_data)

        return {
            'data': result,
            'metadata': {
                'resource_type': 'issue',
                'action': 'update',
                'owner': owner,
                'repo': repo,
                'number': issue_number
            }
        }

    async def _update_pull(self, resource: Dict[str, Any], params: Dict[str, Any]) ->
Dict[str, Any]:
        """PR情報を更新"""
        owner = resource['owner']
        repo = resource['repo']
        pr_number = resource['number']

        if not pr_number:
            raise ValueError("PR number is required for updates")

        # 更新データの準備
        update_data = {}
        for field in ['title', 'body', 'state', 'base']:
            if field in params:
                update_data[field] = params[field]

        if not update_data:
            raise ValueError("No update parameters provided")

        # 更新実行
        result = await self._api_patch(f'/repos/{owner}/{repo}/pulls/{pr_number}',
update_data)

        return {
            'data': result,
            'metadata': {
```

```
                'resource_type': 'pull',
                'action': 'update',
                'owner': owner,
                'repo': repo,
                'number': pr_number
            }
        }

    async def _handle_execute(self, resource: Dict[str, Any], params: Dict[str, Any]) -
> Dict[str, Any]:
        """コマンド実行処理"""
        command = params.get('command')

        if not command:
            raise ValueError("Command parameter is required")

        if command == 'create_issue':
            return await self._create_issue(resource, params)
        elif command == 'create_pull':
            return await self._create_pull(resource, params)
        elif command == 'merge_pull':
            return await self._merge_pull(resource, params)
        elif command == 'add_comment':
            return await self._add_comment(resource, params)
        else:
            raise ValueError(f"Unknown command: {command}")

    async def _create_issue(self, resource: Dict[str, Any], params: Dict[str, Any]) ->
Dict[str, Any]:
        """新規Issueを作成"""
        owner = resource['owner']
        repo = resource['repo']

        # 必須パラメータ確認
        if 'title' not in params:
            raise ValueError("Title is required to create an issue")

        # Issueデータ準備
        issue_data = {
            'title': params['title']
        }

        # オプションフィールド
        for field in ['body', 'assignees', 'labels', 'milestone']:
            if field in params:
                issue_data[field] = params[field]

        # Issue作成実行
        result = await self._api_post(f'/repos/{owner}/{repo}/issues', issue_data)

        return {
            'data': result,
            'metadata': {
                'resource_type': 'issue',
                'action': 'create',
                'owner': owner,
                'repo': repo
```

```
            }
        }

    async def _api_get(self, endpoint: str, params: Dict[str, Any] = None) -> Any:
        """GitHub API GET リクエスト"""
        url = f"{self.api_base_url}{endpoint}"

        response = requests.get(url, params=params, headers=self.default_headers)
        response.raise_for_status()

        return response.json()

    async def _api_post(self, endpoint: str, data: Dict[str, Any]) -> Any:
        """GitHub API POST リクエスト"""
        url = f"{self.api_base_url}{endpoint}"

        response = requests.post(url, json=data, headers=self.default_headers)
        response.raise_for_status()

        return response.json()

    async def _api_put(self, endpoint: str, data: Dict[str, Any]) -> Any:
        """GitHub API PUT リクエスト"""
        url = f"{self.api_base_url}{endpoint}"

        response = requests.put(url, json=data, headers=self.default_headers)
        response.raise_for_status()

        return response.json()

    async def _api_patch(self, endpoint: str, data: Dict[str, Any]) -> Any:
        """GitHub API PATCH リクエスト"""
        url = f"{self.api_base_url}{endpoint}"

        response = requests.patch(url, json=data, headers=self.default_headers)
        response.raise_for_status()

        return response.json()
```

> **若手の疑問解決**:
> *「GitHubのAPIレート制限に引っかかると思うんだけど、それはどう対処するの？」*
>
> GitHubのAPIレート制限はGitHub MCPコネクタでの一般的な課題です。以下の対策が有効です：
>
> 1. **キャッシング**: 頻繁にアクセスされる情報（リポジトリ構造、README等）をキャッシュする
> 2. **条件付きリクエスト**: ETagやLast-Modified等のHTTPヘッダーを活用して304レスポンスを活用
> 3. **バッチリクエスト**: 複数のクエリをGraphQL経由でまとめて処理
> 4. **スマートページネーション**: 本当に必要なデータだけをページ単位で取得
> 5. **レート制限モニタリング**: レート制限ヘッダーを監視し、残りが少なくなったら重要でないリクエストを延期
>
> 企業環境ではGitHub Enterprise APIを使用すると、より高いレート制限が設定できます。

8.2 コードリポジトリとの高度な連携

GitHubリポジトリの情報をより高度に活用するMCP機能を解説します。

8.2.1 セマンティックコード検索

コードベース内のコンセプトや機能を検索する方法：

```
// セマンティックコード検索のクライアント側実装例
async function searchCodeSemantics(mcp, query, options = {}) {
  const {
    repositories = [],
    language = null,
    maxResults = 10,
    includeExamples = true,
    includeDocs = true
  } = options;

  // リポジトリごとのコンテキスト設定
  const repositoryContext = repositories.map(repo => ({
    resource: `github://${repo}`,
    type: 'code-repository'
  }));

  // MCPリクエスト
  const response = await mcp.query({
    requestType: 'semantic-search',
    resource: 'github://code',
    params: {
      query,
      language,
      maxResults,
      includeExamples,
      includeDocs
    },
    context: {
      repositories: repositoryContext
    }
  });

  return {
    results: response.data.results,
    totalFound: response.data.totalFound,
    repositories: response.data.searchedRepositories,
    executionTimeMs: response.metadata.executionTime
  };
}

// 使用例
const searchResults = await searchCodeSemantics(mcpClient,
  "ユーザー認証とセッション管理の実装",
  {
    repositories: ["organization/auth-service", "organization/user-api"],
    language: "javascript",
    includeDocs: true
  }
);
```

```
// 検索結果の処理
searchResults.results.forEach(result => {
  console.log(`File: ${result.path}`);
  console.log(`Relevance: ${result.score}`);
  console.log(`Description: ${result.description}`);
  console.log(`Code Snippet:\n${result.snippet}`);
  console.log('---');
});
```

8.2.2 コードコンテキスト解析

コードの関連情報や背景を解析する機能：

```
// コードコンテキスト解析の実装例
async function analyzeCodeContext(mcp, filePath, options = {}) {
  const {
    repository,
    branch = 'main',
    includeHistory = true,
    includeReferences = true,
    includeDependencies = true
  } = options;

  if (!repository) {
    throw new Error("Repository parameter is required");
  }

  // MCPリクエスト
  const response = await mcp.query({
    requestType: 'analyze',
    resource: `github://${repository}/${filePath}`,
    params: {
      branch,
      includeHistory,
      includeReferences,
      includeDependencies
    }
  });

  return {
    file: response.data.file,
    content: response.data.content,
    analysis: {
      summary: response.data.summary,
      complexity: response.data.complexity,
      dateCreated: response.data.dateCreated,
      lastModified: response.data.lastModified,
      authors: response.data.authors,
      history: response.data.history,
      references: response.data.references,
      dependencies: response.data.dependencies,
      usedBy: response.data.usedBy
    }
  };
}
```

```
// 使用例
const fileContext = await analyzeCodeContext(mcpClient,
  "src/services/authentication.js",
  {
    repository: "organization/auth-service",
    includeDependencies: true
  }
);

console.log("File Summary:", fileContext.analysis.summary);
console.log("Complexity:", fileContext.analysis.complexity);
console.log("Authors:", fileContext.analysis.authors);

// 依存関係の表示
console.log("Dependencies:");
fileContext.analysis.dependencies.forEach(dep => {
  console.log(`- ${dep.path} (${dep.type})`);
});

// 参照元の表示
console.log("Referenced by:");
fileContext.analysis.usedBy.forEach(ref => {
  console.log(`- ${ref.path} (${ref.type})`);
});
```

8.2.3 コード生成と提案

GitHubリポジトリの既存コードスタイルに合わせたコード生成と提案：

```
// コード生成と提案の実装例
async function generateCodeWithContext(mcp, options = {}) {
  const {
    repository,
    description,
    language,
    filePath,
    relatedFiles = [],
    followStyle = true,
    includeTests = true,
    includeComments = true
  } = options;

  // 必須パラメータ確認
  if (!repository || !description) {
    throw new Error("Repository and description parameters are required");
  }

  // 関連ファイルのコンテキスト設定
  const fileContext = relatedFiles.map(file => ({
    resource: `github://${repository}/${file}`,
    type: 'related-file'
  }));

  // MCPリクエスト
  const response = await mcp.query({
    requestType: 'generate',
```

```
    resource: `github://${repository}`,
    params: {
      description,
      language,
      filePath,
      followStyle,
      includeTests,
      includeComments
    },
    context: {
      relatedFiles: fileContext
    }
  });

  return {
    generatedCode: response.data.code,
    description: response.data.description,
    suggestedPath: response.data.suggestedPath || filePath,
    tests: response.data.tests,
    styleSummary: response.data.styleSummary,
    alternatives: response.data.alternatives || []
  };
}

// 使用例
const generatedCode = await generateCodeWithContext(mcpClient, {
  repository: "organization/auth-service",
  description: "JWT認証トークンを検証してユーザー情報を取得するミドルウェア",
  language: "javascript",
  filePath: "src/middleware/auth-validator.js",
  relatedFiles: [
    "src/services/authentication.js",
    "src/models/user.js",
    "src/config/jwt-config.js"
  ],
  includeTests: true
});

console.log("Generated Code:");
console.log(generatedCode.generatedCode);

console.log("\nTests:");
console.log(generatedCode.tests);

console.log("\nStyle Summary:");
console.log(generatedCode.styleSummary);
```

> **プロジェクト事例**: *ある大規模エンタープライズソフトウェア開発チームでは、GitHubとMCPを連携させて「コードナビゲーター」システムを構築しました。開発者は複雑なコードベースを探索する際に、特定の機能や実装パターンを自然言語で検索できるようになりました。例えば「データベース接続プールの設定方法」と検索すると、関連するコードファイルとその実装背景、設計判断理由が提示されます。この仕組みにより、新メンバーの立ち上げ時間が60%短縮され、ベテラン開発者の知識依存度が大幅に減少しました。*

8.3 GitHub Issuesとプロジェクト管理

GitHubのIssues、PR、プロジェクト情報をMCPで活用する方法を解説します。

8.3.1 Issueトラッキングと分析

```
// Issue分析機能の実装例
async function analyzeIssues(mcp, repository, options = {}) {
  const {
    timeRange = { start: null, end: null },
    labels = [],
    assignees = [],
    state = 'all',
    includeComments = true,
    performAnalysis = true
  } = options;

  // クエリパラメータの構築
  const queryParams = {
    state,
    labels: labels.join(','),
    assignees: assignees.join(','),
    since: timeRange.start,
    includeComments,
    performAnalysis
  };

  // MCPリクエスト
  const response = await mcp.query({
    requestType: 'analyze',
    resource: `github://${repository}/issues`,
    params: queryParams
  });

  const analysis = response.data;

  return {
    issues: analysis.issues,
    statistics: {
      totalIssues: analysis.totalCount,
      openIssues: analysis.openCount,
      closedIssues: analysis.closedCount,
      averageResolutionTime: analysis.averageResolutionTime,
      issuesByLabel: analysis.labelDistribution,
      issuesByAssignee: analysis.assigneeDistribution,
      trendsOverTime: analysis.trends
    },
    insights: analysis.insights,
    recommendedActions: analysis.recommendedActions
  };
}

// 使用例
const issueAnalysis = await analyzeIssues(mcpClient, "organization/project", {
  timeRange: {
    start: "2023-01-01T00:00:00Z",
    end: "2023-03-31T23:59:59Z"
  },
```

```
  labels: ["bug", "feature", "documentation"],
  state: "all",
  performAnalysis: true
});

console.log("Issue Statistics:");
console.log(`Total Issues: ${issueAnalysis.statistics.totalIssues}`);
console.log(`Open Issues: ${issueAnalysis.statistics.openIssues}`);
console.log(`Closed Issues: ${issueAnalysis.statistics.closedIssues}`);
console.log(`Average Resolution Time: ${issueAnalysis.statistics.averageResolutionTime}
days`);

console.log("\nInsights:");
issueAnalysis.insights.forEach((insight, index) => {
  console.log(`${index + 1}. ${insight}`);
});

console.log("\nRecommended Actions:");
issueAnalysis.recommendedActions.forEach((action, index) => {
  console.log(`${index + 1}. ${action}`);
});
```

8.3.2 プルリクエスト管理

PRの処理やレビュー管理をMCPで行う例：

```
// PR検索と分析の実装例
async function analyzePullRequests(mcp, repository, options = {}) {
  const {
    state = 'all',
    sort = 'created',
    direction = 'desc',
    reviewStatus = null,
    authorFilter = null,
    reviewerFilter = null,
    timeRange = { start: null, end: null },
    includeReviewComments = true,
    includeCommits = true,
    includeFiles = true,
    analyzeTrends = true
  } = options;

  // クエリパラメータの構築
  const queryParams = {
    state,
    sort,
    direction,
    reviewStatus,
    author: authorFilter,
    reviewer: reviewerFilter,
    since: timeRange.start,
    until: timeRange.end,
    includeReviewComments,
    includeCommits,
    includeFiles,
    analyzeTrends
```

```
  };

  // MCPリクエスト
  const response = await mcp.query({
    requestType: 'analyze',
    resource: `github://${repository}/pulls`,
    params: queryParams
  });

  return {
    pullRequests: response.data.pullRequests,
    statistics: {
      totalPRs: response.data.totalCount,
      openPRs: response.data.openCount,
      mergedPRs: response.data.mergedCount,
      closedPRs: response.data.closedCount,
      averageMergeTime: response.data.averageMergeTime,
      averageReviewTime: response.data.averageReviewTime,
      fileChangeDistribution: response.data.fileChanges,
      authorDistribution: response.data.authorDistribution,
      reviewerDistribution: response.data.reviewerDistribution
    },
    codeQualityInsights: response.data.codeQualityInsights,
    reviewProcess: {
      cycleTime: response.data.cycleTime,
      bottlenecks: response.data.bottlenecks,
      reviewThoroughness: response.data.reviewThoroughness
    },
    recommendations: response.data.recommendations
  };
}

// PRレビュー支援の実装例
async function assistPullRequestReview(mcp, pullRequestUrl, options = {}) {
  const {
    focusAreas = ["logic", "security", "performance", "style"],
    codeQualityCheck = true,
    suggestImprovements = true,
    checkAgainstGuidelines = true,
    guidelines = []
  } = options;

  // URLからリポジトリとPR番号をパース
  const matches = pullRequestUrl.match(/github\.com\/([^\/]+\/[^\/]+)\/pull\/(\d+)/);
  if (!matches) {
    throw new Error("Invalid GitHub pull request URL");
  }

  const repository = matches[1];
  const prNumber = matches[2];

  // MCPリクエスト
  const response = await mcp.query({
    requestType: 'assist',
    resource: `github://${repository}/pulls/${prNumber}`,
    params: {
      focusAreas,
```

```
      codeQualityCheck,
      suggestImprovements,
      checkAgainstGuidelines,
      guidelines
    }
  });

  return {
    summary: response.data.summary,
    changesOverview: response.data.changesOverview,
    review: {
      highPriorityFeedback: response.data.highPriorityFeedback,
      securityConcerns: response.data.securityConcerns,
      performanceImpact: response.data.performanceImpact,
      codeQuality: response.data.codeQuality,
      styleConsistency: response.data.styleConsistency,
      testCoverage: response.data.testCoverage
    },
    suggestedImprovements: response.data.suggestedImprovements,
    fileSpecificFeedback: response.data.fileSpecificFeedback,
    guidelineCompliance: response.data.guidelineCompliance
  };
}
```

> **ベテランの知恵袋**: *「PRレビュー支援はMCPの最も価値ある使い方の一つです。私のチームでは、AIによる自動レビューと人間のレビューを組み合わせることで、初期レビュー時間を70%削減できました。しかし、AIだけにレビューを任せるのではなく、AIによるレビュー提案を人間がさらにレビューするプロセスが重要です。特に、AIがコードの意図を誤解している場合の判断は人間の役割です。」* - シニアエンジニアリングマネージャー

8.3.3 プロジェクト管理の統合

GitHub ProjectsやMilestoneなどのプロジェクト管理機能とMCPを統合する例：

```
// プロジェクト進捗状況の分析
async function analyzeProjectProgress(mcp, repository, projectNumber, options = {}) {
  const {
    includeMilestones = true,
    includeIssues = true,
    includePullRequests = true,
    timeRange = { start: null, end: null },
    calculateVelocity = true,
    forecastCompletion = true
  } = options;

  // MCPリクエスト
  const response = await mcp.query({
    requestType: 'analyze',
    resource: `github://${repository}/projects/${projectNumber}`,
    params: {
      includeMilestones,
      includeIssues,
      includePullRequests,
      since: timeRange.start,
      until: timeRange.end,
```

```
      calculateVelocity,
      forecastCompletion
    }
  });

  return {
    projectInfo: response.data.projectInfo,
    progress: {
      completedPercentage: response.data.completedPercentage,
      itemsTotal: response.data.itemsTotal,
      itemsCompleted: response.data.itemsCompleted,
      itemsInProgress: response.data.itemsInProgress,
      itemsNotStarted: response.data.itemsNotStarted
    },
    milestones: response.data.milestones,
    velocity: response.data.velocity,
    forecast: response.data.forecast,
    bottlenecks: response.data.bottlenecks,
    recommendations: response.data.recommendations
  };
}

// プロジェクトレポート生成
async function generateProjectReport(mcp, repository, options = {}) {
  const {
    projectNumber = null,
    milestoneTitle = null,
    timeRange = { start: null, end: null },
    includeContributorStats = true,
    includeVelocityTrends = true,
    includeBurndownChart = true,
    includeRiskAssessment = true,
    format = "markdown"
  } = options;

  if (!projectNumber && !milestoneTitle) {
    throw new Error("Either projectNumber or milestoneTitle must be specified");
  }

  // リソースURI構築
  let resourceUri = `github://${repository}`;
  if (projectNumber) {
    resourceUri += `/projects/${projectNumber}`;
  } else if (milestoneTitle) {
    resourceUri += `/milestones?title=${encodeURIComponent(milestoneTitle)}`;
  }

  // MCPリクエスト
  const response = await mcp.query({
    requestType: 'report',
    resource: resourceUri,
    params: {
      since: timeRange.start,
      until: timeRange.end,
      includeContributorStats,
      includeVelocityTrends,
      includeBurndownChart,
```

```
      includeRiskAssessment,
      format
    }
  });

  return {
    title: response.data.title,
    summary: response.data.summary,
    reportContent: response.data.content,
    metrics: response.data.metrics,
    charts: response.data.charts,
    dateGenerated: response.data.dateGenerated
  };
}
```

8.4 GitHub Actions連携とワークフロー自動化

GitHub ActionsとMCPを連携させて開発ワークフローを自動化する方法を解説します。

8.4.1 GitHub Actionsの制御

```
// GitHub Actionsワークフロー実行の制御
async function triggerWorkflow(mcp, repository, workflowId, options = {}) {
  const {
    ref = 'main',
    inputs = {},
    reason = 'MCP Workflow Trigger'
  } = options;

  // MCPリクエスト
  const response = await mcp.query({
    requestType: 'execute',
    resource: `github://${repository}/actions/workflows/${workflowId}`,
    params: {
      command: 'dispatch',
      ref,
      inputs,
      reason
    }
  });

  return {
    workflowId: response.data.workflowId,
    runId: response.data.runId,
    status: response.data.status,
    createdAt: response.data.createdAt,
    htmlUrl: response.data.htmlUrl
  };
}

// ワークフロー実行状態の監視
async function monitorWorkflowRun(mcp, repository, runId, options = {}) {
  const {
    waitForCompletion = true,
    pollInterval = 5000,
```

```
    timeout = 3600000  // 1時間
  } = options;

  // 即時状態取得
  const checkStatus = async () => {
    const response = await mcp.query({
      requestType: 'query',
      resource: `github://${repository}/actions/runs/${runId}`
    });

    return {
      runId: response.data.id,
      workflowId: response.data.workflow_id,
      name: response.data.name,
      status: response.data.status,
      conclusion: response.data.conclusion,
      createdAt: response.data.created_at,
      updatedAt: response.data.updated_at,
      htmlUrl: response.data.html_url,
      jobs: response.data.jobs,
      logs: response.data.logs_url
    };
  };

  // 初期状態取得
  const initialStatus = await checkStatus();

  // 完了待ちしない場合は即時返却
  if (!waitForCompletion) {
    return initialStatus;
  }

  // 完了状態を待機
  return new Promise((resolve, reject) => {
    const startTime = Date.now();
    const statusCheck = async () => {
      try {
        const status = await checkStatus();

        // 完了したかチェック
        if (status.status === 'completed') {
          resolve(status);
          return;
        }

        // タイムアウトチェック
        if (Date.now() - startTime > timeout) {
          reject(new Error(`Workflow monitoring timed out after ${timeout}ms`));
          return;
        }

        // 次の確認をスケジュール
        setTimeout(statusCheck, pollInterval);
      } catch (error) {
        reject(error);
      }
    };
```

```
    // 監視開始
    setTimeout(statusCheck, pollInterval);
  });
}
```

8.4.2 ワークフロー定義の自動生成

```
// GitHub Actionsワークフロー定義の自動生成
async function generateWorkflowDefinition(mcp, repository, options = {}) {
  const {
    workflowName,
    description,
    triggerEvents = ["push", "pull_request"],
    language = "javascript",
    buildSteps = true,
    testSteps = true,
    deploySteps = false,
    environments = [],
    customSteps = [],
    secrets = [],
    useCache = true
  } = options;

  // MCPリクエスト
  const response = await mcp.query({
    requestType: 'generate',
    resource: `github://${repository}/actions/workflows`,
    params: {
      workflowName,
      description,
      triggerEvents,
      language,
      buildSteps,
      testSteps,
      deploySteps,
      environments,
      customSteps,
      secrets,
      useCache
    }
  });

  return {
    workflowName: response.data.name,
    filename: response.data.filename,
    yamlContent: response.data.content,
    description: response.data.description,
    suggestedPath: response.data.suggestedPath,
    estimatedExecutionTime: response.data.estimatedExecutionTime,
    requiredSecrets: response.data.requiredSecrets,
    documentation: response.data.documentation
  };
}
```

```
// ワークフローのデプロイ
async function deployWorkflow(mcp, repository, workflowDefinition, options = {}) {
  const {
    path = null,
    commitMessage = "Add GitHub Actions workflow via MCP",
    branch = "main",
    createPullRequest = false,
    pullRequestTitle = null,
    pullRequestBody = null
  } = options;

  // ファイルパスの決定
  const filePath = path || workflowDefinition.suggestedPath ||
                   `.github/workflows/${workflowDefinition.filename ||
workflowDefinition.workflowName.toLowerCase().replace(/\s+/g, '-')}.yml`;

  // デプロイメソッドの決定
  const deployMethod = createPullRequest ? 'pull-request' : 'direct';

  // MCPリクエスト
  const response = await mcp.query({
    requestType: 'update',
    resource: `github://${repository}/${filePath}`,
    params: {
      content: workflowDefinition.yamlContent,
      commitMessage,
      branch,
      deployMethod,
      pullRequestTitle: pullRequestTitle || `Add workflow:
${workflowDefinition.workflowName}`,
      pullRequestBody: pullRequestBody || `Automated addition of GitHub Actions
workflow: ${workflowDefinition.workflowName}\n\n${workflowDefinition.description}`
    }
  });

  return {
    success: response.data.success,
    fileUrl: response.data.fileUrl,
    commitSha: response.data.commitSha,
    pullRequestUrl: response.data.pullRequestUrl,
    pullRequestNumber: response.data.pullRequestNumber
  };
}
```

8.4.3 MCPトリガーのワークフロー

MCPとGitHub Actionsの連携ワークフローの例：

```
# .github/workflows/mcp-code-analysis.yml
name: MCP Code Analysis

on:
  push:
    branches: [ main, develop ]
  pull_request:
    branches: [ main, develop ]
```

```yaml
  workflow_dispatch:
    inputs:
      focused_analysis:
        description: 'Focus analysis on specific area'
        required: false
        default: 'all'
        type: choice
        options:
        - all
        - security
        - performance
        - complexity

jobs:
  analyze:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout repository
        uses: actions/checkout@v3
        with:
          fetch-depth: 0

      - name: Set up MCP Client
        uses: mcp-actions/setup-mcp@v1
        with:
          mcp-token: ${{ secrets.MCP_TOKEN }}

      - name: Get changed files
        id: changed-files
        uses: tj-actions/changed-files@v35
        with:
          all_changed_files: true

      - name: Analyze code with MCP
        id: mcp-analysis
        uses: mcp-actions/code-analysis@v1
        with:
          repository: ${{ github.repository }}
          files: ${{ steps.changed-files.outputs.all_changed_files }}
          focus: ${{ github.event.inputs.focused_analysis || 'all' }}
          output-format: 'json'

      - name: Generate report
        run: |
          echo "# MCP Code Analysis Report" > report.md
          echo "" >> report.md
          echo "Analysis performed on $(date)" >> report.md
          echo "" >> report.md

          # Process the JSON output from the previous step
          cat ${{ steps.mcp-analysis.outputs.result-file }} | jq -r '.summary' >>
report.md
          echo "" >> report.md

          echo "## Key Findings" >> report.md
          cat ${{ steps.mcp-analysis.outputs.result-file }} | jq -r '.findings[] | "- "
+ .' >> report.md
```

```
          echo "" >> report.md

          echo "## Recommendations" >> report.md
          cat ${{ steps.mcp-analysis.outputs.result-file }} | jq -r '.recommendations[]
| "- " + .' >> report.md

      - name: Upload analysis report
        uses: actions/upload-artifact@v3
        with:
          name: mcp-analysis-report
          path: report.md

      - name: Comment on PR
        if: github.event_name == 'pull_request'
        uses: thollander/actions-comment-pull-request@v2
        with:
          filePath: report.md
          comment_tag: mcp-analysis

      - name: Fail if critical issues found
        if: ${{ steps.mcp-analysis.outputs.critical-issues > 0 }}
        run: |
          echo "::error::Found ${{ steps.mcp-analysis.outputs.critical-issues }}
critical issues"
          exit 1
```

> **プロジェクト事例**: *大手金融機関では、GitHubとMCPを統合したコンプライアンス検証システムを構築しました。コード変更がプッシュされると、GitHub ActionsがMCPを通じて変更内容を分析し、社内ポリシーやセキュリティ要件への準拠を自動チェックします。この仕組みにより、規制要件へのコンプライアンスチェック工数が75%削減され、開発者がリアルタイムでフィードバックを得られるようになりました。*

GitHub MCP連携チェックリスト

- [ ] GitHub APIアクセストークンの作成と適切な権限設定
- [ ] MCPサーバーとGitHub連携コネクタのインストール
- [ ] リポジトリアクセスのスコープと権限設定
- [ ] API利用制限の対応策の実装
- [ ] 認証情報の安全な管理と定期的なローテーション
- [ ] ログ記録とモニタリングの設定
- [ ] コード分析機能のカスタマイズとチューニング
- [ ] GitHub Actionsとの連携ワークフロー設定
- [ ] フィードバックループの確立と改善プロセスの定義

第9章: 知識管理との統合 - Redmine、Obsidianとの連携実装

9.1 Redmine MCPコネクタの構築

プロジェクト管理ツールRedmineとMCPを連携させるコネクタの実装方法を解説します。

9.1.1 Redmineコネクタの基本構造

```python
# RedmineMCPコネクタ基本実装（Python）
import requests
import json
from typing import Dict, Any, List, Optional

class RedmineConnector:
    """Redmine MCPコネクタの実装"""

    def __init__(self, config: Dict[str, Any]):
        """コネクタの初期化

        Args:
            config: コネクタ設定（APIキー、URL等）
        """
        self.base_url = config.get('base_url').rstrip('/')
        self.api_key = config.get('api_key')

        self.headers = {
            'Content-Type': 'application/json',
            'X-Redmine-API-Key': self.api_key
        }

    async def handle_request(self, request: Dict[str, Any]) -> Dict[str, Any]:
        """MCPリクエストを処理

        Args:
            request: MCPリクエスト

        Returns:
            処理結果
        """
        request_type = request.get('requestType', '')
        resource = request.get('resource', '')
        params = request.get('params', {})

        resource_info = self._parse_resource_uri(resource)

        if request_type == 'query':
            return await self._handle_query(resource_info, params)
        elif request_type == 'update':
            return await self._handle_update(resource_info, params)
        elif request_type == 'create':
            return await self._handle_create(resource_info, params)
        else:
            raise ValueError(f"Unsupported request type: {request_type}")

    def _parse_resource_uri(self, uri: str) -> Dict[str, Any]:
```

```
        """Redmine URIのパース

        例: redmine://project-id/issues/123
            redmine://project-id/wiki/page-name

        Args:
            uri: リソースURI

        Returns:
            パース結果
        """
        if not uri.startswith('redmine://'):
            raise ValueError(f"Invalid Redmine resource URI: {uri}")

        # プロトコルを削除
        path = uri[10:]
        parts = path.split('/')

        if len(parts) < 1:
            raise ValueError(f"Invalid Redmine resource URI format: {uri}")

        result = {
            'project_id': parts[0],
            'resource_type': 'project',
            'resource_id': None
        }

        if len(parts) > 1:
            result['resource_type'] = parts[1]

            if len(parts) > 2:
                result['resource_id'] = parts[2]

                if len(parts) > 3:
                    result['subresource_type'] = parts[3]

                    if len(parts) > 4:
                        result['subresource_id'] = parts[4]

        return result

    async def _handle_query(self, resource: Dict[str, Any], params: Dict[str, Any]) ->
Dict[str, Any]:
        """クエリ処理

        Args:
            resource: リソース情報
            params: クエリパラメータ

        Returns:
            クエリ結果
        """
        resource_type = resource['resource_type']

        if resource_type == 'project':
            return await self._query_project(resource, params)
        elif resource_type == 'issues':
```

```
            return await self._query_issues(resource, params)
        elif resource_type == 'wiki':
            return await self._query_wiki(resource, params)
        else:
            raise ValueError(f"Unsupported resource type: {resource_type}")

    async def _query_project(self, resource: Dict[str, Any], params: Dict[str, Any]) ->
Dict[str, Any]:
        """プロジェクト情報を取得

        Args:
            resource: リソース情報
            params: クエリパラメータ

        Returns:
            プロジェクト情報
        """
        project_id = resource['project_id']
        include = params.get('include', [])

        # プロジェクト基本情報の取得
        url = f"{self.base_url}/projects/{project_id}.json"

        if include:
            url += f"?include={','.join(include)}"

        response = requests.get(url, headers=self.headers)
        response.raise_for_status()

        project_data = response.json()

        # 追加情報の取得（パラメータに応じて）
        if params.get('include_members', False):
            members_url = f"{self.base_url}/projects/{project_id}/memberships.json"
            members_response = requests.get(members_url, headers=self.headers)

            if members_response.status_code == 200:
                project_data['members'] = members_response.json()['memberships']

        if params.get('include_versions', False):
            versions_url = f"{self.base_url}/projects/{project_id}/versions.json"
            versions_response = requests.get(versions_url, headers=self.headers)

            if versions_response.status_code == 200:
                project_data['versions'] = versions_response.json()['versions']

        return {
            'data': project_data,
            'metadata': {
                'resource_type': 'project',
                'project_id': project_id
            }
        }

    async def _query_issues(self, resource: Dict[str, Any], params: Dict[str, Any]) ->
Dict[str, Any]:
        """課題情報を取得
```

```
        Args:
            resource: リソース情報
            params: クエリパラメータ

        Returns:
            課題情報
        """
        project_id = resource['project_id']
        issue_id = resource.get('resource_id')

        # 単一の課題を取得
        if issue_id:
            url = f"{self.base_url}/issues/{issue_id}.json"

            # 付属情報の指定
            include = params.get('include', [])
            if include:
                url += f"?include={','.join(include)}"

            response = requests.get(url, headers=self.headers)
            response.raise_for_status()

            return {
                'data': response.json(),
                'metadata': {
                    'resource_type': 'issue',
                    'project_id': project_id,
                    'issue_id': issue_id
                }
            }
        else:
            # 課題一覧を取得
            url = f"{self.base_url}/issues.json?project_id={project_id}"

            # フィルターパラメータの追加
            query_params = []
            for param, value in params.items():
                if param in ['status_id', 'tracker_id', 'priority_id',
'assigned_to_id',
                             'author_id', 'fixed_version_id', 'cf_id', 'created_on',
'updated_on']:
                    query_params.append(f"{param}={value}")

            if query_params:
                url += '&' + '&'.join(query_params)

            # ソートとページネーション
            if 'sort' in params:
                url += f"&sort={params['sort']}"

            if 'offset' in params:
                url += f"&offset={params['offset']}"

            if 'limit' in params:
                url += f"&limit={params['limit']}"
```

```
            response = requests.get(url, headers=self.headers)
            response.raise_for_status()

            return {
                'data': response.json(),
                'metadata': {
                    'resource_type': 'issues',
                    'project_id': project_id,
                    'params': params
                }
            }

    async def _query_wiki(self, resource: Dict[str, Any], params: Dict[str, Any]) ->
Dict[str, Any]:
        """Wikiページ情報を取得

        Args:
            resource: リソース情報
            params: クエリパラメータ

        Returns:
            Wikiページ情報
        """
        project_id = resource['project_id']
        page_id = resource.get('resource_id')

        if not page_id:
            # Wikiページ一覧を取得
            url = f"{self.base_url}/projects/{project_id}/wiki/index.json"
            response = requests.get(url, headers=self.headers)
            response.raise_for_status()

            return {
                'data': response.json(),
                'metadata': {
                    'resource_type': 'wiki',
                    'project_id': project_id
                }
            }
        else:
            # 特定のWikiページを取得
            url = f"{self.base_url}/projects/{project_id}/wiki/{page_id}.json"

            # バージョン指定
            if 'version' in params:
                url += f"?version={params['version']}"

            response = requests.get(url, headers=self.headers)
            response.raise_for_status()

            return {
                'data': response.json(),
                'metadata': {
                    'resource_type': 'wiki_page',
                    'project_id': project_id,
                    'page_id': page_id
                }
```

```
            }

    async def _handle_create(self, resource: Dict[str, Any], params: Dict[str, Any]) ->
Dict[str, Any]:
        """リソース作成処理

        Args:
            resource: リソース情報
            params: 作成パラメータ

        Returns:
            作成結果
        """
        resource_type = resource['resource_type']

        if resource_type == 'issues':
            return await self._create_issue(resource, params)
        elif resource_type == 'wiki':
            return await self._create_wiki_page(resource, params)
        else:
            raise ValueError(f"Creation not supported for resource type:
{resource_type}")

    async def _create_issue(self, resource: Dict[str, Any], params: Dict[str, Any]) ->
Dict[str, Any]:
        """課題を作成

        Args:
            resource: リソース情報
            params: 作成パラメータ

        Returns:
            作成結果
        """
        project_id = resource['project_id']

        # 必須パラメータの確認
        if 'subject' not in params:
            raise ValueError("Subject is required to create an issue")

        # 課題データの準備
        issue_data = {
            'issue': {
                'project_id': project_id,
                'subject': params['subject']
            }
        }

        # オプションフィールドの追加
        for field in ['description', 'tracker_id', 'status_id', 'priority_id',
                      'assigned_to_id', 'parent_issue_id', 'start_date', 'due_date',
                      'estimated_hours', 'done_ratio', 'custom_fields']:
            if field in params:
                issue_data['issue'][field] = params[field]

        # 課題作成リクエスト
        url = f"{self.base_url}/issues.json"
```

```
        response = requests.post(url, headers=self.headers, json=issue_data)
        response.raise_for_status()

        return {
            'data': response.json(),
            'metadata': {
                'resource_type': 'issue',
                'project_id': project_id,
                'action': 'create'
            }
        }

    async def _handle_update(self, resource: Dict[str, Any], params: Dict[str, Any]) ->
Dict[str, Any]:
        """リソース更新処理

        Args:
            resource: リソース情報
            params: 更新パラメータ

        Returns:
            更新結果
        """
        resource_type = resource['resource_type']

        if resource_type == 'issues':
            return await self._update_issue(resource, params)
        elif resource_type == 'wiki':
            return await self._update_wiki_page(resource, params)
        else:
            raise ValueError(f"Update not supported for resource type:
{resource_type}")

    async def _update_issue(self, resource: Dict[str, Any], params: Dict[str, Any]) ->
Dict[str, Any]:
        """課題を更新

        Args:
            resource: リソース情報
            params: 更新パラメータ

        Returns:
            更新結果
        """
        issue_id = resource.get('resource_id')

        if not issue_id:
            raise ValueError("Issue ID is required for updates")

        # 更新データの準備
        issue_data = {
            'issue': {}
        }

        # 更新フィールドの追加
        for field in ['subject', 'description', 'tracker_id', 'status_id',
'priority_id',
```

```
                    'assigned_to_id', 'parent_issue_id', 'start_date', 'due_date',
                    'estimated_hours', 'done_ratio', 'custom_fields', 'notes']:
            if field in params:
                issue_data['issue'][field] = params[field]

        if not issue_data['issue']:
            raise ValueError("No update parameters provided")

        # 課題更新リクエスト
        url = f"{self.base_url}/issues/{issue_id}.json"
        response = requests.put(url, headers=self.headers, json=issue_data)
        response.raise_for_status()

        # 成功時は204 No Contentが返されるため、更新後の最新データを取得
        get_response = requests.get(f"{self.base_url}/issues/{issue_id}.json",
headers=self.headers)
        get_response.raise_for_status()

        return {
            'data': get_response.json(),
            'metadata': {
                'resource_type': 'issue',
                'issue_id': issue_id,
                'action': 'update'
            }
        }
```

9.1.2 Redmineの高度な連携機能

Redmineの特化機能との連携例：

```
# プロジェクトの進捗状況分析
async def analyze_project_progress(mcp, project_id, options=None):
    if options is None:
        options = {}

    include_versions = options.get('include_versions', True)
    include_issues = options.get('include_issues', True)
    calculate_burndown = options.get('calculate_burndown', True)

    # MCPリクエスト
    response = await mcp.query({
        'requestType': 'analyze',
        'resource': f'redmine://{project_id}',
        'params': {
            'include_versions': include_versions,
            'include_issues': include_issues,
            'calculate_burndown': calculate_burndown
        }
    })

    data = response.get('data', {})

    return {
        'project': data.get('project', {}),
        'progress': {
```

```
            'percent_complete': data.get('percent_complete', 0),
            'open_issues': data.get('open_issues', 0),
            'closed_issues': data.get('closed_issues', 0),
            'total_issues': data.get('total_issues', 0)
        },
        'versions': data.get('versions', []),
        'burndown': data.get('burndown', {}),
        'velocity': data.get('velocity', {}),
        'recommendations': data.get('recommendations', [])
    }

# チケット関連のAI生成
async def generate_issue_content(mcp, project_id, options=None):
    if options is None:
        options = {}

    description = options.get('description', '')
    type_id = options.get('type_id')
    related_issues = options.get('related_issues', [])

    # MCPリクエスト
    response = await mcp.query({
        'requestType': 'generate',
        'resource': f'redmine://{project_id}/issues',
        'params': {
            'description': description,
            'type_id': type_id,
            'related_issues': related_issues
        }
    })

    data = response.get('data', {})

    return {
        'subject': data.get('subject', ''),
        'description': data.get('description', ''),
        'acceptance_criteria': data.get('acceptance_criteria', ''),
        'suggested_priority': data.get('suggested_priority'),
        'suggested_assignee': data.get('suggested_assignee'),
        'estimated_hours': data.get('estimated_hours'),
        'suggested_custom_fields': data.get('custom_fields', {})
    }
```

> **若手の疑問解決**: *「Redmineって古いシステムだけど、本当にAIと連携できるの？」*
>
> Redmineは確かに比較的古いシステムですが、安定したRESTful APIを持っているため、MCPコネクタとの連携は十分可能です。APIを通じてプロジェクト、課題、Wikiなどのデータにアクセスできます。制限としては、APIが提供する以上の情報にはアクセスできない点や、リアルタイム通知の仕組みがないことがありますが、適切なポーリング戦略やWebhookの追加実装により、これらの制限も克服できます。むしろ、古いシステムゆえにAPI経由で機能拡張できることが、MCPの大きな価値となります。

9.2 Obsidian MCPコネクタの実装

個人知識管理ツールObsidianとMCPを連携させるコネクタの実装方法を解説します。

9.2.1 Obsidianコネクタの基本アーキテクチャ

```
// Obsidian MCPコネクタの基本実装（Node.js）
const fs = require('fs').promises;
const path = require('path');
const { glob } = require('glob');
const matter = require('gray-matter');
const markdownIt = require('markdown-it');
const md = new markdownIt();

class ObsidianConnector {
  /**
   * Obsidian MCPコネクタの初期化
   * @param {Object} config - コネクタ設定
   * @param {string} config.vaultPath - Obsidianボールトのパス
   * @param {boolean} config.useCache - キャッシュを使用するかのフラグ
   */
  constructor(config) {
    this.vaultPath = config.vaultPath;
    this.useCache = config.useCache || false;
    this.cache = new Map();
    this.lastCacheUpdate = 0;
    this.cacheValidity = 60000; // 1分間キャッシュ有効
  }

  /**
   * MCPリクエストを処理
   * @param {Object} request - MCPリクエスト
   * @returns {Promise<Object>} - 処理結果
   */
  async handleRequest(request) {
    const requestType = request.requestType || '';
    const resource = request.resource || '';
    const params = request.params || {};

    // リソースURIの解析
    const resourceInfo = this._parseResourceUri(resource);

    // リクエストタイプに基づいて処理を分岐
    if (requestType === 'query') {
      return await this._handleQuery(resourceInfo, params);
    } else if (requestType === 'update') {
      return await this._handleUpdate(resourceInfo, params);
    } else if (requestType === 'create') {
      return await this._handleCreate(resourceInfo, params);
    } else if (requestType === 'search') {
      return await this._handleSearch(resourceInfo, params);
    } else {
      throw new Error(`Unsupported request type: ${requestType}`);
    }
  }

  /**
   * Obsidian URIのパース
   * @param {string} uri - リソースURI
   * @returns {Object} - パース結果
```

```
   */
  _parseResourceUri(uri) {
    if (!uri.startsWith('obsidian://')) {
      throw new Error(`Invalid Obsidian resource URI: ${uri}`);
    }

    // プロトコルを削除
    const path = uri.substring(11);
    const parts = path.split('/');

    const result = {
      type: 'vault',
      path: '',
      query: {}
    };

    if (parts.length > 0) {
      // クエリパラメータの抽出
      const lastPart = parts[parts.length - 1];
      const queryIndex = lastPart.indexOf('?');

      if (queryIndex !== -1) {
        const queryString = lastPart.substring(queryIndex + 1);
        parts[parts.length - 1] = lastPart.substring(0, queryIndex);

        // クエリパラメータの解析
        const queryParams = new URLSearchParams(queryString);
        for (const [key, value] of queryParams.entries()) {
          result.query[key] = value;
        }
      }

      if (parts.length === 1) {
        // ボールト全体を指定
        result.type = 'vault';
      } else {
        // ファイルパスを指定
        result.type = 'note';
        result.path = parts.slice(1).join('/');

        // 拡張子が指定されていない場合は .md を付加
        if (!result.path.endsWith('.md')) {
          result.path += '.md';
        }
      }
    }

    return result;
  }

  /**
   * クエリ処理
   * @param {Object} resource - リソース情報
   * @param {Object} params - クエリパラメータ
   * @returns {Promise<Object>} - クエリ結果
   */
  async _handleQuery(resource, params) {
```

```
    const resourceType = resource.type;

    if (resourceType === 'vault') {
      return await this._queryVault(resource, params);
    } else if (resourceType === 'note') {
      return await this._queryNote(resource, params);
    } else {
      throw new Error(`Unsupported resource type: ${resourceType}`);
    }
  }

  /**
   * ボールト情報を取得
   * @param {Object} resource - リソース情報
   * @param {Object} params - クエリパラメータ
   * @returns {Promise<Object>} - ボールト情報
   */
  async _queryVault(resource, params) {
    const includeStats = params.includeStats || false;
    const includeStructure = params.includeStructure || false;

    // キャッシュチェック
    const cacheKey = `vault_${includeStats}_${includeStructure}`;

    if (this.useCache && this.cache.has(cacheKey) &&
        (Date.now() - this.lastCacheUpdate) < this.cacheValidity) {
      return this.cache.get(cacheKey);
    }

    // ボールト内のすべてのマークダウンファイルを検索
    const files = await glob('**/*.md', { cwd: this.vaultPath });

    const result = {
      data: {
        files: files.length,
        fileList: files
      },
      metadata: {
        resource_type: 'vault',
        timestamp: new Date().toISOString()
      }
    };

    // 統計情報の取得
    if (includeStats) {
      const stats = {
        totalFiles: files.length,
        totalSize: 0,
        avgFileSize: 0,
        largestFile: {
          path: '',
          size: 0
        },
        modifiedLast24h: 0,
        tags: new Map(),
        backlinks: new Map()
      };
```

```
      // ファイル情報を収集
      for (const file of files) {
        const filePath = path.join(this.vaultPath, file);
        const fileStat = await fs.stat(filePath);

        stats.totalSize += fileStat.size;

        if (fileStat.size > stats.largestFile.size) {
          stats.largestFile.path = file;
          stats.largestFile.size = fileStat.size;
        }

        const modifiedTime = new Date(fileStat.mtime);
        const now = new Date();
        const hoursDiff = (now - modifiedTime) / (1000 * 60 * 60);

        if (hoursDiff <= 24) {
          stats.modifiedLast24h++;
        }

        // ファイル内容の解析
        const content = await fs.readFile(filePath, 'utf8');
        const { data: frontmatter, content: markdownContent } = matter(content);

        // タグの収集
        const tags = this._extractTags(frontmatter, markdownContent);
        for (const tag of tags) {
          stats.tags.set(tag, (stats.tags.get(tag) || 0) + 1);
        }

        // バックリンクの収集
        const links = this._extractLinks(markdownContent);
        for (const link of links) {
          stats.backlinks.set(link, (stats.backlinks.get(link) || 0) + 1);
        }
      }

      stats.avgFileSize = stats.totalFiles > 0 ? Math.round(stats.totalSize /
stats.totalFiles) : 0;

      // Map型をオブジェクトに変換
      result.data.stats = {
         ...stats,
        tags: Object.fromEntries([...stats.tags.entries()].sort((a, b) => b[1] -
a[1]).slice(0, 20)),
        backlinks: Object.fromEntries([...stats.backlinks.entries()].sort((a, b) =>
b[1] - a[1]).slice(0, 20))
      };
    }

    // フォルダ構造の取得
    if (includeStructure) {
      const structure = {};

      for (const file of files) {
        const parts = file.split('/');
```

```
        let current = structure;

        for (let i = 0; i < parts.length - 1; i++) {
          const part = parts[i];
          if (!current[part]) {
            current[part] = {};
          }
          current = current[part];
        }

        const fileName = parts[parts.length - 1];
        if (!current[fileName]) {
          current[fileName] = null;
        }
      }

      result.data.structure = structure;
    }

    // キャッシュに保存
    if (this.useCache) {
      this.cache.set(cacheKey, result);
      this.lastCacheUpdate = Date.now();
    }

    return result;
  }

  /**
   * ノート情報を取得
   * @param {Object} resource - リソース情報
   * @param {Object} params - クエリパラメータ
   * @returns {Promise<Object>} - ノート情報
   */
  async _queryNote(resource, params) {
    const notePath = resource.path;
    const format = params.format || 'markdown';
    const includeBacklinks = params.includeBacklinks || false;

    const filePath = path.join(this.vaultPath, notePath);

    try {
      // ファイル読み込み
      const content = await fs.readFile(filePath, 'utf8');

      // フロントマターの解析
      const { data: frontmatter, content: markdownContent } = matter(content);

      // 結果の作成
      const result = {
        data: {
          path: notePath,
          content: markdownContent,
          frontmatter
        },
        metadata: {
          resource_type: 'note',
```

```
          timestamp: new Date().toISOString()
        }
      };

      // HTMLへの変換
      if (format === 'html') {
        result.data.html = md.render(markdownContent);
      }

      // ファイル情報の取得
      const fileStat = await fs.stat(filePath);
      result.data.fileInfo = {
        size: fileStat.size,
        created: fileStat.birthtime,
        modified: fileStat.mtime
      };

      // バックリンクの取得
      if (includeBacklinks) {
        result.data.backlinks = await this._findBacklinks(notePath);
      }

      return result;
    } catch (error) {
      if (error.code === 'ENOENT') {
        throw new Error(`Note not found: ${notePath}`);
      }
      throw error;
    }
  }

  /**
   * リソース更新処理
   * @param {Object} resource - リソース情報
   * @param {Object} params - 更新パラメータ
   * @returns {Promise<Object>} - 更新結果
   */
  async _handleUpdate(resource, params) {
    const resourceType = resource.type;

    if (resourceType !== 'note') {
      throw new Error(`Update not supported for resource type: ${resourceType}`);
    }

    return await this._updateNote(resource, params);
  }

  /**
   * ノートを更新
   * @param {Object} resource - リソース情報
   * @param {Object} params - 更新パラメータ
   * @returns {Promise<Object>} - 更新結果
   */
  async _updateNote(resource, params) {
    const notePath = resource.path;
    const filePath = path.join(this.vaultPath, notePath);
```

```
    try {
      // 現在のファイル内容を取得
      const currentContent = await fs.readFile(filePath, 'utf8');
      const { data: currentFrontmatter } = matter(currentContent);

      // 更新内容の準備
      const newContent = params.content;
      const newFrontmatter = params.frontmatter || currentFrontmatter;

      // フロントマターとコンテンツを結合
      let updatedContent;

      if (Object.keys(newFrontmatter).length > 0) {
        updatedContent = matter.stringify(newContent, newFrontmatter);
      } else {
        updatedContent = newContent;
      }

      // ファイル書き込み
      await fs.writeFile(filePath, updatedContent, 'utf8');

      // 更新したノートの情報を返す
      return await this._queryNote(resource, {});
    } catch (error) {
      if (error.code === 'ENOENT') {
        throw new Error(`Note not found: ${notePath}`);
      }
      throw error;
    }
  }

  /**
   * リソース作成処理
   * @param {Object} resource - リソース情報
   * @param {Object} params - 作成パラメータ
   * @returns {Promise<Object>} - 作成結果
   */
  async _handleCreate(resource, params) {
    const resourceType = resource.type;

    if (resourceType !== 'note') {
      throw new Error(`Creation not supported for resource type: ${resourceType}`);
    }

    return await this._createNote(resource, params);
  }

  /**
   * ノートを作成
   * @param {Object} resource - リソース情報
   * @param {Object} params - 作成パラメータ
   * @returns {Promise<Object>} - 作成結果
   */
  async _createNote(resource, params) {
    const notePath = resource.path;
    const filePath = path.join(this.vaultPath, notePath);
```

```
    // ディレクトリの存在確認と作成
    const dirPath = path.dirname(filePath);

    try {
      await fs.mkdir(dirPath, { recursive: true });
    } catch (error) {
      if (error.code !== 'EEXIST') {
        throw error;
      }
    }

    // ファイルの存在確認
    try {
      await fs.access(filePath);
      throw new Error(`Note already exists: ${notePath}`);
    } catch (error) {
      if (error.code !== 'ENOENT') {
        throw error;
      }
    }

    // 内容の準備
    const content = params.content || '';
    const frontmatter = params.frontmatter || {};

    // フロントマターとコンテンツを結合
    let noteContent;

    if (Object.keys(frontmatter).length > 0) {
      noteContent = matter.stringify(content, frontmatter);
    } else {
      noteContent = content;
    }

    // ファイル書き込み
    await fs.writeFile(filePath, noteContent, 'utf8');

    // 作成したノートの情報を返す
    return await this._queryNote(resource, {});
  }

  /**
   * 検索処理
   * @param {Object} resource - リソース情報
   * @param {Object} params - 検索パラメータ
   * @returns {Promise<Object>} - 検索結果
   */
  async _handleSearch(resource, params) {
    const query = params.query;
    const searchType = params.type || 'text';
    const limit = params.limit || 10;

    if (!query) {
      throw new Error('Search query is required');
    }

    // ボールト内のすべてのマークダウンファイルを検索
```

```
    const files = await glob('**/*.md', { cwd: this.vaultPath });

    const results = [];

    for (const file of files) {
      const filePath = path.join(this.vaultPath, file);
      const content = await fs.readFile(filePath, 'utf8');

      let match = false;
      let matchContext = null;

      // 検索タイプに応じた処理
      if (searchType === 'text') {
        // テキスト検索
        const lowerContent = content.toLowerCase();
        const lowerQuery = query.toLowerCase();

        if (lowerContent.includes(lowerQuery)) {
          match = true;

          // マッチした箇所の前後のコンテキストを取得
          const index = lowerContent.indexOf(lowerQuery);
          const start = Math.max(0, index - 60);
          const end = Math.min(content.length, index + query.length + 60);

          matchContext = content.substring(start, end);
        }
      } else if (searchType === 'tag') {
        // タグ検索
        const { data: frontmatter, content: markdownContent } = matter(content);
        const tags = this._extractTags(frontmatter, markdownContent);

        if (tags.includes(query)) {
          match = true;
        }
      } else if (searchType === 'frontmatter') {
        // フロントマター検索
        const { data: frontmatter } = matter(content);

        const searchParts = query.split(':');
        if (searchParts.length === 2) {
          const [key, value] = searchParts;
          if (frontmatter[key] && frontmatter[key].toString().includes(value)) {
            match = true;
          }
        }
      }

      if (match) {
        results.push({
          path: file,
          matchContext
        });

        if (results.length >= limit) {
          break;
        }
```

```
      }
    }

    return {
      data: {
        query,
        type: searchType,
        results,
        totalResults: results.length
      },
      metadata: {
        resource_type: 'search',
        timestamp: new Date().toISOString()
      }
    };
  }

  /**
   * タグを抽出
   * @param {Object} frontmatter - フロントマター
   * @param {string} content - マークダウンコンテンツ
   * @returns {string[]} - 抽出されたタグ
   */
  _extractTags(frontmatter, content) {
    const tags = new Set();

    // フロントマターからタグを抽出
    if (frontmatter && frontmatter.tags) {
      if (Array.isArray(frontmatter.tags)) {
        frontmatter.tags.forEach(tag => tags.add(tag));
      } else if (typeof frontmatter.tags === 'string') {
        frontmatter.tags.split(/[,\s]+/).forEach(tag => {
          if (tag) tags.add(tag);
        });
      }
    }

    // コンテンツからハッシュタグを抽出
    const hashtagRegex = /#([a-zA-Z0-9_\-/]+)/g;
    let match;

    while ((match = hashtagRegex.exec(content)) !== null) {
      tags.add(match[1]);
    }

    return [...tags];
  }

  /**
   * リンクを抽出
   * @param {string} content - マークダウンコンテンツ
   * @returns {string[]} - 抽出されたリンク
   */
  _extractLinks(content) {
    const links = new Set();

    // Wiki形式のリンクを抽出 [[Link]]
```

```
    const wikiLinkRegex = /\[\[([^\]|]+)(?:\|[^\]]+)?\]\]/g;
    let match;

    while ((match = wikiLinkRegex.exec(content)) !== null) {
      links.add(match[1]);
    }

    // マークダウン形式のリンクを抽出 [text](link)
    const markdownLinkRegex = /\[([^\]]+)\]\(([^)]+)\)/g;

    while ((match = markdownLinkRegex.exec(content)) !== null) {
      // ローカルリンクのみ追加（HTTP(S)で始まらないリンク）
      const link = match[2];
      if (!link.startsWith('http://') && !link.startsWith('https://')) {
        links.add(link);
      }
    }

    return [...links];
  }

  /**
   * バックリンクを検索
   * @param {string} notePath - ノートのパス
   * @returns {Promise<Array>} - バックリンクの配列
   */
  async _findBacklinks(notePath) {
    const noteNameWithoutExtension = path.basename(notePath, '.md');
    const files = await glob('**/*.md', { cwd: this.vaultPath });

    const backlinks = [];

    for (const file of files) {
      if (file === notePath) continue;

      const filePath = path.join(this.vaultPath, file);
      const content = await fs.readFile(filePath, 'utf8');

      // Wiki形式のリンクを検索
      const wikiLinkRegex = new RegExp(`\\[\\[${noteNameWithoutExtension}(?:\\|
[^\\]]+)?\\]\\]`, 'g');

      // マークダウン形式のリンクを検索
      const markdownLinkRegex = new RegExp(`\\]\\(${notePath.replace(/\./g,
'\\.')}\\)`, 'g');

      if (wikiLinkRegex.test(content) || markdownLinkRegex.test(content)) {
        backlinks.push({
          path: file,
          type: wikiLinkRegex.test(content) ? 'wikilink' : 'markdown'
        });
      }
    }

    return backlinks;
  }
}
```

```
module.exports = ObsidianConnector;
```

9.2.2 Obsidianナレッジグラフの活用

Obsidianの特化機能との連携例：

```
// ナレッジグラフ分析の実装例
async function analyzeKnowledgeGraph(mcp, options = {}) {
  const {
    vault = null,
    includeTags = true,
    includeLinks = true,
    includeNodes = true,
    centralityMetrics = true,
    clusterAnalysis = true,
    maxNodes = 100
  } = options;

  if (!vault) {
    throw new Error("Vault parameter is required");
  }

  // MCPリクエスト
  const response = await mcp.query({
    requestType: 'analyze',
    resource: `obsidian://${vault}`,
    params: {
      type: 'knowledge-graph',
      includeTags,
      includeLinks,
      includeNodes,
      centralityMetrics,
      clusterAnalysis,
      maxNodes
    }
  });

  return {
    graphData: {
      nodes: response.data.nodes,
      links: response.data.links,
      tags: response.data.tags
    },
    metrics: {
      nodeCount: response.data.nodeCount,
      linkCount: response.data.linkCount,
      density: response.data.density,
      avgConnections: response.data.avgConnections,
      centralNodes: response.data.centralNodes
    },
    clusters: response.data.clusters,
    recommendations: response.data.recommendations
  };
}
```

```
// ナレッジベースのクエリ例
async function queryKnowledge(mcp, queryText, options = {}) {
  const {
    vault = null,
    maxResults = 5,
    includeContext = true,
    searchType = 'semantic'
  } = options;

  if (!vault) {
    throw new Error("Vault parameter is required");
  }

  // MCPリクエスト
  const response = await mcp.query({
    requestType: 'query',
    resource: `obsidian://${vault}`,
    params: {
      query: queryText,
      maxResults,
      includeContext,
      searchType
    }
  });

  return {
    query: queryText,
    results: response.data.results,
    relevanceScores: response.data.scores,
    relatedConcepts: response.data.relatedConcepts,
    suggestedQueries: response.data.suggestedQueries
  };
}

// ノート生成の実装例
async function generateNote(mcp, options = {}) {
  const {
    vault = null,
    title = null,
    description = null,
    relatedNotes = [],
    tags = [],
    template = null,
    includeBacklinks = true,
    suggestReferences = true
  } = options;

  if (!vault || !title) {
    throw new Error("Vault and title parameters are required");
  }

  // MCPリクエスト
  const response = await mcp.query({
    requestType: 'generate',
    resource: `obsidian://${vault}`,
    params: {
      title,
```

```
      description,
      relatedNotes,
      tags,
      template,
      includeBacklinks,
      suggestReferences
    }
  });

  return {
    title: response.data.title,
    content: response.data.content,
    frontmatter: response.data.frontmatter,
    suggestedPath: response.data.path,
    suggestedTags: response.data.suggestedTags,
    references: response.data.references,
    relatedConcepts: response.data.relatedConcepts
  };
}
```

> **プロジェクト事例**: *ある研究機関では、ObsidianとMCPを連携させて「知識探索システム」を構築しました。研究者は自分のObsidianノートに書き込んだアイデアや論文メモに対して、機関内の他の研究ノートや公開論文データベースからの関連情報を自動的に取得できるようになりました。例えば「量子コンピューティングの誤り訂正」に関するノートを書くと、システムは機関内の関連ノートや最新の研究動向、類似コンセプトを提示します。これにより、アイデアの連鎖が促進され、分野横断的な研究協力が30%増加したと報告されています。*

9.3 カスタムMCPコネクタの一般原則

独自サービスやツールのMCPコネクタを開発するための一般原則を解説します。

9.3.1 コネクタの基本インターフェース

```
// TypeScriptによるMCPコネクタインターフェース定義
interface MCPConnector {
  /**
   * コネクタの初期化
   * @param config コネクタ設定
   */
  initialize(config: any): Promise<void>;

  /**
   * リクエスト処理
   * @param request MCPリクエスト
   * @returns 処理結果
   */
  handleRequest(request: MCPRequest): Promise<MCPResponse>;

  /**
   * メタデータ取得
   * @returns コネクタのメタデータ
   */
  getMetadata(): ConnectorMetadata;

  /**
```

```
   * リソースキャパビリティ取得
   * @param resourceUri リソースURI
   * @returns リソースのキャパビリティ
   */
  getCapabilities(resourceUri: string): Promise<ResourceCapabilities>;

  /**
   * コネクタの後処理
   */
  dispose(): Promise<void>;
}

interface MCPRequest {
  requestType: string;            // リクエストタイプ (query, update, create, execute,
etc.)
  resource: string;               // リソースURI
  params?: Record<string, any>;   // リクエストパラメータ
  context?: RequestContext;       // リクエストコンテキスト
}

interface MCPResponse {
  data: any;                      // レスポンスデータ
  metadata?: ResponseMetadata;    // レスポンスメタデータ
  error?: ErrorInfo;              // エラー情報（失敗時）
}

interface ConnectorMetadata {
  id: string;                     // コネクタID
  name: string;                   // コネクタ名
  version: string;                // コネクタバージョン
  description: string;            // コネクタ説明
  protocolVersion: string;        // 対応するMCPプロトコルバージョン
  supportedRequestTypes: string[]; // サポートするリクエストタイプ
  resourceUriPattern: string;     // リソースURIパターン
}

interface ResourceCapabilities {
  canQuery: boolean;              // クエリ可能か
  canUpdate: boolean;             // 更新可能か
  canCreate: boolean;             // 作成可能か
  canDelete: boolean;             // 削除可能か
  canExecute: boolean;            // 実行可能か
  supportedParams: string[];      // サポートするパラメータ
  supportedOperations: string[];  // サポートする操作
}

interface RequestContext {
  sessionId?: string;             // セッションID
  previousRequests?: MCPRequest[]; // 以前のリクエスト
  auth?: AuthInfo;                // 認証情報
  metadata?: Record<string, any>; // その他のメタデータ
}

interface ResponseMetadata {
  timestamp: string;              // レスポンスタイムスタンプ
  source: string;                 // データソース
  processingTime?: number;        // 処理時間（ミリ秒）
```

```
  cacheStatus?: 'hit' | 'miss';   // キャッシュステータス
  resourceVersion?: string;       // リソースバージョン
}

interface ErrorInfo {
  code: string;                   // エラーコード
  message: string;                // エラーメッセージ
  details?: any;                  // 詳細情報
}

interface AuthInfo {
  type: string;                   // 認証タイプ
  token?: string;                 // 認証トークン
  credentials?: Record<string, string>; // 認証情報
}
```

9.3.2 エラー処理と回復メカニズム

コネクタでのエラー処理の一般的なパターン：

```
// エラー処理の実装例
class MCPConnectorError extends Error {
  constructor(code, message, details = null) {
    super(message);
    this.name = 'MCPConnectorError';
    this.code = code;
    this.details = details;
  }

  toResponse() {
    return {
      error: {
        code: this.code,
        message: this.message,
        details: this.details
      }
    };
  }
}

// エラー処理を組み込んだリクエスト処理関数
async function safeHandleRequest(connector, request) {
  try {
    // タイムアウト処理の追加
    const timeoutPromise = new Promise((_, reject) => {
      setTimeout(() => {
        reject(new MCPConnectorError(
          'TIMEOUT',
          'Request processing timed out',
          { request, timeoutMs: 30000 }
        ));
      }, 30000); // 30秒タイムアウト
    });

    // 実際のリクエスト処理
    const resultPromise = connector.handleRequest(request);
```

```
    // タイムアウトとリクエスト処理を競合
    const result = await Promise.race([timeoutPromise, resultPromise]);

    return result;
  } catch (error) {
    // エラーの種類に応じた処理
    if (error instanceof MCPConnectorError) {
      // 既知のエラータイプ
      console.error(`MCP Connector Error: ${error.code} - ${error.message}`);
      return error.toResponse();
    } else if (error.code === 'ECONNREFUSED' || error.code === 'ENOTFOUND') {
      // 接続エラー
      return {
        error: {
          code: 'CONNECTION_ERROR',
          message: 'Failed to connect to the service',
          details: { originalError: error.message }
        }
      };
    } else if (error.response && error.response.status) {
      // HTTP エラー
      return {
        error: {
          code: `HTTP_${error.response.status}`,
          message: error.response.statusText || 'HTTP Error',
          details: {
            status: error.response.status,
            data: error.response.data
          }
        }
      };
    } else {
      // 未知のエラー
      console.error('Unexpected error in connector:', error);
      return {
        error: {
          code: 'INTERNAL_ERROR',
          message: 'An unexpected error occurred',
          details: { message: error.message }
        }
      };
    }
  }
}

// リトライメカニズムの実装
async function withRetry(fn, options = {}) {
  const {
    maxRetries = 3,
    initialDelay = 1000,
    backoffFactor = 2,
    retryableErrors = ['CONNECTION_ERROR', 'TIMEOUT', 'HTTP_429', 'HTTP_503']
  } = options;

  let retries = 0;
  let delay = initialDelay;
```

```
  while (true) {
    try {
      return await fn();
    } catch (error) {
      const errorCode = error.code || (error.error && error.error.code);

      // リトライ可能なエラーかチェック
      const isRetryable = retryableErrors.includes(errorCode);

      if (!isRetryable || retries >= maxRetries) {
        throw error;
      }

      // リトライ前に待機
      console.log(`Retrying after error: ${errorCode} (retry ${retries +
1}/${maxRetries})`);
      await new Promise(resolve => setTimeout(resolve, delay));

      // バックオフ遅延を増加
      retries++;
      delay *= backoffFactor;
    }
  }
}
```

9.3.3 コンテキスト管理とセッション処理

```
// コンテキスト管理の実装例
class ContextManager {
  constructor(options = {}) {
    this.contexts = new Map();
    this.defaultTtl = options.defaultTtl || 3600000; // 1時間
    this.maxContexts = options.maxContexts || 1000;

    // 定期的なクリーンアップ
    setInterval(() => this.cleanup(), 300000); // 5分ごと
  }

  /**
   * コンテキストの作成または取得
   * @param {string} sessionId - セッションID（省略時は新規作成）
   * @param {Object} initialData - 初期データ
   * @returns {Object} コンテキスト
   */
  getOrCreateContext(sessionId = null, initialData = {}) {
    // 新規セッションの場合はIDを生成
    if (!sessionId) {
      sessionId = this.generateSessionId();
    }

    // 既存コンテキストの取得または新規作成
    if (this.contexts.has(sessionId)) {
      const context = this.contexts.get(sessionId);
      context.lastAccessed = Date.now();
```

```
    return context;
  }

  // 最大コンテキスト数のチェック
  if (this.contexts.size >= this.maxContexts) {
    this.evictOldestContext();
  }

  // 新規コンテキストの作成
  const newContext = {
    sessionId,
    data: { ...initialData },
    created: Date.now(),
    lastAccessed: Date.now(),
    expiresAt: Date.now() + this.defaultTtl
  };

  this.contexts.set(sessionId, newContext);
  return newContext;
}

/**
 * コンテキストの更新
 * @param {string} sessionId - セッションID
 * @param {Object} updates - 更新データ
 * @returns {Object} 更新されたコンテキスト
 */
updateContext(sessionId, updates) {
  if (!this.contexts.has(sessionId)) {
    throw new Error(`Context not found: ${sessionId}`);
  }

  const context = this.contexts.get(sessionId);

  // データの深いマージ
  context.data = this.deepMerge(context.data, updates);
  context.lastAccessed = Date.now();

  // TTLの延長
  context.expiresAt = Date.now() + this.defaultTtl;

  return context;
}

/**
 * コンテキストの削除
 * @param {string} sessionId - セッションID
 * @returns {boolean} 削除成功したか
 */
removeContext(sessionId) {
  return this.contexts.delete(sessionId);
}

/**
 * 古いコンテキストのクリーンアップ
 */
cleanup() {
```

```
    const now = Date.now();
    let expiredCount = 0;

    for (const [sessionId, context] of this.contexts.entries()) {
      if (context.expiresAt <= now) {
        this.contexts.delete(sessionId);
        expiredCount++;
      }
    }

    if (expiredCount > 0) {
      console.log(`Cleaned up ${expiredCount} expired contexts`);
    }
  }

  /**
   * 最も古いコンテキストの削除
   */
  evictOldestContext() {
    let oldestSessionId = null;
    let oldestAccess = Date.now();

    for (const [sessionId, context] of this.contexts.entries()) {
      if (context.lastAccessed < oldestAccess) {
        oldestAccess = context.lastAccessed;
        oldestSessionId = sessionId;
      }
    }

    if (oldestSessionId) {
      this.contexts.delete(oldestSessionId);
      console.log(`Evicted oldest context: ${oldestSessionId}`);
    }
  }

  /**
   * セッションIDの生成
   * @returns {string} 生成されたセッションID
   */
  generateSessionId() {
    return 'session-' + Math.random().toString(36).substring(2, 15) +
           Math.random().toString(36).substring(2, 15);
  }

  /**
   * オブジェクトの深いマージ
   * @param {Object} target - ターゲットオブジェクト
   * @param {Object} source - ソースオブジェクト
   * @returns {Object} マージされたオブジェクト
   */
  deepMerge(target, source) {
    const output = { ...target };

    for (const key in source) {
      if (Object.prototype.hasOwnProperty.call(source, key)) {
        if (typeof source[key] === 'object' && source[key] !== null) {
          if (typeof target[key] === 'object' && target[key] !== null) {
```

```
            output[key] = this.deepMerge(target[key], source[key]);
          } else {
            output[key] = { ...source[key] };
          }
        } else {
          output[key] = source[key];
        }
      }
    }

    return output;
  }
}
```

> **ベテランの知恵袋**: *「カスタムMCPコネクタを開発する際の最大の教訓は、初めからすべての機能を実装しようとしないことです。まず、基本的なクエリ機能だけを実装して動作確認し、そこから更新、作成といった機能を段階的に追加していくアプローチが効果的です。また、エラー処理とロギングは後回しにせず、最初から組み込むことで、開発中のデバッグが格段に容易になります。」* - エンタープライズインテグレーションアーキテクト

クラウドサービス連携MCPコネクタチェックリスト

- [ ] サービスAPIのエンドポイントとバージョンの確認
- [ ] 認証方式の選択と実装（OAuth、APIキー等）
- [ ] リソースURI形式の設計とパースロジックの実装
- [ ] 基本的なCRUD操作のマッピング
- [ ] エラー処理とリトライロジックの実装
- [ ] レート制限対応の実装
- [ ] キャッシュ戦略の決定
- [ ] コンテキスト管理メカニズムの実装
- [ ] ロギングとモニタリングの統合
- [ ] セキュリティリスク評価の実施

第10章: 自作MCPサーバーの実装 - Python/React/C#による実装例

10.1 Python実装のMCPサーバー

FastAPIを使った軽量MCPサーバーの実装例を解説します。

10.1.1 基本構造と設定

```python
# main.py - FastAPIを使ったMCPサーバーの基本構造
import os
import uuid
import json
import logging
from datetime import datetime, timedelta
from typing import Dict, Any, List, Optional, Union

import uvicorn
from fastapi import FastAPI, HTTPException, Depends, Header, WebSocket,
WebSocketDisconnect
from fastapi.middleware.cors import CORSMiddleware
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm
from jose import JWTError, jwt
from passlib.context import CryptContext
from pydantic import BaseModel, Field

# ロギング設定
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    handlers=[
        logging.StreamHandler(),
        logging.FileHandler("mcp_server.log")
    ]
)
logger = logging.getLogger("mcp_server")

# 設定
SECRET_KEY = os.getenv("MCP_SECRET_KEY", "development-secret-key")
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES = 30

# JWT認証ユーティリティ
pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")
oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

# モデル定義
class User(BaseModel):
    username: str
    email: Optional[str] = None
    full_name: Optional[str] = None
    disabled: Optional[bool] = None

class UserInDB(User):
    hashed_password: str
```

```python
class Token(BaseModel):
    access_token: str
    token_type: str

class TokenData(BaseModel):
    username: Optional[str] = None

class MCPRequest(BaseModel):
    version: str = Field(..., description="MCP protocol version")
    requestType: str = Field(..., description="Request type (query, update, create,
etc.)")
    resource: str = Field(..., description="Resource URI")
    params: Dict[str, Any] = Field(default_factory=dict, description="Request
parameters")
    context: Optional[Dict[str, Any]] = Field(default=None, description="Request
context")

class MCPResponse(BaseModel):
    version: str = Field(..., description="MCP protocol version")
    status: int = Field(..., description="Response status code")
    requestId: str = Field(..., description="Request ID")
    data: Optional[Any] = Field(default=None, description="Response data")
    error: Optional[Dict[str, Any]] = Field(default=None, description="Error
information")
    context: Optional[Dict[str, Any]] = Field(default=None, description="Updated
context")

# ダミーユーザーデータベース
fake_users_db = {
    "admin": {
        "username": "admin",
        "full_name": "Admin User",
        "email": "admin@example.com",
        "hashed_password": pwd_context.hash("adminpass"),
        "disabled": False,
    },
    "user": {
        "username": "user",
        "full_name": "Test User",
        "email": "user@example.com",
        "hashed_password": pwd_context.hash("userpass"),
        "disabled": False,
    }
}

# コネクタマネージャーの定義
class ConnectorManager:
    def __init__(self):
        self.connectors = {}

    def register_connector(self, connector_id, connector):
        self.connectors[connector_id] = connector
        logger.info(f"Registered connector: {connector_id}")

    def get_connector(self, connector_id):
        return self.connectors.get(connector_id)
```

```
    def get_connector_for_resource(self, resource_uri):
        # リソースURIからコネクタを特定
        if "://" in resource_uri:
            protocol = resource_uri.split("://")[0]
            return self.connectors.get(protocol)
        return None

# コンテキストマネージャーの定義
class ContextManager:
    def __init__(self):
        self.contexts = {}

    def get_context(self, session_id):
        return self.contexts.get(session_id)

    def create_context(self, initial_data=None):
        session_id = str(uuid.uuid4())
        context = {
            "sessionId": session_id,
            "created": datetime.now().isoformat(),
            "lastAccessed": datetime.now().isoformat(),
            "data": initial_data or {}
        }
        self.contexts[session_id] = context
        return context

    def update_context(self, session_id, updates):
        if session_id not in self.contexts:
            return None

        context = self.contexts[session_id]
        context["data"].update(updates)
        context["lastAccessed"] = datetime.now().isoformat()

        return context

    def cleanup_old_contexts(self, max_age_hours=24):
        """古いコンテキストを削除"""
        cutoff = datetime.now() - timedelta(hours=max_age_hours)
        to_delete = []

        for session_id, context in self.contexts.items():
            last_accessed = datetime.fromisoformat(context["lastAccessed"])
            if last_accessed < cutoff:
                to_delete.append(session_id)

        for session_id in to_delete:
            del self.contexts[session_id]

        return len(to_delete)

# WebSocketコネクション管理
class ConnectionManager:
    def __init__(self):
        self.active_connections: Dict[str, WebSocket] = {}

    async def connect(self, websocket: WebSocket, client_id: str):
```

```
        await websocket.accept()
        self.active_connections[client_id] = websocket
        logger.info(f"WebSocket client connected: {client_id}")

    def disconnect(self, client_id: str):
        if client_id in self.active_connections:
            del self.active_connections[client_id]
            logger.info(f"WebSocket client disconnected: {client_id}")

    async def send_message(self, client_id: str, message: Any):
        if client_id in self.active_connections:
            websocket = self.active_connections[client_id]
            await websocket.send_json(message)

    async def broadcast(self, message: Any):
        for client_id, websocket in self.active_connections.items():
            await websocket.send_json(message)

# アプリケーション初期化
app = FastAPI(title="MCP Server", version="1.0.0")

# CORS設定
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],  # 本番環境では適切に制限する
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# グローバルインスタンス
connector_manager = ConnectorManager()
context_manager = ContextManager()
connection_manager = ConnectionManager()

# 認証関数
def verify_password(plain_password, hashed_password):
    return pwd_context.verify(plain_password, hashed_password)

def get_password_hash(password):
    return pwd_context.hash(password)

def get_user(db, username: str):
    if username in db:
        user_dict = db[username]
        return UserInDB(**user_dict)
    return None

def authenticate_user(fake_db, username: str, password: str):
    user = get_user(fake_db, username)
    if not user:
        return False
    if not verify_password(password, user.hashed_password):
        return False
    return user

def create_access_token(data: dict, expires_delta: Optional[timedelta] = None):
```

```
    to_encode = data.copy()
    if expires_delta:
        expire = datetime.utcnow() + expires_delta
    else:
        expire = datetime.utcnow() + timedelta(minutes=15)
    to_encode.update({"exp": expire})
    encoded_jwt = jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)
    return encoded_jwt

async def get_current_user(token: str = Depends(oauth2_scheme)):
    credentials_exception = HTTPException(
        status_code=401,
        detail="Could not validate credentials",
        headers={"WWW-Authenticate": "Bearer"},
    )
    try:
        payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
        username: str = payload.get("sub")
        if username is None:
            raise credentials_exception
        token_data = TokenData(username=username)
    except JWTError:
        raise credentials_exception
    user = get_user(fake_users_db, username=token_data.username)
    if user is None:
        raise credentials_exception
    return user

async def get_current_active_user(current_user: User = Depends(get_current_user)):
    if current_user.disabled:
        raise HTTPException(status_code=400, detail="Inactive user")
    return current_user

# 認証エンドポイント
@app.post("/token", response_model=Token)
async def login_for_access_token(form_data: OAuth2PasswordRequestForm = Depends()):
    user = authenticate_user(fake_users_db, form_data.username, form_data.password)
    if not user:
        raise HTTPException(
            status_code=401,
            detail="Incorrect username or password",
            headers={"WWW-Authenticate": "Bearer"},
        )
    access_token_expires = timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
    access_token = create_access_token(
        data={"sub": user.username}, expires_delta=access_token_expires
    )
    return {"access_token": access_token, "token_type": "bearer"}

@app.get("/users/me", response_model=User)
async def read_users_me(current_user: User = Depends(get_current_active_user)):
    return current_user
```

10.1.2 MCPリクエスト処理実装

```python
# MCP処理エンドポイント
@app.post("/mcp", response_model=MCPResponse)
async def process_mcp_request(
    request: MCPRequest,
    authorization: Optional[str] = Header(None),
    current_user: User = Depends(get_current_active_user)
):
    logger.info(f"Received MCP request: {request.requestType} to {request.resource}")

    # リクエストIDの生成
    request_id = str(uuid.uuid4())

    try:
        # バージョン確認
        if request.version not in ["1.0", "1.1"]:
            return MCPResponse(
                version=request.version,
                status=400,
                requestId=request_id,
                error={
                    "code": "UNSUPPORTED_VERSION",
                    "message": f"Unsupported MCP version: {request.version}"
                }
            )

        # コンテキスト処理
        session_context = None
        if request.context and "sessionId" in request.context:
            session_context = context_manager.get_context(request.context["sessionId"])

            if not session_context:
                # セッションが見つからない場合は新規作成
                session_context = context_manager.create_context()
        else:
            # 新しいコンテキストを作成
            session_context = context_manager.create_context()

        # リソースからコネクタを特定
        connector = connector_manager.get_connector_for_resource(request.resource)
        if not connector:
            return MCPResponse(
                version=request.version,
                status=404,
                requestId=request_id,
                error={
                    "code": "CONNECTOR_NOT_FOUND",
                    "message": f"No connector available for resource:
{request.resource}"
                }
            )

        # リクエスト処理の実行
        result = await connector.handle_request({
            "requestType": request.requestType,
            "resource": request.resource,
            "params": request.params,
```

```
            "context": {
                "sessionId": session_context["sessionId"],
                "data": session_context["data"],
                "user": {
                    "username": current_user.username,
                    "roles": ["user"]  # 実際の環境では適切なロール管理が必要
                }
            }
        })

        # コンテキストの更新
        if result.get("context"):
            context_manager.update_context(
                session_context["sessionId"],
                result["context"].get("data", {})
            )

        # 成功レスポンス
        return MCPResponse(
            version=request.version,
            status=200,
            requestId=request_id,
            data=result.get("data"),
            context={
                "sessionId": session_context["sessionId"],
                "updated": datetime.now().isoformat()
            }
        )

    except Exception as e:
        logger.exception(f"Error processing MCP request: {str(e)}")

        # エラーレスポンス
        return MCPResponse(
            version=request.version,
            status=500,
            requestId=request_id,
            error={
                "code": "INTERNAL_ERROR",
                "message": "An internal error occurred while processing the request",
                "details": str(e)
            },
            context={
                "sessionId": session_context["sessionId"] if session_context else None
            }
        )
```

10.1.3 WebSocketストリーミング実装

```
# WebSocketエンドポイント
@app.websocket("/mcp/stream")
async def websocket_endpoint(websocket: WebSocket):
    client_id = str(uuid.uuid4())

    try:
```

```
        # 接続確立
        await connection_manager.connect(websocket, client_id)

        # 初期応答
        await websocket.send_json({
            "type": "connection_established",
            "clientId": client_id,
            "timestamp": datetime.now().isoformat()
        })

        # メッセージ処理ループ
        while True:
            # メッセージ受信
            data = await websocket.receive_json()

            try:
                # ストリーミングリクエスト処理
                if "requestType" in data and "resource" in data:
                    # セッション処理
                    session_id = None
                    if "context" in data and "sessionId" in data["context"]:
                        session_id = data["context"]["sessionId"]
                        context = context_manager.get_context(session_id)

                    if not session_id or not context:
                        context = context_manager.create_context()
                        session_id = context["sessionId"]

                    # リソースからコネクタを特定
                    connector =
connector_manager.get_connector_for_resource(data["resource"])

                    if not connector:
                        await websocket.send_json({
                            "type": "error",
                            "code": "CONNECTOR_NOT_FOUND",
                            "message": f"No connector available for resource:
{data['resource']}"
                        })
                        continue

                    # ストリーミング処理（非同期）
                    async def process_streaming():
                        try:
                            async for chunk in connector.handle_streaming({
                                "requestType": data["requestType"],
                                "resource": data["resource"],
                                "params": data.get("params", {}),
                                "context": {
                                    "sessionId": session_id,
                                    "data": context["data"]
                                }
                            }):
                                # クライアントが切断されていないか確認
                                if client_id not in
connection_manager.active_connections:
                                    break
```

```
                            # チャンクをクライアントに送信
                            await connection_manager.send_message(client_id, {
                                "type": "chunk",
                                "data": chunk,
                                "timestamp": datetime.now().isoformat()
                            })

                        # 完了通知
                        await connection_manager.send_message(client_id, {
                            "type": "complete",
                            "timestamp": datetime.now().isoformat()
                        })

                    except Exception as e:
                        logger.exception(f"Streaming error: {str(e)}")

                        # エラー通知
                        await connection_manager.send_message(client_id, {
                            "type": "error",
                            "code": "STREAMING_ERROR",
                            "message": str(e),
                            "timestamp": datetime.now().isoformat()
                        })

                # 非同期処理を開始
                import asyncio
                asyncio.create_task(process_streaming())

            # 他のメッセージタイプの処理
            elif "type" in data and data["type"] == "ping":
                await websocket.send_json({
                    "type": "pong",
                    "timestamp": datetime.now().isoformat()
                })

            else:
                await websocket.send_json({
                    "type": "error",
                    "code": "INVALID_MESSAGE",
                    "message": "Invalid message format",
                    "timestamp": datetime.now().isoformat()
                })

        except Exception as e:
            logger.exception(f"WebSocket message processing error: {str(e)}")

            # エラー通知
            await websocket.send_json({
                "type": "error",
                "code": "PROCESSING_ERROR",
                "message": str(e),
                "timestamp": datetime.now().isoformat()
            })

except WebSocketDisconnect:
    # 接続切断時の処理
```

```
        connection_manager.disconnect(client_id)

    except Exception as e:
        logger.exception(f"WebSocket error: {str(e)}")

        # 既に接続されている場合のみ切断処理
        if client_id in connection_manager.active_connections:
            connection_manager.disconnect(client_id)
```

10.1.4 サンプルコネクタの登録とサーバー起動

```
# サンプルコネクタの実装
class EchoConnector:
    """エコーコネクタ（デモ用）"""

    async def handle_request(self, request):
        """リクエスト処理"""
        return {
            "data": {
                "echo": request,
                "timestamp": datetime.now().isoformat()
            }
        }

    async def handle_streaming(self, request):
        """ストリーミング処理"""
        # テスト用のストリーミングレスポンス
        for i in range(5):
            await asyncio.sleep(1)  # 1秒ごとにチャンクを送信
            yield {
                "chunk": i + 1,
                "data": f"Echo streaming chunk {i + 1}",
                "progress": (i + 1) * 20
            }

# サンプルファイルコネクタ
class FileConnector:
    """ローカルファイルコネクタ（デモ用）"""

    def __init__(self, base_path):
        self.base_path = base_path

    async def handle_request(self, request):
        """リクエスト処理"""
        request_type = request["requestType"]
        resource = request["resource"]
        params = request.get("params", {})

        # リソースURIからファイルパスを抽出
        if not resource.startswith("file://"):
            return {
                "error": {
                    "code": "INVALID_RESOURCE",
                    "message": "Resource must start with file://"
                }
```

```
            }

        file_path = resource[7:]  # "file://" を削除

        # セキュリティチェック - パストラバーサル防止
        normalized_path = os.path.normpath(file_path)
        if normalized_path.startswith("..") or normalized_path.startswith("/"):
            return {
                "error": {
                    "code": "SECURITY_ERROR",
                    "message": "Invalid file path"
                }
            }

        full_path = os.path.join(self.base_path, normalized_path)

        # リクエストタイプに応じた処理
        if request_type == "query":
            # ファイル読み込み
            try:
                if not os.path.exists(full_path):
                    return {
                        "error": {
                            "code": "FILE_NOT_FOUND",
                            "message": f"File not found: {file_path}"
                        }
                    }

                # ディレクトリの場合はファイル一覧を返す
                if os.path.isdir(full_path):
                    files = os.listdir(full_path)
                    return {
                        "data": {
                            "type": "directory",
                            "path": file_path,
                            "files": files
                        }
                    }

                # ファイルの場合は内容を返す
                with open(full_path, "r", encoding="utf-8") as f:
                    content = f.read()

                return {
                    "data": {
                        "type": "file",
                        "path": file_path,
                        "content": content,
                        "size": os.path.getsize(full_path),
                        "modified":
datetime.fromtimestamp(os.path.getmtime(full_path)).isoformat()
                    }
                }

            except Exception as e:
                return {
                    "error": {
```

```
                        "code": "FILE_READ_ERROR",
                        "message": str(e)
                    }
                }

        elif request_type == "update":
            # ファイル更新
            try:
                content = params.get("content")
                if content is None:
                    return {
                        "error": {
                            "code": "MISSING_CONTENT",
                            "message": "Content parameter is required for update"
                        }
                    }

                # ディレクトリが存在しない場合は作成
                os.makedirs(os.path.dirname(full_path), exist_ok=True)

                # ファイル書き込み
                with open(full_path, "w", encoding="utf-8") as f:
                    f.write(content)

                return {
                    "data": {
                        "type": "file",
                        "path": file_path,
                        "size": os.path.getsize(full_path),
                        "modified":
datetime.fromtimestamp(os.path.getmtime(full_path)).isoformat(),
                        "status": "updated"
                    }
                }

            except Exception as e:
                return {
                    "error": {
                        "code": "FILE_WRITE_ERROR",
                        "message": str(e)
                    }
                }

        else:
            return {
                "error": {
                    "code": "UNSUPPORTED_REQUEST_TYPE",
                    "message": f"Unsupported request type: {request_type}"
                }
            }

    async def handle_streaming(self, request):
        """ストリーミング処理"""
        resource = request["resource"]

        # リソースURIからファイルパスを抽出
        if not resource.startswith("file://"):
```

```
            yield {
                "error": {
                    "code": "INVALID_RESOURCE",
                    "message": "Resource must start with file://"
                }
            }
            return

        file_path = resource[7:]  # "file://" を削除

        # セキュリティチェック
        normalized_path = os.path.normpath(file_path)
        if normalized_path.startswith("..") or normalized_path.startswith("/"):
            yield {
                "error": {
                    "code": "SECURITY_ERROR",
                    "message": "Invalid file path"
                }
            }
            return

        full_path = os.path.join(self.base_path, normalized_path)

        try:
            if not os.path.exists(full_path) or not os.path.isfile(full_path):
                yield {
                    "error": {
                        "code": "FILE_NOT_FOUND",
                        "message": f"File not found: {file_path}"
                    }
                }
                return

            # ファイルサイズを取得
            file_size = os.path.getsize(full_path)

            # ファイルを分割して読み込み
            chunk_size = 1024  # 1KB
            bytes_read = 0

            with open(full_path, "r", encoding="utf-8") as f:
                while True:
                    chunk = f.read(chunk_size)
                    if not chunk:
                        break

                    bytes_read += len(chunk.encode("utf-8"))
                    progress = min(100, int(bytes_read / file_size * 100))

                    yield {
                        "data": chunk,
                        "progress": progress,
                        "total_size": file_size,
                        "bytes_read": bytes_read
                    }

                    await asyncio.sleep(0.1)  # 読み込み間隔
```

```
        except Exception as e:
            yield {
                "error": {
                    "code": "STREAMING_ERROR",
                    "message": str(e)
                }
            }

# サーバー初期化時のコネクタ登録
@app.on_event("startup")
async def startup_event():
    # エコーコネクタの登録
    connector_manager.register_connector("echo", EchoConnector())

    # ファイルコネクタの登録
    file_connector = FileConnector(base_path="./data")
    connector_manager.register_connector("file", file_connector)

    logger.info("MCP Server started with connectors: echo, file")

# 定期的なメンテナンス処理
@app.on_event("startup")
async def schedule_maintenance():
    async def maintenance_task():
        while True:
            # 古いコンテキストのクリーンアップ
            cleaned = context_manager.cleanup_old_contexts(max_age_hours=24)
            if cleaned > 0:
                logger.info(f"Cleaned up {cleaned} old contexts")

            # 他のメンテナンス処理...

            # 1時間ごとに実行
            await asyncio.sleep(3600)

    # バックグラウンドタスクとして実行
    asyncio.create_task(maintenance_task())

# メインエントリーポイント
if __name__ == "__main__":
    # サーバー起動
    uvicorn.run(
        "main:app",
        host="0.0.0.0",
        port=8000,
        reload=True  # 開発時のみ有効にする
    )
```

10.1.5 高度なセキュリティ実装

```
# セキュリティ強化機能
from fastapi import Request, Response
import secrets
import hashlib
```

```python
from fastapi.middleware.trustedhost import TrustedHostMiddleware
from fastapi.middleware.gzip import GZipMiddleware
import time

# セキュリティヘッダーミドルウェア
@app.middleware("http")
async def add_security_headers(request: Request, call_next):
    response = await call_next(request)

    # セキュリティヘッダーの追加
    response.headers["X-Content-Type-Options"] = "nosniff"
    response.headers["X-Frame-Options"] = "DENY"
    response.headers["X-XSS-Protection"] = "1; mode=block"
    response.headers["Strict-Transport-Security"] = "max-age=31536000;
includeSubDomains"
    response.headers["Content-Security-Policy"] = "default-src 'self'; script-src
'self'"

    return response

# レート制限ミドルウェア
class RateLimiter:
    def __init__(self, requests_per_minute=60):
        self.requests_per_minute = requests_per_minute
        self.request_records = {}  # {ip: [(timestamp, count), ...]}
        self.window_size = 60  # 1分間

    def is_rate_limited(self, ip: str) -> bool:
        current_time = time.time()

        # 古いレコードを削除
        if ip in self.request_records:
            self.request_records[ip] = [
                record for record in self.request_records[ip]
                if current_time - record[0] < self.window_size
            ]
        else:
            self.request_records[ip] = []

        # 現在のウィンドウ内のリクエスト数をカウント
        total_requests = sum(record[1] for record in self.request_records[ip])

        # レート制限をチェック
        if total_requests >= self.requests_per_minute:
            return True

        # リクエストを記録
        self.request_records[ip].append((current_time, 1))

        return False

rate_limiter = RateLimiter(requests_per_minute=120)  # 1分あたり120リクエスト

@app.middleware("http")
async def rate_limit_middleware(request: Request, call_next):
    # クライアントIPの取得
    client_ip = request.client.host
```

```
    # APIエンドポイントのみレート制限を適用
    if request.url.path.startswith("/mcp"):
        if rate_limiter.is_rate_limited(client_ip):
            return Response(
                content=json.dumps({
                    "error": {
                        "code": "RATE_LIMITED",
                        "message": "Too many requests, please try again later"
                    }
                }),
                status_code=429,
                media_type="application/json"
            )

    response = await call_next(request)
    return response

# リクエストログミドルウェア
@app.middleware("http")
async def log_requests(request: Request, call_next):
    start_time = time.time()

    # リクエスト情報をログに記録
    logger.info(
        f"Request: {request.method} {request.url.path} "
        f"Client: {request.client.host}"
    )

    # リクエスト処理
    response = await call_next(request)

    # レスポンス情報をログに記録
    process_time = (time.time() - start_time) * 1000
    logger.info(
        f"Response: {response.status_code} "
        f"Process time: {process_time:.2f}ms"
    )

    return response

# 信頼できるホストの制限
app.add_middleware(
    TrustedHostMiddleware,
    allowed_hosts=["localhost", "127.0.0.1", "mcp-server.example.com"]
)

# GZip圧縮
app.add_middleware(GZipMiddleware, minimum_size=1000)

# APIキー認証（HTTPベーシック認証の代わりに）
class APIKey:
    def __init__(self, key_value: str, owner: str, scopes: List[str]):
        self.key_value = key_value
        self.owner = owner
        self.scopes = scopes
```

```python
# サンプルAPIキー（本番環境では安全な方法でキーを管理する）
api_keys = {
    "test-api-key-1": APIKey(
        "test-api-key-1",
        "test-client",
        ["read", "write"]
    ),
    "test-api-key-2": APIKey(
        "test-api-key-2",
        "readonly-client",
        ["read"]
    )
}

async def get_api_key(api_key: str = Header(None, alias="X-API-Key")):
    if api_key is None:
        raise HTTPException(
            status_code=403,
            detail="API key is missing"
        )

    if api_key not in api_keys:
        raise HTTPException(
            status_code=403,
            detail="Invalid API key"
        )

    return api_keys[api_key]

# APIキー認証を使用するエンドポイント
@app.post("/api/v1/mcp", response_model=MCPResponse)
async def api_mcp_request(
    request: MCPRequest,
    api_key: APIKey = Depends(get_api_key)
):
    # スコープチェック
    if request.requestType in ["update", "create", "delete"] and "write" not in
api_key.scopes:
        return MCPResponse(
            version=request.version,
            status=403,
            requestId=str(uuid.uuid4()),
            error={
                "code": "INSUFFICIENT_SCOPE",
                "message": "The API key does not have write permission"
            }
        )

    # リクエスト処理（認証ユーザーなしでコンテキストを使用）
    # 実際の実装ではここで適切なコンテキスト管理と権限チェックを行う

    # メインのMCPリクエスト処理と同様の実装
    # ...

    # サンプルレスポンス
    return MCPResponse(
        version=request.version,
```

```
        status=200,
        requestId=str(uuid.uuid4()),
        data={
            "message": "Request processed via API key",
            "owner": api_key.owner
        }
    )
```

10.1.6 プラグイン機能の実装

```
# プラグイン機能実装
import importlib.util
import inspect
import sys
from typing import Dict, List, Type, Callable

class MCPPlugin:
    """MCPプラグインの基底クラス"""

    id: str = "base-plugin"  # プラグインID
    name: str = "Base Plugin"  # プラグイン名
    version: str = "1.0.0"  # プラグインバージョン

    def __init__(self, server):
        """
        プラグインの初期化

        Args:
            server: MCPサーバーインスタンス
        """
        self.server = server

    async def initialize(self):
        """初期化処理"""
        pass

    async def on_request(self, request: Dict[str, Any], context: Dict[str, Any]) ->
Dict[str, Any]:
        """
        リクエスト処理前のフック

        Args:
            request: MCPリクエスト
            context: リクエストコンテキスト

        Returns:
            変更されたリクエスト
        """
        return request

    async def on_response(self, response: Dict[str, Any], context: Dict[str, Any]) ->
Dict[str, Any]:
        """
        レスポンス処理前のフック
```

```
        Args:
            response: MCPレスポンス
            context: リクエストコンテキスト

        Returns:
            変更されたレスポンス
        """
        return response

    async def shutdown(self):
        """シャットダウン処理"""
        pass

class PluginManager:
    """MCPプラグイン管理"""

    def __init__(self, server):
        self.server = server
        self.plugins: Dict[str, MCPPlugin] = {}
        self.plugin_hooks: Dict[str, List[Callable]] = {
            "on_request": [],
            "on_response": [],
        }

    def register_plugin(self, plugin_class: Type[MCPPlugin]):
        """
        プラグインを登録

        Args:
            plugin_class: プラグインクラス
        """
        plugin = plugin_class(self.server)
        plugin_id = plugin.id

        if plugin_id in self.plugins:
            logger.warning(f"Plugin with ID '{plugin_id}' is already registered")
            return

        self.plugins[plugin_id] = plugin

        # フックの登録
        if hasattr(plugin, "on_request"):
            self.plugin_hooks["on_request"].append(plugin.on_request)

        if hasattr(plugin, "on_response"):
            self.plugin_hooks["on_response"].append(plugin.on_response)

        logger.info(f"Registered plugin: {plugin.name} (v{plugin.version})")

    async def initialize_plugins(self):
        """すべてのプラグインを初期化"""
        for plugin_id, plugin in self.plugins.items():
            try:
                await plugin.initialize()
                logger.info(f"Initialized plugin: {plugin.name}")
            except Exception as e:
                logger.error(f"Failed to initialize plugin '{plugin_id}': {str(e)}")
```

```python
    async def shutdown_plugins(self):
        """すべてのプラグインをシャットダウン"""
        for plugin_id, plugin in self.plugins.items():
            try:
                await plugin.shutdown()
                logger.info(f"Shutdown plugin: {plugin.name}")
            except Exception as e:
                logger.error(f"Failed to shutdown plugin '{plugin_id}': {str(e)}")

    async def apply_request_hooks(self, request: Dict[str, Any], context: Dict[str,
Any]) -> Dict[str, Any]:
        """
        リクエストフックを適用

        Args:
            request: MCPリクエスト
            context: リクエストコンテキスト

        Returns:
            変更されたリクエスト
        """
        current_request = request

        for hook in self.plugin_hooks["on_request"]:
            try:
                current_request = await hook(current_request, context)
            except Exception as e:
                logger.error(f"Error in request hook: {str(e)}")

        return current_request

    async def apply_response_hooks(self, response: Dict[str, Any], context: Dict[str,
Any]) -> Dict[str, Any]:
        """
        レスポンスフックを適用

        Args:
            response: MCPレスポンス
            context: リクエストコンテキスト

        Returns:
            変更されたレスポンス
        """
        current_response = response

        for hook in self.plugin_hooks["on_response"]:
            try:
                current_response = await hook(current_response, context)
            except Exception as e:
                logger.error(f"Error in response hook: {str(e)}")

        return current_response

    def load_plugins_from_directory(self, directory: str):
        """
        ディレクトリからプラグインを読み込む
```

```
        Args:
            directory: プラグインディレクトリのパス
        """
        plugin_files = glob.glob(os.path.join(directory, "*.py"))

        for plugin_file in plugin_files:
            try:
                # モジュール名を取得
                module_name = os.path.basename(plugin_file)[:-3]  # .py を削除

                # モジュールをロード
                spec = importlib.util.spec_from_file_location(module_name, plugin_file)
                module = importlib.util.module_from_spec(spec)
                sys.modules[module_name] = module
                spec.loader.exec_module(module)

                # プラグインクラスを検索
                for item_name in dir(module):
                    item = getattr(module, item_name)

                    if (inspect.isclass(item) and
                        issubclass(item, MCPPlugin) and
                        item is not MCPPlugin):
                        # プラグインクラスを登録
                        self.register_plugin(item)

            except Exception as e:
                logger.error(f"Failed to load plugin from '{plugin_file}': {str(e)}")

# プラグインマネージャーの初期化
plugin_manager = PluginManager(app)

# サンプルプラグイン
class LoggingPlugin(MCPPlugin):
    """リクエスト/レスポンスのロギングプラグイン"""

    id = "logging-plugin"
    name = "Logging Plugin"
    version = "1.0.0"

    async def initialize(self):
        logger.info("Logging plugin initialized")

    async def on_request(self, request, context):
        logger.info(f"MCP Request: {request['requestType']} to {request['resource']}")
        return request

    async def on_response(self, response, context):
        if "error" in response:
            logger.warning(f"MCP Response Error: {response['error']}")
        return response

    async def shutdown(self):
        logger.info("Logging plugin shutdown")

class MetricsPlugin(MCPPlugin):
```

```python
    """メトリクス収集プラグイン"""

    id = "metrics-plugin"
    name = "Metrics Plugin"
    version = "1.0.0"

    async def initialize(self):
        self.request_count = 0
        self.error_count = 0
        self.response_times = []
        logger.info("Metrics plugin initialized")

    async def on_request(self, request, context):
        self.request_count += 1
        context["metrics_start_time"] = time.time()
        return request

    async def on_response(self, response, context):
        if "error" in response:
            self.error_count += 1

        if "metrics_start_time" in context:
            response_time = time.time() - context["metrics_start_time"]
            self.response_times.append(response_time)

            # 定期的にメトリクスをログに出力
            if self.request_count % 100 == 0:
                avg_response_time = sum(self.response_times) / len(self.response_times)
if self.response_times else 0
                logger.info(f"Metrics: {self.request_count} requests,
{self.error_count} errors, "
                            f"avg response time: {avg_response_time:.3f}s")

                # 最新の100件だけ保持
                self.response_times = self.response_times[-100:]

        return response

# サーバー起動時にプラグインを登録
@app.on_event("startup")
async def setup_plugins():
    # 内蔵プラグインを登録
    plugin_manager.register_plugin(LoggingPlugin)
    plugin_manager.register_plugin(MetricsPlugin)

    # 外部プラグインをディレクトリから読み込む
    plugin_manager.load_plugins_from_directory("./plugins")

    # プラグインを初期化
    await plugin_manager.initialize_plugins()

# サーバーシャットダウン時にプラグインをクリーンアップ
@app.on_event("shutdown")
async def cleanup_plugins():
    await plugin_manager.shutdown_plugins()

# プラグインを組み込んだMCPリクエスト処理
```

```python
@app.post("/mcp/v2", response_model=MCPResponse)
async def process_mcp_request_with_plugins(
    request: MCPRequest,
    current_user: User = Depends(get_current_active_user)
):
    logger.info(f"Received MCP v2 request: {request.requestType} to
{request.resource}")

    # リクエストIDの生成
    request_id = str(uuid.uuid4())

    # コンテキスト準備
    context = {
        "user": {
            "username": current_user.username,
            "email": current_user.email
        },
        "request_id": request_id,
        "timestamp": datetime.now().isoformat()
    }

    try:
        # リクエストをディクショナリに変換
        request_dict = request.dict()

        # プラグインのリクエストフックを適用
        modified_request = await plugin_manager.apply_request_hooks(request_dict,
context)

        # コネクタ処理（省略）...
        connector =
connector_manager.get_connector_for_resource(modified_request["resource"])
        # ...

        # サンプルレスポンス
        response = {
            "version": request.version,
            "status": 200,
            "requestId": request_id,
            "data": {
                "message": "Request processed with plugins",
                "original_resource": request.resource,
                "modified_resource": modified_request["resource"]
            }
        }

        # プラグインのレスポンスフックを適用
        modified_response = await plugin_manager.apply_response_hooks(response,
context)

        return MCPResponse(**modified_response)

    except Exception as e:
        logger.exception(f"Error processing MCP request: {str(e)}")

        # エラーレスポンス
        error_response = {
```

```
            "version": request.version,
            "status": 500,
            "requestId": request_id,
            "error": {
                "code": "INTERNAL_ERROR",
                "message": str(e)
            }
        }

        # プラグインのレスポンスフックを適用（エラー時も）
        modified_error = await plugin_manager.apply_response_hooks(error_response,
context)

        return MCPResponse(**modified_error)
```

10.2 React/TypeScriptによるMCPクライアント実装

MCPクライアントのReact/TypeScript実装を解説します。

10.2.1 クライアントライブラリの基本構造

```typescript
// src/lib/mcp-client.ts
import axios, { AxiosInstance, AxiosRequestConfig, AxiosResponse } from 'axios';
import WebSocket from 'isomorphic-ws';

export interface MCPRequestOptions {
  timeout?: number;
  headers?: Record<string, string>;
  signal?: AbortSignal;
}

export interface MCPRequest {
  version: string;
  requestType: string;
  resource: string;
  params?: Record<string, any>;
  context?: {
    sessionId?: string;
    [key: string]: any;
  };
}

export interface MCPResponse {
  version: string;
  status: number;
  requestId: string;
  data?: any;
  error?: {
    code: string;
    message: string;
    details?: any;
  };
  context?: {
    sessionId?: string;
    [key: string]: any;
```

```
  };
}

export interface MCPStreamOptions {
  onChunk?: (chunk: any) => void;
  onComplete?: () => void;
  onError?: (error: any) => void;
}

export interface MCPAuthOptions {
  type: 'bearer' | 'api-key' | 'basic';
  token?: string;
  apiKey?: string;
  username?: string;
  password?: string;
}

export class MCPClient {
  private baseUrl: string;
  private httpClient: AxiosInstance;
  private sessionId: string | null = null;
  private websocket: WebSocket | null = null;
  private wsListeners: Map<string, Set<(data: any) => void>> = new Map();

  constructor(baseUrl: string, authOptions?: MCPAuthOptions) {
    this.baseUrl = baseUrl.endsWith('/') ? baseUrl.slice(0, -1) : baseUrl;

    // Axiosクライアントの初期化
    const axiosConfig: AxiosRequestConfig = {
      baseURL: this.baseUrl,
      timeout: 30000, // デフォルトタイムアウト: 30秒
      headers: {
        'Content-Type': 'application/json'
      }
    };

    // 認証設定
    if (authOptions) {
      switch (authOptions.type) {
        case 'bearer':
          if (authOptions.token) {
            axiosConfig.headers = {
              ...axiosConfig.headers,
              'Authorization': `Bearer ${authOptions.token}`
            };
          }
          break;

        case 'api-key':
          if (authOptions.apiKey) {
            axiosConfig.headers = {
              ...axiosConfig.headers,
              'X-API-Key': authOptions.apiKey
            };
          }
          break;
```

```
        case 'basic':
          if (authOptions.username && authOptions.password) {
            const basicAuth = btoa(`${authOptions.username}:${authOptions.password}`);
            axiosConfig.headers = {
              ...axiosConfig.headers,
              'Authorization': `Basic ${basicAuth}`
            };
          }
          break;
      }
    }

    this.httpClient = axios.create(axiosConfig);
  }

  /**
   * MCPリクエストを送信
   */
  public async query(request: Partial<MCPRequest>, options?: MCPRequestOptions):
Promise<MCPResponse> {
    // リクエストの準備
    const mcpRequest: MCPRequest = {
      version: request.version || '1.0',
      requestType: request.requestType || 'query',
      resource: request.resource || '',
      params: request.params || {},
      context: {
        ...(request.context || {}),
        sessionId: this.sessionId || undefined
      }
    };

    try {
      // リクエスト送信
      const response: AxiosResponse<MCPResponse> = await this.httpClient.post(
        '/mcp',
        mcpRequest,
        {
          timeout: options?.timeout,
          headers: options?.headers,
          signal: options?.signal
        }
      );

      const mcpResponse = response.data;

      // セッションIDの保存
      if (mcpResponse.context?.sessionId) {
        this.sessionId = mcpResponse.context.sessionId;
      }

      return mcpResponse;
    } catch (error) {
      if (axios.isAxiosError(error)) {
        // Axiosエラーの処理
        if (error.response) {
          // サーバーからのエラーレスポンス
```

```
          return error.response.data as MCPResponse;
        } else {
          // ネットワークエラーなど
          return {
            version: mcpRequest.version,
            status: 0,
            requestId: '',
            error: {
              code: 'NETWORK_ERROR',
              message: error.message
            }
          };
        }
      } else {
        // その他のエラー
        return {
          version: mcpRequest.version,
          status: 0,
          requestId: '',
          error: {
            code: 'UNKNOWN_ERROR',
            message: String(error)
          }
        };
      }
    }
  }

  /**
   * WebSocket接続を確立
   */
  public async connect(): Promise<boolean> {
    return new Promise((resolve, reject) => {
      if (this.websocket) {
        // 既に接続済み
        resolve(true);
        return;
      }

      // WebSocket URLの構築
      const wsUrl = this.baseUrl.replace(/^http/, 'ws') + '/mcp/stream';

      // WebSocketの初期化
      this.websocket = new WebSocket(wsUrl);

      // 認証トークンの設定（必要に応じて）
      if (this.sessionId) {
        this.websocket.onopen = () => {
          this.websocket?.send(JSON.stringify({
            type: 'auth',
            sessionId: this.sessionId
          }));
        };
      }

      // イベントハンドラの設定
      this.websocket.onmessage = (event) => {
```

```
        try {
          const data = JSON.parse(event.data as string);
          const messageType = data.type || 'unknown';

          // リスナー通知
          if (this.wsListeners.has(messageType)) {
            const listeners = this.wsListeners.get(messageType)!;
            for (const listener of listeners) {
              listener(data);
            }
          }

          // 接続確立応答
          if (messageType === 'connection_established') {
            resolve(true);
          }
        } catch (error) {
          console.error('Error parsing WebSocket message:', error);
        }
      };

      this.websocket.onerror = (error) => {
        console.error('WebSocket error:', error);
        reject(error);
      };

      this.websocket.onclose = () => {
        this.websocket = null;
        console.log('WebSocket connection closed');
      };
    });
  }

  /**
   * WebSocketストリーミングリクエスト
   */
  public async stream(request: Partial<MCPRequest>, options?: MCPStreamOptions):
Promise<void> {
    // WebSocket接続確立
    if (!this.websocket) {
      await this.connect();
    }

    if (!this.websocket) {
      throw new Error('Failed to establish WebSocket connection');
    }

    // メッセージリスナーの設定
    const onChunkListener = (data: any) => {
      if (data.type === 'chunk') {
        options?.onChunk?.(data);
      } else if (data.type === 'complete') {
        options?.onComplete?.();

        // リスナー削除
        this.removeListener('chunk', onChunkListener);
        this.removeListener('complete', onChunkListener);
```

```
        this.removeListener('error', onErrorListener);
      }
    };

    const onErrorListener = (data: any) => {
      if (data.type === 'error') {
        options?.onError?.(data);

        // リスナー削除
        this.removeListener('chunk', onChunkListener);
        this.removeListener('complete', onChunkListener);
        this.removeListener('error', onErrorListener);
      }
    };

    // リスナー登録
    this.addListener('chunk', onChunkListener);
    this.addListener('complete', onChunkListener);
    this.addListener('error', onErrorListener);

    // リクエストの準備
    const mcpRequest: any = {
      version: request.version || '1.0',
      requestType: request.requestType || 'query',
      resource: request.resource || '',
      params: request.params || {},
      context: {
        ...(request.context || {}),
        sessionId: this.sessionId || undefined
      }
    };

    // リクエスト送信
    this.websocket.send(JSON.stringify(mcpRequest));
  }

  /**
   * WebSocketイベントリスナーを追加
   */
  private addListener(event: string, callback: (data: any) => void): void {
    if (!this.wsListeners.has(event)) {
      this.wsListeners.set(event, new Set());
    }

    this.wsListeners.get(event)!.add(callback);
  }

  /**
   * WebSocketイベントリスナーを削除
   */
  private removeListener(event: string, callback: (data: any) => void): void {
    if (this.wsListeners.has(event)) {
      this.wsListeners.get(event)!.delete(callback);
    }
  }

  /**
```

```
   * WebSocket接続を閉じる
   */
  public disconnect(): void {
    if (this.websocket) {
      this.websocket.close();
      this.websocket = null;
    }
  }

  /**
   * セッションIDを設定
   */
  public setSessionId(sessionId: string): void {
    this.sessionId = sessionId;
  }

  /**
   * セッションIDを取得
   */
  public getSessionId(): string | null {
    return this.sessionId;
  }
}
```

10.2.2 Reactフックの実装

```
// src/hooks/useMCP.ts
import { useState, useEffect, useCallback, useRef } from 'react';
import { MCPClient, MCPRequest, MCPResponse, MCPAuthOptions } from '../lib/mcp-client';

interface UseMCPOptions {
  baseUrl: string;
  auth?: MCPAuthOptions;
  autoConnect?: boolean;
}

interface UseMCPResult {
  client: MCPClient;
  isConnected: boolean;
  isLoading: boolean;
  error: Error | null;
  sessionId: string | null;
  query: (request: Partial<MCPRequest>) => Promise<MCPResponse>;
  stream: (request: Partial<MCPRequest>, options: {
    onChunk: (chunk: any) => void;
    onComplete?: () => void;
    onError?: (error: any) => void;
  }) => Promise<void>;
  connect: () => Promise<boolean>;
  disconnect: () => void;
}

export function useMCP(options: UseMCPOptions): UseMCPResult {
  const { baseUrl, auth, autoConnect = false } = options;
```

```
// クライアントをrefに保持（再レンダリングで再作成されないように）
const clientRef = useRef<MCPClient | null>(null);

// 状態
const [isConnected, setIsConnected] = useState(false);
const [isLoading, setIsLoading] = useState(false);
const [error, setError] = useState<Error | null>(null);
const [sessionId, setSessionId] = useState<string | null>(null);

// クライアント初期化
useEffect(() => {
  if (!clientRef.current) {
    clientRef.current = new MCPClient(baseUrl, auth);
  }

  // セッションID監視
  const checkSessionId = () => {
    const id = clientRef.current?.getSessionId();
    if (id !== sessionId) {
      setSessionId(id);
    }
  };

  const interval = setInterval(checkSessionId, 1000);

  return () => {
    clearInterval(interval);
    // コンポーネントアンマウント時に接続を閉じる
    clientRef.current?.disconnect();
  };
}, [baseUrl, auth, sessionId]);

// 自動接続
useEffect(() => {
  if (autoConnect && clientRef.current && !isConnected) {
    connect().catch(error => {
      console.error('Failed to auto-connect:', error);
    });
  }
}, [autoConnect, isConnected]);

// WebSocket接続
const connect = useCallback(async (): Promise<boolean> => {
  if (!clientRef.current) {
    throw new Error('MCP client not initialized');
  }

  try {
    setIsLoading(true);
    setError(null);

    const result = await clientRef.current.connect();
    setIsConnected(result);
    return result;
  } catch (err) {
    const error = err instanceof Error ? err : new Error(String(err));
    setError(error);
```

```
      throw error;
    } finally {
      setIsLoading(false);
    }
  }, []);

  // 接続を閉じる
  const disconnect = useCallback(() => {
    if (clientRef.current) {
      clientRef.current.disconnect();
      setIsConnected(false);
    }
  }, []);

  // クエリ実行
  const query = useCallback(async (request: Partial<MCPRequest>): Promise<MCPResponse>
=> {
    if (!clientRef.current) {
      throw new Error('MCP client not initialized');
    }

    try {
      setIsLoading(true);
      setError(null);

      const response = await clientRef.current.query(request);

      // エラーチェック
      if (response.error) {
        const error = new Error(response.error.message);
        (error as any).code = response.error.code;
        (error as any).details = response.error.details;
        setError(error);
      }

      return response;
    } catch (err) {
      const error = err instanceof Error ? err : new Error(String(err));
      setError(error);
      throw error;
    } finally {
      setIsLoading(false);
    }
  }, []);

  // ストリーミングリクエスト
  const stream = useCallback(async (
    request: Partial<MCPRequest>,
    options: {
      onChunk: (chunk: any) => void;
      onComplete?: () => void;
      onError?: (error: any) => void;
    }
  ): Promise<void> => {
    if (!clientRef.current) {
      throw new Error('MCP client not initialized');
    }
```

```
    // 接続されていない場合は接続
    if (!isConnected) {
      await connect();
    }

    try {
      setIsLoading(true);
      setError(null);

      await clientRef.current.stream(request, {
        onChunk: options.onChunk,
        onComplete: () => {
          setIsLoading(false);
          options.onComplete?.();
        },
        onError: (err) => {
          const error = new Error(err.message);
          (error as any).code = err.code;
          (error as any).details = err.details;

          setError(error);
          setIsLoading(false);
          options.onError?.(err);
        }
      });
    } catch (err) {
      const error = err instanceof Error ? err : new Error(String(err));
      setError(error);
      setIsLoading(false);
      throw error;
    }
  }, [isConnected, connect]);

  return {
    client: clientRef.current!,
    isConnected,
    isLoading,
    error,
    sessionId,
    query,
    stream,
    connect,
    disconnect
  };
}

// クエリ用フック
interface UseQueryOptions {
  request: Partial<MCPRequest>;
  enabled?: boolean;
  refetchInterval?: number;
  onSuccess?: (data: any) => void;
  onError?: (error: Error) => void;
}

interface UseQueryResult {
```

```
  data: any;
  error: Error | null;
  isLoading: boolean;
  isError: boolean;
  refetch: () => Promise<void>;
}

export function useMCPQuery(client: MCPClient, options: UseQueryOptions):
UseQueryResult {
  const { request, enabled = true, refetchInterval, onSuccess, onError } = options;

  const [data, setData] = useState<any>(null);
  const [error, setError] = useState<Error | null>(null);
  const [isLoading, setIsLoading] = useState(false);

  const fetchData = useCallback(async () => {
    if (!enabled) return;

    try {
      setIsLoading(true);

      const response = await client.query(request);

      if (response.error) {
        const err = new Error(response.error.message);
        (err as any).code = response.error.code;
        (err as any).details = response.error.details;

        setError(err);
        onError?.(err);
      } else {
        setData(response.data);
        setError(null);
        onSuccess?.(response.data);
      }
    } catch (err) {
      const error = err instanceof Error ? err : new Error(String(err));
      setError(error);
      onError?.(error);
    } finally {
      setIsLoading(false);
    }
  }, [client, request, enabled, onSuccess, onError]);

  // 初回フェッチと再フェッチ
  useEffect(() => {
    fetchData();

    // 定期的な再フェッチ
    let interval: NodeJS.Timeout | undefined;

    if (refetchInterval && enabled) {
      interval = setInterval(fetchData, refetchInterval);
    }

    return () => {
      if (interval) {
```

```
        clearInterval(interval);
      }
    };
  }, [fetchData, refetchInterval, enabled]);

  return {
    data,
    error,
    isLoading,
    isError: !!error,
    refetch: fetchData
  };
}
```

10.2.3 Reactコンポーネント実装例

```
// src/components/MCPProvider.tsx
import React, { createContext, useContext, ReactNode } from 'react';
import { MCPClient, MCPAuthOptions } from '../lib/mcp-client';
import { useMCP } from '../hooks/useMCP';

// MCPコンテキスト型定義
interface MCPContextType {
  client: MCPClient;
  isConnected: boolean;
  isLoading: boolean;
  error: Error | null;
  sessionId: string | null;
  connect: () => Promise<boolean>;
  disconnect: () => void;
}

// コンテキスト作成
const MCPContext = createContext<MCPContextType | null>(null);

// プロバイダープロパティ
interface MCPProviderProps {
  baseUrl: string;
  auth?: MCPAuthOptions;
  autoConnect?: boolean;
  children: ReactNode;
}

// MCPプロバイダーコンポーネント
export function MCPProvider({ baseUrl, auth, autoConnect = false, children }:
MCPProviderProps) {
  const { client, isConnected, isLoading, error, sessionId, connect, disconnect } =
useMCP({
    baseUrl,
    auth,
    autoConnect
  });

  return (
    <MCPContext.Provider value={{
```

```
      client,
      isConnected,
      isLoading,
      error,
      sessionId,
      connect,
      disconnect
    }}>
      {children}
    </MCPContext.Provider>
  );
}

// MCPコンテキストフック
export function useMCPContext() {
  const context = useContext(MCPContext);

  if (!context) {
    throw new Error('useMCPContext must be used within a MCPProvider');
  }

  return context;
}
```

```
// src/components/RepoExplorer.tsx
import React, { useState, useEffect } from 'react';
import { useMCPContext } from './MCPProvider';
import { useMCPQuery } from '../hooks/useMCP';

interface RepoFile {
  name: string;
  path: string;
  type: 'file' | 'directory';
  size?: number;
}

interface RepoExplorerProps {
  repository: string;
  initialPath?: string;
}

export function RepoExplorer({ repository, initialPath = '' }: RepoExplorerProps) {
  const { client } = useMCPContext();
  const [currentPath, setCurrentPath] = useState(initialPath);

  // GitHubリポジトリのファイル一覧を取得
  const { data, error, isLoading, refetch } = useMCPQuery(client, {
    request: {
      requestType: 'query',
      resource: `github://${repository}/${currentPath}`,
      params: {
        includeMetadata: true
      }
    },
    enabled: true
  });
```

```
// ファイル一覧
const files: RepoFile[] = data?.files || [];

// ディレクトリ変更処理
const navigateToDirectory = (dirPath: string) => {
  setCurrentPath(dirPath);
};

// 親ディレクトリへ移動
const navigateUp = () => {
  const parts = currentPath.split('/');
  parts.pop();
  setCurrentPath(parts.join('/'));
};

// パス表示
const breadcrumbs = [
  { name: 'Root', path: '' },
  ...currentPath.split('/').filter(Boolean).map((part, index, array) => {
    const path = array.slice(0, index + 1).join('/');
    return { name: part, path };
  })
];

// ファイルの種類によるアイコン
const getFileIcon = (file: RepoFile) => {
  if (file.type === 'directory') {
    return '📁';
  }

  const extension = file.name.split('.').pop()?.toLowerCase();

  switch (extension) {
    case 'js':
    case 'jsx':
    case 'ts':
    case 'tsx':
      return '📄 JS';
    case 'css':
    case 'scss':
    case 'sass':
      return '📄 CSS';
    case 'html':
      return '📄 HTML';
    case 'md':
      return '📄 MD';
    case 'json':
      return '📄 JSON';
    case 'jpg':
    case 'jpeg':
    case 'png':
    case 'gif':
    case 'svg':
      return '🖼️';
    default:
      return '📄';
```

```
    }
  };

  if (isLoading && !data) {
    return <div>Loading...</div>;
  }

  if (error) {
    return (
      <div className="error-container">
        <h3>Error loading repository</h3>
        <p>{error.message}</p>
        <button onClick={() => refetch()}>Retry</button>
      </div>
    );
  }

  return (
    <div className="repo-explorer">
      <div className="breadcrumbs">
        {breadcrumbs.map((crumb, index) => (
          <React.Fragment key={crumb.path}>
            {index > 0 && <span className="separator">/</span>}
            <button
              className="breadcrumb-link"
              onClick={() => navigateToDirectory(crumb.path)}
            >
              {crumb.name}
            </button>
          </React.Fragment>
        ))}
      </div>

      <div className="file-list">
        {currentPath && (
          <div className="file-item" onClick={navigateUp}>
            <span className="file-icon">📁</span>
            <span className="file-name">..</span>
          </div>
        )}

        {files.map((file) => (
          <div
            key={file.path}
            className="file-item"
            onClick={() => file.type === 'directory' ? navigateToDirectory(file.path) :
null}
          >
            <span className="file-icon">{getFileIcon(file)}</span>
            <span className="file-name">{file.name}</span>
            {file.size && <span className="file-size">{formatSize(file.size)}</span>}
          </div>
        ))}

        {files.length === 0 && (
          <div className="empty-message">
            This directory is empty
```

```
          </div>
        )}
      </div>
    </div>
  );
}

// ファイルサイズのフォーマット
function formatSize(bytes: number): string {
  if (bytes < 1024) {
    return `${bytes} B`;
  } else if (bytes < 1024 * 1024) {
    return `${(bytes / 1024).toFixed(1)} KB`;
  } else if (bytes < 1024 * 1024 * 1024) {
    return `${(bytes / (1024 * 1024)).toFixed(1)} MB`;
  } else {
    return `${(bytes / (1024 * 1024 * 1024)).toFixed(1)} GB`;
  }
}
```

```
// src/components/MCPChat.tsx
import React, { useState, useEffect, useRef } from 'react';
import { useMCPContext } from './MCPProvider';

interface Message {
  id: string;
  role: 'user' | 'assistant' | 'system';
  content: string;
  timestamp: string;
}

interface MCPChatProps {
  initialMessages?: Message[];
  assistantName?: string;
}

export function MCPChat({ initialMessages = [], assistantName = 'MCP Assistant' }:
MCPChatProps) {
  const { client, isConnected, connect } = useMCPContext();
  const [messages, setMessages] = useState<Message[]>(initialMessages);
  const [input, setInput] = useState('');
  const [isStreaming, setIsStreaming] = useState(false);
  const messagesEndRef = useRef<HTMLDivElement>(null);

  // メッセージが追加されたら自動スクロール
  useEffect(() => {
    messagesEndRef.current?.scrollIntoView({ behavior: 'smooth' });
  }, [messages]);

  // WebSocketが未接続なら接続
  useEffect(() => {
    if (!isConnected) {
      connect().catch(error => {
        console.error('Failed to connect to MCP server:', error);
        addSystemMessage('Failed to connect to MCP server. Some features may be
limited.');
```

```
      });
    }
  }, [isConnected, connect]);

  // システムメッセージの追加
  const addSystemMessage = (content: string) => {
    setMessages(prev => [
      ...prev,
      {
        id: `system-${Date.now()}`,
        role: 'system',
        content,
        timestamp: new Date().toISOString()
      }
    ]);
  };

  // メッセージ送信処理
  const sendMessage = async () => {
    if (!input.trim()) return;

    // ユーザーメッセージを追加
    const userMessage: Message = {
      id: `user-${Date.now()}`,
      role: 'user',
      content: input,
      timestamp: new Date().toISOString()
    };

    setMessages(prev => [...prev, userMessage]);
    setInput('');

    // アシスタントメッセージの準備
    const assistantMessage: Message = {
      id: `assistant-${Date.now()}`,
      role: 'assistant',
      content: '',
      timestamp: new Date().toISOString()
    };

    setMessages(prev => [...prev, assistantMessage]);

    try {
      // ストリーミングAPIを使用
      setIsStreaming(true);

      // 会話履歴の作成
      const conversationHistory = messages.map(msg => ({
        role: msg.role === 'system' ? 'assistant' : msg.role,
        content: msg.content
      }));

      await client.stream(
        {
          requestType: 'chat',
          resource: 'mcp://assistant',
          params: {
```

```
            messages: [
              ...conversationHistory,
              { role: 'user', content: input }
            ],
            stream: true
          }
        },
        {
          onChunk: (chunk) => {
            // ストリーミングレスポンスの処理
            setMessages(prev => {
              const updatedMessages = [...prev];
              const assistantIdx = updatedMessages.findIndex(m => m.id ===
assistantMessage.id);

              if (assistantIdx !== -1) {
                updatedMessages[assistantIdx] = {
                  ...updatedMessages[assistantIdx],
                  content: updatedMessages[assistantIdx].content + (chunk.data || '')
                };
              }

              return updatedMessages;
            });
          },
          onComplete: () => {
            setIsStreaming(false);
          },
          onError: (error) => {
            setIsStreaming(false);

            addSystemMessage(`Error: ${error.message || 'Failed to get response'}`);

            console.error('Streaming error:', error);
          }
        }
      );
    } catch (error) {
      setIsStreaming(false);

      console.error('Failed to send message:', error);

      addSystemMessage('Failed to get response. Please try again.');
    }
  };

  // Enterキーでメッセージ送信
  const handleKeyDown = (event: React.KeyboardEvent) => {
    if (event.key === 'Enter' && !event.shiftKey) {
      event.preventDefault();
      sendMessage();
    }
  };

  return (
    <div className="mcp-chat">
      <div className="chat-header">
```

```
        <h2>{assistantName}</h2>
        <div className="connection-status">
          <span className={`status-indicator ${isConnected ? 'connected' :
'disconnected'}`} />
          {isConnected ? 'Connected' : 'Disconnected'}
        </div>
      </div>

      <div className="chat-messages">
        {messages.map((message) => (
          <div key={message.id} className={`message ${message.role}`}>
            <div className="message-content">{message.content}</div>
            <div className="message-timestamp">
              {new Date(message.timestamp).toLocaleTimeString()}
            </div>
          </div>
        ))}
        <div ref={messagesEndRef} />
      </div>

      <div className="chat-input">
        <textarea
          value={input}
          onChange={(e) => setInput(e.target.value)}
          onKeyDown={handleKeyDown}
          placeholder="Type your message..."
          disabled={isStreaming}
        />
        <button
          onClick={sendMessage}
          disabled={isStreaming || !input.trim()}
        >
          {isStreaming ? 'Sending...' : 'Send'}
        </button>
      </div>
    </div>
  );
}
```

10.2.4 アプリケーション統合例

```
// src/App.tsx
import React from 'react';
import { MCPProvider } from './components/MCPProvider';
import { MCPChat } from './components/MCPChat';
import { RepoExplorer } from './components/RepoExplorer';
import './App.css';

// アプリケーション設定
const MCP_SERVER_URL = process.env.REACT_APP_MCP_SERVER_URL || 'http://localhost:8000';
const MCP_AUTH_TOKEN = process.env.REACT_APP_MCP_AUTH_TOKEN;

function App() {
  return (
    <MCPProvider
```

```
      baseUrl={MCP_SERVER_URL}
      auth={MCP_AUTH_TOKEN ? { type: 'bearer', token: MCP_AUTH_TOKEN } : undefined}
      autoConnect={true}
    >
      <div className="app-container">
        <header className="app-header">
          <h1>MCP Client Demo</h1>
        </header>

        <main className="app-content">
          <section className="repo-explorer-section">
            <h2>Repository Explorer</h2>
            <RepoExplorer repository="example-org/example-repo" />
          </section>

          <section className="chat-section">
            <MCPChat assistantName="MCP Knowledge Assistant" />
          </section>
        </main>

        <footer className="app-footer">
          <p>MCP Protocol Demo Client &copy; 2025</p>
        </footer>
      </div>
    </MCPProvider>
  );
}

export default App;
```

```
/* src/App.css */
.app-container {
  display: flex;
  flex-direction: column;
  min-height: 100vh;
  font-family: system-ui, -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto,
Oxygen, Ubuntu, Cantarell, 'Open Sans', 'Helvetica Neue', sans-serif;
}

.app-header {
  background-color: #2c3e50;
  color: white;
  padding: 1rem;
  text-align: center;
}

.app-content {
  display: flex;
  flex: 1;
  padding: 1rem;
}

.repo-explorer-section {
  flex: 1;
  margin-right: 1rem;
  border: 1px solid #ddd;
```

```
  border-radius: 4px;
  overflow: hidden;
}

.chat-section {
  flex: 1;
  border: 1px solid #ddd;
  border-radius: 4px;
  overflow: hidden;
}

.app-footer {
  background-color: #ecf0f1;
  padding: 1rem;
  text-align: center;
  font-size: 0.8rem;
  color: #7f8c8d;
}

/* RepoExplorer Component */
.repo-explorer {
  display: flex;
  flex-direction: column;
  height: 100%;
}

.breadcrumbs {
  padding: 0.5rem;
  background-color: #f8f9fa;
  border-bottom: 1px solid #e9ecef;
  display: flex;
  flex-wrap: wrap;
  align-items: center;
}

.breadcrumb-link {
  background: none;
  border: none;
  color: #0366d6;
  cursor: pointer;
  padding: 0;
  font-size: 0.9rem;
}

.breadcrumb-link:hover {
  text-decoration: underline;
}

.separator {
  margin: 0 0.5rem;
  color: #6c757d;
}

.file-list {
  flex: 1;
  overflow-y: auto;
  padding: 0.5rem;
```

```css
}

.file-item {
  display: flex;
  align-items: center;
  padding: 0.5rem;
  border-radius: 4px;
  cursor: pointer;
}

.file-item:hover {
  background-color: #f8f9fa;
}

.file-icon {
  margin-right: 0.5rem;
}

.file-name {
  flex: 1;
}

.file-size {
  color: #6c757d;
  font-size: 0.8rem;
}

.empty-message {
  text-align: center;
  padding: 2rem;
  color: #6c757d;
}

.error-container {
  padding: 1rem;
  color: #721c24;
  background-color: #f8d7da;
  border: 1px solid #f5c6cb;
  border-radius: 4px;
  margin: 1rem;
}

/* MCPChat Component */
.mcp-chat {
  display: flex;
  flex-direction: column;
  height: 100%;
}

.chat-header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 0.5rem 1rem;
  background-color: #f8f9fa;
  border-bottom: 1px solid #e9ecef;
}
```

```css
.chat-header h2 {
  margin: 0;
  font-size: 1.2rem;
}

.connection-status {
  display: flex;
  align-items: center;
  font-size: 0.8rem;
}

.status-indicator {
  width: 8px;
  height: 8px;
  border-radius: 50%;
  margin-right: 0.5rem;
}

.status-indicator.connected {
  background-color: #28a745;
}

.status-indicator.disconnected {
  background-color: #dc3545;
}

.chat-messages {
  flex: 1;
  overflow-y: auto;
  padding: 1rem;
  display: flex;
  flex-direction: column;
}

.message {
  max-width: 80%;
  margin-bottom: 1rem;
  padding: 0.75rem;
  border-radius: 1rem;
  position: relative;
}

.message.user {
  align-self: flex-end;
  background-color: #0084ff;
  color: white;
  border-bottom-right-radius: 0.25rem;
}

.message.assistant {
  align-self: flex-start;
  background-color: #f1f0f0;
  color: black;
  border-bottom-left-radius: 0.25rem;
}
```

```css
.message.system {
  align-self: center;
  background-color: #ffc107;
  color: #212529;
  font-size: 0.9rem;
  padding: 0.5rem 0.75rem;
  border-radius: 0.5rem;
}

.message-timestamp {
  font-size: 0.7rem;
  margin-top: 0.25rem;
  opacity: 0.7;
  text-align: right;
}

.chat-input {
  display: flex;
  padding: 0.5rem;
  border-top: 1px solid #e9ecef;
}

.chat-input textarea {
  flex: 1;
  resize: none;
  padding: 0.5rem;
  border: 1px solid #ced4da;
  border-radius: 4px;
  min-height: 2.5rem;
  font-family: inherit;
}

.chat-input button {
  margin-left: 0.5rem;
  padding: 0 1rem;
  background-color: #0366d6;
  color: white;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}

.chat-input button:disabled {
  background-color: #6c757d;
  cursor: not-allowed;
}
```

> **ベテランの知恵袋**: *「MCPクライアントライブラリを開発する際は、エラー処理とリカバリメカニズムに特に注力することをお勧めします。ネットワークの問題やサーバーの一時的な障害に対して、適切なリトライとフォールバック戦略を実装することで、ユーザー体験を大幅に向上できます。また、デバッグのしやすさも重要です。詳細なログ出力と明確なエラーメッセージにより、問題解決が格段に容易になります。」* - フロントエンドアーキテクト

10.3 C#実装のMCPサーバー

.NET 6を使ったMCPサーバーの実装例を解説します。

10.3.1 基本プロジェクト構造

```
// Program.cs
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using MCPServer.Services;
using MCPServer.Connectors;
using MCPServer.Middleware;
using System.Text.Json.Serialization;

var builder = WebApplication.CreateBuilder(args);

// 設定の追加
builder.Configuration.AddJsonFile("appsettings.json", optional: false, reloadOnChange:
true);
builder.Configuration.AddJsonFile($"appsettings.
{builder.Environment.EnvironmentName}.json", optional: true);
builder.Configuration.AddEnvironmentVariables();

// Controllerの追加
builder.Services.AddControllers()
    .AddJsonOptions(options =>
    {
        options.JsonSerializerOptions.DefaultIgnoreCondition =
JsonIgnoreCondition.WhenWritingNull;
        options.JsonSerializerOptions.PropertyNamingPolicy = null;
        options.JsonSerializerOptions.Converters.Add(new JsonStringEnumConverter());
    });

// CORS設定
builder.Services.AddCors(options =>
{
    options.AddPolicy("AllowAll", policy =>
    {
        policy.AllowAnyOrigin()
              .AllowAnyMethod()
              .AllowAnyHeader();
    });
});

// WebSocketの追加
builder.Services.AddWebSockets(options =>
{
    options.KeepAliveInterval = TimeSpan.FromMinutes(2);
});

// Swagger（OpenAPI）ドキュメント生成の追加
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

// MCP関連サービスの登録
```

```
builder.Services.AddSingleton<IConnectorManager, ConnectorManager>();
builder.Services.AddSingleton<IContextManager, ContextManager>();
builder.Services.AddSingleton<IConnectionManager, ConnectionManager>();
builder.Services.AddSingleton<IAuthService, JwtAuthService>();

// プラグインマネージャーの登録
builder.Services.AddSingleton<IPluginManager, PluginManager>();

// サンプルコネクタの登録
builder.Services.AddSingleton<IConnector, EchoConnector>();
builder.Services.AddSingleton<IConnector, FileConnector>();

// アプリケーションの構築
var app = builder.Build();

// 開発環境ではSwaggerを使用
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
    app.UseDeveloperExceptionPage();
}
else
{
    app.UseExceptionHandler("/error");
    app.UseHsts();
}

// HTTPSリダイレクト
app.UseHttpsRedirection();

// CORSの設定
app.UseCors("AllowAll");

// WebSocketの設定
app.UseWebSockets();

// カスタムミドルウェアの追加
app.UseMiddleware<RequestLoggingMiddleware>();
app.UseMiddleware<RateLimitingMiddleware>();

// 認証と認可
app.UseAuthentication();
app.UseAuthorization();

// WebSocketハンドラーのセットアップ
app.UseWebSocketHandler();

// Controllerルーティング
app.MapControllers();

// コネクタの初期化
var connectorManager = app.Services.GetRequiredService<IConnectorManager>();
await connectorManager.InitializeConnectorsAsync();

// プラグインの初期化
var pluginManager = app.Services.GetRequiredService<IPluginManager>();
```

```
await pluginManager.LoadPluginsAsync();

// アプリケーションの実行
app.Run();
```

10.3.2 MCPモデル定義

```
// Models/MCPModels.cs
using System.Text.Json.Serialization;

namespace MCPServer.Models
{
    /// <summary>
    /// MCPリクエストモデル
    /// </summary>
    public class MCPRequest
    {
        /// <summary>
        /// MCPプロトコルバージョン
        /// </summary>
        public string Version { get; set; } = "1.0";

        /// <summary>
        /// リクエストタイプ (query, update, create, execute等)
        /// </summary>
        public string RequestType { get; set; } = string.Empty;

        /// <summary>
        /// リソースURI
        /// </summary>
        public string Resource { get; set; } = string.Empty;

        /// <summary>
        /// リクエストパラメータ
        /// </summary>
        public Dictionary<string, object>? Params { get; set; }

        /// <summary>
        /// リクエストコンテキスト
        /// </summary>
        public RequestContext? Context { get; set; }
    }

    /// <summary>
    /// リクエストコンテキストモデル
    /// </summary>
    public class RequestContext
    {
        /// <summary>
        /// セッションID
        /// </summary>
        public string? SessionId { get; set; }

        /// <summary>
        /// メタデータ
```

```
        /// </summary>
        public Dictionary<string, object>? Metadata { get; set; }
    }

    /// <summary>
    /// MCPレスポンスモデル
    /// </summary>
    public class MCPResponse
    {
        /// <summary>
        /// MCPプロトコルバージョン
        /// </summary>
        public string Version { get; set; } = "1.0";

        /// <summary>
        /// ステータスコード
        /// </summary>
        public int Status { get; set; }

        /// <summary>
        /// リクエストID
        /// </summary>
        public string RequestId { get; set; } = Guid.NewGuid().ToString();

        /// <summary>
        /// レスポンスデータ
        /// </summary>
        public object? Data { get; set; }

        /// <summary>
        /// エラー情報
        /// </summary>
        public ErrorInfo? Error { get; set; }

        /// <summary>
        /// レスポンスコンテキスト
        /// </summary>
        public ResponseContext? Context { get; set; }
    }

    /// <summary>
    /// レスポンスコンテキストモデル
    /// </summary>
    public class ResponseContext
    {
        /// <summary>
        /// セッションID
        /// </summary>
        public string? SessionId { get; set; }

        /// <summary>
        /// 更新日時
        /// </summary>
        public DateTime Updated { get; set; } = DateTime.UtcNow;

        /// <summary>
        /// メタデータ
```

```
        /// </summary>
        public Dictionary<string, object>? Metadata { get; set; }
    }

    /// <summary>
    /// エラー情報モデル
    /// </summary>
    public class ErrorInfo
    {
        /// <summary>
        /// エラーコード
        /// </summary>
        public string Code { get; set; } = string.Empty;

        /// <summary>
        /// エラーメッセージ
        /// </summary>
        public string Message { get; set; } = string.Empty;

        /// <summary>
        /// 詳細情報
        /// </summary>
        public object? Details { get; set; }
    }

    /// <summary>
    /// WebSocketメッセージモデル
    /// </summary>
    public class WebSocketMessage
    {
        /// <summary>
        /// メッセージタイプ
        /// </summary>
        public string Type { get; set; } = string.Empty;

        /// <summary>
        /// メッセージデータ
        /// </summary>
        public object? Data { get; set; }

        /// <summary>
        /// タイムスタンプ
        /// </summary>
        public DateTime Timestamp { get; set; } = DateTime.UtcNow;
    }
}
```

10.3.3 コネクタインターフェースとサービス定義

```
// Services/IConnector.cs
using MCPServer.Models;

namespace MCPServer.Services
{
    /// <summary>
```

```
    /// MCPコネクタインターフェース
    /// </summary>
    public interface IConnector
    {
        /// <summary>
        /// コネクタID
        /// </summary>
        string Id { get; }

        /// <summary>
        /// コネクタ名
        /// </summary>
        string Name { get; }

        /// <summary>
        /// コネクタバージョン
        /// </summary>
        string Version { get; }

        /// <summary>
        /// サポートするリソースURIスキーム
        /// </summary>
        string[] SupportedSchemes { get; }

        /// <summary>
        /// 初期化処理
        /// </summary>
        Task InitializeAsync();

        /// <summary>
        /// リクエスト処理
        /// </summary>
        /// <param name="request">コネクタリクエスト</param>
        /// <returns>処理結果</returns>
        Task<ConnectorResponse> HandleRequestAsync(ConnectorRequest request);

        /// <summary>
        /// ストリーミング処理
        /// </summary>
        /// <param name="request">コネクタリクエスト</param>
        /// <returns>ストリーミング結果のイテレーター</returns>
        IAsyncEnumerable<object> HandleStreamingAsync(ConnectorRequest request);

        /// <summary>
        /// リソースURIの解析
        /// </summary>
        /// <param name="resourceUri">リソースURI</param>
        /// <returns>解析結果</returns>
        ResourceInfo ParseResourceUri(string resourceUri);

        /// <summary>
        /// 後処理・クリーンアップ
        /// </summary>
        Task DisposeAsync();
    }

    /// <summary>
```

```
    /// コネクタリクエストモデル
    /// </summary>
    public class ConnectorRequest
    {
        /// <summary>
        /// リクエストタイプ
        /// </summary>
        public string RequestType { get; set; } = string.Empty;

        /// <summary>
        /// リソース情報
        /// </summary>
        public ResourceInfo Resource { get; set; } = null!;

        /// <summary>
        /// リクエストパラメータ
        /// </summary>
        public Dictionary<string, object>? Params { get; set; }

        /// <summary>
        /// リクエストコンテキスト
        /// </summary>
        public ConnectorContext Context { get; set; } = null!;
    }

    /// <summary>
    /// コネクタレスポンスモデル
    /// </summary>
    public class ConnectorResponse
    {
        /// <summary>
        /// レスポンスデータ
        /// </summary>
        public object? Data { get; set; }

        /// <summary>
        /// エラー情報
        /// </summary>
        public ErrorInfo? Error { get; set; }

        /// <summary>
        /// コンテキスト更新情報
        /// </summary>
        public ConnectorContext? Context { get; set; }

        /// <summary>
        /// 成功レスポンスの作成
        /// </summary>
        /// <param name="data">レスポンスデータ</param>
        /// <param name="context">コンテキスト更新情報</param>
        /// <returns>コネクタレスポンス</returns>
        public static ConnectorResponse Success(object? data = null, ConnectorContext?
context = null)
        {
            return new ConnectorResponse
            {
                Data = data,
```

```
                Context = context
            };
        }

        /// <summary>
        /// エラーレスポンスの作成
        /// </summary>
        /// <param name="code">エラーコード</param>
        /// <param name="message">エラーメッセージ</param>
        /// <param name="details">詳細情報</param>
        /// <returns>コネクタレスポンス</returns>
        public static ConnectorResponse Error(string code, string message, object?
details = null)
        {
            return new ConnectorResponse
            {
                Error = new ErrorInfo
                {
                    Code = code,
                    Message = message,
                    Details = details
                }
            };
        }
    }

    /// <summary>
    /// コネクタコンテキストモデル
    /// </summary>
    public class ConnectorContext
    {
        /// <summary>
        /// セッションID
        /// </summary>
        public string SessionId { get; set; } = string.Empty;

        /// <summary>
        /// コンテキストデータ
        /// </summary>
        public Dictionary<string, object> Data { get; set; } = new Dictionary<string,
object>();

        /// <summary>
        /// ユーザー情報
        /// </summary>
        public UserInfo? User { get; set; }
    }

    /// <summary>
    /// ユーザー情報モデル
    /// </summary>
    public class UserInfo
    {
        /// <summary>
        /// ユーザーID
        /// </summary>
        public string Id { get; set; } = string.Empty;
```

```
        /// <summary>
        /// ユーザー名
        /// </summary>
        public string Username { get; set; } = string.Empty;

        /// <summary>
        /// ロール
        /// </summary>
        public List<string> Roles { get; set; } = new List<string>();
    }

    /// <summary>
    /// リソース情報モデル
    /// </summary>
    public class ResourceInfo
    {
        /// <summary>
        /// リソースURI
        /// </summary>
        public string Uri { get; set; } = string.Empty;

        /// <summary>
        /// スキーム
        /// </summary>
        public string Scheme { get; set; } = string.Empty;

        /// <summary>
        /// パス
        /// </summary>
        public string Path { get; set; } = string.Empty;

        /// <summary>
        /// クエリパラメータ
        /// </summary>
        public Dictionary<string, string> QueryParams { get; set; } = new
Dictionary<string, string>();

        /// <summary>
        /// 追加プロパティ
        /// </summary>
        public Dictionary<string, object> Properties { get; set; } = new
Dictionary<string, object>();
    }
}
```

```
// Services/IConnectorManager.cs
using MCPServer.Models;

namespace MCPServer.Services
{
    /// <summary>
    /// コネクタマネージャーインターフェース
    /// </summary>
    public interface IConnectorManager
    {
```

```
        /// <summary>
        /// コネクタの登録
        /// </summary>
        /// <param name="connector">コネクタインスタンス</param>
        void RegisterConnector(IConnector connector);

        /// <summary>
        /// コネクタIDによるコネクタの取得
        /// </summary>
        /// <param name="connectorId">コネクタID</param>
        /// <returns>コネクタインスタンス</returns>
        IConnector? GetConnector(string connectorId);

        /// <summary>
        /// リソースURIからコネクタを特定
        /// </summary>
        /// <param name="resourceUri">リソースURI</param>
        /// <returns>コネクタインスタンスとリソース情報のタプル</returns>
        (IConnector connector, ResourceInfo resourceInfo)?
GetConnectorForResource(string resourceUri);

        /// <summary>
        /// リクエスト処理
        /// </summary>
        /// <param name="request">MCPリクエスト</param>
        /// <param name="context">リクエストコンテキスト</param>
        /// <returns>処理結果</returns>
        Task<MCPResponse> ProcessRequestAsync(MCPRequest request, ConnectorContext
context);

        /// <summary>
        /// すべてのコネクタを初期化
        /// </summary>
        Task InitializeConnectorsAsync();
    }

    /// <summary>
    /// コネクタマネージャーの実装
    /// </summary>
    public class ConnectorManager : IConnectorManager
    {
        private readonly Dictionary<string, IConnector> _connectors = new();
        private readonly Dictionary<string, IConnector> _schemeConnectors = new();
        private readonly ILogger<ConnectorManager> _logger;

        public ConnectorManager(ILogger<ConnectorManager> logger,
IEnumerable<IConnector> connectors)
        {
            _logger = logger;

            // 提供されたコネクタを自動登録
            foreach (var connector in connectors)
            {
                RegisterConnector(connector);
            }
        }
```

```
        /// <inheritdoc />
        public void RegisterConnector(IConnector connector)
        {
            _connectors[connector.Id] = connector;

            // スキームとコネクタのマッピングを登録
            foreach (var scheme in connector.SupportedSchemes)
            {
                _schemeConnectors[scheme.ToLowerInvariant()] = connector;
            }

            _logger.LogInformation("Registered connector: {ConnectorId}
({ConnectorName} v{Version})",
                connector.Id, connector.Name, connector.Version);
        }

        /// <inheritdoc />
        public IConnector? GetConnector(string connectorId)
        {
            return _connectors.TryGetValue(connectorId, out var connector) ? connector
: null;
        }

        /// <inheritdoc />
        public (IConnector connector, ResourceInfo resourceInfo)?
GetConnectorForResource(string resourceUri)
        {
            if (string.IsNullOrEmpty(resourceUri))
            {
                return null;
            }

            // スキームを抽出
            int schemeEndIndex = resourceUri.IndexOf("://");
            if (schemeEndIndex <= 0)
            {
                return null;
            }

            string scheme = resourceUri.Substring(0,
schemeEndIndex).ToLowerInvariant();

            if (_schemeConnectors.TryGetValue(scheme, out var connector))
            {
                var resourceInfo = connector.ParseResourceUri(resourceUri);
                return (connector, resourceInfo);
            }

            return null;
        }

        /// <inheritdoc />
        public async Task<MCPResponse> ProcessRequestAsync(MCPRequest request,
ConnectorContext context)
        {
            var connectorInfo = GetConnectorForResource(request.Resource);
            if (connectorInfo == null)
```

```
            {
                return new MCPResponse
                {
                    Version = request.Version,
                    Status = 404,
                    RequestId = Guid.NewGuid().ToString(),
                    Error = new ErrorInfo
                    {
                        Code = "CONNECTOR_NOT_FOUND",
                        Message = $"No connector available for resource:
{request.Resource}"
                    }
                };
            }

            var (connector, resourceInfo) = connectorInfo.Value;

            try
            {
                var connectorRequest = new ConnectorRequest
                {
                    RequestType = request.RequestType,
                    Resource = resourceInfo,
                    Params = request.Params,
                    Context = context
                };

                var result = await connector.HandleRequestAsync(connectorRequest);

                return new MCPResponse
                {
                    Version = request.Version,
                    Status = result.Error != null ? 400 : 200,
                    RequestId = Guid.NewGuid().ToString(),
                    Data = result.Data,
                    Error = result.Error,
                    Context = result.Context != null ? new ResponseContext
                    {
                        SessionId = result.Context.SessionId,
                        Metadata = result.Context.Data
                    } : null
                };
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Error processing request with connector
{ConnectorId}", connector.Id);

                return new MCPResponse
                {
                    Version = request.Version,
                    Status = 500,
                    RequestId = Guid.NewGuid().ToString(),
                    Error = new ErrorInfo
                    {
                        Code = "INTERNAL_ERROR",
                        Message = "An internal error occurred while processing the
```

```
request",
                        Details = ex.Message
                    }
                };
            }
        }

        /// <inheritdoc />
        public async Task InitializeConnectorsAsync()
        {
            foreach (var connector in _connectors.Values)
            {
                try
                {
                    await connector.InitializeAsync();
                    _logger.LogInformation("Initialized connector: {ConnectorId}",
connector.Id);
                }
                catch (Exception ex)
                {
                    _logger.LogError(ex, "Failed to initialize connector:
{ConnectorId}", connector.Id);
                }
            }
        }
    }
}
```

```
// Services/IContextManager.cs
namespace MCPServer.Services
{
    /// <summary>
    /// コンテキストマネージャーインターフェース
    /// </summary>
    public interface IContextManager
    {
        /// <summary>
        /// コンテキストの取得
        /// </summary>
        /// <param name="sessionId">セッションID</param>
        /// <returns>コンテキスト</returns>
        ConnectorContext? GetContext(string sessionId);

        /// <summary>
        /// コンテキストの作成
        /// </summary>
        /// <param name="initialData">初期データ</param>
        /// <returns>新しいコンテキスト</returns>
        ConnectorContext CreateContext(Dictionary<string, object>? initialData = null);

        /// <summary>
        /// コンテキストの更新
        /// </summary>
        /// <param name="sessionId">セッションID</param>
        /// <param name="updates">更新データ</param>
        /// <returns>更新されたコンテキスト</returns>
```

```
        ConnectorContext? UpdateContext(string sessionId, Dictionary<string, object>
updates);

        /// <summary>
        /// 古いコンテキストのクリーンアップ
        /// </summary>
        /// <param name="maxAgeHours">最大保持時間（時間）</param>
        /// <returns>クリーンアップされたコンテキスト数</returns>
        int CleanupOldContexts(int maxAgeHours = 24);
    }

    /// <summary>
    /// コンテキストマネージャーの実装
    /// </summary>
    public class ContextManager : IContextManager
    {
        private class ContextEntry
        {
            public ConnectorContext Context { get; set; } = null!;
            public DateTime Created { get; set; }
            public DateTime LastAccessed { get; set; }
        }

        private readonly Dictionary<string, ContextEntry> _contexts = new();
        private readonly SemaphoreSlim _lock = new(1, 1);
        private readonly ILogger<ContextManager> _logger;

        public ContextManager(ILogger<ContextManager> logger)
        {
            _logger = logger;

            // 定期的なクリーンアップタスクの開始
            StartCleanupTask();
        }

        /// <inheritdoc />
        public ConnectorContext? GetContext(string sessionId)
        {
            if (string.IsNullOrEmpty(sessionId))
            {
                return null;
            }

            _lock.Wait();
            try
            {
                if (_contexts.TryGetValue(sessionId, out var entry))
                {
                    // 最終アクセス時間を更新
                    entry.LastAccessed = DateTime.UtcNow;
                    return entry.Context;
                }

                return null;
            }
            finally
            {
```

```
                _lock.Release();
            }
        }

        /// <inheritdoc />
        public ConnectorContext CreateContext(Dictionary<string, object>? initialData =
null)
        {
            _lock.Wait();
            try
            {
                var sessionId = Guid.NewGuid().ToString();
                var context = new ConnectorContext
                {
                    SessionId = sessionId,
                    Data = initialData ?? new Dictionary<string, object>()
                };

                _contexts[sessionId] = new ContextEntry
                {
                    Context = context,
                    Created = DateTime.UtcNow,
                    LastAccessed = DateTime.UtcNow
                };

                return context;
            }
            finally
            {
                _lock.Release();
            }
        }

        /// <inheritdoc />
        public ConnectorContext? UpdateContext(string sessionId, Dictionary<string,
object> updates)
        {
            if (string.IsNullOrEmpty(sessionId))
            {
                return null;
            }

            _lock.Wait();
            try
            {
                if (!_contexts.TryGetValue(sessionId, out var entry))
                {
                    return null;
                }

                // コンテキストデータを更新
                foreach (var kvp in updates)
                {
                    entry.Context.Data[kvp.Key] = kvp.Value;
                }

                // 最終アクセス時間を更新
```

```
                entry.LastAccessed = DateTime.UtcNow;

                return entry.Context;
            }
            finally
            {
                _lock.Release();
            }
        }

        /// <inheritdoc />
        public int CleanupOldContexts(int maxAgeHours = 24)
        {
            var cutoff = DateTime.UtcNow.AddHours(-maxAgeHours);
            var keysToRemove = new List<string>();

            _lock.Wait();
            try
            {
                foreach (var kvp in _contexts)
                {
                    if (kvp.Value.LastAccessed < cutoff)
                    {
                        keysToRemove.Add(kvp.Key);
                    }
                }

                foreach (var key in keysToRemove)
                {
                    _contexts.Remove(key);
                }

                return keysToRemove.Count;
            }
            finally
            {
                _lock.Release();
            }
        }

        private void StartCleanupTask()
        {
            Task.Run(async () =>
            {
                while (true)
                {
                    try
                    {
                        // 1時間ごとにクリーンアップを実行
                        await Task.Delay(TimeSpan.FromHours(1));

                        int cleaned = CleanupOldContexts();
                        if (cleaned > 0)
                        {
                            _logger.LogInformation("Cleaned up {Count} old contexts",
cleaned);

                        }
```

```
                    }
                    catch (Exception ex)
                    {
                        _logger.LogError(ex, "Error in context cleanup task");
                    }
                }
            });
        }
    }
}
```

10.3.4 ウェブソケット処理の実装

```
// Services/IConnectionManager.cs
using System.Collections.Concurrent;
using System.Net.WebSockets;
using System.Text;
using System.Text.Json;
using MCPServer.Models;

namespace MCPServer.Services
{
    /// <summary>
    /// WebSocket接続マネージャーインターフェース
    /// </summary>
    public interface IConnectionManager
    {
        /// <summary>
        /// クライアント接続の追加
        /// </summary>
        /// <param name="clientId">クライアントID</param>
        /// <param name="webSocket">WebSocketインスタンス</param>
        void AddConnection(string clientId, WebSocket webSocket);

        /// <summary>
        /// クライアント接続の削除
        /// </summary>
        /// <param name="clientId">クライアントID</param>
        /// <returns>削除に成功したか</returns>
        bool RemoveConnection(string clientId);

        /// <summary>
        /// クライアントへのメッセージ送信
        /// </summary>
        /// <param name="clientId">クライアントID</param>
        /// <param name="message">送信メッセージ</param>
        /// <returns>送信タスク</returns>
        Task SendMessageAsync(string clientId, WebSocketMessage message);

        /// <summary>
        /// すべてのクライアントへのメッセージ送信
        /// </summary>
        /// <param name="message">送信メッセージ</param>
        /// <returns>送信タスク</returns>
        Task BroadcastMessageAsync(WebSocketMessage message);
```

```
        /// <summary>
        /// クライアントセッションの取得
        /// </summary>
        /// <param name="clientId">クライアントID</param>
        /// <returns>セッション情報</returns>
        WebSocketSession? GetSession(string clientId);

        /// <summary>
        /// クライアントセッションの更新
        /// </summary>
        /// <param name="clientId">クライアントID</param>
        /// <param name="sessionInfo">セッション情報</param>
        void UpdateSession(string clientId, WebSocketSession sessionInfo);
    }

    /// <summary>
    /// WebSocketセッション情報
    /// </summary>
    public class WebSocketSession
    {
        /// <summary>
        /// MCPセッションID
        /// </summary>
        public string? SessionId { get; set; }

        /// <summary>
        /// ユーザー情報
        /// </summary>
        public UserInfo? User { get; set; }

        /// <summary>
        /// 接続時間
        /// </summary>
        public DateTime ConnectedAt { get; set; } = DateTime.UtcNow;

        /// <summary>
        /// セッションメタデータ
        /// </summary>
        public Dictionary<string, object> Metadata { get; set; } = new();
    }

    /// <summary>
    /// 接続マネージャーの実装
    /// </summary>
    public class ConnectionManager : IConnectionManager
    {
        private readonly ConcurrentDictionary<string, WebSocket> _connections = new();
        private readonly ConcurrentDictionary<string, WebSocketSession> _sessions =
new();
        private readonly ILogger<ConnectionManager> _logger;

        public ConnectionManager(ILogger<ConnectionManager> logger)
        {
            _logger = logger;
        }
```

```
        /// <inheritdoc />
        public void AddConnection(string clientId, WebSocket webSocket)
        {
            if (_connections.TryAdd(clientId, webSocket))
            {
                _sessions[clientId] = new WebSocketSession();
                _logger.LogInformation("Client connected: {ClientId}", clientId);
            }
        }

        /// <inheritdoc />
        public bool RemoveConnection(string clientId)
        {
            if (_connections.TryRemove(clientId, out var webSocket))
            {
                _sessions.TryRemove(clientId, out _);
                _logger.LogInformation("Client disconnected: {ClientId}", clientId);
                return true;
            }

            return false;
        }

        /// <inheritdoc />
        public async Task SendMessageAsync(string clientId, WebSocketMessage message)
        {
            if (!_connections.TryGetValue(clientId, out var webSocket))
            {
                _logger.LogWarning("Cannot send message to unknown client: {ClientId}",
clientId);
                return;
            }

            if (webSocket.State != WebSocketState.Open)
            {
                _logger.LogWarning("WebSocket not in open state for client:
{ClientId}", clientId);
                RemoveConnection(clientId);
                return;
            }

            try
            {
                string json = JsonSerializer.Serialize(message);
                byte[] buffer = Encoding.UTF8.GetBytes(json);
                await webSocket.SendAsync(
                    new ArraySegment<byte>(buffer),
                    WebSocketMessageType.Text,
                    true,
                    CancellationToken.None);
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Error sending message to client: {ClientId}",
clientId);
                RemoveConnection(clientId);
            }
```

```
        }

        /// <inheritdoc />
        public async Task BroadcastMessageAsync(WebSocketMessage message)
        {
            var tasks = new List<Task>();

            foreach (var clientId in _connections.Keys)
            {
                tasks.Add(SendMessageAsync(clientId, message));
            }

            await Task.WhenAll(tasks);
        }

        /// <inheritdoc />
        public WebSocketSession? GetSession(string clientId)
        {
            return _sessions.TryGetValue(clientId, out var session) ? session : null;
        }

        /// <inheritdoc />
        public void UpdateSession(string clientId, WebSocketSession sessionInfo)
        {
            _sessions[clientId] = sessionInfo;
        }
    }
}
```

```
// Middleware/WebSocketHandlerMiddleware.cs
using System.Net.WebSockets;
using System.Text;
using System.Text.Json;
using MCPServer.Models;
using MCPServer.Services;

namespace MCPServer.Middleware
{
    /// <summary>
    /// WebSocket処理ミドルウェア
    /// </summary>
    public class WebSocketHandlerMiddleware
    {
        private readonly RequestDelegate _next;
        private readonly ILogger<WebSocketHandlerMiddleware> _logger;

        public WebSocketHandlerMiddleware(RequestDelegate next,
ILogger<WebSocketHandlerMiddleware> logger)
        {
            _next = next;
            _logger = logger;
        }

        public async Task InvokeAsync(HttpContext context,
                                      IConnectionManager connectionManager,
                                      IConnectorManager connectorManager,
```

```
                                    IContextManager contextManager)
        {
            if (context.Request.Path == "/mcp/stream")
            {
                if (context.WebSockets.IsWebSocketRequest)
                {
                    WebSocket webSocket = await
context.WebSockets.AcceptWebSocketAsync();
                    string clientId = Guid.NewGuid().ToString();

                    // 接続を登録
                    connectionManager.AddConnection(clientId, webSocket);

                    // 接続確立メッセージを送信
                    await connectionManager.SendMessageAsync(clientId, new
WebSocketMessage
                    {
                        Type = "connection_established",
                        Data = new
                        {
                            clientId,
                            timestamp = DateTime.UtcNow
                        }
                    });

                    // クライアントからのメッセージを処理
                    await ProcessWebSocketMessagesAsync(webSocket, clientId,
connectionManager, connectorManager, contextManager);
                }
                else
                {
                    context.Response.StatusCode = StatusCodes.Status400BadRequest;
                }
            }
            else
            {
                await _next(context);
            }
        }

        private async Task ProcessWebSocketMessagesAsync(WebSocket webSocket,
                                                         string clientId,
                                                         IConnectionManager
connectionManager,
                                                         IConnectorManager
connectorManager,
                                                         IContextManager contextManager)
        {
            var buffer = new byte[4096];
            WebSocketReceiveResult result;

            try
            {
                while (webSocket.State == WebSocketState.Open)
                {
                    using var ms = new MemoryStream();
```

```
                    do
                    {
                        result = await webSocket.ReceiveAsync(new ArraySegment<byte>
(buffer), CancellationToken.None);
                        ms.Write(buffer, 0, result.Count);
                    } while (!result.EndOfMessage);

                    ms.Seek(0, SeekOrigin.Begin);

                    if (result.MessageType == WebSocketMessageType.Text)
                    {
                        string message = Encoding.UTF8.GetString(ms.ToArray());
                        await HandleWebSocketMessageAsync(message, clientId,
connectionManager, connectorManager, contextManager);
                    }
                    else if (result.MessageType == WebSocketMessageType.Close)
                    {
                        await webSocket.CloseAsync(WebSocketCloseStatus.NormalClosure,
"Closing", CancellationToken.None);
                        connectionManager.RemoveConnection(clientId);
                        break;
                    }
                }
            }
            catch (WebSocketException ex)
            {
                _logger.LogWarning(ex, "WebSocket error for client {ClientId}",
clientId);
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Error processing WebSocket messages for client
{ClientId}", clientId);
            }
            finally
            {
                connectionManager.RemoveConnection(clientId);

                if (webSocket.State != WebSocketState.Closed && webSocket.State !=
WebSocketState.Aborted)
                {
                    try
                    {
                        await
webSocket.CloseAsync(WebSocketCloseStatus.InternalServerError, "Internal error",
CancellationToken.None);
                    }
                    catch (Exception ex)
                    {
                        _logger.LogError(ex, "Error closing WebSocket for client
{ClientId}", clientId);
                    }
                }
            }
        }

        private async Task HandleWebSocketMessageAsync(string messageJson,
```

```
                                                    string clientId,
                                                    IConnectionManager
connectionManager,
                                                    IConnectorManager
connectorManager,
                                                    IContextManager contextManager)
        {
            try
            {
                var message = JsonSerializer.Deserialize<Dictionary<string,
JsonElement>>(messageJson);
                if (message == null)
                {
                    return;
                }

                // メッセージタイプによる処理分岐
                if (message.TryGetValue("type", out var typeElement) &&
typeElement.ValueKind == JsonValueKind.String)
                {
                    string type = typeElement.GetString() ?? string.Empty;

                    switch (type)
                    {
                        case "ping":
                            await connectionManager.SendMessageAsync(clientId, new
WebSocketMessage
                            {
                                Type = "pong",
                                Data = new { timestamp = DateTime.UtcNow }
                            });
                            break;

                        case "auth":
                            if (message.TryGetValue("sessionId", out var
sessionElement) &&
                                sessionElement.ValueKind == JsonValueKind.String)
                            {
                                string sessionId = sessionElement.GetString() ??
string.Empty;
                                var session = connectionManager.GetSession(clientId);
                                if (session != null)
                                {
                                    session.SessionId = sessionId;
                                    connectionManager.UpdateSession(clientId, session);
                                }
                            }
                            break;

                        default:
                            // MCPリクエストとして処理
                            await HandleMCPRequestAsync(message, clientId,
connectionManager, connectorManager, contextManager);
                            break;
                    }
                }
                else
```

```
                {
                    // MCPリクエストとして処理
                    await HandleMCPRequestAsync(message, clientId, connectionManager,
connectorManager, contextManager);
                }
            }
            catch (JsonException ex)
            {
                _logger.LogError(ex, "Error parsing WebSocket message: {Message}",
messageJson);

                await connectionManager.SendMessageAsync(clientId, new WebSocketMessage
                {
                    Type = "error",
                    Data = new
                    {
                        code = "INVALID_JSON",
                        message = "Invalid JSON in message",
                        details = ex.Message
                    }
                });
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Error handling WebSocket message");

                await connectionManager.SendMessageAsync(clientId, new WebSocketMessage
                {
                    Type = "error",
                    Data = new
                    {
                        code = "INTERNAL_ERROR",
                        message = "Internal server error",
                        details = ex.Message
                    }
                });
            }
        }

        private async Task HandleMCPRequestAsync(Dictionary<string, JsonElement>
message,
                                                 string clientId,
                                                 IConnectionManager connectionManager,
                                                 IConnectorManager connectorManager,
                                                 IContextManager contextManager)
        {
            if (!message.TryGetValue("requestType", out var requestTypeElement) ||
                requestTypeElement.ValueKind != JsonValueKind.String ||
                !message.TryGetValue("resource", out var resourceElement) ||
                resourceElement.ValueKind != JsonValueKind.String)
            {
                await connectionManager.SendMessageAsync(clientId, new WebSocketMessage
                {
                    Type = "error",
                    Data = new
                    {
                        code = "INVALID_REQUEST",
```

```
                message = "Missing required fields: requestType, resource"
            }
        });
        return;
    }

    string requestType = requestTypeElement.GetString() ?? string.Empty;
    string resource = resourceElement.GetString() ?? string.Empty;

    // パラメータの抽出
    Dictionary<string, object>? parameters = null;
    if (message.TryGetValue("params", out var paramsElement) &&
        paramsElement.ValueKind == JsonValueKind.Object)
    {
        parameters = JsonSerializer.Deserialize<Dictionary<string, object>>(
            paramsElement.GetRawText());
    }

    // コンテキスト情報の取得
    var session = connectionManager.GetSession(clientId);
    var context = session?.SessionId != null
        ? contextManager.GetContext(session.SessionId)
        : contextManager.CreateContext();

    if (context == null)
    {
        context = contextManager.CreateContext();

        if (session != null)
        {
            session.SessionId = context.SessionId;
            connectionManager.UpdateSession(clientId, session);
        }
    }

    // セッション情報を更新
    if (session != null)
    {
        context.User = session.User;
    }

    // リソースからコネクタを特定
    var connectorInfo = connectorManager.GetConnectorForResource(resource);
    if (connectorInfo == null)
    {
        await connectionManager.SendMessageAsync(clientId, new WebSocketMessage
        {
            Type = "error",
            Data = new
            {
                code = "CONNECTOR_NOT_FOUND",
                message = $"No connector available for resource: {resource}"
            }
        });
        return;
    }
```

```
            var (connector, resourceInfo) = connectorInfo.Value;

            // ストリーミング処理の開始
            try
            {
                var request = new ConnectorRequest
                {
                    RequestType = requestType,
                    Resource = resourceInfo,
                    Params = parameters,
                    Context = context
                };

                // ストリーミング非同期イテレーターを取得
                var stream = connector.HandleStreamingAsync(request);

                // 非同期処理をバックグラウンドで実行
                _ = Task.Run(async () =>
                {
                    try
                    {
                        await foreach (var chunk in stream)
                        {
                            // クライアントが切断されていないか確認
                            if
(!connectionManager.GetSession(clientId)?.SessionId.Equals(context.SessionId) ?? true)
                            {
                                break;
                            }

                            // チャンクをクライアントに送信
                            await connectionManager.SendMessageAsync(clientId, new
WebSocketMessage
                            {
                                Type = "chunk",
                                Data = chunk
                            });
                        }

                        // 完了通知
                        await connectionManager.SendMessageAsync(clientId, new
WebSocketMessage
                        {
                            Type = "complete",
                            Data = new { timestamp = DateTime.UtcNow }
                        });
                    }
                    catch (Exception ex)
                    {
                        _logger.LogError(ex, "Error in streaming for client
{ClientId}", clientId);

                        // エラー通知
                        await connectionManager.SendMessageAsync(clientId, new
WebSocketMessage
                        {
                            Type = "error",
```

```
                            Data = new
                            {
                                code = "STREAMING_ERROR",
                                message = "Error during streaming",
                                details = ex.Message
                            }
                        });
                    }
                });
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Error starting streaming for client {ClientId}",
clientId);

                await connectionManager.SendMessageAsync(clientId, new WebSocketMessage
                {
                    Type = "error",
                    Data = new
                    {
                        code = "STREAMING_INIT_ERROR",
                        message = "Error initializing streaming",
                        details = ex.Message
                    }
                });
            }
        }
    }

    /// <summary>
    /// WebSocketハンドラーミドルウェア拡張メソッド
    /// </summary>
    public static class WebSocketHandlerMiddlewareExtensions
    {
        public static IApplicationBuilder UseWebSocketHandler(this IApplicationBuilder
app)
        {
            return app.UseMiddleware<WebSocketHandlerMiddleware>();
        }
    }
}
```

10.3.5 認証サービスの実装

```
// Services/IAuthService.cs
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;
using Microsoft.IdentityModel.Tokens;

namespace MCPServer.Services
{
    /// <summary>
    /// 認証サービスインターフェース
    /// </summary>
```

```
    public interface IAuthService
    {
        /// <summary>
        /// ユーザー認証
        /// </summary>
        /// <param name="username">ユーザー名</param>
        /// <param name="password">パスワード</param>
        /// <returns>認証結果と認証トークン</returns>
        Task<(bool success, string? token)> AuthenticateAsync(string username, string
password);

        /// <summary>
        /// トークン検証
        /// </summary>
        /// <param name="token">認証トークン</param>
        /// <returns>検証結果とユーザー情報</returns>
        Task<(bool success, UserInfo? user)> ValidateTokenAsync(string token);

        /// <summary>
        /// パスワードのハッシュ化
        /// </summary>
        /// <param name="password">パスワード</param>
        /// <returns>ハッシュ値</returns>
        string HashPassword(string password);

        /// <summary>
        /// パスワード検証
        /// </summary>
        /// <param name="password">パスワード</param>
        /// <param name="hashedPassword">ハッシュ値</param>
        /// <returns>検証結果</returns>
        bool VerifyPassword(string password, string hashedPassword);
    }

    /// <summary>
    /// JWT認証サービスの実装
    /// </summary>
    public class JwtAuthService : IAuthService
    {
        private readonly IConfiguration _configuration;
        private readonly ILogger<JwtAuthService> _logger;

        // テスト用の簡易ユーザーストア（本番環境ではデータベースを使用）
        private readonly Dictionary<string, (string hashedPassword, List<string>
roles)> _users = new();

        public JwtAuthService(IConfiguration configuration, ILogger<JwtAuthService>
logger)
        {
            _configuration = configuration;
            _logger = logger;

            // サンプルユーザーの初期化
            InitializeUsers();
        }

        private void InitializeUsers()
```

```
        {
            // サンプルユーザー追加（本番環境では実際のデータソースを使用）
            _users["admin"] = (HashPassword("adminpass"), new List<string> { "admin",
"user" });
            _users["user"] = (HashPassword("userpass"), new List<string> { "user" });
        }

        /// <inheritdoc />
        public async Task<(bool success, string? token)> AuthenticateAsync(string
username, string password)
        {
            // ユーザーの存在確認
            if (!_users.TryGetValue(username, out var userData))
            {
                return (false, null);
            }

            // パスワード検証
            if (!VerifyPassword(password, userData.hashedPassword))
            {
                return (false, null);
            }

            // JWTトークン生成
            var token = GenerateJwtToken(username, userData.roles);
            return (true, token);
        }

        /// <inheritdoc />
        public async Task<(bool success, UserInfo? user)> ValidateTokenAsync(string
token)
        {
            try
            {
                var tokenHandler = new JwtSecurityTokenHandler();
                var key = Encoding.ASCII.GetBytes(_configuration["Jwt:Secret"] ??
"default_secret_key_for_development_only");

                var tokenValidationParameters = new TokenValidationParameters
                {
                    ValidateIssuerSigningKey = true,
                    IssuerSigningKey = new SymmetricSecurityKey(key),
                    ValidateIssuer = true,
                    ValidIssuer = _configuration["Jwt:Issuer"] ?? "MCPServer",
                    ValidateAudience = true,
                    ValidAudience = _configuration["Jwt:Audience"] ?? "MCPClients",
                    ValidateLifetime = true,
                    ClockSkew = TimeSpan.Zero
                };

                // トークン検証
                var principal = tokenHandler.ValidateToken(token,
tokenValidationParameters, out _);

                // クレームからユーザー情報を取得
                var usernameClaim = principal.FindFirst(ClaimTypes.Name)?.Value;
                if (string.IsNullOrEmpty(usernameClaim))
```

```
                {
                    return (false, null);
                }

                var rolesClaims = principal.FindAll(ClaimTypes.Role).Select(c =>
c.Value).ToList();

                var user = new UserInfo
                {
                    Id = principal.FindFirst(ClaimTypes.NameIdentifier)?.Value ??
usernameClaim,
                    Username = usernameClaim,
                    Roles = rolesClaims
                };

                return (true, user);
            }
            catch (Exception ex)
            {
                _logger.LogWarning(ex, "Token validation failed");
                return (false, null);
            }
        }

        /// <inheritdoc />
        public string HashPassword(string password)
        {
            // 実際の実装ではBCryptなどのセキュアなハッシュアルゴリズムを使用
            // この簡略実装は例示目的のみで、本番環境では使用しないでください
            return Convert.ToBase64String(
                System.Security.Cryptography.SHA256.Create().ComputeHash(
                    Encoding.UTF8.GetBytes(password + "salt")));
        }

        /// <inheritdoc />
        public bool VerifyPassword(string password, string hashedPassword)
        {
            // 実際の実装ではBCryptなどのセキュアな検証方法を使用
            var computedHash = HashPassword(password);
            return computedHash == hashedPassword;
        }

        private string GenerateJwtToken(string username, List<string> roles)
        {
            var tokenHandler = new JwtSecurityTokenHandler();
            var key = Encoding.ASCII.GetBytes(_configuration["Jwt:Secret"] ??
"default_secret_key_for_development_only");

            var claims = new List<Claim>
            {
                new Claim(ClaimTypes.Name, username),
                new Claim(ClaimTypes.NameIdentifier, username),
            };

            // ロールのクレームを追加
            foreach (var role in roles)
            {
```

```
                claims.Add(new Claim(ClaimTypes.Role, role));
            }

            var tokenDescriptor = new SecurityTokenDescriptor
            {
                Subject = new ClaimsIdentity(claims),
                Expires =
DateTime.UtcNow.AddHours(double.Parse(_configuration["Jwt:ExpiryHours"] ?? "24")),
                Issuer = _configuration["Jwt:Issuer"] ?? "MCPServer",
                Audience = _configuration["Jwt:Audience"] ?? "MCPClients",
                SigningCredentials = new SigningCredentials(
                    new SymmetricSecurityKey(key),
                    SecurityAlgorithms.HmacSha256Signature)
            };

            var token = tokenHandler.CreateToken(tokenDescriptor);
            return tokenHandler.WriteToken(token);
        }
    }
}
```

10.3.6 サンプルコネクタの実装

```
// Connectors/EchoConnector.cs
using MCPServer.Services;

namespace MCPServer.Connectors
{
    /// <summary>
    /// エコーコネクタ（デモ用）
    /// </summary>
    public class EchoConnector : IConnector
    {
        private readonly ILogger<EchoConnector> _logger;

        public string Id => "echo";
        public string Name => "Echo Connector";
        public string Version => "1.0.0";
        public string[] SupportedSchemes => new[] { "echo" };

        public EchoConnector(ILogger<EchoConnector> logger)
        {
            _logger = logger;
        }

        public Task InitializeAsync()
        {
            _logger.LogInformation("Echo connector initialized");
            return Task.CompletedTask;
        }

        public Task<ConnectorResponse> HandleRequestAsync(ConnectorRequest request)
        {
            _logger.LogInformation("Echo request: {RequestType} to {Resource}",
                request.RequestType, request.Resource.Uri);
```

```
            // リクエストをそのままエコー
            return Task.FromResult(ConnectorResponse.Success(new
            {
                echo = request,
                timestamp = DateTime.UtcNow
            }));
        }

        public async IAsyncEnumerable<object> HandleStreamingAsync(ConnectorRequest
request)
        {
            // テスト用のストリーミングレスポンス
            for (int i = 0; i < 5; i++)
            {
                await Task.Delay(1000); // 1秒ごとにチャンクを送信

                yield return new
                {
                    chunk = i + 1,
                    data = $"Echo streaming chunk {i + 1}",
                    progress = (i + 1) * 20,
                    timestamp = DateTime.UtcNow
                };
            }
        }

        public ResourceInfo ParseResourceUri(string resourceUri)
        {
            if (!resourceUri.StartsWith("echo://"))
            {
                throw new ArgumentException($"Invalid echo resource URI:
{resourceUri}");
            }

            // 簡易的なパース処理
            var path = resourceUri.Substring(7); // "echo://" を削除
            var queryIndex = path.IndexOf('?');

            var queryParams = new Dictionary<string, string>();

            if (queryIndex >= 0)
            {
                var queryString = path.Substring(queryIndex + 1);
                path = path.Substring(0, queryIndex);

                foreach (var param in queryString.Split('&'))
                {
                    var parts = param.Split('=');
                    if (parts.Length == 2)
                    {
                        queryParams[parts[0]] = Uri.UnescapeDataString(parts[1]);
                    }
                }
            }

            return new ResourceInfo
```

```
            {
                Uri = resourceUri,
                Scheme = "echo",
                Path = path,
                QueryParams = queryParams
            };
        }

        public Task DisposeAsync()
        {
            _logger.LogInformation("Echo connector disposed");
            return Task.CompletedTask;
        }
    }
}
```

```
// Connectors/FileConnector.cs
using MCPServer.Services;

namespace MCPServer.Connectors
{
    /// <summary>
    /// ファイルコネクタ（デモ用）
    /// </summary>
    public class FileConnector : IConnector
    {
        private readonly ILogger<FileConnector> _logger;
        private readonly string _basePath;

        public string Id => "file";
        public string Name => "File Connector";
        public string Version => "1.0.0";
        public string[] SupportedSchemes => new[] { "file" };

        public FileConnector(IConfiguration configuration, ILogger<FileConnector>
logger)
        {
            _logger = logger;
            _basePath = configuration["FileConnector:BasePath"] ??
Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "data");
        }

        public Task InitializeAsync()
        {
            // データディレクトリの作成確認
            if (!Directory.Exists(_basePath))
            {
                Directory.CreateDirectory(_basePath);
            }

            _logger.LogInformation("File connector initialized. Base path: {BasePath}",
_basePath);
            return Task.CompletedTask;
        }

        public async Task<ConnectorResponse> HandleRequestAsync(ConnectorRequest
```

```
request)
        {
            var filePath = GetFullPath(request.Resource.Path);

            // セキュリティチェック
            if (!IsPathSafe(filePath))
            {
                return ConnectorResponse.Error("SECURITY_ERROR", "Invalid file path");
            }

            switch (request.RequestType)
            {
                case "query":
                    return await QueryFileAsync(filePath, request.Params);

                case "update":
                    return await UpdateFileAsync(filePath, request.Params);

                case "create":
                    return await CreateFileAsync(filePath, request.Params);

                case "delete":
                    return await DeleteFileAsync(filePath);

                default:
                    return ConnectorResponse.Error("UNSUPPORTED_REQUEST_TYPE",
                        $"Unsupported request type: {request.RequestType}");
            }
        }

        public async IAsyncEnumerable<object> HandleStreamingAsync(ConnectorRequest
request)
        {
            var filePath = GetFullPath(request.Resource.Path);

            // セキュリティチェック
            if (!IsPathSafe(filePath))
            {
                yield return new
                {
                    error = new
                    {
                        code = "SECURITY_ERROR",
                        message = "Invalid file path"
                    }
                };
                yield break;
            }

            if (!File.Exists(filePath))
            {
                yield return new
                {
                    error = new
                    {
                        code = "FILE_NOT_FOUND",
                        message = $"File not found: {request.Resource.Path}"
```

```
                    }
                };
                yield break;
            }

            try
            {
                // ファイル情報を取得
                var fileInfo = new FileInfo(filePath);
                long fileSize = fileInfo.Length;

                // ファイルを分割して読み込み
                int chunkSize = 1024; // 1KB
                using var fileStream = new FileStream(filePath, FileMode.Open,
FileAccess.Read);
                using var reader = new StreamReader(fileStream);

                char[] buffer = new char[chunkSize];
                int bytesRead;
                long totalRead = 0;

                while ((bytesRead = await reader.ReadAsync(buffer, 0, chunkSize)) > 0)
                {
                    string chunk = new string(buffer, 0, bytesRead);
                    totalRead += bytesRead;
                    int progress = (int)Math.Min(100, (totalRead * 100) / fileSize);

                    yield return new
                    {
                        data = chunk,
                        progress,
                        total_size = fileSize,
                        bytes_read = totalRead
                    };

                    await Task.Delay(100); // チャンク間の遅延
                }
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Error streaming file: {FilePath}", filePath);

                yield return new
                {
                    error = new
                    {
                        code = "STREAMING_ERROR",
                        message = ex.Message
                    }
                };
            }
        }

        public ResourceInfo ParseResourceUri(string resourceUri)
        {
            if (!resourceUri.StartsWith("file://"))
            {
```

```
                throw new ArgumentException($"Invalid file resource URI:
{resourceUri}");
            }

            // リソースURIのパース
            var path = resourceUri.Substring(7); // "file://" を削除
            var queryIndex = path.IndexOf('?');

            var queryParams = new Dictionary<string, string>();

            if (queryIndex >= 0)
            {
                var queryString = path.Substring(queryIndex + 1);
                path = path.Substring(0, queryIndex);

                foreach (var param in queryString.Split('&'))
                {
                    var parts = param.Split('=');
                    if (parts.Length == 2)
                    {
                        queryParams[parts[0]] = Uri.UnescapeDataString(parts[1]);
                    }
                }
            }

            return new ResourceInfo
            {
                Uri = resourceUri,
                Scheme = "file",
                Path = path,
                QueryParams = queryParams
            };
        }

        public Task DisposeAsync()
        {
            _logger.LogInformation("File connector disposed");
            return Task.CompletedTask;
        }

        // プライベートヘルパーメソッド

        private string GetFullPath(string relativePath)
        {
            return Path.Combine(_basePath, relativePath.TrimStart('/'));
        }

        private bool IsPathSafe(string fullPath)
        {
            // パストラバーサル防止
            var normalizedPath = Path.GetFullPath(fullPath);
            var normalizedBasePath = Path.GetFullPath(_basePath);

            return normalizedPath.StartsWith(normalizedBasePath);
        }

        private async Task<ConnectorResponse> QueryFileAsync(string filePath,
```

```
Dictionary<string, object>? parameters)
        {
            try
            {
                // ファイルが存在しない場合
                if (!File.Exists(filePath))
                {
                    // ディレクトリであれば一覧を返す
                    if (Directory.Exists(filePath))
                    {
                        var files = Directory.GetFileSystemEntries(filePath)
                            .Select(Path.GetFileName)
                            .ToList();

                        return ConnectorResponse.Success(new
                        {
                            type = "directory",
                            path = GetRelativePath(filePath),
                            files
                        });
                    }

                    return ConnectorResponse.Error("FILE_NOT_FOUND", $"File not found:
{GetRelativePath(filePath)}");
                }

                // ファイル内容を読み込み
                string content = await File.ReadAllTextAsync(filePath);
                var fileInfo = new FileInfo(filePath);

                return ConnectorResponse.Success(new
                {
                    type = "file",
                    path = GetRelativePath(filePath),
                    content,
                    size = fileInfo.Length,
                    modified = fileInfo.LastWriteTimeUtc
                });
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Error reading file: {FilePath}", filePath);
                return ConnectorResponse.Error("FILE_READ_ERROR", ex.Message);
            }
        }

        private async Task<ConnectorResponse> UpdateFileAsync(string filePath,
Dictionary<string, object>? parameters)
        {
            try
            {
                // パラメータチェック
                if (parameters == null || !parameters.TryGetValue("content", out var
contentObj))
                {
                    return ConnectorResponse.Error("MISSING_CONTENT", "Content
parameter is required for update");
```

```
                }

                string content = contentObj?.ToString() ?? string.Empty;

                // ファイルの親ディレクトリを確認・作成
                var directory = Path.GetDirectoryName(filePath);
                if (!string.IsNullOrEmpty(directory) && !Directory.Exists(directory))
                {
                    Directory.CreateDirectory(directory);
                }

                // ファイル書き込み
                await File.WriteAllTextAsync(filePath, content);

                var fileInfo = new FileInfo(filePath);

                return ConnectorResponse.Success(new
                {
                    type = "file",
                    path = GetRelativePath(filePath),
                    size = fileInfo.Length,
                    modified = fileInfo.LastWriteTimeUtc,
                    status = "updated"
                });
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Error updating file: {FilePath}", filePath);
                return ConnectorResponse.Error("FILE_WRITE_ERROR", ex.Message);
            }
        }

        private async Task<ConnectorResponse> CreateFileAsync(string filePath,
Dictionary<string, object>? parameters)
        {
            try
            {
                // ファイルが既に存在する場合
                if (File.Exists(filePath))
                {
                    return ConnectorResponse.Error("FILE_EXISTS", $"File already
exists: {GetRelativePath(filePath)}");
                }

                // パラメータチェック
                if (parameters == null || !parameters.TryGetValue("content", out var
contentObj))
                {
                    return ConnectorResponse.Error("MISSING_CONTENT", "Content
parameter is required for create");
                }

                string content = contentObj?.ToString() ?? string.Empty;

                // ファイルの親ディレクトリを確認・作成
                var directory = Path.GetDirectoryName(filePath);
                if (!string.IsNullOrEmpty(directory) && !Directory.Exists(directory))
```

```
                {
                    Directory.CreateDirectory(directory);
                }

                // ファイル作成
                await File.WriteAllTextAsync(filePath, content);

                var fileInfo = new FileInfo(filePath);

                return ConnectorResponse.Success(new
                {
                    type = "file",
                    path = GetRelativePath(filePath),
                    size = fileInfo.Length,
                    modified = fileInfo.LastWriteTimeUtc,
                    status = "created"
                });
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Error creating file: {FilePath}", filePath);
                return ConnectorResponse.Error("FILE_CREATE_ERROR", ex.Message);
            }
        }

        private async Task<ConnectorResponse> DeleteFileAsync(string filePath)
        {
            try
            {
                // ファイルが存在しない場合
                if (!File.Exists(filePath))
                {
                    return ConnectorResponse.Error("FILE_NOT_FOUND", $"File not found:
{GetRelativePath(filePath)}");
                }

                // ファイル削除
                File.Delete(filePath);

                return ConnectorResponse.Success(new
                {
                    type = "file",
                    path = GetRelativePath(filePath),
                    status = "deleted"
                });
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Error deleting file: {FilePath}", filePath);
                return ConnectorResponse.Error("FILE_DELETE_ERROR", ex.Message);
            }
        }

        private string GetRelativePath(string fullPath)
        {
            return fullPath.Substring(_basePath.Length).Replace('\\',
'/').TrimStart('/');
```

```
        }
    }
}
```

10.3.7 MCPコントローラーの実装

```
// Controllers/MCPController.cs
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using MCPServer.Models;
using MCPServer.Services;

namespace MCPServer.Controllers
{
    [ApiController]
    [Route("mcp")]
    public class MCPController : ControllerBase
    {
        private readonly IConnectorManager _connectorManager;
        private readonly IContextManager _contextManager;
        private readonly IAuthService _authService;
        private readonly IPluginManager _pluginManager;
        private readonly ILogger<MCPController> _logger;

        public MCPController(
            IConnectorManager connectorManager,
            IContextManager contextManager,
            IAuthService authService,
            IPluginManager pluginManager,
            ILogger<MCPController> logger)
        {
            _connectorManager = connectorManager;
            _contextManager = contextManager;
            _authService = authService;
            _pluginManager = pluginManager;
            _logger = logger;
        }

        /// <summary>
        /// MCPリクエスト処理エンドポイント
        /// </summary>
        [HttpPost]
        [Authorize]
        public async Task<ActionResult<MCPResponse>> ProcessRequest([FromBody]
MCPRequest request)
        {
            _logger.LogInformation("Received MCP request: {RequestType} to {Resource}",
                request.RequestType, request.Resource);

            // リクエストIDの生成
            string requestId = Guid.NewGuid().ToString();

            try
            {
                // バージョン確認
```

```
                if (request.Version != "1.0" && request.Version != "1.1")
                {
                    return new MCPResponse
                    {
                        Version = request.Version,
                        Status = 400,
                        RequestId = requestId,
                        Error = new ErrorInfo
                        {
                            Code = "UNSUPPORTED_VERSION",
                            Message = $"Unsupported MCP version: {request.Version}"
                        }
                    };
                }

                // コンテキスト処理
                ConnectorContext? sessionContext = null;
                if (request.Context != null &&
!string.IsNullOrEmpty(request.Context.SessionId))
                {
                    sessionContext =
_contextManager.GetContext(request.Context.SessionId);

                    if (sessionContext == null)
                    {
                        // セッションが見つからない場合は新規作成
                        sessionContext = _contextManager.CreateContext(
                            request.Context.Metadata as Dictionary<string, object>);
                    }
                }
                else
                {
                    // 新しいコンテキストを作成
                    sessionContext = _contextManager.CreateContext();
                }

                // ユーザー情報の設定
                var username = User.Identity?.Name;
                if (!string.IsNullOrEmpty(username))
                {
                    sessionContext.User = new UserInfo
                    {
                        Id = username,
                        Username = username,
                        Roles = User.Claims
                            .Where(c => c.Type ==
System.Security.Claims.ClaimTypes.Role)
                            .Select(c => c.Value)
                            .ToList()
                    };
                }

                // リクエストをプラグインのリクエストフックに通す
                var modifiedRequest = await
_pluginManager.ApplyRequestHooksAsync(request, sessionContext);

                // リクエスト処理
```

```
                var response = await
_connectorManager.ProcessRequestAsync(modifiedRequest, sessionContext);

                // レスポンスをプラグインのレスポンスフックに通す
                var modifiedResponse = await
_pluginManager.ApplyResponseHooksAsync(response, sessionContext);

                return modifiedResponse;
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Error processing MCP request");

                return new MCPResponse
                {
                    Version = request.Version,
                    Status = 500,
                    RequestId = requestId,
                    Error = new ErrorInfo
                    {
                        Code = "INTERNAL_ERROR",
                        Message = "An internal error occurred while processing the
request",
                        Details = ex.Message
                    }
                };
            }
        }

        /// <summary>
        /// プラグイン使用のMCPリクエスト処理
        /// </summary>
        [HttpPost("v2")]
        [Authorize]
        public async Task<ActionResult<MCPResponse>>
ProcessRequestWithPlugins([FromBody] MCPRequest request)
        {
            _logger.LogInformation("Received MCP v2 request: {RequestType} to
{Resource}",
                request.RequestType, request.Resource);

            return await ProcessRequest(request);
        }

        /// <summary>
        /// APIキー認証用のエンドポイント
        /// </summary>
        [HttpPost("api")]
        [ApiKeyAuth]
        public async Task<ActionResult<MCPResponse>> ProcessApiRequest([FromBody]
MCPRequest request)
        {
            return await ProcessRequest(request);
        }
    }

    /// <summary>
```

```
    /// 認証コントローラー
    /// </summary>
    [ApiController]
    [Route("auth")]
    public class AuthController : ControllerBase
    {
        private readonly IAuthService _authService;
        private readonly ILogger<AuthController> _logger;

        public AuthController(IAuthService authService, ILogger<AuthController> logger)
        {
            _authService = authService;
            _logger = logger;
        }

        public class LoginModel
        {
            public string Username { get; set; } = string.Empty;
            public string Password { get; set; } = string.Empty;
        }

        public class LoginResponse
        {
            public string Token { get; set; } = string.Empty;
            public string TokenType { get; set; } = "bearer";
            public int ExpiresIn { get; set; } = 86400; // 24時間
        }

        /// <summary>
        /// ログイン処理
        /// </summary>
        [HttpPost("login")]
        public async Task<ActionResult<LoginResponse>> Login([FromBody] LoginModel
model)
        {
            try
            {
                var (success, token) = await
_authService.AuthenticateAsync(model.Username, model.Password);

                if (!success || string.IsNullOrEmpty(token))
                {
                    return Unauthorized(new { message = "Invalid username or password"
});
                }

                return new LoginResponse
                {
                    Token = token,
                    TokenType = "bearer",
                    ExpiresIn = 86400 // 24時間
                };
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Login error");
                return StatusCode(500, new { message = "Internal server error" });
```

```
            }
        }
    }

    /// <summary>
    /// APIキー認証属性
    /// </summary>
    [AttributeUsage(AttributeTargets.Class | AttributeTargets.Method)]
    public class ApiKeyAuthAttribute : Attribute, IAsyncActionFilter
    {
        private const string ApiKeyHeaderName = "X-API-Key";

        public async Task OnActionExecutionAsync(ActionExecutingContext context,
ActionExecutionDelegate next)
        {
            // ヘッダーからAPIキーを取得
            if (!context.HttpContext.Request.Headers.TryGetValue(ApiKeyHeaderName, out
var potentialApiKey))
            {
                context.Result = new UnauthorizedResult();
                return;
            }

            var configuration =
context.HttpContext.RequestServices.GetRequiredService<IConfiguration>();
            var apiKeys = configuration.GetSection("ApiKeys").Get<List<string>>() ??
new List<string>();

            // APIキーの検証
            var apiKey = potentialApiKey.ToString();
            if (!apiKeys.Contains(apiKey))
            {
                context.Result = new UnauthorizedResult();
                return;
            }

            await next();
        }
    }
}
```

10.3.8 プラグインシステムの実装

```
// Services/IPluginManager.cs
using MCPServer.Models;

namespace MCPServer.Services
{
    /// <summary>
    /// プラグインマネージャーインターフェース
    /// </summary>
    public interface IPluginManager
    {
        /// <summary>
        /// プラグインのロード
```

```
        /// </summary>
        Task LoadPluginsAsync();

        /// <summary>
        /// プラグインの登録
        /// </summary>
        /// <param name="plugin">プラグインインスタンス</param>
        void RegisterPlugin(IMCPPlugin plugin);

        /// <summary>
        /// プラグインのリクエストフック適用
        /// </summary>
        /// <param name="request">MCPリクエスト</param>
        /// <param name="context">コンテキスト</param>
        Task<MCPRequest> ApplyRequestHooksAsync(MCPRequest request, ConnectorContext
context);

        /// <summary>
        /// プラグインのレスポンスフック適用
        /// </summary>
        /// <param name="response">MCPレスポンス</param>
        /// <param name="context">コンテキスト</param>
        Task<MCPResponse> ApplyResponseHooksAsync(MCPResponse response,
ConnectorContext context);
    }

    /// <summary>
    /// MCPプラグインインターフェース
    /// </summary>
    public interface IMCPPlugin
    {
        /// <summary>
        /// プラグインID
        /// </summary>
        string Id { get; }

        /// <summary>
        /// プラグイン名
        /// </summary>
        string Name { get; }

        /// <summary>
        /// プラグインバージョン
        /// </summary>
        string Version { get; }

        /// <summary>
        /// 初期化処理
        /// </summary>
        Task InitializeAsync();

        /// <summary>
        /// リクエスト処理前のフック
        /// </summary>
        /// <param name="request">MCPリクエスト</param>
        /// <param name="context">コンテキスト</param>
        Task<MCPRequest> OnRequestAsync(MCPRequest request, ConnectorContext context);
```

```
        /// <summary>
        /// レスポンス処理前のフック
        /// </summary>
        /// <param name="response">MCPレスポンス</param>
        /// <param name="context">コンテキスト</param>
        Task<MCPResponse> OnResponseAsync(MCPResponse response, ConnectorContext
context);

        /// <summary>
        /// シャットダウン処理
        /// </summary>
        Task ShutdownAsync();
    }

    /// <summary>
    /// プラグインマネージャーの実装
    /// </summary>
    public class PluginManager : IPluginManager
    {
        private readonly List<IMCPPlugin> _plugins = new();
        private readonly ILogger<PluginManager> _logger;
        private readonly IServiceProvider _serviceProvider;
        private readonly IConfiguration _configuration;

        public PluginManager(
            ILogger<PluginManager> logger,
            IServiceProvider serviceProvider,
            IConfiguration configuration)
        {
            _logger = logger;
            _serviceProvider = serviceProvider;
            _configuration = configuration;
        }

        /// <inheritdoc />
        public async Task LoadPluginsAsync()
        {
            // 内蔵プラグインの登録
            RegisterBuiltInPlugins();

            // 外部プラグインのロード
            await LoadExternalPluginsAsync();

            // プラグインの初期化
            await InitializePluginsAsync();
        }

        /// <inheritdoc />
        public void RegisterPlugin(IMCPPlugin plugin)
        {
            if (_plugins.Any(p => p.Id == plugin.Id))
            {
                _logger.LogWarning("Plugin with ID '{PluginId}' is already registered",
plugin.Id);
                return;
            }
```

```
            _plugins.Add(plugin);
            _logger.LogInformation("Registered plugin: {PluginName} (v{Version})",
plugin.Name, plugin.Version);
        }

        /// <inheritdoc />
        public async Task<MCPRequest> ApplyRequestHooksAsync(MCPRequest request,
ConnectorContext context)
        {
            var currentRequest = request;

            foreach (var plugin in _plugins)
            {
                try
                {
                    currentRequest = await plugin.OnRequestAsync(currentRequest,
context);
                }
                catch (Exception ex)
                {
                    _logger.LogError(ex, "Error in request hook for plugin
'{PluginId}'", plugin.Id);
                }
            }

            return currentRequest;
        }

        /// <inheritdoc />
        public async Task<MCPResponse> ApplyResponseHooksAsync(MCPResponse response,
ConnectorContext context)
        {
            var currentResponse = response;

            foreach (var plugin in _plugins)
            {
                try
                {
                    currentResponse = await plugin.OnResponseAsync(currentResponse,
context);
                }
                catch (Exception ex)
                {
                    _logger.LogError(ex, "Error in response hook for plugin
'{PluginId}'", plugin.Id);
                }
            }

            return currentResponse;
        }

        private void RegisterBuiltInPlugins()
        {
            // ロギングプラグイン
            RegisterPlugin(new LoggingPlugin(_logger));
```

```
            // メトリクスプラグイン
            RegisterPlugin(new MetricsPlugin(_logger));
        }

        private async Task LoadExternalPluginsAsync()
        {
            var pluginsPath = _configuration["Plugins:Path"] ??
Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "Plugins");

            if (!Directory.Exists(pluginsPath))
            {
                _logger.LogInformation("Plugins directory not found: {Path}",
pluginsPath);
                return;
            }

            var pluginDlls = Directory.GetFiles(pluginsPath, "*.dll");

            foreach (var pluginDll in pluginDlls)
            {
                try
                {
                    var assembly =
System.Runtime.Loader.AssemblyLoadContext.Default.LoadFromAssemblyPath(pluginDll);

                    var pluginTypes = assembly.GetTypes()
                        .Where(t => typeof(IMCPPlugin).IsAssignableFrom(t) &&
!t.IsAbstract && !t.IsInterface)
                        .ToList();

                    foreach (var pluginType in pluginTypes)
                    {
                        try
                        {
                            // プラグインのインスタンス化（依存性注入を試みる）
                            IMCPPlugin? plugin = null;

                            try
                            {
                                plugin =
(IMCPPlugin)ActivatorUtilities.CreateInstance(_serviceProvider, pluginType);
                            }
                            catch
                            {
                                // フォールバック: パラメータなしコンストラクタでインスタンス化
                                plugin =
(IMCPPlugin)Activator.CreateInstance(pluginType)!;
                            }

                            if (plugin != null)
                            {
                                RegisterPlugin(plugin);
                            }
                        }
                        catch (Exception ex)
                        {
                            _logger.LogError(ex, "Failed to instantiate plugin type:
```

```
{Type}", pluginType.FullName);
                        }
                    }
                }
                catch (Exception ex)
                {
                    _logger.LogError(ex, "Failed to load plugin from {Path}",
pluginDll);
                }
            }
        }

        private async Task InitializePluginsAsync()
        {
            foreach (var plugin in _plugins)
            {
                try
                {
                    await plugin.InitializeAsync();
                    _logger.LogInformation("Initialized plugin: {PluginName}",
plugin.Name);
                }
                catch (Exception ex)
                {
                    _logger.LogError(ex, "Failed to initialize plugin '{PluginId}'",
plugin.Id);
                }
            }
        }
    }

    /// <summary>
    /// ロギングプラグイン実装
    /// </summary>
    public class LoggingPlugin : IMCPPlugin
    {
        private readonly ILogger _logger;

        public string Id => "logging-plugin";
        public string Name => "Logging Plugin";
        public string Version => "1.0.0";

        public LoggingPlugin(ILogger logger)
        {
            _logger = logger;
        }

        public Task InitializeAsync()
        {
            _logger.LogInformation("Logging plugin initialized");
            return Task.CompletedTask;
        }

        public Task<MCPRequest> OnRequestAsync(MCPRequest request, ConnectorContext
context)
        {
            _logger.LogInformation("MCP Request: {RequestType} to {Resource}",
```

```
request.RequestType, request.Resource);
            return Task.FromResult(request);
        }

        public Task<MCPResponse> OnResponseAsync(MCPResponse response, ConnectorContext
context)
        {
            if (response.Error != null)
            {
                _logger.LogWarning("MCP Response Error: {Code} - {Message}",
                    response.Error.Code, response.Error.Message);
            }

            return Task.FromResult(response);
        }

        public Task ShutdownAsync()
        {
            _logger.LogInformation("Logging plugin shutdown");
            return Task.CompletedTask;
        }
    }

    /// <summary>
    /// メトリクスプラグイン実装
    /// </summary>
    public class MetricsPlugin : IMCPPlugin
    {
        private readonly ILogger _logger;
        private int _requestCount = 0;
        private int _errorCount = 0;
        private readonly List<double> _responseTimes = new();
        private readonly object _lock = new();

        public string Id => "metrics-plugin";
        public string Name => "Metrics Plugin";
        public string Version => "1.0.0";

        public MetricsPlugin(ILogger logger)
        {
            _logger = logger;
        }

        public Task InitializeAsync()
        {
            _logger.LogInformation("Metrics plugin initialized");
            return Task.CompletedTask;
        }

        public Task<MCPRequest> OnRequestAsync(MCPRequest request, ConnectorContext
context)
        {
            lock (_lock)
            {
                _requestCount++;

                // コンテキストに開始時間を記録
```

```
                if (context != null)
                {
                    context.Data["metrics_start_time"] = DateTime.UtcNow;
                }
            }

            return Task.FromResult(request);
        }

        public Task<MCPResponse> OnResponseAsync(MCPResponse response, ConnectorContext
context)
        {
            lock (_lock)
            {
                // エラーカウント
                if (response.Error != null)
                {
                    _errorCount++;
                }

                // レスポンスタイム計測
                if (context != null && context.Data.TryGetValue("metrics_start_time",
out var startTimeObj))
                {
                    if (startTimeObj is DateTime startTime)
                    {
                        var responseTime = (DateTime.UtcNow -
startTime).TotalMilliseconds;
                        _responseTimes.Add(responseTime);

                        // 最新の100件だけ保持
                        if (_responseTimes.Count > 100)
                        {
                            _responseTimes.RemoveAt(0);
                        }
                    }
                }

                // 定期的にメトリクスをログに出力（100リクエストごと）
                if (_requestCount % 100 == 0)
                {
                    var avgResponseTime = _responseTimes.Count > 0 ?
_responseTimes.Average() : 0;

                    _logger.LogInformation(
                        "Metrics: {RequestCount} requests, {ErrorCount} errors, " +
                        "avg response time: {AvgResponseTime:F2}ms",
                        _requestCount, _errorCount, avgResponseTime);
                }
            }

            return Task.FromResult(response);
        }

        public Task ShutdownAsync()
        {
            // 最終メトリクスをログに出力
```

```
            var avgResponseTime = _responseTimes.Count > 0 ? _responseTimes.Average() :
0;

            _logger.LogInformation(
                "Final Metrics: {RequestCount} requests, {ErrorCount} errors, " +
                "avg response time: {AvgResponseTime:F2}ms",
                _requestCount, _errorCount, avgResponseTime);

            return Task.CompletedTask;
        }
    }
}
```

10.3.9 レート制限とセキュリティミドルウェア

```
// Middleware/RateLimitingMiddleware.cs
using System.Collections.Concurrent;
using System.Net;

namespace MCPServer.Middleware
{
    /// <summary>
    /// レート制限ミドルウェア
    /// </summary>
    public class RateLimitingMiddleware
    {
        private readonly RequestDelegate _next;
        private readonly ILogger<RateLimitingMiddleware> _logger;
        private readonly RateLimitingOptions _options;

        // クライアントごとのリクエスト記録
        private readonly ConcurrentDictionary<string, List<DateTime>> _requestRecords =
new();

        public RateLimitingMiddleware(
            RequestDelegate next,
            ILogger<RateLimitingMiddleware> logger,
            IConfiguration configuration)
        {
            _next = next;
            _logger = logger;

            // 設定の読み込み
            _options = new RateLimitingOptions();
            configuration.GetSection("RateLimiting").Bind(_options);
        }

        public async Task InvokeAsync(HttpContext context)
        {
            // APIエンドポイントのみレート制限を適用
            if (context.Request.Path.StartsWithSegments("/mcp") ||
                context.Request.Path.StartsWithSegments("/api"))
            {
                string clientIp = GetClientIpAddress(context);
```

```
                if (IsRateLimited(clientIp))
                {
                    _logger.LogWarning("Rate limit exceeded for client: {ClientIp}",
clientIp);

                    context.Response.StatusCode = (int)HttpStatusCode.TooManyRequests;
                    context.Response.ContentType = "application/json";

                    await context.Response.WriteAsync(
                        "{\"error\":{\"code\":\"RATE_LIMITED\",\"message\":\"Too many
requests, please try again later\"}}");

                    return;
                }

                // リクエストを記録
                RecordRequest(clientIp);
            }

            await _next(context);
        }

        private bool IsRateLimited(string clientIp)
        {
            // ホワイトリストのチェック
            if (_options.Whitelist.Contains(clientIp))
            {
                return false;
            }

            // ブラックリストのチェック
            if (_options.Blacklist.Contains(clientIp))
            {
                return true;
            }

            var now = DateTime.UtcNow;
            var windowStart = now.AddSeconds(-_options.WindowSeconds);

            // 現在のウィンドウ内のリクエストを取得
            if (!_requestRecords.TryGetValue(clientIp, out var records))
            {
                return false;
            }

            // 古いレコードの削除
            records.RemoveAll(r => r < windowStart);

            // レート制限の確認
            return records.Count >= _options.MaxRequests;
        }

        private void RecordRequest(string clientIp)
        {
            var now = DateTime.UtcNow;

            _requestRecords.AddOrUpdate(
```

```
                clientIp,
                // キーが存在しない場合は新しいリストを作成
                _ => new List<DateTime> { now },
                // キーが存在する場合は既存のリストに追加
                (_, records) =>
                {
                    records.Add(now);
                    return records;
                });
        }

        private string GetClientIpAddress(HttpContext context)
        {
            // X-Forwarded-Forヘッダーのチェック
            if (context.Request.Headers.TryGetValue("X-Forwarded-For", out var
forwardedIps))
            {
                var ips = forwardedIps.ToString().Split(',',
StringSplitOptions.RemoveEmptyEntries);
                if (ips.Length > 0)
                {
                    return ips[0].Trim();
                }
            }

            // リモートIPアドレスを使用
            return context.Connection.RemoteIpAddress?.ToString() ?? "0.0.0.0";
        }
    }

    /// <summary>
    /// レート制限オプション
    /// </summary>
    public class RateLimitingOptions
    {
        /// <summary>
        /// ウィンドウ期間（秒）
        /// </summary>
        public int WindowSeconds { get; set; } = 60;

        /// <summary>
        /// 最大リクエスト数
        /// </summary>
        public int MaxRequests { get; set; } = 100;

        /// <summary>
        /// ホワイトリスト
        /// </summary>
        public List<string> Whitelist { get; set; } = new List<string>();

        /// <summary>
        /// ブラックリスト
        /// </summary>
        public List<string> Blacklist { get; set; } = new List<string>();
    }

    /// <summary>
```

```
    /// レート制限ミドルウェア拡張メソッド
    /// </summary>
    public static class RateLimitingMiddlewareExtensions
    {
        public static IApplicationBuilder UseRateLimiting(this IApplicationBuilder app)
        {
            return app.UseMiddleware<RateLimitingMiddleware>();
        }
    }
}
```

```
// Middleware/RequestLoggingMiddleware.cs
using System.Diagnostics;

namespace MCPServer.Middleware
{
    /// <summary>
    /// リクエストログミドルウェア
    /// </summary>
    public class RequestLoggingMiddleware
    {
        private readonly RequestDelegate _next;
        private readonly ILogger<RequestLoggingMiddleware> _logger;

        public RequestLoggingMiddleware(RequestDelegate next,
ILogger<RequestLoggingMiddleware> logger)
        {
            _next = next;
            _logger = logger;
        }

        public async Task InvokeAsync(HttpContext context)
        {
            var stopwatch = Stopwatch.StartNew();

            // リクエスト情報をログに記録
            _logger.LogInformation(
                "Request: {Method} {Path} - Client: {ClientIp} - User: {User}",
                context.Request.Method,
                context.Request.Path,
                context.Connection.RemoteIpAddress,
                context.User?.Identity?.Name ?? "Anonymous");

            // リクエスト本文のロギング（開発環境のみ）
            if (_logger.IsEnabled(LogLevel.Debug))
            {
                var originalBodyStream = context.Response.Body;

                try
                {
                    // リクエスト本文の読み取り（必要に応じて）
                    if (context.Request.ContentLength > 0 &&
context.Request.ContentLength < 10000)
                    {
                        context.Request.EnableBuffering();
```

```
                        using var reader = new StreamReader(
                            context.Request.Body,
                            encoding: System.Text.Encoding.UTF8,
                            detectEncodingFromByteOrderMarks: false,
                            leaveOpen: true);

                        var body = await reader.ReadToEndAsync();
                        _logger.LogDebug("Request Body: {Body}", body);

                        context.Request.Body.Position = 0;
                    }
                }
                catch (Exception ex)
                {
                    _logger.LogWarning(ex, "Error logging request body");
                }
            }

            // レスポンスの処理
            await _next(context);

            // レスポンス情報をログに記録
            stopwatch.Stop();

            _logger.LogInformation(
                "Response: {StatusCode} - Duration: {ElapsedMilliseconds}ms",
                context.Response.StatusCode,
                stopwatch.ElapsedMilliseconds);
        }
    }

    /// <summary>
    /// リクエストログミドルウェア拡張メソッド
    /// </summary>
    public static class RequestLoggingMiddlewareExtensions
    {
        public static IApplicationBuilder UseRequestLogging(this IApplicationBuilder
app)
        {
            return app.UseMiddleware<RequestLoggingMiddleware>();
        }
    }
}
```

10.3.10 設定と展開の考慮事項

```
// appsettings.json
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
```

```
  "Jwt": {
    "Secret": "your_secret_key_at_least_32_bytes_long_for_security",
    "Issuer": "MCPServer",
    "Audience": "MCPClients",
    "ExpiryHours": 24
  },
  "ApiKeys": [
    "test-api-key-1",
    "test-api-key-2"
  ],
  "RateLimiting": {
    "WindowSeconds": 60,
    "MaxRequests": 120,
    "Whitelist": [
      "127.0.0.1",
      "::1"
    ],
    "Blacklist": []
  },
  "FileConnector": {
    "BasePath": "./data"
  },
  "Plugins": {
    "Path": "./Plugins"
  }
}
```

```
<!-- MCPServer.csproj -->
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>net6.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
    <GenerateDocumentationFile>true</GenerateDocumentationFile>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.Authentication.JwtBearer"
Version="6.0.13" />
    <PackageReference Include="Swashbuckle.AspNetCore" Version="6.5.0" />
    <PackageReference Include="System.IdentityModel.Tokens.Jwt" Version="6.26.0" />
  </ItemGroup>

  <ItemGroup>
    <None Update="data\README.md">
      <CopyToOutputDirectory>PreserveNewest</CopyToOutputDirectory>
    </None>
  </ItemGroup>

  <ItemGroup>
    <Folder Include="Plugins\" />
  </ItemGroup>

</Project>
```

デプロイ方法については、以下のアプローチが考えられます：

1. **コンテナ化（Docker）**：
   MCPサーバーをコンテナ化することで、環境に依存しない一貫したデプロイが可能になります。

```
FROM mcr.microsoft.com/dotnet/aspnet:6.0 AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build
WORKDIR /src
COPY ["MCPServer.csproj", "./"]
RUN dotnet restore "MCPServer.csproj"
COPY . .
RUN dotnet build "MCPServer.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "MCPServer.csproj" -c Release -o /app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .

# データディレクトリの作成
RUN mkdir -p /app/data
RUN mkdir -p /app/Plugins

ENTRYPOINT ["dotnet", "MCPServer.dll"]
```

2. **クラウドホスティング**：
   Azure App ServicesやAWS Elastic Beanstalkなどのクラウドプラットフォームを使用することで、スケーラビリティと管理性を向上させることができます。
3. **オンプレミスデプロイ**：
   自社サーバー環境でのデプロイも可能です。その場合は、以下の点に注意してください：
   - IIS (Windows) または Nginx (Linux) などのリバースプロキシの設定
   - SSL証明書の適用
   - 監視とロギングの設定

展開時の考慮事項：

1. **セキュリティ**：
   - 本番環境では必ず強力なJWTシークレットキーを使用
   - APIキーの安全な管理
   - HTTPS/TLSの強制
   - 適切なCORSポリシーの設定
2. **スケーラビリティ**：
   - 水平スケーリングのためのステートレス設計
   - セッションやコンテキストの分散ストレージ（例：Redis）
   - 負荷分散の設定
3. **監視と運用**：

- APM（Application Performance Monitoring）ツールの統合
- ヘルスチェックエンドポイントの実装
- ログ集約と分析システムの導入

4. **ライフサイクル管理**：
   - CIパイプラインを使った自動デプロイ
   - バージョン管理と更新プロセス
   - プラグインの配布とアップデート機構

> **ベテランの知恵袋**: *「MCPサーバーのデプロイでは、環境変数を使った設定の外部化が非常に重要です。特に認証情報やAPIキーなどのセンシティブな情報は、直接コードやappsettings.jsonに含めるべきではありません。また、コンテナ化されたデプロイはスケーリングとバージョン管理が容易になるため、初期の開発段階からDockerコンテナを使った開発環境を整えておくことをお勧めします。」* - クラウドアーキテクト

10.4 MCPサーバー実装の比較と選択ガイド

これまで3つの異なる実装アプローチを紹介してきました。ここでは、それぞれの特性を比較し、用途に応じた選択指針を提供します。

10.4.1 実装比較

特性	Python (FastAPI)	React/TypeScript	C# (.NET)
言語の特性	動的型付け、迅速な開発	フロントエンド向け、型安全	静的型付け、高パフォーマンス
主な用途	小〜中規模サーバー、プロトタイピング	クライアント実装、ブラウザ統合	大規模エンタープライズサーバー
学習曲線	緩やか	中程度	やや急
パフォーマンス	中程度	クライアント次第	高い
スケーラビリティ	中程度	低い（主にクライアント）	高い
コード量	少ない	中程度	多い
静的型安全性	オプション (型ヒント)	高い (TypeScript)	高い (.NET)
デプロイ容易性	高い	高い（静的ホスティング）	中程度
エコシステム	豊富なパッケージ	npm/React エコシステム	.NET エコシステム
統合容易性	広範なライブラリ	ブラウザAPIとの統合	企業システムとの統合

10.4.2 選択ガイド

Python (FastAPI) を選ぶべき場合:

- 迅速な開発とプロトタイピングが最優先
- 小〜中規模のサーバー実装
- データサイエンスや機械学習との統合が重要

- 開発リソースが限られている
- フレキシブルな実行環境が必要

React/TypeScript を選ぶべき場合:

- クライアントサイド実装が主目的
- ウェブブラウザやモバイルアプリでの実行
- 既存のウェブアプリケーションへの統合
- UIとの密接な連携が必要
- ユーザー体験の最適化が重要

C# (.NET) を選ぶべき場合:

- エンタープライズレベルの高い信頼性が必要
- 大規模システムでの運用
- 高いパフォーマンス要件がある
- 厳格な型安全性が求められる
- Windowsベースの環境との統合が必要
- 長期的なメンテナンスと拡張性が重要

10.4.3 混合アプローチの可能性

実際のシステムでは、これらの実装を組み合わせることも考えられます：

1. **マイクロサービスアーキテクチャ**:
   - コア機能は高性能なC#で実装
   - データ処理やAI連携はPythonで実装
   - ユーザーインターフェースはReactで実装
2. **プログレッシブアプローチ**:
   - まずはPythonでプロトタイピングと検証
   - ユーザーインターフェースをReactで構築
   - 本番稼働時にはコア機能をC#に移行
3. **専門性に基づく分業**:
   - バックエンド開発者はPythonまたはC#
   - フロントエンド開発者はReact
   - データサイエンティストはPython

> **プロジェクト事例**: *ある金融機関のAI統合プロジェクトでは、Python FastAPIによるMCPサーバーでプロトタイピングを行い、ユーザー体験の検証を早期に完了。その後、高いスケーラビリティとセキュリティ要件を満たすため、コア機能をC#に移行しながら、フロントエンドはReactで開発を継続しました。この混合アプローチにより、プロジェクトの初期段階から価値を提供しながら、最終的には堅牢なエンタープライズシステムを構築することに成功しました。*

第10章まとめ

この章では、MCPサーバーとクライアントの自作実装について、Python、React/TypeScript、C#の3つの異なるアプローチで詳細に解説しました。

主要なポイント

1. **実装アプローチの多様性**:
   - 各言語とフレームワークは、それぞれの強みと適用分野を持つ
   - 用途に応じた技術選択が重要
2. **共通の設計原則**:
   - コンテキスト管理の重要性
   - プラグインアーキテクチャによる拡張性
   - エラー処理とリカバリーメカニズムの実装
   - セキュリティ対策の組み込み
3. **実装上の考慮点**:
   - スケーラビリティと性能の確保
   - セキュリティと認証の実装
   - コネクタの標準化と再利用性
   - デプロイと運用の簡便性

自作MCPサーバー実装チェックリスト

- [ ] 要件と用途に基づく言語・フレームワークの選定
- [ ] プロトコル仕様の理解と遵守
- [ ] コアコンポーネント（コネクタ、コンテキスト管理、認証）の設計
- [ ] エラー処理と例外管理の実装
- [ ] テストケースの作成と実行
- [ ] セキュリティレビューの実施
- [ ] パフォーマンステストとベンチマーク
- [ ] デプロイメントパイプラインの構築
- [ ] モニタリングと運用計画の策定
- [ ] ドキュメントとAPI仕様の作成

次のステップ

MCPサーバーとクライアントの実装を進める際に考慮すべき次のステップ：

1. **標準コネクタの開発**:
   - 主要サービス（GitHub, Obsidian, Redmineなど）への標準コネクタ実装
   - コネクタ開発のためのSDKと開発者ツールの提供
2. **セキュリティの強化**:
   - 認証と認可の高度な実装
   - エンドツーエンド暗号化の検討
   - 監査とコンプライアンス機能の追加
3. **パフォーマンス最適化**:
   - 大規模データ処理におけるストリーミング技術の活用
   - キャッシュ戦略の高度化
   - 分散システムアーキテクチャの採用
4. **フェデレーション機能**:
   - 複数MCPサーバー間の相互接続
   - 分散環境での検索と情報統合
   - クロスドメインのコンテキスト共有

MCPサーバーとクライアントの実装は、単なる技術的な課題ではなく、組織のナレッジ管理と生成AIの効果的な活用を可能にする戦略的な取り組みです。本章で紹介した実装例を出発点として、組織固有のニーズに合わせたカスタマイズと拡張を進めることで、生成AI時代における知識管理の新たな可能性を開拓できるでしょう。

第11章: 結論と将来展望 - MCPプロトコルの進化と可能性

11.1 MCPプロトコルの現状

Message Capacity Protocol（MCP）は、生成AIと各種開発・知識管理ツールを効率的に連携させるための通信プロトコルとして誕生し、急速に発展しています。本書で見てきたように、MCPは以下のような特徴を持っています：

11.1.1 MCPの核となる強み

1. **統一されたコンテキスト管理**

   MCPの最大の強みは、異なるツールやサービス間でコンテキストを維持・共有する能力です。これにより、ユーザーは複数のツールを使用しながらも一貫した対話体験を得ることができます。
2. **拡張性と柔軟性**

   プラグインアーキテクチャと標準化されたコネクタシステムにより、新しいサービスやツールを容易に統合できます。これは、技術環境が急速に変化する現代において極めて重要な特性です。
3. **セマンティック処理の統合**

   単なるデータ交換を超え、情報の意味的関連性を理解・活用する機能を持つことで、より高度な知識探索と統合が可能になっています。
4. **実装の多様性**

   Python、JavaScript/TypeScript、C#など、様々な言語とフレームワークでの実装が可能であり、組織の既存技術スタックや要件に合わせた選択ができます。

11.1.2 現在の課題

一方で、MCPの普及と発展には以下のような課題も存在します：

1. **標準化の進展段階**

   MCPはまだ発展途上のプロトコルであり、完全な標準化には至っていません。実装間の互換性を確保するための取り組みが続いています。
2. **セキュリティモデルの成熟**

   特に機密情報や個人情報を扱う場合の高度なセキュリティモデルについては、さらなる発展が期待されています。
3. **大規模データ処理の最適化**

   大量のデータを扱う際のパフォーマンスと効率性については、継続的な改善が必要です。
4. **開発者エコシステムの拡大**

   MCPの広範な普及のためには、開発者ツール、ドキュメント、サンプル実装などのエコシステムのさらなる充実が必要です。

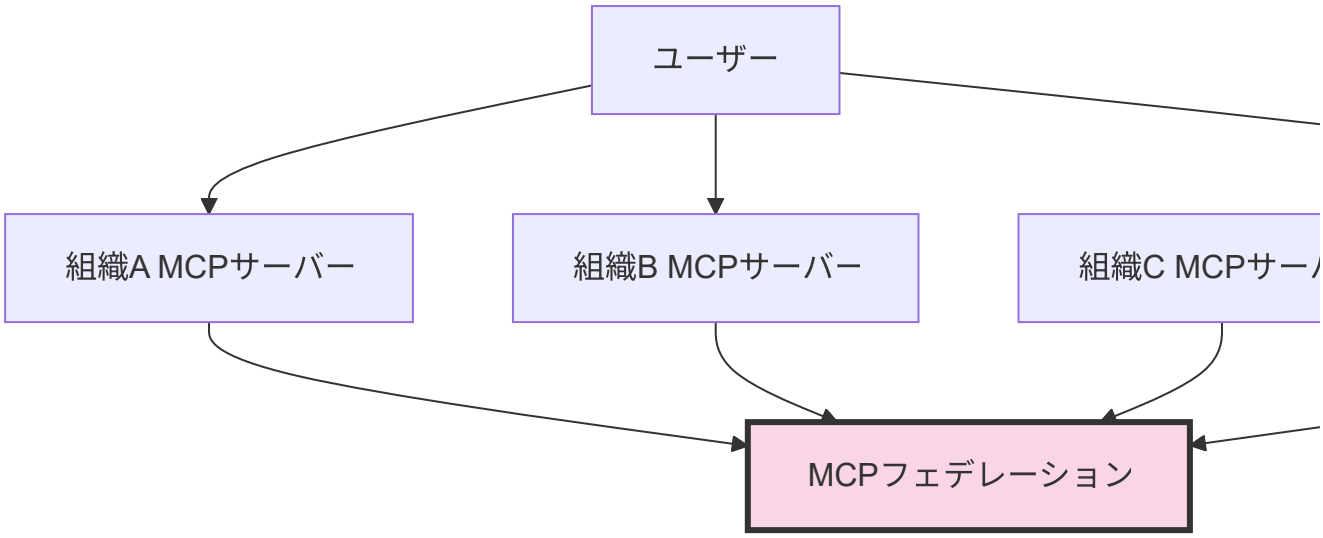
> **ベテランの知恵袋**: *「現時点でのMCPは、電子メールが初期の頃に似ています。基本的な機能と価値は明らかですが、広範な標準化とエコシステムの成熟はこれからです。しかし、初期段階から参加することで、組織は競争優位性を確立し、プロトコルの進化に影響を与えることができます。」* - エンタープライズアーキテクト

11.2 未来への展望

MCPの今後の発展について、いくつかの重要な方向性が見えてきています：

11.2.1 技術的進化

1. **フェデレーションモデルの強化**



   複数のMCPサーバー間で情報を検索・共有できるフェデレーションモデルの発展により、組織横断的な知識共有と協働が促進されるでしょう。
2. **マルチモーダル対応の強化**

   テキストだけでなく、画像、音声、動画などのマルチモーダルコンテンツも含めたコンテキスト管理と処理能力が向上することで、より豊かな情報交換が可能になります。
3. **リアルタイム協調機能**

   複数のユーザーとAIが同時に対話し、協働作業を行うためのリアルタイム機能が強化され、より自然な協調作業環境が実現するでしょう。
4. **エッジコンピューティングの統合**

   ローカル処理とクラウド処理を効率的に組み合わせる「エッジMCP」の発展により、プライバシー保護とパフォーマンス向上の両立が図られるでしょう。

11.2.2 応用領域の拡大

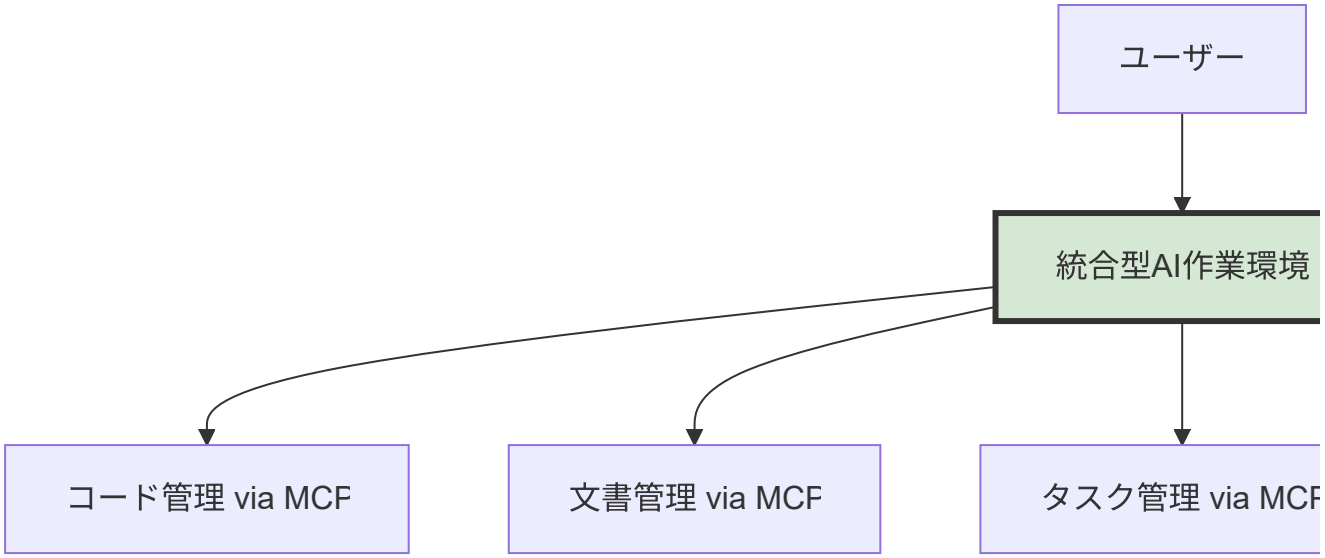
1. **企業知識管理の革新**

   MCPは企業の知識管理システムに革命をもたらす可能性があります。過去のドキュメント、メール、チャット履歴、コードリポジトリなどが統合され、AIを通じて容易にアクセス・活用できるようになります。
2. **教育・学習環境への統合**

   教育現場では、学習管理システム、参考資料、対話型チュートリアル、評価システムなどが統合され、個別最適化された学習体験を提供できるようになるでしょう。
3. **研究開発プロセスの加速**

   科学研究や製品開発において、過去の実験データ、論文、特許情報などを統合的に参照・活用できるようになり、イノベーションサイクルが加速するでしょう。

4. **統合型ワークスペースの実現**



   様々なツールが統合された、AIを中心とする新世代のワークスペースが誕生し、業務の分断と文脈切り替えのコストが大幅に削減されるでしょう。

11.2.3 組織的・社会的影響

1. **ナレッジワーカーの役割変化**

   MCPとAIの統合により、情報の検索・整理・要約といった作業の多くが自動化され、ナレッジワーカーはより創造的・戦略的な業務に集中できるようになります。
2. **組織の知識民主化**

   従来は少数の専門家や長期勤続者に集中していた暗黙知が、より広く組織全体で共有・活用されるようになり、意思決定の質と速度が向上するでしょう。
3. **新しい協働モデルの出現**

   人間とAIの新しい協働モデルが生まれ、チームの構成や仕事の進め方が根本的に変わる可能性があります。
4. **情報アクセスの公平性課題**

   一方で、MCPのような高度な情報統合技術へのアクセスの格差が、組織間や社会間の新たな不均衡を生み出す可能性にも注意が必要です。

> **プロジェクト事例**: *ある国際的な研究機関では、MCPを活用して世界中の研究拠点のデータベース、論文リポジトリ、実験装置のログを統合する「フェデレーテッド・リサーチ・ハブ」を構築しました。研究者はこれにより、以前は数週間かかっていた関連研究のレビューと統合を数時間で完了できるようになり、研究サイクルが大幅に加速しました。特に分野横断的な研究において、従来は見落とされていた関連性の発見が増加しています。*

11.3 MCPエコシステム構築のステップ

組織内でMCPエコシステムを構築・発展させるための段階的アプローチを考えてみましょう：

11.3.1 初期導入フェーズ

1. **ニーズと機会の分析**

- 現在のワークフローにおける情報の分断点の特定
- 生成AIとの統合による価値創出機会の評価
- パイロットプロジェクトの選定

2. **基本インフラの構築**
   - 必要なMCPサーバーの選定または開発
   - 核となるコネクタの実装（例：コード管理、ドキュメント管理）
   - 認証・セキュリティ基盤の整備
3. **パイロット運用**
   - 限定されたユーザーグループでの試験的利用
   - フィードバックループの確立
   - 初期成果の測定と評価

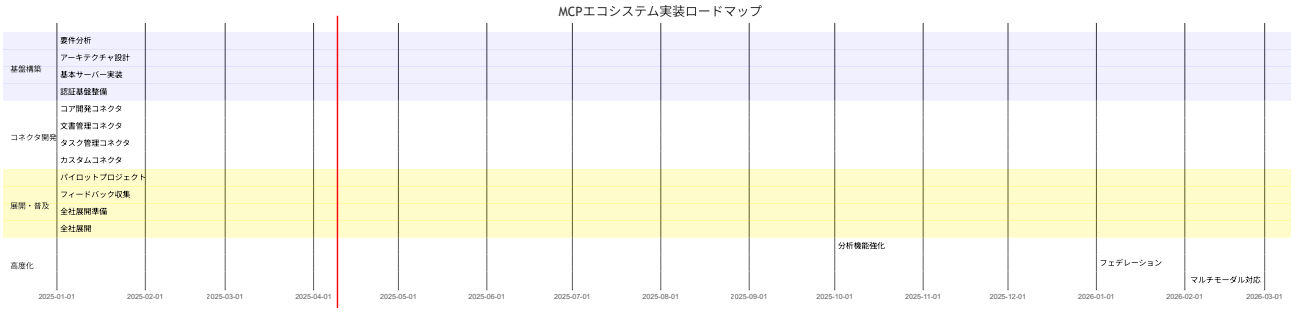
11.3.2 拡張フェーズ

1. **コネクタエコシステムの拡大**
   - 追加サービスへのコネクタ開発
   - カスタムコネクタ開発のためのガイドラインと支援体制の整備
   - コネクタの品質保証プロセスの確立
2. **ユーザーエクスペリエンスの最適化**
   - 各部門・役割に特化したUIの開発
   - ワークフロー統合の深化
   - トレーニングプログラムの実施
3. **組織的な展開**
   - 部門横断的な利用の促進
   - 成功事例の共有と横展開
   - 効果測定の体系化

11.3.3 成熟フェーズ

1. **高度な統合と自動化**
   - 業務プロセスとの深い統合
   - ワークフロー自動化の拡大
   - 予測的機能の導入
2. **フェデレーションの実現**
   - 部門間、組織間のMCP連携
   - 外部パートナーとの安全な知識共有
   - フェデレーテッド検索と統合分析
3. **継続的イノベーション**
   - 新技術の積極的な取り込み
   - ユーザー主導のイノベーションプログラム
   - コミュニティへの貢献と協調的進化

11.3.4 実装ロードマップの例



> **若手の疑問解決**: *「MCP導入の最初のステップとして、何に注力すべきでしょうか？」*
>
> MCPの初期導入で最も重要なのは、「小さく始めて迅速に価値を示す」アプローチです。組織全体のすべてのシステムをMCPに接続しようとするのではなく、特定のチームや部門における具体的な課題（たとえば開発者のコードとドキュメントの分断、営業部門の顧客情報へのアクセス改善など）に焦点を当て、その解決に直接寄与するMCP接続から始めましょう。これにより、具体的な成功事例を早期に作り出し、組織内での採用を促進する基盤になります。また、初期パイロットから得られるフィードバックは、その後の大規模展開における貴重な指針となります。

11.4 MCP導入の成功要因

MCPエコシステム構築の成功を左右する重要な要因は以下の通りです：

11.4.1 技術的成功要因

1. **モジュール性と拡張性の重視**
   - 疎結合のコンポーネント設計
   - 明確なAPI契約と標準規格の遵守
   - 将来の拡張を考慮した設計
2. **セキュリティファースト**
   - 最小権限の原則
   - アクセス制御と監査の徹底
   - データプライバシーの保護
3. **パフォーマンスの最適化**
   - 適切なキャッシュ戦略
   - 非同期・ストリーミング処理の活用
   - 段階的なスケーリング計画
4. **堅牢なエラー処理**
   - グレースフルデグラデーション（段階的縮退）
   - 自己修復メカニズム
   - 明確なエラー通知とロギング

11.4.2 組織的成功要因

1. **明確な価値提案**
   - 具体的な業務課題への紐付け
   - 測定可能な成果指標の設定
   - 短期的成果と長期的可能性のバランス
2. **段階的な変革管理**
   - 抵抗の少ない入り口からの導入

- 成功体験の共有と横展開
- 継続的なトレーニングと支援

3. **組織文化の考慮**
   - 既存の業務プロセスとの調和
   - 組織の価値観との整合性
   - 知識共有インセンティブの設計
4. **エコシステム思考**
   - 内部開発者コミュニティの育成
   - 外部エコシステムとの連携
   - オープンな協業モデルの採用

11.4.3 MCP導入の長期的成功メトリクス

MCPエコシステム導入の長期的成功を評価するための主要なメトリクス：

カテゴリ	メトリクス	測定方法
生産性	コンテキスト切替時間削減	タスク間遷移時間の測定
	情報検索時間削減	情報検索にかかる平均時間
	重複作業の削減	同一作業の繰り返し頻度
知識活用	知識再利用率の向上	既存資産の参照・引用回数
	意思決定速度の向上	決定プロセスの所要時間
	組織学習の加速	ベストプラクティス普及速度
コラボレーション	分野横断協業の増加	異なる専門チーム間の協働頻度
	アイデア創出の活性化	新規提案・発想の数と質
	知識伝達の効率化	新メンバーの習熟時間短縮
技術的健全性	システム利用率	アクティブユーザー数と使用頻度
	拡張性の維持	新機能追加の容易さと速度
	障害耐性	障害時の影響範囲と回復時間

> **失敗から学ぶ**: *「あるグローバル企業では、MCPシステムを一度に全社展開しようとして失敗しました。技術的には優れたシステムでしたが、各部門の既存ワークフローやツールセットを十分に考慮せず、また現場の声を取り入れる仕組みが不足していたのです。後に改めて、部門ごとの『MCPチャンピオン』を育成し、各部門の具体的課題に焦点を当てた段階的アプローチに切り替えたところ、導入が進み、大きな成果を上げ始めました。技術の優れさだけでなく、組織の受容性と変革管理が成功の鍵だったのです。」* - 変革管理コンサルタント

11.5 結びに代えて - MCPの未来を共に創る

Message Capacity Protocol（MCP）は、単なる技術プロトコルを超えた、知識とAIの新時代を形作る重要な要素です。コンテキストを中心に据えた設計思想は、分断された情報環境からシームレスな知識エコシステムへの移行を可能にします。

本書で紹介した概念、アーキテクチャ、実装例は、あくまでも現時点での最良の実践であり、MCPエコシステムは今後も急速に進化していくでしょう。重要なのは、この進化を受動的に待つのではなく、積極的に参加し、共に形作っていくことです。

組織の規模や業種を問わず、あらゆる知的作業において、MCPの原則とアプローチを適用することで、人間とAIのより自然で生産的な協働関係を構築できるでしょう。そして、その過程で得られた知見、開発したコネクタ、構築したワークフローを、広いコミュニティと共有することで、MCPエコシステム全体の発展に貢献できます。

未来のナレッジワークは、孤立したツールやデータの集合ではなく、有機的につながり、相互に高め合う統合されたエコシステムとなるでしょう。MCPはその重要な架け橋となります。

あなたとあなたの組織がこの旅に参加し、ナレッジワークの新たな可能性を開拓されることを願っています。

付録A: MCP仕様リファレンス

A.1 MCPプロトコル基本仕様

A.1.1 リクエスト構造

```
{
  "version": "1.1",
  "requestType": "query",
  "resource": "service://resource/path",
  "params": {
    "key1": "value1",
    "key2": "value2"
  },
  "context": {
    "sessionId": "session-123456",
    "metadata": {
      "key1": "value1"
    }
  }
}
```

必須フィールド

- `version`: MCPプロトコルバージョン（"1.0"または"1.1"）
- `requestType`: リクエストタイプ（"query", "update", "create", "execute", "stream"など）
- `resource`: リソースURI（形式: "scheme://path"）

オプションフィールド

- `params`: リクエスト固有のパラメータ
- `context`: リクエストコンテキスト
  - `sessionId`: セッションID（存在する場合）
  - `metadata`: その他のコンテキストメタデータ

A.1.2 レスポンス構造

```
{
  "version": "1.1",
  "status": 200,
  "requestId": "req-abcdef123456",
  "data": {
    "result": "value"
  },
  "error": null,
  "context": {
    "sessionId": "session-123456",
    "updated": "2025-04-09T10:30:45Z"
  }
}
```

必須フィールド

- `version`: MCPプロトコルバージョン
- `status`: ステータスコード（HTTP状態コードに準拠）
- `requestId`: リクエスト識別子

条件付きフィールド

- `data`: 成功時のレスポンスデータ（`status`が2xx系の場合）
- `error`: エラー情報（`status`が4xx系または5xx系の場合）
  - `code`: エラーコード
  - `message`: エラーメッセージ
  - `details`: 詳細情報（オプション）

オプションフィールド

- `context`: 更新されたコンテキスト情報
  - `sessionId`: セッションID
  - `updated`: 更新日時
  - `metadata`: その他のコンテキストメタデータ

A.1.3 標準リクエストタイプ

リクエストタイプ	説明	一般的な用途
`query`	情報検索リクエスト	リソースの読み取り、検索、情報取得
`update`	情報更新リクエスト	既存リソースの更新・修正
`create`	情報作成リクエスト	新規リソースの作成
`delete`	情報削除リクエスト	リソースの削除
`execute`	コマンド実行リクエスト	アクションの実行、処理の開始
`stream`	ストリーミングリクエスト	継続的なデータストリームの要求
`search`	検索リクエスト	複数リソースにまたがる検索
`analyze`	分析リクエスト	リソースの分析と洞察抽出

A.1.4 標準エラーコード

エラーコード	説明	HTTPステータスコード
`INVALID_REQUEST`	リクエスト形式が不正	400
`AUTHENTICATION_REQUIRED`	認証が必要	401
`PERMISSION_DENIED`	権限がない	403
`RESOURCE_NOT_FOUND`	リソースが見つからない	404
`METHOD_NOT_ALLOWED`	リクエストタイプがサポートされていない	405
`CONFLICT`	リソース競合	409
`RATE_LIMITED`	レート制限超過	429

エラーコード	説明	HTTPステータスコード
`INTERNAL_ERROR`	サーバー内部エラー	500
`SERVICE_UNAVAILABLE`	サービス利用不可	503
`CONNECTOR_ERROR`	コネクタ固有のエラー	502
`CONTEXT_ERROR`	コンテキスト管理エラー	500

A.2 リソースURI形式仕様

A.2.1 基本URI構造

```
scheme://authority/path/to/resource?param1=value1&param2=value2
```

- `scheme`: サービス種別（github, redmine, obsidian, file など）
- `authority`: サービス識別子（組織名、ユーザー名、サーバー名など）
- `path`: リソースパス
- `?param1=value1&...`: クエリパラメータ（オプション）

A.2.2 標準スキーム例

スキーム	URI例	説明
`github`	`github://owner/repo/path/to/file.md`	GitHubリポジトリ内のファイル
`redmine`	`redmine://project-id/issues/123`	Redmineプロジェクトの課題
`obsidian`	`obsidian://vault/folder/note.md`	Obsidianボールト内のノート
`file`	`file://path/to/document.txt`	ローカルファイルシステム上のファイル
`mcp`	`mcp://assistant`	MCPネイティブサービス

付録B: トラブルシューティングガイド

B.1 一般的な問題と解決策

B.1.1 接続問題

症状	考えられる原因	解決策
MCPサーバーに接続できない	ネットワーク構成の問題	ファイアウォール設定の確認、ネットワーク経路のトレース
	サーバー停止	サーバーステータスの確認、ログの調査
	TLS/SSL設定の不一致	証明書と暗号化設定の確認
WebSocket接続が切断される	タイムアウト設定	キープアライブ間隔の調整
	サーバーリソース不足	サーバーリソースの増強、接続数制限の調整

B.1.2 認証・認可問題

症状	考えられる原因	解決策
認証エラー	無効なトークンや証明書	認証情報の更新、再取得
	トークン有効期限切れ	認証フローの再実行
	クロックドリフト	サーバー時刻の同期確認
アクセス権限エラー	不十分な権限設定	必要な権限の付与、ロールの確認
	権限キャッシュの問題	権限キャッシュのクリア、再ログイン

B.1.3 パフォーマンス問題

症状	考えられる原因	解決策
レスポンスが遅い	サーバー負荷	サーバーリソースのモニタリングと増強
	非効率なクエリ	クエリ最適化、インデックス作成
	ネットワークレイテンシ	CDNの利用、エッジロケーションの活用
メモリ使用量の増加	リソースリーク	メモリプロファイリング、リソース解放の確認
	大きすぎるペイロード	ペイロードサイズの制限、ページネーションの実装

B.2 コネクタ固有の問題

B.2.1 GitHub コネクタ

症状	考えられる原因	解決策
APIレート制限エラー	短時間での過剰なリクエスト	レート制限対応の実装、キャッシュの活用
	認証なしリクエスト	適切な認証トークンの使用
リポジトリへのアクセスエラー	権限不足	適切なリポジトリ権限の付与
	プライベートリポジトリ	認証トークンとスコープの確認

B.2.2 Obsidian コネクタ

症状	考えられる原因	解決策
ボールトにアクセスできない	ファイルパスの問題	絶対パスと相対パスの確認
	権限の問題	ファイルシステム権限の確認
Obsidian Publishへの接続エラー	APIキーの問題	APIキーの更新と権限確認
	CORS制限	CORS設定の調整

B.3 サーバー運用問題

B.3.1 スケーリングの問題

症状	考えられる原因	解決策
負荷増大時のクラッシュ	リソース不足	自動スケーリングの設定、負荷分散の実装
	コネクションプールの枯渇	プール設定の最適化、接続管理の改善
コンテキスト管理の問題	共有ストレージの競合	分散ロックの実装、アトミック操作の保証
	セッションの整合性	セッションアフィニティの設定

B.3.2 監視とアラート

MCP環境の効果的な監視のために注目すべき主要メトリクス：

カテゴリ	メトリクス	警告閾値の目安
可用性	サーバー稼働時間	< 99.9%
	エンドポイント応答率	< 99.5%
性能	リクエスト平均応答時間	> 500ms
	99パーセンタイル応答時間	> 2000ms
	WebSocketドロップ率	> 1%
リソース	CPU使用率	> 80%
	メモリ使用率	> 85%
	ディスクI/O待ち時間	> 100ms

カテゴリ	メトリクス	警告閾値の目安
トラフィック	秒間リクエスト数	サーバー容量による
	帯域幅使用量	ネットワーク容量の70%以上
エラー	4xx/5xxレスポンス率	> 5%
	認証失敗率	> 10%
コネクタ	コネクタエラー率	> 5%
	コネクタ応答時間	> 1000ms

付録C: サンプルコード一覧

C.1 Python/FastAPI MCPサーバー 主要コンポーネント

```python
# コネクタマネージャーの実装サンプル
class ConnectorManager:
    def __init__(self):
        self.connectors = {}

    def register_connector(self, connector_id, connector):
        self.connectors[connector_id] = connector

    def get_connector(self, connector_id):
        return self.connectors.get(connector_id)

    def get_connector_for_resource(self, resource_uri):
        if "://" in resource_uri:
            protocol = resource_uri.split("://")[0]
            return self.connectors.get(protocol)
        return None

# コンテキストマネージャーの実装サンプル
class ContextManager:
    def __init__(self):
        self.contexts = {}

    def get_context(self, session_id):
        return self.contexts.get(session_id)

    def create_context(self, initial_data=None):
        session_id = str(uuid.uuid4())
        context = {
            "sessionId": session_id,
            "created": datetime.now().isoformat(),
            "lastAccessed": datetime.now().isoformat(),
            "data": initial_data or {}
        }
        self.contexts[session_id] = context
        return context

    def update_context(self, session_id, updates):
        if session_id not in self.contexts:
            return None

        context = self.contexts[session_id]
        context["data"].update(updates)
        context["lastAccessed"] = datetime.now().isoformat()

        return context
```

C.2 React/TypeScript MCPクライアント サンプル

```
// MCPクライアントフックの使用例
import React, { useState, useEffect } from 'react';
import { useMCP } from '../hooks/useMCP';

function MCPExampleComponent() {
  const { client, isConnected, query } = useMCP({
    baseUrl: 'https://mcp-server.example.com',
    auth: { type: 'bearer', token: 'YOUR_AUTH_TOKEN' },
    autoConnect: true
  });

  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState(null);

  const fetchData = async () => {
    setLoading(true);
    setError(null);

    try {
      const response = await query({
        requestType: 'query',
        resource: 'github://organization/repository/README.md'
      });

      setData(response.data);
    } catch (err) {
      setError(err.message);
    } finally {
      setLoading(false);
    }
  };

  useEffect(() => {
    if (isConnected) {
      fetchData();
    }
  }, [isConnected]);

  return (
    <div className="mcp-example">
      <h2>MCP Data Example</h2>

      <div className="connection-status">
        Status: {isConnected ? 'Connected' : 'Disconnected'}
      </div>

      {loading && <div className="loading">Loading data...</div>}

      {error && (
        <div className="error">
          Error: {error}
          <button onClick={fetchData}>Retry</button>
        </div>
      )}
```

```
      {data && (
        <div className="data-display">
          <h3>Repository README</h3>
          <pre>{data.content}</pre>
        </div>
      )}
    </div>
  );
}
```

C.3 C# MCPコネクタ サンプル

```
/// <summary>
/// GitHub MCPコネクタの実装例
/// </summary>
public class GitHubConnector : IConnector
{
    private readonly ILogger<GitHubConnector> _logger;
    private readonly HttpClient _httpClient;
    private readonly string _apiBaseUrl;
    private string? _authToken;

    public string Id => "github";
    public string Name => "GitHub Connector";
    public string Version => "1.0.0";
    public string[] SupportedSchemes => new[] { "github" };

    public GitHubConnector(IHttpClientFactory httpClientFactory,
ILogger<GitHubConnector> logger, IConfiguration configuration)
    {
        _logger = logger;
        _httpClient = httpClientFactory.CreateClient("GitHub");
        _apiBaseUrl = configuration["GitHub:ApiBaseUrl"] ?? "https://api.github.com";
    }

    public Task InitializeAsync()
    {
        _logger.LogInformation("GitHub connector initialized");
        return Task.CompletedTask;
    }

    public async Task<ConnectorResponse> HandleRequestAsync(ConnectorRequest request)
    {
        try
        {
            // 認証トークンの取得
            _authToken = GetAuthToken(request.Context);

            if (string.IsNullOrEmpty(_authToken))
            {
                return ConnectorResponse.Error("AUTHENTICATION_REQUIRED", "GitHub API
token is required");
            }

            // リクエストタイプに応じた処理
```

```
            switch (request.RequestType)
            {
                case "query":
                    return await HandleQueryAsync(request);
                case "update":
                    return await HandleUpdateAsync(request);
                case "create":
                    return await HandleCreateAsync(request);
                default:
                    return ConnectorResponse.Error("INVALID_REQUEST_TYPE",
$"Unsupported request type: {request.RequestType}");
            }
        }
        catch (HttpRequestException ex)
        {
            _logger.LogError(ex, "GitHub API request error");
            return ConnectorResponse.Error("API_ERROR", ex.Message);
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "GitHub connector error");
            return ConnectorResponse.Error("INTERNAL_ERROR", ex.Message);
        }
    }

    private async Task<ConnectorResponse> HandleQueryAsync(ConnectorRequest request)
    {
        var resourceInfo = request.Resource;
        var owner = resourceInfo.Properties["owner"]?.ToString();
        var repo = resourceInfo.Properties["repo"]?.ToString();
        var path = resourceInfo.Properties["path"]?.ToString();

        if (string.IsNullOrEmpty(owner) || string.IsNullOrEmpty(repo))
        {
            return ConnectorResponse.Error("INVALID_RESOURCE", "Owner and repository
are required");
        }

        // パスが指定されている場合はファイル内容を取得
        if (!string.IsNullOrEmpty(path))
        {
            var fileContent = await GetFileContentAsync(owner, repo, path);
            return ConnectorResponse.Success(fileContent);
        }

        // リポジトリ情報の取得
        var repoInfo = await GetRepositoryInfoAsync(owner, repo);
        return ConnectorResponse.Success(repoInfo);
    }

    // 他のハンドラーメソッド...

    private string? GetAuthToken(ConnectorContext context)
    {
        // コンテキストからトークンを取得
        if (context.Data.TryGetValue("github_token", out var token) && token is string
tokenString)
```

```
        {
            return tokenString;
        }

        // ユーザー情報からトークンを取得（実際の実装ではより安全な方法を使用）
        return context.User?.Id;
    }

    // API呼び出しヘルパーメソッド...

    public ResourceInfo ParseResourceUri(string resourceUri)
    {
        // リソースURIの解析
        // github://owner/repo/path/to/file

        if (!resourceUri.StartsWith("github://"))
        {
            throw new ArgumentException($"Invalid GitHub resource URI: {resourceUri}");
        }

        var path = resourceUri.Substring(9); // "github://" を削除
        var parts = path.Split('/');

        if (parts.Length < 2)
        {
            throw new ArgumentException($"Invalid GitHub resource URI format:
{resourceUri}");
        }

        var resourceInfo = new ResourceInfo
        {
            Uri = resourceUri,
            Scheme = "github",
            Path = parts.Length > 2 ? string.Join("/", parts.Skip(2)) : ""
        };

        resourceInfo.Properties["owner"] = parts[0];
        resourceInfo.Properties["repo"] = parts[1];

        if (parts.Length > 2)
        {
            resourceInfo.Properties["path"] = string.Join("/", parts.Skip(2));
        }

        return resourceInfo;
    }

    public Task DisposeAsync()
    {
        _logger.LogInformation("GitHub connector disposed");
        return Task.CompletedTask;
    }

    public IAsyncEnumerable<object> HandleStreamingAsync(ConnectorRequest request)
    {
        // ストリーミング処理の実装
        throw new NotImplementedException("GitHub streaming not implemented");
```

```
    }
}
```

C.4 MCPプラグイン サンプル

```python
# Pythonプラグイン実装サンプル
class SecurityAuditPlugin:
    """セキュリティ監査プラグイン（デモ用）"""

    def __init__(self):
        self.id = "security-audit-plugin"
        self.name = "Security Audit Plugin"
        self.version = "1.0.0"
        self.sensitive_patterns = [
            r'\bpassword\b',
            r'\bsecret\b',
            r'\bapi[_-]?key\b',
            r'\baccess[_-]?token\b',
            r'\bcredential\b'
        ]

    async def initialize(self):
        """初期化処理"""
        logging.info("Security Audit plugin initialized")

    async def on_request(self, request, context):
        """リクエスト処理前のフック"""
        # 監査ログ
        log_entry = {
            "timestamp": datetime.now().isoformat(),
            "user": context.get("user", {}).get("username", "anonymous"),
            "resource": request["resource"],
            "requestType": request["requestType"]
        }

        logging.info(f"SecurityAudit: {json.dumps(log_entry)}")

        # センシティブ情報のチェック
        if self._contains_sensitive_data(request):
            logging.warning(f"SecurityAudit: Potential sensitive information detected
in request")

            # コンテキストにフラグを追加
            if context and "data" in context:
                context["data"]["security_check"] = {
                    "sensitive_data_detected": True,
                    "timestamp": datetime.now().isoformat()
                }

        return request

    async def on_response(self, response, context):
        """レスポンス処理前のフック"""
        # センシティブ情報が含まれていないか確認
        if self._contains_sensitive_data(response.get("data", {})):
```

```
            logging.warning(f"SecurityAudit: Potential sensitive information detected
in response")

            # レスポンスにセキュリティ警告を追加
            if "metadata" not in response:
                response["metadata"] = {}

            response["metadata"]["security_warning"] = {
                "message": "Potential sensitive information detected",
                "timestamp": datetime.now().isoformat()
            }

        return response

    async def shutdown(self):
        """シャットダウン処理"""
        logging.info("Security Audit plugin shutdown")

    def _contains_sensitive_data(self, data):
        """センシティブ情報が含まれているか確認"""
        if not data:
            return False

        # 文字列化してパターンチェック
        data_str = json.dumps(data)

        for pattern in self.sensitive_patterns:
            if re.search(pattern, data_str, re.IGNORECASE):
                return True

        return False
```