**# 法令知識ベース連携モジュール 詳細設計書**

**## 目次**

1. [モジュール概要](#1-モジュール概要)

2. [システム構成](#2-システム構成)

3. [ファイル構成と役割](#3-ファイル構成と役割)

4. [主要機能の動作](#4-主要機能の動作)

5. [APIエンドポイント仕様](#5-apiエンドポイント仕様)

6. [データフロー](#6-データフロー)

7. [入出力仕様](#7-入出力仕様)

8. [依存関係](#8-依存関係)

9. [実行環境](#9-実行環境)

10. [モジュール結合時の注意点](#10-モジュール結合時の注意点)

11. [トラブルシューティング](#11-トラブルシューティング)

---

**## 1. モジュール概要**

**### 1.1 目的**

本モジュールは、e-Gov法令APIから日本法の条文を取得し、パース（解析）してデータベースに格納することで、他のモジュール（対話理解モジュール、論争解析モジュール）が法令情報を利用できるようにするための基盤システムです。

**### 1.2 主要機能**

1. **\*\*法令データ取得\*\***: e-Gov APIから法令のXMLデータを取得

2. **\*\*XML解析\*\***: XML形式の法令データを構造化されたJSON形式に変換

3. **\*\*条文要約\*\***: AI（Gemini）を使用した条文の要約生成

4. **\*\*論点抽出\*\***: 複数条文から法的論点を自動抽出

5. **\*\*全文検索\*\***: 法令・条文の検索機能

6. **\*\*キャッシュ機能\*\***: Redisによる高速化

**### 1.3 システム全体における位置**

```

┌─────────────────────────────────────────────────┐

│ Law Chat Bot 全体システム │

├─────────────────────────────────────────────────┤

│ WP2-1 対話理解モジュール │

│ ├─ ユーザー入力の理解 │

│ └─ ─────────────┬────────────── │

│ ↓ API呼び出し │

│ WP2-3 法令知識ベース連携モジュール ← \*\*こちら\*\* │

│ ├─ 法令データ取得 │

│ ├─ 条文要約 │

│ └─ ─────────────┬────────────── │

│ ↓ API呼び出し │

│ WP2-2 論争解析モジュール │

│ └─ 論争の解析と対立点抽出 │

└─────────────────────────────────────────────────┘

```

---

**## 2. システム構成**

**### 2.1 アーキテクチャ図**

```

┌─────────────────────────────────────────────────────┐

│ 外部システム │

├──────────┬─────────────┬──────────────────────────┤

│ e-Gov API│ Gemini API │ PostgreSQL / Redis │

│ (XML取得)│ (AI処理) │ (データ保存) │

└──────────┴─────────────┴──────────────────────────┘

↓ ↓ ↓

┌─────────────────────────────────────────────────────┐

│ 法令知識ベース連携モジュール │

├─────────────────────────────────────────────────────┤

│ FastAPI Web Server (Port 8083) │

│ ├─ /health ヘルスチェック │

│ ├─ /laws/list 法令リスト取得 │

│ ├─ /laws/{id} 法令詳細取得 │

│ ├─ /laws/{id}/articles/{no} 条文取得 │

│ ├─ /summarize 条文要約 │

│ └─ /extract\_topics 論点抽出 │

└─────────────────────────────────────────────────────┘

↓

┌─────────────────────────────────────────────────────┐

│ 上流モジュール │

│ ┌──────────────┬──────────────┬──────────────┐ │

│ │ 対話理解 │ 論争解析 │ その他 │ │

│ └──────────────┴──────────────┴──────────────┘ │

└─────────────────────────────────────────────────────┘

```

**### 2.2 データベース設計**

```

┌─────────────────────────────────────────────────┐

│ PostgreSQL データベース │

├─────────────────────────────────────────────────┤

│ legal\_refs (法令マスタ) │

│ ├─ law\_id (PK): 法令ID │

│ ├─ title: 法令名 │

│ ├─ law\_no: 法令番号 │

│ ├─ raw\_xml: 元のXMLデータ │

│ └─ updated\_at: 更新日時 │

│ │

│ articles (条文) │

│ ├─ article\_id (PK): 条文ID │

│ ├─ law\_id (FK): 法令ID │

│ ├─ article\_no: 条番号 │

│ ├─ text: 条文本文 │

│ └─ parsed\_json: 構造化データ │

│ │

│ article\_embeddings (埋め込みベクトル) │

│ ├─ embedding\_id (PK) │

│ ├─ article\_id (FK) │

│ └─ embedding: ベクトルデータ │

│ │

│ sync\_log (同期ログ) │

│ ├─ sync\_id (PK) │

│ ├─ sync\_type: 同期種別 │

│ ├─ status: 状態 │

│ └─ finished\_at: 終了時刻 │

└─────────────────────────────────────────────────┘

```

---

**## 3. ファイル構成と役割**

**### 3.1 ディレクトリ構成**

```

WP2-3 法令知識ベース連携/

├── app/ # メインアプリケーション

│ ├── \_\_init\_\_.py # パッケージ初期化

│ ├── main.py # FastAPI サーバー起動

│ ├── config.py # 設定管理（環境変数読み込み）

│ ├── logger.py # ログ設定

│ ├── schemas.py # API リクエスト/レスポンス定義

│ │

│ ├── api/ # API ルーター

│ │ ├── \_\_init\_\_.py

│ │ ├── laws.py # 法令API エンドポイント

│ │ └── middleware.py # レート制限・エラーハンドリング

│ │

│ ├── services/ # ビジネスロジック

│ │ ├── \_\_init\_\_.py

│ │ ├── egov\_client.py # e-Gov API 連携クライアント

│ │ ├── xml\_parser.py # XML 解析処理

│ │ ├── summarizer.py # 条文要約サービス

│ │ ├── topic\_extractor.py # 論点抽出サービス

│ │ └── cache\_service.py # Redis キャッシュ管理

│ │

│ ├── clients/ # 外部API クライアント

│ │ ├── \_\_init\_\_.py

│ │ └── gemini\_client.py # Google Gemini API 連携

│ │

│ ├── models/ # データベースモデル

│ │ ├── \_\_init\_\_.py

│ │ └── models.py # SQLAlchemy ORM 定義

│ │

│ ├── scripts/ # バッチスクリプト

│ │ ├── \_\_init\_\_.py

│ │ └── sync\_egov.py # e-Gov データ同期処理

│ │

│ └── utils/ # ユーティリティ

│ ├── \_\_init\_\_.py

│ └── error\_mapping.py # エラーハンドリング

│

├── tests/ # テスト

│ ├── \_\_init\_\_.py

│ ├── test\_parser.py # XML パーサーテスト

│ └── test\_api.py # API エンドポイントテスト

│

├── prompt\_templates/ # AIプロンプトテンプレート

│ ├── summarize\_ja.txt # 要約プロンプト

│ └── extract\_topics\_ja.txt # 論点抽出プロンプト

│

├── sample\_xml/ # サンプルXML

│ └── law\_civil\_001.xml

│

├── Dockerfile # Docker イメージ定義

├── requirements.txt # Python依存パッケージ

├── README.md # プロジェクトドキュメント

├── openapi.yaml # OpenAPI 仕様書

└── .env # 環境変数（要作成）

```

**### 3.2 主要ファイルの役割**

**#### 3.2.1** `app/main.py`

**\*\*役割\*\***: FastAPIアプリケーションのエントリーポイント

**\*\*主要処理\*\***:

- FastAPIアプリケーションの初期化

- ミドルウェア（CORS、レート制限、エラーハンドリング）の設定

- API ルーターの登録

- ヘルスチェックエンドポイントの提供

**\*\*機能\*\***:

```python

@app.get("/") # ヘルスチェック

@app.get("/health") # 詳細ヘルスチェック

@app.get("/v1/models") # モデル情報取得

```

**#### 3.2.2** `app/api/laws.py`

**\*\*役割\*\***: 法令関連APIエンドポイントの実装

**\*\*提供するエンドポイント\*\***:

- `GET /laws/list` - 法令リスト取得

- `GET /laws/{law\_id}` - 法令詳細取得

- `GET /laws/{law\_id}/articles/{article\_no}` - 条文取得

- `POST /laws/{law\_id}/articles/{article\_no}/summarize` - 条文要約

- `POST /laws/extract\_topics` - 論点抽出

- `POST /laws/search` - 法令検索

**#### 3.2.3** `app/services/egov\_client.py`

**\*\*役割\*\***: e-Gov APIとの通信処理

**\*\*主要関数\*\***:

- `get\_law\_list()` - 法令リスト取得

- `get\_law\_details()` - 法令詳細取得

- `get\_article()` - 条文取得

**#### 3.2.4** `app/services/xml\_parser.py`

**\*\*役割\*\***: XML形式の法令データを構造化データに変換

**\*\*主要関数\*\***:

- `parse\_xml()` - XMLをパースして辞書形式に変換

- `\_extract\_law\_info()` - 法令基本情報を抽出

- `\_extract\_articles()` - 条文データを抽出

- `normalize\_article\_no()` - 条番号を正規化

**#### 3.2.5** `app/services/summarizer.py`

**\*\*役割\*\***: 条文の要約処理

**\*\*主要関数\*\***:

- `summarize\_article()` - 条文を要約

- `\_apply\_local\_reduction()` - ローカル短縮処理

- `\_extract\_paragraphs()` - 段落抽出

- `\_format\_summary()` - 要約結果を整形

**#### 3.2.6** `app/services/topic\_extractor.py`

**\*\*役割\*\***: 法的論点の抽出処理

**\*\*主要関数\*\***:

- `extract\_topics()` - 論点抽出

- `\_format\_topics()` - 結果を整形

- `\_extract\_key\_phrases()` - 重要語句抽出

**#### 3.2.7** `app/clients/gemini\_client.py`

**\*\*役割\*\***: Google Gemini APIとの連携

**\*\*主要関数\*\***:

- `generate\_summary()` - 要約生成

- `extract\_topics()` - 論点抽出

- `\_call\_api()` - API呼び出し

**#### 3.2.8** `app/services/cache\_service.py`

**\*\*役割\*\***: Redis キャッシュ管理

**\*\*主要関数\*\***:

- `get()` - キャッシュから取得

- `set()` - キャッシュに保存

- `delete()` - キャッシュ削除

- `make\_key()` - キャッシュキー生成

---

**## 4. 主要機能の動作**

**### 4.1 法令リスト取得フロー**

```

1. ユーザー → GET /laws/list?page=1&per\_page=20

2. laws.py (list\_laws) がリクエスト受信

3. egov\_client.py の get\_law\_list() を呼び出し

4. e-Gov API に HTTP リクエスト送信

5. e-Gov API から JSON レスポンス受信

6. レスポンスを整形して返却

```

**\*\*具体例\*\***:

```python

# リクエスト

GET http://localhost:8083/laws/list?page=1&per\_page=20

# レスポンス

{

"laws": [

{

"law\_id": "CIVIL\_LAW\_001",

"title": "民法",

"law\_no": "明治29年法律第89号",

"enact\_date": "1896-04-27T00:00:00"

}

],

"total": 1,

"page": 1,

"per\_page": 20

}

```

**### 4.2 条文要約フロー**

```

1. ユーザー → POST /laws/{id}/articles/{no}/summarize

2. laws.py (summarize\_article) がリクエスト受信

3. egov\_client.py で条文本文を取得

4. summarizer.py でローカル短縮処理

5. gemini\_client.py でGemini API呼び出し

6. 要約結果を整形して返却

```

**\*\*具体例\*\***:

```python

# リクエスト

POST http://localhost:8083/laws/CIVIL\_LAW\_001/articles/第1条/summarize

{

"max\_length": 200,

"style": "plain"

}

# レスポンス

{

"summary\_text": "私権は公共の福祉に適合する必要がある...",

"original\_reference": {

"law\_id": "CIVIL\_LAW\_001",

"article\_no": "第1条"

},

"citations": [],

"style": "plain",

"word\_count": 150

}

```

**### 4.3 XML解析フロー**

```

1. e-Gov API から XML データを取得

2. xml\_parser.py の parse\_xml() を呼び出し

3. XML を DOM ツリーに変換

4. \_extract\_law\_info() で法令情報抽出

5. \_extract\_articles() で条文データ抽出

6. 構造化された JSON 形式に変換

```

**\*\*XML 例\*\***:

```xml

<Law>

<Article number="第1条" heading="私権の内容">

<Paragraph>

私権は、公共の福祉に適合しなければならない。

</Paragraph>

</Article>

</Law>

```

**\*\*JSON 結果\*\***:

```json

{

"law\_id": "CIVIL\_LAW\_001",

"articles": [

{

"article\_no": "第1条",

"heading": "私権の内容",

"text": "私権は、公共の福祉に適合しなければならない。"

}

]

}

```

---

**## 5. APIエンドポイント仕様**

**### 5.1 一覧表**

| メソッド | エンドポイント | 機能 | 入力 | 出力 |

|---------|---------------|------|------|------|

| GET | `/health` | ヘルスチェック | なし | システム状態 |

| GET | `/laws/list` | 法令リスト取得 | page, per\_page | 法令リスト |

| GET | `/laws/{law\_id}` | 法令詳細取得 | law\_id | 法令詳細 + 条文一覧 |

| GET | `/laws/{law\_id}/articles/{article\_no}` | 条文取得 | law\_id, article\_no | 条文データ |

| POST | `/laws/{law\_id}/articles/{article\_no}/summarize` | 条文要約 | law\_id, article\_no, options | 要約結果 |

| POST | `/laws/extract\_topics` | 論点抽出 | texts[], mode | 論点リスト |

| POST | `/laws/search` | 法令検索 | keyword | 検索結果 |

**### 5.2 詳細仕様**

**#### 5.2.1** `GET /laws/list`

**\*\*入力\*\***:

```http

GET /laws/list?law\_type=法律&page=1&per\_page=20

```

**\*\*出力\*\***:

```json

{

"laws": [

{

"law\_id": "CIVIL\_LAW\_001",

"title": "民法",

"law\_no": "明治29年法律第89号",

"law\_type": "法律",

"enact\_date": "1896-04-27T00:00:00"

}

],

"total": 100,

"page": 1,

"per\_page": 20

}

```

**#### 5.2.2** `POST /laws/{law\_id}/articles/{article\_no}/summarize`

**\*\*入力\*\***:

```json

{

"max\_length": 200,

"style": "plain" // "plain" | "legal\_summary" | "for\_layperson"

}

```

**\*\*出力\*\***:

```json

{

"summary\_text": "要約された条文の内容...",

"original\_reference": {

"law\_id": "CIVIL\_LAW\_001",

"article\_no": "第1条"

},

"citations": [],

"style": "plain",

"word\_count": 150

}

```

---

**## 6. データフロー**

**### 6.1 全体データフロー図**

```

外部システム 本モジュール 上流モジュール

────────────────────────────────────────────────────────────

│

[e-Gov API] ──────────▶│ (egov\_client.py)

(XML 法令データ) │ ↓

│ [XML解析]

│ (xml\_parser.py)

│ ↓

│ [PostgreSQL]

[Gemini API] ──────────▶│ (gemini\_client.py)

(AI処理) │ ↓

│ [要約・論点抽出]

│ ↓

│ [API エンドポイント]

│ ↓

└───────────▶ [対話理解モジュール]

[論争解析モジュール]

```

**### 6.2 具体例: 条文要約リクエストの流れ**

```

1. [ユーザー]

POST /laws/CIVIL\_001/articles/第1条/summarize

↓

2. [FastAPI Router]

laws.py の summarize\_article() 関数が呼び出される

↓

3. [サービス層]

EGOvClient().get\_article()

→ e-Gov API から条文を取得

↓

4. [要約処理]

ArticleSummarizer().summarize\_article()

→ ローカル短縮処理

↓

5. [AI処理]

GeminiClient().generate\_summary()

→ Gemini API で要約生成

↓

6. [結果返却]

整形された要約をJSON形式で返却

```

---

**## 7. 入出力仕様**

**### 7.1 リクエスト形式**

全てのAPIリクエストはHTTP/HTTPSプロトコルを使用します。

**\*\*ベースURL\*\***: `http://localhost:8083` (開発環境)

**\*\*認証\*\***: 現在は不要（将来はAPI Key認証を追加予定）

**\*\*レスポンス形式\*\***: JSON

**### 7.2 エラーレスポンス**

```json

{

"detail": "エラーメッセージ",

"error": "ERROR\_CODE"

}

```

**\*\*HTTPステータスコード\*\***:

- `200`: 成功

- `404`: リソースが見つからない

- `429`: レート制限超過

- `500`: サーバーエラー

**### 7.3 具体例: 条文要約API**

**\*\*リクエスト例\*\***:

```bash

curl -X POST \

"http://localhost:8083/laws/CIVIL\_LAW\_001/articles/第1条/summarize" \

-H "Content-Type: application/json" \

-d '{

"max\_length": 200,

"style": "plain"

}'

```

**\*\*レスポンス例\*\***:

```json

{

"summary\_text": "私権は公共の福祉に適合する必要がある。",

"original\_reference": {

"law\_id": "CIVIL\_LAW\_001",

"article\_no": "第1条"

},

"citations": [],

"style": "plain",

"word\_count": 25

}

```

---

**## 8. 依存関係**

**### 8.1 外部システムへの依存**

- **\*\*e-Gov API\*\***: 法令データ取得

- **\*\*Google Gemini API\*\***: AI処理（要約・論点抽出）

- **\*\*PostgreSQL\*\***: データベース

- **\*\*Redis\*\***: キャッシュ（オプション）

**### 8.2 Pythonパッケージ依存**

**\*\*主要ライブラリ\*\***:

```

fastapi==0.115.0 # Web フレームワーク

uvicorn==0.30.6 # ASGI サーバー

pydantic==2.9.2 # データ検証

httpx==0.27.2 # HTTP クライアント

sqlalchemy==2.0.29 # ORM

redis==5.0.5 # キャッシュ

transformers==4.36.0 # ML ライブラリ

```

**\*\*完全な依存関係は** `requirements.txt` **を参照\*\***

**### 8.3 モジュール間の依存関係**

```

app/main.py

├── app/api/laws.py

│ ├── app/services/egov\_client.py

│ ├── app/services/summarizer.py

│ └── app/services/topic\_extractor.py

└── app/api/middleware.py

app/services/egov\_client.py

└── app/services/xml\_parser.py

app/services/summarizer.py

└── app/clients/gemini\_client.py

app/services/topic\_extractor.py

└── app/clients/gemini\_client.py

```

---

**## 9. 実行環境**

**### 9.1 必要な環境**

- **\*\*OS\*\***: Linux / macOS / Windows

- **\*\*Python\*\***: 3.10以上

- **\*\*メモリ\*\***: 最低2GB、推奨4GB以上

- **\*\*ディスク\*\***: 最低1GB（データベース除く）

**### 9.2 セットアップ手順**

**#### Step 1: 依存パッケージのインストール**

```bash

cd "WP2-3 法令知識ベース連携"

pip install -r requirements.txt

```

**#### Step 2: 環境変数の設定**

```bash

cp env.example .env

# .env ファイルを編集

```

**\*\*必須設定\*\***:

```env

GEMINI\_API\_KEY=your\_key\_here # Gemini API キー

E\_GOV\_API\_KEY=your\_key\_here # e-Gov API キー

DATABASE\_URL=postgresql://... # データベース接続URL

```

**#### Step 3: サーバー起動**

```bash

# 方法1: Pythonモジュールとして起動

python -m app.main

# 方法2: uvicornで起動

uvicorn app.main:app --reload --port 8083

```

**#### Step 4: 動作確認**

```bash

# ヘルスチェック

curl http://localhost:8083/health

# API ドキュメント

open http://localhost:8083/docs

```

**### 9.3 Docker での実行**

```bash

# イメージビルド

docker build -t law-kb-module .

# コンテナ起動

docker run -p 8083:8083 --env-file .env law-kb-module

```

---

**## 10. モジュール結合時の注意点**

**### 10.1 他のモジュールとの連携**

**#### 10.1.1 対話理解モジュール（WP2-1）との連携**

```

対話理解モジュール側:

- ユーザーの「民法第1条を要約して」というリクエストを解析

- このモジュールの API を呼び出し

本モジュール側:

- POST /laws/extract\_topics エンドポイント

- POST /laws/{id}/articles/{no}/summarize エンドポイント

```

**\*\*注意点\*\***:

- レート制限（60リクエスト/分）を遵守

- エラーハンドリングを実装

- タイムアウト処理（30秒）を設定

**#### 10.1.2 論争解析モジュール（WP2-2）との連携**

```

論争解析モジュール側:

- 論争に関連する条文を取得

- 論点を抽出して使用

本モジュール側:

- GET /laws/{id}/articles/{no} エンドポイント

- POST /laws/extract\_topics エンドポイント

```

**\*\*注意点\*\***:

- 大容量データの処理（複数条文の同時取得）

- キャッシュを活用してパフォーマンス向上

**### 10.2 API呼び出し例**

**#### 10.2.1 Python (httpx)**

```python

import httpx

# 条文要約API呼び出し

async with httpx.AsyncClient() as client:

response = await client.post(

"http://localhost:8083/laws/CIVIL\_001/articles/第1条/summarize",

json={"max\_length": 200, "style": "plain"},

timeout=30.0

)

result = response.json()

```

**#### 10.2.2 cURL**

```bash

curl -X POST \

"http://localhost:8083/laws/CIVIL\_001/articles/第1条/summarize" \

-H "Content-Type: application/json" \

-d '{"max\_length": 200, "style": "plain"}'

```

**### 10.3 共通注意事項**

**#### 10.3.1 エラーハンドリング**

```python

try:

response = await client.post(url, json=data)

response.raise\_for\_status()

return response.json()

except httpx.HTTPStatusError as e:

if e.response.status\_code == 404:

# リソースが見つからない

pass

elif e.response.status\_code == 429:

# レート制限超過 → リトライ

await asyncio.sleep(1)

return await call\_api()

except Exception as e:

# その他のエラー

logger.error(f"API call failed: {e}")

```

**#### 10.3.2 タイムアウト設定**

```python

# 推奨タイムアウト: 30秒

async with httpx.AsyncClient(timeout=30.0) as client:

response = await client.post(url, json=data)

```

**#### 10.3.3 レート制限への対応**

```python

# レート制限: 60リクエスト/分

import asyncio

from datetime import datetime, timedelta

request\_times = []

async def call\_with\_rate\_limit(url, data):

now = datetime.now()

# 1分以内のリクエストをフィルタ

recent = [t for t in request\_times if now - t < timedelta(minutes=1)]

if len(recent) >= 60:

# 待機

await asyncio.sleep(1)

request\_times.append(now)

response = await client.post(url, json=data)

return response

```

**### 10.4 パフォーマンス最適化**

**#### 10.4.1 キャッシュ活用**

```python

# 同一リクエストはキャッシュから取得

cache\_key = f"law\_{law\_id}\_article\_{article\_no}"

cached\_result = cache\_service.get(cache\_key)

if cached\_result:

return cached\_result

# なければAPI呼び出し

result = await call\_api()

cache\_service.set(cache\_key, result, ttl=86400)

```

**#### 10.4.2 並列処理**

```python

import asyncio

# 複数条文を並列取得

async def get\_multiple\_articles(law\_id, article\_nos):

tasks = [

client.get(f"/laws/{law\_id}/articles/{no}")

for no in article\_nos

]

results = await asyncio.gather(\*tasks)

return results

```

---

**## 11. トラブルシューティング**

**### 11.1 よくある問題**

**#### 問題1: インポートエラー**

```

ModuleNotFoundError: No module named 'app'

```

**\*\*解決方法\*\***:

```bash

# PYTHONPATH を設定

export PYTHONPATH="${PYTHONPATH}:$(pwd)"

# または

cd "WP2-3 法令知識ベース連携"

python -m app.main

```

**#### 問題2: ポートが既に使用されている**

```

Address already in use

```

**\*\*解決方法\*\***:

```bash

# 別のポートを使用

uvicorn app.main:app --reload --port 8084

```

**#### 問題3: Gemini API キー未設定**

```

Warning: GEMINI\_API\_KEY is not set. Mock mode will be used.

```

**\*\*解決方法\*\***:

- `.env` ファイルに `GEMINI\_API\_KEY=your\_key` を追加

- またはモックモードで動作（機能制限あり）

**#### 問題4: データベース接続エラー**

```

OperationalError: could not connect to database

```

**\*\*解決方法\*\***:

1. PostgreSQL が起動しているか確認

2. `DATABASE\_URL` の接続情報を確認

3. データベースが存在するか確認

**### 11.2 デバッグ方法**

**#### ログレベルの変更**

```env

# .env ファイル

LOG\_LEVEL=DEBUG

```

**#### 詳細なエラー情報を取得**

```python

import logging

logging.basicConfig(level=logging.DEBUG)

```

**### 11.3 パフォーマンス調整**

**#### キャッシュTTL の調整**

```env

# .env ファイル

CACHE\_TTL=86400 # 24時間（秒）

```

**#### レート制限の調整**

```env

# .env ファイル

RATE\_LIMIT\_PER\_MINUTE=60 # 1分あたりのリクエスト数

```

---

**## 12. まとめ**

本モジュールは、法令知識ベースとして以下を提供します：

**### 主要機能**

1. ✅ 法令データの取得・解析

2. ✅ 条文の要約生成

3. ✅ 法的論点の抽出

4. ✅ 高速な全文検索

5. ✅ キャッシュによる最適化

**### 技術スタック**

- FastAPI（Web フレームワーク）

- PostgreSQL（データベース）

- Redis（キャッシュ）

- Google Gemini API（AI処理）

**### 利用方法**

```bash

# 起動

uvicorn app.main:app --reload --port 8083

# API ドキュメント

http://localhost:8083/docs

```

詳細情報は `README.md` を参照してください。