方式設計書

バージョン 1.0

NTT DATA CCS Corporation

改版履歴

版数	日付	担当者
	改版内容	
1.0	2017/02/13	武田
	初版作成	

方式設計書　目次

目次

1. 本書について 1

1.1. 概要 1
1.2. 対象読者 1

2. 全体概要 2

2.1. アプリケーションアーキテクチャ全体構成 2
2.2. 各層の役割 2
2.2.1. フロントエンド 2
2.2.2. サーバサイド 3

3. 技術要素 5

3.1. Node.js 5
3.1.1. 概要 5
3.1.2. 導入 5
3.1.3. 使用方法 6
3.2. nodist 6
3.2.1. 導入 6
3.2.2. 使用方法 6
3.3. npm 7
3.3.1. 概要 7
3.3.2. 導入 7
3.3.3. 使用方法 8
3.4. TypeScript 10
3.4.1. 概要 10
3.4.2. 特徴 10
3.5. Typings 12
3.5.1. 導入 12
3.5.2. 使用方法 12
3.6. OpenLayers 13

3.6.1. 概要 13
3.6.2. サンプル 13
3.7. Knockout 20
3.7.1. 概要 21
3.7.2. サンプル 21
3.8. LESS 23
3.8.1. 概要 23
3.8.2. コンパイル 24
3.8.3. 記法 24

4. Web アプリケーションセキュリティ 26

4.1. SQL インジェクション 26
4.2. XSS（クロスサイト・スクリプティング） 28
4.3. CSRF（クロスサイト・リクエスト・フォージェリ） 28
4.4. セキュリティ診断 30
4.4.1. 非共有の主題図 30
4.4.2. アクセス制御の不備 30
4.4.3. URL に重要情報の保持 31
4.4.4. 内部サーバエラーの検出 31

5. Web サービス 32

5.1. 例外ハンドリング 32
5.1.1. 例外の発生 32
5.1.2. 例外の捕捉 33
5.2. データアクセス 34
5.2.1. ORM 34
5.2.2. Entity Framework 34
5.3. DI（Dependency Injection） 37
5.3.1. DI とは 37
5.3.2. Ninject 37
5.4. ロギング 39
5.4.1. Nlog 39
5.4.2. アクションメソッドの実行ログ 40
5.4.3. エラーログ 41
5.5. マッパー 41
5.5.1. マッパーとは 41
5.5.2. AutoMapper 42

6. Map サービス 44

6.1. 例外ハンドリング 44
6.2. データアクセス 44
6.2.1. Arc GIS Server REST API 44
6.3. DI（Dependency Injection） 45
6.4. ロギング 45

7. 権限及び機能 46
7.1. 権限の種類 46
7.1.1. 石連事務局員 46
7.1.2. 対応オペレーター 46
7.1.3. 元売分析閲覧（自社） 46
7.1.4. 元売分析閲覧（自社以外） 46
7.1.5. エネ庁・石油精製備蓄課 46
7.1.6. エネ庁・石油流通課 46
7.1.7. エネ庁・資源燃料部政策課 46
7.2. 機能の種類 46
7.2.1. 共通機能 47
7.2.2. 主題図一覧 47
7.2.3. 外部機関提供地図一覧 48
7.2.4. ナビゲーション 48
7.2.5. コンテンツ 49
7.2.6. 外部機関提供地図追加 49
7.2.7. テーブル 49
7.2.8. メモ 50
7.2.9. 日時 50
7.2.10. 主題図保存 51
7.3. 各権限の閲覧・機能制限について 51
7.3.1. 各権限で閲覧可能な情報 51
7.3.2. 各権限で使用可能な機能 51

8. 開発ガイド 53
8.1. レイヤ 53
8.1.1. 概要 53
8.1.2. LayerMetadata 53
8.1.3. Metadata 54
8.2. レイヤ定義 54
8.2.1. 概要 54
8.2.2. LayerDefinitionDto 55

8.3. フィルタリング 55
8.3.1. 概要 55
8.3.2. 対象 56
8.3.3. FilteringCondition 56
8.3.4. 検証項目 56
8.3.5. 条件式構築 57
8.4. コンテンツ 57
8.4.1. 方式 57
8.4.2. JsTreeNode 57
8.4.3. LayerDto 58
8.4.4. イベントハンドラ 58
8.5. レイヤ画像 58
8.5.1. 概要 58
8.5.2. 対象 58
8.5.3. 方式 59
8.6. メモ 59
8.6.1. 概要 59
8.6.2. レイヤ表示 59
8.6.3. 登録 60
8.6.4. 編集 60
8.6.5. 削除 60
8.7. 日時コントロール 60
8.7.1. 概要 60
8.7.2. 方式 60
8.8. 主題図 61
8.8.1. 概要 61

1. 本書について

1.1. 概要

本書は、石連災害システム開発に関する全体的な設計・実装における考え方の概要を示すことによって設計・実装における共通的な知識を供すると共に、個別の方式設計における指針となることを目的としている。

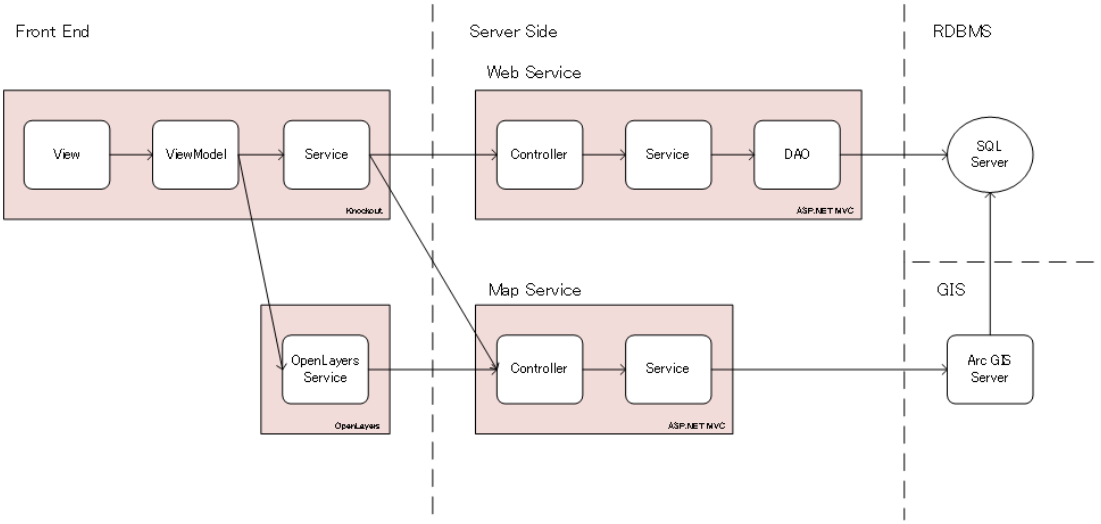
1.2. 対象読者

石連災害システムを開発する関係者を想定する。

2. 全体概要

2.1. アプリケーションアーキテクチャ全体構成

アプリケーションアーキテクチャの全体構成を以下に示す。ピンク色の箇所が今回の開発範囲である。



2.2. 各層の役割

以下に、各層が担当する責務を示す。

2.2.1. フロントエンド

HTML、CSS、JavaScript で構成される Web ブラウザで動作するシステムを指す。フロントエンドはプレゼンテーション層を実現し、画面の表示やユーザの操作を受け取る役割等を担う。本システムでは Knockout を採用し、MVVM パターンで実装することによりプレゼンテーション層の中でも各層の役割を明確にし、保守性の向上を目指す。以下にフロントエンドの構成要素について述べる。

2.2.1.1. View

View では、クライアントに対するユーザインタフェースの提供を行う。Knockout によるデータバインド機能によって ViewModel のプロパティが View に表示される。

2.2.1.2. ViewModel

ViewModel では、ユーザからの操作を受け、Model に処理を委譲した後、Model から受け取った結果を View に反映する。ViewModel のプロパティ更新に伴う View への伝播は Knockout の責務であ

る。ASP.NET MVC で扱う ViewModel とは異なることに留意すること（ASP.NET MVC における ViewModel は View に表示するデータを表し、View-Controller 間でデータを受け渡す際に用いられる）。

2.2.1.3. Model

Model では Web サービスを呼び出し、ViewModel に結果を返却する。Web サービスの呼び出しは View や ViewModel で行わず、必ず Model でのみ行うこと。Model がフロントエンドとサーバサイドの境界である。

2.2.1.4. OpenLayersService

地図操作ライブラリとして OpenLayers を採用する。OpenLayers の設計が MVVM パターンとは異なるため、本システムでは OpenLayersService を Model とは別に用意し、地図操作に関する責務を OpenLayersService で担うこととする。具体的には以下のような処理を担当する。

- 地図の表示
- レイヤの表示
- 地図操作（移動、回転、縮尺）
- フィーチャの描画

レイヤの取得等を行う際は Arc GIS Server REST API を直接呼び出さず、必ず Map Service を呼び出すこと。

2.2.2. サーバサイド

プレゼンテーション層をフロントエンドで実現させることによりサーバサイドはプレゼンテーション層を意識することなく、ビジネスロジック層及びデータアクセス層の実現に徹することが出来る。本システムでは ASP.NET MVC を採用し、MVC パターンで実装する。

2.2.2.1. Web サービス

地図以外の処理を担当する。Controller、Service、DAO で構成される。

1. Controller
   Controller では、クライアントからのリクエストを受け、Service に処理を委譲した後、Service から受け取った結果をクライアントに返却する。
2. Service
   Service では、地図以外のビジネス処理を実現する。データベースへのアクセスは DAO に処理を委譲する。
3. DAO
   DAO はデータアクセスを担当する。本システムでは ORM として Entity Framework を採用する。

2.2.2.2. Map サービス

地図に関する処理を担当する。Arc GIS サーバとの通信は必ず Map サービスで行い、フロントエンドから Arc GIS サーバを隠蔽すること。Controller、Service で構成される。

1. Controller

   Controller では、クライアントからのリクエストを受け、Service に処理を委譲した後、Service から受け取った結果をクライアントに返却する。

2. Service

   Serviec では、地図に関するビジネス処理を実現する。Arc GIS Server REST API の呼び出しは Service でのみ行うこと。

3. 技術要素

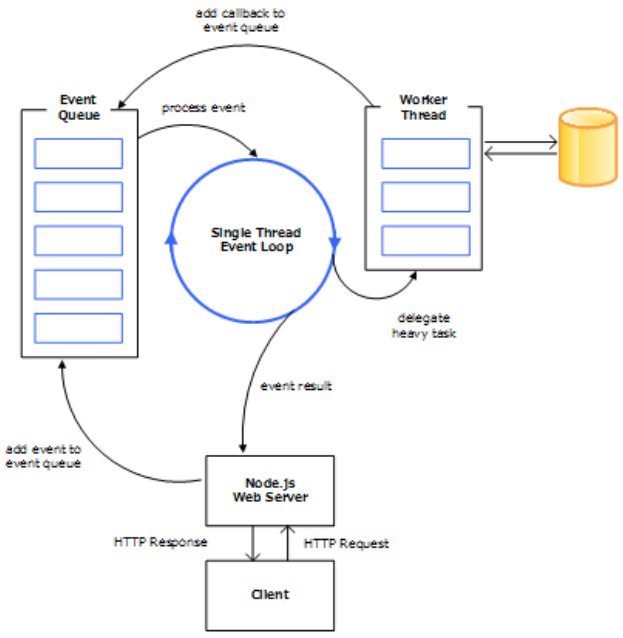
3.1. Node.js

サーバサイド JavaScript 環境である。具体的には、Google が開発する JavaScript 実行エンジンである V8 JavaScript エンジンをサーバで実行出来るようにしたものである。Node.js はノンブロッキング I/O、イベントループを採用していることから、軽量かつ効率的に多くの処理を捌くことが出来る。

3.1.1. 概要

Node.js はシングルスレッドを立ち上げ、イベントループを開始する。また、リクエストやコールバック等をイベントとして扱う。イベントはイベントキューに格納され、先に登録されたイベントから取り出され、処理される。データベースアクセスや I/O タスク等負荷が高い処理はワーカースレッドにデリゲートし、非同期で処理される。ワーカースレッドの処理が完了すると、コールバックがイベントキューに追加される。このデリゲートによって、シングルスレッドはビジーにならず、多くの要求を受け付けることが出来る。

図 3.1. Node.js の処理フロー



3.1.2. 導入

公式ページ(https://nodejs.org/ja/)からインストーラをダウンロードし、実行する。

3.1.3. 使用方法

3.1.3.1. バージョン確認

下記コマンドで、Node.js のバージョンを確認出来る。

```
node --version
v4.1.1
```

3.2. nodist

Node.js のバージョン管理ツールである。

3.2.1. 導入

Node.js を既にインストールしている場合はアンインストールする。公式ページ(https://github.com/marcelklehr/nodist/releases)から最新版のインストーラをダウンロードし、実行する。

3.2.2. 使用方法

3.2.2.1. 使用可能な Node.js の確認

下記コマンドで使用可能な Node.js のバージョン一覧が表示される。

```
nodist dist


...
  6.11.3
  6.11.4
  6.11.5
  6.12.0
  6.12.1
  7.0.0
  7.1.0
  7.2.0
  7.2.1
  7.3.0
  7.4.0
  7.5.0
...
```

3.2.2.2. Node.js のインストール

例えば 7.1.0 をインストールする場合は下記コマンドを実行する。

```
Nodist + 7.1.0
```

Node.js のバージョンを確認すると、指定した Node.js がインストールされたことが分かる。

```
node -v
v7.1.0
```

3.2.2.3. インストールした Node.js の管理

nodist でインストールした Node.js は下記コマンドで確認出来る。

```
nodist
6.1.0
> 7.1.0(global: 7.1.0)
```

Node.js のバージョンを切り替える場合は下記コマンドを実行する。

```
nodist global 6.1.0
```

3.3. npm

Node Package Manager の略で、Node.js のモジュールを管理するツールである。

3.3.1. 概要

3.3.1.1. package.json

npm はフロントエンドで使用するパッケージのバージョン管理を行うのに用いられる。npm では、npm でインストールしたパッケージのバージョン情報を package.json に格納する。この package.json からパッケージを一括でインストールすることが出来るため、package.json を一元管理することによりフロントエンドで使用するパッケージのバージョン管理を一元化することが出来る。

3.3.1.2. インストールの種類

npm ではパッケージをインストールする方法がグローバルインストールとローカルインストールの 2 種類存在する。

グローバルインストールでは npm のルート配下にある node_modules にパッケージがインストールされる。Node.js インストール時に Node.js の実行モジュールのインストール先にパスが通されているため、グローバルインストールされたパッケージを全てのプロジェクトで使用することが出来る。

ローカルインストールではプロジェクトのルート配下にある node_modules にパッケージがインストールされる。ローカルインストールされたパッケージは対象のプロジェクトのみで使用することが出来る。

3.3.2. 導入

Node.js をインストールすると同時に npm もインストールされる。

3.3.3. 使用方法

3.3.3.1. バージョン確認

下記コマンドで、npm のバージョンを確認出来る。

```
npm --version
2.14.4
```

3.3.3.2. 初期化

プロジェクトのルートディレクトリで下記コマンドを実行する。

```
npm init
```

これにより、プロジェクトのルートディレクトリ配下に package.json が作成される。

3.3.3.3. パッケージの復元

プロジェクトのルートディレクトリにパッケージ情報が記載された package.json を置き、下記コマンドを実行する。

```
npm install
```

これにより、package.json に記載されたパッケージが一括でインストールされる。

3.3.3.4. パッケージのインストール

グローバルインストールする場合は下記コマンドを実行する。

```
npm install -g jquery
```

ローカルインストールする場合はプロジェクトのルート配下で下記コマンドを実行する。

```
npm instal jquery
```

--save オプションを付与すると package.json の dependencies に依存関係が追記される。dependencies には公開する際に必要なパッケージを記録する。

```
npm install --save jquery
```

--save-dev オプションを付与すると package.json の devDependencies に依存関係が追記される。devDependencies には開発時に必要なパッケージを記録する（テストツールやタスクランナー等）。

```
npm install --save-dev jquery
```

バージョンを指定してインストールする場合はパッケージ名の後にバージョン名を指定する。

```
npm install jquery@1.1.0
```

3.3.3.5. パッケージのバージョン確認

リリースされたパッケージのバージョン一覧を下記コマンドで取得出来る。

```
npm info jquery versions
```

ローカルインストール済みのパッケージのバージョン一覧は下記コマンドで確認することが出来る。

```
npm list --depth=0
```

グローバルインストール済みのパッケージのバージョン一覧は下記コマンドで確認することが出来る。

```
npm list --depth=0 -g
```

3.3.3.6. パッケージの更新

下記コマンドで未更新のパッケージを確認出来る。

```
npm outdated
```

下記コマンドで package.json に記載されているパッケージのバージョンに更新される。

```
npm update
```

ただし、最新のバージョンを使用するには、package.json に記載されているパッケージのバージョンを更新する必要がある。これを手動で一つ一つ最新か確認するのは高コストである。そこで、npm-check-updates を使用する。npm-check-updates をグローバルインストールする。

```
npm install -g npm-check-updates
```

下記コマンドで package.json のパッケージのバージョンを一括で最新にすることが出来る。

```
npm-check-updates -u
```

この後に npm update を実行することで、一括で最新にすることが出来る。

3.3.3.7. パッケージのアンインストール

下記コマンドで、グローバルインストールしたパッケージをアンインストールすることが出来る。

```
npm uninstall -g jquery
```

ローカルインストールしたパッケージをアンインストールする場合は下記コマンドを実行する。

```
npm uninstall jquery
```

package.json の dependencies から依存関係を削除する場合は--save オプションを付与する。

```
npm uninstall --save jquery
```

package.json の devDependencies から依存関係を削除する場合は--save-dev オプションを付与する。

```
npm uninstall --save-dev jquery
```

3.4. TypeScript

Microsoft 社製の altJS である。

3.4.1. 概要

TypeScript を JavaScript にトランスパイルする必要があるが、TypeScript を使用することにより下記のメリットを受けることが出来る。

- 静的型付けにより、コンパイル時に静的解析が可能となる。
- 型定義ファイルを導入することが出来る。
- 型定義ファイルの導入により、IDE で強力なコード補完を受けることが出来る。
- IDE でデバッグを行うことが出来る。

以下に本システムにおける TypeScript の開発環境を記す。

表 3.1. 本システムにおける TypeScript 開発環境

IDE	Visual Studio 2017
TypeScript の Version	2.3
型定義ファイルの管理	Typings

3.4.2. 特徴

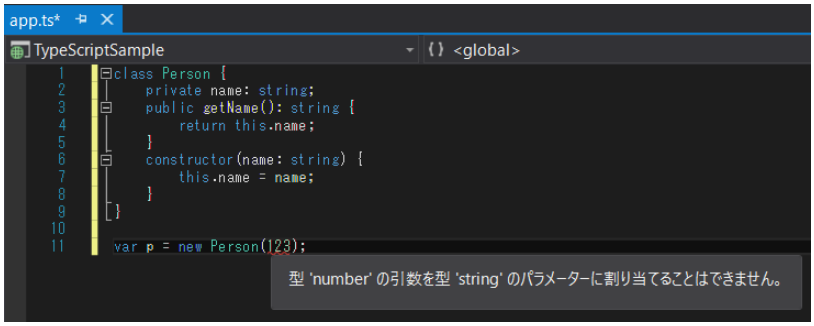
3.4.2.1. 静的型付け及び型チェック

例えば Person クラスを定義する。constructor の引数の型を string として定義している。

```
class Person{
    private name: string;
    public getName(): string{
        return this.name;
    }
    constructor(name: string) {
        this.name = name;
    }
}
```

引数に number を入れると、型チェックにより指摘される。

図 3.2. 型チェックによる指摘



このように静的に型チェックを行うことが出来る。

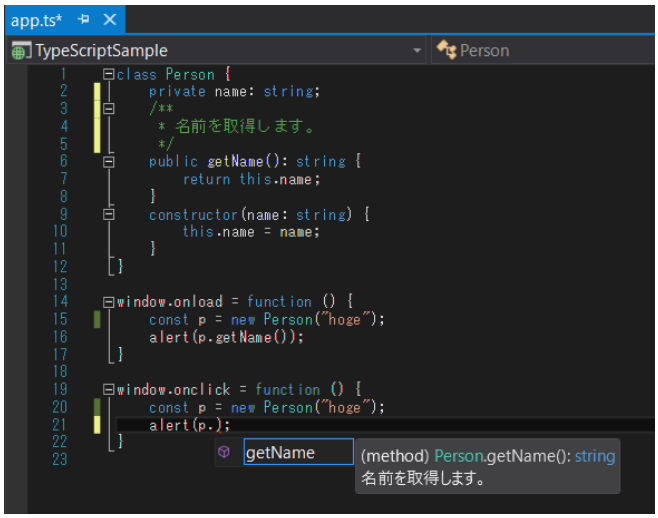
3.4.2.2. 型定義ファイル

型宣言のみを記述した TypeScript ファイルを型定義ファイルといい、拡張子は.d.ts とする。この型定義ファイルは自作することも出来るが、有名な JS ライブラリの場合、既に型定義ファイルが公開されている。型定義ファイルの管理には Typings が用いられる。

3.4.2.3. コード補完

IDE でコード補完を行うことが出来る。

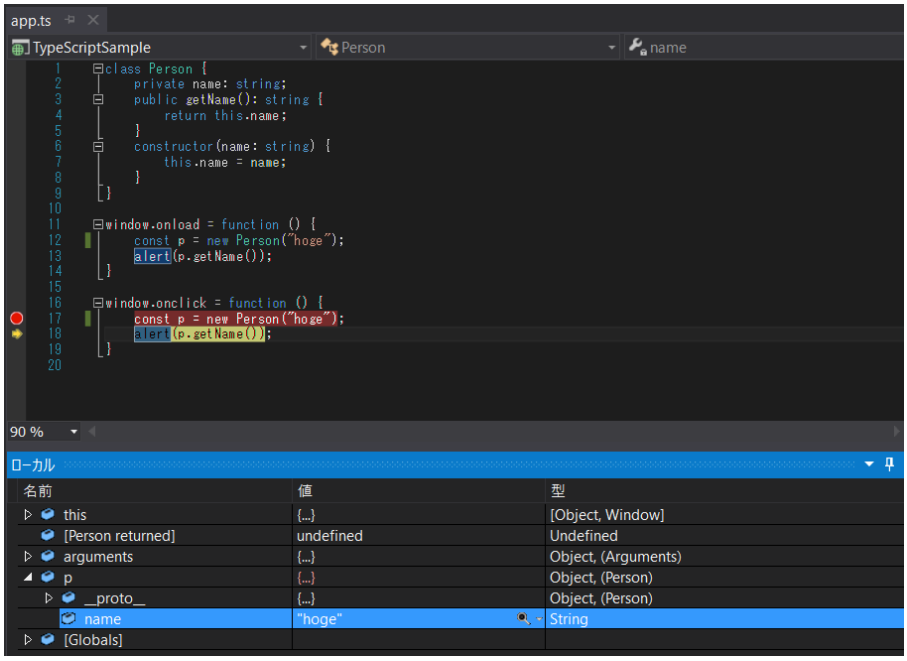
図 3.3. コード補完



3.4.2.4. デバッグ

IDE でデバッグを行うことが出来る。

図 3.4. デバッグしている様子



3.5. Typings

TypeScript の型定義ファイルを管理するツールである。

3.5.1. 導入

npm で取得する。

```
npm install -g typings
```

3.5.2. 使用方法

3.5.2.1. 初期化

プロジェクトのルートディレクトリで下記コマンドを実行する。

```
typings init
```

上記コマンドを実行することにより typings.json が作成される。typings.json に typings でインストールした型定義ファイルの情報が格納される。

3.5.2.2. 型定義ファイルの検索

下記コマンドで型定義ファイルを検索する。

```
typings search jquery
```

3.5.2.3. 型定義ファイルのインストール

下記コマンドをプロジェクトのルートディレクトリで実行し、型定義ファイルをインストールする。

```
typings install dt~jquery --global --save
```

projectRoot/typings/index.d.ts に取得した型定義ファイルへの参照が追加される。

3.5.2.4. 型定義ファイルの使用

ソースに projectRoot/typings/index.d.ts への参照を追加する。

```
/// <reference path=”typings/index.d.ts/>
```

3.5.2.5. 型定義ファイルの削除

下記コマンドで型定義ファイルを削除する。

```
typings remove jquery --global --save
```

3.6. OpenLayers

フロントエンドで地図操作を実現可能にする JavaScript ライブラリである。

3.6.1. 概要

OpenLayers では下記のような機能を提供する。

- 地図の表示
- レイヤの表示
- 地図操作（移動、回転、縮尺）
- フィーチャの描画

3.6.2. サンプル

3.6.2.1. 地図の表示及びコントロール

画面に OpenLayers.js 及び付随する css を読み込む。

Map.html

```
<html>
   <head>
      <link rel="stylesheet" href="https://openlayers.org/en/v4.6.4/c
ss/ol.css" type="text/css">
      <script  src="https://openlayers.org/en/v4.6.4/build/ol.js"></sc
ript>
      <script src="map.js"></script>
```

```
    </head>
    <body>
        <div id="map"></div>
    </body>
</html>
```

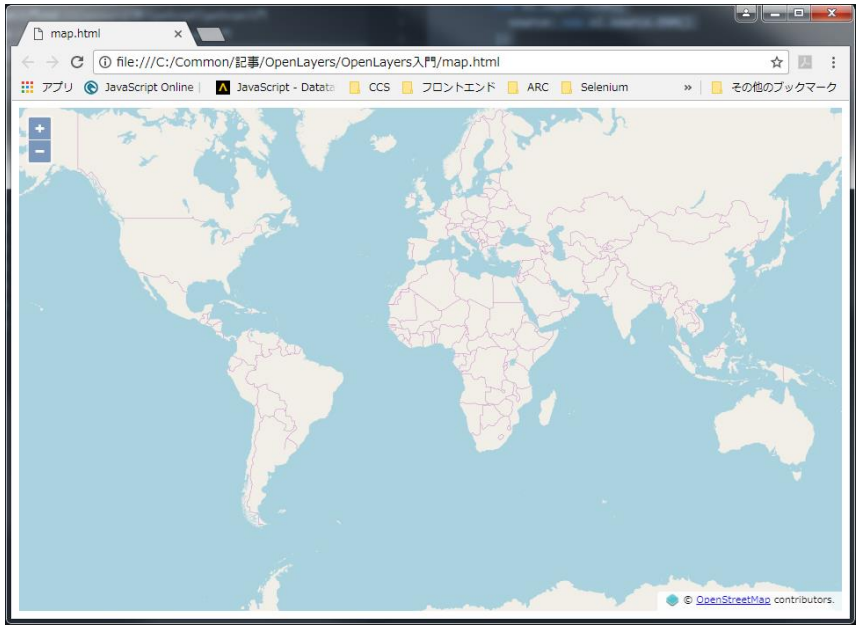
id が map の要素に地図を追加する。

map.js

```
window.onload = function(){
    var map = createMap();
}

/** 地図を作成します。 */
function createMap() {
    return new ol.Map({
        layers: [
            new ol.layer.Tile({
                source: new ol.source.OSM()
            })
        ],
        target: 'map',
        controls: ol.control.defaults({
            attributionOptions: {
                collapsible: false
            }
        }),
        view: new ol.View({
            center: [0, 0],
            zoom: 2
        })
    });
}
```

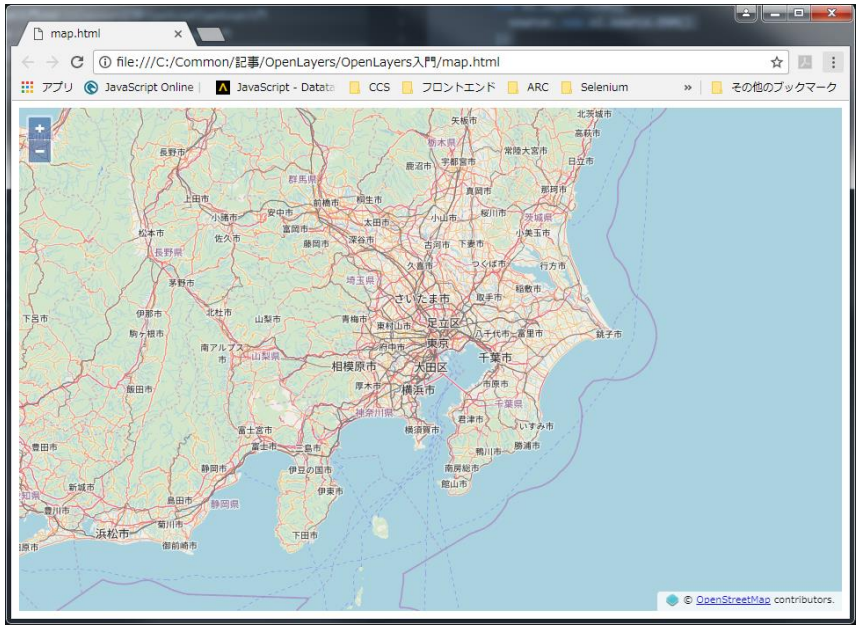
Map.html をブラウザで表示すると画面に地図が現れる。

図 3.5. 地図画面



マウス操作に応じて地図の移動や拡大・縮小を行うことが出来る。

図 3.6. 移動や拡大・縮小後の地図画面



3.6.2.2. フィーチャの表示

フィーチャを外部から取得し、ベクターレイヤを表示する例を以下に示す。

ReadFeature.html

```
<html>
    <head>
        <link rel="stylesheet" href="https://openlayers.org/en/v4.6.4/c
```

```
ss/ol.css" type="text/css">
       <script  src="https://openlayers.org/en/v4.6.4/build/ol.js"></sc
ript>
       <script src="readFeature.js"></script>
   </head>
   <body>
       <div id="map"></div>
   </body>
</html>
```

ベクターレイヤーを作成する際に取得元の URL、フォーマット等を指定する。今回の例では KML を指定している。

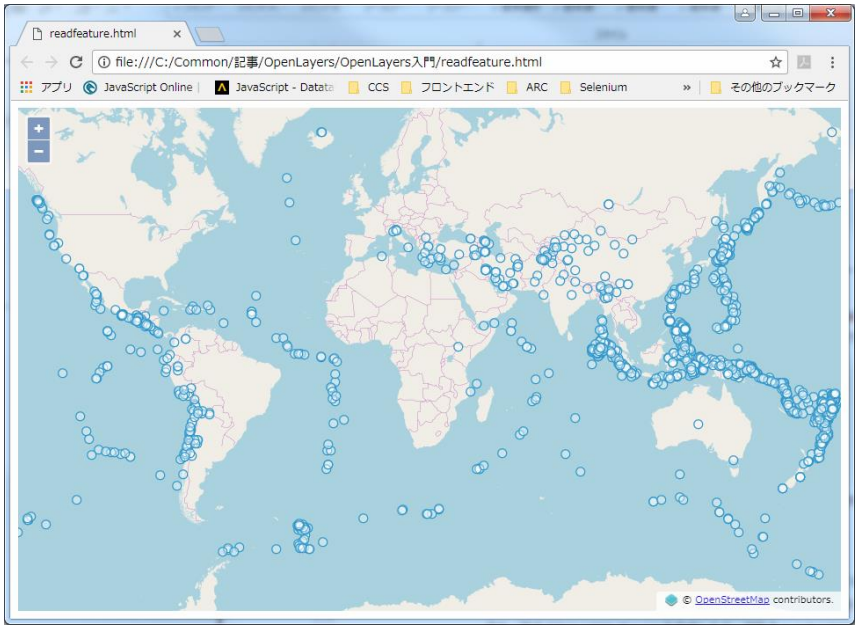
readFeature.js

```
window.onload = function(){
   var map = createMap();
   var vectorLayer = createVectorLayer();
   map.addLayer(vectorLayer);
}

/** 地図を作成します。 */
function createMap() {
   return new ol.Map({
       layers: [
           new ol.layer.Tile({
               source: new ol.source.OSM()
           })
       ],
       target: 'map',
       controls: ol.control.defaults({
           attributionOptions: {
               collapsible: false
           }
       }),
       view: new ol.View({
           center: [0, 0],
           zoom: 2
       })
   });
}
```

```
/** ベクターレイヤを作成します。 */
function createVectorLayer() {
    return new ol.layer.Vector({
        source: new ol.source.Vector({
            url: 'https://openlayers.org/en/v4.6.4/examples/data/kml/201
2_Earthquakes_Mag5.kml',
            format: new ol.format.KML({
              extractStyles: false
            })
        })
    });
}
```

ReadFeature.html をブラウザで表示すると地図にフィーチャが表示される。

図 3.7. フィーチャの表示



3.6.2.3. フィーチャの描画

フィーチャを地図に描画出来るようにする例を以下に示す。プルダウンで描画するフィーチャの種類を切り替える。

DrawFeature.html

```
<html>
    <head>
        <link rel="stylesheet" href="https://openlayers.org/en/v4.6.4/c
```

```
ss/ol.css" type="text/css">
       <script  src="https://openlayers.org/en/v4.6.4/build/ol.js"></sc
ript>
       <script src="createMap.js"></script>
       <script src="drawInteraction.js"></script>
       <script src="drawFeature.js"></script>
   </head>
   <body>
       <div id="map" style="height: calc(100% - 20px);"></div>
       <label>Geometry type  </label>
       <select id="type">
         <option value="Point">Point</option>
         <option value="LineString">LineString</option>
         <option value="Polygon">Polygon</option>
         <option value="Circle">Circle</option>
       </select>
   </body>
</html>
```

描画用のレイヤを用意し、地図に描画出来るように設定する。また、プルダウンの切替に応じて描画するフィーチャの種類を切り替える。

drawFeature.js

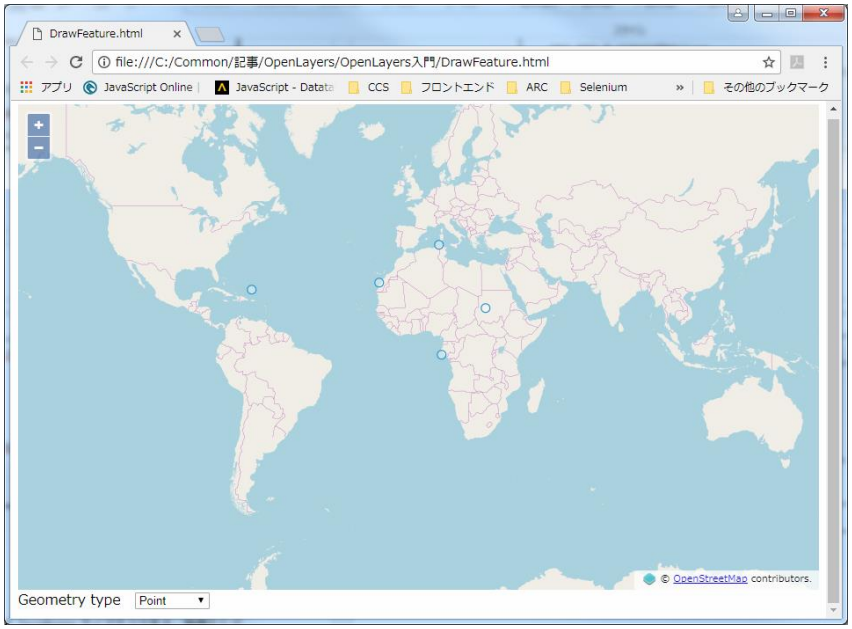
```
window.onload = function(){
   var map = createMap();
   var typeSelect = document.getElementById('type');

   addDrawingLayer(map);
   addDrawInteraction(map, typeSelect.value);

   typeSelect.onchange = function(){
       removeDrawInteraction(map);
       addDrawInteraction(map, typeSelect.value);
   }
}

/** 地図を作成します。 */
function createMap() {
   return new ol.Map({
       layers: [
```

```
            new ol.layer.Tile({
                source: new ol.source.OSM()
            })
        ],
        target: 'map',
        controls: ol.control.defaults({
            attributionOptions: {
                collapsible: false
            }
        }),
        view: new ol.View({
            center: [0, 0],
            zoom: 2
        })
    });
}

var drawingSource = new ol.source.Vector();
var drawingLayer = new ol.layer.Vector({
    source: drawingSource
});
var drawInteraction;

/** 描画用のレイヤを地図に追加します。 */
function addDrawingLayer(map){
    map.addLayer(drawingLayer);
}

/** フィーチャを描画出来るようにします。 */
function addDrawInteraction(map, type){
    // 指定された種類のフィーチャを描画出来るようにする
    drawInteraction = new ol.interaction.Draw({
        source: drawingSource,
        type: type
    });
    map.addInteraction(drawInteraction);
}

/** フィーチャの描画を解除します。 */
```

```
function removeDrawInteraction(map){
    map.removeInteraction(drawInteraction);
}
```

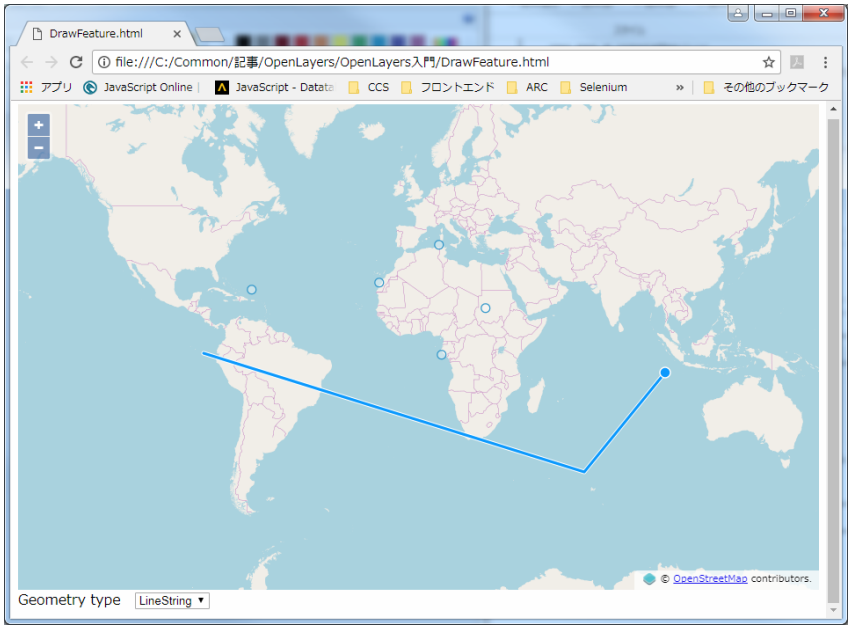
DrawFeature.html をブラウザで表示し地図をクリックするとポイントを描画することが出来る。

図 3.8. ポイントを描画した際の画面



また、プルダウンで種類を変えることにより描画出来るフィーチャの種類が変わる。

図 3.9. フィーチャの種類を変えた際の画面



3.7. Knockout

フロントエンドでデータバインディング機能を提供する JavaScript ライブラリである。

3.7.1. 概要

特徴として下記が挙げられる。

- 軽量である（59 kb）。
- 他の JavaScript ライブラリと依存関係が無いため、競合しない。
- 公式のサンプルが豊富である。

Knockout を使用することにより DOM を直接操作することなく、ViewModel のプロパティを更新するだけで View に更新を反映することが可能となる。

3.7.2. サンプル

3.7.2.1. ViewModel のプロパティを View に表示する

ViewModel のプロパティを View に表示する例を以下に示す。

Knockout では ko.applyBindings で View に ViewModel をバインドさせる。

simpleBindingViewModel.js

```
var viewModel = {
    msg: "Hello Knockout World!"
};

window.onload = function () {
    ko.applyBindings(viewModel);
}
```

画面では data-bind 属性を指定することにより ViewModel のプロパティを表示することが出来る。

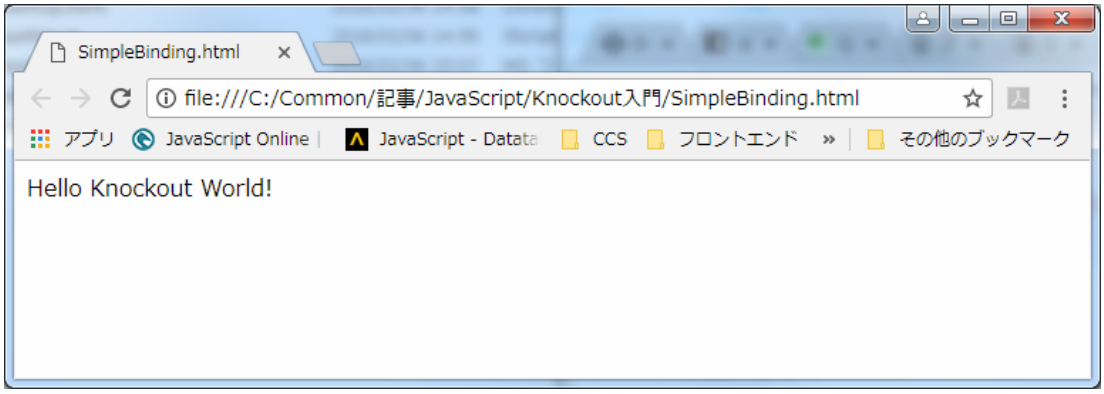
SimpleBinding.html

```
<html>

<head>
    <script type="text/javascript" src="https://cdnjs.cloudflare.com/aj
ax/libs/knockout/3.4.2/knockout-min.js"></script>
    <script type="text/javascript" src="simpleBindingViewModel.js"></sc
ript>
</head>

<body>
    <div data-bind="text: msg"></div>
</body>
```

```
</html>
```

SimpleBinding.html をブラウザで表示すると ViewModel のプロパティである「Hello Knockout World!」が画面に表示される。

図 3.10. ViewModel のプロパティが View に表示される例



3.7.2.2. View のイベントを検知し ViewModel のプロパティを更新する

View のイベントを検知し ViewModel のプロパティを更新する例を以下に示す。

更新するプロパティを observable とする。プロパティを取得するには引数なしの observable を呼ぶ。プロパティを更新する際には、引数に新しい値を入れ observable を呼ぶ。

countUpViewModel.js

```
var viewModel = {
    clickedCount: ko.observable(0)
};


function countUp() {
    cnt = viewModel.clickedCount();
    viewModel.clickedCount(cnt + 1);
}


window.onload = function () {
    ko.applyBindings(viewModel);
}
```

ボタンに関数をバインドする。

CountUp.html

```
<html>


<head>
    <script type="text/javascript" src="https://cdnjs.cloudflare.co
```

```
m/ajax/libs/knockout/3.4.2/knockout-min.js"></script>
    <script type="text/javascript" src="countUpViewModel.js"></scri
pt>
</head>

<body>
    <div><button data-bind="click: countUp">Click Me!</button></div>
    <div>Clicked Count: <span data-bind="text: clickedCount"></span>
</div>
</body>

</html>
```

CountUp.html をブラウザで表示すると、countUp が実行され、クリック数が増える。

図 3.11. ボタンクリック前の画面

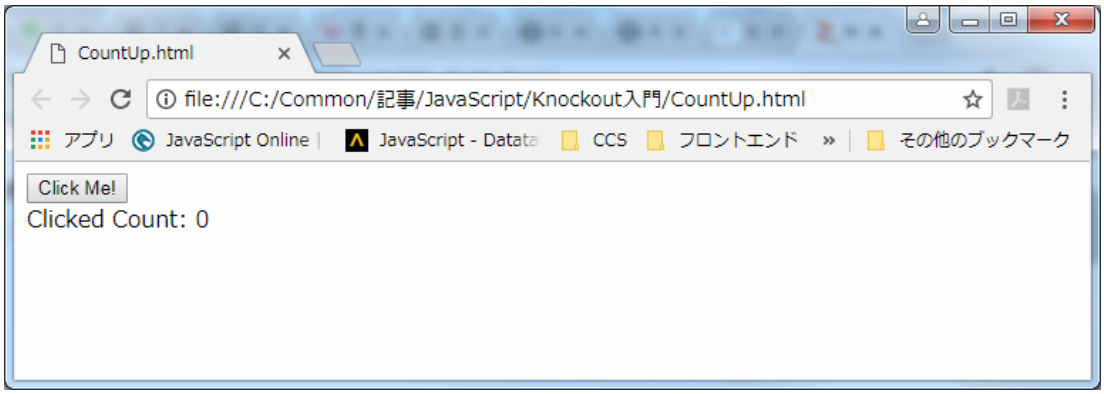
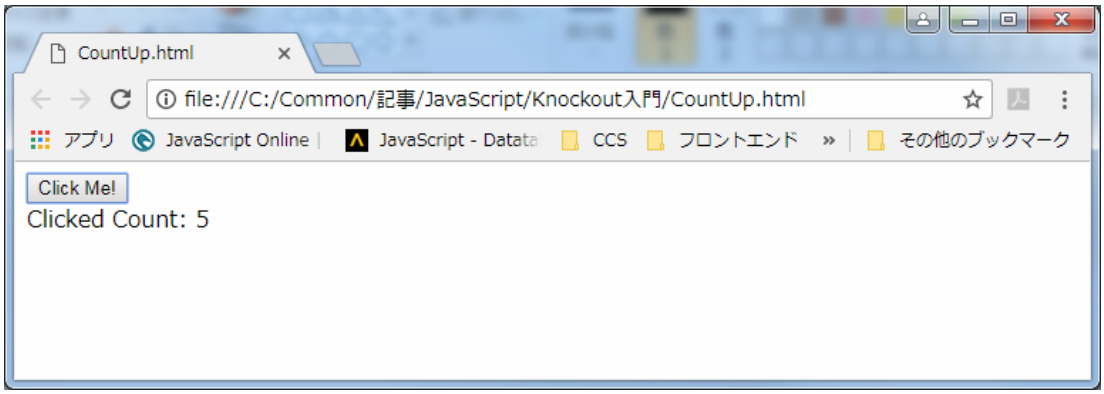


図 3.12. ボタンクリック後の画面



3.8. LESS

CSS の拡張メタ言語である。

3.8.1. 概要

CSS にコンパイルする必要があるが、LESS を使用することにより下記のメリットを受けることが

出来る。

- 変数を使用することが出来る。
- 分割した LESS ファイルを 1 つの CSS ファイルにコンパイルすることが出来る

変数を使用することが可能になるため、共通で使用する設定（色、幅等）を局所化することが出来る。また、部品ごとに LESS ファイルを作成したり、定数を 1 つの LESS ファイルにしたりすることにより保守性の向上が期待される。

3.8.2. コンパイル

3.8.2.1. コンパイラーの導入

npm でグローバルインストールする。

```
npm install -g less@2.7.2
```

3.8.2.2. コンパイル方法

下記コマンドで less ファイルを css ファイルにコンパイルする。-x オプションで圧縮した css ファイルを生成する。

```
lessc -x src.less dst.css
```

3.8.3. 記法

3.8.3.1. 変数

変数は下記のように記述する。

```
@hoge-color: rgba(178, 222, 255, 0.5);
@hoge-width: 300px;
```

3.8.3.2. インポート

hoge.less に const.less をインポートする場合下記のように記述する。

```
@import const.less;
```

less ファイルを分割した場合は 1 つの less ファイルに全てインポートし、コンパイルすれば 1 つの css ファイルにまとめられる。

3.8.3.3. エスケープ

例えば css で横幅を計算した値に設定する場合下記のように記述する。

```
.hoge-width{
    width: calc(100vw - 200px);
```

```
}
```

これを less で記述する場合はエスケープする必要がある。

```
.hoge-width{
    width: calc(~”100vw - 200px”);
}
```

エスケープする場合は文字列の先頭に~を付ける。

4. Web アプリケーションセキュリティ

4.1. SQL インジェクション

4.1.1. 概要

SQL インジェクションとは、アプリケーションのセキュリティ上の不備を突き、アプリケーションが想定しない SQL を実行させることによってデータベースを不正に操作する攻撃のことである。対策として下記の方法が挙げられる。

- 入力値に SQL インジェクションで使用される特殊文字（以下、特殊文字）が含まれる場合、適切にエスケープする。
- 特殊文字の入力を制限する。
- 言語、フレームワークで提供されている機能を利用して SQL を発行する。

4.1.2. 対策

本システムでは、SQL インジェクションを防ぐために以下の対策を実施する。

4.1.2.1. Web サービス

Web サービスでは、ORM に Entity Framework を、SQL の発行に LINQ to SQL を使用することによりライブラリ側で SQL インジェクションを防止する。

4.1.2.2. Map サービス

Map サービスでは、Arc GIS サーバにクエリを直接送信する必要があるため、一般的な Web サービスと対策が異なる。

旧分析システムでは、クエリをフロントエンドで作成し、それを Arc GIS サーバに送信していた。そのため、フロントエンドで任意のクエリに書き換えて、Arc GIS サーバに直接送信することが可能であった。

```
https://baltan.science.nttdata-ccs.co.jp/dev/Portal/Bunseki/app/proxy.
ashx?https://arcgis/rest/services/dev_pajdis_request/MapServer/0/quer
y?returnGeometry=true&geometryType=esriGeometryEnvelope&geometry={"xmi
n":11851371.4285714,"ymin":2660000,"xmax":19048628.5714286,"ymax":5940
000,"spatialReference":{"wkid":102100}}&inSR=102100&spatialRel=esriSpa
```

```
tialRelIntersects&where=対象日 <= '201801151616' and '201801151616' < 報
告対象日時_T and インシデント ID = '43E42961-0229-43E9-9DAA-D8B1F6225A55'
and '201610270000' <= 対象日 and (  状況 = '登録' AND 対応 LIKE '%石連%' )&o
utSR=102100&outFields=*&f=json&
```

本システムでは、フロントエンドでのクエリ構築を禁止する。フロントエンドからはクエリの構築に必要なパラメータのみをサーバに送信させ、サーバサイドでクエリを構築し、Arc GIS サーバと通信する方式を取る。これによりフロントエンドで任意のクエリに書き換えることを防ぐとともに、Arc GIS サーバを外部から隠蔽することが可能となる。以下に具体的な対策を示す。

まず、フロントエンドで LayerDefinitionDto にレイヤの取得条件を設定し、Map サービスに送信する。LayerDefinitionDto については 8.2.2 を参照すること。LayerDefinitionDto を使用することにより下記のセキュリティ上のメリットがある。

- 項目名の代わりに MetadataId を使用することにより悪意のあるユーザによる不正な条件の推測を抑制する。
- 条件式の代わりに conditionalOperatorCd を使用することにより悪意のあるユーザによる不正な条件式の推測を抑制する。

```
http://localhost:2286/AttributeTable/FetchAttributeTableDataAsync?laye
rDef={"filteringConditions":[{"metadataId":"2","value":"登録","andOrCd
":0,"conditionalOperatorCd":"0"},{"metadataId":"3","value":"石連","and
OrCd":0,"conditionalOperatorCd":"6"}],"layerId":0,"layerGroupCd":0,"in
cidentId":"43e42961-0229-43e9-9daa-d8b1f6225a55","startDate":"2016/10/
27 10:00:00","endDate":"2018/01/15 16:27:07"}&xmin=12479584.57423778&y
min=2426546.752122817&xmax=21872166.60992024&ymax=6829319.581348969&pr
ojection=3857&isImportantVisible=true
```

次に、Map サービスでリクエストを検証する。value はユーザが任意に入力することが可能なため下記検証を行う。

- 文字数が指定した項目の最大入力可能文字数以下であること。
- 入力値に特殊文字が含まれていないこと。

検証後、クエリを構築し、Arc GIS サーバに送信する。

```
http://pigmon.science.nttdata-ccs.co.jp/arcgis/rest/services/dev_pajdi
s_request/MapServer/0/query?returnGeometry=true&geometryType=esriGeome
tryEnvelope&geometry={"xmin":12479584.57423778,"ymin":2426546.75212281
7,"xmax":21872166.60992024,"ymax":6829319.581348969,"spatialReference
":{"wkid":3857}}&inSR=3857&spatialRel = esriSpatialRelIntersects&where
=対象日 <= '201801151627' AND '201801151627' < 報告対象日時_T and インシデ
ント ID = '43e42961-0229-43e9-9daa-d8b1f6225a55' AND '201610270000' <=
```

```
対象日 AND ( 状況 = '登録' AND 対応 LIKE '%25 石連%25')&outSR=3857&outFields
=*&f=json
```

以上の対策により、フロントエンドで任意のクエリを作成し、サーバで実行されることを防ぐ。

4.2. XSS（クロスサイト・スクリプティング）

4.2.1. 概要

XSS とは、アプリケーションのセキュリティ上の不備を突き、悪意を持ったスクリプトを埋め込み、ユーザに埋め込んだスクリプトを実行させる攻撃手法のことである。対策として下記の方法が挙げられる。

- 入力値に XSS で使用される特殊文字（以下、特殊文字）が含まれる場合、適切にエスケープする。
- 特殊文字の入力を制限する。
- 画面で値を表示する際に特殊文字をエスケープする。

ユーザによる入力値をエスケープ、または制限することにより悪意のあるスクリプトが埋め込まれることを防ぐことが出来る。さらに、表示する際にエスケープすることによって悪意のあるスクリプトの実行を防止することが出来る。

4.2.2. 対策

本システムでは XSS を防ぐために以下の対策を実施する。

4.2.2.1. 入力の制限

フロントエンドで特殊文字（「&」「<」「>」「'」「"」「/」「*」）の入力を制限することにより悪意のあるスクリプトを埋め込むことが出来ないようにする。サーバサイドについては、使用しているフレームワーク（ASP.NET）で XSS の疑いのある入力が制限される（Request.Params からリクエストパラメータを取得した時）。

4.2.2.2. 特殊文字のエスケープ

フロントエンドでテーブルやテキストボックスに値を表示する際に使用している Knockout では、特殊文字を自動でエスケープする。ただし、レイヤツリーの表示に使用している JsTree では特殊文字がエスケープされないため、別途特殊文字のエスケープを行う。

4.3. CSRF（クロスサイト・リクエスト・フォージェリ）

4.3.1. 概要

CSRF とは、ユーザが認証されている信頼済みのサイトを使用している際に悪意のあるリンクを押下した場合に、ログインしたユーザの権限で攻撃者のスクリプトを実行させる攻撃手法である。対策とし

て、クライアントが推測困難なトークンを付与したリクエストをサーバに送信し、サーバサイドでそのトークンを検証する方法がある。この方法により攻撃者はトークンを偽造することが出来ないため、攻撃者の作成した悪意のあるスクリプトをサーバで検出することが出来る。

4.3.2. 対策

本システムでは CSRF を防ぐために以下の対策を実施する。

フロントエンドで ASP.NET で提供されている AntiForgery.GetTokens()で推測が困難な偽造防止トークンを作成する。

```
<script>
    public string GenerateRequestValificationToken()
    {
        string cookieToken, formToken;
        AntiForgery.GetTokens(null, out cookieToken, out formToken);
        return cookieToken + ":" + formToken;
    }
</script>

<input type="hidden" id="requestValificationToken" value="@GenerateReq
uestValificationToken()" />
```

クライアントで作成したトークンをヘッダーに付与しリクエストをサーバに送信する。

```
$.ajax({
    url: url,
    headers: {
        'RequestVerificationToken': $('#requestVerificationToken').val
()
    }
})
```

サーバサイドで ASP.NET で提供されている AntiForgery.Validate()でトークンを検証することにより正規のユーザからのリクエストであるかを判定する。

```
void ValidateAntiForgeryToken()
{
    string cookieToken = "";
    string formToken = "";
    var tokenHeader = Request.Headers["RequestVerificationToken"];
    if (!String.IsNullOrEmpty(tokenHeader))
    {
```

```
        string[] tokens = tokenHeader.Split(':');
        if (tokens.Length == 2)
        {
            cookieToken = tokens[0].Trim();
            formToken = tokens[1].Trim();
        }
    }
    AntiForgery.Validate(cookieToken, formToken);
}
```

4.4. セキュリティ診断

4.4.1. 非共有の主題図

本システムは主題図一覧画面と地図画面の 2 画面で構成される。主題図一覧画面で主題図を選択すると別タブで地図画面が開くフローとなっている。主題図一覧画面にはログインユーザが作成した、もしくは共有フラグが有効な主題図のみが表示される。従って、主題図一覧画面から非共有の主題図を選択することは出来ない。

本システムでは主題図情報を主題図一覧画面から地図画面へ受け渡す際に SessionStorage を使用する。SessionStorage のスコープは 1 ウィンドウ内かつ同一ドメイン内であり、異なるドメインから本システムの SessionStorage を改ざんすることは不可能である。従って、SessionStorage を改ざんすることによって非共有の主題図情報を地図画面に表示するといった攻撃は成功しない。

以上をまとめると、下記方法によって非共有の主題図を地図画面に表示することを防ぐ。

- 主題図一覧画面に表示する主題図をログインユーザが作成した、もしくは非共有フラグが有効なものに制限する。
- SessionStorage を利用し、主題図情報を受け渡す。

4.4.2. アクセス制御の不備

本システムでは Controller の各メソッドに AuthorizeAttribute アノテーションを付与することによってアクセス制御を行う。この対応により、権限のないユーザへの意図しない情報公開を防ぐ。以下に地図画面を返却する Controller の実装例を示す。

```
public class MapController : Controller
{
    [Authorize(Roles = AnalysisConst.EnabledRoles)]
    public ActionResult Index()
    {
        return View();
    }
}
```

4.4.3. URL に重要情報の保持

旧分析システムではユーザ ID 等の重要情報が URL に含まれていた。

```
/dev/Portal/Bunseki/app/proxy.ashx?https://bws01.pajdis.jp/arcgis/rest
/services/dev_pajdis_memo/FeatureServer/0/query?returnGeometry=true&sp
atialRel=esriSpatialRelIntersects&where=(((HOKOKU_DATETIME_TO > '2017/
02/08 09:22:00') AND (HOKOKU_DATETIME <= '2017/02/08 09:22:00' ))) AND
((USER_ID = 'sekiren') OR (SHARE_FLG = '1')) AND (INCIDENT_ID = '01c474
81-b516-4175-bb6c-405ed6e332a1')&outSR=102100&outFields=*&f=json&_ts=6
362217493135214 73& HTTP/1.1
```

本システムではサーバサイドでユーザ ID 等の重要情報を必要に応じて付与するようにする。この対応により、重要情報の漏洩を防止する。

4.4.4. 内部サーバエラーの検出

内部サーバエラーが発生した場合は、Controller クラスを継承した BaseController クラスの OnException メソッドで補足する。エラー詳細はログに出力し、クライアントにはエラーを隠蔽したオブジェクトを JsonResult に格納し、HTTP ステータスコード 200 で返却する。上記の対応を行うことにより、内部サーバエラーの詳細を攻撃者に知られることを防ぐ。

```
public class BaseController : Controller
{
    protected override void OnException(ExceptionContext filterContext)
    {
        Logger.Error(filterContext.Exception.Message);
        filterContext.Result = new JsonResult()
        {
            Data = new JsonResultDto()
            {
                Status = AnalysisConst.ResultStatus.Error,
                Messages = new List<string> { "エラーが発生しました。" },
                Data = null
            },
            JsonRequestBehavior = JsonRequestBehavior.AllowGet
        };
        filterContext.ExceptionHandled = true;
    }
}
```

5. Web サービス

5.1. 例外ハンドリング

以降、本システムの Web サービスにおける例外ハンドリング方式について述べる。

5.1.1. 例外の発生

例外の基底クラスとして BaseException クラスを用意し、作成する例外クラスは全て BaseException クラスを継承させる。

```
public class BaseException : Exception
{
    /// <summary>
    /// Result Status
    /// </summary>
    public ResultStatus Status { get; set; }

    /// <summary>
    /// エラーメッセージ
    /// </summary>
    public IList<string> Messages { get; set; }

    /// <summary>
    /// Json Data
    /// </summary>
    public Object ResultData { get; set; }

    public BaseException(ResultStatus status, IList<string> messages)
    {
        this.Status = status;
        this.Messages = messages;
    }

    public BaseException(ResultStatus status, IList<string> messages, O
```

```
bject data) : this(status, messages)
    {
        this.ResultData = data;
    }
}
```

検証時の例外を表すクラスとして ValidateException クラスを用意し、サーバサイドで検証に失敗した場合に ValidateException クラスを throw する。

```
public class ValidateException : BaseException
{
    public ValidateException(IList<string> messages) : base(ResultStatu
s.ValidateError, messages)
    {
    }

    public ValidateException(IList<string> messages, Object data) : bas
e(ResultStatus.ValidateError, messages, data)
    {
    }
}
```

ビジネス処理の例外を表すクラスとして BusinessException クラスを用意し、サーバサイドでビジネス処理時に例外が発生した場合に BussinessException クラスを throw する。

5.1.2. 例外の捕捉

Controller の基底クラスとして BaseController クラスを用意し、作成する Controller クラスは全て BaseController クラスを継承させる。BaseController クラスの OnException メソッドによって例外を補足する。

```
public class BaseController : Controller
{
    protected override void OnException(ExceptionContext filterContext)
    {
        Logger.Error(filterContext.Exception.Message);
        filterContext.Result = new JsonResult()
        {
            Data = new JsonResultDto()
            {
                Status = AnalysisConst.ResultStatus.Error,
```

```
                Messages = new List<string> { "エラーが発生しました。" },
                Data = null
            },
            JsonRequestBehavior = JsonRequestBehavior.AllowGet
        };
        filterContext.ExceptionHandled = true;
    }
}
```

5.2. データアクセス

本システムでは RDBMS として SQL Server を使用している。Web サービスでは、データアクセスに SQL Server と相性が良い Entity Framework を利用する。以降、ORM と Entity Framework について述べる。

5.2.1. ORM

アプリケーションの開発に採用されるオブジェクト指向では、データをオブジェクトとして扱う。しかし、RDB ではデータをタプルとフィールドを持つテーブルで扱い、テーブルのフィールド同士を関連付けることでリレーションを表現する。オブジェクト指向と RDB の設計思想は異なり、オブジェクトとリレーショナルモデルとの間にはインピーダンスミスマッチがある。

例えば、オブジェクト指向では、多対多の関係を 2 つのクラスで表現することが出来る。ユーザに複数の権限を持たせる場合、ユーザクラスと権限クラスを用意し、ユーザクラスのプロパティに権限一覧を持たせればよい。しかし、RDB では、多対多の関係を表現できないため、ユーザテーブル、権限テーブルの他に、ユーザ権限テーブルのような連関テーブルが必要になる。複数権限を持つユーザを RDB に登録するには、ユーザテーブル、権限テーブル、ユーザ権限テーブルそれぞれにレコードを追加する必要がある。また、複数権限を持つユーザを RDB から取得する場合は、レコードからオブジェクトに変換する必要がある。

オブジェクトとリレーショナルモデルとの間にあるインピーダンスミスマッチを解消するツールが ORM（Object-Relational Mapper）である。ORM を利用することで下記のメリットが受けられる。

・コネクションの取得に関する記述を隠蔽することが出来る。

・オブジェクトを解釈し、SQL を発行する必要が無くなる。

・結果セットからオブジェクトに変換する必要が無くなる。

5.2.2. Entity Framework

Entity Framework ではデータを概念モデルで扱う。このデータのことをエンティティと呼ぶ。また、概念モデルのことを EDM（Entity Date Model）と呼び、EDM に対して処理を行うことによってデータベースに対する操作を行うことが出来る。EDM ではリレーションをアソシエーションで表す。

EDM の作成方法は 2 通りある。まず 1 つ目はデータベースファーストである。データベースファーストでは、データベースから EDM を自動生成する。2 つ目はモデルファーストである。モデルファー

ストでは、EDM を手動で設計、作成し、データベースを自動生成する。本システムではデーターベースファーストを採用する。

EDM を作成/更新すると、Entity Framework で下記 2 クラスが自動生成される。

・エンティティクラス

・コンテキストクラス

エンティティクラスは、EDM で定義したエンティティに対応するクラスである。アプリケーションではデータベースのデータをエンティティとして利用することが出来る。

コンテキストクラスは、データベースへのコネクションを隠蔽し、データベースへの変更を反映させる等の機能を持つ DbContext クラスを実装したクラスである。Entity Framework を使用する場合、まず始めにコンテキストクラスのインスタンスの作成を行う。

コンテキストクラスは、各エンティティのコレクションを DbSet として持つ。DbSet クラスは検索、追加、更新、削除操作をもつクラスである。コンテキストクラスの持つエンティティの DbSet に対して操作を行うことで、データベースからデータを取得したり、データを変更したりすることが出来る。ただし、データの変更はコンテキストクラスの SaveChanges メソッドを呼ばないとデータベースに保存されないことに留意すること。

以下にコンテキストクラスの例を示す。

```
public partial class Entities : DbContext
{
    public Entities()
        : base("name=Entities")
    {
    }

    protected override void OnModelCreating(DbModelBuilder modelBuilde
r)
    {
        throw new UnintentionalCodeFirstException();
    }

    public virtual DbSet<PAJ_B_MAP_STATE> PAJ_B_MAP_STATE { get; set; }
}
```

以下に、データベースからデータを取得する例を示す。コンテキストクラスのインスタンスを生成し、インスタンスのエンティティの DbSet に対して条件を指定すればよい。

```
public PAJ_B_MAP_STATE FindOne(int? id)
{
    using (var entities = new Entities())
    {
```

```
        return Entities.PAJ_B_EXTERNAL_MAP.Where(e => e.ID.Equals(id)).S
ingle();
    }
}
```

以下にデータベースにデータを追加する例を示す。コンテキストクラスのインスタンスを生成し、インスタンスのエンティティの DbSet に新しくエンティティを追加し、インスタンスの SaveChange メソッドを呼び出せばよい。

```
public void Create(PAJ_B_MAP_STATE entity)
{
    using (var entities = new Entities())
    {
        entities.PAJ_B_MAP_STATE.Add(entity);
        entities.SaveChanges();
    }
}
```

以下にデータベースのデータを更新する例を示す。コンテキストクラスのインスタンスを生成し、インスタンスのエンティティの DbSet から対象のエンティティを取得し、更新した後にインスタンスの SaveChange メソッドを呼び出せばよい。

```
public void Update(PAJ_B_MAP_STATE entity)
{
    using (var entities = new Entities())
    {
        var old = entities.PAJ_B_MAP_STATE.Where(e => e.ID.Equals(entit
y.ID)).SingleOrDefault();
        old.COMMENT = entity.COMMENT;
        entities.SaveChanges();
    }
}
```

以下にデータベースからデータを削除する例を示す。コンテキストクラスのインスタンスを生成し、インスタンスのエンティティの DbSet から対象のエンティティを取得し、DbSet からエンティティを削除した後にインスタンスの SaveChange メソッドを呼び出せばよい。

```
public void DeleteById(int id)
{
    using (var entities = new Entities())
    {
```

```
        var entity = entities.PAJ_B_MAP_STATE.Where(e => e.ID.Equals(i
d)).SingleOrDefault();
        entities.PAJ_B_MAP_STATE.Remove(entity);
        entities.SaveChanges();
    }
}
```

5.3. DI（Dependency Injection）

本システムでは、Ninject を使用して DI を実現する。以降、DI の詳細、及び Ninject の導入方法について述べる。

5.3.1. DI とは

DI とはオブジェクト間の依存性を必要に応じて実行時に外部から注入することである。DI を導入することによって、下記のメリットが得られる。

- アプリケーションの各層を美しく分離した設計が容易になる。
- テスト時にモックを注入することで正しい単体テストを作成することが出来る。

本システムでは、サービス層、データベースアクセス層、ロガー、データベースコンテキストを DI の対象とする。

5.3.2. Ninject

NuGet で Ninject、Ninject.MVC5、Ninject.Web.Common、Ninject.Web.Common.WebHost を導入する。導入後、App_Start 配下に NinjectWebCommon クラスが作成される。NinjectWebCommon クラスの RegisterServices メソッドに依存性を記述する。下記に依存性の記述例を示す。

```
/// <summary>
/// Load your modules or register your services here!
/// </summary>
/// <param name="kernel">The kernel.</param>
private static void RegisterServices(IKernel kernel)
{
    // service
    kernel.Bind<IUserService>().To<UserService>();
    // dao
    kernel.Bind<IUserDao>().To<UserDao>();
    // Logging
    kernel.Bind<ILogger>().ToMethod(ctx =>
    {
```

```
        return  LogManager.GetLogger(ctx.Request.Target.Member.Declaring
Type.FullName);
    });
    // EF
    kernel.Bind<IEntities>().To<Entities>();
```

Ninject によって依存性を注入する場合は Inject アノテーションを DI の対象のプロパティに付与すればよい。下記に UserController クラスに UserService クラスを注入する場合の記述を示す。

```
public class UserController : BaseController
{
    #region DI

    [Inject]
    public IUserService Service { get; set; }

    #endregion

    [HttpGet]
    [Authorize(Roles = AnalysisConst.EnabledRoles)]
    [OutputCache(Location = OutputCacheLocation.None, NoStore = true)]
    public ActionResult FindUserAndCompanyById()
    {
        ValidateAntiForgeryToken();
        return new JsonResult()
        {
            Data = new JsonResultDto()
            {
                Status = AnalysisConst.ResultStatus.Success,
                Data = Service.FindUserAndCompanyById(User.Identity.Name)
            },
            JsonRequestBehavior = JsonRequestBehavior.AllowGet
        };
    }
}
```

5.4. ロギング

ロギングライブラリを利用することにより、ログ出力の統一化を行う。本システムでは　ロギングライブラリとして NLog を採用する。

5.4.1. Nlog

NuGet で NLog、NLog.Config、NLog.Schema、NLog.Web を導入する。以下にログ出力する際の例を示す。以下の例では、Logger インスタンスを DI で注入している。

```
[Inject]
public ILogger Logger { get; set; }


Logger.Info("test message");
```

ログ出力の設定は NLog.config に記載する。以下に、NLog.config を示す。以下の例では、C:/Temp/Log 配下に yyyyMMdd_PAJDIS_NEW_ANALYSIS.log ファイルを作成し、日時、ログの重要度（INFO、ERROR 等）、ユーザ ID、メッセージをログ出力する。

```
<?xml version="1.0" encoding="utf-8" ?>
<nlog xmlns="http://www.nlog-project.org/schemas/NLog.xsd"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">


  <targets>
    <target
      name="f"
      xsi:type="File"
      layout="${longdate} [${uppercase:${level:padding=-5}}], [ユーザ ID:
 ${aspnet-user-identity}], [${message}]"
      fileName="C:/Temp/Log/${date:format=yyyyMMdd}_PAJDIS_NEW_ANALYSI
S.log"
      encoding="UTF-8"
      archiveEvery="Day"/>
  </targets>


  <rules>
    <logger name="*" minlevel="Debug" writeTo="f" />
  </rules>
</nlog>
```

以降、本システムで取得するログについて述べる。

5.4.2. アクションメソッドの実行ログ

BaseController クラスの OnActionExecuting メソッドによって、アクションメソッドの実行前にアクションメソッドの実行情報をログ出力する。出力内容を以下に示す。

- セッション ID
- コントローラ名
- アクションメソッド名
- GET リクエストの場合クエリストリング
- POST リクエストの場合フォームデータ

また、以下に OnActionExecuting メソッドの実装例を示す。

```
public class BaseController : Controller
{
    #region DI
    [Inject]
    public ILogger Logger { get; set; }
    #endregion

    /// <summary>
    /// アクション メソッドの実行前に呼び出されます。
    /// </summary>
    /// <param name="filterContext">フィルター コンテキスト。</param>
    protected override void OnActionExecuting(ActionExecutingContext fi
lterContext)
    {
        if (filterContext == null) throw new System.ArgumentNullExceptio
n("filterContext");
        var values = from v in filterContext.RouteData.Values where v.Va
lue.GetType() != typeof(HttpPostedFileBase) select String.Format(Cultu
reInfo.CurrentCulture, "[{0} : {1}]", v.Key, v.Value);

        IEnumerable<string> queryString = null;
        if (Request.HttpMethod.Equals(HttpMethod.Get.ToString()))
        {
            queryString = from x in Request.QueryString.AllKeys select St
ring.Format(CultureInfo.CurrentCulture, "{0}: {1}", x, Request.QuerySt
ring[x]);
        }
        IEnumerable<string> form = null;
        if (Request.HttpMethod.Equals(HttpMethod.Post.ToString()))
```

```
        {
            form = from x in Request.Form.AllKeys select String.Format(Cu
ltureInfo.CurrentCulture, "{0}: {1}", x, Request.Form[x]);
        }

        StringBuilder sb = new StringBuilder();
        if (filterContext.HttpContext.Session != null)
            sb.AppendFormat("{0}, [SessionID : {1}]", string.Join(", ", v
alues), filterContext.HttpContext.Session.SessionID);
        else
            sb.AppendFormat(string.Join(", ", values));
        if(queryString != null && 0 < queryString.Count())
            sb.AppendFormat(", [QueryString : ¥n{0}]", string.Join(Envir
onment.NewLine, queryString));
        if (form != null && 0 < form.Count())
            sb.AppendFormat(", [Form : ¥n{0}]", string.Join(Environment.
NewLine, form));

        Logger.Info(message: sb.ToString());
    }
}
```

5.4.3. エラーログ

5.1.2 で述べたように、BaseController クラスの OnException メソッドで全てのエラーを捕捉していることから、 BaseController クラスの OnException メソッドにてエラー内容をログ出力する。

5.5. マッパー

本システムではマッパーとして AutoMapper を使用する。以降、マッパーの詳細、及び AutoMapper の利用方法について述べる。

5.5.1. マッパーとは

Web アプリケーションを設計する際に、ユーザインタフェースとビジネスロジックを分離するために、3 層アーキテクチャが適用される。この 3 層アーキテクチャでは、構成をプレゼンテーション層、ビジネスロジック層、データアクセス層に分ける。3 層アーキテクチャを導入することで、ユーザインタフェース、ビジネスロジック、データアクセスの結合度を下げ、ある層の変更による他の層への影響を減らすことが出来る。これにより、Web アプリケーションの保守性、拡張性が向上する。

3 層アーキテクチャを実現するため、画面を表示するために使用するオブジェクト、リクエストを送受信するために使用するオブジェクト、データベースにデータを格納するために使用するオブジェクトを使用する。それぞれのオブジェクトで求めるプロパティが若干異なり、オブジェクト間に微妙な差異が生じることがある。この差異のことをインピーダンスミスマッチという。コーディング作業の中で大きな割合を占めるのがインピーダンスミスマッチを解消する作業である。インピーダンスミスマッチを解消する作業にかかる時間を削減するツールとしてマッパーが用いられる。

マッパーとは、オブジェクトの詰め替えを自動で行うツールである。マッパーを利用することで、オブジェクトの詰め替え作業を都度記述する必要が無くなるため、コーディング作業量を削減することが出来る。また、オブジェクト間の対応を局所化することが出来るため、仕様変更による影響範囲を減らすことが出来る。

5.5.2. AutoMapper

NuGet から AutoMapper を取得する。App_Start 配下の AutoMapperConfig クラスにオブジェクト間の対応を記述する。下記はインシデントをエンティティから DTO に変換する例である。

```
public class AutoMapperConfig
{
    public static void RegisterAutoMappings()
    {
        Mapper.Initialize(cfg =>
        {
            cfg.CreateMap<PAJ_C_INCIDENT, IncidentDto>()
                .ForMember(d => d.Id, o => o.MapFrom(s => s.INCIDENT_ID))
                .ForMember(d => d.Name, o => o.MapFrom(s => s.INCIDENT_NAM
E))
                .ForMember(d => d.StartedDate, o => o.MapFrom(s => s.INCID
ENT_START_DATE.ToString("yyyy/MM/dd HH:mm:ss")))
                .ForMember(d => d.EndedDate, o => o.MapFrom(s => s.INCIDEN
T_END_DATE != null ? s.INCIDENT_END_DATE.Value.ToString("yyyy/MM/dd HH:
mm:ss") : string.Empty))
                .ForMember(d => d.Visibled, o => o.MapFrom(s => s.VISIBLE
D))
                .ForMember(d => d.Deleted, o => o.MapFrom(s => s.DELETE_FL
AG));
        });
    }
}
```

Global.asax の Application_Start メソッドで AutoMapperConfig クラスの RegisterAutoMappings メソッドを呼び出す。

```
public class MvcApplication : System.Web.HttpApplication
{
    protected void Application_Start()
    {
        AutoMapperConfig.RegisterAutoMappings();
    }
}
```

実際にマッピングを行う際は、下記のように記述する。

```
public class IncidentDao : IIncidentDao
{
    [Inject]
    public IEntities Entities { get; set; }

    /// <summary>
    /// 現在有効なインシデントを取得します。
    /// </summary>
    /// <returns>現在有効なインシデント一覧</returns>
    public IList<IncidentDto> FindByCurrentlyValid()
    {
        return  Mapper.Map<List<IncidentDto>>(Entities.PAJ_C_INCIDENT.Wh
ere(e => e.DELETE_FLAG == 0 && e.VISIBLED == "0").OrderByDescending(e =
> e.INCIDENT_START_DATE).ToList());
    }
}
```

6. Map サービス

6.1. 例外ハンドリング

Web サービスと同様である。詳細は 5.1 を参照すること。

6.2. データアクセス

レイヤ画像やレイヤ情報、フィーチャの取得は Arc GIS Server から行う。以降、クライアントが Map サービスを介して Arc GIS Server と通信する方式を示す。

6.2.1. Arc GIS Server REST API

クライアントから受け取ったリクエストを基に Arc GIS Server REST API で要求するリクエストパラメータを生成する。また、クライアントから受け取ったリクエストを基に URL を作成する。以下に、Arc GIS Server REST API の URL 一覧を示す。

対象サービス	Arc GIS Server REST API URL
レイヤ一覧取得	https://<arcGISServerName>/arcgis/rest/services/<layerGroupName>/MapServer/?f=pjson
メモレイヤ設定取得	https://<arcGISServerName>/arcgis/rest/services/<layerGroupName>/FeatureServer/<layerId>?f=pjson
レイヤ画像取得	https://<arcGISServerName>/arcgis/rest/services/<layerGroupName>/MapServer/export
属性テーブルデータ取得	https://<arcGISServerName>/arcgis/rest/services/<layerGroupName>/MapServer/<layerId>/query
メモフィーチャ取得	https://<arcGISServerName>/arcgis/rest/services/<layerGroupName>/FeatureServer/<layerId>/query
メモフィーチャ編集	https://<arcGISServerName>/arcgis/rest/services/<layerGroupName>/FeatureServer/<layerId>/query

[1] arcGISServerName は Arc GIS Server 名を表す。

[2] layerGroupName はレイヤグループ名を表す。

[3] layerId はレイヤ ID を表す。レイヤ ID はレイヤグループ内で一意の識別子である。

リクエストパラメータ及び URL の作成完了後、ASP.NET で提供される HttpClient で Arc GIS Server REST API を呼び出す。Arc GIS Server より取得したデータはクライアントの求めるフォーマットに加工し、返却する。

6.3. DI（Dependency Injection）

Web サービスと同様である。詳細は 5.3 を参照すること。

6.4. ロギング

Web サービスと同様である。詳細は 5.4 を参照すること。

7. 権限及び機能

7.1. 権限の種類

分析システムを利用できる権限を以下に示す。

7.1.1. 石連事務局員

管理者権限である。

7.1.2. 対応オペレーター

対応システムで自身に割り当てられた要請に対し、「現地確認」「供給者調整」「配送手配」を行う。

7.1.3. 元売分析閲覧（自社）

元売企業・共同タンク事業者に割り当てられる権限の一つである。分析システムで自社の事業所、給油所について情報を参照することが出来る。

7.1.4. 元売分析閲覧（自社以外）

元売企業・共同タンク事業者に割り当てられる権限の一つである。分析システムで自社以外の事業所、給油所について情報を参照することが出来る。

7.1.5. エネ庁・石油精製備蓄課

経済産業省・資源エネルギー庁に割り当てられる権限の一つである。

7.1.6. エネ庁・石油流通課

経済産業省・資源エネルギー庁に割り当てられる権限の一つである。

7.1.7. エネ庁・資源燃料部政策課

経済産業省・資源エネルギー庁に割り当てられる権限の一つである。

7.2. 機能の種類

分析システムが有する機能を以下に示す。

7.2.1. 共通機能

7.2.1.1. 文字サイズ変更

文字サイズを変更する。初期値はディスプレイのサイズに応じて自動で設定される。

7.2.1.2. ポータル遷移

ポータル画面に遷移する。

7.2.1.3. ログアウト

災害情報収集システムをログアウトする。

7.2.2. 主題図一覧

7.2.2.1. 一覧表示

主題図一覧画面を表示する。対応オペレータかそれ以外の権限かで表示される主題図は異なる。

1. 対応オペレータの場合

   共有が有効でありかつ自身が担当するインシデントに紐付く、または自身が作成した主題図
2. それ以外の場合

   共有が有効または自身が作成した主題図

7.2.2.2. 主題図を開く

主題図一覧画面から対象の主題図を開く。

7.2.2.3. 新規作成

対応オペレータかそれ以外の権限かで動作が異なる。

1. 対応オペレータの場合

   自身が担当するインシデントの主題図が新規作成される。
2. それ以外の場合

   インシデント選択画面が開き、インシデントを選択することで新しく主題図が作成される。

7.2.2.4. コメント編集

主題図一覧画面でコメントを直接編集する。自身が作成した主題図のみ編集可能である。

7.2.2.5. 共有可否変更

主題図一覧画面で共有可否を変更する。自身が作成した主題図のみ変更可能である。

7.2.2.6. 削除

主題図一覧画面から対象の主題図を削除する。

7.2.3. 外部機関提供地図一覧

7.2.3.1. 一覧表示

共有が有効または自身が作成した外部機関提供地図を一覧表示する。

7.2.3.2. 新規作成

外部機関提供地図を新規作成する。

7.2.3.3. コメント編集

外部機関提供地図一覧画面でコメントを直接編集する。自身が作成した外部機関提供地図のみ編集可能である。

7.2.3.4. 共有可否変更

外部機関提供地図一覧画面で共有可否を変更する。自身が作成した外部機関提供地図のみ変更可能である。

7.2.3.5. 削除

外部機関提供地図一覧画面から対象の外部機関提供地図を削除する。

7.2.4. ナビゲーション

7.2.4.1. 移動

地図を移動する。

7.2.4.2. 拡大/縮小

地図を拡大/縮小する。

7.2.4.3. 回転

地図を回転する。

7.2.4.4. 回転初期化

回転した地図を初期化する。

7.2.4.5. 初期化

地図の中心座標とズームレベルを初期化する。

7.2.5. コンテンツ

7.2.5.1. レイヤー覧表示

レイヤ一覧をツリー表示する。

7.2.5.2. レイヤグループ折畳

レイヤグループを折り畳む。

7.2.5.3. レイヤ表示/非表示

チェックボックスの ON/OFF と連動してレイヤの表示/非表示を切り替える。

7.2.5.4. テーブル表示

レイヤのラベルをクリックした際にテーブルを表示する。

7.2.6. 外部機関提供地図追加

7.2.6.1. 一覧表示

追加可能な外部機関提供地図一覧を表示する。

7.2.6.2. 追加

外部機関提供地図一覧から対象の外部機関提供地図を地図に追加する。

7.2.6.3. 削除

追加した外部機関提供地図を地図から削除する。

7.2.7. テーブル

7.2.7.1. 表示

テーブルを表示する。

7.2.7.2. 詳細被災総合判定

選択した事業所の被災総合判定を行う。複数の事業所に対して一括で被災総合判定を行うことが出来る。

7.2.7.3. 重要項目表示/非表示

重要項目の表示/非表示を切り替える。

7.2.7.4. ラベル表示/非表示

ラベルレイヤの表示/非表示を切り替える。

7.2.7.5. 透過率変更

選択したレイヤの透過率を変更する。

7.2.7.6. 初期化

フィルタリング条件、ソート等を全て初期化し、最新のデータを取得する。

7.2.7.7. 更新

フィルタリング条件、ソート等を維持したまま、最新のデータを取得する。

7.2.7.8. フィルタリング

選択したレイヤの取得条件を設定する。フィルタリング条件はレイヤ毎に維持され、主題図保存した場合は条件も保存される。

7.2.7.9. 簡易総合判定

被災状況プルダウンリストから被災状況を選択し、対象の事業所の総合判定を簡易的に行う。

7.2.8. メモ

7.2.8.1. 追加

アイコンを選択もしくはライン/ポリゴンを描画し、これにコメント等を付与したものをメモとし、地図に追加する。

7.2.8.2. 編集

地図に追加されたメモを編集する。自身が作成したメモのみ編集可能である。

7.2.8.3. 削除

地図に追加されたメモを削除する。自身が作成したメモのみ削除可能である。

7.2.9. 日時コントロール

7.2.9.1. リアルタイム

レイヤ、テーブルを定期的に最新に更新する。

7.2.9.2. 日時設定

カレンダー、リストから日時を設定し、指定した日時のレイヤ、テーブルを取得する。

7.2.10. 主題図保存

7.2.10.1.別名保存

主題図を新規で保存する。地図の中心座標、ズームレベル、レイヤの表示状態、フィルタリング条件、外部機関提供地図追加状態を保存する。

7.2.10.2.上書保存

主題図を上書保存する。保存内容は別名保存と同様である。

7.3.　各権限の閲覧・機能制限について

7.3.1.　各権限で閲覧可能な情報

以下に各権限で閲覧可能な情報を示す。

表 7.1. 各権限で閲覧可能な情報

情報 / 権限	対応情報		被災状況		メモ
	要請情報	候補施設情報	事業所情報	給油所情報	
石連事務局員	○（全て）	○（全て）	○（全て）	○（全て）	○※
対応オペレータ	○（自担当）	×	×	×	○※
エネ庁・石油精製備蓄課	○（全て）	×	○（全て）	○（全て）	○※
エネ庁・石油流通課	○（全て）	×	○（全て）	○（全て）	○※
エネ庁・資源燃料部政策課	○（全て）	×	○（全て）	○（全て）	○※
元売分析閲覧（自社）	×	×	○（自社）	○（自社）	○※
元売分析閲覧（自社以外）	×	×	○（自社以外）	○（自社以外）	○※
元売分析閲覧（自社&自社以外）	×	×	○（全て）	○（全て）	○※

※共有しているメモは全て閲覧可能である。ただし、非共有のメモは自身が登録したもののみ閲覧可能とする。

7.3.2.　各権限で使用可能な機能

以下に各権限で使用可能な機能を示す。

表 7.2. 各権限で使用可能な機能

機能 / 権限		石連事務局員	対応オペレータ	エネ庁・石油精製備蓄課	エネ庁・石油流通課	エネ庁・資源燃料部政策課	元売分析閲覧（自社）	元売分析閲覧（自社以外）	元売分析閲覧（自社&自社以
共通機能	文字サイズ変更	○	○	○	○	○	○	○	○
	ポータル遷移	○	○	○	○	○	○	○	○
	ログアウト	○	○	○	○	○	○	○	○

主題図一覧	一覧表示	○	○	○	○	○	○	○	○
	主題図を開く	○	○	○	○	○	○	○	○
	新規作成	○	○	○	○	○	○	○	○
	コメント編集	○	○	○	○	○	○	○	○
	共有可否変更	○	○	○	○	○	○	○	○
	削除	○	○	○	○	○	○	○	○
外部機関提供地図一覧	一覧表示	○	○	○	○	○	○	○	○
	新規作成	○	○	○	○	○	○	○	○
	コメント編集	○	○	○	○	○	○	○	○
	共有可否変更	○	○	○	○	○	○	○	○
	削除	○	○	○	○	○	○	○	○
ナビゲーション	移動	○	○	○	○	○	○	○	○
	拡大/縮小	○	○	○	○	○	○	○	○
	回転	○	○	○	○	○	○	○	○
	回転初期化	○	○	○	○	○	○	○	○
	初期化	○	○	○	○	○	○	○	○
コンテンツ	レイヤ一覧表示	○	○	○	○	○	○	○	○
	レイヤグループ折畳	○	○	○	○	○	○	○	○
	レイヤ表示/非表示	○	○	○	○	○	○	○	○
	テーブル表示	○	○	○	○	○	○	○	○
外部機関提供地図追加	一覧表示	○	○	○	○	○	○	○	○
	追加	○	○	○	○	○	○	○	○
	削除	○	○	○	○	○	○	○	○
テーブル	表示	○	○	○	○	○	○	○	○
	詳細被災総合判定	○	×	×	×	×	×	×	×
	重要項目表示/非表示	○	○	○	○	○	○	○	○
	ラベル表示/非表示	○	○	○	○	○	○	○	○
	透過率変更	○	○	○	○	○	○	○	○
	初期化	○	○	○	○	○	○	○	○
	更新	○	○	○	○	○	○	○	○
	フィルタリング	○	○	○	○	○	○	○	○
	簡易被災総合判定	○	×	×	×	×	×	×	×
メモ	追加	○	○	○	○	○	○	○	○
	編集	○	○	○	○	○	○	○	○
	削除	○	○	○	○	○	○	○	○
日時	リアルタイム	○	○	○	○	○	○	○	○
	日時設定	○	○	○	○	○	○	○	○
主題図保存	別名保存	○	○	○	○	○	○	○	○
	上書保存	○	○	○	○	○	○	○	○

8. 開発ガイド

8.1. レイヤ

8.1.1. 概要

各レイヤで保持するプロパティをサーバサイドで定義する。サーバサイドで定義する理由を下記に示す。

- 画面に表示するレイヤの項目名、順序、重要項目、フィルタリングに使用する項目等アプリケーションの仕様により変更する可能性のある設定を一元管理するため。
- これまでフィルタリングで検証していなかったデータ型、入力文字数等をサーバサイドで検証できるようにするため。
- Arc GIS Server の設定を変更することなく、アプリケーションの改修のみで各レイヤのプロパティの設定を行うことができるようにするため。

8.1.2. LayerMetadata

各レイヤのプロパティを下記フォーマットで保持する。

- LayerId
  レイヤを一意に識別する ID。
- LayerGroupCd
  レイヤグループコード。対応情報、被災状況、メモのいずれのグループかを表す。
- LayerTitle
  属性テーブルに表示するレイヤのタイトル。
- IsFilteringEnabled
  フィルタリングを使用できるレイヤかを表すフラグ。
- IsSimpleJudgementEnabled
  簡易総合判定を使用できるレイヤかを表すフラグ。
- IsOverallJudgment
  詳細総合判定を使用できるレイヤかを表すフラグ。
- MetadataList
  レイヤに紐付くメタデータ一覧。

8.1.3. Metadata

各レイヤの各項目を下記フォーマットで保持する。

各項目に保持する内容については、属性テーブル項目一覧.xlsm を参照。

- Id

  Metadata を一意に識別する識別子。

- Key

  Arc GIS Server で定義されている項目名。

- DispTitle

  画面に表示する項目名。

- DataTypeCd

  型コード。

- MaxLength

  最大入力可能文字数。

- DispOrder

  表示順。

- IsImportant

  重要項目フラグ。

- IsFilteringColumn

  フィルタリング項目フラグ。

- IsEnabled

  属性テーブルデータとして利用するかを表すフラグ。

- WidthLength

  画面に表示する際の項目幅[1]。

- IsFixed

  画面に表示する際の項目幅が固定なのかを表すフラグ。

8.2. レイヤ定義

8.2.1. 概要

Arc GIS Server REST API でレイヤ画像を取得する場合下記のようなリクエストを送る。

```
https://arcgis/rest/services/dev_pajdis_request/MapServer/0/query?retu
rnGeometry=true&geometryType=esriGeometryEnvelope&geometry={"xmin":118
51371.4285714,"ymin":2660000,"xmax":19048628.5714286,"ymax":5940000,"s
patialReference":{"wkid":102100}}&inSR=102100&spatialRel=esriSpatialRe
lIntersects&where=対象日 <= '201801151616' and '201801151616' < 報告対象
日時_T and インシデントID = '43E42961-0229-43E9-9DAA-D8B1F6225A55' and '
```

[1] 単位は em

```
201610270000' <= 対象日&outSR=102100&outFields=*&f=json&
```

このリクエストではレイヤの取得条件を where に設定している。このレイヤの取得条件のことを「レイヤ定義」と呼ぶ。

このレイヤ定義をフロントエンドで作成する場合、悪意のあるユーザによってレイヤ定義が自由に書き換えられてしまうことから、セキュリティ上望ましくない。従って、本システムでは 8.2.2 に示す LayerDefinitionDto でレイヤ定義に必要なパラメータをクライアントから送り、サーバサイドでレイヤ定義を構築することとする。

8.2.2. LayerDefinitionDto

下記に保持する内容を示す。

- LayerId
  対象のレイヤ ID。レイヤ ID はレイヤグループコード毎に一意である。
- SymbolLabelCd
  シンボルかラベルを表すコード。コードに応じてシンボルレイヤかラベルレイヤかを判定する。
- LayerGroupCd
  レイヤグループコード。対応情報、被災状況、メモのいずれのグループかを表す。
- IncidentId
  対象のインシデント ID。
- StartDate
  開始日。
- EndDate
  終了日。
- FilteringConditions
  フィルタリング条件式一覧。フィルタリングについては 8.3 に記載する。

8.3. フィルタリング

8.3.1. 概要

状況が「登録」かつ対応に「石連」を含むシンボルのレイヤ画像を取得する場合、Arc GIS Server REST API に下記のようなリクエストを送る。

```
https://baltan.science.nttdata-ccs.co.jp/dev/Portal/Bunseki/app/proxy.ashx?https://arcgis/rest/services/dev_pajdis_request/MapServer/0/query?returnGeometry=true&geometryType=esriGeometryEnvelope&geometry={"xmin":11851371.4285714,"ymin":2660000,"xmax":19048628.5714286,"ymax":5940000,"spatialReference":{"wkid":102100}}&inSR=102100&spatialRel=esriSpatialRelIntersects&where=対象日 <= '201801151616' and '201801151616' < 報
```

```
告対象日時_T and インシデントID = '43E42961-0229-43E9-9DAA-D8B1F6225A55'
and '201610270000' <= 対象日 and (  状況 = '登録' AND 対応 LIKE '%石連%' )&o
utSR=102100&outFields=*&f=json&
```

このリクエストではレイヤ定義に詳細なレイヤ取得条件を加えている。

8.2.1 で述べたように、レイヤ定義をクライアントで作成する場合、悪意のあるユーザによって詳細なレイヤ取得条件が自由に書き換えられてしまうことから、セキュリティ上望ましくない。従って、本システムでは 8.3.3 で示す FilteringCondition でフィルタリングに必要なパラメータをクライアントから送り、サーバサイドでフィルタリング条件を構築することとする。

8.3.2. 対象

対応情報、被災状況をフィルタリングの対象レイヤグループとする。また、レイヤの表示、属性テーブルの表示を対象サービスとする。

8.3.3. FilteringCondition

下記にフィルタリング条件で保持する内容を示す。

- MetadataId
  選択した項目のメタデータ ID。サーバサイドで定義した Metadata.Id に対応する。
- Value
  入力値。
- ConditionalOperatorCd
  条件演算子コード。
- AndOrCd
  かつまたはコード。

上記の設計により、下記メリットが受けられる。

- MetadataId を使用することで、悪意のあるユーザから項目名による不正な条件式の構築を防ぐ。
- ConditionalOperatorCd を使用することで、悪意のあるユーザから条件演算子による不正な条件式の構築を防ぐ。
- IsFilteringColumn が true な Metadata の Id のみを条件式として使用するだけで不正な条件式の構築を防ぐ。

8.3.4. 検証項目

フィルタリングでは下記項目を検証対象とする。

- 選択項目がフィルタリング対象項目であること
  Metadata.IsFilteringColumn を確認すればよい。
- 入力値が選択項目の型であること

Metadata.DataTypeCd から判定すればよい。

- 入力値の文字数が選択項目の最大入力可能文字数以下であること
  Metadata.MaxLength と比較すればよい。
- 選択された条件式が選択項目で使用可能な条件式であること
  データ型で使用可能な条件演算子コード一覧を定数として定義し、それと比較すればよい。
- 入力値が SQL インジェクション攻撃に該当しないこと
  正規表現で SQL インジェクション攻撃に使用されるトークンが含まれないか検証すればよい。

8.3.5. 条件式構築

必要に応じて下記項目を設定し、条件式を構築する。

- かつまたはコードに応じて AND/OR を設定する。
- 条件演算子コードに応じて条件演算子を設定する。
- サーバサイドで定義したメタデータからデータ型を取得し、データ型から入力値を加工、設定する。
- サーバサイドで定義したメタデータから、Arc GIS Server で定義されている項目名を取得、設定する。

8.4. コンテンツ

8.4.1. 方式

本システムでは、クライアントの負荷を軽減するため、Map サービスで Arc GIS Server REST API を呼び出し、取得したレイヤ一覧を jsTree で認識できるフォーマットに加工する。

8.4.2. JsTreeNode

jsTree のノードを表す抽象クラス。jsTree の表示に必要なプロパティを持たせる。

- Id
  jsTree のノードを一意に識別する
- Parent
  jsTree の親ノード
- Text
  jsTree で表示するテキスト
- AAttr
  jsTree の a タグ属性
- State
  Jstree で node の状態を設定する

8.4.3. LayerDto

レイヤを表すクラス。JsTreeNode を継承する。

- LayerId
  レイヤを一意に識別する識別子
- LayerGroupCd
  レイヤグループコード
- SubLayerIds
  サブレイヤ ID 一覧

8.4.4. イベントハンドラ

ツリーの折り畳みは jsTree で提供されているため、実装する必要が無い。しかし、チェックボックス ON/OFF によるレイヤの表示/非表示やラベルクリックによる属性テーブルの表示等は別途実装する必要がある。

8.4.4.1. check_node.jstree

jsTree が提供しているイベントで、ノードがチェックされた際に発火する。イベントハンドラにこのイベントを追加することで、チェックボックスを ON にしたときのレイヤの表示を実装することが出来る。

8.4.4.2. uncheck_node.jstree

jsTree が提供しているイベントで、ノードのチェックが外れた際に発火する。イベントハンドラにこのイベントを追加することで、チェックボックスを OFF にしたときのレイヤの非表示を実装することが出来る。

8.4.4.3. select_node.jstree

jsTree が提供しているイベントで、ノードが選択された際に発火する。イベントハンドラにこのイベントを追加することで、レイヤを選択したときの属性テーブル表示を実装することが出来る。

8.5. レイヤ画像

8.5.1. 概要

レイヤにはシンボルレイヤとラベルレイヤの 2 種類が存在する。シンボルレイヤはアイコンを、ラベルレイヤはテキストを表示する。

本システムではレイヤを画面サイズの画像 1 枚で表す。

8.5.2. 対象

レイヤを画像で取得する対象は、対応情報のシンボル/ラベル、被災状況のシンボル/ラベル、メモの

ラベルである。

8.5.3. 方式

8.5.3.1. レイヤ画像の取得

Map サービス（LayerController.MapServer）でレイヤ画像を取得する。Map サービスでは、クライアントから LayerDefinitionDto を受け取り、レイヤ定義を構築し、Arc GIS Server REST API の MapServer よりレイヤ画像を取得し、返却する。

8.5.3.2. レイヤ画像の描画

フロントエンドでは OpenLayers の ImageArcGISRest で扱う。ImageArcGISRest の仕様により、Map サービスのサービス名は MapServer とする必要がある。

8.6. メモ

8.6.1. 概要

本システムでは、地図上に配置したアイコンや線、図形にコメント等を付与したものを「メモ」と呼ぶ。GIS ではアイコンや線、図形を「フィーチャ」といい、それぞれをポイント、ライン、ポリゴンと分類する。ポイントは点で表され、1 つの座標で表現できる。ラインは線で表され、例えば直線の場合は始点と終点の 2 つの座標で表現できる。ポリゴンは面で表され、例えば長方形の場合は頂点の 4 つの座標で表現できる。このように分類によって幾何情報が異なる。この幾何情報のことを「ジオメトリ」という。

8.6.2. レイヤ表示

8.5.1 で述べたように、対応情報のシンボル/ラベル、被災状況のシンボル/ラベル、メモのラベルではレイヤ画像を取得/表示しているが、メモのシンボルではフィーチャ一覧をサーバから取得し、ベクターレイヤを表示している。理由としては、フィーチャをサーバから取得することにより画面で直接フィーチャの選択を行うことが出来るからである。本システムでは、フィーチャを選択した際に、編集ダイアログを表示する。編集の仕様については 8.6.4 で述べる。

8.6.2.1. フィーチャー覧の取得

Map サービス（LayerController.FetchMemoLayerFeaturesAsync）でフィーチャ一覧を取得する。Map サービスでは、クライアントから LayerDefinitionDto を受け取り、レイヤ定義を構築し、Arc GIS Server REST API の FeatureServer よりフィーチャ一覧を取得し、返却する。

8.6.2.2. ベクターレイヤの描画

フロントエンドでは OpenLayers の Vector で扱う。

8.6.3. 登録

本システムでは、画面でフィーチャを描画する。また、ライン、ポリゴンについてはフリーハンドで作成することが出来る。フィーチャの描画が完了次第、情報を入力し、Map サービスにリクエストを送信し、サーバサイドで Arc GIS Server REST API を呼び出してメモを登録する。

8.6.4. 編集

OpenLayers で地図のクリックイベントを拾う。イベントでクリックした座標を取得し、その座標の近傍にフィーチャが存在するかをチェックする。該当するフィーチャが存在した場合は編集ダイアログを表示する。

フィーチャの編集が完了次第、Map サービスにリクエストを送信し、サーバサイドで Arc GIS Server REST API を呼び出してメモを更新する。

8.6.5. 削除

メモは物理削除せず、論理削除とする。削除のフローは編集と同様である。

8.7. 日時コントロール

8.7.1. 概要

本システムでは日時を指定することで、レイヤ/テーブルの取得条件を変更することが出来る。また、リアルタイムのチェックを ON にすることで、定期的に最新のレイヤ/テーブルを取得することが出来る。

8.7.2. 方式

8.7.2.1. 日時指定可能範囲

インシデントの開始日から終了日 23:59 までを選択可能範囲とする。

8.7.2.2. 日時指定

リアルタイムのチェックが OFF の場合、日時指定が可能とする。日時を指定した場合、インシデントの開始日から指定日時までを取得範囲とし、レイヤ/テーブルを更新する。

8.7.2.3. リアルタイム

終了日が設定されていないインシデントのみ、リアルタイムを有効にすることが出来ることとする。リアルタイムのチェックが ON の場合、日時選択を不可とする。また、5 分毎にインシデントの開始日からシステム時刻までを取得範囲とし、レイヤ/テーブルを更新する。

8.8. 主題図

8.8.1. 概要

主題図では、下記情報を保持する。

- 中心座標
- ズームレベル
- レイヤ表示状態
  レイヤのチェック状態を保持する。
- 外部機関提供地図レイヤ表示状態
  コンテンツに追加する外部機関提供地図と外部機関提供地図レイヤのチェック状態を保持する。
- フィルタリング条件
  各レイヤのフィルタリング条件を保持する。