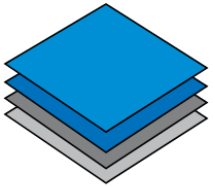
WebGIS初級編

OpenLayersで簡単作成



FOSS4G Hokkaido(jp.foss4g.hokkaido@gmail.com)

- この資料はFOSS4G 2013 Hokkaidoハンズオンセッション「 WebGIS初級編 - OpenLayersで簡単作成」の配付資料に加筆・修正したものです
- 配布データは FOSS4G 2013 Hokkaido ハンズオンセッションのページ (https://sites.google.com/site/foss4ghokkaido/foss4g-2013-hokkaido/handson)からダウンロードしてください

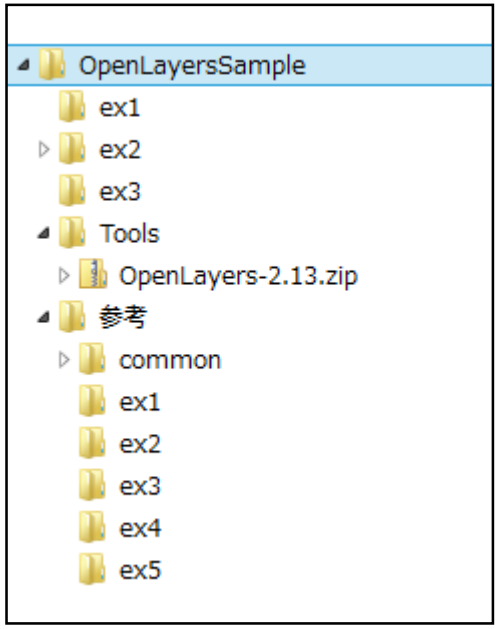
はじめに

- 代表的な地図表示用のJavaScriptライブラリ　「OpenLayers」を使用して、基本的なWebGIS(データビューワー)を作成します。

- 演習はインターネット接続もサーバーが無くても実行できる範囲で行います。
- 資料内でフォルダを表すときは、末尾に/(スラッシュ)を付けています。
- 配付資料からソースコードをコピー&ペーストすると正しく動作しないことがあります。配布データにある「参考」フォルダからコピーしてください。

配布データについて

- 配布したデータを適当な場所に展開します。
  "c:¥OpenLayersSample"など



ex1/ ex2/ ex3・・・演習作業フォルダ

Tools・・・OpenLayersのライブラリや
テキストエディタなどのツール

参考・・・実習の最終コード
実習によってはソースが掲載しきれないので
ここのコードを参考にしてください

本セッションの内容

○OpenLayers概説

- OpenLayersとは
- OpenLayersで出来ること
- GoogleMaps APIとなにが違う?

○地図を表示する

- OpenLayersの基本インストール
- 一枚の画像を表示する
- タイル画像を表示する
- ベクトル(ポイント・ライン・ポリゴン)を表示する

○GISソフトらしくする(コントロール)

- コントロールとは?
- 主なコントロールを一通り使ってみる
- 図形をクリックして属性表示

○+α(開発のコツなど)

- オンラインマップの紹介
- OpenLayersとJavaScriptライブラリの組み合わせ
- 参考リンクなど

OpenLayers概説

OpenLayersとは

OpenLayersはJavaScriptで作成された地図表示用のライブラリです。

サーバーサイドで動作するWebGISからの出力や各種対応地図データの読み込みと表示、ブラウザーサイドでのユーザーインタフェース部分を担当します。
OpenLayersを使用することで、Googleマップに代表される画面遷移無しでのスクロール、拡大縮小といった動作を実現することができます。

ブラウザだけでなく、QGISをはじめとするアプリケーションに組み込まれたり、
最近では電子国土Webシステムも従来の独自プラグインから、OpenLayersベースとなりました。

OpenLayersで出来ること-1

- OpenLayersで出来ること
  - 画像やポイント・ライン・ポリゴンやテキスト(注記)を表示する。
  - 画像は1ファイルだけでも、ルールに基づいたファイル名の画像群を敷き詰めて表示すること(タイリング)もできる。
  - ベクトルデータの表示色、サイズなどの設定ができる。
  - WebGISエンジン(Mapserver、GeoServerなど)に動的な画像生成をリクエストし、その結果を表示できる。
  - レイヤの表示/非表示、透過表示などができる。
  - スクロール表示、拡大縮小表示を連続的に行える。
  - 「コントロール」を使用してGISに必要なUIを提供できる。
    スケール・スケールバー、移動・拡大縮小、位置図、レイヤ一覧、
    クリックによるイベント発生、属性表示 etc...
  - 座標の投影変換などの演算APIがある。
  - ブラウザ間の違いを吸収してくれる。

OpenLayersで出来ること-2

- OpenLayersで出来ないこと
  - バイナリGISデータ(Shapefile、GeoTIFFなど)を直接扱うこと
  - データベース(MySQL、PostGIS、Spatialliteなど)に直接接続すること
  - 乗り換え・経路探索、住所と座標の相互変換(ジオコーディング)などの地理情報サービス

注)頑張れば出来なくも無いですが・・・

OpenLayersで出来ること = JavaScriptで出来ること

「出来ないこと」は他のJavaScriptライブラリやサーバー(WebGISエンジン、外部サービス)と組み合わせて実現する。

Google Maps APIとなにが違う?

- Web地図の代表格と言えばGoogleマップ
- Google Maps APIを利用するとWebGISが作成できる
  - マーカーの設置、ベクター図形の描画
  - 経路検索、ジオコーディング。その結果の描画。
  - コントロールのカスタマイズ
  - 地図表現のカスタマイズ
- 上記のようなカスタマイズが比較的簡単にできる

Google Maps APIとの比較表

ライセンスの詳細、制限事項に関してはGoogleに確認してください

	OpenLayers	Google Maps API(無料の範囲)
商用利用	OK(制約無し)	基本OKだが制約あり アクセス数制限やサイトの一般公開義務
背景地図	どのサービス、データでもOK ただし自分で確保する	Googleマップ利用が原則 地図以外のデータだけ使いたい→NG
経路探索 ジオコーディング	自分でサービスを確保する。 専用APIは無い。	Googleが提供しているサービスをAPIで利用 地図以外のデータだけ使いたい→NG
インターネット 接続	オフラインでも使用できる サーバーレスでも使用できる	必須
ソース解析 カスタマイズ	オープンソースなので自由 ソース、ツール一式が用意されている	オープンソースでは無い 難読化されているのでデバッグも難しい
オーバーレイ 外部データ利用	豊富なAPI(クラス)で簡単に利用できる	用意されているAPIは便利で強力 反面APIに合わせるのが面倒なケースも

使い分けの基準

- Google Maps(サービス)とGoogle Maps APIは不可分
- Googleが提供する強力なサービスを利用するにはGoogle Maps API利用が大前提
- Googleのサービスは運用環境に制約がある
- OpenLayersでGoogleマップの地図画像を表示できる

∴OpenLayersとGoogle Maps APIは意外にバッティングしない

要件に適したライブラリを使いましょう

地図を表示する

インストール-1

最小限(基本)構成

1. OpenLayersのサイトからアーカイブファイル(zip/tar.gz)をダウンロード(今回はOpenLayers-2.13.zipを配布します)し、展開する
2. 公開に最小限必要な以下のファイル、フォルダをセットで(ブラウザで参照できる場所に)コピーする

   注)ファイル・フォルダの相対位置を変えない

   - OpenLayers.js (ライブラリソース)
   - img/ (アイコン、UI部品)
   - theme/ (スタイルシートとUI部品)
3. HTMLソースには OpenLayers.jsをリンクする

- リファレンスマニュアルはdoc/apidocs/index.html

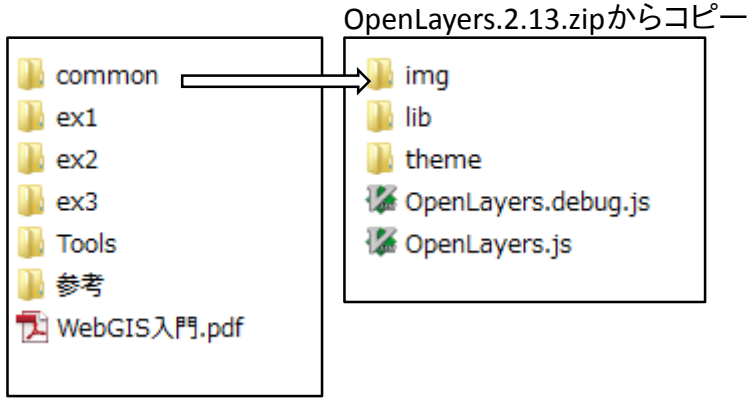
インストール-2

開発(デバッグ)構成

1. 最小構成に加えて以下をコピーする
   - lib/ (OpenLayers.jsの元になっているモジュール毎のJavaScriptソース群)
   - OpenLayers.debug.js (OpenLayers.jsの非圧縮版。lib/をまとめただけ)

2. HTMLソースには lib/OpenLayers.js、またはOpenLayers.debug.jsをリンクする
   自分が読みやすい方、使いやすい方を選択する。
   本セッションではOpenLayers.debug.jsを使用します。

3. 本セッションでは配付資料を展開したフォルダに
   common/ フォルダを作成、開発構成の
   ファイル・フォルダをコピーして使用する。



実習1

一枚の画像を表示する

(OpenLayers.Layer.Image)

実習1 - 一枚の画像を表示する

- この実習では、OpenLayersの基本的な構成を学ぶために、一枚の画像を表示するスクリプトを作成する。

1. 実習データのex1/を使用する。
   このフォルダのmap.png(画像サイズ=1280x720)を表示する。
2. index.htmlをテキストエディタとウェブブラウザで開く。
   ウェブブラウザはデバッグコンソールも起動する(エラーチェックのため)。
3. headにcommon/OpenLayers.debug.jsとmap.cssへのリンクを追加。
4. map.js、mac.cssを次ページ以降のように書き換える。
5. ブラウザをリロードして動作確認する。
6. numZoomLevelsとresolutionsのコメントアウトを入れ替えて動作確認する。

実習1 - map.js

ソースコード(map.js)

```
window.onload = function () {
 var map = new OpenLayers.Map(
    "map", // 地図を表示するdivのID
    {
      numZoomLevels: 4 // 地図の表示段階数
      // resolutions: [0.25, 0.5, 1.0, 2.0] // 地図の表示解像度
    }
  );
  var layer = new OpenLayers.Layer.Image(
    "Image Layer", // レイヤ名
    "map.png",  // 画像のURL
    new OpenLayers.Bounds(0,0,1280,720), // 画像を投影する矩形の左下・右上座標
    new OpenLayers.Size(1280,720) // 画像サイズ
  );
  map.addLayers([layer]);
  map.setCenter(new OpenLayers.LonLat(640,360));
};
```

OpenLayersの基本フロー

Mapオブジェクト生成
↓
Layerオブジェクト生成
↓
MapにLayerを追加する
↓
初期表示位置を指定

※このフローは一例です

実習1 - index.html, map.css

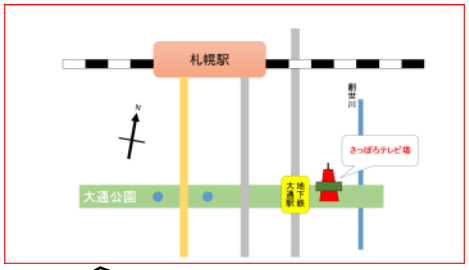
ソースコード(index.html)

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>OpenLayers Example - 1</title>
    <link rel="stylesheet" href="map.css" type="text/css">
    <script src="map.js"></script>
    <script src="../common/OpenLayers.debug.js"></script>
  </head>
  <body>
    <h1 id="title">OpenLayers Example</h1>
    <div id="map" class="mapdiv"></div>
  </body>
</html>
```

ソースコード(map.css)

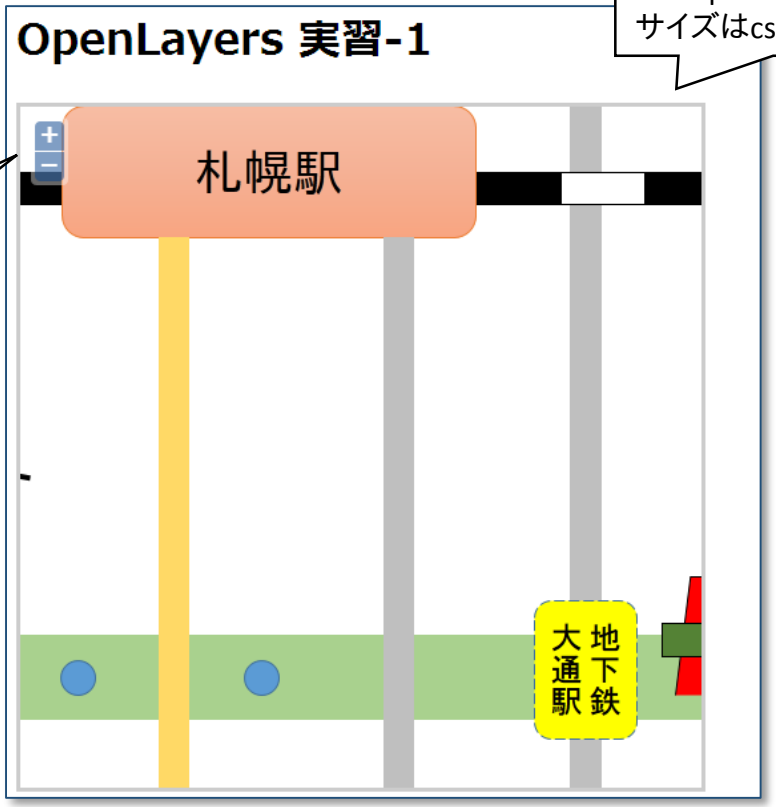
```
.mapdiv {
  width: 512px;
  height: 512px;
  border: 3px solid #ccc;
}
```

実習1 - 実行結果



表示する画像ファイルは
正しい地図ではない

+/-クリックで
拡大縮小



id:mapのdivに表示される
サイズはcssで指定

実習1 - ポイント

- window.onload()はOpenLayers.jsの実行を待つため。
  <body onload="map_init();">などでも置き換え可
- HTMLの地図を表示する にはCSSで高さ・幅を指定する。
- HTMLの地図を表示する のIDをJavaScriptのOpenLayers.Mapで指定する。
  (DOMElementでもOK)
- このサンプルのように、必ずしも投影法定義や正しい地理座標は必要ない。
- ブラウザの画像拡大、縮小機能を使用するため、resolutionが1.0を離れるほど表示画質が低下する。
- 広範囲を表示するには画素を増やす必要があり、ブラウザやネットワークの負荷を考えると無駄が多く、限界が低い。

実習2

タイル画像を表示する

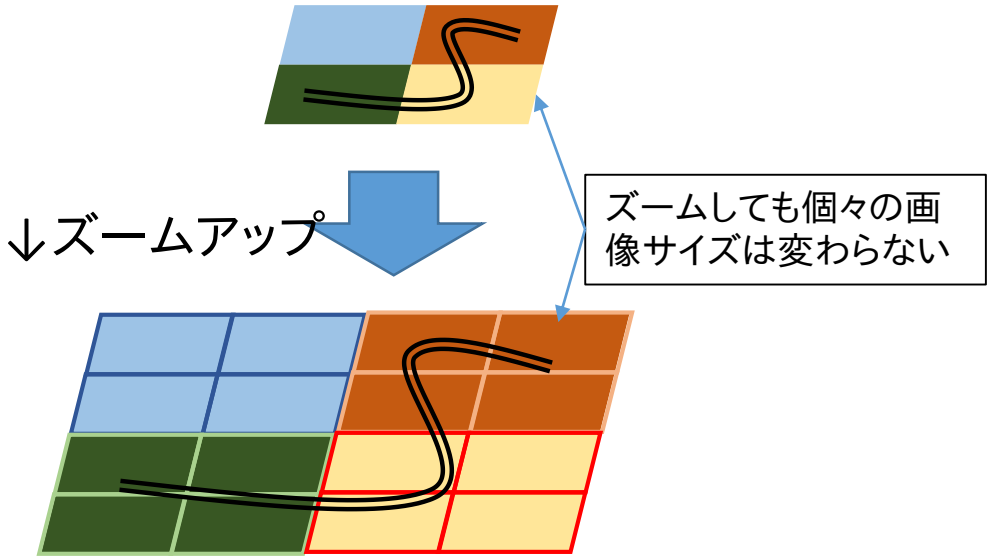
(OpenLayers.Layer.TMS)

実習2 - タイル画像を表示する

- この実習では、実践的かつ扱いやすいデータ形式であるタイル画像を表示する。
- WebGISで言うところのタイル画像とは、ルールに基づいたファイル名(データパス)の複数の画像群である。一般的には静的かつ同一サイズで構成される。
  - Imageレイヤに対するメリット・・・広範囲でも初期表示が高速、縮尺を増やしても高画質
  - 動的生成(WMS)に対するメリット・・・サーバーの負担が低い、単純なストレージのみで提供できる
  - デメリット・・・(広範囲、多スケールになるにつれ)作り置きの時間、ディスクスペースの要求量が増える傾向にある
- 普段見ているGoogleマップもタイル画像。主な地図サービスは殆どがタイル画像で提供されている。(電子国土Version4も!)

実習2 - タイル画像の構造

- タイル画像は右図のように、OpenLayersの縮尺毎にリサイズされた画像を用意する。
- 画像サイズ(tilesize)は全て同じ
- ルールに則ったフォルダ名やファイル名をつける。(独自のルールでも良い)



実習2 - タイル画像を表示する

1. 実習データのex2/を使用する。
   このフォルダのmapimages/landsatフォルダにタイル画像がある。(画像の詳細は後述)
2. index.htmlをテキストエディタとウェブブラウザで開く。
   ウェブブラウザはデバッグコンソールも起動する。
3. map.jsを次ページのように書き換える。
4. ブラウザをリロードして動作確認する。
5. デバッガのネットワークパネルを表示させ、スクロールや拡大縮小による画像読み込みの様子を観察する。
6. デバッガでブレークポイントを関数getTIleURLに設定し、入出力値や計算の過程を観察する。
7. map.cssを編集し、. olTileImage のコメントアウトを外してみる。

実習2 - map.js

ソースコード(map.js)抜粋

ブレークポイント

```
～前略～

  // 画像パスを計算する関数
  // OpenLayers.Layer.TMSのgetURLメソッドをオーバーライドする
  // MapTilerが生成するソースを参考にしている
  var getTileURL = function (bounds) {
    var res = this.map.getResolution(); // thisはOpenLayers.Layer.TMSを指す
    var x = Math.round((bounds.left - googleMaxExtent.left) / (res * this.tileSize.w));
    var y = Math.round((bounds.bottom - this.tileOrigin.lat) / (res * this.tileSize.h));
    var z = this.map.getZoom();

    var mapBounds = this.maxExtent;

    if (mapBounds.intersectsBounds(bounds) && z >= this.mapMinZoom && z <= this.mapMaxZoom ) {
      var imgurl = this.url + this.layername + "/" + z + "/" + x + "/" + y + "." + this.type;
    } else {
      var imgurl = "../common/img/blank.gif";
    }

    return imgurl;
  };

～つづく～
```

実習2 - map.js

ソースコード(map.js)抜粋

```
～つづき～

  var layer = new OpenLayers.Layer.TMS( // Tile Map Service
    "Tile Map Overlay",  // レイヤ名
    "mapimg/", // 画像のあるURL
    { // TMSオプション
      mapMinZoom:7,
      mapMaxZoom:13,
      maxExtent: dataMaxExtent,
      layername: "landsat",
      type: 'jpg', // 画像形式
      getURL: getTileURL // 画像のパス(URL)を求める関数
    });

  map.addLayers([layer]);

～後略～
```

実習2 - 実行結果

実習2 - ポイント

- タイルレイヤでは見える範囲(＋α)の画像だけが読み込まれる。
- JavaScriptのオーバーライドを用いて、APIの振る舞いを変えている。
  これを応用すると、変則的なタイル画像にも対応できる。
  実習1のような手書き・イラスト地図、ノースアップでは無い地図も扱える。
- Imageレイヤと同様、表示画像自体はウェブブラウザで表示できる形式で、ジオデータではない(GISソフトとの違い)。
- 座標値やスケールを把握するために"コントロール"を追加した。
- タイル画像を扱う主なレイヤ。

  ○ローカルファイル(オフライン)にも対応
  - OpenLayers.Layer.TMS
  - OpenLayers.Layer.TileCache

  ○既存サービス向け
  - OpenLayers.Layer.Google
  - OpenLayers.Layer.Bing
  - OpenLayers.Layer.OSM
  - OpenLayers.Layer.WMS(動的生成サーバーにタイルリクエストを送る)

実習2 - タイル画像の作成方法について

- 本セッションでは、GeoTIFF(Landsat)をMapTilerで分割した画像、及びコード(getURLメソッド)を利用している。
  タイルの仕様によってgetURLメソッドの内容も変わる。
- MapTilerはGoogleマップに合わせたタイル画像を生成できるので、
  Google/OpenStreetMap/電子国土と重ねたい場合に便利。
- MapTilerはgdalツールのgdal2tiles.pyがベースとなっている。Windows以外やバッチで走らせたい場合はgdal2tile.pyを使用する。
- 他にもTilecache[http://tilecache.org]のtilecache_seed.pyが画像分割→OpenLayersに便利。
- ImageMagickのconvert -crop ← getURLは自作

実習3

ベクトルデータを表示する

(OpenLayers.Layer.Vector)

実習3 - ベクトルデータを表示する

- この実習では、ベクトルデータの読み込みと表示の基本を学ぶ。
- GISでベクトルと言えばShapefileだが、OpenLayersでは扱えない。
- OpenLayersで利用できるベクトルはテキストファイル(KML、GML、GeoJSONなど)。
- 今回はJSON(JavaScript Object Notation)の地理的拡張であるGeoJSONを使用する。
  GeoJSONは属性も持てる。(オフラインで確実に読み込める外部ファイル形式)

実習3 - ベクトルデータを表示する

1. 実習データのex3/を使用する。
2. index.htmlをテキストエディタとウェブブラウザで開く。
   ウェブブラウザはデバッグコンソールも起動する。
3. headにfeatures.jsへのリンクを追加する。features.jsの内容も確認する。
4. map.jsにレイヤ定義関数(loadPointLayer(), loadLineLayer(), loadPolygonLayer())を追加する。
5. ブラウザをリロードして動作確認する。
6. レイヤの追加順を入れ替えて、Feature(各図形)の上下関係が変わることを確認する。

実習3 - map.js

ソースコード map.js(抜粋)

```
var map = new OpenLayers.Map(
   "map", // 地図を表示するdivのID
   { // 地図全般のオプション
     allOverlays: true, // ベースマップ無しでMap定義
     projection: projection,
     displayProjection: displayProjection,
     units: "m",
     maxExtent: maxExtent,
     numZoomLevels: 5,
     controls: []
   }
 );

 loadPointLayer(map);
 loadLineLayer(map);
 loadPolygonLayer(map);

 if (!map.getCenter()) {
   map.setCenter(new OpenLayers.LonLat(141.350864, 43.068640).transform(displayProjection, projection), 2);
 }

 map.addControl(new OpenLayers.Control.LayerSwitcher());
```

図形種別毎にレイヤを追加する関数
次ページを参考に追加する

実習3 - map.js

ソースコード map.js 追加するコード

```
function loadPointLayer(map)
{
  var style = new OpenLayers.StyleMap({
    'default': new OpenLayers.Style ({
      graphicName:"circle",
      strokeColor: "#ff0000",
      fillColor: "#ff00ff",
      strokeOpacity: 1.0,
      fillOpacity: 0.5,
      pointRadius: 15 // pixel
    })
  });

  var layer = new OpenLayers.Layer.Vector(
    'Layer of Point',
    { styleMap: style }
  );
  map.addLayer(layer);

  var geojson_format = new OpenLayers.Format.GeoJSON({
    externalProjection: new OpenLayers.Projection('EPSG:4326'),
    internalProjection: new OpenLayers.Projection('EPSG:900913')
  });

  loaded_data = geojson_format.read(input_geojson_point);
  layer.addFeatures(loaded_data);
  return;
}
```

図形の描画
スタイル設定

ベクトルレイヤを追加
この時点では図形無し

GeoJSONのI/Oクラスを生成
投影変換オプションを追加

readメソッドでGeoJSONデータをロードする。
引数はJavaScriptの変数。
layer.addFeatureメソッドでレイヤにFeatureを追加する。
(layer.featuresで確認できる)

実習3 - map.js

ソースコード map.js 追加するコード

```
function loadLineLayer(map)
{
  var style = new OpenLayers.StyleMap({
    'default': new OpenLayers.Style ({
      strokeColor: "${color}",
      strokeOpacity: 1.0,
      strokeWidth: 3
    })
  });

  var layer = new OpenLayers.Layer.Vector(
    'Layer of Line',
    { styleMap: style }
  );
  map.addLayer(layer);

  var geojson_format = new OpenLayers.Format.GeoJSON({
    externalProjection: new OpenLayers.Projection('EPSG:4326'),
    internalProjection: new OpenLayers.Projection('EPSG:900913')
  });

  loaded_data = geojson_format.read(input_geojson_line);
  layer.addFeatures(loaded_data);
  return;
}
```

"${変数名}"とすることで、
GeoJSONの属性値(properties)
を利用できる

実習3 - map.js

ソースコード map.js 追加するコード

```
function loadPolygonLayer(map)
{
  var style = new OpenLayers.StyleMap({
    'default': new OpenLayers.Style ({
      strokeColor: "#909090",
      fillColor: "#f0f0f0",
      strokeOpacity: 0.5,
      fillOpacity: 0.9,
    })
  });

  var layer = new OpenLayers.Layer.Vector(
    'Layer of Polygon',
    { styleMap: style }
  );
  map.addLayer(layer);

  var geojson_format = new OpenLayers.Format.GeoJSON({
    externalProjection: new OpenLayers.Projection('EPSG:4326'),
    internalProjection: new OpenLayers.Projection('EPSG:900913')
  });

  loaded_data = geojson_format.read(input_geojson_polygon);
  layer.addFeatures(loaded_data);
  return;
}
```

実習3 - features.js(GeoJSON)

<script src="features.js"></script>で追加する(抜粋)

```
var input_geojson_line =
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "LineString",
        "coordinates": [
          [ 141.351637, 43.062128 ],
          [ 141.353241, 43.055616 ],
          [ 141.361738, 43.056832 ],
          [ 141.363164, 43.056658 ],
          [ 141.374632, 43.049406 ]
        ]
      },
      "properties": {
        "no": 36,
        "color": "#ff0000"
      }
    },
    {
      "type": "Feature",
      "geometry": {
        "type": "LineString",
        "coordinates": [
          [ 141.351518, 43.062085 ],
          [ 141.373147, 43.065037 ],
          [ 141.379742, 43.065992 ],
          [ 141.380931, 43.064255 ],
          [ 141.396558, 43.056224 ]
        ]
      },
      "properties": {
        "no": 12,
        "color": "#00ff00"
      }
    },
    {
      "type": "Feature",
      "geometry": {
        "type": "LineString",
        "coordinates": [
          [ 141.358292, 43.056440 ],
          [ 141.356746, 43.062692 ],
          [ 141.353716, 43.073891 ],
          [ 141.352943, 43.076496 ],
          [ 141.352052, 43.078926 ],
          [ 141.352052, 43.078926 ]
        ]
      },
      "properties": {
        "no": 5 ,
        "color": "#0000ff"
      }
    }
  ]
}
;
```

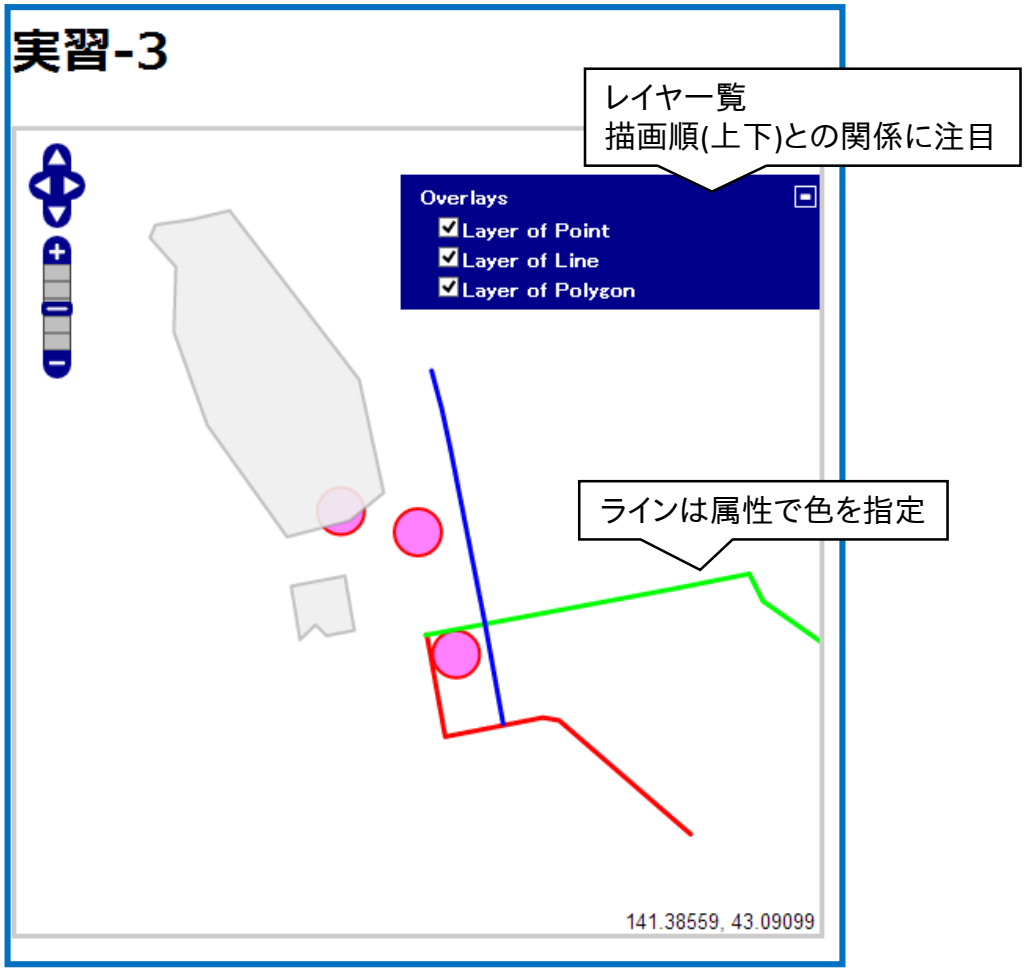
geometry:図形情報

properties:属性情報

属性名は自由に定義できる

個々の図形(features)は、JavaScriptのArrayの要素

実習3 - 実行結果



実習3 - ポイント

- ベクトルなのでズームによる画質の悪化が防げ、スケール毎のデータ作成が省ける。
- クライアントPCの性能、環境とデータ量によっては動作が重い。
- オンラインではWFS(Web Feature Service)、Ajax、OpenLayers.Layer.Vectorのprotocolオプションで随時読み込みを行う方が良い。
- OpenLayers.Styleのプロパティに ${*プロパティ名*} を使用することで、図形毎の描画設定を変えられる。
- 既存DBやGISとデータ連携したい→OpenLayersで読み込めるベクトルデータ(GeoJSON、KML、GMLなど)は、QGISやgdalツール(ogr2ogr)で変換、作成できる。
- QGISは編集も出来るので便利!

GISソフトらしくする(コントロール)

コントロールとは?

- OpenLayersは地図データを表示するだけではなく、GISソフトとして有用な部品・機能も提供する。それらはOpenLayers.Controlクラスとして利用できる。
- デフォルトで有効なコントロール
  - Navigation・・・マウスによって地図操作ができるようにする。
  - Zoom・・・ズームアップ/ダウン(ボタン)を表示する。
  - Attribution・・・レイヤのattributionプロパティを表示する(出典、著作権表示)。
  - ArgParser・・・URLから座標、スケールを解析する。Permalinkコントロールと対になる。
- 主なコントロール
  - PanZoomBar・・・スクロール(スライダー)とズームアップ/ダウン(ボタン)を表示する。
  - LayerSwitcher・・・表示ON/OFFチェックボックス付きのレイヤー覧を表示する。
  - MousePosition・・・マウスカーソル位置を常に表示する。
  - Scale、ScaleLine・・・縮尺記号を表示する(スケール数値(1:xxx)、距離の目安)。
  - PermaLink・・・固定リンクURLを表示する。(地図上に"Permalink"と表示)
  - SelectFeature・・・Featureを選択したときにイベントを実行できるようにする。

実習4 - 主なコントロールを使ってみる

- この実習では、よく使われるコントロールを地図に追加する。
- コントロールの簡単なカスタマイズも行う。

1. 実習3のフォルダ(ex3/)をコピーしてex4/とする。
2. index.htmlをテキストエディタとウェブブラウザで開く。
   ウェブブラウザはデバッグコンソールも起動する。
3. map.jsの map.addControlsの行を全て削除する
4. map.jsを次ページのように書き換える。
5. ブラウザをリロードして動作確認する。

実習4 - map.js

ソースコード map.js の変更、追加箇所

```
var controls = [
    new OpenLayers.Control.Navigation(),
   new OpenLayers.Control.KeyboardDefaults(),
   new OpenLayers.Control.ArgParser(),
   new OpenLayers.Control.Permalink(),
   new OpenLayers.Control.PanZoomBar(),
   new OpenLayers.Control.LayerSwitcher(),
   new OpenLayers.Control.MousePosition(),
   new OpenLayers.Control.Attribution(),
   new OpenLayers.Control.Scale(),
   new OpenLayers.Control.ScaleLine()
 ];

 var map = new OpenLayers.Map(
   "map", // 地図を表示するdivのID
   { // 地図全般のオプション
     allOverlays: true,
     projection: projection,
     displayProjection: displayProjection,
     units: "m", // 座標単位はメートル
     maxExtent: maxExtent,
     numZoomLevels: 5,
     controls: controls
   }
 );
```

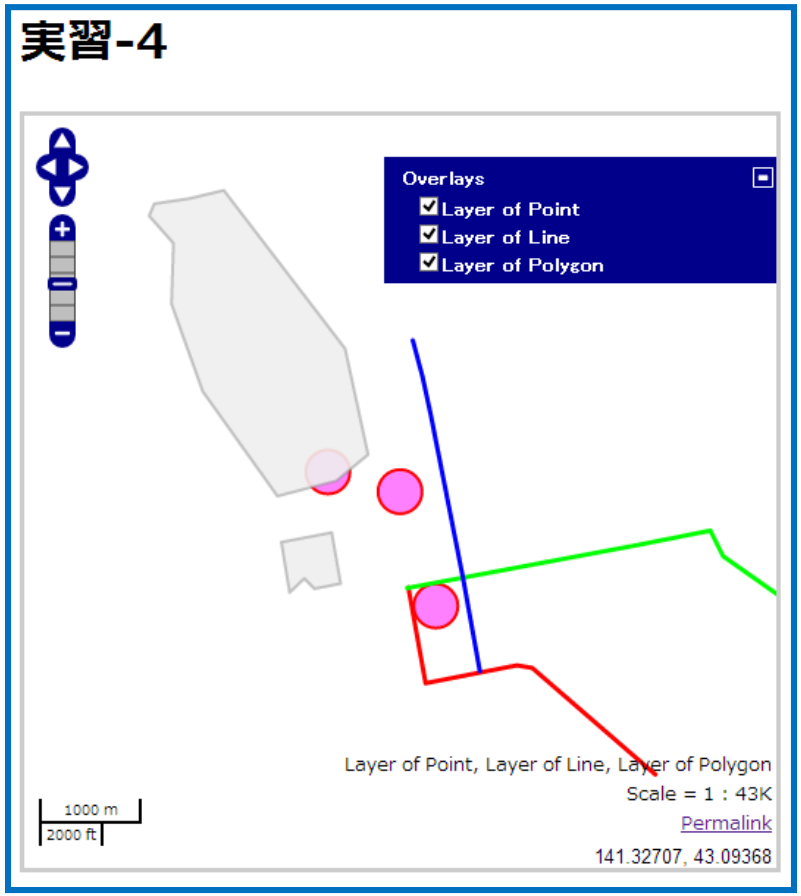
```
 var layer = new OpenLayers.Layer.Vector(
    'Layer of Line',
    { styleMap: style }
  );
  map.addLayer(layer);

↓

 var layer = new OpenLayers.Layer.Vector(
    'Layer of Line',
    { styleMap: style, attribution: 'Layer of Line' }
  );
  map.addLayer(layer);
```

他のレイヤも同様にattributionプロパティを追加する

実習4 - 実行結果


実習-4
Overlays
Layer of Point
Layer of Line
Layer of Polygon
Layer of Point, Layer of Line, Layer of Polygon
Scale = 1 : 43K
Permalink
141.32707, 43.09368
1000 m
2000 ft

実習4 - コントロールのカスタマイズ

1. 最初にインストールしたtheme/default/style.cssをエディタで参照する。
2. ブラウザのDOM Inspectorを使ってみる。
3. index.html,map.js,map.cssを次ページのように編集する。
4. ブラウザをリロードして動作確認する。

実習4 - コントロールのカスタマイズ

index.html

```
<body>
  <h1 id="title">実習4</h1>
  <div id="map" class="mapdiv"></div>
  <div id="ext_permalink" ></div>
</body>
```

map.js

```
new OpenLayers.Control.ArgParser(),
new OpenLayers.Control.Permalink({
    div: OpenLayers.Util.getElement("ext_permalink")
}),
new OpenLayers.Control.PanZoomBar(),
```

map.css

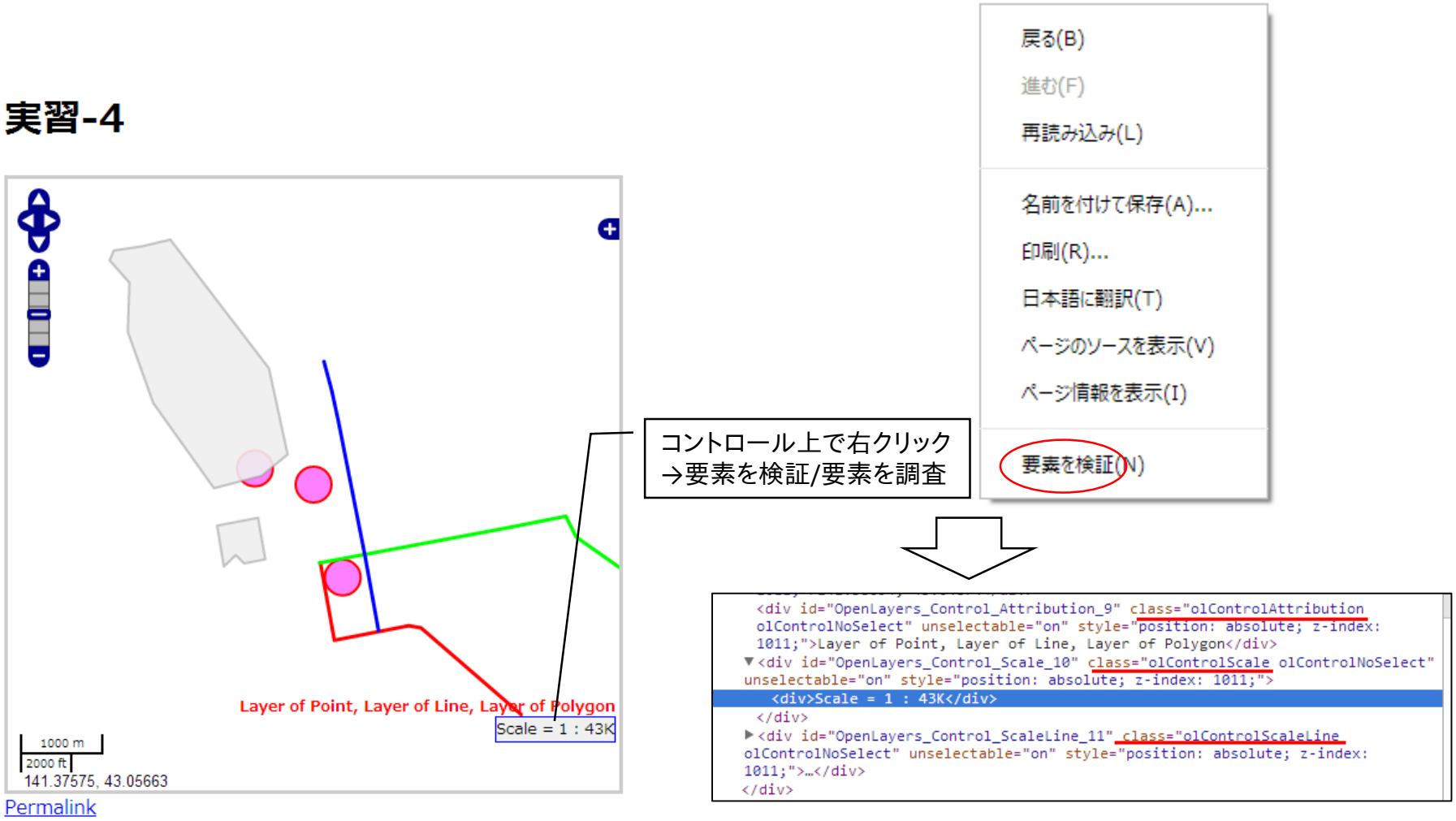
```
.mapdiv {
  width: 512px;
  height: 512px;
  border: 3px solid #ccc;
}

.olControlScale {
  border: 1px solid blue;
  background-color: #f0f0f0;
}

.olControlMousePosition {
  top: 1em;
}

.olControlAttribution {
  font-weight: bold;
  color: #ff0000;
}
```

実習4 - コントロールのカスタマイズ - 実行結果



実習4 - ポイント

- バーやボタンなど”見えるインターフェイス”だけではなく、マウス・キーボード操作などのイベント処理も、コントロールによって提供されている。
- JavaScript(プロパティ変更、オーバーライド)やCSSの変更によって、コントロールをカスタマイズできる。
- CSSカスタマイズにはブラウザの開発ツールを活用する。

実習5 - クリックされた図形の属性を表示する

- この実習では、クリックされた図形の属性を取得、表示する。

1. 実習4のフォルダ(ex4/)をコピーしてex5/とする。
2. index.htmlをテキストエディタとウェブブラウザで開く。
   ウェブブラウザはデバッグコンソールも起動する。
3. map.jsを次ページのように書き換える。
4. ブラウザをリロードして動作確認する。
5. 地図をスクロールさせて、枠ギリギリの図形をクリックしてみる。
6. feature.jsのpropertiesを書き換えてみる。(値の変更、属性種別の追加)

実習5 - map.js

ソースコード map.js の追加ソース

```
function createControlSelectFeature(map, layerNames)
{
  var targetLayers = [];
  for (var i = 0; i < layerNames.length; i++) {
    var layers = map.getLayersByName(layerNames[i]);
    targetLayers.push(layers[0]);
  }

  var control = new OpenLayers.Control.SelectFeature(
    targetLayers,
    {
      onSelect: onFeatureSelect,
      onUnselect: onFeatureUnselect
    });
  map.addControl(control);
  control.activate();
}

function onFeatureSelect(feature) {
  var selectedFeature = feature;
  var selectControl = this;

  var onPopupClose = function (evt) {
    selectControl.unselect(selectedFeature);
  };
  var html = "<div>";
  for (var key in feature.attributes) {
    html += key + ":" + feature.attributes[key] + "<br>";
  }
  html += "</div>";

  var popup = new OpenLayers.Popup.FramedCloud("popup",
    feature.geometry.getBounds().getCenterLonLat(),
    null, html, null, true, onPopupClose);

  feature.popup = popup;
  this.map.addPopup(popup);
  return;
}

function onFeatureUnselect(feature) {
  this.map.removePopup(feature.popup);
  feature.popup.destroy();
  feature.popup = null;
  return null;
}
```

選択対象のレイヤ配列

SelectFeatureコントロールは
activate()しないと有効に出来ない

実習5 - map.js

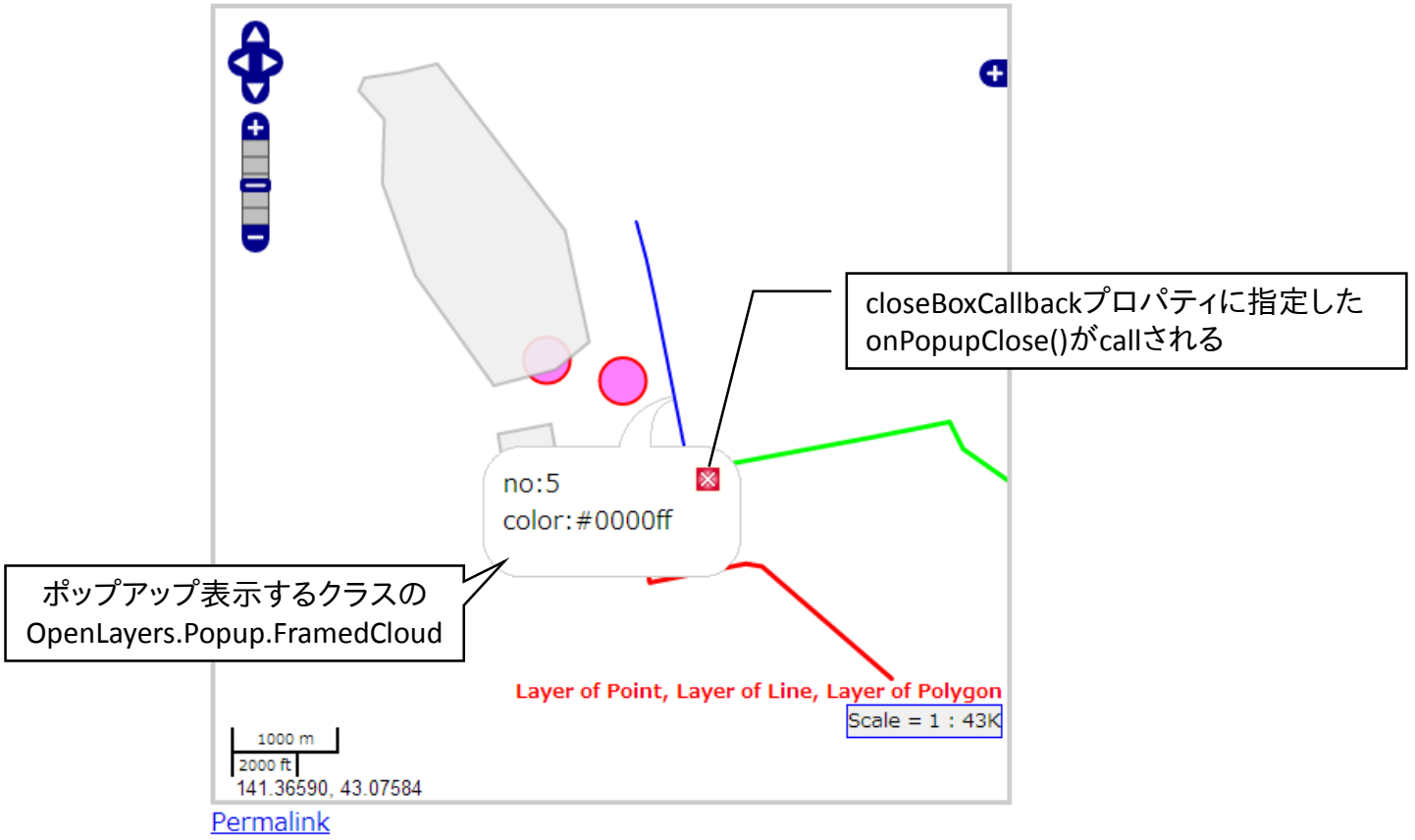
ソースコード map.js の変更(追加)箇所

```
  loadPointLayer(map);
  loadLineLayer(map);
  loadPolygonLayer(map);

  createControlSelectFeature(map, ['Layer of Point', 'Layer of Line', 'Layer of Polygon']);

  if (!map.getCenter()) {
    map.setCenter(new OpenLayers.LonLat(141.350864, 43.068640).transform(displayProjection, projection), 2);
  }
```

実習5 - 実行結果


closeBoxCallbackプロパティに指定した
onPopupClose()がcallされる
no:5
color:#0000ff
ポップアップ表示するクラスの
OpenLayers.Popup.FramedCloud
Layer of Point, Layer of Line, Layer of Polygon
Scale = 1 : 43K
1000 m
2000 ft
141.36590, 43.07584
Permalink

実習5 - ポイント

- 図形を選択すると属性がポップアップ表示される処理のフロー
  1. マウスでクリックする(Featureが選択状態になる)
  2. 選択イベントが発生し、onSelectプロパティに指定したコールバック関数が実行される
  3. 2.のコールバック関数(この関数をコーディングする)はポップアップ表示の処理を行う。
  4. 他のFeatureが選択されたり、何も無い場所をクリックすると選択が解除され、onUnselectプロパティに指定したコールバック関数が実行される。
  5. 4.のコールバック関数(この関数をコーディングする)はポップアップの非表示、削除処理を行う。
     ＊選択や解除の仕様はselectFeatureクラスのプロパティで変更できる
     (キーワード:hover,clickout, toggle)
- selectFeatureコントロールを定義するだけでは選択イベントは発生しない。
  activateメソッドを実行する。(一時的に無効にするにはdeactivate)

＋α (開発のコツなど)

オンラインマップの紹介

- 背景地図(ベースマップ)にオススメのサービス・クラスの紹介
  - OpenLayers.Layer.Google・・・Google Maps API経由のGoogleマップ

    現在は次のURLで呼び出すのが良さそう。

    <script type="text/javascript"
    src="http://maps.google.com/maps/api/js?v=3&sensor=false&language=ja&region=jp"></script>
    - Google Maps API Ver.3 (ver.2はサポート終了、廃止予定)
    - APIキー不要(Ver.2の古い資料ではAPIキー必要となってる)
    - 日本国内で使うにはlanguage=ja&region=jpと付けた方が良い(東海騒動)

  - OpenLayers.Layer.OSM・・・OpenStreetMaps(WMS)
    自由な地図
    - 追加スクリプトは必要なし

  - webtis.Layer.BaseMap・・・電子国土Web.next

    <script type="text/javascript" src="http://portal.cyberjapan.jp/sys/v4/webtis/webtis_v4.js" ></script>
    - 詳細は http://portal.cyberjapan.jp/portalsite/version/v4/

OpenLayersとJavaScriptライブラリの組み合わせ

WebGISはOpenLayersのみでも構築出来るが、他のJavaScriptと組み合わせることにより
UIの利便性やクロスブラウザ対応など、開発生産性を高められる。

- jQuery系列
  - jQuery(http://jquery.com/)・・・DOM走査、Ajaxなど全般的に
  - jQuery UI(http://jqueryui.com/) ・・・タブ、ポップアップなど
  - jQuery UI Layout Plug-in(http://layout.jquery-dev.net/)・・・フレームレイアウト

- GeoExt(http://www.geoext.org/)
  - ExtJS(Sencha)のOpenLayers拡張(ExtJSも必要)
  - 各種APIが揃っており、親和性が高い
  - オープンソースだが、ExtJSはGPL 3.0/商用ライセンスなので注意
  - 現行のExtJS 4.xには未対応

参考リンクなど

- 参考リンク
  - OpenLayers.org http://www.openlayers.org/
  - OpenLayers User discussion http://lists.osgeo.org/mailman/listinfo/openlayers-users
  - OSGeo.jp discussion http://lists.osgeo.org/mailman/listinfo/osgeojapan-discuss
  - OpenなGISのこと http://blog.godo-tys.jp/

- 参考書
  - FOSS4G HANDBOOK http://www.amazon.co.jp/dp/4759101314

以上、お疲れ様でした

誤り･クレームなどは↓まで
北海道地図株式会社 原田英夫 (@hcc_hh)
h-harada@hcc.co.jp / harada.hideo@gmail.com

2013/07/06 FOSS4G 2013 Hokkaido ハンズオンセミナー