



OpenLayers 1

Librairies JavaScript de Webmapping

Deux librairies principales pour du webmapping 2D open source:

	
Simple de base mais peut-être étendue par des plugins	Beaucoup de fonctionnalités de base + plugins
Grande communauté, beaucoup d'exemples sur le web	Moins grande communauté mais évolue rapidement
Projection suisse peu supportée	Très présente sur le marché Suisse

D'autres librairies existent:

- Mapbox et MapLibre: plutôt pour des vector tiles
- Cesium: globe 3D
- Here Maps API, Google Maps API: propriétaire
- Carto: styliser des cartes avec CSS

OpenLayers: exemples d'utilisation

- GeoAdmin : map.geo.admin.ch
- SuisseMobile : map.wanderland.ch
- Luxembourg : map.geoportail.lu
- EPFL : plan.epfl.ch
- Transports publics en temps réel : tracker.geops.ch
- NE Mobilité 2030 : nemobilite2030.ch
- Région de Nyon : map.cartolacote.ch

Première carte!

```
...  
<div id="map"></div>  
<script src="https://cdn.jsdelivr.net/npm/ol@v7.1.0/dist/ol.js"></script>  
<script type="text/javascript">  
  const map = new ol.Map({  
    target: "map", // l'id de l'élément HTML où on veut mettre la carte  
  
    layers: [  
      new ol.layer.Tile({  
        source: new ol.source.OSM(),
```

```

    }},
  ],

  view: new ol.View({
    center: ol.proj.fromLonLat([6.6, 46.6]),
    zoom: 10,
  }),
});
</script>

```

Voir `ol-1_carte_basique.html`

Comment ça marche?

1. On charge OpenLayers dans une balise `<script>`. Rien de nouveau par rapport à jQuery. On charge également la CSS d'OpenLayers.
2. On crée, dans l'HTML un `<div>` qui contiendra la carte. On lui donne un `id` qui servira de référence à OpenLayers. Exemple: `<div id="map">`. On peut également définir la hauteur et largeur de la carte en CSS:

```

#map {
  height: 400px;
  width: 100%;
}

```

3. Avec JavaScript on crée la carte en utilisant la variable globale `ol`.

API & exemples

<https://openlayers.org/>

new ?

JavaScript n'a pas de classes, c'est un langage à objets. On peut cependant utiliser le mot-clé `new` sur des fonctions. Les fonctions se comportent comme des constructeurs et créent des instances de la fonction:

```

function Car() {}

const myCar = new Car();
console.log(myCar instanceof Car); // true

```

import ?

Les exemples d'OpenLayers utilisent [les modules JavaScript](#). Les modules permettent de ne charger qu'une partie d'OpenLayers mais cela sort du périmètre de ce cours.

Bonne nouvelle, OpenLayers s'utilise aussi de façon traditionnelle et il est possible de "traduire" la syntaxe `import` en syntaxe de type `ol.module.classe`.

Prenons cet [exemple](#):

```
// ES6 imports
import 'ol/ol.css';
import Map from 'ol/Map';
import OSM from 'ol/source/OSM';
import TileLayer from 'ol/layer/Tile';
import View from 'ol/View';

const map = new Map({
  layers: [
    new TileLayer({
      source: new OSM(),
    }) ],
  target: 'map',
  view: new View({
    center: [0, 0],
    zoom: 2,
  }),
});

// Variable globale ol
const map = new ol.Map({
  layers: [
    new ol.layer.Tile({
      source: new ol.source.OSM(),
    }) ],
  target: 'map',
  view: new ol.View({
    center: [0, 0],
    zoom: 2,
  }),
});
```

Pour utiliser OpenLayers de manière traditionnelle (avec la variable globale `ol`), on ignore les lignes d'import et on appelle les constructeurs (`new`) et les méthodes à l'aide d'une notation complète basée sur le chemin des `import`.

OpenLayers: principes

ol.Map

C'est l'élément de base. La carte contrôle où elle sera placée dans l'HTML grâce à sa propriété `target`, elle contient une `view` et des `layers` [entre autres](#).

ol.View

Obligatoire: contient le niveau de `zoom` et le `center` [entre autres](#)

ol.layer

C'est un module contenant plusieurs types de couches. Dans notre premier exemple, nous n'avons vu qu'un seul layer: `ol.layer.Tile`. Nous discuterons des layers et de leurs sources plus tard.

OpenLayers: toute la documentation dans l'API!

En règle générale, dans des bibliothèques complexes, on attribue pas des valeurs directement aux propriétés (exemple `ol.Map.view.zoom = 10`). On passe par des *getters* et *setters*. Exemples de méthodes de la classe `ol.View`:

Get:

- Centre : `getCenter()`
- Zoom : `getZoom()`
- Orientation : `getRotation()`
- Projection : `getProjection().getCode()`

Set :

- Centre : `setCenter(center)`
 - avec `center` : tableau de coordonnées (ex. `[2600000, 1200000]`)
- Zoom : `setZoom(zoom)`
 - avec `zoom` : entier (ex. `15`)
- Orientation : `setRotation(rotation)`
 - avec `rotation` : angle en radians (`3.14`)

Voir `ol-2_methodes_vue.html`

Contrôles

Les contrôles sont des éléments permettant de manipuler la carte ou d'afficher une information.

Par défaut, `ol.Map` en charge 3:

- Zoom `ol.control.Zoom`
- Orientation `ol.control.Rotate` (apparaît dès que la carte est tournée)
- Attribution `ol.control.Attribution`

De nombreux autres contrôles existent:

- Barre d'échelle : `ol.control.ScaleLine`
- Carte d'aperçu : `ol.control.OverviewMap`
- Position curseur : `ol.control.MousePosition`
- Plein écran : `ol.control.FullScreen`
- Zoom sur étendue max : `ol.control.ZoomToExtent`
- Curseur de zoom : `ol.control.ZoomSlider`

Il est même possible de créer ses propres contrôles!

Voir `ol-3_controles.html`

Couches

Le nombre possible de types de couches sont nombreux mais peuvent être divisés en 2 catégories:

- Raster (par ex: `ol.layer.Tile`, `ol.layer.Image`)
- Vectoriel (par ex: `ol.layer.Vector`, `ol.layer.VectorTile`)

Comme dans QGIS, une couche représente une donnée, la source de la donnée est par conséquent une propriété d'un `ol.layer` et on peut à nouveau les séparer en 2 catégories:

- Raster (par ex: `ol.source.Tile`, `ol.source.Image`)
- Vectoriel (par ex: `ol.source.Vector`, `ol.source.VectorTile`)

Couches tuilées

Quelques exemples de couches tuilées `ol.layer.Tile` et leurs sources:

- OpenStreetMap : `ol.source.OSM`
 - Ne pas utiliser cette source en production! Elle est disponible à des fins de démo.
- Bing : `ol.source.BingMaps`
 - `imagerySet` : *Road, Aerial, AerialWithLabels*
 - `key` : clé à obtenir sur *bingmapsportal.com*
- Stamen : `ol.source.Stamen`
 - `layer` : *terrain, toner, toner-lite, watercolor, terrain-labels*
- etc.

Voir *ol-4_couches_tuiles.html*

WMS (Web Map Service)

Il y a deux façons de lire du WMS avec OpenLayers:

`ol.layer.Tile` avec `ol.source.TileWMS`

- OpenLayers découpe l'étendue de la vue en une mosaïque d'images carrées qu'il va demander au serveur
- Performant sur des WMS raster, les images sont mises en cache
- Plus lent lorsque le serveur doit calculer un rendu complexe

`ol.layer.Image` avec `ol.source.ImageWMS`

- OpenLayers demande au serveur WMS une image de la taille de la carte
- Pas de cache, à chaque zoom, une nouvelle image est demandée
- Intéressant quand le rendu d'une couche est complexe

Une source WMS demandera toujours:

- une `url` du service WMS
- les `params` standards WMS, c'est-à-dire les paramètres GetMap tels que `LAYERS`, `FORMAT`, etc.
- si le WMS est soumis à des droits d'utilisation, il faut indiquer les `attributions`

Voir *ol-5_couches_wms.html*

Exercices

Exercice *ol-1_fond_plan.html*. Inspirez-vous des exemples!