OpenLayers

Styles

L'ensemble des Styles OpenLayers constitue le moyen de contrôler le style des éléments attachés aux couches vecteur dans OpenLayers, tels que des points, lignes, et polygones. Les possibilités offertes correspondent aux fonctionnalités fournies par des standards tel que le SLD, permettant l'utilisation de styles avancés avec des propriétés et des règles.

Les 'Style Classes'

Lorsqu'un élément est ajouté sur la carte, son information de style peut provenir de l'une des trois sources:

- Une série de paramètres de style placée directement sur l'élément. Ceci est inhabituel, mais peut arriver lors de l'analyse de données distantes qui contiennent une information de style au niveau de chaque élément, comme certains contenus KML.
- Une série de paramètres de style associée à une couche comme layer.style.
- Un objet Style ou StyleMap attaché à une couche comme layer.styleMap.

Une série de paramètres de style est un simple objet JavaScript, avec des paires clé/valeur décrivant les valeurs à utiliser.

```
{
  'strokeWidth': 5,
  'strokeColor': '#ff0000'
}
```

Un objet StyleMap est un élément plus descriptif, qui permet l'utilisation de styles avancés. On utilise à la fois les objets OpenLayers.Style et OpenLayers.StyleMap. Un objet StyleMap supporte différentes 'intentions de rendu': façons dont un élément doit être dessiné. OpenLayers utilise trois 'intentions de rendu' différentes en interne:

- 'default': Utilisé dans la plupart des cas
- 'select': Utilisé quand un élément est sélectionné

- ‘temporary’: Utilisé lors de l’ébauche d’un élément.

Lorsqu’un élément est dessiné, il est possible de passer une de ces intentions de rendu à la fonction ‘drawFeature’ de la classe Layer, ou de mettre la propriété ‘renderIntent’ de l’élément à l’une de ces trois intentions.

Chaque intention de rendu dans le StyleMap a un objet OpenLayers.Style qui lui est associé.

Les objets OpenLayers.Style sont des descriptions de la manière dont les éléments doivent être rendus. Quand un élément est ajouté à une couche, celle-ci combine la propriété de style avec l’élément pour créer un ‘symboliseur’ – décrit ci-dessus comme un ensemble de propriétés de style qui seront utilisées lors du rendu de la couche. (En interne, ceci est fait via la fonction ‘createSymbolizer’.)

Syntaxe de Remplacement par Attribut

La manière la plus courante d’accéder aux attributs d’un élément lors de la création d’un style correspond à la syntaxe de remplacement par attribut. En utilisant des valeurs de style telles que `${varname}`, OpenLayers peut remplacer ces chaînes de caractères par les attributs des éléments. Par exemple, si vous avez un fichier GeoJSON où chaque élément a un attribut `thumbnail` décrivant une image à utiliser, vous pouvez employer quelque chose comme:

```
var s = new OpenLayers.Style({
  'pointRadius': 10,
  'externalGraphic': '${thumbnail}'
});
```

De cette manière, le style peut contenir une information de rendu dépendante de chaque élément.

StyleMap

Les objets du type simple OpenLayers Style sont instanciés en passant un ensemble de paramètres au constructeur du style. Ce Style peut ensuite être passé à un constructeur StyleMap:

```
new OpenLayers.StyleMap(s);
```

Par commodité, vous pouvez aussi créer un StyleMap en passant directement l'ensemble de paramètres. (Le StyleMap va alors créer l'objet Style pour vous.)

```
new OpenLayers.StyleMap({'pointRadius': 10,
                         'externalGraphic': '${thumbnail}'});
```

Dans presque tous les cas simples, cela sera suffisant. Par défaut, passer un objet Style fera que toutes les intentions de rendu du StyleMap auront le même style. Si vous souhaitez avoir différents objets Style pour différentes intentions de rendu, passez plutôt une série d'objets Style ou une série d'ensembles de paramètres:

```
var defaultStyle = new OpenLayers.Style({
  'pointRadius': 10,
  'externalGraphic': '${thumbnail}'
});

var selectStyle = new OpenLayers.Style({
  'pointRadius': 20
});

new OpenLayers.StyleMap({'default': defaultStyle,
                         'select': selectStyle});
```

L'intention de rendu 'default' a un rôle particulier: si la propriété extentDefault du StyleMap est mise à 'true' (valeur par défaut), les 'symboliseurs' calculés pour les autres intentions de rendu vont étendre le 'symboliseur' calculé pour l'intention 'default'. Donc si nous voulons que les éléments sélectionnés aient juste une taille ou une couleur différente, nous devons uniquement changer une propriété (dans cet exemple: pointRadius).

Utiliser les objets Style

Une fois l'objet Style créé, il est nécessaire de le passer à une couche afin qu'il puisse être utilisé. L'objet StyleMap doit être passé via l'option 'styleMap' de la couche:

```
var styleMap = new OpenLayers.StyleMap({pointRadius: 10,
                         externalGraphic: '${thumbnail}'});
var l = new OpenLayers.Layer.Vector("Vector Layer",
                                    {styleMap: styleMap});
```

Les éléments ajoutés à la couche seront rendus suivant le StyleMap défini.

Styles Basés sur des Règles

En plus des styles basés sur un simple attribut, OpenLayers supporte également des styles basés sur des règles – où une propriété sur un élément peut déterminer les autres styles en usage. Par exemple, si vous avez un attribut, ‘size’, qui est ‘large’ ou ‘small’, et qui détermine la taille désirée d’une icône, vous pouvez utiliser cette propriété pour contrôler le ‘pointRadius’.

OpenLayers fournit deux manières différentes de réaliser cela. Beaucoup de cas simples peuvent être solutionnés avec la fonction commode addUniqueValueRules, alors que les cas plus complexes nécessitent de créer vos propres règles.

addUniqueValueRules

Afin d’utiliser la fonction addUniqueValueRules, vous créez d’abord un StyleMap avec les propriétés ‘partagées’ du style. Comme dans le cas ci-dessus, nous imaginons que nous chargeons des éléments avec des URLs dans l’attribut ‘thumbnail’:

```
var styleMap = new OpenLayers.StyleMap({externalGraphic: '${thumbnail}'});
```

Nous créons alors une correspondance entre la valeur de l’attribut et la valeur du symboliseur, ensuite nous ajoutons les règles au symboliseur par défaut qui vérifie l’attribut “size” et applique le symboliseur défini dans cette variable:

```
var lookup = {
  "small": {pointRadius: 10},
  "large": {pointRadius: 30}
}

styleMap.addUniqueValueRules("default", "size", lookup);
```

Ceci ajoute des règles aux Styles dans l’intention de rendu ‘default’, actant que le Style doit changer le ‘pointRadius’ sur la base de l’attribut ‘size’ de l’élément.

Les symboliseurs à l’intérieur des règles ne doivent pas être des symboliseurs complets, car ils étendent le symboliseur par défaut passé au constructeur de OpenLayers.Style ou

OpenLayers.StyleMap.

L'exemple Unique Values example illustre l'utilisation de la fonction addUniqueValueRules.

Règles personnalisées

OpenLayers supporte de nombreux types de Règles et de Filtres. La fonction addUniqueValueRules créé des règles de Comparaison, avec l'opérateur EQUAL_TO. Nous pouvons aussi créer des règles qui nous permettent d'appliquer des styles sur la base du fait qu'une valeur est supérieure ou inférieure à une autre valeur, ou qu'elle correspond à une certaine chaîne de caractères, et plus encore.

Nous illustrons, ici, comment créer des filtres en utilisant les opérateurs LESS_THAN et GREATER_THAN_OR_EQUAL_TO:

```
var style = new OpenLayers.Style();

var ruleLow = new OpenLayers.Rule({
  filter: new OpenLayers.Filter.Comparison({
      type: OpenLayers.Filter.Comparison.LESS_THAN,
      property: "amount",
      value: 20,
  }),
  symbolizer: {pointRadius: 10, fillColor: "green",
               fillOpacity: 0.5, strokeColor: "black"}
});

var ruleHigh = new OpenLayers.Rule({
  filter: new OpenLayers.Filter.Comparison({
      type: OpenLayers.Filter.Comparison.GREATER_THAN_OR_EQUAL_TO,
      property: "amount",
      value: 20,
  }),
  symbolizer: {pointRadius: 20, fillColor: "red",
               fillOpacity: 0.7, strokeColor: "black"}
});

style.addRules([ruleLow, ruleHigh]);
```

Chacune de ces règles utilise un filtre de Comparaison. Il existe différents types de filtres:

- Comparison Filters: Ces filtres prennent un opérateur – un des types de filtre de comparaison supportés – et une ou deux valeurs. On évalue si l'élément satisfait la comparaison.
- FeatureId Filters: Prennent une liste de 'Feature IDs'. On évalue si l'ID de l'élément est dans la liste.
- Logical Filters: Ces filtres combinent les autres types de filtres, ce qui permet de construire des règles plus complexes en utilisant des opérateurs booléens (AND, OR, NOT). Une règle 'Logique' peut avoir des règles enfants (excepté avec NOT).

Chaque règle peut aussi avoir une propriété minScaleDenominator et maxScaleDenominator. Ceci nous permet de spécifier des plages d'échelle pour lesquelles la règle doit être appliquée. Nous pourrions par exemple vouloir montrer des petits points aux petites échelles, mais des imagettes aux grandes échelles. Le résultat de telles règles est illustré dans l' Exemple SLD: zoomer à un certain niveau transformera les lacs en bleu. Les styles et règles de cet exemple ne proviennent pas d'objets de style et de règle créés en javascript, mais d'un document SLD lu grâce à OpenLayers.Format.SLD.

Avec les SLD, les styles sont groupés par couches (NamedLayer), qui contiennent elles-même un ensemble de styles utilisateur (UserStyle). C'est la raison pour laquelle un objet Style possède aussi les propriétés 'layerName' et 'name'. Pour chaque couche, il peut y avoir un style par défaut. Ceci est noté en mettant la propriété 'isDefault' de l'objet Style à 'true'.

Propriétés de Style

Les propriétés que vous pouvez utiliser pour les styles sont les suivantes:

- fillColor

  Valeur par défaut: `#ee9900`. C'est la couleur utilisée pour remplir les Polygones. C'est aussi utilisé dans le centre des marques pour les points: la couleur intérieure des cercles et autres formes. Ce n'est pas utilisé si un 'externalGraphic' est appliqué au point.

- fillOpacity

  Valeur par défaut: `0.4`. C'est l'opacité utilisée pour remplir les Polygones. C'est aussi utilisé dans le centre des marques pour les points: la couleur intérieure des cercles et autres formes. Ce n'est pas utilisé si un 'externalGraphic' est appliqué au point.

- strokeColor

  Valeur par défaut: `#ee9900`. C'est la couleur de la ligne des éléments. Pour les polygones et les points, il s'agit du contour de l'élément. Pour les lignes, c'est la représentation de l'élément.

- strokeOpacity

  Valeur par défaut: `1`. C'est l'opacité de la ligne des éléments. Pour les polygones et les points, il s'agit du contour de l'élément. Pour les lignes, c'est la représentation de l'élément.

- strokeWidth

  Valeur par défaut: `1`. C'est la largeur de la ligne des éléments. Pour les polygones et les points, il s'agit du contour de l'élément. Pour les lignes, c'est la représentation de l'élément.

- strokeLinecap

  Valeur par défaut: `round`. Les options sont `butt`, `round`, `square`. Cette propriété est similaire à la propriété SVG stroke-linecap. Cela détermine à quoi doit ressembler la fin des lignes. Voir le lien SVG pour les exemples d'image.

- strokeDashstyle

  Valeur par défaut: `solid`. Les options sont:

  - `dot`
  - `dash`
  - `dashdot`
  - `longdash`
  - `longdashdot`
  - `solid`

- pointRadius

  Valeur par défaut: `6`.

- pointerEvents:

  Valeur par défaut: `visiblePainted`. Utilisé uniquement par le moteur de rendu SVG. Voir la définition SVG pointer-events pour plus d'info.

- cursor

  Curseur utilisé lorsque le pointeur de la souris est sur l'élément. La valeur par défaut est une chaîne vide, et est héritée des éléments parents.

- externalGraphic

  Une image externe utilisée pour représenter un point.

- graphicWidth, graphicHeight

  Ces propriétés définissent la hauteur et la largeur d'un externalGraphic. C'est une alternative à la propriété pointRadius à utiliser lorsque votre graphique a des tailles différentes suivant les axes X et Y.

- graphicOpacity

  Opacité de l'image externe.

- graphicXOffset, graphicYOffset

  Indique où doit se situer le centre d'un externalGraphic.

- graphicName

  Nom d'un type de symbole à utiliser pour une marque de point.

- display

  Peut être mis à 'none' pour cacher les éléments lors du rendu.