管理地图的堆栈层

OpenLayers地图使我们能够 可视化来自不同种类的层的信息，这为我们带来了管理与之相连的层的方法。

在本食谱中，我们将学习一些有关如何控制层的技术：添加，分组，管理堆栈顺序以及其他层操作。学习这些非常常见的操作非常重要，因为几乎每个Web映射应用程序都需要这些类型的任务。

该应用程序将在左侧显示一个地图，在右侧显示一个控制面板，其中包含可以拖动的层列表，您可以对其进行排序。这就是我们的最终结果：

您可以在中找到此食谱的源代码 `ch01/ch01-map-layers/` 。

> **注意**
>
> 在此食谱中创建小部件（例如可排序列表）时，我们将使用jQuery UI库（https://jqueryui.com (https://jqueryui.com)），它对jQuery具有单一依赖性（https://jquery.com (https://jquery.com)）。这样做将有助于我们将注意力集中在OpenLayers代码上，而不是用于创建高级UI组件的常规JavaScript代码上。

怎么做...

1. 我们从创建开始 一个HTML文件，用于组织应用程序布局并链接到资源：

复制

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Managing map's stack layers | Chapter 1</title>
  <link rel="stylesheet" href="ol.css">
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <div id="js-map" class="map"></div>
  <div class="pane">
    <h1>Layers</h1>
    <p>Drag the layer you wish to view over the satellite imagery into the box.</p>
    <ul id="js-layers" class="layers"></ul>
  </div>
  <script src="ol.js"></script>
  <script src="jquery.js"></script>
  <script src="jquery-ui.js"></script>
  <script src="script.js"></script>
```

2 创建 CSS文件， `style.css` 并在其中添加以下内容：

复制

```
.map {
  position: absolute;
  top: 0;
  bottom: 0;
  left: 0;
  right: 20%;
}

.pane {
  position: absolute;
  top: 0;
  bottom: 0;
  right: 0;
  width: 20%;
  background: ghostwhite;
  border-left: 5px solid lightsteelblue;
  box-sizing: border-box;
  padding: 0 20px;
}
```

3 创建 在 `script.js` JavaScript文件，并添加了下列文件：

复制

```
var map = new ol.Map({
  layers: [
    new ol.layer.Tile({
      source: new ol.source.MapQuest({
        layer: 'sat'
      }),
      opacity: 0.5,
      zIndex: 1
    })
  ],
  view: new ol.View({
    zoom: 4,
    center: [2120000, 0]
  }),
  target: 'js-map'
});

var layerGroup = new ol.layer.Group({
  layers: [
```

怎么运行的...

HTML包含 地图和控制面板的标记。如本食谱前面所述，我们已链接到jQuery UI和jQuery的本地副本。如果您不使用提供的源代码，则需要自己下载这些库以进行后续操作。

The CSS organizes the layout so that the map takes up 80% width of the screen with 20% left over for the control panel. It also provides the styling for the list of layers so that the first item in the list is outlined to represent the layer that is currently in view. We won't go into any more detail about the CSS, as we'd like to spend more of our time taking a closer look at the OpenLayers code instead.

Let's begin by breaking down the code in our custom JavaScript file:

Copy

```
var map = new ol.Map({
  layers: [
    new ol.layer.Tile({
      source: new ol.source.MapQuest({
        layer: 'sat'
      }),
      opacity: 0.5,
      zIndex: 1
    })
  ],
  view: new ol.View({
    zoom: 4,
    center: [2120000, 0]
  }),
  target: 'js-map'
});
```

We've introduced a new layer source here, `ol.source.MapQuest`. OpenLayers provides easy access to this tile service that offers multiple types of `layers`, from which we've chosen type `sat`, which is an abbreviation of satellite. We're going to use this layer as our always-visible backdrop. In order to produce this desired effect, we've passed in some properties to `ol.layer.Tile` to set `opacity` to 50% (`0.5`) and `zIndex` to `1`.

The reason why we set `zIndex` to `1` is to ensure that this layer is not hidden by the layer group that's added on top of this layer. This will be better explained when we continue looking through the next piece of code, as follows:

Copy

```
var layerGroup = new ol.layer.Group({
  layers: [
    new ol.layer.Tile({
      source: new ol.source.MapQuest({
        layer: 'osm'
      }),
      title: 'MapQuest OSM'
    }),
    new ol.layer.Tile({
      source: new ol.source.MapQuest({
        layer: 'hyb'
      }),
      title: 'MapQuest Hybrid',
      visible: false
    }),
    new ol.layer.Tile({
      source: new ol.source.OSM(),
      title: 'OpenStreetMap',
      visible: false
```

We instantiate a new instance of `ol.layer.Group`, which expects a layers collection. One useful benefit of creating a layer group is when you want to apply the same actions against many layers at once, such as setting a property.

We instantiate three new instances of `ol.layer.Tile`, two of which are different layer types offered from `ol.source.MapQuest` (`osm` and `hyb`). The other tile service source is the familiar `ol.source.OSM` layer source (OpenStreetMap) from previous recipes.

We have set the `visible` property on two of the three tile layers to `false`. When the page loads, the `MapQuest osm` layer will be the only visible layer from this layer group.

Optionally, we could have set the `opacity` to 0 for the layers that we didn't want to display. However, there's a performance benefit from setting the visibility to `false`, as OpenLayers doesn't make any unnecessary HTTP requests for the tiles of layers that aren't visible.

The `title` property that we set on each layer isn't actually part of the OpenLayers API. This is a custom property, and we could have named it almost anything. This allows us to create arbitrary properties and values on the `layer` objects, which we can later reference in our application. We will use the title information for some layer-switching logic and to display this text in the UI.

Lastly, a customization has been applied to all the layers inside the `layer` group by setting the `zIndex` property to `0` on the layer group instance. However, why have we done this?

Internally, OpenLayers stores `layers` in an array, and they are rendered in the same order that they are stored in the array (so the first element is the bottom layer). You can think of the map as storing layers in a stack and they are rendered from bottom to top, so the above layers can hide beneath the below layers depending on opacity and extent.

With this in mind, when this layer group is added to the map, it'll naturally render above our first layer containing the satellite imagery. As the layers in the group are all opaque, this will result in hiding the satellite imagery layer. However, by manually manipulating the map layer stack order, we force the layer group to be at the bottom of the stack by setting `zIndex` to `0`, and we force the satellite imagery layer to the top of the stack by setting `zIndex` to `1` so that it'll render above this layer group.

Note

The default `zIndex` property for a layer group is `0` anyway. This means that we could have just set the `zIndex` property of the satellite layer to `1`, and this would leave us with the same result. We've explicitly set this here to help explain what's going on.

由于我们一直希望将卫星图像放在最上面，因此值得一提的是它 `ol.layer.Layer` 提供了一种 `setMap` 方法。tile图层（ `ol.layer.Tile` ）是的子类 `ol.layer.Layer` ，因此，如果通过 `setMap` 方法将卫星图像tile图层添加到地图中，则无需自己手动调整 `zIndex` 属性，因为该属性会自动显示在顶部。无论如何，这是一个很好的机会来展示 `zIndex` 实际行动。

复制

```
map.addLayer(layerGroup);
```

图层组仅添加到地图实例中。您会注意到，此方法可用于添加单个图层或一组图层。

复制

```
var $layersList = $('#js-layers');
layerGroup.getLayers().forEach(function(element, index, array) {
  var $li = $('<li />');
  $li.text(element.get('title'));
  $layersList.append($li);
});
```

现在，我们开始利用jQuery库来执行一些DOM操作。我们将 `js-layers` ID 的元素存储到一个变量中，即 `$layersList` 。在变量前面加上美元符号是一种约定，可以将结果表示为jQuery对象。此选择器将定位先前的HTML：

复制

```
<ul id="js-layers" class="layers"></ul>
```

为了在面板中动态填充图层列表，我们使用了来自图层组实例的方法 `getLayers` 。这会返回 `ol.collection` 给定组的所有层的列表（），然后我们将其链接到 `forEach` 方法（可从中使用另一种方法 `ol.collection` ）。

在内部， `forEach` 方法从Google Closure库调用实用程序方法。该 `forEach` 方法中可用的参数是元素，索引和数组。元素是迭代时的层，索引是迭代时该层在组中的位置，而数组是我们要遍历的层的组。在本例中，我们仅使用element参数。

我们使用jQuery创建一个 `li` 元素并设置文本内容。文本值源自图层的标题值，这是我们赋予组中每个图层的自定义属性，以标识它们。OpenLayers提供了一种方便的 `get` 方法来检索此值。然后，我们使用jQuery将这个 `li` 元素附加到 `ul` 元素上。

复制

```
$layersList.sortable({
  update: function() {
    var topLayer = $layersList.find('li:first-child').text();

    layerGroup.getLayers().forEach(function(element) {
      element.setVisible(element.get('title') === topLayer);
    });
  }
});
```

为了使列表项能够重新排序，我们使用jQuery UI可排序小部件并将其应用于HTML中的层列表。列表中的某个项目一旦移动，就会触发更新事件。这是我们执行一些OpenLayers逻辑的地方。

获取最顶层的文本内容，因为这是用户希望看到的层。文本存储在 `topLayer` 变量内。该文本将对应于图层标题之一。

我们使用相同的 `getLayers` 层组上的方法和 `forEach` 该方法 `ol.collection` 如前。根据文本是否与图层标题匹配，我们使用 `setVisible` 方法相应地切换图层可见性。

还有更多...

对于此配方，我们选择一次仅显示其他一层。如果需要使所有图层保持可见，而是动态更改图层的堆叠顺序，则可以使用layer `setZIndex` 方法来管理哪些图层位于其他图层之上。

有了图层的集合，例如返回的 `ol.Map.getLayers()` ，您可以 `setAt` 在 `ol.collection` layers对象上使用方法对图层进行重新排序，从而改变其堆叠顺序。这实际上与更改 `zIndex` 属性相同。

还有许多其他方法可以操纵地图图层。在此配方中，我们仅看到了一些：添加，设置标准和任意属性，层堆栈排序等。但是，您可以找到更多方法，例如，删除图层/图层组，更改图层源等。

也可以看看

- 该**管理地图的控制**食谱
- 在**周围的地图视图搬家**食谱
- 的**制约地图范围**的配方

上一节 (/book/web_development/9781785287756/1/ch01lvl1sec11/playing-with-the-map-s-opt

下一节 (/book/web_development/9781785287756/1/ch01lvl1sec13/managing-the-map-s-contro