样式交互呈现意图

有一些 地图交互（例如**Draw**和**Select)** ，它们会根据特征的当前状态（即当前是否正在绘制或**选择**）来更改特征的样式。还有其他创建临时几何图形的 `DragZoom` 交互，例如交互，它在地图上绘制了一个矩形，代表您希望查看的范围。

此食谱向我们展示了如何修改用于这些渲染意图的样式以更改应用程序的外观。

我们将在地图上方创建一个面板，允许用户启用绘图或选择交互。他们将能够 `DragZoom` 随时随地表演。可以在中找到源代码 `ch06/ch06-styling-interaction-render-intents` ，并且**Draw**交互渲染意图的样式将类似于以下屏幕截图：

怎么做...

了解如何 样式交互按照以下说明呈现意图：

1. 创建一个新的HTML文件并添加OpenLayers依赖项，包括 `div` 用于保存 `map` 实例的元素。特别是，创建 `select` JavaScript将要访问的菜单，如下所示：

复制

```
<select id="js-draw-or-select">
  <option value="draw" selected>Draw</option>
  <option value="select">Select</option>
</select>
```

2. 在单独的CSS文件中，添加以下声明以自定义 `DragZoom` 交互样式：

复制

```
.ol-dragzoom {
  border: 3px dotted white;
  background: rgba(50, 186, 132, 0.5);
}
```

3. 设置矢量层和 `map` 实例，如下所示：

复制

```
var vectorLayer = new ol.layer.Vector({
  source: new ol.source.Vector()
});
var map = new ol.Map({
  view: new ol.View({
    zoom: 11, center: [-8238306, 4987133]
  }),
  target: 'js-map',
  layers: [
    new ol.layer.Tile({
      source: new ol.source.Stamen({layer: 'toner'})
    }), vectorLayer
  ]
});
```

4 创建一个函数，将生成一个 `ol.style.Circle` 带有dynamic 的实例 `radius` ：

复制

```
var imageCircle = function(radius) {
  return new ol.style.Circle({
    stroke: new ol.style.Stroke({
      color: 'red', width: 2
    }), radius: radius
  });
};
```

5 创建 使用自定义样式绘制交互，如下所示：

复制

```
var drawInteraction = new ol.interaction.Draw({
  style: [
    new ol.style.Style({
      fill: new ol.style.Fill({
        color: 'rgba(153, 202, 255, 0.5)'
      }),
      stroke: new ol.style.Stroke({
        color: 'blue', width: 2, lineDash: [8, 10]
      }),
      image: imageCircle(15)
    }),
    new ol.style.Style({
      image: imageCircle(10)
    }),
    new ol.style.Style({
      image: imageCircle(5)
    })
  ],
  type: 'Polygon',
```

6 创建 选择与自定义样式的交互，如下所示：

复制

```
var selectInteraction = new ol.interaction.Select({
  style: new ol.style.Style({
    fill: new ol.style.Fill({
      color: 'rgba(255, 255, 31, 0.8)'
    }),
    stroke: new ol.style.Stroke({
      color: 'rgba(255, 154, 31, 0.9)', width: 4
    })
  })
});
```

7 在地图上添加以下内容 `Draw` 和 `DragZoom` 互动内容：

复制

```
map.addInteraction(new ol.interaction.DragZoom());
map.addInteraction(drawInteraction);
```

8 订阅菜单 `change` 上的事件 `select` 并相应地删除或添加 `Draw` 或 `Select` 交互：

复制

```
document.getElementById('js-draw-or-select')
  .addEventListener('change', function() {
    var oldInteraction = (this.value === 'draw')
      ? 'select' : 'draw';
    map.removeInteraction(window[oldInteraction + 'Interaction']);
    map.addInteraction(window[this.value + 'Interaction']);
  });
```

怎么运行的...

我们已经排除了显示所有CSS和HTML的可能性，但是请查看本书的源代码以获取完整的实现。包括在内的各部分应提供足够的隔离性，以便可以正常运行。

You may have observed that the `DragZoom` interaction (`ol.interaction.DragZoom`) does not have a `style` property like the other two interactions in this example. This is because when the `DragZoom` interaction draws out an extent over the map to zoom to, the rectangle is actually an HTML element rather than part of the map canvas. With this in mind, it should make sense as to why the box is styled with the following CSS:

Copy

```
.ol-dragzoom {
  border: 3px dotted white;
  background: rgba(50, 186, 132, 0.5);
}
```

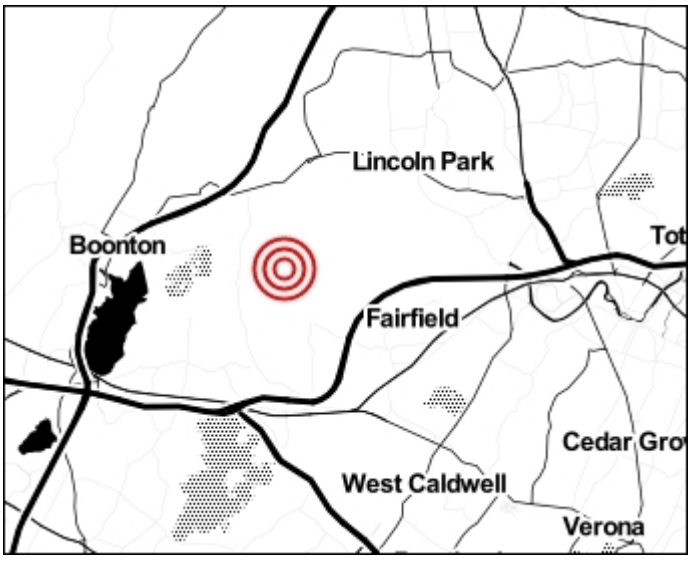
We chose to style the border as a dotted white line and applied a semitransparent turquoise color for the background. It's a good idea to put custom CSS that overrides the OpenLayers defaults within a separate style sheet for easier maintenance down the line.

Unlike `DragZoom`, the `Draw` and `Select` interactions do have a `style` property that we must take 的优势，以便自定义渲染意图的样式。让我们仔细看看它们是如何实现的：

复制

```
var imageCircle = function(radius) {
  return new ol.style.Circle({
    stroke: new ol.style.Stroke({
      color: 'red', width: 2
    }), radius: radius
  });
};
```

对于 `Draw` 交互，我们决定使用三个圆的组合构造点图像（该点用于标识要素中下一个几何点的位置），使其看起来像一个目标。由于 `ol.style.Circle` 构造函数将被多次调用，但 `radius` 大小不同，因此我们将其捕获到可重用的函数中，即 `imageCircle` 。每个圆圈图像都包含一个简单的红色笔触，没有填充颜色。结果显示在以下屏幕截图中：



复制

```
var drawInteraction = new ol.interaction.Draw({
  style: [
    new ol.style.Style({
      fill: new ol.style.Fill({
        color: 'rgba(153, 202, 255, 0.5)'
      }),
      stroke: new ol.style.Stroke({
        color: 'blue', width: 2,
        lineDash: [8, 10]
      }),
      image: imageCircle(15)
    }),
    new ol.style.Style({
      image: imageCircle(10)
    }),
    new ol.style.Style({
      image: imageCircle(5)
    })
  ]
```

我们砍掉了 `Draw` 这段代码中交互实例化的最后几个属性，因为其他配方看起来很熟悉。我们想提请您注意我们分配给 `style` 酒店的物品。

实际上，我们 `ol.style.Style` 在数组中创建了三个实例。第一个主要样式包含多边形填充和笔触的样式，以及点样式的第一个最外环，这是由我们的 `imageCircle` 函数生成的。该 `fill color` 属性是RGBA形式的字符串-OpenLayers接受的另一个变量。

数组中的以下两种样式创建该点的其他内环。所有 `image` 属性都分配了我们 `imageCircle` 函数的结果；每个都经过不同大小的半径，从而产生我们希望达到的目标效果。

了解样式可以在数组中建立很有用。您可以设计一些非常有创意的样式，如果您有需要或希望这样做的话。

该 `Select` 还与一些通过自己的方式提供的交互 `style` 特性，以颜色选择多边形用浅黄色填充和厚厚的暗橙色中风。

复制

```
document.getElementById('js-draw-or-select')
  .addEventListener('change', function() {
    var oldInteraction = (this.value === 'draw') ? 'select' : 'draw';
    map.removeInteraction(window[oldInteraction + 'Interaction']);
    map.addInteraction(window[this.value + 'Interaction']);
  });
```

我们将在本食谱中讨论的最后一段代码在将活动的交互切换到或从 `Draw` 或切换后进行了艰苦的工作 `Select` 。

当 `select` 菜单选项更改，新值的两种可能性之一： `draw` 或 `select` 。由此，我们可以确定地图上先前进行的互动是什么。我们使用地图的 `removeInteraction` 方法删除了这种互动，因为两者相互干扰。

互动变量是 `window` 对象的属性， `window.drawInteraction` 或者 `window.selectInteraction` 。使用数组表示法进行对象查找，我们可以动态删除或添加适用的交互。例如，如果用户从菜单切换到“**选择**”交互 `select` ，则 `oldInteraction` 变量将包含 `'drawInteraction'` 字符串，并且要应用的交互变为 `'selectInteraction'` 。因此， `window['selectInteraction']` 与 `window[this.value + 'Interaction']` 此方案中的相同。

还有更多...

如果不立即可见， `DragZoom` 请在**按住**键盘上的**Shift**键然后单击鼠标左键以开始拖动时激活交互。

该 `DragZoom` 互动有一个叫做物业 `condition` 能够为了使拖动，而不是使用提供了不同的事件**转变**的关键。默认条件值为 `ol.events.condition.shiftKeyOnly` ，但随时可以探索更适合您的应用程序的其他条件。

也可以看看

- **基于几何类型**配方的**样式功能**
- 第5章 (/book/web_development/9781785287756/5)，**添加控件中的“修改功能”**配方 (/book/web_development/9781785287756/5)
- 该**造型层**配方

上一节 (/book/web_development/9781785287756/6/ch06lvl1sec57/styling-based-on-feature-at

下一节 (/book/web_development/9781785287756/6/ch06lvl1sec59/styling-clustered-features)