修改图层外观

我们已经看到了如何 在执行一些基本的层的外观修改**变化层的不透明度**在配方第2章(/book/web_development/9781785287756/2)，**添加栅格图层**。我们还看到了如何重新排序图层堆栈以及显示或隐藏图层。对于大多数Web映射应用程序而言，这种层修改可能就足够了，但是我们将介绍此配方的其他一些更高级的技术。

在此配方期间，我们将修改从图块服务返回的栅格图块的外观。众所周知，栅格图块将作为图像返回，这意味着当通过画布渲染器应用图像时，我们可以操纵图像的颜色。

在将修改后的图块返回到浏览器之前，还可以在服务器（可能是代理服务器）上执行颜色操作。另外，如果您可以控制图块服务本身，则可以提供其他颜色方案的栅格图块。

当然，我们将研究如何在客户端的JavaScript代码中实现这一点。我们将 `ol.source.Raster` 在本书中首次使用图层源。如我们指定的，它提供了一种将输入像素值转换为不同像素值以进行输出的方法。

我们将提供在黑色，白色和其他颜色之间切换图层的功能。我们还将允许用户调整图层的亮度。可以在中找到源代码 `ch07/ch07-modifying-layer-appearance` 。接下来显示完成的应用程序的屏幕快照，用户已将图层切换为黑白模式。

请注意，对于栅格图块的每个像素，我们将在较低级别上执行操作。这确实消耗了计算能力，并且性能问题可能很明显（也就是说，该层可能需要一些时间才能从一种配色方案转换为另一种配色方案）。考虑您在应用程序中支持的用户，并为此类任务适当地选择最佳技术和功能。

做好准备

对于一些颜色操作，我们将在不同的色彩空间之间进行转换（RGB到HCL，反之亦然）。我们还将修改颜色的亮度。我们将依靠伟大的D3颜色库（https://github.com/d3/d3-color (https://github.com/d3/d3-color)）为我们处理繁重的工作，以便我们专注于应用程序代码。下载D3颜色库，因为我们将在HTML中引用它。

怎么做...

1. 创建一个新的HTML文件，并包含OpenLayers依赖项和一个 `div` 用于保存地图的元素。如本食谱的“**准备就绪**”部分所述，还包括 `D3` 颜色库。特别是，我们在侧面板上的控件具有以下标记：

复制

```
<form>
  <select id="js-colour">
    <option value="colour">Colour</option>
    <option value="blackAndWhite">Black &amp;amp; White</option>
  </select>
  <button id="js-darker"></button>
  <button id="js-lighter"></button>
</form>
```

2. 创建一个自定义JavaScript文件并订阅color select `change` 事件，强制重新渲染源：

复制

```
var selectElem = document.getElementById('js-colour');
selectElem.addEventListener('change', function() {
  raster.changed();
});
```

3 设置默认 较暗和较亮的属性，订阅 `click` 每个按钮上的事件，更新相关的亮度属性，并强制重新渲染源：

复制

```
var goDarker = { enable: false, level: 0.1 };
document.getElementById('js-darker')
    .addEventListener('click', function() {
        goDarker.enable = true;
        raster.changed();
    });

var goLighter = { enable: false, level: 0.1 };
document.getElementById('js-lighter')
    .addEventListener('click', function() {
        goLighter.enable = true;
        raster.changed();
    });
```

4 使用像素操作创建栅格源以修改图层外观：

复制

```
var raster = new ol.source.Raster({
  sources: [
    new ol.source.Stamen({
      layer: 'watercolor'
    })
  ],
  threads: 0,
  operation: function(pixels, data) {
    if (pixels[0][0] == 0 && pixels[0][1] == 0 && pixels[0][2] == 0) {
      return [0, 0, 0, 0];
    }

    var rgb = d3_color.rgb(
      pixels[0][0],
      pixels[0][1],
      pixels[0][2]
    );

    if (data.blackAndWhite) {
```

5 设置正确 栅格源的属性如下：

复制

```
raster.setAttributions(ol.source.Stamen.ATTRIBUTIONS);
```

6 在像素操作实际发生之前，为图层操作准备一些配置：

复制

```
raster.on(ol.source.RasterEventType.BEFOREOPERATIONS, function(event) {
  var data = event.data;
  data.blackAndWhite = selectElem.value === 'blackAndWhite';

  if (goDarker.enable) {
    data.goDarker = true;
    data.level = goDarker.level;
    goDarker.enable = false;
    goDarker.level += 0.1;
    goLighter.level -= 0.1;
  }
  else if (goLighter.enable) {
    data.goLighter = true;
    data.level = goLighter.level;
    goLighter.enable = false;
    goLighter.level += 0.1;
    goDarker.level -= 0.1;
  }
});
```

7 实例化一个新实例 `map` ：

复制

```
var map = new ol.Map({
  layers: [
    new ol.layer.Image({source: raster})
  ], target: 'js-map',
  view: new ol.View({
    center: [-4383204, 6985732], zoom: 3
  })
});
```

怎么运行的...

我们已经使用了Bootstrap CSS框架以样式化和组织HTML，为简洁起见，已将其省略。请查看随附的源代码以获取全部详细信息。

在UI中，有一些控件可以控制图层的外观。该 `select` 菜单中包含的颜色或黑色和白色之间转换的瓷砖两个选项。我们订阅该 `change` 事件，并通过 `raster.changed()` 从处理程序中调用来强制重新渲染栅格源：

复制

```
selectElem.addEventListener('change', function() {
  raster.changed();
});
```

通过开始渲染，将执行像素操作，其说明如下：

复制

```
var goDarker = { enable: false, level: 0.1 };
document.getElementById('js-darker')
    .addEventListener('click', function() {
        goDarker.enable = true;
        raster.changed();
    });
```

我们设置一个 `goDarker` 对象，指定一些当前状态。它跟踪是否已启用它以及当前级别。在执行像素操作之前，我们需要知道发生了哪些用户操作来触发图层的重新渲染。为了实现这一目标，我们相应地使该对象保持最新。

如果用户第一次按下较暗的（减号）按钮，则图块将变暗一定数量 `0.1` 的原始图，并且级别将增加到 `0.2` 。如果用户想再次变暗，则增加级别将是 `0.2` 这次，从而使图层总的 `0.2` 颜色比原始颜色暗，依此类推。在后面的说明中，我们将看到如何增加级别。

我们订阅 `click` 按钮上的事件，将对象的 `enable` 属性设置 `goDarker` 为true，然后触发栅格源重新渲染。

我们对 `lighter` 按钮应用了相同的设置和逻辑，因此在此不再赘述，因为在前面的讨论中它应该是不言而喻的。

栅格源设置 跨越多行，因此让我们将其分解成小块进行解释：

复制

```
var raster = new ol.source.Raster({
  sources: [
    new ol.source.Stamen({
      layer: 'watercolor'
    })
  ],
```

栅格源允许我们对像素值执行任意操作，这正是我们想要做的。第一个属性， `sources` 期望其中的一个数组 `sources` 将提供操作输入。我们传入了来自 `Stamen` 图块提供者的源。

复制

```
threads: 0,
```

下一个属性是 `threads` ，我们将其设置为零。默认情况下，OpenLayers在工作线程中执行像素操作以提高性能。这样做会引起范围的复杂性，必须将函数和值传递给线程上下文。否则，它们将不可用。出于演示目的，我们避免了这种复杂性。对于一个很好的介绍工作线程，请访问 http://www.html5rocks.com/en/tutorials/workers/basics/ (http://www.html5rocks.com/en/tutorials/workers/basics/)。实施这些高级功能时，请记住检查浏览器支持（http://caniuse.com/#search=workers (http://caniuse.com/#search=workers)）。

复制

```
operation: function(pixels, data) {
    if (pixels[0][0] == 0 && pixels[0][1] == 0 && pixels[0][2] == 0) {
      return [0, 0, 0, 0];
    }
```

该 `operation` 属性随函数一起提供，该函数将处理输入的像素数据，然后返回修改后的像素数据，该数据将分配给栅格源。

所述 `pixels` 参数是一个数组。数组的第一项包含它们自己的数组中的RGB值。这是我们最感兴趣的数据。

如果数组中的每个像素的红色，绿色和蓝色的值均为零，那么我们将 `return` 尽早避免执行任何可能触发JavaScript `NaN` （而非数字）错误的转换。我们返回一个零数组，该数组与传递给该操作的数据相同。

复制

```
    var rgb = d3_color.rgb(
      pixels[0][0],
      pixels[0][1],
      pixels[0][2]
    );
```

为了使用D3颜色库执行一些像素颜色操作，我们首先获取像素输入，然后使用 `d3_color.rgb` 方法将其转换为D3 RGB对象，并将结果存储在变量中，即 `rgb` 。该 `pixels[0][0]` 输入对应红色，`pixels[0][1]` 对应绿色，等等。

复制

```
    if (data.blackAndWhite) {
      var hcl = d3_color.hcl(rgb);
      hcl.c = 0;
      rgb = d3_color.rgb(hcl);
    }
```

如果 `data.blackAndWhite` 财产为true，这意味着用户将 `select` 菜单值设置为**Black&White**。要将颜色转换为灰度，我们必须将 `chroma` HCL颜色空间的属性设置为零。为此，我们首先使用该 `d3_color.hcl` 方法将颜色从RGB颜色空间转换为HCL颜色空间。在HCL中获得颜色后，将 `chroma` 属性（ `hcl.c` ）更新为零。然后，我们返回OpenLayers期望的RGB颜色空间（ `rgb = d3_color.rgb(hcl)` ）。

复制

```
    if (data.goDarker) {
      rgb = rgb.darker(data.level);
    }
    else if (data.goLighter) {
      rgb = rgb.brighter(data.level);
```

> 注意
>
> 这些值是 `data` 从操作前事件处理程序中分配给对象的，我们将在后面解释。

如果用户要求变暗或变亮，我们可以 `rgb` 通过调用方法 `darker` 或 `brighter` 在D3 RGB对象上并提供当前的亮度等级（ `data.level` ）为变量分配新的RGB颜色。

复制

```
return [rgb.r, rgb.g, rgb.b, 255];
```

最后，我们 `return` 从D3 RGB对象中检索了一个数组，其索引0-2构成了受操纵的RGB内容。也就是说， `rgb.r` 是红色的。

这样就完成了栅格源的设置，但是我们还需要在栅格源操作之前执行一些任务。为此，我们在栅格源上订阅了before操作事件：

复制

```
raster.on(ol.source.RasterEventType.BEFOREOPERATIONS,function(event) {
  var data = event.data;
  data.blackAndWhite = selectElem.value === 'blackAndWhite';
```

发布操作之前的事件时，它为我们提供了扩展 `event.data` 对象的机会，该对象将传递给 `operations` 栅格源的方法（ `data` 如我们所见，也被称为栅格源）。

我们将 `blackAndWhite` 属性附加到 `data` 对象上，该对象根据条件的评估（用户是否从菜单中选择了“**黑白**”选项 `select` ）被分配为true或false的值。

复制

```
  if (goDarker.enable) {
    data.goDarker = true;
    data.level = goDarker.level;
    goDarker.enable = false;
    goDarker.level += 0.1;
    goLighter.level -= 0.1;
  }
```

如果用户单击 在减号按钮上变暗时， `goDarker` 分配数据的属性 `true` 。我们还检索了要应用的当前暗度（ `goDarker.level` ），并将此值分配给 `data.level` 操作函数内部使用。

由于这些更改（变暗）将应用于下一个操作，但不一定应用于该操作之后的操作，因此我们必须相应地更新 `goDarker` 对象。我们将其标记为已禁用（ `goDarker.enable = false` ），并增加用于下一个黑暗阶段（ `goDarker.level += 0.1` ）的黑暗程度。

重要的是，我们还降低了 `goLighter` 对象的级别。亮度必须与黑暗有关，因此，如果黑暗增加，亮度会降低。这样可确保在亮度级别之间进行调整时保持平滑的一致性。

如果用户使图层变浅而不是变暗，我们将执行类似（但相反）的逻辑。我们希望前面的解释也将涵盖此细节。

我们对before操作的事件类型使用了详细查找： `ol.source.RasterEventType.BEFOREOPERATIONS` 。结果是一个字符串 `'beforeoperations'` ，我们可以代替使用。但是，了解这些事件在OpenLayers代码中的何处注册很有趣。

还有一个操作后事件（ `ol.source.RasterEventType.AFTEROPERATIONS` ）。作为一个想法，如果您认为您的操作可能会长时间运行并影响性能，则可以考虑在操作前显示加载图形，然后在发布操作后事件后隐藏该加载图形。

也可以看看

- 第2章 (/book/web_development/9781785287756/2)，**添加栅格图层中**的**更改图层不透明度**配方 (/book/web_development/9781785287756/2)
- 第2章 (/book/web_development/9781785287756/2)，**添加栅格图层中**的**创建图像图层**配方 (/book/web_development/9781785287756/2)

上一节 (/book/web_development/9781785287756/7/ch07lvl1sec66/drawing-in-freehand-mode)

下一节 (/book/web_development/9781785287756/7/ch07lvl1sec68/adding-features-to-the-vec