OpenLayers 3 入门教程

摘要

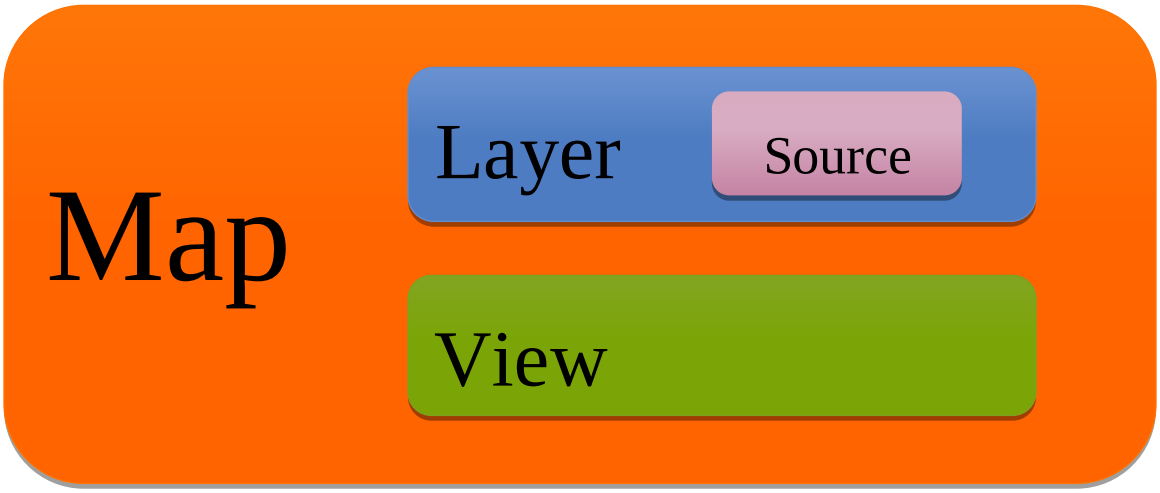
OpenLayers 3 对 OpenLayers 网络地图库进行了根本的重新设计。版本 2 虽然被广泛使用，但从 JavaScript 开发的早期发展阶段开始，已日益现实出它的落后。 OL3 已运用现代的设计模式从底层重写。

最初的版本旨在支持第 2 版提供的功能，提供大量商业或免费的瓦片资源以及最流行的开源矢量数据格式。与版本 2 一样，数据可以被任意投影。最初的版本还增加了一些额外的功能，如能够方便地旋转地图以及显示地图动画。

OpenLayers 3 同时设计了一些主要的新功能，如显示三维地图，或使用 WebGL 快速显示大型矢量数据集，这些功能将在以后的版本中加入。

目录

基本概念



Map

OpenLayers3 的核心部件是 Map（`ol.Map`）。它被呈现到对象 `target` 容器（例如，包含在地图的网页上的 `div` 元素）。所有地图的属性可以在构造时进行配置，或者通过使用 setter 方法，如 setTarget（ ）。

```
<div id="map"style="width: 100%, height: 400px"></div>
<script>
var map = new ol.Map({target: 'map'});
</script>
```

View

`ol.View` 负责地图的中心点，放大，投影之类的设置。

一个 `ol.View` 实例包含投影 `projection`，该投影决定中心 `center` 的坐标系以及分辨率的单位，如果没有指定（如下面的代码段），默认的投影是球墨卡托（EPSG：3857），以米为地图单位。

放大 `zoom` 选项是一种方便的方式来指定地图的分辨率，可用的缩放级别由 `maxZoom`（默认值为 28）、`zoomFactor`（默认值为 2）、`maxResolution`（默认由投影在 256×256 像素瓦片的有效成都来计算）决定。起始于缩放级别 0，以每像素 `maxResolution` 的单位为分辨率，后续的缩放级别是通过 `zoomFactor` 区分之前的缩放级别的分辨率来计算的，直到缩放级别达到 `maxZoom`。

```
map.setView(new ol.View({
  center: [0, 0],
  zoom: 2
 }));
```

Source

OpenLayers3 使用 `ol.source.Source` 子类获取远程数据图层，包含免费的和商业的地图瓦片服务，如 OpenStreetMap、Bing、OGC 资源（WMS 或 WMTS）、矢量数据（GeoJSON 格式、KML 格式...）等。

```
var osmSource = new ol.source.OSM();
```

Layer

一个图层是资源中数据的可视化显示，OpenLayers3 包含三种基本图层类型：`ol.layer.Tile`、`ol.layer.Image` 和 `ol.layer.Vector`。

`ol.layer.Tile` 用于显示瓦片资源，这些瓦片提供了预渲染，并且由特定分别率的缩放级别组织的瓦片图片网格组成。

`ol.layer.Image` 用于显示支持渲染服务的图片，这些图片可用于任意范围和分辨率。

`ol.layer.Vector` 用于显示在客户端渲染的矢量数据。

```
var osmLayer = new ol.layer.Tile({source: osmSource});
 map.addLayer(osmLayer);
```

总结

上述片段可以合并成一个自包含视图和图层的地图配置：

```
<divid="map"style="width: 100%, height: 400px"></div>
<script>
new ol.Map({
   layers: [
new ol.layer.Tile({source: new ol.source.OSM()})
   ],
   view: new ol.View({
     center: [0, 0],
     zoom: 2
   }),
   target: 'map'
 });
</script>
```

Openlayers 3 实践

1 地图显示

1.1 创建一副地图

在 openlayers 中，Map 是图层、各种交互以及处理用户交互控件的集合，地图由三个基本成分生成：标记，样式声明和初始化代码。以下是一个完整的 OpenLayers3 地图示例。

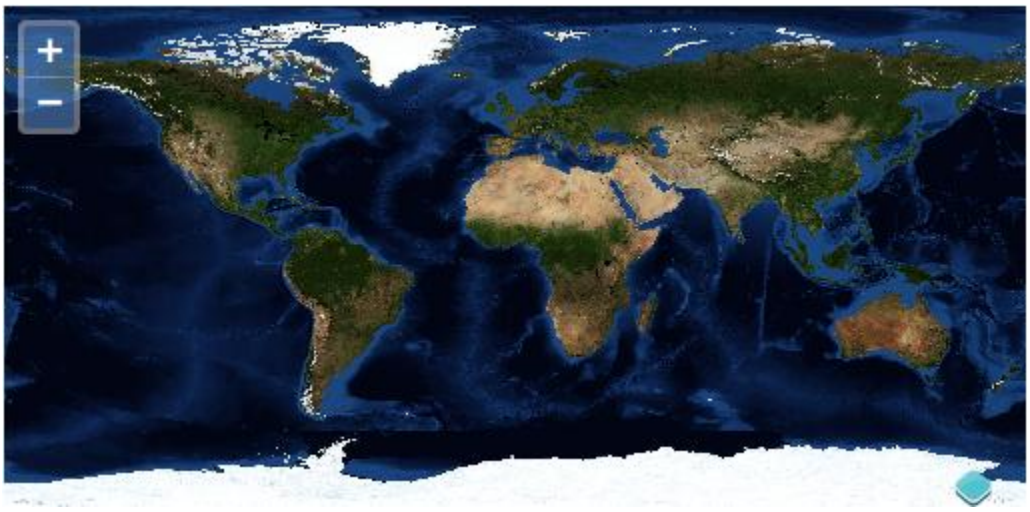
```
<!doctype html>
<htmllang="en">
<head>
<linkrel="stylesheet"href="ol3/ol.css"type="text/css">
<style>
#map{
height:256px;
width:512px;
}
</style>
<title>OpenLayers 3 example</title>
<script src="ol3/ol.js"type="text/javascript"></script>
</head>
<body>
<h1>My Map</h1>
<divid="map"></div>
<script type="text/javascript">
varmap=newol.Map({
target:'map',
layers:[
newol.layer.Tile({
title:"Global Imagery",
source:newol.source.TileWMS({
url:'http://maps.opengeo.org/geowebcache/service/wms',
params:{LAYERS:'bluemarble',VERSION:'1.1.1'}
})
})
],
view:newol.View({
```

```
projection:'EPSG:4326',
center:[0,0],
zoom:0,
maxResolution:0.703125
})
});
</script>
</body>
</html>
```

（1） 下载 https://github.com/openlayers/ol3-workshop/archive/resources.zip，并将该文件夹放在网络服务器的根目录下；

（2） 创建一个新的文件，命名为 `map.html`，将以上代码复制进该文件后放入下载的文件夹的根目录下；

（3） 在浏览器中输入：http://localhost:8000/ol_workshop/map.html，我们将打开一个工作的地图。



成功地创建了第一张地图，我们将继续关注地图的组成部分，详见 1.2 剖析你的地图。

1.2 剖析你的地图

正如前一部分演示的那样，一副地图通过将标记，样式声明和初始化代码三部分组织在一起而生成，接下来将详细的介绍这三个组成部分。

1.2.1 地图标记

标记为上例中的地图生成的一个文档元素：

```
<divid="map"></div>
```

在此示例中，我们用`<div>`元素作为地图显示的容器，其他块级元素也能做视图的容器。在这种情况下，我们设置容器的`id`属性，所以我们可以将其作为地图的对象。

1.2.2 地图样式

OpenLayers 带有一个默认的样式表，指定地图相关的元素应如何显示，我们明确的将样式表引用到`map.html`页面中。

OpenLayers 不对地图的大小做预定义，因此在默认样式表之后，我们需要包括至少一个自定义样式声明来说明地图在页面上的空间。

```
<linkrel="stylesheet"href="ol3/ol.css"type="text/css">
<style>
#map {
height:256px;
width:512px;
 }
</style>
```

在该示例中，我们使用地图容器的`id`值作为选择器，并明确定义地图容器的高为 256px，宽为 512px。样式声明直接包含在文档的<head>部分。在大多数情况下，地图相关的样式说明是链接到外部样式表的一个大型主题网站的一部分。

1.2.3 地图初始化

生成地图的下一步包含一些初始化代码，在该示例中，我们在文档的`<body>`前添加`<script>`元素来实现。

```
<script>
var map =new ol.Map({
   target:'map',
   layers: [
new ol.layer.Tile({
       source:new ol.source.TileWMS({
         url:'http://maps.opengeo.org/geowebcache/service/wms',
         params: {LAYERS:'bluemarble', VERSION:'1.1.1'}
       })
     })
   ],
   view:new ol.View({
     projection:'EPSG:4326',
     center: [0, 0],
     zoom:0,
     maxResolution:0.703125
   })
 });
</script>
```

注：这些步骤的顺序很重要，OpenLayers 库必须在在自定义脚本执行之前加载，在此示例中，OpenLayers 库在文档的`<head>`部分加载（`<script src="ol3/ol.js"></script>`）。同样的，在文档中作为显示容器的元素（该实例中为`<div id="map"></div>`）准备好之前，自定义地图初始化代码是不能执行的，将初始化代码添加到文档中`<body>`的后面，我们能在地图生成前，确保库已加载，显示容器已准备好。

接下来，将详细介绍初始化脚本的内容。脚本创建了一个包含一些配置选项的 `ol.Map` 对象：

```
target:'map'
```

我们使用显示容器的 `id` 属性来告诉地图构造函数将地图交付到何处，在该示例中，我们通过字符串“map”作为地图构造函数的对象。这个语法方便快捷，也可以更详细的使用元素的直接引用（e.g. `document.getElementById("map")`）。

图层配置创建了一个显示在地图中的图层：

```
layers: [
```

```
new ol.layer.Tile({
   source:new ol.source.TileWMS({
     url:'http://maps.opengeo.org/geowebcache/service/wms',
     params: {LAYERS:'bluemarble', VERSION:'1.1.1'}
   })
  })
],
```

不用担心对以上的语法不了解，图层创建在后续章节中会有详细的介绍。最重要的是理解地图显示的是图层的集合。为了显示一副地图，至少需要添加一个图层。

最后一步是定义视图，指定投影、中心点，放大级别，在该示例中，还指定了 maxResolution，以确保请求的范围不超过 GeoWebCache 能处理的范围。

```
view:new ol.View({
  projection:'EPSG:4326',
  center: [0, 0],
  zoom:0,
  maxResolution:0.703125
})
```

以上，成功剖析了一副地图的显示，接下来将介绍更多关于 OpenLayers 的开发。

1.3 Openlayers 的资源

OpenLayers 库提供丰富的功能，尽管开发者对每个功能都提供了示例，并且让其他有经验的程序员找到属于他们自己的方式来组织代码，很多用户仍觉得从头开始是一个挑战。

1.3.1 通过示例学习

新用户很可能会发现研究 OpenLayer 的示例代码以及库的功能是开始起步最有效的方式。

- http://openlayers.org/en/master/examples/

1.3.2 查看 API 参考

在理解了构成以及控制一幅地图的基本组成之后，搜索 API 帮助文档以了解方法签名、对象属性的详细信息。

- http://openlayers.org/en/master/apidoc/

2 图层与资源

2.1 网络地图服务图层

当向地图中添加图层，图层的资源通常用来获取将要显示的数据，数据请求可以是影像数据，也可以是矢量数据。栅格数据是服务端提供的图片信息，矢量数据是服务器交付的结构化信息，在客户端（浏览器）进行显示。

有许多不同类型的服务提供栅格地图数据，这部分涉及到符合 OGC 网络地图服务（WMS）规范的供应商。

2.1.1 创建图层

在 1.1 创建一副地图创建的地图示例的基础上，修改图层来理解其运行机制。

```
layers: [
new ol.layer.Tile({
   title:"Global Imagery",
   source:new ol.source.TileWMS({
     url:'http://maps.opengeo.org/geowebcache/service/wms',
     params: {LAYERS:'bluemarble', VERSION:'1.1.1'}
   })
 })
]
```

完整示例代码详见 1.1 创建一副地图。

2.1.2 The ol.layer.Tile 构造函数

在 1.1 创建一副地图创建的示例中，使用的数据资源是 `ol.source.TileWMS`，我们可以从 title 关键字的字面上理解它的含义，但是基本上来说，这里的关键字可以是任意名称，在

OpenLayers 3 中，图层和资源有一个区别，而在 OpenLayers 2 中，这两部分一起组成了一个图层。

`ol.layer.Tile` 表示图像的规则网格，而 `ol.layer.Image` 表示单个图像，基于图层类型，我们将使用不同的资源（`ol.source.TileWMS` 与 `ol.source.ImageWMS`）。

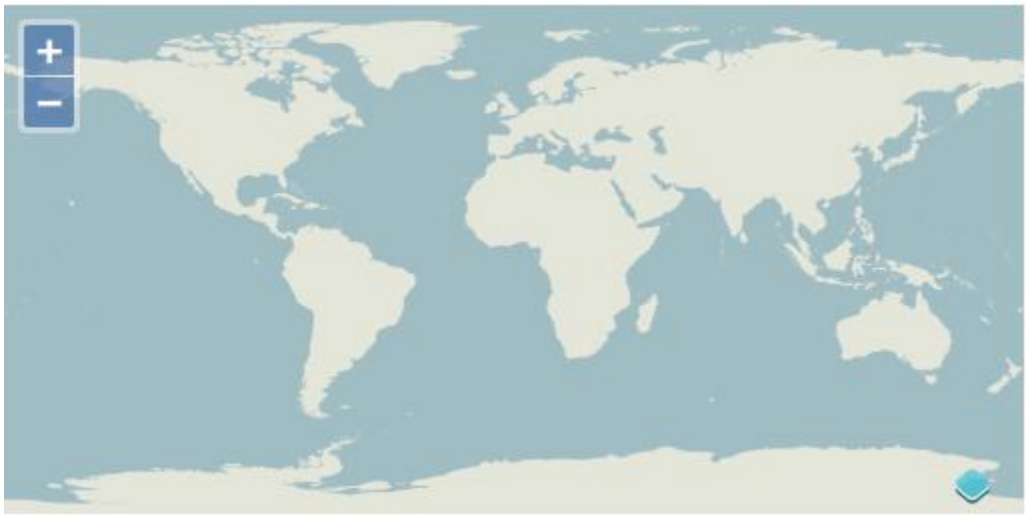
2.1.3 Theol.source.TileWMS 构造函数

url 指向的是 WMS 服务的在线资源，params 是对象参数名称及其值，由于在 OpenLayers 中，默认 WMS 版本是 1.3.0，如果 WMS 不支持该版本，需要在 params 中提供一个低版本。

（1） 该示例中 WMS 提供了一个名为“openstreetmap”的图层，将 LAYERS 参数的值由“bluemarble”改为“openstreetmap”，代码如下：

```
new ol.layer.Tile({
  title:"Global Imagery",
  source:new ol.source.TileWMS({
    url:'http://maps.opengeo.org/geowebcache/service/wms',
    params: {LAYERS:'openstreetmap', VERSION:'1.1.1'}
  })
})
```

（2） 改变图层与资源，用单一图像取代瓦片，再这个过程中，需要将资源的 url 修改为 http://suite.opengeo.org/geoserver/wms，将 LAYERS 参数的值修改为“`opengeo:countries`”，完成修改后，利用浏览器的开发工具，确保请求的是单一图像而不是 256×256 像素的瓦片。



了解了从网络地图服务动态获取数据的机制，接下来将深入了解瓦片缓存服务。

2.2 瓦片缓存

默认情况在，瓦片图层请求一个 256×256 像素的图像来填充地图视窗，当你平移或缩放地图，将发出更多图片请求来填充你没有访问过的地方。浏览器会缓存一些请求的图像，这通常需要大量的处理器来动态渲染图像。

由于瓦片图层以规律的网格请求图像，这使得服务器能够缓存这些图片请求并且在下次浏览相同区域的时候返回该缓存结果，从而获得更好的性能。

2.2.1 ol.source.XYZ

网络地图服务规范使得客户端能够请求的内容具有灵活性，如果没有限制，在实践中，缓存会变得困难甚至不可能实现。

另一种几段情况是，服务器可能只提供固定缩放级别和规律网格的瓦片，这种情况可以概括为 XYZ 资源的瓦片图层——X/Y 代表网格的行与列，Z 代表缩放级别。

2.2.2ol.source.OSM

OpenStreetMap (OSM)投影是为了收集并免费提供世界地图的数据，OSM 提供了一些不同的数据渲染作为瓦片缓存集，这些渲染符合基本的 XYZ 网格配置，并且可以在 OpenLayers 地图中使用。`ol.source.OSM` 图层可以访问 OpenStreetMap 瓦片资源。

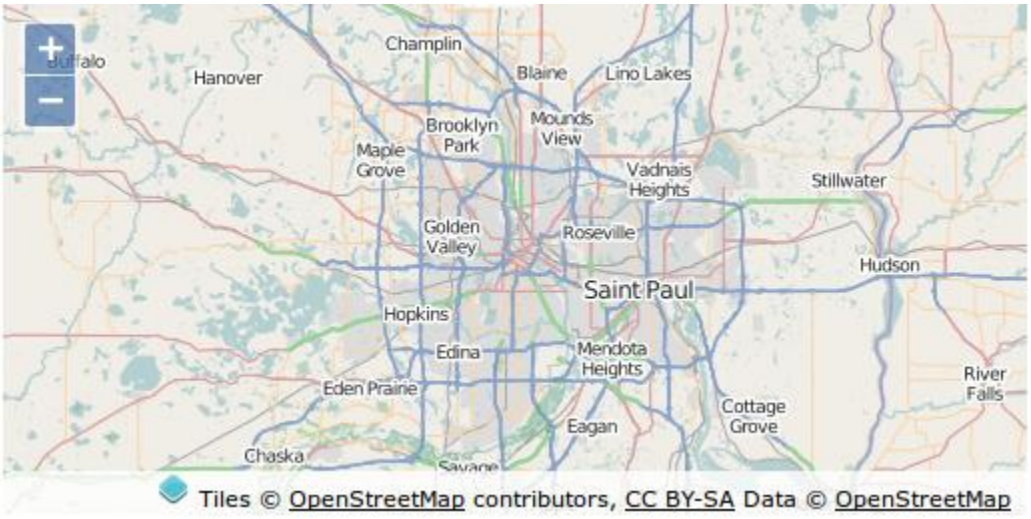
（1） 打开 1.1 创建一副地图创建的 map.html 文件，将地图初始化代码替换为以下代码：

```
<script>
    var map =new ol.Map({
  target:'map',
  layers: [
new ol.layer.Tile({
      source:new ol.source.OSM()
    })
  ],
  view:new ol.View({
    center: ol.proj.transform([-93.27, 44.98], 'EPSG:4326', 'EPSG:3857'),
    zoom:9
  }),
  controls: ol.control.defaults({
    attributionOptions: {
      collapsible:false
    }
  })
 });
</script>
```

（2） 在文档的`<head>`中，添加以下图层属性的样式说明：

```
<style>
#map {
width:512px;
height:256px;
  }
.ol-attributiona {
color:black;
  }
</style>
```

（3） 保存修改，在浏览器中查看该页面：http://localhost:8000/ol_workshop/map.html：



2.2.2.1 投影

回顾地图的视图定义：

```
view:new ol.View({
 center: ol.proj.transform([-93.27, 44.98], 'EPSG:4326', 'EPSG:3857'),
 zoom:9
})
```

地理空间数据可能来自各种坐标参照系，一个数据集可能是以度为单位的地理（经纬）坐标系，另一个可能是以米为单位的投影坐标系，对坐标系的全面讨论超出了本实践的范围，但了解其基本概念是很重要的。

OpenLayers 3 需要知道所使用数据的坐标系，在内部，由 ol.proj.Projection 对象展现，ol.proj 命名空间中的 transform 方法使用字符串表示坐标参考系（上述示例中的`"EPSG:4326"`以及`"EPSG:3857"`）。

OpenStreetMap 瓦片数据是墨卡托投影，因此，我们需要使用墨卡托坐标了设置初始化时的中心点。由于一个地方的地理坐标相对来说更容易知道，使用`ol.proj.transform`方法将地理坐标系（`"EPSG:4326"`）转化为墨卡托坐标系（`"EPSG:3857"`）。

OpenLayers 3 包含地理坐标系与墨卡托坐标系间相互转换的方法，因此我们可以使用 `ol.proj.transform` 方法而不需要任何额外的工作。如果想要使用其他投影的数据，再使用 `ol.proj.transform` 方法之前需要添加一些额外的信息。

例如，使用"EPSG:21781" 坐标参照系的数据，添加以下两条 script 标签到页面中：

```
<script src="http://cdnjs.cloudflare.com/ajax/libs/proj4js/2.2.1/proj4.js"
        type="text/javascript"></script>
<script src="http://epsg.io/21781-1753.js"type="text/javascript"></script>
```

然后在应用程序代码中，注册该投影并设置其有效范围，代码如下：

```
// This creates a projection object for the EPSG:21781 projection
// and sets a "validity extent" in that projection object.
var projection = ol.proj.get('EPSG:21781');
projection.setExtent([485869.5728, 76443.1884, 837076.5648, 299941.7864]);
```

2.2.2.2 图层创建

```
layers: [
new ol.layer.Tile({
   source:new ol.source.OSM()
 })
],
```

之前的示例中，创建一个新图层后，将其添加到地图配置对象的图层数组中，以上代码接受资源的所有默认选项。

2.2.2.3 样式

```
.ol-attributiona {
color:black;
}
```

如何处理地图控件是本章节意外的内容，但是这里的样式声明让你先睹为快。默认情况下 `ol.control.Attribution` 控件被添加到所有地图中，这使得地图视窗中显示图层资源的归属信息来源，上述声明改变了地图中归属信息的样式（版权行再地图的右下方）。

掌握了公开可用的缓存瓦片集的图层用法，接下来将介绍专有的栅格图层，详见 2.3 专有栅格图层。

2.2.2.4 属性控件配置

默认情况下，`ol.control.Attribution` 控件在页面上添加了一个 `i`（information）按钮，点击即可显示归属地信息。为了符合 OpenStreetMap 的使用条款，并且将 OpenStreetMap 的归属地信息一直展现出来，可添加以下代码在 `ol.Map` 构造函数中最为可选对象。

```
controls:ol.control.defaults({
attributionOptions:{
collapsible:false
}
})
```

这段代码移除了 `i` 按钮，使得归属地信息一直展现在视图中。

2.3 专有栅格图层（Bing）

在前面的章节中，图层是基于符合标准的 WMS 以及自定义瓦片缓存显示的，在线地图（或者瓦片地图客户端）主要是通过可用的专有地图瓦片服务进行广泛推广，OpenLayers 提供的图层类型能通过使用它们的 API 来调用这些专有服务。

本章节使用的示例，是在上一章节示例的基础上，添加一个使用 Bing 瓦片的图层。

（1） 将 `map.html` 文件中配置 OSM 资源的代码替换为 `ol.source.BingMaps`：

```
source:new ol.source.BingMaps({
  imagerySet:'Road',
  key:'Ak-dzM4wZjSqTlzveKz5u0d4IQ4bRzVI309GxmkgSVr1ewS6iPSrOvOKhA-CJlm3'
})
```

注：Bing 瓦片 API 要求用户注册一个 API 密钥，该密钥将在地图应用程序中使用，示例中的密钥不能再产品中使用。

（1） 保存修改，在浏览器中查看该页面：http://localhost:8000/ol_workshop/map.html：

完整示例代码如下：

```
<!doctype html>
<htmllang="en">
<head>
<linkrel="stylesheet"href="ol3/ol.css"type="text/css">
<style>
#map {
height:256px;
width:512px;
    }
.ol-attributiona {
color:black;
    }
</style>
<script src="ol3/ol.js"type="text/javascript"></script>
<title>OpenLayers 3 example</title>
</head>
<body>
<h1>My Map</h1>
<divid="map"class="map"></div>
<script type="text/javascript">
var map =new ol.Map({
       target:'map',
       layers: [
new ol.layer.Tile({
```

```
        source:new ol.source.BingMaps({
          imagerySet:'Road',
          key:'Ak-dzM4wZjSqTlzveKz5u0d4IQ4bRzVI309GxmkgSVr1ewS6iPSrOvOKhA-CJlm3'
        })
      })
    ],
    view:new ol.View({
      center: ol.proj.transform([-93.27, 44.98], 'EPSG:4326', 'EPSG:3857'),
      zoom:9
    })
  });
</script>
</body>
</html>
```

2.4 矢量图层

矢量图层由 `ol.layer.Vector` 展示，并处理客户端矢量数据的显示。以下将使用最初的 WMS 示例来获取一个世界地图，并在其基础上添加一个带有一些要素的矢量图层。

```
<!doctype html>
<htmllang="en">
<head>
<linkrel="stylesheet"href="ol3/ol.css"type="text/css">
<style>
#map {
height:256px;
width:512px;
    }
</style>
<title>OpenLayers 3 example</title>
<script src="ol3/ol.js"type="text/javascript"></script>
</head>
<body>
<h1>My Map</h1>
<divid="map"></div>
<script type="text/javascript">
var map =new ol.Map({
```

```
      target:'map',
      layers: [
new ol.layer.Tile({
         title:"Global Imagery",
         source:new ol.source.TileWMS({
           url:'http://maps.opengeo.org/geowebcache/service/wms',
           params: {LAYERS:'bluemarble', VERSION:'1.1.1'}
         })
       })
      ],
      view:new ol.View({
       projection:'EPSG:4326',
       center: [0, 0],
       zoom:0,
       maxResolution:0.703125
      })
    });
</script>
</body>
</html>
```

2.4.1 添加矢量图层

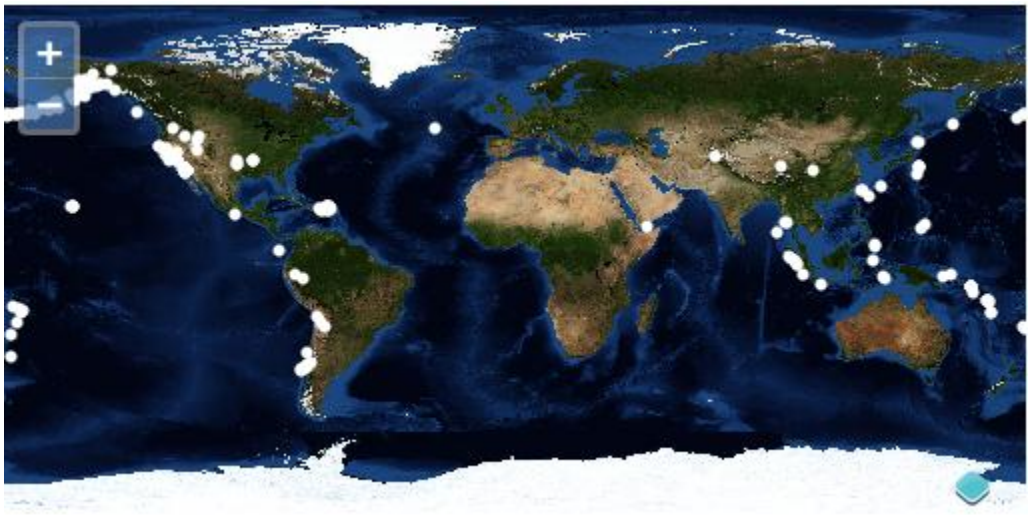
（1）打开 `map.html` 文件，将初始化 WMS 的示例复制其中，保存修改后在浏览器中确定地图正常显示：http://localhost:8000/ol_workshop/map.html。

（2）在地图初始化代码中，找到瓦片图层的加载，在其后添加一下新的图层，以下代码实现请求一组存放在 GeoJSON 中的要素：

```
new ol.layer.Vector({
 title:'Earthquakes',
 source:new ol.source.GeoJSON({
  url:'data/layers/7day-M2.5.json'
 }),
 style:new ol.style.Style({
  image:new ol.style.Circle({
   radius:3,
   fill:new ol.style.Fill({color:'white'})
```

```
  })
 })
})
```

以上示例显示了世界地图，附以白色的圆圈代表地震带。



注：GeoJSON 数据坐标系与地图视图的相同，均为 `EPSG:4326`，因此无需再次设置投影，只有在资源与视图的投影不同的情况下，才需要在资源中明确指定 `projection` 属性来表示要素缓存的投影，这以为着地图视图的投影通常可以被指定。

2.4.2 详细说明

上述示例中，设置图层的标题 `title` 为“Earthquakes”，使用 `ol.source.GeoJSON` 类型的资源 source，该资源指向一个明确的 `url`。

如果你希望要素的样式基于其属性，可以使用一个样式函数替代 `ol.style.Style`，从而配置 `ol.layer.Vector` 的样式。

（1）上述示例中地图上白色的圆圈代表 `ol.layer.Vector` 图层中的 `ol.Feature` 对象，每一个要素都包含 title 和 summary 属性信息。地图中注册一个命名为 forEachFeatureAtPixel 的单击监听事件，并在地图视图下在显示地震信息。

（2） 矢量图层的数据来自于美国地质调查局（USGS）公布的地震资料(http://earthquake.usgs.gov/earthquakes/catalogs/)，找到 OpenLayers 3 支持格式的矢量图层信息保存为文档，将该文档放置在项目的 data 文件夹下，就能在地图中显示该矢量图层。

2.5 矢量影像

在上一章节的示例中，使用了 `ol.layer.Vector` 类，在动态缩放的过程中，要素不断重新渲染（点符号大小保持固定），在矢量图层中，OpenLayers 基于每一动画帧重新渲染资源数据，这使得在视图的分辨率变化后，笔划、点符号、标签持续的渲染。

另一种渲染策略是避免在视图转换的过程中重渲染，并将之前视图的状态下的渲染输出重定位和改变其规模。通过使用包含 `ol.source.ImageVector` 的 `ol.layer.Image` 可以实现以上效果。这种结合，使得当视图没有动态变化时，保存渲染数据的“快照”，在视图转换过程中，重利用这些“快照”。

以下示例使用了包含 `ol.source.ImageVector` 的 `ol.layer.Image` 类，实现分块渲染，使用该类可以仅渲染数据的一小部分，这种结合将适用于包含大量相对静态数据渲染的应用程序。

2.5.1 ol.source.ImageVector

回顾 2.4 矢量图层添加的包含地震数据的地图示例，将此示例改为分块渲染，将矢量图层替换为如下代码：

```
new ol.layer.Image({
 title:'Earthquakes',
 source:new ol.source.ImageVector({
  source:new ol.source.GeoJSON({
   url:'data/layers/7day-M2.5.json'
  }),
  style:new ol.style.Style({
   image:new ol.style.Circle({
   radius:3,
    fill:new ol.style.Fill({color:'white'})
   })
```

```
  })
 })
})
```

通过以上方式，矢量数据由图像描绘，但视觉上仍是要素的形式，实现了性能和质量之间本质的折中。

2.5.2 详细说明

在上述代码中，使用 `ol.layer.Image` 代替 `ol.layer.Vector`，然而，仍可以通过 `ol.source.ImageVector` 连接原始的 `ol.source.GeoJSON` 类型数据，从而使用是矢量数据，这里的样式提供了对 `ol.source.ImageVector` 的配置，而不是直接配置图层。

3 控件与交互

3.1 显示比例尺

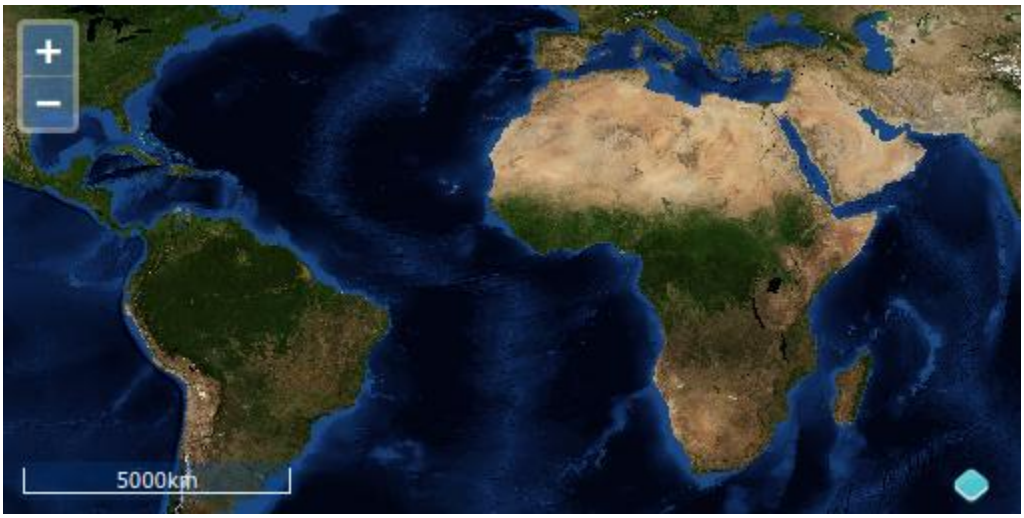
比例尺是显示在地图上的典型窗口小部件，OpenLayers 3 提供了 `ol.control.SaleLine` 来实现。

3.1.1 创建比例尺

在地图配置的范围内，添加如下代码，给地图创建一个新的比例尺控件：

```
controls:ol.control.defaults().extend([
newol.control.ScaleLine()
]),
```

一个默认的比例尺将出现在地图视图的左下角。



3.1.2 移动比例尺控件

如果觉得比例尺控件在图形中看不清，一下提供几种策略来提高比例尺的可见性。在文档的 CSS 中添加一些样式声明，可以包括背景色，填充等，以下代码可作为参考：

```
.ol-scale-line, .ol-scale-line:not([ie8andbelow]) {
  background: black;
  padding: 5px;
}
```

如果地图视图拥挤难耐，为了避免过度拥挤，可以将比例尺控件移到其他位置。实现此功能，需要在标记中创建一个额外元素来存放比例尺控件。

（1） 在 `map.html` 页面的 `<body>` 范围内创建一个新的块级元素，为了更好的指向该元素，设置其 `id` 属性 `scale-line`，代码如下：

```
<divid="scale-line"class="scale-line"></div>
```

将以上代码放置在 `<divid="map"></div>` 后最合理。

（2） 修改比例尺控件的创建代码，使其指向 `scale-line` 元素：

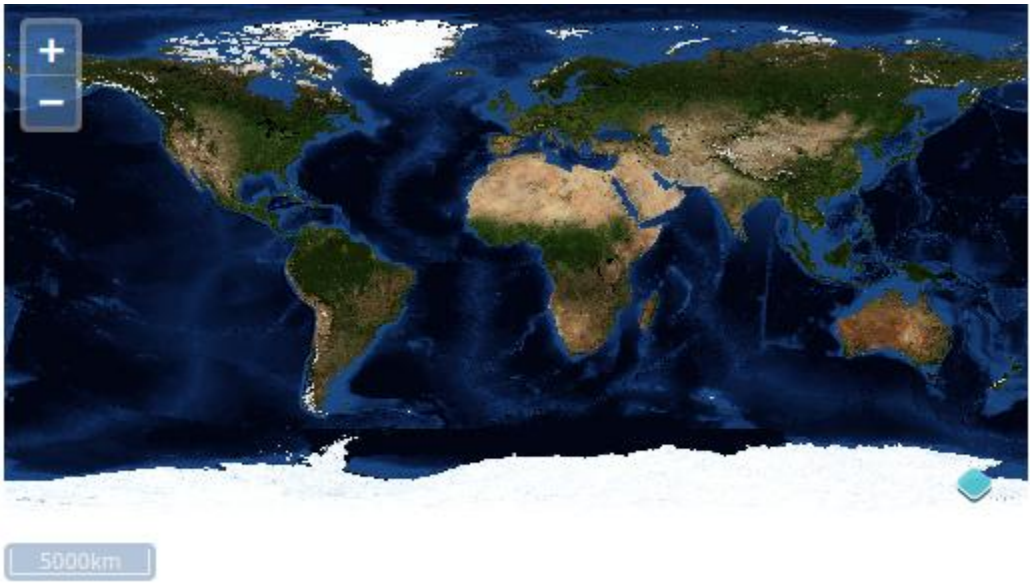
```
controls:ol.control.defaults().extend([
newol.control.ScaleLine({
    className:'ol-scale-line',
```

```
    target:document.getElementById('scale-line')
    })
]),
```

（3） 添加比例尺控件的样式声明：

```
.scale-line {
 position: absolute;
 top: 350px;
}
.ol-scale-line {
 position: relative;
 bottom: 0px;
 left: 0px;
}
```

保存修改，在浏览器中查看该页面：http://localhost:8000/ol_workshop/map.html：



实现比例尺控件在地图视图以外。

注：想要创建自定义控件，可以从 `ol.control.Control` 继承（通过使用 `ol.inherits`）。

3.2 选择要素

矢量数据服务的一大优点是，用户可以和数据交互，在本节示例中，将创建一个矢量图层，用户可以选择和查看要素信息。

之前的示例展示了 `ol.control.Control` 的用法，控件可以在地图上直接显示，或者在文档中添加一个 DOM 元素，`ol.interaction.Interaction` 负责处理用户交互，但是通常没有视觉上的展现。一下示例将展示 `ol.interaction.Select` 的使用，实现与矢量图层上的要素交互。

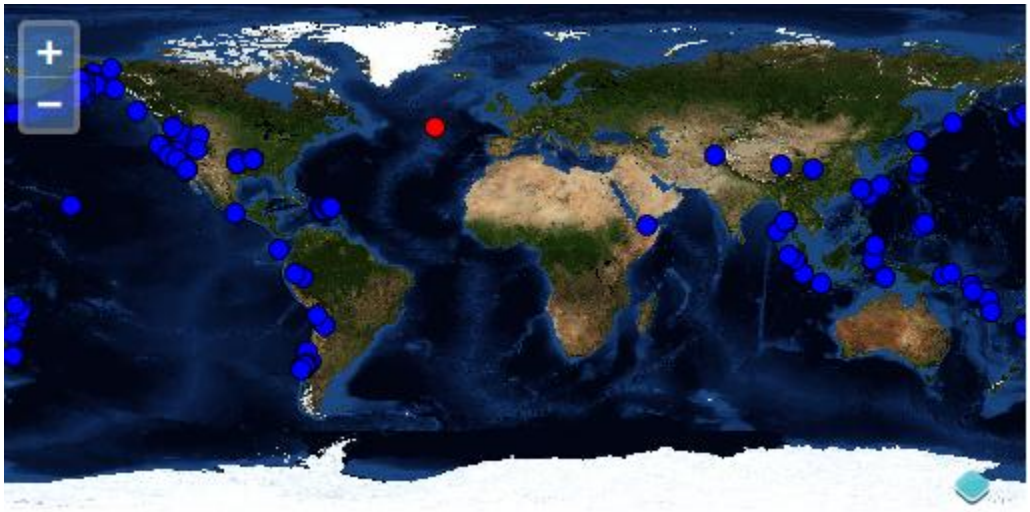
回顾 2.4 矢量图层中矢量图层的创建，在其基础上，添加选择交互工具，完整代码如下：

```
<!doctype html>
<htmllang="en">
<head>
<linkrel="stylesheet"href="ol3/ol.css"type="text/css">
<style>
#map {
height:256px;
width:512px;
   }
</style>
<script src="ol3/ol.js"type="text/javascript"></script>
<title>OpenLayers 3 example</title>
</head>
<body>
<h1>My Map</h1>
<divid="map"></div>
<script type="text/javascript">
var map =new ol.Map({
      interactions: ol.interaction.defaults().extend([
new ol.interaction.Select({
          style:new ol.style.Style({
            image:new ol.style.Circle({
              radius:5,
              fill:new ol.style.Fill({
                color:'#FF0000'
              }),
              stroke:new ol.style.Stroke({
                color:'#000000'
              })
```

```
            })
          })
        })
      ]),
      target:'map',
      layers: [
new ol.layer.Tile({
        title:"Global Imagery",
        source:new ol.source.TileWMS({
          url:'http://maps.opengeo.org/geowebcache/service/wms',
          params: {LAYERS:'bluemarble', VERSION:'1.1.1'}
        })
      }),
new ol.layer.Vector({
        title:'Earthquakes',
        source:new ol.source.GeoJSON({
          url:'data/layers/7day-M2.5.json'
        }),
        style:new ol.style.Style({
          image:new ol.style.Circle({
            radius:5,
            fill:new ol.style.Fill({
              color:'#0000FF'
            }),
            stroke:new ol.style.Stroke({
              color:'#000000'
            })
          })
        })
      }),
      view:new ol.View({
        projection:'EPSG:4326',
        center: [0, 0],
        zoom:1
      })
    });
</script>
</body>
```

```
</html>
```

保存修改，在浏览器中打开：http://localhost:8000/ol_workshop/map.html.使用鼠标点击事件选择地震带，查看要素选择的效果。



3.3 绘制要素

通过使用 `ol.interaction.Draw` 可以绘制新的要素，一个绘制交互工具由矢量资源和几何类型构成。

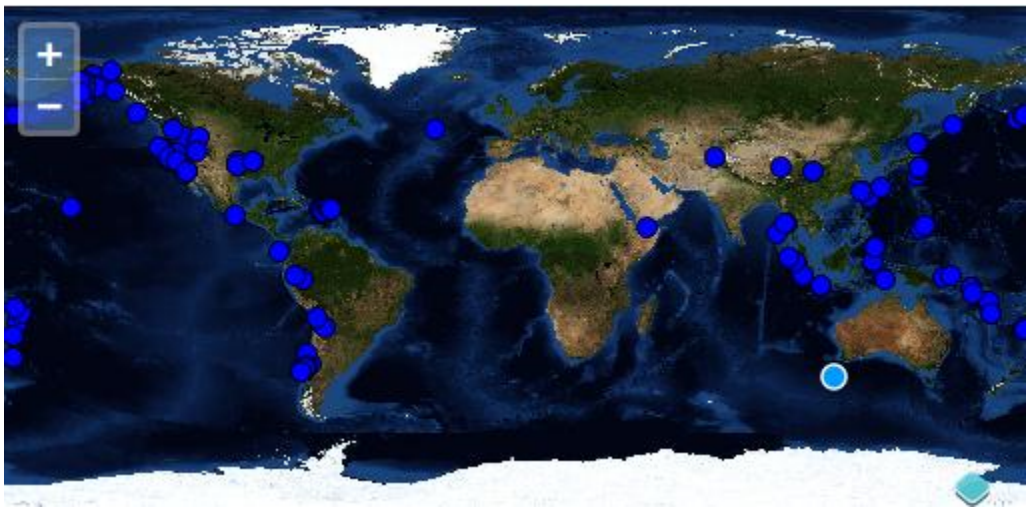
回顾 2.4 矢量图层中矢量图层的创建，在其基础上，添加绘制交互工具，完整代码如下：

```
<!doctype html>
<htmllang="en">
<head>
<linkrel="stylesheet"href="ol3/ol.css"type="text/css">
<style>
#map {
height:256px;
width:512px;
   }
</style>
<script src="ol3/ol.js"type="text/javascript"></script>
<title>OpenLayers 3 example</title>
</head>
```

```
<body>
<h1>My Map</h1>
<divid="map"></div>
<script type="text/javascript">
var source =new ol.source.GeoJSON({
       url:'data/layers/7day-M2.5.json'
     });
var draw =new ol.interaction.Draw({
       source: source,
       type:'Point'
     });
var map =new ol.Map({
       interactions: ol.interaction.defaults().extend([draw]),
       target:'map',
       layers: [
new ol.layer.Tile({
          title:"Global Imagery",
          source:new ol.source.TileWMS({
            url:'http://maps.opengeo.org/geowebcache/service/wms',
            params: {LAYERS:'bluemarble', VERSION:'1.1.1'}
          })
        }),
new ol.layer.Vector({
          title:'Earthquakes',
          source: source,
          style:new ol.style.Style({
            image:new ol.style.Circle({
              radius:5,
              fill:new ol.style.Fill({
                color:'#0000FF'
              }),
              stroke:new ol.style.Stroke({
                color:'#000000'
              })
            })
          })
        })
       ],
       view:new ol.View({
```

```
        projection:'EPSG:4326',
        center: [0, 0],
        zoom:1
      })
    });
</script>
</body>
</html>
```

保存修改，在浏览器中打开：http://localhost:8000/ol_workshop/map.html.点击地图来添加新的要素，实现点几何对象的绘制。



3.4 修改要素

修改要素需要联合使用 `ol.interaction.Select` 和 `ol.interaction.Modify`，它们有一个共同的要素集(ol.Collection)。`ol.interaction.Modify` 对 `ol.interaction.Select` 选择的要素进行修改。

回顾 2.4 矢量图层中矢量图层的创建，在其基础上，添加选择与修改交互工具，完整代码如下：

```
<!doctype html>
<htmllang="en">
<head>
<linkrel="stylesheet"href="ol3/ol.css"type="text/css">
<style>
```

```
#map {
height:256px;
width:512px;
   }
</style>
<script src="ol3/ol.js"type="text/javascript"></script>
<title>OpenLayers 3 example</title>
</head>
<body>
<h1>My Map</h1>
<divid="map"></div>
<script type="text/javascript">
var source =new ol.source.GeoJSON({
       url:'data/layers/7day-M2.5.json'
     });
var style =new ol.style.Style({
       image:new ol.style.Circle({
         radius:7,
           fill:new ol.style.Fill({
           color: [0, 153, 255, 1]
         }),
         stroke:new ol.style.Stroke({
           color: [255, 255, 255, 0.75],
           width:1.5
         })
       }),
       zIndex:100000
     });
var select =new ol.interaction.Select({style: style});
var modify =new ol.interaction.Modify({
       features: select.getFeatures()
     });
var map =new ol.Map({
       interactions: ol.interaction.defaults().extend([select, modify]),
       target:'map',
       layers: [
new ol.layer.Tile({
           title:"Global Imagery",
           source:new ol.source.TileWMS({
```

```
            url:'http://maps.opengeo.org/geowebcache/service/wms',
            params: {LAYERS:'bluemarble', VERSION:'1.1.1'}
          })
        }),
new ol.layer.Vector({
          title:'Earthquakes',
          source: source,
          style:new ol.style.Style({
            image:new ol.style.Circle({
              radius:5,
              fill:new ol.style.Fill({
                color:'#0000FF'
              }),
              stroke:new ol.style.Stroke({
                color:'#000000'
              })
            })
          })
        })
      ],
      view:new ol.View({
        projection:'EPSG:4326',
        center: [0, 0],
        zoom:1
      })
    });
</script>
</body>
</html>
```

保存修改，在浏览器中打开：http://localhost:8000/ol_workshop/map.html.使用鼠标点击事件选择地震带，拖动该点，实现要素修改。

4 矢量样式

4.1 矢量图层格式

基础的 `ol.layer.Vector` 构造函数提供了一个相对固定的图层类型，默认情况下，当创建一个新的矢量图层时，并不知道图层的要素来源，因为这是的 `ol.source.Vector` 内容。接下来将介绍自定义渲染风格，以及矢量数据的基本格式。

4.1.1 ol.format

OpenLayers 3 中的 `ol.format` 类负责解析服务器中代表是两要素的数据，大多数情况下，不会直接使用该类，但是会使用其相应的资源（比如：`ol.source.KML`）。该格式将原始要素数据转变为 `ol.Feature` 对象。

考虑以下两个数据块，它们都表示同一个 `ol.Feature` 对象（西班牙巴塞罗那的一个点）。第一种序列化为 GeoJSON（使用 `ol.format.GeoJSON` 解析），第二种序列化为 KML（使用 `ol.format.KML` 解析）。

（1）GeoJSON Example

```
{
"type":"Feature",
"id":"OpenLayers.Feature.Vector_107",
"properties": {},
"geometry": {
"type":"Point",
"coordinates": [-104.98, 39.76]
   }
}
```

（2）KML Example

```
<?xml version="1.0" encoding="utf-8"?>
<kmlxmlns="http://earth.google.com/kml/2.2">
<Placemark>
<Point>
<coordinates>-104.98,39.76</coordinates>
</Point>
</Placemark>
</kml>
```

4.2 矢量图层样式

设置 HTML 元素样式，使用如下 CSS：

```
.someClass {
   background-color: blue;
   border-width: 1px;
   border-color: olive;
}
```

`.someClass` 是一个选择器（选择所有 `Class` 属性值为`"someClass"`的对象），后面的内容是一组已命名的属性与其对应的值，称之为样式声明。

4.2.1 基础样式

一个矢量图层可以有样式，更明确的说法是，一个矢量图层可以由一个 `ol.style.Style` 对象、一组 `ol.style.Style` 对象或者一个返回值为一组 `ol.style.Style` 对象的方法配置。

（1）由静态样式配置的矢量图层：

```
var layer =new ol.layer.Vector({
  source:new ol.source.Vector(),
  style:new ol.style.Style({
// ...
  })
});
```

（2）由样式方法配置的矢量图层，将所有 `Class` 属性值为`"someClass"`的对象设置为同一样式：

```
var layer =new ol.layer.Vector({
  source:new ol.source.Vector(),
  style:function(feature, resolution) {
if (feature.get('class') ==='someClass') {
// create styles...
return styles;
    }
  },
```

```
});
```

4.2.2 符号化

OpenLayers 3 中的符号化相当于 CSS 中声明的块（下面是关于 ol.style.*的典型示例）。在一个蓝色的背景下用 1 像素宽的线条绘制区要素，将运用的如下两个符号化对象：

```
new ol.style.Style({
  fill:new ol.style.Fill({
    color:'blue'
  }),
  stroke:new ol.style.Stroke({
    color:'olive',
    width:1
  })
});
```

不同的符号化对象使用于不同的几何类型，区可以由线组成，但没有填充，点可以用 `ol.style.Circle` 和 `ol.style.Icon` 进行样式说明。前者呈现圆形的形状，后者使用文件中的图片（比如：png 图像）。下面是用圆形样式的示例：

```
new ol.style.Circle({
  radius:20,
  fill:new ol.style.Fill({
    color:'#ff9900',
    opacity:0.6
  }),
  stroke:new ol.style.Stroke({
    color:'#ffcc00',
    opacity:0.4
  })
});
```

4.2.3 ol.style.Style

一个 `ol.style.Style` 对象有四个关键属性：fill, image, stroke 和 text，外加一个 `zIndex` 属性选项。下面的样式方法将返回一组 `ol.style.Style` 对象。

以下示例实现将 class 属性值为的"someClass"对象中的要素设置为 1 像素宽的蓝色，其余所有要素设置为红色。（最好将样式方法在对象外创建，以便重用，但以下代码为简单起见，将方法写在对象内。）

```
style: (function() {
var someStyle = [new ol.style.Style({
    fill:new ol.style.Fill({
      color:'blue'
    }),
    stroke:new ol.style.Stroke({
      color:'olive',
      width:1
    })
  })];
var otherStyle = [new ol.style.Style({
    fill:new ol.style.Fill({
      color:'red'
    })
  })];
returnfunction(feature, resolution) {
if (feature.get('class') ==="someClass") {
return someStyle;
    } else {
return otherStyle;
    }
  };
}())
```

注：一个要素也有样式配置选项，只需设置 resolution 参数就能实现对个别要素的样式设置（基于分辨率）。

4.2.4 伪类

CSS 允许选择器中存在伪类，这限制了基于上下文的样式声明的应用，使得样式声明不容器在选择器中表示，比如鼠标位置，附近的元素或者浏览器历史。在 OpenLayers 3 中，有一种类似的概念：在 ol.interaction.Select 中添加样式配置选项，示例代码如下：

```
var select =new ol.interaction.Select({
  style:new ol.style.Style({
    fill:new ol.style.Fill({
      color:'rgba(255,255,255,0.5)'
    })
  })
});
```

了解了基础的样式，接下来将介绍矢量图层的样式设置。

4.3 设置矢量图层的样式

本节示例将展示矢量图层中的建筑平面图，完整示例代码如下：

```
<!doctype html>
<htmllang="en">
<head>
<linkrel="stylesheet"href="ol3/ol.css"type="text/css">
<style>
#map {
background-color:gray;
height:256px;
width:512px;
    }
</style>
<title>OpenLayers 3 example</title>
<script src="ol3/ol.js"type="text/javascript"></script>
</head>
<body>
<h1>My Map</h1>
<divid="map"></div>
<script type="text/javascript">
var map =new ol.Map({
        target:'map',
        layers: [
new ol.layer.Vector({
            title:'Buildings',
            source:new ol.source.KML({
              url:'data/layers/buildings.kml',
```

```
          extractStyles:false
        }),
        style:new ol.style.Style({
          stroke:new ol.style.Stroke({color:'red', width:2})
        })
      })
    ],
    view:new ol.View({
      projection:'EPSG:4326',
      center: [-122.791859392, 42.3099154789],
      zoom:16
    })
  });
</script>
</body>
</html>
```

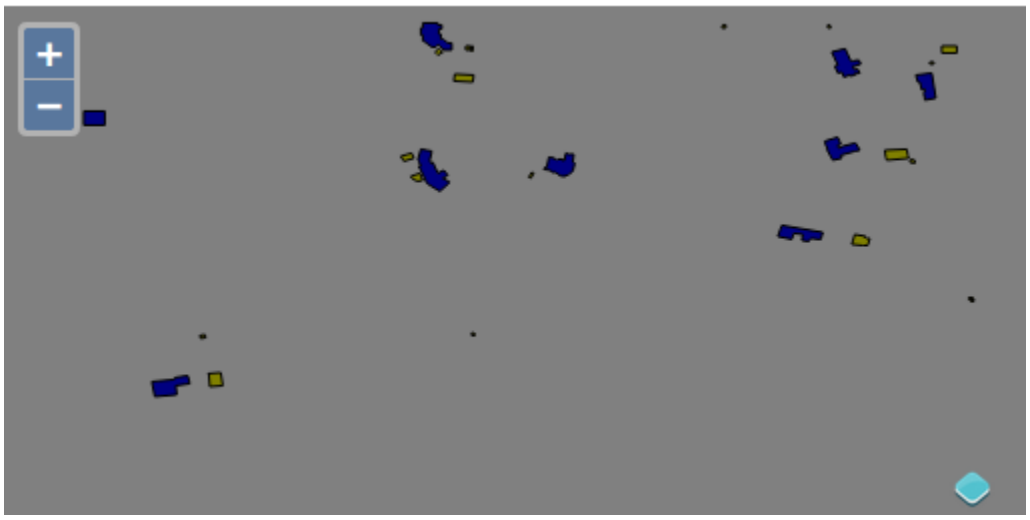
在浏览器中打开 `map.html` 文件：http://localhost:8000/ol_workshop/map.html，将看到带有红色轮廓的建筑群。

创建一个样式方法，根据面积的大小，使建筑显示为不同的颜色，将一下代码，将 Buildings 图层的样式配置选项替换为如下代码：

```
style: (function() {
var defaultStyle = [new ol.style.Style({
   fill:new ol.style.Fill({color:'navy'}),
   stroke:new ol.style.Stroke({color:'black', width:1})
 })];
var ruleStyle = [new ol.style.Style({
   fill:new ol.style.Fill({color:'olive'}),
   stroke:new ol.style.Stroke({color:'black', width:1})
 })];
returnfunction(feature, resolution) {
if (feature.get('shape_area') <3000) {
return ruleStyle;
   } else {
return defaultStyle;
   }
```

```
  };
})()
```

保存修改，在浏览器中查看：http://localhost:8000/ol_workshop/map.html。



最后给建筑添加标签，为了简单化，我们使用黑色轮廓的样式，并只使用一种标签。

```
style: (function() {
var stroke =new ol.style.Stroke({
    color:'black'
  });
var textStroke =new ol.style.Stroke({
    color:'#fff',
    width:3
  });
var textFill =new ol.style.Fill({
    color:'#000'
  });
returnfunction(feature, resolution) {
return [new ol.style.Style({
      stroke: stroke,
      text:new ol.style.Text({
        font:'12px Calibri,sans-serif',
        text: feature.get('key'),
        fill: textFill,
        stroke: textStroke
```

```
    })
  })];
 };
})()
```

保存修改，在浏览器中查看：http://localhost:8000/ol_workshop/map.html。