论文领域地图版块的逐层加载与前端展示设计

杜鹏翔
517030910372

2020.6

摘　要　论文网络地图往往有较高的数据量，当网络图中节点数目极大时，如果使用矢量图来进行缩放展示会导致效果很差，严重影响用户体验。本文主要讲解了如何使用多层位图分层加载的方法来解决此问题，并借助Leaflet实现了能对约两百万节点的论文网络快速展示的网页。

关键词　论文网络，切片图，分层加载，网页制作

1　简介

论文领域的网络展示往往涉及到超大规模的节点数和边数，如果使用Gephi这类工具制作一张超大型的矢量图来进行缩放展示，会存在加载耗时较长、操作反应迟缓等缺点，影响用户体验。

联想到各类地图APP展示时不同放大层级展示的实际上不是同一张图片，而是针对每一个放大层级分别制作一系列图片，在加载时只需要加载一部分用到的图而不是全部的数据。因此将同样的思路用到超大规模论文网络的展示中，我们不是将全部数据一次性制作成一张大图，而是首先根据节点大小划分多个放大层级，在每个层级中采用切片绘图的方法绘制多个切片图。在展示时，只需要加载相应放大层数和位置的少数图片即可。从而在加载速度上有较大提升，并且操作时的延迟感明显下降。

2　方法和原理

本文使用的是基于分层加载法的超大规模网络展示方法。该方法主要分为三步：

1. 使用基于网格划分的重叠去除算法对原始网络布局中节点重叠现象进行去除；

   由于原始网络布局中存在大量的节点重叠和接触现象，所以首先应当进行重叠去除处理。同时因为网络规模巨大，使用普通的重叠去除算法效果较差，所以使用先切片后对每张切片单独重叠去除处理。通过微调节点之间的位置，使相互接触或者相互交叠的节点互相不再接触，并保留原网络结构。

2. 使用切片绘图的方法对超大规模网络进行切片化绘制；

   切片绘图实际上就是先将原始网络图按照网格化划分，将每一个网格制作为一张切片图。

3. 使用分层加载技术对绘制完成的切片进行多层缩放展示。

   这一部分是本文的主要部分，也是我工作的主要内容。其主要思想是根据上一步得到的不同放大层级的多组切片图，在我们逐渐放大时在页面内分别展示相应放大层数和位置的多张切片图。由于不同层的切片都有绘制，所以可以保证放大后分辨率保持不变。另一方面由于我们只需要加载页面内能看到的部分图片而非全部图片，从而保证了页面加载速度。

   通过HTML+CSS+Javascript的方式实现，通过调用开源的地图Javascript库Leaflet，我们可以实现对于切片的分层加载。

3 实验部分

3.1 原始数据

因为本文主的主要目的是实现论文领域地图版块的逐层加载与前端展示设计，所以使用的原始数据“Nature杂志引用关系网络”是已经经过重叠去除算法处理之后的。由两部分组成，一是一个包含所有节点信息和边信息的JSON文件，另一个是包含文章题目与节点ID对应关系的csv文件。但是此csv文件中所使用的ID为六进制数且并不是第一个文件中的ID数字的直接转换。而是通过另一个JSON文件来记录其一一对应关系。

第一个文件的内容格式如下图所示，是一个由很多个字典组成的数组。

{'label': 'Biology', 'x': -15.654354095458984, 'y': 433.5774841308594, 'id': '954692', 'attributes': {'Class': 'Biology'}, 'color': 'rgb(255,56,175)', 'size': 0.10579727590084076}
{'label': 'Biology', 'x': -509.22784423828125, 'y': -97.87203216552734, 'id': '1440948', 'attributes': {'Class': 'Biology'}, 'color': 'rgb(255,56,175)', 'size': 0.1077297031879425}
{'label': 'Biology', 'x': -337.72930908203125, 'y': -327.26373291015625, 'id': '107953', 'attributes': {'Class': 'Biology'}, 'color': 'rgb(255,56,175)', 'size': 0.10193242877721786}
{'label': 'Biology', 'x': -180.80601501464844, 'y': -427.2701416015625, 'id': '1635566', 'attributes': {'Class': 'Biology'}, 'color': 'rgb(255,56,175)', 'size': 0.10193242877721786}
{'label': 'Geology', 'x': 353.40570068359375, 'y': 452.5256652832031, 'id': '317632', 'attributes': {'Class': 'Geology'}, 'color': 'rgb(100,87,255)', 'size': 0.111594557762146}
{'label': 'Biology', 'x': 1500.531005859375, 'y': -672.7166137695312, 'id': '694265', 'attributes': {'Class': 'Biology'}, 'color': 'rgb(255,56,175)', 'size': 0.10020291805267334}
{'label': 'Biology', 'x': 503.1953430175781, 'y': -238.0445098876953, 'id': '1208109', 'attributes': {'Class': 'Biology'}, 'color': 'rgb(255,56,175)', 'size': 0.10020291805267334}
{'label': 'Medicine', 'x': -185.58747863769531, 'y': -495.4329528808594, 'id': '261240', 'attributes': {'Class': 'Medicine'}, 'color': 'rgb(0,245,17)', 'size': 0.10020291805267334}
{'label': 'Mathematics', 'x': 802.1954956054688, 'y': 445.3265380859375, 'id': '1237738', 'attributes': {'Class': 'Mathematics'}, 'color': 'rgb(255,163,207)', 'size': 0.10020291805267334}
{'label': 'Biology', 'x': 133.3516082763672, 'y': -162.6007843017578, 'id': '439182', 'attributes': {'Class': 'Biology'}, 'color': 'rgb(255,56,175)', 'size': 0.10020291805267334}

{'source': '382638', 'target': '752184', 'id': '2325537', 'attributes': {}, 'color': 'rgb(255,154,92)', 'size': 1.0}
{'source': '1490929', 'target': '789361', 'id': '589261', 'attributes': {}, 'color': 'rgb(127,150,96)', 'size': 1.0}
{'source': '180365', 'target': '724462', 'id': '1374209', 'attributes': {}, 'color': 'rgb(255,154,92)', 'size': 1.0}
{'source': '1173211', 'target': '1209343', 'id': '313074', 'attributes': {}, 'color': 'rgb(255,56,175)', 'size': 1.0}
{'source': '1000766', 'target': '693025', 'id': '831761', 'attributes': {}, 'color': 'rgb(130,152,215)', 'size': 1.0}
{'source': '498689', 'target': '837169', 'id': '2013490', 'attributes': {}, 'color': 'rgb(127,150,96)', 'size': 1.0}
{'source': '1273413', 'target': '358354', 'id': '2799947', 'attributes': {}, 'color': 'rgb(100,87,255)', 'size': 1.0}

图 1: nature_3.json.json

第一张图展示了节点信息的结构，每个节点组成一个字典，包含其标签、位置、ID、尺寸等信息，第二张图则为节点的信息结构，包含每条边的起始节点和终止节点的ID等信息。其中这些节点的位置信息是已经经过重叠去除算法处理了的，我们可以直接用这些数据进行第二步的切片图绘制。

3.2 切片图绘制

3.2.1 节点划分

在放大的层数比较低时，虽然视野的范围更大，但是一般也不需要更为详细的展示，而是在放大的比较大时再展示的更为详细。也就是说，我们前几层的图片可以仅展示部分节点和边。

这里我们采取的策略是先遍历所有节点，将度高于50，也就是比较关键的节点选出来，保证他们都能一直被展示出来。然后对于其他的节点，我们按照指数增长的速度把剩余节点分到10个json文件中，$0-x$这x个json文件代表放大到第x层的所有节点。例如，“0.json”中有20000个节点的ID，“1.json”中有40000个节点的ID，而这两个文件记录我们放大到第二层时所有的节点。

```
for node in tqdm(G.node):
    if G.degree(node) > 50:
        NodesList.add(int(D[node], 16))
        Nodes.remove(node)
for i in tqdm(range(10)):
    while True:
        if len(NodesList) < (2**(i + 1))*10000:
            NodesList.add(int(D[Nodes.pop()], 16))
        else:
            break
    NodeList_json_str = json.dumps(list(NodesList))
    with open(project_name + "//json//" + str(i) + ".json", "w") as file:
        json.dump(NodeList_json_str, file)
```

3.2.2 切片绘制

根据上一步所分得的十个JSON文件，我们可以分别绘制各个放大层级的切片图，其中切片图的数量与放大层级的关系为$n = 4^{level}$.这一步所占用的内存空间会很大，尤其是到了后面几层的时候，会涉及到十万数量级图片信息的使用。而且在绘制时，会涉及到多个JSON文件的使用，这其中也会包含部分无用信息。因此我不单单把后面几层分开各自绘制，还使用一个中间过程存储所用到的信息，把绘制的函数单独拿出来利用这些信息而非全部原始信息来绘制图片。同时，借助进程池来加速这个过程。

3.3 网页设计与制作

网页的制作是通过HTML+CSS+Javascript的方式实现，主要使用的库是Leaflet。Leaflet是一个开源的并且对于移动设备友好的交互式地图JavaScript库。

实现的论文领域地图板块的基本功能有以下几种。

- 加载切片图
- 缩放
- 拖拽
- 文章搜索
- 其他小功能

我们首先使用Leaflet的map函数初始化一个地图。

```
var map = L.map('image-map', {
    minZoom: 0,
    maxZoom: 9,
    zoom: 0,
});
```

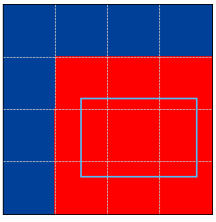
这里的‘image-map’便是我们在html中声明的一个div。通过这段代码我们便可以在网页中获得一个初始缩放水平为0，最大缩放水平为9的原始地图。这时地图并不会展示任何东西，所以我们需要靠下面的步骤来完善这个地图。另外，在我们初始化地图之后，实际上相当于生成了一个有x轴y轴的坐标系。这个坐标系只会随缩放拖拽等操作做出相应的变化，但是点的坐标不会发生变化。比如说我鼠标拖着某个点移动地图，但这个点的坐标并不会变，而是相当于整个坐标系在做平移。这是能够保证我们以下功能实现的关键所在。

3.3.1 图片加载

通过查看Leaflet的官方文档，我们了解到加载一张图片的方式是使用ImageOverlay，用于在地图的特定边界上加载和显示单个图像。因为有些时候要展示多张图片，所以又使用LayerGroup将多个图片Layer合并将其作为一个层处理。

最开始，我制作的只是在某一放大层级，将改层的全部图片一股脑加载出来。我们发现这在前面两三层还算可以，但是放大到后面的时候，由于图片总数指数式上升，加载耗时也会指数式上升。这并不符合我们快速加载的要求。所以便想到像真正的地图加载方式那样，我们只加载在页面显示范围之内的图片。这一方法可以由下图展示的那样。

如图所示是在放大到第二层，图片总数为16时的情况。图中蓝色的方框表示我们页面内能看到的部分，红色的放个代表我们需要加载的图片，而蓝色的图片则为不需要加载的图片。这种加载方式在前几层速度并不会差别多少，但在放大层数较高时，由于最多只需要加载九张图片，可以保证图片加载的高效性。



3.3.2 缩放时的图片切换

此类展示方法精髓就在于我们放大图像时实际上不是对原图像的放大，而是展示另外几张原本制作时就比较大的图像。所以这里就涉及到了图片缩放时的图片切换问题。

最开始我想的是对鼠标的滚轮事件进行监听，我尝试了Leaflet和JavaScript两种对鼠标滚轮事件的监听，但是前者没有返回监听信息，后者返回的鼠标位置是相对于整个网页的，而我想要的是相对于地图坐标系的。所以这两种方法都不行。

后来我才意识到我们并不需要这些操作时的鼠标位置信息，我们只需要屏幕中央的点相对于地图坐标系的坐标就可以了。而Leaflet的$map.getCenter()$可以满足这一要求。

我们只需要在缩放事件结束时，通过中心坐标和当前放大层数，加载出我们所需要的图片。但是如果只加载图片会导致在同一位置上多张图片的堆叠，数量一多就会导致页面变得非常卡顿。所以我们还需要将在之前层加载的图片删除。

一开始我使用的方法是先将全部图片清除，再根据位置加载新的图片。这种方式非常简单，但也有一点不好处在于旧的图片清除与新的图片加载完成之前不可避免的会有时间间隔，这会页面就会闪一下，影响用户体验。所以后来改进为先加载新的图片，经过短暂的延迟后再将之前的图片删除。由于使用的是同一个LayerGroup，这会变得比较复杂，稍有不慎便删错图片。因为0、1层分别是1、4张图，高层级在移动到边缘时也可能只需要六张，角落则为四张。这些情况都需要我们考虑。

这里主要用到了两个自定义的函数，一个是setPics用来加载图片，一个是delPics用来删除图片。

```
var imgs = new L.layerGroup();
function setPics(level,center_x,center_y){
  for (var i = -1;i < 2;i++){
    for (var j = -1;j < 2;j++){
      if (x+i>=0 && x+i<n && y+j>=0 && y+j<n) {
        var southWest1 = map.unproject([(x+i)*wb, (y+j)*wb],0);
        var northEast1 = map.unproject([(x+1+i)*wb,(y+1+j)*wb], 0);
        var bounds1 = new L.LatLngBounds(southWest1, northEast1);
        L.imageOverlay('img/'+level+'/'+(x+i)+'/'+(y+j)+'.png', bounds1).addTo(imgs
            );
      }
    }
  }
  imgs.addTo(map);
};
function delPics(level,cur_level,center_x,center_y){
  for (var im = 0; im < n; im++){
                imgs.removeLayer(x0[im]);
        }
};
```

这里只展示了两个函数的关键部分。

3.3.3 拖拽功能

拖拽功能的实现原理基本上与缩放相似，我们只需要通过对拖拽事件进行监听，在拖拽结束时通过中心位置坐标来加载新的图片，并删除之前的图片。

3.3.4 几个小功能

1. 地图标记.

   地图标记是Leaflet实现的一个功能，我们可以在监听到鼠标左键点击时在所点击的位置生成一个marker并添加到地图中展示。

2. 小地图.

   MiniMap是一个很常用的Leaflet小地图插件。其内部是借助tileLayer实现的地图。可以实现与大地图同步的拖拽或缩放，也可以借助小地图来对大地图进行操作。

3.3.5 搜索功能

搜索功能的最终版本已经可以通过文章的题目进行搜索，可以是完整的标题也可以是一部分。

其中搜索框是通过vue.js实现的，而搜索则是通过JavaScript的正则表达式匹配实现。由于原始数据过大，所以只选取了一部分文章，通过原始数据中各种对应关系，生成了一个记录文章题目和坐标对应关系的JSON文件。这种方法比较粗糙，以后可能会考虑使用数据库等方式来实现模糊搜索等功能。

```
var search_by_title = new Vue({
    el: '#search',
    data:{
      message: ''
    },
    methods: {
        fly:function(event){
            var search_result = [];
            var reg = new RegExp(this.message);
            for (let title in pos) {
                if (reg.test(title)) {
                    var x = -1 * pos[title][1] * 960;
                    var y = pos[title][0]*960;
                    search_result.push([[x,y],title]);}
            }
            switch(search_result.length){
                case 0:
                    alert("Not found!");
                    break;
                case 1:
                    map.flyTo(search_result[0][0],7);
                    L.marker(search_result[0][0]).addTo(map);
                    break;
                default:
                    var t = [];
                    for (let v of search_result){t.push(v[1]);}
                    alert("Find may results:\n"+t.join('\n\n'));
            }
        }
      }
    })
```

3.4 TileLayer

在完成上述地图的制作之后，在我查阅相关资料时发现我之前忽视的一种实现方式。Leaflet官方文档中就有这么一个类，是专门用来展示这种切片图组成的分层地图的。他的初始化也很简单。简单来讲就是下面这行代码。

```
L.tileLayer('{z}/{x}/{y}.png').addTo(map);
```

TileLayer就是专门用来加载切片地图的，可惜我之前并没有注意到这个。所以我在所实现的网页的下面实现了以此类为基础的第二个地图。通过操作可以感受到，这类地图相较于我之前实现的第一个地图会更加顺滑、流畅，并且有更好的鲁棒性。

4 总结

在本学期的移动互联网课程大作业中，选择这个课题的目的是对网页制作比较感兴趣。在本学期之前的寒假中恰好自己简单的学习了django、vuejs等和网页比较相关的知识，但并没有做几个网页。所以就自己选择了这个题目。但是Leaflet对我来说是个从没听说过的东西，所以就自己搜索相关资料，查看Leaflet的官方文档等，一步步自己的摸索。从最开始的单张图片加载与四角切换、到多张图片的加载，发现问题、思考解决方法和完善各种功能的这个过程还是比较有趣的。虽然最后制作的第一张地图相对于使用TileLayer实现的地图显得有些笨拙，但自己在这个过程中还是收货颇多的。最后感谢老师和助教们的指导!

本文所涉及的代码已上传至Github。`https://github.com/Dpxx/dpxx.github.io`

网页展示见 `https://dpxx.github.io`