



WEBGIS小智团队出品

# WebGIS 快速开发教程

Quick development tutorial of web GIS

5

released in Dec 2023

## 目录

开场白 .....	5
如何阅读本书 .....	7
第一章：预备知识 .....	8
第 1 节：互联网知识铺垫 .....	8
第 2 节：前端基础 .....	11
第 3 节：图形学基础 .....	29
第 4 节：地理学基础 .....	33
第二章：GIS 数据结构基础 .....	36
第 1 节：GIS 数据介绍及分类 .....	36
第 2 节：栅格数据 .....	37
第 3 节：矢量数据 .....	38
第 4 节：三维数据 .....	46
第三章：地图学知识基础 .....	48
第 1 节：地图核心要素 .....	48
第 2 节：坐标系与投影 .....	50
第 3 节：常用坐标系及转换 .....	52
第四章：底层渲染原理 .....	55
第 1 节：Dom 渲染方式 .....	55
第 2 节：canvas 渲染方式 .....	58
第 3 节：图层渲染原理 .....	61
第 4 节：框架与技术选型 .....	63
第五章：Leaflet 精讲 .....	66
第 1 节：基础入门 .....	66
第 2 节：基础底图 .....	73
第 3 节：矢量图层操作 .....	77
第 4 节：marker 与 dom .....	84
第 5 节：栅格图层操作 .....	89
第 6 节：交互及事件监听 .....	91
第六章：Mapbox 精讲 .....	94
第 1 节：基础入门 .....	94
第 2 节：基础底图 .....	106
第 3 节：矢量图层操作 .....	112
第 4 节：marker 与 dom .....	120
第 5 节：交互及事件监听 .....	125
第 6 节：专题地图制作 .....	131
第 7 节：图形绘制与测量 .....	142
第七章：Cesium 精讲 .....	144
第 1 节：基础入门 .....	144
第 2 节：各类图层加载 .....	152
第 3 节：相机教程 .....	156
第 4 节：模型与 3d tiles .....	161

第 5 节：交互及事件 .....	165
第 6 节：WebGL 与着色器知识 .....	170
第八章：OpenLayers 精讲 .....	174
第 1 节：基础入门 .....	174
第 2 节：基础底图加载 .....	179
第 3 节：矢量图层加载 .....	184
第 4 节：栅格图层的加载 .....	194
第 5 节：marker 与 dom .....	196
第 6 节：交互与事件 .....	202
第 7 节 图形绘制与测量 .....	208
第九章：框架编程思想 .....	208
第 1 节：抽象框架知识 .....	208
第 2 节：掌握开发技巧 .....	210
第 3 节：矢量样式美化 .....	210
第 4 节：掩膜技巧 .....	212
第 5 节：地图定位技巧 .....	217
第十章：GIS 软件及中间件 .....	219
第 1 节：常用软件介绍 .....	219
第 2 节：常用中间件介绍 .....	223
第十一章：GIS 服务端构建 .....	225
第 1 节：开源地图服务 .....	225
第 2 节：GeoServer 学习 .....	229
第 3 节：构建自己的服务端 .....	241
第十二章：GIS 数据库基础 .....	252
第 1 节：postgresql 基础 .....	252
第 2 节：postGIS 基础 .....	259
第 3 节：空间查询 .....	260
第 4 节：空间分析 .....	262
第十三章：GIS 架构设计 .....	264
第 1 节：GIS 组件化 .....	264
第 2 节：嵌入式 GIS 架构 .....	265
第 3 节：个性化解决方案 .....	266
第 4 节：构建低代码平台 .....	267
项目实战案例 .....	268
实战案例一 .....	269
1. 项目背景及需求 .....	269
2. app 设计思路 .....	269
3. 实现过程 .....	270
4. 思考与延伸 .....	274
实战案例二 .....	275
1. 项目背景及需求 .....	275
2. app 设计思路 .....	275
3. 实现过程 .....	276
4. 思考与延伸 .....	277

实战案例三.....	279
1. 项目背景及需求 .....	279
2. app 设计思路 .....	280
3. 实现过程 .....	280
4. 思考与延伸 .....	284
常见问题.....	285
结语 .....	286

# 开场白

本书为想学习 WebGIS 的同学量身打造，从一个小白的角度带着大家一步一步走入 WebGIS 开发的大门。本书适用于未毕业的，一点开发经验都没有的学生，也适用于从其他开发行业转到 WebGIS 行业的有开发经验的朋友。本书的目的和宗旨是带着大家快速的入门并且学会 WebGIS 的开发，重点关注“快速开发”四个字。学习的过程中同时掌握一些基础的理论知识，为以后的深入研发做出铺垫。这本书不会跟大家长篇大论的去讲述很多很深奥的理论，所以不用担心会很难而学不会。这本书一共分为十二个章节，从预备知识到基础理论再到基础开发再到深入学习，最后还附带三个实战案例。另外。如果各位在读完这本书的时候有任何的错误性的问题，你可以在我的博客或者短视频账号（B 站、CSDN、等相关媒体平台搜索 **WebGIS 小智** 即可）私信或者留言，我将亲解决你的问题。

现如今的互联网行业内卷严重，传统的前端、后端不是面临着被优化就是面临着降薪调岗，更直接的有人还会面临失业甚至无业的风险，因此大家也都希望自己有着核心竞争力，有着不被企业优化的自信，有不可替代性。这本书就是解决大家这个问题。毕竟现如今 WebGIS 开发的从业人员还比较少。市场还比较有竞争力，希望大家都能够具有一些独特的，不可被替代的本领。

本书的语言比较直白，因为作者本人不喜欢那些高大上的忽悠人的概念。简简单单能让大家把知识学会了就好了。本书的内容包括书籍本体（电子书 pdf 版本）和书籍配套代码文件夹，文件夹中保存的是本书所有的代码案例。本套系列书籍实行“一次购买，永久更新”的方案。即不管读者是从哪个版本购买的，只要购买一次，后续发行的版本都可以免费更新。更新的方式也很简单，只需要联系到作者本人，验证您购买过书籍之后，将会分享给您最新的书籍链接。作者本人在写这本书的时候真是呕心沥血，没日没夜希望大家尊重一下作者劳

动成果，不要在网上或者线下随意传播。在这里拜托大家了！

# 如何阅读本书

本书共 13 个章节。从最基础的铺垫知识到数据基础，地图学基础，渲染原理，再到具体的 WebGIS 前端框架教学，再到 GIS 软件和中间件，再到服务端，数据库，架构等。提供 WebGIS 开发“一条龙”服务。我们围绕“快速学会开发”这个目标，尽可能的让大家在最短的时间内能够上手，拥有实战开发的能力。因此在阅读的过程中你可能会觉得知识讲的很简单，很少的篇幅就能够讲完一个知识，其实并不是笔者不想深入讲解，而是过深的讲述会耽误大家的学习进程，特别是着急做项目，只想完成领导交代的任务的这类人，这样做会偏离宗旨，因此我们就以开发过程中最常见的问题和功能需求为核心，教会大家如何开发，尽可能多的举一反三实现一些思考，提高变换能力。

在阅读本书的过程中，尤其是涉及到具体的功能点实现的时候，我还是建议大家一边阅读一边动手敲一些代码。古话说的好“眼过千遍不如手过一遍”。眼睛看会了那不是会了，只有动手敲出来结果，并且是脱离书中讲解之后能自己敲出来，那才叫真正的学会了。建议大家在阅读到关键部分的时候放慢速度，停下来一边阅读一边思考，一边动手敲代码，这样会大大加快学习效率。本书讲解的绝大多数涉及到代码的地方都有源码文件夹，会在各位购买到此书的同时开放给大家。各位在编写代码的过程中如果遇到了问题，可以参考源码对比一下问题所在。

在阅读本书的过程中，需要引起大家注意的名词、概念、理论、说明、以及结论会用 **紫色** 标注出来，比较重要的概念会用 **加粗体** 展示，非常重要的概念会用 **紫色加粗字体** 展示。

# 第一章：预备知识

## 第 1 节：互联网知识铺垫

在入门 WebGIS 之前，需要各位先掌握一些互联网的基础知识，尤其是还未毕业，没有工作经验的学生们（[已经有从业经验的小伙伴你可以跳过前 2 节，直接从图形学基础开始](#)），这一点尤为重要。只有先从大体上宏观上认识了整个互联网行业的底层逻辑才能在后面的道路上有方向、有目标、同时也能够更好的理解每一个细微的知识点。因此我们先来聊聊互联网的一些基础知识。

不知道大家是否思考过这样一个场景，我们在使用一些软件的时候，例如美团，支付宝等软件，这些软件的运行逻辑是怎样的？我们可能平时只关心怎么用，可能也没思考过这个软件它从研发到投入使用经历了怎样的一个过程？

首先一个软件的终端肯定是用户，是老百姓，是我们大家，因此我们接触到的这个软件的部分通常称为“**客户端**”。对于一个网站来讲也是如此。凡是客户接触到的部分都称作是“**客户端**”或者是“**前端**”。与之相对应的还有一个终端我们叫做“**服务端**”，在程序员行业也称作是“**后端**”。前端/客户端的职责很明显，就是与客户（用户）直接打交道，而后端/服务端的职责通常是整个客户群体所产生的数据的维护。举个例子，你使用美团点餐之前是首先要注册一个账号和密码，然后你下完一单以后这个订单信息还能作为“历史记录”数据被你一直能看到，所以这部分数据存储在哪里？哪里给你提供这一部分数据？哪里来验证你输入的手机号和密码是不是正确的？没错，就是我们的后端/服务端。

了解清楚前端后端的概念以后，我们再来说一说具体的代码开发。实际上在企业中，有这么两个岗位就叫做前端和后端，前端开发人员的职责就是负责开发和用户产生交互的页面，以及交互逻辑。后端人员就是负责给这个应用提供数据和服务。保证在用户使用的时候能够存储、查询、编辑、删除用户的信息。前端开发页面使用的编程语言包括但不限于：**HTML、CSS、javaScript**。我们称为“**前端三剑客**”。这三种语言配合就能够开发我们的前端/客户端的页面，就包括你看到的所有网页的样式，页面的内容，以及一部分手机 app 的页面还包括我们的小程序页面都是前端工作者利用这三剑客开发的。后端的开发语言有很多比较主流的有 **java、python、c++、c#、node.js、go** 等。这些语言各自有着不同的特点，有着不同的适用场景。要注意就像王者荣耀里的英雄一样，没有强弱之分，只是各自的

职责和适用场景不同，我们在不同的开发场景中就会选择不同的语言。所以语言没有好坏之分，不要单纯的去比较语言的好坏。顺便说个小事情，`javaScript` 和 `java` 语言就像是周杰与周杰伦，老婆与老婆饼，没有任何的关系。一个是前端的开发语言，另一个是后端的语言。毫无关系。

我们的 WebGIS 开发也分为前端和后端。一个 WebGIS 应用也是需要一部分人来开发用户界面，一部分人来管理数据服务。但是由于这个行业实在是从业人员比较少，因此大多数情况下都需要各位既会一些前端，又得会一些后端。当然这也正是这个行业工资普遍不低的原因。我们的 WebGIS 是建立在传统的互联网架构基础之上的。换句话说，传统的互联网应用怎么构建，传统的网站怎么开发，我们就怎么开发，只不过我们的核心逻辑和关注点在 GIS 上面。计算机和网络这样的硬件逻辑一天不变，我们的软件逻辑也不会变。

下面是一副互联网基础架构的逻辑图。我们一起来解读一下：

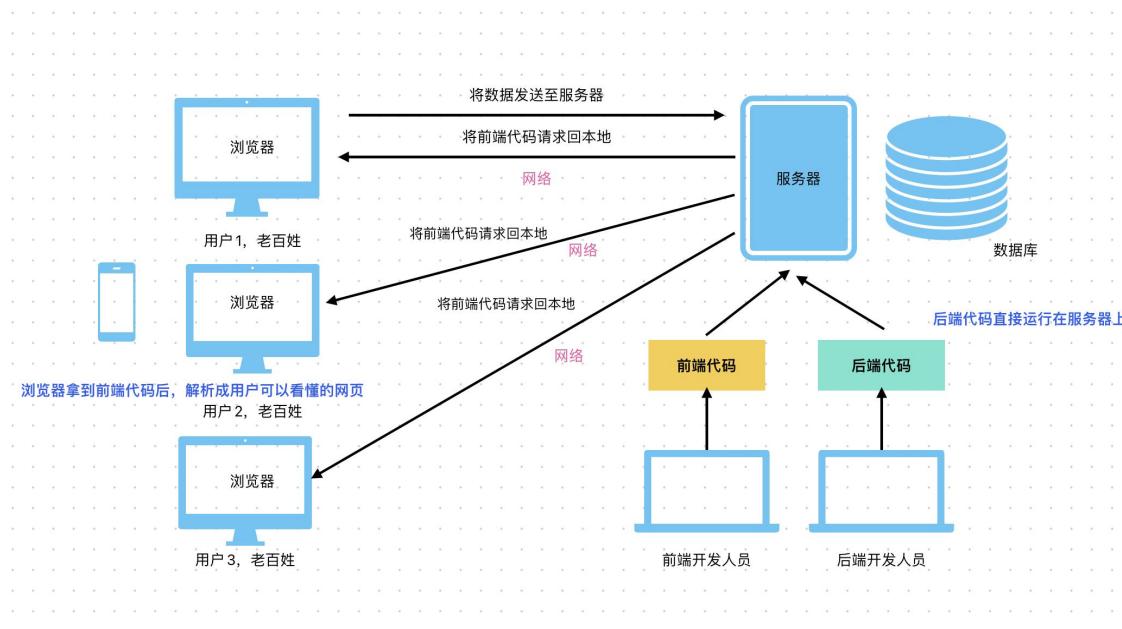


图 1.1.1-互联网基础架构

我们刚才说到了前端和后端，实际上无论前端和后端我们都是写代码的（这听起来确实有点卑微……）。那么前端和后端写完代码之后呢？这个网站/应用要怎么投入使用呢？这就要介绍我们的浏览器和服务器的知识了。浏览器大家应当很熟悉，都用过浏览器看过网页，其实浏览器就是相当于一个手机屏幕，尽管里面的内容会不断变化，但是它本身却是固定的。重点说一下服务器，服务器顾名思义提供服务的。服务是什么？服务这个概念对于初学者来说很难理解？其实服务解释开来就两句话：1.我需要查询什么你就给我什么 2.我需要你帮我

计算什么，你帮我计算，并且把计算结果给我。这就是服务。因此当用户在浏览器上点击按钮把商品添加到购物车的时候，服务器就收到了用户的请求，服务器就知道了“OK，你想买这几样东西，我帮你记录一下商品信息，并且计算一下多少钱然后告诉你”。那么如果你没结账，等你下次打开软件的时候你发现你购物车里的东西还在。这就是服务器在帮你保存这些数据。服务器从物理的概念上可以理解为“一个电脑主机”。和我们家中的电脑主机没什么区别，只不过它有三个特点：1.海量的数据存储能力（比你家里的电脑存储的多）2.超强的计算能力（比你家里的电脑计算的快）3.强大的通信能力（能够同时跟很多台计算机进行通信）。

按照图 1.1.1 的解释，我们的服务器要同时承受很多用户（浏览器）发送的数据和交互，要和很多浏览器通信，那么具体的这些通信过程中产生的数据存储在哪里了呢？那就是我们服务器上的数据库里面。数据库也可以理解为一个软件，专门用来存储数据的，常用的数据仓库有 MySQL, PostgreSQL, SQL Server 等等。

我们的前端代码和后端代码写完之后都会打包（该压缩压缩，该优化优化）扔到服务器上。服务器上准备好环境然后一行命令就能让这些程序跑起来（正常使用）。然后我们的用户打开浏览器输入了一个网址比如说淘宝网，这时候用户的电脑会把这串地址解析并且转发到淘宝的服务器上，告诉服务器“有个用户她想‘剁手’了”。然后服务器就会把服务器上的前端代码，注意是前端的代码你没听错，发送给用户。用户这时候还停留在浏览器面前等待网页的展示是吧？那用户的浏览器收到了服务器发送过来的代码之后，会把这一部分代码解析成，转换成，翻译成我们眼睛里看到的网页。实际上浏览器看到的都是代码，只不过这个好心人能够把这些代码翻译给我们看。然后我们看到网页之后如果想买东西，那势必会点击某些按钮之类的，然后这时候浏览器会再次把我们的数据和信息记录下来发送给服务器，服务器根据我们的操作做出响应。这就是一个网站（app）完整的闭环逻辑。

如果看到这里你都能够看明白。恭喜你，我们的 WebGIS 也是这样一套逻辑。只不过我们的用户在浏览器上看到的是地图，我们的服务器完成的是地图数据的发布和地图中的某些计算和分析。所以这也正是为什么第一章要给大家做这个铺垫。

总结一下：本节必须搞懂的几个概念：**前端/客户端、后端/服务端、服务器、浏览器、前端三剑客、后端编程语言**。

## 第 2 节：前端基础

上一节说了前端和后端的概念，我们这一节详细的来说一下一些入门之前的准备知识，因为你的 WebGIS 学习之路必然也是从前端开始的。前端和 WebGIS 之间的关系就像是你要学会解二元一次方程的前提是你必须会加减乘除，前端现在发展的越来越快，如果你是个传统的前端，你得一直保持学习否则 2、3 年你就会掉队。就会跟不上现在的发展趋势。因此前端知识是你入门 WebGIS 开发者工作岗位必会的知识储备，但这并不是你的目的地，我们还是要以 GIS 本身作为发展的核心方向。但是前端有意思的是，不管怎么变化和发展，始终是万变不离其宗，HTML、CSS、javaScript 这老三样始终还是鼻祖，还是标准，还是你必会的东西。

其实在 WebGIS 书籍里面讲前端知识听起来确实挺怪异的，但是为了照顾毫无编程基础的小伙伴们，我们再三思考还是决定增加这部分内容，没办法其实我也是一个很宠粉的博主~ 不过我们只讲基础，程度就是满足我们 WebGIS 开发即可，过多深入的东西请大家选择其他的资料或者视频进行深入学习研究。笔者在此书中讲前端知识是为了各位在后续阅读 gis 相关开发代码中不至于看不懂，而并非真正的传统意义上的系统前端教学。

因此各位可以根据自己的实际情况，如果愿意看我讲的就继续往下读，如果喜欢在其他地方学习前端知识，那本节剩余的部分各位就可以跳过了，等学习完前端知识回来之后，再继续阅读第二章的内容。

前端该怎么学，从哪里学其实是个挺难回答的问题。这要看各位对于互联网的认知和知识水平了。我们姑且认为您是一个 0 基础的小白，我们一点点从基础开始跟各位讲解。通过第 1 节的学习我想各位应该大概对于一个网站，或者说一个 app 整体的运作逻辑有了些许的了解。前端的核心其实就是在编写页面，无论是 javaScript 也好，css 也好，html 也好，甚至是 vue、react 等各种框架也好。都是为了页面而服务的。说白了前端一直追求的方向就是写出好看、大方、美观、高效、炫酷、丰富多彩的页面。对于 html、css、javaScript 三者的作用和关系如果还有不明白的小伙伴，可以听听我的理解。就拿一个人举例子，我们人本身是由物质、细胞构成，那么我们的 html 就相当于我们的躯干，页面你也得有实体的代码存在，html 就能够搭建基础骨架，让人“成型”，而 css 的作用相当于你采用某种手法，比如去整形，去美容让自己变得好看一些，变得美丽一些，css 美化了人的外表。同样都是人有的长得好看有的长得丑，同样都是 html，不同的 css 作用上去好看程度天壤之别。而 javaScript 就更强大了，它相当于人的行为，让人可以“动起来”，可以

除了静态看起来好看之外还能有动作, 有交互。这就是前端三剑客之间的关系和各自的作用。接下来我们分别简单的讲一下这三个部分。本节编写的代码都在本书附带的代码文件夹中, 各位可以使用 **vs code** 编辑器, 下载一个 “**Live Server**” 的插件然后在页面部分鼠标右键选择 “**Open with Live Server**” 来查看网页的效果。

**HTML 部分:** HTML 是一种超文本标记语言, 很多书上都这么说, 我们所看到的绚丽的网页内容都是由 HTML 标签代码翻译过来的。其实它很简单, 它的核心就是用标签来描述一切。什么是标签?

```
<name>张三</name>
<age>18</age>
<sex>男</sex>
```

这, 就是标签, 一目了然这段内容想表达什么对吗? 假设有个人它叫张三, 年龄 18, 性别为男。`html` 其实就是这样的方式, 只不过它不描述什么张三, 它描述的时候网页上的元素, 常用的元素有 `div` 标签 (就是内容块区域, 理解为一个盒子, 一块区域)。`span` 标签 (一行区域)。`input` 标签 (用户输入框)。`img` 标签 (展示图片) 等等等等。

你随手打开电脑新建一个后缀名为.`txt` 的文本文件, 然后把下面这段代码粘进去:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta name="viewport" content="width=device-width,
initial-scale=1.0" />
<title>Document</title>
</head>
<body>
我是一个网页
</body>
</html>
```

保存, 再修改后缀名为.`html`, 再用浏览器打开这个文件, 你会直接看到网页。也就是说其实计算机本身是支持这种文件格式的, 并且浏览器能够解析这种格式, 这也就意味着你

只要按照某种规则去写这个文件，计算机就能够根据规则进行翻译转换，将网页效果通过浏览器翻译在你面前，那么也就是说，规则和语法都是提前定义好的，你写代码的过程其实就是一个使用的过程，并不是真正意义上的开发过程。只不过我们把编写和生产网页的过程叫做开发。写 HTML 其实主要就是一个记忆的过程，要学会常用的标签，比较重要的标签有以下几个：

1. **div 标签**，这是最常用的标签，你可以理解为盒子，容器，用于承载内容的标签，div 是块状元素，也就是意味着它可以有宽高，可以多行展示，并且它的内部还可以嵌套很多元素，你可以在内部直接写文本，也可以在内部嵌套其他的标签。总之 div 是最基本的容器。你可以给 HTML 内的任何标签起 id，用于标记这个标签，后续在操作和使用这个标签的时候可以通过 id 找到它。div 通常决定了你页面的布局，你想要左右分栏的页面还是从上至下滚动展示的页面，还是不可滚动的页面等等，在最初的设计的阶段就要想好布局多少个容器，分别是什么位置。

```
<div>这里可以放置一段文字</div>

<div><div>div 里面也可以潜逃其他元素</div></div>

<div id="aa">

你可以给 div 起个 id 用于标记它，后续给这个 div 设置样式的时候就可以通过这个 id 查找到 div
</div>

<div style="width: 20px; height: 30px">

你还可以直接在 div 里面写样式，使用 style 关键字
</div>
```

2. **img 标签**，用于展示图片。你可以将图片的位置、路径、或者是网络请求地址填入此标签的 src 属性当中，这样就能够展示图片了。同时你可以设置这个标签的 width 属性和 height 属性来设置展示的图片的宽和高：

```

```

3. **input 标签**，input 标签的表面意思是输入框，即在页面上给用户一个输入框让用户输入内容，不过 input 不仅仅是输入框这么简单，它还可以是单选按钮，还可以是复选框设置可以是一个按钮。决定 input 类型的是它的 type 属性：

```

<input type="text" />这是普通的文本输入框

<input
  type="password"
/>这是用于输入密码的文本框，输入的内容不会显示在页面上

<input type="number" />这是用于输入数字的输入框，限制用户只能输入数字

<input type="radio" />这是单选按钮，常用于从多个内容中选择一个

<input type="checkbox" />这是复选框，常用于从所有内容中勾选多个内容

<input type="button" />这是按钮类型的 input

<input type="file" />这是用于文件上传的 input

```

4. **button 标签**，即按钮标签，书写这个标签就是在页面上新添加了一个按钮。
5. **style 标签**，这是用于我们编写 css 的标签，我们可以在这个标签内部写 css 的代码，我们写 css 的时候需要通过标签的 id 或者是类名来书写。要注意 css 标签要写在 head 标签内部，而不是 body 标签里，这一点要注意，我们在讲 css 的时候再细说。

```

<!DOCTYPE html>

<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width,
      initial-scale=1.0" />
    <title>Document</title>
    <style>
      /* 这是为 id 为 aa 的标签写的样式 */
      #aa {
        width: 100px;
        height: 100px;
      }
    </style>

```

```
</style>
</head>
<body>
<div id="aa"></div>
</body>
</html>
```

6. **script** 标签, 我们在这个标签内部可以写 **javaScript** 代码, 并且我们的 **js** 代码也只能在这个标签当中书写。 **script** 标签有两种写法, 第一种是我们想要从外部引入 **js** 文件, 这种方式的话 **script** 最好是写在 **head** 标签里面, 当然也有特殊情况, 遇到的时候我们在讲解。第二种就是我们直接在 **html** 中写这个标签然后在里面书写 **js** 代码, 这种情况通常是把标签写在 **body** 标签的外面, 并且是 **body** 标签的下方:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta name="viewport" content="width=device-width,
initial-scale=1.0" />
<title>Document</title>
<script src="./index.js"></script>
</head>
<body></body>
<script>
const a = "我是变量";
</script>
</html>
```

**CSS 部分:** 在 **html** 部分的时候我们简单说过写 **css** 样式的时候要通过标签的 **id** 先

找到标签，然后再为标签写样式，那么这个样式具体可以怎么写呢，都有哪些属性呢？我们挑一些常用的来跟大家讲讲：

1. **width 宽度**，控制元素的宽度，单位可以有很多种，可以是以屏幕像素为单位的 **px**，也可以是相对单位 **em**。还可以是百分比。
2. **height 高度**，控制元素的高度，单位同上
3. **background 背景**，控制背景，属性可以设置很多值，可以是颜色填充，直接写一个颜色值，也可以是图片，那么就可以直接写图片的 **url**。
4. **position 定位**，控制元素以何种定位方式出现在页面上，比如可以选择 **fixed** 固定在窗口，也可以选择 **absolute** 相对于父元素固定在某个位置，配合 **left** 和 **top** 属性来控制相对于父元素左上角的距离。定位方式有很多在这里花 2 页纸也讲不完，大家百度百度吧。
5. **display 布局**，通常可选的至有 **none** 表示隐藏不显示，**block** 表示以块元素显示，也就是说可以换行，**inline** 以行要素显示，即不能换行，还有取值为 **flex**，即弹性盒子布局，可以通过属性控制元素的布局方式，这个大家也可以百度深入了解。
6. **border 边框**，元素的边框属性，可以设置粗细，颜色。虚线还是实线等等。
7. **box-shadow 阴影**，可以设置元素的阴影，让元素看起来有立体的 **3D** 的效果。
8. **z-index 纵向层级**，在网页上显示元素的时候。总有元素在前面，有的元素显示在后面，**z-index** 就是用来控制元素层级的。
9. **color 内容颜色**，元素内部的文本颜色
10. **font-size 字体尺寸**，即元素内部字体的大小。

```
#aa {  
    width: 200px;  
    height: 300px;  
    background: black;  
    border: 2px solid red;  
    color: white;  
    position: absolute;  
    left: 10px;  
    top: 20px;
```

```

font-size: 20px;
display: flex;
z-index: 20;
box-shadow: 2px 2px 2px yellow;
}

```

**11. JavaScript 部分:** js 作为一门独立的编程语言，在前端中发挥着核心的作用，一方面它可以操作 html 和 css。另一方面它可以和后端服务进行交互。因此是前端最核心的部分。按照别人教学，学习 js 肯定要从下面几个概念入手：关键字、变量、作用域、数据类型、函数、类、逻辑判断、原型及原型链。但是我的方式向来和别人不一样。我们换种方式来学习。假设现在让你设计一门编程语言，和 java、python、go、以及 js 等语言齐平的语言，能够让用户拿去开发，你该如何去做？这听起来很难，我连编程开发我都不会，我一门语言都不会你还让我设计一门语言，这不是开玩笑吗？

别急，我们静下心来慢慢来思考这个问题，计算机之所以能读懂我们的编程语言，都是因为这些编程语言最后都有一个编译的阶段，也就是要把我们写的代码转换成计算机能够读懂的二进制代码即 000111001001，可能一句代码最后给了计算机，计算机接收到的就是 0011001100111100110011，这些 0, 1 你肯定是看不懂，那没关系计算机能懂就行了，计算机 cpu 在进行计算的时候读取的就是这些 000111，我们暂且不管我们的代码是如何翻译成这些 000111 的，我们要考虑我们设计的编程语言让用户用起来有逻辑，有足够的基础能力能让用户在上层进行各种复杂的操作。因此我们必须把语言设计的足够简单。

计算机一个非常强大的功能就是存储，那么我们写代码的过程中也需要存储一些数据，因此我们必须设计一种格式或者写法来存储我们的数据——变量，油然而生。变量就是用来标记、存储相关数据来使用的。比如我要记录一个人的名字我可以这样写：

*name='张三'*

比如我想记录一个人的年龄我可能会这样写：

*age=19*

这样一来我们就可以通过变量的方式来记录一些数据，这里的变量和你中学学过的 x、y 变量没有任何区别的，你中学的时候不也经常写 *x=1* 吗？其本质都是通过赋值来记录数

据。那么理解到这里问题就来了，如果我们就这么简单的写 `a=b` 可不可以？会有什么问题？貌似看起来是可以没有什么问题，但是仔细想想还是有一些问题的，比如说如果采用上述的记录方式，我们好像阅读起来不太清楚这两个变量到底是记录什么数据的（如果不叫 `name` 或者 `age`），另外随着变量的增多，我们肯定会有没法确认的数据类型，因此很多语言要求声明变量的时候必须加上类型。比如 `java` 语言会要求这样写：

```
String name='张三'  
Integer age=19
```

那么紧接着下一个问题就来了，如果说我们在声明变量的时候还需要指定类型的话，我们是不是得先确定我们常用的数据类型有哪些？通常我们的文本类型的数据比如姓名、简介、地址等等数据都属于**字符串类型**的数据。而像年龄、分数等数据通常都是**整数类型**的数据。再比如我们想记录平均分这种数据，通常伴随着小数点的数据，我们可以用**浮点数数据类型**来记录。再比如我们如果想记录一个班级的所有学生我们可以使用**数组类型**，即`[‘A同学’, ‘B同学’, ‘C同学’, ‘D同学’]`，再比如我们想记录一个人的一些属性信息，我们可以用**对象数据类型**`{name: "tom", age: 19, gender: "male"}`，总之我们有很多类型的数据。**但是我们在 js 当中声明变量的时候不用区分类型**。我们可以统一用一个 `var` 关键字来定义变量：

```
var a='张三', var b=10
```

如果你不高兴，这个 `var` 你不写也可以（但我不推荐），这就是 `js` 的特点，随性随意，因为 `js` 被称作是脚本语言，脚本嘛，本来就是随便写的。

我们再来详细的介绍一下这些数据类型。关于对象，通常是用来描述一个个体的一堆属性，比如说想表示一个学生可以是：

```
var student={name: "tom", age: 34, gender: "man", hobby: "drink"}
```

想表示一个动物，比如一只猫可以是：

```
var cat={type: "cat", age: 3, color: "white", weight: 12}
```

因此对象这种数据类型是可以描述个体的，尤其是拥有很多复杂的属性的个体。我们访问对象的时候可以直接访问 `student`、`cat` 这样的变量名，如果想访问对象的某个属性，可以这样来写：

```
student['name'] // 这表示我想访问学生的名称这个属性，得到的结果是 tom
```

关于数组，以`[]`的形式进行数据存储，数组里的元素类型可以是任何数据类型，比如：

```
var names=['a', 'b', 'c', 'd'];
```

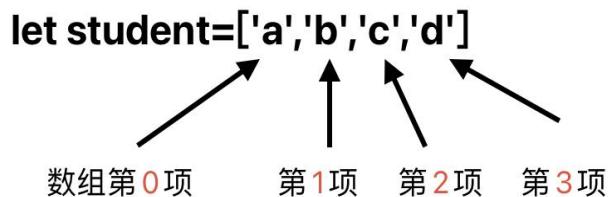
也可以是：

```
var num=[1,3,4,5,6,7]
```

还可以是

```
var student=[{name:'a',age:4},{name:'fh',age:6},{name:"haha"}]
```

如何来访问数组呢？答案是通过下标的方式，在 js 中，当你声明好一个数组之后，js 内部会为数组的每个元素都打一个标记，我们称为下标。下标使用的是数组，从 0 开始，比如说 `student[0]` 表示的是 `{name:'a',age:4}`，而 `student[1]` 表示的是 `{name:'fh',age:6}`



数组有一个很重要的属性叫做 `length`，它表示数组的长度，比如说：

```
student.length 表示的是 student 数组里有多少个元素，这里的结果是 2.
```

数据类型的概念清楚以后我们再来思考一个问题，计算机除了存储之外还有计算的能力。那么如何计算？我们中学的时候学过函数，在编程语言中也是通过函数来实现计算的，即通过像函数内部输入几个参数，来执行一段计算过程，最后将计算结果返回。在 js 中使用 `function` 关键字来定义函数。什么是 **关键字**？就是指设计编程语言的作者留给开发者使用的“暗号”，也就是当开发者写出下面这样的代码之后，编程语言的作者就知道开发者是想写函数，然后就会处理对应的计算机编译代码的逻辑，我们之前写过的 `var`、`String`、`Integer` 等都是关键字：

```
function getAge(age){return age}
```

类似的关键字还有 `if` 表示如果，`else` 表示如果之外的另一种情况，`var` 关键字表示要声明变量，`const` 也表示要声明一个变量，不过 `const` 代表着所声明的变量不再允许改变，即 `const` 用于声明常量，比如

```
const PI=3.14159265358979(后面不允许修改这个 PI 的值)
```

还有 `let` 关键字。`let` 是 `var` 的一种新的替代方案，推荐能使用 `let` 就不要使用 `var`，因为 `var` 存在变量声明先后顺序问题。即 `var` 可以先使用后声明

```
a=5;var a=0
```

这样的代码在 js 中是允许的。这感觉有点变态说实话。

所以引入了 `let` 就不存在这个问题。`let` 也不是完美无瑕，就比如说 `let` 声明的变量是有作用域的，也就是说可能只在某一部分代码中生效。其他部分的代码是无法读取 `let` 的。因此关于 `var` 和 `let` 在具体使用的时候要根据实际情况来使用。

我们再来思考一个问题，其实程序的核心作用就是避免**重复的劳动**，如何来避免重复的劳动？就是把重复的计算和逻辑判断丢给计算机执行，那么计算机编程语言自然要有处理循环往复的计算的能力，这就是编程语言中的**循环**的概念。在 js 中使用

```
for(let i =0;i<n;i++){}
```

这样的语法来实现循环，即 `for` 语句，`for` 也是关键字，这在其他语言中也大差不差，循环是一种思想，当你遇到一门你不会的新语言的时候，你可以搜索如何在 `java` 中实现循环，百度上到处都是答案。所以掌握循环这种思想，用于处理大量的数据。

比如按照上面讲对象时候的例子。对象的属性是可以被遍历被循环的，比如我们可以把一个对象放入到 `for` 循环中：

```
for(let key in student){
```

`key` 表示 `student` 中的每个属性名即 `name`、`age`、`gender` 等

`student[key]` 表示每个属性值即 `tom`、`34`、`white` 等

```
}
```

这里我们看到了 `for` 的其中一种用法，即 `for in`。

再比如，数组也可以被循环遍历。如果使用 `for` 循环来循环遍历数组，可以这样写：

```
for(let i =0;i<student.length;i++){
```

`student[i]` 表示的是数组中的每个元素

```
}
```

这里是 `for` 的第二种用法，也是最经典最基础的 `for` 循环。`let i=0` 表示从数组的哪个位置开始循环，`i<student.length` 表示到什么位置结束循环，`i++` 表示循环从起始点到终止点依次继续。

循环是一种思想，那么编程语言中还有另外一种很重要的思想——**判断**。由于计算机只认识 **0** 和 **1**。这就意味着计算机的世界是对立的，只有是，或者否，开或者关，进或者退。因此在编程语言中出现了 `if` 和 `else` 关键字用来区分是或者否，当然还存在这样一种情况，

就是不仅仅是是和否，有很多种情况需要判断。比如说，小明如果在 10 岁到 20 岁之间，小明属于少年。如果在 30-40 岁之间属于中年。60-80 岁之间属于老年，这就是三种不同的情况，在程序中需要这样来写判断（伪代码）：

```
if(10<age<20){  
    state='少年'  
}  
else if(30<age<40){  
    state='中年'  
}  
else{  
    state='老年'  
}
```

因此，当你有多个情况需要判断的时候你可以用 `if else if... else if... else` 来判断。

我们先暂停下来慢慢回顾总结一下刚才学到的，首先我们说在 js 中定义一个变量的时候不需要声明其类型，只需要根据情况使用 `var`、`let`、以及 `const` 关键字声明即可。其次我们说虽然不需要声明类型但是我们要清楚一般的数据类型有哪些，字符串、整型、数组、对象等等。区分这些是为了方便针对不同的数据采用不同的数据类型去存储。然后我们说编程的时候比较重要的几个逻辑，第一个是用于计算的函数 `function`，然后是用于处理批量数据的一个逻辑——循环 `for`，而后是用于处理判断逻辑的 `if/else`。这些内容已经基本的构成了编程的骨架，再结合之前的 `html` 和 `css` 我们来写个小例子帮助大家巩固所学到的知识。

假设现在班级里有几个学生，我们需要设计一个页面来完成对这几名学生的一个统计，那么首先我们在建立这些学生数据的时候肯定是采用数组来记录了，数组里的每一个元素我们肯定是使用对象这个类型来描述学生了，每个学生自身包含两个属性，一是名字 `name`，而是成绩 `score`。

```
let students = [  
    { name: "a", score: 5 },  
    { name: "b", score: 7 },  
    { name: "c", score: 2 },  
    { name: "d", score: 8 },  
    { name: "e", score: 3 },
```

];

接下来我们需要写一个函数，来计算出这群学生中分数最高的一位，我们势必要使用 `for` 循环来做这件事情了。

```
function getMax(student) {
  let max = {};
  for (let i = 0; i < student.length - 1; i++) {
    if (student[i].score > student[i + 1].score) {
      max = student[i];
    }
  }
  return max;
}

const result = getMax(students);
console.log(result);
```

仔细阅读上面这段代码，首先我们使用 `let` 来声明学生数组。不能使用 `const` 因为后续你的数组中的内容有可能会变化，然后我们编写了一个函数叫做 `getMax` 这个函数用于计算学生中的成绩最好的一位并且返回，函数的参数就是一个学生数组，就是把我们的 `students` 传入即可。

在这里顺便说一下函数的使用，在 js 中，函数分为声明阶段和调用阶段。声明阶段的意思就是先定义函数，先写明白函数有哪些参数，要执行哪些操作，要返回什么，声明的方式是 `function` 关键字+函数名称再加()内部可以填参数，再加{}内部写函数的执行逻辑。函数的调用阶段是指要使用这个函数，调用方式为函数名字加()如果有参数将参数放在()中。紧接着我们又在函数中定义了一个变量 `max`，它的类型是一个对象，{}中没有内容表示它是一个空对象。`max` 的作用是等待记录数组中成绩最好的学生。随后我们写了一个 `for` 循环开始循环学生数组，然后紧接着我们写了 `if` 语句用于判断。判断什么呢？在这我们采用了一种比较方式，即每相邻的两个学生进行比较，分数较高的学生会赋值给 `max`，这样一来，随着循环的进行。`max` 记录的总是成绩最高的那个学生。在这里要注意，我们 `for` 循环内部的条件写的是 `i < student.length - 1` 为什么是这样呢？请大家思考如果写成 `i < student.length`，那么循环走到数组最后一项的时候 `student[i+1]` 岂不是没有值了？因此我们应该让 `student[i+1]` 成为数组的最后一项，这样一来，我们数组

的长度就必须是 `student.length - 1` 了。请大家仔细揣摩。

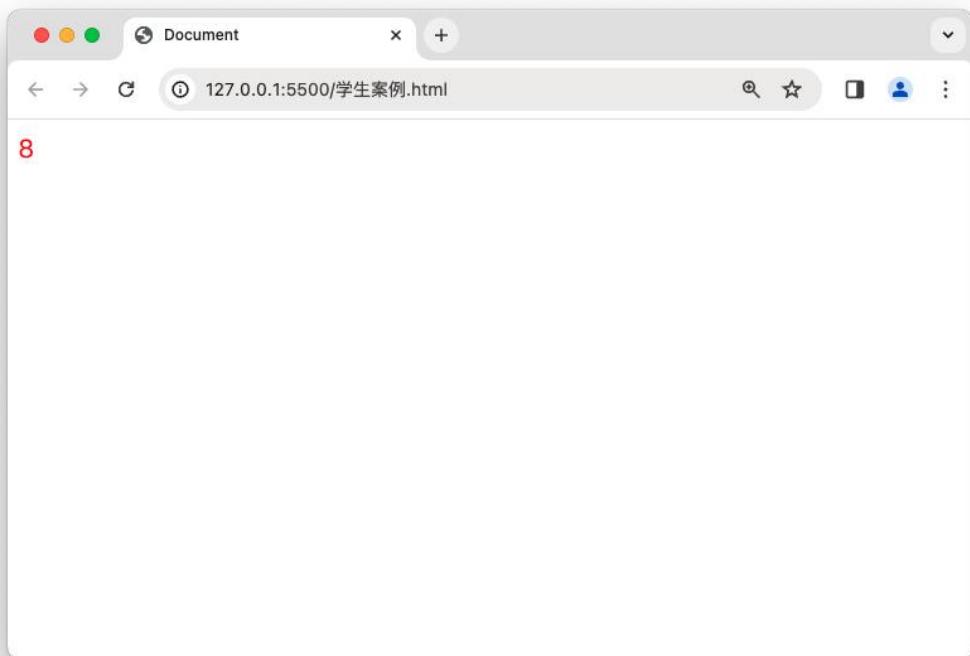
明白了上述的 `js` 操作之后，我们再来结合 `html` 和 `css` 让这个结果显示在页面上，我很早就讲过 `js` 可以操作 `html`，如何来操作呢？我们要使用一个关键的 `document`。在 `js` 中 `html` 其实就相当于一个文档，`js` 认为我如果想用代码操作这个文档，那么首先我要把所有的操作方法都集成在一个对象中，就给起了个名字叫 `document`。比如说我们要在 `js` 中获取 `id` 为 `aa` 的一个 `div` 元素。我们要这样写：

```
const div=document.getElementById('aa');
```

如果我们想把刚才计算出来的成绩最好的学生成绩填充到 `id` 为 `aa` 的 `div` 元素中，应该这样写：

```
div.innerText=result.score;
```

这样操作下来页面上就展示出这个学生成绩，再配合我们学过的 `css`，我想在页面让这个学生成绩展示成红色：



完整的案例代码如下，初学者可以仔细回顾理解：

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta name="viewport" content="width=device-width,
initial-scale=1.0" />
<title>Document</title>
<style>
#aa {
color: red;
font-size: 18px;
}
</style>
</head>
<body>
<div id="aa"></div>
</body>
<script>
const students = [
{ name: "a", score: 5 },
{ name: "b", score: 7 },
{ name: "c", score: 2 },
{ name: "d", score: 8 },
{ name: "e", score: 3 },
];
function getMax(student) {
let max = {};

```

```
for (let i = 0; i < student.length - 1; i++) {
  if (student[i].score > student[i + 1].score) {
    max = student[i];
  }
}
return max;
}

const result = getMax(students);
const div = document.getElementById("aa");
div.innerText = result.score;

</script>
</html>
```

到目前为止相信你已经了解清楚前端的一些基础知识了，但这还远远不够，接下来我们稍微深入的讲解一些进阶知识。

**构造函数与类：**我们之前说可以用对象来表示一个个体，表示一个学生，一只猫等等，但是会有一个这样的问题，群体如何来表示？或者说一种类别的物体该如何表示？比如人类？比如猫科动物类？汽车类？等等，自然界有很多种类别的物质，同一类别之下的个体有共性也有差异。如何来描述这种共性和差异呢？

编程语言提出了类的概念，如果是学习 `java` 等后端语言的小伙伴对此肯定熟悉不过了，但是在早期的 `js` 语言中，有类的概念但是没有类的关键字，在别的语言中可以通过 `class` 关键字类声明一个类，但是在早期的 `js` 中是没有的。因此 `js` 的开发者们就使用构造函数的方式来代替类。什么是构造函数呢？其实就是给函数换一种使用方式，之前我们学过函数的声明是 `function` 关键字加`()`再加`{ }` 使用的时候直接是函数名字加`()`。但是构造函数的使用需要用 `new` 关键字来使用，而且有一个不成文的规定是**构造函数的首字母要求大写**，以做到和普通函数有区分比如现在我们使用构造函数来模拟一个人类：

```

<script>

function Person(name, age) {
    this.name = name;
    this.age = age;
    this.eat = function () {
        console.log("i can eat");
    };
    this.say = function () {
        console.log("i can say");
    };
}

</script>

```

我们可以看到，构造函数内部使用 `this` 关键字，来接收一些参数，并且在函数内部可以在此声明方法，也是使用 `this` 关键字，为什么要这样做呢？答案很简单——构造模板。我们都应该早起的活字印刷术，雕刻一个模具，然后把纸张按在上面印刷，实际上自然界的所有的类别的物质都是按照某个模板去复制的。因此写构造函数的过程就是构建模板的过程。那么为什么是 `this.name=name` 呢？这里就是用来体现个性化差异的，因为每个人的名字不一样，所以你传给我什么名字，我就叫什么名字。我们称 `name`、`age` 这些能够决定未来被构造个体的特征的变量成为属性。像 `say` 和 `eat` 这样的能够决定未来被构造的个体的行为的函数叫做方法。模板构建好了之后我们就要考虑批量生产的问题了，如何来使用上述的构造函数来创建一个“人”？

```
let p1 = new Person("小明", 18);
```

如何再构建一个“人”？

```
let p2 = new Person("小张", 24);
```

如何让第一个人说话？

```
p1.say();
```

如何让第二个人吃饭？

```
p2.eat();
```

是的，就是使用 `new` 关键字不断的创建新的个体。不过我们在程序中不叫个体，而叫 **实例**，我们认为构造函数所定义的内容是抽象的概括性的，而具体的实例才是真正投入使用。因此我们在将来的学习中也会经常遇到这样的方式，就比如我们的地图可以抽象为一个 **Map** 类，后续在页面上显示地图的时候可以构建一个 `map` 的实例，如果需要多个也可以重复构建，我们把从构造函数到具体实例的构建过程叫做**实例化**。

理解清楚构造函数之后其实类的概念也就理解了，在 `js` 中这两个概念是相同的，只不过早期没有类的时候我们只能用构造函数，现在有了类关键字 `class`，我们就可以直接使用 `class` 来写一个类：

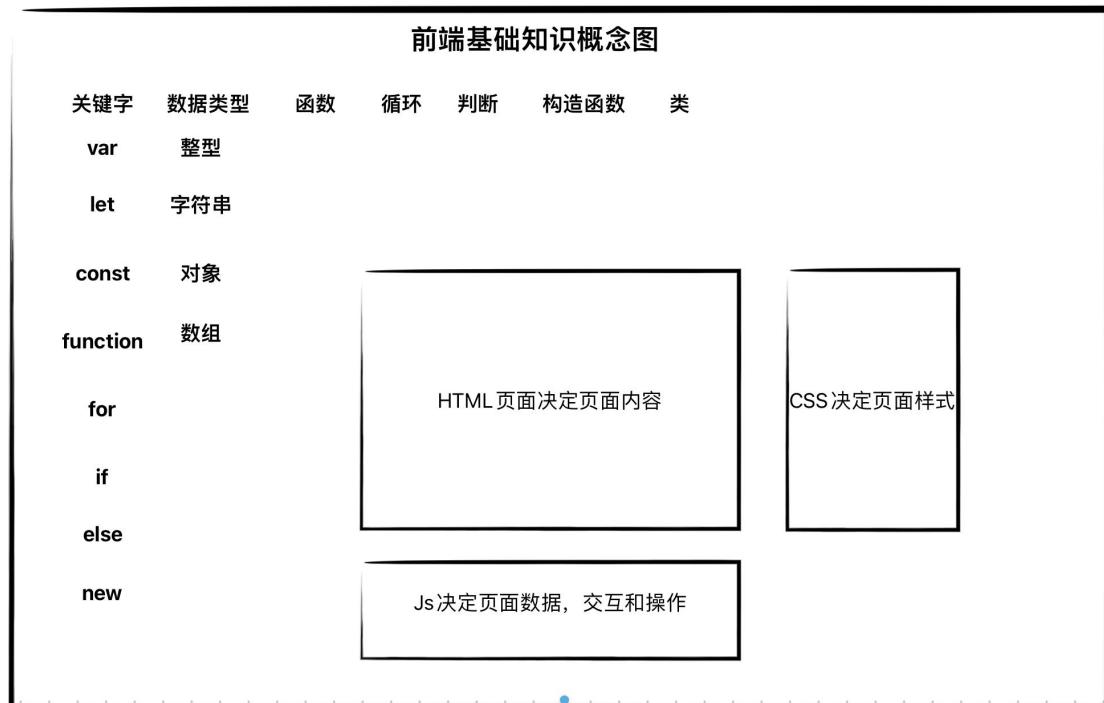
```
<script>
  class Person {
    constructor(name, age) {
      this.name = name;
      this.age = age;
    }
    eat() {
      console.log("i can eat");
    }
    say() {
      console.log("i can say");
    }
  }
</script>
```

在 `js` 中类的声明只是和构造函数语法不一样，其本质都是相同的。在 `class` 声明的类中，使用 `constructor` 来作为构造方法，就是代替之前的 `this` 接收参数，这样做比较规范一些，所有的属性值得接收都在这个 `constructor` 内部，而方法也不用再写 `this` 了直接写方法名加()再加{}，大家只需要记住这个写法即可。类在具体的创建实例的时候和构造函数创建实例的写法一模一样，在这里我就不重复了。

关于类和构造函数现在 `js` 都支持，实际上也不可能丢弃掉其中一个，因为本质上类是

构造函数的语法糖 (就是语法让人用起来简便一些的一种书写方式)。所以各位在开发过程中使用哪个都可以的。

学海无涯，长路漫漫，js 这门语言有很庞大的知识体系，不要说 GIS 开发，就是普通的前端也需要很长时间的学习才能达到一个不错的水平，这本书里留给我们讲前端的篇幅已经不够了，因此更加深奥的 js 知识就留给各位前去探索吧。笔者以上所讲的知识对于入门 WebGIS 开发似乎也足够了，还希望大家能够细细揣摩好好理解。



前端要掌握到什么程度？这是个好问题。大家不妨学完本章之后先接着往下学习，如果能够读懂第 5、6、7、8 四章的内容，那我想已经没必要在学习前端知识了，反之可能需要花一些时间，大概 2~3 周的样子，先掌握一些 JavaScript 基础知识包括：变量、数组、对象、函数、构造函数、类、原型、原型链、计时器、事件及事件监听、以及一些 ES6 的新语法等等，然后使用这些概念能写一个基本的购物网页，要求不复杂实现简单的商品排列，点击添加到购物车，然后前端静态的结账就好了。这里面涉及到一些 css 的知识比如说：布局，定位，弹性布局，阴影、动画效果等等。大家可以花点时间研究一下。在这里也给大家推荐一点学习前端的好资源，大家可以在 B 站上搜索自己喜欢的前端教程，没有什么最好的，适合自己就是最好的，那个能听进去觉得有意思，就学习哪个（我在这里比较推荐大家学习尚硅谷张天禹老师的前端视频），我个人认为他讲的比较通俗易懂，适合各位学习。

关于前端的框架在这里不要求大家必须掌握。现在 vue、react 火的一塌糊涂，但是大家在工作中同时学会两个这也不太现实，最关键的是，我们的 GIS 开发不是很依赖与前

端用什么框架，用什么对于我们来讲都没太大所谓。当然如果你会使用框架，那我相信你对于原生 js 的理解一定也不会差。这也是为什么本书我不会特别多的讲 vue、react 的原因。实际上这两个框架我都使用过，WebGIS 最重要的还是原理性的知识，掌握了原理你再去把它应用到不同的项目场景是一件非常简单的事情。其实不光光是前端，我们服务端也会使用各种语言和框架，例如 python 的 Django，java 的 spring boot，javaScript 的 node 等等。我们都会用到，所以如果你问我老师能不能教教我 spring boot，能，但是太累了。明天再来一个人问我 python，后天再来个人问我 node，我直接.....所以我不会教大家具体的某个跟 GIS 关系不太大的框架，但是我会教给大家，以 GIS 为核心，对应的框架都有哪些用处，该以什么样的方式去用以及各自的优点特点是什么。

## 第 3 节：图形学基础

上一节我们讲了前端的一些基础知识，如果说为了应付常规的开发，应该是足够了。但是很多时候我们需要用的技术相对来说比一般的前端要复杂的多，这时候就体现出来差距了。为何 WebGIS 从业者要学习的东西如此多如此复杂，可能这里也是其中一部分原因。但是各位可以换个角度思考问题，一旦我们掌握了更加有难度更加深层次的知识。是否我们的行业竞争力也会有所提高？是否我们的“身价”也会有所上涨？好，言归正传本节图形学基础，是想让大家对 canvas 有一个深层次的认识。方便在后续的开发中能够理解更多复杂的操作。

首先 canvas 也是一个 dom 元素，和 div 等 dom 元素有着类似的 API。但是不同点在于几乎是不能通过 html 来操作 canvas。只能简单的在 canvas 标签内部写 width 和 height 等属性来设置宽和高：

```
<canvas id="cav" width="800" height="600"></canvas>
```

更为复杂的操作需要借助 js 来完成，我们说 canvas 是一个画布元素，其核心用处就是在这一块画布上面画各种内容，这有点类似于其他任何画图软件，比如 iPad 以及 Mac 上的无边际等软件，就是用来写写画画。因此我们在绘画之前要了解 canvas 的用法，首先我们在绘画之前要获取一只画笔，你得有笔你才能画啊，这个画笔我们可以通过以下代码来获取：

```
const canvas = document.getElementById("cav");
const ctx = canvas.getContext("2d");
```

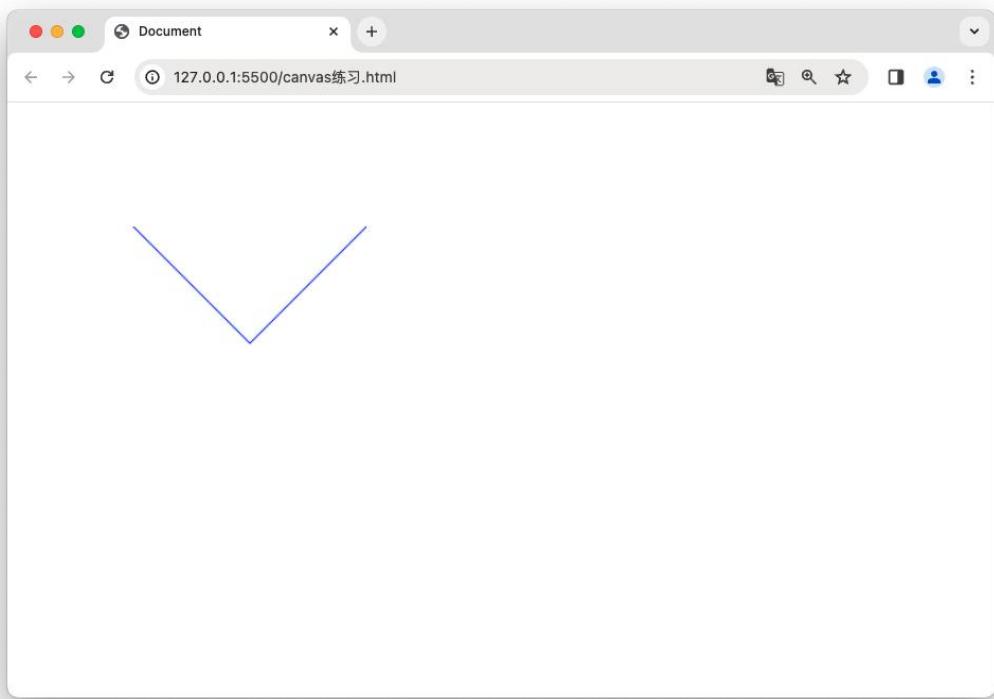
首先根据 id 获取到 canvas 对象，然后使用 getContext() 方法获取到这只画笔，

`getContext` 方法的参数中我传递了一个 “`2d`” , 其意思是我要采取 `2d` 模式来画图, 那么 `canvas` 除了 `2d` 模式之外, 还有 `webgl` 模式。`webgl` 模式就是 `3D` 模式。可以在画布当中渲染三维立体效果, 我们后面学习的 `cesium` 框架以及 `mapbox` 框架都是 `webgl` 模式。而早期的 `openlyaers` 框架是 `2d` 模式。不过最新版的 `openlayers` 也在渐渐向 `webgl` 模式迁移。有同学会问了老师那你直接讲 `webgl` 模式不就可以了吗? 不不不, 没学会走路之前学习跑步肯定是不现实的。`2d` 模式是基础, 只有掌握深刻了才能慢慢走向 `webgl` 模式。话不多说获取到画笔以后我们开始简单的使用画笔来绘制一些东西。比如说我们使用 `canvas` 来绘制一条线段:

```
const canvas = document.getElementById("cav");
const ctx = canvas.getContext("2d");

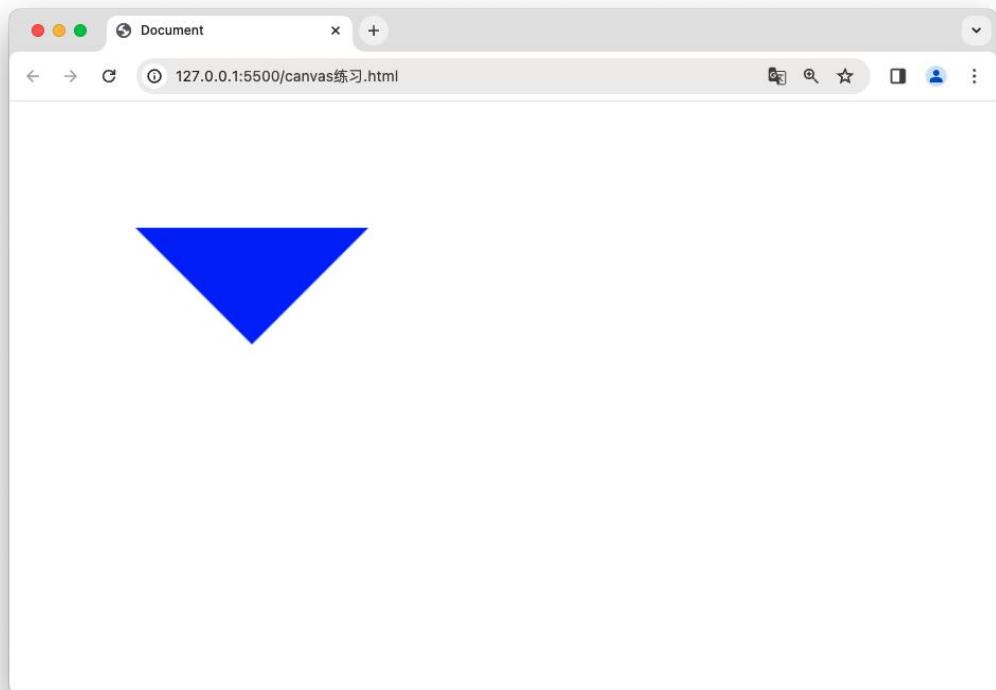
function drawLine() {
    ctx.beginPath();
    ctx.moveTo(100, 100);
    ctx.lineTo(200, 200);
    ctx.lineTo(300, 100);
    ctx.strokeStyle = "blue";
    ctx.stroke();
    ctx.closePath();
}

drawLine();
```



绘制线段的逻辑很简单，首先要告诉 `canvas` 我准备开始画了，于是会写 `ctx.beginPath`，然后 `ctx.moveTo()` 表示要把起始点定在哪里，这里的单位都是像素，也就是找屏幕上像素坐标为 `100, 100` 的点，然后 `lineTo` 表示要往哪个坐标点画如果要画很多次，那就需要多次调用 `lineTo` 这个方法，最后通过 `ctx.strokeStyle` 来给刚才所画的线条设置颜色，通过 `ctx.stroke()` 方法来确认你画的是一条线。为什么要这样写呢？因为前面 `moveTo` 和 `lineTo` 等方法只是确定位置，我们知道线和面其实本质上都是有多个点位组成的，因此如果不写 `ctx.stroke()` 的话，`canvas` 是无法知道你想要画线还是画面的。那么顺势我们来说在 `canvas` 里如何绘制一个面：

```
function drawFill() {  
    ctx.beginPath();  
    ctx.moveTo(100, 100);  
    ctx.lineTo(200, 200);  
    ctx.lineTo(300, 100);  
    ctx.fillStyle = "blue";  
    ctx.fill();  
    ctx.closePath();  
}  
  
drawFill();
```



其实在 `canvas` 当中还有很多复杂的操作，比如说如何在 `canvas` 中绘制一张图片并且修改图片的颜色，裁剪图片，以及在 `cavans` 中写一些炫酷的动画，甚至是使用 `canvas` 制作游戏等，都是 `canvas` 能够完成的。当然针对于 GIS 行业，`canvas` 中的操作就包括如何转换屏幕坐标与地理坐标？如何渲染对应的图形数据？如何加载栅格瓦片等等，这些深

层次的知识还需要各位在今后的学习中进一步的探索，当前的版本我们仅仅是为大家做一个铺垫和基础入门教学。因为这里的内容相对于底层基础，因此在后续的版本中还会不断的更新以加深这部分的知识，让各位在后续的开发中能够更加深入了解 WebGIS 的底层知识。

## 第 4 节：地理学基础

WebGIS 这 6 个字母的核心其实还是 GIS，所以搞 GIS 不知道地理学知识肯定是不行的。我清楚这本书的读者除了有学生之外还会有已经从业的，想要转行做 GIS 的开发人员。有趣的是学校里的学生大多数是懂地理的，但是不会编程。已经工作的朋友是会编程，但是不懂地理。因此前几节的内容是为了照顾还没毕业的学生，这一节是为了照顾一下非地理学相关专业的朋友。

其实这一节我也不想讲太多太难太深奥的概念，深奥的概念我们放在第三章再好好烧脑。这一节谈一点轻松的话题。

首先说说我们的地球吧，地球呢实际上是一个不规则的球体。并不是大家在各个地方看到的美丽的渲染图那样，其实真实的地球并不是那样的。真实的地球是一个坑坑洼洼，凹凸不平，有高有低，有山有水的大椭球。有网友画的一幅图很形象：

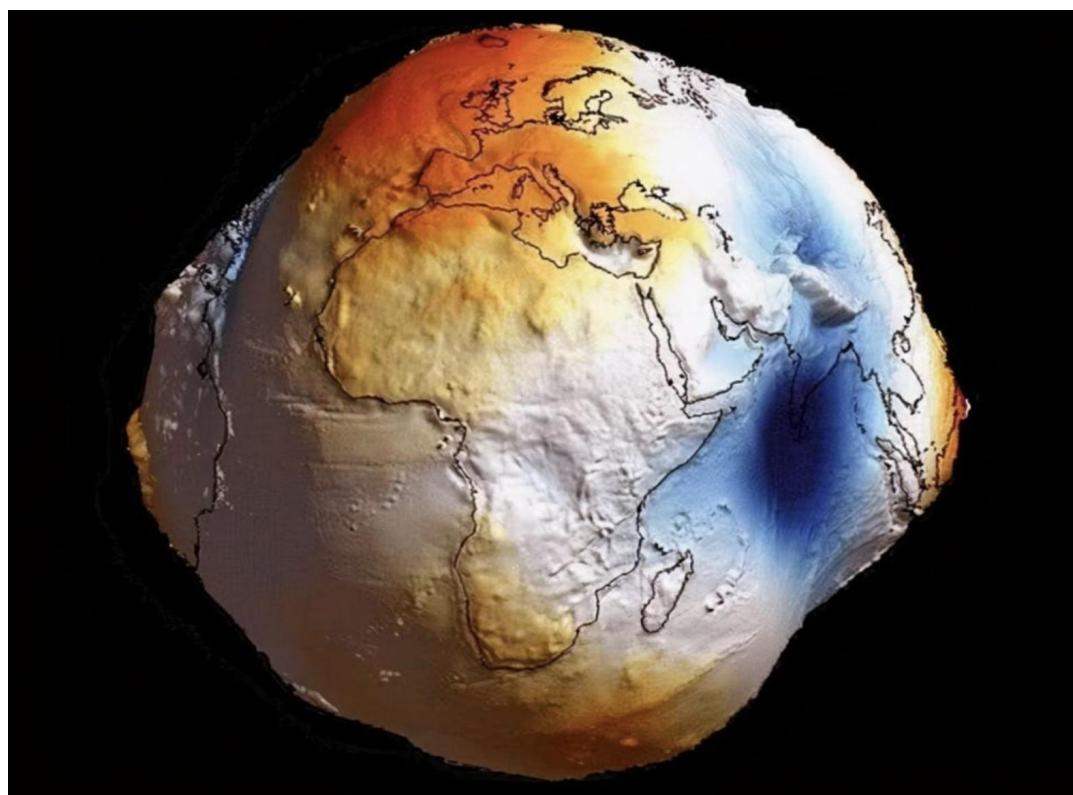


图 1.4.1-地球真实样貌想象

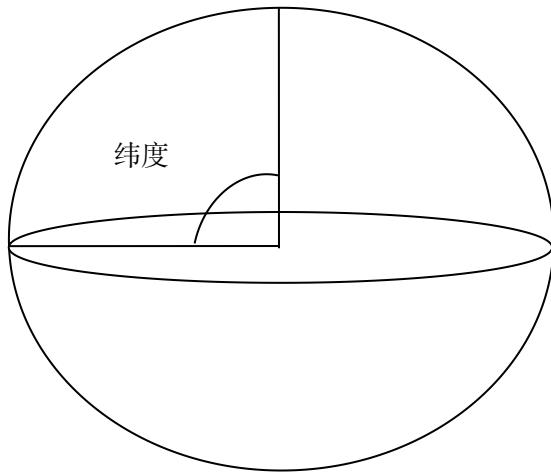
为什么说地球是一个大椭球，是因为真实的地球是两级略短赤道略长的球体。什么？你还不知道赤道和两极？快去百度百度！

那这样的话我们的地球在数学上就不能进行标准的计算了。因为它是个不规则图形。所以我们地理学要做的第一步就是把这个不规则的椭球体变成规则的。我们把它抽象为，简化为一个标准的椭球。方便计算。

所以我们的经纬度是怎么来的？就是简化成标准的椭球体以后，早在古希腊时期就有地理学家定义了经度和纬度的概念，经度用于表示地球的水平方向上的变化，也就是地球的东和西，例如中国位于地球的东半球，而英国位于西半球。纬度用于表示地球的竖直方向上的变化，也就是南北，例如俄罗斯在北半球，澳大利亚在南半球。

其实经纬度像极了我们中学学过的平面直角坐标系里面的  $x$ 、 $y$  轴 的概念， $x$  对应经度表示的范围， $y$  对应纬度表示的范围。经纬度通常会成对出现，例如  $120.4243234, 29.756756$  这其实就像是我们象限上的任意一个点都有自己的唯一坐标  $(x,y)$  经纬度的发明也就是为了确定地球上的唯一一个点。

经度的范围是 $-180^\circ$  到  $180^\circ$ 。why？按照常理来说，一个椭球体的半径切面应该是个圆。一个圆应该是  $360^\circ$ ，为什么经度的范围不是  $0^\circ \sim 360^\circ$  而是  $-180^\circ \sim 180^\circ$ ，为什么？答案还得去找英国人，英国人最先提出了经纬度的计算方式，并且把穿过自己国家的一条经线称作是  $0$  度经线，即 **本初子午线**。英国人认为：如果把经度定义成了  $0^\circ \sim 360^\circ$ ，你随便说个经度，我都判断不出来这个经度所在的位置离我远不远。因此英国人为了很好地掌握距离他本国的远近，就借助了数轴的概念，我们都知道在数轴上，不管正数还是负数，绝对值越大那就意味着距离原点越远。所以把经度定义成了  $-180^\circ \sim 180^\circ$  之后，只要经度的绝对值越大，英国人就知道这个位置肯定离他比较远。纬度的范围是 $-90^\circ \sim 90^\circ$ 。为什么呢？因为纬度的定义是地球重力方向的铅垂线和赤道平面的夹角。注意是夹角，如下图所示，当然是  $90^\circ$  了。



所以我们的世界范围是 $[-90, -180, 90, 180]$ 。

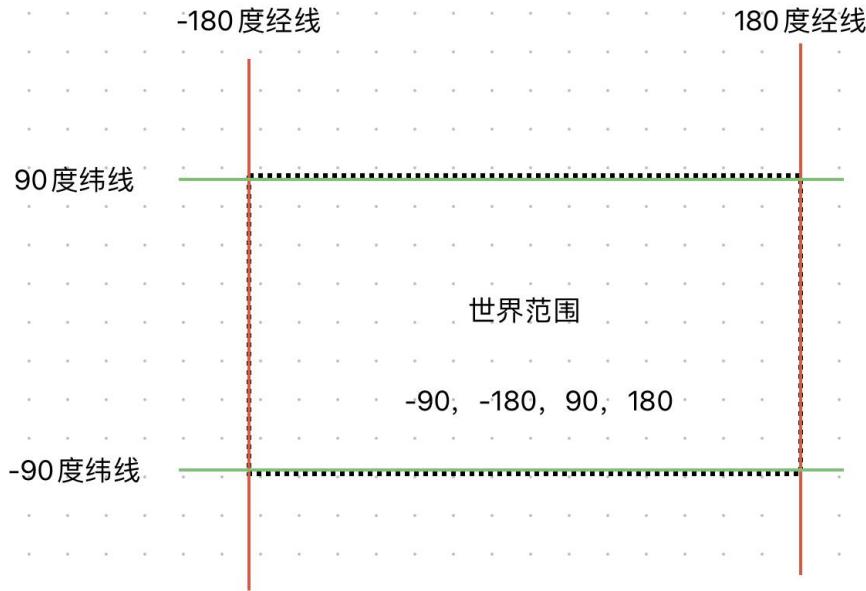


图 1.4.2-世界经纬度范围

也就是说，在我们的平面范围内（通常指浏览器的窗口范围内，最上端是北纬 90 度，最下端是南纬 90 度，也记作-90 度，最左侧为西经 180 度，也记作-180 度，最右侧为东经 180 度）。因此在平面内大家要熟悉，在东半球（中国范围内）来讲约往右边伴随着经度值越大，越往上端纬度值越大。好了我们在第一章不讲过于深奥和复杂的知识，我们暂且休息，第三章再来认真研究这个问题。

## 第二章：GIS 数据结构基础

### 第 1 节：GIS 数据介绍及分类

其实很多数据都可以作为我们的 GIS 数据。比如像卫星、航天飞机、以及现在特别火的无人机都可以拍摄一些照片，我们 GIS 行业称为影像或者说**遥感影像**。这种照片一般分辨率比较高，就是放大到很大的程度还是很清晰的能看到地物。所以这种数据也成了我们最常用的地图数据。这种数据的格式通常是 tiff 格式，即后缀名为.tiff。而不是我们常规印象里所理解的那种 png、jpeg 格式的照片。

为什么是 tiff？很简单，因为 tiff 可以携带其他信息，而不仅仅是图像本身的信息。**tiff 格式的文件是携带经纬度（坐标）信息的**。而且这个经纬度信息还能够被其他的软件或者工具所读取并且使用。那么为什么要携带经纬度信息？其实就是为了准确的定位。试想一下你给我一张照片并且不告诉我经纬度范围我怎么知道这张照片所拍摄的地物在哪里呢？我们定位为一个区域的逻辑很简单就是知道这个区域的经纬度范围。比如杭州的经纬度范围是：

[118.344957, 29.188757, 120.721946, 30.566516]，这就代表着从东经  $18.344957^{\circ}$  到东经  $120.721946^{\circ}$  范围内，从北纬  $29.188757^{\circ}$  到北纬  $30.566516^{\circ}$  范围内都是杭州。配合世界的范围是不是很轻松就确定了杭州的范围？

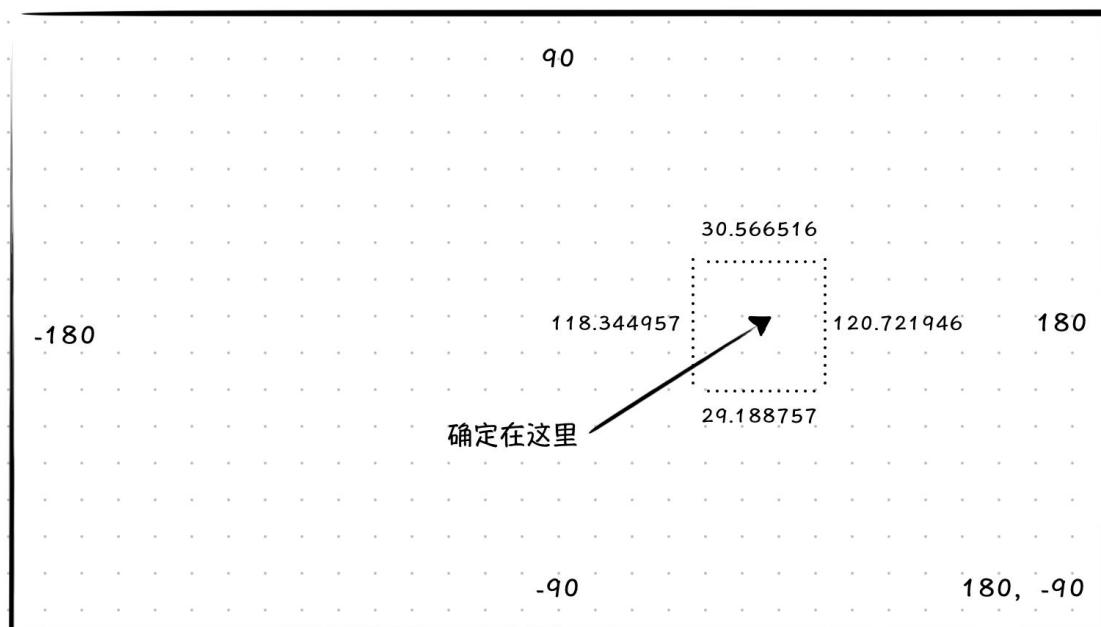


图 2.1.1-图层定位原理

虽然遥感影像是我们比较常用的数据类型，但是我们的 **png** 格式, **jpg** 格式的图片也不是不可以作为地图的数据源。就比如说我现在只有一张无人机拍摄的照片，它的格式是 **png**, 还没有经过处理变成 **tiff**。那么我们也能把它加载到地图上，另外一些手工地图，包括一些旅游区景点的手绘地图都是图片格式，我们也支持把它加载到地图上。加载的原理和 **tiff** 是一样的。**png** 和 **jpg** 格式不携带经纬度信息，那么就需要手动的在写代码的时候补充上这个信息。这一点我们会在讲解具体框架的时候给大家讲解。

除了影像之外，还有一些**测绘数据**也是我们**GIS**数据的主要来源。测绘数据包括一些经纬度仪器获取到的经纬度点信息，一些地块的形状面积等信息，以及道路河流等这样的线状信息。

最后还包括一些**人工衍生**的数据，比如从影像中提取出来（矢量化）的地物信息，类似于居民区，道路，绿地，河流等等要素。

总的来说，我们的**GIS**数据大致可以分为两类，专业的名称叫做矢量数据与栅格数据。形象的理解就是一些图形数据和图像数据，矢量就指的是图形数据，栅格就指的是图像数据。因为计算机能够渲染的也就是这两种类型的数据，我们地图中使用这两种数据也完全能够表示出地理学要素。

## 第 2 节：栅格数据

上一节中我们所说的遥感影像，图片格式的**GIS**数据我们统称为**栅格数据** (**raster data**)。这个概念很重要，后续的学习中会经常用到，在**GIS**领域里，一切的图片，图像，影片，影像格式的数据都称作是栅格数据。栅格数据其实就是图像数据。



图 2.2.1-栅格影像数据

### 第 3 节：矢量数据

与栅格数据相对应的是[矢量数据](#) (**vector data**)。我们接下来来介绍一下：

矢量数据是指一些人工手绘的，或者经过测绘而得来的一些图形数据。这种数据往往只有一些简单的轮廓构成，比如一个行政区可能只用轮廓边界表示，一条河流可能只用一条线段表示，一个商店只用一个点表示。这就是我们的矢量数据。

矢量数据往往来源于测绘或者是由栅格数据中得出。比如说现在我要调查杭州市所有的道路信息。我就可以实地去勘测然后描绘，或者我也可以从影像图中把道路都描绘出来。两种方式皆可。矢量数据其实就是[图形数据](#)。



图 2.3.1-矢量数据

WebGIS 开发过程中，矢量数据的主要格式有 **shape** 格式和 **geojson** 格式两种。

**shape** 格式是由 ESRI 公司提出并研发的一种用来表示几何图形及其属性信息的文件，**shape** 格式的文件一般包含 6 个文件，分别是后缀名为.**shp** 的图形文件，后缀名为.**dbf** 的数据库文件、后缀名为.**prj** 的坐标系信息文件、后缀名为.**shx** 的索引文件，以及后缀名为.**sbn** 或者.**sbx** 的空间索引文件。其中最后面两个是非必要的，一般来说前四个文件就能够完成 **shape** 文件的读写操作。一个基本的 **shape** 文件如下图：

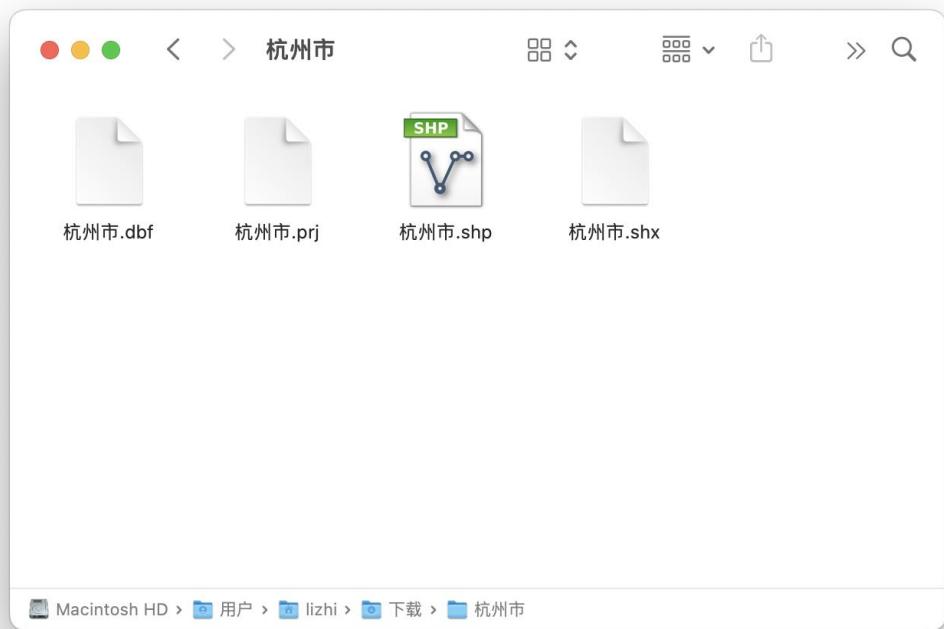


图 2.3.2-shape 文件组成结构

`geojson` 格式的矢量数据是我们 WebGIS 开发过程当中最常打交道的数据文件，没有之一。因为我们的浏览器直接读取 `shape` 文件比较费劲，因此大多数时候我们浏览器还是读取的 `geojson` 文件。此文件的后缀名可以是`.json` 或者是`.geojson` 都可以。说清楚 `geojson` 文件之前我们还得给大家科普科普 `json`。我们在预备知识那一章跟大家说过标签表示万物的方式，实际上标签表示万物的语法叫做 `xml` 语法，表示的文件叫做 `xml` 文件，`xml` 是早起互联网进行数据传输过程中的数据格式。

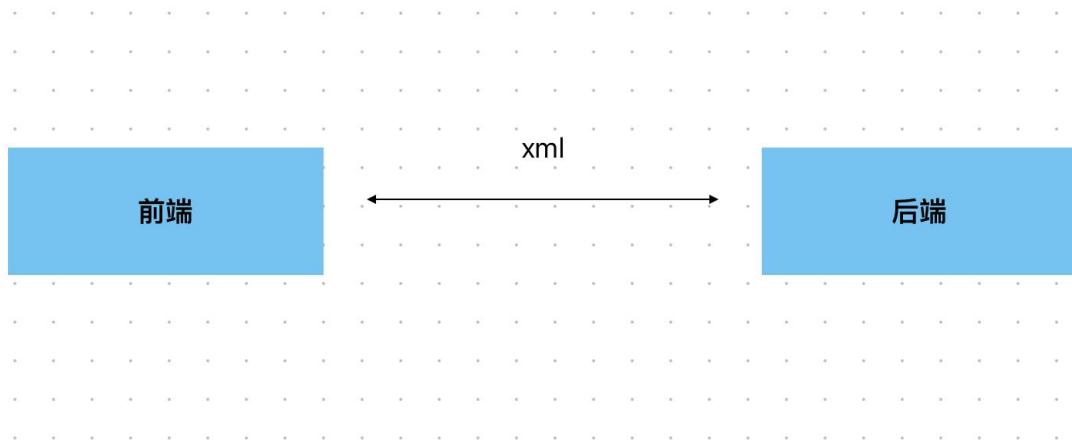


图 2.3.3-xml 作为传输数据

但是后来人们发现这样描述万物的效率太低了，每一组标签都需要有 2 个，`<name>张三</name>`这样才能描述一个 name 属性，那为什么不是：“name”：“张三” 呢，这样岂不是更简单？于是乎 json 格式就诞生了，并且逐渐的取代了 xml 的方式。

所以 json 格式很简单，就是什么是什么来描述万物，属性名:属性值，这样的方式也能够更好的跟编程语言所融合。

所以我们的张三同学要被表示成这样了

```
{
  "name": "张三",
  "age": 18,
  "sex": "男"
}
```

这样是不是又简单又直观。所以我们的地理矢量数据呢也采用了这种方式，我们的 json 叫做 geojson。一段标准的 geojson 应该是这样的：

```
{
  "type": "Feature",
  "properties": { "name": "杭州市" },
  "geometry": { "type": "Point", "coordinates": [120.3422342,
29.23423] }
}
```

其中第 1 个字段 **type** 表示类型，是一个 feature，feature 是要素的意思，什么是一个要素？一个点，一条线，一个面都叫做一个要素。这个字段通常是**固定**的。第 2 个字段 **properties** 代表着属性，这个字段用于存储 geojson 数据的属性，比如里面的 name: 杭州，就是存储这个点位的名字是杭州，属性可以自定义的编辑。第 3 个字段 **geometry** 表示几个图形字段，里面存储了这个要素的图形信息。图形信息是依靠**图形类别**和**坐标数据**（大多数时候是经纬度）来存储的。**geometry** 里面有两个字段，第一个 type 表示这个图形的类型，有 6 中类型可以选择，分别是 Point(点)、MultiPoint(多点)、LineString(线)、MultiLineString(多线)、Polygon(面)、MultiPolygon(多面)。点线面应该都好理解。在这里解释一下多面/多线/多点这种类型。举个例子什么是多面？比如我国的浙江省是附带很多岛屿的，

那如果我想表示浙江省的话势必会连同岛屿一起展示，但是浙江省是一个要素，一个要素又存在多个空间上不相连的面，这就是多面的概念。简单的概括一句话就是一个要素存在多个空间上不相连接的图形，用老百姓的话说，这叫**飞地**。



图 2.3.4-浙江省行政区

然后是 **coordinates** 字段，表示数据的具体的坐标信息，这个坐标可以是经纬度来表示也可以是投影坐标来表示。**coordinates** 这个字段也是特别的重要。初学者很容易在这个上面犯错。而且这种错误框架本身不会提示的，但是在页面上看不到效果。要注意区分 **coordinates** 里面数组的层数，记住一句话，一个面就是一个数组。如果是 **polygon** 类型的，**coordinates** 里面只有两层就像下面这样：

```
geometry: {
  type: "Polygon",
  coordinates: [
    [
      [119.996332, 30.181536],
      [119.987988, 30.174901],
      [119.963101, 30.170464],
      [119.959274, 30.168005],
    ],
  ]
}
```

但是如果 MultiPolygon 就有 3 层了, 像下面这样:

```
geometry: {
  type: "MultiPolygon",
  coordinates: [
    [
      [
        [
          [119.996332, 30.181536],
          [119.987988, 30.174901],
          [119.980168, 30.17405],
          [119.974296, 30.175107],
          [119.964669, 30.172854],
          [119.963101, 30.170464],
          [119.959274, 30.168005],
        ],
      ],
      [
        [
          [119.987988, 30.174901],
          [119.980168, 30.17405],
          [119.974296, 30.175107],
          [119.964669, 30.172854],
        ],
      ],
    ]
  ]
}
```

这一点细节各位一定要留心, 在今后的开发中这有可能将成为各位经常犯错和调试的部分。

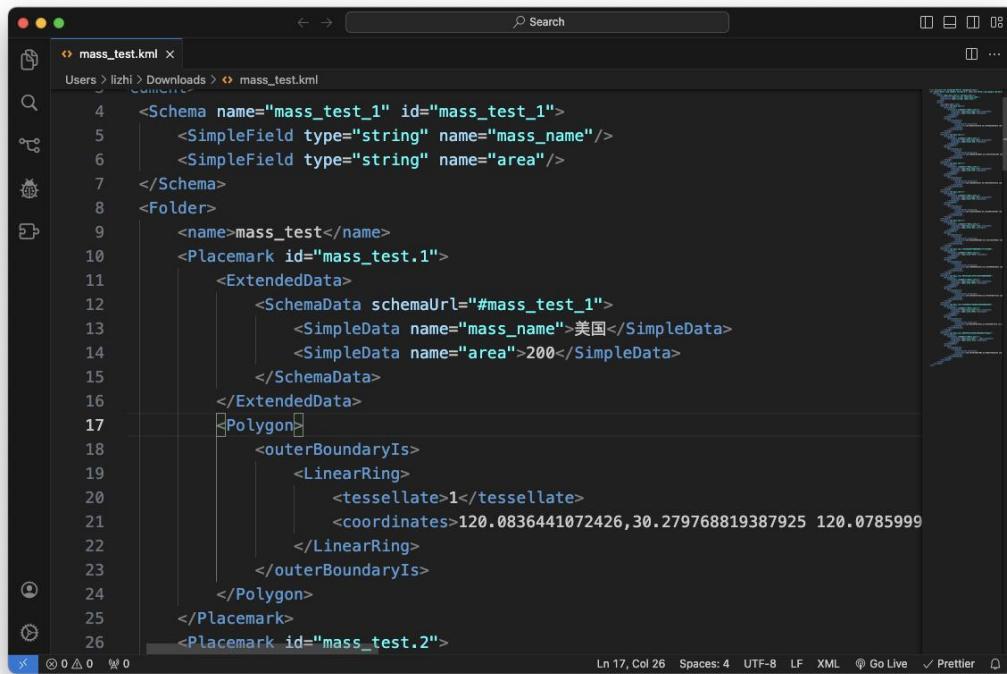
除此之外 geojson 数据还有另一种表示方法: 以集合的形式表示。在实际的业务开发当

中，有的时候需要一次性表示多个要素，那么就会有一个要素集合的概念。因此会把每一个 `feature` 都装进一个数组当中，然后外部再包裹一层对象，这样的话就表示成了我们的 `features` 集合。

```
{  
  type: "FeatureCollection",  
  features: [  
    {  
      type: "Feature",  
      properties: { name: "宁波市" },  
      geometry: {  
        type: "Point",  
        coordinates: [120.3422342, 29.23423],  
      },  
    },  
    {  
      type: "Feature",  
      properties: { name: "杭州市" },  
      geometry: {  
        type: "Point",  
        coordinates: [120.3422342, 29.23423],  
      },  
    },  
  ],  
};
```

因为 `geojson` 格式最经常被用到，所以它很重要，希望大家能够认真的学习并且掌握这种格式，有不少同学在实际的开发过程中因为 `geojson` 格式的基本功不扎实，导致产生了很难排查出来的 `bug` 而影响开发进程。并且后续的开发过程中，很多炫酷的特效以及动态渲染相关的知识都需要你对 `GeoJSON` 有一个非常清楚的认知。

矢量数据最常用的格式就是 `geojson`，但是还有一些其他格式的矢量数据需要我们了解，比如说 `kml`、`gml`、`wkt` 等。我们上文说过 `xml` 格式。其实 `kml` 和 `gml` 就是使用 `xml` 语法来记录地理数据，例如一段 `kml` 数据的格式是这样的：



```

<Schema name="mass_test_1" id="mass_test_1">
  <SimpleField type="string" name="mass_name"/>
  <SimpleField type="string" name="area"/>
</Schema>
<Folder>
  <name>mass_test</name>
  <Placemark id="mass_test.1">
    <ExtendedData>
      <SchemaData schemaUrl="#mass_test_1">
        <SimpleData name="mass_name">美国</SimpleData>
        <SimpleData name="area">200</SimpleData>
      </SchemaData>
    </ExtendedData>
    <Polygon>
      <outerBoundaryIs>
        <LinearRing>
          <tessellate>1</tessellate>
          <coordinates>120.0836441072426,30.279768819387925 120.0785999
          </coordinates>
        </LinearRing>
      </outerBoundaryIs>
    </Polygon>
  </Placemark>
  <Placemark id="mass_test.2">

```

图 2.3.5-kml 表示矢量数据

我们可以看到 kml 也可以记录属性数据和图形数据，图形数据的记录方式也很简单，就是直接记录坐标。类似的。gml 也不难理解，它就是谷歌提出的一种 xml 记录地理数据的方式。和我前文中讲解的原因一样，随着 xml 这种格式不如 json 便捷，kml 和 gml 也逐渐慢慢被淘汰。只有少量的比较古老的系统中采用到此类数据类型。对于这两种类型大家能够认识，并且会使用工具转换即可。工具我会在软件及中间件中讲解。

接下来我们还需要认识一个比较重要的矢量数据格式那就是 wkt 格式。一段 wkt 记录的地理数据如下：

```
POLYGON ((120.0836441072426 30.279768819387925, 120.07859992548151
30.215278518888496, 120.19966028772456 30.200457202491933,
120.19966028772456 30.266699882015928, 120.13711243389929
30.292836016463923, 120.0836441072426 30.279768819387925))
```

没错，正如你看到的，wkt 数据无法记录属性信息。它只能够记录坐标信息（图形信息）。属性无法记录，这是它的致命缺陷。但是反过来讲，这也是它最大的优点，那就是数据体积小，所有的矢量数据格式中，体积最小的一定是 wkt。如果你的需求中不关心属性数据，那么使用 wkt 将是最好的选择。

矢量数据和栅格数据各有优缺点。栅格数据的优点在于能够精准而真实的展示地物的样

貌，最真实的还原地球表面的状态。所以通常一些影像图能够作为底图。栅格数据的缺点在于可编辑行可交互性差，通常你不能对一张图片进行什么高端的操作（点击高亮，下钻，展示属性等等）。矢量数据的优点正好就是栅格数据的缺点，你可以对矢量数据进行交互操作。而且矢量数据很简单，数据体积往往不大。方便展示和交互。矢量数据的缺点在于表达的信息比较少，信息很有限而且不能知晓真实的地物信息。

矢量数据和栅格数据各有优缺点，适用的场景也不一样，所以还要根据具体的业务场景来定夺。

## 第 4 节：三维数据

三维数据是近些年来才不断产生的数据类型。随着科技与狠活的发展，三维数据一共分成 4 种类型给大家介绍。分别是倾斜摄影、手工模型、建筑信息模型、点云数据。如果各位将来要从事三维方面的开发，这些三维数据类型还是需要大家掌握的。但是如果各位的岗位职责仅限于二维的 GIS 方面的开发，就可以跳过这一小节了。

首先一些软件能够把航天设备拍摄的照片合成一个三维立体的格式，比如说[倾斜摄影](#)。这种数据就是我们航天飞机从上、前、后、左、右五个面拍摄 5 张照片，最后丢到软件里面一合成，就能够被我们的某些软件和框架解析成三维的立体的数据了。



图 2.4.1-倾斜摄影

除此之外，三维数据还有一些手工制作的[模型](#)，这种模型依赖于一些三维的制作软件例如 3d max 等。这种模型的格式通常是 gltf 或者 glb。

还有一种跟建筑相关的三维数据，就是**建筑信息模型** (Building Information Model 简称 BIM) 数据，这种数据来自于专业的建筑建模软件，但是这种数据它可以携带一些经纬度之类的信息，因此也被常用于三维 WebGIS 的开发场景中。

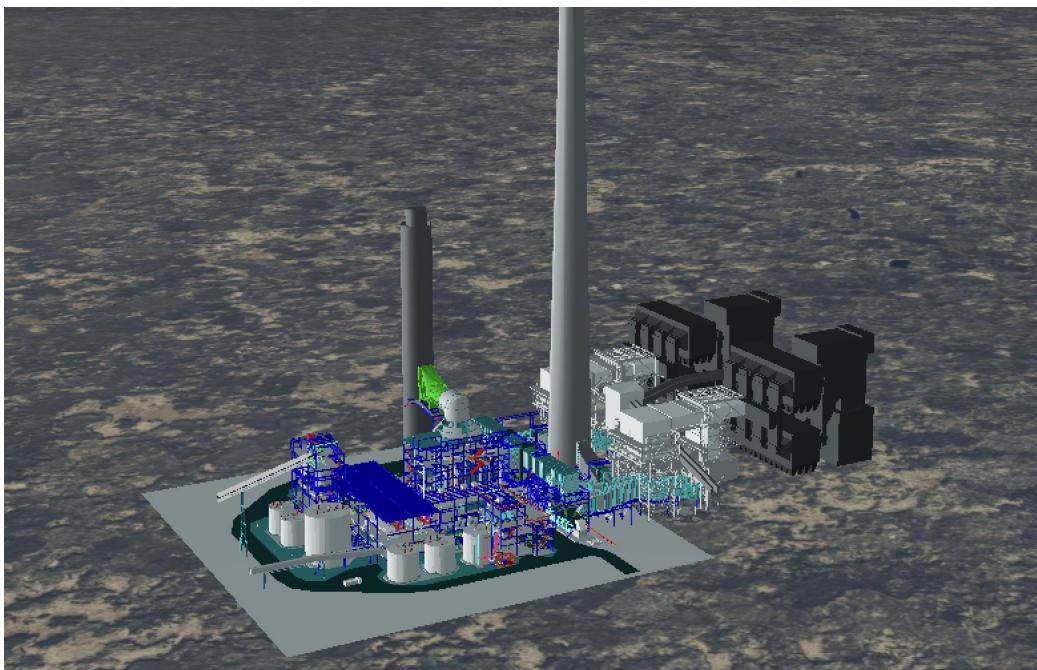


图 2.4.2-BIM 模型

最后一种三维数据是**点云数据**。点云的原理是来自与激光雷达设备。通过发射信号并返回信号来测算空间位置信息。有点像 iPhone 15 Pro Max 手机背部激光雷达的原理。通过发射一些信号计算空间内物体的轮廓为止，最后再使用很多的点描绘出一个空间三维的形状，这就是点云数据。

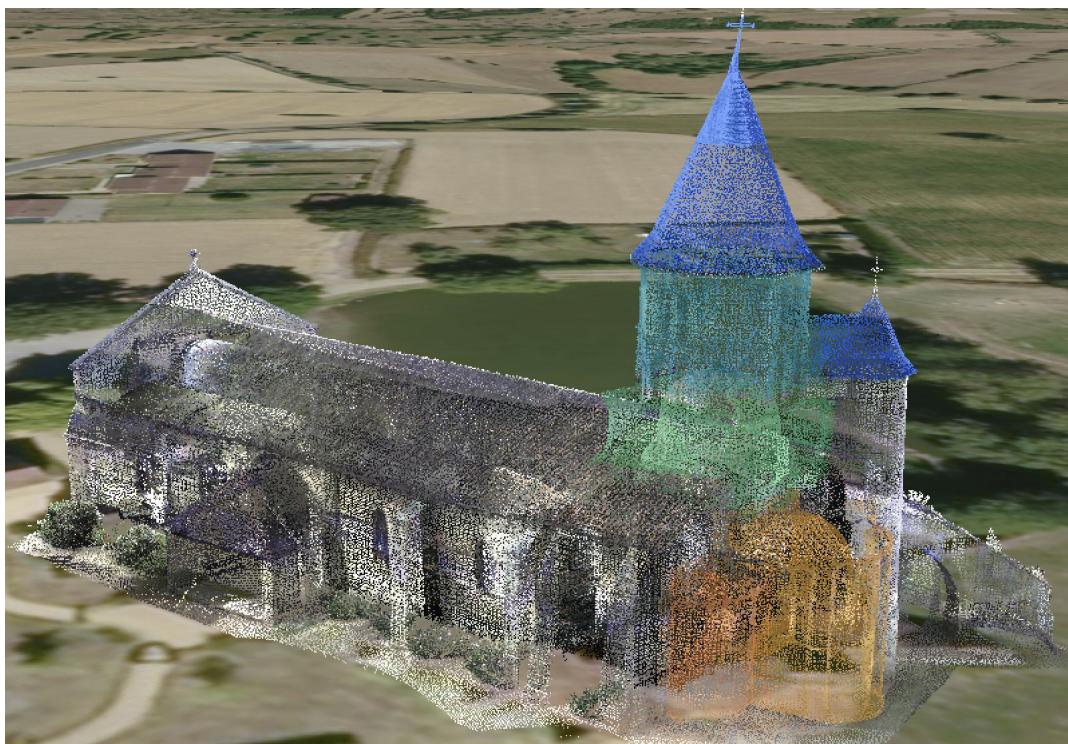


图 2.4.3-点云数据

在第二章数据结构基础中，我们只需要认识和了解这些三维数据的格式和样貌，有兴趣的同学可以多了解了解这些数据的生产过程，在实际的开发过程中，我们不太关心这些数据的来源，只需要学会如何加载以及操作这些数据即可，关于这一点，我们会在第七章给大家详细讲解。

## 第三章：地图学知识基础

### 第 1 节：地图核心要素

一幅地图的组成要素有哪些？我们先来根据经验想象一下，最主要的地图本身肯定是要素之一了。除此之外还会有地图名称（**图名**），有的时候还需要一些文字性的东西来描述地图中的一些图层内容，我们称之为**图例**，为了让读者清楚的知道地图当前表示的内容和实际内容之间的比例大小关系，我们还会加上**比例尺**，为了让用户区分地图的东南西北朝向我们还可能会加上**指北针**。这些都是基本的地图要素。一副标准的地图如下：



图 3.1.1-地图组成要素示意图

因为是显示在浏览器上的电子地图，我们还要对地图中的常用参数有一个了解，这样在后续开发中才能理清头绪。地图是可以放大缩小的，那么我们衡量地图放大和缩小的程度有个关键参数叫做 `zoom`，这是你在任何框架中都会见到的，`zoom` 用于表示缩放层级，通常取值为 0-22 之间，值越大代表着当前地图被放大的越大。通常我们设定 `zoom` 的值就可以改变地图的显示分辨率，这就好比相机的焦距一样，通过调整放大倍数来拍摄到清晰的照片。我们距离地图最远的时候，地图不能再被缩小的那个层级是第 0 级，每放大一次 `zoom` 就增加 1，地图放到最大的时候一般是 22，当然最大的 `zoom` 根据具体情况而定，取决于地图的清晰程度。

除了缩放层级之外，还有一个边界范围的概念，地图控制定位的方式一共有 2 种，第一是确定一个中心点——`center`，这个概念太基础了我就不介绍了，`center` 的取值通常是一个数组，即经纬度数组（如 `[120.43453453, 29.435345]`），使用中心点来确定地图位置通常需要配合 `zoom` 一起，这一点我们在第五、六两章会看到地图初始化时的定位方式。第二种定位的方式是使用 `extent`，`extent` 的意思是范围，是一个矩形的范围，当你确定好之后，地图会根据范围的经纬度差距来自动换算当前的 `zoom` 和 `center`。这个道理很简单，比如你告诉我一个矩形范围是 `[120.234234, 20.3242342, 122.453453, 23.5446]`，我是可

以知道当前的矩形的中心点，而且根据两个经度的差值还可以确定大致相距多远，进而能换算出当前地图大概到了什么级别的缩放。因此这种方式更适合于定位图层，比如加载一个行政区或者一些点位等业务图层之后，采用这种方式会更加准确的让图层适配到屏幕范围。

## 第 2 节：坐标系与投影

坐标的概念我们在预备知识一章稍微提过一些。我们之前讲了经纬度和坐标的概念。其实坐标系就是指一种以某几个参数作为标准的一套坐标体系。例如我们的平面直角坐标系，就是指在一个平面上，由 x 轴和 y 轴组成的具有正负方向的二维平面坐标系。这个坐标系就能够表示一个平面上的任何一点。那么我们的地理坐标系也是如此。就是在地球的球面上以某些参数（长轴，短轴，扁率，偏心率等我们后面会讲到）所构建的一套能够表示地球上任意一个位置的坐标系。

在地理学中，坐标系分为**地理坐标系**和**投影坐标系**。

地理坐标系又称为**经纬度坐标系**。顾名思义，地理坐标系的核心就是用经纬度来表示位置。不管是在球面上还是把球面投影到平面上，如果你想用地理坐标系来表示一幅地图的任意位置，那这个位置的表示方式一定是经纬度。反之，如果一幅地图的单位以经纬度来表示，则这幅地图的坐标系一定是经纬度坐标系。

**投影坐标系**是指基于地理坐标系进行的一些投影变换，投影坐标系的单位通常是米 (m) 或者千米，即距离长度为单位。投影坐标系的诞生是为了满足某些特殊的地理需求，例如**墨卡托投影**，就是为了方便航海家们确认方向。每个地理坐标系都可以通过数据计算和变换产生出很多的投影坐标系。例如基于 wgs84 坐标系（地理坐标系，第 3 节会讲到）的 web 墨卡托投影，基于我国 GCGS2000 坐标系（第 3 节会讲到）的各类地方坐标系，都是投影坐标系。

一般的地图厂商都会提供两种坐标系的地图，我们用国内最常用的天地图为例，天地图提供的经纬度坐标系的地图以\_c 关键字出现在 url 中，而投影坐标系以\_w 坐标系出现，\_w 表示的就是 web 墨卡托投影坐标系。下面两幅图分别是 wgs84 坐标系的天地图和 web 墨卡托投影的天地图：

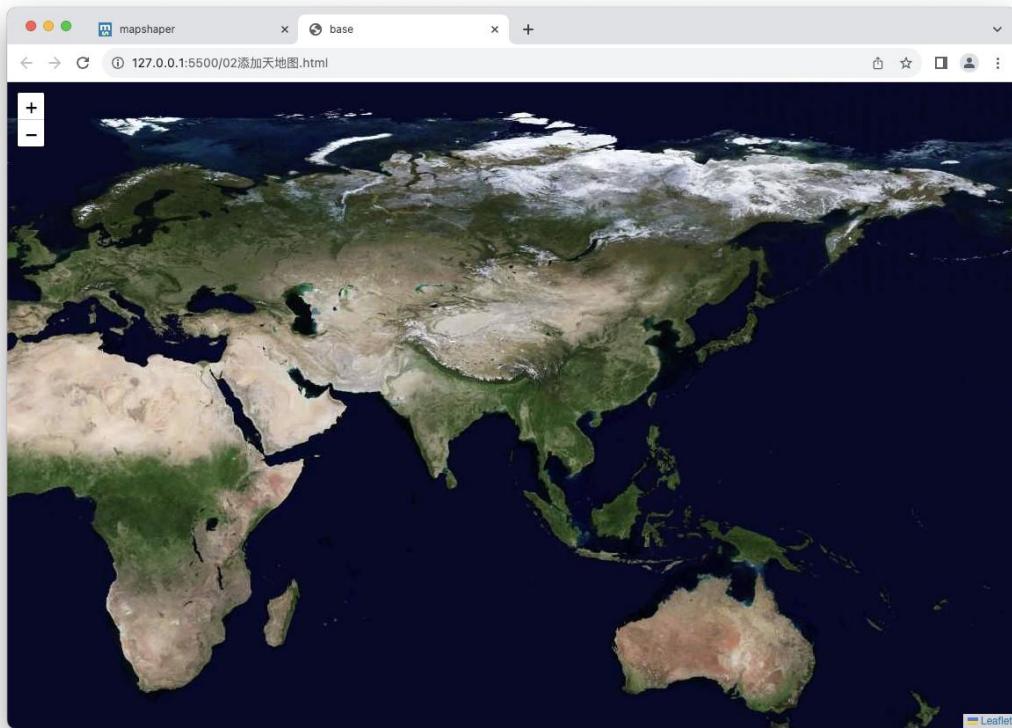


图 3.2.1-wgs84 坐标系天地图

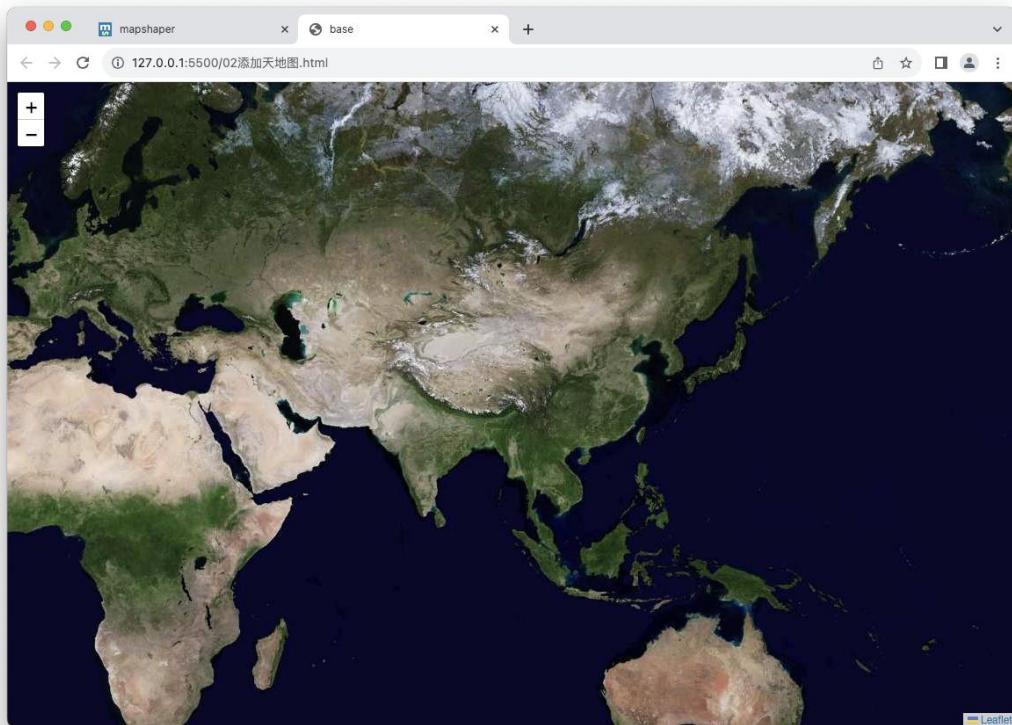


图 3.2.2-web 墨卡托坐标系天地图

区分二者的方法也很简单，经纬度坐标系的地图通常略微扁一些，靠近两极的地方横向拉伸严重，而 web 墨卡托投影的特点是竖直方向被拉伸的很明显，同时整个地图的轮廓是正方形，而 wgs84 坐标系的轮廓是长方形。

## 第 3 节：常用坐标系及转换

在国内，常用的地理坐标系有两个，分别是 **wgs84 坐标系** 和 **GCGS2000 坐标系**。**wgs84** 坐标系是美国在 1984 年综合了卫星数据，地面测绘数据，以及各类实验室传感器的数据等等最后计算出了地球的真实长半轴、短半轴、偏心率、扁率、等参数之后，建立出的一套球面地理坐标系，史称 **wgs84 坐标系**。

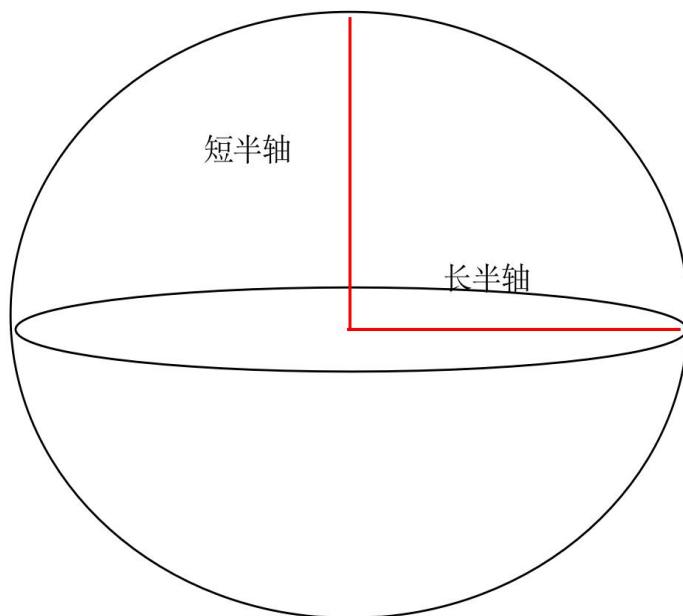


图 3.3.1-长半轴与短半轴

**wgs84** 坐标系直到今天还仍然是世界范围内应用最广泛的地理坐标系之一。因此掌握 **wgs84** 坐标系是必须的。**GCGS2000** 坐标系是我国于 2000 年发布的经过我国地理学家和实验室测算的长半轴，短半轴，扁率，偏心率等数据以后，汇总而成的一套地理坐标系，史称 **GCGS2000 坐标系**。我们的 **2000 坐标系**使用的范围也很广，也是世界范围内使用最广泛的地理坐标系之一。

那么，**GCGS2000 坐标系**和 **wgs84 坐标系**有什么区别呢？首先最大的区别就是两个坐标系所测算的短半轴的数据是不一样的。因此会造成一个问题，那就是两者所描绘的地球的

扁率是不一样的。

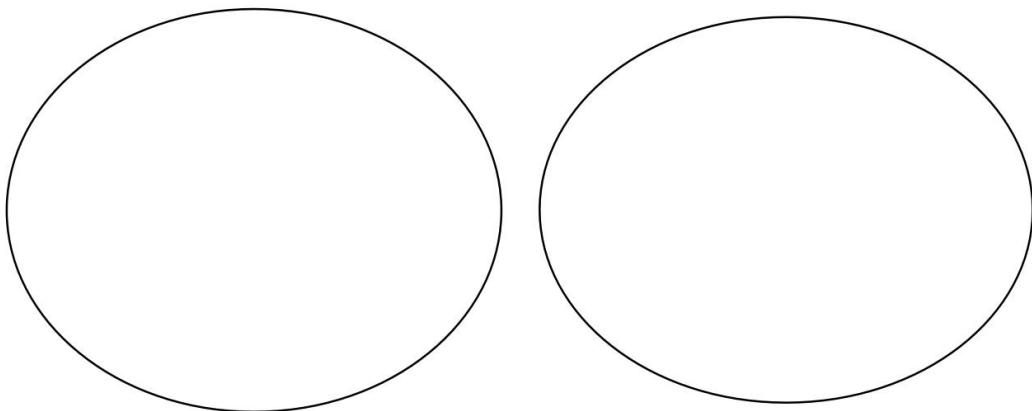


图 3.3.2-短半轴差异

没错，就是你看到的上面有着细微差别的类似于两颗鸡蛋一样的模型。就是一个扁一点另一个没有那么扁，你要问我具体是哪个更扁一点，我也记不住，你可以自行百度，或者你也可以不用百度因为也没那个必要，你只需要知道，这两个坐标系在国内被认为是一致的，就是在经纬度的表示和计算上面视两者相同。因为二者实际上只在高纬度地区有细微的差别，这个差别都是毫米级别的，所以在国内（我国大部分国土不属于高纬度地区）基本上就不存在这个差别，因此我们认为：**在国内，wgs84 坐标系表示的经纬度和 CGCS2000 表示的经纬度是等同的。**

国内除了这两种坐标系以外，还有一些常见的企业级别的坐标系，即高德的 GCJ02 坐标系和百度的 bd09 坐标系，这两种坐标系都是以我们上文中提及的经纬度坐标系为基础，进行一些算法上的加密和偏移而生成的。所以在日常的开发中如果直接获取到这些商业坐标系是不能直接使用的，需要经过坐标系的转换，将百度坐标系和高德坐标系转换成 wgs84 坐标系才能够正常的显示位置，具体的坐标之间的转换方法我放在本书附带的源码文件夹中了，大家可以阅读使用。



我们常用的投影坐标系有很多，最主要的是 **web 墨卡托投影坐标系**，这个坐标系是经过 **wgs84** 坐标系投影而来的，它更方便与展示 **web** 端的平面地图，因此很火很盛行。另外就是我们国内各个地方可能基于我们的 **2000** 坐标系进行了各种转换得到了我们的地方坐标系，这个就具体讨论了，因为基于一个地理坐标系可以投出无数个投影坐标系。关于 **web 墨卡托** 和 **经纬度坐标系** 之间的转换网上有一大堆办法，首先像 **arcgis**, **qgis** 这样的软件可以转换，另外一个工具库例如 **geotools**、**proj4** 等很多语言都有其坐标转换的框架，**github** 上面一搜一大把，如果你懒得搜，OK 我在本书附带的代码文件里已经给你准备好了二者互转的代码，你直接复制粘贴可使用。涉及到国内各个地方坐标系和 **GCGS2000** 或者 **WGS84** 坐标系之间的转换，建议大家使用 **arcmap** 这款软件，这样是最方便快捷的。

在实际的开发过程中，还需要大家了解一个知识，那就是 **EPSG** 代码。首先 **EPSG** 是欧洲一个石油组织的简称，这个组织提出了一种规范，用代号的形式来表示坐标系。比如我们上文中讲到的 **wgs84** 坐标系，**GCGS2000** 坐标系以及 **web 墨卡托坐标系** 都是有其自己的代号的，我们平时开发的过程中也是使用这个代号来表示相应的坐标系。比如 **wgs84** 坐标系我们会表示成 **EPSG:4326**。**GCGS2000** 坐标系我们会表示成 **EPSG:4490**。**web 墨卡托坐标系** 我们会表示成 **EPSG:900913** 或者 **EPSG:3857** 等等。所以当各位在开发的过程中遇到了这些代号，需要做一个转换，清楚其要表示的坐标系即可。这一点我们在后续章节中也会有体现。

# 第四章：底层渲染原理

## 第 1 节：Dom 渲染方式

这一章是要从全局从底层去了解一下我们的 WebGIS 地图是怎样渲染的。为什么要直接讲述底层性的原理性的东西，因为只有从底层和原理上把握一门知识和一门学问才不会出现方向上的错误和偏差，才知道后期应该怎样走出新的道路。这就是为什么我们很多制造行业在认真的学习原理性的东西由制造行业慢慢转换为研发行业。从 made in China 变成 created in China。

在之前 GIS 数据基础一章我们详细的讨论过 GIS 的一些数据类型，包括栅格数据和矢量数据两大类。栅格数据主要包含 tiff 影像和一些 png/jpg 格式的图片。矢量数据主要包括 shape 数据和一些 geojson 数据。那么我们的网页该如何渲染这两种数据类型呢？首先我们要清楚一个网页如何展示一张图片，毫无疑问我们采用 img 标签对吧

```

```

很简单的一句代码就能够展示一张图片。那么我们的栅格数据不就是图片吗，即使是 tiff 影像也可以被转换成图片啊，所以我们使用 img 标签肯定能够展示栅格数据咯。

再说说矢量数据，说矢量数据格式之前呢先要熟悉网页如何展示一个图形，或者我们平常看到的那些图标，是怎么展示的？有的图标可以展示在各个终端，手机上，平板上，电脑上，都没有任何问题，而且清晰度也没发生任何变化，这就用到我们的 svg 标签，svg 标签能够展示一段矢量图形，这个图形是可伸缩的，也就是意味着你对这个图形进行放大缩小操作它是不会发生模糊失真等现象的。



图 4.1.1-svg 图标

所以我们的矢量数据也是图形，也可以转换成 `svg` 标签进行渲染。OK! 到此我们已经清楚了这两种数据的渲染方式，因此我们的 WebGIS 渲染地图的**第一种底层原理就是 dom 方式**。`dom` 指的是我们**使用 html 原生的标签来展示我们的地图**。简单来讲就是使用 `svg` 标签渲染矢量图形，使用 `img` 标签渲染栅格图像。我们的 WebGIS 前端有很多框架，在后面的章节中我们也会介绍使用，在这里可以提前跟大家透露一下，我们下一章要讲的 `leaflet` 框架就是采用了这种原理：

下面要给大家展示一些证据：



图 4.1.2-leaflet 使用 img 标签渲染栅格数据

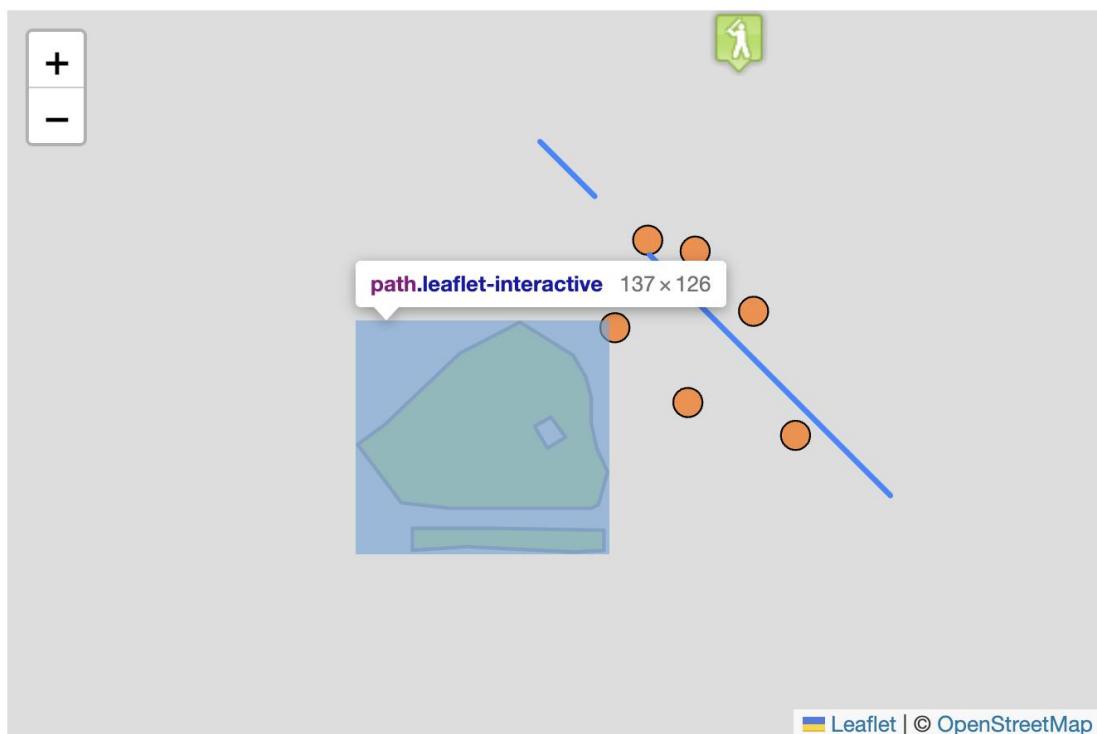


图 4.1.3-leaflet 使用 svg 标签渲染矢量数据

img 标签很好理解就是<img src=''/>, path 其实是 svg 标签的内容, 大家可以随便找一个 svg 图标, 然后用 vscode 打开就可以查看其代码。

```
<?xml version="1.0" encoding="UTF-8"?><svg
width="20"
height="20"
viewBox="0 0 48 48"
fill="none"
xmlns="http://www.w3.org/2000/svg"
>
<path
d="M10 6C10 4.89543 10.8954 4 12 4H36C37.1046 4 38 4.89543 38 6V44L31
39L24 44L17 39L10 44V6Z"
fill="none"
stroke="#6b6b6b"
stroke-width="4"
stroke-linecap="round"
stroke-linejoin="round"
/>
</svg>
```

所以请同学们知晓，普通的 `dom` 元素就可以渲染我们的地图的。但是因为普通 `dom` 元素开发渲染地图难度比较大，而且渲染效率并不高，因此我们在实际的开发中也不会自己手动去使用原生的 `dom` 元素进行渲染。

## 第 2 节：canvas 渲染方式

我们再来说另一种渲染地图的方式，使用 `canvas` 来渲染地图。先说清楚什么是 `canvas`。  
`canvas` 其实就是一个绘图画布。我们的计算机上都会有相应的绘画软件，支持我们绘制各种图形和图像。比如 `windows` 系统里面的画图软件和 `Mac` 系统里的 `无边际` 等软件，`canvas` 就是一个网页上的绘图画布，你可以在其中绘制各种各样的图形和图像。那当然也能够绘制我们的地图了（具体细节我们在第一章也有讲过）。事实上现在绝大部分的网页端处理图像，

修整图片的网站都脱离不了 canvas。canvas 也是一个 dom 元素，只不过它下面不会再有子元素了，也就是说对于 canvas 的操作是封闭的，你无法像普通的 dom 元素一样去给 canvas 里面的元素再写 css 等属性。举个例子你使用 canvas 在网页上绘制一条线可以这样写

```
<!DOCTYPE html>

<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport"
      content="width=device-width,initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <canvas width="600" height="600"></canvas>
  </body>
  <script>
    const canvas = document.querySelector("canvas");
    const ctx = canvas.getContext("2d");
    ctx.beginPath();
    ctx.moveTo(100, 100);
    ctx.lineTo(400, 400);
    ctx.strokeStyle = "red";
    ctx.stroke();
    ctx.closePath();
  </script>
</html>
```

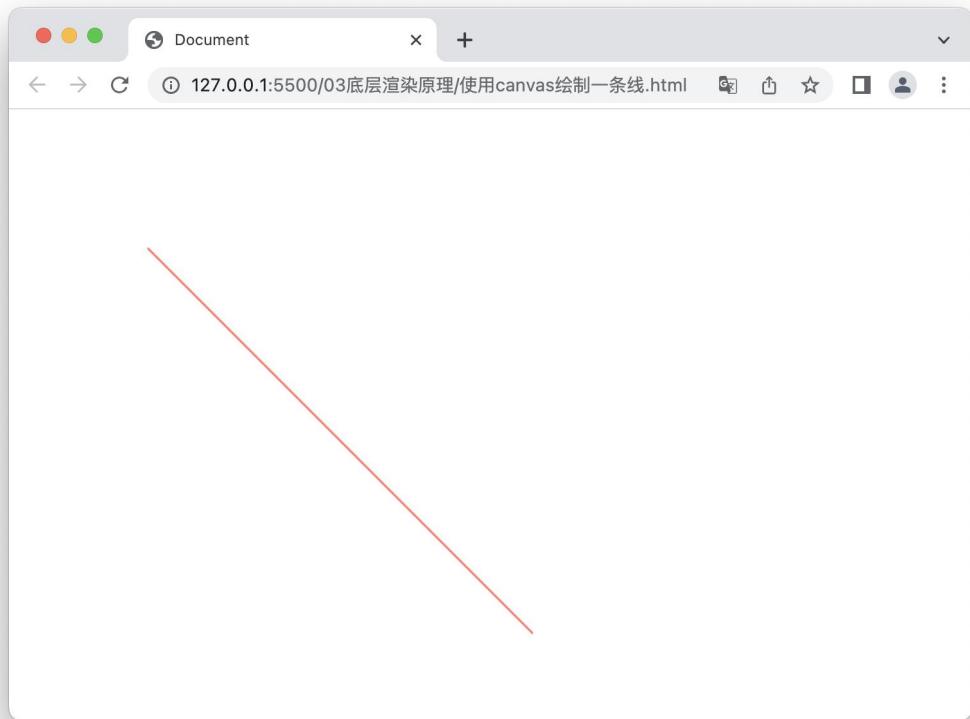
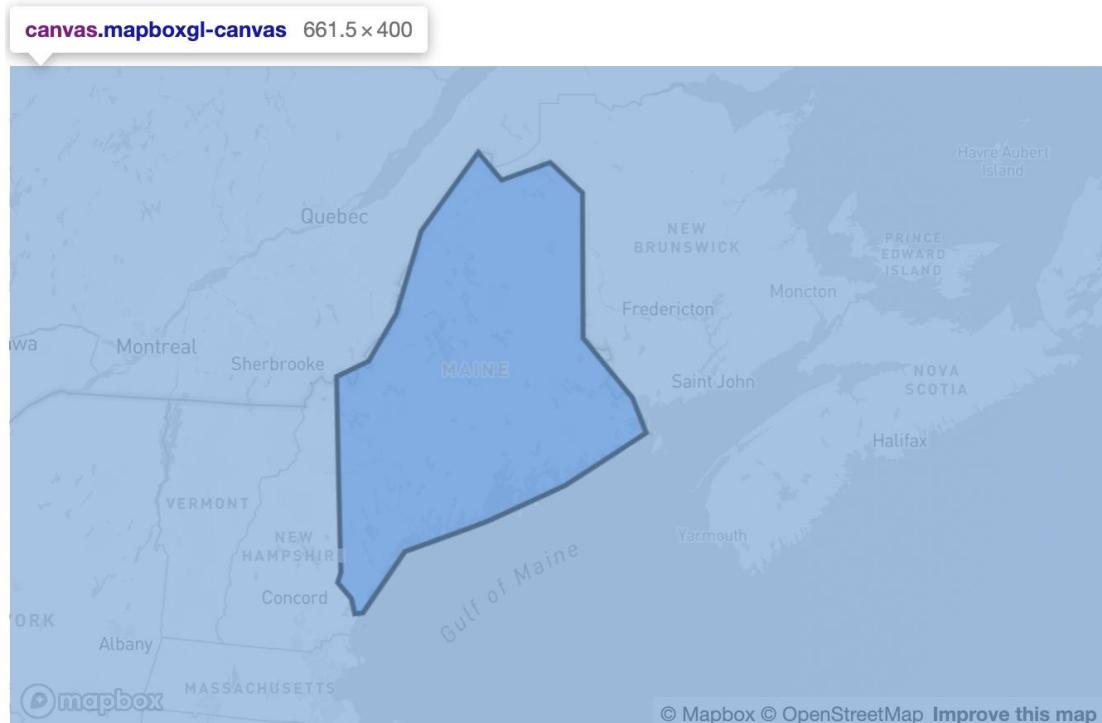


图 4.2.1-canvas 绘制线要素

怎么样是不是非常的简单？所以我们的地图的两大数据类型栅格数据和矢量数据将来也可以通过 `canvas` 渲染在网页上，这也是我们 `mapbox` 框架（在第 6 章会讲到）所使用的底层原理。



#### 图 4.2.2-mapbox 使用 canvas 渲染地图

我们了解了底层渲染原理并不意味着我们就要用这些方式去手动的渲染地图。一是麻烦，二是没必要。编程的最大魅力其实就是解决重复性劳动，业界已经有很多大佬写了很多成熟好用的工具框架，我们不妨就直接用一用，你不用显得它没有价值。不过用归用，可别沉迷于其中，什么都依赖框架。因为总有框架解决不了的问题，需要你自己动脑解决的问题，到那个时候，你必然就要用到我们的底层原理了。

我们之所以要告诉大家地图在浏览器上的渲染方式有两种是因为这两种方式对应着不同的使用场景。这涉及到技术选型的问题。例如 `leaflet` 是使用 `dom` 来渲染地图，这意味着不可大量的滥用 `dom` 元素，这意味着 `wmts` 的地图瓦片将会渲染成很多的 `img` 标签，因此 `leaflet` 它只适合中小型应用。更适合移动端的应用。再换个层面思考，为何以 `canvas` 原理渲染的地图就适合大型复杂应用？那是因为 `canvas` 的绘图能力和渲染能力到远超 `img` 标签，`svg` 标签。但是 `canvas` 在移动端的浏览器上兼容性还有待推敲。这也是很多体量比较大的栅格数据在移动端浏览器上显示不正常的原因。

## 第 3 节：图层渲染原理

要彻底搞清楚这两种渲染方式的不同，恐怕我们还得对 `dom` 和 `canvas` 有一个深入的了解。我们在这里称的 `canvas` 其实也属于 `dom` 元素之一，但是我们在 `gis` 领域所称的 `dom` 元素通常不包含 `canvas`。这二者本质上有什么区别呢？我在这里用一段话总结给大家  
**canvas 相比与 img 等标签元素有着更加复杂丰富的图形图像渲染能力，并且也有着更高的效率，但是这也意味着需要更加复杂的代码开发和更高更深层次的理解。反之 img 等普通的 dom 标签元素有着更加简单的操作和更加轻量的渲染方式。**

我们举一个具体的例子来说，比如现在我们需要渲染一些切片（瓦片）地图，我们如果使用 `img` 标签就需要使用循环来建立多个 `img` 标签，每个 `img` 标签渲染其中的一副图片。而 `canvas` 就是一个整体，你需要把所有的瓦片都渲染到一个 `canvas` 内部，在内部你可以通过不断的 `drawImage` 来渲染每个瓦片地图。

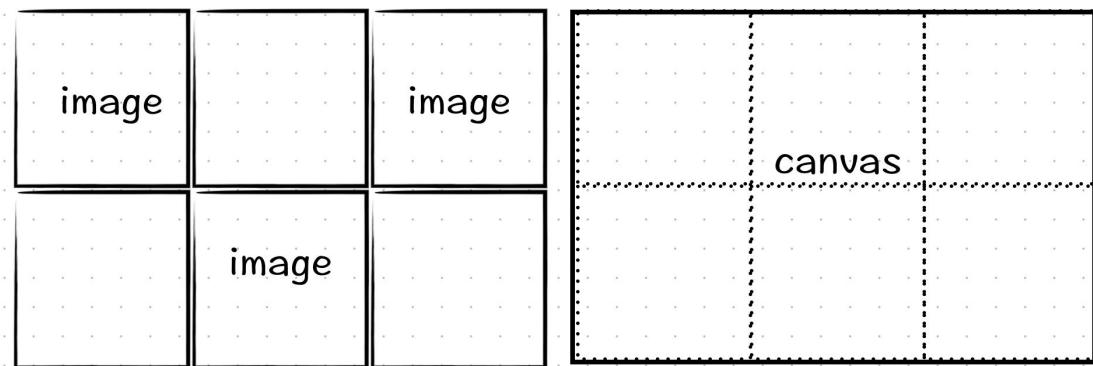


图 4.3.1-dom 与 canvas 对比

无论是使用 `dom` 元素来渲染地图。或者是使用 `canvas` 来渲染地图，其原理都是一样的，只不过是采用的技术手段不同而已，通常一个框架的渲染过程都遵循以下的流程：

1. 确定挂载容器
2. 以浏览器屏幕的实际像素尺寸与地理坐标范围做映射
3. 根据挂载容器的实际尺寸换算需要展示的地图范围
4. 根据栅格图片的四个角的坐标范围来确定栅格图层的渲染位置
5. 根据矢量数据的边界坐标范围来确定矢量数据的渲染位置

如果各位分析过这些框架的底层源码就知道，其实手写一个 WebGIS 框架最难的部分就在于地理学的相关单位和真实的计算机位置的换算关系。比如现在你清楚世界的经纬度范围是  $[-180, -90, 180, 90]$ ，可能你还清楚地球的半径是 6378.137km，那么如何将这个平面地图渲染在浏览器上。并且知道每个经纬度所对应的屏幕位置呢？这确实不是一件容易的事，也不适合在这本书里讲这个问题，但是大家要明确。WebGIS 图层渲染的根本原理就是以矩形的经纬度（坐标）范围来确定图像或者矢量图形的位置。无论是栅格数据还是矢量数据，统统都是如此。想要加载图层，必须首先搞清楚图层的最西边的经度，最东边的经度，以及最北边的纬度和最南边的纬度，四个边界的经纬度信息。只有知道了图层所处的位置信息，才能够精准的加载图层。这一点是大家要牢记的，在今后的开发中也是经常用到的，希望大家重视。

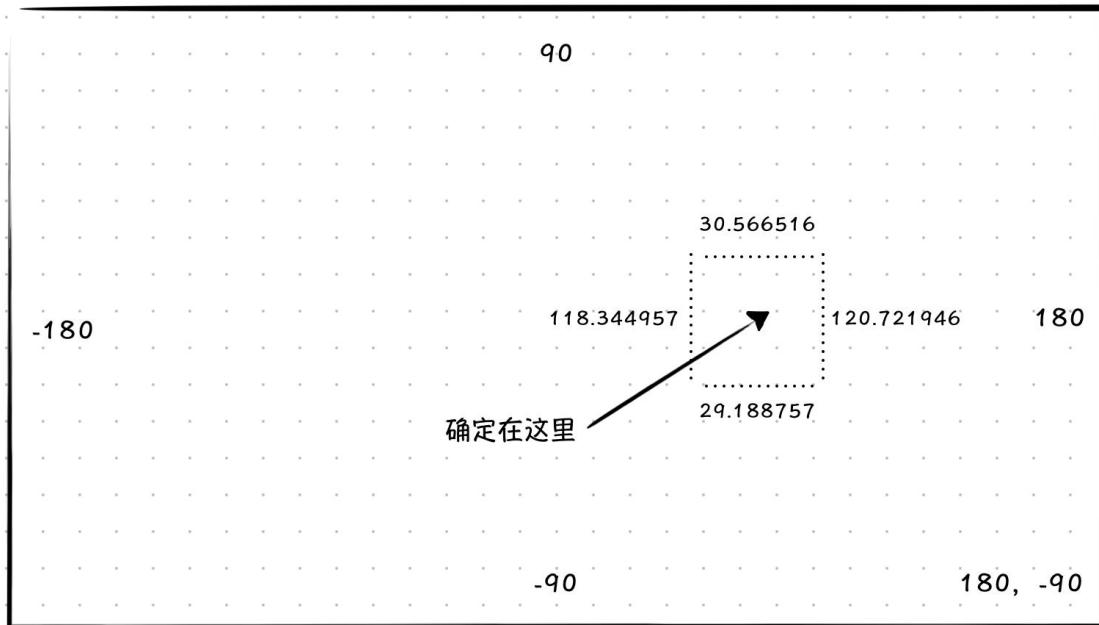


图 4.3.2-图层渲染原理

## 第 4 节：框架与技术选型

目前我们常用的 WebGIS 前端渲染框架一共有 5 个，分别是 leaflet、openlayers、mapbox、cesium 和 maptalks。这几个框架没有好坏之分，都有各自的擅长领域，我们在学习的时候不必全部学会，只需要学会其中 2-3 个即可。还有小伙伴会问什么不说 arcgis API for JavaScript。这个框架有很多弊端：首先它并非完全开源，也不完全支持现有的开源服务和数据，其次是 esri 有自己的数据格式和服务规则，比如我们之前学过的 geojson，在 esri 的生态体系中会有 esrijson。一些后续我们要用到的 WFS 以及 WMTS 服务它也有自己的方式。因此这个框架不太好与开源的数据结构所兼容。因此在进行平台对接等开发时，会有很大困难，目前也处在下降和被淘汰的趋势，在这里就不给大家讲解了。

各个框架的设计思路虽然不同，但是其使用逻辑开发流程都相差不大（本质上其实都受限于 js 这门语言的设计思路）因此无论哪个框架我们在使用的时候都会采用面线对象的编程思想，比如上述的 5 个框架各位在初始化地图的时候都需要 new 一个 map 对象（实例）来进行页面容器的绑定和一些中心点、缩放层级等参数的绑定。在使用的过程中也是接触到各种类和方法来进行 API 的调用。

接下来我们详细介绍一下每个框架的特点以及他们之间的对比。

首先，leaflet 是一个特别适合移动端 gis 开发的框架，我们在本章的前两节也讲过采用

**dom** 渲染方式的框架只要不是滥用 **dom**, 兼容性都要好过 **canvas**。另外 **leaflet** 特别的小而且轻量，很适合初学者上手。

**mapbox** 和 **openlayers** 其实是 **canvas 2D** 的典型代表了。这两个框架很相似，互相都能完成绝大多数的日常需求（国内）。非要讲细微的差别，主要有以下的不同：

1. **mapbox** 其实本质上使用的是 **canvas webgl** 模式，因此支持 **3D** 渲染。而 **openlayers** 是纯 **2D** 的。

2. **mapbox** 所渲染的地图相对来讲更加优美，样式细节更加丰富。**openlayers** 渲染的样式比较基础。

3. **openlayers** 更加能够照顾开源开发者，因为 **openlayers** 本身就是开源的，因此社区以及官网有丰富的例子，支持各种开源的数据类型和服务。支持的数据格式也相对比较多。这方面 **mapbox** 相对来讲就比较闭塞了。举个例子 **mapbox** 现如今支持的矢量数据类型只有 **geojson**。而 **openlayers** 支持 **wkt**、**kml**、**gml** 等在内的多种矢量数据格式。**mapbox** 完整意义上并不算开源，因为很多地图服务和样式文件都是依赖其官网的。只有完全摆脱掉这些东西才算做开源。另外就是让人厌烦的 **token**。

**cesium** 是 **3D** 开发的框架，而且主要集中的是微观上的细节上的东西。我们说基于 **cesium** 开发的 **app** 基本上都是细粒度的。精确到某几栋楼，某厂房等等。换句话说，如果你使用 **cesium** 想要展示全国的 **POI**，那简直是多此一举，毫无意义。

**maptalks** 是国内大神开发的 **gis** 框架，它同时支持 **2**、**3D**。虽然是国内大神编写的开源的地图框架，**maptalks** 做到了将 **openlayer** 和 **mapbox** 相糅合，一方面将地图展示为 **2**、**3** 维一体化，另一方面又要保证地图有足够的用户编辑权限，允许用户进行各式各样的矢量边界，图层加载等。并且 **maptalks** 也是开源的，**api** 文档写的也挺详细。

**maptalks** 的缺点在于稳定性相对来说不是很好，性能上与其他几个框架相比也稍微有点差，毕竟国内开源的，而且还是由个人编写的，背后没有资本的注入，确实没办法将框架有质的提升，不过对于新手练习，简单地图应用的构建还是完全够用的。

以下是几个 **gis** 框架的多方面对比，仅供大家参考：

框架	代码体积	运行效率	渲染效果	支持 3D	上手难度
openlayers	中等	较高	好	不支持	较难
mapbox	较大	较高	好	支持	一般
leaflet	较小	较高	好	不支持	容易
maptalks	中等	一般	较好	支持	一般
cesium	较大	较差	一般	支持	较难

表 4-4-1:主流框架比较

上面 5 种框架的基本组成也是相似的，地图核心类，图层类，编辑和控制，交互，数据和服务的加载等等，基本上都是围绕这几个方面进行的。其实如果你足够清楚 webgis 的原理，使用哪个框架其实都大致相似，基本原理都是一样的。

其实单单从应用的角度讲，上面四种框架你使用哪个都能构建一个基础的可以使用的 GIS 系统。就像编程语言一样，对于框架从来就没有优劣之分，只不过不同的框架所擅长的事情不同。如果你关注的是地图的美，前端渲染的炫酷，强调展示层面，那你选择 mapbox 准没错，如果你的地图应用有很多客户提出的个性化需求，对地图有很多频繁的操作需求，那你不妨试试 ol，如果你要做移动端的 app，可以使用 leaflet，如果你要支持国产，那你也同样可以使用 maptalks。

我们常用的 WebGIS 框架还有很多，不过本书我们只提比较核心的 Leaflet、OpenLayers、Mapbox、Maptalks、Cesium。但是这几个框架中有很多东西都是重复的，也就是同一个需求其实不同框架都能实现而且实现的思路都差不多，所以我们重点挑其中 2 个（leaflet 和 mapbox）跟大家讲解一下，之所以选这两个是因为比较容易入手，对学生比较好一些，大家如果能把这两个框架掌握明白了，相信大家也有能力去自学剩下的其他的框架。openlayers 和 mapbox 其实高度重合。他们两个就像 Android 和 iPhone，vue 和 react，相爱相杀各自有优势，整体上来看却又非常的相似，因此如果你选择学习 openlayers 其实也无妨，代码设计思路以及渲染原理都是很相似的，只不过具体的 api 名字不同，可以参考 mapbox 和 openlayers 的官网。

我们在针对具体的 app 需求时可以根据上述内容做一个技术选型确定。例如现在需要开发一个小程序。并且功能也比较简单的话毫无疑问选择 leaflet，快速而又简单。如果涉及到相对比较复杂的功能，或者页面有很大的变化和操作，可以选择 openlayers 和 mapbox。

在这里还是强烈建议各位**移动端 app 尽可能的使用 leaflet**，因为低版本的 mapbox 在移动端有很大的兼容性问题，很多安卓手机会不支持 mapbox 而导致地图无法渲染。web 端的应用如果是纯二维场景，并且不注重样式美观，可以直接 **openlayers**，如果是特别关心样式的渲染（比如很多大屏项目）可以考虑 **mapbox**。如果是涉及到一点三维场景，那毫无疑问也是 **mapbox**。如果 web 端应用比较注重细节的，场景化的展示（例如建筑、道路、智慧社区等等此类需求）可以考虑 **cesium**。另外还会有相当一部分系统是既要支持二维也要支持三维，也就是所谓的二三维一体化。这时候技术选型就比较推荐 **openlayers+cesium** 了。并且社区也有针对二者结合的各种插件。如果二三维一体化平台的需求比较简单的话，可以考虑只使用 **mapbox**。毕竟现在 **mapbox** 也有能够渲染一部分三维模型的能力。实际上 **cesium** 这个框架虽然强大，但是笔者还是建议能不选就不选，因为这个框架开发过程中遇到的问题太多太细小，就像它框架本身一样，很容易让人崩溃。你会发现在 **cesium** 中稍微写错一点东西就直接渲染错误，崩溃掉，系统崩溃的多了，人也就崩溃了。

## 第五章：Leaflet 精讲

### 第 1 节：基础入门

leaflet 是我用过的最**轻量**的，简单的框架没有之一。

leaflet 官网地址：<https://leafletjs.com/>。

这个框架已经有十几年的历史了，早起的很多国内的大厂都是使用 **leaflet** 来做 **GIS** 方面的开发，直到现在也有很多公司在使用 **leaflet**。这也是为什么本书要讲 **leaflet** 的原因。

**leaflet** 官网有这么一句话：

*an open-source JavaScript library for mobile-friendly interactive maps*  
一个开源的对于移动端交互性地图非常友好的 *JavaScript* 库

官网也说了，**leaflet** 是一个对于移动端交互性地图非常友好的库，简而言之就是对移动端展示比较好。我在这里解释一下为什么对于移动端比较好，这跟我们之前说过的底层原理是密切相关的，因为 **leaflet** 是基于 **dom** 方式渲染的，移动端的浏览器也是支持 **dom** 渲染。并且 **dom** 渲染方式只要你不是滥用 **dom** 不会存在太大的性能问题。然而 **canvas** 就不一样了，移动端的浏览器对于 **canvas** 的支持很考验性能，尤其是一些安卓浏览器在加载体积过大的时候就会出现卡顿，加载影像数据会显示成小黑块等问题。这也是 **canvas** 对于移动端

交互性地图不友好的原因之一，这一点在上一章中也有讲到过。

**leaflet** 框架的组织结构也特别的简单，如图：

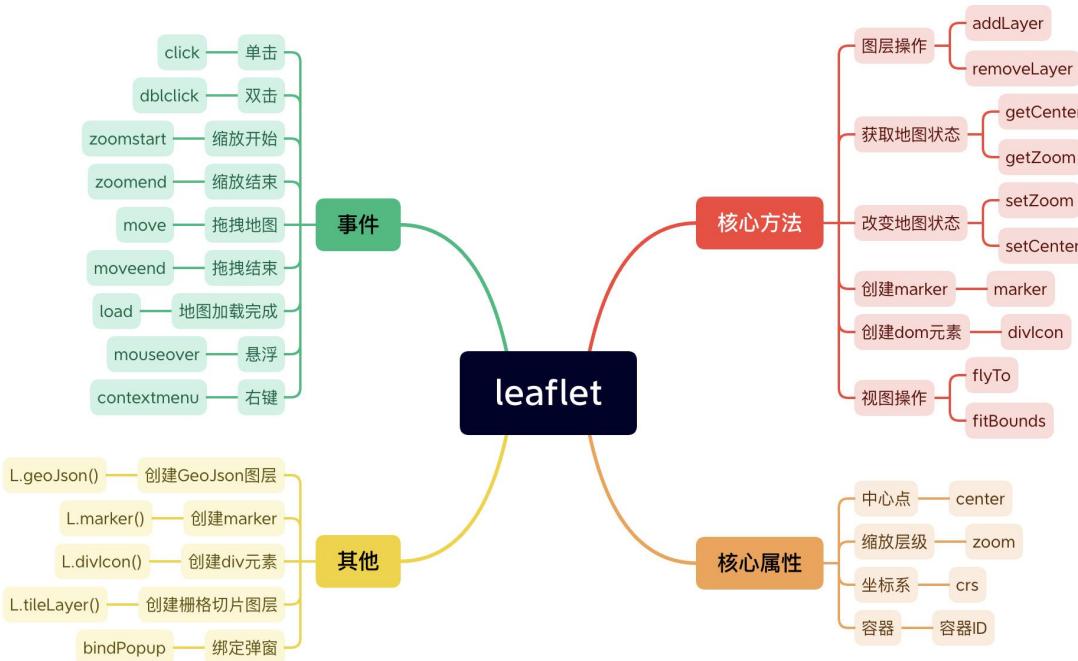


图 5.1.1-leaflet 架构 (简化版)

怎么样是不是特别的简单？开个玩笑。其实任何一个框架都会包含很多的组织结构，只不过看你能否一眼就看到这个框架的骨干部分。上面这幅图只是简单的罗列了 **leaflet** 框架中的一小部分知识体系。其实一副地图的组成就那么几个方面，首先是基础的展示层面，这就涉及到图层的加载和展示。然后是图层的类型，图层的数据源，图层的样式，图层的交互等等，总之一个地图的核心其实就是图层。其次从宏观上说整个地图作为一个可操作的对象也一定是包含很多属性和方法和事件的。就像理解我们的 json 一样，一个地图下面会有很多子元素对象，例如一些属性，一些方法，一些图层，图层又会有自己的子元素比如说图层的样式，图层的类型等等。如果你觉得这些内容不好理解的话，你可以根据功能来划分，你的 **GIS** 应用里需要哪些功能，这些功能需要用到 **leaflet** 里的那些属性和方法或者哪些概念跟你的需求是相关的，这样探索式的学习会减轻很多压力。

关于 **leaflet** 官网的使用技巧，在这里跟大家分享一下。首先官网分为 6 个部分：

即概览、案例、api 文档、下载、插件、博客等。其中比较重要的是案例和文档还有插件三个部分。

**leaflet** 案例部分比较少，只有几个基础的例子，如果大家在开发过程中遇到不会的问题可以先来案例部分查询以下。如果没有答案还是推荐大家查看插件里，插件里不仅有介绍还

有 demo 案例。大家可以仿照着编写代码，或者直接跳进 [github](#) 查阅源码是如何编写的。

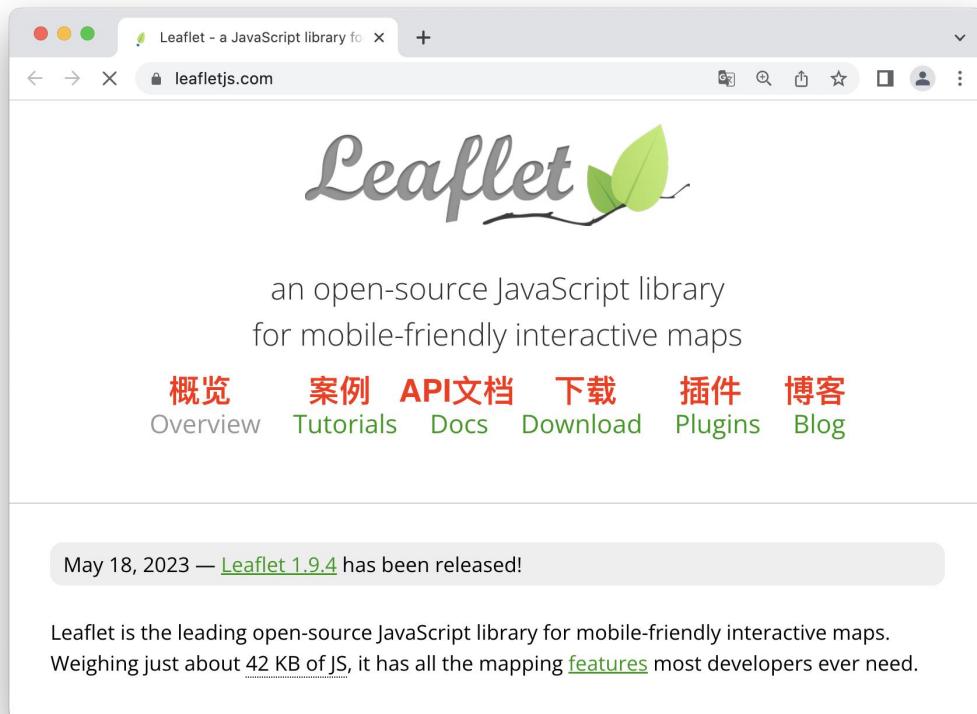


图 5.1.2-leaflet 官网

在讲 `leaflet` 具体的项目代码结构之前需要各位明白一个 js 的基础知识，那就是类的概念，这一点我们在预备知识章节也说到过。

`leaflet` 本身其实就是一个类，这个类的名字叫 L。你没听错就是一个大写的字母 L。这个类身上有很多的方法和属性，其中最重要的一个方法就是 `L.map()`，这个方法就能够初始化完成一个地图。除此之外 `leaflet` 的很多图层也是由这个 L 类的某些方法来调用的，比如说你想添加一个 geojson 图层，需要用到 `L.geojson()` 方法来添加 geojson 数据。比如你想在地图上添加一个标记点，你可以使用 `L.marker()` 方法来添加。总之一句话概括万物皆可 L。

接下来我们就初始化一个 `leaflet` 应用。有两种初始化的方式，如果你打算使用原生的 html, css, js 进行开发，那么你需要进入 `leaflet` 的两个关键的文件和一个静态资源文件夹，你可以从官网上找到 `download` 先把资源文件下载下来，然后只保留其中的 `leaflet.css` 和 `leaflet.js` 两个文件还有一个 `images` 文件夹



图 5.1.3-初始化 leaflet 所需文件

然后新建一个文件夹并且把这三个文件放进文件夹里, 用你的代码编辑器打开你新建的文件夹, 新建一个 html 文件, 名字自己起, 比如就叫 leaflet-init.html.

```

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta http-equiv="X-UA-Compatible" content="IE=edge" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<title>Document</title>
</head>
<body></body>
</html>

```

然后我们引入两个关键的文件:

```

<link rel="stylesheet" href="../leaflet.css" />

<script src="../leaflet.js"></script>

```

然后我们需要准备一个 dom 容器, 通常是一个 div 用来装载地图, 地图将来会挂载到这个 div 内, 然后 div 的尺寸大小等样式就决定了地图的样式。所以我们要给这个 div 写一个 css 样式:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta http-equiv="X-UA-Compatible" content="IE=edge" />
<meta name="viewport" content="width=device-width,
initial-scale=1.0" />
<title>Document</title>
<link rel="stylesheet" href="./leaflet.css" />
<script src="./leaflet.js"></script>
<style>
#map {
  width: 100%;
  height: 100vh;
  position: absolute;
  left: 0;
  top: 0;
}
</style>
</head>
<body>
<div id="map"></div>
</body>
</html>
```

接下来就要初始化我们的 leaflet 代码了。首先要构建一个地图对象 map，然后给这个 map 传入一些配置参数，比如地图的中心点、缩放层级、坐标系、地图所要挂载的容器等。

```
<script>
const map = new L.map("map", {
center: [29.4234234, 120.646456],
zoom: 8,
crs: L.CRS.EPSG4326,
});
</script>
```

如果你看到这样的效果，那就证明你的初始化已经成功了：

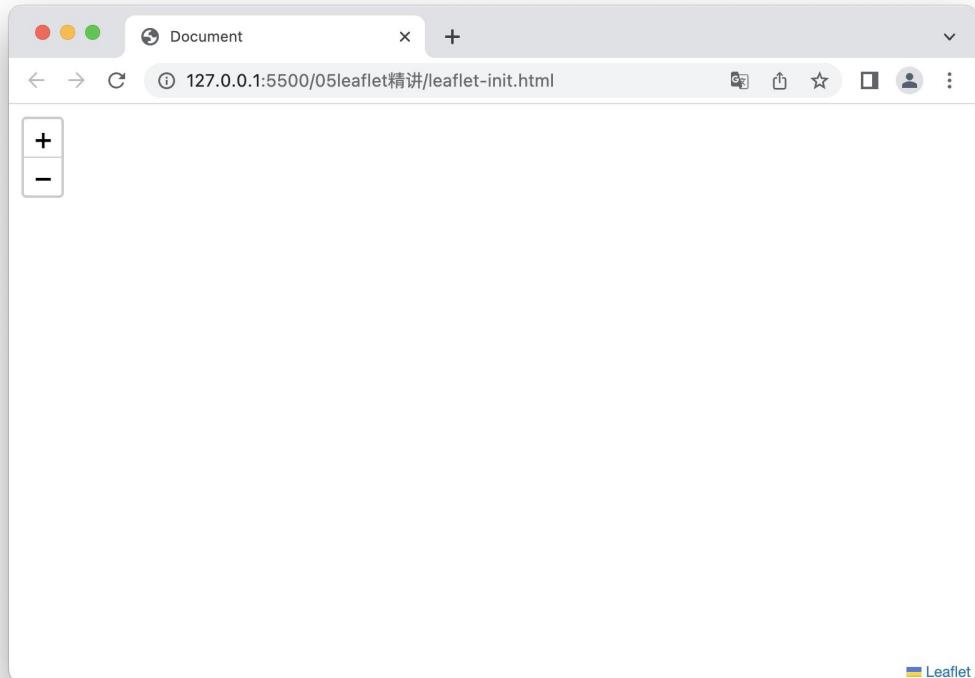


图 5.1.4-初始化成功

对于工程话的项目，比如你使用了 vue、react 等框架构建的前端项目，那么你需要采用 npm 包引入的方式来构建 leaflet。

第一步你需要在项目所在的终端执行命令 `npm install leaflet --save` 这样就可以把 leaflet 的资源包引入到项目的 `node_modules` 里面了。然后在具体使用的时候可以使用 `import` 语句导入 leaflet。

```
import L from "leaflet";
import "leaflet/dist/leaflet.css";
```

然后 js 部分你也可以写和第一种初始化方式一样的代码了，运行项目之后你也会看到相同的结果。这里以 vue 组件为例给大家附上完整代码：

```
<template>
  <div id="map"></div>
</template>
<script>
  import L from "leaflet";
  import "leaflet/dist/leaflet.css";
  export default {
    name: "LeafLetMap",
    data() {
      return {};
    },
    mounted() {
      const map = L.map("map", {
        center: [29.2342342, 120.45345],
        zoom: 9,
        crs: L.CRS.EPSG4326,
      });
    },
  };
</script>
<style scoped>
#map {
  position: absolute;
  width: 100%;
  height: 100vh;
  left: 0;
  top: 0;
}
</style>
```

到此，我们就完成了 leaflet 的初始化工作。后面讲的所有案例我都采用原生 html 的方式讲解吧，一是比较快捷方便，二是对于学生党比较友好（他们可能没学过工程化的搭建），反正原生的对于已经有工作经验的同志们也能看得懂。初始化的代码中有几个需要注意的点，首先你的地图将来所要挂载的容器的样式就决定了地图的样式，所以外部容器 div 的样式需要配合页面其他元素进行组合式布局。另外 leaflet 中的经纬度顺序要特别注意。其他框架中都是先经度后纬度，但是 leaflet 是先纬度后经度。leaflet 支持 2 个坐标系。即 EPSG:4326 和 EPSG:3857（第三章讲过）当然你也可以通过 proj4 等工具转换地图的坐标系。进而加载地方坐标系的数据。这样也是可行的。

## 第 2 节：基础底图

在国内，基础底图的选择相对来说很少，除了使用一些在线的国外的图源（bing 地图、mapbox 官方图层），90% 的场景都需要使用天地图。我们这一小节来使用 leaflet 添加天地图。天地图是我们国家地理信息公共服务平台，平台提供了包含矢量、影像、地形、以及三维等多种类型的服务。我们可以调用其中的某些服务来加载影像图，因为天地图是官方的，是国家标准，因此天地图的影像图和矢量地图被用作大多数 GIS 项目的底图。

天地图数据服务官网：<http://lbs.tianditu.gov.cn/server/MapService.html>

地图服务列表			
缩略图	图层名称	服务地址	投影类型
	矢量底图	http://t0.tianditu.gov.cn/vec_c/wmts?tk=您的密钥	经纬度投影
		http://t0.tianditu.gov.cn/vec_w/wmts?tk=您的密钥	球面墨卡托投影
	矢量注记	http://t0.tianditu.gov.cn/cva_c/wmts?tk=您的密钥	经纬度投影
		http://t0.tianditu.gov.cn/cva_w/wmts?tk=您的密钥	球面墨卡托投影
	影像底图	http://t0.tianditu.gov.cn/img_c/wmts?tk=您的密钥	经纬度投影
		http://t0.tianditu.gov.cn/img_w/wmts?tk=您的密钥	球面墨卡托投影
	影像注记	http://t0.tianditu.gov.cn/cia_c/wmts?tk=您的密钥	经纬度投影

### 图 5.2.1-天地图官网服务

我们只需要根据相关的 url 和标准去访问就可以了。在使用之前各位需要去天地图官网注册登录并且申请一个 token (密钥)，有了这个 token 之后才能使用天地图的服务。

天地图提供的地图服务一般都是同时支持两个坐标系的，一个是 wgs84 经纬度坐标系，另一个是 web 墨卡托坐标系。因此在加载的时候要先确定框架使用的和所支持的坐标系，选择对应的坐标系的地图资源，我们的 leaflet 同时支持两种坐标系，所以我们在里以 wgs84 坐标系为例来介绍一下如何加载天地图。

天地图提供了两种访问方式，一种是 wmts 方式，一种是 xyz 的访问方式（第十一章会讲到）两种方式的基础 url 如下：

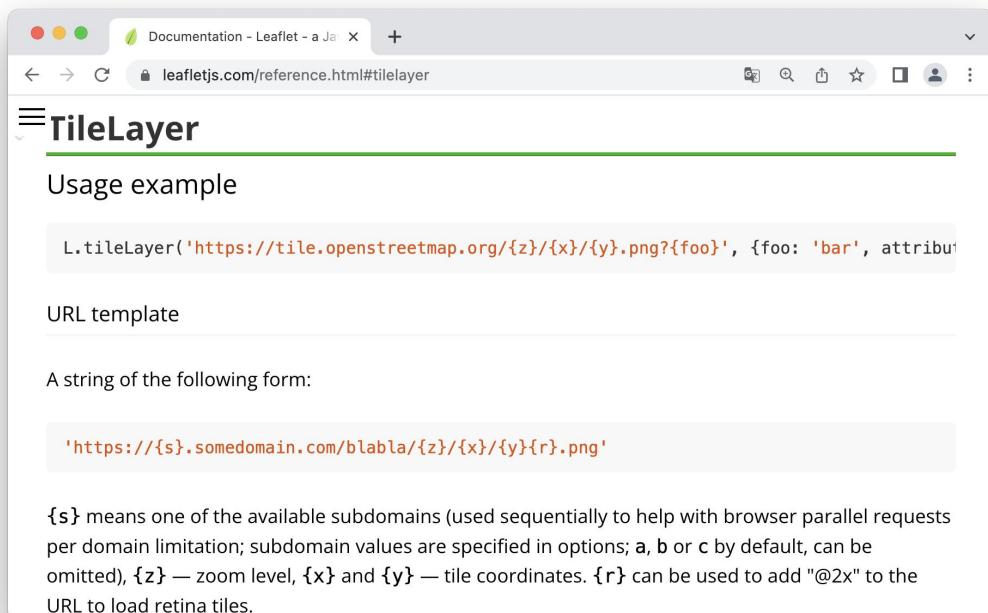
#### xyz 方式：

```
"https://t0.tianditu.gov.cn/DataServer?T=img_c&x={x}&y={y}&l={z}&tk="
```

#### wmts 方式：

```
"http://t0.tianditu.gov.cn/img_c/wmts?SERVICE=WMTS&REQUEST=GetTile&VERSION=1.0.0&LAYER=img&STYLE=default&TILEMATRIXSET=c&FORMAT=.tiles&TILEMATRIX={z}&TILEROW={y}&TILECOL={x}&tk="
```

无论使用哪一种 url 我们都必须使用 L.tileLayer 这个方法来加载，因为他们都属于瓦片图层（第九章第 1 节会详细介绍）因此我们就需要了解一下 L.tileLayer 这个方法的用法：



## 图 5.2.2-tileLayer 用法

第一个参数是一个 url, 剩下的参数是一些配置项, 因此我们可以这样写来添加天地图:

```
const TDT_TOKEN = tdt_token;
//xyz 方式
const tdt_url =
  "https://t0.tianditu.gov.cn/DataServer?T=img_c&x=
  {x}&y={y}&l={z}&tk=";
const layer = L.tileLayer(tdt_url + TDT_TOKEN, {
  zoomOffset: 1, //缩放偏移, 调整 url 里的层级
  tileSize: 256, //地图瓦片的尺寸
  maxZoom: 18, //最大缩放层级
});
layer.addTo(map);
```

解释一下这段代码, 首先我们从天地图官网申请的 token 需要放在代码中, 因为我们的请求地址 url 中是需要 token 的。然后我们调用 L.tileLayer() 这个方法来新建一个 layer 对象, , 传入两个参数, 第一个参数就是请求地址, 第二个参数是瓦片图层的一些配置, 比如切片本身的尺寸大小, 以及最大缩放层级, 那个 zoomOffset 是用来手动调整 url 里的缩放层级的。 url 里的缩放层级体现在哪呢? 大家可以先保留这个问题, 到第十一章的时候自然就明白了。

如果我们需要开发移动端相关的 app, 那么使用天地图矢量风格的地图作为底图可能更加合适, 因此我们只要把 url 里的 img\_w 换成 vec\_w 即可。

最后我们在把这个瓦片图层添加到地图上 layer.addTo(map), 当然你也可以写 map.addLayer(layer)。两者的效果是一样的。完整的源码在本书附带的源码文件夹里。

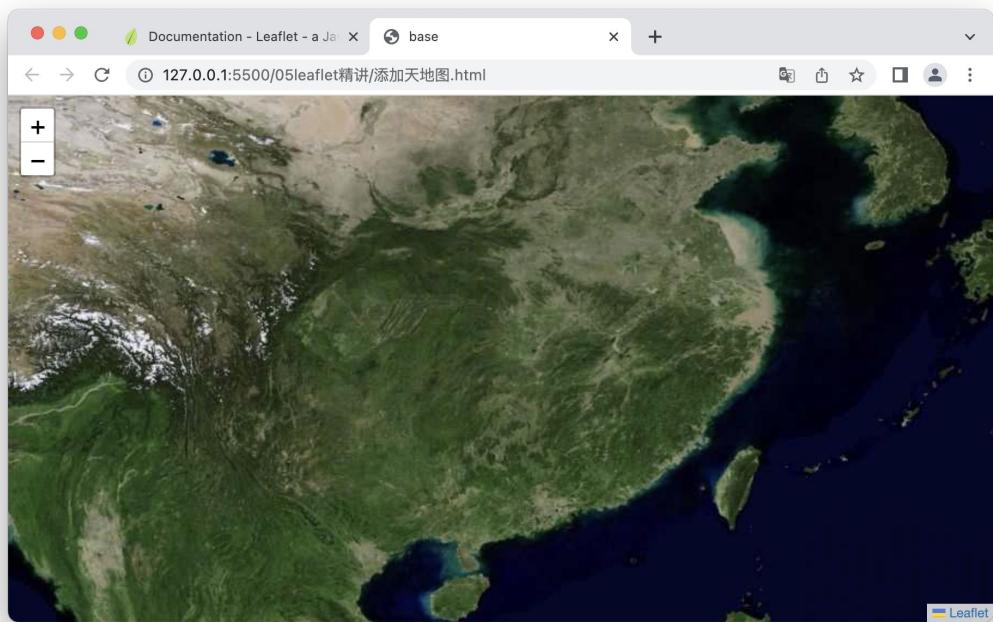
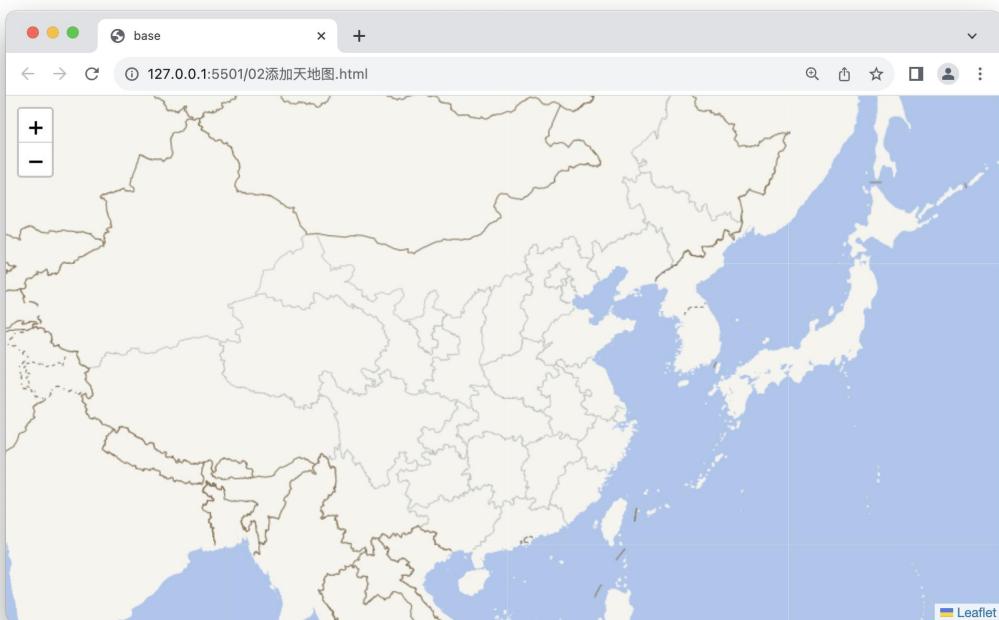


图 5.2.3-leaflet 加载天地图

天地图的风格样式有很多种，如果需要类似于高德百度地图风格的地图，可以选择将参数中的 `img_w` 修改成 `vec_w`，即请求地址为：

```
"https://t0.tianditu.gov.cn/DataServer?T=vec_c&x={x}&y={y}&l={z}&tk=你的token"
```

，这样你就得到了一个矢量风格的底图。

对于 wmts 方式的方式加载大家要格外的注意，一定要引入 leaflet 的 css 文件，不引入的后果就是地图图片是错乱的。另外如果是采用 wgs84 坐标系，一定要记得打开 zoomOffset 参数。

## 第 3 节：矢量图层操作

这一小节来学习一下如何加载矢量图层，这也是很常见的一一个业务场景。比如有的时候需要加载行政区或者是一些地块数据。我们首先给大家准备了一份行政区的数据，包括杭州市以及其下属的区县，格式为 geojson 格式（这个在数据基础章节详细讲过了），在 leaflet 中我们加载 geojson 数据需要用到 L.geoJson() 这个方法，来看下怎么用吧：

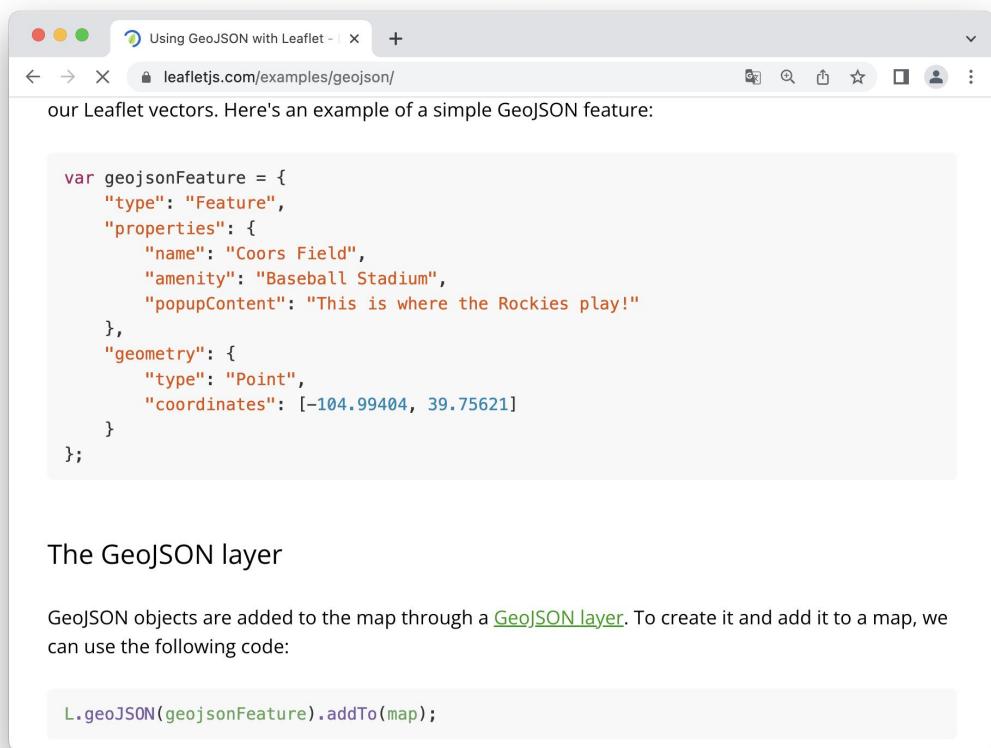


图 5.3.1-leaflet 添加 geojson

官网的案例写起来确实很简单，第一步声明一个 `feature` 对象，第二步直接加载到地图上。我们也来这样写一下：

```
L.geoJson(hangzhou).addTo(map);
```

直接就可以有结果，非常的简单（大家可以查看附带源码）。在实际的项目开发过程中。这份 `geojson` 数据有可能是本地的一个静态的 `json` 文件。也有可能是通过网络请求得到

的 json 字符串。例如 geoserver 的 WFS 服务 (第十一章会介绍)。总之我们实际开发中不需要管这个数据怎么来的，只要拿到了这个数据。就按照静态的 geojson 的方式去加载即可。

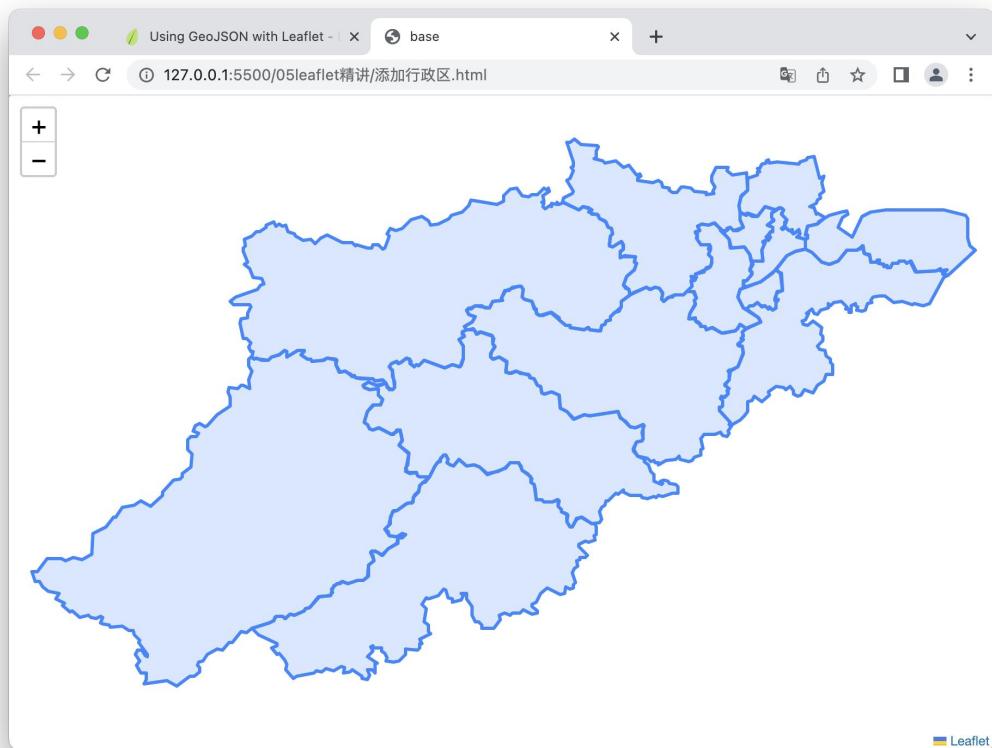


图 5.3.2-leaflet 加载杭州行政区

其实加载一个行政区数据确实很简单。但是需求往往没有这么简单，例如我们现在加载的结果是有问题的，由于地图当前的中心点的位置和杭州行政区的中心点不一致，就导致现在初始的时候的时候杭州行政区并不在地图的正中心，因此我们需要加一句代码来控制行政区处于地图的中心，这就要介绍一下我们的矢量数据图层是如何确定位置的，其实和第二章跟大家讲过的栅格数据的位置确定的方式是一样的，需要给这个多边形确定一个边界经纬度范围。实际上任何的图层在添加到地图上的时候都是这个原理，我们在之前的章节也重点讲过这个原理。

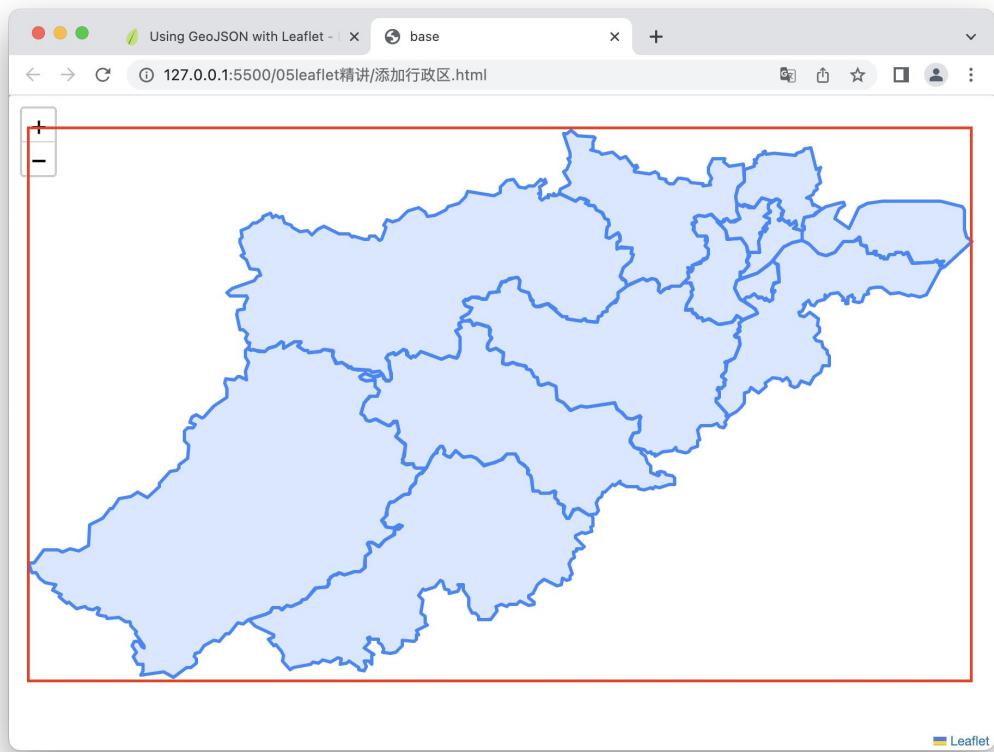
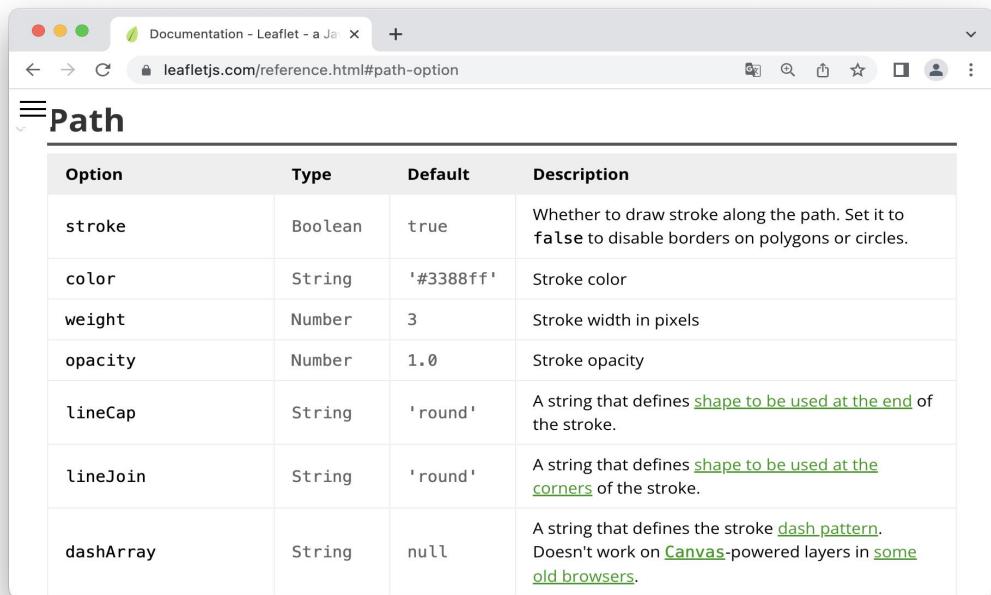


图 5.3.3-矢量图层定位原理

经纬度的范围就是上面图中红色方框表示的区域。因此我们要获取这个边界并且让地图适配到这个边界，这一点我们在第四章第 3 节也讲过。我们使用 `layer.getBounds()` 和 `map.fitBounds()` 两个方法来完成这个操作：

```
const layer = L.geoJson(hangzhou);
layer.addTo(map);
map.fitBounds(layer.getBounds());
```

顾名思义 `getBounds()` 意为获取图层的边界范围，而 `fitBounds()` 意为飞行到适配到这个边界范围。到目前为止加载并且定位到行政区已经完成了，接下来我们给行政区设置一下样式，这就需要在添加行政区的时候传入第二个参数，这个参数是一个对象，里面可以写很多的配置。



The screenshot shows a browser window displaying the Leaflet.js documentation at [leafletjs.com/reference.html#path-option](https://leafletjs.com/reference.html#path-option). The page title is "Path". Below the title is a table with the following columns: Option, Type, Default, and Description. The table lists seven properties: stroke, color, weight, opacity, lineCap, lineJoin, and dashArray.

Option	Type	Default	Description
stroke	Boolean	true	Whether to draw stroke along the path. Set it to <code>false</code> to disable borders on polygons or circles.
color	String	'#3388ff'	Stroke color
weight	Number	3	Stroke width in pixels
opacity	Number	1.0	Stroke opacity
lineCap	String	'round'	A string that defines <a href="#">shape to be used at the end</a> of the stroke.
lineJoin	String	'round'	A string that defines <a href="#">shape to be used at the corners</a> of the stroke.
dashArray	String	null	A string that defines the stroke <a href="#">dash pattern</a> . Doesn't work on <a href="#">Canvas</a> -powered layers in <a href="#">some old browsers</a> .

图 5.3.4-矢量图层设置样式

废话不多说我们直接上代码吧:

```
const myStyle = {  
    fillColor: "yellow", //填充颜色  
    weight: 3, //线条颜色  
    fillOpacity: 1, //填充透明度  
};  
  
const layer = L.geoJson(hangzhou, {  
    style: myStyle,  
});  
layer.addTo(map);  
map.fitBounds(layer.getBounds());
```

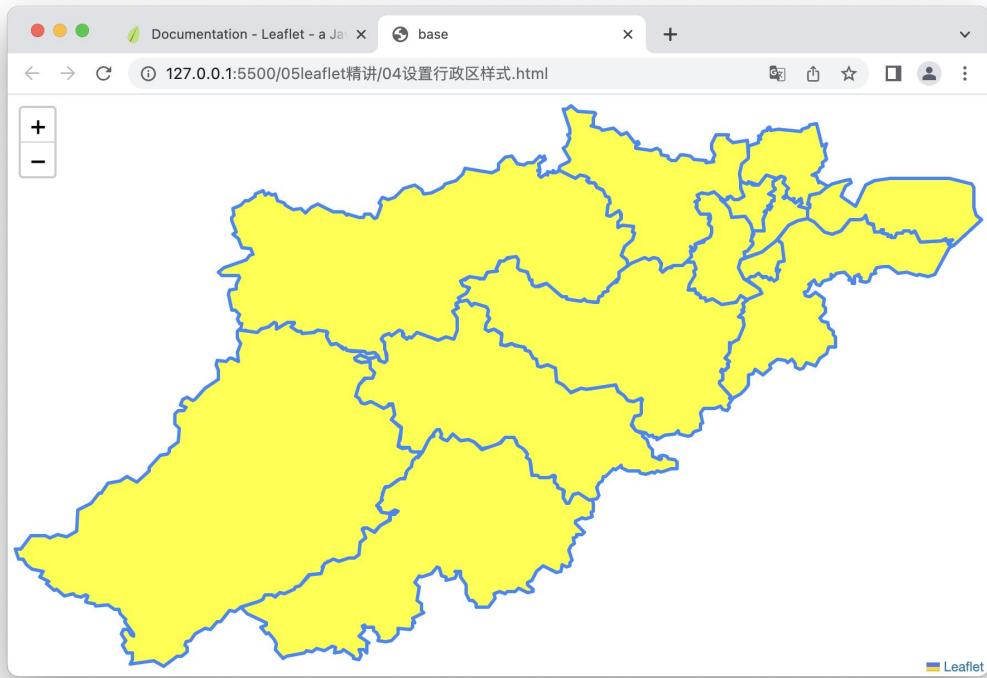


图 5.3.5-修改填充样式

好更改样式我们就做完了是不是很简单，只需要根据官网上给出的配置项设置对应的颜色，宽度等属性就行了。

我们设置颜色的时候除了可以把参数写成一个对象之外还可以写成一个函数，我们称作回调函数（js 基础概念，不清楚的要认真学一学哦）这样写有什么用呢？答案是可以根据数据中的某些属性动态的灵活的配置颜色。比如现在我需要将杭州市的这几个行政区显示成五颜六色的怎么办，就像这样：

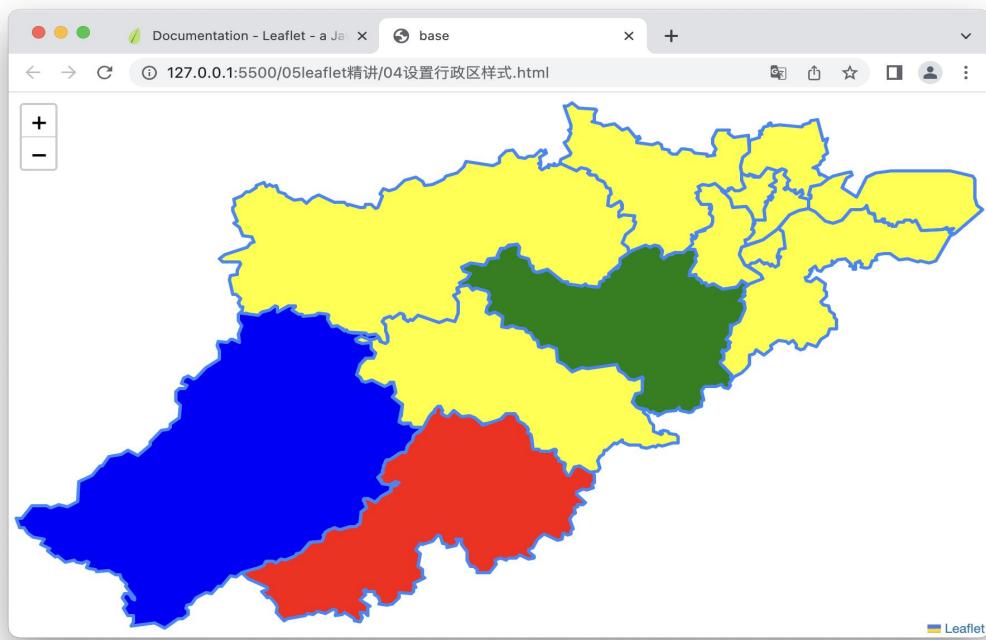


图 5.3.6-分级着色

那我们就必须给 `style` 传递一个回调函数，然后这个函数的返回值来决定样式，并且不同的条件能够决定不同的返回值：

```
//用函数的方式配置样式
const layer = L.geoJson(hangzhou, {
  style: function (feature) {
    //这里的 feature 是框架帮助我们获取到的，可以任意操作
    //这个方法的返回值必须是一个样式对象，像上面的 myStyle 一样
    const prop = feature.properties.name;
    if (prop === "淳安县") {
      return {
        fillColor: "blue",
        weight: 3,
        fillOpacity: 1,
      };
    } else if (prop === "建德市") {
      return {
        fillColor: "red",
        weight: 3,
        fillOpacity: 1,
      };
    }
  }
});
```

```

} else if (prop === "富阳区") {
  return {
    fillColor: "green",
    weight: 3,
    fillOpacity: 1,
  };
} else {
  return {
    fillColor: "yellow",
    weight: 3,
    fillOpacity: 1,
  };
},
});
layer.addTo(map);

```

样式修改了之后我们得想想交互操作了，如果现在有个需求，点击某个要素的时候让这个要素高亮，其余要素保持不变该如何做？这就需要用到我们的图层遍历参数，`leaflet` 在加载图层的时候会给我们提供一个配置项叫做 `onEachFeature`，这个配置项也是需要我们写一个函数，而且 `leaflet` 会帮我们把每一个 `feature` 和整个图层的 `layer` 对象都作为参数传递给我们，因此我们可以这样写：

```

const myStyle = {
  fillColor: "yellow",
  weight: 3,
  fillOpacity: 1,
};

const layer = L.geoJson(hangzhou, {
  style: myStyle,
  onEachFeature: function (feature, layer) {
    layer.on({ click: highlight });
  },
});

```

```

layer.addTo(map);
map.fitBounds(layer.getBounds());
function highLight(e) {
    layer.setStyle(myStyle);
    var newLayer = e.target;
    newLayer.setStyle({
        fillColor: "red",
    });
    newLayer.bringToFront();
}

```

总之我们要慢慢习惯这种配置方式，这一点在其他的框架中也是经常有所体现，就是参数的形式是函数的时候，就可以动态的配置某些属性，并且框架一般会在回调函数中把必要的数据都传递给我们，这样我们只需要对图层的 click 事件进行监听就可以完成相应的操作了。当用户点击的时候，先清空上次的图层样式的变化，然后再给用户选中的要素设置一个新的样式。

好了我们的矢量图层的操作就暂时告一段落了，因为按照需求去跟大家讲案例是永远讲不完的，我希望通过这几个例子能够对大家有所启发，后面根据这个几个案例修改出其他的完成需求的方案。

## 第 4 节：marker 与 dom

marker 在 leaflet 中或者是其他框架中都有类似的概念，就是表示一个点，并且是用图标来表示一个点，最简单的写法就像官网上的例子那样：

```
L.marker([120.345332, 29.394863]).addTo(map);
```

就一句代码非常的简单，但是按照这样的写法加载的是默认的样式的图标点，有的人会觉得 emm....有点丑。

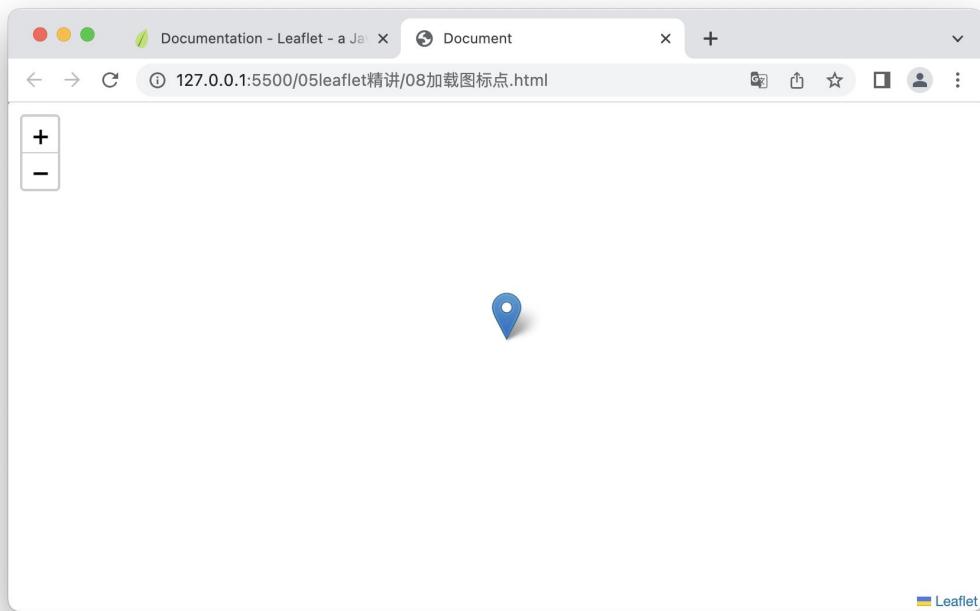


图 5.4.1-leaflet 原生 marker

所以我们可以修改一些配置参数来改变这个图标样式，使用自己的图标。

```
const myIcon = L.icon({
  iconUrl: "../icons/a.png", //图标地址
  //iconSize 表示图标的尺寸大小，单位为像素，数组中第一项表示宽度，第二项表示高度
  iconSize: [30, 30],
});
L.marker([29.4755343, 120.40342], { icon: myIcon }).addTo(map);
```

我们使用了 `L.icon()` 这个方法，配置了一个自定义的图标，并且给图标设置了尺寸，然后在 `marker` 方法中传入了第二个参数，它是一个对象，这个对象里面可以自定义点位的图标，因此我们就完成了自定义图标的操作：

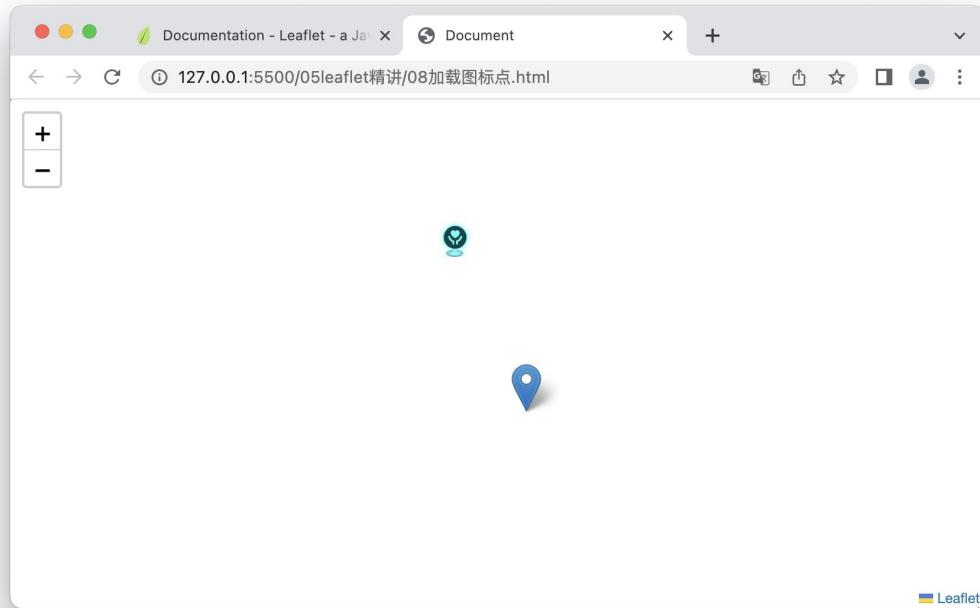


图 5.4.2-自定义图标

这个 `icon` 参数可以是一个图标对象，也可以是一个 `dom` 元素。

早在讲底层渲染原理的时候就说过我们的地图上可以叠加 `dom` 元素，我们这一就来练习一个小示例来看看具体怎么玩。还是以杭州市行政区为例，我们现在想要让各个区县的名字展示在地图上，应该如何做？`leaflet` 的属性配置里面可没有可以配置文字的地方，所以我们就要用到 `dom` 元素叠加在地图上的操作。我们在 `onEachFeature` 这个配置项里传入一个函数，这个函数也是回调函数，框架会帮助我们把 `feature` 和整个 `layer` 都返回给我们，然后我们在里面声明一段 `dom` 字符串，使用 `marker` 把 `dom` 字符串固定在指定的经纬度即可。`marker` 里面的 `icon` 也可以是 `dom` 元素，因此我们就实现了在地图上任意位置展示文字的操作。沿着这个思路我们还可以实现气泡图、散点图等跟 `dom` 元素相关的操作。大家有时间可以多去探究探究。

```
const layer = L.geoJson(hangzhou, {
  style: myStyle,
  onEachFeature: function (feature, layer) {
    const div =
      "<div style='width:200px>" + feature.properties.name + "</div>";
    const divIcon = L.divIcon({ className: "test", html: div });
    layer.setIcon(divIcon);
  }
});
```

```
L.marker(feature.properties.centroid.reverse(), {
  icon: divIcon,
}).addTo(map);
},
});
```

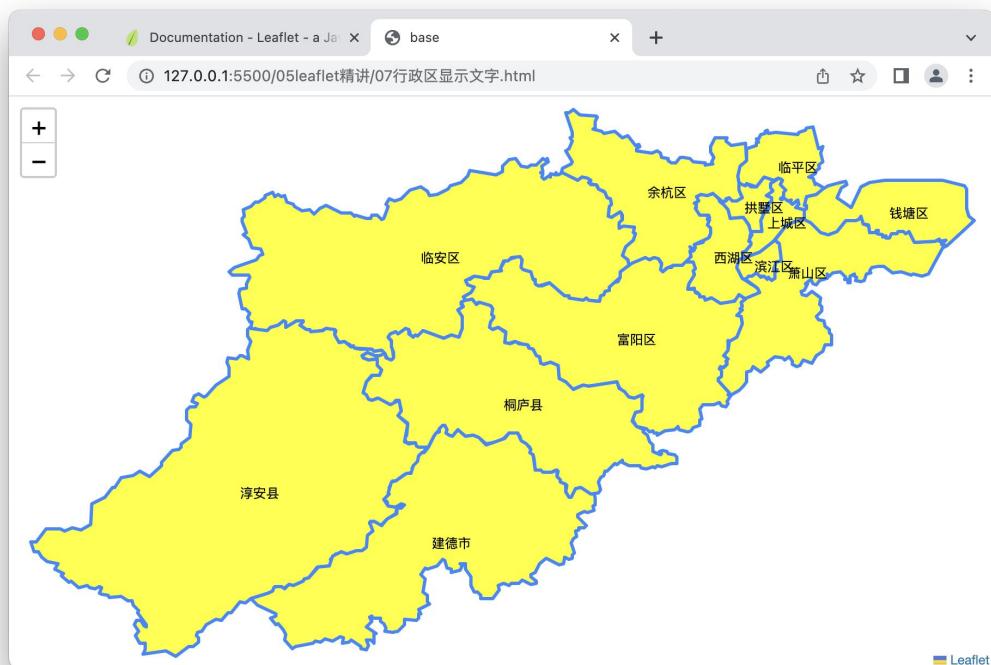


图 5.4.3-行政区展示名称

很多时候在页面渲染图标的时候，会有按照某种类别来渲染不同的图标的需求。如果是在 leaflet 中，可以根据 if 语句判断类型来构建不同的 divIcon，实现渲染多种不同类型图标的需求：

```
let poi = [
  { name: "希望小学", lon: "124.789412", lat: "47.347325", type: "school" },
  { name: "蜜雪冰城", lon: "103.007012", lat: "39.53535", type: "school" },
```

```
{ name: "天猫超市", lon: "117.872028", lat: "27.100653", type: "market" },
{ name: "如家酒店", lon: "124.642233", lat: "48.042886", type: "market" },
{ name: "人民医院", lon: "87.038159", lat: "31.42261", type: "hospital" },
{ name: "和平饭店", lon: "117.038159", lat: "31.62261", type: "school" },
{ name: "银泰城", lon: "121.038159", lat: "29.42261", type: "market" },
{ name: "海底捞", lon: "111.038159", lat: "28.42261", type: "school" },
];
function addPOI(data) {
  data.forEach((d) => {
    let url = "";
    if (d.type === "market") {
      url = "./icons/a.png";
    } else if (d.type === "school") {
      url = "./icons/b.png";
    } else {
      url = "./icons/c.png";
    }
    const myIcon = L.icon({
      iconUrl: url,
      iconSize: [30, 30],
    });
    L.marker([parseFloat(d.lat), parseFloat(d.lon)], {
      icon: myIcon,
    }).addTo(map);
  });
}
```

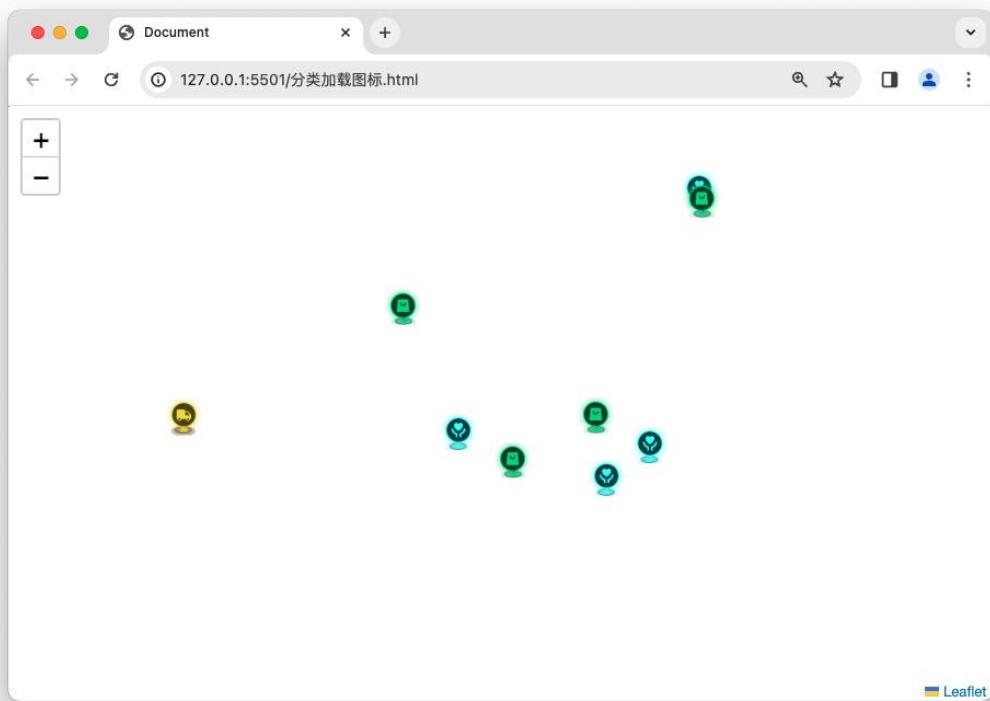


图 5.4.4-分类展示图标

## 第 5 节：栅格图层操作

对于栅格图层的操作要比矢量图层简单的多。因为栅格数据的加载我们在第二章说过，只需要确定栅格数据的边界范围就可以了。我在这里为大家准备了一份栅格图片数据，叫做 shaoxing.png，然后我们把它加载到地图上：

```
//绍兴市的边界
const bound = [119.889067, 29.225091, 121.231536, 30.286319];
var imageUrl = "../shaoxing.jpg",
imageBounds = [
[29.225091, 119.889067],
[30.286319, 121.231536],
];
L.imageOverlay(imageUrl, imageBounds).addTo(map);
```

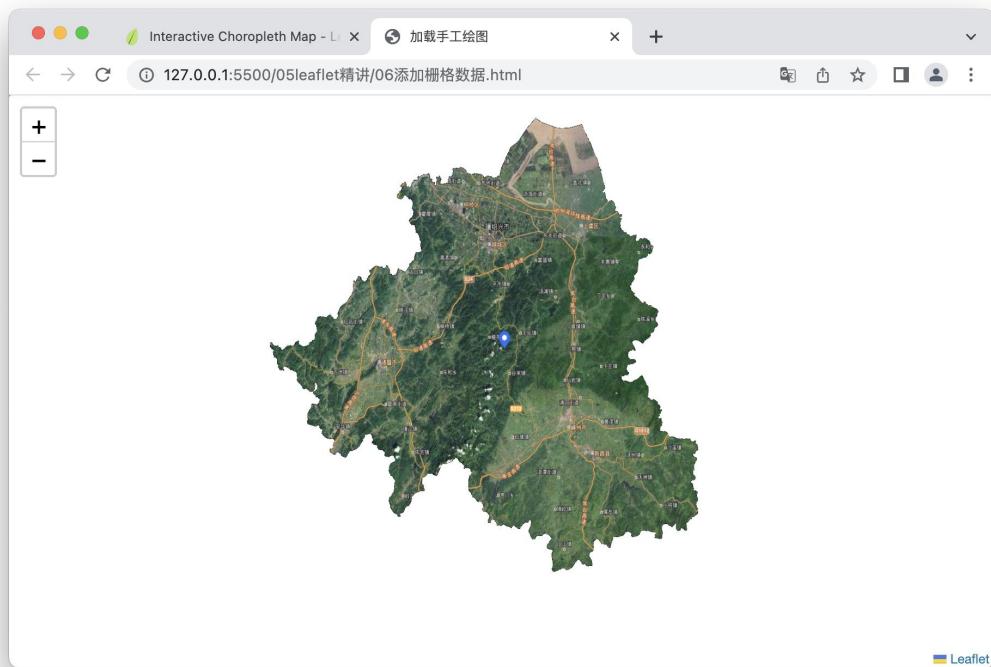


图 5.5.1-添加栅格数据

加载非常的简单，将图像的经纬度边界作为 `imageOverlay` 的参数传入即可。这个边界可以通过一些软件生成空间参考而获得。大家要学会举一反三，这个例子虽然看起来很简单只是添加了一张图片，但是在实际的开发场景中用处很多，比如有些时候你的系统中只想展示某个行政区内部的信息，比如我只想看杭州市内部的信息，杭州市以外的部分我不关心，这个时候你就需要使用这种方式加载杭州市的影像数据，其余的部分都被省略掉。类似的需求还有一些没有坐标系信息的手工绘图，例如下面这样的景区旅游地图，有的时候需要将它与真实地图做重合展示。也会用到我们上面讲的方式。



图 5.5.2-添加手绘地图

## 第 6 节：交互及事件监听

假设现在用户有一个需求，就是在点击行政区或者图标的时候，弹窗展示一些内容，我们使用 leaflet 该如何做呢？我们首先需要跟大家介绍一下，图层点击事件监听的概念：在 leaflet 中，对于图层的所有操作都可以被框架所监听，比如说点击图层，双击图层，鼠标在图层上滑动，缩放图层，拖拽图层等，这些操作是可以被 leaflet 框架所捕捉的。因此我们需要写这部分代码来监听用户对于图层操作，我们可以这样写：

```
const layer = L.geoJson(hangzhou, {
  onEachFeature: (f, l) => {
    //f 表示每一个要素, l 表示整个图层
    l.on({
      //监听图层的点击事件
      click: (e) => {
        //e.target.feature 表示当前鼠标点击后获取到的要素
        console.log(e.target);
      },
    });
  },
});
```

那么完整的做一个点击要素弹窗展示属性的例子源代码如下:

```
const map = new L.map("map", {
  center: [29.4234234, 120.646456],
  zoom: 8,
  crs: L.CRS.EPSG4326,
});

const layer = L.geoJson(hangzhou, {
  onEachFeature: (f, l) => {
    //f 表示每一个要素, l 表示整个图层
    l.on({
      //监听图层的点击事件
      click: (e) => {
        //e.target.feature 表示当前鼠标点击后获取到的要素
        console.log(e.target);
      },
    });
  },
});
```

```
var popup = L.popup()
.setLatLng([e.latlng.lat, e.latlng.lng])
.setContent(e.target.feature.properties.name)
.openOn(map);
},
});
},
});
layer.addTo(map);
map.fitBounds(layer.getBounds());
```

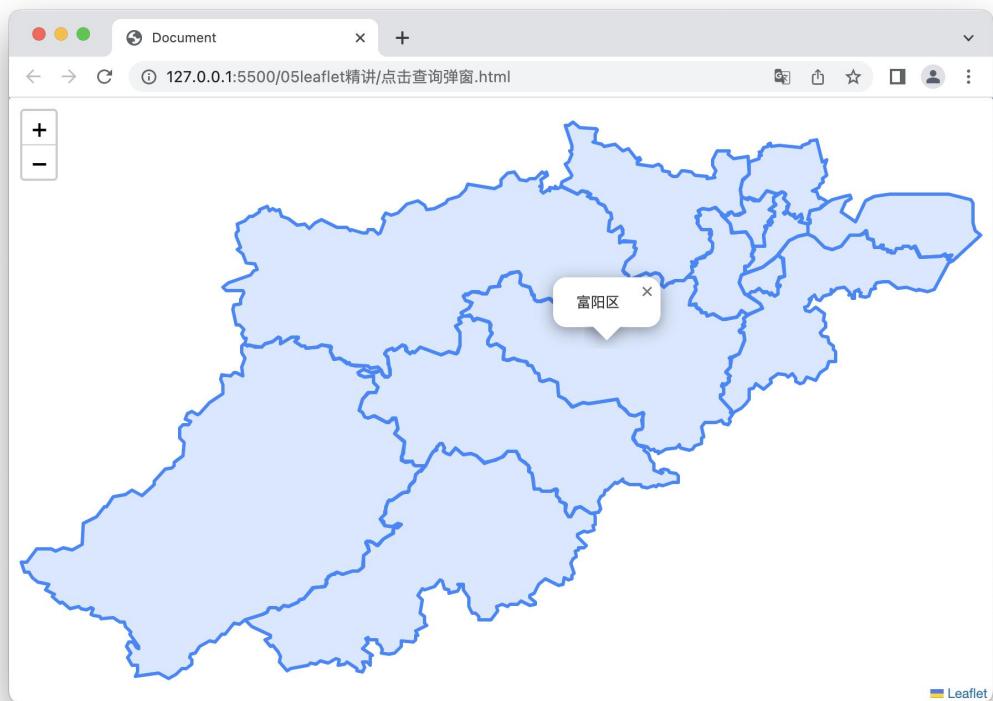


图 5.6.1-点击查询弹窗

还是那句话。我们在做每一个案例的时候都要学会举一反三，上面我们学会了如何在点击行政区要素的时候弹窗展示一些信息。我们就要想一个问题，假设用户现在需要点击某个点位的时候也弹窗展示一些内容。该如何做？或者说现在用户想改变操作方式，想让鼠标悬浮到图标上方的时候展示弹窗，鼠标离开时弹窗消失，又该如何做？这两个问题作为课后作业留给大家，大家可以思考一下如何来实现。

# 第六章： Mapbox 精讲

## 第 1 节： 基础入门

mapbox 是一个生态丰富的 GIS 框架， mapbox 不仅仅是支持 web 端的开发，它还支持移动端的 GIS 开发、还支持桌面端应用、还有一些地图服务以及静态地图资源。同时 mapbox 也是一个同时支持 2D/3D 的框架。所以 mapbox 框架本身是非常强大的。另外 mapbox 比较注重地图的样式，他们希望地图是美观的，是优美的。因此 mapbox 也使用了 canvas 的 ‘webgl’ 模式来渲染地图。mapbox 提供了一系列的图层样式调整措施，后面我们会跟大家讲到。非常的详细。

mapbox 其实严格意义上并不属于一个完全开源的框架，这一点从其 accessToken 就能看出来。你不申请 token 都无法使用 mapbox 这个框架。所以 mapbox 的 api 没有那么的底层和自由，可修改性、自定义的程度不高，使用起来让人有一种很拘束的感觉，我用下来给我的感受就是 mapbox 说什么我做什么， mapbox 让我玩什么我玩什么，不让我玩的东西我也玩不了……相比于 openlayers 和 leaflet， mapbox 非常的约束开发者的自由。下面是我和 mapbox 的一段对话：

mapbox：“我给你写了这么多 API， 你拿去玩吧”。

我：“我想自己玩”

mapbox：“不行， 我写什么你玩什么， 我不写的你不能玩”

我：“…………啊， 这”

按照惯例我们还是先来跟大家介绍一下 mapbox 的体系架构：

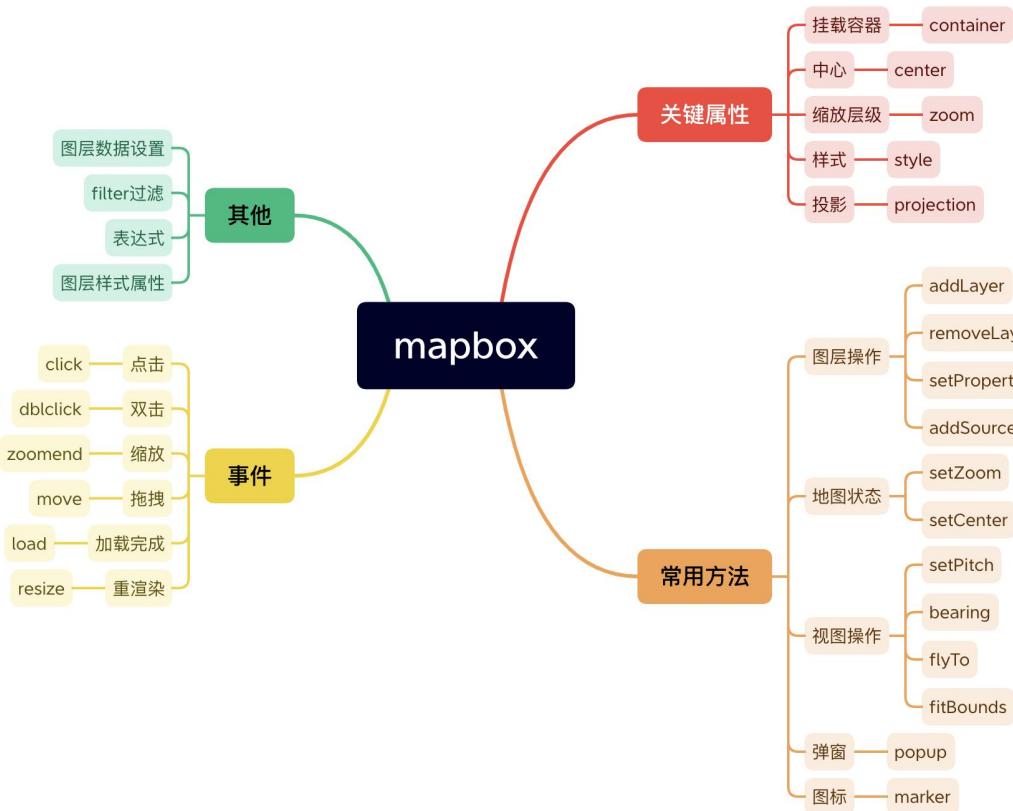


图 6.1.1-mapbox 体系架构

mapbox 框架和 leaflet 很类似，其实现有的这个 WebGIS 框架都很类似，这其实都是 js 的语言特性导致的，任何框架的核心都是一个类或者构造函数，然后指定一些属性和方法，监听一些事件，最后再对一些参数进行类化。mapbox 常用的属性也就是对于地图经常要用到的调整参数，比如缩放级别、中心点、倾斜角度、投影坐标系等等。核心方法通常就是设置核心参数和对于图层和组件的操作。包括设置中心点、倾斜角度、添加图层、移除图层、弹窗等等。事件其实几乎和 leaflet 一样的，监听包括地图的单击、双击、缩放、拖拽、以及加载完成事件。

mapbox 官网的使用大致分为以下几个部分，首先是案例部分：

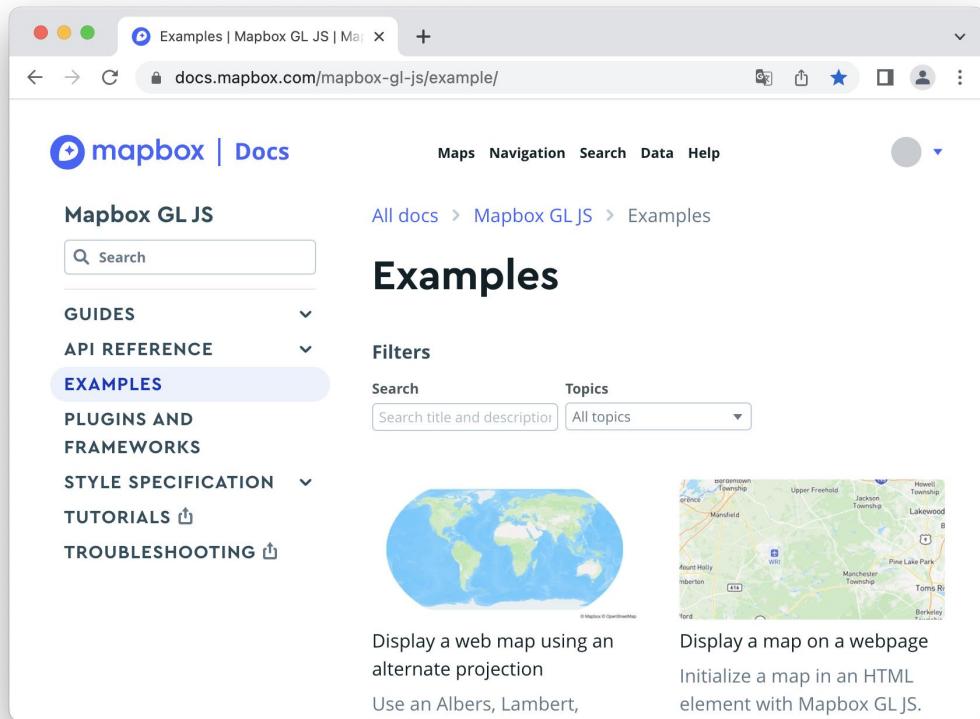


图 6.1.2-maobox 官网案例

案例部分提供了很多例子，几乎可以涵盖行业内 70% 左右的需求。很多需求都可以根据案例来改写举一反三。其次是 API Reference 部分，解释了 mapbox 中所有的 api 的使用规则，并且也提供了些许案例。

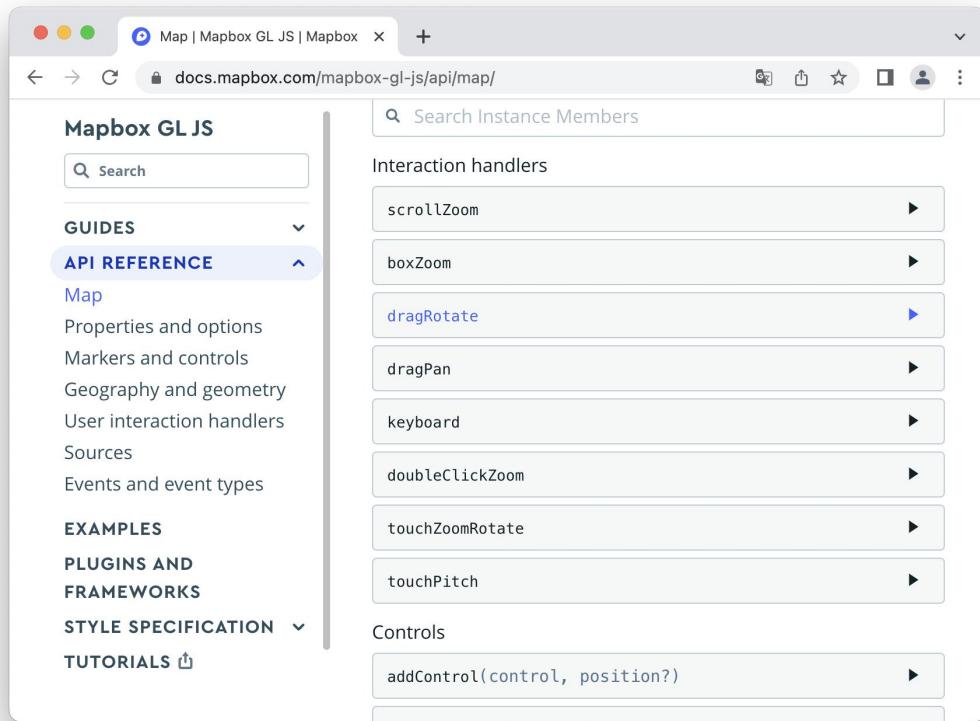


图 6.1.3-mapbox 官网 api 文档

最后是 **STYLE SPECIFICATION** 部分，这里介绍了 mapbox 中图层样式的配置方案。

样式配置有很多并且很复杂，需要大家仔细研读说明文档来区分那些细微的差异。

所以官网上我们经常访问的地方也就那么几个，首先如果你是零基础的初学者可能要看 **GUIDES** 部分学习如何安装和引入 mapbox。如果你是有点经验的开发者。可能后面经常会查看这些案例 **EXAMPLES** 以及 方法属性参考 **API REFERENCE**。可能有的时候你碰到了一些比较罕见或者高阶的需求，你发现 **example** 里并没有你需要的内容，那你就需要访问 **PLUGINS** 里的内容了。最后。不管你是什么阶段的开发者，你都会经常的去访问 **STYLE SPECIFICATION** 这一部分。因为 mapbox 的样式配置实在太多太详细了，通常情况下根本记不住，因此需要经常查看一些样式属性。

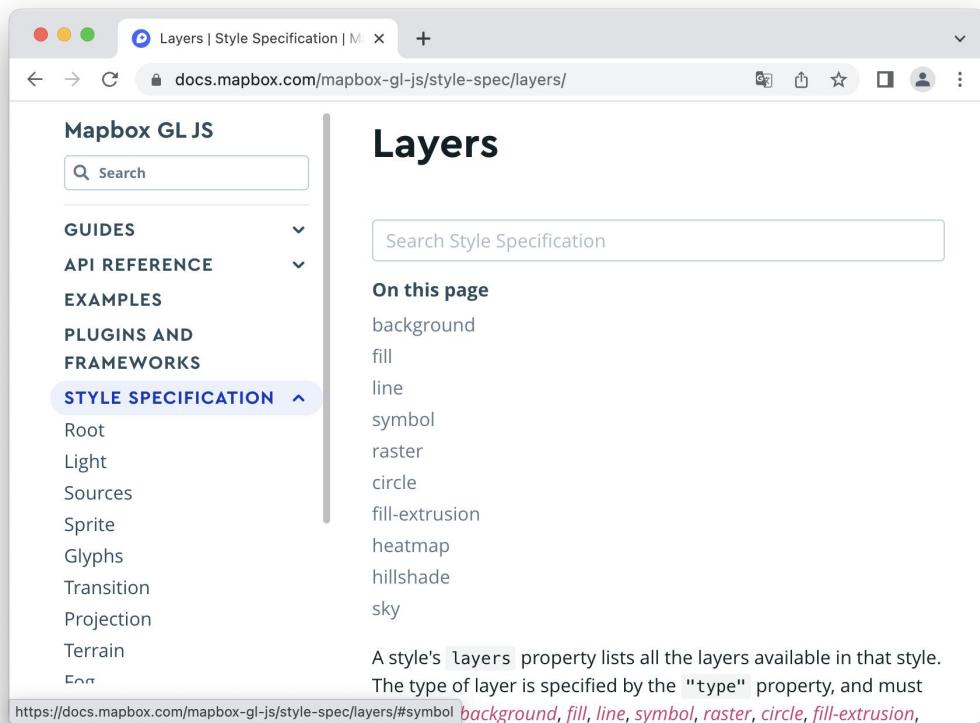


图 6.1.4-mapbox 官网样式标准

在正式开始写代码之前，还有一个小消息需要大家了解，那就是 mapbox 中只支持 web 墨卡托坐标系，不支持经纬度坐标系。因此基于经纬度坐标系栅格类型的数据是没有办法在 mapbox 中渲染的，矢量数据不用担心这个问题，mapbox 官方会自动帮我们转换成墨卡托坐标系再进行渲染。但是在输出坐标的时候又会给我们输出经纬度，这涉及到屏幕坐标换算的相关知识，当各位研究到更深层次的时候会彻底理解这样的操作。日常的开发场景大家了解就好了。

好了言归正传我们先来初始化一下 mapbox，首先 mapbox 也是有两种初始化方式，如果你需要写原生的 html，那还是需要引入 css 标签和 js 标签才能使用。所以你需要写下面两行代码：

```
<script
src="https://api.mapbox.com/mapbox-gl-js/v2.14.1/mapbox-gl.js"></script>
<link
href="https://api.mapbox.com/mapbox-gl-js/v2.14.1/mapbox-gl.css"
rel="stylesheet"
/>
```

很明显一个是官方提供的 **js** 文件，一个是官方的 **css** 样式文件，但是大家要注意这种方式是 **cdn** 的方式，也就意味着资源都是在线的，如果你的开发环境通常是不允许连接外网（互联网）的，或者网络环境较差的，那你不妨把这两个文件下载到本地然后再引入。下载的方式也很简单，可以直接将这两个地址放在浏览器地址栏里，然后打开调试控制台，选择网络（**network**）一栏，把文件另存为本地即可。

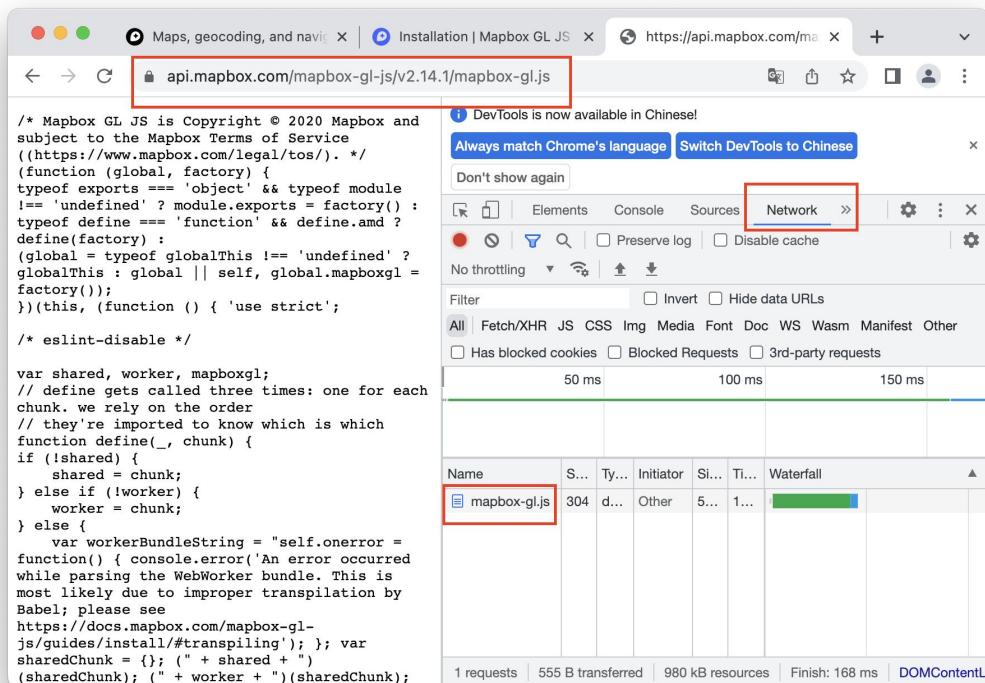


图 6.1.5-下载 maobox 资源文件

所以我们后面的开发案例当中都是采用本地静态引入的方式。

```
<script src="./mapbox-gl.js"></script>
<link href="./mapbox-gl.css" rel="stylesheet" />
```

在初始化 **mapbox** 之前，先跟大家分享一个背景故事。这个故事虽然跟写代码没关系但是它很有用，那就是 **leaflet** 的作者是个乌克兰人，自创立了 **leaflet** 框架之后，被美国人发现并且觉得是个人才，于是乎 **mapbox** 公司就聘请了 **leaflet** 的作者作为公司开发 **mapbox gl js** 的主要成员，所以在编写 **mapbox** 代码的时候。总感觉会有那么一点点 **leaflet** 的影子，语法上有的时候跟 **leaflet** 很像，甚至说某些方面的设计简直是一毛一样，所以当有些语法想不起来或者不知道怎么写的时候可以参考一下 **leaflet** 的写法。

引入好资源文件以后首先第一步还是把访问 **token** 写好（自己在 **mapbox** 官网上申请一个 **token**）。然后我们还是像 **leaflet** 一样初始化一个地图对象 **map**。首先还是准备一个

地图挂载的容器，也就是下方代码中的 `container` 属性，将来用来决定地图渲染的尺寸和位置。再传入一些挂载容器、中心点、缩放层级、样式等必要的参数，这样就能够初始化好地图配置。

```
mapboxgl.accessToken = mapboxtoken;  
  
const map = new mapboxgl.Map({  
  container: "map",  
  center: [120.536363, 29.243242],  
  zoom: 8,  
  style: "",  
});
```

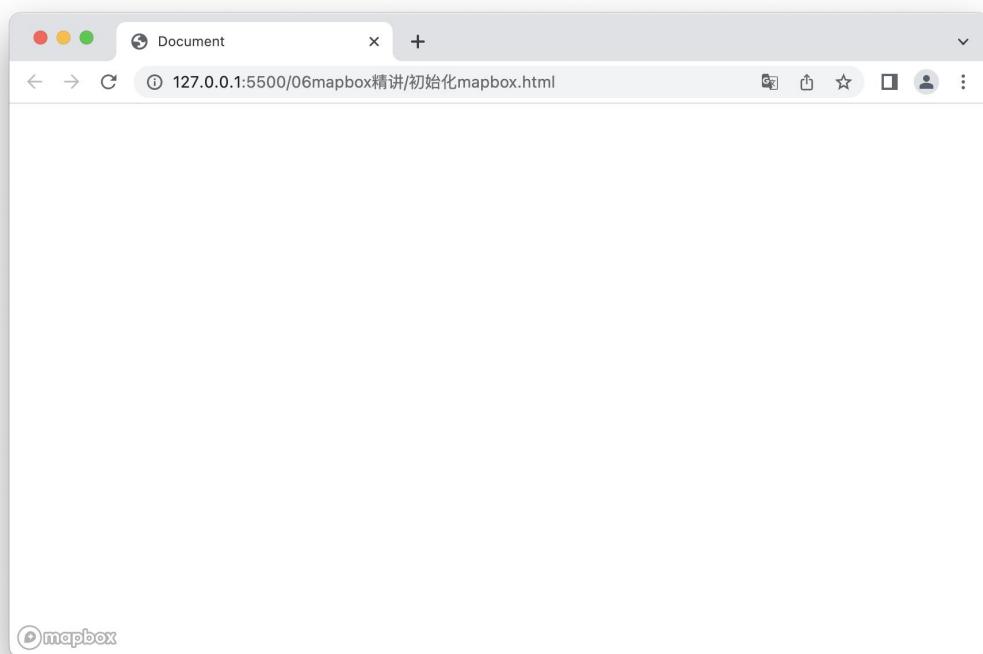


图 6.1.6-mapbox 初始化成功

看到页面左下角出现 mapbox 的商标就证明初始化成功了，与 leaflet 不同的是 mapbox 官方是自带一些底图的：

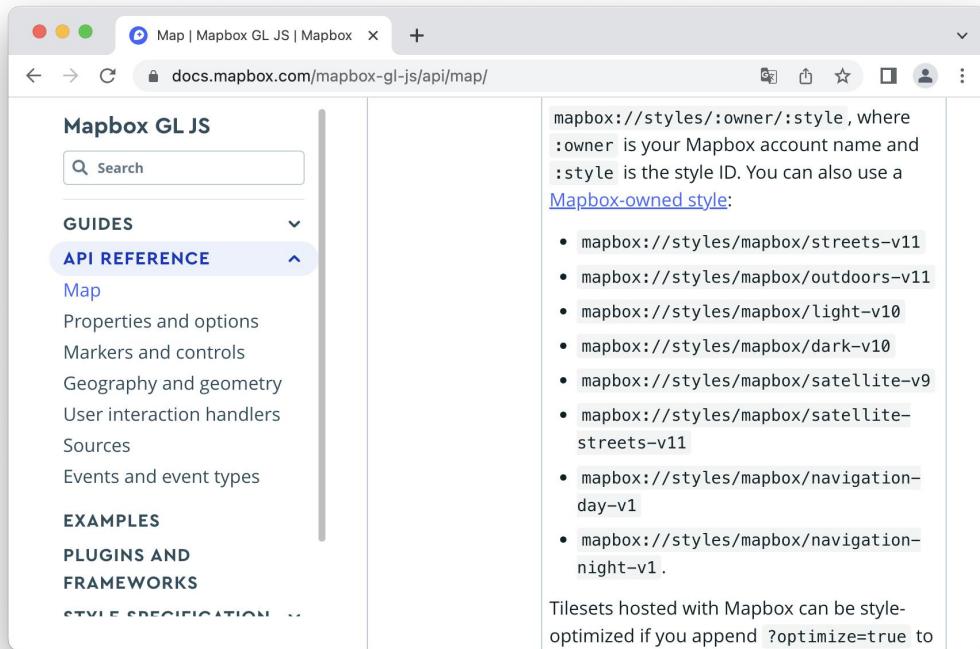


图 6.1.7-mapbox 的地图样式

我们只需要将其中的任何一个样式填写在 style 的配置项里即可。

```
mapboxgl.accessToken = mapboxtoken;

const map = new mapboxgl.Map({
    container: "map",
    center: [120.536363, 29.243242],
    zoom: 8,
    style: "mapbox://styles/mapbox/satellite-v9",
});
```

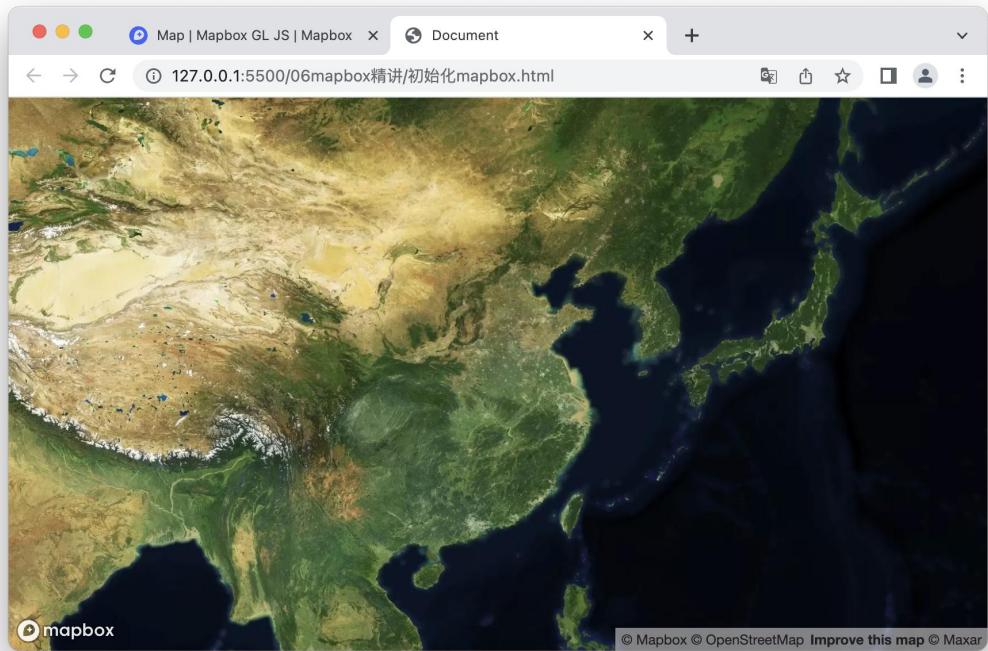


图 6.1.8-mapbox 设定影像底图

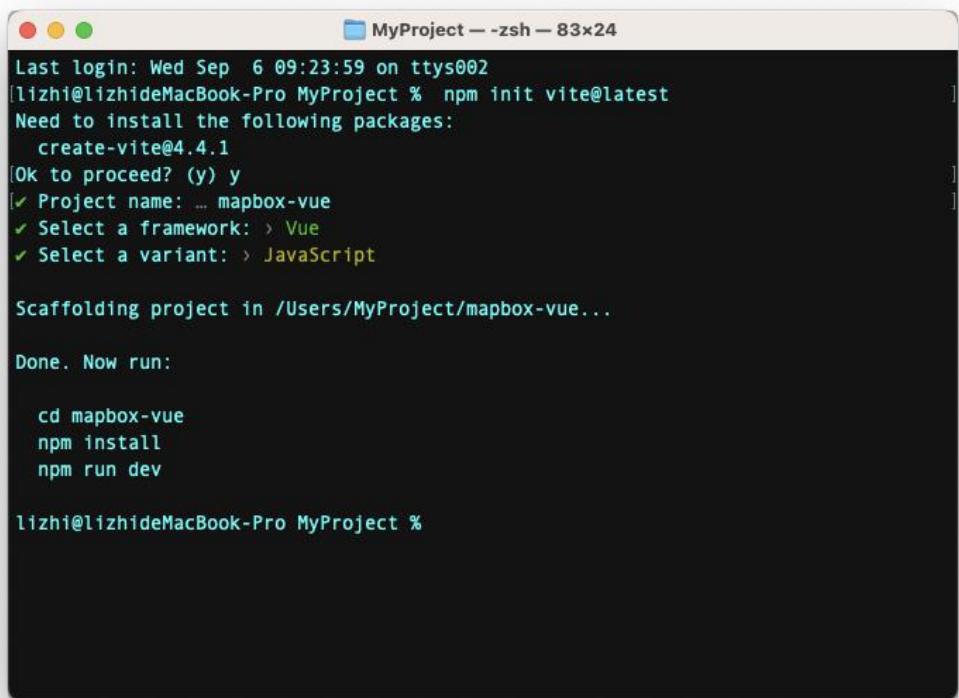
这样就完成了 mapbox 的初始化。其中 mapbox 提供了很多样式的底图，大家可以選擇其他的底图样式应用在不同的项目中。

如果使用了 vue、react 这样的前端工程化脚手架，我们的初始化方式其实是类似的。大家要注意一个点，那就是 vue、react 只是简化了前端的开发，提高了前端的开发效率，优化了前端页面的性能，但是它本质上还是没有脱离 JavaScript，换句话说，你之前在 html 里面怎么写的 js，到了 vue 还是同样的逻辑，只不过是语法习惯要遵循 vue 作者所提出的规范。这一点其实与 GIS 无关，大家只需按照自己所在公司的技术选型，单独地去了解这些前端框架即可，相比于 GIS 本身的原理，这些框架其实都非常的简单。

话不多说我们还是用 vite 脚手架创建一个项目：

```
npm init vite@latest
```

然后按照提示输入项目的名字，我们在这里起个名字叫 mapbox-vue。



```
MyProject — zsh — 83x24
Last login: Wed Sep 6 09:23:59 on ttys002
[lizhi@lizhidemacBook-Pro MyProject % npm init vite@latest
Need to install the following packages:
  create-vite@4.4.1
[OK to proceed? (y) y
✓ Project name: ... mapbox-vue
✓ Select a framework: > Vue
✓ Select a variant: > JavaScript

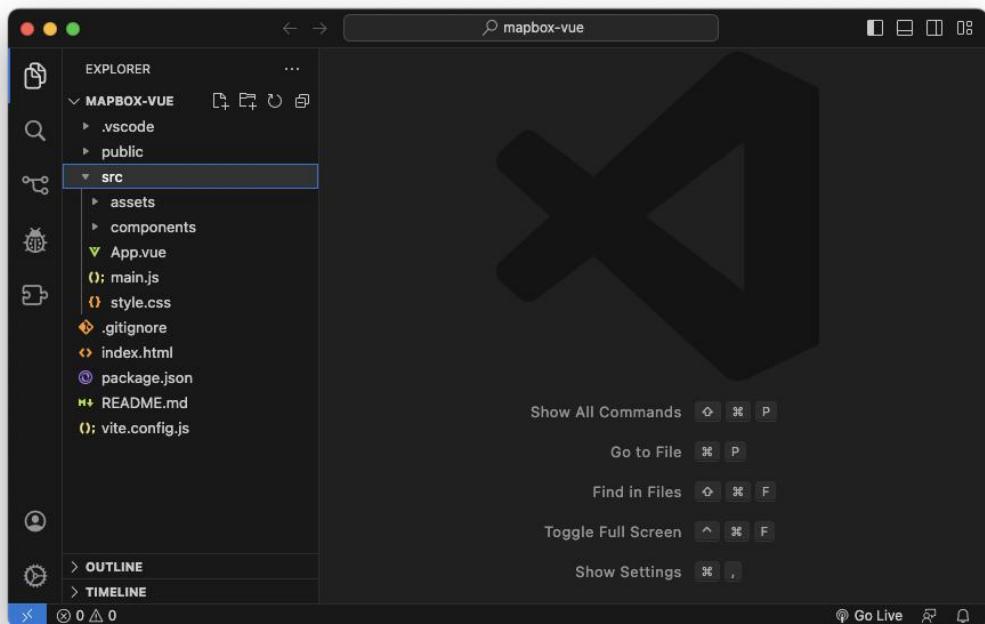
Scaffolding project in /Users/MyProject/mapbox-vue...

Done. Now run:

  cd mapbox-vue
  npm install
  npm run dev

lizhi@lizhidemacBook-Pro MyProject %
```

创建好之后我们的工程应该长这个样子：

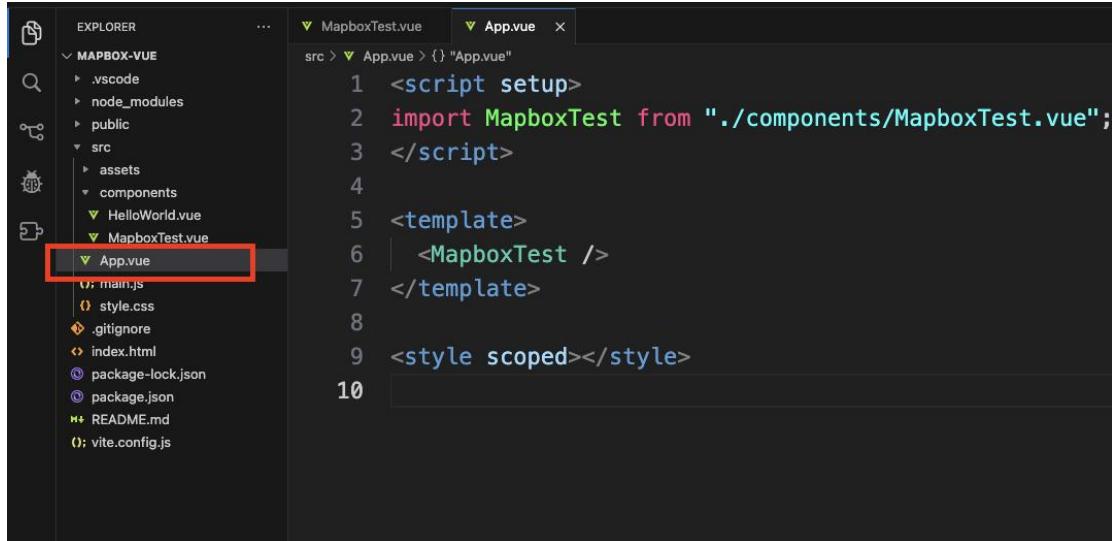


接下来我们需要首先安装 mapbox 的依赖，在终端输入：

```
npm install mapbox-gl
```

等待下载安装完成以后，创建一个新的组件叫做 MapboxTest.vue，然后在组件里面准

备写 maobox 的相关内容，不过，在这之前得先去把 app.vue 文件中的 helloworld 删除，然后把我们新写的 MapboxTest.vue 组件引入：



```

src > ▾ App.vue > {} "App.vue"
1 <script setup>
2 import MapboxTest from "./components/MapboxTest.vue";
3 </script>
4
5 <template>
6   <MapboxTest />
7 </template>
8
9 <style scoped></style>
10

```

接下来我们就可以放心的去写我们组件里面的 GIS 逻辑了。我们先来简单做个地图的初始化工作，正常的去写我们地图初始化逻辑。

和原生的一样，我们也需要在组件里准备一个容器 div 用于放置地图，然后也需要给这个 div 写样式：

```

<template>
  <div id="map"></div>
</template>
<style>
  #map {
    width: 100%;
    height: 100vh;
    position: absolute;
    inset: 0;
  }
</style>

```

然后我们初始化一个 map 对象，填写对应的配置参数，在这里要提醒大家，**尽量不要将 map 对象放在 data 选项里，在 vue3 中尽量不要把 map 做成 reactive 的**，因为 map 对象本身绑定了许多的属性方法和事件，vue 的好处虽然是可以做到数据变化页面自动刷新，但是在 GIS 中我们并不希望这样，反而这样的操作会消耗浏览器很大的资源。因此我们还是按照常规的 js 逻辑去初始化 map 对象，如果想让 map 对象是全局变量，做到任何

方法里都可以用，在 vue2 中可以把初始化的过程放在 script 脚本的最前面，也就是 import 之后，在 vue3 中可以灵活的控制其为非响应式对象。以 vue2 为例，代码可以这样写：

```
import mapboxgl from "mapbox-gl";
var map = null;
export default {
  name: "MapboxTest",
  data() {
    return {};
  },
  mounted() {
    this.init();
  },
  methods: {
    init() {
      mapboxgl.accessToken = "你的 token";
      map = new mapboxgl.Map({
        container: "map",
        center: [120.536363, 29.243242],
        zoom: 8,
        style: "mapbox://styles/mapbox/satellite-v9",
      });
    },
  },
};

</script>
```

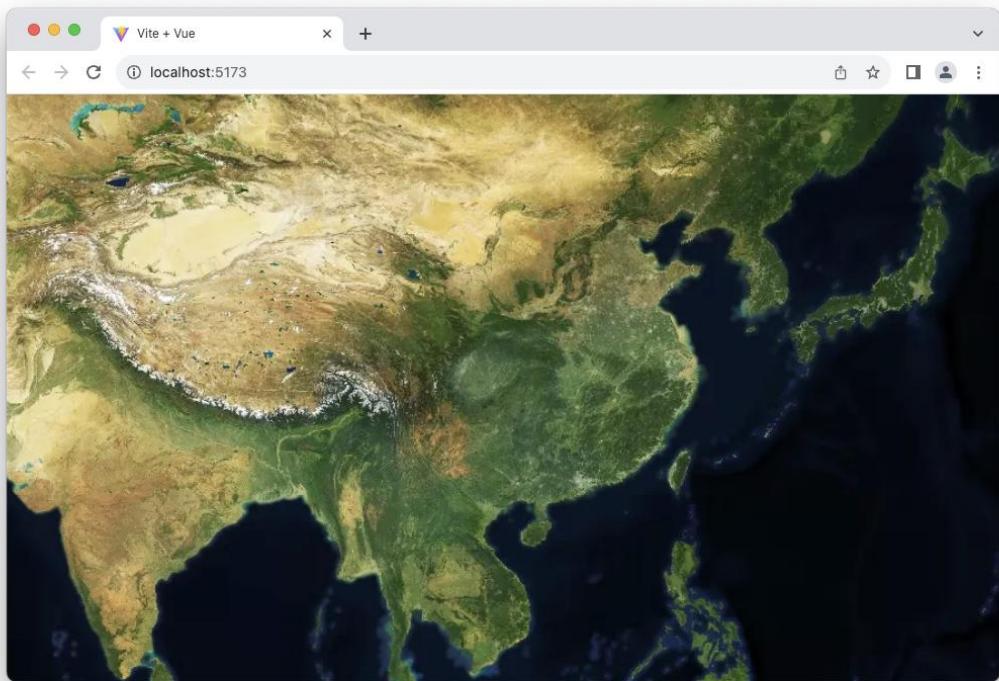


图 6.1.9-mapbox 与 vue 结合

这样我们就成功的将 `mapbox` 与 `vue` 做了一次融合。后面的开发过程大家可以仿照这个例子，将自己的功能都封装在一个一个的组件里面，核心的 js 逻辑都是一样的，比如现在想写一个测量组件，那么我们可以新建一个 `Measure.vue`，然后把组件的 ui 写在 `template` 里和 `style` 里。把核心逻辑封装成方法，然后把需要传递的参数预留好位置，至于父子组件之间的传参以及交互在这里就不展开细讲了，大家可以去官网查阅相关用法。因为这些前端框架是针对所有前端开发者做的，因此不会因为我们是 GIS 开发就有什么特殊的和不一样的地方，该学习的 `vue` 本身的知识一样也不能少。

## 第 2 节：基础底图

不管你是用什么框架，在国内的项目开发中加载天地图都是一个**非常基础且重要的需求**。因此在 `mapbox` 中也不能省略这一部分内容。加载天地图之前我们还是要把天地图的 `token` 申请好。和 `leaflet` 相似，我们通过服务 `url` 就可以加载。不过需要注意的是。在 `mapbox` 中的**所有操作都必须发生在地图加载完成以后**。也就是说地图加载完之前任何操作都是无效的，甚至是会报错的。那么什么时候叫地图加载完呢？`mapbox` 框架为我们提供了一个地图加载完成之后的监听回调，也就是 `map.on('load', e=>{ })`.`map` 会去监听地图加载完成的事

件，这个事件名字叫做 **load**，当加载完成以后 **mapbox** 会提供一个回调函数。我们的逻辑必须写在这个回调函数里面。

```
map.on("load", (e) => {
    //下面写的代码会正常执行，因为这时候 mapbox 已经初始化完成了
    const tdt_url =
        "https://t0.tianditu.gov.cn/DataServer?T=img_w&x={x}&y={y}&l={z}&tk=";
    //服务地址
    map.addSource("tdt-source", {
        type: "raster", //类型为栅格类型
        tiles: [tdt_url + tdttoken], //天地图地址以及请求 token
        tileSize: 256, //瓦片尺寸大小
    });
    map.addLayer({
        id: "tdt-layer",
        type: "raster",
        source: "tdt-source",
    });
});
```

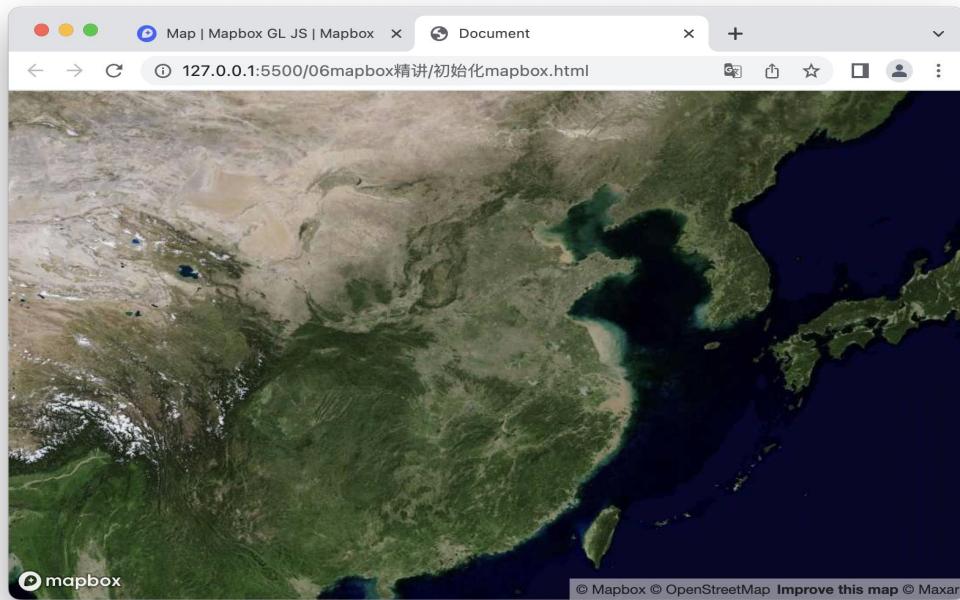
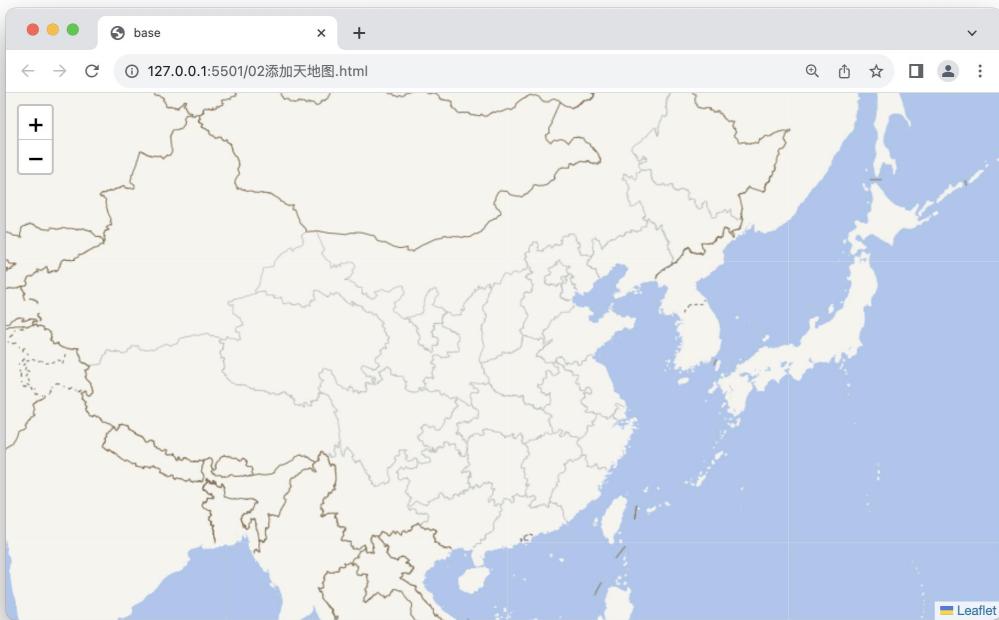


图 6.2.1-mapbox 添加天地图

从上面这段代码中不难看出。mapbox 在加载图层的时候，会先加载一个 **source**（图层源）对象。这个图层源有很多种，但是大体上还是分为我们的栅格数据源和矢量数据源，然后 **source** 相关配置参数都填写好以后，才会调用 **addLayer** 方法将图层添加在地图上。如果我们需要开发移动端相关的 app，那么使用天地图矢量风格的地图作为底图可能更加合适，因此我们只要把 url 里的 `img_w` 换成 `vec_w` 即可，即请求地址为：

```
"https://t0.tianditu.gov.cn/DataServer?T=vec_c&x={x}&y={y}&l={z}&tk=你的token"
```

，这样你就得到了一个矢量风格的底图。



还是要再次强调 `mapbox` 中的这个机制。在 `mapbox` 中地图的渲染是异步的，不是瞬时完成的，并且 `mapbox` 官方也没有对渲染做等待处理，而是把渲染完成之后的回调函数交给了用户处理，这样就相当于把犯错的可能交给了用户，因为在很多时候大家容易忽略这个 `load` 回调，导致很多莫名其妙的错误发生。比如有的时候发现一个图层只渲染了一部分，另一个部分丢失，比如刷新页面有的时候有些图层也无法正常显示，再比如有的图层的样式和我们所设定的样式不一致，这大多都是因为并没有等待 `mapbox` 初始化结束就开始操作。这一点在实际的开发过程中大家一定要注意。

另外在有些场景中（大屏），客户可能需要地图能够美观的呈现，比如说让地图变的立体并且是只显示客户关心的地图区域，其他区域则不显示，对于这样的需求我们有两种解决方案。其中一种方案是采用贴图图片的方式，即我们将一张图片贴在地图上，至于图片的美观这就需要你公司的 UI 同事帮忙了。加入我们现在从 UI 同事那里获取到一张具有 3D 立体效果的 `png` 图片：

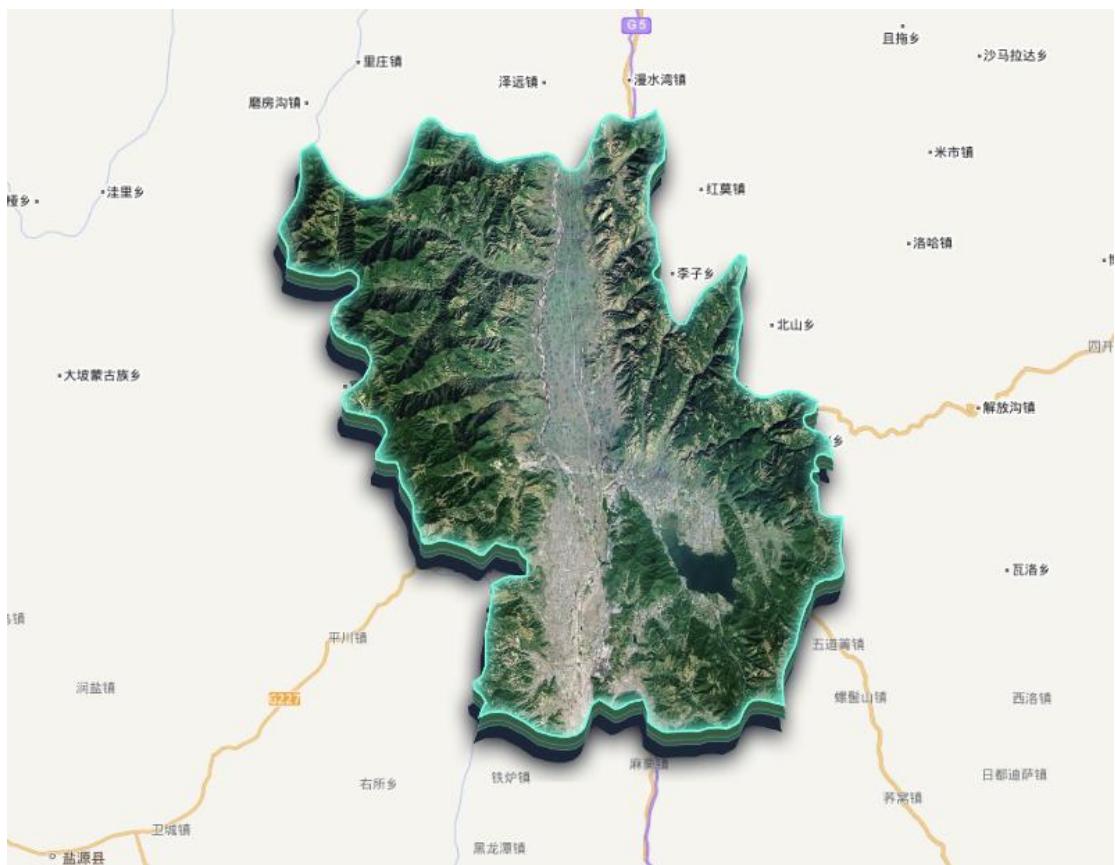


现在你只需要考虑如何将它加载到地图上。你需要使用 mapbox 的 `image` 类型的 `source` 和 `raster` 类型的 `layer` 来添加此数据。具体代码如下：

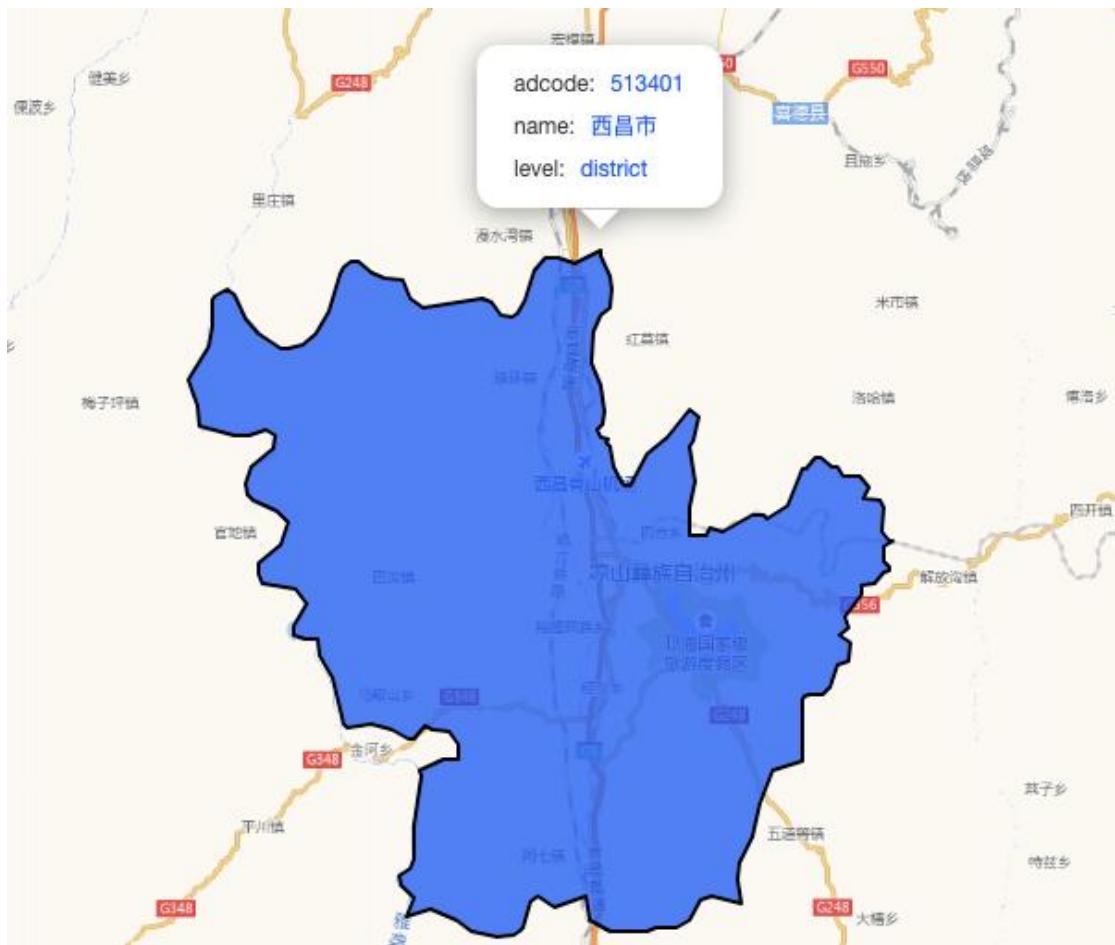
```
//图片存放路径
let url = "./icons/xc.png";
//图片经纬度边界
let extent = [101.73497559, 27.50030558, 102.43557243,
28.18097996];
map.addSource("ui", {
  type: "image",
  url: url,
  coordinates: [
    [extent[0], extent[3]],
    [extent[2], extent[3]],
```

```
[extent[2], extent[1]],
[extent[0], extent[1]],
],
});
map.addLayer({
  id: "ui",
  type: "raster",
  source: "ui",
});

```



我们来解释一下原理，其实很简单，我们首先要有一个类似于地理配准的过程，即我们应当知道图片所处的地理范围边界，即最西边的经度，最东边的经度，最南边的纬度和最北边的纬度，四个控制点来决定该图片的位置。这样我们就能在地图上唯一锁定一张图片的位置。清楚其原理之后，同学们可能还会问如何获取图片的边界经纬度范围呢？在这里给大家提供一种方式，即找到与图片对应的矢量数据，获取其矢量数据的边界范围，应用在图片上再加以调整即可，比如像上述的例子，我只需找到一份 **geojson** 格式的矢量数据。



再通过 turf.js 等工具，使用 turf.bbox() 函数获取其 bbox（经纬度边界范围）即可获取该图片的经纬度范围。也就是示例代码中的 extent 值，这样一通操作下来，一张具有立体感的三维感的影像地图就成功加载了！

### 第 3 节：矢量图层操作

在讲矢量图层操作之前我想先说说 mapbox 当中的图层都有哪些类型，我们常用的图层类型一共有 7 种分别是：**面状要素图层 (fill)**、**线状要素图层 (line)**、**点或者圆要素图层 (circle)**、**图标和文字点要素图层 (symbol)**、**背景图层 (background)**、**栅格图层 (raster)**、以及**三维面要素图层 (fill-extrusion)**。

首先面状要素图层就是用来加载一些多边形面要素，例如行政区、地块等要素。线要素图层就是线条要素例如河流、道路等。点或者圆要素就是点位，但是这个“点”是一个有半径的圆面，只不过圆非常小的时候就是一个点了。图标图层也是用来表示点位要素只不过这个点的位置可以不是 canvas 所渲染的圆面，而是一个图标，图片。背景图层可以设置底图

的背景，比如你的页面中加载了杭州市的行政区，但是地图背景现在是空白的。你可以将其设置为透明背景。这样你在 `map` 所挂载的 `div` 容器底部的内容（颜色，背景图片）等就都可以显示出来了。栅格图层通常是用来加载我们的栅格数据，或者是一些栅格瓦片服务。三维面要素图层是在面要素的基础上增加了立体扩展，让整个图层具有 3D 效果。

我们还是像 `leaflet` 一样来添加杭州市的行政区数据并且设置一些样式和交互。首先添加行政区图层的过程还是发生在 `load` 事件的回调函数里面。只不过这次我们要加载的图层类型是矢量数据。我们的数据还是和 `leaflet` 一样使用杭州市的 `geojson` 数据，并且附带下属的各个区县的边界。我们直接来添加：

```
map.on("load", (e) => {
  map.addSource("xzq-source", { type: "geojson", data: hangzhou });
  map.addLayer({
    id: "xzq-layer",
    type: "fill", //类型为面状要素 fill
    source: "xzq-source",
    //paint 为样式设置字段，可设置填充颜色，填充透明度，填充图案等等
    paint: {
      "fill-color": "yellow",
      "fill-opacity": 0.8,
    },
  });
});
```

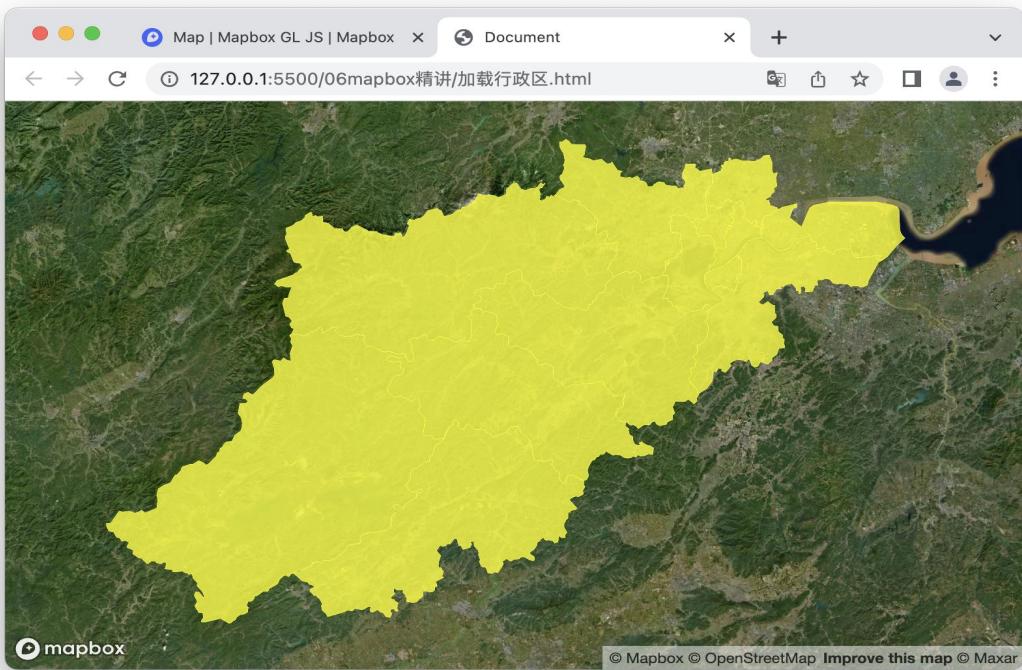


图 6.3.1-mapbox 添加行政区

那么如何来设置行政区边界的样式呢，在 mapbox 中不能像在 leaflet 和 openlayers 中那样直接设置线条样式和填充样式。而是需要添加两个图层，一个图层设置填充样式，另一个图层设置线条样式。比如你需要将上面的行政区边界线的颜色设置成红色。就需要再加上下面这段代码：

```
map.addLayer({
  id: "xzq-line",
  type: "line",
  source: "xzq-source",
  paint: {
    "line-color": "red",
    "line-width": 2,
  },
});
```

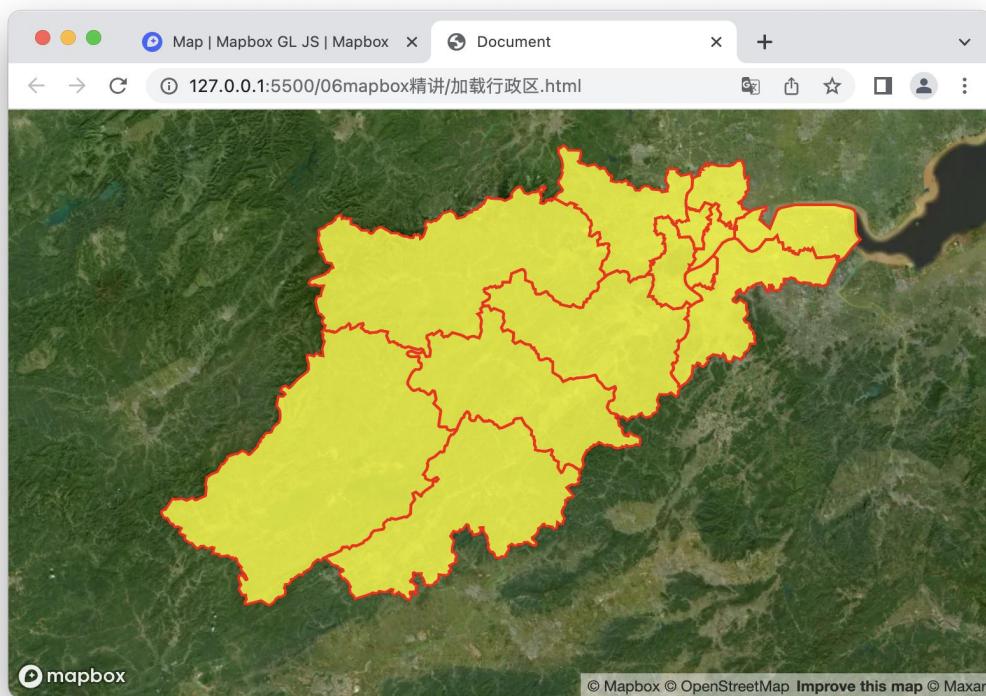


图 6.3.2-mapbox 修改行政区样式

对于矢量图层的样式配置一般会有两个参数分别是 `paint` 和 `layout`。这一点在官网中也有体现。这两个有什么区别？一个是绘图参数，另一个是布局参数。大部分跟样式有关的都会在 `paint` 里，跟数据相关的都会在 `layout` 里，不过这也不绝对。由于 mapbox 的样式属性特别多，而且经常性分不清楚某个属性是属于 `paint` 还是属于 `layout`。因此在实际开发中可能都需要我们不断的去阅读：

<https://docs.mapbox.com/mapbox-gl-js/style-spec/layers/>

可配置的属性非常多，大家也都不需要记住，需要用到的时候再去官网阅读说明然后按照要求填写对应的参数即可。并且还可以通过不断的改变参数的值来调试最终的效果。

在 mapbox 中显示行政区的名称要比在 leaflet 中简单一些，那就是再叠加一个 `symbol` 图层专门用于显示文字，并且显示文字的位置也不用我们自己指定，mapbox 会帮助我们自动计算多边形的几何中心，并且将文字置于几何中心，但我们还是要给 mapbox 指定要显示的文字字段，这个字段就是数据中的 `properties` 里面的某个属性，比如说我们在这里就使用 `properties` 里的 `name` 字段：

```
map.addLayer({  
    id: "xzq-text",  
    type: "symbol",  
    source: "xzq-source",  
    paint: {  
        "text-color": "red",  
    },  
    layout: {  
        "text-field": "{name}",  
    },  
});
```



图 6.3.3-展示行政区名称

如何将行政区颜色进行色彩分类? (如何将行政区展示成五颜六色的效果?) 这一点跟在 leaflet 中的操作是不同的, 在 mapbox 中需要用到表达式的概念去操作。什么是表达式? 其实就是一个或者一系列的条件, mapbox 框架通过解析这些条件去帮你执行某些操作, 其

实就相当于我们在写 leaflet 分层设色的时候自己写的那些 if/else 条件现在 mapbox 框架帮我们做了，但是你要把条件的取值告诉 mapbox。比如现在同样还是要对行政区颜色进行划分。只需要改动 fill-color 这个属性就行了，这个属性除了能传递一个静态的颜色字符串之外还可以传递一个表达式，具体这么写：

```
map.addLayer({
  id: "xzq-layer",
  type: "fill", //类型为面状要素 fill
  source: "xzq-source",
  paint: {
    "fill-color": [
      "match",
      ["get", "name"],
      "建德市",
      "green",
      "淳安县",
      "white",
      "桐庐县",
      "blue",
      "yellow",
    ],
    "fill-opacity": 0.8,
  },
});
```

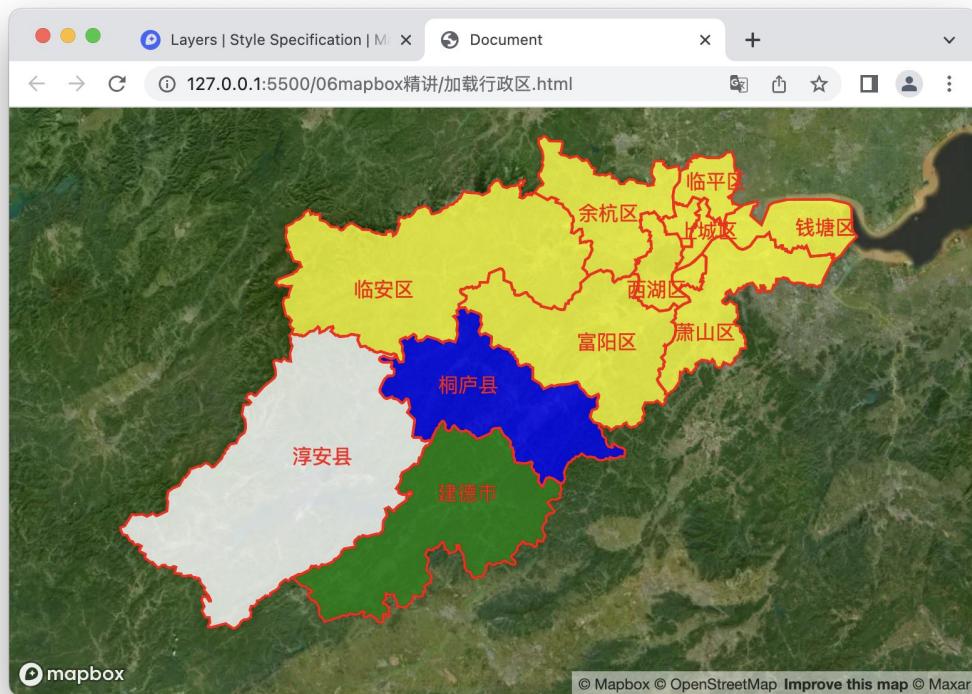


图 6.3.4-分级着色

跟大家解释一下表达式的具体语法，首先表达式通常是一个数组的形式，数组的第 0 项，也就是初始项表示表达式的模式，模式一共有很多种。像匹配模式，相等模式、大于等于、小于等于、`like` 模糊、等等。我们这里使用的是匹配模式 `match`，就是告诉 mapbox 我们的颜色是按照数据中的某个属性进行匹配的，第二个参数是一个数组，第二个参数整体表示要使用哪个属性进行匹配，这里是通过框架内置的方法 `get` 获取到对应的属性 `name`，也就是说我们是使用属性 `properties` 里的 `name` 字段进行颜色匹配的，后面的参数就是一一对应了，什么字段对应什么颜色需要我们自己写，比如“桐庐县”要对应蓝色就是“桐庐县”，“blue”这样。要注意如果你仅仅对其中的某几个行政区匹配了颜色，那么你需要在最后写一个默认的颜色用于表示没有被匹配到的行政区的颜色，就比如说上面的例子中其实我们只为 3 个行政区匹配了颜色，剩下的行政区我们没写颜色，那么我们需要在最后写上一个默认颜色 `yellow` 来表示没有匹配到的都是黄色。

除了匹配模式 `match` 之外还有很多常用的表达式，我们需要使用的到的时候时候可以去官网进行详细了解，在这里再给大家举一个例子，比方说现在我们如果需要根据某个指标的数值来定义颜色该如何做？就比如现在我们调查清楚了全国各省拥有的茶企业数量，我们想根据每个省拥有的茶企业的数量来排个名，在地图上就是通过颜色的深浅来区分不同省份的茶企业的数量多少。那么我们就需要用到阶梯分段表达式 `step`。具体的写法如下：

```
map.addLayer({
  id: "china",
  source: "china",
  type: "fill",
  paint: {
    "fill-color": [
      "step",
      ["get", "count"],
      "rgb(190, 216, 205)",
      2,
      "rgb(164, 206, 165)",
      5,
      "rgb(151, 202, 148)",
      8,
      "rgb(95, 181, 108)",
      10,
      "rgb(91, 174, 103)",
      20,
      "rgb(81, 154, 92)",
      40,
      "rgb(74, 141, 84)",
      50,
      "rgb(182, 213, 193)",
      60,
      "rgb(70, 134, 80)",
      ],
    },
  });
});
```

我们通过观察代码可以看到，在写表达式的时候我们的模式指定成了 `step`，要参与阶梯分类的字段是数据中的 `count` 字段，然后我们可以根据自己的设计将 `count` 分成几个阶梯，并且给每个阶梯赋值一定的颜色，这样一来 `mapbox` 框架就会根据我们的数据不同，自动渲染为不同的颜色。实现的效果如下图：



图 6.3.5-阶梯分类

从 mapbox 的表达式的设计也不难看出，mapbox 官方分想法可能是好的，将我们日常所需要用到的所有东西能封装起来的就尽可能的封装起来，但是殊不知有的时候封装的过于复杂也不见得是件好事，因为随着代码的封装程度的不断提高，API 会越来越丰富，丰富到不好记忆，甚至出现难以理解的现象，因此有的时候车轮也不能够造的过于圆滑还是要留给开发者足够的空间。这也印证了我在本章第一节中就讲过的 mapbox 极大程度的限制了 API 的使用，让开发者有很少的选择，这一点我觉得是不如 openlayers 的地方。

## 第 4 节：marker 与 dom

在实际的项目开发中，一些点位（兴趣点 POI）的展示需求也是必不可少的。那么这节我们来带着大家加载一下这些 poi 的数据，在本书附带的代码文件夹中给大家准备了一份 poi 数据。这份数据只是一个简单的 json 数据。叫做 poi.js，像下面这样：

```

var poi = [
{ name: "希望小学", lon: "124.789412", lat: "47.347325", type: "school" },
{ name: "蜜雪冰城", lon: "103.007012", lat: "39.53535", type: "school" },
{ name: "天猫超市", lon: "117.872028", lat: "27.100653", type: "market" },
{ name: "如家酒店", lon: "124.642233", lat: "48.042886", type: "market" },
{ name: "人民医院", lon: "87.038159", lat: "31.42261", type: "hospital" },
{ name: "和平饭店", lon: "117.038159", lat: "31.62261", type: "school" },
{ name: "银泰城", lon: "121.038159", lat: "29.42261", type: "market" },
{ name: "海底捞", lon: "111.038159", lat: "28.42261", type: "school" },
];

```

没什么大不了的，就是一个普通的 json 数组。数据里面包含经纬度信息。到时候用于确定位置。有一个 type 字段到时候用于区分 poi 的类型。因为可能会出现不同类型的 poi 对应不同的图标的需求。

好了第一步我们就要想如何加载这个数据，我们可以用 symbol 的图层类型来加载这是毫无疑问的。但是我们的 symbol 图层也只是支持 geojson 格式的数据。这种普通的 json 是不支持的，那我们第一步就是需要先把普通的 json 转化成 geojson 了。普通的 json 和 geojson 之间有什么不同？不就是 geojson 那些字段比较固定吗？什么 type 是 feature, properties 里面存储属性，geometry 里面存储图形信息，因此我们可以构造一个 geojson 模版然后使用遍历的方式把这些点位都转换成 geojson，我是采用下面的转换方法的，各位也可以根据自己的思路来写：

```

function convert2Geo(json) {
  const result = { type: "FeatureCollection", features: [] };
  json.forEach((j) => {
    const feature = {
      type: "Feature",
      properties: {},
      geometry: { type: "Point", coordinates: [] },
    };
    feature.properties = j;
    feature.geometry.coordinates = [j.lon, j.lat];
    result.features.push(feature);
  });
  return result;
}

```

那么我们现在只需要把普通的 json 当作参数传入到这个函数中，即可完成 geojson 的转换。接下来我们就需要使用 symbol 的方式加载图标类型图层了。加载之前要跟大家说一个 mapbox 里面的规则。如果你想加载一个图标类图层，那么你首先需要通过 map.loadImage()方法把这个图标资源请求回来。然后使用 map.addImage()方法把这个图标加载到地图上之后，才能使用加载 source 和加载 layer 那一套逻辑。因此我们需要把加载图层的逻辑放在图片加载成功的回调函数里。

```

map.on("load", (e) => {
  const poigeo = convert2Geo(poi);
  map.loadImage("../icons/a.png", (error, image) => {
    map.addImage("a", image);
    map.addSource("poi-source", { type: "geojson", data: poigeo });
    map.addLayer({
      id: "poi-layer",
      source: "poi-source",
      type: "symbol",
    });
  });
}

```

```
layout: {
  "icon-image": "a",
  "icon-size": 0.6,
},
});
});
});
```

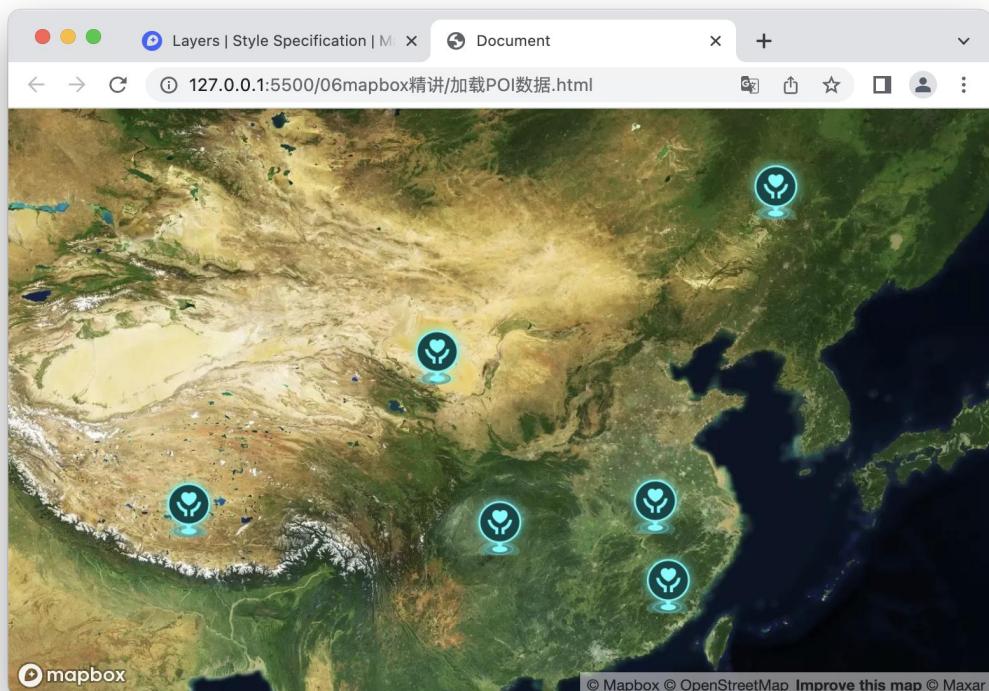


图 6.4.1-mapbox 加载图标图层

可以看到图标加载成功，不过现在的图标只有一种类型。如果我们想通过数据中的 type 字段来区分图标的类型该怎么办呢？答案还是使用我们的表达式，不过在使用表达式之前，你需要把所有会用到的图标先通过 `loadImage` 方法加载到地图上，你可以写写循环遍历添加。也可以绑定到数据里（依据具体场景定）。总之你需要把图标都添加到地图上并且给每一个图标起一个唯一的 ID。然后把 `icon-image` 这个参数改成一个表达式即可。

```
map.loadImage("../icons/a.png", (error, image) => {
    map.addImage("a", image);
});
map.loadImage("../icons/b.png", (error, image) => {
    map.addImage("b", image);
});
map.loadImage("../icons/c.png", (error, image) => {
    map.addImage("c", image);
});
map.addSource("poi-source", { type: "geojson", data: poigeo });
map.addLayer({
    id: "poi-layer",
    source: "poi-source",
    type: "symbol",
    layout: {
        "icon-image": [
            "match",
            ["get", "type"],
            "hospital",
            "a",
            "school",
            "b",
            "c",
        ],
        "icon-size": 0.6,
    },
});
```

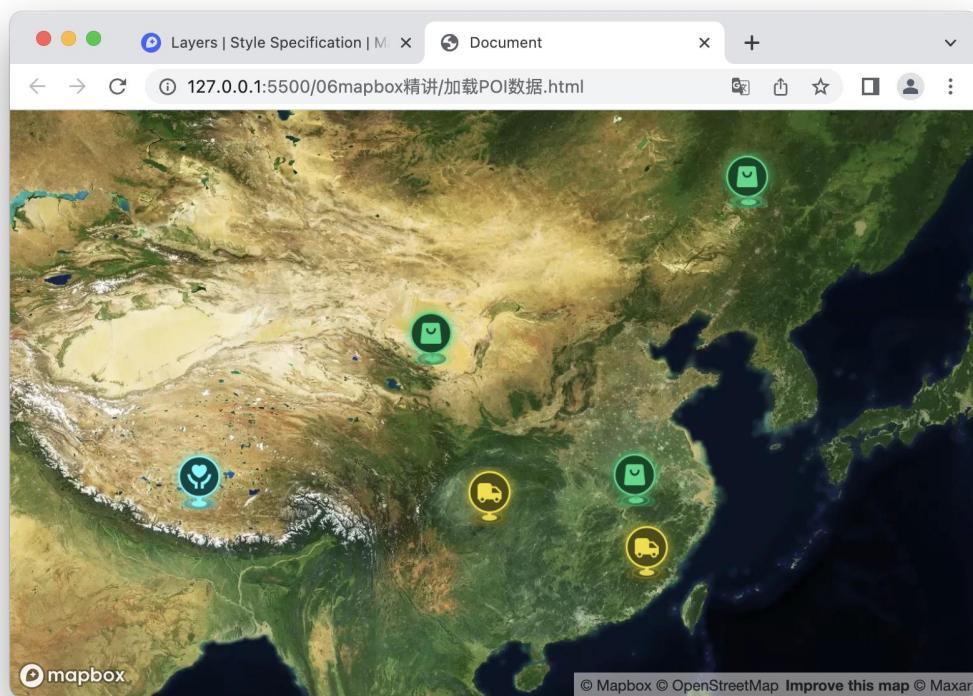


图 6.4.2-mapbox 图标分类实现

这里的分类的表达式和之前行政区分级着色的道理是一样的, 因此在这里就不重复讲解了。OK 到此位置我们实现了 `poi` 数据的加载和分类展示的功能。关于 `poi` 的一些交互和事件我们将在下一小节进行讲解。

## 第 5 节：交互及事件监听

这一小节我们来了解一下 `mapbox` 当中的一些交互和事件, 我们通过对前两个小节的案例的完善来跟大家介绍 `mapbox` 中常用的一些交互和事件。

首先对于行政区经常性的会有这样一个需求, 就是点击某个行政区的时候这个行政区高亮, 在 `mapbox` 中点击高亮有两种方式可以做。一种是按照常规逻辑, 在用户点击的区域再叠加一个不同颜色的要素 `feature` 即可, 另一种是使用 `mapbox` 中的 `filter`。

我们先来说常规方法也是最好理解的方法, `mapbox` 框架为我们提供了地图点击的监听叫做 `click`。这个监听会带有一个回调。回调函数中会把我们点击的要素作为参数返还给我们:

```
map.on("click", "xzq-layer", (e) => {
  console.log(e.features[0]);
});
```

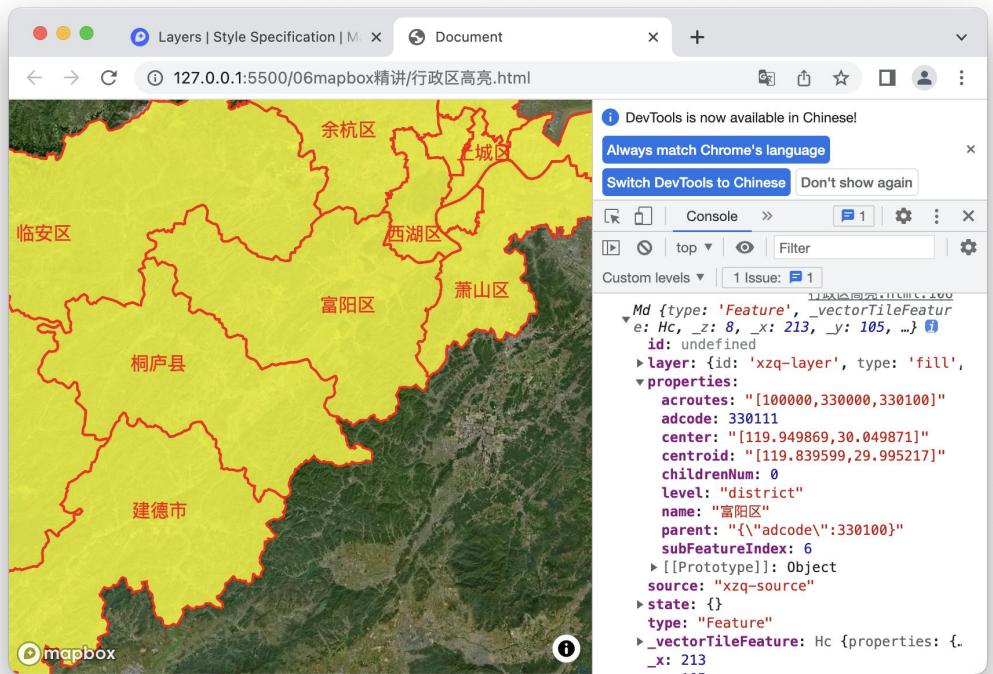


图 6.5.1-点击拾取要素

我们在控制台可以看到，当我用鼠标点击“富阳区”这个要素的时候控制台打印出了这个要素。也就是说框架会帮我们返回我们点击到的要素。那么我们只需要把这个要素再次以另一种样式添加到地图上就可以了。但是要注意：**mapbox 中所有的图层 layer 和数据源 source 都必须有一个唯一的 ID，也就是同一 ID 的数据源或图层不允许重复添加，两次添加的 source 的 id 如果一样就会报错**。因为我们的点击高亮是随着用户切换行政区，高亮颜色跟随变化的，所以我们需要在每次高亮之前先清除掉上一次的高亮。这样的话我们就必须在添加这个高亮要素之前先清除掉高亮所对应的 source 和 layer。

```
map.on("click", "xzq-layer", (e) => {
  console.log(e.features[0]);
  if (map.getLayer("xzq-hl")) {
    map.removeLayer("xzq-hl");
    map.removeSource("hl-source");
  }
  map.addSource("hl-source", { type: "geojson", data:
  e.features[0] });
  map.addLayer({
    id: "xzq-hl",
    type: "fill",
    source: "hl-source",
    paint: {
      "fill-color": "lightblue",
      "fill-opacity": 0.7,
    },
  });
});
```

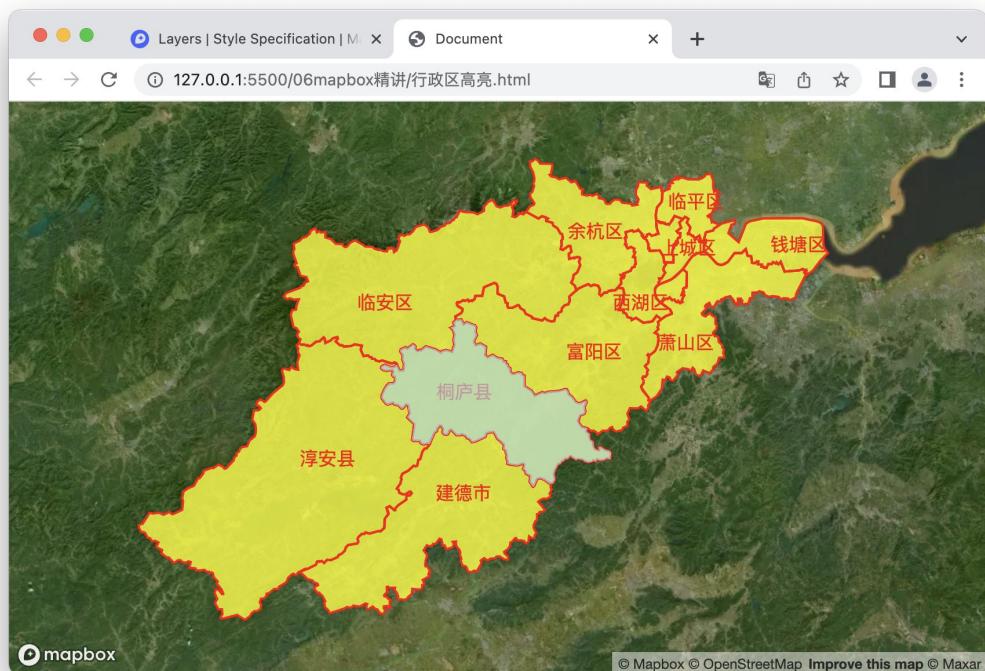


图 6.5.2-点击高亮

还有第二种方式就是使用 **mapbox** 官方为我们提供的思路: **filter**。它的原理是这样的,

首先和行政区一样同时在地图上加载一个高亮图层。但是这个高亮图层是不显示的，如何让它不显示？只需要给这个图层设置一个 `filter` 条件，当这个条件不满足的时候图层自然就不会显示。因为 mapbox 官方对于 `filter` 的解释就是，满足 `filter` 条件的要素会展示在地图上，不满足条件的则不会展示在地图上。那如何高亮呢？那就是在点击要素的时候。获取要素的属性，然后重新修改之前隐藏的图层的 `filter` 条件，这样就能让它重新展示。具体的代码如下：

```
map.on("click", "xzq-layer", (e) => {
  map.setFilter("xzq-filter", [
    "in",
    "name",
    e.features[0].properties.name,
  ]);
});
```

这样写起来比较简单。也不用考虑每次变换要清除上一次高亮的元素，因为这一点框架底层已经为我们处理掉了，但是这种方式可能不是很好理解，需要大家多揣摩一下。

说完了行政区的高亮问题我们再来说说弹窗展示属性的问题。无论是行政区还是 `poi` 兴趣点都会存在这样的需求，点击某个要素的时候弹窗展示一些东西。实际上在 mapbox 中的弹窗还是 `dom` 元素，也就是说我们的 `canvas` 元素上层是允许叠加 `dom` 元素的，你可以把一个 `div` 放在 `canvas` 元素上面。那么我们就使用 mapbox 中为我们提供的 `popup` 即可。

```
map.on("click", "poi-layer", (e) => {
  new mapboxgl.Popup()
    .setLngLat(e.features[0].geometry.coordinates)
    .setHTML("<div>" + e.features[0].properties.name + "</div>")
    .addTo(map);
});
```

上面这段代码应该很好理解吧。首先创建一个 `popup` 弹窗对象。然后给这个弹窗设置一个展示时的经纬度，在给这个弹窗设置一个展示的内容 `html` 字符串。最后把这个弹窗添加到地图上。

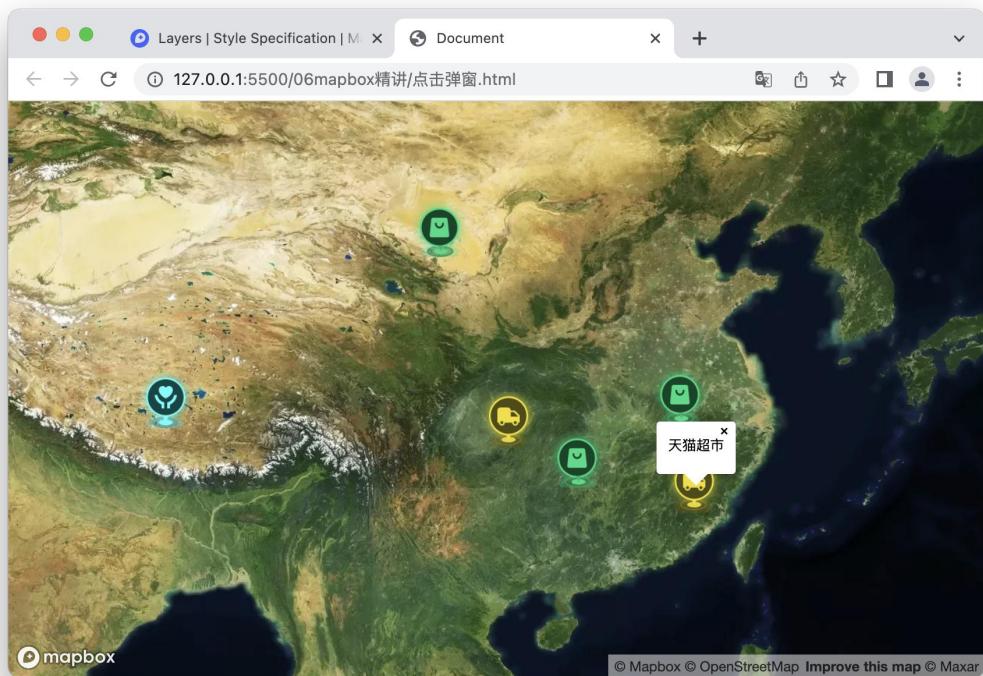


图 6.5.3-点击弹唱展示信息

这样我们就实现了点击要素的时候弹窗展示一些属性内容，对于行政区也是类似操作在这里我就省略了，大家有兴趣可以自己下去练习一下。另外我们也像讲 leaflet 的时候一样，引导大家思考，如果现在用户改变了交互方式，比如用户想当鼠标悬浮到图标上的时候展示弹窗，而鼠标离开的时候弹窗关闭，请大家思考这个需求应该怎么做。

上面讲的内容针对的是对于图层的交互操作，针对于地图本身也有许多交互和操作，比如我们都知道控制地图缩放的参数是 `zoom`，通常取值是 0-22 左右。控制地图拖拽的参数是 `dragpan`。比如说现在我们如果想当地图放大到一定层级的时候隐藏掉某个图层，缩小到某个层级的时候再显示这个图层，该如何操作呢？那我们第一步首先还是要监听地图的缩放结束事件“`zoomend`”，然后在这个事件回调中写隐藏掉图层的逻辑：

```
// 监听地图的缩放事件
map.on("zoomend", (e) => {
  const currentZoom = map.getZoom();
  if (currentZoom > 8) {
    map.setLayoutProperty("xzq-layer", "visibility", "none");
  } else {
    map.setLayoutProperty("xzq-layer", "visibility", "visible");
  }
});
```

最后各位在使用 `mapbox` 开发移动端项目时要注意，移动端的事件监听和 PC 端是不同的。比如在 PC 端点击地图的事件是 `click`，而在移动端点击地图的事件是 `touchend`，这个细节需要格外注意，具体的其他的事件可以参考 `mapbox` 官网上对于事件的介绍。

<https://docs.mapbox.com/mapbox-gl-js/api/events/>

再比如现在假设有这样一个需求，很多小程序内需要一个在地图上通过点击拾取点位从而自动获取经纬度的功能，如果我们使用 `mapbox` 来做的话，只需要监听地图的点击事件 `click` 就好了，当用户点击完地图之后，通过 `e.lngLat` 获取到用户点击位置的经纬度，将经纬度发送给需要的地方即可，例如可以将经纬度作为参数来获取高德的地址定位接口数据，从而拿到该经纬度对应的地址信息。

```
const map = new mapboxgl.Map({
  container: "map",
  center: [120.536363, 29.243242],
  zoom: 8,
  style: "mapbox://styles/mapbox/satellite-v9",
});
map.on("click", (e) => {
  alert("当前的经纬度位置: " + e.lngLat);
});
```

## 第 6 节：专题地图制作

关于专题地图的制作其实我们在本章的第 3 节已经介绍过一部分了，就是我们常规的按照不同的颜色去对要素进行分类，我们称作是分类着色。这也算是专题图中的一种。接下来我们给大家介绍一些更加深入的专题图制作的案例。

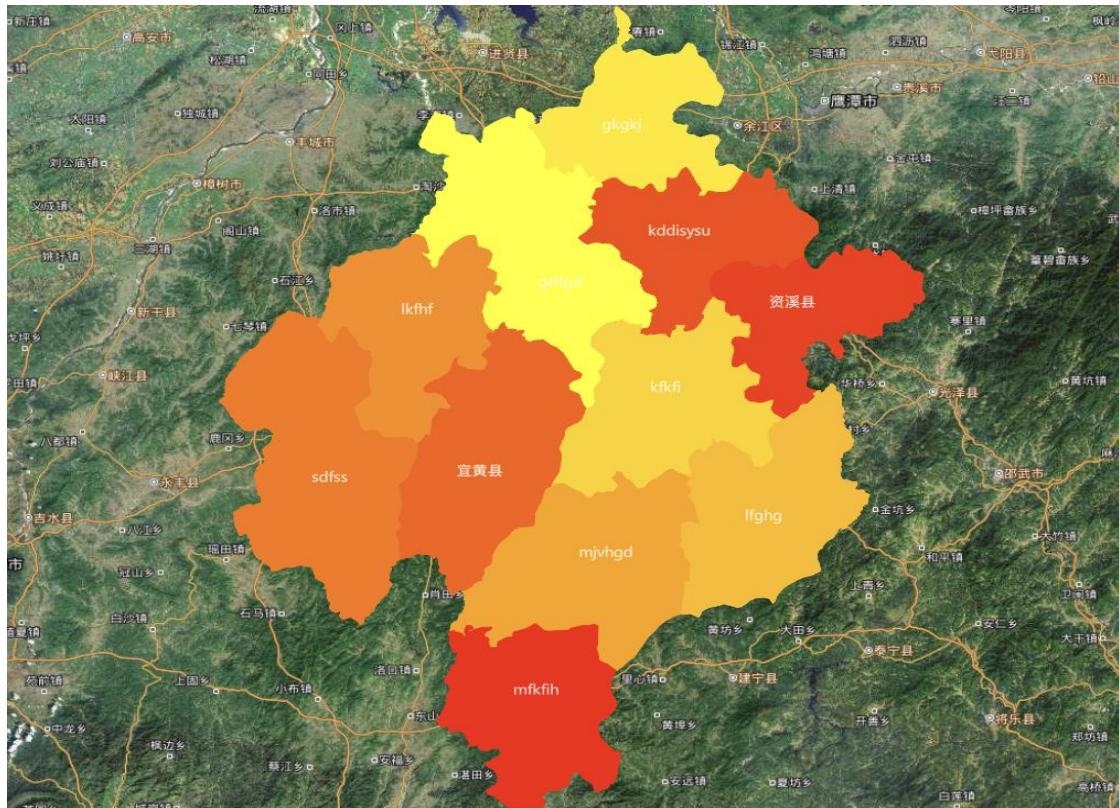


图 6.6.1-分类着色专题图

### 1. 热力图的制作

热力图要用到 mapbox 中 7 大图层类型 (第 3 节讲过) 之一的 heatmap 类型。和分类着色不同的是，heatmap 通常要表示的就是不同位置不同的属性值来渲染不同的颜色，因为将数据细化成了点，因此才更加容易的看出空间位置上的分布状况，哪里聚集，哪里分散一目了然。在 mapbox 中如果要使用热力图类型的图层，需要你传入一些点位数据，像下面这样：

```
var tentea = {
    type: "FeatureCollection",
    features: [
        {
            type: "Feature",
            properties: {
                name: "西湖龙井",
                dis: "杭州市西湖区",
                temp: 19.21,
                rank: 2,
            },
            geometry: {
                type: "Point",
                coordinates: [120.07637104262216, 30.110987373108287],
            },
        },
        {
            type: "Feature",
            properties: {
                name: "安溪铁观音",
                dis: "泉州市安溪县",
                temp: 19.03,
                rank: 3,
            },
            geometry: {
                type: "Point",
                coordinates: [117.93324898736677, 24.798769462568657],
            },
        },
    ],
};
```

然后我们在添加图层的时候要这样写:

```
map.addSource("heat", { type: "geojson", data: tentea });
map.addLayer({
  id: "heat",
  type: "heatmap",
  source: "heat",
  paint: {
    "heatmap-weight": [
      "interpolate",
      ["linear"],
      ["get", "temp"],
      15.38,
      0,
      20.45,
      1,
    ],
    "heatmap-intensity": [
      "interpolate",
      ["linear"],
      ["zoom"],
      0,
      1,
      19,
      13,
    ],
  },
});
```

```
"heatmap-color": [
    "interpolate",
    ["linear"],
    ["heatmap-density"],
    0,
    "rgba(255, 254, 191,0)",
    0.2,
    "rgb(91, 182, 255)",
    0.4,
    "rgb(0, 48, 255)",
    0.6,
    "rgb(255, 202, 114)",
    0.8,
    "rgb(255, 121, 60)",
    1,
    "rgb(255, 18, 5),
],
"heatmap-radius": ["interpolate", ["linear"], ["zoom"], 0, 15,
19, 110],
"heatmap-opacity": ["interpolate", ["linear"], ["zoom"], 7, 1,
19, 0],
},
});
```

跟大家来解释以下这些关键的参数含义：首先"heatmap-weight"代表着热力图的权重，也就是说哪个字段，哪个值决定热力图的颜色，我这里使用的是数据中的 temp (温度) 字段，"heatmap-intensity"代表着热力图的密度，通常由缩放层级来决定，当然你也可以自己指定这个值的变化依据。"heatmap-color"代表着热力图的颜色，这里使用的是按照密度来定义颜色，不同的密度值对应不同的颜色。"heatmap-radius"和"heatmap-opacity"分别代表着热力图的圆圈半径和透明度，也是有缩放层级来决定，通常我们希望地图放大时能够显示更大的半径以便用户看的清楚。

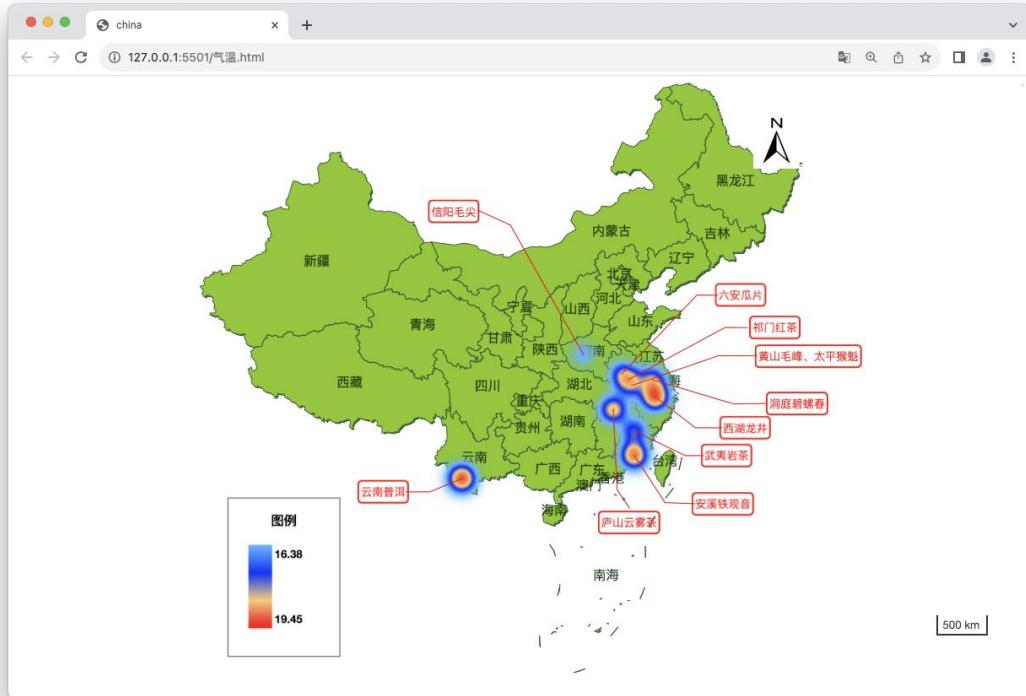


图 6.6.2-热力图

## 2. 聚合图的制作

聚合图实现的方式有很多种，随着大家深入的学习应该也能使用多种方式来实现。我们在这里跟大家介绍一种 mapbox 原生的方式。即我们在加载一些点位图层的 `source` 时候在配置参数里面增加一个 `cluster` 参数，类似于下面这样：

```
map.addSource("earthquakes", {
  type: "geojson",
  data: point,
  cluster: true,
  clusterMaxZoom: 11,
  clusterRadius: 100,
});
```

同时还可以指定聚合的最大缩放层级和最小缩放层级，意思就是当达到某个缩放层级的时候才执行聚合，否则就不聚合。还可以指定聚合的半径 `clusterRadius`，这个根据具体的项目需求来调整。

添加一个聚合图层的 `layer` 代码如下：

```
map.addLayer({
  id: "clusters",
  type: "circle",
  source: "earthquakes",
  filter: ["has", "point_count"],
  paint: {
    "circle-color": [
      "step",
      ["get", "point_count"],
      "rgba(255,218,60,1)",
      20,
      "rgba(255,218,60,1)",
      50,
      "rgba(255,218,60,1)",
      ],
    "circle-radius": [
      "step",
      ["get", "point_count"],
      14,
      100,
      20,
      200,
      28,
      500,
      28,
      ],
    },
  });
});
```

这段代码中需要注意的是"point\_count"这个值，注意它不是存在数据中的某个字段，而是 mapbox 框架帮我们统计出来的周围的点的数量值，因此这个值是不变的，也不能变，只能这么写。然后剩下的参数含义我想大家经过上一个热力图的学习应该也能够看懂了。

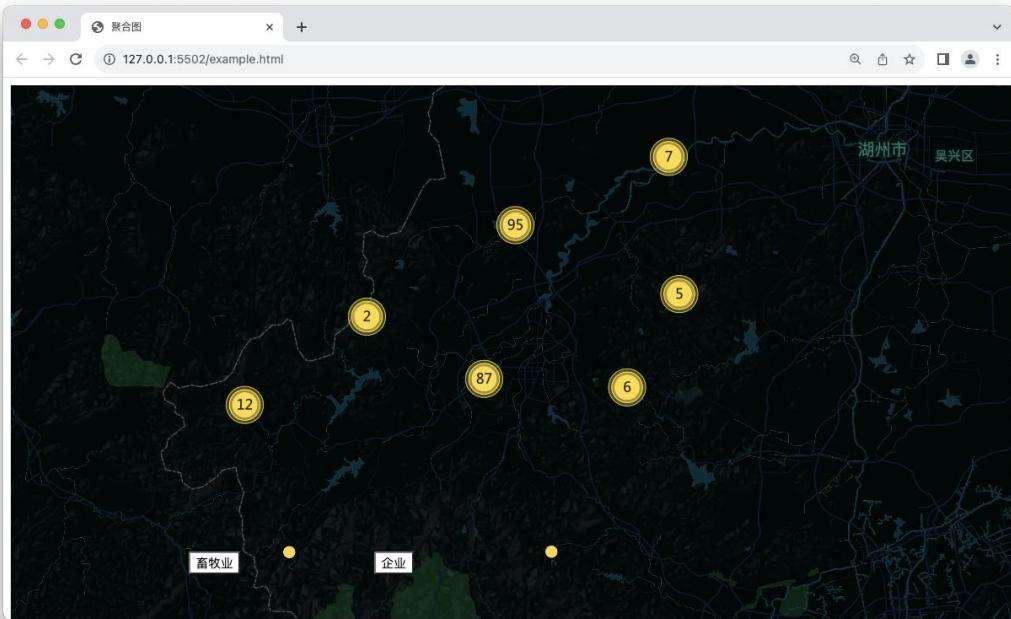


图 6.6.3-聚合图

### 3. 迁徙图的制作

迁徙图相对来说比较麻烦，当然如果不愿意从 WebGIS 的角度去做这件事的话，可以选择使用市面上比较流行的图标工具库，例如 echarts 等。如果使用 WebGIS 的方式我们需要这样做，首先要准备迁徙和流向的城市数据，比较简单，只需要城市名字和对应的经纬度位置即可。就像下面这样：

```
var citys = {
    北京: [116.404269, 39.915156],
    大连: [121.615878, 38.927457],
    杭州: [120.153293, 30.343855],
    南京: [118.773496, 32.083122],
    上海: [121.477898, 31.264951],
    厦门: [118.097395, 24.494698],
    福州: [119.308167, 26.084711],
    成都: [104.125798, 30.651546],
    重庆: [106.62783, 29.628137],
};
```

为了方便接下来的讲解和大家的理解。我们不妨先把结果图放在这里：

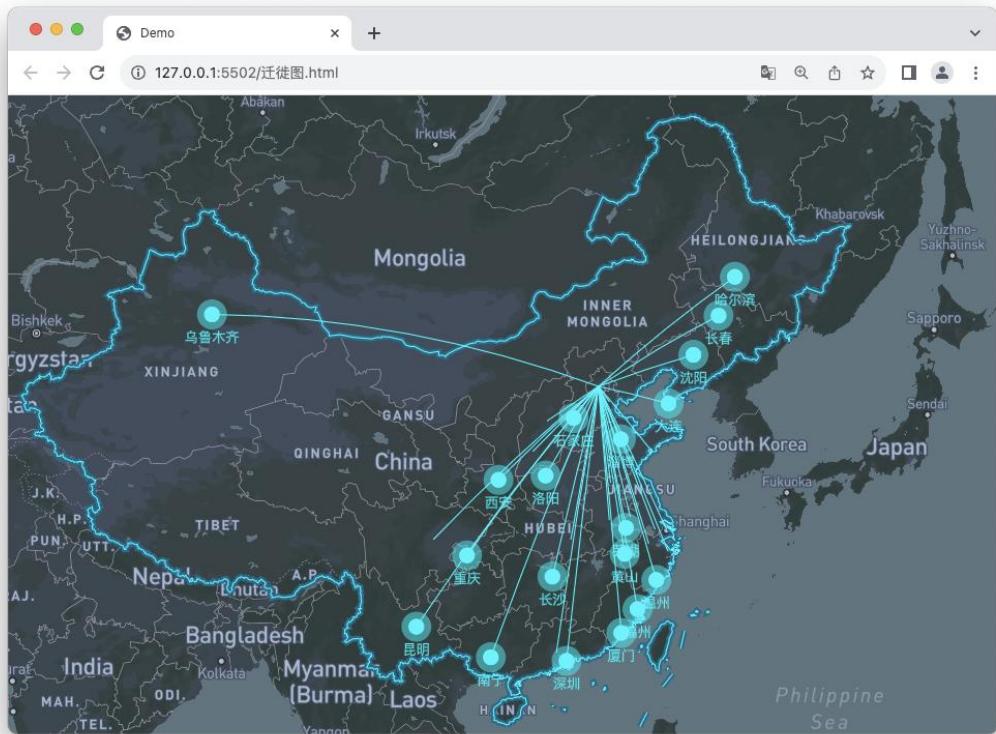
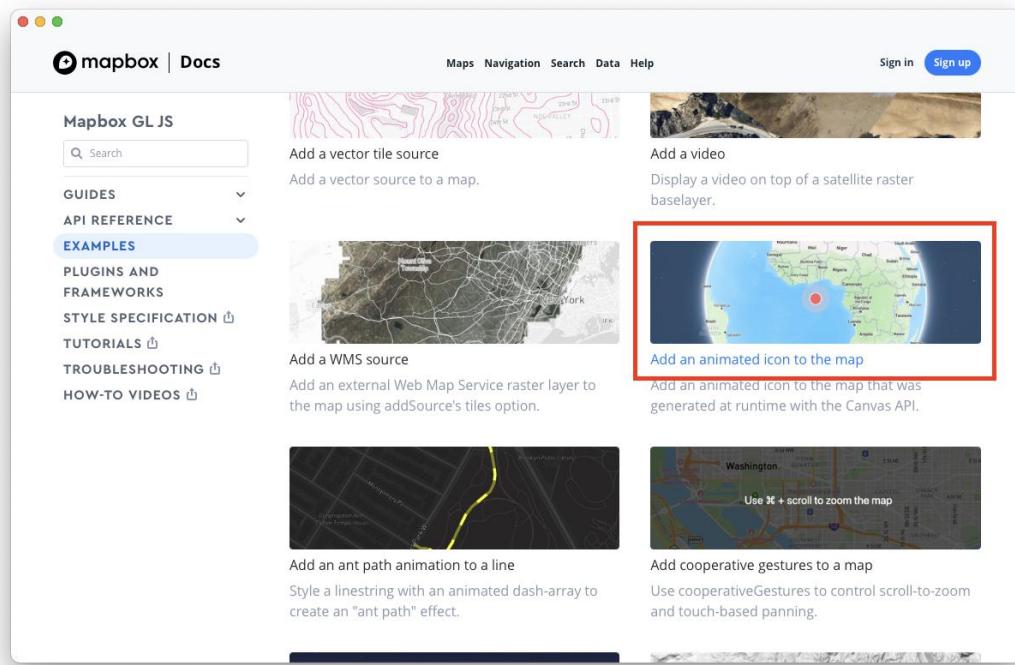


图 6.6.4-迁徙图

第二步我们需要对渲染的图层进行分类，实际上我们可以按照两个部分来渲染，第一部分是这些城市的点位，第二部分是点位之间的路径渲染，相当于一个 `point` 图层，一个 `line` 图层。但是这里面有很多细节需要注意，第一是我们尽可能的要把这种图做的比较炫酷，因此最好点位带一点动态效果，这一点 `echarts` 也是这样做的。第二是点位与点位之间的路径连线要尽可能的优美圆滑。除此之外我们还必须清楚一个道理，那就是迁徙图（流向图）意味着只有一个起点和多个终点，或者只有一个终点和多个起点。大家好好揣摩这句话。

理解了这一点之后我们就分开来写代码，首先先写点位渲染的，这里我们借助了官网上案例，这也是之前跟大家讲的如果没有灵感，可以上官网的案例进行参考：



具体的点位渲染的代码如下：

```
//构建动态点
function generatePoint(points) {
  const size = 80;
  const pulsingDot = {
    width: size,
    height: size,
    data: new Uint8Array(size * size * 4),
    onAdd: function () {
      const canvas = document.createElement("canvas");
      canvas.width = this.width;
      canvas.height = this.height;
      this.context = canvas.getContext("2d");
    },
    render: function () {
      const duration = 1000;
      const t = (performance.now() % duration) / duration;
      const radius = (size / 2) * 0.3;
      const outerRadius = (size / 2) * 0.7 * t + radius;
      const context = this.context;
    }
}
```

```
context.clearRect(0, 0, this.width, this.height);
context.beginPath();
context.arc(
  this.width / 2,
  this.height / 2,
  outerRadius,
  0,
  Math.PI * 2
);
context.fillStyle = `rgba(0, 246, 255, ${1 - t})`;
context.fill();
context.beginPath();
context.arc(this.width / 2, this.height / 2, radius, 0, Math.PI
* 2);

context.fillStyle = "rgba(0, 246, 255, 1)";
context.strokeStyle = "rgba(0, 246, 255, 1)";
context.lineWidth = 2 + 4 * (1 - t);
context.fill();
context.stroke();
// Update this image's data with data from the canvas.
this.data = context.getImageData(0, 0, this.width,
this.height).data;
map.triggerRepaint();
return true;
},
};

map.addImage("pulsing-dot", pulsingDot, { pixelRatio: 2 });
for (let i in points) {
  map.addSource("dot-point" + i, {
    type: "geojson",
    data: {
      type: "FeatureCollection",
      features: [
        {
          type: "Feature",
          properties: { name: i },
          geometry: {
            type: "Point",
            coordinates: [points[i].lon, points[i].lat]
          }
        }
      ]
    }
  });
}

// Create a pulsing dot
function pulsingDot(t) {
  const radius = 10;
  const outerRadius = 15;
  const width = 100;
  const height = 100;
  const map = this;
  const context = this.getContext("2d");
  const data = this.data;
  const points = this.points;
  const pulsingDot = document.createElement("img");
  pulsingDot.src = "pulsing-dot.png";
  pulsingDot.style.position = "absolute";
  pulsingDot.style.left = 0;
  pulsingDot.style.top = 0;
  pulsingDot.style.width = width + "px";
  pulsingDot.style.height = height + "px";
  pulsingDot.style.zIndex = 1000;
  pulsingDot.style.pointerEvents = "none";
  pulsingDot.style.filter = "blur(10px)";
  pulsingDot.style.opacity = 0;
  pulsingDot.style.transition = "filter 1s ease-in-out, opacity 1s ease-in-out";
  pulsingDot.style.display = "block";
  pulsingDot.style.margin = "auto";
  pulsingDot.style.borderRadius = "50%";

  this.pulsingDot = pulsingDot;
  this.map = map;
  this.context = context;
  this.data = data;
  this.points = points;
  this.radius = radius;
  this.outerRadius = outerRadius;
  this.width = width;
  this.height = height;
  this.t = t;
}
```

```

    type: "Point",
    coordinates: points[i], // icon position [lng, lat]
  },
},
],
},
},
});
map.addLayer({
  id: "layer-with-pulsing-dot" + i,
  type: "symbol",
  source: "dot-point" + i,
  layout: {
    "icon-image": "pulsing-dot",
    "icon-allow-overlap": true,
    "text-field": "{name}",
    "text-anchor": "top",
    "text-size": 12,
    "text-offset": [0, 1],
  },
  paint: {
    "text-color": "rgba(0, 246, 255, 1)",
  },
});
}
}

```

需要注意的是，点位的渲染本质上用到了 `canvas` 的知识，没错就是我们之前一直在铺垫的 `canvas`。这样看来其实我们讲的底层原理都是很有用的，基础打好之后后面的学习很轻松。我们之前说过 `mapbox` 的底层渲染原理就是使用的 `canvas`，那么当然 `mapbox` 可以随意的操作 `canvas`，并将其与地图相结合，实现各种动态效果，上面这段代码的核心就是使用 `canvas` 绘制一个动态的圆，关于如何使用 `canvas` 来绘制动态的圆以及各种动画各位可以关注我的抖音，有相关的免费教程，在这里我就不展开讲了。

接下来我们来说下路径渲染的思路，我们首先要借助 `turf.js` 来学会如何根据两个点生成一条线，其次我们要学会如何让一段曲线变得更加圆滑，根据起点和终点生成线段我们可以使用 `turf.js` 中的 `greatCircle` 这个函数。而将线段变得圆滑我们可以使用 `bezierSpline` 这个函数。这样的话我们可以使用循环的方式，比如以北京为起点，那么剩下所有的点都是终

点，使用循环不断的生成线段，再使其变的圆润，就完成了我们路径的渲染。

```
function generateTrack(data) {
  const start = turf.point(Object.values(data)[0]);
  for (let i in data) {
    var end = turf.point(data[i]);
    var greatCircle = turf.greatCircle(start, end, {
      properties: { name: "Seattle to DC" },
      npoints: 500,
    });
    var curved = turf.bezierSpline(
      turf.lineString(greatCircle.geometry.coordinates)
    );
    map.addSource("circle" + i, { data: curved, type: "geojson" });
    map.addLayer({
      id: "circle-layer" + i,
      type: "line",
      source: "circle" + i,
      paint: {
        "line-color": "rgb(0,246,255)",
        "line-width": 1,
      },
    });
  }
}
```

## 第 7 节：图形绘制与测量

图形的绘制也是 WebGIS 开发过程中非常常见的功能，这节我们教给大家如何封装一个绘制组件和测量组件，后续可以不断复用。并且给大家解释清楚其中的原理。我们本小节给大家提供两种方式来实现图形绘制。

首先我们要清楚在 mapbox 中绘制的功能是被官方已经封装好的，并且以插件的形式引入使用。如果想使用此插件，必须在 html 中引入 mapbox-gl-draw.js 文件。

mapbox 官方已经为各位写好了按钮的 css 样式，只要通过 new MapboxDraw()方法实例化之后，就可以在页面上看到绘制相关的按钮，并且可以直接使用。如果不喜欢单方提

供的样式可以自己修改样式或者另写按钮，然后调用此按钮的方法即可。这里就要考验各位的 css 功底了。

官方的 draw 插件具体用法: <https://github.com/mapbox/mapbox-gl-draw>

使用组件的核心代码很简单，就几行代码：

```
const draw = new MapboxDraw({
  displayControlsDefault: false,
  controls: {
    polygon: true,
    lineString: true,
    point: true,
    trash: true,
  },
});
map.addControl(draw);
```

完整的 demo 我放在附带源码文件夹里了，各位可以参考研究。

如果想要自己写一个绘制的组件其实也不难，这中间要涉及到很多对于事件监听的理解。实际上绘画就是一个按照步骤操作的过程，我们用绘制一条线来举例，首先需要开启地图的 click 监听，当第一次点击地图时触发，记录用户点击位置的坐标，然后随着用户鼠标移动，下一次点击时再次记录坐标，这样就在两点之间连成一条线，线段的绘制就完成了绘制面要素的道理其实是相同的。只不过最后一步要加上一个多边形闭合的操作，将首尾坐标相连即可。

测量其实就是绘制增加了一个计算的操作。在绘制结束之后计算绘制线条的长度或者绘制多边形的面积，计算的方式有很多种，可以使用现有的库比如 turf.js (我们的老盆友了)。或者也可使用经纬度换算距离的公式，二者皆可看自己喜好。因为代码高度类似，在这里我们只提供测量的代码给大家，如果大家想要用作图形绘制，只需要把计算的部分删掉即可。

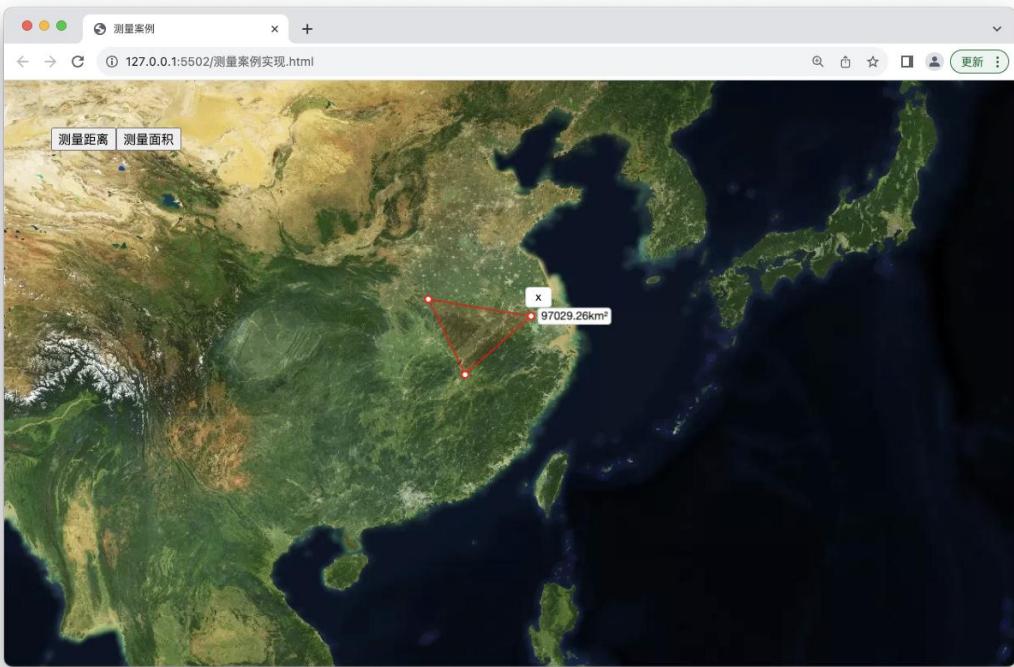


图 6.7.1-测量案例的实现

## 第七章：Cesium 精讲

### 第 1 节：基础入门

Cesium 作为现在几乎必选的 WebGIS 三维框架，已经被广大 GIS 开发者所熟知。如果是做智慧城市、数字孪生、数字城市相关的开发工作，学会 cesium 将会是你职业生涯的起点。废话不多说我们先来了解一个大概，早在第 2 章数据结构基础就跟大家介绍了三维数据的一些类型。虽然只是简单的提了一下，但是这些内容很重要，做三维开发还是要对这些常见的数据类型有一个深刻的认识。

cesium 从开发难度上讲应该是所有的 WebGIS 框架中最难的。有这么几个原因：

1. 本身的 API 风格酷似 java，方法名和参数长且细致，看起来让人头大
2. 涉及到的很多 API 比较底层，需要有一些计算机图形学的知识作为铺垫
3. 对于空间想象能力有一定的要求，立体几何不好的同学要加油了
4. 很多时候要与浏览器性能打交道，很多时候写 cesium 的应用不是在完成功能，而是在优化性能。

## 5. 数据源类型和数据处理过程丰富且麻烦，需要掌握和学习的东西比较多。

我们来看一下 **cesium** 官网所提供的一些 API:

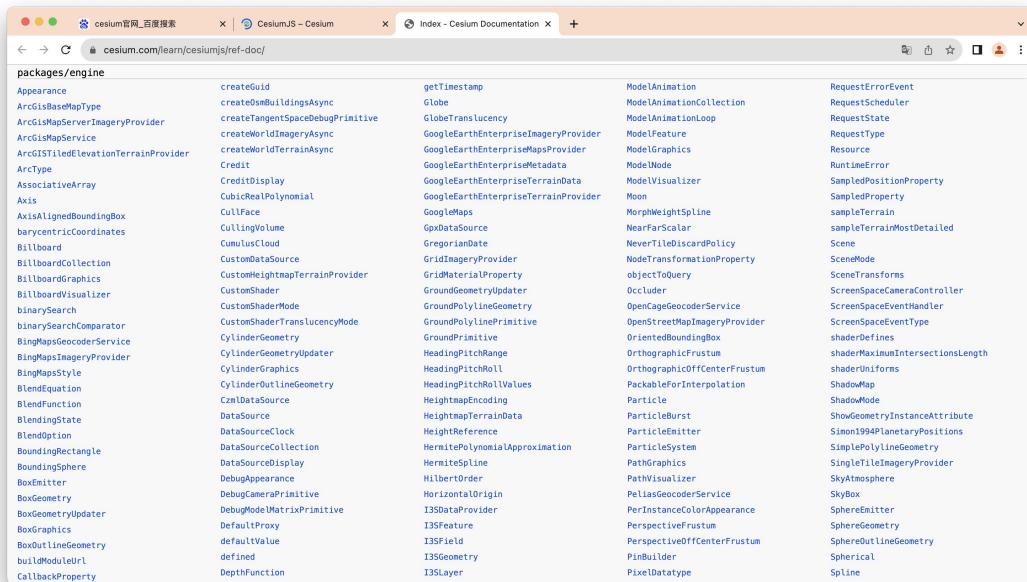


图 7.1.1-cesium API

这还远远没完，往下还有很多，而且点进去每个 API 里面还有很多详细的介绍，所以各位在学习 **cesium** 开发之前一定要做好打持久战的准备。会很累很麻烦很容易掉头发（但是截止到目前笔者写这段内容的时候也没发现 **cesium** 开发者的工资比其他 GIS 开发者的 salary 高多少）不管了，我们还是以知识为主。

首先，我们要去 **cesium** 的官网上熟悉一下内容，得知道一些基本的使用。

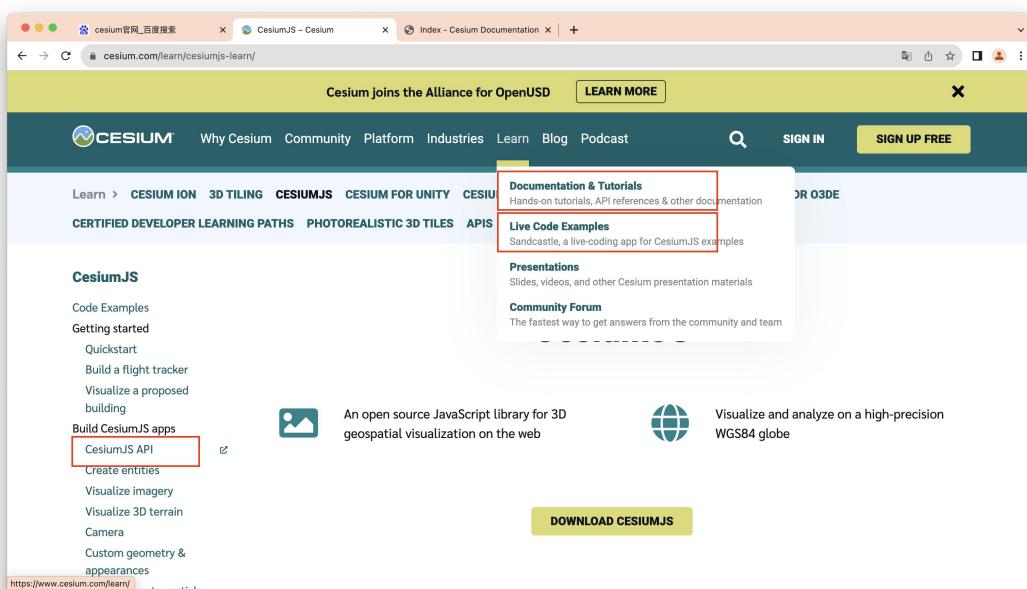


图 7.1.1-cesium 官网

图中这红色方框标出的这几部分内容将会是我们今后开发中经常访问的内容。

Document 和 Tutorials 自然不用说是官方的文档和示例, Live Code Examples 也是官网在线案例, code pan 形式的案例。

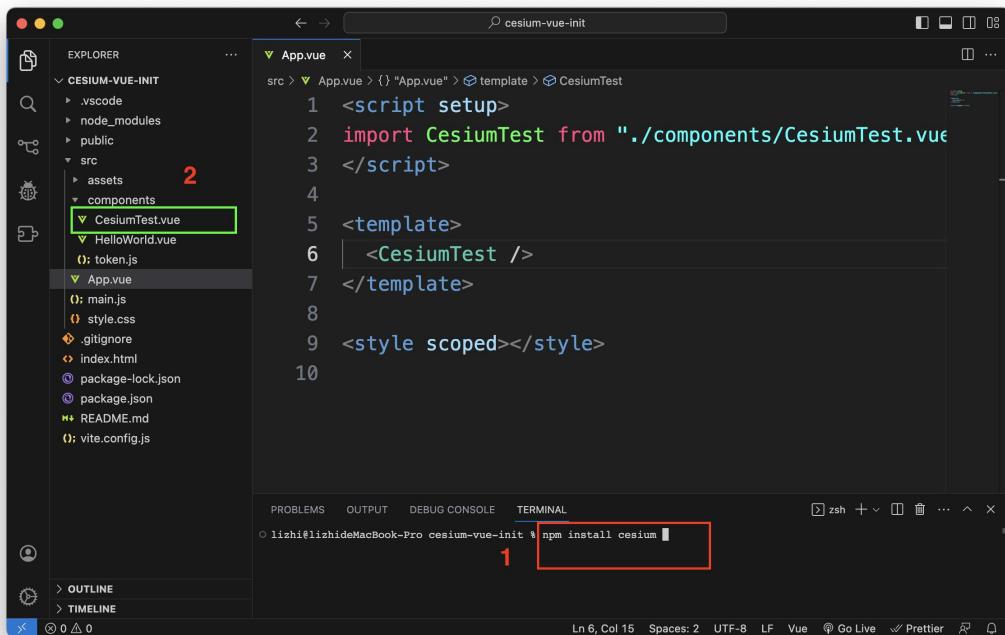
那我们现在就来搭建一个 cesium 的示例工程吧。鉴于各位之前对于 vue 的呼声很高都希望我讲一讲 vue 与 cesium 的结合, 那么我们这一章就是用 vue 来讲吧。在讲之前还需要大家自行到 cesium 的官网上申请一个开发者 token。另外就是作者建议各位使用 cesium 的稳定版本版本号为 1.88, 后续的很多版本会有问题。

首先我们通过 vite 脚手架搭建一个 vue 工程, 考虑到很多同学是刚入门, 我们还是使用 vue2 来讲吧, 我相信会 vue3 的同学 vue2 也不成问题。

使用 vite 搭建 vue 工程我就不再说了, 这个太基础了, 大家直接上网百度吧。折腾好了以后我们直接在项目的终端输入:

```
npm install cesium
```

通过这个命令来安装我们工程中所需要的 cesium 依赖。安装好了以后我们新建一个组件叫做 CesiumTest.vue 来初始化一个 cesium 的工程。



然后不用管 helloworld 这个组件, 直接在 App.vue 中将 helloWorld 组件删除, 重新引入我们的 CesiumTest 组件, 并且把 App.vue 中无关的内容都删除。

紧接着我们来写 CesiumTest 这个组件。我们可以直接查看 cesium 官网的 quick start,

引入一些必要的方法和变量，然后新建一个 `Viewer` 来初始化项目。在初始化之前要跟大家讲清楚一个细节，`cesium` 中的一些静态资源是需要手动指定路径的，比如地球的宇宙背景图，比如按钮的背景图等等，这些静态资源正常是在 `cesium` 的 `npm` 包里，但是我们如果不配置路径的话是没有办法访问这些静态资源的，因此我们需要做一个操作，就是把 `cesium` 包中的静态资源拷贝到我们项目工程的 `public` 文件夹之下然后再把静态资源路径指向它。

这一点在官网中也有详细说明：

#### Configuring CESIUM\_BASE\_URL

CesiumJS requires a few static files to be hosted on your server, like web workers and SVG icons. Configure your module bundler to copy the following four directories and serve them as static files:

- `node_modules/cesium/Build/Cesium/Workers`
- `node_modules/cesium/Build/Cesium/ThirdParty`
- `node_modules/cesium/Build/Cesium/Assets`
- `node_modules/cesium/Build/Cesium/Widgets`

The `window.CESIUM_BASE_URL` global variable must be set before CesiumJS is imported. It must point to the URL where those four directories are served.

For example, if the image at `Assets/Images/cesium_credit.png` is served with a `static/Cesium/` prefix under `http://localhost:8080/static/Cesium/Assets/Images/cesium_credit.png`, then you would set the base URL as follows:

```
window.CESIUM_BASE_URL = '/static/Cesium/';
```

因此我们需要按照下面的步骤将静态文件进行迁移，具体步骤如下

1. 找到项目工程中的 `node_modules` 文件夹，找到里面的 `cesium`，再找到 `Build` 文件夹，再找到之下的 `Cesium` 文件夹，将整个文件夹拷贝
2. 将拷贝的文件夹粘贴到项目的 `public` 文件夹之下
3. 在组件编写的时候加上红框中的代码

```

App.vue          CesiumTest.vue
src > components > CesiumTest.vue > {"CesiumTest.vue" > style > .cesium-viewer-toolbar
24   mounted() {
25     this.init();
26   },
27   methods: {
28     async init() {
29       Ion.defaultAccessToken = token;
30       window.CESIUM_BASE_URL = "./Cesium/";
31       const viewer = new Viewer("cesiumContainer",
32         // terrainProvider: Cesium.createWorldTerrain(),
33         navigationHelpButton: false,
34         timeline: false,
35         fullscreenButton: false,
36         animation: false,

```

关于示例代码的编写，我们首先可以准备一个容器将来用于放置 cesium 的渲染内容，即一个 div 就可以，通常我们是需要这个 div 撑满整个屏幕的。

```

App.vue          CesiumTest.vue
src > components > CesiumTest.vue > {"CesiumTest.vue" > style
1  <template>
2  <div id="cesiumContainer"></div>
3  </template>
4  <style>
5  #cesiumContainer {
6    position: absolute;
7    height: 100vh;
8    width: 100%;
9    inset: 0;
10 }
11 .cesium-viewer-bottom {
12   display: none !important;
13 }
14 .cesium-viewer-toolbar

```

然后我们新建一个 viewer 来将 cesium 挂载到准备好的容器里，这一点其实和 leaflet、mapbox 都是一样的。

```
<template>
    <div id="cesiumContainer"></div>
</template>
<style>
    #cesiumContainer {
        position: absolute;
        height: 100vh;
        width: 100%;
        inset: 0;
    }
    .cesium-viewer-bottom {
        display: none !important;
    }
    .cesium-viewer-toolbar {
        display: none !important;
    }
</style>
<script>
window.CESIUM_BASE_URL = "/Cesium/";
import {
    Cartesian3,
    createOsmBuildingsAsync,
    Ion,
    Math as CesiumMath,
    Terrain,
    Viewer,
} from "cesium";
import "cesium/Build/Cesium/Widgets/widgets.css";
import { token } from "./token";
export default {
    name: "CesiumTest",
    data() {
        return {};
    },
    mounted() {
        this.init();
    },
},
```

```
methods: {
  async init() {
    Ion.defaultAccessToken = token;
    window.CESIUM_BASE_URL = "./Cesium/";
    const viewer = new Viewer("cesiumContainer", {
      // terrainProvider: Cesium.createWorldTerrain(),
      navigationHelpButton: false,
      timeline: false,
      fullscreenButton: false,
      animation: false,
      BaseLayerPicker: false,
      HomeButton: false,
      ProjectionPicker: false,
    });
  },
},
};

</script>
```

到此为止，我们的基础框架就算是已经搭建好了。我们只需要启动项目，就可以看到：

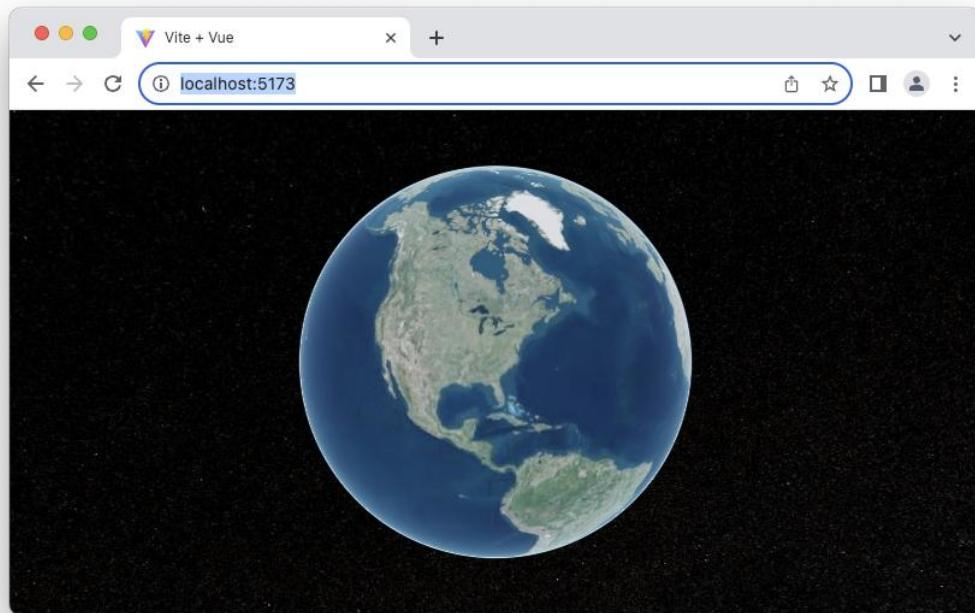


图 7.1.2-cesium+vue 初始化

剩下的工作就是去实现各种功能了，在这里就不跟大家解释这些详细的参数了，像 `navigationHelpButton`、`Homebutton` 这种组件是否开启这种配置大家自行阅读官网。由于篇幅原因我们不能将官网上的 API 一个一个放到这里讲，我们就拣几个非常常见且重要的 API 跟大家介绍一下：

1. `Viewer` 视图容器，主要用于地图和数据的展示，可以设置页面上的元素、功能按钮、背景、地图、地形等等，是 `cesium` 最核心的也是必须掌握的 API（组件）之一。我们在上面的例子中新建一个 `cesium` 的 demo 工程也必须用到 `viewer`。`viewer` 是创建 `cesium` 应用的基础。`viewer` 在 `cesium` 中就好比其他框架中的构造函数（`mapbox` 中的 `Map`, `leaflet` 中的 `L` 等），`cesium` 中很多子类和方法也都是挂在 `viewer` 之下的。

2. `widgets`，页面小组件，即页面上的各种按钮，小组件，包括地图选择器，视角选择器，事件播放轴，`widgets` 里有非常重要的。我们用来呈现内容的。

packages/widgets
Animation
AnimationViewModel
BaseLayerPicker
BaseLayerPickerViewModel
Cesium3DTilesInspector
Cesium3DTilesInspectorViewModel
CesiumInspector
CesiumInspectorViewModel
ClockViewModel
Command
createCommand
FullscreenButton
FullscreenButtonViewModel
Geocoder
GeocoderViewModel
HomeButton
HomeButtonViewModel
InfoBox
InfoBoxViewModel
NavigationHelpButton
NavigationHelpButtonViewModel
PerformanceWatchdog
PerformanceWatchdogViewModel
ProjectionPicker
ProjectionPickerViewModel
ProviderViewModel
SceneModePicker
SceneModePickerViewModel
SelectionIndicator
SelectionIndicatorViewModel
SvgPathBindingHandler
Timeline
ToggleButtonViewModel
Viewer
viewerCesiumInspectorMixin
viewerDragDropMixin
viewerPerformanceWatchdogMixin
viewerVoxelInspectorMixin
VoxelInspector
VoxelInspectorViewModel
VRButton
VRButtonViewModel

3. `Billboard`，广告牌，实际上是我们用作在 `cesium` 上叠加 dom 元素的一种方式，比如在地图上添加一些文字和图片等元素的时候可以使用 `billboard`。

4. `Camera`，相机，`cesium` 中重中之重的类，所有视图方面的操作，动画，运动等相关的需求都脱离不了相机。轻则各种视角的切换，重则各种飞行漫游，都是基于相机

5. `Cesium3DTileset`，3d tiles 数据加载类。用于 3d tiles 数据的加载。

6. `Entity`，实体，即 `cesium` 中几何图形渲染所需要的类

7. `ImageProvider`，影像提供器。提供各类影像地图

8. `Primitive`，原始绘画渲染类，以更加底层和高效的方式去渲染图形

9. `WebMapServiceImageryProvider`，wms 服务地图提供器

10. `WebMapTileServiceImageryProvider`，wmts 服务地图提供器

11. `ScreenSpaceEventHandler`，屏幕事件句柄，屏幕事件相关操作

12. `CzmlDataSource`，czml 格式数据源，一种动态的用于描述路径的数据格式

.....还有很多，我在之后的新版本不断再给各位介绍。

除了 API 需要强调一部分之外，还有一件非常重要的事情需要跟大家讲一下，那就是我们在讲 `vue`、`react` 等框架与 `cesium` 做结合的时候，**不要将 cesium 中的一些变量或者对**

象轻易的做成响应式的。因为我们都清楚，像 `vue`、`react` 这样的框架都有响应式的特性，即数据已发生变化页面立即重新渲染，虽然在常规的业务系统中这是一件好事情，给用户更好更快的操作体验，但是在 WebGIS 的开发中这并不是一件好事情。因为你一旦将某个变量做成响应式的，那么这个变量就会绑定很多的监听（`getter` 和 `setter`），如果这个变量是一个对象类型的数据，还意味着深层次的嵌套的监听，这样一来，反倒会造成很大的资源浪费和性能下降，因为很多时候我们页面的内容并不是频繁的改变，即使改变我们也不见得多么希望变化很快，实际上我们可以通过动画等过渡效果来解决这一问题。如果按照响应式的思路，那么这个变化将是非常大的，就比如你将 `viewer` 对象做成响应式的，那么 `viewer` 以及 `viewer` 下的任何一个变量发生变化，页面都要重新渲染，这听起来简直是太扯了！因此非常不建议大家将 `cesium` 中的变量做成响应式的。

## 第 2 节：各类图层加载

毕竟各位使用 `cesium` 还是要进行 `gis` 相关的开发，那么我们就避免不了跟一些地图服务打交道，在前两章中我们也教给了大家如何加载天地图，在这一章也不例外我们先来加载天地图。我们加载天地图需要使用到 `cesium` 中为我们提供的一个类叫做：

### ***WebMapTileServiceImageryProvider***

怎么样是不是特别长，但是长的好处就在于阅读起来能够顾名思义，我严重怀疑写 `cesium` 框架的这个人是 `java` 转过来的～

具体的代码我们这么写：

```
<script>
const key = tdt_key;
Cesium.Ion.defaultAccessToken = cesium_token;
window.CESIUM_BASE_URL = "./Cesium";
const viewer = new Cesium.Viewer("cesiumContainer", {
    // terrainProvider: Cesium.createWorldTerrain(),
    navigationHelpButton: false,
    timeline: false,
    fullscreenButton: false,
    animation: false,
    imageryProvider: new
Cesium.WebMapTileServiceImageryProvider({
    url: "http://t0.tianditu.gov.cn/img_w/wmts?tk=" + key,
    layer: "img",
    style: "default",
    tileMatrixSetID: "w",
    format: "tiles",
    maximumLevel: 18,
}),
});
viewer.imageryLayers.addImageryProvider(
new Cesium.WebMapTileServiceImageryProvider({
    url: "http://t0.tianditu.gov.cn/cia_w/wmts?tk=" + key,
    layer: "cia",
    style: "default",
    tileMatrixSetID: "w",
    format: "tiles",
    maximumLevel: 18,
})
);
</script>
```

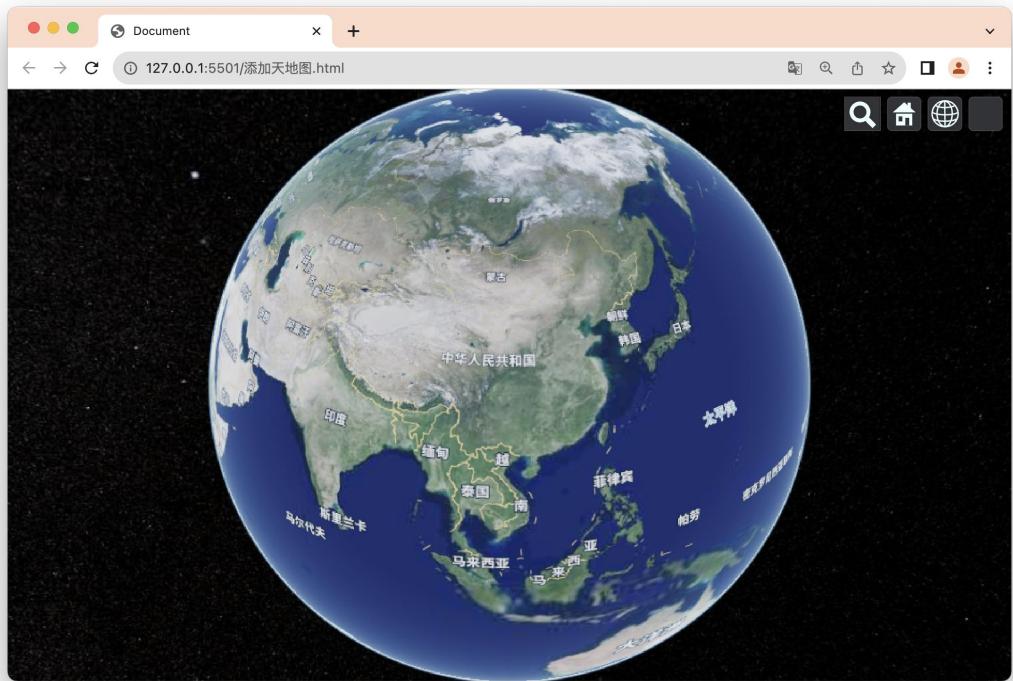


图 7.2.1-cesium 加载天地图

天地图也需要各位开发者自行去官网申请一个 key，这个在前两章也有说过，我们来解释一下这段代码，首先我们要创建一个 `imageProvider` 来为 cesium 提供地图，这个 `imageProvider` 的来源有很多，可以是各大厂商的地图，我们这里使用

`WebMapTileServiceImageryProvider` 是因为天地图遵循 WMTS 规范，因此我们还是把天地图当成了和其他普通的 WMTS 服务一样来加载。

参数我想应该不用过多的解释了，`url` 代表地图服务端地址，`layer` 表示图层名称，`style` 表示图层样式，`tileMatrixSetID` 表示瓦片矩阵 ID，其实在这里可以简单的理解成为天地图的坐标系，是 c 的经纬度坐标系还是 w 的墨卡托坐标系。后面的两个参数 `format` 和 `maximumLevel` 可以不填的，分别表示格式和最大缩放层级。

除了要学会加载天地图之外还要学会加载一些矢量图形，由于 cesium 本身是不支持矢量切片的（虽然有部分大神写了支持矢量切片的插件，但是我个人觉得不必费劲去研究，因为 cesium 本身底层是希望大家使用 3d tiles 的，否则官方肯定早就支持这种数据类型了）。那我们就来跟大家演示一些如何加载正常的矢量数据（我们的老朋友 `geojson`）吧。

在 cesium 中加载 `geojson` 数据需要用到一个函数叫做：

`Cesium.GeoJsonDataSource.load()`，这个函数的参数可以是一个 `url`，即访问数据的路径，当然也可以直接是数据本身。具体的代码示例如下：

```
addGeojson("./data/金华市.json");
async function addGeojson(url) {
    let result = await Cesium.GeoJsonDataSource.load(url);
    viewer.dataSources.add(result);
    let entities = result.entities.values;
    for (let i = 0; i < entities.length; i++) {
        let entity = entities[i];
        entity.polygon.material = Cesium.Color.YELLOW;
        //entity.polygon.extrudedHeight = Math.floor(2000); // 高度
        // 随机数, 单位是米
        entity.polyline.material = new
        Cesium.PolylineGlowMaterialProperty({
            glowPower: 0.1,
            color: Cesium.Color.fromCssColorString("#ff0000"),
        });
    }
}
```

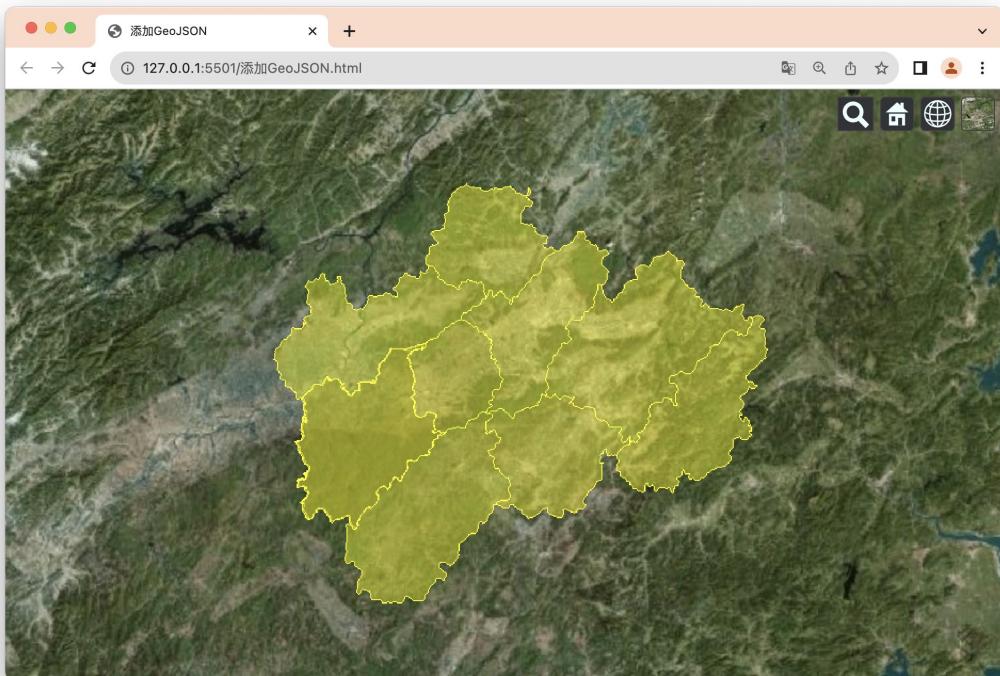


图 7.2.2-cesium 加载 geojson 数据

在这里我们必须跟大家提出 cesium 中一个及其重要的概念了——entity。dataSource.load()执行的意义只是将 geojson 数据加载到了 viewer 当前的数据源中，但是还没有进行渲染，页面上看不到效果的，真正的渲染是 entity 进行的。所谓 entity 就是其中

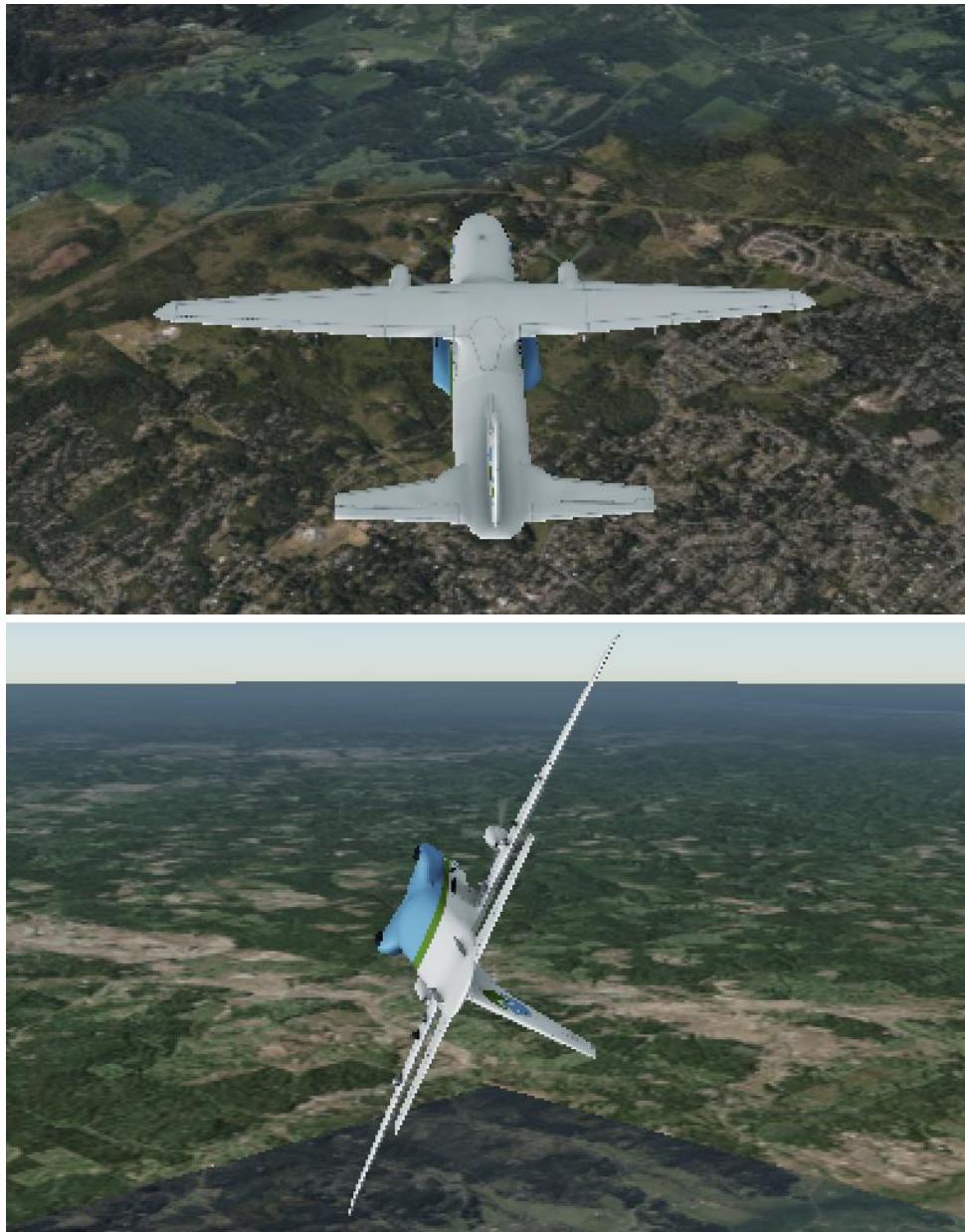
文翻译“实体”的意思。但是这个实体所包含的数据范围特别的广，在 cesium 中，一份 geojson 格式的数据可以称之为是一个 entity 实体，一个手工模型也可以称之为是一个实体，你所绘制的长方体，球体等图形也是实体。所以实体的含义其实就是数据，各种数据，都可以被概括为是实体（影像地图资源除外）。

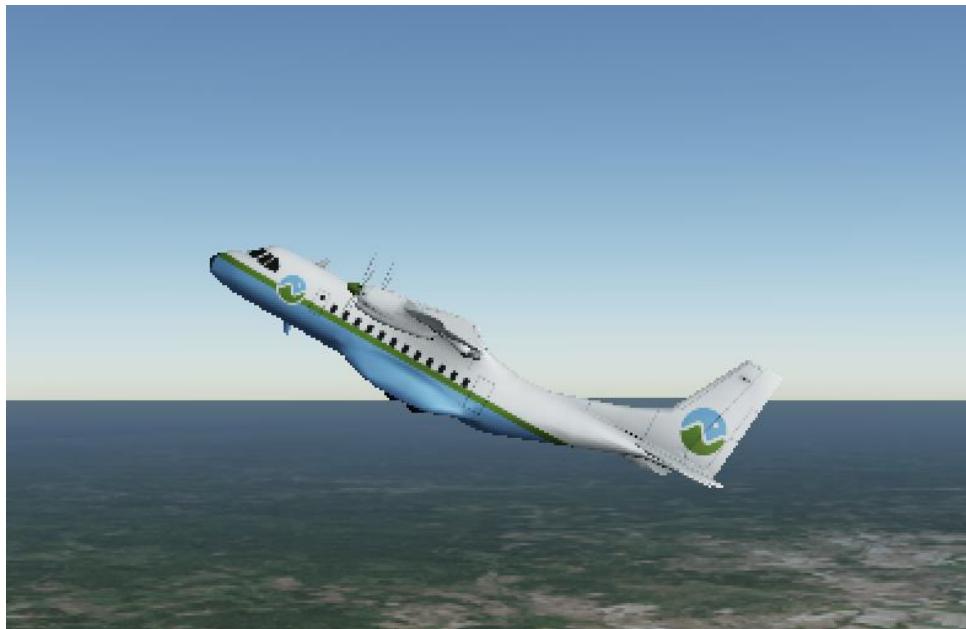
在 cesium 中渲染图形还有另一种特别重要的方式——primitive。换句话说上面的 geojson 数据的加载其实也可以用 primitive 的方式完成。它与 entity 的作用是一样的，但是二者有很大的区别，各有优势和特点。首先 entity 开发难度比较低，使用起来相对比较容易，很适合新手入门，但是缺点在于 entity 绘制的图形渲染效率比较低，可能会造成页面卡顿，资源消耗上升，性能出现问题。primitive 方式开发难度比较高，因为它更偏于底层，理解起来要更费劲一些，但是一旦用好了，渲染效率比 entity 要高很多。同时 primitive 方式也更为灵活，可以自定义很多样式和材质。实际上大家可以理解成 entity 是基于 primitive 由进行了一次封装。虽然方便了开发，但是却牺牲了一部分渲染效率。

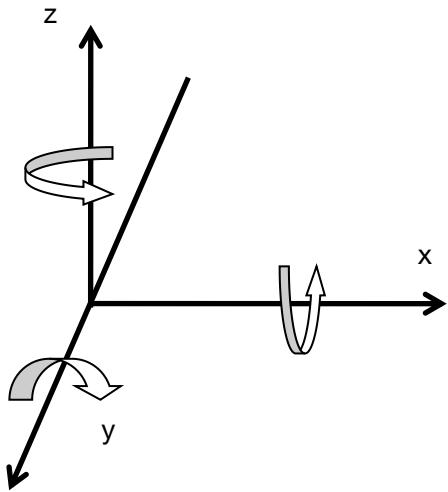
## 第 3 节：相机教程

在学习 cesium 的相机知识之前，需要大家先明确一些空间上的概念和名词。假设现在驾驶着一架飞机在空中飞行，高中空间几何我们学过 xyz 坐标系，那么我们的飞机可以发生如下的操作：









所以大家在学习相机之前应当搞清楚三个英文单词即 **heading**、**pitch**、**roll**。

首先 **heading** 表示朝向，即水平面内的方向角，即你想去东南西北哪个方向，**roll** 表示 **y** 轴上的倾斜角度，即飞机的左翅膀高还是右翅膀高？**pitch** 表示与地面的倾斜夹角，即飞机是处于起飞阶段（机头朝上）还是向地面俯冲？

其实在 **cesium** 中，视角的变动不止一个方法可以完成，**viewer** 本身就是有 **flyTo** 的方法的。不过 **viewer** 的 **flyTo** 和 **camera** 的 **flyTo** 还是有些区别的。首先 **viewer** 的 **flyTo** 的参数及用法如下：

```
viewer.flyTo(target,options)
```

这里面的 **target** 是指要飞行到的目标。这个目标一般就是数据或者位置。这个数据所支持的类型有很多种，凡是可以以 **entities** 的方式加载的数据都可以，比如说一些倾斜摄影模型，手工模型，**geojson** 数据源等等，以及一些已经被标准化的 **3D tiles** 数据，但是要注意，这个 **target** 他不可是坐标或者位置。只能是一个目标数据，他定位的原理是找到目标数据的包围盒，然后以包围盒子的集合中心为基准点进行飞行。并且这个飞行的特点是，必须等待数据完全加载完成之后才会飞行，如果数据没有加载完成，它是不会飞行的。

至于后面的 **options** 参数，是用来调整相机的参数的。它是一个对象，一共包含三个参数分别是 **duration**，**maximumHeight** 以及 **offset** 比如，飞行过程中的所花费的时间的参数：**duration**，如果你希望飞行的慢一点，那么你可以把这个时间设置的长一点。**maximumHeight** 参数是指飞行过程中相机的最大高度。**offset** 参数是指相机的偏移量。**offset** 里面又可以包含 3 个参数，分别是用于控制相机朝向的 **heading** 和用于控制倾斜

视角的 `pitch` 以及用于控制相机到目标中心点距离偏移的参数 `range`。

接下来我们介绍 `camera` 的 `flyTo` 方法。要注意，尽管 `viewer` 可以调用 `camera`，并且 `viewer` 在执行飞行的过程中也是调用了相机，但是 `viewer` 所调用的相机是经过进一步的封装的。因此 `camera` 的 `flyTo` 方法更为原始和底层。

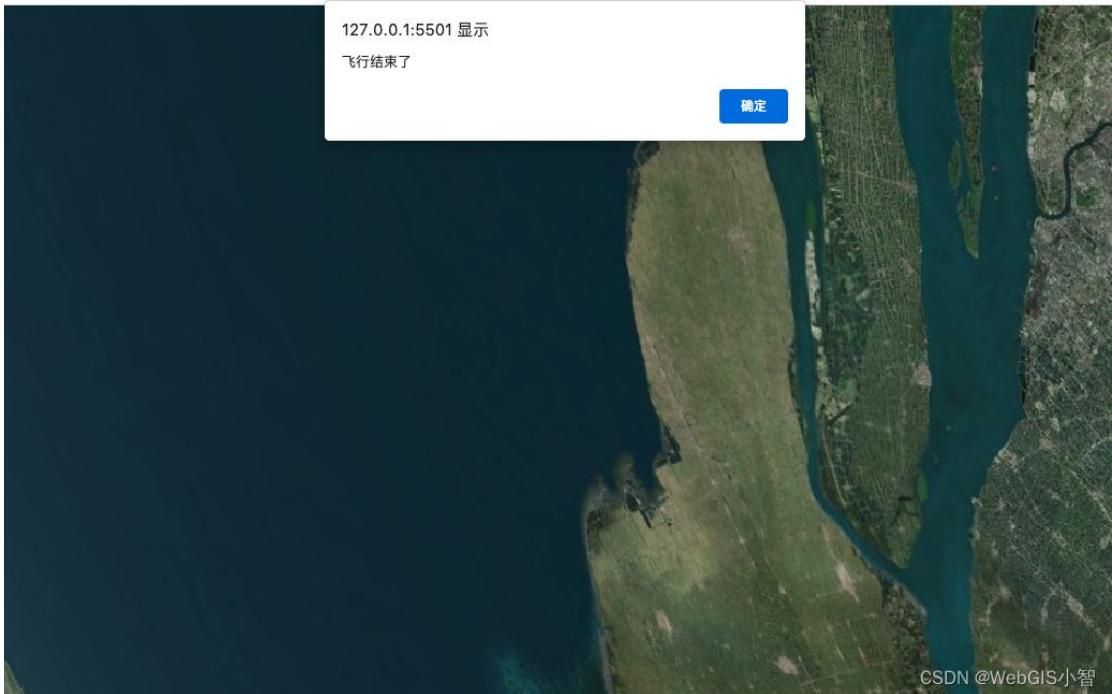
接下来我们还得说一下 `destination` 参数，这个参数是配置要飞到的目标地方的位置，注意是位置，那么这个参数必须是一个坐标。这个坐标可以是一个经纬度，也可以是一个带有高度的经纬度。也可以是一个经纬度的范围，就像下面这样写：

```
Cesium.Rectangle.fromDegrees(west, south, east, north)
```

但是它的目标不可以是一个 `entity` 之类的。接下来的三个参数我们一起说，因为这三个参数虽然含义不一样，但是性质都是一样的，因为他们都是回调函数，当某个状态是才能触发。你可以在触发之后做一下操作。

比如第一个 `complete` 方法，就是飞行完成之后才能够触发。你可以在这个回调函数里面提前写好到时候飞行完成之后你要执行的操作。

```
viewer.camera.flyTo({
  destination: Cesium.Cartesian3.fromDegrees(
    121.465538,
    30.241737,
    150000.0
  ),
  orientation: {
    heading: 0,
    pitch: 345,
    roll: 2.0,
  },
  complete: function () {
    //如果你想在飞行结束之后做操作，可以把操作写在这个方法里
    alert("飞行结束了");
  },
  cancel: function (e) {
    console.log(e);
    //如果你想在飞行取消之后做操作，可以把操作写在这个方法里
  },
});
```



如果你非要使用 `camera` 飞到某个实体上。飞到某个物体上，你可以使用 `flyToBoundingSphere` 方法飞到这个物体的包围球上。具体用法如下：

```
viewer.camera.flyToBoundingSphere(newCesium.BoundingSphere(center,
radius),options)
```

## 第 4 节：模型与 3d tiles

`cesium` 的研发设计思想其实就是集中于三维立体模型的渲染和展示。因此 `cesium` 的底层设计也是尽可能的支持各种三维模型，但是在研发的过程中难免会遇到很多实际的问题，最大的问题就是大批量的三维模型该如何以最高效的方式渲染？通常我们现在所做的智慧城市也好，数字孪生也好，都是需要很多很多的城市真实的三维实景模型渲染到浏览器上，如果不进行优化处理，必然也无法支撑如此需求。

因此，`3d tiles` 的概念就因此诞生了。要注意：**`3dtiles` 是一个规范，不是一种具体的数据格式，是设计者为了抹平各种 `3d` 数据的差异，采用统一的标准和规范来使数据跨越各类型平台方便的流通，同时也为了让 `cesium` 这样的引擎更高效的加载 `3d` 数据**这是很多新手容易误解和难以理解的地方。很多人在最初学习 `cesium` 的时候以为 `3d tiles` 就是一种格式，要把数据转换成这种格式才能被 `cesium` 所加载。实则不然，`cesium` 官方其实是希望用户将大批量的数据按照某个规范的标准进行处理，将处理之后的符合规范的数据丢进 `cesium`

中进行加载。市面上的三维数据类型其实特别的多，这些数据不见得都能转换成某种固定的格式，但是这些数据可以按照某个固定的规范进行处理。比如说一个分层的思想，也就是 LOD 的概念。某个模型，你在距离它很远的时候其实是看不清它的细节的，你只能从宏观上了解到它的全貌。随着距离的逐渐逼近，你才能够慢慢的了解到它各个部分的细节。这就是 LOD 的思想。万物，在距离你足够远的地方其实都是一个点。

3D tiles 具体的标准内容是

关于 3D tiles 的更多细节，大家可以前往 [github](#) 查看：

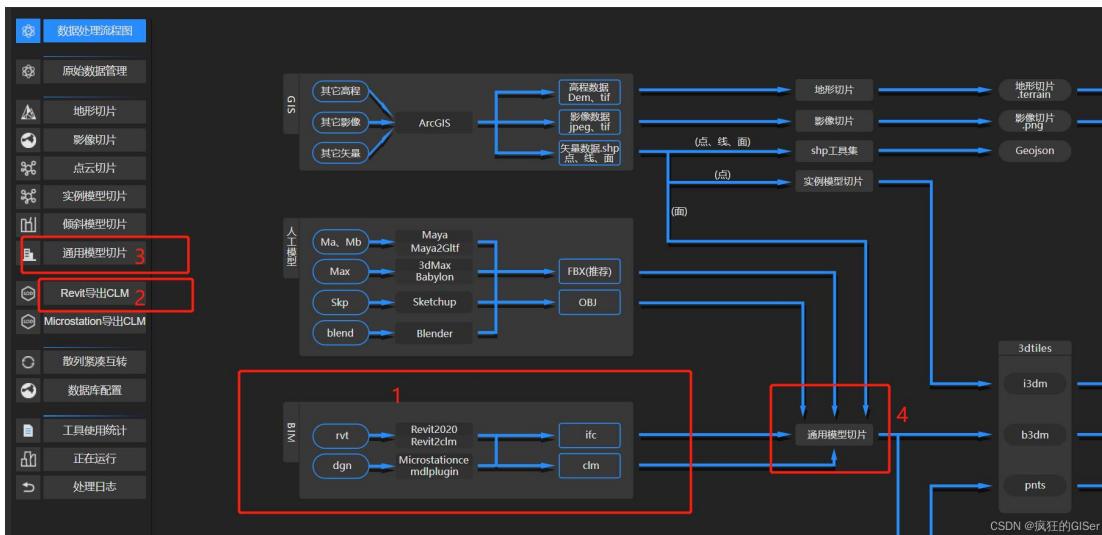
<https://github.com/CesiumGS/3d-tiles>

我们在日常的开发过程中，是如何将各种数据按照 3d tiles 的标准来进行处理的呢？

首先我们拿到各种类型的三维数据之后首先要确定其具体的格式，是 gltf、BIM、shp 倾斜摄影 b3dm 还是点云？确定好数据格式以后我们通常可以使用一个非常强大的客户端工具 [cesium lab](#)。它是一个将各种市面上常见的三维数据按照 3d tiles 标准进行处理的软件。

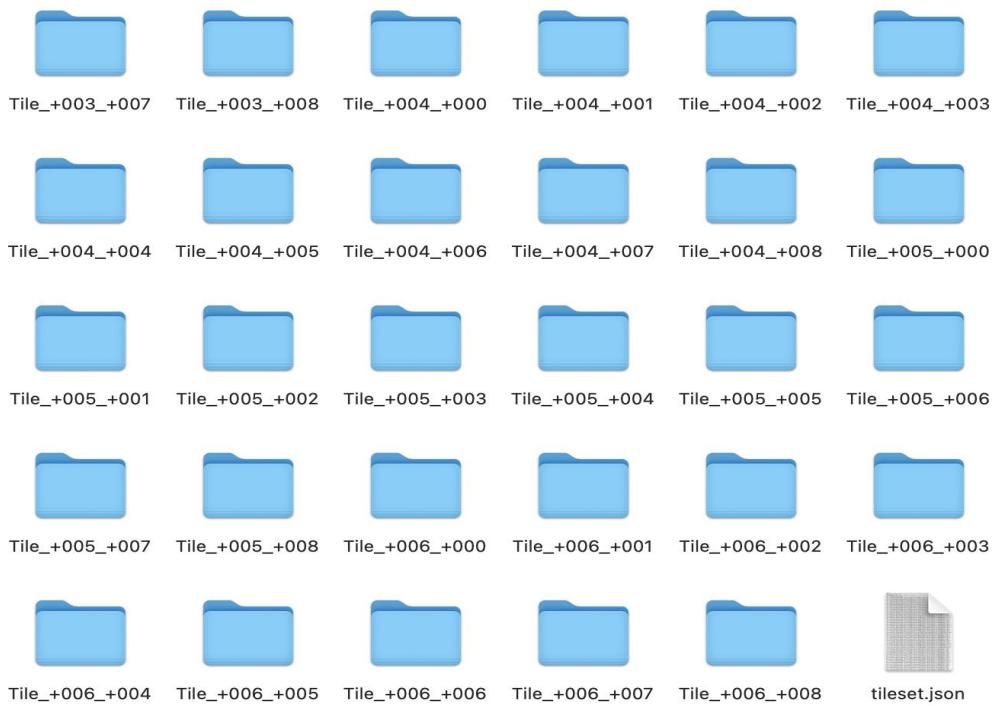


注意这个软件只支持 window 平台。[mac](#) 暂时是不支持的。软件内部提供了很多的转换流程，所以在前文中要让大家确定自己的数据格式，好在这一步选择对应的流程进行处理。



接下来我们以倾斜摄影为例子, 来跟大家演示如何将一批倾斜摄影数据从拍摄到最后呈现在浏览器上。首先倾斜摄影最初的形式其实就是 png 或者 jpg 格式的照片, 无人机在飞行过程中会按照指定的行列位置进行拍摄, 这些照片会经过 3d max 等软件进行第一次的处理和合成, 将平面的照片进行拼接, 因为飞机飞行的时候会从前后左右以及正上方进行 5 个角度的拍摄, 这样就很容易将对应的照片进行拼接旋转角度等操作, 总之这一步实际上就是想将二维的照片处理成可以从空间角度查看的三维模型。

通常处理好的三维数据数据量还特别的大, 而且原生格式 b3dm 也不是 cesium 能够直接加载的格式, 这时候我们就用到上文提到的 cesium lab 软件进行处理, 处理之后的文件就变成了如下的样子:



怎么样？是不是像极了我们之前了解的 2 维中的栅格瓦片和矢量切片的概念，实际上对于倾斜摄影数据按照 3d tiles 标准处理就相当于是对数据进行切片，将数据按照层级分割成很多个瓦片，这里面还有一个非常重要的文件 `tileset.json`，它相当于一个头文件，这些数据的关键信息和组织信息都在这个 `json` 文件里。将来我们在 `cesium` 中加载倾斜摄影的时候也是先请求这个文件，那到这里我们不妨也直接告诉大家如何加载倾斜摄影：

```
const viewer = new Cesium.Viewer("cesiumContainer", {
    // terrainProvider: Cesium.createWorldTerrain(),
    navigationHelpButton: false,
    timeline: false,
    fullscreenButton: false,
    animation: false,
});
// 加载倾斜模型
var tileset = new Cesium.Cesium3DTileset({
    url: "http://localhost:8080/baoli/tileset.json",
});
viewer.scene.primitives.add(tileset);
viewer.zoomTo(tileset);
```

其实也非常的简单，就用到了 `cesium` 中加载 3d tiles 的通用方法 `Cesium3DTileset()` 方法。

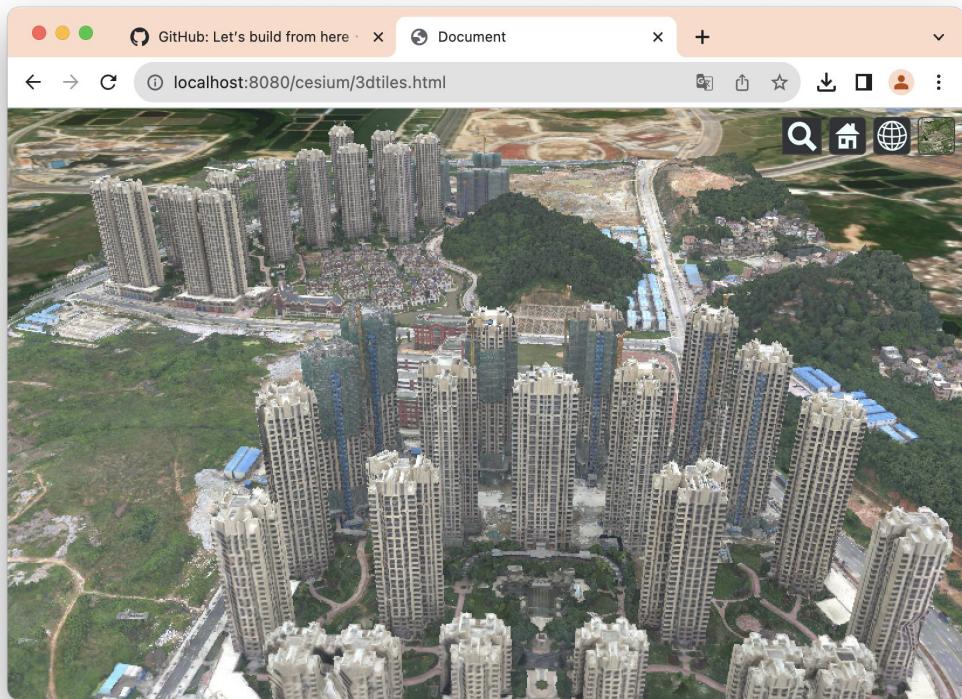


图 7.4.1-cesium 加载倾斜摄影

我们在第二章第 4 节中讲过的那些三维的数据类型都必须遵循 3d tiles 规范，并且处理过程也都是和上述的倾斜摄影数据的处理过程类似。我们在实际开发的过程中可以按照上面所讲的步骤，再结合 cesium 具体的 API 版本变化，进行对应的开发。

## 第 5 节：交互及事件

想要学会如何用 cesium 进行业务系统的开发，交互和事件也是必不可少的一部分。和其他 gis 框架一样，我们还是要熟悉一些鼠标和键盘事件的使用。本节我们以 ceisum 中进行绘制图形和自定义点击图形弹框展示属性两个功能为例子来让大家掌握 cesium 中的事件和交互。

我们来做一个 cesium 中的绘制工具，希望实现的效果是能够完成点线面的绘制。我们先来捋清思路，要实现绘制，肯定脱离不了鼠标事件，这一点在任何框架中都是这样设计的，而且大致思路都是监听鼠标事件，当鼠标移动的时候记录下来鼠标所经过的每个位置的经纬度坐标，最后把这些坐标整合到一起以 entity 的方式渲染成图形。

```

let handler = new
Cesium.ScreenSpaceEventHandler(viewer.scene.canvas);
// 监听鼠标左键
handler.setInputAction(function (movement) {
let ray = viewer.camera.getPickRay(movement.position);
position = viewer.scene.globe.pick(ray, viewer.scene);
let point = drawPoint(position);
tempEntities.push(point);
}, Cesium.ScreenSpaceEventType.LEFT_CLICK);
function drawPoint(position, config) {
let config_ = config ? config : {};
return viewer.entities.add({
name: "Point",
position: position,
point: {
color: Cesium.Color.SKYBLUE,
pixelSize: 10,
outlineColor: Cesium.Color.YELLOW,
outlineWidth: 3,
disableDepthTestDistance: Number.POSITIVE_INFINITY,
heightReference: Cesium.HeightReference.CLAMP_TO_GROUND,
},
});
}

```

解释一下这段代码。首先我们声明了一个 `handler` 对象，用于操控鼠标事件，这个 `handler` 是一个监听屏幕的事件句柄 `ScreenSpaceEventHandler`，调用这个函数的时候必须把当前所在的场景 `scene` 传入，`handler` 有个关键的函数 `setInputAction` 这个函数的参数是一个回掉函数，`cesium` 框架会在这个回掉函数的参数中将当前鼠标所在位置的经纬度坐标返回给你，然后我们利用 `cesium` 返回的经纬度调用 `drawPoint` 函数来将坐标渲染成点。要注意这个 `handler` 对象不仅可以监听鼠标左键单击事件，还可以监听右击双击等等的事件你可以在事件中完成你想要的操作。绘制线和绘制面与绘制点大同小异，无非是对监听回来的坐标做了一些处理，比如绘制一个面的话，需要加上这段代码：

```
handler.setInputAction(function (click) {
    let cartesian = viewer.camera.pickEllipsoid(
        click.position,
        viewer.scene.globe.ellipsoid
    );
    if (cartesian) {
        let tempLength = tempPoints.length;
        if (tempLength < 3) {
            alert("至少选择 3 个点完成多边形闭合");
        } else {
            //闭合最后一条线
            let pointline = drawPolyline([
                tempPoints[tempPoints.length - 1],
                tempPoints[0],
            ]);
            tempEntities.push(pointline);
            drawPolygon(tempPoints);
            tempEntities.push(tempPoints);
            handler.destroy(); //关闭事件句柄
            handler = null;
        }
    }
}, Cesium.ScreenSpaceEventType.RIGHT_CLICK);
```

这里明显多了几步操作，使用 `drawPolyline` 函数将图形闭合。`drawPolygon` 函数将坐标数组进行面渲染。

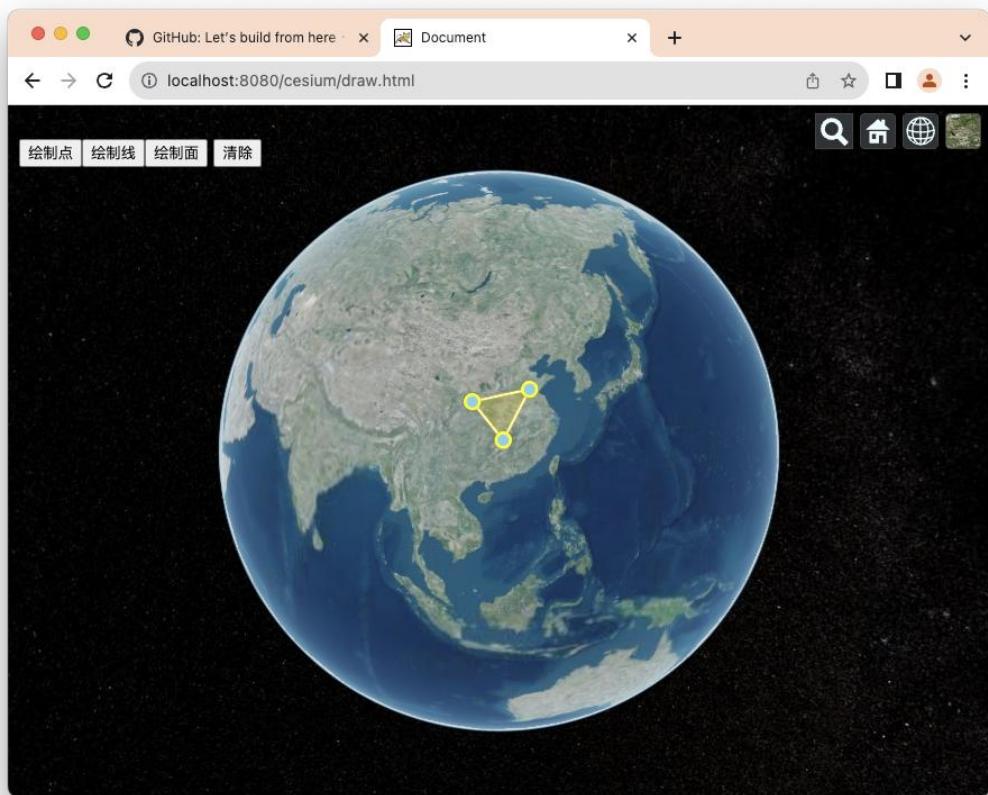


图 7.5.1-cesium 编写绘制组件

我们再来看第二个跟事件交互有关的例子，如何不使用 cesium 自带的相机弹窗来完成用户点击图形之后展示属性的需求。

```
let handler = new
Cesium.ScreenSpaceEventHandler(viewer.scene.canvas);
handler.setInputAction(function (event) {
    let pick = viewer.scene.pick(event.position);
    if (pick && pick.id.id) {
        //拾取到要素
        document.getElementById("stateShow").style.display = "block";
        let ray = viewer.camera.getPickRay(event.position);
        cartesian = viewer.scene.globe.pick(ray, viewer.scene);
        //将要素的属性信息填充在 dom 元素里
        document.getElementById("title").innerHTML =
        pick.id.data.title;
        document.getElementById("state").innerHTML =
        pick.id.data.state;
        document.getElementById("info").innerHTML = pick.id.data.info;
```

```
// 实时更新位置
viewer.scene.postRender.addEventListener(updatePosition);
} else {
document.getElementById("stateShow").style.display = "none";
// 移除事件监听
viewer.scene.postRender.removeEventListener(updatePosition);
}
}, Cesium.ScreenSpaceEventType.LEFT_CLICK);
// 位置更新
function updatePosition() {
// 将 WGS84 坐标中的位置转换为窗口坐标
let windowPosition =
Cesium.SceneTransforms.wgs84ToWindowCoordinates(
viewer.scene,
cartesian
);
// 数值是样式中定义的宽高
if (windowPosition == undefined) return;
document.getElementById("stateShow").style.left =
windowPosition.x - 220 / 2 + "px";
document.getElementById("stateShow").style.top =
windowPosition.y - 150 + "px";
}
```

其实我们可以看到大概的逻辑都是差不多的，只不过在监听到点击事件内部做了一些变化，我们通过 `viewer.scene.pick` 获取到了鼠标点击的要素图形，然后取出这个图形所携带的属性数据，自定义一个 `dom` 元素，将属性填充到 `dom` 元素里。然后再将拾取位置的经纬度转换为屏幕坐标，屏幕上的 `xy` 坐标能够控制 `dom` 元素的位置，我们把 `dom` 元素设置成绝对定位 `absolute`，然后去根据屏幕坐标 `xy` 去控制它的 `top` 和 `left` 边距，这样就能够让 `dom` 元素呈现在图形要素之上。



图 7.5.2-cesium 自定义弹窗

## 第 6 节：WebGL 与着色器知识

早在第 4 章底层渲染原理我就讲过，在浏览器上渲染图形图像就 2 种主流的方法，要么是 `dom` 方式(`img+svg`)，要么是 `canvas` 元素来渲染。前者只适用于轻型小型应用。`canvas` 又分为 `2d` 模式和 `webgl` 模式。实际上知识都是环环相扣的，有很多同学一上来就说要学习 `cesium`，但是一些基础的知识都不清楚，很难走远，最终也只能成为和框架接触的工具人而已。实际上 `webgl` 确实是 `cesium` 的底层。`webgl` 来自于 `opengl`（基于 `c++` 的计算机图形语言），`opengl` 是运行在操作系统上的，是客户端渲染图形的主流方式。比如画图和一些图标的软件。`webgl` 相当于将 `opengl` 搬到了浏览器上。

要学好 `cesium`，达到深层次级别的水平，是必须要掌握 `webgl` 的。因为一些复杂的需求是肯定需要各位会使用 `webgl` 的相关知识解决的。

说到 `webgl` 首先想到的是着色器语言。即 `shader`。使用着色器语言在 `canvas` 上绘制图形图像。说到这里得提前跟大家铺垫一个知识。其实屏幕上的任何你看到的元素其实都可以用着色器的思路去解释。首先，一个物体可能自身存在着多种颜色，并且这些颜色还可能会随着光效发生变化，进而产生阴影等。我们在绘制一个物体的时候实际上可以把物体本身分为若干个三角形。然后我们在渲染这个物体的时候会对物体的不同位置进行分类，主要分为顶点位置和非顶点位置（片元）。这就是我们经常提的概念——**渲染管线**。在计算机渲染

这些图形的时候会根据顶点位置的特殊性渲染对应的颜色，法线等，然后对剩余的片元部分进行另一种着色方案，这就是**顶点着色器**和**片元着色器**的概念。

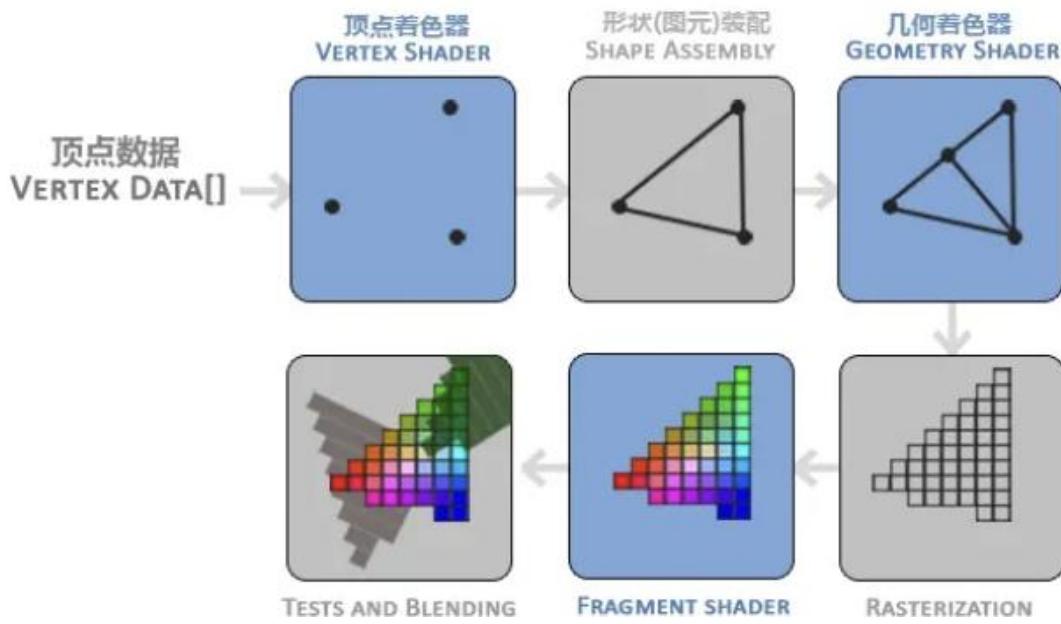


图 7.6.1-图形渲染管线

例如我们现在用顶点着色器和片元着色器的知识来在浏览器页面上绘制一个正方形，我们可以这样写：

```
const ctx = document.getElementById("canvas");
const gl = ctx.getContext("webgl");
// 创建着色器源码
const VERTEX_SHADER_SOURCE = `

uniform vec4 uPosition;
// 只传递顶点数据
attribute vec4 aPosition;
void main() {
    gl_Position = aPosition; // vec4(0.0,0.0,0.0,1.0)
    gl_PointSize = 10.0;
}
`;
```

```
// 顶点着色器
// 顶点着色器需要指定精度，如下指定精度为中级精度
const FRAGMENT_SHADER_SOURCE = `
precision mediump float;
uniform vec2 uColor;
void main() {
    gl_FragColor = vec4(uColor.r, uColor.g, 0.0,1.0); // vec4
}
`;
// 片元着色器
const program = initShader(
    gl,
    VERTEX_SHADER_SOURCE,
    FRAGMENT_SHADER_SOURCE
);
const aPosition = gl.getAttribLocation(program, "aPosition");
const uColor = gl.getUniformLocation(program, "uColor");
const points = [];
ctx.onclick = function (ev) {
    // 坐标
    const x = ev.clientX;
    const y = ev.clientY;
    const domPosition = ev.target.getBoundingClientRect();
    const domx = x - domPosition.left;
    const domy = y - domPosition.top;
    // 当前画布宽度的一半
    const halfWidth = ctx.offsetWidth / 2;
    // 当前画布高度的一半
    const halfHeight = ctx.offsetHeight / 2;
    const clickX = (domx - halfWidth) / halfWidth;
    const clickY = (halfHeight - domy) / halfHeight;
    points.push({
        clickX,
        clickY,
    });
}
```

```
for (let i = 0; i < points.length; i++) {
    gl.vertexAttrib2f(aPosition, points[i].clickX,
    points[i].clickY);
    gl.uniform2f(uColor, points[i].clickX, points[i].clickY);
    gl.drawArrays(gl.POINTS, 0, 1);
}
};
```

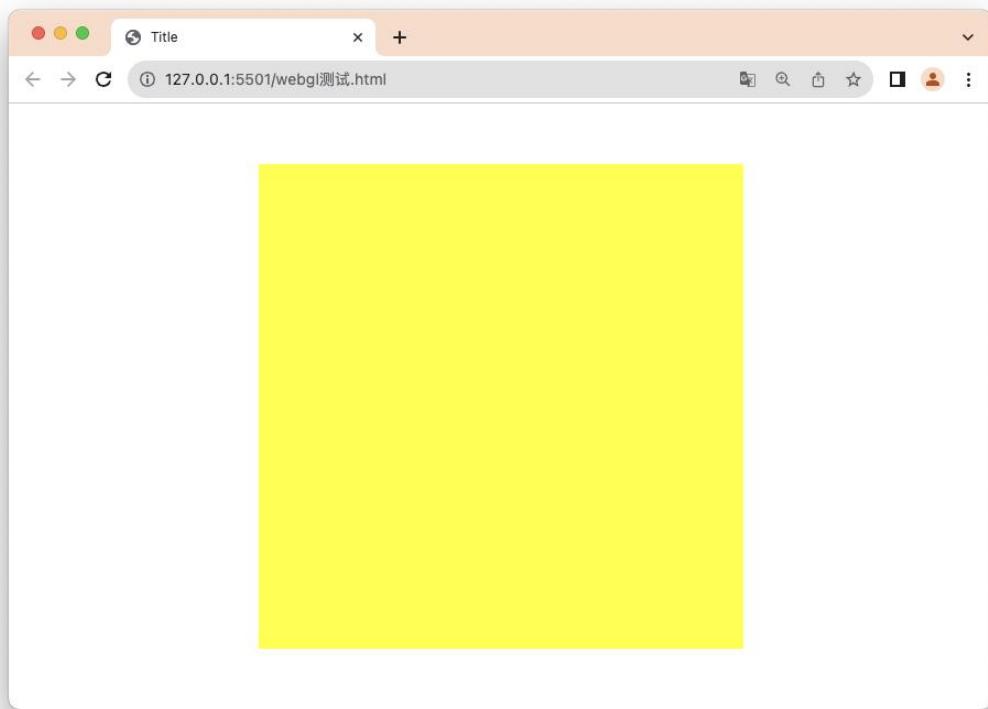


图 7.6.2-webgl 绘制正方形

我们仔细看代码可以得知，实际上着色器语言写的就是 C/C++ 代码，然后以字符串变量的格式交给浏览器渲染引擎 V8 去解释。浏览器渲染引擎底层还是调用了操作系统的 API 来解析这些着色器代码。这其实也是 webgl 由 opengl 迁移而来的佐证。那到这里如果各位向再往深层次走下去，再进一步的了解更加底层的原理和知识的话，就要学习 opengl 和计算机图形学相关的知识了，在本书中将不再为大家进一步讲解，仅为大家介绍入门和发展方向，因为这相比于我们的 gis，有了发展方向上的偏移。之所以这一节还要讲这个知识是大家在进行 cesium 开发的时候避免不了要去分析源码，cesium 的源码中会有很多着色器的部分，为了不阻碍大家的学习进程特此在这一章节进行简单的介绍。

# 第八章：OpenLayers 精讲

## 第 1 节：基础入门

在早期版本的此书中，是不讲 openlayers 这个框架的，因为它和 mapbox 太相似了，很多地方都是近似的，甚至是相同的。包括二者对于具体的业务功能，以及渲染思路都是及其相似的。关于二者的具体差异我在第四章第 4 节框架与技术选型中已经详细讲过了，再此就不赘述了，我们直接来进入 openlayers 的开发。

首先我们还是要学会使用官网：

<https://openlayers.org/>

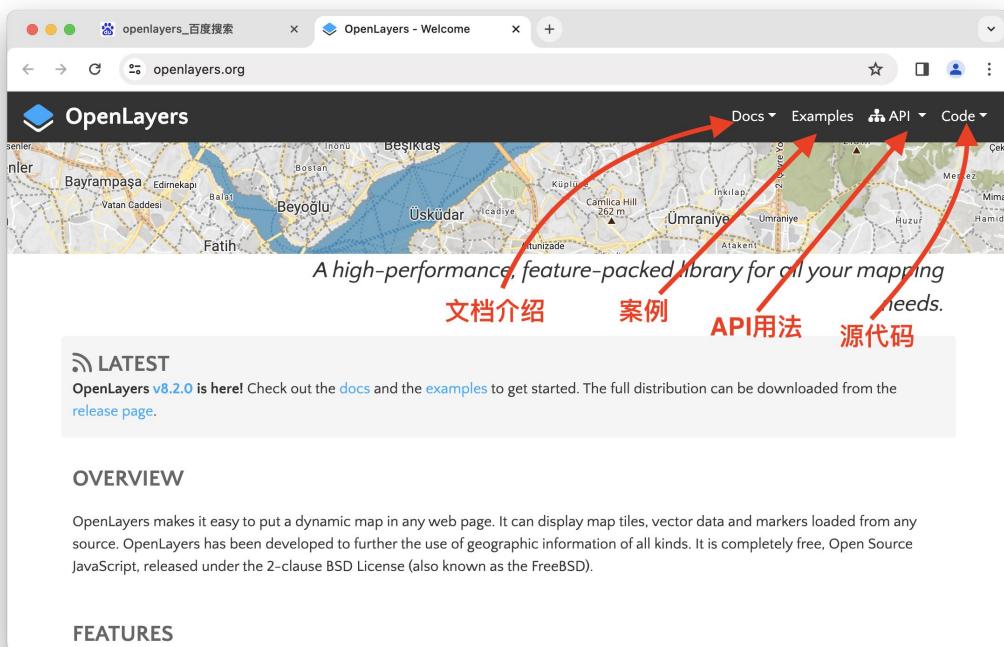


图 8.1.1-openlayers 官网

常言道授人渔不如授人以渔，因此书籍中不可能把所有的知识给大家讲到，只能“领大家进门”后续的学习需要大家掌握学习方法之后自己突破。因此官网的使用是非常重要的一部分。**ol** 的官网大致分为四个板块，基本的文档介绍，以及演示案例，API 具体用法，源代码四个部分。这当中大家需要经常打开学习的是案例和 API 用法两个部分。很多需求和功能的开发没有思路的时候，就可以借助现有的案例进行改造，这是开发过程中再常见不过的方式了。笔者在写本书的时候 **openlayers** 已经默默的更新到了 8.2.0，但是笔者第一次接

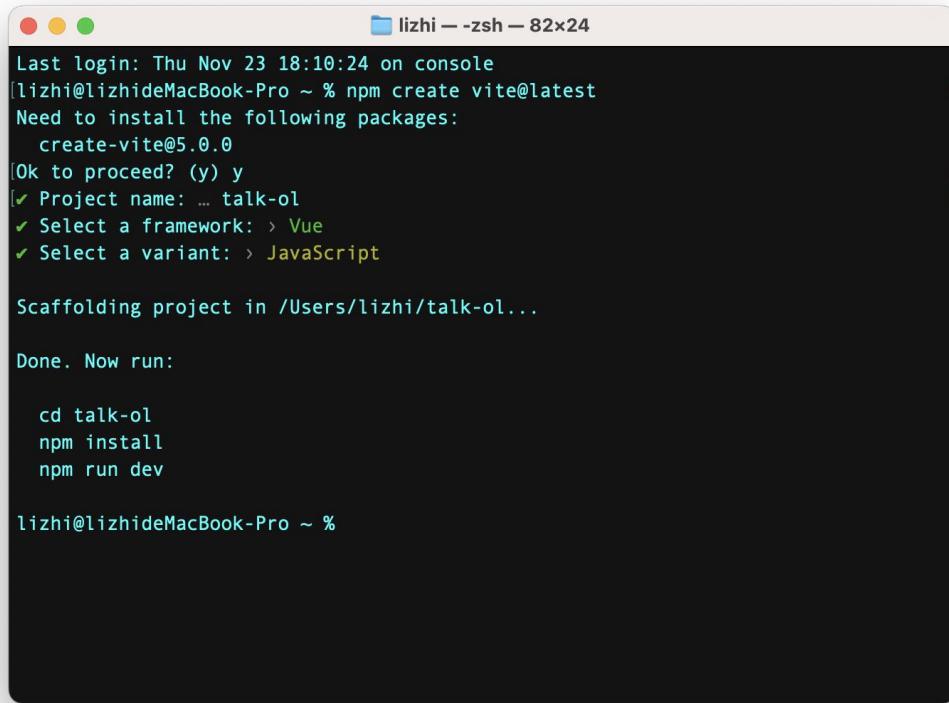
触 **openlayers** 的时候版本才 3.2.1，可见这些年 **openlayers** 发展速度如此之快。不仅发展速度快，**openlayers** 也是紧跟着开源行业的节奏，不断的将自己的代码结构进行优化，从最初的纯 **html** 和 **js** 以及 **css** 文件配合，到如今的工程化构建，甚至 **openlayers** 的官方都在使用当前最新最流行的 **vite** 脚手架，真的是非常的“时髦”。

跟大家讲这个背景的意思就是我们在接下来讲解 **openlayers** 的时候会结合 **vue** 框架进行讲解，因为纯原生的 **html** 和 **js** 用现在的版本构建起来相反会更麻烦。不过和 **vue** 框架结合也是过去各位读者的心声。所以没有 **vue** 基础的小伙伴可能需要先简单的学习一下 **vue** 了，我们对各位 **vue** 的水平要求不高，会使用 **vue2** 写一个简单的商品页面或者是官网页面即可。同时也是为了照顾基础比较差的同学以及一些不愿意在学习新技术的同学，我们在编码的过程中尽量还是使用 **vue2** 的语法，采用对新手比较友好的选项式 API 的写法。

好下面我们来使用 **openlayers+vue** 来初始化一个工程，脚手架我们使用 **openlayers** 官网也用的 **vite**，如果喜欢使用 **vue-cli** 脚手架的小伙伴们就自行百度吧，我在这里就不重复讲解了。首先我们保证自己的计算机上有 **node** 环境，大家可以去官网下载一个 **node.js** 并安装，安装成功之后可以搜索 **vite** 官网，进入到官网页面之后点击“Get Start”，然后在你计算机的控制台输入：

```
npm create vite@latest
```

然后输入项目名称，以及要使用的框架，我们这里起个名字叫做“**talk-ol**”，框架我们选择 **vue**，语言选择 **javaScript**，这样的话就会自动为你更新并且创建一个 **vite** 项目。



```
lizhi@lizhidemacBook-Pro ~ % npm create vite@latest
Need to install the following packages:
  create-vite@5.0.0
Ok to proceed? (y) y
✓ Project name: ... talk-ol
✓ Select a framework: > Vue
✓ Select a variant: > JavaScript

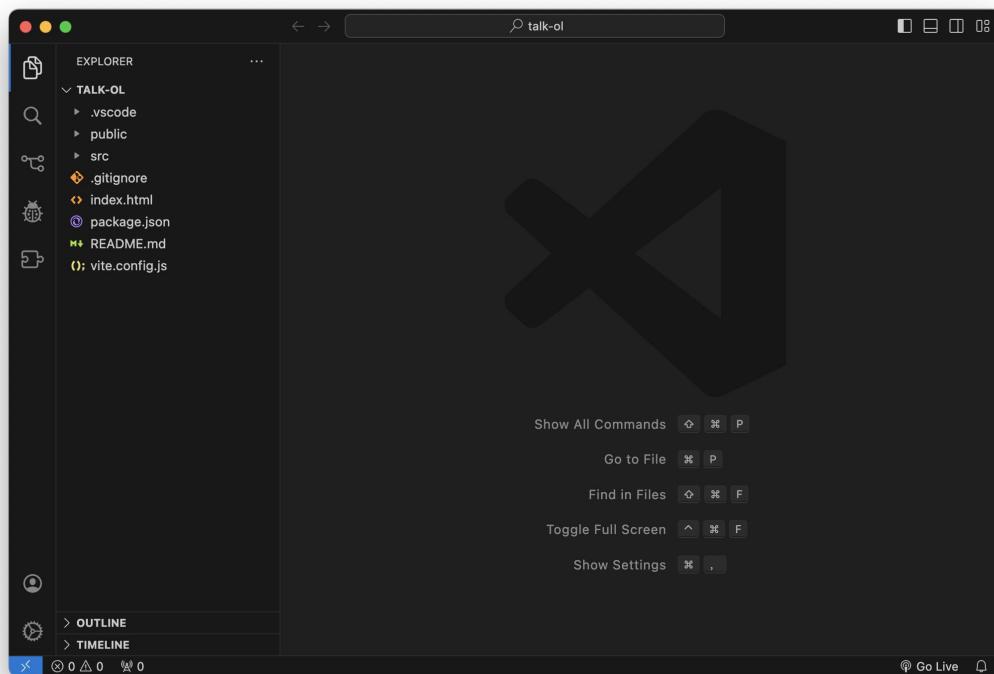
Scaffolding project in /Users/lizhi/talk-ol...

Done. Now run:

  cd talk-ol
  npm install
  npm run dev

lizhi@lizhidemacBook-Pro ~ %
```

使用 `vs code` 编辑器打开我们的工程文件夹可以看到如下:



好，我们接下来就可以使用这个工程来进行编码了，第一步我们要在工程中先引入 `openlayers` 的依赖，我们在终端使用 `npm install ol` 的命令来安装最新的依赖。同时

也提醒各位在使用本书附带的源码之前也记得使用 `npm install` 来安装对应的依赖。

接下来我们就可以使用 `openlayers` 来开发我们的第一个组件了。我们可以新建一个 `BaseMap` 组件作为基础底图展示的组件，然后测试这个组件能否展示 OSM 官方提供的底图，如果可以就证明我们的基础环境搭建好了。我们可以参考 `openlayers` 官网的写法，查看如何开始搭建一个 `hello world` 级别的应用，我们可以在官网示例页面中搜索“OGC Map tiles”然后观察这个示例，首先引入了构建组件需要的依赖资源：

```
import Map from "ol/Map.js";
import OGCMaptile from "ol/source/OGCMaptile.js";
import TileLayer from "ol/layer/Tile.js";
import View from "ol/View.js";
import "ol/ol.css";

//引入资源的时候要注意引入 ol 官方自带的 css 文件。随后就是地图代码的编写
const map = new Map({
  target: "map",
  layers: [
    new TileLayer({
      source: new OGCMaptile({
        url:
"https://maps.gnosis.earth/ogcapi/collections/blueMarble/map/tiles/WebMercatorQuad",
      })
    })
  ],
  view: new View({
    center: [0, 0],
    zoom: 1,
  })
});
```

在这里跟大家介绍一下，`openlayers` 的核心类名称叫做 `Map`，使用 `new Map()` 来创建一个地图实例。`openlayers` 和 `leaflet` 以及 `mapbox` 框架一样，初始化的时候都必须找到一个 `dom` 容器（一般是一个 `div`）作为挂载对象，也就是你必须首先制定地图要渲染在哪个容器当中，这样才好决定地图的宽高和位置。因此指定了属性 `target` 为 `map`，这个 `target` 的参数值是 `dom` 容器 (`div`) 的 `id` (注意只能是 `id` 而不是 `class`)，这样就能够告诉 `openlayers` (后文我们简称 `ol`) 地图的渲染位置和大小。因此我们需要提前指定好 `dom` 容器的样式。

至于其他的参数都比较好理解了，`view` 是 `ol` 中用于操作视图的对象，它可以控制你观察地图的状态，具体的有 `zoom` 表示缩放层级，表示指定地图初始化时处于什么样的缩放级别，`center` 表示地图的中心点，`layers` 表示地图中有哪些图层，毕竟地图可以叠加很多个图层。这样我们的一个基础的底图组件就搭建好了。各位不必担心其他不认识的概念，先理解这些已知的概念，后续的学习中会慢慢理解的更多。我们接下来可以将项目工程中自带的 `hello world` 组件删除掉，然后在根组件 `App.vue` 中引入我们刚才编写的 `BaseMap` 组件，随后启动项目就能看到我们的初始化页面啦～

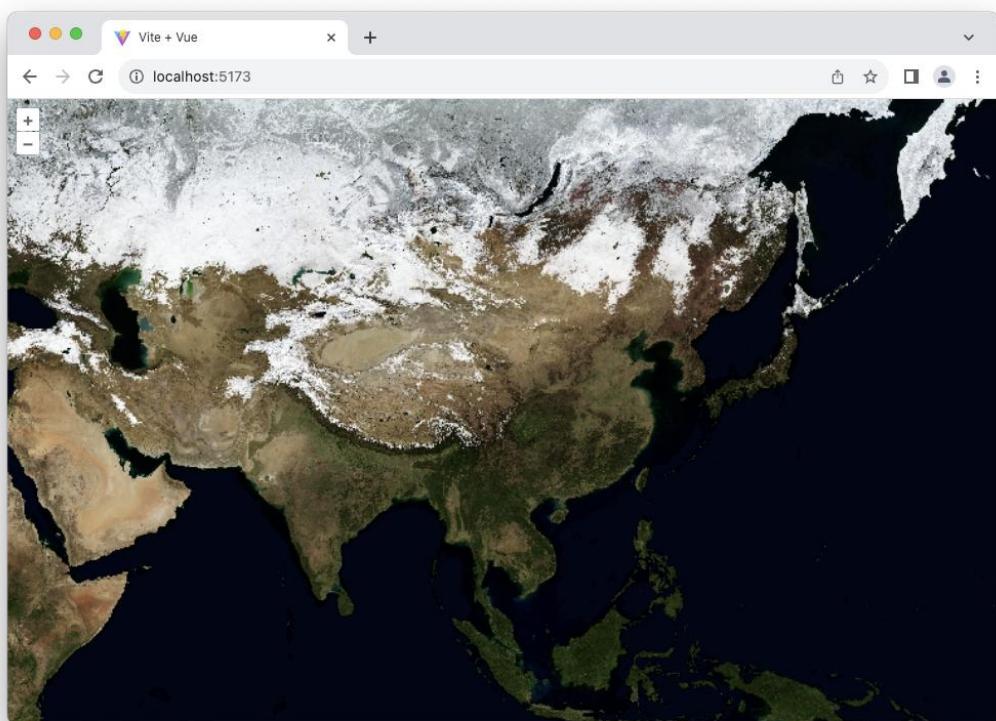


图 8.1.2-初始化 openlayers

那么到此为止我们就完成了 `ol` 项目的初始化工作，在这里给大家分享一些细节上的问题，在真实的项目开发过程中，设计肯定是多个组件之间协调配合工作，因此很多依赖资源都是组件中会引入的，我们不妨就把一些公共会用到的抽取出来放在头文件 `main.js` 中，这样以来所有的子组件都能够读取到这些依赖资源，后续编写代码的过程中就可以避免重复性的编写依赖引入内容了。

```

import { createApp } from "vue";
import "./style.css";
import App from "./App.vue";
//引入 css 文件，后续不必重复引入
import "ol/ol.css";
createApp(App).mount("#app");

```

另外还有一个非常重要的知识需要跟大家提前铺垫清楚，在 ol 中，也是支持双坐标系的。即同时支持 web 墨卡托 (EPSG: 3857) 和 wgs84 坐标系 (EPSG: 4326)，严格意义上来说 ol 不仅支持这两个坐标系，而是支持很多的坐标系，不过其他的坐标系需要开发者来自定义并转换，比如我们国家的 2000 坐标系，ol 也是支持的。

在 ol 中切换坐标系的方法很简单，ol 默认是 web 墨卡托坐标系，如果各位想使用 wgs84 坐标系（经纬度坐标系），可以在 view 对象中设置属性 projection: 'EPSG:4326'，即：

```

view: new View({
  projection: "EPSG:4326",
  center: [101.73497559, 27.50030558],
  zoom: 2,
  maxZoom: 8,
}),

```

## 第 2 节：基础底图加载

上一节给大家分享了如何初始化一个 ol 的应用，这一节我们来说一下真实开发场景中对于基础底图的加载，在国内 90% 的场景基础底图都会选择加载天地图，因此使用 openlayers 加载天地图也是各位必须学会的内容。关于天地图的知识我们就不多介绍了，我们需要知道的是天地图官方为我们提供了两种坐标系的底图服务，我们在前端 GIS 框架中只需要调用对应的服务即可正常的加载天地图的底图。因为天地图提供了两种方式的地图服务，一种是 xyz 的方式（相关知识会在第十一章第一节开源服务介绍中讲解），一种是 WMTS 的方式。那么在 ol 中有两种方式可以加载天地图，我们先来说第一种 xyz 方式，既然是 xyz 的方式，就必须使用 ol 中一个关键的 API 就叫做 xyz。添加天地图的代码如下：

天地图的 token 各位自行申请：

```
const token = "你的天地图 token";
const map = new Map({
  target: "map",
  layers: [
    new TileLayer({
      source: new XYZ({
        url:
          "https://t0.tianditu.gov.cn/DataServer?T=img_w&
          x={x}&y={y}&l={z}&tk=" +
          token,
      }),
    }),
  ],
  view: new View({
    center: [472202, 7530279],
    zoom: 10,
  }),
});
```

我们可以看到代码中一个非常关键的部分是 `new XYZ()` 这个方法就是用来加载所有符合 xyz 切片方式的地图服务。我们也可以看到 `new XYZ()` 的上层调用了 `new TileLayer()` 这个类，这个类是所有的瓦片图层的总类，我们下面将要给大家介绍的第二种加载天地图的方式也必须使用该类。

那么我们顺势开始第二种加载天地图的方式——WMTS 方式。

这种方式代码写起来就比较复杂了，而且需要各位的专业知识也比较多，我们先把源代码给大家放在下面，针对重点的地方在进行解释。

```
const token = "你的天地图 token";
const projection = getProjection("EPSG:3857");
const projectionExtent = projection.getExtent();
const size = getWidth(projectionExtent) / 256;
const resolutions = new Array(19);
const matrixIds = new Array(19);
for (let z = 0; z < 19; ++z) {
    resolutions[z] = size / Math.pow(2, z);
    matrixIds[z] = z;
}
const map = new Map({
    layers: [
        new TileLayer({
            source: new WMTS({
                url: "http://t0.tianditu.gov.cn/img_w/wmts?tk=" + token,
                layer: "img",
                matrixSet: "w",
                format: "tiles",
                projection: projection,
                tileGrid: new WMTSTileGrid({
                    origin: getTopLeft(projectionExtent),
                    resolutions: resolutions,
                    matrixIds: matrixIds,
                }),
                style: "default",
                wrapX: true,
            }),
        })
    ],
    target: "map",
    view: new View({
        center: [-11158582, 4813697],
        zoom: 4,
    }),
});
```

在上述的代码中，`const projection = getProjection("EPSG:3857");`代表着定义坐标系，当前使用的是 web 墨卡托投影坐标系。直到 `const map = new Map` 这一句之前都是计算瓦片层级以及行列号的部分，其实读到这里如果不清楚瓦片的请求加载逻辑的同学可以先去第十一章第一节先了解清楚瓦片的加载机制再回来读这段代码会清晰很多，这段代码根据分辨率和瓦片的尺寸，动态的算出了对应层级之下应该请求的瓦片的行列号，为什么在其他的框架中不用写这段代码呢？是因为其他框架已经把这个逻辑封装在框架内部了，不需要开发者再来手动写这段代码了。这样做其实有好处也有不好的地方。这段代码让用户自己写那么用户就可以根据自己定义的规则去切瓦片。然后自己根据规则计算行列号再加载瓦片。自定义的程度更高，这也是我们之前说 ol 的开源程度相比于 mapbox 要高的多的原因之一。相反 mapbox 虽然避免让开发者自己来写这段代码，但是同时也失去了自由。

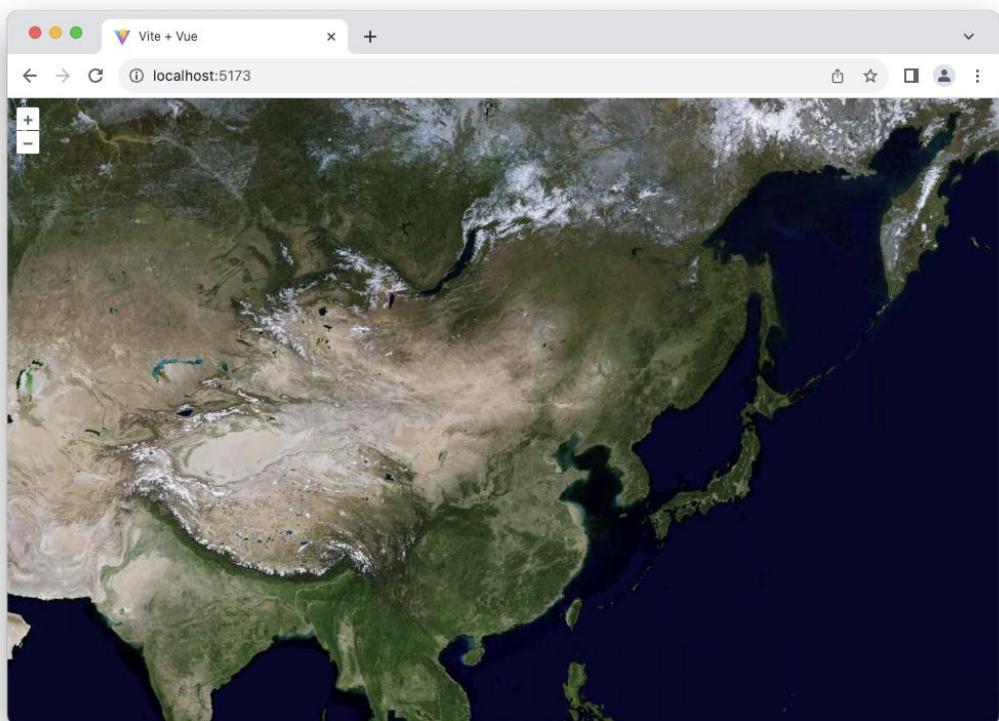


图 8.2.1-openlayers 加载天地图

从 `const map = new Map` 再往下就是常规的加载 WMTS 图层的逻辑了，这个我们上文也说过了，使用 `new TileLayer()` 和 `new WMTS()` 正常加载即可。这里面填的参数其实 WMTS 标准服务都告诉我们了，就拿天地图来举例，天地图官方提供的 WMTS 的地址为：

[http://t0.tianditu.gov.cn/img\\_w/wmts?SERVICE=WMTS&REQUEST=GetTile&VERSION=1.0.0&LAYER=img&STYLE=default&TILEMATRIXSET=w&FORMAT=til](http://t0.tianditu.gov.cn/img_w/wmts?SERVICE=WMTS&REQUEST=GetTile&VERSION=1.0.0&LAYER=img&STYLE=default&TILEMATRIXSET=w&FORMAT=til)

[es&TILEMATRIX={z}&TILEROW={y}&TILECOL={x}&tk=您的密钥](#)

可以看到 new WMTS 里面填的参数除了是计算出来的行列号以及投影信息之外，都可以在这个 url 中找到。对应填上就可以。

至此，各位已经学会了两种在 ol 中添加天地图的方式。如果各位想换不同风格的天地图，只需要在 url 参数中更换不同的关键字即可，比如各位现在如果想加载天地图提供的地形图风格的瓦片服务，就可以这样写：

```
new TileLayer({
  source: new WMTS({
    url: "http://t0.tianditu.gov.cn/ter_w/wmts?tk=" + token,
    layer: "ter",
    matrixSet: "w",
    format: "tiles",
    projection: projection,
    tileGrid: new WMTSTileGrid({
      origin: getTopLeft(projectionExtent),
      resolutions: resolutions,
      matrixIds: matrixIds,
    }),
    style: "default",
    wrapX: true,
  }),
});
```

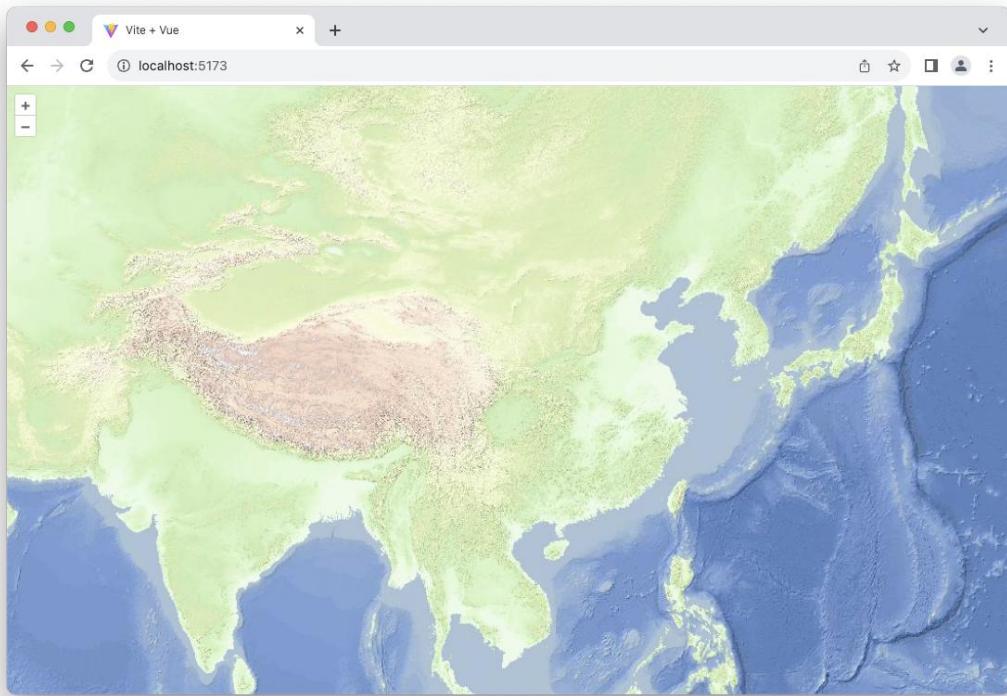


图 8.2.2-openlayers 加载天地图地形图

### 第 3 节：矢量图层加载

矢量图层也是项目开发过程中避免不了要打交道的。在 ol 中对于矢量数据格式的支持相比于 mapbox 要丰富的多，它支持 kml、gml、wkt、geojson 等。加载矢量图层用到关键的类叫做 `VectorLayer()` 它涵盖了我上述讲过的所有矢量数据类型，也就是说，想要在 ol 中加载矢量图层，就必须使用这个类，比方说我们使用这个类来加载一些 geojson 数据，我们可以像下面这样写代码：

```
import { hangzhou } from "../assets/hangzhou";
import GeoJSON from "ol/format/GeoJSON.js";
import Map from "ol/Map.js";
import View from "ol/View.js";
import { Fill, Stroke, Style } from "ol/style.js";
import { Vector as VectorSource } from "ol/source.js";
import { Vector as VectorLayer } from "ol/layer.js";
export default {
  name: "AddVector",
```

```
mounted() {
    //方式一读取本地静态文件进行加载
    const vectorSource = new VectorSource({
        features: new GeoJSON().readFeatures(hangzhou),
    });

    const vectorLayer = new VectorLayer({
        source: vectorSource,
        //给图层设置样式
        style: new Style({
            stroke: new Stroke({
                color: "rgb(35, 176, 189)",
                width: 2,
            }),
            fill: new Fill({
                color: "rgba(255, 255, 0, 0.1)",
            }),
        }),
    });

    //构建地图对象，并把刚才定义好的 layer 加入到图层数组中
    const map = new Map({
        layers: [vectorLayer],
        target: "map",
        view: new View({
            center: [0, 0],
            zoom: 2,
        }),
    });

    //定位聚焦到加载的矢量图层
    map.getView().fit(vectorSource.getExtent(), { size:
        map.getSize() });
};

});
```

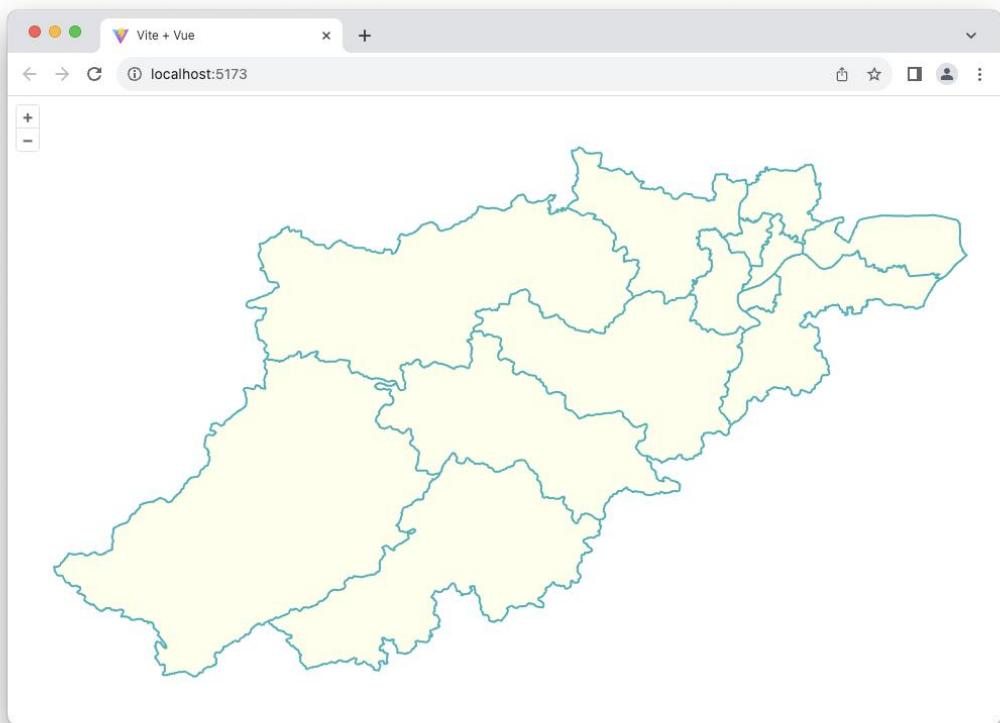


图 8.3.1-加载矢量数据

由于 openlayers 的 API 比较复杂，我们来认真解释一下上面这段代码。首先还是相关的需要用到的函数以及类进行导入，在 ol 中，任何图层的构建都遵循一个规范，那就是先构建一个 source（数据源），然后再构建一个 layer，将 source 作为参数传入，最后再通过 map 的 layers 参数或者是 map.addlayer()方法将图层添加到地图上，这一点在其他框架中也是类似。最后一段代码 map.getView().fit()方法是 ol 中用于定位的方法，它的主要参数是一个经纬度或者投影坐标范围，提供范围后 ol 框架会根据这个范围和当前浏览器的尺寸来找到一个自适应的缩放层级和中心点以定位到图层。

这是整体的逻辑，捋清楚以后我们再来看代码，确实是先 new 了一个 VectorSource 然后通过指定 features 这个参数将数据源传入，在这里要注意，我们选择了本地的 geojson 数据作为数据源，当然在这里还有很多其他的方式，比如你可以从服务器获得一份 geojson 数据，那么你的代码就得这样写了：

```
//方式二加载在线 geojson 数据
const vectorSource = new VectorSource({
  url: "https://openlayers.org/data/vector/ecoregions.json",
  format: new GeoJSON(),
});
```

只需要在 `url` 处填上服务器上访问 `geojson` 数据资源的地址即可。当然这里也可以是你本地静态资源的访问地址。

紧接着往下就是加载图层的部分了，要注意 `VectorLayer` 的几个关键参数，一个是刚才我们建立好的 `source`，另外就是 `style`，也就是控制矢量图层的样式参数，`style` 的写法也是多种多样的，`ol` 版本更新到 8.2.0 之后。`style` 的写法也借鉴了 `mapbox` 那一套，开始使用表达式的方式进行编写了。关于表达式的使用大家可以参考 `mapbox` 章节中表达式的用法。在这之前，`style` 只能通过自己指定边界样式 `stroke`、填充样式 `fill` 等写法来编写。

```
const vectorLayer = new VectorLayer({
  background: "#1a2b39",
  source: new VectorSource({
    url: "https://openlayers.org/data/vector/ecoregions.json",
    format: new GeoJSON(),
  }),
  style: {
    "fill-color": ["string", ["get", "COLOR"], "#eee"],
  },
});
```

说到样式我们就顺势来说一说如何在实际的项目中开发出用户喜欢的各种丰富样式的地图。首先是分类着色，这个是最经常用到的手法，即根据不同的属性值来给对应的区域着色。比如说我们现在统计杭州市各个区县的经济产值，得出了每个县区的具体生产值和排名情况，我们根据颜色的深浅来划分产值的不同，让用户从地图上一目了然的能够看到哪个区排名靠前哪个区还需努力。这听起来好像对比较差的区域比较残酷～好废话不多说我们直接来实现这个功能。

我们要实现分类着色的核心思路还是要对矢量图层的样式进行设置，因此入手还是关注这个 `style`，`style` 的写法除了上述两种之外还有第三种写法，那就是给一个回调函数，就像下面这样写：

```
const vectorLayer = new VectorLayer({
  source: vectorSource,
  //给图层设置样式
  style: this.styleFunction,
});

//构建地图对象，并把刚才定义好的 layer 加入到图层数组中
const map = new Map({
  layers: [vectorLayer],
  target: "map",
  view: new View({
    center: [0, 0],
    zoom: 2,
  }),
});

//定位聚焦到加载的矢量图层
map.getView().fit(vectorSource.getExtent(), { size:
map.getSize() });

function styleFunction(feature) {
  //注意观察这个打印，ol 会把每个要素作为参数返回给我们
  console.log(feature);
}
```

就像在 leaflet 中一样，我们如果在 `style` 的位置写了一个函数作为参数，那么这个函数将会作为回调函数来使用，回调函数的参数就是每个要素，回调函数的返回值将作为当前要素的 `style`。当我们 `console.log` 的时候就会展示出所有的要素，在这个示例中，每个要素 (`feature`) 对应的是杭州市的每个区县，那么我们就可以在回调函数中来操作这个 `feature` 或者是根据条件来判断 `feature`，根据不同的条件设置不同的颜色并且返回对应的颜色，所以我们可以像下面这样写：

```
function styleFunction(feature) {
    let value = feature.getProperties().value;
    let color = "";
    switch (true) {
        case 1000 < value && value < 2000:
            color = "rgb(255, 184, 48)";
            break;
        case 2000 < value && value < 3000:
            color = "rgb(255, 114, 73)";
            break;
        case 3000 < value && value < 4000:
            color = "rgb(255, 83, 73)";
            break;
        case 4000 < value && value < 5000:
            color = "rgb(177, 50, 84)";
            break;
        case value > 5000:
            color = "rgb(71, 19, 55)";
            break;
        default:
            color = "rgb(255, 184, 48)";
    }
    return new Style({
        stroke: new Stroke({
            color: "rgb(213, 217, 224)",
            width: 2,
        }),
        fill: new Fill({
            color: color,
        }),
    });
}
```

我们首先可以通过 `feature.getProperties()` 方法获取到 `geojson` 数据中的要素属性，我们的数据中有 `value` 字段来表示产值，那么我们获取到产值之后可以对产值做一个阶段性划分，并且给每个阶段赋上相应的颜色，最后我们再返回一个 `new Style()` 实例，这个实例中的 `color` 色值会根据 `switch` 的情况返回对应的颜色，这样就形成了我们的分类着色。

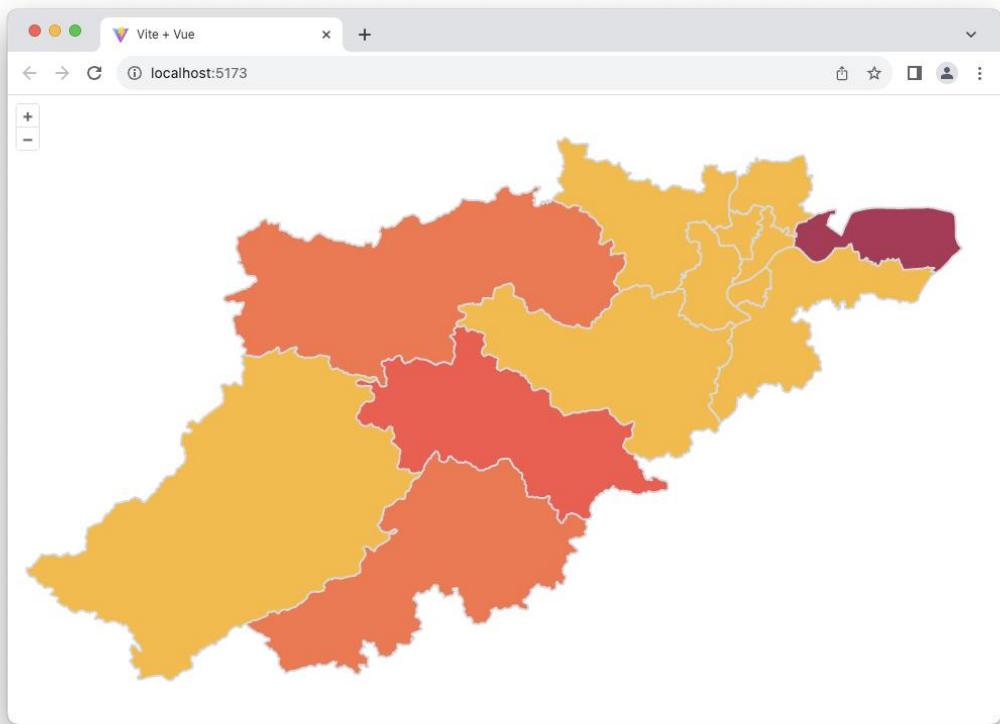


图 8.3.2-分类着色

其他的各种颜色附着也是类似的道理，在具体的实践开发中大家可以根据具体的业务场景来设置不同的颜色。

似乎现在的地图相比于最开始看起来好看多了，我们再来让这个地图更加美观一些，那么我们首先应该为地图添加文字，最起码每个行政区要显示出自己的名称来，那么这个操作也不难，我们也是需要在 `styleFunction` 中获取到要素的属性，然后再返回一个 `text` 样式即可，具体的代码如下：

```
import { Fill, Stroke, Style, Text } from "ol/style.js";
return new Style({
  stroke: new Stroke({
    color: "rgb(213, 217, 224)",
    width: 2,
  }),
  fill: new Fill({
    color: color,
  }),
  text: new Text({
    font: "16px sans-serif",
    fill: new Fill({
      color: "black",
    }),
    text: feature.get("name"),
  }),
});
});
```

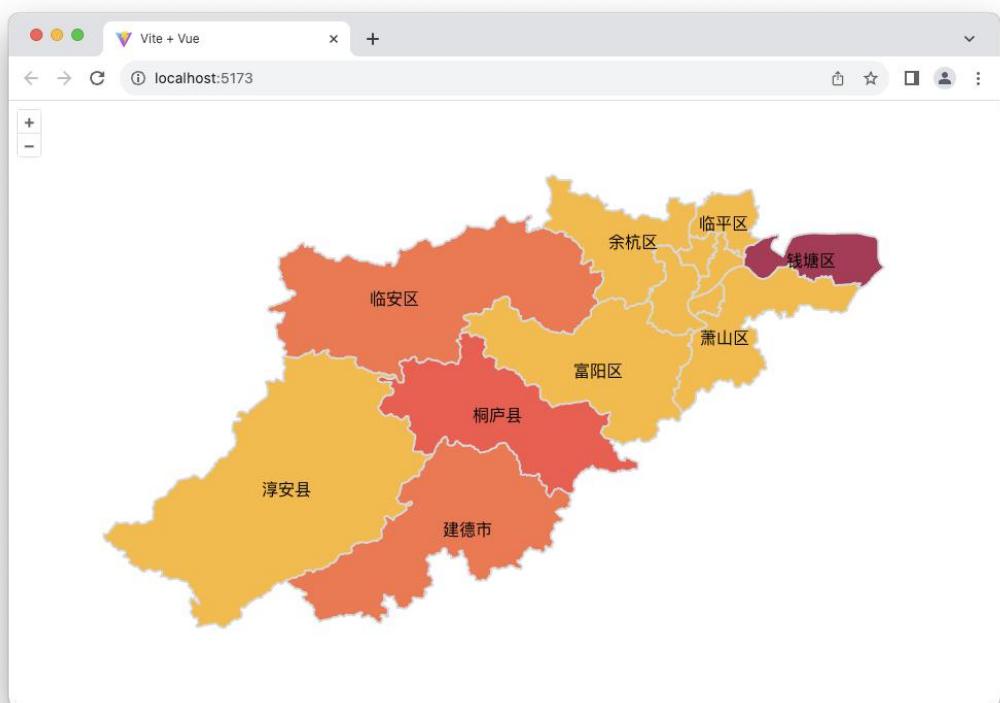


图 8.3.3-显示名称

需要注意的是，在使用的时候一定要记得引入 `Text` 这个类，也就是上述代码中的第一行，否则你会发现你可能写的没问题，但就是报错不出结果。

我们之前说过 `ol` 的特点是支持多种多样的矢量数据格式，我们不妨举个例子，继续给大家讲解在 `ol` 中是如何加载其他格式的矢量数据的，我们以 `wkt` 格式为例：

```
const wkt =
  "POLYGON((120.29126154543951 30.340552956159428,
120.28428156767069 " +
  "30.21655254907006, 120.40792688811769 30.213967541831877,
120.4757323864253 " +
  "30.32161859274214, 120.29126154543951 30.340552956159428))";

const format = new WKT();
const feature = format.readFeature(wkt);
const vector = new VectorLayer({
  source: new VectorSource({
    features: [feature],
  }),
  style: new Style({
    stroke: new Stroke({
      color: "rgb(213, 217, 224)",
      width: 2,
    }),
    fill: new Fill({
      color: "red",
    }),
  }),
});
```

```
const map = new Map({
  layers: [vector],
  target: "map",
  view: new View({
    center: [0, 0],
    zoom: 4,
  }),
});
map.getView().fit(vector.getSource().getExtent(), { size:
  map.getSize() });

```

我们还是为大家来解释下上述代码，在 ol 中加载 wkt 格式的矢量数据其核心做法就是转换，即把 wkt 转换成内部支持的（例如 geojson）等可以被框架本身所解析的数据格式，然后在对其进行加载，所以这当中就用到了非常关键的 `new WKT()`，这个实例中有可以用于读取 wkt 字符串的 `readFeature()` 方法。执行这个方法的返回值就是 ol 所能够支持的矢量数据。对于 wkt 格式的展现形式还不清楚的小伙伴们可以阅读第二章第 3 节矢量数据。剩余的加载矢量数据的 `source` 和 `layer` 和加载 geojson 数据都是一样的，在此就不赘述了。

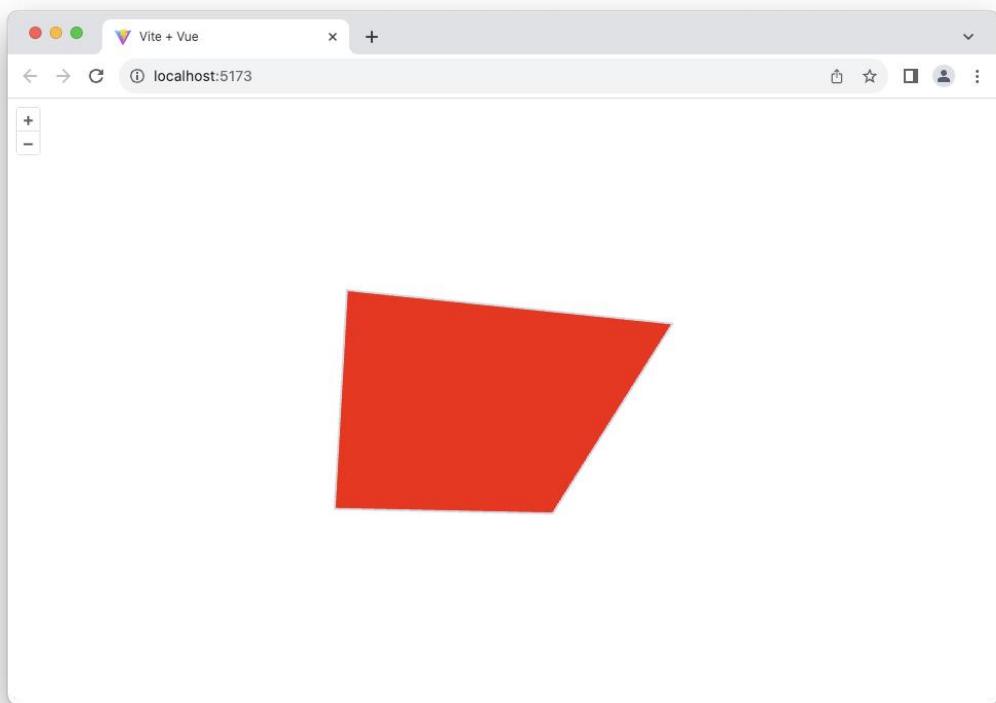


图 8.3.4-加载 wkt

讲到这里我们的矢量图层加载就要告一段落了，剩余的矢量格式的数据，例如 `gml`, `kml` 等大家如果有兴趣可以自己探究该如何加载，因为项目开发过程中这两种格式用的也不是很多我们就不在这里浪费篇幅讲了。关于如何和矢量图层进行交互我们会放在本章的第 5 节进行讲解。

## 第 4 节：栅格图层的加载

上一节我们讲了矢量图层的加载，我们这一节来研究一下栅格数据该如何加载，其实我们本章的第 2 节基础底图的加载就算是一部分栅格图层加载的知识，因为 WMTS 服务本身也属于栅格数据服务。我们这一小节再来跟大家讲一下其他形式的栅格数据的加载。

假设现在我们有一个需求，客户要求我们展示一张非常炫酷的拥有 3D 立体效果的某个地区的影像图，作为数字大屏的门面。按照常理，这个需求好像只需要前端同事将这幅图片贴在页面上即可，但是用户说我们希望这张图真的被应用起来，我们将来还要在这张地图上展示一些其他的数据，类似于点位，地块等等。那么我们就要确保加载到地图上的这张图片是有经纬度坐标信息的，这样才能与其他的数据进行比对，才能将其加载到精准的位置，这个原理我们在之前的章节中也有讲过，那么在 `ol` 中我们需要这样来写代码：

```
const extent = [101.73497559, 27.50030558, 102.43557243,  
28.18097996];  
  
const projection = new Projection({  
    code: "EPSG:4326",  
    extent: extent,  
});  
  
const map = new Map({  
    layers: [  
        new ImageLayer({  
            source: new Static({  
                url: image,  
                projection: projection,  
                imageExtent: extent,  
            }),  
        }),  
    ],  
    target: "map",
```

```
view: new View({
    projection: projection,
    center: getCenter(extent),
    zoom: 1,
}),
});
```

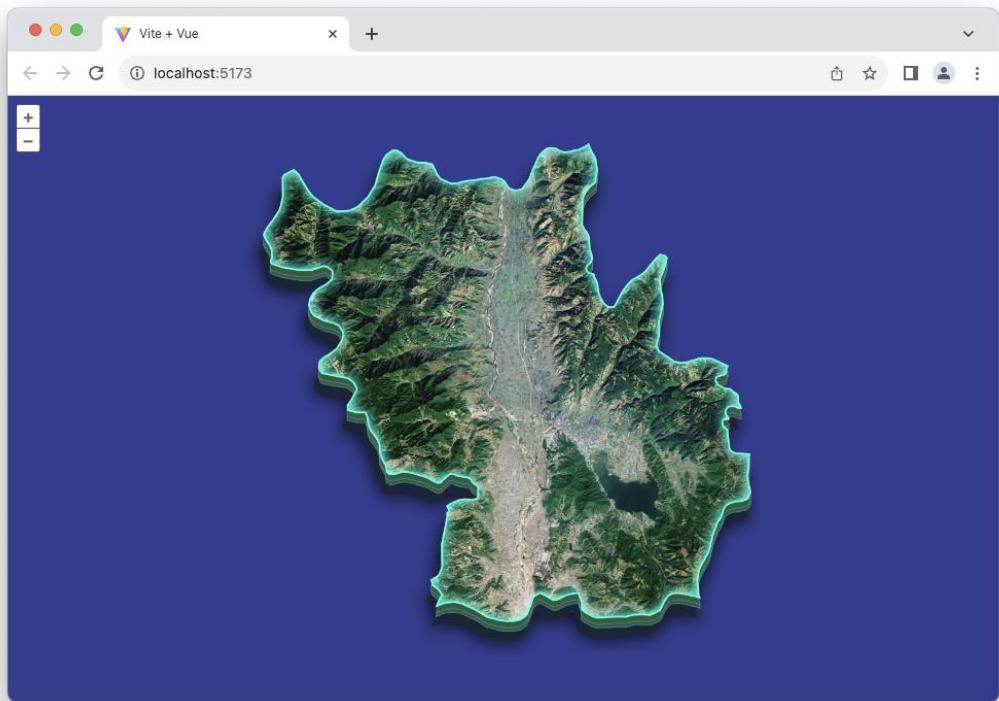


图 8.4.1-加载栅格图片

通过阅读上面的代码我们可以大体知道其思路，和 leaflet 以及 mapbox 等框架其实没有太大区别，核心思路都是一样的，先知道图片四个角的经纬度，然后在加载图片。不过不同的是，在 ol 中写法确实复杂了很多，因为 ol 中坐标系和投影参数是在 view 中定义的。所以在这里我们必须首先把 view 的 projection 改为 EPSG: 4326. 同时我们也限制了投影的边界范围为图片的四个角的经纬度范围。另外在加载图片的时候我们用到了 imageLayer 这个类，然后填入图片的 url 以及 projection 还有图片的经纬度范围即可。有的同学可能会问，图片的四个角的经纬度该如何获得？其实是这样的，这种具有 3D 效果的图片通常都是由 UI 同学提供的，因此图片本身是不具有任何经纬度信息，因此经纬度信息需要我们自己指定，那么问题就来了，这张炫酷的 3D 立体效果的图片出自于哪里？难道是 UI 同学凭空获

得的吗？肯定是我们 GIS 专业的同学给到 UI 的啊，那么如何制作一张这样的图片？我们可以现在地图上添加一个矢量图形，然后采用掩膜裁剪的技术（第九章会讲到）将我们的行政区部分提取出来，然后将这张图片给到 UI 同事，那么这张图片的四个角的经纬度岂不就是这个行政区的边界经纬度？行政区边界经纬度的获取不用我再重复了吧，你可以通过 `getBounds()` 或者借助 `turf.js` 等等手段获取一个矢量图层的 `bounds`，然后将这个边界范围用在上述代码中岂不是就完美的将这张图片加载到了地图上？如果你觉得可能会有不准和误差，那么你可以将行政区和图片同时添加到地图上观察二者的重合程度，二者完全重合的时候，这张图片的位置就没有任何问题了。

## 第 5 节：marker 与 dom

我们接连学会了如何加载矢量图层和栅格图层，但这远远还不够，因为地图上显示一些兴趣点和一些富文本信息也是非常常见的需求。比方说最简单的我们需要将一些兴趣点为位以图标的方式加载到地图上。我们该如何实现？我们一起来探究实现一下。

首先如果我们按照常规的思路，POI 兴趣点数据如果也是标准的 `geojson` 格式的数据，那么我们正常加载到地图上肯定官方会为我们提供相应的设置图标样式的方法。那么带着这个问题我们去 `ol` 的官网上寻找案例，我们果然搜索到解决方案，跟我们设想的一模一样，我们只需要在加载成功 `geojson` 格式的点数据之后设置其 `style` 中的 `image` 即可。那我们不如就直接进入深度一点的学习，我们直接按照不同类别使用不同图标的方式来加载这些 POI 数据，我们的示例数据如下：

```
let poi = [
  { name: "希望小学", lon: "124.789412", lat: "47.347325", type: "school" },
  { name: "蜜雪冰城", lon: "103.007012", lat: "39.53535", type: "school" },
  { name: "天猫超市", lon: "117.872028", lat: "27.100653", type: "market" },
  { name: "如家酒店", lon: "124.642233", lat: "48.042886", type: "market" },
  { name: "人民医院", lon: "87.038159", lat: "31.42261", type: "hospital" },
  { name: "和平饭店", lon: "117.038159", lat: "31.62261", type: "school" },
  { name: "银泰城", lon: "121.038159", lat: "29.42261", type: "market" },
  { name: "海底捞", lon: "111.038159", lat: "28.42261", type: "school" },
];
```

对于 POI 数据我们最关心的字段其实就只有 2 个，即经度和纬度。其他的属性字段对于我们来讲其实都不重要，因为肯定会随着业务的不同而发生变化。那么我们如果想把上面的数据以 geojson 格式加载到地图上，势必首先得经过一个转换的过程了，这个转换的过程我们在讲 mapbox 的时候也讲过，比较考验大家的前端基础，下面我将转换方法直接贴出，如果各位能够自己手写出来最好，如果写不出来，着急开发项目，那么也可以直接拿去用。

```
function json2Geojson(json) {  
    const result = { type: "FeatureCollection", features: [] };  
    json.forEach((j) => {  
        const feature = {  
            type: "Feature",  
            properties: {},  
            geometry: { type: "Point", coordinates: [] },  
        };  
        feature.properties = j;  
        feature.geometry.coordinates = [j.lng, j.lat];  
        result.features.push(feature);  
    });  
    return result;  
}
```

转换好之后我们可以就可以进行加载过程啦，加载点位和加载行政区等面数据其实使用的 API 都是一样的，只需要在设置样式的时候做一些变化：

```
styleFunction(feature) {  
    let type = feature.getProperties().type;  
    let icon = void 0;  
    switch (type) {  
        case "school":  
            icon = a;  
            break;  
        case "market":  
            icon = b;  
            break;  
        default:  
            icon = c;  
    }  
}
```

```
return new Style({
  image: new Icon({
    crossOrigin: "anonymous",
    src: icon,
    scale: 0.6,
  }),
});
}

let geojson_poi = this.json2Geojson(poi);
const vectorSource = new VectorSource({
  features: new GeoJSON().readFeatures(geojson_poi),
});

const vectorLayer = new VectorLayer({
  source: vectorSource,
  //给图层设置样式
  style: this.styleFunction,
});

//构建地图对象，并把刚才定义好的 layer 加入到图层数组中
const map = new Map({
  layers: [vectorLayer],
  target: "map",
  view: new View({
    center: [0, 0],
    zoom: 2,
  }),
});
//定位聚焦到加载的矢量图层
map.getView().fit(vectorSource.getExtent(), { size:
map.getSize() });
}
```

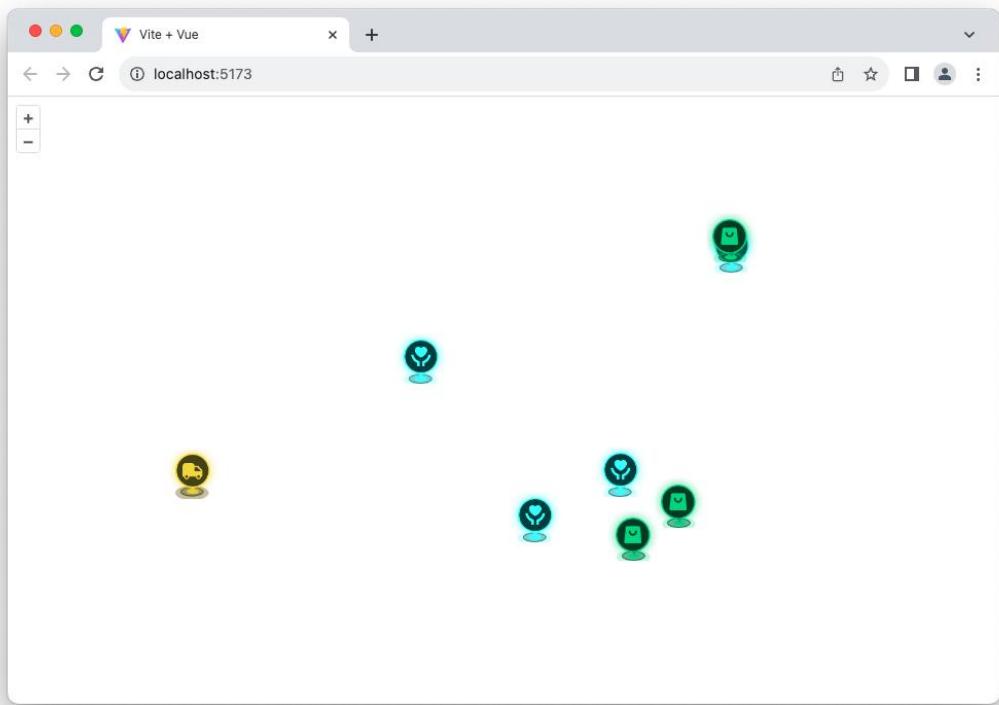


图 8.5.1-加载图标点位

在 leaflet 和 mapbox 中有专门的一个类别叫做 marker，但是在 ol 中，抱歉，对不起，没有。ol 中只有一个 overlay 可以使用，有的同学会问：老师，ol 中的 overlay 好像可以高度自定义 html，我在 overlay 里面写 img 元素，然后把这个图标以 overlay 的形式展现在地图上不就可以了嘛？是的可以，但是这样做不好，因为这样做实际上相当于是把很多 dom 元素叠加到 canvas 元素之上，少量的几个图标还好，如果几千个上万个的时候。这样的操作会把页面卡爆的同学，因此不建议在 ol 中使用 overlay 操作图标。

既然说到了 overlay，那么我们顺势来说一说 overlay 的用法，overlay 这个类的核心作用就是在地图上展示 dom 元素 (div, img, span 等等)，最常用的地方其实就是点击弹窗，比方说我现在想要实现点击地图之后弹窗展示我点击之处的经纬度信息，我们可以这样写代码：

```
let container = document.getElementById("info");
let overlay = new Overlay({
  element: container,
  positioning: "center-center",
  autoPan: {
    animation: {
      duration: 250,
    },
  },
});
const map = new Map({
  target: "map",
  layers: [
    new TileLayer({
      source: new XYZ({
        url:
"https://t0.tianditu.gov.cn/DataServer?T=img_w&x={x}&y={y}&l={z}&tk=" +
        token,
      }),
    }),
  ],
  overlays: [overlay],
  view: new View({
    center: [0, 0],
    zoom: 1,
  }),
});

map.on("singleclick", (e) => {
  container.innerText = e.coordinate;
  overlay.setPosition(e.coordinate);
});
```

思路也很简单，第一步你需要定义好一个你需要展示的 dom 元素，比如写好一个 div，然后新建一个 overlay，把你的 dom 元素作为 container 传入，这样 ol 就知道你想展示哪个

div，接着你需要在地图的配置项 `overlay` 中写入你声明好的 `overlay`，地图 `map` 对象的 `overlays` 参数是一个数组，也就意味着你可以写多个 `overlay` 传入，这样的话你可以在地图上展示不同类型的 dom 元素。最后就是监听鼠标的点击事件了，即当鼠标点击地图的时候，我们对这个行为进行监听，当用户点击完成后获取点击位置的坐标信息，然后将坐标信息作为弹窗内容展示在弹窗里。这一点如果不清楚不要紧，我们下一节会重点讲这个问题，本节大家只需要掌握如何创建 `overlay` 即可。

## 第 6 节：交互与事件

我们接着上一节的内容来讲，地图通常不是仅仅展示在页面上这么简单，很多时候地图也要与用户进行交互，地图的交互的方式也有很多，首先仅对于地图来讲，就有缩放，双击，单击，右击、拖拽等很多个事件，在这些事件中都可以写交互逻辑。对于图层来讲，也有很多的交互类型比如鼠标悬浮于图层之上，以及鼠标点击图层等等。我们这一小节来了解一下 `ol` 中的交互与事件。

首先对于地图的交互来讲，我们可以使用 `map.on()` 进行监听，这一点和 leaflet 以及 mapbox 都是一样的。就那我们上一小节的例子来讲，我们使用：

```
map.on('singleclick', e=>{ })
```

这样的写法来监听用户单击地图的事件，在回调函数中写监听到此行为的后续操作，上一节当中我们监听之后将用户点击的坐标信息展示在了弹窗里。再举个例子比方说我们想实现一个效果，当缩放层级小于 6 时我们展示天地图影像图，反之我们显示天地图的矢量风格地图，我们需要这样写代码：

```
const token = "你的天地图 token";
const tdt_img = new TileLayer({
  source: new XYZ({
    url:
      "https://t0.tianditu.gov.cn/DataServer?T=img_w&x={x}&y={y}&l={z}&tk=" +
      token,
  }),
  visible: false,
});
```

```

const tdt_vec = new TileLayer({
  source: new XYZ({
    url:
      "https://t0.tianditu.gov.cn/DataServer?T=vec_w&x
      ={x}&y={y}&l={z}&tk=" +
      token,
  }),
  visible: true,
});

const map = new Map({
  target: "map",
  layers: [tdt_img, tdt_vec],
  view: new View({
    center: [472202, 7530279],
    zoom: 0,
  }),
});
map.on("moveend", (e) => {
  const zoom = map.getView().getZoom();
  if (zoom > 6) {
    tdt_img.setVisible(true);
    tdt_vec.setVisible(false);
  } else {
    tdt_img.setVisible(false);
    tdt_vec.setVisible(true);
  }
});

```

我们来解释一下思路，首先我们需要把天地图两种类型的图层都加载到地图上，然后我们通过 map 的 moveend 事件来监听地图的缩放，一旦用户缩放了地图就会自动触发这个事件，然后我们在这个事件中监听地图的缩放层级 zoom，如果 zoom 比 6 大我们展示天地图影像图，反之我们展示天地图的矢量风格地图。设置图层的显示隐藏可以调用图层的 setVisible()方法，传参 true 代表显示，false 代表隐藏。具体的实现效果各位可以在本书附

带的源码文件夹中 `MapEvent.vue` 组件中查看。

接下来我们再来了解一下图层的交互和事件，假设我们现在想要实现一个效果：当鼠标悬浮到某个矢量图层要素上方的时候，该要素高亮显示，当点击这个要素的时候，弹窗展示该要素的属性信息。如何展示弹窗这个功能我们已经在上一节中跟大家分享过了，主要我们现在是要清楚如何捕捉到用户鼠标悬浮，点击到图层的事件。鼠标悬浮到元素时元素高亮显示，在 `ol` 中我们这样来做：

```
let selected = null;
map.on("pointermove", (e) => {
  if (selected !== null) {
    selected.setStyle(undefined);
    selected = null;
  }
  map.forEachFeatureAtPixel(e.pixel, (f) => {
    selected = f;
    f.setStyle(
      new Style({
        stroke: new Stroke({
          color: "rgb(13, 247, 219)",
          width: 2,
        }),
        fill: new Fill({ color: [13, 247, 219, 0.2] }),
        text: new Text({
          font: "16px sans-serif",
          fill: new Fill({
            color: "black",
          }),
          text: f.get("name"),
        }),
      })
    );
    return true;
  });
});
```

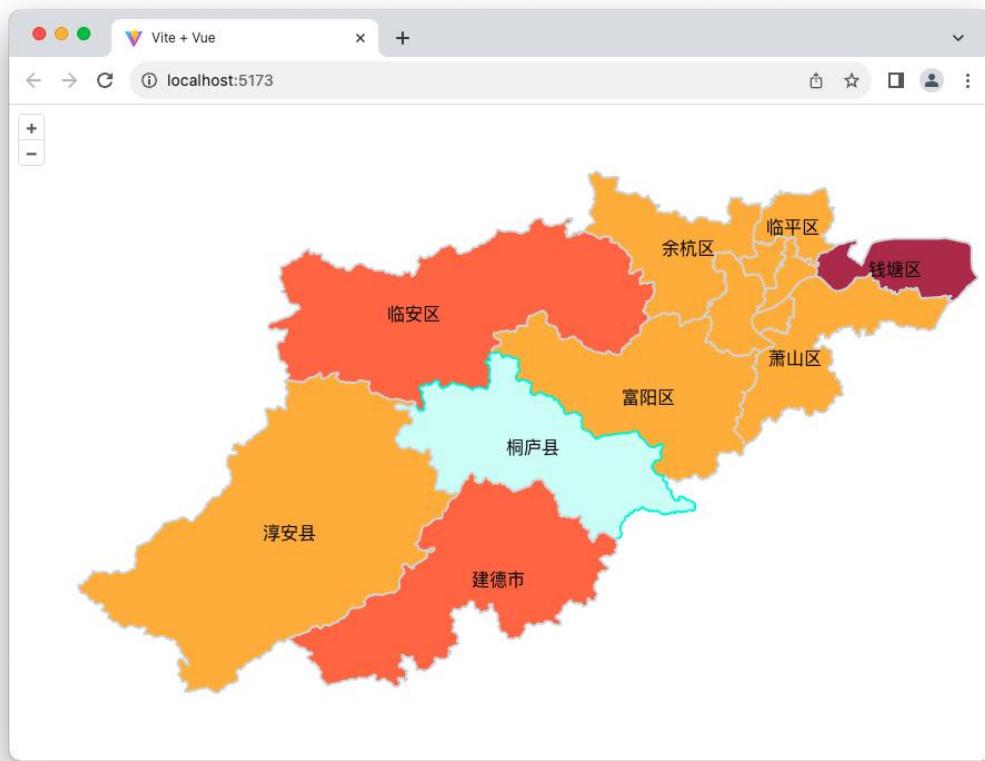


图 8.6.1-鼠标悬浮高亮

这段代码看起来极其复杂，对于新手来讲可能要懵圈了，我们来详细讲解一下，首先我们正常的去加载一个矢量图层，随意加载，加载成五颜六色也行。然后我们去写一个地图的监听事件，监听鼠标在地图上移动的事件，事件名称叫做“`pointermove`”，类似于普通的鼠标悬浮事件 `mousemove`。然后在这个事件的监听回调函数当中调用一个函数来查询地图上的矢量要素，方法叫做“`forEachFeatureAtPixel`”，这个方法根据名字也能知道是什么意思，即根据像素遍历要素，像素在这里值得就是鼠标的位置，也就是说这个方法是根据鼠标的位置来获取要素的，那么只需要把鼠标的位置作为参数传入即可，那么我们可以看到调用这个方法的时候我们传入的是 `e.pixel`，可以理解为传入鼠标点所在的像素，`ol` 框架内部会根据这个像素去比对当前地图上这个像素位置是否有矢量图层要素，如果有，将会把这个要素 `feature` 返回给我们，这也就是这个方法的回调函数中所返回的那个“`f`”，“`f`”即为 `ol` 检索到的要素，拿到这个要素之后的操作就简单了，肯定是给这个要素设置新的样式，但是你还不能简单的只是给他设置一个新样式，因为你要考虑当鼠标移开要素的时候，我们还必须恢复原来的样式，那么我们肯定需要一个状态来判断当前鼠标所在的位置是否有要素，如果有我们设置新样式，如果没有，我们要把刚才设置的新样式清除掉。于是乎我们在最外面设置一个变量 `selected`。这个变量用于记录我们选中的要素，比如说我们在第一次的鼠标悬浮

过程中选中了一个要素 a，那么我们就把 a 要素赋值给 `selected`，这样的话下一次鼠标离开 a 要素的时候，我们就知道了刚才选中的是 a 要素，然后我们使用

```
selected.setStyle(undefined)
```

来取消掉之前给 a 元素赋值的新样式，并且把 `selected` 置空以方便其接收下一次选中的要素。整个逻辑思路整理起来好像真的很复杂，但这也正是 ol 的特点，给你足够的灵活性，但是如果基础不好，可能这段路走起来异常艰难。

建议各位花点时间好好消化一下上部分内容，毕竟确实很难理解，尤其对于新手来讲。

我们接下来来思考如何实现点击弹窗展示要素的属性信息。我想大致思路我们已经能够捋清楚了。首先我们还是监听地图的点击事件，然后通过鼠标所在的位置获取到我们点击的要素，然后获取到要素的属性信息，和我们点击位置的坐标。然后我们利用上一节学过的 `overlay` 来展示我们的属性信息，具体的代码如下：

```
map.on("singleclick", (e) => {
  const feature = map.getFeaturesAtPixel(e.pixel);
  if (feature.length > 0) {
    let f = feature[0];
    container.style.display = "block";
    container.innerHTML =
      "<div>名称: " +
      f.get("name") +
      "</div>" +
      "<div>代码: " +
      f.get("adcode") +
      "</div>";
    overlay.setPosition(e.coordinate);
  } else {
    container.style.display = "none";
  }
});
```

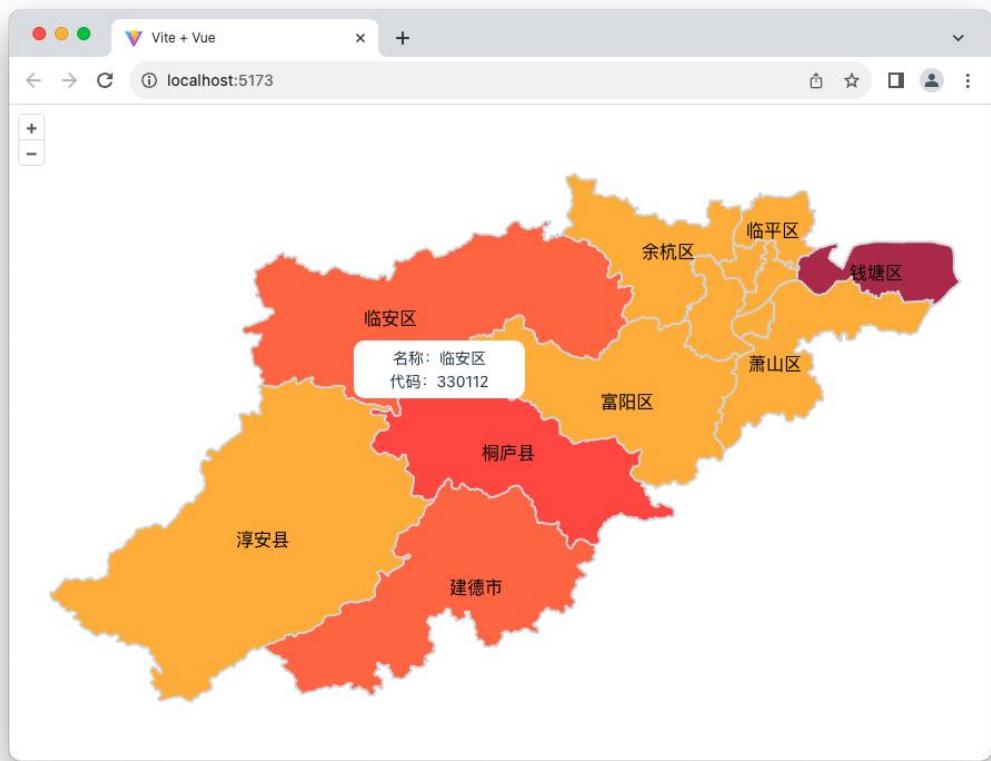


图 8.6.2-点击弹窗查询

声明 `overlay` 和写 `container` 的 `html` 部分就不在这里给大家展示了，大家可以去附带源码中查看具体的细节代码。在这里给大家换了一个函数，我们不用“`forEachFeatureAtPixel`”，而是采用另一个函数叫做“`getFeaturesAtPixel()`”，当然用之前的函数也可以实现这个功能，只不过比较麻烦，用这个函数可以省略鼠标位置是否捕捉到了要素的判断过程。也就是说我们可以直接通过鼠标位置去拿要素，如果能拿到那么这个函数会给我们返回一个数组，里面有所有当前鼠标位置下的要素（因为会有多个图层重叠的情况发生。所以框架本身会穿透拾取要素返回给我们），因此我们可以判断返回来的数组长度，如果是 0 则证明没有拾取到要素，那么这个时候我们可以清除掉页面上所有的弹窗。反之如果我们获取到要素，可以使用 `feature.get()` 方法将要素的属性拼接成 `html` 字符串，然后展示在 `overlay` 当中。

我在写这本书说的最多的一句话可能就是“举一反三”。我们要通过有限的例子学会无限的知识和技能。现在比如我们有一个需求是点击某些 `POI` 点位的时候弹窗展示属性，也就是把第 5 节中展示点位的案例与本节弹窗展示属性信息相结合，请各位先思考该如何操作，我会把答案放在附带的源码里 `IconPopup.vue` 组件中，供各位对证参考。

## 第 7 节 图形绘制与测量

由于 ol 高自由度的设计，在 ol 中绘制各种图形有无限的可能。可以说只有你想不到的，没有画不出来的。也正是因为这天马行空的想象力，官方很早就为大家准备了标准的绘制案例，大家可以直接在官网案例部分搜索“draw features”，即可看到官方为大家准备的丰富绘制案例，可绘制点线面等多种类型的要素。

<https://openlayers.org/en/latest/examples/draw-features.html>

官方已经高度封装了这些 API，只给各位留下了类型让大家填空，具体的绘制（draw）过程对于各位来讲是不可操作的。如果各位只是为了项目上使用那么可以直接 copy 案例代码了，因为已经写的非常成熟了，如果想知道原理和逻辑，那么不妨听我往下讲：其实任何框架的绘制原理都是一样的，都是几个状态之间的记录，比如当鼠标第一次落于地图之上我们需要记录鼠标的位置作为几何图形的起点，当然如果只是绘制一个点，那么这个记录的结果就是最终结果了。如果是线或者面，那么这个点只是起点。随着鼠标的移动。还需要记录移动过程中鼠标当前的位置，虽然这个位置不需要记录，但是要在这个位置和起点之间连成一条线，以提醒用户，现在所画的位置和方向。我在这里讲的位置其实是指经纬度，或者是投影坐标系下的坐标。都是一个道理。然后伴随着鼠标第二次，第三次，第四次……落下，分别记录每次落点的位置，直到最后一次鼠标落下（一般是双击结束绘制），我们把记录过的位置拼起来即能够形成一条线或者是一个多边形。具体的代码操作各位如果感兴趣可以阅读我在 mapbox 章节中讲过的自定义绘制及测量。清楚 geojson 结构的同学我讲到这里应该已经悟了，因此绘制并不是一件难事，反而很简单。

关于测量，其实官方也为大家封装好了成熟可用的代码，就差给各位界面让各位当作组件直接去使用了，同样的各位也可以在官网案例中直接搜索“measure”即可，代码都很成熟，几乎不需要改。即拿即用。

# 第九章：框架编程思想

## 第 1 节：抽象框架知识

我们可以对第五章、第六章、第七章，包括第八章的知识进行一个总结，从总结中得出经验。我们来回顾一下 WebGIS 开发中的几个关键要素。地图首先需要挂载到某个 div 容器

中，这是为了让用户自定义地图要素所要展示的实际尺寸大小。其次地图通常可以设置一些**基础底图**，而且通常这些底图的格式都是栅格数据。我们说在国内我们通常使用的是天地图提供的栅格瓦片服务。另外地图可以添加一些**业务图层**，这些图层通常都是业务方提出，而且图层的数据类型和数据格式都是多样的，从数据类型讲可以有点线面甚至是体要素。从数据格式讲有栅格数据也有矢量数据。所以图层的种类是丰富多样的。这样看来，底图也算是一个特殊的图层了。只不过底图通常用于表示全局范围的，大面积要素下的参考地物。市面上所有的 WebGIS 框架也都提供了各种支持底图加载的工具类。我们在加载这些业务图层的时候需要指定对应的数据源，在各个框架中几乎都是需要这样的操作，而且也都对数据源对象做了分类，大致分为栅格数据源和矢量数据源两大类，矢量数据源在 **maobox** 中比较单一几乎就是指 **geojson** 数据源，但是在 **openlayers** 和 **leaflet** 等框架中可能还会支持 **kml**、**gml** 等格式的数据源。

除了地图的基础数据，对于地图的**操作交互**也是很重要的一部分。我们对于栅格数据的操作比较单一，基本上只能浏览，缩放，改变透明度，对比度，饱和度等属性。对于矢量数据的操作交互通常比较丰富，一是能够灵活的修改矢量图层的样式，二是能够实时的跟矢量数据产生交互（点击查询、点击高亮等操作）。因此对于矢量数据的操作是地图的主要操作部分。其次还有一些非地图本身所能承载的要素，但是需要和地图一起呈现。就比如我们在 **mapbox** 那一章讲的 **dom** 元素的使用。利用 **dom** 元素其实可以展示出各种丰富的地图效果，这一点大家要明确并且牢记。

在这里我为大家整理我们通常解决项目需求时的一般开发思路。

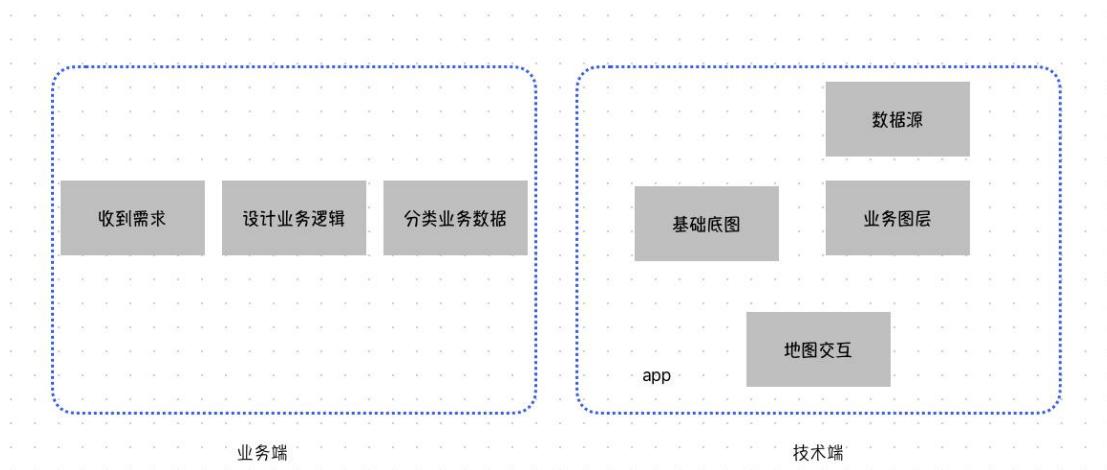


图 9.1.1-开发思路

## 第 2 节：掌握开发技巧

通过第五章、第六章、第七章以及第八章的学习各位应该有所体会。其实 WebGIS 前端框架的设计都是很相似的，尽管渲染原理不同，但是代码的设计思路都是相似的。举个例子，在初始化地图的时候都是需要首先去实例化一个类，这个类通常封装了对于地图的所有操作（属性，方法等），然后再通过一些参数的配置和传递来初始化一副地图。再比如在添加图层的时候，这些框架的做法都是先让你填写配置一个数据源 `source`，然后再配置一个 `layer` 对象，尽管在 leaflet 中你可以直接写 `addLayer()`，但是这个 `source` 创建的过程也是作者在源码中为大家内置好的，后续如果大家有精力去学习别的框架，例如 openlayers、cesium 等，大家也会发现也是类似的思路。再比如说地图当中的一些监听和事件操作。作者在创建框架的时候就为其中的对象属性添加了监听和事件回调，这会导致大家不太明白为什么可以这样操作。其实大家也不必纠结这些，如果你清楚 js 的事件监听和派发机制，你就会明白这并不算一件难事。js 的语言特性导致了这些框架的作者在设计的时候会采用符合语言本身标准的方式来跟开发者产生交互。

所以各位在实际的开发过程中要多去揣摩 API 的目的，一本书教不完一个框架内的所有 api，但是可以教给大家使用 api 的方法。每当你想要完成一个需求的时候，通常的做法是先去百度搜索别人是怎么做的，然后顺着别人的代码找到了一些案例，甚至是一些写好的代码，自己尝试之后可能会发现漏洞百出，各种错误频繁爆发，没关系，这都是正常的。于是你开始仔细研读别人的代码想要自己写出来这部分功能，然后你就聚焦到了官网的具体的某个 api。所以最终你还是需要仔细的去了解 api，多做一些测试和尝试你就明白这个 api 它到底想干什么？设计者在设计这个 api 的时候他想满足用户哪些需求？所以与其不断的百度别人，不如自己静下心来慢慢的去研究。其实不需要每个框架每个 api 都学习。很多时候，你只需要学习一遍，就能举一反三，应用到其他的框架中去。

## 第 3 节：矢量样式美化

关于矢量图层的样式问题，我们直接跟大家举个特别实际的例子，mapbox 官网提供了很多样式参数，用于调整图层的样式，这些参数不可能快速的一次性掌握，但是大家可以在空闲的时候花点时间去揣摩揣摩这些参数，因为框架的开发作者既然为大家提供了这个 API 就注定是希望大家使用的，否则就没有了意义。比如说一个特别有意思但是容易被忽略的

API——'line-blur'。这是用来模糊线条的，就是让线要素变得模糊，那正好我们可以利用这一点，既然可以模糊，是否可以考虑尝试渐变？是否以后在加载行政区图形的时候对其边界做一定的处理？能否形成阴影效果？能否形成发光效果？

于是乎我们可以利用如下的代码进行尝试：

```
const small = turf.transformScale(  
  turf.polygon(huangshan.features[0].geometry.coordinates),  
  0.97  
);  
  
map.addSource("huangshan-source", {  
  type: "geojson",  
  data: huangshan,  
});  
  
map.addSource("huangshan-small", {  
  type: "geojson",  
  data: small,  
});  
  
map.on("click", (e) => {  
  map.addLayer({  
    id: "haungshan-layer",  
    type: "fill",  
    source: "huangshan-source",  
    paint: {  
      "fill-color": "rgba(6, 248, 248, 0.2)",  
    },  
  });  
  
  map.addLayer({  
    id: "haungshan-line",  
    type: "line",  
    source: "huangshan-source",  
    paint: {  
      "line-color": "rgba(6, 248, 248, 1)",  
      "line-width": 2,  
    },  
  });  
});
```

```
map.addLayer({
  id: "haungshan-inner",
  type: "line",
  source: "huangshan-small",
  paint: {
    "line-color": "rgba(6, 248, 248, 0.3)",
    "line-width": 18,
    "line-blur": 8,
  },
});
});
```

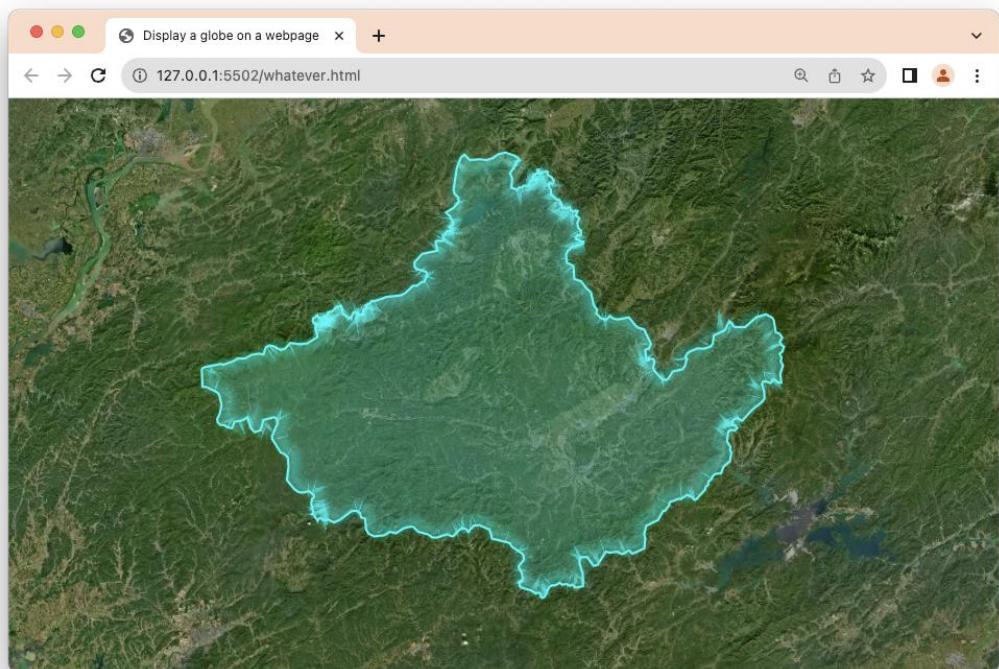


图 9.3.1-制造模糊发光效果

## 第 4 节：掩膜技巧

再比如现在很多项目中要求只展示客户关心的区域的地图。比如说杭州市的项目，在页面上只需要展示杭州市的地图，其他区域不想展示，我们该如何来做？一个非常常规的想法

就是将杭州周围的地图都进行裁剪，只保留杭州区域，但是这样做的成本很大，每次都需要裁剪不说，如果是遇到系统中需要随时切换其他城市并且只展示该城市的需求。这样的方式就行不通了。那么我们不妨可以换种思路，既然是只想看到本城市，其余的地方不关心，那是否可以把其余的地方抹掉？如果说我们有办法把其他的地方遮盖起来，这样也能达到相应的效果不是吗？因此我们一个非常关键而又重要的技术——**掩膜**就诞生了。

掩膜的原理正如我上面所说，将我们不关心的地方遮盖起来。如何遮盖成了目前最关键的问题，其实方式有很多种，我在这里给大家讲最常见的一种，那就是相交取反。我在之前第一章预备知识铺垫的时候就跟大家讲过世界的经纬度范围是  $[-180, -90, 180, 90]$ ，那么我们就可以把这个范围看作是一个矩形，如果我们用这个矩形和我们关心的城市图形数据做一个相交操作，再取除交集之外的部分，不就是我们需要的部分吗，我们再对这个相交取反的图形赋上一定的颜色就可以做到掩盖其余地图的功能。



图 9.4.1-掩膜技巧

这样的话我们只需要获取到上图中蓝色的部分对其进行着色即可遮盖住地图的其余部分。如果是需要展示影像图，那么我们完全可以在关心的城市区域覆盖一个透明的多边形，然后再地图的其余部分覆盖一个半透明的图层，即将上述蓝色部分多边形着色为半透明，得到如下的效果：



图 9.4.2-掩膜提取

再比如，我们如果把周围的地图全部遮盖住，也可以在白色的页面中做到影像地图凸显的效果：



图 9.4.3-聚焦区域

清楚了原理之后我们就要进入实践操作了，实际的相交操作取反操作也有很多种办法，大家可以借助 turf.js 完成，或者直接将 geojson 数据和世界范围的  $[-180, -90, 180, 90]$

进行相交也可以，即将 `[-180, -90, 180, 90]` 直接丢进 geojson 数据的 `geometry` 的 `coordinates` 中即可，这就考验各位的基本功底了，早在讲 geojson 的时候就埋过伏笔 geojson 格式真的很重要，掌握清楚其原理才能想明白为何这样做是可以的，实际上 geojson 记录多边形就是按照多边形的每个经纬度坐标来记录的，那么我们如果向数据中加入了 `[-180, -90, 180, 90]` 就意味着构建了一个新的多边形，多边形，以 `-180, 90` 开始，也是以 `-180, 90` 结束，那么就构成了一个凹多边形，这个多边形就是整个世界的范围，但是除了我们关心的城市区域。下面将核心代码提供给大家以做参考：

```
function generateMask(geojson) {
  let coord = [
    [
      [-180, -90],
      [-180, 90],
      [180, 90],
      [180, -90],
      [-180, -90],
    ],
  ];
  var country = turf.polygon(coord);
  if (geojson.features.length > 1) {
    geojson = this.unionPolygon(geojson);
  }
  if (geojson.features[0].geometry.type === "MultiPolygon") {
    var feature = turf.multiPolygon(
      geojson.features[0].geometry.coordinates
    );
  } else {
    var feature =
      turf.polygon(geojson.features[0].geometry.coordinates);
  }
  var diff = turf.difference(country, feature);
  return diff;
}
```

```
// 合并多个多边形，不管是 multipolygon 的集合还是 polygon 的集合
function unionPolygon(featureCollection) {
  let newCollection = {
    type: "FeatureCollection",
    features: [
      {
        type: "Feature",
        properties: {},
        geometry: {
          type: "MultiPolygon",
          coordinates: [],
        },
      },
    ],
  };
  if (featureCollection.features.length > 0) {
    featureCollection.features.forEach((f) => {
      if (f.geometry.type === "MultiPolygon") {
        newCollection.features[0].geometry.coordinates.push([
          f.geometry.coordinates[0][0],
        ]);
      } else if (f.geometry.type === "Polygon") {
        newCollection.features[0].geometry.coordinates.push([
          f.geometry.coordinates[0],
        ]);
      }
    });
  };
  return newCollection;
}
}
```

像掩膜这样的操作是可以无视框架的，无论你使用何种框架，都可以借鉴这种思路，来完成展示区域的集中性、独立性，这是一个非常不错的思路，也是项目实战中使用比较多的方式。

## 第 5 节：地图定位技巧

在 WebGIS 的开发过程当中，地图的定位技巧有很多中，但是总得来说只有 2 种。下面我们分别来认识一下：第一种是基于中心点和缩放层级来定位，也就是依靠 `center` 参数和 `zoom` 参数来进行定位。类似的代码我们在前几章讲解框架的时候也没少见。

```
mapboxgl.accessToken = mapboxtoken;
const map = new mapboxgl.Map({
    container: "map",
    center: [120.536363, 29.243242],
    zoom: 8,
    style: "",
});

```

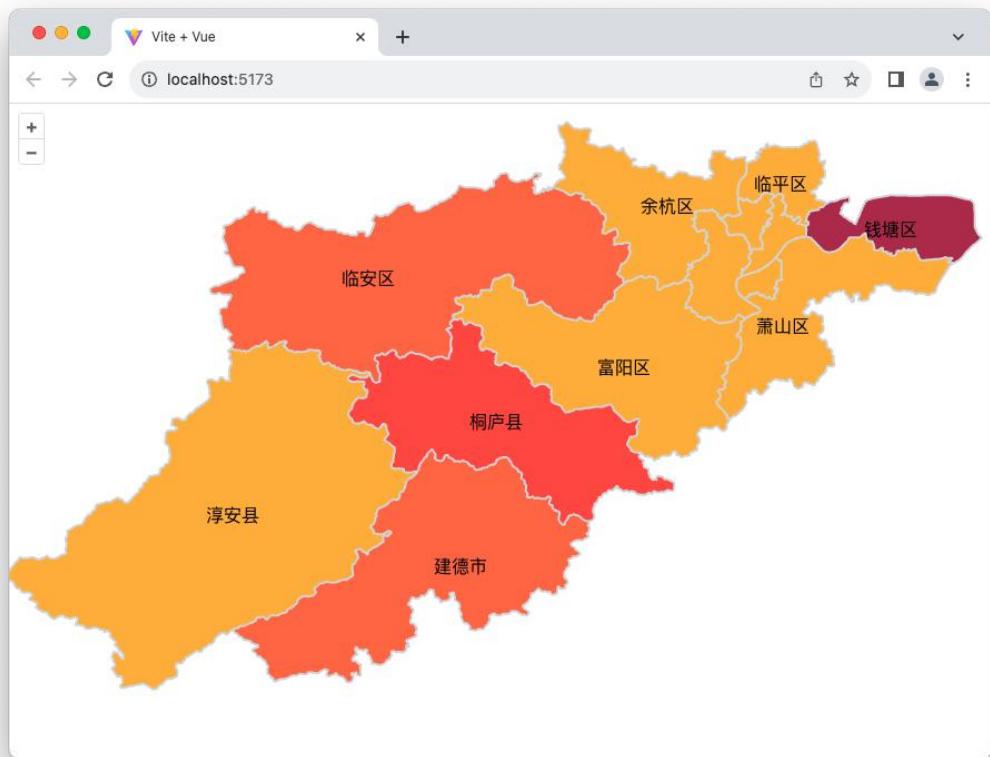
通常这种定位发生在初始化地图的时候，这种定位的好处是能够高度自定义地图的缩放大小和中心位置。大家要注意地图的缩放层级不仅仅可以是整数哦，它还可以是小数，因此这种定位方式能够精准的控制地图的缩放层级和中心点。第二种定位方式是根据数据的边界来进行定位。即当你加载了矢量或者栅格图层的时候，希望定位到该图层，这时候你大概会使用一个跟 `fit` 相关的方法来定位，在 leaflet 和 mapbox 中都叫 `fitBounds()`，而在 openlayers 中叫做 `fit`：

```
const layer = L.geoJson(hangzhou);
layer.addTo(map);
map.fitBounds(layer.getBounds());
```

这种定位的好处是能够聚焦到图层，让图层主要的内容显示在浏览器上的核心区域，但是这种方式也有缺点，那就是很难调整具体的大小，虽然它能过将图层自适应在屏幕上，但是有的时候可能会过于大或者过于小就比如说其实我们想要的是以下这样的效果：



而你调用 `fit` 方法之后页面上是这样的效果:



因此我们说这种方式虽然简便，但是还是不推荐使用，因为他不够灵活。当然你也可以采用设置缓冲变量的方式来手动调节经纬度边界范围，这样更加麻烦而且效率不高，因此关于这两种定位方式我的方案是：除非页面上图层特别多数据是动态的，经常变化的，否则还是推荐第一种定位方式。

# 第十章：GIS 软件及中间件

## 第 1 节：常用软件介绍

做 GIS 开发只会开发是不行的，很多时间你不是在写代码开发程序，而是在处理数据。所以学习一些常用的软件是非常必要的。下面给大家罗列一下几款常用的软件：

### 第一款：ArcGIS

这是 ESRI 公司开发的一套桌面端的 GIS 处理软件，功能非常的强大，能够进行栅格影像读取、重采样、图形的合并分割、坐标系的转换、空间分析等功能。它几乎涵盖了全行业 90% 的 GIS 数据处理功能，但是 ArcGIS 有个很致命的缺点就是它不支持 mac 系统。无法在苹果机器上运行。所以只限于 windows 用户使用。Arcmap 是 arcgis 系列产品中最常用的桌面软件。通常在开发之前，需要用其进行数据转换和处理工作，例如我们经常获取的数据源是国内各地方坐标系的数据，我们要将其转换成 wgs84 或者 GCGS2000 坐标系才能使用，这时候就需要 arcmap 出马了，只需要进行简单的要素投影即可完成对于要素的坐标系转换。

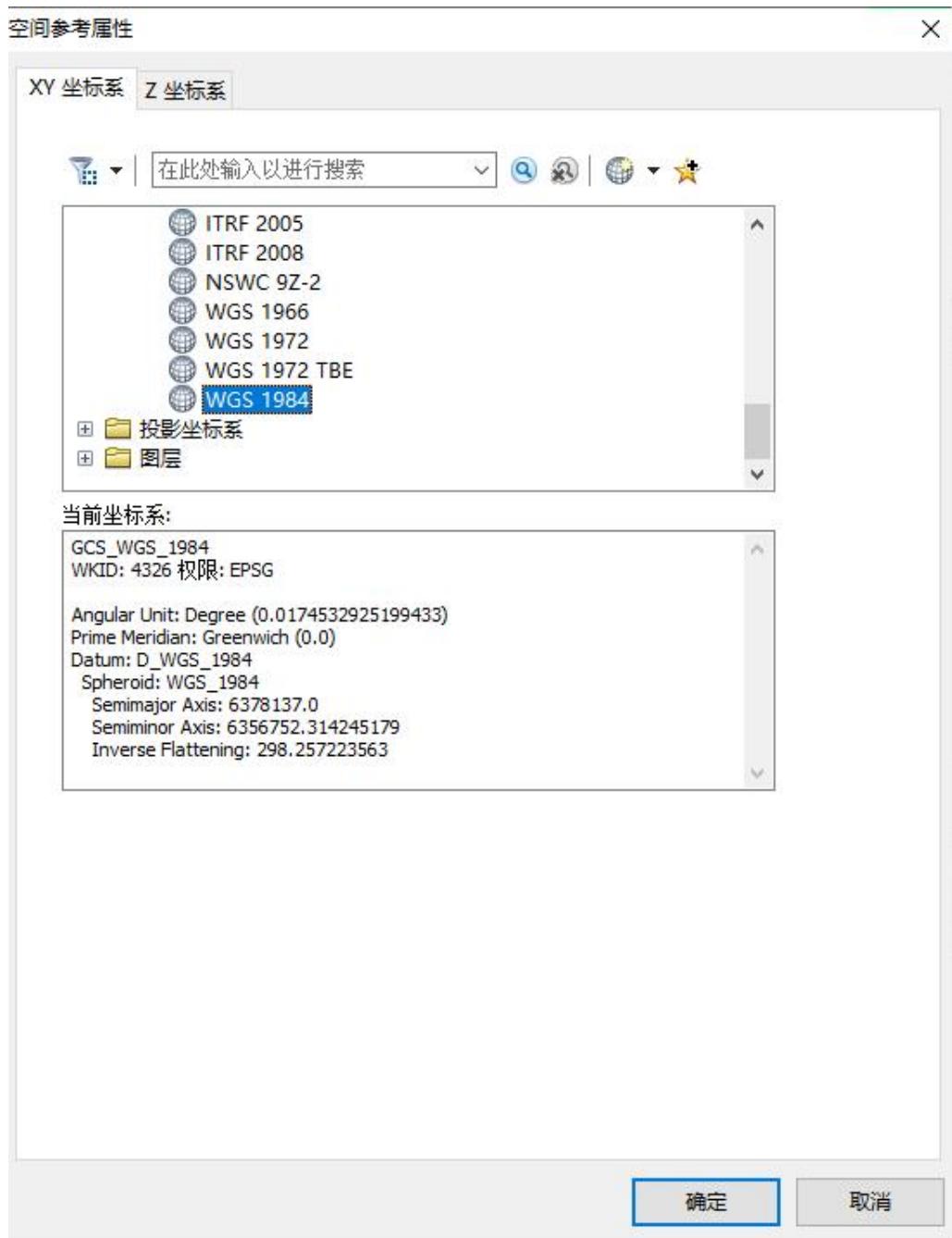
我们来演示一下如何使用 arcmap 来进行坐标系的转换，首先你的手头获取了一份地方坐标系的数据。因为是地方坐标系，因此其坐标都不是经纬度为单位的，

```
[{"type": "FeatureCollection", "features": [
  {"type": "Feature", "geometry": {"type": "Polygon", "coordinates": [[[570952.832600003, 3246218.9528], [570928.054500006, 3246228.266200004], [570892.507500003, 3246221.281099993], [570826.464500007, 3246206.771099995], [570872.3991, 3246207.522800004], [570856.570846.999099997, 3246213.8728], [570844.459099996, 3246211.54450001], [570839.378999997, 3246212.700699999, 3246226.99620001], [570815.672299997, 3246232.711200007], [570794.505599998, 3246233.98630599998, 3246229.112800002], [570770.5872, 3246225.514499996], [570771.010599997, 3246220.8577999593899997, 3246217.047800008], [570794.717299997, 3246217.047800008], [570810.380599998, 3246217.893246206.464500007], [570828.795699997, 3246191.012800006], [570833.029000001, 3246183.181099997], [3246168.575999994], [570846.364099999, 3246165.612700004], [570848.904099999, 3246159.686000007], [3246146.985999996], [570838.955699998, 3246138.731000006], [570839.590699996, 3246134.497600004], [3246122.22100001], [570855.677400004, 3246119.6809], [570859.910699999, 3246119.4693], [570865.62579570872.187400002, 3246124.126], [570878.114099999, 3246119.2576], [570894.412499996, 3246107.827600725800001, 3246101.900900007], [570907.535799997, 3246098.5142], [570908.805800002, 3246092.7992000004200003], [570906.265800002, 3246078.829199996], [570902.667500004, 3246079.252499995], [570900.602499997], [570887.427500002, 3246083.485899997], [570883.829099997, 3246080.099199997], [570883.19779200007], [570896.740799998, 3246050.042500004], [570910.287499996, 3246039.247500005], [570917.9570928.490899996, 3246019.5624], [570933.359199998, 3246020.832399997], [570940.767599999, 3246024.985899997, 3246045.8091], [570955.795900003, 3246051.1008], [570976.989099999, 3246053.711400004], [571049], [571030.038200003, 3246043.3926], [571031.890300001, 3246040.217599997], [571032.816300001, 3246076700001, 3246025.004000007], [571048.294499998, 3246022.358200006], [571054.247600004, 3246018.17]
```

这时候我们把数据丢在 arcmap 里。然后找到 arcmap 的 toolbox (工具箱) , 找到“数据管理工具” - “投影和变换” - “要素” - “投影”。



这时候将你的图层输入，然后输入坐标系软件会自动识别，你只需要指定输出坐标系，找到 world 里的 wgs84 坐标系或者是 GCGS2000 坐标系就可以将数据转为经纬度为单位的地理坐标系数据。



## 第二款：QGIS

QGIS 是基于 C++ 开发的一款可以跨平台的 GIS 处理软件，它可以同时被 Mac 系统和 windows 系统所支持。QGIS 也能够完成 arcgis 的绝大部分功能，除此之外 QGIS 还能够实现对数据的切片（包括矢量切片和栅格切片）。除此之外，QGIS 也成为了 mac 系统中跟 postgis 相连接的最常用工具。因此使用苹果系统的同学们要好好学习一下 QGIS 了。

## 第三款：ENVI

这是一款遥感影像处理的软件，可以对遥感影像进行波段读取、波段计算、NDVI 计算等多种遥感影像处理工作，如果你的行业偏重于遥感图像方面的，那这个软件必须要好好了解一下。

#### 第四款：ArcGIS Pro

ArcGIS Pro 是易智瑞公司研发的一款比 `arcgis` 系列产品更加专业。更加进阶的，融合了 AI 分析计算在内的高端 GIS 产品。如果你的行业比较专注于遥感图像处理分析，图像识别等领域，可以研究一下 ArcGIS Pro.

## 第 2 节：常用中间件介绍

光会软件也不行，因为有的时候环境不允许你使用软件，有的时候使用软件太麻烦。所以必须借助一些中间件来使用，我们这里的中间件的概念可能不同于传统前端的中间件，我们这个更像是一个小工具，或者说工具类网站。

### 1. geojson.io

这是由 mapbox 所维护的一个能够自己绘制点线面要素并导出的网站。网站地址为：  
<https://geojson.io/>，这个网站允许你制造一些类似于点位数据，地块数据，河流道路数据等，还允许你上传自己的影像图，总之你可以像在 `arcgis` 里面一样对影像图进行矢量化操作。绘制结束以后你可以直接以 `geojson` 的格式导出，这个网站通常作为我们构建 GIS 模拟数据的来源。

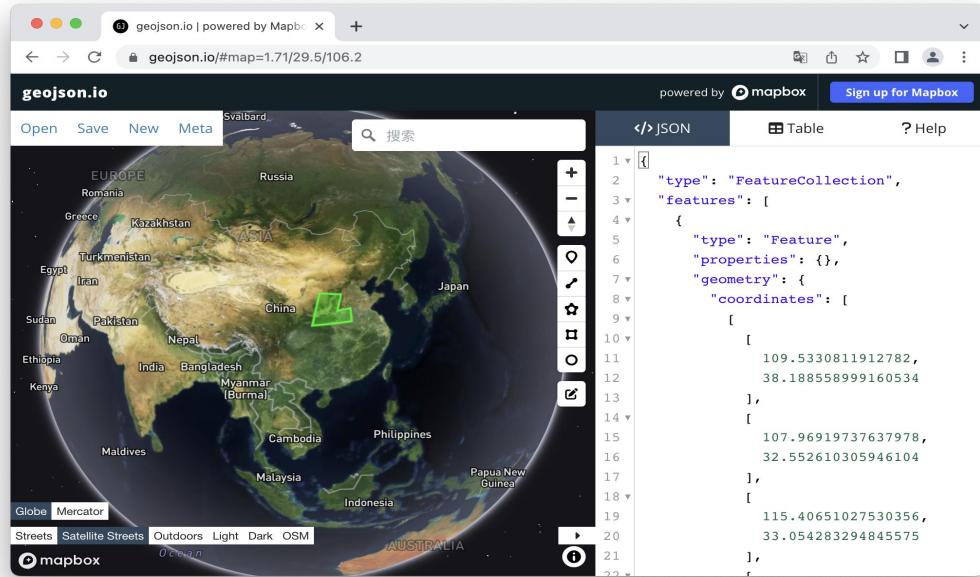


图 10.2.1-geojson.io

## 2. turf.js

turf.js 是一个前端的地理空间分析库，使用它可以完成计算要素中心点，计算边界，计算面积，判断是否相交，判断是否包含，图形的合并，图形的拆分，平移，旋转，缩放等等操作，总之你能想到的对于图形的所有操作它都能够完成。就是这么的强大。

官网地址：<https://turfjs.org/>

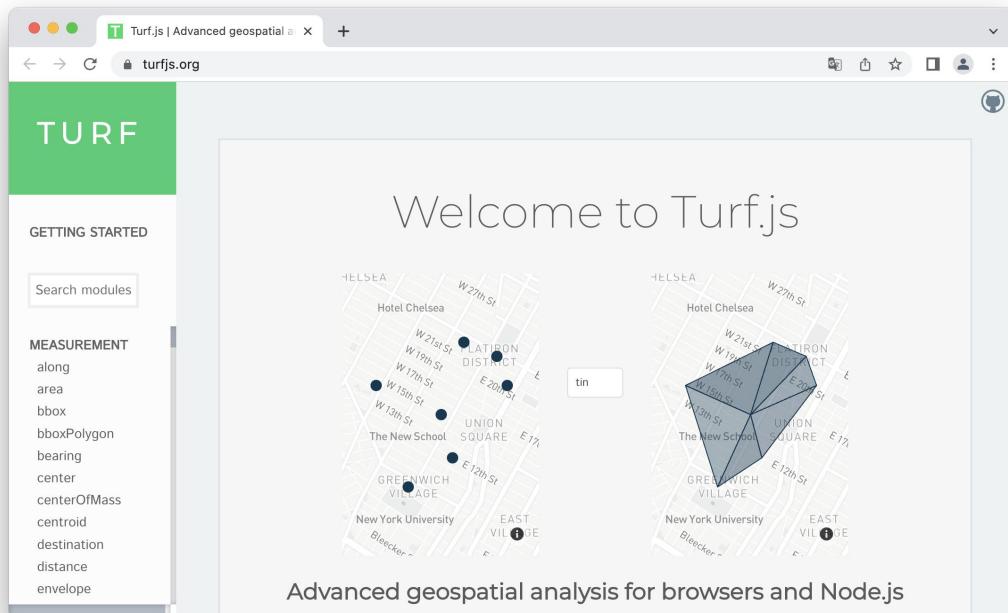


图 9.2.2-turf.js

### 3. mapshaper

mapshaper 是一个常用于数据格式转换的网站，做常用的就是 shape 数据和 geojson 数据互转。导入 shape 数据可以以 geojson 格式导出，导入 geojson 数据也可以导出成 shape 数据。官网地址：<https://mapshaper.org/>

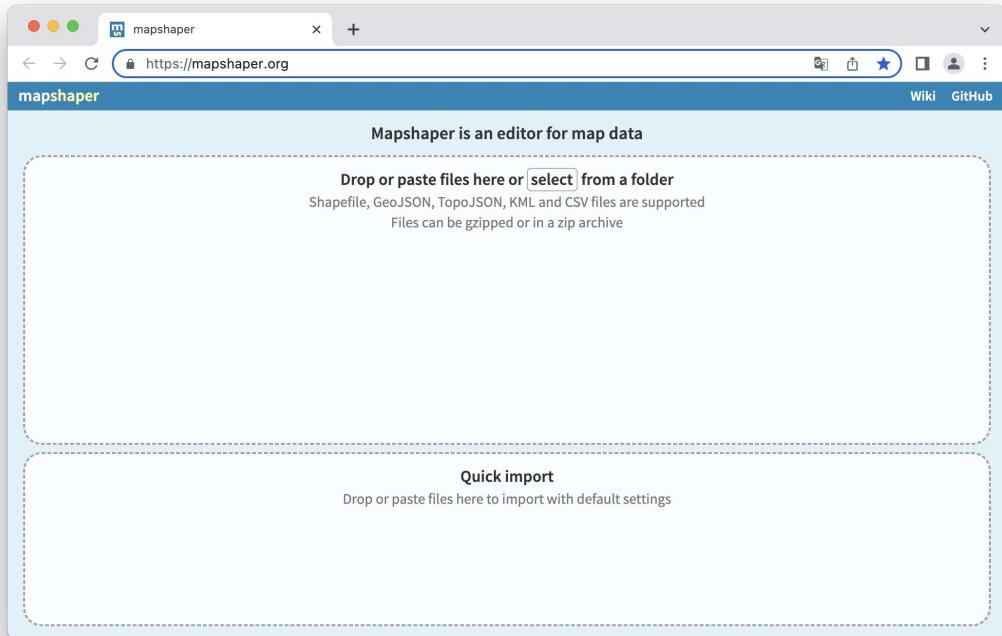


图 10.2.3-mapshaper

这些网站都非常的简单，几乎打开就可以直接使用，所以我在这里就不给各位啰嗦了。

## 第十一章：GIS 服务端构建

### 第 1 节：开源地图服务

前几章我们在进行开发的时候使用的都是静态的数据，一直也没跟大家讲这些数据平时在实际项目中都是怎么来的。实际上我们不管是栅格数据也好还是矢量数据也好，在真实的开发场景中这些数据通常都来自于服务。服务的概念我在预备知识里面讲的很详细了这里就不说了。那我们的 GIS 数据是通过什么样的服务来获取的呢？

在这里要给大家讲一个背景知识，虽然是背景但是很重要。要知道我们地理信息系统行业有一个非常重要的组织——OGC（全称 Open Geospatial Consortium 开放地理空间信息

联盟），OGC 提出了很多 GIS 行业的标准性的东西，例如从事 Cesium 三维开发所使用的 3D tiles 标准也属于 OGC 的标准。我们本书讲过的所有框架都支持 OGC 的服务标准，并且各位在日后的开发过程中也要遵循 OGC 的标准和规范。

其中经常用到的大致有三个分别是：

**WMS 服务、WFS 服务、WMST 服务。**首先对于栅格数据有两种服务需要大家掌握，即 WMS 服务和 WMST 服务。

WMS 服务全称为 Web Map Service 即网络地图服务。核心就是提供一副网络地图。如何提供？实际上就是给你一张图片，这张图片的内容是一副地图，我们之前说过栅格数据最终渲染到浏览器上都被转换成了图片，那 wms 就是将地图提前处理成图片格式然后发布出来以供浏览器调用。所以 wms 的核心就是一句话：为你提供一张图片格式的地图。

栅格数据的第二种服务为 WMST 服务，全称：Web Map Tile Service，即网络地图切片服务。与 WMS 不同的是 WMST 增加了一个新的名词，切片。什么意思？原因来自于一个非常实际的问题，就是浏览器能够请求的数据量大小是非常有限制的，一般来说网络请求的数据体量都是 kb 级别，几 k 几十 k 大小的数据请求起来还行，换成几 M 或者几十 M 的话很容易就发生阻塞了。所以如果一张地图图片的尺寸非常大的话一次性请求回来肯定是行不通的。因此需要把地图分割成很多份，每一份的大小控制成固定的，然后按照顺序和需求去请求对应的图片，不需要的就不请求，这样以来就大大的减少了浏览器资源消耗。提高了性能。这就是 WMST 诞生的原因。

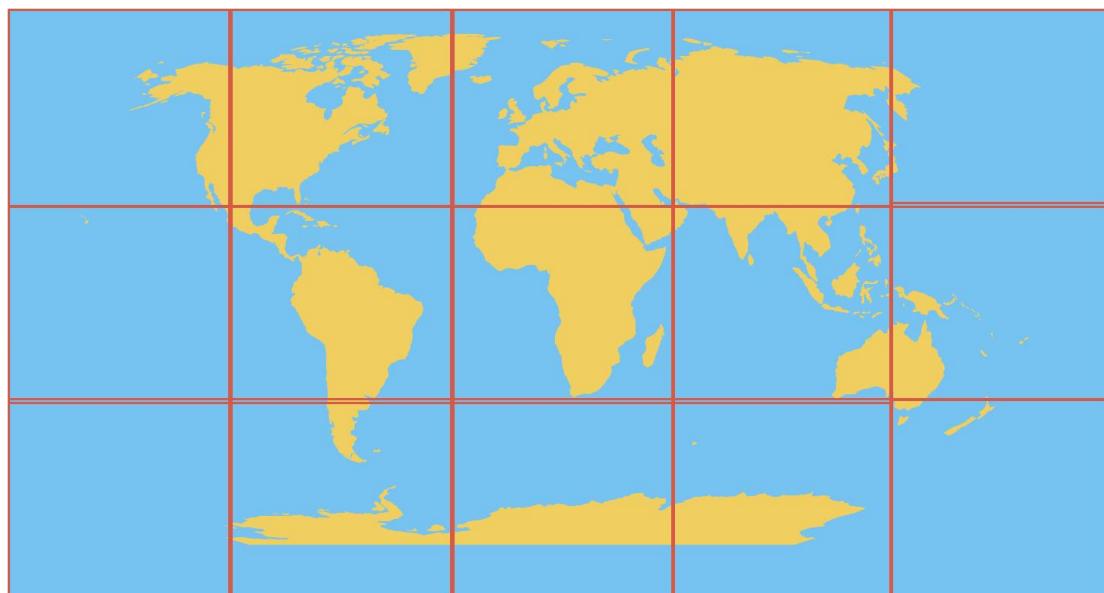


图 11.1.1-地图瓦片

WMST 就是把地图分成许多等份，每一份通常是 256x256 像素大小的图片。用户在浏

览器端查看地图的时候，根据用户当前所处的缩放层级，来加载对应的该层级之下的切片。

在这里解决一下之前在学习 leaflet 加载天地图所产生的一些疑问。wmts 对于地图的切片是这样的，首先以一整张图片的左上角为切片原点，然后按照 256 像素\*256 像素对图片进行切割，切割完之后再对图片进行编号，编号的规则是这样的：首先要确定当前的层级（也就是当前的比例尺）我们简称为 y，然后在按照行和列的概念对每一副小地图进行编号，就像下面这样，

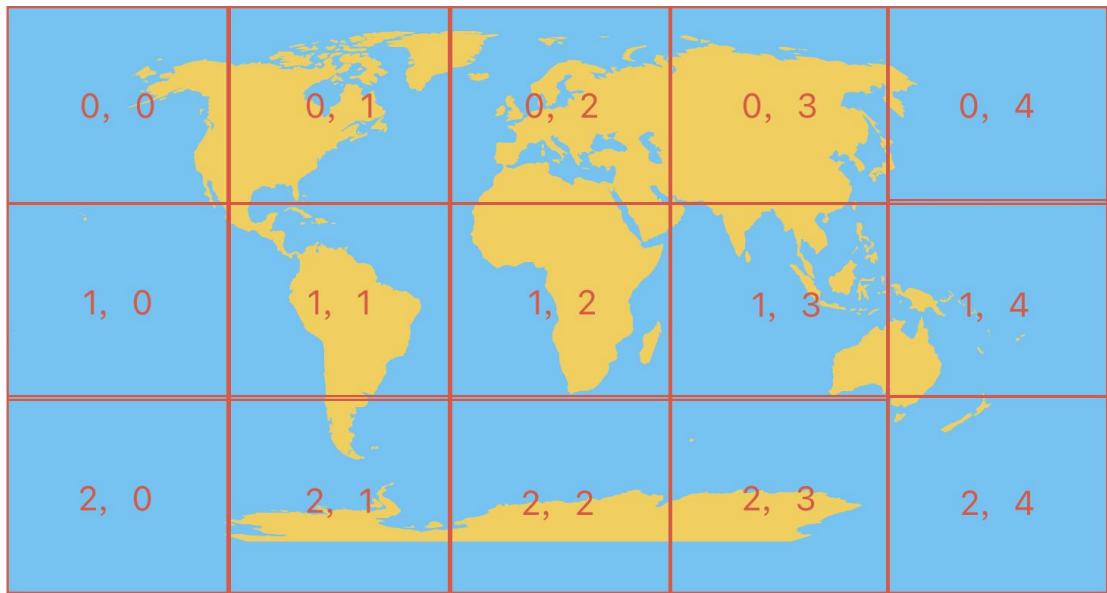


图 11.1.2-切片原理

行号我们记作 x，列号记作 y。因此根据三个数字 xyz 就能够获取唯一一张图片，这也是在 wmts 的 url 里出现 zxy 的原因。另外根据这个原理，其实我们完全可以在服务器上自定义一个新的规则来获取瓦片那就是按照文件夹来划分 z 和 y，然后每个文件夹里面按照 x 的编号来放置图片。这就是民间较为流行但并没有成为官方标准的 xyz 方式，实际上这也是所谓的 TMS 的规范和标准。

很多同学读到这里还是不太清楚 WMS 服务和 WMTS 的具体的区别在哪里，我们来详细描述一下这个过程。一个标准的 WMS 的过程是客户端根据浏览器当前的缩放层级和分辨率计算出需要请求的地图范围，这个范围即请求参数中的 width 和 height，然后发送请求到服务端，服务端根据请求参数里的宽和高从地图数据中裁剪出应有的范围，然后将这个范围内的部分以图片 png/jpeg 的格式返回到前端，也就是每一次鼠标的拖拽以及地图的缩放，只请求回来一张图片。而 WMTS 则是根据当前的分辨率（缩放层级）自动计算出当前层级下的瓦片行列号，讲这些瓦片全都返回，注意是全部，尽管你在屏幕上只看到了几十张瓦片，但是它确确实实是按照层级一次行全部返回的。也就是说 WMTS 随着地图的缩放请求一次

要返回很多张图片。但是仅拖拽不缩放是不会发起请求的。

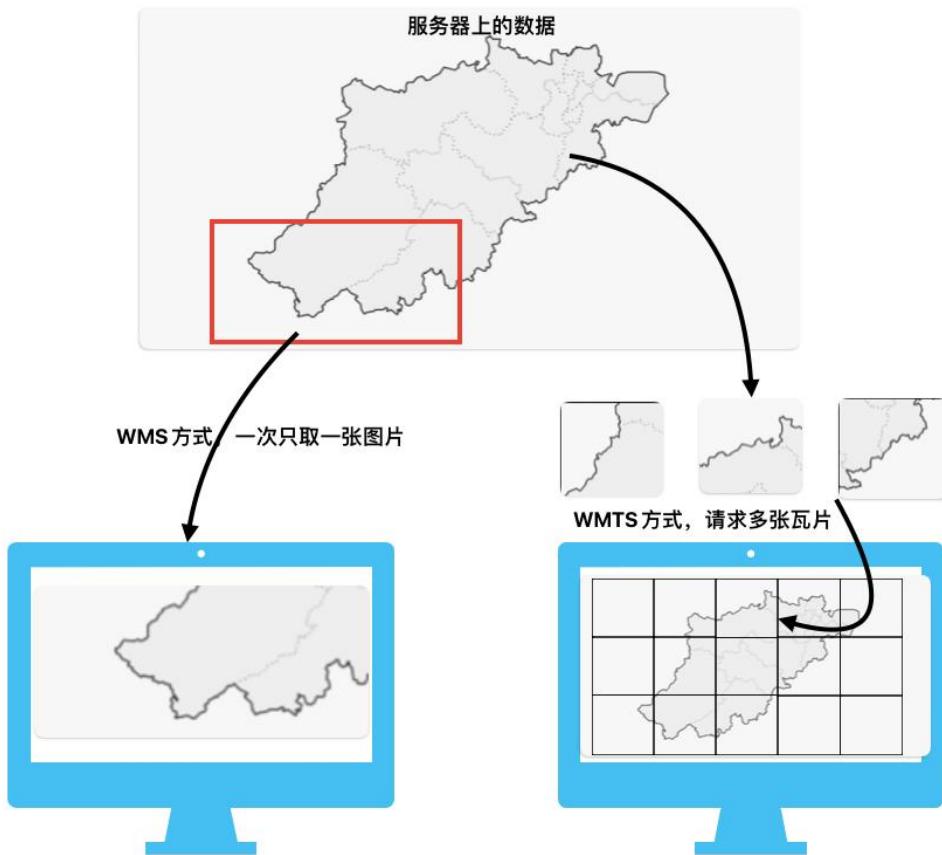


图 11.1.3-WMS 与 WMTS 对比

对于矢量数据类型也有两种地图服务分别是 WFS 服务和矢量切片服务。WFS 全称 Web Feature Service，即网络要素服务。如何理解这个要素，还记得我们在 GIS 数据结构基础这一章跟大家讲过 **feature** 的概念吗，一个点，一条线，一个面都可以被称作是一个要素，那么 WFS 服务其实就是返回一些要素数据。再进一步讲，WFS 服务通常返回的就是 GeoJSON 数据。如果说栅格数据的两种服务返回的是图片数据，WFS 服务返回的就是要素数据。矢量切片 (Vector Tile) 诞生的原因其实和 WMTS 诞生的原因一样，都是大量的数据会造成浏览器性能的卡顿，矢量切片其实就是对矢量数据进行切片，切片的原理和栅格切片类似，也是按照其数据范围进行切片，只不过不需要像栅格那样去裁切图像，而是将数据组织进行变换，按照层级和范围去划分数据，另外矢量切片的结果通常是需要进行数据压缩的。mapbox 提供了一种矢量切片的标准叫做 MVT。谷歌提出了一种矢量切片数据的压缩格式.pbf。

最后一种服务既可以访问栅格数据也可以访问矢量数据，即 TMS 服务。全称 Tile Map Service，瓦片地图服务。对于栅格数据来讲，和 WMTS 类似，但是切片的起始原点和编号

都与 WMTS 不同。对于矢量数据来讲，TMS 也是遵循了大众的习惯直接以 xyz 的方式去访问矢量切片的结果 pbf。因此 TMS 比较全面。

到此为止已经给大家介绍完了所有的开源服务类型，其中 WMS，WMTS，WFS 服务是大家必须要掌握的，也是开发过程中经常会遇到的。

## 第 2 节：GeoServer 学习

这里要给大家介绍一个非常重要的东西，你可以称它为一个软件，也可以称它为一个中间件，甚至你可以称它为一个网站，但是它最准确的称呼应该是 **GIS 服务器**——GeoServer。GeoServer 的出现是为了避免我们手动的频繁的自己写数据服务，并且 GeoServer 实现了多种常用的矢量和栅格数据的处理办法。我们只需下载和部署就可以直接使用它来发布数据服务。在这个服务器上发布的数据会生成相应的访问地址，然后客户端会根据这些地址请求到对应的数据。这就解决了我们在前几章学知识的过程中所遇到的数据来源的问题。在之前的章节。我跟大家说业务中遇到的数据类型大多数时候是 geojson 格式的。可是普通的后端开发人员是不认识 geojson 这种格式的，每次开发我们都自己去写服务端又显得有些麻烦，这个时候我们就可以考虑 geoserver。geoserver 在这种场景下就类似与 nginx 和 tomcat 一样，扮演着一个数据发布服务器的角色。

geoserver 首先需要下载安装，官网：<https://geoserver.org/>。

使用 geoserver 之前请先确保你的电脑有 **java 环境**。如何安装 java 环境大家自行百度吧，因为这种教程网上太多了，放在我的书里简直是占用空间。同时在这里建议大家下载 geoserver 的时候选择最新的稳定版本，比如笔者在撰写这本书的时候下载的就是 2.22.3 版本，各位在开发的时候尽量高于或者与此版本保持一致。

找到官网首页点击 **stable** 下面的版本号然后跳转之后按照下面的方式下载：

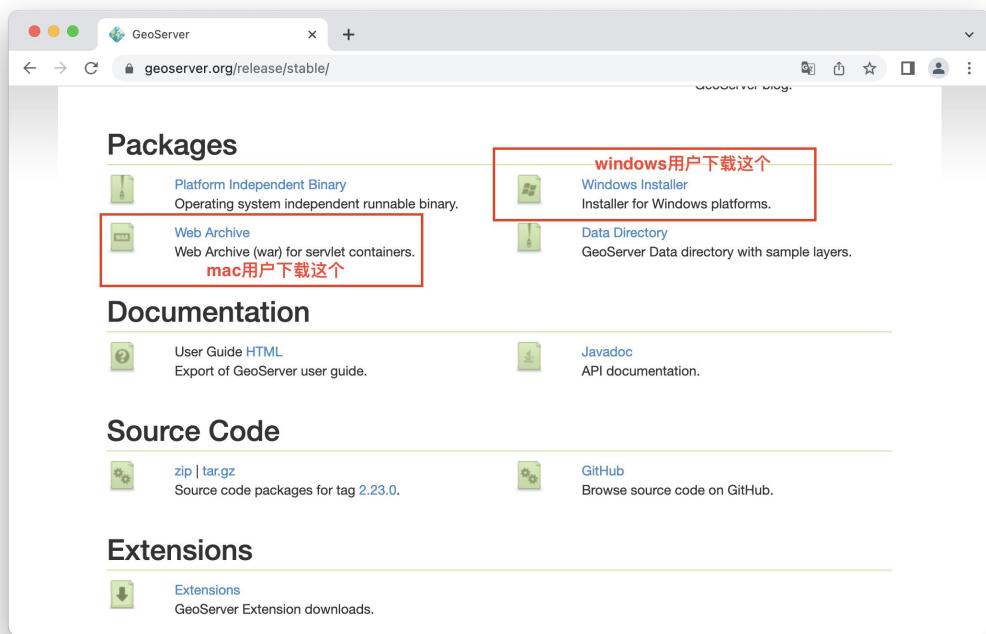


图 11.2.1-下载 geoserver

按照各自对应的操作系统去下载就可以了,当然 windows 用户也可以下载 web archive 版本, 因为是 war 包部署的, 只要由 tomcat 无所谓什么操作系统。

下面讲讲两种下载的安装方式, 对于 windows installer 很简单, 一路根据提示无脑默认安装就行了。对于 web Archive 方式的, 你必须先在自己的机器上安装一个 tomcat (tomcat 的下载和安装也异常的简单哦) 然后把这个下载下来的文件扔到 tomcat 的 webapps 文件夹里面就 OK 了。

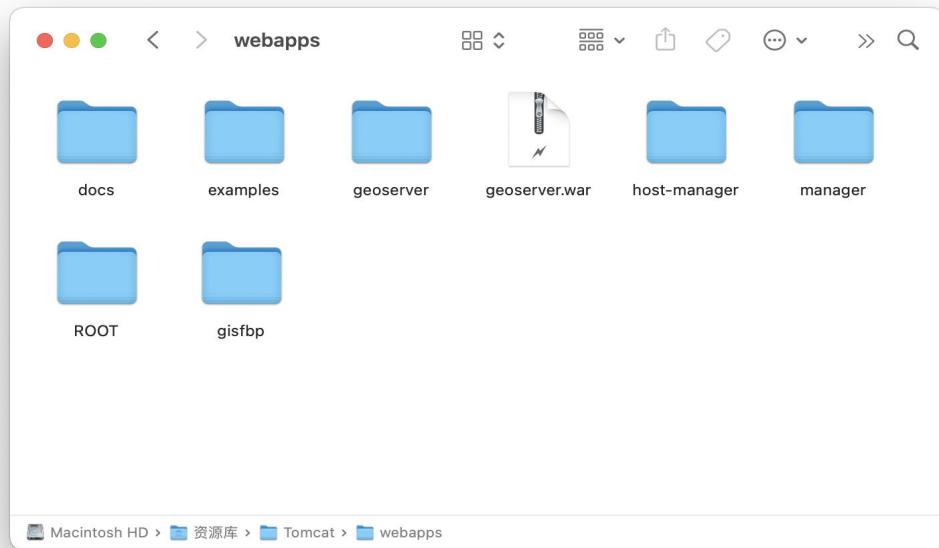


图 11.2.2-war 包部署 geoserver

如何启动 GeoServer? 是个好问题。不管你是以哪种方式部署的。启动 GeoServer 的关键都是找到 `startup.bat` 或者 `startup.sh` 文件。如果你是 windows 系统, 你可以直接找到安装目录下面的 `bin` 文件夹然后双击 `startup.bat`, 等待一会就启动成功了。

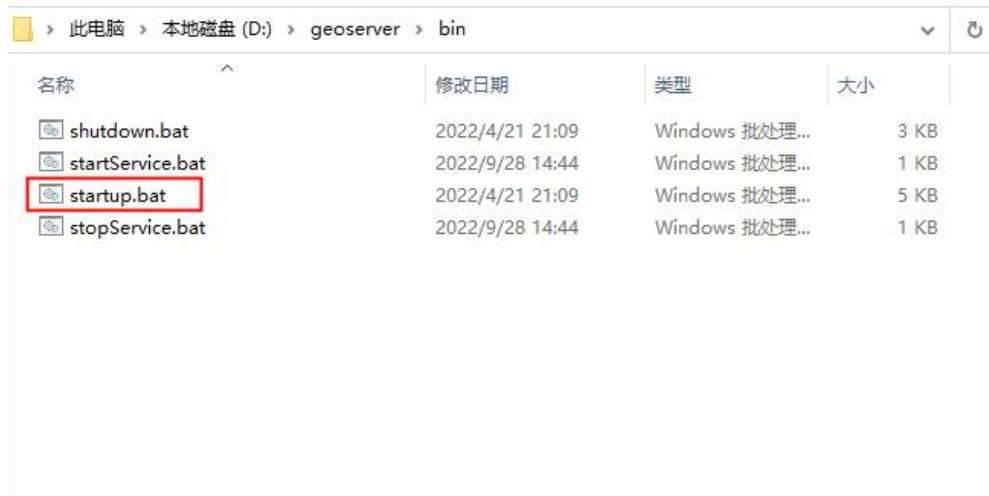


图 11.2.3-启动 geoserver

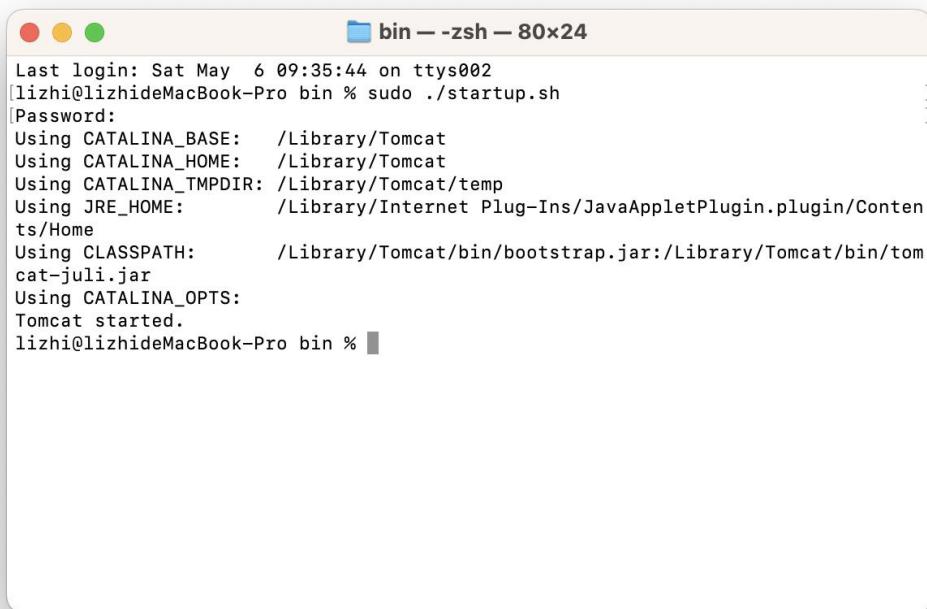
如果你是 mac 系统, 你需要在 `bin` 文件夹的位置打开你的终端, 然后输入:

```
sudo ./startup.sh
```



A screenshot of a Mac OS X terminal window titled "bin — -zsh — 80x24". The window shows the standard OS X window controls (red, yellow, green) at the top left. The title bar is centered. The main area of the terminal is completely blank, indicating no input or output has been displayed.

然后输入开机密码之后按下回车，看到以下内容也证明启动成功了。



A screenshot of a Mac OS X terminal window titled "bin — -zsh — 80x24". The window shows the standard OS X window controls (red, yellow, green) at the top left. The title bar is centered. The terminal displays the following text:

```
Last login: Sat May 6 09:35:44 on ttys002
[lizhi@lizhideMacBook-Pro bin % sudo ./startup.sh
>Password:
Using CATALINA_BASE: /Library/Tomcat
Using CATALINA_HOME: /Library/Tomcat
Using CATALINA_TMPDIR: /Library/Tomcat/temp
Using JRE_HOME: /Library/Internet Plug-Ins/JavaAppletPlugin.plugin/Contents/Home
Using CLASSPATH: /Library/Tomcat/bin/bootstrap.jar:/Library/Tomcat/bin/tomcat-juli.jar
Using CATALINA_OPTS:
Tomcat started.
lizhi@lizhideMacBook-Pro bin % ]
```

然后在浏览器地址栏上输入 `localhost:8080/geoserver` 即可成功访问 geoserver。

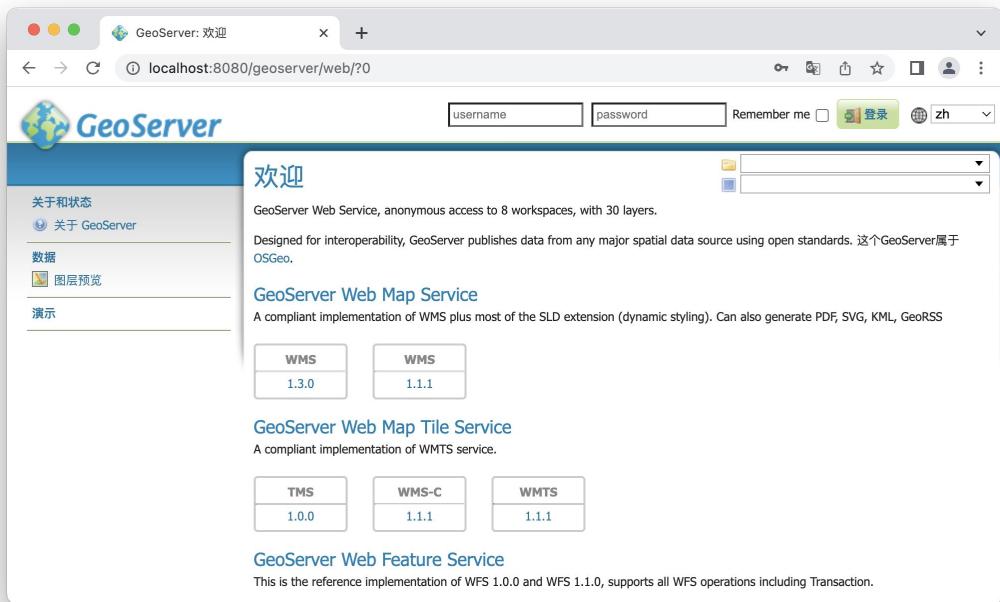


图 11.2.4-访问 geoserver

geoserver 的默认登录名为 admin，密码为：geoserver。登录以后就可以开始正常的操作了。

下面我们以一份杭州市的行政区数据为例，给大家介绍一下 geoserver 的用法。

我们首先来说说数据的发布，geoserver 支持发布三种类型的数据，geotiff（就是我们的 tiff 影像），静态矢量数据，以及链接数据库进行数据的发布。我们登录以后找到左侧的“存储仓库”（也可能叫数据存储之类的，总之是和存储相关）然后选择“添加新的存储仓库”，在弹出的页面中你可以看到我上面说的三种类型的数据发布方式。

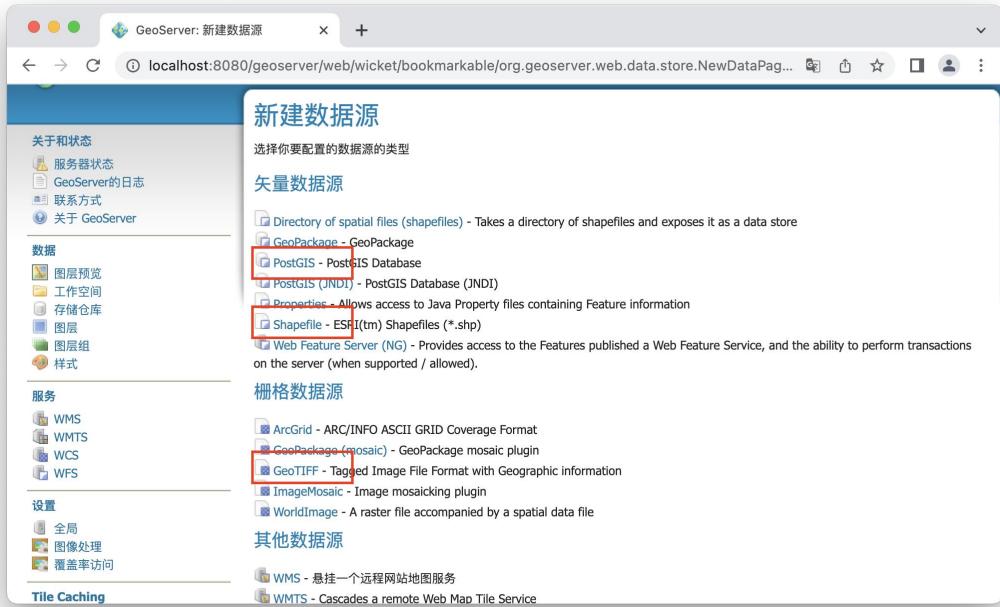


图 11.2.5-geoserve 发布数据

注意这里的三种方式可不仅仅是只有这三种，只不过这三种是我们最常用的。我们把我们准备好的杭州市的 shape 数据进行发布，大家手头如果有 shape 数据就用自己的数据，没有数据的直接 arcgis 里或者 QGIS 里画一个。我们选择上面图中的 shapefile 选项，然后通过本地路径找到我们的 shape 数据，

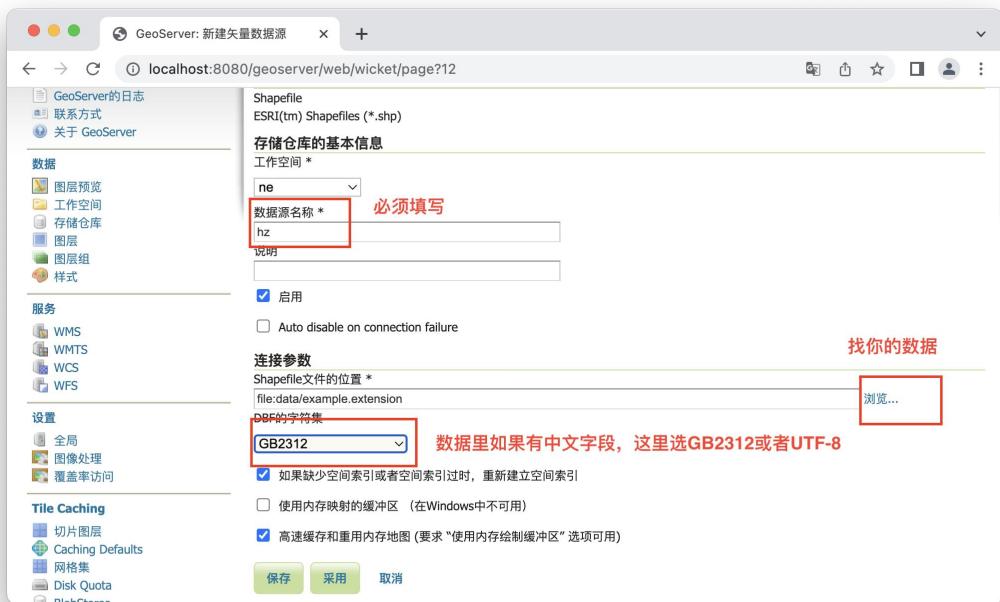


图 11.2.6-发布矢量数据

然后点击保存，新页面中点击发布即可进入到发布数据的页面，这里需要填写几项配置，

首先是数据的经纬度范围, 这是必填的, 不过通常 geoserver 会帮助我们自动计算这个范围, 填写完成之后点击下方的保存即可成功的发布数据。

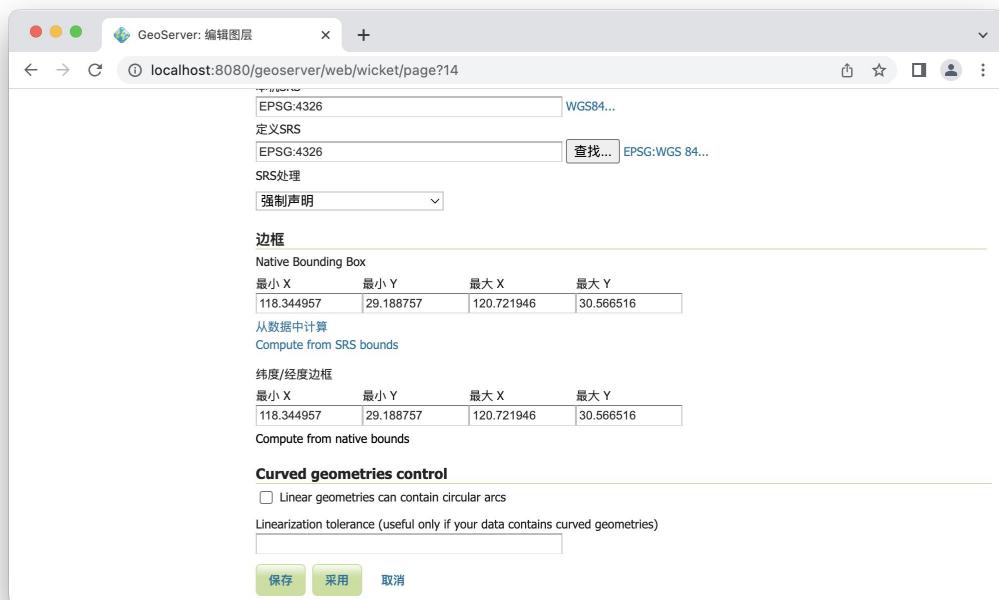


图 11.2.6-确定数据的边界

如果发布的 tiff 格式的影像数据。流程几乎相似。而且更加简单。在选择数据源的时候选择 Geotiff 即可, 然后计算边框数据, 最后保存将数据发布。

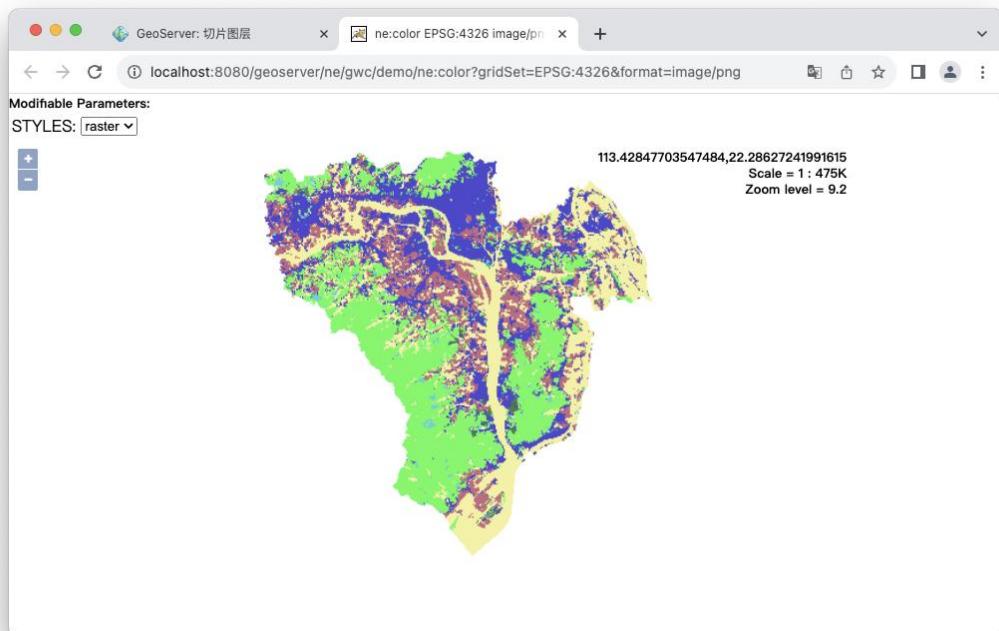


图 11.2.7-发布栅格影像数据

数据发布完成之后点击左侧的菜单栏当中的“图层预览”，在众多图层中找到你刚才发

布的那个图层，点击 **openlayers** 即可以 wms 的方式查看你的地图。

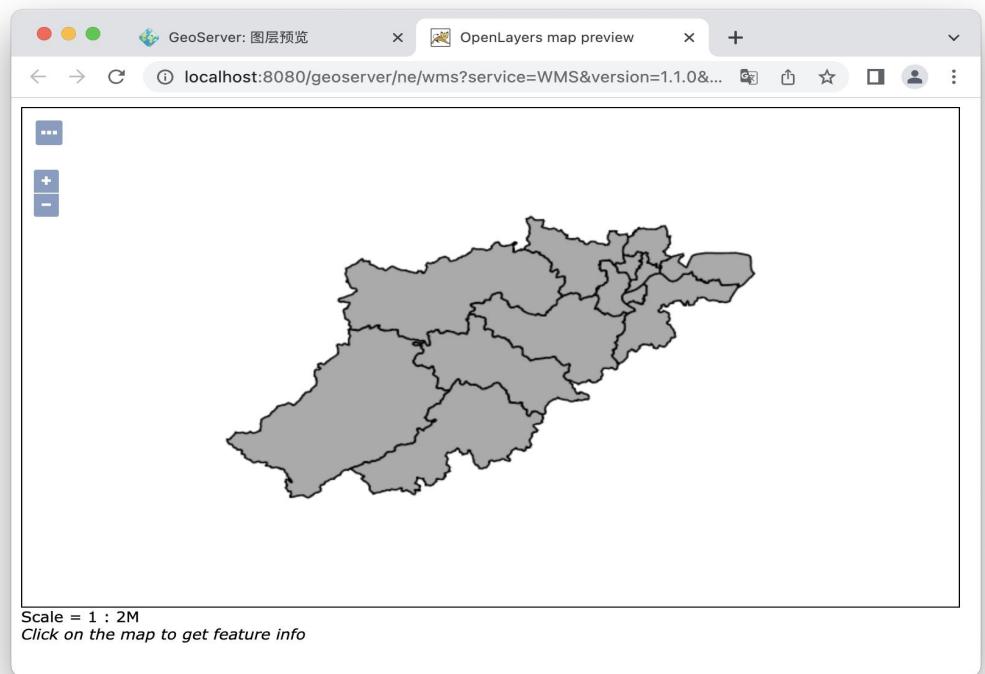


图 11.2.8-以 wms 方式访问地图

此页面的浏览器地址栏里的地址就是 WMS 服务的访问方式。如果想在浏览器页面展示 wms 结果，直接在框架（mapbox、leaflet、openlayers）内部调取这个 url 就可以了。

在调用之前需要说明一个小细节，常规情况下是需要打开 geoserver 的跨域限制的。geoserver 默认是阻止跨域请求的，因此很多时候你在代码里写了正确的请求，而没有出现效果，打开浏览器控制台就会发现是 geoserver 阻止了请求。打开的方式也很简单，找到 geoserver 安装在的目录，找到“WEB-INF”文件夹，再找到 web.xml 文件，打开如下的两段注释即可（原本这两段内容是被注释起来的。表示不生效，即不打开跨域限制，我们把注释去掉之后就表示打开跨域），具体的注释内容可能跟版本有关系，大家只需关心 cross-origin 关键字即可，参照对应版本的 xml 文件内容进行修改。

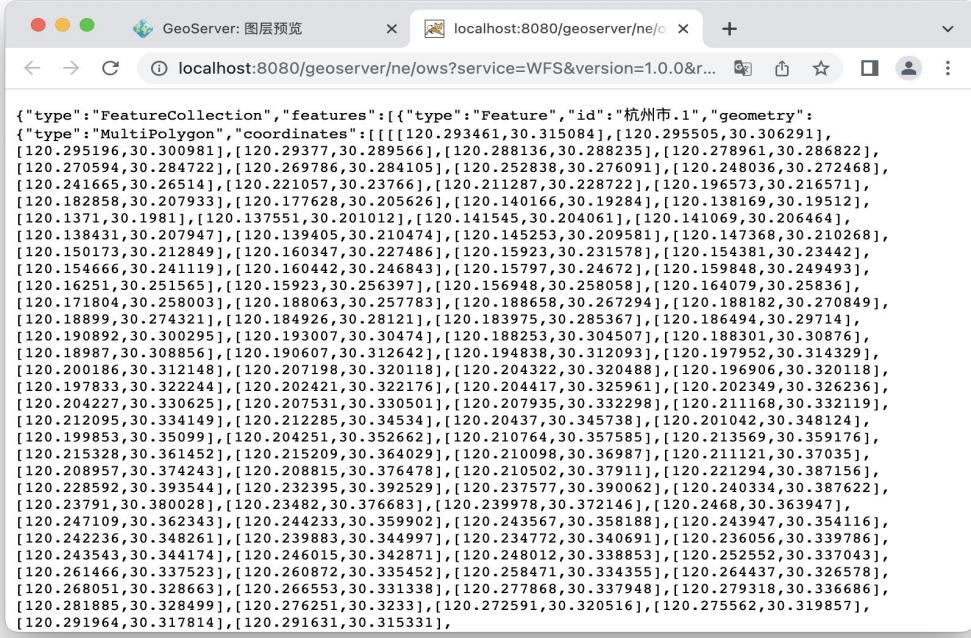
```

<filter>
  <filter-name>cross-origin</filter-name>
  <filter-class>org.apache.catalina.filters.CorsFilter</filter-
class>
  <init-param>
    <param-name>cors.allowed.origins</param-name>

```

```
<param-value>*</param-value>
</init-param>
<init-param>
<param-name>cors.allowed.methods</param-name>
<param-value>GET,POST,PUT,DELETE,HEAD,OPTIONS</param-value>
</init-param>
<init-param>
<param-name>cors.allowed.headers</param-name>
<param-value>*</param-value>
</init-param>
</filter>
<-----第二段----->
<filter-mapping>
<filter-name>cross-origin</filter-name>
<url-pattern>/*</url-pattern>
</filter-mapping>
```

如果你想以 WFS 的方式访问这个图层，可以回到刚才的页面，在 openlayers 旁边有一个 select one，然后点击 wfs 底下的 geojson 即可查看该图层的 WFS 服务。



```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "id": "杭州市.1",
      "geometry": {
        "type": "MultiPolygon",
        "coordinates": [
          [
            [
              [
                [
                  [120.295196, 30.300981], [120.293461, 30.315084], [120.295505, 30.306291], [120.29377, 30.289566], [120.288136, 30.288235], [120.278961, 30.286822], [120.270594, 30.284722], [120.269786, 30.284105], [120.252838, 30.276091], [120.248036, 30.272468], [120.241665, 30.265141], [120.221057, 30.237661], [120.211287, 30.228722], [120.196573, 30.216571], [120.182858, 30.207933], [120.177628, 30.205626], [120.140166, 30.192841], [120.138169, 30.195121], [120.137130, 30.1981], [120.137551, 30.201012], [120.141545, 30.204061], [120.141069, 30.206464], [120.138431, 30.207947], [120.139405, 30.210474], [120.145253, 30.209581], [120.147368, 30.210268], [120.150173, 30.212849], [120.160347, 30.227486], [120.15923, 30.231578], [120.154381, 30.234421], [120.154666, 30.241119], [120.160442, 30.246843], [120.15797, 30.24672], [120.159848, 30.249493], [120.16251, 30.251565], [120.15923, 30.256397], [120.156948, 30.258058], [120.164079, 30.25836], [120.171804, 30.258003], [120.188063, 30.257783], [120.188658, 30.267294], [120.188182, 30.270849], [120.18899, 30.274321], [120.184926, 30.28121], [120.183975, 30.285367], [120.186494, 30.29714], [120.190892, 30.300295], [120.193007, 30.30474], [120.188253, 30.304507], [120.188301, 30.308761], [120.18987, 30.308856], [120.190607, 30.312642], [120.194838, 30.312093], [120.197952, 30.314329], [120.200186, 30.312148], [120.207198, 30.320118], [120.204322, 30.320488], [120.196906, 30.320118], [120.197833, 30.322244], [120.202421, 30.322176], [120.204417, 30.325961], [120.202349, 30.326236], [120.204227, 30.330625], [120.207531, 30.330501], [120.207935, 30.332298], [120.211168, 30.332119], [120.212095, 30.334149], [120.212285, 30.34534], [120.20437, 30.345738], [120.201042, 30.348124], [120.199853, 30.35099], [120.204251, 30.352662], [120.210764, 30.357585], [120.213569, 30.359176], [120.215328, 30.361452], [120.215209, 30.364029], [120.210098, 30.36987], [120.211121, 30.37035], [120.208957, 30.374243], [120.208815, 30.376478], [120.210502, 30.37911], [120.221294, 30.387156], [120.228592, 30.393544], [120.232395, 30.392529], [120.237577, 30.390062], [120.240334, 30.387622], [120.23791, 30.380028], [120.23482, 30.376683], [120.239978, 30.372146], [120.2468, 30.363947], [120.247109, 30.362343], [120.244233, 30.359902], [120.243567, 30.358188], [120.243947, 30.354116], [120.242236, 30.348261], [120.239883, 30.344997], [120.234772, 30.340691], [120.236056, 30.339786], [120.243543, 30.344174], [120.246015, 30.342871], [120.248012, 30.338853], [120.252552, 30.337043], [120.261466, 30.337523], [120.260872, 30.335452], [120.258471, 30.334355], [120.264437, 30.326578], [120.268051, 30.328663], [120.266553, 30.331338], [120.277868, 30.337948], [120.279318, 30.336686], [120.281885, 30.328499], [120.276251, 30.3233], [120.272591, 30.320516], [120.275562, 30.319857], [120.291964, 30.317814], [120.291631, 30.315331]
      ]
    }
  ]
}
```

图 11.2.9-以 WFS 方式访问数据

同样的，如果你想以 WFS 的方式访问数据，只需请求地址栏里的 url 即可。

如果你想以 WMTS 的方式查看地图，需要点击左侧菜单栏里的切片图层选项，然后找到我们的图层，点击“select one”选项随便点击一个，这样就是以 WMTS 的方式去访问我们的图层。

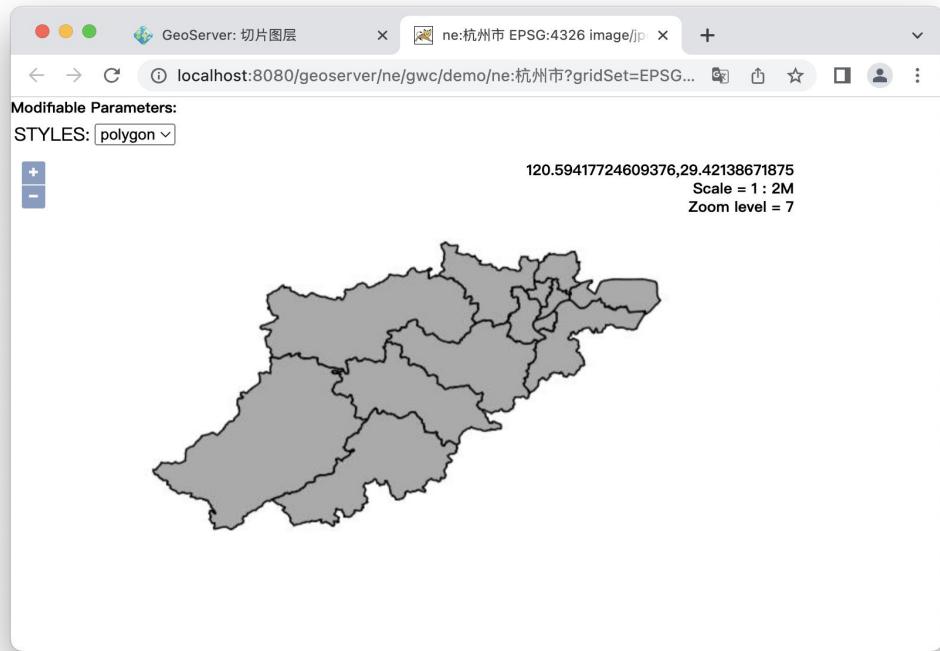


图 11.2.10-以 wmts 方式访问地图

这三种方式地址栏里的地址通常就是服务的请求地址, 只不过需要对这些地址里的参数进行一些修改。具体的参数也要根据前端使用的 GIS 框架的不同来决定。

一个 wms 服务的请求参数通常如下:

```
http://localhost:8080/geoserver/ne/wms?service=WMS&version=1.1.0&request=GetMap&layers=ne%3Apoint&bbox=102.87412663444218%2C25.159433245949472%2C116.00213373311556%2C41.643412502187545&width=611&height=768&srs=EPSG%3A4326&styles=&format=image/png
```

其中 `http://localhost:8080/geoserver/ne/wms?` 为请求的地址, 一般为服务器域名加端口号再加 `/geoserver/ne/wms?`, 如果 geoserver 在本地部署, 则 ip 为 `localhost` 即可。`service=WMS` 代表服务类型为 wms 服务。`version=1.1.0` 代表使用的是 1.1.0 版本的 wms 服务。`request=GetMap` 代表该服务使用的请求方法为 GetMap。`layers=ne%3Apoint` 代表请求的图层为 ne:point。

`bbox=102.87412663444218%2C25.159433245949472%2C116.00213373311556%2C41.643412502187545` 代表该图层的经纬度范围。`srs=EPSG%3A4326` 代表该图层的坐标系为 wgs84 坐标系。剩下的参数都不确定, 可改动。不同框架调用 wms 的时候所填写的参数也不同。另外 wms 可以进行样式的配置和操作, geoserver 提供了 sld 样式配置

的方案，只需按照规则就可以对图层进行各种各样的颜色，文字等样式的配置。

WFS 的本质是返回矢量数据。这对应我们在第二章第三节给大家讲过的内容。在 WebGIS 中，没有办法直接返回 shape 数据，所以会将 shape 转换为更简单轻量的 geojson 数据。WFS 服务就封装了这样一种需求。如果说 WMS 服务返回的是栅格图片的话，WFS 返回的就是多种格式的矢量数据，除了 geojson 以外还可以返回 kml, gml, wkt, svg 等格式。

一个标准的 wfs 服务请求格式如下：

```
http://localhost:8080/geoserver/ne/ows?service=WFS&version=1.0.0&request=GetFeature&typeName=ne%3Apoint&maxFeatures=50&outputFormat=json
```

其中 “<http://localhost:8080/geoserver/ne/ows?>” 为基础地址 service 和 version 都和上文一样，请求方法呢为 GetFeature 注意 wfs 中的图层名字的写法是：typeName=ne:point 后面的 maxFeature=50 可去掉，这表示限制要素返回的数量为 50 个。去除后将全量返回。最后指定格式为 json 格式。返回的数据的格式为 geojson 格式。

一个 WMTS 的请求格式为：

```
http://localhost:8080/geoserver/ne/gwc/service/wmts?layer=ne%3Apoint&style&tilematrixset=EPSG%3A900913&Service=WMTS&Request=GetTile&Version=1.0.0&Format=image%2Fpng&TileMatrix=EPSG%3A900913%3A{z}&TileCol={x}&TileRow={y}
```

其中 <http://localhost:8080/geoserver/ne/gwc/service/wmts?> 是基础地址。后面的一些参数，需要注意的是 TileMatrix=EPSG:900913 代表的是图层的坐标系为 3857 坐标系，也就是墨卡托坐标系。后面的 {z}&TileCol={x}&TileRow={y} 表示这个 wmts 是以 xyz 的方式请求的。z 代表当前所处的层级，x 代表图片是第几行，y 代表图片是第几列。

其实我们还有很多方式去访问 geoserver 发布的服务，并且每个服务里都有很多的细节，但是本书目的是教给大家快速开发的教程，这些知识也足够大家应付项目需求了。如果还想学习深入的进阶的知识大家可以报名我的直播课。直播课中会跟大家讲解一些深入的技巧性的东西

## 第 3 节：构建自己的服务端

有些场景不允许使用开源服务器 `geoserver`，或者说 `geoserver` 的承载能力是有限的，碰到大数据的场景或者数据类型丰富且常变的场景 `geoserver` 就显得很鸡肋，这时候就需要我们自己动手搭建 `GIS` 服务端。`GIS` 的服务端搭建有很多种方式，而且很多语言也支持搭建 `GIS` 服务端。实际上我们的 `GIS` 服务端还是使用常规的服务端搭建技术，只不过数据管理，分析和传输层面专注于 `GIS` 方面。换句话说我们可以使用任何服务端语言，只需要针对与不同的场景即可。

例如，你的项目对于 `GIS` 服务端的需求比较简单，比如说行政区服务，你希望通过某些接口按照行政区名称和代码获得相关的行政区，那么你可以用 `node.js`（一款 `js` 语言为基础的服务端框架）构建一个服务端，然后专注于写一些获取行政区要素的 `SQL` 即可。

如果你的项目中涉及到了更加复杂的操作，比如说，数据格式转换。坐标转换，矢量栅格互转等等需求，你就需要使用 `java` 作为服务端比较合适。因为 `java` 支持很多的 `GIS` 分析计算库，例如 `GDAL`, `Geotools` 等。有的时候你需要对遥感影像有深度的操作，比如进行一些 `NDVI` 之类的计算和分析，那么你可能使用 `python` 作为服务端会更好，`python` 可以使用 `rasterio`, `pyproj` 等工具效率比使用 `java` 要高得多，还是那句话，`python` 天生适合分析和计算。因为本书主打的是基础和入门，所以这些比较深入的内容将会在以后的书籍中或者视频课中再给大家详细讲解。

具体的服务端项目搭建在这里就不跟大家演示了，如果是从未搭建过服务端项目的学生建议可以先去学习如何使用 `spring boot` (`spring cloud`) 搭建一个服务端，或者也可以选择使用 `node.js` 下的 `express` 框架搭建一个服务端。能实现基本的 `http` 响应和搭建 `restful` 风格的接口即可。其实服务端的基础搭建还是比较简单的。只不过项目中的一些业务会让项目逻辑变的很复杂，例如使用 `node.js` 搭建服务端也需要简单的几行代码：

```

const express = require("express");
const Service = require("./service");
var morgan = require("morgan");
var fs = require("fs");
var path = require("path");
var accessLog = fs.createWriteStream(path.join(__dirname, "log") {
  flags: "a",
});
const app = express();
app.use(morgan("common", { stream: accessLog }));
app.use(express.urlencoded({ extended: true }));
app.use(express.json());
const globeConfig = require("./config.json");
const port = globeConfig.port;
app.listen(port, () => {
  console.log(`Example app listening on port ${port}`);
});

```

其实我们 GIS 服务端的重心不在于什么语言和框架。不同的语言和框架只是能适用于不同的场景而已。真正的核心原理都是相通的。空间分析和数据转换以及计算其实都围绕着数据库操作而进行的（下一章我们会讲到）。因此不管是使用何种语言框架作为服务端，其本质还是要结合几段非常漂亮而又高效的 sql。

即使你使用当下最流行的服务端框架 Spring Boot,我们在编写绝大部分的 GIS 服务端功能的时候也需要借助 postGIS。熟悉 spring 的同学都知道，spring 以“分层”而出名，首先有控制处理前端请求的 controller 层，还有对于业务进行抽象接口定义的 service 层。还有用于和数据打交道的 dao 层，还有与数据库表一一映射用于持久化的 do 层 (entity、bean、pojo) 等等。但是这些东西其实和我们的 GIS 没太大关系，我们的 GIS 服务注重的是分析和计算，而并非传统的增删改查（当然前提是增删改查你要会）。因此我们不必在写代码的时候顾虑太多。但是有些基础的公用的知识我们还是必须学会的，比如 java 语言基础，以及一些 maven 插件的使用，还有项目中一些常见的依赖包的使用。

那么我们以 spring boot 为例子来构建一个简单的 gis 服务端。首先还是打开 IntelliJ IDEA，我们新建一个 spring boot 项目。使用 spring initializr 或者是 maven 创建都可以，这里随大家喜好。

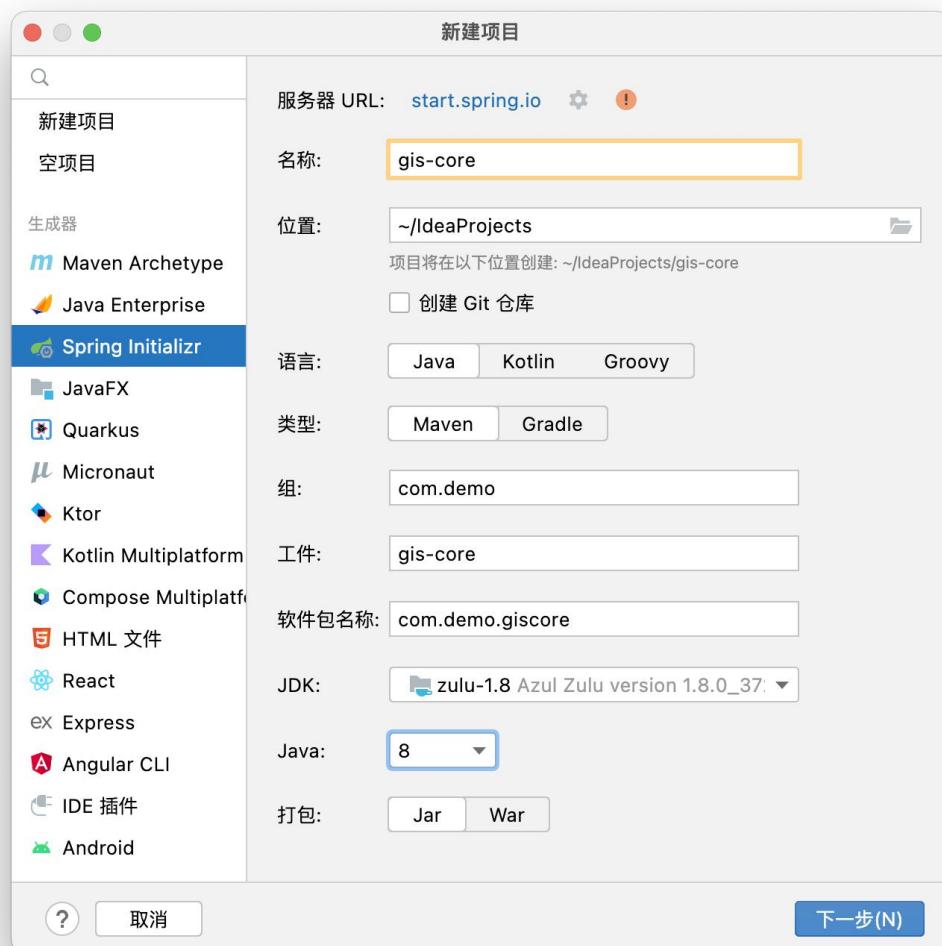


图 11.3.1-新建 GIS 服务端 spring boot 工程

然后填写一些项目中需要的 spring 组件:

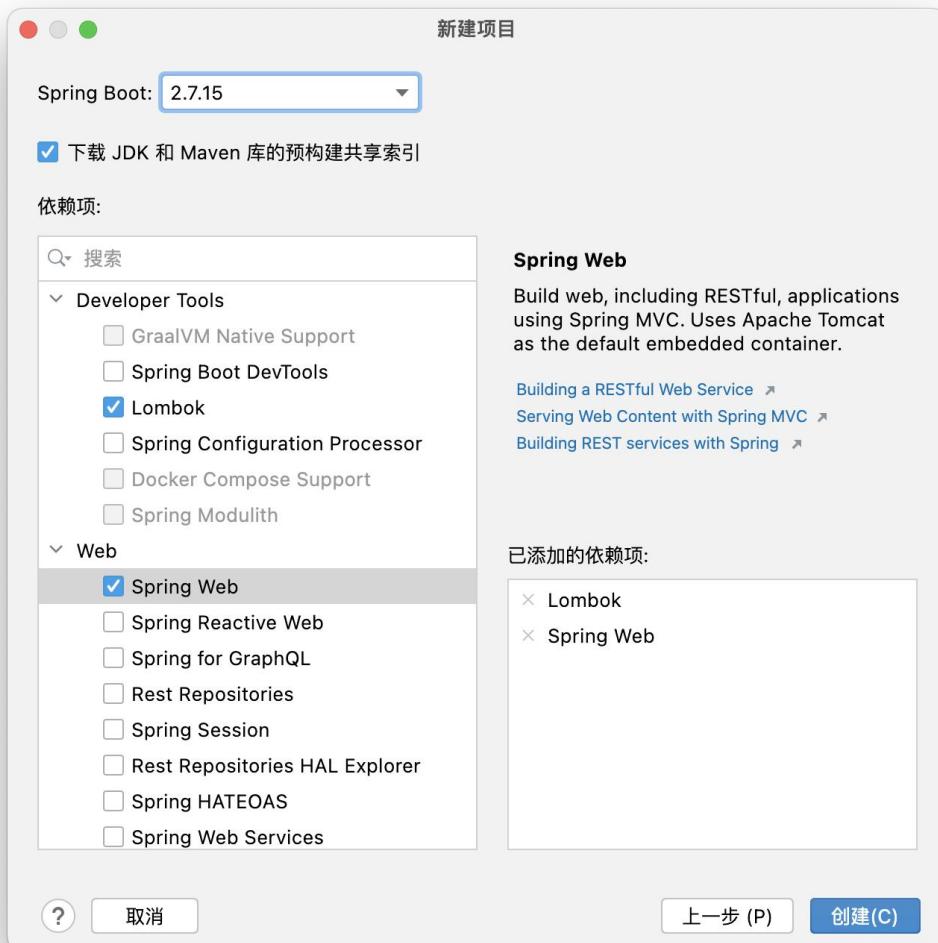


图 11.3.2-选择对应组件

我这里只选择了 Lombok (java 实体类操作工具) 和 web。后续如果需要用到其他的组件我们也可以再次加载。

构建好以后我们的项目工程大概就是这个样子，这个时候我们可以先停下来脚步，进行一个服务端程序的设计，不必着急写代码，首先要整理好设计好你的业务需求，比如说我们现在有这样一个需求，有一个高标准农田项目，客户提供了一些地块数据，要求在我们的系统里能够对这些地块进行一个简单的增删改查操作。按照常理大家可能会选择使用 geoserver，但是它有一个缺点就是查询还行，增删改不是那么的方便。还有一个致命的缺点是 geoserver 不太能够适应海量数据操作，事实上可能当地块数量达到 5000 个左右的时候就会开始出现性能瓶颈。这时候就需要我们自己来编写服务端程序。对地块进行操作。

现在需求明确了，本质上就是对于矢量图形的增删改查操作。前后端交互的数据格式也可以基本断定就是 geojson 格式。那么我们现在就需要对这些操作的接口进行设计。首先对

于地块的查询接口大致可以约定成 2 个，一个是获取全部地块列表。另一个是按照条件获取满足条件的地块，条件可由前端自由指定。地块的新增操作可以是 1 个接口，即批量新增一批地块，当然这批地块可以是单个地块。编辑（保存）接口可以是 1 个，即按照客户端提供的条件 `update` 相关地块，编辑操作我们就暂时不考虑批量修改的业务逻辑了。删除接口也可以设计成 1 个，即按照条件单独删除一个地块，暂时不考虑批量删除接口。

接口设计明确之后呢我们就可以进行代码的编写了。编写之前大家需要考虑一个基础的问题。我们在编写代码的时候可以按照传统的多分层结构的方式来编写代码。即首先建立一个地块的实体类比如就叫 `MassDo`，然后和数据库中的地块表一一映射。按照正常的思路来写增删改查。但是这样做会有一些问题，第一，数据库中地块的图形字段 `geom` 在 java 中没有对应的数据类型能够支持，我们暂时只能使用 `String` 类型来对应 `geom`。其次我们这样查出来的结果无论是 `List` 还是 `Map`，`geom` 字段都是一个十六进制的乱码，这样的数据格式浏览器肯定是不支持的（如果各位不明白，在读完第十二章第 3 节之后就理解了）。不过也不是毫无办法，我们可以使用 `postgis` 所提供的一些函数将这个十六进制的数据转换成 `String` 类型的数据，这样就能够正常的返回数据了。

鉴于这中间涉及到太多数据库和 `postgis` 的知识，我建议各位先读第十二章，读完之后再回来这个部分接着读效果会更好一些。

在这里以一个标准的 `spring boot restful` 风格的接口给大家示例如何完成一个地块采集的服务接口。包含的功能是新增地块，编辑地块（包含图形编辑和属性编辑），删除地块和查询地块列表等功能。

Controller 层：

```
@RestController

@RequestMapping("/mass")

@RequestBody

@Api(value="地块服务模块",tags = "")

public class MassController {

    /**
     * 功能清单
}
```

```

* 1.获取所有的地块列表 (普通 json 的格式, 图形字段以字符串的形式返回)

* 2.获取所有的地块列表 (geojson 格式, 直接以 featureCollection 返回)

* 3.按照条件获取一批地块, 条件可自由指定

* 4.新增地块

* 5.编辑修改单个地块

* 6.按照条件删除地块

*/

```

@Autowired

```

private MassService massService;

@GetMapping("/get-mass-list")

@ApiOperation(value="获取所有的地块",notes="返回值为常规 json")

public List<MassDo> getMassList(){

    return massService.getMassList();

}

@GetMapping("/get-mass-list-geojson")

@ApiOperation(value="获取地块列表",notes="以 geojson 返回")

public List<Map<String, Object>> getMassListCollection(){

    return massService.getMassListCollection();

}

@ApiOperation(value="按条件获取地块数据",notes="返回 geojson 格式的数据")

@PostMapping ("/get-need-mass")

public List<MassDo> getMassByRule(@RequestParam("field") String field,

```

```
@RequestParam("value") String value){

    return massService.getMassByRule(field,value);

}

@ApiOperation(value="新增地块数据",notes="按照数组批量新增")

@RequestMapping ("/add-mass")

public void addMass(@RequestBody MassDo massdo){

    Integer total= massService.addMass(massdo);

    System.out.println(total);

}

ApiOperation(value="编辑地块数据",notes="单条编辑保存")

@PostMapping ("/edit-mass")

public Integer editMass(@RequestBody MassDo massdo){

    return massService.editMass(massdo);

}

ApiOperation(value="删除地块数据",notes="按照条件批量删除")

@PostMapping ("/delete-mass")

public Integer deleteMass(@RequestBody Map<String, Object> map{

    String id=(String)map.get("id");

    return massService.deleteMass(id);

}

}
```

Service 层:

```
public interface MassService {  
  
    List<MassDo> getMassList();  
  
    List<Map<String, Object>> getMassListCollection();  
  
    List<MassDo> getMassByRule(String field, String value);  
  
    Integer addMass(MassDo massdo);  
  
    Integer editMass(MassDo massdo);  
  
    Integer deleteMass(String id);  
  
}
```

ServiceImpl 层:

```
@Service  
  
@Slf4j  
  
public class MassServiceImpl implements MassService {  
  
    @Value("${mybatis.configuration.variables.tablename}")  
  
    private String tableName;  
  
    @Autowired  
  
    JdbcTemplate jdbcTemplate;  
  
    @Autowired  
  
    private MassDao massdao;  
  
    @Override  
  
    public List<MassDo> getMassList(){  
  
        return massdao.getMassList();  
  
    }
```

```
@Override
```

```
public List<Map<String, Object>> getMassListCollection(){  
  
    String sql="select json_build_object(\n" +  
  
        " 'type','FeatureCollection',\n" +  
  
        " 'features',jsonb_agg(st_asgeojson(mt.*)::json)) as results\n" +  
  
        " from "+tableName+" mt";
```

```
List<Map<String, Object>> sqlList = jdbcTemplate.queryForList(sql);
```

```
    return sqlList;
```

```
}
```

```
@Override
```

```
public List<MassDo> getMassByRule(String field, String value){
```

```
    return massdao.getMassByRule(field,value);
```

```
}
```

```
public Integer addMass(MassDo massdo){
```

```
    return massdao.addMass(massdo);
```

```
}
```

```
public Integer editMass(MassDo massdo){
```

```
    return massdao.editMass(massdo);
```

```
}
```

```
public Integer deleteMass(String id){
```

```
    return massdao.deleteMass(id);
```

```

    }

}

```

数据访问 Dao 层：

```

@Mapper

public interface MassDao {

    List<MassDo> getMassList();

    List<Map<String, Object>> getMassListCollection();

    List<MassDo> getMassByRule(String field, String value);

    Integer addMass(MassDo massdo);

    Integer editMass(MassDo massdo);

    Integer deleteMass(String id);

}

```

Mapper 层

```

<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE mapper

    PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="com.hyperspace.giscore.dao.MassDao">

    <resultMap id="massList" type="com.hyperspace.giscore.entity.MassDo">

        <id column="id" property="Id"></id>

        <result column="area" property="area"></result>

        <result column="geom" property="geom"></result>
    
```

```
<result column="mass_name" property="massName"></result>

</resultMap>

<select id="getMassList" resultMap="massList">

    select id,mass_name,area,st_asgeojson(mt.*) as geom from ${tablename} mt

</select>

<select id="getMassByRule" resultMap="massList">

    select id,mass_name,area,st_asgeojson(mt.*) as geom from ${tablename} mt where

${field}=#${value}

</select>

<insert id="addMass" parameterType="com.hyperspace.giscore.entity.MassDo" >

    insert into ${tablename}(id,mass_name,area,geom) values

(#{id},#{massName},#{area},ST_GeomFromGeoJSON(#{geom}))

</insert>

<update id="editMass">

    update ${tablename} set

mass_name=#{massName},area=#{area},geom=ST_GeomFromGeoJSON(#{geom}) where

id=#{id}

</update>

<delete id="deleteMass">

    delete from ${tablename} mt where id=#{id}
```

```

</delete>

</mapper>

```

数据实体 DO 层就省略了大家可以根据 sql 自己创建对应的实体类，另外我这里将数据库的表名做了动态配置。在 application.properties 文件里

```
mybatis.configuration.variables.tablename=mass_test
```

数据库表结构：

列名	# 数据类型	标识	排序规则	非空	默认值
id	1 varchar(50)		default	[v]	nextval('mass_t')
geom	2 geometry(polygon)			[ ]	
mass_name	3 varchar		default	[ ]	
area	5 varchar(10)		default	[ ]	

图 11.3.3-数据库表结构

这样下来的话，一个简单且使用的地块采集的服务端就完成了。只需要配合前端发起对应的请求就可以进行地块数据的增加、删除、修改、查询业务了。还是老生常谈的一个问题，举一反三，我们试想一下，如果把上述的内容换成点位数据，是否这样的代码编写方式和逻辑还能通用？是否一个关于点位的后端服务就构建成功？因此关于 cesium 中加载一些点位的后端服务这个问题，就留给大家作为作业了，我相信这个问题也是非常简单的。

## 第十二章：GIS 数据库基础

### 第 1 节：postgresql 基础

全世界范围内的数据库产品有很多，但是适合与 GIS 行业的数据库寥寥无几。其中 MySQL 和 PostgreSQL 是比较适合 webgis 开发所使用的数据库，但是二者相比较来看，最合适的还是 PostgreSQL (后文简称 pg)，MySQL 虽然也有空间扩展模块，但是相比于 postgresql 来说局限性还是很大，况且 API 数量也没 PostgreSQL 多。pg 首先是一个空间数据库而非传统的关系型数据库，也就是说它对于空间数据有着很好的支持。从存储层面讲，pg 支持对矢量数据以 16 进制的方式来存储图形数据。从操作层面讲 pg 也能够很方便的对

空间数据进行管理和分析。所以我们最流行的空间数据库其实还是 pg。

我们这一节简单的跟大家分享一下 pg 数据库的下载和安装以及基本使用。安装之前要提醒大家的是：尽量下载较高版本的 pg，尽量是 pg14 或者 pg15 及以上，因为低版本的 pg 有很多的 sql 语法是不支持的。

首先我们访问 pg 的官网：<https://www.postgresql.org/>

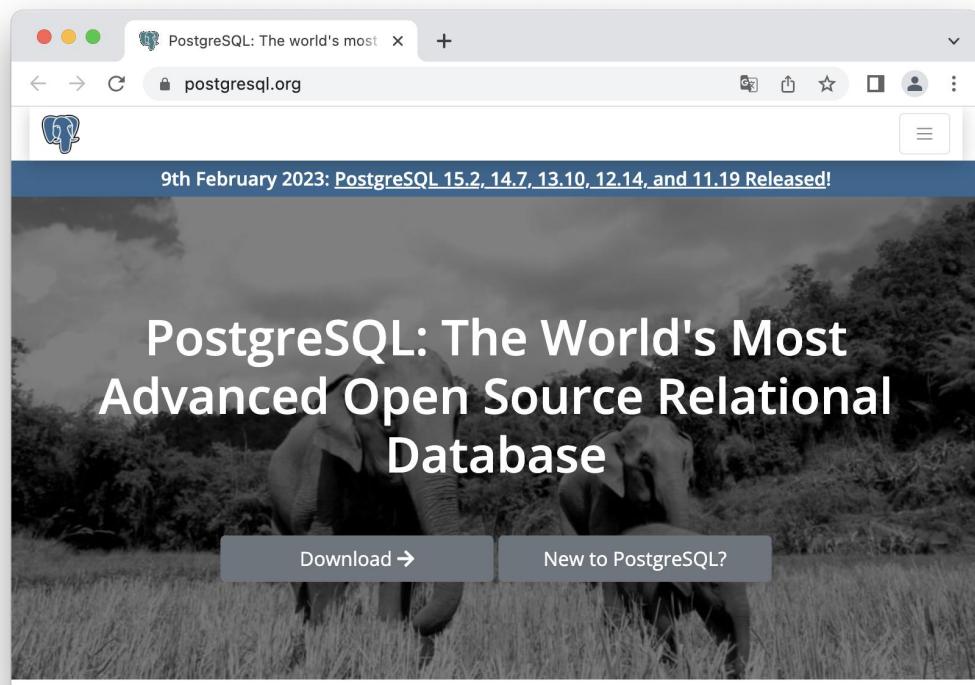


图 12.1.1-下载 postgresql 数据库

直接点击 Download 即可跳入下载页面，无论是 windows 系统还是 mac 系统，它的下载页面只有一个：

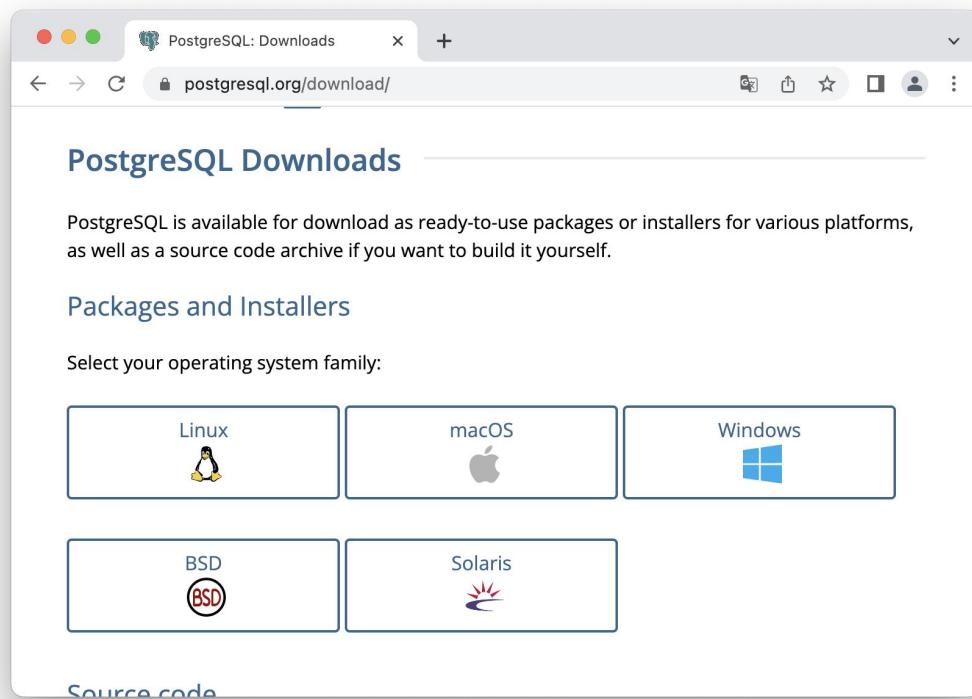


图 12.1.2-选择对应版本下载

选择你系统对应的版本去下载就好了，注意不要下载过高的版本和过低的版本，我本人下载的是 pg15 版本。

下载和安装几乎都很简单，傻瓜式的一路跟着提示走就行了。但是 mac 系统的用户在最后一步需要注意一下，安装完成的时候会提示要安装一个叫做 `satck builder` 的东西，大家要选择安装：

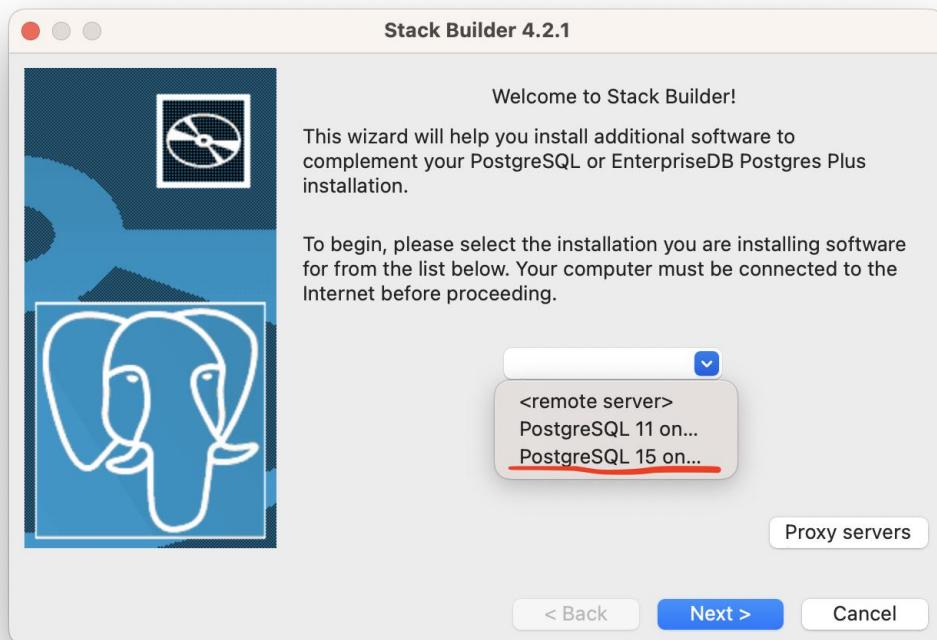


图 12.1.3-安装 stack builder

因为这个 **stack builder** 里面有 **postgis** 插件 (一个进行空间分析和计算的插件库, 下一节会讲到)。

对于 **windows** 用户而言你可以选择和上面的方式一样去安装这个 **stack builder**, 当然你也可以不安装这个, 然后去百度搜索 **postgis** 官网, 打开以后找到对应的 **windows** 版本进行下载也可以。同样还是提醒大家不要下载版本过低的 **postgis**, 笔者的版本是 **3.3**, 各位在下载的时候尽量选择高版本。**windows** 用户下载的 **postgis** 插件会带有一个自动将 **shape** 导入到 **pg** 库的小工具, 叫做 **shapefile and dbf loader exporter**。这个小工具非常的实用, 能够很轻松的将一个 **shape** 文件倒入到数据库中变成一张表。

对于 **mac** 用户而言这个小工具就没办法玩了, **mac** 用户只能采用命令行或者是 **QGIS** 软件把 **shape** 文件导入到数据库中, 命令行操作对于新手来讲太困难, 我们还是用 **QGIS** 吧。

安装完成以后我们就可以在本地找到我们的 **pgAdmin** 进行启动, 初次启动需要你输入用户名和密码, 自己定一个就好。下面要进行一步关键的操作, 就是要安装我们的 **postgis** 插件, 我们打开左侧的数据库, 或者你也可以新建一个自己的数据库, 总之 **postgis** 插件是要安装在数据库之下作为扩展的。打开 **extensions** 选项, 按照下面的图进行操作, 搜索 **postgis** 然后点击保存, 我们就成功的引入了 **postgis** 的支持。

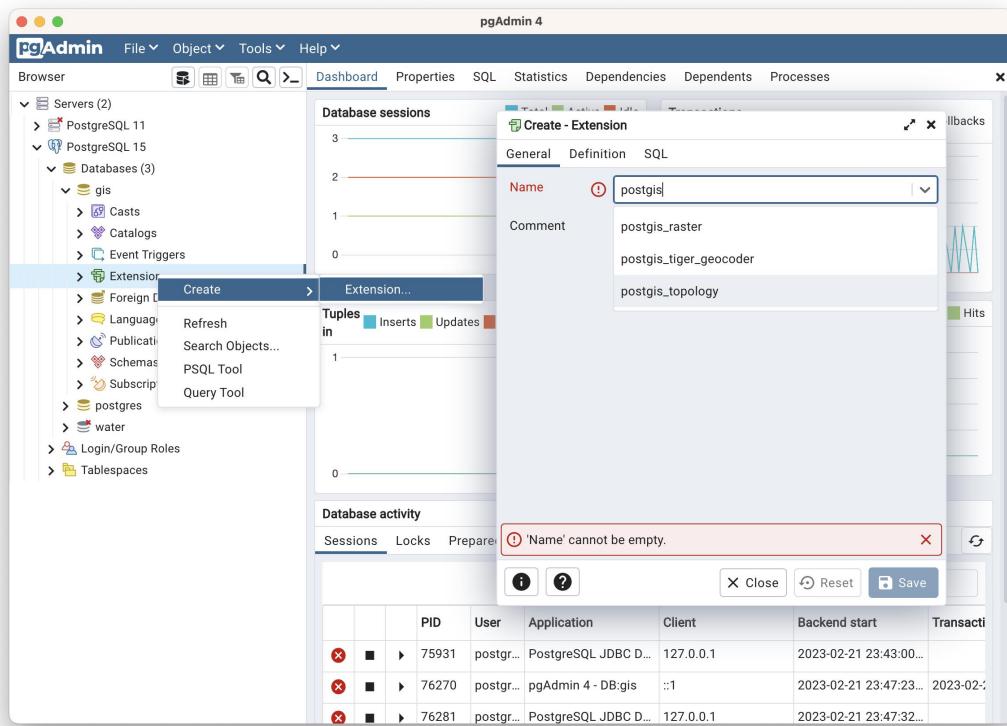


图 12.1.4-安装 postgis 扩展

到目前为止已经算是配置好了 pg 库的基本环境，我们可以关掉 pgAdmin。然后我们需要下载一个链接数据库的工具，可能有从业经验的人这时候马上想到的是 navicat，但是很抱歉我们 GIS 行业不太喜欢用这个玩意，第一它破解起来很费劲，有的时候还像 IDEA 一样有时间限制，过了一段时间又得重新破解。第二也是最主要的原因，它服务于传统的关系型数据库还好，我们的空间信息很多时候是需要跟图形打交道的，因此这里给大家推荐一个免费的开源的超级好用的开源软件叫做 **DBeaver**。这款软件非常的强大好用，我觉得比 navicat 强太多了。而且它同时支持 mac 和 windows，所以我极力推荐大家下载安装它（它并没有找我做广告，只是我一厢情愿的比较喜欢它）同时这个软件的内部还嵌入了 leaflet，能够展示矢量数据的图形信息，这一点简直太赞了，我们能够在数据库中直接查看图形信息。

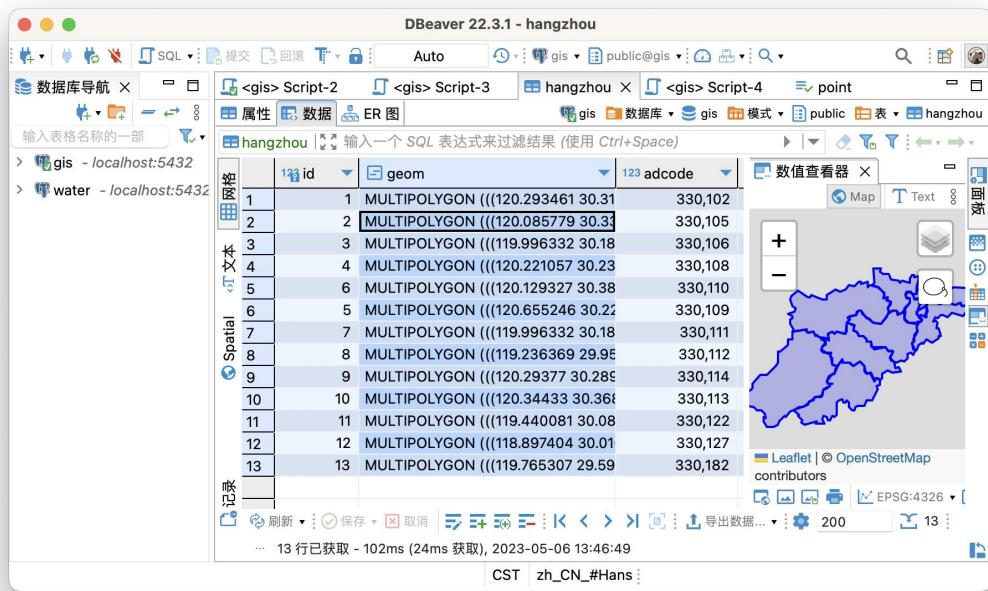


图 12.1.5-使用dbeaver

好了我们现在可以借助工具或者 QGIS 导入一个 shape 数据然后写个简单的 sql 测试一下我们的 postgis 了。把准备好的 shape 数据使用 QGIS 打开。然后在 QGIS 的界面上找到 Database→DB Manager。

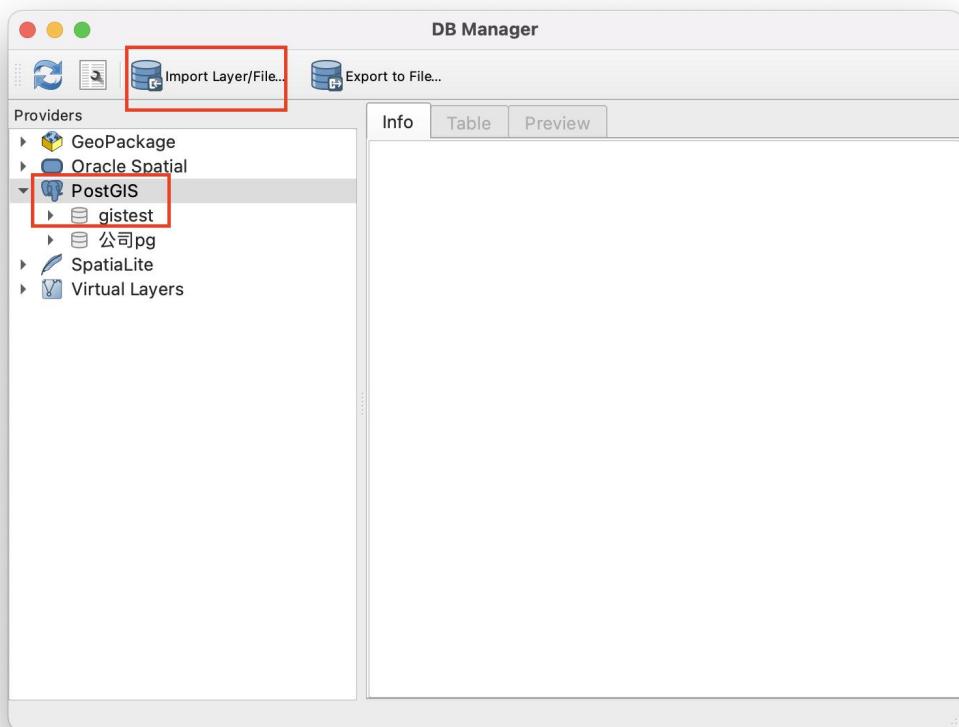


图 12.1.6-QGIS 导入数据至 pg 库

连接好 pg 数据库，然后选择导入我们的图层，根据选项填一些配置项：

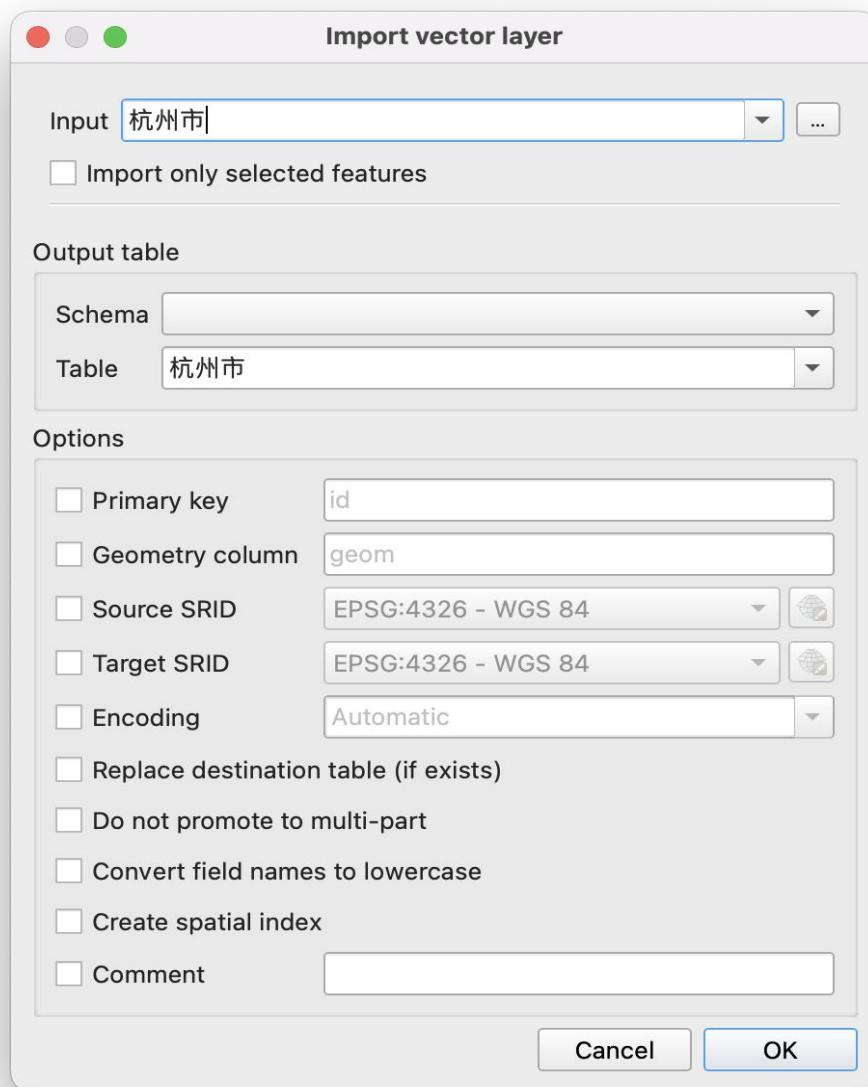


图 12.1.7-导入 shape 数据配置项

最后导入成功以后我们就能够在 pg 库里看到我们导入的数据了。

```
select st_asgeojson(h.*) from public.hangzhou h
```

```

结果 1 ×
select st_asgeojson(h.*) from publi
输入一个 SQL 表达式来过滤结果 (使用 Ctrl+Space)

1 {"type": "Feature", "geometry": {"type": "MultiPolygon", "coordinates": [[[120.293461,30.315084],[[120.295505,30.306291],[120.295196,30.300981],[120.29377,30.289566],[120.288136,30.288235],[120.278961,30.286822],[120.270594,30.284722],[120.269786,30.284185],[120.252838,30.276091],[120.248036,30.272468],[120.241665,30.26514],[120.221057,30.23766],...]
```

... 13 行已获取 - 57ms (1ms 获取), 2023-05-10 16:14:20

CST zh\_CN\_#Hans 可写 智能插入 3 : 1 [48] S:

图 12.1.8-shape 导入成功

## 第 2 节: postGIS 基础

在上一节中我们已经配置好了 pg 数据库并且也安装了 postgis 插件，这一节我们来说说如何使用。postgis 其实就是一个提供了很多 SQL 函数的一个库，我们之前在讲中间件的时候我们提到过一个前端的空间分析库叫做 turf.js。其实 postgis 的功能就和 turf.js 很相似，但是它要比 turf 更强大一点。首先我们要去他的官网上面了解一下它：

<http://postgis.net/docs/manual-3.3/>

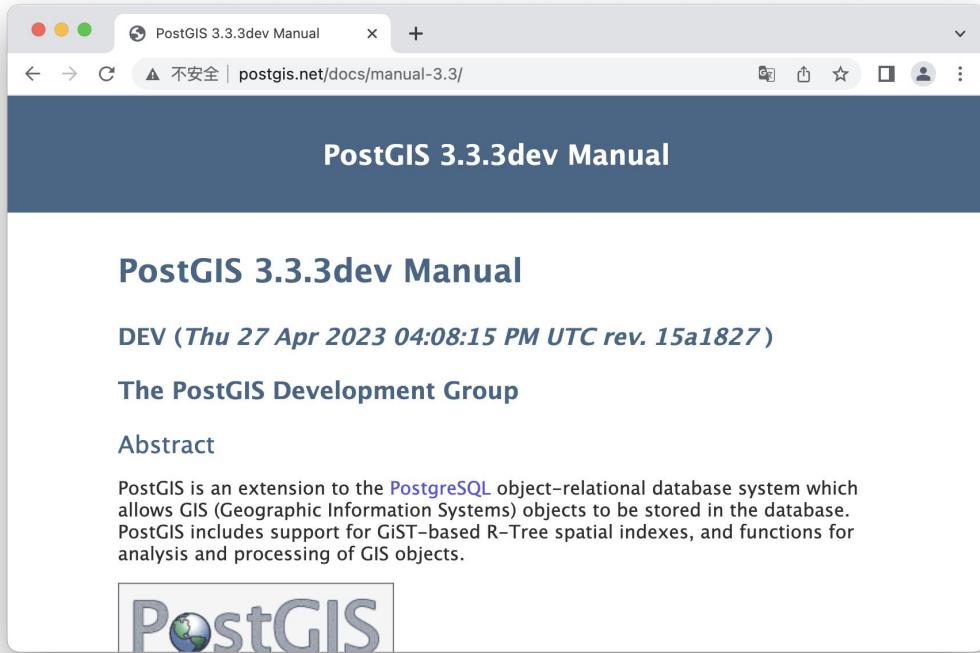


图 12.2.1-postgis 官网

在官方网站上它介绍了 postgis 的一些常用的功能函数，我们重点看第 8 部分的内容，我列

举一下：

1. **st\_Point/st\_polygon** 创建点/面要素
2. **st\_SetSRID** 为要素设置坐标系
3. **ST\_GeomFromText** 将文本数据转换成要素图形
4. **ST\_AsGeoJSON** 将数据作为 geojson 格式返回
5. **ST\_AsEWKT** 将数据作为 wkt 格式返回
6. **st\_intersects** 判断两个图形是否相交
7. **st\_Within** 判断一个图形是否完全在另一个图形内部
8. **st\_Equals** 判断两个图形是否完全相等
9. ....太多了我发现我根本列举不完

功能真的很多，需要具体用到的时候去官网上看下接口 API 具体是怎么用的。我们这本书就挑几个比较重要的说说吧。我们把这些接口大体上分成两类，一类是数据格式转换，一类是空间分析。分别在下面两小节单独讲述。

## 第 3 节：空间查询

数据格式转换是后端 GIS 开发中非常基础的业务，比如说前端向你提交了一些数据，这些数据是 geojson 格式的，你该如何将它们存储到数据库中？前面我们只说了如何将一个 shape 导入到数据库中，却没教过大家如何把 geojson 存储到数据库中。又或者是现在前端需要从数据库中拿一些 feature，我们该如何使用 sql 查询一些要素并且以 geojson 的格式返回？我们这一小节就来解决一下这两个问题。

首先假设你是一个服务端开发人员，前端像向你提交了一段 geojson 数据，像下面这样：

```
{
  type: "Feature",
  properties: { name: "杭州市" },
  geometry: {
    type: "Point",
    coordinates: [120.3422342, 29.23423],
  },
};
```

这时候作为服务端的你如何把这个数据存入数据库呢？当然直接存肯定没有问题，把它当成一个普通的 json 对象。然后使用常规的 `insert` 语句去存储它肯定是没有问题的。但是那是非 GIS 行业后端的做法，作为专业搞 GIS 的你，你必须将其以专业的方式存储起来，那就用到了我们的 `postgis`。

我们可以利用 `ST_GeomFromGeoJSON` 这个函数将 `geojson` 中的图形字段转换成数据库中的一条图形数据。这就不得不介绍我们的图形数据在数据库里是如何存储的。

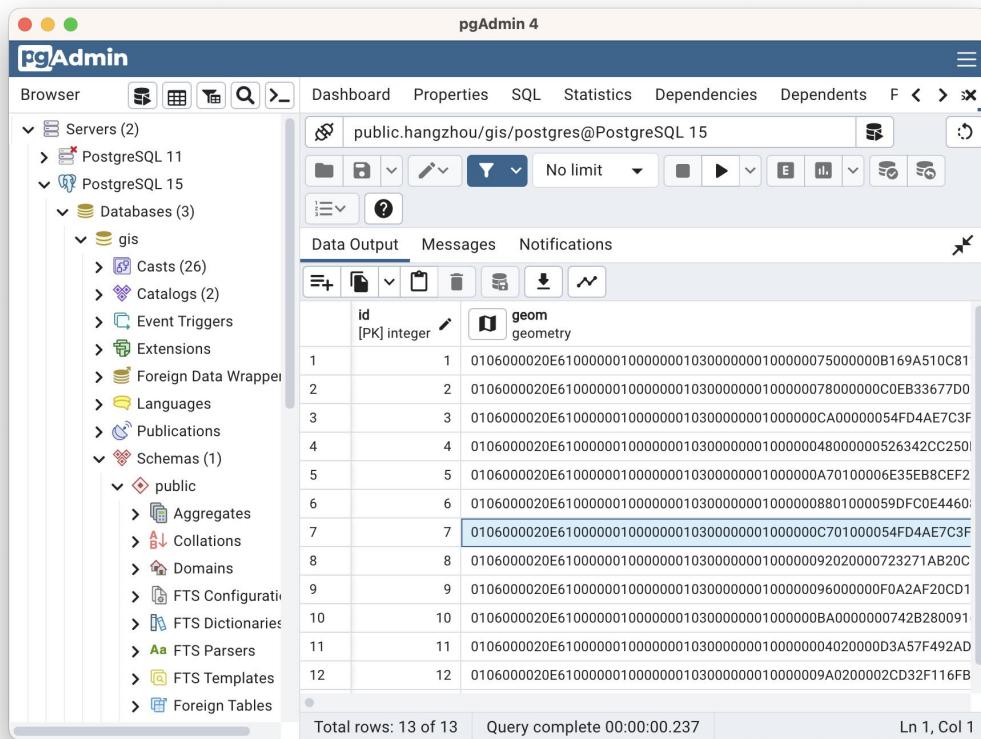


图 12.3.1-十六进制存储图形字段

实际上图形的数据在数据库中并不是直接存储经纬度数据，而是存储的十六进制的数据，这样做第一能够减少数据体积，第二能够加快查询和检索速度。所以我们必须借助

postgis 将经纬度数据也转换成十六进制的数据存储。

```
insert into public.point("id" , "geom", "name") values
(8,ST_GeomFromGeoJSON('{"type":
```

```
"Point", "coordinates": [120.34223423453, 29.837465233]}'), '杭州市')
```

现在向数据库里插入一条 geojson 相信大家已经没问题了，那么如何从数据库中取出 geojson 呢？我们要用到一个关键的函数叫做 ST\_AsGeoJSON()。具体写法如下：

```
select st_asgeojson(h.*) from public.hangzhou h where name='西湖区'
```

## 第 4 节：空间分析

WebGIS 的空间分析必然是一个系统最能体现能力的地方，通常空间分析都依赖于服务端去做，服务端所使用的最常见的空间分析手段也是 postgis。因为它提供了太多的空间分析函数。比如现在我们要判断两个图形是否相交，我们使用 st\_intersects(geom1,geom2) 就可以判断。

```
select st_intersects((select geom from public.point where
```

```
id=2), (select geom from public.point where id=3))
```

```
--不相交返回 false
```

如果判断两个图形在空间上是否存在包含关系。也就是一个图形是否完全位于另一个图形的内部，可以像下面这样写：

```
select st_within('SRID=4326;POINT(120.217685
```

```
30.216143)::geometry, (select geom from public.hangzhou h where
```

```
id=4));
```

```
--返回 true
```

类似于这样的分析函数还有很多，在这里就不给大家一一举例子了。大家可以根据自己真实的项目场景所需要的分析找到对应的分析函数查看其用法，然后对应修改参数即可。

另外大家对于这些分析函数一定要认真研读其含义，因为有的时候参数的顺序不一样结果就会完全不一样。比如说 `st_within(geom1,geom2)` 和 `st_contains(geom1,geom2)` 都表示包含。可是前者表示的是 `geom2` 是否包含 `geom1`，而后者表示的是 `geom1` 是否包含 `geom2`。这点细微的区别还是要靠认真阅读官方文档才可知晓。另外对于一些空间关系判断的理解上也需要大家多花些功夫。比如说 `touch` 和 `intersects` 相交有什么区别？何为 `touch`？

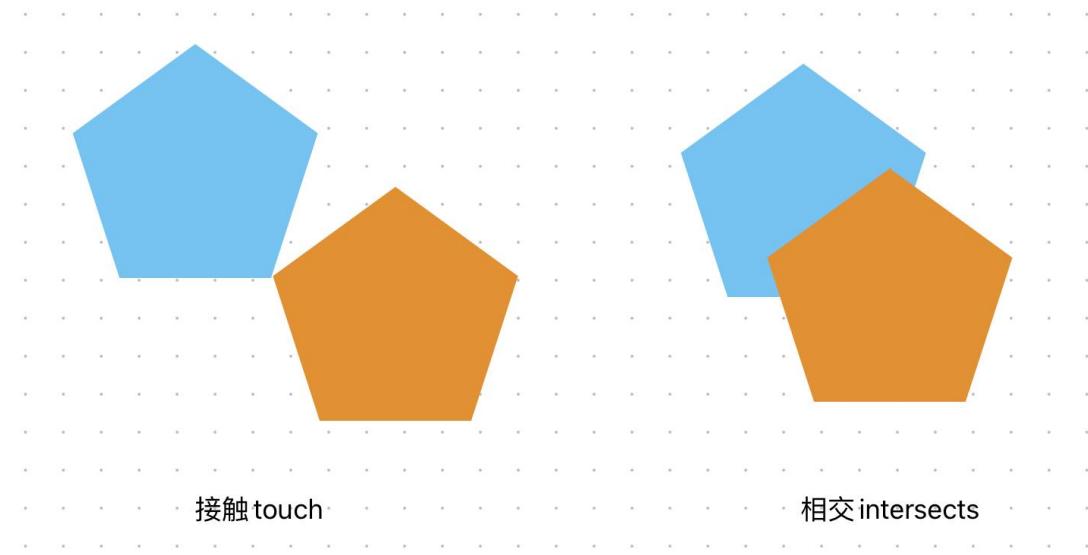


图 12.4.1-空间关系

`touch` 指的是两个图形有接触，但是有且仅有一个触点，但是相交的含义是至少有一个触点。也就是相交肯定是包含接触，但接触不一定包含相交。这一点在做一些地下管线的空间分析项目中可能会用到。另外随着 `postgis` 的越来越完善，越来越强大，`postgis` 甚至能分析 3D 的一些空间关系。大家有兴趣可以下去研究研究。

# 第十三章：GIS 架构设计

## 第 1 节：GIS 组件化

随着现代前端的飞速发展，使用框架来进行组件化的开发已经非常盛行了。`vue`、`react` 等框架几乎已经风靡了整个前端行业。大大小小的项目现在几乎都是使用组件化的方式去开发，组件化有两个优点，第一是开发速度比较快，组建复用性强的话就大大减少了开发的工作量，第二是渲染效率高，虚拟 `dom` 和响应式的一些概念能够有效的减少对真实 `dom` 的操作，因此能大大的提高渲染的效率和性能。前面也说过我们的 GIS 发展跟传统的互联网行业的发展息息相关。行业中任何新鲜事物的产生都能够对 GIS 发展产生影响。因此一个 WebGIS 应用如何去设计，去构建也是一个非常重要的话题。设计方式和架构的不同就直接决定了这个应用的适用场景，也直接决定了应用的优点和缺点。

一个 GIS 应用的架构可以有很多种设计方式，本章给大家讲两种最常用的也是最好用的两种设计方式。

第一种就是我们的 GIS 组件化。什么意思呢？相信大家应该使用过一些前端的 UI 组件库，例如 `Element UI`、`Vant4`、`Ant Design` 等。还没用过的小伙伴抓紧时间去百度一下，我在这里等你 5 分钟，回来我们接着讲.....其实这就是把前端常用的一些 `html` 元素抽象成组件，例如一些输入框、按钮、单选按钮、复选框等等抽象成组件，这样开发人员在编写页面的时候就可以通过简单的传几个参数直接使用。我们的 GIS 组件也是一样，要把我们 GIS 的一些常用的功能抽象封装成组件，然后通过传递不同的参数来灵活的复用组件，大大提高了开发效率。

具体该如何设计呢？首先我们的 GIS 常规应用应该大致有以下这么几个方面的功能：  
GIS 底图、各类型图层加载、定位、导航、地址搜索、测量、绘制、地图导出、等等。如果你的项目里有具体的应用你可以具体再添加个性化的组件。所以我们可以按照这些功能来设计一套组件库。

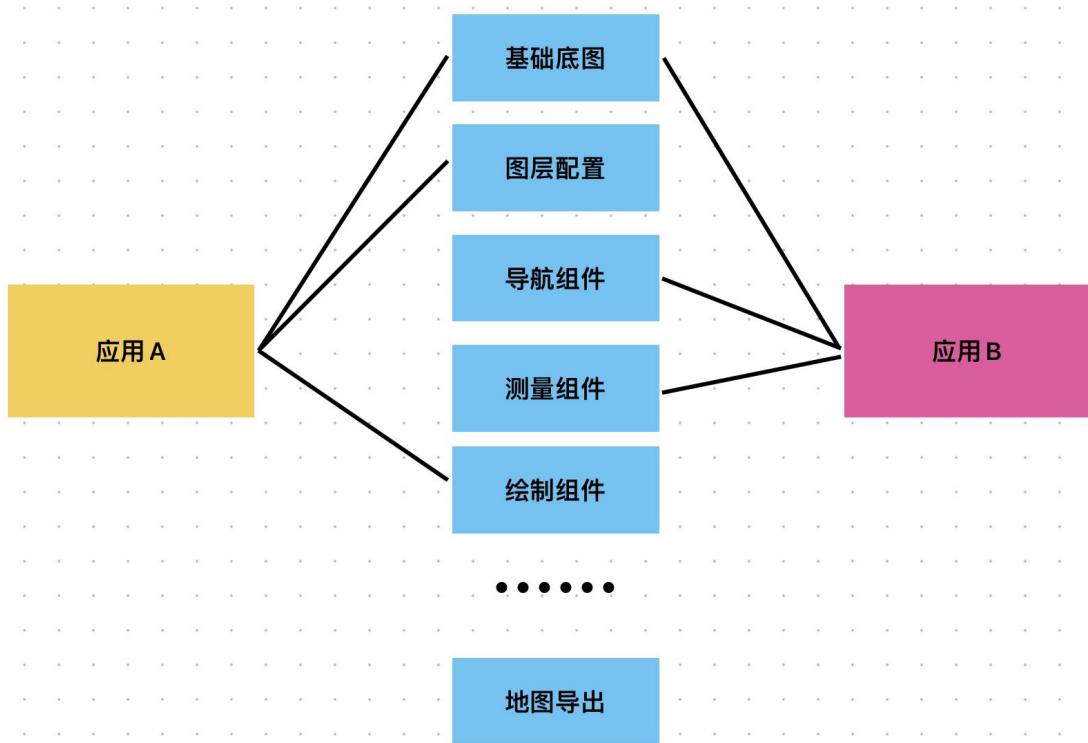


图 13.1.1-组件化 GIS

具体的技术路线有很多，在这里给大家提供一种思想。实际上你可以使用 vue、react 等框架去写一写这些组件，结合你学到的组件之间的数据传递，组件参数的传递等知识灵活的去复用这些组件。这就是我们 GIS 组件化的架构设计。

## 第 2 节：嵌入式 GIS 架构

除了组件化的 GIS 应用之外呢还有一种嵌入式的 GIS 架构。其思想也是现将 GIS 部分和非 GIS 的业务部分分开。将 GIS 部分的一些常用的页面逻辑进行总结和分类，然后将代码逻辑进行封装，具体可以抽象成一些配置类或者配置函数，然后只要通过参数的不同来渲染出不同的效果。就比如说要展示一个行政区这样基础的需求，其实完全就可以抽象成一个函数，后续只需要提供数据和设置样式，剩下的都可以重复的依靠代码来完成。经过一系列的抽象之后，这些功能点其实都被抽象成了模块，然后我们根据具体的业务场景来决定这些功能点的配置。最后为这个应用生成一份可执行的代码文件，通常是一个文件夹，然后再将 GIS 部分和其他的业务部分进行融合。这样也能够做到快速开发且可重复使用。

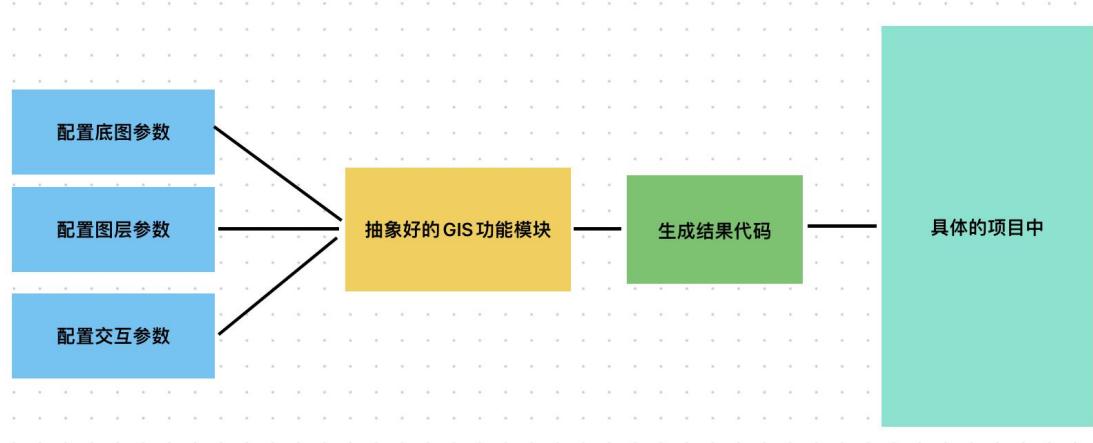


图 13.2.1-嵌入式 GIS

## 第 3 节：个性化解决方案

其实任何架构都有优点和缺点，不存在最完美的架构设计，而只有最合适，性价比最高的架构体系。GIS 组件化的设计的优点在于可复用性高，组件库一旦成型之后后续的开发都能够非常高效快捷的完成。但是缺点在于前期开发组件的周期比较长，因此通常适合大型中型公司去做，再有就是组件需要一直维护，因为需求的不同导致组件的数据传递方式和使用方式需要一直有变化或者是需要同时提供多套组件。对于嵌入式 GIS 架构好处也是适合复用，提高开发效率，低代码开发。但是缺点在于后期维护起来比较麻烦，因为 GIS 部分和常规业务部分是剥离开来的，因此维护数据会比较麻烦。具体你的 GIS 应用如何设计如何安排取决于具体的业务场景、公司的人员配置、投入的资金等等、有很多影响因素。同学们在进行应用设计或者框架设计的时候要认真的考虑清楚，以实现的目的和需求为首要，还有很多很优秀的设计方案等着大家不断的总结和探索。

为什么本书不多讲 vue 框架和 react 框架与 WebGIS 的结合？这是阅读过早期版书籍后的一部分同学提出来的问题。主要的原因有两个，第一是因为前端框架很多，不仅仅只有 vue 和 react，现在仍有一部分人在使用 angular。如果结合前端框架去讲的话太复杂太累，实际上我们说过支持这些框架最好的方式就是原生，原生的 html、css、js 无论是什么样的前端框架都可以完美支持。并且学会原生的编写代码的逻辑之后，再将结果移植到 vue、react 等框架中是一件很容易的事，两者没有太大区别，在 html 里写的核心代码可以完美的移植到 vue 里，只不过语法格式需要稍作修改，比如方法不用再写 function，数据有专门的数据选项 data 等等。略微研读 vue 官方文档即可轻松完成转换。另外这些前端框架更新的速度

很快很频繁，而我们的 **gis** 框架相对没有那么的快，因此不做结合有一点原因也是因为笔者没有时间频繁的因为前端框架的更新而更新此书籍。第二个原因是后续笔者将会推出新的专门研究 **gis** 框架与前端框架结合的数据。例如《leaflet+vue 实战》等等。在那本书中大家可以敞开了研究 **gis** 框架与前端框架想融合的一些细节。

## 第 4 节：构建低代码平台

随着当前 AI 技术的火爆，市场对于程序员的需求大大下降，但是对于优秀的程序员的需求却未发生任何变动。当你开发过足够多的 GIS 应用你就会发现，其实绝大多数的需求都是可以被重复抽象的。绝大多数的应用只是数据不同，研究区域不同，其实从功能上和操作上都是非常相似的。你可能会说第 1 节中的 GIS 组件化不就能够解决这个问题吗？其实不然，GIS 组件化只是降低了开发难度，或者说提高了开发效率，但是还是需要人去开发，即使这个开发的人或许已经是 N 次开发了。即使有一批人封装好了一部分组件，那也需要另一部分人在实际的项目中去使用这些组件去开发项目，就好比 Element UI 或者 Ant Design 这样的产品。你在做实际项目的时候不还是需要使用它们再开发一次吗？

那么有没有一种可以不用开发只通过简单的配置就直接可以自动生成 GIS 应用的方式呢？当然是有的。我们仔细来思考以下这个逻辑，首先我们所使用的任何一个 GIS 框架的 API 都是固定的，API 都是固定去完成某个需求的，就比如说 `setZoom()` 方法就是用来调整地图缩放层级的。那么我们就可以围绕这些 API 来研究接口的参数，实际上这些参数的不同就意味着不同的使用场景或者说应用场景，那么我们只需要根据常用的需求和应用场景概括出需要用到的 API 和其参数即可。另一方面我们可以把项目中与数据有强关联的部分抽出来，将这部分做成可配置的即可。

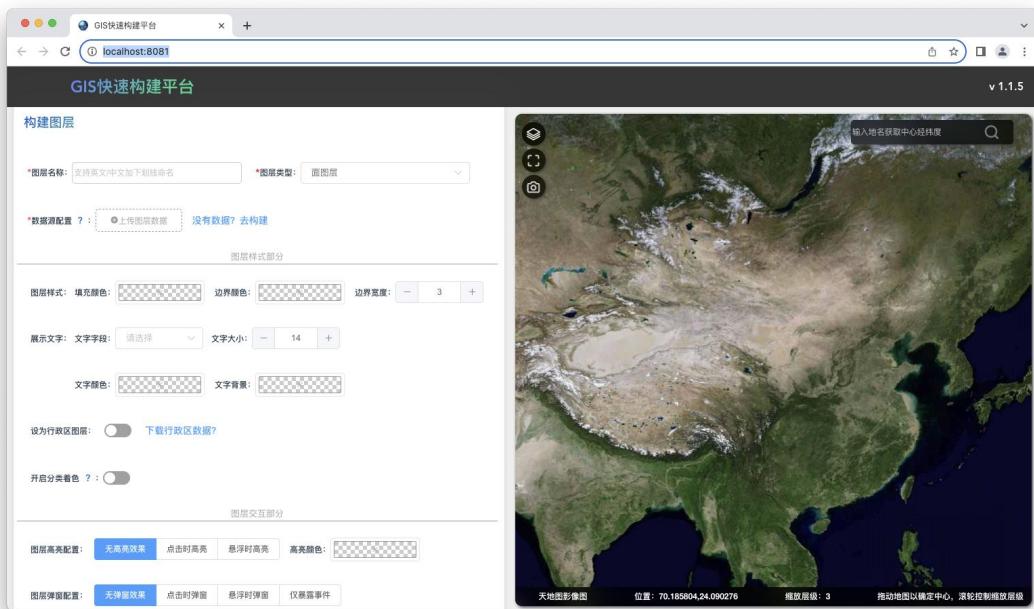


图 13.4.1-低代码平台思路

这是我即将推出的 GIS 低代码平台，它能够实现的功能非常的强大。从数据获取、应用配置到功能交互、生产上线整个一体化流程都做了处理。毫不夸张的说。这个平台可以在几分钟之内就构建好一个成熟的可用的 **GIS APP**。80% 场景下的项目需求都可以通过这个平台快速配置，配置结束即可投入使用。

快速构建平台有两个重点（特色），第一是“低代码”甚至可以说是无代码。整个过程中不需要写任何代码。将功能逻辑和参数进行分离，用户只需要关心 app 的表现和交互以及功能，不需要进行任何的代码编写。通过配置的结果可以直接在线查看样式，不满意可进行调整和修改。确定好之后可以直接选择打包。第二是广泛适宜性，从这个平台打包出来的应用可以用在任何技术架构的项目中（react、vue、甚至是 angular）都可以完美的支持。

所以这个平台也会作为我第一个成熟的产品进行销售。针对的用户可以是公司也可以是个人。如果是个人，会省去大部分开发重复项目的时间。如果是公司。可能.....可以省去好几个员工的成本。因此这个平台我不会开源。而且也不会低价出售。

## 项目实战案例

我们 5、6、7 三章分别讲了 leaflet、mapbox、cesium，那么我们三个实战案例就对应这三章的内容，以具体的需求场景为例子带大家实战 webgis 项目开发。只有实践才能让各

位的知识体系掌握的更加牢固，在开始讲解之前，各位需要注意两个问题：

1. 各位需要申请自己的 token，用于访问 mapbox、cesium、天地图等。
2. 三个实战案例都是采用 vue 编写，因此大家需要熟悉 vue 和工程化知识，启动项目之前记得 npm install。
3. 受限于时间和主题，项目中与 GIS 相关性不大的内容省略掉了，另外很多数据也没有部署服务，采用静态测试数据。如关心服务端的内容，可以阅读第十章的内容自行改写。

## 实战案例一

### 1. 项目背景及需求

第一个项目实战的案例我们简单一些，然后带大家循序渐进，现在我们第一个实战案例就是用 leaflet 来实现一些移动端的需求。现在有一个旅游相关的小程序项目，杭州市西湖为旅游景区，需要做一个旅游小程序，实现一些功能，具体的需求如下：

- (1) 获取用户当前的位置，并且把当前位置标记在地图上
- (2) 展示西湖景区的地图，包含文字注记。
- (3) 展示该景区内的各个景点的位置，并且弹窗展示各个景点的图片和文字介绍
- (4) 要支持用户搜索景点的功能，当检索到用户心仪的景点位置之后，点击景点名称地图要跳转放大到对应的景点位置，并且弹窗展示景点详细信息，提供外链以便用户使用高德等导航软件导航到景点。

### 2. app 设计思路

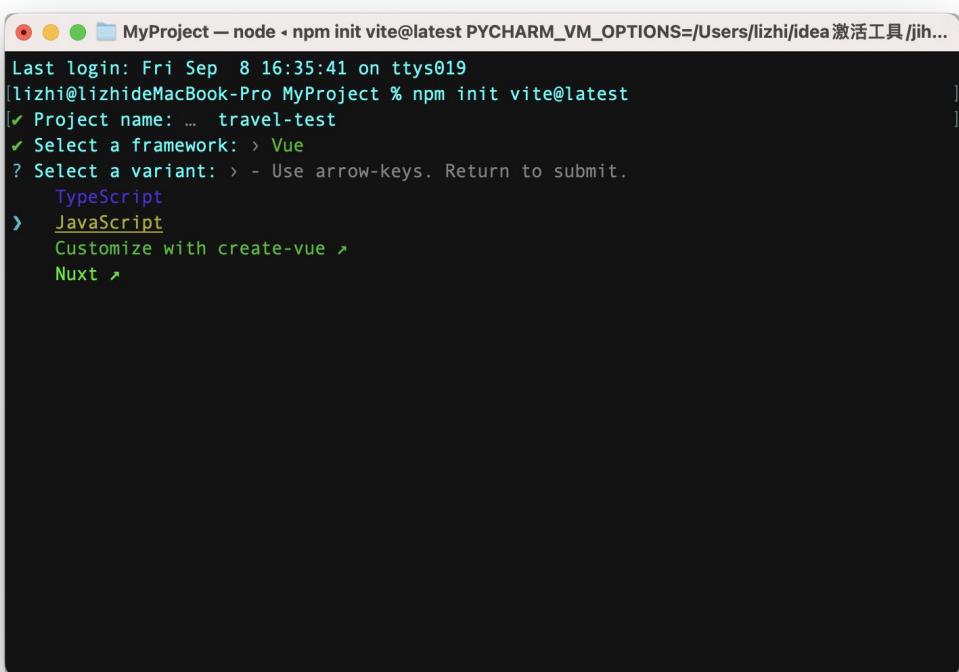
这四个需求其实都是移动端 GIS 开发的常见需求，首先第一个，获取用户当前的位置并且把位置标记在地图上，这是非常常见的需求，实现这个需求的方法和思路有很多，比如说我们可以通过申请获取浏览器的定位，得到当前用户所处位置的坐标，也可已通过高德所提供的接口获取当前的位置，实战场景中需要根据具体的 app 应用场景选择对应的获取位置的方式。这里需要注意的一点是，如果使用的浏览器原生的定位方式，需要在项目上线后将访问地址升级成 https 的，否则谷歌浏览器是不允许对 http (不安全) 的网页开放定位接口的。但是在开发阶段各位可以使用 Safari 浏览器或者火狐 (Firefox) 来查看当前位置的效果。第二个需求展示西湖景区内的地图和文字注记这也应该是 GIS 的最基本需求了，我

们在这个案例中采用天地图的矢量风格地图，文字注记也采用天地图的文字注记。第三个需求展示景点位置以及所对应的弹窗需要我们使用 `L.marker()` 方法和 `popup` 相关方法来操作。第四个需求，搜索景点及定位并且弹窗展示景点详情，这需要大家有一些 `h5` 和 `css` 的基础，来做一些页面操作逻辑。

### 3. 实现过程

那鉴于需求整体都比较简单那么我们就直接使用 `vue` 来给大家把整个项目还原一下吧。第一步还是新建一个 `vue` 工程，由于是小程序，单页面应用，我们直接准备一个 `vue` 组件就好了。

执行以下命令在你准备好的文件夹里新建一个工程，名字叫做 `travel-test`。



```
MyProject — node - npm init vite@latest PYCHARM_VM_OPTIONS=/Users/lizhi/idea激活工具/jih...
Last login: Fri Sep  8 16:35:41 on ttys019
lizhi@lizhidemacBook-Pro MyProject % npm init vite@latest
[✓] Project name: ... travel-test
[✓] Select a framework: > Vue
? Select a variant: > - Use arrow-keys. Return to submit.
  TypeScript
  > JavaScript
  Customize with create-vue ✘
  Nuxt ✘
```

很快就创建好了，使用 `vs code` 打开项目工程，在终端中输入

```
npm install
```

命令先为项目安装对应的依赖包。

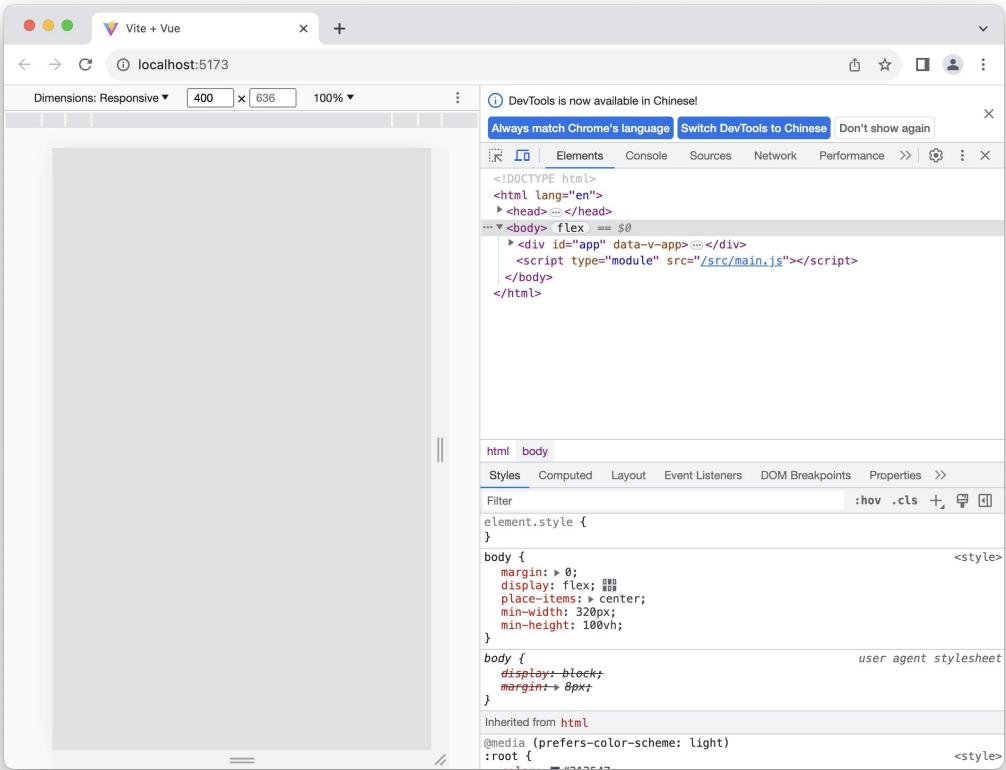
然后在 `components` 文件夹之下新建一个 `vue` 组件叫做 `MainPage.vue`。写好其基本结构。这里我就不细讲了，这是 `vue` 基础知识，不懂得同学可以下去专门学习一下。接下来我们开始构建 `leaflet` 相关的内容，首先还是引入 `leaflet` 相关的依赖：

```
npm install leaflet
```

然后开始给小程序的页面写一些基本的样式，首先小程序是移动端的应用，我们的页面样式通常要适配移动端：

```
<style>
#main {
    width: 100%;
    height: 100vh;
    inset: 0;
    margin: 0;
    position: absolute;
    background: #e1e1e1;
}
</style>
```

写好样式之后我们可以启动项目，把 App.vue 中的 helloworld 相关的删除，换成我们的 MainPage.vue 组件，启动后我们用浏览器的移动端模式查看：

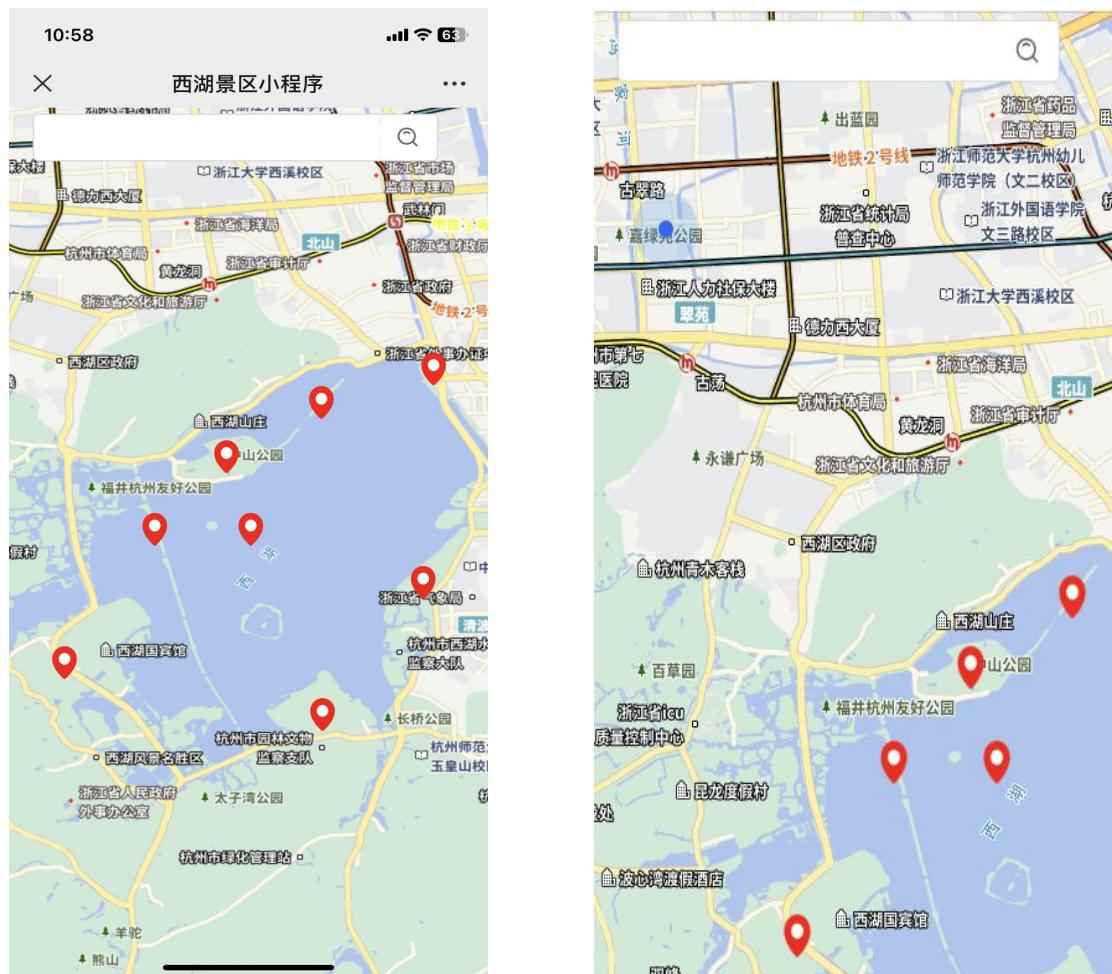


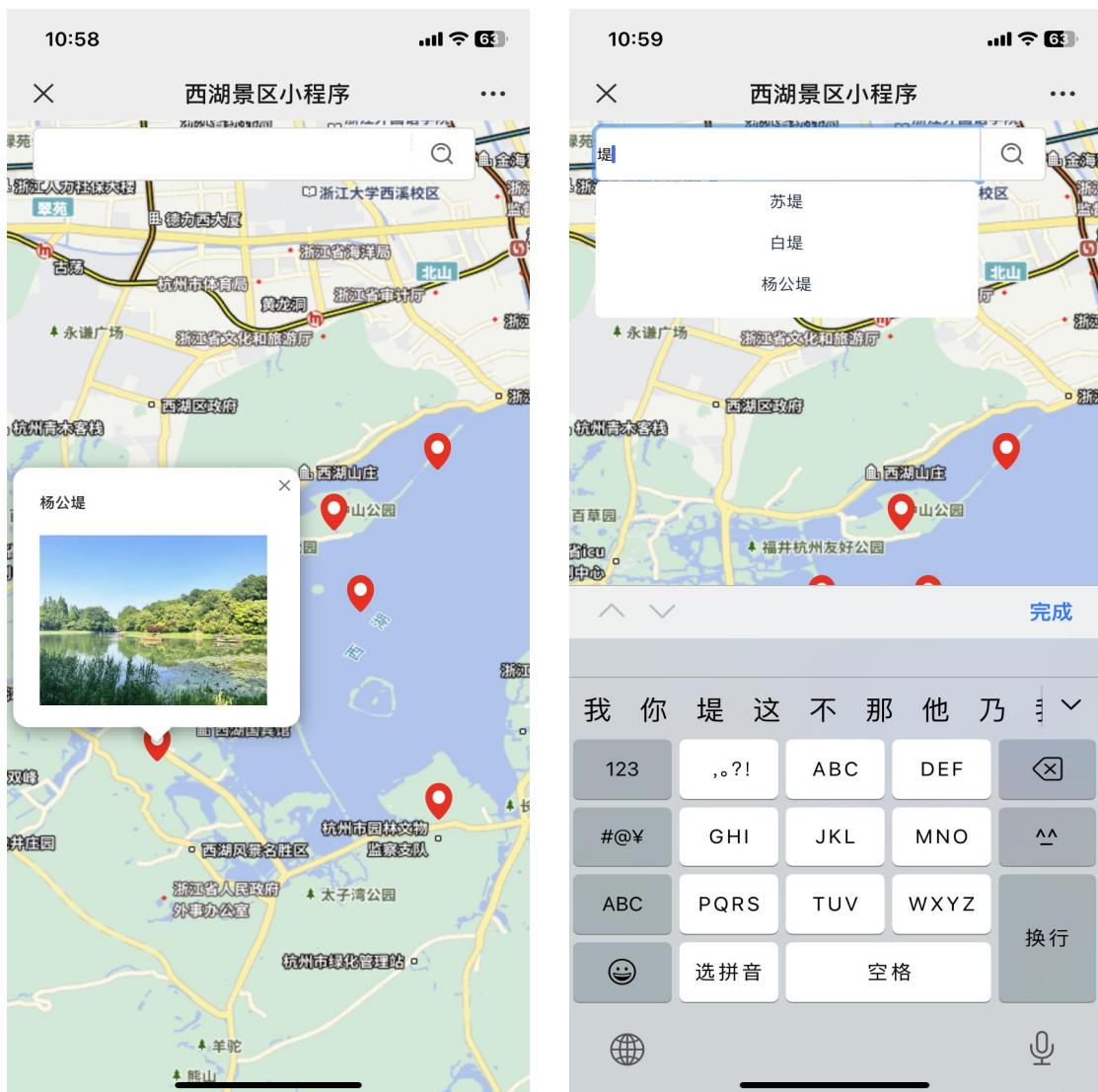
好接下来我们开始写 GIS 部分的代码，首先第一步我们要给一副基础的地图，我们使用天地图的矢量版，关于如何加载天地图我在第 5 章也讲过了在这里就不重复讲了，代码也不必粘贴在这里了，本书在卖给大家的时候会附带源码文件夹，大家自行打开查看。在这

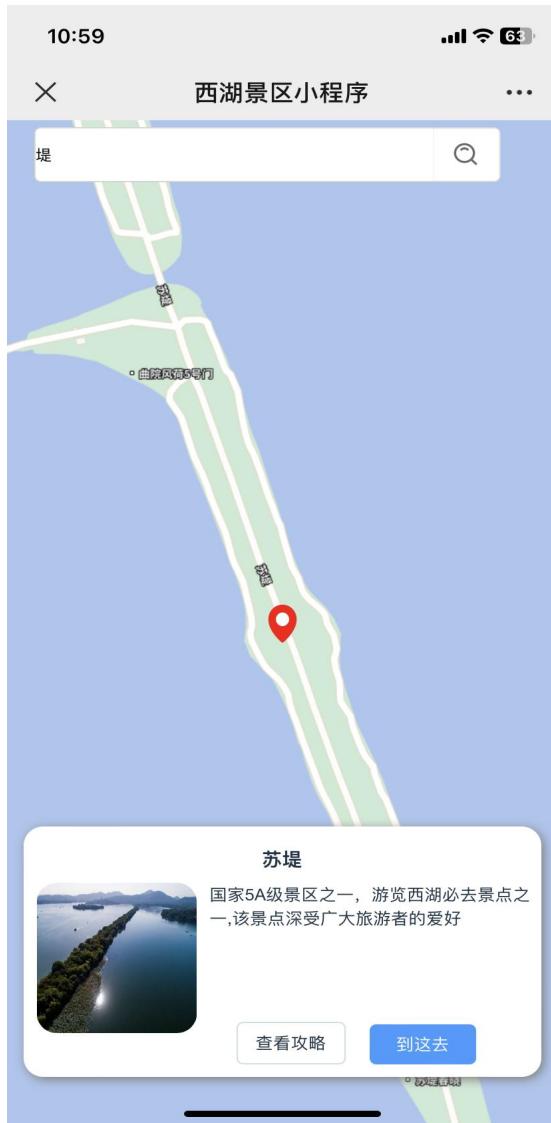
里我们只强调两个特别关键的问题，第一要注意在 leaflet 中经纬度的顺序是反着的先是纬度后是经度。因此需要调换天地图切片地址中的 x 和 y 的顺序。第二如果不指定坐标系我默认使用墨卡托坐标系的天地图切片。

我们通过之前第 5 章的核心代码先来加载好天地图的切片，由于重复性代码太多，在这里我就不粘贴源代码了，详细代码大家可以查看附带的源码文件夹，我给大家写好了详细的注释。这里也建议大家一边阅读数据理清思路，一边参考源码学习如何实现，这样的方式比较高效。

我们在页面设计上采用单页面应用，页面上的元素很简单，就是一个地图，一个搜索栏，当用户选择了搜索结果之后弹出对应内容并且地图放大聚焦展示当前的点位。然后点击地图上的点位弹窗展示名称和图片。对应的代码各位可以阅读源码文件 `MainPage.vue`。







## 4. 思考与延伸

leaflet 素以轻量的、高效的移动端 GIS 框架而著称，所以各位今后在做移动端项目的时候要好好考虑 leaflet。另外就是各位在做移动端 app 的时候要特别的注重样式，确保你的页面在移动端有着合适的尺寸适配和交互反应，西湖旅游小程序这个项目中为了突出演示 GIS 的功能，在样式上没有做过多的整饰，大家在实际的开发中可以加入更多的与用户友好的操作，比如一些动画，底部展示景点详情那个弹窗可以由下向上弹出等等。同时各位在开发移动端项目的时候还避免不了与设备本身的应用程序或者系统级别的信息打交道。通常我们现在的 webgis 能力很难做到导航功能，最主要的是我们没有庞大的数据基础。因此在进行具体的导航功能的时候还是不得不调用第三方的接口。这就需要我们了解 h5 页面调用设备

app 的相关接口。最后要说的就是性能问题，在实际的开发中还是建议各位使用私有的或者客户提供的数据源，比如景区的手绘地图来代替这个项目中的天地图，关于手绘地图的加载我在第 5 章中也有提到过。

## 实战案例二

### 1. 项目背景及需求

现如今 WebGIS 行业需求点最多的场景可能就是大屏了。各种各样的花里胡哨的数字大屏。通常情况下都需要 GIS 的支持。现在有一个浙江省土壤质量检测相关的项目，甲方提出了以下几个需求，请各位考虑一下，我们作为开发人员该去如何实现这些项目需求：

- (1) 该系统共分为 2 层行政区等级，首页展示的是浙江省各个城市，以及各个城市土壤监测点数量的汇总统计，然后按照城市土壤监测点数量来渲染城市颜色，土壤监测点数量越多颜色越深。
- (2) 行政区划要求有点击下钻的功能，点击行政区划之后要下钻到对应的行政区，并且自动展示该行政区的土壤检测点的数量或者是具体监测点位的信息。
- (3) 具体的检测点位的信息必须弹窗展示，当用户鼠标悬浮时自动弹窗显示信息，该区县的土壤监测点的信息，鼠标离开后隐藏掉弹窗，并且要伴随有轮播功能，每隔 3 秒自动播放每个区县的土壤监测点的信息。
- (4) 还要有行政区返回功能，从浙江省下钻到某城市之后，再返回浙江省，回到初始状态。

### 2. app 设计思路

第一个需求，行政区颜色的渲染要根据土壤的质量来进行渲染，那么必然也是业务同事会返回给我们土壤质量的数据，而且这个数据 必须与行政区代码唯一对应，这样我们也是根据行政区代码对应的数据去将地图渲染成不同的颜色即可，这是典型的分层设色的一个应用场景，非常的常规。

第二个需求，行政区要有下钻的功能，那么这个功能的实现通常要和数据进行挂钩了，要想实现行政区的下钻功能，必须清楚的知道行政区的上下级之间的对应关系，比如浙江省

的行政区代码是 330000，那么杭州市的行政区代码是 330100，如果想从浙江省下钻到杭州市，那么在杭州市的数据中就必须有一个字段来记录杭州市的上级行政区是谁，通常我们会使用例如 `parent` 字段来记录，例如在杭州市得数据中 `parent` 包含了 33 表示其上级行政区代码对应的是浙江省，反之当我们从省级往下寻找的时候也是通过这个 `parent` 字段去寻找其下属的行政区。再说每个行政区的中央要显示该行政区内部的土壤检测点的总数数量，这个也是需要业务端同事给我们这份数据，我们渲染到对应的位置上即可。

第三个需求其实是首先要把这些点位加载到地图上，第二步是鼠标悬浮到点位上的时候展示弹窗，这都很简单，在第 6 章中都有讲过，对于各位来讲比较难的事情就是轮播了，这需要大家有一些 js 的基础，会使用 `setInterval` 这样的定时器函数。

第四个需求实际上是我们了解 mapbox 当中的图层加载机制，当你添加相同 `id` 的 `source` 和 `layer` 时，mapbox 会报错提示，已经存在相同 `id` 的 `source` 或者是 `layer`。因此我们如果循环往复的想利用同一个函数加载不同的数据，在这里有两个方案，第一是在添加新的数据之前先把过去的 `source` 和 `layer` 清除掉。第二个是先添加一个空数据的 `layer`。然后再采用 `setData()`方法将数据添加到图层中。

### 3. 实现过程

清楚了每个需求的实现方案之后，我们就进入到开发的阶段，首先确认数据源都准备齐全，然后选择使用什么样的方式开发，在之前讲一些功能的实现逻辑的以后我几乎都只是写了最核心部分的代码，那么现在是项目实战，我们给大家完整的提供源码，我们使用 vue2 来实现上述的需求，原因就不解释了，vue2 是最经典的版本也是适用性友好性最好的版本，vue3 不适合新手，react 学习成本又太高。

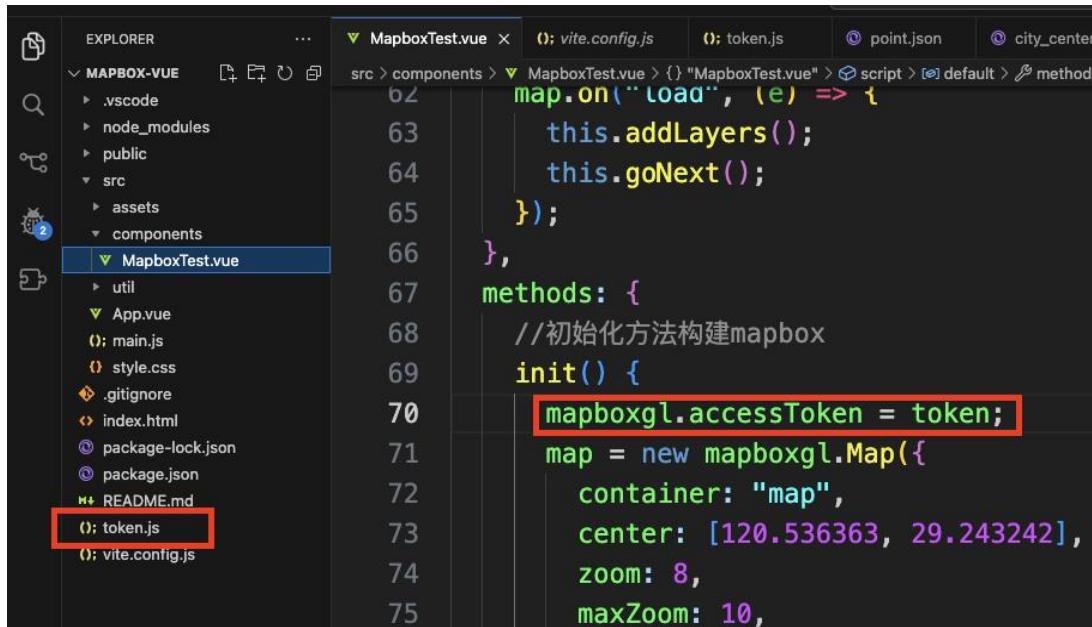
具体的项目工程的搭建大家可以参考上一个案例，过程都是一样的在这里就不重复了。不一样的时候我们要注意引入 mapbox 的依赖库 js 和 css，以及一个空间计算 js 库 turf.js（我们在第九章讲过）。

```
import mapboxgl from "mapbox-gl";
import "mapbox-gl/dist/mapbox-gl.css";
import * as turf from "turf";
```

我们还用到了一个前端常用的 http 请求库 axios：

```
import axios from "axios";
```

剩下的代码部分各位就需要结合上面所讲过的思路参照源代码进行学习了, 建议大家一边调试一边查看结果, 这样学习起来会比较快。另外各位再启动项目之前需要配置自己的 token, 项目工程中有 `token.js` 文件, 里面存放了 token, 大家换成自己的 token 即可启动项目。



```

EXPLORER          ...      MapboxTest.vue x 0; vite.config.js 0; token.js 0; point.json 0; city_center
MAPBOX-VUE        ...
  .vscode
  node_modules
  public
  src
    assets
    components
      MapboxTest.vue
        util
        App.vue
        main.js
        style.css
        .gitignore
        index.html
        package-lock.json
        package.json
        README.md
        0; token.js
        0; vite.config.js
src > components > MapboxTest.vue > {} "MapboxTest.vue" > script > default > methods
62   map.on("load", (e) => {
63     this.addLayers();
64     this.goNext();
65   });
66 },
67 methods: {
68   //初始化方法构建mapbox
69   init() {
70     mapboxgl.accessToken = token;
71     map = new mapboxgl.Map({
72       container: "map",
73       center: [120.536363, 29.243242],
74       zoom: 8,
75       maxZoom: 10,

```

## 4. 思考与延伸

大屏类的项目现如今已经是铺天盖地, 几乎所有的企业都要做大屏, 美其名曰企业数字化。GIS 也成为大屏项目中经常出现的组成元素之一, 现如今看到一个大屏, 如果没有地图, 都会感觉到不适应。我们在大屏开发过程中需要掌握很多的技能。在这个例子中只体现了 20% 的技术内容, 虽然少, 但是很基础, 案例中涉及到的知识点都是必会的。另外各位可能还需要掌握一些例如掩膜、随机散点、切片服务加载, 影像贴图、动态点位等等技巧。随着开发的逐渐深入, 各位还可能会涉及到一些调优问题, 也可能会遇到形形色色的 bug。这都是正常现象。

最后在强调一点就是如果大家使用 `mapbox` 开发, 切记两点: 1.所有操作都必须发生在地图 `load` 成功之后 2.尽量不要把 `map` 对象做成响应式的 `this.map` 这不是一件好事。





## 实战案例三

### 1. 项目背景及需求

上一个案例我们做了常见的大屏的需求，这个案例我们来做一下三维的项目，现在有一个中国南部的某城市需要构建一个三维的数字孪生平台。具体的需求如下：

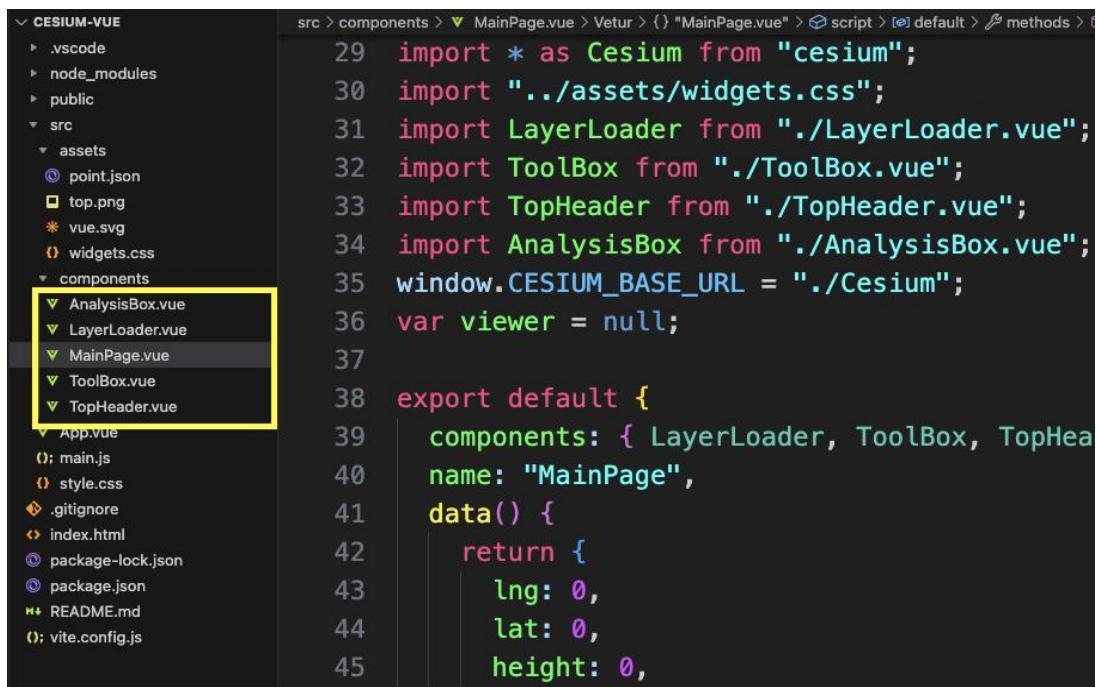
- (1) 平台要有加载基础的地图服务的能力，包括天地图和 geoserver 发布 WMTS 服务，因为客户通常需要城市的高程信息 (DEM 分布)，这种图的格式通常是 tiff，因此需要发布成 wmts 服务进行展示
- (2) 要有能够加载倾斜影像数据的能力，客户还提供了一部分建筑倾斜摄影，需要我们展示。
- (3) 要有能够展示城市关键 POI 点位和点位信息的能力，兴趣点是市中心范围内的一些经常容易出现排水阻塞的位置，需要展示。
- (4) 要有绘制图形的能力，方便标记提供绘制点线面的能力。
- (5) 要有动态预警，标记预警范围的能力，即指定某个位置和区域，再该区域生成预警信息和预警特效在地图上显示。
- (6) 要有一些简单的分析能力，该城市是一个多雨的城市，经常发生洪涝灾害，因此需要提供淹没分析的功能。

## 2. app 设计思路

我们还是逐条需求来分析，首先构建一个三维数字孪生这样的关键词我们就必须使用 **cesium** 了。第一个需求比较基础，就是加载天地图和 **geoserver** 发布的 **wmts** 服务，实际上这是项目中非常常见的客户提供了一些遥感分析的数据，需要我们通过 **geoserver** 发布成切片数据，然后添加到 **cesium** 上进行渲染。在完成是要注意掌握 **wmts** 服务的一些关键参数的加载，第二个需求是加载倾斜影像数据，这也是比较常见的，我们需要吧倾斜影像数据处理成 **3d tiles** 然后加载。要注意把倾斜影响发布成服务在加载，在这个需求中最难的恐怕不是 **cesium** 中 **API** 的使用，而是数据的发布和处理。倾斜摄影数据除了要经过 **cesium lab** 的处理之外还需要借助 **nginx** 发布成为静态资源，并且开发环境中我们还必须解决掉跨域问题。第三个需求要有展示点位和点位信息，这有点像我们上一个案例中的需求，但是这是 **cesium** 中，我们可能要使用不一样的 **api** 和接口来实现此功能。第四个需求，绘制点线面是一个基础的需求，几乎代码都是固定的，类似的还有测量工具。第五个需求，要有动态的预警的功能，这个是要使用到 **cesium** 中的 **entity** 的相关知识，绘制一些类似于雷达特效，炫光特效等，然后指定预警的中心点和预警的影响范围，扩散半径等。要注意其中的难点其实并不来自于 **cesium**，而是动画计算和计时器相关的知识。这一点在其他的 **GIS** 框架中也是类似，凡是跟动画、动态效果有关的，强制要求各位对计时器和 **js** 的动画有一定的理解。第六个需求相对来说上了一些难度，但是也还好，主要用到了 **cesium** 中的地形分析和动态绘制的知识，我们需要借助 **cesium** 中的地形数据，当然也可以自己提供地形数据 (**DEM**) 来做分析需要大家用心掌握和理解。

## 3. 实现过程

**vue** 和 **cesium** 的结合必然会存在很多坑，尤其是使用了新版的脚手架 **vite** 之后，各种问题，另外由于框架本身的不稳定性，开发过程中经常遇到崩溃那也是家常便饭。所以研究 **cesium** 一定要有耐心。在开发的过程中，围绕需求中的功能点，我设计了多个组件，各自去完成一些特定的功能



The screenshot shows the project structure and a portion of the `MainPage.vue` component's code.

```

CESIUM-VUE
├── .vscode
├── node_modules
├── public
└── src
    ├── assets
    │   └── point.json
    ├── components
    │   ├── AnalysisBox.vue
    │   ├── LayerLoader.vue
    │   ├── MainPage.vue (highlighted)
    │   ├── ToolBox.vue
    │   └── TopHeader.vue
    └── App.vue
    └── vite.config.js

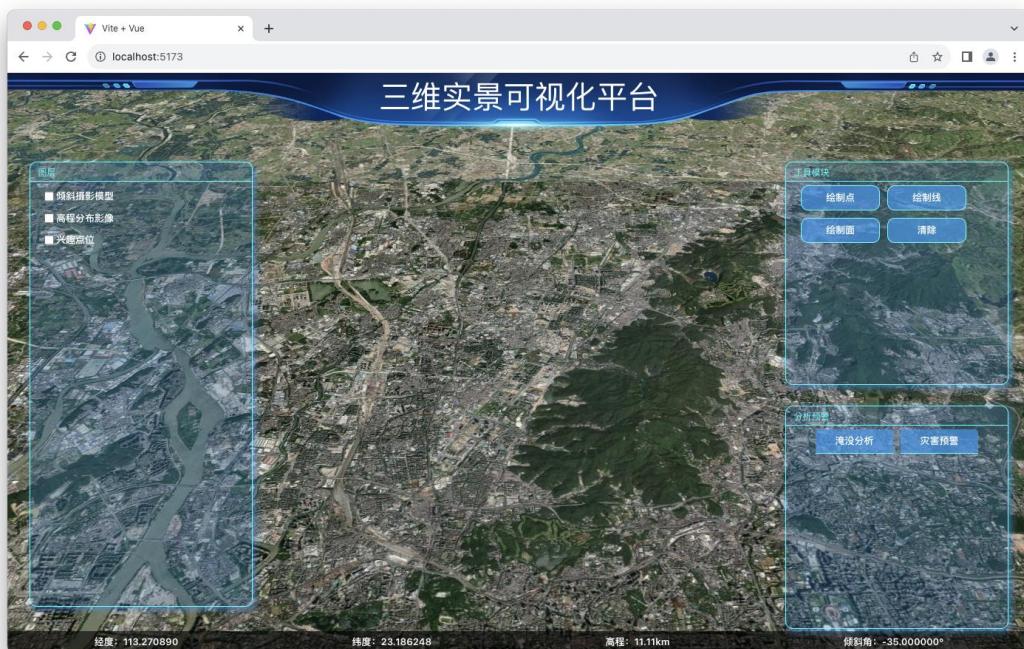
```

```

src > components > MainPage.vue > Vetur > {" MainPage.vue" } > script > default > methods > 
29 import * as Cesium from "cesium";
30 import "../assets/widgets.css";
31 import LayerLoader from "./LayerLoader.vue";
32 import ToolBox from "./ToolBox.vue";
33 import TopHeader from "./TopHeader.vue";
34 import AnalysisBox from "./AnalysisBox.vue";
35 window.CESIUM_BASE_URL = "./Cesium";
36 var viewer = null;
37
38 export default {
39     components: { LayerLoader, ToolBox, TopHeader },
40     name: "MainPage",
41     data() {
42         return {
43             lng: 0,
44             lat: 0,
45             height: 0,

```

其中 `MainPage` 是主页面，也就是 `cesium` 所挂载的页面，同时也是其他组件的父组件，`TopHeader` 是顶部的样式标题组件，`ToolBox` 是绘制点线面的组件，`LayerLoader` 是添加图层的组件，`AnalysisBox` 是分析功能组件。因为最终的渲染还是要发生在主页面，因此绝大多数的组件之间的数据传递我都选择了子组件向父组件传递数据，父组件渲染的方式。



开发过程中需要注意的几个要点如下：

1. 倾斜摄影 3d tiles 数据的发布可以选择 nginx 或者 tomact 发布，这个你的运维同事应该

会帮助你，反正我们访问数据的宗旨只有一个，那就是必须访问到 `tileset.json` 文件。

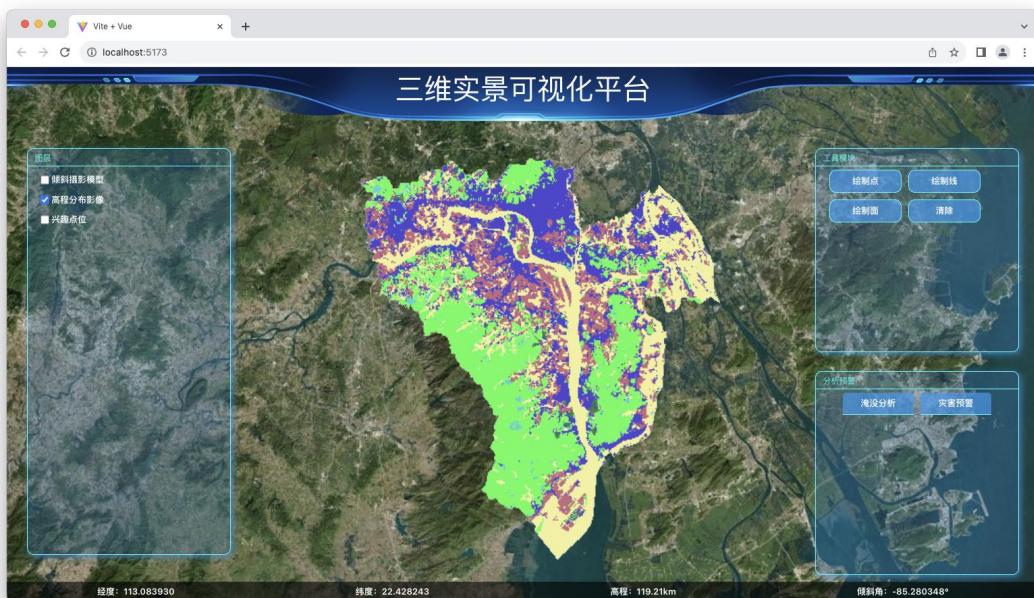
2. 加载城市高程信息分布的遥感影像时可以使用 `geoserver` 先将影像发布成 WMTS 的方式，再使用 `cesium` 加载。这就涉及到了我们第十章第 2 节所讲过的关于 `geoserver` 的知识。
3. 进行淹没分析的时候要注意一定要确保 `cesium` 的地形数据能够正常加载，其实淹没的过程很简单，就是随着高度的不断升高，多边形渲染能够覆盖的部分，因此必须要使用到计时器相关的知识，要么使用 `cesium` 自带的 `ontick` 要么就使用 `setInterval`。
4. 在进行预警渲染时用到了部分着色器的知识。各位如果有兴趣的话可以深入研究，如果没兴趣的话。代码基本上都是固定的，可以直接复用。

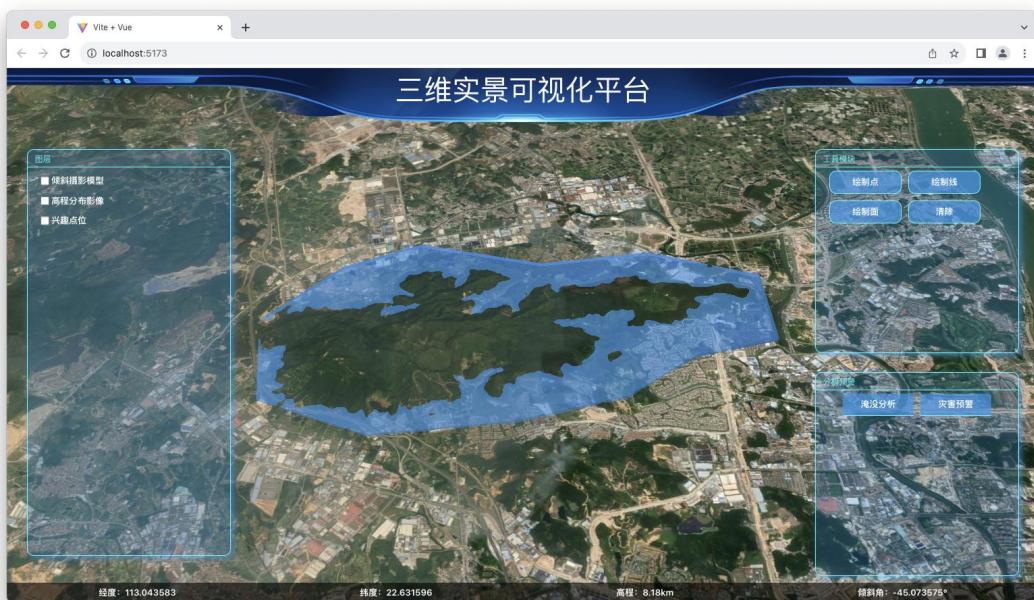
开发过程中所涉及到的所有数据如果体量不是很大我都放在本书所附带的源码文件夹里了。像切好的 `3d tiles` 文件我放在了百度网盘上，链接如下：

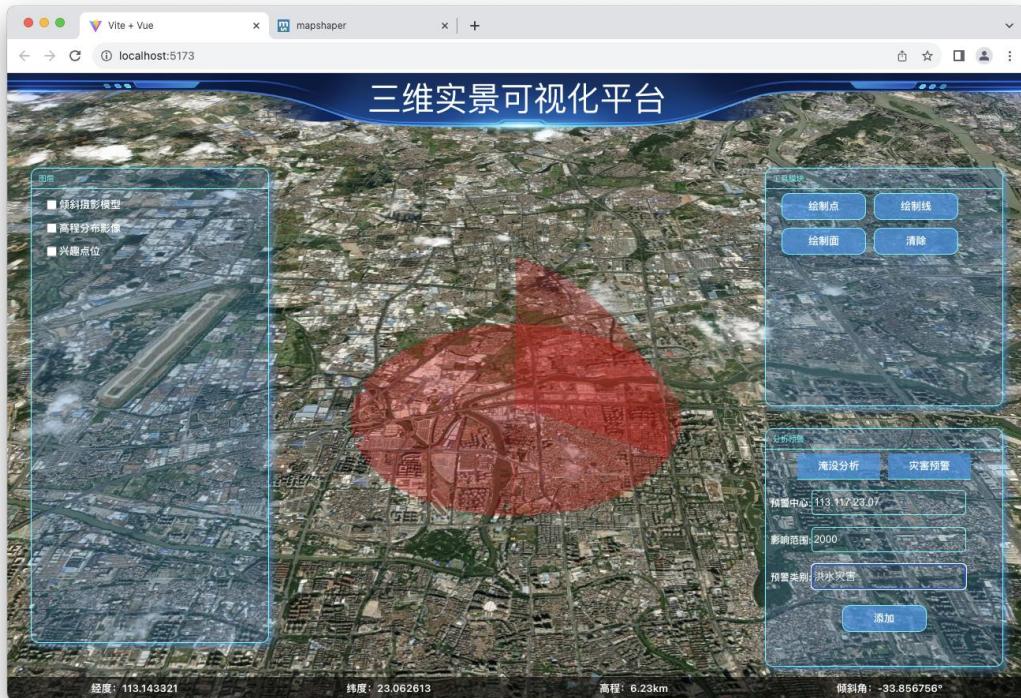
链接：<https://pan.baidu.com/s/1BB2fahTLikhE-SohTmMRuQ>

提取码：my2j

大家下载后解压部署到服务器上即可。







## 4. 思考与延伸

三维 GIS 开发相比于二维来讲确实难了不少，但是不见得难就薪资高，说实话学习三维的 GIS 开发很累很苦很烧脑，我觉得除非是真正的爱好，否则很少有人愿意从事这类开发。现如今数字孪生智慧城市的热度要远大于二维 GIS 所应对的自然资源、农业等项目，这些项目通常的需求其实都很简单，绝大多数的需求其实都是数据的加载，渲染，只要能展示出来就已经解决了很大的问题。例如加载倾斜摄影数据，加载 BIM 建筑模型，加载 gltf 手工模型，加载三维管线模型，等等。加载完成之后其实就是属性的查询，geojson 数据的查询，entity 的查询，以及模型单体化。最后就是一些分析工作，日照分析，以及淹没分析、视域分析等等，这些分析的原理其实都很简单，只不过需要大家花点时间搞懂原理，然后结合 cesium 提供的一些 api 将功能实现即可。

# 常见问题

1. 做开发使用什么编译器比较好？

分前后端和语言，前端作者推荐 vscode，后端推荐 IDEA，如果使 python 推荐 pycharm。

2. WebGIS 应用领域、应用范围很广吗？

请把吗字去掉，可以说是非常广，涉及到自然资源、国土、农业、智慧城市、数字孪生、图形图像、人工智能等各个领域都有 gis 的身影。

3. WebGIS 就业前景好吗？薪资待遇如何？

就业前景相对来说还是不错的，尽管这两年整个互联网行业都很卷，大环境不好，但是 webgis 相对于传统的前端来说还是不错的。

4. 做 gis 倾向于前端好一点还是后端好一点？

看个人喜好，没有绝对的好坏，学好了，哪个都很强！

5. 做三维 gis 开发需要准备些什么？

如果专门做三维要重点读本书的后面 5 章，另外还有要 cesium 相关基础，关于 cesium 可以留意我的直播课或者视频课。

6. 感觉 webgis 入门好难，感觉知识好多好复杂，坚持不下去了怎么办？

这都是正常现象，任何一个刚入门的同学都会有这种感觉，这也正是这个行业的魔力所在，但是一旦听过入门阶段，后面的学习会断崖式轻松，甚至各位都能举一反三自己创新出新的知识和花样。

7. WebGIS 发展方向有哪些？

偏前端的发展天花板几乎是 WebGL 和计算机图形学。偏后端的将来可能会踏入大数据时空数据分析等领域。

## 结语

创作不易，分享更不易。作者本人为了写这本书可以说是呕心沥血，没日没夜。没奢求过同过这本书能够赚多少 money。这本书更大用处在于解决想入门 WebGIS 行业的各位的刚需（毕竟大家觉得直播课和视频课内容过于繁重，价格也比较高）。这本书可能教不会你太多东西，但是应该能给各位很多思想上的启发。也就是常言道：授人鱼不如授人以渔。更多的是希望大家掌握了思想和方法，后面再学习其他的知识的时候能够有自己的一套方法论。书里大白话式的语言应该也能让各位对过去某些困惑的知识有了新的认知。知识从来就不是越多越好，而是在有限的知识里能有无限的思考。本书随着版本迭代的更新也会不断的加入更多的内容，有趣的是，笔者更新书有点像软件升级一样，短期内就会更迭一个版本，新的版本除了解决旧版本的问题之外，还会新增很多实用的知识，像极了系统更新时提供的很多新功能。有的时候我也怀疑自己不是在写书，而是在写操作系统，定期的有大版本和小版本的更新。像极了不断的在解决问题的 IOS。其实任何事物的发展都逃不过其固定规律，一个新生事物的诞生必先经历辉煌，再到平淡，最后被时代所淘汰。趁着目前我们的书籍还处在辉煌的阶段，我也会尽可能的加快版本更迭速度。最后，感谢各位对笔者本人的支持！我从最初发布第一版书籍的时候就承诺过：一次购买，永久免费。所以我也特别欣慰一些最初就关注我支持我的老粉，一直找我不断的置换最新的书籍。这说明我确实给大家带来了微薄的帮助，也说明这本书也在不断的发挥着它的作用。

# 5



WEBGIS小智团队出品

“WebGIS 快速开发教程是一本针对于零基础学者专门打造的一本入门级书籍，它的特点是“轻”、“快”，力求以最快的速度让大家理解 WebGIS 开发过程中的知识，问题，并且以最快的速度进入开发状态，拥有开发的能力。希望大家从此书中能够学到现有认知之外的东西，同时也能够解决各位遇到的问题。”

——作者寄语

