**基本环境搭建**

1 搭建webpack 脚手架和配置webpack.config.js 文件

**搭建初步的webpack 脚手架**

//webpack 局部安装指令: npm install webpack webpack-cli@3 –D

//HtmlWebpackPlugin 安装指令: npm i -s -d html-webpack-plugin

Webpack 是一个模块打包器，它的主要目标是将JavaScript 文件打包在一起，打包后的文件用于在浏览器中使用。

Webpack 最大的用处就是分析一个网页的各种依赖，并且自动化地将这些依赖打包在一起并且压缩，供网页使用。比如Webpack 中的loader，可以将JavaScript ES6(很多老浏览器不一定支持)转换成支持更加多浏览器的老版本JavaScript。

**webpack.config.js 配置：**

入口(entry)，指示webpack 应该使用哪个模块，默认值为./src。

输出(output)，output.path 告诉webpack 在哪里输出它所创建的bundles，以及output.filename 如何命名这些文件，默认值为./dist。

webpack 是模块打包⼯具，⽽模块不仅仅是js，还可以是css，图⽚或者其他格式，但是webpack 默认只知道如何处理js 模块，那么其他格式的模块处理，和处理⽅式就需要loader。

module: {

rules: [{

test: /\.(png|jpe?g|gif)$/,

use: {

loader: "file-loader",

options: {

outputPath: 'img/'

}

}

}]

},

**插件(plugins)HtmlWebpackPlugin**

htmlwebpackplugin 会在打包结束后，⾃动⽣成一个html⽂件，并把打包生成的js模块引⼊到该html中。

2 webpack-dev-server 启动网页和source-Map 方便调试

//生成package.json 指令：npm init –y 生成package.json

//webpack-dev-server 安装指令：npm install webpack-dev-server -D

//源代码与打包后的代码的映射关系:source-Map

"start": "webpack --config webpack.Config.js"

每次改完代码都需要重新打包一次，打开浏览器，很麻烦，可以安装使用webpackdevserver 来改善体验

修改下 package.json "scripts": {

"server": "webpack-dev-server"

},

在 webpack.config.js 配置：

devServer: {

contentBase: "./dist",

open: true,

port: 8089

}

**source-Map源代码与打包后的代码的映射关系**

在 dev 模式中，默认开启，关闭的话 可以在配置⽂件⾥提升开发效率的利器

每次改完代码都需要重新打包⼀次，打开浏览器，刷新⼀次，很麻烦我们可以安装使⽤webpackdevserver 来改善这块的体验

devtool:"none"

//cheap-module-eval-source-map 开发环境配置推荐

//cheap-module-source-map 线上环境配置推荐

//devtool: "source-map", //方便寻找错误，可以快速定位到错误发生位置devtool: "cheap-module-eval-source-map",

3 区分模式打包，Config 配置文件区分开发和生产环境

共用配置代码webpack.common.js, 开发环境配置代码webpack.dev.js,生产环境，上线后的配置代码webpack.prod.js

devlopment vs Production 区分模式打包 npm install webpack-merge -D

4 通过 Babel 把 ES6 转换为 ES5 实现浏览器兼容

Babel 是一个广泛使用的转码器，babel 可以将 ES6 代码完美地转换为ES5 代码，所以我们不用等到浏览器的支持就可以在项目中使用 ES6 的特性。

npm i babel-loader @babel/core @babel/preset-env -D

babel-loader 是 webpack 与 babel 的通信桥梁，不会做把 es6 转成 es5 的⼯作，这部分⼯作需要⽤到@babel/preset-env 来做

@babel/preset-env⾥包含了 es6 转 es5 的转换规则

**webpack - babel 配置**

babel 是一个 javascript 编译器，是前端开发中的一个利器。它突破了浏览器实现 es 标准的限制，使我们在开发中可以使用最新的 javascript 语法。

通过构建 babel，可以使用最新 js 语法进行开发，最后自动编译成用于浏览器的代码。

**Three.js 3D地图门户绘制**

1 导入 threejs 库及 threejs 常用的插件

安装轨道控制插件:npm install three-orbit-controls

插件引入之前必须确认 three.js 库已经引入

OrbitControls = require('three-orbit-controls')(THREE)

使用：controls = new OrbitControls(camera);

安装加载.obj 和.mtl 文件的插件 npm i --save three-obj-mtl-loader

插件引入之前必须确认 three.js 库已经引入

该插件包括加载.obj 和.mtl 文件的加载器

import {MTLLoader,OBJLoader} from 'three-obj-mtl-loader';

使用： mtlLoader = new MTLLoader(); objLoader = new OBJLoader();

安装 three-css2drender 插件， npm i --save three-css2drender

插件引入之前必须确认 three.js 库已经引入

import {CSS2DRenderer,CSS2DObject} from 'three-css2drenderer';

使用：labelRenderer = new CSS2DRenderer(); label = new CSS2DObject( text );

**基本环境搭建完成后package.json**

"dependencies": {

"@babel/polyfill": "^7.7.0",

"@babel/runtime": "^7.7.7",

"axios": "^0.19.0",

"d3": "^5.15.0",

"file-loader": "^5.0.2",

"html-webpack-plugin": "^3.2.0",

"jquery": "^3.4.1",

"three": "^0.112.1",

"three-css2drender": "^1.0.0",

"three-obj-mtl-loader": "^1.0.3",

"three-orbit-controls": "^82.1.0"

},

"devDependencies": {

"@babel/core": "^7.7.7",

"@babel/plugin-transform-runtime": "^7.7.6",

"@babel/preset-env": "^7.7.7",

"babel-loader": "^8.0.6",

"webpack": "^4.46.0",

"webpack-cli": "^3.3.12",

"webpack-dev-server": "^3.11.2",

"webpack-merge": "^4.2.2"

},

## 2 通过 jquery 和 axios 在界面中引入 geojson 数据

import $ from 'jquery';

import axios from 'axios';

import { decrypt } from './dist/decrypt';

import ThreeMap from './ThreeMap';

import ThreeMapLightBar from './ThreeMapLightBar';

$.get('/map/china.json', data => {

const mapData = decrypt.decode(data);

//console.log(data);

const map = new ThreeMapLightBar(decrypt.decode(data));

map.drawLightBar(lightBarData);

//const map2 = new ThreeMap(data);

})

3 geoJson 及点线面等在线效果

json 是通过键值对表示数据对象的一种格式，可以很好地表达数据，json 可以应用的范围很广，不止于前端，非常轻量、容易解析。geojson 是是 json 的子集，用 json 的语法表达和存储地理数据。

"type": "FeatureCollection",

"features": [

{"type":"Feature",

"properties":{},

"geometry":{

"type":"Point",

"coordinates":[105.380859375,31.57853542647338]

}

}

]

}

geojson 将所有的地理要素分为Point、MultiPoint、LineString、MultiLineString、Polygon、MultiPolygon、GeometryCollection。首先是将这些要素封装到单个的 geometry 里，然后作为一个个的Feature（也就是要素）；要素放到一个要素集合里，从树状结构来理解 FeatureCollection 就是根节点

{

"type": "FeatureCollection",

"features": []

}

**点要素 Point**

点要素是最简单的，类型type 对应 Point，然后坐标是一个 1 维的数组，里面有两个元素（如果是立体的坐标就是三维 x,y,z），分别为经度和纬度。properties 里面可以封装各种属性，例如名称、标识颜色等等。

{"type":"Feature",

"properties":{},

"geometry":{

"type":"Point",

"coordinates":[105.380859375,31.57853542647338]

}

}

**多点要素 MultiPoint**

{"type":"Feature",

"properties":{},

"geometry":{

"type":"MultiPoint",

"coordinates":[[105.380859375,31.57853542647338],

[105.580859375,31.52853542647338]

]

}

}

4 墨卡托坐标及经纬度转换三维坐标方法实现

**墨卡托坐标介绍**

Google Maps、Virtual Earth 等网络地理所使用的地图投影，常被称作 Web Mercator 或 Spherical Mercator，它与常规墨卡托投影的主要区别就是把地球模拟为球体而非椭球体。

//经纬度转三维坐标

lnglatToVector(lnglat) {

if (!this.projection) {

this.projection = d3

.geoMercator() //获取墨卡托坐标方法

.center([112.946332, 28.236672])

.scale(60)

//.rotate(Math.PI / 4)

.translate([0, 0]);

}

//const projection = d3.geoMercator().center([108.904496, 32.668849]).scale(80);

const [y, x] = this.projection([...lnglat]);

let z = 0;

return [y, x, z];

}

5 用 geoJson 数据在场景中绘制中国地图

drawMap() {

//console.log(this.mapData)

//console.log(this.mapData);

this.vector3Json = [];

this.mapData.features.forEach(element => {

const areas = element.geometry.coordinates[0];

//es6解构 ...data

//var a = { name: '张三', age: 25 } var b = {job: 'web前端',...a}

// 相当于 var c = {job: 'web前端',name: '张三',age: 25}

const areaData = {...element.properties, coordinates: [] };

//通过循环，区分坐标或数组

areas.forEach((area, i) => {

//如果是数组

if (area[0] instanceof Array) {

areaData.coordinates[i] = [];

area.forEach(areaInner => {

//console.log(pointInner);

areaData.coordinates[i].push(this.lnglatToVector(areaInner))

})

} else {

areaData.coordinates.push(this.lnglatToVector(area))

}

})

this.vector3Json[element.properties.name] = areaData;

this.vector3Json.push(areaData);

});

//console.log('vector3json', this.vector3Json)

//绘制模块

const group = new THREE.Group();

const lineGroup = new THREE.Line();

this.vector3Json.forEach(provinces => {

//var a=[[1,2,3],[4,5,6]]

//访问：a[0][0]=1,a[1][2]=6 （起始值0）

if (provinces.coordinates[0][0] instanceof Array) {

provinces.coordinates.forEach(area => {

const mesh = this.getAreaMesh(area);

group.add(mesh);

const line = this.drawLine(area);

lineGroup.add(line);

})

} else { //单面

//console.log(provinces.coordinates)

const mesh = this.getAreaMesh(provinces.coordinates);

group.add(mesh);

const line = this.drawLine(provinces.coordinates);

lineGroup.add(line);

}

});

this.group = group;

group.rotation.y = Math.PI;

lineGroup.rotation.y = Math.PI;

this.scene.add(group);

this.scene.add(lineGroup);

}



6 光线投射Raycaster 鼠标点击地图中某个区域时改变该区域的颜色

**光线投射 Raycaster**

这个类用于进行 raycasting（光线投射）。 光线投射用于进行鼠标拾取（在三维空间中计算出鼠标移过了什么物体）。

mouseEvent(event) {

if (!this.raycaster)

this.raycaster = new THREE.Raycaster();

if (!this.mouse)

this.mouse = new THREE.Vector2();

// 将鼠标位置归一化为设备坐标。x 和 y 方向的取值范围是 (-1 to +1)

this.mouse.x = (event.clientX / window.innerWidth) \* 2 - 1;

this.mouse.y = -(event.clientY / window.innerHeight) \* 2 + 1;

// 通过摄像机和鼠标位置更新射线

this.raycaster.setFromCamera(this.mouse, this.camera);

// 计算物体和射线的焦点

const intersects = this.raycaster.intersectObjects(this.group.children);

this.group.children.forEach(mesh => {

mesh.material.color.set('#005fc3');

})

for (var i = 0; i < intersects.length; i++) {

intersects[i].object.material.color.set(0xff0000);

}

7 地图区域内中心点生成光柱特效和光柱底盘绘制

drawLightBar(data) {

const group = new THREE.Group();

const texture = new THREE.TextureLoader().load(pic1);

//texture.rotation.x = Math.PI;

data.forEach((d, i) => {

//console.log(this.vector3Json[d.name]);

const { cp } = this.vector3Json[d.name];

const [x, y, z] = this.lnglatToVector(cp);

var geomentry = new THREE.PlaneGeometry(1, d.value / 5);

var material = new THREE.MeshBasicMaterial({

//map: texture,

map: this.textures[i % 2], ////颜色贴图

color: '#ffff00',

transparent: true,

opacity: 0.7,

depthTest: false, //深度测试属性

blending: THREE.AdditiveBlending, //滤镜选择

side: THREE.DoubleSide

})

var plane = new THREE.Mesh(geomentry, material);

plane.position.set(x, y, -(z + d.value / 5 / 2));

plane.rotation.x = Math.PI / 2;

group.add(plane);

var plane2 = plane.clone();

plane2.rotation.y = Math.PI / 2;

group.add(plane2);

group.add(this.addButtomPlate([x, y, z]));

});

group.rotation.y = Math.PI;

this.scene.add(group);

}



**部署上线后的测试环境**

采用docker来部署整个项目，云端服务器系统为Ubuntu20.04，docker采用CentOS 8.3.2011(Py3.7.9)镜像作为容器，借助了宝塔Linux面板作为环境的搭建工具。

项目运行环境搭建如下：

WebServer: Nginx 1.18.0 Apache Tomcat 9.0.0.M18

数据库: MySQL 5.6.50

MySQL管理工具: phpMyAdmin 4.9

PHP版本: PHP-7.2

文件服务器: Pure-Ftpd 1.0.49

其他工具/插件: Java项目一键部署 1.0

服务端框架: node.js v15.10.0 npm 7.6.0

Web3D技术： Three.js R112 D3.js 5.15.0