

Leaflet.js 入门教程

介绍

概述

本文将讲述在 vue.js 的工程中结合 leaflet.js 开发地图相关的功能，通过自己在实际工作中所学习到开发经验分享给读者，总结归纳出常见的地图功能的实现示例，仅供读者参考。Vue.js 是时下流行的 MVVM 的前端框架，简单易学。Leaflet 是一个为建设移动设备友好的互动地图，而开发的现代的、开源的 JavaScript 库。它具有开发人员开发在线地图的大部分功能。

环境搭建

开发机器：window10.

开发工具：webStorm

1、node.js 安装。 参考官网 <https://nodejs.org/en/>

2、vue-cli 标准工具必须安装。

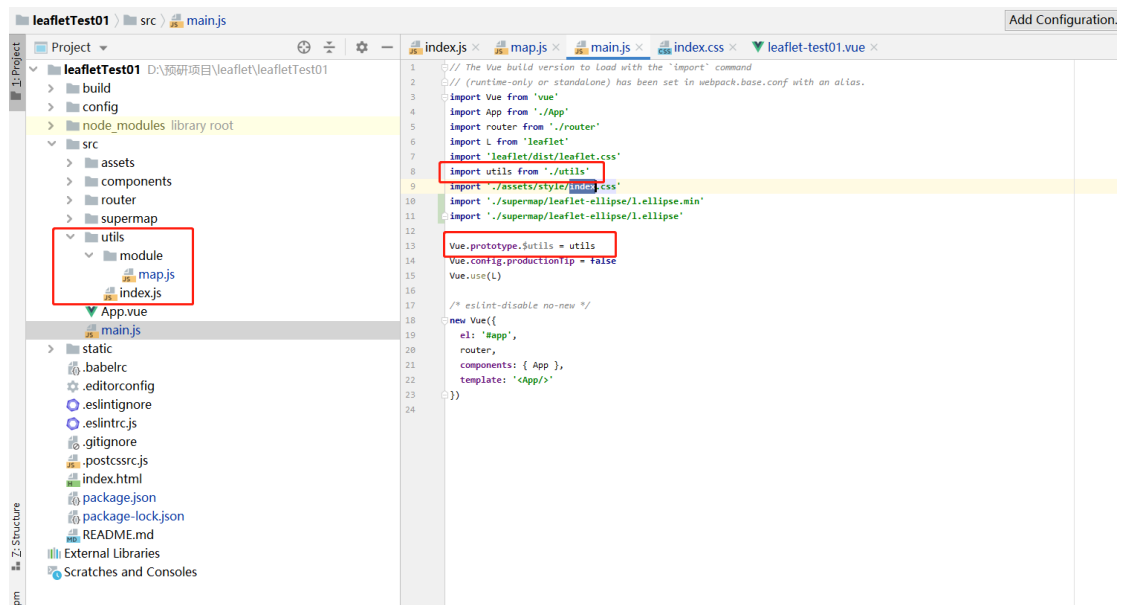
Npm install -g @vue/cli

创建 Vue-CLI 工程

Vue-CLI 安装完成之后，创建一个 Vue 工程，运行一下命令。

Vue create leafletTest01

在项目中创建一个工具集模块，同时修改 main.js 中的引用，将工程中的工具集模块挂载在 Vue 对象上：



在 vue 中安装 leaflet

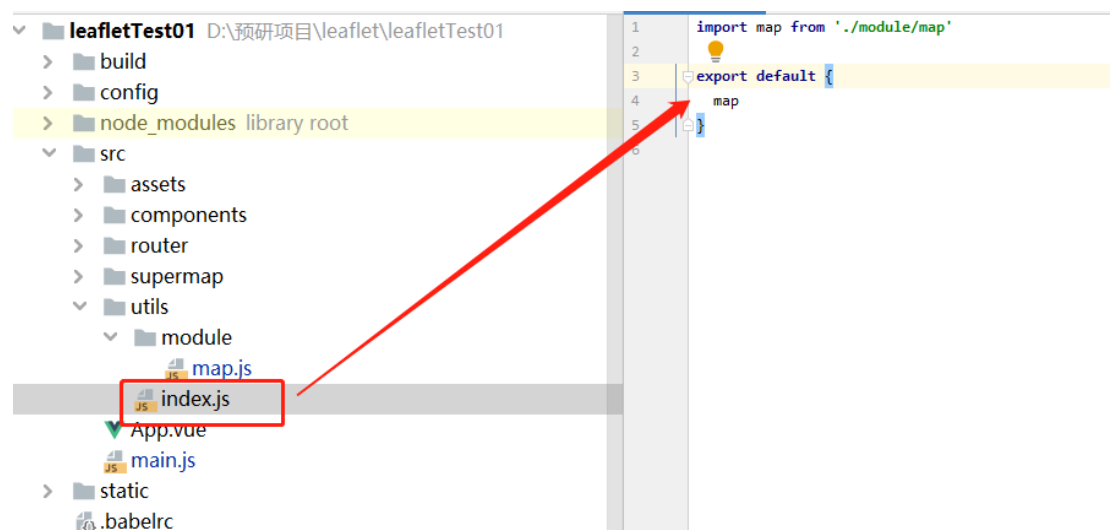
npm install leaflet --save

leaflet 包安装成功之后，需要在 Vue 工程中将它引入至工程。将 leaflet 作品为地图模块的工具单独封装一个模块里 map.js。这样在 map.js 维护可以使代码更加简洁，提高代码的易读性，可维护性。

注意：引用 leaflets 的时候需要同时引入 leaflet.css 样式。



同时将地图模块挂载在工具集中：



显示一个地图

1、创建地图对象

地图对象相对容以地理资源的容器。而地理资源则包括了在地图视图窗口中看到的所有内容，如常见地图服务资源，地理图形（点，线，面），以及地图控件等多种元素。创建的方法也很简单。在工具模块中地图模块 map.js 添加创建地图对象的方法如下：

```

ex.js ×  JS map.js ×  JS main.js ×  CSS index.css ×  leaflet-
import 'leaflet/dist/leaflet.css'
import $L from 'leaflet'
import { chinaProvider } from 'leaflet.chinatmsproviders'
import L from 'leaflet'

// 创建地图对象
const createMap = (divId, options) => {
  let map = $L.map(divId, options)
  L.ellipse([31.52, 117.17], [500, 100], 90).addTo(map)
  return map
}

```

地图对象创建成功后，创建地图视图窗口//src/components/leaflet-test01.vue:

```

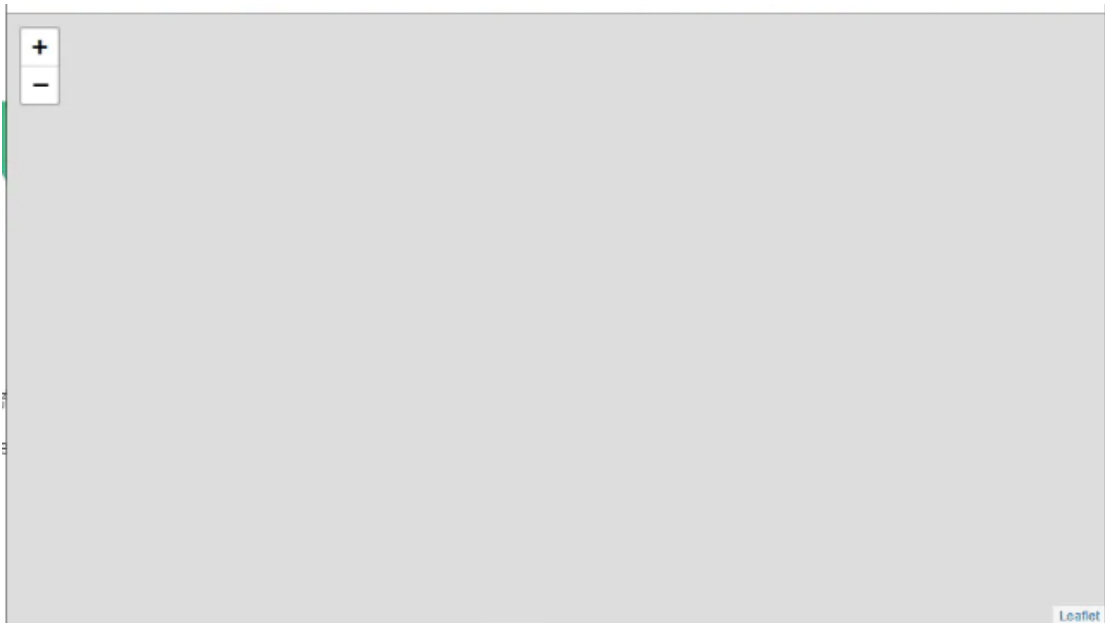
<template>
  <div class="map-container" id="map-container"></div>
</template>

<script>
export default {
  name: "map",
  components: {},
  data(){
    return {
      map: null
    }
  },
  mounted() {
    this.map = this.$utils.map.createMap("map-container");
  }
};
</script>

<style lang="less">
.map-container {
  position: absolute;
  left: 0;
  top: 0;
  width: 100%;
  height: 100%;
}
</style>

```

此时加载页面



2、加载地图服务

有了地图容器之后，并不代表就可以看到地图。地图视图需要添加图层或图形才会有内容显示。

首先在 `map.js` 中添加图层加载的方法。需要注意一下，`leaflet` 中创建图层的方法是异步方法，所以这里用到了 `async` 和 `await` 来处理异步的问题。

```
// 加载地图服务图层
const createTileLayer = async (map, url, options) => {
  let tileLayer = await $L.tileLayer(url, options)
  tileLayer.addTo(map)
  return tileLayer
}
```

接着，再次地图页面调用该方法显示地图 `//src/components/leaflet-test01.vue`

```

<template>
  <div class="map-container" id="map-container"></div>
</template>

<script>
export default {
  name: "mapView",
  components: {},
  data() {
    return {
      map: null,
      OSMUrl: "https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
    };
  },
  mounted() {
    this.map = this.$utils.map.createMap("map-container");

    // 加载 open street map 图层服务
    this.$utils.map.createTileLayer(this.map, this.OSMUrl, {});

    // 设施地图视图 中心位置
    this.map.setView([51.505, -0.09], 13);
  }
};
</script>
<style lang="less">
.map-container {
  position: absolute;
  left: 0;
  top: 0;
  width: 100%;
  height: 100%;
}
</style>

```

定位到中国地图

```

// 设施地图视图 中心位置
this.map.setView([34.3227, 108.5525], 5)

```

目前便会在浏览器中显示一个地图了。

地图基本操作

1、功能分析

地图加载成功之后，接下来要开始对地图的常见功能的实现，一个地图的基本功能包括：地图显示、平移、放大、缩小、定位等功能。

2、实现思路

1) 平移

通常 webGIS 中地图平移是最为基本且常用的功能，地图回默认开启平移功能。通常情况下都不需要开发者自己去实现平移的功能。

2) 放大与缩小

放大、缩小也一样，地图通常会默认开启此项功能。而大多数情况下，webGIS 库会提供一些自带的控件来实现一级一级地放大的功能。因此，放大与缩小实现的办法有两种。

1、自带控件

```
L.map('map', {  
  zoomControl: true,  
  scrollWheelZoom: true //默认开启鼠标滚轮缩放  
});
```

2、通过地图的 API 实现

```
// 逐级放大, delta 默认 1  
map.zoomIn( ?delta )  
  
// 逐级缩小, delta 默认 1  
map.zoomOut( ?delta )
```

编辑地图

添加 marker, polyline, polygon, circle


一、功能分析

WebGIS 中添加标记或者图形是很常见的功能，如查询结果的图文联动，定位标记点，点图查询等都以此功能展开。而添加 graphics 是一切图形显示、绘制，编辑等功能的基础。

二、代码实现

添加 marker

Leaflet 中 marker 对象构造较为简单，可在官网提供的 API 文档中可以看到主要构造参数：

Options			
Option	Type	Default	Description
 icon	Icon	*	Icon instance to use for rendering the marker. See icon documentation for details on how to customize the marker icon. If not specified, a common instance of L.Icon.Default is used.
keyboard	Boolean	true	Whether the marker can be tabbed to with a keyboard and clicked by pressing enter.
title	String	''	Text for the browser tooltip that appear on marker hover (no tooltip by default).
alt	String	''	Text for the alt attribute of the icon image (useful for accessibility).
zIndexOffset	Number	0	By default, marker images zIndex is set automatically based on its latitude. Use this option if you want to put the marker on top of all others (or below), specifying a high value like 1000 (or high negative value, respectively).
opacity	Number	1.0	The opacity of the marker.
riseOnHover	Boolean	false	If true, the marker will get on top of others when you hover the mouse over it.
riseOffset	Number	250	The z-index offset used for the riseOnHover feature.
pane	String	'markerPane'	Map pane where the markers icon will be added.
bubblingMouseEvents	Boolean	false	When true, a mouse event on this marker will trigger the same event on the map (unless L.DomEvent.stopPropagation is used).

其中最主要的较为重要的参数为 icon 是显示结果的关键参数，iconUrl

Options

Option	Type	Default	Description
iconUrl	String	null	(required) The URL to the icon image (absolute or relative to your script path).
iconRetinaUrl	String	null	The URL to a retina sized version of the icon image (absolute or relative to your script path). Used for Retina screen devices.
iconSize	Point	null	Size of the icon image in pixels.
iconAnchor	Point	null	The coordinates of the "tip" of the icon (relative to its top left corner). The icon will be aligned so that this point is at the marker's geographical location. Centered by default if size is specified, also can be set in CSS with negative margins.
popupAnchor	Point	[0, 0]	The coordinates of the point from which popups will "open", relative to the icon anchor.
tooltipAnchor	Point	[0, 0]	The coordinates of the point from which tooltips will "open", relative to the icon anchor.
shadowUrl	String	null	The URL to the icon shadow image. If not specified, no shadow image will be created.
shadowRetinaUrl	String	null	
shadowSize	Point	null	Size of the shadow image in pixels.
shadowAnchor	Point	null	The coordinates of the "tip" of the shadow (relative to its top left corner) (the same as iconAnchor if not specified).
className	String	''	A custom class name to assign to both icon and shadow images. Empty by default.

Icon 支持的图片格式，支持常用的.jpg .png .gif
在 map.js 分别添加 icon 和 maker 的两个创建方法

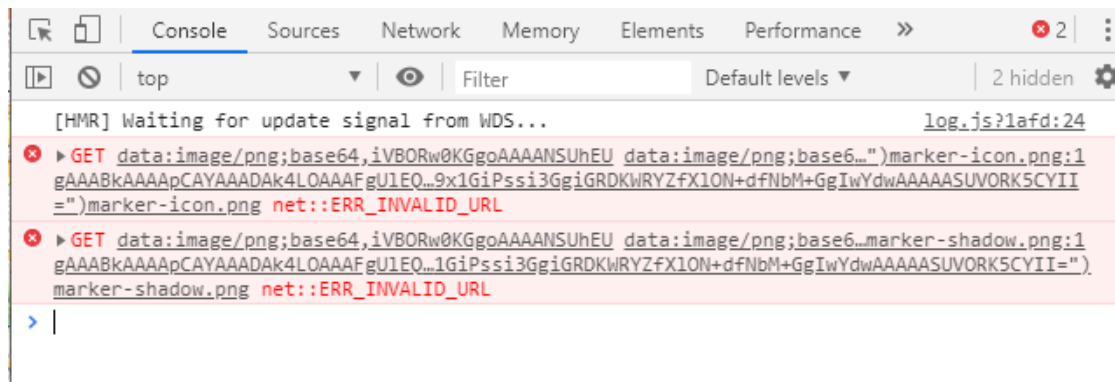
```
const createIcon = options => {
  return $L.icon(options);
};

/**
 * 通过 [x, y] 坐标添加 Maker
 *
 * @param {Object} map
 * @param {Object} latLng
 * @param {Object} options
 *
 */
const createMakerByXY = (map, coordinate, options = {}) => {
  let marker = $L.marker($L.latLng(coordinate[1], coordinate[0s]), options);
  marker.addTo(map);
  return marker;
};
```

在地图页面创建 maker 并显示在地图上//src/components/leaflet-test01.vue

```
addMarker() {
  this.$utils.map.createMakerByXY(this.map, [51.505, -0.09]);
},
```


这里添加 `marker` 方法默认是可以不传入任何参数，`leaflet` 会加载其默认的 `marker` 样式。但是如果我们直接调用刚刚写的 `marker` 方法，看不到地图视图中有 `marker` 显示，在控制台中会出现某一个路径错误。其实 `leaflet` 默认的 `icon` 的图片路径不对。倒是 `leaflet` 加载默认 `marker` 失败。



解决方法如下，添加到 `map.js`

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

// 解决 leaflet 默认 maker 无法显示的问题
import icon from "leaflet/dist/images/marker-icon.png";
import iconShadow from "leaflet/dist/images/marker-shadow.png";
let DefaultIcon = $L.icon({
  iconUrl: icon,
  shadowUrl: iconShadow
});
$L.Marker.prototype.options.icon = DefaultIcon;
```

自定义 `marker` 样式，则先创建不同的 `icon`，修改一下地图页面 `//src/components/leaflet-test01.vue` 中添加 `addMarker` 方法

```


addMaker() {
  this.$utils.map.createMakerByXY(this.map, [-0.09, 51.505]);

  let gifIcon = this.$utils.map.createIcon({
    imageUrl: require("../assets/images/sample.gif"),
    iconSize: [32, 32]
  });
  this.$utils.map.createMakerByXY(this.map, [-0.095, 51.505], {
    icon: gifIcon
  });

  let pngJpgIcon = this.$utils.map.createIcon({
    imageUrl: require("../assets/images/tree.png"),
    iconSize: [52, 42]
  });
  this.$utils.map.createMakerByXY(this.map, [-0.09, 51.490], {
    icon: pngJpgIcon
  });
}

```

添加 polyline

Creation	
Factory	Description
 L.polyline(<LatLng[]> <i>LatLngs</i> , <Polyline options> <i>options?</i>)	Instantiates a polyline object given an array of geographical points and optionally an options object. You can create a Polyline object with multiple separate lines (MultiPolyline) by passing an array of arrays of geographic points .

Leaflet 中构造线要素，面要素中关键的是弄清楚其构造参数中的第一个参数。第一个参数，决定了要素的形状位置，是一个坐标数组。这个坐标数组是也可以包含多个要素组成复合多线要素或是复合多边形。

Polyline

A class for drawing polyline overlays on a map. Extends [Path](#).


Usage example

```
// create a red polyline from an array of LatLng points
var latlngs = [
  [45.51, -122.68],
  [37.77, -122.43],
  [34.04, -118.2]
];
var polyline = L.polyline(latlngs, {color: 'red'}).addTo(map);
// zoom the map to the polyline
map.fitBounds(polyline.getBounds());
```

You can also pass a multi-dimensional array to represent a [MultiPolyline](#) shape:

```
// create a red polyline from an array of arrays of LatLng points
var latlngs = [
  [[45.51, -122.68],
  [37.77, -122.43],
  [34.04, -118.2]],
  [[40.78, -73.91],
  [41.83, -87.62],
  [32.76, -96.72]]
];
```

Creation

Factory	Description
 <code>L.polyline(<LatLng[]> latlngs, <Polyline options> options?)</code>	Instantiates a polyline object given an array of geographical points and optionally an options object. You can create a Polyline object with multiple separate lines (MultiPolyline) by passing an array of arrays of geographic points.

Options

Option	Type	Default	Description
<code>smoothFactor</code>	Number	1.0	How much to simplify the polyline on each zoom level. More means better performance and smoother look, and less means more accurate representation.
<code>noClip</code>	Boolean	false	Disable polyline clipping.

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 shape to be used at the end of the stroke.
lineJoin	String	'round'	A string that defines shape to be used at the corners of the stroke.
dashArray	String	null	A string that defines the stroke dash pattern . Doesn't work on Canvas -powered layers in some old browsers .
dashOffset	String	null	A string that defines the distance into the dash pattern to start the dash . Doesn't work on Canvas -powered layers in some old browsers .
fill	Boolean	depends	Whether to fill the path with color. Set it to <code>false</code> to disable filling on polygons or circles.
fillColor	String	*	Fill color. Defaults to the value of the color option
fillOpacity	Number	0.2	Fill opacity.
fillRule	String	'evenodd'	A string that defines how the inside of a shape is determined.
bubblingMouseEvents	Boolean	true	When true, a mouse event on this path will trigger the same event on the map (unless L.DomEvent.stopPropagation is used).
renderer	Renderer		Use this specific instance of Renderer for this path. Takes precedence over the map's default renderer .
className	String	null	Custom class name set on an element. Only for SVG renderer.

在 `map.js` 中添加创建线要素的方法

```
/**
 * 创建线要素
 *
 * @param {Object} map
 * @param {Array} linePath
 * @param {Object} lineOpts
 */

const createPolyline = (map, linePath, lineOpts) => {
  let polyline = $L.polyline(linePath, lineOpts);
  polyline.addTo(map);
  return polyline;
};
```

在地图页面/`src/components/leaflet-test01.vue` 添加 `data` 存放创建线的坐标数组

```

data() {
  return {
    singleLine: [
      [51.517288954651875, -0.16685485839843753],
      [51.51194758264939, -0.1474571228027344],
      [51.51878442657495, -0.13320922851562503],
      [51.530426064343594, -0.1419639587402344],
      [51.53640593191922, -0.11209487915039064]
    ],
    multipleLine: [
      [
        [51.49282033577651, -0.11432647705078126],
        [51.48010001366223, -0.10265350341796876],
        [51.48084836613703, -0.08222579956054689],
        [51.49591970845512, -0.08239746093750001]
      ],
      [
        [51.47614423230301, -0.08909225463867188],
        [51.47571655971428, -0.05973815917968751],
        [51.4829864484029, -0.03398895263671876],
        [51.49025517833079, -0.008239746093750002],
        [51.477641054786694, 0.008583068847656252],
        [51.487796767005534, 0.021800994873046875]
      ]
    ]
  };
}

```

在地图页面/src/components/leaflet-test01.vue 的 method 存放创建线要素的方法

```

methods: {
  addPolyline() {
    let singleLineStyle = {
      stroke: true,
      color: "#de0000",
      weight: 3,
      opacity: 1
    };

    this.$utils.map.createPolyline(
      this.map,
      this.singleLine,
      singleLineStyle
    );

    let multipleLineStyle = {
      stroke: true,
      color: "#0085fb",
      weight: 4,
      opacity: 1
    };
    this.$utils.map.createPolyline(
      this.map,
      this.miltipleLine,
      multipleLineStyle
    );
  }
}

```

运行



添加 polygon

Creation	
Factory	Description
<code>L.polygon(<LatLng[]> latlngs, <Polyline_options> options?)</code>	

Polygon 的创建和 polyline 的创建方法几乎是一样的，包括坐标串的组织方式也基本一样。
在 map.js 中添加创建面要素的方法

```
/**
 * 创建面要素
 *
 * @param {Object} map
 * @param {Array} areaPath
 * @param {Object} areaOpts
 */

const createPolygon = (map, areaPath, areaOpts) => {
  let polygon = $L.polyline(areaPath, areaOpts);
  polygon.addTo(map);
  return polygon;
};
```

在地图页面/src/components/leaflet-test01.vue 的 method 存放创建面要素的方法，下面为坐标串的组织方式。

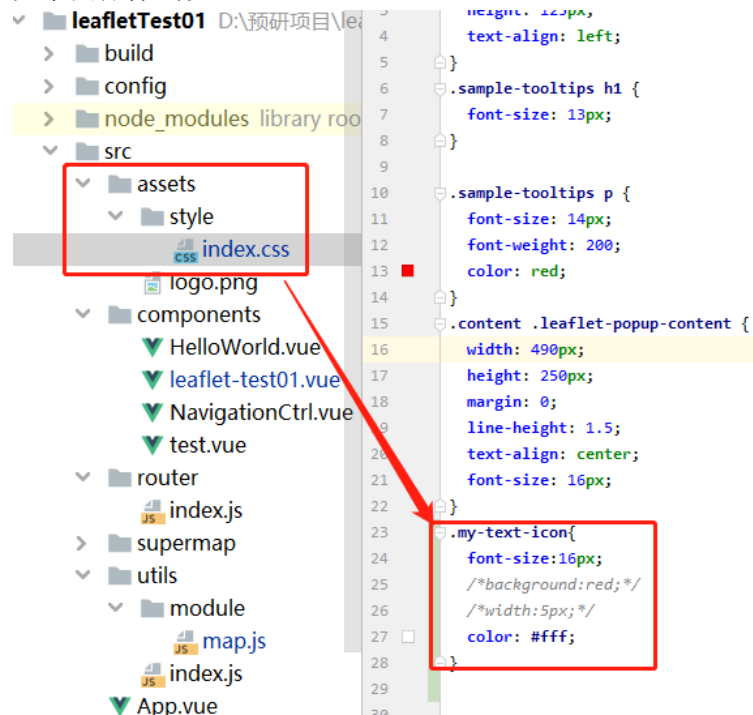
```
data() {  
  return {  
    singlePolygon: [  
      [51.50203767899114, -0.13977527618408206],  
      [51.505777518488806, -0.13072013854980472],  
      [51.505109712517786, -0.1296043395996094],  
      [51.50388092395907, -0.12921810150146487],  
      [51.50345351147583, -0.12921810150146487],  
      [51.50302609498369, -0.12947559356689456],  
      [51.502545246638114, -0.12973308563232425],  
      [51.50219796412198, -0.12990474700927737],  
      [51.50177053585362, -0.12990474700927737],  
      [51.5014232474337, -0.12999057769775393],  
      [51.50043479667606, -0.13891696929931643],  
      [51.50134310357634, -0.1399040222167969],  
      [51.50195753621433, -0.13973236083984378],  
      [51.50195753621433, -0.13973236083984378]  
    ],  
    multiplePolygon: [  
      [  
        [51.481703611072156, -0.09407043457031251],  
        [51.480313829908056, -0.09080886840820312],  
        [51.481703611072156, -0.08531570434570314],  
        [51.482131227525315, -0.07415771484375001],  
        [51.48394855271953, -0.07415771484375001],  
        [51.48426924964768, -0.07638931274414064],  
        [51.486941636341456, -0.07604598999023438],  
        [51.485552014806856, -0.07947921752929689],  
        [51.48426924964768, -0.0830841064453125],  
        [51.48320025111633, -0.08754730224609376],  
        [51.4826657424533, -0.08943557739257814],  
        [51.481489801341986, -0.09441375732421875],  
        [51.481489801341986, -0.09441375732421875]  
      ],  
      [  
        [51.49869827721546, -0.05613327026367188],  
        [51.498377681772325, -0.05922317504882813],  
        [51.49506473014368, -0.05802154541015626],  
        [51.49132401147376, -0.05407333374023438],
```



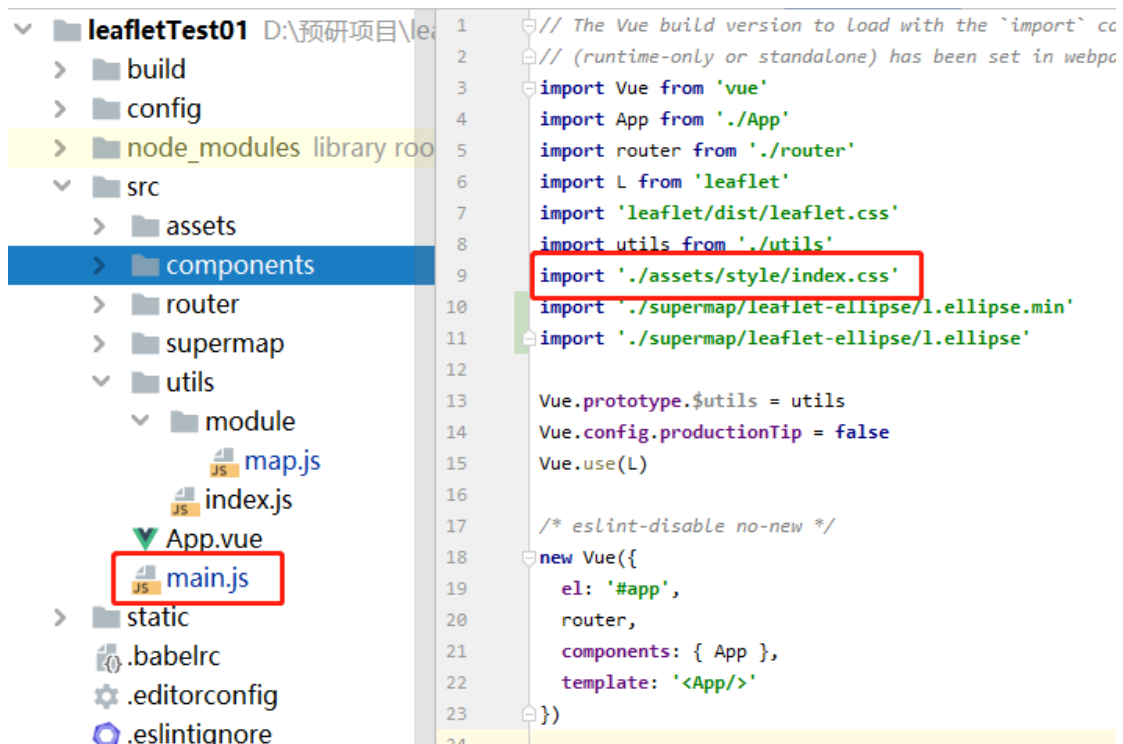

添加地图文字标注

在地图上面进行一些文字的标注,有些场景我们会用到,这这里我们文字标注用到了 DivIcon 图标,通过与 mark 相结合,将 mark 的图标设置为 DivIcon 图标,进行文字标注。

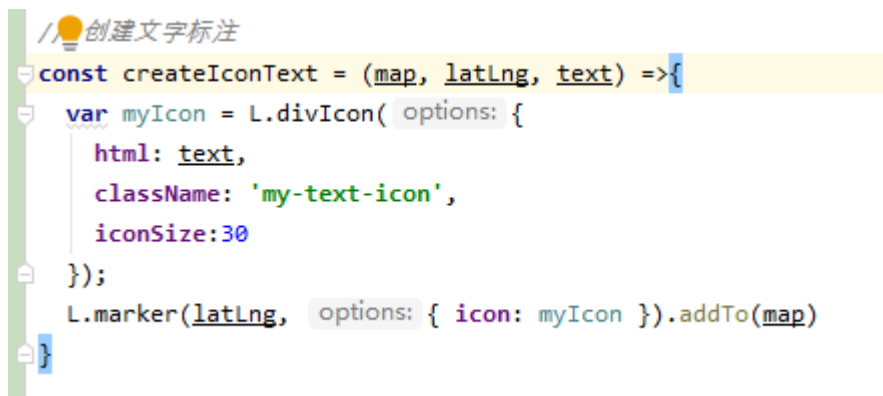
文字图标样式设置



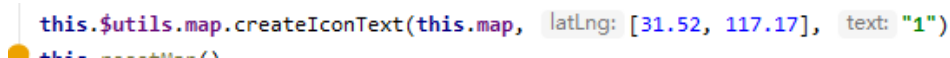
不要忘记在 main.js 中全局引入

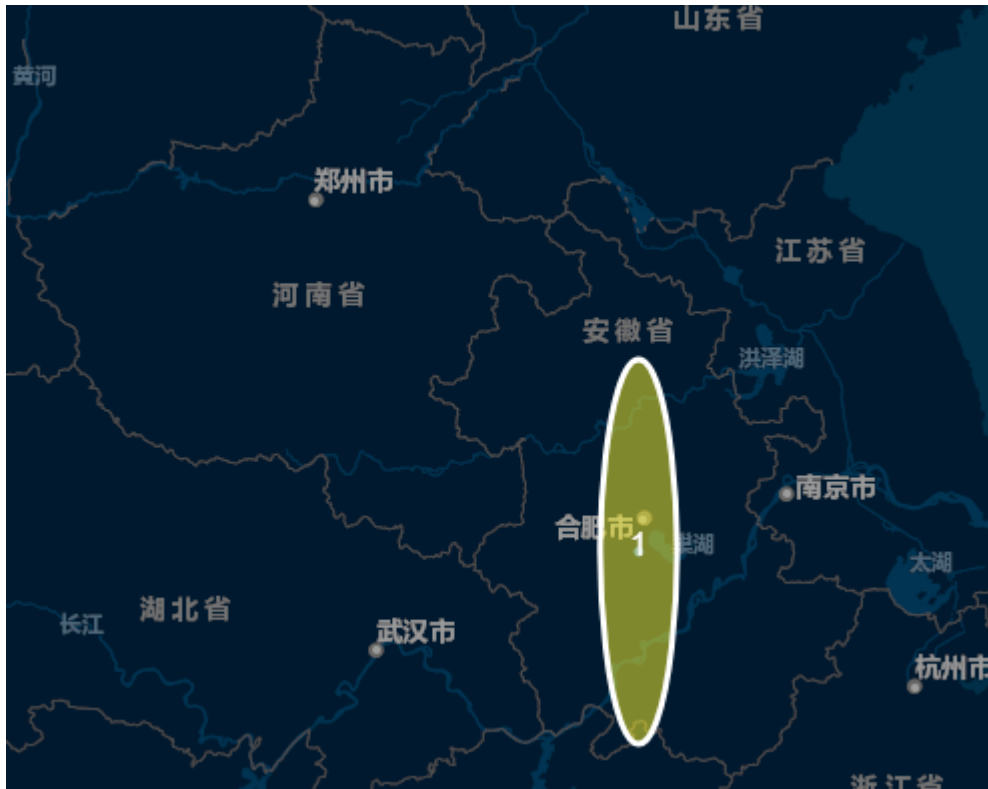


在 map.js 添加创建文字图标的方法



在地图页面/src/components/leaflet-test01.vue 调用此方法显示文字标注





在地图上面画圆

在 map.js 中添加创建圆形的方法

```
// 创建波菜圆点
const createCircle = async (map, latlng, options, number) => {
  // let circleName = '波菜' + number
  // 画一个circle
  const circle = L.circle(latlng, options).addTo(map)
  // 绑定一个悬浮框
  // let tooltips = `<h4>` + circleName + `</h4>`
  // circle.bindTooltip(tooltips)
  // 绑定一个自定义弹窗
  // const html = `<h1>` + circleName + `</h1><br><Link>点击查看详情</Link></h1>`
  circle.bindPopup(html, { maxHeight: 250, maxWidth: 490, className: 'sample-tooltips', offset: [0, 0] }).on('popupopen', function (params) {
    // console.log(params)
  })
  // })
  map.flyTo(circle.getLatLng())
  // map.fitBounds(circle.getBounds())
}
```

在地图页面/src/components/leaflet-test01.vue 调用此方法显示圆形画面

```
this.array = [[30.31, 117.02], [32.56, 117.21], [33.52, 115.47], [33.30, 119.09], [34.36, 119.33], [31.26, 119.29],
[31.48, 121.39], [32.05, 121.03], [41.48, 123.25], [40.51, 122.43], [39.55, 116.24], [26.05, 119.18], [20.54, 110.
for (let [key, val] of this.array.entries()) {
  console.log(key, val)
  this.$utils.map.createCircle(this.map, val, options: {
    text: '1',
    color: '#fff', // 描边色
    fillColor: '#fff82d', // 填充色
    fillOpacity: 0.5, // 透明度
    radius: 100000 // 半径, 单位米
  }, number: key+1)
}
console.log('test')
```

运行结果:



在地图上面画椭圆

Leaflet 并没有原生的绘制椭圆接口，所以绘制起来比较麻烦，因此要引入插件。

下载插件

Npm install leaflet.ellipse – save

Leaflet.ellipse	Leaflet.ellipse place ellipses on map by specifying center point, semi-major axis, semi-minor axis, and tilt degrees from west.	JD Fergason
---------------------------------	---	-----------------------------

插件引入

在 main.js 中全局引入

```

import Vue from 'vue'
import App from './App'
import router from './router'
import L from 'leaflet'
import 'leaflet/dist/leaflet.css'
import utils from './utils'
import './assets/style/index.css'
import 'leaflet-ellipse/l.ellipse.min'
import 'leaflet-ellipse/l.ellipse'

Vue.prototype.$utils = utils
Vue.config.productionTip = false
Vue.use(L)

/*eslint-disable no-new */

new Vue({
  el: '#app',
  router,
  components: { App },
  template: '<App/>'
})

```

创建画椭圆形的的方法

在 map.js 中创建画椭圆形的的方法

```

import 'leaflet/dist/leaflet.css'
import $L from 'leaflet'
import { chinaProvider } from 'leaflet.chinatsmproviders'
import L from 'leaflet'

// 画椭圆形波束
const ellipseMaket = (map, latLng, radii, tilt, options) => {
  L.ellipse(latLng, radii, tilt, options).addTo(map)
}

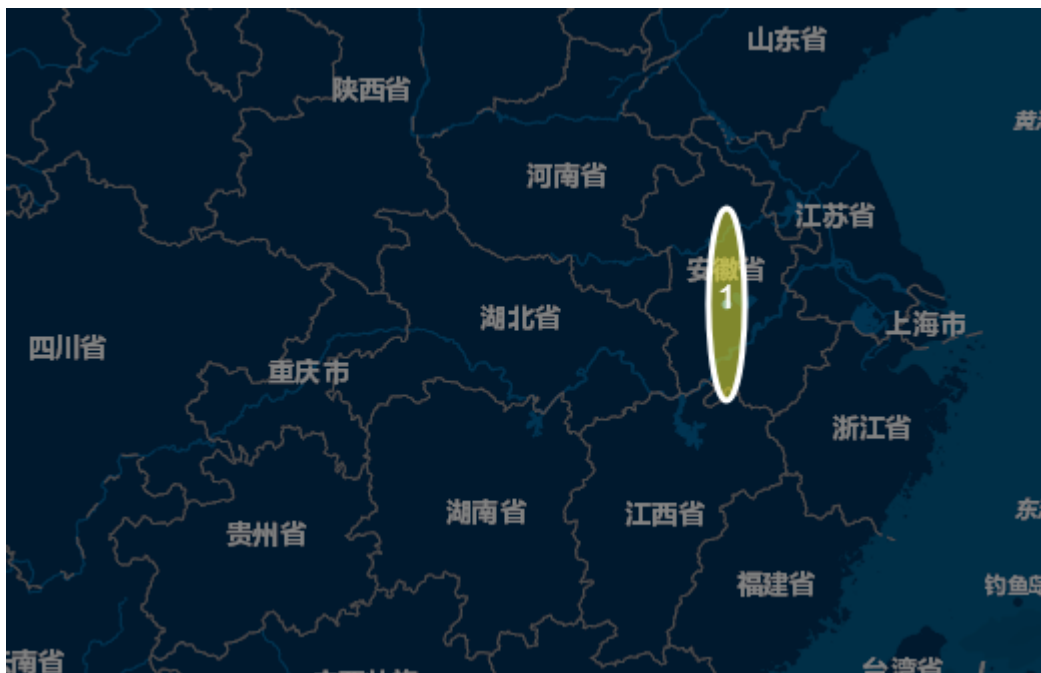
```

在地图页面/src/components/leaflet-test01.vue 中调用此方法，显示椭圆，并调用地图文本标注，设置的坐标相同，可以实现椭圆上面具有文本标注的效果：

```

this.$utils.map.ellipseMaket(this.map, latLng: [31.52, 117.17], radii: [200000, 40000], tilt: 90, options: {
  color: '#fff', // 描边色
  fillColor: '#fff82d', // 填充色
  fillOpacity: 0.5, // 透明度
})
this.$utils.map.createIconText(this.map, latLng: [31.52, 117.17], text: "1")
this.resetMap()

```



参数解释

API

Factory method

```
L.ellipse( <LatLng> latlng, <Radii> radii, <Number> tilt,
           <Path options> options? )
```

- latlng - The position of the center of the ellipse.
- radii - The semi-major and semi-minor axis in meters
- tilt - The rotation of the ellipse in degrees from west
- options - Options dictionary to pass to L.Path

依次是椭圆中心点坐标；椭圆半径数组（上半径、下半径）、椭圆方向（90 度，180 度等）

Leaflet 与 Echart 结合使用

待用待写、、、、、、、

切换地图底图

因为 **leaflet** 提供的是自家的底图，服务器放在美国，并且是英文样式，在中国区域的数据也不全，基本上无法详细展示我们所要的信息。因此大部门用户考虑使用国内的免费地图，如

天地图、高德等发布的地图服务。而下面的插件便是将国内的底图封装出来，下载安装后直接调用相应的接口，切换底图。

下载插件

Npm i leaflet.chinatmsproviders

Leaflet.ChineseTmsProviders	Contains configurations for various Chinese tile providers — TianDiTu, MapABC, GaoDe, etc.	Tao Huang
	A set of tools for using ArcGIS services with	

创建切换底图图层的方法

在 map.js 中引入 [Leaflet.ChineseTmsProviders](#) 插件,并创建切换底图的方法:

```
import { chinaProvider } from 'leaflet.chinatmsproviders'

// 改变底图图层
const changeLayer = async (map) => {
  let normalm3 = await L.tileLayer.chinaProvider( type: 'Geoq.Normal.PurplishBlue', options: {})
  normalm3.addTo(map)
}
```

参数解释

上述代码的 type 代表着底图图层类型，常见的底图图层类型如下：

- TianDiTu
 - TianDiTu.Normal.Map
 - TianDiTu.Normal.Annotation
 - TianDiTu.Satellite.Map
 - TianDiTu.Satellite.Annotation
 - TianDiTu.Terrain.Map
 - TianDiTu.Terrain.Annotation
- GaoDe
 - GaoDe.Normal.Map (include Annotation)
 - GaoDe.Satellite.Map
 - GaoDe.Satellite.Annotation
- Google
 - Google.Normal.Map (include Annotation)
 - Google.Satellite.Map (exclude Annotation)
 - Google.Satellite.Map (include Annotation)
- Geoq
 - Geoq.Normal.Map
 - Geoq.Normal.PurplishBlue
 - Geoq.Normal.Gray
 - Geoq.Normal.Warm
 - Geoq.Normal.Hydro
- OSM
 - OSM.Normal.Map
- Baidu
 - Baidu.Normal.Map
 - Baidu.Satellite.Map (exclude Annotation)
 - Baidu.Satellite.Annotation

底图图层的类型，代码实现

```
/**
 * 智图地图内容
 */
let normalm1 = L.tileLayer.chinaProvider( type: 'Geoq.Normal.Map', options: {
  attribution: 'Haut-Gis-Org © 智图地图'
})
let normalm2 = L.tileLayer.chinaProvider( type: 'Geoq.Normal.Color', options: {
  attribution: 'Haut-Gis-Org © 智图地图'
})
let normalm3 = L.tileLayer.chinaProvider( type: 'Geoq.Normal.PurplishBlue', options: {
  attribution: 'Haut-Gis-Org © 智图地图'
})
let normalm4 = L.tileLayer.chinaProvider( type: 'Geoq.Normal.Gray', options: {
  attribution: 'Haut-Gis-Org © 智图地图'
})
let normalm5 = L.tileLayer.chinaProvider( type: 'Geoq.Normal.Warm', options: {
  attribution: 'Haut-Gis-Org © 智图地图'
})
let normalm6 = L.tileLayer.chinaProvider( type: 'Geoq.Normal.Cold', options: {
  attribution: 'Haut-Gis-Org © 智图地图'
})
...

```

```
/**
 * 天地图内容
 */
let normalm = L.tileLayer.chinaProvider( type: 'TianDiTu.Normal.Map', options: {
  attribution: 'Haut-Gis-Org © 天地图'
})
let normala = L.tileLayer.chinaProvider( type: 'TianDiTu.Normal.Annotation', options: {
  attribution: 'Haut-Gis-Org © 天地图'
})
let imgm = L.tileLayer.chinaProvider( type: 'TianDiTu.Satellite.Map', options: {
  attribution: 'Haut-Gis-Org © 天地图'
})
let imga = L.tileLayer.chinaProvider( type: 'TianDiTu.Satellite.Annotation', options: {
  attribution: 'Haut-Gis-Org © 天地图'
})
let normal = L.layerGroup( layers: [normalm, normala])
let image = L.layerGroup( layers: [imgm, imga])

```

```
* 谷歌
*/
let normalMap = L.tileLayer.chinaProvider( type: 'Google.Normal.Map', options: {
  attribution: 'Haut-Gis-Org © Google Map'
})
let satelliteMap = L.tileLayer.chinaProvider( type: 'Google.Satellite.Map', options: {
  attribution: 'Haut-Gis-Org © Google Map'
})
...

```

```

/**
 * 高德地图
 */
let Gaode = L.tileLayer.chinaProvider( type: 'GaoDe.Normal.Map', options: {
  attribution: 'Haut-Gis-Org @ AMap'
})
let Gaodimgem = L.tileLayer.chinaProvider( type: 'GaoDe.Satellite.Map', options: {
  attribution: 'Haut-Gis-Org @ AMap'
})
let Gaodimga = L.tileLayer.chinaProvider( type: 'GaoDe.Satellite.Annotation', options: {
  attribution: 'Haut-Gis-Org @ AMap'
})
let OpenStreetMap = L.tileLayer(
  url: 'http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', options: {
    attribution: 'Haut-Gis-Org @ OpenStreetMap'
  }
)

'Esri卫星图': EsriWorldImagery,
'Esri-灰白': EsriDarkGreyCanvas,
'智图地图': normalm1,
'智图午夜蓝': normalm3,
'智图灰色': normalm4,
'智图暖色': normalm5,
'天地图': normal,
'天地图影像': image,
'谷歌地图': normalMap,
'谷歌影像': satelliteMap,
'高德地图': Gaode,
'高德影像': Gaodimage,
'OpenStreetMap': OpenStreetMap,
'MapBox': MapBox

```

以上均创建了各个底图类型的图层，仅仅需要添加到 map 中便可切换地图底图图层了。
例如：

```

let normalm3 = await L.tileLayer.chinaProvider( type: 'Geoq.Normal.PurplishBlue', options: {})
normalm3.addTo(map)

```