参与项目：开发GIS工具库hxol

项目介绍：hxol是基于OpenLayers及ol-ext专门为Web GIS 客户端开发提供的JavaScript类库包，用于在Web上创建交互式地图。hxol支持初始加载包含多个源地图的Map容器，利用Lit框架构建地图控件UI，可直接调整CSS设计控件样式。主要功能包括：地图加载、设备定位、图例、元素聚合、聚合动画交互、图层选择器、服务器元素更新、图形绘制及元素动画等。

本人负责：

1. 分析项目需求，搭建项目框架，构建项目结构、根据项目需求进行技术选型，进行babel、rollup、jest、gulp等技术配置
2. 开发地图，交互、动画等功能类库、基于lit框架开发地图控件及样式
3. 编写浏览器及node环境下的各功能demo，并使用parcel打包、基于jest编写测试单元
4. 使用Vue编写工具库开发文档
5. 使用git进行项目管理，与团队进行协作，进行分支管理、代码版本管理。

介绍：

因为我们多个项目对于GIS方面的功能，很多功能是相似的，代码也是可复用的，这样会反复写很多重复的代码，并且维护也比较麻烦，所以我们就基于多个项目的需求，基于openlayers和ol-ext开发了GIS组件库hxol。

hxol主要包括地图容器、元素聚合及动画、定位组件、图例等这些功能组件。

因为hxol是面向对象开发的，每一个功能模块都是一个对象，最终我们是要将多个js文件合并成一个js文件，所以打包工具选择的是rollup。

然后是编写浏览器环境下及node环境下的demo。浏览器环境下就是直接编写HTML文件，通过script标签引入hxol，直接浏览器打开。Node环境下的demo是通过npm导入我们的组件，然后通过parcel直接打包HTML文件，打包出一个轻量的dist包，然后再浏览器运行localhost:1234查看代码demo。

开发中也使用了babel（js语法编译器）将es6代码编译成ES5的代码。

将所有的scss文件编译成css文件（sass有相应的转换指令）输出到hxol文件中，并对生成的css文件利用gulp进行压缩（删除空格、换行等）。

Gulp：js前端自动化构建工具，主要是来设定程序自动处理静态资源的工作。Gulp是一个基于任务的工具，也就是说，gulp的每个功能都是一个任务，压缩css的任务，合并文件的任务，gulp规定任务要写在一个gulpfile.js的文件中，在这个文件中用来配置所有任务。

开发中组件UI部分原先是用web component编写自定义的组件UI。web component可以基于HTMLElement自定义自己的元素模板，但是定义的模板没办法响应式变化（视图数据变化的时候，视图不会自动更新），而且语法比较麻烦，需要原生创建dom、操作DOM，所以使用了一个基于web component的框架lit。

Lit组件是一个可重用的UI、比较快速轻量：

1. 可以开发自定义的模板，编写模板时跟我们平常再HTML中编写模板语法一样，并且可以使用表达式，以及一些自定义指令，可以构建响应式的模板。
2. 反应性属性：每当反应式属性发生变化时，组件就会重新渲染。（响应式模板）
3. 更新速度快：因为lit会跟踪UI的动态部分，并且只在底层状态发生变化时更新这些部分—无需重建整个虚拟树。

难点：构建组件UI

一开始我们是使用纯js构建的组件UI，但是构建的模板没办法复用，并且定义自定义方法比较难，所以就用web component构建UI模板，但是后来写的过程中，web component没办法在数据更新或者数据改变时，响应式的更新模板，所以从web component的模板中选择了lit框架。

参与项目：NMS（运维管理系统）COPEL项目

项目介绍：COPEL为巴西交付项目，产品主要以首页、监控、维护三大功能菜单，包含首页、网络监测、设备监测、固件升级、任务管理等菜单子功能实时监测设备运行，对设备各项数据进行多维度的统筹分析，大幅提升系统运维效率及管理水平。其中首页及详情页包括统筹展示系统各项参数，如设备告警及运行状态等，并以GIS及拓扑图的方式展示各设备部署情况，让用户全局审视系统全貌及运行状态，对海量不同设备实现聚合及点击展开设备功能，同时也提供设备搜索、设备快速定位及展示该设备状态等详细信息。

本人负责：

1. 根据业务需求，讨论页面、功能设计，利用Vue绘制页面。
2. 实现首页、详情页利用WMS方式加载多个底图及设备拓扑结构。WFS方式加载多种矢量设备，根据设备不同状态筛选设备，设置样式、集合、交互、定位、查看详情等功能。
3. 利用webSocket对设备数据进行轮询更新，更新设备详情及样式。
4. 对测试及现场人员提供技术支持、利用git与其他组人员共同管理项目代码。

参与项目：海兴网络规划平台（HXNP）

项目介绍：HXNP基于海兴空间信息服务平台完成的软件研发，核心服务位于hxnpserver后台，引入无线信号覆盖预测模型，结合空间分析算法提供无线设备的信号覆盖仿真服务、无线网络拓扑自动生成等服务，浏览器端页面主要功能包括数据管理、规划仿真、任务、在线编辑、数据浏览等。

本人负责：

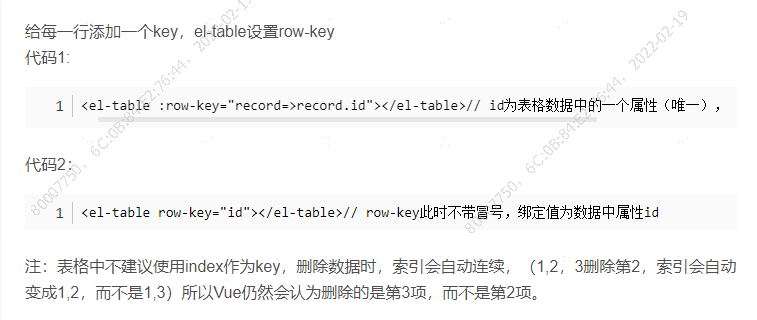
1. 实现HXNP参数模板、图层收藏及缓存、数据导出及发布、多用户图层保护、国际化等功能开发。
2. 实现多图层可视范围内加载、拓扑可视化等功能开发。
3. 开发图例、设备搜索定位及设备收藏等功能。
4. 图层数据处理、测试支撑、代码管理。

难点：

1. 场景：我们中会有收藏图层、设备这些功能，当我们取消收藏一个设备时，设备表格的焦点会自动聚焦在下一行的按钮上。给人的视觉效果不是很好。问题虽然不是大问题，但是也用了一两天去思考解决这个问题。

过程：刚开始的思路是点击删除一行数据后，自动失去焦点，但这时候的效果时，下一行的按钮会聚焦再失焦。

然后也是查看了element-table文档和源码发现table有个属性row-key，这个就是利用的vue的key的作用，key主要是给组件元素添加了唯一标识，当更新组件的时候，可以通过key值来判断两个节点是否相同，如果相同会复用，不相同就创建新的。当没有给table设置row-key的时候，比如三行数据删除第二行，vue不会认为我们删除了第二行，只会把第二行的数据变成第三行，但是按钮的状态会就地复用。设置key之后，操作组件，就会通过key找到对应的节点，而不会就地复用。

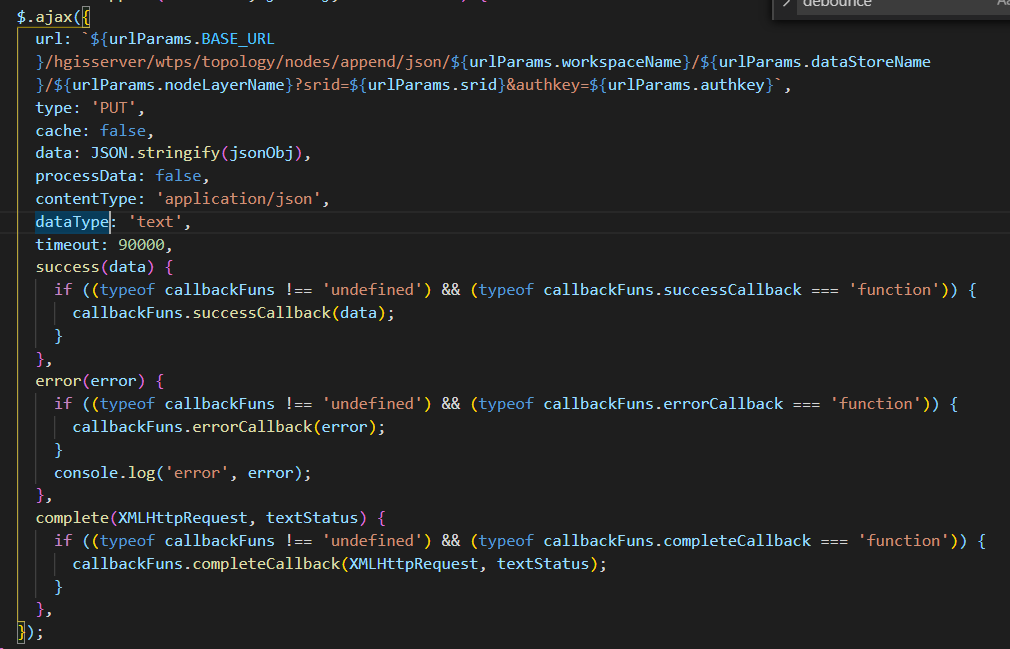


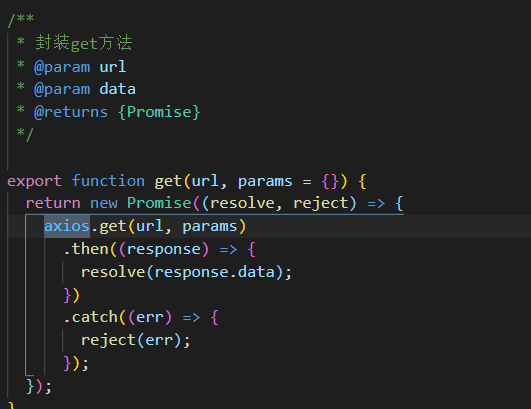
# 问题总结

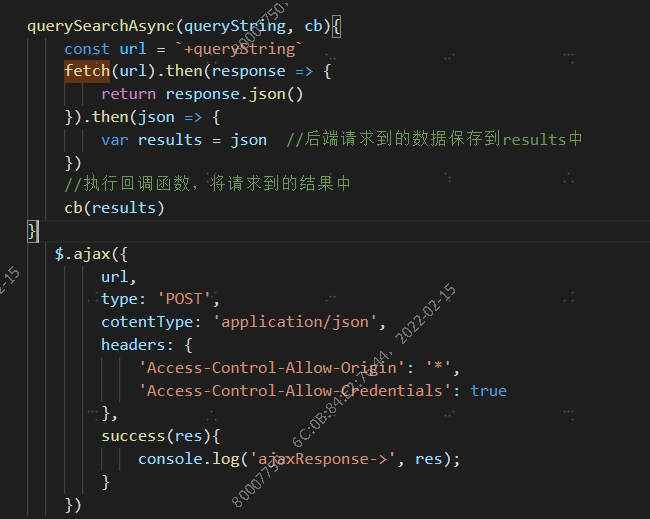
## ES5+ES6：

同步事件->微任务（promise）->宏任务（setTimeout）

请求：







## 构建Vue项目：

1. 安装node、@vue/cli（2.X版本已弃用，使用3.x+创建项目）
2. 如果安装旧vue，需要全局卸载再全局安装vue3.x+ npm i -g @vue/cli
3. vue create projectName创建vue项目，可通过自定义构建项目，选择自定义模块。或者vue init webpack name创建webpack项目，因为Vue-cli 3.x+和vue-cli 2.x使用了相同的 vue 命令，所以 vue-cli 2.x 被覆盖了。

自定义js工具文件（数值、对象、类、方法）：

Js文件其实就是vue文件中script中的部分，可import可以定义变量，也可以直接export default{}导出一个对象。也可以导出以下：

export const A = a/{};

export function A(){};

export class A{};

export default 如果加了default导入的时候，不加{}；不加default，导入的时候，加{}。

* **浏览器渲染过程**

1. 浏览器会将HTML解析成一个DOM树。

DOM树的构建过程是一个深度遍历过程，当前节点的所有子节点都构建好后才会去构建当前节点的下一个兄弟节点。

1. 将css解析成CSSOM树。

Css文档中，所有元素都是节点，与HTML文件中的标签节点一一对应，css中各节点之间同意拥有层级关系，彼此相连构成CSSOM树。

解析dom树期间，如果遇到js，阻塞了dom和cssom树的构建，优先加载js文件，加载完毕后，再继续构建DOM和CSSOM树。

原因是：js会对DOM进行操作，浏览器无法预测未来的DOM节点的具体内容，为了防止无效操作，节省资源，只能先记载js文件，阻塞DOM树的构建。

1. 根据DOM和CSSOM树构建渲染树，三者之间无先后顺序，而是会有交叉、并行构建。因此会形成一边加载、一边解析、一边渲染的现象。

渲染树不等同于DOM树，像display:none的元素是没必要存在于render树中的。

1. 有了渲染树之后，浏览器已经知道网页中有哪些节点、各个节点的css定义以及从属关系，下一步操作就是计算出每个节点在屏幕中的位置。
2. 然后就是绘制页面。渲染完成后，如果js操作了DOM，浏览器就会对页面进行重绘或者回流（重排）。

重绘：元素样式发生变化，但是节点位置、大小、宽高等没有发生变化。

回流：节点的几何属性发生变化，导致节点位置发生变化，就会回流，重新生成渲染树。

重排一定重绘，重绘不一定重排。



重绘和重排比较耗时，导致浏览器卡慢。如何避免重排：

1. 浏览器自己的优化：浏览器会维护1个队列，把所有会引起回流、重绘的操作放入这个队列，等队列中的操作到了一定的数量或者到了一定的时间间隔，浏览器就会flush队列，进行一个批处理。这样就会让多次的回流、重绘变成一次回流重绘。

2. 我们要注意的优化：

* 改变样式时，直接改变元素的className；
* display:none；先设置元素display：none，使元素脱离文档流，这时候对元素进行操作，不会引起重排，操作完成后，再将display设为block，这样的话只会有两次重排（显示和隐藏时）。
* 将需要多次重排的元素，position设置为absolute和fixed，元素脱离文档流，变化不会影响到其他元素，不会重排。
* 尽量不使用table布局。
* **ES6+JS总结**
* **let与var的区别**

1. var声明的变量存在变量提升，并且在全局内都有效；而let声明的不存在变量提升，并且只在（声明的代码块内）块级作用域内有效；

2. var可以重复声明，let不允许在相同的作用域内重复声明同一个变量；

const声明的变量是一个只读的常量，不可修改，基本数据类型的话就是值不可以变，对于引用数据类型，内存地址不可以变，但是对象内部的属性可以发生变化。

* **Js数据类型**

基本数据类型：

String：typeof instance === ‘string’

Number：typeof instance === ‘number

Boolean：typeof instance === ‘boolean

Null：typeof instance === ‘Object

undefined：typeof instance === ‘undefined’

Symbol：typeof instance === ‘symbol’

BingInt：typeof instance === ‘bigint’,可以用任意精度表示证书，可以安全的储存和操作大整数。Bigint是通过在数字末尾加n或者调用构造函数来创建的。

> const x = 2n \*\* 53n;

9007199254740992n

> const y = x + 1n;

9007199254740993n

引用数据类型：

Object：typeof instance === ‘object’

大写和小写意义不一样，比如String是构造函数，有属性和方法，而string是变量的一种类型。

检测数据类型：typeof、instanceof、。

typeof检测数据类型，只能检测到属于对象object，并不能检测出是哪一种对象（function、array）。

instanceof检测的是原型，判断构造函数的prototype属性是否在实例对象的原型链上。即判断A是否为B的实例。

a instanceof String：构造函数String.prototype是否在实例对象a的原型链上。在a的原型链上中层层查找，如果一直找到顶层null，也没有String.prototype，返回false，否则返回true。只能用来判断两个对象是否属于实例关系，不能判断一个对象具体属于哪种类型。

Object.prototype.toString.call(target)：任何值调用Object原生的toString方法，都会返回一个描述对象的字符串。默认toString返回"[object Type]"，其中Type是对象类型。

判断是否为一个数组也可以用：Array.isArray([1,2]);

判断是否为DOM元素，利用的DOM对象特有的nodeType属性，如果等于1即为DOM元素。

* **Class类**

class Point {

constructor(x, y) {

this.x = x;

this.y = y;

}

toString() {

return '(' + this.x + ', ' + this.y + ')';

}

}

事实上，类的所有方法都定义在类的prototype属性上面。因此，在类的实例上面调用方法，其实就是调用原型上的方法。

1. ES6类中定义的方法是不可枚举的，ES5中可以；

2. 类中this指向的是实例对象；

3. 类都要有constructor方法，new时自动调用该方法，如果没有显示定义，会默认添加一个空的constructor；

4. 类中定义的所有方法都会被实例继承，如果在一个方法前加上static关键字，表示该方法不会被实例继承，而是直接通过类调用，这就是“静态方法”。静态方法中的this指向类，而不是实例。（A.toString()）

5. 静态属性定义在class本身的属性，而不是定义在实例对象this上的属性，直接通过类去调用静态属性。

6. 私有属性和私有方法只能在类内部调用，目前常用区别：私有方法“\_fun”,但只是命名上的区别，在外部仍然可以调用到。另一种可以将私有方法移出类，定义在类外部。

私有属性“#count”即表示一个私有属性，只能在类内部使用this.#count；#也可以放在方法前面，表示一个私有方法。

ES6通过关键字extends继承一个类。子类构造函数内要必须有super()，super()执行的是父类的构造函数。ES5实现继承是‘先实例对象，再继承’，ES6继承机制是‘先继承，再实例对象’所以ES6中子类必须先执行super()，才能继承父类。执行super后才可以使用关键字this。super内部的this指向的是子类的实例对象。

super关键字作为函数时，执行的是父类的构造函数。

super作为对象时，再普通函数中，指向父类的原型对象。在静态方法中指向父类。当super指向父类原型对象时，定义在父类实例上的属性或方法（挂载在this下的），是无法通过super调用的。定义在原型上（类中定义的方法，是直接挂载在原型下的），可以通过super调用到。

7. 扩展运算符...：对于数组是将数据转化为逗号分隔的参数序列[1,2] =>1,2;

引入对象后，是取出参数对象的所有可遍历属性，拷贝到当前对象之中。let z = { a: 3, b: 4 };let n = { ...z };n // { a: 3, b: 4 }

* **全局作用域、块级作用域与函数作用域：**

全局作用域：全局都可以使用；

函数作用域： 变量在定义的函数内及嵌套的子函数内都有用

块级作用域：声明的变量只在当前语句块中有作用，即在{}花括号内的域

例如：

fun(c = 0) {

if (c == 0) {

var a = 1; //var声明的在整个函数内都有效

let b = 2; //let声明的变量只在他所在的代码块{}内有效

}

console.log("a:>>>", a); // 1或者undefined

console.log("b:>>>", b); // 'b' is not defined no-undefined

},

* **v-if 与 v-show 的区别**

v-if是动态添加dom的，当值为false的时候，是完全移除该元素，即dom树中不存在该元素，只有为true时才会渲染元素；

v-show元素总是会被渲染，仅仅是控制DOM元素的display样式，来控制元素显隐。值为false时，通过设置元素的display：none来控制元素显隐。（与display: none效果一样）

v-if判断是否加载，可以减轻服务器的压力，在需要时加载,但有更高的切换开销;v-show调整DOM元素的CSS的dispaly属性，可以使客户端操作更加流畅，但有更高的初始渲染开销。如果需要非常频繁地切换，则使用 v-show 较好；如果在运行时条件很少改变，则使用 v-if 较好。

v-if与v-for永远不要在一个元素上同时使用，因为vue3.x中v-if的优先级比v-for高，所以如果同时使用可能会导致变量还不存在，

可以结合计算属性或者包裹一个template。



* **for和forEach（增强for循环）、map的区别**

for循环中可以使用continue，break、return false来控制循环和跳出循环，但是forEach不可打断。

forEach循环时会将空值跳过，for不会；

for可以控制循环的七点（i值），forEach只能从0开始；

for循环中可以修改索引i（比如i++，i=i+2这些，下次循环就会从修改过的i值开始），foreach不可以修改i（底层控制index自增，无法修改）；

forEach循环返回的是undefined、map返回的是符合条件的新数组。

* **普通函数与箭头函数**

声明式函数：function A(){}；函数表达式：const A = function（）{}；

箭头函数只能用函数表达式声明：const test = () =>{}；

普通函数的this只有在调用时才确定，指向调用函数的对象。

箭头函数的this，在定义时就确定了，箭头函数默认不创建自己的this，而是会从自己的作用域链的上一层继承this，与外层的this保持一致。也不可以使用call、apply、bind等方法改变this指向。

多层嵌套的箭头函数this，与最近一层普通函数所在的环境this保持一致。

* **HTTP状态码**

1xx(临时响应)

100: 请求者应当继续提出请求。

101(切换协议) 请求者已要求服务器切换协议，服务器已确认并准备进行切换。

2xx(成功)

200：正确的请求返回正确的结果

201：表示资源被正确的创建。比如说，我们 POST 用户名、密码正确创建了一个用户就可以返回 201。

202：请求是正确的，但是结果正在处理中，这时候客户端可以通过轮询等机制继续请求。

3xx(已重定向)

300：请求成功，但结果有多种选择。

301：请求成功，但是资源被永久转移。

303：使用 GET 来访问新的地址来获取资源。

304：请求的资源并没有被修改过，客户端有缓存并且没有过期。

4xx(请求错误)

400：请求出现错误，比如请求头不对等。

401：没有提供认证信息。请求的时候没有带上 Token 等。

402：为以后需要所保留的状态码。

403：请求的资源不允许访问。就是说没有权限。

404：请求的内容不存在。

5xx(服务器错误)

500：服务器错误。

501：请求还没有被实现。

* **VUE总结**
* **vue组件之间的通信（调用方法/使用其数值/传值）：父子组件、兄弟组件、祖孙组件**

父>>子：props方式向子组件传值；也可以通过refs获取子组件的实例，修改子组件data属性；也可以直接通过$chidren调用子组件方法或者获取数值。

子>>父：子组件向父组件传值的话，是通过事件进行传递，可以在子组件中通过$emit派发一个函数并携带参数，然后在父组件中监听该事件，并操作数据。子组件中也可以通过$parent调用父组件实例，将参数传递给父组件。

兄弟组件之间：可以通过他们共同的父组件进行传值，通过$parent访问父组件实例，调用父组件方法或者获取父组件数据，比如子组件B通过this.$parent.a=“”，子组件C可以通过this.$parent.a访问该值，即实现兄弟组件之间的通信。

父组件调用子组件方法：$refs方式获取子组件实例，在调用其方法；

子组件中调用父组件方法：子组件中通过$emit派发一个函数，当父组件监听到该函数时，再执行父组件中的方法。或者通过this$parent调用父组件方法。

兄弟组件之间调用方法：可以通过$emit和$refs实现，例如在A组件中利用$emit派发一个函数，父组件监听到该函数后，通过refs调用B组件组件中的方法。

也可以通过事件总线EventBus、$emit/$on、vuex实现父子、兄弟及隔代组件之间的通信。

**事件总线EventBus：**

创建EventBus：

// 1.event-bus.js

import Vue from 'vue'

export const EventBus = new Vue()

// 2.main.js

Vue.prototype.$EventBus = new Vue()

实质上EventBus是一个不具备DOM的组件，发送和监听主要是用到：

// 发送消息

EventBus.$emit(channel: string, callback(payload1,…))

// 监听接收消息

EventBus.$on(channel: string, callback(payload1,…))

// 移出事件

EventBus.$off(channel:string)

## VUEX：

是为vue开发的状态管理模式。核心是store仓库，相当于一个容器，包含项目中大部分的状态，和全局对象有以下两点不同：

1. vuex的状态存储是响应式的。当vue组件从state中读取状态时，如果状态发生变化，那么相应的组件也会更新；

2. 不能直接改变state状态，只能通过mutation改变

state：状态对象，声明一些状态变量；在组件通过this.$srore.state. 调用状态；

mutation：用于改变仓库中的state，类似于事件， 需要在组件中通过this.$store.commit('事件名称'，额外参数)；只能是同步事件。

action：类似于mutation，但是action提交的是mutation，而不是去直接改变state中的值。action可以包含任意异步操作。通过this.$store.dispatch（‘事件名称’）触发。

modules：Vuex允许我们将store分割成module（模块），每个模块拥有自己的state、mutation、action、getter甚至是嵌套子模块。

const moduleA = {

state: () => ({ msgA: 'msgA}),

mutations: { ... },

actions: { ... },

getters: { ... }

}

const moduleB = {

state: () => ({ ... }),

mutations: { ... },

actions: { ... }

}

const store = new Vuex.Store({

modules: {

a: moduleA,

b: moduleB

}

})

store.state.a.msgA // -> moduleA 的状态

store.state.b // -> moduleB 的状态

* **computed和watch属性区别：**

computed是计算属性，依赖于其他的属性，并且computed的值有缓存，只有当他依赖的属性值发生变化时，下一次获取computed的值才会更新计算值。

watch是监听数据的变化，每当监听的数据发生变化时，都会执行回调进行后续的操作。

computed适用于重新计算比较费时不用重复计算的环境，如果一个数据依赖于其他数据，就把这个数据设计为computed。

watcher如果这个数据发生变化时需要做一些处理比如一些异步操作，使用watcher来观察这个数据变化。

vue不能直接根据索引修改数组的值，或者修改数组的长度，可以通过一下几个方法

Vue.set();

this.$set()

array.splice()

* **require、import的区别：**

require是运行时动态加载，加载整个，import的编译时加载，只加载导入的包

import/export 只能在模块顶层使用，不能在函数、判断语句等代码块之中引用；require/exports 可以。

import/export 导出的模块默认调用严格模式。require/exports 默认不使用严格模式，可以自定义是否使用严格模式。

* **MVVM开发模式：**

MVVM是一种基于前端开发的架构模式， Model层是指数据模型，就是后端提供的一些api接口。View指的就是视图层，用户界面主要是由HTML和CSS创建。 ViewModel是视图数据层，处理从Model层获取的数据，然后加载到View层。比如，页面的一些交互功能都是封装在ViewModel层，ViewModel层实现了双向数据绑定，这样前端开发者就不用直接去操作DOM更新视图，更新视图的数据就会自动更新视图。

Vue实现双向数据绑定的原理：

双向数据绑定是指：数据变化更新视图，视图变化更新数据。

主要的实现步骤是：

1. 实现一个监听器Oberver来劫持并监听所有属性，利用Object.defineProperty()对属性都加上setter和getter，当某个值被赋值的时候，就会调用setter，就能监听到数据的变化。

2. 实现一个解析器compile，解析VUE模板指令，将模板中的指令替换成数据，然后初始化渲染页面视图。并且会将每个指令对应的节点绑定更新函数，这样一旦数据发生变化，就会调用更新函数进行数据更新。

3. 实现一个订阅者watcher，订阅者是监听器和解析器之间通信的桥梁，主要任务是订阅监听器中属性的变化的消息，当属性发生变化时，就会触发解析器中对应的更新函数来更新视图。

* **Vue路由：**

路由有三种模式：hash、history、abstract。

hash：使用URL中的hash值作为路由，支持所有的浏览器。默认模式。

* hash值即浏览器地址中的#后面的内容，不包含在实际的HTTP请求中。只是表示浏览器的一种状态。
* hash值的改变都会向浏览器的访问历史中增加一个记录，因此可以通过浏览器的前进、回退按钮控制hash值的改变。
* 可以通过a标签，并设置href，当用户点击这个标签后，URL的hash值会发生变化。或者通过location.hash进行赋值，改变URL中的hash值。
* 也可以使用hashchange事件监听hash值的改变，从而对页面进行跳转。

History：依赖HTML5 history API 和服务器配置，请求地址与浏览器地址完全一致。

Abstract：支持所有的JavaScript运行环境。

* this.$route访问的是当前路由；this.$router访问的是全局路由实例
* **路由跳转**

1. 声明式标签跳转<router-link to=”/home”>
2. 编程式导航this.$router.push()

router.push()会向history栈中添加一个新的记录，所以用户点击浏览器回退按钮时，会回到之前的URL。

点击<router-link to=”/home”>的时候，其实内部调用的就是router.push()。



* Vue中watch中可以监听$route当前路由，然后进行一些路由改变时的操作。
* Router定义动态路由就是路由路径中有动态参数，比如route中定义的路由路径path: ‘/home/:id’，就是动态路由。跳转该路由的时候，要携带id参数，可通过params(this.$route.params获取-传递参数使用router获取参数是用当前路由route获取参数)、也可以通过query携带（this.$route.query获取）。
* Vue-router传参

Params: 只能使用name，不能使用path；参数不会显示在路径上；浏览器强制刷新参数会被清空。

Query：参数会显示在路径上，刷新不会被清空；query可以跟path、name一起使用。

* **路由导航守卫**

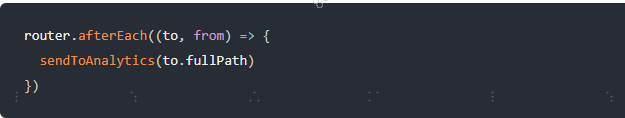
路由提供的导航守卫主要用来通过跳转或者取消的方式守卫导航。有全局的、单个路由独享的、或者是组件级别的。

* 全局导航钩子：

全局前置守卫：router.beforeEach，跳转之前调用。



* 全局后置钩子：router.afterEach，路由跳转后调用，不接受next参数。



在该周期内，可以进行更改页面标题、声明页面等辅助功能。

* 路由独享的钩子：beforeEnter()



* 组件内的守卫钩子：

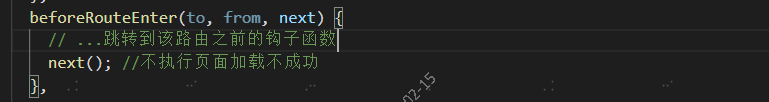
beforeRouteEnter

beforeRouteUpdate

beforeRouteLeave



组件中：



* **H5 新增特性：**

1. Canvas、SVG – 用于绘画的元素，canvas绘制的图片会失真，SVG可绘制矢量图形，不会失真。
2. video、audio – 用于播放视频和音频的媒体
3. Drag、Drop – 用于拖放
4. Geolocation – 用于获取地理位置
5. 新增的客户端存储数据对象，localStorage、sessionStorage

localStorage：没有时间限制的本地存储，需要手动删除；

sessionStorage：关闭浏览器时，数据会被删除；

Cookie也是客户端存储，只能存储4K左右，localStorage和sessionStorage可以达到5M或者更大。

sessionStorage在不同的浏览器窗口中不共享，即使是同一个页面。localStorage、cookie在所有同源窗口中都是共享的。

1. 新的结构标签，比如 article、footer、header、nav、section、aside、mark等
2. 新增了表单的type类型，比如 calendar、date、time、email、url、search

H5的优缺点：

优势：

* 提高可用性和可维护性，改进了用户体验。
* 新增语义化标签，有助于开发人员定义清晰的结构。
* 可以播放视频音频，增加多媒体元素。
* 利用h5动画，友好地替代了flash和silverlight。
* 爬虫抓取网站的时候，对于SEO很友好。
* H5被大量应用于移动应用和游戏开发。
* 可移植性好

缺点：

现在大多数浏览器都支持h5，但是少部分的低版本浏览器不支持H5。

## ES6新增的特性（部分）：

let和const、Symbol属性；

解构赋值；

数组新增方法：

Array.from(将类数组转化为数组)、Array.of()（将一组值转化为数组）、find()和findIndex()、fill()等；

新增的对象方法：assign()复制一个对象、keys();得到一个对象的所有属性等；

使用了ES6的类以及类的继承；

还有Promise

下载文件到本地，通过创建a标签元素，设置H5中新增download属性，指示浏览器下载URL而不是跳转URL。然后执行a标签click事件，模拟点击链接下载文件到本地。

多个异步请求执行成功后，执行后续操作：

1. $.when(

$.ajax(),

$.ajax(),

$.ajax() ).done(fn).fail(fn);

2. Promise.all([promise1,promise2]).then(function(){})

当一个异步执行成功后，在执行一个异步，也可以利用$.when($.ajax).done($.ajax())

* **组件生命周期：**

beforeCreate：组件实例被创建之前，$el和data还没有初始化

created：完成了data的初始化，$el还没有

beforeMount：完成了$el的初始化，但是还没有挂载

Mounted：是完成了挂载，此时组件实例已经完成创建

beforeUpdate：数组更新之前调用，此时data是最新的，但页面不是

update：组件数据更新之后，此时data与页面保持同步，都是最新的

beforeDestory：组件销毁前调用

destoryed：组件销毁后调用

created阶段已经完成了data的初始化，所以created、beforeMounted、mounted阶段进行异步请求。推荐在created钩子函数中调用异步请求，因为在created阶段中调用异步请求有以下优点：

1. 能更快获取到服务端数据，减少页面loading；

2. ssr不支持beforeMount、mounted钩子函数，所以放在created中有助于一致性。

父组件可以监听到子组件的生命周期吗？

比如父组件想要子组件挂载mounted就做一些逻辑处理，可以通过在子组件mounted函数内通过$emit分发一个函数，然后父组件监听。

更简单方式可以通过在父组件引用子组件时通过@hook来监听即可：<child @hook:mounted="dosomeing">，@hook方法不仅可以监听mounted，还可以监听其他的生命周期事件如created，updated等。

* **vue-router的三种模式：hash，history，abstract**

hash模式：即是浏览器地址里面的“#”符号，比如URL“http://www.abc.com/#/hello”，hash 的值为 #/hello。特点是，hash值虽然存在于URL中，但是不会被包括在HTTP请求中，只有符号之前的内容会被包含在请求中。对后端完全没有影响。因此对于后端来说即使没有做到路由全覆盖，也不会返回404。

history模式：利用了H5 History interface中新增的pushState()以及replaceState()方法（需要特定浏览器支持），history模式下，前端的URL必须和实际请求的URL一致，如果后端找不到URL对应的资源会404。

* **数组去重常用方法：**

1. indexOf方法去重：首先定义一个空数组，然后调用indexOf方法对原来数组进行遍历判断，如果元素不在新数组内，就push进新数组。

2. new Set方法

3. 双重for循环，结合splice去重

4. 构建一个新数组，先排序，再判断相邻元素（前一个）是否相等，如果不相等push到新数组

5. 利用arr.filter，返回符合当前元素索引与第一次出现该元素索引相同的元素数组。

* **git常用名指令：**

git status：查看状态；

git pull ：拉取代码；

git add . :添加所有的修改到缓存区；

git commit -m "备注":添加备注并提交；

git push：上传代码；

git stash save '' 暂存到本地；

git stash pop 恢复缓存；

git init 、git clone -b 分支名 url等指令。

git branch –a 查看当前分支;

git checkout –b dev 切换分支

display：none和visiblity:hidden:

两者都可以隐藏元素，display隐藏的元素，不在文档流中占位，浏览器不会解析该元素；

visibility隐藏的元素在文档流中占位，浏览器会解析该元素。

* **深拷贝与浅拷贝**

JS数据类型：String、Number、Boolean、Null、undefined、Object、Symbol。

Js数据类型包括：基本数据类型、引用数据类型。

基本数据类型：名和值都存在栈内存中，当复制一个基本数据类

型时，会在栈中开辟一个新内存，两者互不影响。

引用数据类型：对象存在堆内存中，栈中存的是对象的名和指向该对象内存地址的指针。针对js引用数据类型，浅拷贝是复制了指向对象的指针，复制对象和原对象会互相影响；深拷贝（将对象的每一层属性都复制）是开辟了新的内存空间和指向该对象的指针，复制对象和原对象互不影响。

深拷贝浅拷贝是针对引用数据类型（Object）来说的，对于基本数据类型没有意义。

浅拷贝实现方式： Object.assign()；直接=赋值、解构赋值、Array.concat()、Array.slice()（这两个方法不会修改原数组，只会返回一个浅复制了原数组中的元素的一个新数组）。

深拷贝的实现方式：递归去拷贝所有层级属性；通过JSON对象（JSON.stringify、JSON.parse）实现深拷贝；使用JQuery的extend方法实现深拷贝、lodash库中的\_.cloneDeep函数实现深拷贝、手写递归方法遍历对象、数组直到里面都是基本数据类型，然后再去复制，就可以实现深拷贝。

浅拷贝与深拷贝也是相对的，要看下数据结构，如果是单层数据结构，for..in、Array.slice、Array.concat、Object.assign()就能实现深拷贝，如果是嵌套数据类型，该几种方法实现的就是浅拷贝。

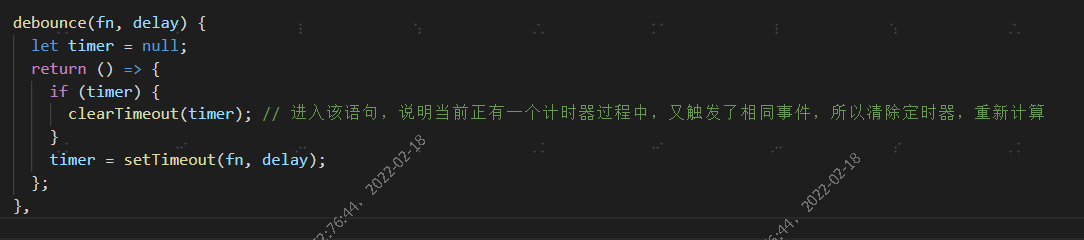
递归遍历深拷贝例子：



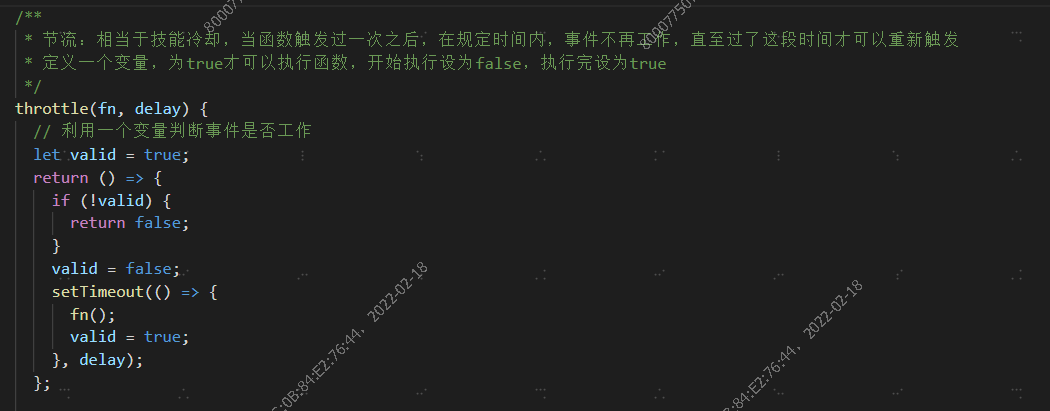
* **防抖和节流：**

防抖和节流主要是为了优化短时间内，频繁请求某事件。防抖是：给定一个时间，若在该时间内再次触发事件，则清除定时器重新定时器重新计时，如果在该时间内没有再次触发事件，则执行该事件。防抖即是：在给定时间内，只执行一次事件。

防抖实现：



节流是指事件触发一次后，在给定时间内，让该事件暂时失效，过了这段时间再重新激活。（即使不断触发，也在规定时间内触发一次，输出结果）。



应用场景：实现一个搜索框input，要支持实时搜索就可以使用节流（间隔一段时间就必须查询一次相关内容）。 页面resize事件，页面适配的时候，需要根据最终的页面情况进行页面dom渲染，就可以只用防抖，只需要判断最后一次的情况。

Lodash：是一个js工具库，lodash中有封装好的防抖节流方法。思想不变，用法简介。

testDebounce: debounce(func, 500, {leading:true})

testThrottle: throttle(func, 5000, {leading: true})

* **Array方法**
* Array.map、Array.filter

两者都可以操作数组，map是对原数组的加工，返回一个符合条件的新数组。filter是筛选符合条件的数组，但是返回的数据结构与原数组结构一致。（比如一个数字数组，map可以返回其他数据类型的数组，而filter只能返回与原数组结构一样的数字数组）

* Array.some

返回一个boolean值，遍历数组中的每一个值，有一个符合条件的，即返回true，若都没有，返回false。

pop删除数组最后一个值、push在数组最后添加元素、shift删除数组第一个值、unshift在数组前面添加值、slice切割子字符串，包含头不包含尾、splice删除，第一个参数索引，第二个是删除的个数、reduce返回按条件计算得到的值、from将类数组转化为数组、concat连接数组、keys遍历数组的key（得到的是一个迭代器，遍历每一项是数组的key）、values遍历数组的值、entries遍历数组的键值、every检测每一项值，如果都符合条件返回true、find找到第一个符合的元素并返回、includes判断是都包含指定值，返回布尔值、join将数组连接成一个字符串并返回、some判断数组中是否有符合条件的元素，返回boolean、toString将数组转换成字符串并返回。

数组方法总结：https://www.cnblogs.com/dhui/p/13750448.html

async/await返回的是一个promise，const result = await promise；返回值是promise resolved时的值，promise状态也有可能是rejected，所以await+promise代码放在try...catch中比较合适， 利用catch捕捉错误。

* **Vue虚拟DOM**

Vue1.0：利用的是Object.defineProperty()每个数据的修改都会通知watcher，进而通知dom去改变，实现双向数据绑定，内存消耗比较大。

Vue2.0引入虚拟DOM，通过diff之后再去通知DOM更新，相对于1.0响应式的级别修改了，watcher只到组件，组件内部使用虚拟DOM（vue2.0组件内部就是虚拟DOM的原理）。

浏览器渲染流程：创建DOM-创建StyleRules（样式表）-创建render树（DOM树+style）-布局layout-绘制painting。

虚拟DOM就是为了解决浏览器性能问题而设计出来的，就是使用js的Object模拟真实的DOM，当状态发生改变时，更新之前利用diff算法比较旧，新两棵DOM树的差异，再将虚拟DOM渲染成真正的DOM，更新视图。当一次操作多次DOM时，虚拟DOM不会立即操作DOM，而是将这几次更新的diff内容保存到一个JS对象（虚拟DOM）中，最终将这个JS对象一次性attch到DOM树上，再进行后续操作，避免大量无谓的计算量。

模板转化为视图的过程：

* Vue.js通过编译将template模板编译成一个代码字符串，然后再渲染函数（render）中执行这个字符串，渲染函数执行后会得到一个虚拟节点树。
* 修改DOM时，利用diff算法比较新旧DOM的差异，并将差异部分利用patch算法映射到DOM上，从而更新视图。

Vue-loader允许我们直接在template中编写模板字符串，将内容提取出来给vue-template-compiler，vue.js通过编译器将模板转换成渲染函数中的内容，执行这个渲染函数h()就会得到一个VNode，使用这个虚拟节点就可以渲染页面了（先编译模板语法再渲染成页面）。

例如，组件中编写dom：

<template>

<img src="../image.png">

</template>

Vue会将该代码编译成：

createElement(‘img’, {

attrs: {

src: require(‘../image.png’)

}

}

事实上，vue编译函数会将代码编译成一个代码字符串，这个代码字符串被包装在渲染函数中执行，生成一份Vnode。

* **数据结构：Map Set 两种新增的数据结构{}**

1. Map类似于对象，对象保存键值对。任何值（对象或者原始值）都可以作为一个键或一个值。

数据结构形式：{ key1=>value1, key2=>value2 }。

Map结构的键实际上是跟内存地址绑定的，只要内存地址不一样就可以视为不同的键。所以如果键是基本数据类型，只要值严格相等（===）就可以视为同一个键。

Map与Object的区别：

* 一个Object的键只能是字符串或者Symbols，但一个Map的键可以是任意值。
* Map中的键值是有序的，而Object中的键则不是。
* Map中键值对的个数可以通过size（map.size）获取，但是Object只能手动计算。
* Object都有自己的原型，原型链上的键名有可能和你自己在Object上设置的键名产生冲突。

const map = new Map();

map.set('a', a);

map.set('{}', object);

map.set('function(){}', function) 设置键值对

map.get('a') 获取值

map.has('a') 是否存在该key

map.delete('a') 删除key值

多次对同一个key放入value时，后面的值会把前面的值覆盖。

Map的迭代：for...of forEach；Map的遍历顺序就是插入顺序。

map进行合并时，如果有重复的键值，则后面的会覆盖前面的。

Set类似于数组，对象允许存储任何类型的唯一值。Set存储的值总是唯一的。有几个特殊值需要特殊对待：

+0与-0算重复值；undefined算重复值；NaN与NaN是不恒等的，但是Set结构中只能存一个，不能重复。

set.add() 添加值；

set.delete() 删除值；（没有键值对所以没有获取某个特定值的方法）

set.has();

set.clear();

Set结构的遍历顺序就是插入顺序。

Vue中的key有什么作用？

key作为Vue中vnode的唯一标记，通过这个key，我们的diff操作可以更准确、更快速。DOM发生改变的时候，diff算法会比较虚拟DOM的新旧DOM，有了key可以更快速正准确的找到DOM更新的节点位置。

更准确：因为带key就不是就地复用了，在SameNode函数a.key===b.key对比中可以避免就地复用的情况。所以会更加准确。

更快速：利用key的唯一性生成map对象来获取对应节点，比遍历方式更快。

Vue Date;

const date = new Date();

date.getFullYera() => 获取当前年；

date.getMonth() +1 => 获取当前月份；

date.getDate() => 获取当前日期（几号）；

Vue 组件中data为什么是一个函数，然后return一个对象，而new Vue中直接是一个对象？

原因：

因为组件是用来复用的，data是一个函数，组件实例化的时候data函数会返回一个对象，分配一个新的内存地址，这样可以保证各组件之间不会相互干扰，改变一个组件状态的时候，其他组件不会改变（相当于深拷贝）。但如果是对象，这样作用域没有隔离（相当于浅拷贝），这样组件之间的data会互相影响。而new Vue中的实例，不会被复用，所以不存在引用对象的问题。

* **事件委托**

事件代理又称为事件委托。即把原本需要绑定在子元素的响应元素（click、keydown。。。）委托给父元素，让父元素监听事件。

事件冒泡：一个元素的事件触发后，会依次一级一级往上调用父元素的同名事件，直到window。当点击了一个元素，不管这个元素有没有click事件，都会往上冒泡调用父元素的click事件（有的话）。

例div>p>span都添加了click事件，当点击了span时，会依次向上触发p、div的click事件。

事件委托就是基于事件冒泡这个原理。

例子：给ul的每一个li添加一个点击事件，打印li元素内容，如果li很多，会消耗很多内存，事件委托就是给父元素ul绑定点击事件，点击li时就会调用ul的点击事件，判断点击的元素是不是li，如果是打印li元素内容。事件委托下，如果有新增的li元素，也会自动绑定上点击事件，不用再手动绑定点击事件。

所以事件委托优点：

1. 节约内存
2. 能为之后新增的DOM元素依然添加事件

* **中断ajax请求**

停止ajax请求有两种方式：

1. 设置超时时间让ajax自动断开；

$.ajax({timeout: 1000(ms)})

1. 手动停止ajax，核心是调用XMLHttpRequest对象上的abort方法。

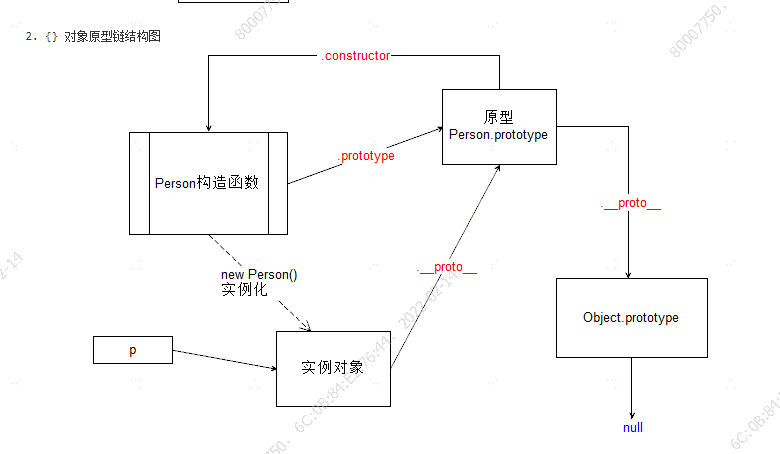
let ajax = $.ajax()

Ajax.abort()

* **Js原型链**

首先明确：函数才有prototype属性，对象（除Object）拥有\_proto\_属性。原型链的顶层是Object.prototype，而这个对象是没有原型对象的，即最终指向null。

当js引擎查找对象的属性时，先查找对象本身是否存在该属性，如果不存在，会在原型链上查找，但不会查找自身的prototype。



构造函数实例化得到一个对象，实例对象的\_proto\_属性指向，构造函数的prototype，构造函数的prototype指向构建函数本身。构造函数中的prototype对象也有\_proto\_属性，它指向的该构造函数的上一层函数Object.prototype（null）

* **Js实现继承**

1. 原型链继承

子类的原型指向父类实例

function Son(){}

Son.property = new Father();

缺点：创建子类实例时，无法向父类构造函数传参。

1. 构造函数继承

在子类中通过call、apply等方法，将父对象的构造函数绑定在子对象上。即将父类中的this指向子类。

function Son(){Father.call(this)}

缺点：只能继承父类实例的属性和方法，不能继承其原型上的属性和方法。

1. 组合继承

原型链继承和构造函数继承结合使用。

* **HTTP、HTTPS**
* http是超文本传输协议。信息是明文传输的。HTTPS是具有安全性的加密传输协议。
* HTTP与HTTPS的连接方式也不一样，用的端口也不一样，HTTP是80，HTTPS是443。

HTTP特点：

* **Get和post-两种HTTP请求的基本方法**

Get和post是HTTP的两种请求方式，HTTP是基于TCP/IP通信的协议，所以get和post的底层也是TCP/IP，本质上都是TCP链接，并没有什么差别，但是由于HTTP的规定以及浏览器/服务器的限制，导致他们在应用过程中体现出一些不同。

* Get只产生一个TCP包，post产生两个（重要；但是Firefox中post只发送一次）

Get请求会将HTTP的header和data一起发送出去，服务器再返回数据，post是先发送header，服务器响应100（continue请求者应该继续提出请求）后，再继续发送data，然后服务器再返回结果200。

* Get请求参数包含在URL中，post请求实在request body中设置参数，所以post请求比get请求更加安全；
* Get请求有长度限制，URL最大长度为2048个字符，post没有长度限制；
* Get请求可以被缓存，post不会被缓存
* Get后退没有影响，post会重新提交
* Get只支持URL编码，post支持多种编码方式



* **Websocket**
* HTTP协议有一个缺陷：通信只能有客户端发起。只能是由客户端向服务端发起请求，服务端返回查询结果。HTTP协议做不到服务端主动向客户端推送消息。这种单向请求，如果服务端有连续的状态变化，客户端要获知就比较麻烦，只能通过轮询的方式，每隔一段时间就询问一次服务端信息。轮询效率低，非常浪费资源。
* WebSocket：是一种双向通信协议，服务端主动向客户端推送消息，客户端也可以主动向服务器端发送消息，是真正的双向平等对话。



* **cookie、session**
* cookie是客户端缓存，session是服务端缓存；
* session默认被存在服务器的一个文件里（不是内存）；
* session的运行依赖session\_id，而session\_id是存在cookie中的，也就是说，如果浏览器禁用了cookie，同时session也会失效（但是可以通过其他方式实现，比如url中传递session\_id）；
* session可以放在文件、数据库、或内存中都可以；
* 用户验证这种场合一般会用session。
* cookie 就是用户通行证（用户名片），session就像用户档案信息表（用户档案信息），里面包含了用户认证信息和登录状态等信息。
* cookie一般存储的数据不超过4kb，session没有对储存数据量的限制，可以保存更为复杂的数据类型。

Session虽然是保存在服务器，对客户端是透明的，但是它的正常运行仍然需要客户端的支持。这是因为session需要使用cookie作为识别标志，HTTP协议是无状态的，所以session不能根据HTTP连接来判断是否为同一个客户，因此服务器向客户端发送一个名为JSESSIONID的cookie，他的值为该session的id，session依据该cookie来识别是否为同一个用户，所以如果浏览器禁用cookie，session也会失效。

用户连接服务器时，会由服务器生成一个sessionID，用该ID作为标识符来存取服务端的session存储空间，而sessionID是存储在客户端cookie中的，用户提交页面时，会将sessionID提交给服务端，来存取session的数据。服务器根据当前的session\_id来判断相应的用户数据标志，以判断用户是否登录或者是否具有某种权限。

* **前端跨域**

跨域一般由后端实现，前端实现方式：JSONP

优点：简单适用，兼容性好；缺点：支持get不支持post。

核心思想：网页通过添加一个script元素（script的src是支持跨域请求的），向服务器请求JOSN数据，服务器收到请求后，将数据放在一个指定名字的回调函数的参数位置传回。

JQuery ajax：将dataType设置jsonp即可。

$.ajax({

url: 'http://www.test.com:8080/login',

type: 'get',

dataType: 'jsonp', // 请求方式为jsonp

jsonpCallback: "handleCallback", // 自定义回调处理函数

data: {}，

jsonp:callback, //传递给请求处理程序或者页面的，用以获得jsonp回调函数名的参数名（一般默认为callback）

success(data){}

});

如果不想再success中处理返回数据，可通过设置：

* jsonp: “callback”, //传递给服务端的回调函数参数名，如果不设置此项，则默认是"callback"
* jsonpCallback: “handleCallback”, //传递给服务端的回调函数名称，如果不设置此项，则默认是形如"jQuery111007837897759742043\_1460657212499"的由jQuery自动生成的函数名称，会自动在$.ajax方法的success中完成调用。

jsonpCallback函数设置后，服务端返回的数据会先调用该回调函数的代码，然后再执行success中的代码，一般情况下不需要设置该参数，在success中处理即可。

* **CSS总结**
* **CSS动画属性**

1. Transform 适用于2D或3D转换的元素

2D转换方法：





**transition过渡**

从一种样式逐渐过渡到另一种样式。要实现这个过渡过程，必须规定两项内容：

* 指定要添加过渡效果的css属性（width等）
* 指定过渡时间（默认为0）

transition需要事件触发（比如点击，悬浮），无法自动加载；是一次性的，不能重复发生，除非一再触发；只能定义开始和结束状态；

例子：

对于一个元素初始样式，和设置悬浮width变化成另一个值，这个width变化过程，使用过渡属性，动画过渡。

div { width:100px, height: 100px, transition: width 2s height:2s}

div:hover { width:200px, height:300px}

**Animation 动画**

**利用@keyframes定义关键帧的样式**

使用@keyframes创建动画，然后把它绑定在一个选择器上，不然动画不会产生任何效果。

@keyframes定义各个状态变化可以使用from，to；也可以使用百分比，Animation可以无限次触发。

例子：

div {

width:100px;

height: 100px;

animation: myAnimation 5s;

}

@keyframes myAnimation {

0% {background:red;};

25% {background:yellow;};

50% {background:blue;};

100% {background:green;};

}

* **CSS盒子模型**

盒子模型包括：content实际内容、paddding内边距、border边框、margin外边距。

盒子模型主要包括w3c盒子模型（标准盒子模型）和IE盒子模型。

标准盒子模型：css中的宽/高 = content的宽/高；

IE盒子模型：css中的宽/高 = content的宽/高 + (padding + border) \* 2 。

* **CSS清除浮动**

简单来说，浮动是因为使用了float：left/right而产生的浮动。

浮动导致的副作用：对子元素设置浮动，会导致父元素不能撑开，就会不能显示背景、父元素的边框也不会撑开、margin/padding不能正确显示。

清除浮动：

1. 设置父元素合适的高度；

2. 设置父元素overflow：hidden；

3. clear：both清除浮动（新建一个子元素并将样式设置为clear：both）

伪元素：

::before/::after，在一个元素前面/后面插入一个元素及设置样式，此元素默认为行内元素。

* **px、em、rem、vh字体单位**

- px是像素单位，是固定的，一旦设置就无法适应页面大小而改变。

- em和rem是相对长度单位，长度不是固定的，更适应于响应式布局。

- em是相对于父元素来设置大小的（参考物是父元素的“font-size”），这样就会存在一个问题，进行元素设置时，都有可能需要知道它父元素的大小。

- 而rem是相对于根元素<html>的，这样就只需要在根元素确定一个参考值。

- vw/vh 全程是viewport width 和 viewport height，视图的宽度和高度，1vw/vh相当于屏幕宽度和高度的1%，不过处理宽度的时候%更合适，处理高度的话vh更合适。

项目中的应用场景（个人）：

- 像素（px）：用于设置元素的边框或者定位。

- em、rem适应于响应式的页面，可能更倾向于rem，因为em是相对于父元素，计算的时候可能比较麻烦。而rem相对于根元素HTML，计算更清晰。

- em会继承父元素的大小，1em/rem=16px(浏览器默认大小)

* **calc、support/media各自的含义及用法：**

@support主要是用于检测浏览器是否支持css的某个属性，其实就是条件判断，如果支持某个属性，你可以写一套样式，如果不支持某个属性，也可以提供另一套样式。

calc()函数用于动态计算长度值。

@media查询，可以针对不同的媒体类型定义不同的样式。

* **元素水平、垂直居中：**

1. 绝对定位+margin：auto

parent {

position:relative;

}

child{

position:absolute;

width:30px;

height:30px;

top:0;

bottom:0;

left:0;

right:0; // top/bottom/left/right设置为相同的值，加上margin：auto即可实现上下左右平等分配

margin:auto

}

1. 绝对定位+translate()平移，可以不设置元素宽高

child{

position:absolute;

top:50%;

left:50%;

transform:translate(-50%, -50%);

}

1. 绝对定位+负margin，元素需要设置宽高，margin的上/左设置为高/宽的一半。

child{

position:absolute,

width:100px;

height:100px;

top:50%,

left:50%;

margin:-50px 0px -50px 0px;

}

1. flex布局

child{

display:flex;

align-items: center; // 垂直居中

justify-content: center; // 水平居中

}

1. table-cell

child{

display:table-cell;

text-align:center; //水平居中

vertical-align: middle;// 非必须，垂直方向上居中

}

* **CSS画一个三角形**

元素的border是一个梯形结构，所以通过设置元素的border可以得到不同的三角形。

例如：

.border{

width: 0px;

height: 0px;

border: 50px solid;

border-color: #96ceb4 #ffeead #d9534f #ffad60;

}



当元素宽高设置为0时，为上述形状，此时就可以通过设置border来得到想要的三角形，如想得到下方红色三角形，将上左右设置为透明即可。

.border{

border-color: transparent transparent #d9534f;

width: 0px;

height: 0px;

border-width: 0 50px 50px; //将上边框的宽度设置为0

}



* **Label标签**

Label标签通常与input标签一起使用，label标签为input元素定义标注（标记）。

Label标签为鼠标用户提供了可用性，当用户点击label标签中的文本时，浏览器会自动将焦点转到和该标签相关联的控件上，扩大了点击范围，提高用户体验。

* **CSS 元素**

CSS元素分为三种：行内元素（inline）、块级元素（block）、行级块元素（inline-block）。

行内元素：

常见行内元素：<a>、<span>、<i>、<em>、<strong>、<label>、<q>、<var>、<cite>、<code>。

1. 和其他元素在一行上，不可以自动换行；
2. 元素的高度、宽度、padding、margin的上下边距设置无效，左右可以设置。
3. 元素宽度即为内容的宽度，不可修改。
4. 可以通过display变成其他元素

块级元素：

常见块级元素：

<div>、<p>、<h1>...<h6>、<ol>、<ul>、<stable>、<address>、<form>

特点：

1. 独占一行，可以自动换行
2. 宽、高以及边距都可以设置
3. 宽度默认是容器的100%

行级块元素：

既有行内元素的特点，也有块级元素的特点。常见元素：<img>、<input>。

特点：

1. 和其他元素在一行，不可以自动换行
2. 元素的宽、高以及边距都可以设置

* **position**

position：static（默认值）

正常流，元素在文档流中当前的布局位置，此时的top、right、bottom、left属性无效，不脱离文档流。

position：relative

相对定位，相对于自身静态位置定位。不脱离文档流。

Position： absolute

绝对定位，相对于最近的父元素，且该父元素的position不为static。脱离文档流。

Position：fixed

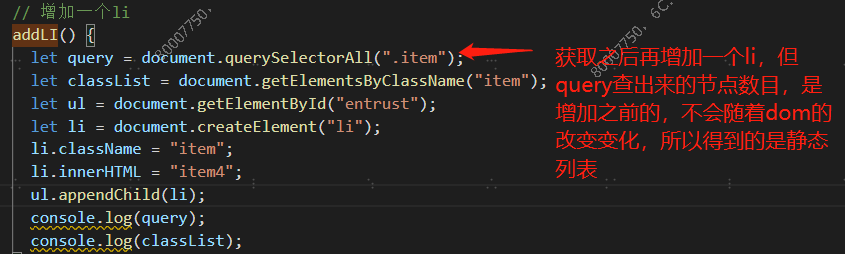
固定定位，相对于浏览器窗口进行定位，在拖拽页面滚动条时，该元素不会随之一起移动。

脱离文档流导致的问题：absolute以及fixed会导致元素脱离文档流，就会产生相应的问题：子元素脱离文档流，父元素无法撑开。

解决办法：给父元素设置高度等于子元素的高度 。

* **querySelectorAll和getElementsByClassName获取元素的区别**

querySelectAll(‘.className’)—参数是css选择器；获取的是静态节点列表，不会动态随DOM发生变化，不影响文档操作。



而getElementsByClassName得到的是动态节点列表，是动态查询的过程，会随着DOM结构的变化，得到的节点列表也会随之变化。



从性能上来讲，getElementByClass要更快。

* **Css box-sizing:**

Css中设置元素宽高时，如果元素有border和padding，再元素布局时就要时刻注意border和padding的值，而box-sizing就可以调整这个。

box-sizing有两个属性：

content-box：是默认值。如果你设置一个元素的宽为100px，那么这个元素的内容区会有100px 宽，并且任何边框和内边距的宽度都会被增加到最后绘制出来的元素宽度中。（border这些值不会覆盖content部分，相当于标准盒子模型的宽度）

border-box ：告诉浏览器：你想要设置的边框和内边距的值是包含在width内的。也就是说，如果你将一个元素的width设为100px，那么这100px会包含它的border和padding，内容区的实际宽度是width减去(border + padding)的值。大多数情况下，这使得我们更容易地设定一个元素的宽高。（相当于设置的border这些值，是盖在content上的，不是包裹，相当于IE盒子模型的宽度（包含content、border、padding不包含margin外边距）））