自我介绍 围绕项目提问，vue和angular双向绑定的核心原理及其区别。vue是数据劫持，angular是脏值检查。

vue路由导航的生命周期。记住几个路由钩子。

项目难点，怎么解决，

css和js基础 选择器，权重，垂直居中，闭包，继承，封装插件，ztree和echarts为什么使用某项技术，http和https1

根据项目技术细节题出问题，json数据处理

页面布局

写一个html5页面 ajax，cookie和session区别 网络的七层协议

显示器分辨率不同如何让网页适应各种显示屏 页面刷新

兼容性问题

js全通

css两栏布局，垂直居中，清除浮动 js变量提升，重载，数据拷贝 求中位数的题

slider html和css怎么写

我遇到的，二叉数中序遍历，meta参数，字符串取最后三个字节（几种方法实现），异步发送A、B、C,如何做让A、B、C成功发送后，再发送D。还有个布局问题。有个程序考验对this的理解 问了闭包定时器就过了

看简历问问题

vue相关 如何看待mvc框架，如何介绍mvc框架，es6新特征 考算法

来海康能为其做什么 使用技术栈

期望薪资

VUE三大要素

1.响应式，监听到数据变化

vue.js是采用数据劫持，并**结合发布者——订阅者**的模式：通过Object.defineProperty()来劫持vue中各个属性的setter、getter，**在数据变动的时候，再发布消息给订阅者**

**答到关键点就好了，通过 getter 和 setter 劫持了对对象赋值的过程，在这个过程中可以进行更新 dom 操作等等。**

**Vue 的双向绑定只是一个语法糖**

**订阅者模式**

使用自定义的data属性在HTML代码中指明绑定。所有绑定起来的JavaScript对象以及DOM元素都将“订阅”一个发布者对象。任何时候如果JavaScript对象或者一个HTML输入字段被侦测到发生了变化，我们将代理事件到发布者-订阅者模式，这会反过来将变化广播并传播到所有绑定的对象和元素。**在vue里，就是把数据订阅到标签的data-xxx 里.再由**

Object.defineProperty(obj,prop, descriptor)劫持数据,该方法接收3个参数   
obj： 要在其上定义属性的对象。 prop： 要定义或修改的属性的名称。   
descriptor： 给对象的属性添加特性描述，目前提供两种形式：数据描述和存取器描述。   
var obj = {};

Object.defineProperty(obj, 'hello', {

get: function() {

console.log('get val:'+ val);

return val;

　 },

　　set: function(newVal) {

val = newVal;

console.log('set val:'+ val);

}

});

返回值为第一个参obj

第三个参里，{}可选配置，根本上，这个Object.definePropety方法是为obj写入或修改属性

1. writable:false默认，值是否可重写

#### 2.enumerable：false,是否可被使用for...in或Object.keys()）枚举

#### 3. configurable：false，是否可被删除和重新设置

如果用了本方法也没有任何配置，则全false

手写实现

**第一步：需要observe的数据对象进行递归遍历**，包括子属性对象的属性，都加上 setter和getter  
这样的话，给这个对象的某个值赋值，就会触发setter，那么就能监听到了数据变化

**第二步：compile解析模板指令**，将模板中的变量替换成数据，然后初始化渲染页面视图，并将每个指令对应的节点绑定更新函数，添加监听数据的订阅者，一旦数据有变动，收到通知，更新视图

**第三步：Watcher订阅者是Observer和Compile之间通信的桥梁**，主要做的事情是:  
1、在自身实例化时往属性订阅器(dep)里面添加自己  
2、自身必须有一个update()方法  
3、待属性变动dep.notice()通知时，能调用自身的update()方法，并触发Compile中绑定的回调，则功成身退。

**第四步：MVVM作为数据绑定的入口，整合Observer、Compile和Watcher三者**，通过Observer来监听自己的model数据变化，通过Compile来解析编译模板指令，最终利用Watcher搭起Observer和Compile之间的通信桥梁，达到数据变化 -> 视图更新；视图交互变化(input) -> 数据model变更的双向绑定效果。

1.基本配置

window.onload = function() { var app = new myVue({

el:'#app', data: { number: 0 }, methods: { increment: function() { console.log(this.number)}, } }) }

2.定义vue

function myVue(options) { this.\_init(options); }

myVue.prototype.\_init = function (options)

{ this.$options = options; this.$el = document.querySelector(options.el); this.$data = options.data; this.$methods = options.methods;//遍历配置的属性

this.\_binding = {};绑定 this.\_obverse(this.$data); //劫持监听属性

this.\_complie(this.$el); }//解析指令

3.定义obverse劫持

myVue.prototype.\_obverse = function (obj) { //**obj=this.$data**

var value;

for (key in obj) { //对data中每个项做操作

if (obj.hasOwnProperty(key)) { //检验可否检索到key

this.\_binding[key] = { \_directives: [] }; //数据结构为 \_binding:{key:\_directives:[],......}

value = obj[key];

if (typeof value === 'object')

{ this.\_obverse(value); }//data中如果是对象则迭代

var binding = this.\_binding[key]; //则binding=\_directives：[],

**Object.defineProperty(this.$data, key, { enumerable: true, configurable: true,**

**//即为获取值时触发函数**

**get: function () { console.log(`获取${value}`); return value; },**

**//为修改值时触发函数**

**set: function (newVal) { console.log(`更新${newVal}`);**

**if (value !== newVal)**

**{ value = newVal;**

**binding.\_directives.forEach(function (item) { item.update(); }**)//如果数据改变，则调用更新方法

} } }) } } }

4.定义解析指令，数据绑定到指令

myVue.prototype.\_complie =function(root) { root = el

var\_this =this;

varnodes = root.children;//el.children

for(vari = 0; i < nodes.length; i++) {

Var node = nodes[i];

if(node.children.length) {

this.\_complie(node);//节点还有子节点则迭代

}

if(node.hasAttribute('v-click')) {

node.onclick = (function() {

Var attrVal = nodes[i].getAttribute('v-click');

Return \_this.$methods[attrVal].bind(\_this.$data);

})();

}

if(node.hasAttribute(**'v-bind**')) {

varattrVal = node.getAttribute('v-bind');

\_this.\_binding[attrVal].\_directives.push(**new Watcher**(

'text',

node,

\_this,

attrVal,

'innerHTML'

))//结合下面，相当于 this.node[innerHTML]=\_this.$data[this.attrVal]

}

}

}

function Watcher(name, el, vm, exp, attr) {

this.name = name; //指令名称，例如文本节点，该值设为"text"

this.el = el; //指令对应的DOM元素

this.vm = vm; //指令所属myVue实例

this.exp = exp; //指令对应的值，本例如"number"

this.attr = attr; //绑定的属性值，本例为"innerHTML"

this.update();

}

//watcher+update 等于简单更新 el的html里的订阅attr标签相应属性值

Watcher.prototype.update =function() {

this.el[this.attr] =this.vm.$data[this.exp];}

<keep-alive></keep-alive> 包裹动态组件时，会缓存不活动的组件实例,主要用于保留组件状态或避免重新渲染

生命周期钩子 **activated: keep-alive组件激活时调用**

由于 Vue 不允许动态添加根级响应式属性，所以你必须在初始化实例前声明根级响应式属性，哪怕只是一个空值，如果你未在 data 选项中声明 message，Vue 将警告你渲染函数正在试图访问的属性不存在。

This.$nextTick()

Vue 异步执行 DOM 更新。只要观察到数据变化，Vue 将开启一个队列，并缓冲在同一事件循环中发生的所有数据改变。如果同一个 watcher 被多次触发，只会被推入到队列中一次。如果要立即操作更新后的DOM，则使用nextTick

**nextTick的触发时机：**

　　　　在同一事件循环中的数据变化后，DOM完成更新，立即执行nextTick(callback)内的回调。

用途：

1.在 created 和 mounted 阶段，如果需要操作渲染后的试图，也要使用 nextTick 方法。

mounted: function () {

this.$nextTick(function () {})}

1. dom修改后

点击按钮显示原本以 v-show = false 隐藏起来的输入框，并获取焦点。

showsou(){

this.showit = true //修改 v-show

document.getElementById("keywords").focus() //在第一个 tick 里，获取不到输入框，自然也获取不到焦点

}

修改为：

showsou(){

this.showit = true

this.$nextTick(function () {

// DOM 更新了

document.getElementById("keywords").focus()

})

}

根本上来讲，nextTick就是把个中事件放入下一次事件循环里。

 **created:** 实例已经创建完成之后调用,在这一步,实例已经完成数据观测, 属性和方法的运算, watch/event事件回调. 然而, 挂载阶段还没有开始, $el属性目前还不可见

 **mounted:** el被新创建的 vm.$el 替换，并挂载到实例上去之后调用该钩子。如果 root 实例挂载了一个文档内元素，当 mounted 被调用时 vm.$el 也在文档内。

 **activated:**:keep-alive组件激活时调用

#### 第一次页面加载会触发哪几个钩子

答：第一次页面加载时会触发 beforeCreate, created, beforeMount, mounted 这几个钩子

**请详细说下你对vue生命周期的理解？**

答：总共分为8个阶段创建前/后，载入前/后，更新前/后，销毁前/后。

创建前/后： 在beforeCreate阶段，vue实例的挂载元素el和数据对象data都为undefined，还未初始化。在created阶段，vue实例的数据对象data有了，el还没有。

载入前/后：在beforeMount阶段，vue实例的$el和data都初始化了，但还是挂载之前为虚拟的dom节点，data.message还未替换。在mounted阶段，vue实例挂载完成，data.message成功渲染。

更新前/后：当data变化时，会触发beforeUpdate和updated方法。

销毁前/后：在执行destroy方法后，对data的改变不会再触发周期函数，说明此时vue实例已经解除了事件监听以及和dom的绑定，但是dom结构依然存在

在谈到Vue的生命周期的时候，我们首先需要创建一个实例，也就是在 new Vue ( ) 的对象过程当中，首先执行了init（init是vue组件里面默认去执行的），在init的过程当中首先调用了beforeCreate，然后在injections（注射）和reactivity（反应性）的时候，它会再去调用created。所以在init的时候，事件已经调用了，我们在beforeCreate的时候千万不要去修改data里面赋值的数据，最早也要放在created里面去做（添加一些行为）。

当created完成之后，它会去判断instance（实例）里面是否含有“el”option（选项），如果没有的话，它会调用vm.$mount(el)这个方法，然后执行下一步；如果有的话，直接执行下一步。紧接着会判断是否含有“template”这个选项，如果有的话，它会把template解析成一个render function ，这是一个template编译的过程，结果是解析成了render函数：

render (h) {

return h('div', {}, this.text)

}

render函数里面的传参h就是Vue里面的createElement方法，return返回一个createElement方法，其中要传3个参数，第一个参数就是创建的div标签；第二个参数传了一个对象，对象里面可以是我们组件上面的props，或者是事件之类的东西；第三个参数就是div标签里面的内容，这里我们指向了data里面的text。

使用render函数的结果和我们之前使用template解析出来的结果是一样的。render函数是发生在beforeMount和mounted之间的，这也从侧面说明了，在beforeMount的时候，$el还只是我们在HTML里面写的节点，然后到mounted的时候，它就把渲染出来的内容挂载到了DOM节点上。这中间的过程其实是执行了render function的内容。

在使用.vue文件开发的过程当中，我们在里面写了template模板，在经过了vue-loader的处理之后，就变成了render function，最终放到了vue-loader解析过的文件里面。这样做有什么好处呢？原因是由于在解析template变成render function的过程，是一个非常耗时的过程，vue-loader帮我们处理了这些内容之后，当我们在页面上执行vue代码的时候，效率会变得更高。

beforeMount在有了render function的时候才会执行，当执行完render function之后，就会调用mounted这个钩子，在mounted挂载完毕之后，这个实例就算是走完流程了。

后续的钩子函数执行的过程都是需要外部的触发才会执行。比如说有数据的变化，会调用beforeUpdate，然后经过Virtual DOM，最后updated更新完毕。当组件被销毁的时候，它会调用beforeDestory，以及destoryed。

这就是vue实例从新建到销毁的一个完整流程，以及在这个过程中它会触发哪些生命周期的钩子函数。那说到这儿，可能很多童鞋会问，钩子函数是什么意思？

钩子函数，其实和回调是一个概念，当系统执行到某处时，检查是否有hook，有则回调。说的更直白一点，每个组件都有属性，方法和事件。所有的生命周期都归于事件，在某个时刻自动执行。

其实，当你跟面试官阐述到这儿的时候，面试官基本上已经满意你的回答了，隐约看到了你的技术功底。当然，如果你还想更进一步，让面试官对你刮目相看，达到加分的效果，你还可以这样说：

在这个过程当中，Vue为我们提供了renderError方法，这个方法只有在开发的时候它才会被调用，在正式打包上线的过程当中，它是不会被调用的。它主要是帮助我们调试render里面的一些错误。

   beforecreate : 可以在这加个loading事件，在加载实例时触发

· created : 初始化完成时的事件写在这里，如在这结束loading事件，异步请求也适宜在这里调用

·  mounted : 挂载元素，获取到DOM节点

·  updated : 如果对数据统一处理，在这里写上相应函数

·  beforeDestroy : 可以做一个确认停止事件的确认框

·  nextTick : 更新数据后立即操作dom

**vue的整个实现流程**

1.解析模板成render函数

template 创建标签的过程，并解析指令，如v-modal，v-for之类的绑定和处理，render**函数返回的是一个vnode节点**。模板中所有信息被render函数包含，

2.响应式开始监听

Object.defineProperty,然后把所有data属性代理到vm（即new出的vue实例）上。

3.首次渲染，显示页面，绑定依赖

首次渲染，没有则把vnode加入容器，有则执行后面的比较过程，

则执行render函数时，会访问到data数据get，于是触发响应

在get上监听是由于，data数据并不是所有都会直接用到页面上，不用到的不会触发get，则其set也不用管，于是可以避免多余的渲染。

修改属性被set监听到，set中执行updateComponent，触发比较diff函数，更新vnode



4.data变化，触发rerender

VUEX

一个应用可以看作是由上面三部分组成: **View, Actions,State**,数据的流动也是从View => Actions => State =>View 以此达到数据的单向流动.但是项目较大的, 组件嵌套过多的时候, 多组件共享同一个State会在数据传递时出现很多问题.Vuex就是为了解决这些问题而产生的.

Vuex可以被看作项目中所有组件的数据中心,我们将所有组件中共享的State抽离出来,任何组件都可以访问和操作我们的数据中心.

**组件之间的传值？**

**1:父组件与子组件传值**

父组件通过标签上面定义传值  
子组件通过props方法接受数据

**2:子组件向父组件传递数据** 子组件通过$emit方法传递参数

### 为什么使用key？

当有相同标签名的元素切换时，需要通过 key 特性设置唯一的值来标记以让 Vue 区分它们，否则 Vue 为了效率只会替换相同标签内部的内容。

当有相同标签名的元素切换时，需要通过 key 特性设置唯一的值来标记以让 Vue 区分它们，否则 Vue 为了效率只会替换相同标签内部的内容。

### VNode是什么？虚拟 DOM是什么？

Vue在 页面上渲染的节点，及其子节点称为“虚拟节点 (Virtual Node)”，简写为“VNode”。“虚拟 DOM”是由 Vue 组件树建立起来的整个 VNode 树的称呼。

虚拟dom在于以js模拟dom操作，最大避免dom的渲染，dom操作非常昂贵，从而最好的性能。js效率高。

vdom为何使用diff算法

1.dom节点昂贵2.找出本次dom需要更新的节点去更新，其余不更新

**Vdom核心：1.h函数2.patch函数**

h作为createElement的别名

**new** Vue({

el: '#root',

render: h => h(App)

})

**等同于下面，可见h是直接给绑定节点渲染一个vue组件**

**new Vue({**

**el: '#root',**

**template: '<App></App>',**

**components: {**

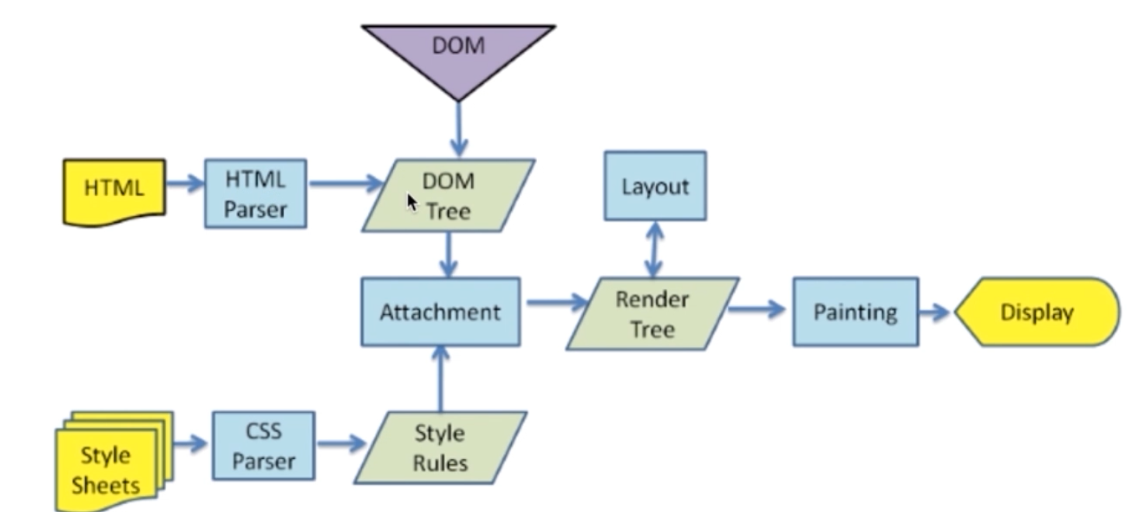
**App**

**}**

**})**

浏览器渲染过程

dom tree css tree =>render tree



重排reflow：

盒子找正确位置。增删dom，移动修改dom，增加css样式，resize窗口，修改字体触发

重绘repaint

盒子各种颜色大小字体确定后，绘制一遍，dom改动和css改动触发。

如何避免最小repaint 不要一个一个添加节点，把他们先接到父节点再加入

第一步，用HTML分析器，分析HTML元素，构建一颗DOM树。

第二步：用CSS分析器，分析CSS文件和元素上的inline样式，生成页面的样式表。

第三步：将上面的DOM树和样式表，关联起来，构建一颗Render树。这一过程又称为Attachment。每个DOM节点都有attach方法，接受样式信息，返回一个render对象（又名renderer）。这些render对象最终会被构建成一颗Render树。

第四步：有了Render树后，浏览器开始布局，会为每个Render树上的节点确定一个在显示屏上出现的精确坐标值。

第五步：Render数有了，节点显示的位置坐标也有了，最后就是调用每个节点的paint方法，让它们显示出来。

普通虚拟dom

var element = {

tagName: 'ul', // 节点标签名

props: { // DOM的属性，用一个对象存储键值对

id: 'list'

},

children: [ // 该节点的子节点

{tagName: 'li', props: {class: 'item'}, children: ["Item 1"]},

{tagName: 'li', props: {class: 'item'}, children: ["Item 2"]},

{tagName: 'li', props: {class: 'item'}, children: ["Item 3"]},

]

}

对应

<ul id='list'>

<li class='item'>Item 1</li>

<li class='item'>Item 2</li>

<li class='item'>Item 3</li>

</ul>

用 JavaScript 对象结构表示 DOM 树的结构；然后用这个树构建一个真正的 DOM 树，插到文档当中 当状态变更的时候，重新构造一棵新的对象树。然后用新的树和旧的树进行比较，记录两棵树差异 把2所记录的差异应用到步骤1所构建的真正的DOM树上，视图就更新了

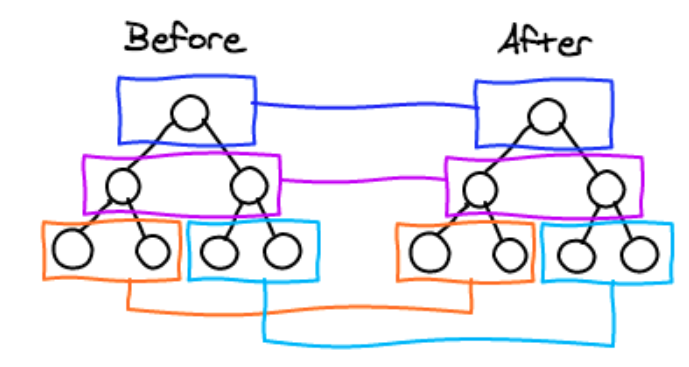
使用document.CreateElement 和 document.CreateTextNode创建的真实节点。

**虚拟dom的diff算法**

设置key和不设置key的区别：  
**不设key，newCh和oldCh只会进行头尾两端的相互比较，设key后，除了头尾两端的比较外，还会从用key生成的对象**oldKeyToIdx**中查找匹配的节点，所以为节点设置key可以更高效的利用dom。**

**分析diff算法 它是linux基本命令，找文本不同，找节点不同**

**比较只会在同层级进行, 不会跨层级比较。**



因为只同层比较，则改变某个元素的层级，则是删掉原有再加上新的这种操作。

**diff的过程就是调用patch函数，就像打补丁一样修改真实dom**。

总的来说，就是顺着节点递归对比

**diff 的实现主要通过两个方法，patchVnode 与 updateChildren 。**

**patchVnode主要分五种情况：**

1.if (oldVnode === vnode)，他们的引用一致，可以认为没有变化。

2. if(oldVnode.text !== null && vnode.text !== null && oldVnode.text !== vnode.text)，文本节点的比较，需要修改，则会调用Node.textContent = vnode.text。

3.if( oldCh && ch && oldCh !== ch ), **两个节点都有子节点，而且它们不一样，这样我们会调用 updateChildren 函数比较子节点，这是diff的核心，后边会讲到。**

4.if (ch)，只有新的节点有子节点，调用createEle(vnode)，vnode.el已经引用了老的dom节点，createEle函数会在老dom节点上添加子节点。

5.if (oldCh)，新节点没有子节点，老节点有子节点，直接删除老节点。

**updateChildren 是关键，这个过程可以概括如下**：

oldCh 和 newCh 各有两个头尾的变量 StartIdx 和 EndIdx ，它们的2个变量相互比较，一共有4种比较方式。如果 4 种比较都没匹配，如果设置了key，就会用key进行比较，在比较的过程中，变量会往中间靠，一旦 StartIdx > EndIdx 表明 oldCh 和 newCh 至少有一个已经遍历完了，就会结束比较。

**为什么使用key？**

当有相同标签名的元素切换时，需要通过 key 特性设置唯一的值来标记以让 Vue 区分它们，否则 Vue 为了效率只会替换相同标签内部的内容。

**为什么避免 v-if 和 v-for 用在一起**

当 Vue 处理指令时，v-for 比 v-if 具有更高的优先级，这意味着 v-if 将分别重复运行于每个 v-for 循环中。通过v-if 移动到容器元素，不会再重复遍历列表中的每个值。取而代之的是，我们只检查它一次，且不会在 v-if 为否的时候运算 v-for。

**自定义指令（v-check、v-focus）的方法有哪些？它有哪些钩子函数？还有哪些钩子函数参数？**

答：全局定义指令：在vue对象的directive方法里面有两个参数，一个是指令名称，另外一个是函数。组件内定义指令：directives

钩子函数：bind（绑定事件触发）、inserted(节点插入的时候触发)、update（组件内相关更新）

钩子函数参数：el、binding

**vue-router是什么？它有哪些组件？**

答：vue用来写路由一个插件。router-link、router-view

**导航钩子有哪些？它们有哪些参数？**

答：导航钩子有：a/全局钩子和组件内独享的钩子。b/beforeRouteEnter、afterEnter、beforeRouterUpdate、beforeRouteLeave

参数：有to（去的那个路由）、from（离开的路由）、next（一定要用这个函数才能去到下一个路由，如果不用就拦截）最常用就这几种

**聊聊你对Vue.js的template编译的理解？ https://www.jianshu.com/p/e1669afa30b8**

答：简而言之，就是先转化成AST树，再得到的render函数返回VNode（Vue的虚拟DOM节点）

详情步骤：

首先，通过compile编译器把template编译成AST语法树（abstract syntax tree 抽象语法树  即 源代码的抽象语法结构的树状表现形式），compile是createCompiler的返回值，createCompiler是用以创建编译器的。另外compile还负责合并option。

然后，AST会经过generate（将AST语法树转化成render funtion字符串的过程）得到render函数，render的返回值是VNode，VNode是Vue的虚拟DOM节点，里面有（标签名、子节点、文本等等）

**Vue complier 实现**

模板解析这种事，本质是将数据转化为一段 html ，最开始出现在后端，经过各种处理吐给前端。随着各种 mv\* 的兴起，模板解析交由前端处理。

总的来说，Vue complier 是将 template 转化成一个 render 字符串。 可以简单理解成以下步骤：

- parse 过程，将 template 利用正则转化成 AST 抽象语法树。

- optimize 过程，标记静态节点，后 diff 过程跳过静态节点，提升性能。

- generate 过程，生成 render 字符串。

VUE-Router

路由懒加载

const Recommend = (resolve) => {

import('components/recommend/recommend').then((module) => {

resolve(module)

})

}

<router-link :to="{name: 'saveJob', params: {job\_id: props.row.job\_id}}" tag="span">

<el-button type="info" size="small" icon="edit">修改</el-button>

</router-link>

<el-button v-if="props.row.job\_id!=get\_user\_info.user.jobid" type="danger" size="small" icon="delete" @click="delete\_job(props.row.job\_id)">删除</el-button>

{

path: '/table/saveJob/:job\_id',

name: 'saveJob',

component: SaveJob,

meta: {

title: "数据修改",

auth: true

}

},

### vue-router有哪几种导航钩子？

三种，一种是全局导航钩子：router.beforeEach(to,from,next)，作用：跳转前进行判断拦截。  
第二种：组件内的钩子；  
第三种：单独路由独享组件

const router = new VueRouter({

routes,

mode: 'hash', //default: hash ,history

scrollBehavior (to, from, savedPosition) {

if (savedPosition) {

return savedPosition

} else {

return {x: 0, y: 0}

}

}

})

router.beforeEach((to, from, next) => {

let toName = to.name

let is\_login = store.state.user\_info.login

if (!is\_login && toName !== 'login') {

next({

name: 'login'

})

} else {

if (is\_login && toName === 'login') {

next({

path: '/'

})

} else {

next()

}

}

})

CSS

1基本选择器2组合选择器3属性选择器4伪类选择器5伪元素选择器

基本：.class #id \*所有 element元素

组合：多元素 a,b 后代a b 子元素a>b 直接相邻 a+b 匹配选择a~b之后选择同级

属性选择器[a]带有属性a的元素 [a=xxx]属性a等于xxx的元素 其中 a~=xxx属性包含字段，a^=xxx起始，a$=xxx结尾 a\*=xxx包含

伪类 ：link所有未访问链接 ：visited所有访问过的链接 ：hover ：active正在活动的链接 ：focus选择元素输入后具有焦点

伪类元素 ：：before/:before 元素前插入内容在content，不在文档树里，：after同上,:first-letter元素中文本首字母，:first-line元素文本第一行，只能用于块级元素

优先级：内联样式1000 > ID 选择器100 > 类选择器10 = 属性选择器 = 伪类选择器 > 元素（类型）选择器 1= 伪元素选择器

计算举例 div.class = 10+1 .test1 .test2 =10+10

var para=document.createElement("p");

var node=document.createTextNode("这是一个新段落。");

para.appendChild(node);

var element=document.getElementById("div1"); element.appendChild(para);

三栏布局

1.float 2.absolute left：0 right:0 3.flex flex：1 4.父元素display:table 子元素display:table-cell 5. 网格布局grid 父元素display:grid; width:100%;grid-template-rows:100px(高);grid-template-columns:300px auto 300px;宽

左定宽右自适应

1.左float 2. 左absolute 右margin-left3.flex 右flex：1 4.5.同上

左侧随右侧高度变化而变化

右float：right height:auto min-height:100%

左 height:100%**;overflow:hidden;**

**padding-bottom:9999px;margin-bottom:-9999px;**float:left

**布局 水平居中**

1) 若是行内元素, 给其父元素设置 text-align:center,即可实现行内元素水平居中.  
2) 若是块级元素, 该元素设置 margin:0 auto即可.  
3) 若子元素包含 float:left 属性, 为了让子元素水平居中, 则可让父元素宽度设置为fit-content,并且配合margin：0 auto,

4) 使用flex , 可以轻松的实现水平居中 justify-content align-item

5) position absolute使用绝对定位方式, 以及负值的margin-left,

以及 transform:-50%

以及left:0;right:0;margin:0 auto;

① text-align:center;② margin:0 auto;③ width:fit-content;④ flex⑤ 盒模型⑥ transform⑦ ⑧ 两种不同的绝对定位方法

**布局 垂直居中**

**单行文本**  
1) 若元素是单行文本, 则可设置 line-height 等于父元素高度  
**行内块级元素**  
2) 若元素是行内块级元素, 基本思想是使用display: inline-block, vertical-align: middle和一个伪元素让内容块处于容器中央.

只有元素属于inline或是inline-block ，vertical-align属性才会起作用。

3. 父元素display:table, 子元素 display:table-cell;vertical-align:middle;

① 单行文本, line-height② 行内块级元素, 使用 display: inline-block, vertical-align: middle; 加上伪元素辅助实现③ vertical-align④ flex⑤ 盒模型⑥ transform

谈谈盒子模型理解

盒模型两种，W3C 标准模型 和IE模型，包括margin、border、padding、content

标准模型宽度width=content的宽度，

IE盒模型元素的width=border + padding + content，

设置style的box-sizing，两个属性content-box和border-box。

Js获取宽高

obj=document.getElementById("test");

1.行内obj.style.height 只取的到内联 obj.curruntStyle.width 都能取到，但ie8不兼容

2.计算 ie: obj.currentStyle.height

window.getComputedStyle(obj,null).height

3.obj.getBoundingClientRect().width 计算元素绝对位置 得四个值left top height width

**BFC块级格式化上下文，解决边距重叠**

如果子元素设置外边距，在没有把父元素变成BFC，父元素也会产生外边距，给父元素添加 overflow：hidden 父元素就变为 BFC，不会随子元素产生外边距，但是父元素的高会变化。 同级元素外边距重叠，取两者绝对值大的,或正负相加

**BFC定位原理（规则）**：1.垂直方向边距重叠2.BFC区域不与浮动元素区域重叠3.BFC为独立元素，外面不会影响里面，里面也不外面4.计算高度时，浮动元素也参与计算

BFC规定了内部的Block Box如何布局。定位方案：

内部的Box会在垂直方向上一个接一个放置。

Box垂直方向的距离由margin决定，属于同一个BFC的两个相邻Box的margin会发生重叠。

每个元素的margin box 的左边，与包含块border box的左边相接触。

BFC的区域不会与float box重叠。

BFC是页面上的一个隔离的独立容器，容器里面的子元素不会影响到外面的元素。

计算BFC的高度时，浮动元素也会参与计算。

**创建BFC**

触发条件，满足之一 根元素，即html float的值不为none（默认） overflow的值不为visible（默认） display的值为inline-block、table-cell、table-caption position的值不为static

**使用场景**

消除边距重叠 从上到下叠了一堆，给相应子元素添加bfc父级别 左右布局给右边一个overflow：auto，即不与左边重叠

html5必有标签

<!doctype html>

<html>

<head>

<title>helloWorld--zxk</title>

</head>

<body>

 </body>

</html>

**JS究极大全**

**DOM**

querySelectorAll 返回的是一个 Static Node List，而 getElementsBy 系列的返回的是一个 Live Node List。动态的 Node List， 每一次调用 lis 都会重新对文档进行查询

1）创建新节点

createDocumentFragment() //创建一个DOM片段

createElement() //创建一个具体的元素

createTextNode() //创建一个文本节点

2）添加、移除、替换、插入

appendChild() //添加

removeChild() //移除

replaceChild() //替换

insertBefore() //插入

3）查找

getElementsByTagName() //通过标签名称

getElementsByName() //通过元素的Name属性的值

getElementById() //通过元素Id，唯一性

dom事件级别 标准 obj=document.getElementById("test")

dom0:obj.onclick=function

dom2:obj.addEventListener("click",function,false（冒泡还是捕获）)

dom3:增加了一些事件类型， 鼠标 键盘 keyup

dom事件模型：捕获和冒泡dom事件流：1.捕获2.目标3.冒泡

捕获的具体流程 window-document-html（document.documentElement）-body

DOM对象事件类 event.preventDefault 阻止默认事件 a跳转

event.stopPropagation

event.stopImmediatePropagation

在按钮a中增加，阻止b，用来事件优先级

事件委托，子元素事件代理到父元素

var toolbar = document.querySelector(".toolbar");

toolbar.addEventListener("click", function(e) {

  var button = e.target;

  if(!button.classList.contains("active"))

    button.classList.add("active");

  else

    button.classList.remove("active");

});

event.currentTarget当前绑定的事件

event.target当前被点击元素

自定义事件

新建event :var eve=new Event("cus"); 或 customEvent("cus",{param1,param2...})

绑定obj.addEventListener("cus",function(){});

触发obj.dispatchEvent(eve);

1、clientWidth 获取网页可视区域宽度（两种用法）

clientHeight 获取网页可视区域高度（两种用法）

调用者不同，意义不同：

盒子调用： 指盒子本身。

body/html调用： 可视区域大小。

2、clientX 鼠标距离可视区域左侧距离（event调用）

clientY 鼠标距离可视区域上侧距离（event调用）

**创建对象的三种方法**

**工厂方法**function createPerson( name, age, gender ) {

var o = {};

o.name = name;

o.age = age;

o.gender = gender;

return o;}

**构造器方法**

function Person(name, age, gender){

this.name = name;

this.age = age;

this.gender = gender;

}

var p = new Person("zhangsan", 19, "男");

# 原型是什么？

在JavaScript中原型是一个prototype对象，用于表示类型之间的关系。

# 原型链是什么？

JavaScript万物都是对象，对象和对象之间也有关系，并不是孤立存在的。对象之间的继承关系，在JavaScript中是通过prototype对象指向父类对象，直到指向Object对象为止，这样就形成了一个原型指向的链条，专业术语称之为原型链。

通过instanceof判断的话，如果在原型链上，恒返回true o3 instanceof Objcet

所以 用child.\_proto\_.constructor ==parent 更加严谨

**New 之后**

1.创建新对象 继承自prototype

2.构造器函数被执行，this指向新实例

3.如果构造器返回对象，则取代new出的结果，如果没有返回对象，则为1创建的对象

**闭包** 是由函数引用其周边状态（ 词法环境 ）绑在一起形成的（封装）组合结构。在 JavaScript 中，闭包在 每个函数被创建时 形成。

闭包就是有权访另一个函数作用域中变量的函数

2.特性：

* 函数嵌套函数；
* 函数内部可以引用函数外部的参数和变量；
* 函数变量和参数不会被垃圾回收机制回收；

3.优缺点：

优点   
- 希望一个变量长期驻扎在内存中   
- 避免全局变量的污染   
- 私有成员的存在   
缺点   
- 常驻内存，增大内存使用量，使用不当回造成内存泄漏；

4.自执行函数的好处

* 隔离作用域，避免全局作用域污染
* 模拟块级作用域

把专用访问函数内部的函数暴露出来才叫闭包

**页面刷新方法**

1. history.go(0)

2. location.reload()

3. location=location

4. location.assign(location)

5. document.execCommand( 'Refresh ')

6. window.navigate(location)

7. location.replace(location)

8. document.URL=location.href

**this指向 bind，apply,call**

1、普通函数中的this指向的是对象，匿名函数中的this指向的是windows，和全局变量一样

2、让匿名函数中的this指向对象的两种方法

可以使用对象冒充强制改变this的指向

将this赋值给一个变量，闭包访问这个变量

**this 永远指向最后调用它的那个对象**

var name = "windowsName";

function a() {

 var name = "Cherry";

 console.log(this.name);   // **windowsName**

 console.log("inner:" + this); // **inner: Window**

fn : function () {

  console.log(this.name);  // windowsName

 }

}

a();

**如果** window.a.fn(); this指向a

**即使赋值过 var b= a.fn; b(); 此时this依然是window，它执行了b**

**改变this指向**

**1使用 ES6 的箭头函数**

箭头函数的 this 始终指向函数定义时的 this，而非执行时。箭头函数中没有 this 绑定，必须通过查找作用域链来决定其值，如果箭头函数被非箭头函数包含，则 this 绑定的是最近一层非箭头函数的 this，否则，this 为 undefined

**2在函数内部使用 \_this = this**

**3call() 和 apply() 这两个方法的基本作用：改变对象的执行上下文，即this的指向**

call（this,param1,param2..）

call方法接受多个参数，第一个参默认为window，后面的依次传入

实现继承

child(x){ parent.call(this,x); this.b=2;}

apply(this,param)

apply方法接受两参数，第一个默认window同上，第二个参数为数组或**类数组**

类数组：形如

{0:1,1:2,2:3，length:3}

我们经常使用的获取dom节点的方法返回的就是一个类数组，在一个方法中使用 arguments关键字获取到的该方法的所有参数也是一个类数组

apply特性可以解开类（似）数组

arr.push.apply( arr, a ); 相当于 arr.push( a[0], a[1] )

同样apply也可以解开数组

**Math.max.apply( null, arr );直接相当于遍历了数组并找出最大值**

**4bind()**

注意，上面apply和call调用它们的必然是function ，之后自执行

bind()方法创建一个新的函数, 当被调用时，将其this关键字设置为提供的值，在调用新函数时，在任何提供之前提供一个给定的参数序列。

可见要再执行

var a ={

name : "Cherry",

fn : function (a,b) {

console.log( a + b)

}

}

var b = a.fn;

b.bind(a,1,2)()   // 3

setTimeout(console.log('a'),0)

console.log('c');

的输出顺序; 就是ac 所以异步，设置setTimeout时间为0是不正确的写法

封装函数 f，使 f 的 this 指向指定的对象

function bindThis(f, oTarget) {

if(f.bind){

return f.bind(oTarget);

} else {

return function(){

return f.apply(oTarget,arguments);

};

}

}

**输入url到页面呈现过程：**

1. DNS解析

DNS解析的过程就是寻找哪台机器上有你需要资源的过程。DNS解析是一个递归查询的过程

1. TCP连接
2. 发送HTTP请求
3. 服务器处理请求并返回HTTP报文
4. 浏览器解析渲染页面
5. 连接结束

**输入url到页面呈现过程：**

什么是DOCTYPE及作用

用来定义xml的文件类型（DTD），浏览器根据它判断用什么协议解析，浏览器模式

布局layout

错误监控类

即时运行错误

1.try catch 2window.onerror

资源加载错误

1.object.onerror 2.performance.getEntries3.error事件处理

HTTP超文本传输协议。

简单快速（统一资源符），灵活（同一数据协议传多种数据），无连接（客户端和服务端两身份，不连接），无状态

**什么是RESTful API？怎么使用?**

答：是一个api的标准，无状态请求。请求的路由地址是固定的，如果是tp5则先路由配置中把资源路由配置好。标准有：.post .put .delete

HTTP就是一个用文本格式描述报文头并用双换行分隔报文头和内容，在TCP基础上实现的请求-响应模式的双向通信协议。

Web客户端常见的又各大厂商的浏览器Web服务器存储Web内容HTTP服务器。

什么是资源？Web资源寄宿在Web服务器上。所有能够提供Web内容的东西都是Web资源。比比如静态文件，能够动态生成内容的软件程序。

MIME类型时一种文本标记，标识一种主要的对象类型和一个特定的子类型，中间由一条斜杠来分隔。如text/html,text/plain,image/ipeg。常见的MIME类型有数百个。

URI：统一资源标识符，在世界范围内唯一标识并定位信息资源。包含URL：统一资源定位符。URL描述了一台特定服务器上某资源的特定位置。URL的第一部分被称为方案（scheme）。说明了访问资源所使用的协议类型。这里使用的时HTTP协议（http://）。URN:统一资源名，URN是作为特定内容的唯一名称使用的，与目前的资源所在地无关。

常见方法get post put将来自客户端的数据存储到一个命名的服务器资源中去delete删除命名资源 head（仅发送命名资源响应中http首部）

200 - 请求成功204无资源返回206成功执行范围请求301 - 资源（网页等）被永久转移到其它URL 302表示暂时转移到其他，头包含localtion内容给新url。304未修改305use proxy必须代理访问 400客户端语法错误 401未授权 404 - 请求的资源（网页等）不存在500 - 内部服务器错误503bad gateway服务器现在无法响应

**http请求报文组成部分**

请求行：包含请求方法、URI、HTTP协议版本信息

请求头 key value值，告诉服务端要的内容 host connetction:keep-alive cache-control

user-agent:浏览器内核 accept：text/html Accept-Encoding,Language Cookie

空行

请求体

 响应报文包含三部分：

状态行：包含HTTP版本、状态码、状态码的原因短语

响应头 Server:nginx Date Content-Type:text/html;Charset=utf-8 connection:keep-alive Expires:时间 Cache-Control Set-Cookie:

空行

响应体

Http协议首部字段？

a、通用首部字段（请求报文与响应报文都会使用的首部字段）

Date：创建报文时间

Connection：连接的管理

Cache-Control：缓存的控制

Transfer-Encoding：报文主体的传输编码方式

b、请求首部字段（请求报文会使用的首部字段）

* Host：请求资源所在服务器
* Accept：可处理的媒体类型
* Accept-Charset：可接收的字符集
* Accept-Encoding：可接受的内容编码
* Accept-Language：可接受的自然语言

c、响应首部字段（响应报文会使用的首部字段）

* Accept-Ranges：可接受的字节范围
* Location：令客户端重新定向到的URI
* Server：HTTP服务器的安装信息

d、实体首部字段request header（请求报文与响应报文的的实体部分使用的首部字段）

* Allow：资源可支持的HTTP方法
* Content-Type：实体主类的类型
* Content-Encoding：实体主体适用的编码方式
* Content-Language：实体主体的自然语言
* Content-Length：实体主体的的字节数
* Content-Range：实体主体的位置范围，一般用于发出部分请求时使用

1、**一次完整的HTTP事务是怎样的一个过程？**

基本流程：

a. 域名解析

b. 发起TCP的3次握手

确认3次

c. 建立TCP连接后发起http请求

d. 服务器端响应http请求，浏览器得到html代码

e. 浏览器解析html代码，并请求html代码中的资源

f. 浏览器对页面进行渲染呈现给用户

HTTP协议类 **持久连接**

使用keep-alive模式 1.1版本支持，使客户端到服务端连接持续有效，客户端后续请求时，避免了重新建立连接

**管线化在持久连接下完成，只有GET、HEAD可以post有限制，服务端不一定支持，要支持。现代浏览器并不默认开启**

**由请求1-响应1-请求2-响应2-请求3**

**变为请求1-请求2-请求3-响应1-响应2-响应3 请求打包过去，响应打包回来**

cookie和session的区别   
什么是cookie   
cookie技术是客户端的解决方案，cookie就是有服务器给客户端的特殊信息，而这些信息以文本文件的方式存放在客户端，然后客户端每次向服务器发送请求的时候都会带上这些特殊的信息（在okhttp中设置cookie需要用到拦截器来完成）   
什么是session   
session是另一种记录客户状态的机制，不同的是cookie保存在客户端浏览器中，而session保存在服务器上，客户端浏览器访问服务器的时候，服务器把客户端信息以某种形式记录在服务器上

Cookie会根据从服务端发送的响应报文内的一个叫做Set-Cookie的首部字段信息，通知客户端保存Cookie。当下次客户端再往该服务器发送请求时，客户端会自动在请求报文中加入Cookie值后发送出去。服务端发现客户端发送过来的Cookie后，会去检查究竟是从哪一个客户端发来的连接请求，然后对比服务器上的记录（Session），最后得到之前的状态信息。所以这里你可以知道Cookie和Session的区别，首先是存放位置，其次是安全性和持久性（存放在服务端的Session更为安全和持久）

session：工作原理   
1.创建session   
2.在创建session的同时，服务器会为该session生成唯一的session id   
3.在session被创建之后，就可以调用session相关的方法往session中添加内容   
4. 当客户端再次发送请求的时候，会将这个session id带上，服务器接受到请求后就会依据session id找到相应的session

区别

1. 存放位置不同
2. 存放方式不同 （session能保存任何内容，cookie 字符串）
3. 安全性不同，session更安全
4. 有效期不同 ，cookie比较长，session如果设置为-1关闭浏览器后就会失去
5. 对服务器压力不同

Http与Https的区别：

1. HTTP 的URL 以http:// 开头，而HTTPS 的URL 以https:// 开头

2. HTTP 是不安全的，而 HTTPS 是安全的  
3. HTTP 标准端口是80 ，而 HTTPS 的标准端口是443  
4. 在OSI 网络模型中，HTTP工作于应用层，而HTTPS 的安全传输机制工作在传输层  
5. HTTP 无法加密，而HTTPS 对传输的数据进行加密  
6. HTTP无需证书，而HTTPS 需要CA机构wosign的颁发的SSL证书  
HTTPS工作原理  
一、首先HTTP请求服务端生成证书，客户端对证书的有效期、合法性、域名是否与请求的域名一致、证书的公钥（RSA加密）等进行校验；  
二、客户端如果校验通过后，就根据证书的公钥的有效， 生成随机数，随机数使用公钥进行加密（RSA加密）；  
三、消息体产生的后，对它的摘要进行MD5（或者SHA1）算法加密，此时就得到了RSA签名；  
四、发送给服务端，此时只有服务端（RSA私钥）能解密。  
五、解密得到的随机数，再用AES加密，作为密钥（此时的密钥只有客户端和服务端知道）。

http1.0所做的优化

* 带宽：现阶段已经解决
* 延迟：1.浏览器阻塞：浏览器对同一个域名，同时只能有4个连接、
* 延迟：2.DNS查询：浏览器需要知道目标服务器的 IP才能建立连接
* 延迟：建立链接：三次握手

http1.0/http1.1的区别

1缓存处理不同。 1.0只是用一个头文件的缓存策略，1.1引入了更多   
2带宽优化及网络连接的使用（1.0存在浪费带宽，不支持断点续传功能。1.1允许只请求资源的部分）   
3Host头处理 (1.0中每台服务器绑定唯一的IP地址。1.1请求消息和响应消息都支持Host头表改进）   
4长连接（最大的区别）   
http1.0/http1.1存在的问题

1. HTTP1.X在传输数据时，每次都需要重新建立连接，无疑增加了大量的延迟时间（主要存在于1.0中，1.1可以解决）
2. HTTP1.x在传输数据时，所有传输的内容都是明文，客户端和服务端都无法验证对方的身份（可以使用https)
3. HTTP1.x在使用时，header里携带的内容过大，在一定程度上增加了传输成本
4. 虽然HTTP1.x支持了keep-alive，来弥补多次创建连接产生的延迟，但是keep-alive使用多了同样会给服务器带来大量的性能压力（比如请求一个文件）在http1.0中，当建立连接后，客户端发送一个请求，服务器端返回一个信息后就关闭连接，当浏览器下次请求的时候又要建立连接，显然这种不断建立连接的方式，会造成很多问题。

在http1.1中，引入了持续连接的概念，通过这种连接，浏览器可以建立一个连接之后，发送请求并得到返回信息，然后继续发送请求再次等到返回信息，也就是说客户端可以连续发送多个请求，而不用等待每一个响应的到来。

11、Http与Https优缺点？

* 通信使用明文不加密，内容可能被窃听，也就是被抓包分析。
* 不验证通信方身份，可能遭到伪装
* 无法验证报文完整性，可能被篡改
* HTTPS就是HTTP加上加密处理（一般是SSL安全通信线路）+认证+完整性保护

12、Http优化

* 利用负载均衡优化和加速HTTP应用
* 利用HTTP Cache来优化网站

13、Http协议有那些特征？

1、支持客户/服务器模式；2、简单快速；3、灵活；4、无连接；5、无状态。

HTTP协议：HTTP请求头，HTTP请求方式哪几种以及区别，cookie在哪里，什么特点，服务器那边对应的是什么，除了cookie还有那些常见的客户端存储。HTTP相应头，状态码

HTTPS协议：说完SSL与TLS后自己又拓展把细节介绍了下，对称密钥与非对称密钥，https优缺点，https证书cname配置。

解释了游览器同源策略，前端跨域解决方案，cors，jsonp，图像ping

断点调试怎么调试？有没有用过抓包工具？

dom实际操作：生成子元素并赋予id，修改css，删除一些节点等等类似简单问题

**http 缓存的好处** 缓存机制（强缓存 + 协商缓存）

1、减少了冗余的数据传输，减少网费  
2、减少服务器端的压力  
3、Web 缓存能够减少延迟与网络阻塞，进而减少显示某个资源所用的时间  
4、加快客户端加载网页的速度

常见 http 缓存的类型

1、私有缓存（一般为本地浏览器缓存）  
2、代理缓存

然后谈谈本地缓存

本地缓存是指浏览器请求资源时命中了浏览器本地的缓存资源，浏览器并不会发送真正的请求给服务器了。它的执行过程是

1、第一次浏览器发送请求给服务器时，此时浏览器还没有本地缓存副本，服务器返回资源给浏览器，响应码是200 OK，浏览器收到资源后，把资源和对应的响应头一起缓存下来。  
2、第二次浏览器准备发送请求给服务器时候，浏览器会先检查上一次服务端返回的响应头信息中的Cache-Control，它的值是一个相对值，单位为秒，表示资源在客户端缓存的最大有效期，过期时间为第一次请求的时间减去Cache-Control的值，过期时间跟当前的请求时间比较，如果本地缓存资源没过期，那么命中缓存，不再请求服务器。  
3、如果没有命中，浏览器就会把请求发送给服务器，进入缓存协商阶段。

HTTP强缓存和协商缓存

强缓存

强缓存是利用http的返回头中的Expires或者Cache-Control两个字段来控制的，用来表示资源的缓存时间。·

Expires

该字段是http1.0时的规范，它的值为一个绝对时间的GMT格式的时间字符串，比如Expires:Mon,18 Oct 2066 23:59:59 GMT。这个时间代表着这个资源的失效时间，在此时间之前，即命中缓存。这种方式有一个明显的缺点，由于失效时间是一个绝对时间，所以当服务器与客户端时间偏差较大时，就会导致缓存混乱。

Cache-Control

Cache-Control是http1.1时出现的header信息，主要是利用该字段的max-age值来进行判断，它是一个相对时间，例如Cache-Control:max-age=3600，代表着资源的有效期是3600秒。cache-control除了该字段外，还有下面几个比较常用的设置值：

no-cache：不使用本地缓存。需要使用缓存协商，先与服务器确认返回的响应是否被更改，如果之前的响应中存在ETag，那么请求的时候会与服务端验证，如果资源未被更改，则可以避免重新下载。•

no-store：直接禁止游览器缓存数据，每次用户请求该资源，都会向服务器发送一个请求，每次都会下载完整的资源。•

•public：可以被所有的用户缓存，包括终端用户和CDN等中间代理服务器。

•private：只能被终端用户的浏览器缓存，不允许CDN等中继缓存服务器对其缓存。

Cache-Control与Expires可以在服务端配置同时启用，同时启用的时候Cache-Control优先级高。

协商缓存

协商缓存就是由服务器来确定缓存资源是否可用，所以客户端与服务器端要通过某种标识来进行通信，从而让服务器判断请求资源是否可以缓存访问，这主要涉及到下面两组header字段，这两组搭档都是成对出现的，即第一次请求的响应头带上某个字段（Last-Modified或者Etag），则后续请求则会带上对应的请求字段（If-Modified-Since或者If-None-Match），若响应头没有Last-Modified或者Etag字段，则请求头也不会有对应的字段。

**web性能优化**

### 1.   减少HTTP请求。CSSSprites,最常用的方式。

### 2.使用内容发布网络（CDN）什么是CDN，是分布在不同位置的服务器，主要存放静态资源（图片,css, js等），以便用户更加快速的访问内容。

### 3.添加本地缓存

加入本地缓存，在第一次访问时，把静态资源缓存到浏览器，第二次访问不会再次发送请求。浏览本身的缓存机制并不是十分靠谱，个人推荐用html5的manifest,只需要一个配置文件就ok。还有一种存放到localStorage的（个人没有尝试过）。

### 4.压缩资源文件

使用精灵图的缺点就是当需要更改图标的时候将会很麻烦，需要手动一个个地去修改，特别是当页面很多的时候，所以在使用的时候也要考虑好哪些东西适合哪些不适合做精灵图，尽量在项目开始前就统一好。

### 5.将CSS样式表放在顶部，把javascript放在底部（浏览器的运行机制决定）

### 6.避免使用CSS表达式

### 7.配置ETag

Etag在请求资源是会对比服务器上次返回的etag是否一致，如果一致，则请求为304（协商缓存）。还是推荐manifest的方式缓存。

### 8.减少DNS查询

DNS查询有时间开销，通常一个浏览器查找一个给定主机名的IP地址需要20-120ms。缓存DNS：缓存DNS查询可以很好地提高网页性能，一旦缓存了DNS查询，之后对于相同主机名的请求就无需进行再次的DNS查找，至少短时间内不需要。所以在使用页面中URL、图片、js文件、css文件等时，不要使用过多不同的主机名。

### 9.使用外部javascript和CSS

### 10.精简javascript移除不必要的字符，包括所有的注释、不必要的空白字符。

### 11.避免重定向

重定向的英文是Redirect，用于将用户从一个URL重新跳转到另一个URL。  
最常见的Redirect就是301和302两种。在我们实际开发中避免重定向最简单也最容易被忽视的一个问题就是，设置URL的时候，最后的“/”，有些人有时候会忽略，其实你少了“/”，这时候的URL就被重定向了，所以在给页面链接加URL的时候切记最后的“/”不可丢。

### 12.删除重复脚本

### 13.使Ajax可缓存

针对页面中主动的Ajax请求返回的数据要缓存到本地，当然这个是针对短期内不会变化的数据。如果不确定数据变化周期的话，可以增加一个修改标识的判断，我正常处理过程中会给一些Ajax请求返回的数据增加一个MD5值的判断，每次请求会判断当前MD5是否变化，如果变化了取最新的数据，如果不变化，则不变.

### 14，图片lazyLoad

代码优化：

减少DOM操作、给图片加上宽高、使用JSON格式进行数据交换、少用全局变量、减少对象查找，如a.b.c.d这种查找方式非常耗性能，尽可能把它定义在变量里、避免404错误。

**HTML&CSS：**

对Web标准的理解、浏览器内核差异、兼容性、hack、CSS基本功：布局、盒子模型、选择器优先级、

手写ajax

var xhr; if(window.XMLHttpRequest){

xhr=new window.XMLHttpRequest

}else{

xhr=ActiveXObject("Microsoft.XMLHTTP")

}

xhr.open("get","localhost:8200/JBB/CS/getvillage",true)

xhr.send()

xhr.onreadystatechange=function(){

if(xhr.readyState==4&xhr.status==200){

document.getElementById("mydiv").innerHTML=xhr.responseText;

}

}

同源策略，限制跨域通信

**跨域的基本方式**

1.Jsonp 只能发get

发出请求 url：www.eduexam.cn/?pass=222&**callback=jsonp**

本地全局函数function jsonp() { console.log(response.name....)}

2.hash url中#号后跟的东西，变化不刷新页面

location.hash + iframe

3.postMessage H5标准 XHR level 2 的api

window.postMessage(msg,targetOrigin)

  获取postMessage传来的消息：为页面添加onmessage事件

window.addEventListener('message',function(e){})

4WebSocket

在JS创建了web socket之后，会有一个HTTP请求发送到浏览器以发起连接。取得服务器响应后，建立的连接会使用HTTP升级从HTTP协议交换为web sockt协议。

var ws =new WebSocket("wss://echo.websocket.org")

ws.onopen ws.onmessage ws.onclose

5Cors 支持跨域通信的ajax， 浏览器识别到ajax，在请求头中加allow-origin

fetch("/url/aaa",{

method:"get"

}).then().catch

安全类

csrf跨域请求伪造 依赖于登录

伪造网站b引诱点击，触发网站a的事件，凭空涨粉

防御，1服务端注册token，2refer认真，检验来源

xss跨域脚本攻击 1攻击：向页面注入脚本，js

2.防御：让脚本无法执行

区别 xss向页面里运行脚本 csrf伪造调用攻击

HTML5、CSS3、Flexbox

数组去重

function uniq2(arr){var r=[];

for(var i= 0;i<arr.length;i++){if(r.indexOf(arr[i])===-1) r.push(arr[i])};

return r;}

ES6 set

function uniq(arr){ return Array.from(new Set(arr));}

希尔排序（不会，囧），写一个堆排序，写一个快速排序。