**需要熟记的知识点**

<http://www.cnblogs.com/jikey/p/3604459.html>

|  |
| --- |
| 经典JS面试题   1. 作用域scope   (function(){var a=b=5}())   1. 创建原生native方法   **'hello'**.**repeatify**(3)  **String.prototype.repeatify=String.prototype.repeatify || function (count){  var str = '';  for (var i = 0; i < count; i++) {  str += this;  }  return str;  }**   1. 变量提升;(4)this的应用;(5)call apply;(6)闭包;(7)数据类型typeof,其中{}、[]、null都是返回object，undefined返回undefined;(8)事件循环;(9)算法   **function** *isPrime*(number) {  **if** (**typeof** number !== **'number'** || !Number.isInteger(number)) {  **return false**;  }  **if** (number < 2) {  **return false**;  }  **if** (number === 2) {  **return true**;  } **else if** (number % 2 === 0) {  **return false**;  }  **var** sqrt = **Math**.sqrt(number);  **for** (**var** i = 3; i < sqrt; i += 2) {  **if** (number % i === 0) {  **return false** }  }  **return true**; } |

|  |
| --- |
| js五种基本数据类型：Number,String,Boolean,Undefined,Null  一种复杂数据类型：Object  3大引用类型：Object Array Function  基础类型VS复杂类型：  1. 基本类型变量存的是值，复杂类型的变量存的是内存地址。  2. 基本类型在赋值的时候拷贝值，复杂类型在赋值的时候只拷贝地址，不拷贝值。  栈内存 堆内存 |

|  |
| --- |
| js继承：1）构造函数绑定----Parent.apply(this,arguments)；2）prototype模式----Child.prototype=new Parent();Child.prototype.constructor=Child；3）直接继承prototype---- Parent.prototype.type=’’;Child.prototype=Parent.prototype; Child.prototype.constructor=Child；4）利用空函数继承，var F=function(){};F.prototype=Parent.prototype;Child.prototype=new F();Child.prototype.constructor=Child;5)拷贝继承 |

|  |
| --- |
| 原型链 |

|  |
| --- |
| Math.max.apply(null,[])  call apply 可以看着是某个对象的方法，通过调用方法的形式间接调用函数,call和apply都是改变上下文中的this并立即执行这个函数，bind方法可以让对应的函数想什么时候调就什么时候调用  call apply bind  call就是挨个传值，apply传一个数组，bind也是挨个传值，但和call和apply还有多少不同，使用call和apply会直接执行这个函数，而bind并不会而是将绑定好的this重新返回一个新函数，什么时候调用由你自己决定  function fn(a,b,c,d){ 　　console.log(a,b,c,d); }  //call fn.call(null,1,2,3);  //apply fn.apply(null,[1,2,3]);  //bind var f = fn.bind(null,1,2,3); f(4);  结果如下： 1 2 3 undefined 1 2 3 undefined 1 2 3 4  arguments.callee 返回拥有该arguments的函数  e.g:  */\*\*  \* arguments.callee,方法迭代  \** ***@param*** *n  \*/* **function** *show*(n) {  **var** arr = []  **return** (**function** () {  arr.unshift(n);  n--;  **if** (n != 0) {  arguments.**callee**()  }  **return** arr  })() }  */\*\*  \* 不使用arguments.callee,给内部函数定义名称  \** ***@param*** *n  \*/* **function** *showFn*(n) {  **var** arr = [];  **return** (**function** fn() {  arr.unshift(n);  n--;  **if** (n != 0) {  *fn*();  }  **return** arr;  })() } |

|  |
| --- |
| 2-8原则，短时间内让用户看到页面，短时间内响应用户的操作  http建立连接与等待阶段花费时间 |

|  |
| --- |
| nodejs require已经加载过的模块不会重新加载  require某个js文件时，nodejs创建了一个新的实例module=new Module(),返回module.exports里面的东西  大致过程： |

|  |  |
| --- | --- |
| webpack打包原理  把所有依赖打包成一个js文件，通过代码分割成单元片段按需加载。入口entry.js  vendor则是通过提取公共模块插件来提取的代码块（webpack本身带的模块化代码部分），而manifest则是在vendor的基础上，再抽取出要经常变动的部分，比如关于异步加载js模块部分的内容  （1）webpack配置proxyTable,跨域解决方法  proxyTable的一般用法 vue-cli使用这个功能是借助http-proxy-middleware插件，一般解决跨域请求api proxyTable: { '/list': { target: 'http://api.xxxxxxxx.com', -> 目标url地址 changeOrigin: true, -> 指示是否跨域 pathRewrite: { '^/list': '/list' -> 可以使用 /list 等价于 api.xxxxxxxx.com/list } } }  webpack打包优化：   |  | | --- | | 1）区分开发环境、生成环境；2）使用代码热替换；3）编译ES6；4）使用ESlint检查js代码规范；5）压缩代码；6）代码分割、按需加载；7）使用DllPlugin和DllReferencePlugin分割代码;8)压缩图片、将图片转成base64；8）给文件添加hash缓存、自动生成页面;10）使用Fetch |   (1) **let *BundleAnalyzerPlugin*** = *require*(**'webpack-bundle-analyzer'**).**BundleAnalyzerPlugin**;  查看打包情况，多少包以及包大小；npm run dev --report  (2) |

|  |
| --- |
| array方法  添加：push（返回修改后数组的长度）、unshift（将元素添加到原数组开头，返回数组长度）、splice（始终返回一个数组，包含从原数组中删除的项，如果没有删除，返回空数组）  删除：pop（返回移除的项）、shift（删除原数组第一项，返回删除元素的值）、splice  连接：concat  截取：slice  拷贝：arr.slice(0)、arr.concat()  排序：sort  反转：reverse  转换成字符串：join,原数组不变  Array.from(new Set([1,2,4,4,6,6]))去重  splice，原数组改变，返回操作项  Array.prototype.slice.call(arguments)将arguments转换成数组    ES5新增  indexOf（从数组开头向后进行查找）、 lastIndexOf（从数组末尾向前进行查找）、 forEach（没有返回值）、map（返回新数组）、 reduce、 filter（返回满足条件的数组）、 every（判断数组中每一项是否满足条件，全部满足才返回true）、 some、 reduceRight    arr.sort(function(a,b){return a-b})//数组升序排序    冒泡排序**function** *sortArr*(arr) {  **for** (**var** i = 0; i < arr.**length**; i++) {  **for** (**var** j = 0; j < arr.**length** - 1 - i; j++) {  **if** (arr[j + 1] > arr[j]) {  **var** tmp = arr[j]  arr[j] = arr[j + 1]  arr[j + 1] = tmp  }  }  }  **return** arr } *sortArr*([6, 1, 4, 9])*//[9,6,4,1]*  插入排序  **function** *InsertionSort*(from) {  **var** arr = *deepCopy*(from)  **var** key;  **var** i;  **for** (**var** j = 1; j < arr.**length**; j++) {  key = arr[j];  i = j - 1;  **while** (i > -1 && arr[i] > key) {  arr[i + 1] = arr[i];  i = i - 1;  }  arr[i + 1] = key;  }  **return** arr }  快速排序 |

|  |
| --- |
| 闭包的定义：就是能够读取其他函数内部变量的函数（即，定义在一个函数内部的函数）  在本质上，闭包就是将函数内部和函数外部连接起来的一座桥梁  匿名函数具有全局性  var name = 'window';  var object = {  name: 'object',  getNameFunc: function() {  return function() {  return this.name;  };  }  };  alert(object.getNameFunc()()); //window  闭包的用途：1 读取函数内部的变量；2 让这些变量的值始终保持在内存中  闭包的应用场景：1 保护函数内的变量安全；2 在内存中维持一个变量 |

|  |
| --- |
| this理解：  如果一个函数中有this，但是它没有被上一级的对象所调用，那么this指向的就是window（排除严格模式）；  如果一个函数中有this，这个函数有被上一级的对象所调用，那么this指向的就是上一级的对象；  如果一个函数中有this，**这个函数中包含多个对象，尽管这个函数是被最外层的对象所调用，this指向的也只是它上一级的对象**  匿名函数,定时器中的函数,由于没有默认的宿主对象,所以默认this指向window  var o = {  a:10,  b:{  a:12,  fn:function(){  console.log(this.a); //undefined  console.log(this); //window  }  }  }  var j = o.b.fn;  j();  this永远指向的是最后调用它的对象，也就是看它执行的时候是谁调用的，例子4中虽然函数fn是被对象b所引用，但是在将fn赋值给变量j的时候并没有执行所以最终指向的是window |

|  |
| --- |
| jQuery绑定事件的方式  bind(.click .change)：把监听事件绑定到目标元素上，动态添加的元素上事件无效；  live：将监听器绑定到this.context上，并非自己（this）；  delegate： 通过selector指定触发事件的目标元素；  on（types,selector,data,fn）,前三者源码中使用on |

|  |
| --- |
| css箭头形成原理：通过截取border的部分拐角实现 |

|  |
| --- |
| VUE生命周期 |

|  |
| --- |
| css 斜切角:  1)css 渐变，渐变接受一个角度参数，色标位置是绝对长度，不受元素和背景尺寸的改变影响    2) svg+background-image  .**skew-corner** {  border: 20**px solid transparent**;  border-image: 1 **url**(**'data:image/svg+xml,\<svg xmlns="http://www.w3.org/2000/svg"\ width="3" height="3" fill="%2358a">\<polygon points="0,1 1,0 2,0 3,1 3,2 2,3 1,3 0,2"/>\</svg>'**);  background: **#58a**;  background-clip: **padding-box**; }  3）clip-path |

|  |
| --- |
| transition animation 区别：  控制粒度不同  1)transition:transition-property transition-duration transition-timing-function transition-delay  transition: 过渡属性 过渡所需要时间 过渡动画函数 过渡延迟时间；  2)animation |

|  |
| --- |
| px em rem  em相对于父元素，层级越深计算越复杂；rem相对于根元素 |

|  |
| --- |
| ajax访问原理：  创建XMLHttpRequest，xhr.setRequestHeader、onreadystatechange、open、send  xhr.state,2\*\*表示成功；3\*\*表示重定向，其中304表示未修改。  浏览器向服务器发送请求 —-> 服务器将请求的html文档发送回浏览器 —-> 浏览器下载文档，从上到下解析文档，生成DOM，会根据页面中的CSS形成CSSOM，DOM和CSSOM会生成一个渲染树 —-> 浏览器根据渲染树的内容计算出各个节点的确切大小和位置，并绘制在浏览器上 |

|  |
| --- |
| css 渐变色原理 |

|  |
| --- |
| export 和 module.exports的区别 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| DOM操作：获取、增、删、查、改等操作   |  |  | | --- | --- | | 方法 | 说明 | | getElementById（返回HTMLDivElement  ）、getElementsByTagName、getElementsByName（返回数组HTMLCollection(NodeList)） | 获取节点 | | getAttribute 、setAttribute、removerAttribute | 对节点的属性进行操作 | | createElement | 创建元素节点 | | createTextNode | 创建文本节点 | | appendChild | 向节点添加最后一个子节点 | | target.insertBefore(newChild,existingChild) | 1)target:被添加节点和现有节点的父节点;2)newChild:将要被插入的节点;3)existingChild:现有的节点，新节点将会被插入到它的前面，此参数值可以为null。 | | node.replaceChild (newnode,oldnode ) | 新节点替换旧节点 | | cloneNode | 复制节点 | | removeChild、removeChildren | 移除节点 |   节点分为元素节点、属性节点、文本节点   |  |  |  |  | | --- | --- | --- | --- | | 节点类型 | nodeName | nodeType | nodeValue | | 元素 | 元素名称 | 1 | null | | 属性 | 属性名称 | 2 | 属性值 | | 文本 | #text | 3 | 文本内容 |  |  |  | | --- | --- | | 层次节点属性 | | | 属性 | 说明 | | childNodes | 获取当前节点的所有子节点 | | firstChild | 获取当前节点的第一个子节点 | | lastChild | 获取当前节点的最后一个子节点 | | owenerDocument | 获取该节点的文档根节点，相当于document | | parentNode | 获取当前节点的父节点 | | previousSibling | 获取当前节点的前一个同级节点 | | nextSibling | 获取当前节点的后一个同级节点 | | attributes | 获取当前元素节点的所有属性节点集合 | |

|  |
| --- |
| 性能优化有哪些原则措施，减少http请求为什么能提高性能？（http请求头数量大、连接开销大-DNS解析、tcp连接等）有没具体实践过？（合并JS、CSS；使用雪碧图；）   1. 措施：    **请求数量——**合并脚本和样式表，CSS Sprites，拆分初始化负载，划分主域   **请求带宽——**开启Gzip，精简JavaScript，移除重复脚本，图像优化   **缓存利用——**使用CDN，使用外部JavaScript和CSS，添加Expires头，减少DNS查找，配置ETag，使Ajax可缓存   **页面结构——**将样式表放在顶部，将脚本放在底部，尽早刷新文档的输出    **代码校验——**避免CSS表达式，避免重定向  TTFB(Time to First Byte最初的网络请求被发起到服务器接收到第一个字节，包含了 TCP连接时间，发送HTTP请求时间和获得响应消息第一个字节的时间  优化方案：1 减少DNS查询；2 使用CDN；3 提早flush(让页面分块，逐步呈现); 4 添加周期头 |

|  |
| --- |
| ReactJs基础  生命周期：   1. Mounting: componentWillMount --> render --> componentDidMount --> 2. Updating: shouldComponentUpdate componentWillUpdate componentDidUpdate 3. Unmounting:componentWillUnmount |

|  |
| --- |
| 浅拷贝（shallow copy）和深拷贝(deep copy)  Object.assign浅拷贝  **function** *deepCopy*(obj) {  **var** str, newobj = obj.**constructor** === Array ? [] : {};  **if** (**typeof** obj !== **'object'**) {  **return**;  } **else if** (**window**.**JSON**) {  str = **JSON**.stringify(obj), *//系列化对象，会丢失function、undefined* newobj = **JSON**.parse(str); *//还原* } **else** {  **for** (**var** i **in** obj) {  newobj[i] = **typeof** obj[i] === **'object'** ?  *deepCopy*(obj[i]) : obj[i];  }  }  **return** newobj; } |

|  |
| --- |
| 数组去重、扁平化  数组去重方法：1）set去重法；2）reduce去重法；3）filter-数组下标去重法；4）遍历去重；5）排序后相邻法去重  [...**new** Set(***arr***)] ; Array.from(new Set(arr))   1. Reduce使arr去重，arr.reduce(function(a,b){   if(a.indexOf(b)===-1){  a.push(b)  }  return a  },[]) |

|  |
| --- |
| ES6中的解构赋值  定义：使用类似数组或对象字面量的语法将数组和对象的属性赋给各种变量  **let *node*** = {  **type**: **"Identifier"**,  **name**: **"foo"** };  let {type,name,value} = node(其中node中没有value属性，其值为undefined)  let [first,...tail]=[1,2,[3,4],9](first=0,tail=[2,[3,4],9]) |

|  |
| --- |
| OOP（面向对象编程） FP（函数式编程） |

|  |
| --- |
| 迭代器、生成器  \*）迭代器是一个拥有next()方法的特殊对象，每次调用next()都返回一个结果对象。  \*）生成器是一种返回迭代器的函数，通过function关键字后的星号（\*）来表示，函数中会用到新的关键字yield。  **function** \**createIterator*(items) {  **for** (**let** i = 0; i < items.**length**; i++) {  **yield** items[i];  } } **let *iterator*** = *createIterator*([1, 2, 3]);  console.log(***iterator***.**next**()); *// "{ value: 1, done: false }"* console.log(***iterator***.**next**()); *// "{ value: 2, done: false }"* console.log(***iterator***.**next**()); *// "{ value: 3, done: false }"* console.log(***iterator***.**next**()); *// "{ value: undefined, done: true }"* |

|  |
| --- |
| 装饰器  **定义**：修饰器是一个对类进行处理的函数，用来修改类的行为  <注>：装饰器只能用来修改类及类的方法, **编译时执行的函数**  修饰器第一个参数是类的原型对象，第二个参数是所要修饰的属性名，第三个参数是该属性的描述对象  **function** *testable*(target) {  target.**prototype**.**isTestable** = **true**; }  @*testable* **class** Person {  @**readonly** name() {  **return `**${**this**.**first**} ${**this**.**last**}**`** } }  **function** *readonly*(target, name, descriptor) {  *// descriptor对象原来的值如下  // {  // value: specifiedFunction,  // enumerable: false,  // configurable: true,  // writable: true  // };* descriptor.**writable** = **false**;  **return** descriptor; } **let *obj*** = **new** Person(); ***obj***.**isTestable** *//true* |

|  |
| --- |
| 柯里化(涉及到了闭包,function.toString，返回函数的源码字符串)  作用：1 **参数复用**；2 **提前返回**；3 **延迟计算/运行**  **function** *addOther*() {  **var** args = [].slice.call(arguments);   **var** *fn* = **function** () {  **var** newArgs = args.concat([].slice.call(arguments));  **return** *addOther*.apply(**null**, newArgs);  }  *fn*.toString = **function** () {  **return** args.reduce(**function** (a, b) {  **return** a + b;  })  }  **return** *fn*; }  **var *t*** = *addOther*(1)(2)(3)(5) **console**.log(**typeof *t***)*//function* |

|  |
| --- |
| Css预处理器 scss less stylus  Sass  变量；计算功能；嵌套；继承；mixin;颜色函数；插入文件；自定义函数等  <**style** lang=**"scss"** scoped>  **@import "../../assets/css/base.css"**;  $**wholeColor**:**#fff**;  $**pColor**:**#1875e7**;  $**mainColor**:**#bffffb**;  $**side**:**left**;  $**varLeft**:3;  .**test-main**{  color:$**wholeColor**;  }  **@mixin left**($**value**,$**color**:**lighten**(**#cc3**, 10%)){  float:**left**;  width:30%;  padding-left:**#{$value}px**;  height:30**px**;  background-color:$**color**;  margin-left:30**px**;  }  .**content**{  height: **calc**(100%);  padding-top:1.2**rem**;  .**test-p**{  color:$**pColor**;  padding-**#{$side}**:**#{$varLeft\*5}px** ;  border:{*/\*border后面必须加上冒号,标准注释,会保留到编译后的文件\*/* color:**red**;  width:2**px**;*/\*no\*/* style:**solid**;  }  &:**hover**,&:**focus**{//伪类,**只保留在SASS源文件中，编译后被省略  background-color**: $**mainColor**;  }  }  .**other-p**{  **@extend** .**test-p**;  margin-top: 40**px**;  }  .**border-left**{  border-**#{$side}**:**solid** 1**px** $**pColor**;*/\*no\*/* }  .**sub-item**{  position:**relative**;  .**left**{  **@include left**(10);  }  **@function getHeight**($**h**){  **@return** $h\*2;  }  **#subItem1**{  height: **getHeight**(25**px**);  }  }  } </**style**>  <style lang=’less’>  **@color**: **#4D926F**; **@the-border**: 1**px**; **@base-color**: **#4D9**; **@red**: **#842210**; .**test-less-main**{ .**rounder-radius**(**@radius**:5**px**){  -webkit-border-radius: **@radius**;  -moz-border-radius:**@radius** ;  border-radius:**@radius** ; } .**mixin** (**dark**, **@color**) {  color: **darken**(**@color**, 10%); } .**mixin** (**light**, **@color**) {  color: **lighten**(**@color**, 10%); } .**mixin** (**@\_**, **@color**) {  display: **block**; }  .**variable-express**{  color:**@color**;  }  .**opt-main**{  position: **relative**;  }  .**radius-div-1**,.**radius-div-2**,.**radius-div-3**{  width: 400**px**;  min-height: 40**px**;  background-color: **@color**;  color:**#fff**;  float: **left**;  } .**radius-div-1**{ .**rounder-radius**(10**px**); } .**radius-div-2**{ .**rounder-radius**(40**px**);  border-style: **solid**;  border-color: **@base-color**\*3;  border-width: **@the-border**\*3; } .**box-shadow** (**@x**: 0, **@y**: 0, **@blur**: 1**px**, **@color**: **red**) {  box-shadow: **@arguments**;  -moz-box-shadow: **@arguments**;  -webkit-box-shadow: **@arguments**; } .**radius-div-3**{ .**box-shadow**(2**px**, 5**px**); .**mixin**(**light**, **#888**); } }  </style>  sass 、less区别：  1）less基于JavaScript，在客户端处理，JavaScript引擎需要额外时间处理代码然后输出CSS到浏览器；sass基于ruby在服务端处理。  清除浮动  .**clearfix**:**after** {  content: **'.'**;  height: 0;  overflow: **hidden**;  clear: **both**;  display: **block**;  visibility: **hidden**; } .**clearfix** {  zoom: 1;*/\*hack\*/* }  淘宝的双翼布局（左右两边固定宽度，中间主内容100%，并且先渲染）:  <**div** class=**"clearfix"**>  <**div** class=**"main"**></**div**>  <**div** class=**"left"**></**div**>  <**div** class=**"right"**></**div**> </**div**>  <**style**>  .**main**, .**left**, .**right** {  float: **left**;  height: 100**px**;  }  .**main** {  width: 100%;  }  .**left** {  width: 20%;  margin-left: -100%;  background-color: **#c1c3ce**;  }  .**right** {  width: 30%;  margin-left: -30%;  background-color: **#c1c3ce**;  } </**style**> |

|  |
| --- |
| Amd cmd |

|  |
| --- |
| Height:100%起作用的方法：   1. 从根的父容器到子容器设置好高度的百分比   2) js获取视窗的宽高，来设置body的宽高，从而是div的height:100%生效  3）给父容器设置位置信息 |

|  |
| --- |
| Vuex-状态管理模式|框架  store/index.js  **import** Vue **from 'vue' import** Vuex **from 'vuex' import** cooperDetail **from './modules/cooperDetail' import** user **from './modules/user' import** state **from './state' import** actions **from './actions' import** getters **from './getters'/\*export default{}\*/ import** mutations **from './mutations'** Vue.use(Vuex) **export default new** Vuex.**Store**({  actions,  getters,  mutations,  state,  **modules**: {  user,  cooperDetail  },  **strict**: **true** })  其中cooperDetail、user中返回export default{state,getters,actions,mutations}  mutatians格式：  **const *mutations*** = {  [types.***PS\_QUERY\_PUBLIC***](state, {data}) {  state.**\_queryPublic** = data }} |
|  |

|  |
| --- |
| Vue-router (active-class)  <**router-link** active-class=**"active"** :to=**"menu.url"** tag=**"li"** v-for=**"menu,index in menus1"** :key=**"menu.name"** v-auth=**"menu.auth"**></**router-link**>  <**router-view**></**router-view**>  动态传入router参数/hello/:id,可通过props解耦(pass *route.params* to props)，设置方式如下几种：  1）{**path**: **'hello/:name'**, **component**: TestLess, **props**: **true**}  2）{**path**: **'static'**, **component**: TestLess, **props**: {**name**: **'world'**}},  3）{**path**: **'dynamic/:years'**, **component**: TestLess, props: *dynamicPropsFn*}  在组件中可通过（this.$attrs获取返回的对象） |

|  |
| --- |
| Vue-directives  **function** *authValidation*(el, binding) {  **if** (binding.value && !*hasAuth*(binding.value)) {  **if** (el.remove) {  el.remove()  } **else** {  el.removeNode(**true**)  }  } } Vue.directive(**'auth'**, {  bind: *authValidation*,  *// inserted: authValidation,* update: *authValidation* })  bind inserted update componentUpdate unbind |

|  |
| --- |
| Vue实现双向绑定原理  1）Object.defineProperty():给一个对象添加新属性或者修改原有属性  语法: Object.defineProperty(obj, prop, descriptor) |

|  |
| --- |
| Vue template编译原理   * compile 函数主要是将 template 转换为 AST，优化 AST，再将 AST 转换为 render函数; * render函数 与数据通过 Watcher 产生关联; * 在数据发生变化时调用 patch 函数，执行此 render 函数，生成新 VNode，与旧 VNode 进行 diff，最终更新 DOM 树。 |

|  |
| --- |
| Vue的响应式原理  将普通的js对象传给Vue的data，data中属性是可观察的。通过Object.defineProperty为每个属性设定get、set |

|  |
| --- |
| 换肤 |

|  |
| --- |
| Css 生成倒影:1) -webkit-box-reflect: **below;**2) |

|  |
| --- |
| JsonP:  利用script标签可以跨域，让服务器端返回可执行的js函数，参数为要回发的函数。  H5特性postMessagge   1. domain\_a.html   <**iframe** id=**"iframe"** src=**"http://localhost:63343/untitled/demo\_cross\_domain\_b.html"** style=**"**display:**none**;**"**></**iframe**> <**script**>  **var *iframe*** = **document**.getElementById(**'iframe'**);  ***iframe***.onload = **function** () {  **var** data = {  **name**: **'aym'** };  ***iframe***.**contentWindow**.postMessage(**JSON**.stringify(data), **'http://localhost:63343'**); *// 向domain\_b传送跨域数据*  };**window**.addEventListener(**'message'**, **function** (e) {*// 接受domain\_b返回数据*  **console**.info(**'data from domain\_b ---> '** + e.**data**)  }, **false**); </**script**>  (2)domain\_b.html  <**script**>**window**.addEventListener(**'message'**, **function** (e) {*// 接收domain\_a的数据*  **console**.info(**'data from domain\_a ---> '** + e.**data**)  **var** data = **JSON**.parse(e.**data**);  **if** (data) {  data.**number** = 16;  *// 处理后再发回domain\_a* **window**.**parent**.postMessage(**JSON**.stringify(data), **'http://localhost:63342'**);  }  }, **false**); </**script**> |

|  |  |  |  |
| --- | --- | --- | --- |
| ES6新特性：  1)默认参数function(w=300){};2)模板语法：`your name is ${first} ${last}`;3)  （1）箭头函数，注意：1）this指向定义时所在的对象；2）不能当做 构造函数；3）不能使用arguments；4）不能使用yield的命令  （2）rest，获取函数多余的参数   |  | | --- | | var let 区别：  1、var会造成变量提升现象，let 作用域是块级；  2、let不可以重复变量  let a=[1,3]  let b=[...a]//[1,3] |  |  | | --- | | 计算属性命名  **let *arr*** = [1, 2, 3] **let *outArr*** = ***arr***.map(***n*** => {  **return** {  [ ***n*** ]: ***n***,  [**`**${***n***}**^2`**]: **Math**.pow(***n***, 2)  }  } )*//[{"1":1,"1^2":1},{"2":2,"2^2":4},{"3":3,"3^2":9}]* |  |  | | --- | | Async await  async用于定义一个异步函数，该函数返回一个Promise。 | |

|  |
| --- |
| CSS样式优先级  内联样式表(InLine ***style***)>内部样式表(Internal ***style*** sheet)>外部样式表(External ***style*** sheet) 例外：但如果外部样式表放在内部样式表下边引用,则外部样式表>内部样式表; 1）内联样式表权值为1000; 2）ID选择器的权值为100; 3）Class类选择器的权值为10; 4）HTML标签选择器的权值为1;  **#redP p** {*/\*权值100+1=101\*/*} **#redP p**.**red em** {*/\*权值100+1+10+1=112\*/*} **#redP** .**red em** {*/\*权值100+10+1=111\*/*} **#redP p span em** {*/\*权值100+1+1+1=103\*/*} |

|  |
| --- |
| 函数式编程**var** *Container* = **function** (x) {  **this**.**\_\_value** = x; } *Container*.**of** = **x** =>**new** *Container*(**x**); *//试试看 Container.of(1); //=> Container(1) Container.of('abcd'); //=> Container('abcd') Container*.**prototype**.map = **function**(f){  **return** *Container*.**of**(f(**this**.**\_\_value**)) } **var *obj***=*Container*.**of**(3)  .map(**x** => **x** + 1)  .***map***(**x** => 1 + **x**)  .***map***(**x** => **'Result is '** + **x**) **console**.info(***obj***.**\_\_value**)*//5* |

|  |
| --- |
| 浏览器常用存储技术：  cookie，会随着HTTP请求头信息一起发送，能存储的数据容量有限，不同浏览器cookie大小不一样，一般为 4kb，而IE6大约只有2K  localStorage sessionStorage IndexedDB  **UserData(IE专属)、GlobalStorage(火狐)、Google Gear(浏览器中若没有flash插件，需要在网页中加flash)** |

|  |
| --- |
| 函数去抖debounce  **function** *debounce*(func, wait, immediate) {  **let** timeout;  **return function** () {  **let** context = **this**, args = arguments;  **let** *later* = **function** () {  timeout = **null**;  **if** (!immediate) func.apply(context, args);  };  **let** callNow = immediate && !timeout;  clearTimeout(timeout);  timeout = setTimeout(*later*, wait);  **if** (callNow) func.apply(context, args);  }; }  函数节流  **function** *throttle*(method, delay, time) {  **var** timeout, startTime = **new** Date();  **return function** () {  **var** context = **this**,  args = arguments,  curTime = +**new** Date();  clearTimeout(timeout);  *// 如果达到了规定的触发时间间隔，触发 handler* **if** (curTime - startTime >= time) {  method.apply(context, args);  startTime = curTime;  } **else** {  *// 没达到触发间隔，重新设定定时器* timeout = setTimeout(method, delay);  }  } } |

|  |
| --- |
| 懒加载 预加载 |

|  |
| --- |
| 模板字符串`` ${}  **const *name*** = **'lux' console**.log(**`hello** ${***name***}**`**) *//hello lux*  const obj = Object.assign({}, objA, objB)  1.当用export default people导出时，就用 import people 导入（不带大括号）  2.一个文件里，有且只能有一个export default。但可以有多个export。  3.当用export name 时，就用import { name }导入（记得带上大括号）  4.当一个文件里，既有一个export default people, 又有多个export name 或者 export age时，导入就用 import people, { name, age }  5.当一个文件里出现n多个 export 导出很多模块，导入时除了一个一个导入，也可以用import \* as example  *// 模拟其他语言中的 sleep，实际上可以是任何异步操作* **const *sleep*** = (timeountMS) => **new** Promise((**resolve**) => {  setTimeout(resolve, timeountMS); }); (**async** () => { *// 声明即执行的 async 函数表达式* **for** (**var *i*** = 0; ***i*** < 5; ***i***++) {  await ***sleep***(1000);  **console**.log(**new** Date, ***i***); } await ***sleep***(1000); **console**.log(**new** Date, ***i***); })(); |

|  |
| --- |
| css编写规范  1.位置属性(position, top, right, z-index, display, float等)  2.大小(width, height, padding, margin)  3.文字系列(font, line-height, letter-spacing, color- text-align等)  4.背景(background, border等)  5.其他(animation, transition等) |

<https://www.jianshu.com/p/609489e8c929>

<https://zhuanlan.zhihu.com/p/24833275>

1 团队工作流程

2 团队人数，人员状况，负责什么样的产品，岗位要求，在公司内部的发展如何？

3 领导风格

4 五险一金