流利说 （挂）8月17

1. Flex布局，flex属性，原理

6个容器属性：

Flex-flow = flex-direction + flex-wrap; 例如：flex: row nowrap;

Flex-direction: row | column | row-reverse | column-reverse

Flex-wrap: nowrap | wap | wrap-reverse

Just-content: flex-start | flex-end | center | space-around | space-between

Align-items: flex-start | flex-end | center | baseline | stretch(默认)

Align-content: flex-start | flex-end | center | space-around | space-between | stretch(默)

6个元素属性：

Order

Flex = flex-grow + flex-shrink + flex-basis

Flex-grow:放大比例，无剩余空间，无效

Flex-shrink:缩放比例，有剩余空间，无效（会考虑元素本身的大小防止缩小后宽度为0）

Flex-basis:初始化尺寸，即前两者生效前的尺寸（不考虑元素大小）

Align-self:属性值和align-items相同，当设置了align-self的值，父元素上的align-items失效

原理：弹性布局，具有伸缩性，元素大小高度依赖父级元素，目的是为了撑满父容器，和position，float根据盒模型可以计算出元素宽高（非百分比），除非溢出，才会依赖父级元素。

Flex默认值：flex: 0 1 auto

几个简写：

Flex:1 => flex: 1 1 0%

Flex:2 =>flex:2 1 0%

Flex:auto => flex:1 1 auto

Flex:none =>flex:0 0 auto(常用于固定尺寸不伸缩)

1. Float

CSS的float（浮动），会使元素向左或向右移动，其周围的元素也会重新排列。**元素的水平方向浮动，意味着元素只能左右移动而不能上下移动。**一个浮动元素会尽量向左或向右移动，直到它的外边缘碰到包含框或另一个浮动框的边框为止。浮动元素之后的元素将围绕它，浮动元素之前的元素将不会受到影响。

**浮动特征：**

1.按照一个指定的方向移动，遇到父级的边界或者其他的元素停下来

2.块元素可以在一行显示，宽由内容撑起来

3.行内元素支持设置宽高

4.脱离文档流 float不是完全脱离文档流的

块元素 ：对于块元素，后面未浮动的元素所占的位置是浮动元素的位置

**注意：**内容会把浮动元素的位置保留，

行内元素：行内元素会接着浮动元素显示，内容会形成环绕浮动元素的样子

**清除浮动方式（四种）：**

1. 给父级元素添加高height
2. 给父级元素增加overflow:hidden/auto;zoom:1
3. 设置父元素伪类

.parent:after{

clear:both;

display:block; //after伪元素添加的内容默认为行内元素,需要设置display:block

content:””;

//visibility:hidden;

//height:0;

//zoom:1;//IE6,IE7兼容性问题，仅IE支持，重新渲染dom

}

1. 跨域，广义定义，实现跨域方式

跨域广义定义：一个域下的资源试图访问另一个域下的资源。

受同源策略限制：所谓同源，即协议、域名、端口号相同，三者有一个不同则视为跨域

**解决跨域方式：**

1. Jsonp
2. 跨站请求伪造
3. 请求数据 ajax fetch 跨域
4. 浏览器缓存 强缓存 、协商缓存 具体哪些数据
5. 第一次请求资源下载后，点击刷新，会在请求一次吗？
6. Git SVN
7. SourceTree是什么？底层原理是什么？存在bug本地
8. 代码提交后push到服务器，存在bug该怎么修改？
9. 怎样部署项目？
10. 如何打包一个项目？命令是什么？
11. 打包工具webpack原理

多益网络（等待结果）8月23

1. 单例模式 例子
2. http协议get/post区别
3. http，https CA认证
4. 编写网页页面时候的安全防范
5. XSS请求伪造 防范
6. 不需要第三变量，交换a,b的值
7. 多人协助项目，css命名不冲突方案？
8. 后端语言，技术热点、技术论坛

360 （挂）8月28

1. 自适应怎么做的？<https://www.php.cn/div-tutorial-410913.html>

**静态布局**：给页面元素设置固定的宽度和高度，单位用px，当窗口缩小，会出现滚动条，拉动滚动条显示被遮挡内容。针对不同分辨率的手机端，分别写不同的样式文件。

**自适应布局**：创建多个静态布局，每个静态布局对应一个屏幕分辨率范围，使用@media媒体查询技术。

**流式布局**：元素的宽高用百分比做单位，元素宽高按屏幕分辨率调整，布局不发生变化。屏幕尺度跨度过大的情况下，页面不能正常显示。

**响应式布局**：采用自适应布局和流式布局的综合方式，为不同屏幕分辨率范围创建流式布局。

**自适应布局：**要点在于使用em和rem单位来定义元素宽度，与流式布局有极大的相似性，但也有不同之处，主要区别在于弹性布局的尺寸主要根据字体大小而变化

1. Flex布局（弹性布局---自适应）
2. Rem px 区别
3. 什么是闭包？作用是什么？
4. 继承哪些方式，优缺点？改进方式？
5. 深拷贝方式 （JSON缺点）
6. 开关定时器方式
7. ==和===区别

* 对于引用类型的变量，==（等同）和===（恒等）只会判断**引用的地址是否相同**，而不会判断对象具体里属性以及值是否相同。因此，如果两个变量指向相同的对象，则返回true。

var arrRef = ['Hi!'];

var arrRef2 = arrRef;

console.log(arrRef === arrRef2); // true

* 如果是不同的对象，即使包含相同的属性和值，也会返回false。

var arr1 = ["Hi!"];

var arr2 = ["Hi!"];

console.log(arr1 === arr2); // false

* 如果想判断两个不同的对象是否真的相同，简单的方法即将它们转换为字符串然后判断。

var arr1str = JSON.stringify(arr1);

var arr2str = JSON.stringify(arr2);

console.log(arr1str === arr2str); // true

另一个方法就是递归地判断每一个属性的值，直到基本类型位置，然后判断是否相同。

1. Vue生命周期
2. 父子组件之间通信方式，非父子组件之间的通信方式
3. 状态管理Vuex简要说明
4. 路由
5. Vue的Render了解?

<https://blog.csdn.net/sansan_7957/article/details/83014838>

1. Css3写过哪些
2. Less，scss基于less
3. 报表图表有无自己写过？无，哈哈~~~菜鸟的蓝瘦
4. 3d，echart 大屏系统显示效果
5. Tree.js sv.js
6. 炫酷效果
7. Webpack原理，打包过程
8. 有没有自己封装过树（没有）

**9月3号 bigo**

1.v-if 和 v-show 区别？v-show怎么控制样式？v-for和v-if的优先级？

v-if和v-show都是动态显示DOM元素。

**区别**：

1. 编译过程：v-if 是真正的条件渲染，因为它会确保在切换过程中条件块内的事件监听器和子组件适当地被销毁和重建。v-show的元素始终会被渲染并保留在 DOM 中。v-show只是简单地切换元素的 CSS 属性display。
2. 编译条件：v-if 是惰性的：如果在初始渲染时条件为假，则什么也不做。直到条件第一次变为真时，才会开始渲染条件块。v-show不管初始条件是什么，元素总是会被渲染，并且只是简单地基于 CSS 进行切换。
3. 性能消耗：v-if有更高的切换消耗。v-show有更高的初始渲染消耗。
4. 应用场景：v-if适合运行时条件很少改变时使用。v-show适合频繁切换。

**v-for和v-if的优先级**

1. v-for的优先级更高，会先进行循环，再进行 if 判断；永远不建议同时使用 v-for 和 v-if，因为会导致性能低下，增加渲染成本；
2. 当真的需要进行判断是否渲染列表的时候，在列表循环的外部添加 v-if；

<template>

<div>

<ul v-if="list.length>5">

<li v-for="(item,index) in list" :key=index>{{item}}</li>

</ul>

</div>

</template>

当需要对循环的每一项进行判断是否渲染的时候，在computed计算属性中先行计算，并直接渲染对应的计算属性即可。

2.v-model实现？

语法糖，在使用v-model进行数据双向绑定时：<input v-model = “something”>

仅仅是一个语法糖：<input v-bind:value = “something” v-on:input = “something = $event.target.value”> 所以在组件中使用，相当于下面简写：<custom v-bind:value = “something” v-on:input = “something = $event.target.value”>

所以要组件的v-model生效，它必须：

* 接受一个value属性
* 在有新的value时触发input事件

在自定义组件中：要在字符串模板中使用上述的第二种方式来实现v-model。

<div id="demo">

<test v-model="name"></test>

<input type="text" v-model="name" >

<p>{{name}}</p>

<button @click="ccc">按钮</button>

</div>

Vue.component("test",{

//props 当一个值传递给一个 prop 特性的时候，它就变成了那个组件实例的一个属性

//v-model 绑定的就是 input等表单元素的value、

//因此此组件中的prop必为value

props:['value'],

template:`

<input

v-on:input="$emit('input', $event.target.value)"

v-bind:value="value"

>`

})

new Vue({

el:"#demo",

data:{

name:"你好",

},

methods:{

ccc:function(){

this.name = "世界";

}

}

})

3.双向数据绑定的原理

Object.defineProperty()的作用就是直接在一个对象上定义一个新属性，或者修改一个已经存在的属性。

Object.defineProperty(obj, prop, desc)

* obj 需要定义属性的当前对象
* prop 当前需要定义的属性名
* desc 属性描述符 可以控制一个对象属性的一些特有操作，比如读写权，是否可枚举.
* configrable 描述属性是否配置，以及可否删除
* enumerable 描述属性是否会出现在for in 或者 Object.keys()的遍历中

configurable: false 时，不能删除当前属性，且不能重新配置当前属性的描述符(有一个小小的意外：可以把writable的状态由true改为false,但是无法由false改为true),但是在writable: true的情况下，可以改变value的值；configurable: true时，可以删除当前属性，可以配置当前属性所有描述符。

**原理：**

vue的双向绑定是由数据劫持结合发布者－订阅者模式（backbone）实现的，通过Object.defineProperty()来劫持对象属性的setter和getter操作。通过MVVM实现数据双向绑定：1.通过观察者来监听数据的setter、getter值，发布给订阅者；2.通过指令解析器compile对每个节点进行扫描和解析，根据指令模板替换数据，以及绑定相应的更新函数；3.watcher是observer和compile的桥梁，能够订阅并收到每个属性变动的通知，执行指令绑定的相应回调函数，从而更新视图。

4.有使用过nextTick？它是做什么的？实现？

用于延迟一段代码，接收两个参数（回调函数、执行回调函数函数的上下文环境），若无回调函数将返回promise对象。

是更新dom的vue方法

一般在生命周期created（）钩子函数进行DOM操作一定会放在vue.nextTick（）的回调函数。原因：再次钩子函数执行时，DOM其实还没有进行任何渲染工作（mounted），进行dom操作是无济于事的，所以要将DOM操作的js代码放在vue.nextTick（）的回调函数中。

总结一句话就是：vue的一步操作，vue.nextTick（）用来指导DOM什么时候更新。

**异步更新队列**：vue在观察导数据变化的时，不是立即更新DOM，而是开启一个队列，并缓冲队列的所有数据改变，并在缓冲室去除重复的数据。在下一个事件循环tick中，vue刷新队列并执行实际操作（去重后）。所以如果用一个for循环来动态改变数据100次，其实它只能应用到最后一次改变，若果没有这种机制，DOM就会重新重排（reflow）100次，加大了开销。

5.有了解过keepalive？

Keep-alive主要用于保留数组状态或重新渲染。

例如：有一个列表页面和一个详情页面，用户就会经常执行打开详情=》返回列表=》打来详情，这样的话 列表 和 详情都是一个频率很高的页面，就可以对列表使用<keep-alive></keep-alive>进行缓存，这样用户每次返回列表的时候，都能从缓存中快速渲染，而不是重新渲染。

用法：包裹动态组件是，会缓存不活动的组件实例，而不是销毁他们。和<transition>相似，<keep-alive>是抽象的组件，他自身并不渲染DOM，也不会出现在父组件链中。

6.keepalive它主要缓存的是什么？

keep-alive来缓存vue的组件信息，使其不再重新加载。如果缓存的组件想要清空数据或者执行初始化方法，在加载组件的时候调用activated钩子函数。activated: function () {

this.data = ‘'

}

7.vue的虚拟dom树，diff算法？怎样提高dom更新效率？绑定key需要注意些什么？

虚拟DOM其实就是普通的js对象，它用来描述一段HTML片段的。

* 当页面渲染的时候会创建虚拟DOM树；
* 当页面发生改变会再创建一颗虚拟的DOM树
* 前后新旧两个DOM数作对比，vue通过diff算法，去记录存在差异的大方；
* 将差异的地方更新到真实的DOM树

**虚拟dom树的作用：**

**Vue中的虚拟dom树只会重新渲染页面修改的地方，大大减少了dom树的操作。虚拟dom树占据内容，但可以提高dom性能。**虚拟DOM树是用空间(虚拟DOM树占空间)换时间(虚拟DOM树可以提高DOM效率)。

**当数据发生变化时，vue是怎么更新节点的？**

先根据真实DOM生成一颗 virtual DOM，当 virtual DOM 某个节点的数据改变后会生成一个新的 Vnode，然后 Vnode 和 oldVnode 作对比，发现有不一样的地方就直接修改在真实的DOM上，然后使 oldVnode 的值为 Vnode 。

**virtual DOM和真实DOM的区别？**

virtual DOM是将真实的DOM的数据抽取出来，以对象的形式模拟树形结构。比如dom是这样的：

<div>

 <p>123</p>

</div>

对应的virtual DOM（伪代码）

var Vnode = {

 tag: 'div',

 children: [

  { tag: 'p', text: '123' }

 ]

};

**diff的比较方式？**

在采取diff算法比较新旧节点的时候，比较只会在**同层级**进行, 不会跨层级比较。

比较后出现四种情况：

1、此节点是否被移除 -> 添加新的节点

2、属性是否被改变 -> 旧属性改为新属性

3、文本内容被改变-> 旧内容改为新内容

4、节点要被整个替换 -> 结构完全不相同 移除整个替换

**传统Diff：计算一棵树形结构转换为另一棵树形结构需要最少步骤，如果使用传统的diff算法通过循环递归遍历节点进行对比，其复杂度要达到O(n^3)，其中n是节点总数，效率十分低下，假设我们要展示1000个节点，那么我们就要依次执行上十亿次的比较。**

**Diff算法：**

diff的过程就是调用名为 patch 的函数，比较新旧节点，一边比较一边给 真实的DOM 打补丁。

一、找到相同的前置元素、后置元素；

1、旧数组为空，将新数组的剩余元素插入；

2、新数组为空，将旧数组的剩余元素删除；

3、新、旧数组都不为空，执行第二步。

二、找到需要被删除、插入、移动的元素

数组p：与新数组的长度相同，与新数组是相互映射关系，

元素在旧数组中的索引 存储在 元素在新数组中的位置

三、找到最少的移动次数

1、找到 P 数组的最长递增子序列来做动态规划，新集合中不属于这个序列的将会被移动。

2、同时尾部遍历新数组和 LIS 序列，查看元素的位置是否能与 LIS 序列的任何一个值匹配：

a：可以匹配，保留位置；

b：不能匹配，移动到到前面；

c：找不到，插入元素；

**React中Diff算法的实现：**

React用 三大策略 将O(n^3)复杂度 转化为 **O(n)复杂度**

**策略一（tree diff）：**

Web UI中DOM节点跨层级的移动操作特别少，可以忽略不计。

**策略二（component diff）：**

拥有相同类的两个组件 生成相似的树形结构，

拥有不同类的两个组件 生成不同的树形结构。

**策略三（element diff）：**

对于同一层级的一组子节点，通过唯一id区分。

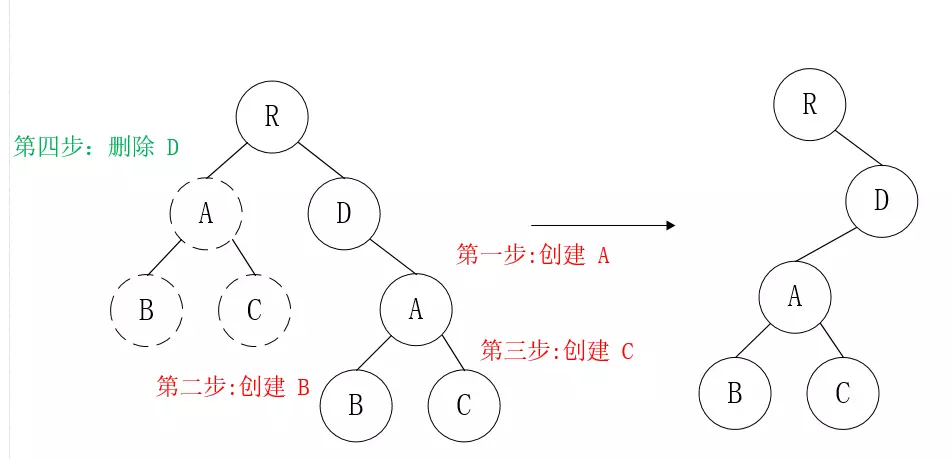
**tree diff**

（1）React通过updateDepth对Virtual DOM树进行层级控制。

（2）对树分层比较，两棵树 只对同一层次节点 进行比较。如果该节点不存在时，则该节点及其子节点会被完全删除，不会再进一步比较。

（3）只需遍历一次，就能完成整棵DOM树的比较。

**那么问题来了，如果DOM节点出现了跨层级操作,diff会咋办呢？**  
答：diff只简单考虑同层级的节点位置变换，如果是跨层级的话，**只有创建节点和删除节点的操作。**



如上图所示，以A为根节点的整棵树会被重新创建，而不是移动，因此 官方建议不要进行DOM节点跨层级操作，可以通过CSS隐藏、显示节点，而不是真正地移除、添加DOM节点。

component diff

React对不同的组件间的比较，有三种策略

（1）同一类型的两个组件，按原策略（层级比较）继续比较Virtual DOM树即可。

（2）同一类型的两个组件，组件A变化为组件B时，可能Virtual DOM没有任何变化，如果知道这点（变换的过程中，Virtual DOM没有改变），可节省大量计算时间，所以 用户 可以通过 shouldComponentUpdate() 来判断是否需要 判断计算。

（3）不同类型的组件，将一个（将被改变的）组件判断为dirty component（脏组件），从而替换 整个组件的所有节点。

**注意：**如果组件D和组件G的结构相似，但是 React判断是 不同类型的组件，则不会比较其结构，而是删除 组件D及其子节点，创建组件G及其子节点。

**element diff**  
当节点处于同一层级时，diff提供三种节点操作：**删除、插入、移动**。

插入：组件 C 不在集合（A,B）中，需要插入

删除：（1）组件 D 在集合（A,B,D）中，但 D的节点已经更改，不能复用和更新，所以需要删除 旧的 D ，再创建新的。

2）组件 D 之前在 集合（A,B,D）中，但集合变成新的集合（A,B）了，D 就需要被删除。

**移动**：组件D已经在集合（A,B,C,D）里了，且集合更新时，D没有发生更新，只是位置改变，如新集合（A,D,B,C），D在第二个，无须像传统diff，让旧集合的第二个B和新集合的第二个D 比较，并且删除第二个位置的B，再在第二个位置插入D，而是 **（对同一层级的同组子节点） 添加唯一key进行区分，移动即可。**

**vue与react的区别：**

**vue：渲染过程中是跟踪每一个组件的依赖，即更改了哪个组件渲染哪个。双向**

**react：会重新渲染全部子组件。单向**

8.有了解vue3走向？vue3双向绑定哪些部分改动？

Vue.js 3.0的一些新特性，其中一个很重要的改变就是Vue3 将使用 **ES6的Proxy** 作为其观察者机制，取代之前使用的Object.defineProperty。

**为什么要取代Object.defineProperty？**

**在Vue中，Object.defineProperty无法监控到数组下标的变化，导致直接通过数组的下标给数组设置值，不能实时响应。 为了解决这个问题，经过vue内部处理后可以使用以下几种方法来监听数组（ Vue为什么不能检测数组变动）**

**push()，pop()，shift()，unshift()，splice()，sort()，reverse()**

**Object.defineProperty只能劫持对象的属性，因此我们需要对每个对象的每个属性进行遍历。Vue中是通过递归以及遍历data对象来实现数据的监控的，**如果属性值也是对象那样可以深度遍历，显然可以劫持一个完整的对象，不管是对操作性还是性能都会有一个很大的提升**。**

**什么是Proxy？**

Proxy 是ES6中新增的一个特性，可以理解成，在目标对象之前架设一层“拦截”，外界对该对象的访问，都必须先通过这层拦截，因此提供了一种机制，可以对**外界的访问**进行**过滤和改写**。Proxy 这个词的原意是代理，用在这里表示由它来“代理”某些操作，可以译为“代理器”。

**基本用法：**

//ES6 原生提供 Proxy 构造函数，用来生成 Proxy 实例。

var proxy = new Proxy(target, handler);

**参数解释**

1) target: 是用Proxy包装的被代理对象（可以是任何类型的对象，包括原生数组，函数，甚至另一个代理）。

2)handler: 是一个对象，其声明了代理target 的一些操作，其属性是当执行一个操作时定义代理的行为的函数。Proxy 对象的所有用法，都是上面这种形式，不同的只是handler参数的写法

1. 移动端适配有了解过吗？
2. ES6有用过什么？set,map有用过吗？

**Set：**

Set:返回值是对象，可用Array.from(new Set([1,2])),或者用[...new Set(1,2)]转化成数组；

set--结构**类似于数组**，但是成员唯一，不会有重复的元素。

创建：let set = new Set([1,2.3]); let set = new Set(); set.add(1)

**作用**：**移除数组的重复元素**，判断重复依据：**===**。特别注意，NaN === NaN的结果为false，但是向 Set 实例添加了两次NaN，但是只会加入一个。这表明，在 Set 内部，两个NaN是相等的。

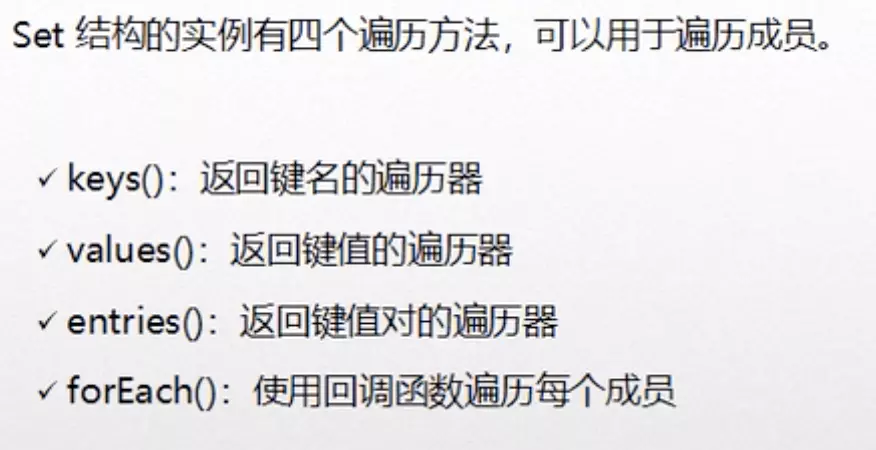
方法：add:添加元素，返回长度和set结果。delete：删除某个值，返回布尔值。has:判断是否包含某个元素，返回布尔值。clear清除set，没有返回值。

遍历方法:keys,values,entries,forEach。其中遍历的顺序就是插入的顺序，因此，可以保证回调函数的顺序执行。

**Map：**

JavaScript 的对象（Object），本质上是键值对的集合（Hash 结构），但是传统上只能用字符串当作键。这给它的使用带来了很大的限制。为了解决这个问题，**ES6 提供了 Map 数据结构。它类似于对象，也是键值对的集合，但是“键”的范围不限于字符串，各种类型的值（包括对象）都可以当作键。**也就是说，**Object 结构**提供了“**字符串—值**”的对应，**Map 结构**提供了“**值—值**”的对应，是一种更完善的 Hash 结构实现。如果你需要“键值对”的数据结构，Map 比 Object 更合适。

1. a={key:1,key2:2},console.log(new Set([a,a]))//[a]



12.js值传递和引用值传递

将对应的值拷贝了一份，然后赋值给新的变量。我们把它称作**值传递**。

对象是通过**引用传递**，而不是值传递。也就是说，**变量赋值只会将地址传递过去**。

* 栈：自动分配的内存空间，大小确定会自动释放。存放变量/局部变量/形参等。在js中存放简单数据段（五种基本数据类型：Number、String、Boolean、Null、Undefined），他们是按值存放的，可以直接访问。
* 堆：动态分配的内存，大小不定并且不会自动释放。存放在堆内存中的对象，栈中的变量实际保存的是一个指针，这个指针指向堆中的某一个位置。三种类型: Array、Function和Object，它们通过引用来传递。三者都是对象。

对于一个函数，给定一个输入，返回一个唯一的输出。除此之外，不会对外部环境产生任何附带影响，称该函数为**纯函数**。所有函数内部定义的变量在函数返回之后都被垃圾回收掉。

但是，如果函数的输入是对象(Array, Function, Object)，那么传入的是一个引用。对该变量的操作将会影响到原本的对象。这样的编程手法将产生附带影响，是的代码的逻辑复杂和可读性变低。

因此，很多数组函数，比如Array.map和Array.filter是以纯函数的形式实现。虽然它们的参数是一个数组变量，但是通过深度拷贝并赋值给一个新的变量，然后在新的数组上操作，来防止原始数组被更改。

13.promise有了解吗？状态？变化状态？

14.vue项目用什么做请求？ajax,fetch, axios区别？原生该怎么写都需要了解下？

普通的Ajax请求，用XHR发送一个json 请求

// xhr.readyState 有个5个状态 0 1 2 3 4 5

// xhr.readyState == 0 请求未初始化

// xhr.readyState == 1 服务器连接已建立

// xhr.readyState == 2 请求已接收

// xhr.readyState == 3 请求处理中

// xhr.readyState == 4 请求已完成，且响应已就绪

function ajax(options) {

options = options || {};

options.type = (options.type || "GET").toUpperCase();

options.dataType = options.dataType || "json";

var params = formatParams(options.data);

//创建xhr对象 - 非IE6

if (window.XMLHttpRequest) {

//1.创建一个XMLHttpRequest对象

var xhr = new XMLHttpRequest();

} else { //IE6及其以下版本浏览器

var xhr = new ActiveXObject('Microsoft.XMLHTTP');

}

//GET POST 两种请求方式

if (options.type == "GET") {

//2.请求方式，请求路径,是否异步

xhr.open("GET", options.url + "?" + params, true);

//

xhr.send(null);

} else if (options.type == "POST") {

xhr.open("POST", options.url, true);

//设置表单提交时的内容类型//如果是post请求，下面一行一定是设置请求头(固定写法)

xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

xhr.send(params);

}

//接收

xhr.onreadystatechange = function () {

if (xhr.readyState == 4) {

var status = xhr.status;

if (status >= 200 && status < 300) {

//发送成功

options.success && options.success(xhr.responseText);

} else {

//404 请求失败

options.fail && options.fail(status);

}

}

}

}

//格式化参数 //发送数据，只能发送字符串，想发送类似对象格式的数据，用 '&' 连接

function formatParams(data) {

var arr = [];

for (var name in data) {

arr.push(encodeURlComponent(name) + "=" + encodeURIComponent(data[name]));

}

arr.push(("v=" + Math.random()).replace(".",""));

return arr.join("&");

}

15.css3动画？

<div class = “rotate”></div>

.rotate{

width:300px;

height:300px;

border:1px solid red;

}

animation: rotate 3s linear infinite;

@keyframes rotate{

from{

transform: rotate(0deg)

}

To{

Tranform:rotate(360deg)

}

}

16.选择器有哪些？

1.标签选择器 2.类选择器、属性选择器（[rel=up]）和伪类选择器链接样式，a元素的伪类，4种状态：link,visited,hover,active） 3.ID选择器 4.通配符\* 5.组合选择器（.head .head\_logo）6.继承选择器(div p,注意两个选择器用空格分开) 7.元素和伪元素

伪元素选择器只包含以下几种::after，::before，::first-letter，::first-line，::selecton

**从0开始，一个行内样式+1000，一个id+100，一个属性选择器/class或者伪类+10，一个元素名，或者伪元素+1.**

17.选择第几个，:nth-child(n)选择器；表示父元素的第n个子元素。

18.css垂直居中几种方式？

19.js基本类型,引用类型，分别如何去判断？

20.call、apply、bind区别？bind实现，用构造函数如何实现bind?

参考答案：三者都可以把一个函数应用到其他对象上，注意不是自身对象．apply,call是直接执行函数调用，bind是绑定，执行需要再次调用．apply和call的区别是apply接受数组作为参数，而call是接受逗号分隔的无限多个参数列表，

代码演示

function Person() {

}

Person.prototype.sayName() { alert(this.name); }

var obj = {name: 'nn'}; // 注意这是一个普通对象，它不是Person的实例

1) apply

Person.prototype.sayName.apply(obj, [param1, param2, param3]);

2) call

Person.prototype.sayName.call(obj, param1, param2, param3);

3) bind

var sn = Person.prototype.sayName.bind(obj);

sn([param1, param2, param3]); // bind需要先绑定，再执行

sn(param1, param2, param3); // bind需要先绑定，再执行

**caller,callee之间的关系**就像是employer和employee之间的关系，就是调用与被调用的关系，二者返回的都是函数对象引用．**arguments**是函数的所有参数列表，它是一个类数组的变量．

代码演示

function parent(param1, param2, param3) {

child(param1, param2, param3);

}

function child() {

console.log(arguments); // { '0': 'mqin1', '1': 'mqin2', '2': 'mqin3' }

console.log(arguments.callee); // [Function: child]

console.log(child.caller); // [Function: parent]

}

parent('mqin1', 'mqin2', 'mqin3');

21.防抖节流，如何去优化？

22.js本地缓存？

23.ajax请求相关知识？

24.有了解过https吗？http,https区别？

25.last-modified(304),根据什么字段来判断有无修改？

26.vue封装组件

27.封装一个搜索框如何去封装？

28.webpack如何将scss转换成css?

29.webwapck解决跨域该如何解决？

* 通过jsonp跨域

jsonp的原理

script 标签的src属性可以跨域引用文件，jsonp是请求之后后台包装好一段json，并且把数据放在一个callback函数，返回一个js文件，动态引入这个文件，下载完成js之后，会去调用这个callback,通过这样访问数据。

在浏览器端定义一个回调函数，并将函数名通过src传至服务器端；服务器端将数据包装成为一段js数据，并返回js函数格式的js文件，接着拿到这个js文件之后函数自动调用，拿到后端返回的数据

* document.domain + iframe跨域
* location.hash + iframe
* window.name + iframe跨域
* postMessage跨域
* 跨域资源共享（CORS）

普通跨域请求：只服务端设置Access-Control-Allow-Origin即可，前端无须设置，若要带cookie请求：前后端都需要设置。

* nginx代理跨域

---nginx反向代理接口跨域

跨域原理： 同源策略是浏览器的安全策略，不是HTTP协议的一部分。服务器端调用HTTP接口只是使用HTTP协议，不会执行JS脚本，不需要同源策略，也就不存在跨越问题。

实现思路：通过nginx配置一个代理服务器（域名与domain1相同，端口不同）做跳板机，反向代理访问domain2接口，并且可以顺便修改cookie中domain信息，方便当前域cookie写入，实现跨域登录。

* nodejs中间件代理跨域
* WebSocket协议跨域

30.Apache？nginx?

**正向代理:客户端 <一> 代理 一>服务端**

**反向代理:客户端 一>代理 <一> 服务端**

正向代理的代理对象是客户端,反向代理的代理对象是服务端。

反向代理，其实**客户端**对代理是**无感知**的，因为客户端不需要任何配置就可以访问，只需要将请求发送到反向代理服务器，由反向代理服务器去选择目标服务器获取数据后，在返回给客户端，此时反向代理服务器和目标服务器对外就是一个服务器，暴露的是代理服务器地址，隐藏了真实服务器IP地址。