0、css布局方式

1、table布局（现在少用）

2、flex布局

3、float布局

4、响应式布局

1、说一下盒子模型（常问）

标准和模型和IE盒子模型

这两个的区别主要是

IE盒子模型的宽高包括content和padding还有border，标准盒子模型 不包括，

box-sizing:content-box 标准盒模型

box-sizing:border-box IE盒模型

2、Html5新标签

- **canvas 新元素**

标签	描述
<canvas>	标签定义图形，比如图表和其他图像。该标签基于 JavaScript 的绘图 API

新多媒体元素

标签	描述
<audio>	定义音频内容
<video>	定义视频（video 或者 movie）
<source>	定义多媒体资源 <video> 和 <audio>
<embed>	定义嵌入的内容，比如插件。
<track>	为诸如 <video> 和 <audio> 元素之类的媒介规定外部文本轨道。

新的语义和结构元素

HTML5提供了新的元素来创建更好的页面结构：

标签	描述
<article>	定义页面独立的内容区域。
<aside>	定义页面的侧边栏内容。
<bdi>	允许您设置一段文本，使其脱离其父元素的文本方向设置。
<details>	用于描述文档或文档某个部分的细节
<dialog>	定义对话框，比如提示框
<summary>	标签包含 details 元素的标题
<figure>	规定独立的流内容（图像、图表、照片、代码等等）。
<figcaption>	定义 <figure> 元素的标题
<footer>	定义 section 或 document 的页脚。
<header>	定义了文档的头部区域
<mark>	定义带有记号的文本。
<meter>	定义度量衡。仅用于已知最大和最小值的度量。
<nav>	定义导航链接的部分。
<progress>	定义任何类型的任务的进度。

3、BFC

`BFC`（Block Formatting Context），即块级格式化上下文，它是页面中一个独立的容器，容器中的元素不会影响到外面的元素

触发条件

触发`BFC`的条件包含不限于：

- 根元素，即HTML元素
- 浮动元素：float值为left、right
- overflow值不为 visible，为 auto、scroll、hidden
- display的值为inline-block、inltable-cell、table-caption、table、inline-table、flex、inline-flex、grid、inline-grid
- position的值为absolute或fixed

4、浏览器运行机制

1、创建DOM树

2、构建渲染树，CSS渲染

3、布局渲染，每个元素的大小、位置

4、绘制渲染树、再画出来

重绘：改变元素的外观属性例如div的color、background-color、等属性发生改变时

重排（回流）：元素的规模尺寸、布局、隐藏改变时

代价：耗时，导致浏览器卡慢

5、居中的方式

• 垂直居中的方式

行高=高

绝对定位 top50%，自身宽度的50%的负值

flex布局 align---center

• 水平居中的方式

绝对定位

flex布局 juest---center

text-align center

6、rem、em、vh、px各自代表的含义?

px：绝对单位，页面按精确像素展示

em：相对单位，基准点为父节点字体的大小，如果自身定义了 `font-size` 按自身来计算，整个页面内 `1em` 不是一个固定的值

rem：相对单位，可理解为 `root em`，相对根节点 `html` 的字体大小来计算

vh、vw：主要用于页面视口大小布局，在页面布局上更加方便简单

7、有哪些方式可以隐藏页面元素？区别？

通过`css`实现隐藏元素方法有如下：

- display:none
- visibility:hidden
- opacity:0
- 设置height、width模型属性为0
- position:absolute
- clip-path

关于`display: none`、`visibility: hidden`、`opacity: 0`的区别，如下表所示：

	display: none	visibility: hidden	opacity: 0
页面中	不存在	存在	存在
重排	会	不会	不会
重绘	会	会	不一定
自身绑定事件	不触发	不触发	可触发
transition	不支持	支持	支持
子元素可复原	不能	能	不能
被遮挡的元素可触发事件	能	能	不能

8、什么是响应式设计？响应式设计的基本原理是什么？如何做？

响应式网站设计（Responsive Web design）是一种网络页面设计布局，页面的设计与开发应当根据用户行为以及设备环境(系统平台、屏幕尺寸、屏幕定向等)进行相应的响应和调整

响应式网站常见特点：

- 同时适配PC + 平板 + 手机等
- 标签导航在接近手持终端设备时改变为经典的抽屉式导航
- 网站的布局会根据视口来调整模块的大小和位置

实现响应式布局的方式有如下：

- 媒体查询（我们可以设置不同类型的媒体条件，并根据对应的条件，给相应符合条件的媒体调用相对应的样式表）
- 百分比
- vw/vh
- rem

响应式设计实现通常会从以下几方面思考：

- 弹性盒子（包括图片、表格、视频）和媒体查询等技术
- 使用百分比布局创建流式布局的弹性UI，同时使用媒体查询限制元素的尺寸和内容变更范围
- 使用相对单位使得内容自适应调节
- 选择断点，针对不同断点实现不同布局和内容展示

9、css选择器有哪些？优先级？

关于`css`属性选择器常用的有：

- id选择器（#box），选择id为box的元素
- 类选择器（.one），选择类名为one的所有元素
- 标签选择器（div），选择标签为div的所有元素
- 后代选择器（#box div），选择id为box元素内部所有的div元素
- 子选择器（.one>one_1），选择父元素为.one的所有.one_1的元素
- 相邻同胞选择器（.one+.two），选择紧接在.one之后的所有.two元素
- 群组选择器（div,p），选择div、p的所有元素

还有一些使用频率相对没那么多的选择器：

- 伪类选择器

```
:link ：选择未被访问的链接
:visited：选取已被访问的链接
:active：选择活动链接
:hover ：鼠标指针浮动在上面的元素
:focus ：选择具有焦点的
:first-child：父元素的首个子元素
```

- 伪元素选择器

```
:first-letter ：用于选取指定选择器的首字母
:first-line ：选取指定选择器的首行
:before ： 选择器在被选元素的内容前面插入内容
:after ： 选择器在被选元素的内容后面插入内容
```

- 属性选择器

```
[attribute] 选择带有attribute属性的元素
[attribute=value] 选择所有使用attribute=value的元素
[attribute≃value] 选择attribute属性包含value的元素
[attribute⊨value]: 选择attribute属性以value开头的元素
```

在`CSS3`中新增的选择器有如下：

- 层次选择器（p~ul），选择前面有p元素的每个ul元素
- 伪类选择器

```
:first-of-type 父元素的首个元素
:last-of-type 父元素的最后一个元素
:only-of-type 父元素的特定类型的唯一子元素
:only-child 父元素中唯一子元素
:nth-child(n) 选择父元素中第N个子元素
:nth-last-of-type(n) 选择父元素中第N个子元素，从后往前
:last-child 父元素的最后一个元素
:root 设置HTML文档
:empty 指定空的元素
:enabled 选择被禁用元素
:disabled 选择被禁用元素
:checked 选择选中的元素
:not(selector) 选择非 <selector> 元素的所有元素
```

- 属性选择器

```
[attribute*=value]: 选择attribute属性值包含value的所有元素
[attribute^=value]: 选择attribute属性开头为value的所有元素
[attribute$=value]: 选择attribute属性结尾为value的所有元素
```

- **优先级**

内联 > ID选择器 > 类选择器 > 标签选择器

10、清除浮动的方法

方法一：使用带 clear 属性的空元素

在浮动元素后使用一个空元素,并在 CSS 中赋 予.clear{clear:both;}属性即可清理浮动。

方法二：使用 CSS 的 overflow 属性

给浮动元素的容器添加 overflow:hidden;或 overflow:auto;可以清除浮动，另外在 IE6 中还 需要触发 hasLayout ，例如为父元素设置容器宽高或设置 zoom:1。 在添加 overflow 属性后，浮动元素又回到了容器层，把容器高度撑起，达到了清理浮动 的效果。

方法三：给浮动的元素的容器添加浮动

给浮动元素的容器也添加上浮动属性即可清除内部浮动，但是这样会使其整体浮动，影 响布局，不推荐使用。

方法四：使用 CSS 的:after 伪元素

结合:after 伪元素（注意这不是伪类，而是伪元素，代表一个元素之后最近的元素）和 IEhack ， 可以完美兼容当前主流的各大浏览器，这里的 IEhack 指的是触发 hasLayout。 给浮动元素的容器添加一个 clearfix 的 class，然后给这个 class 添加一个:after 伪元素实 现元素末尾添加一个看不见的块元素清除浮动

11、常见的行内元素、块级元素

1、块级元素（div,p,h1...h6,ol,ul,table）

每个块级元素都是独自占一行、元素的高度宽度都是可以设置的

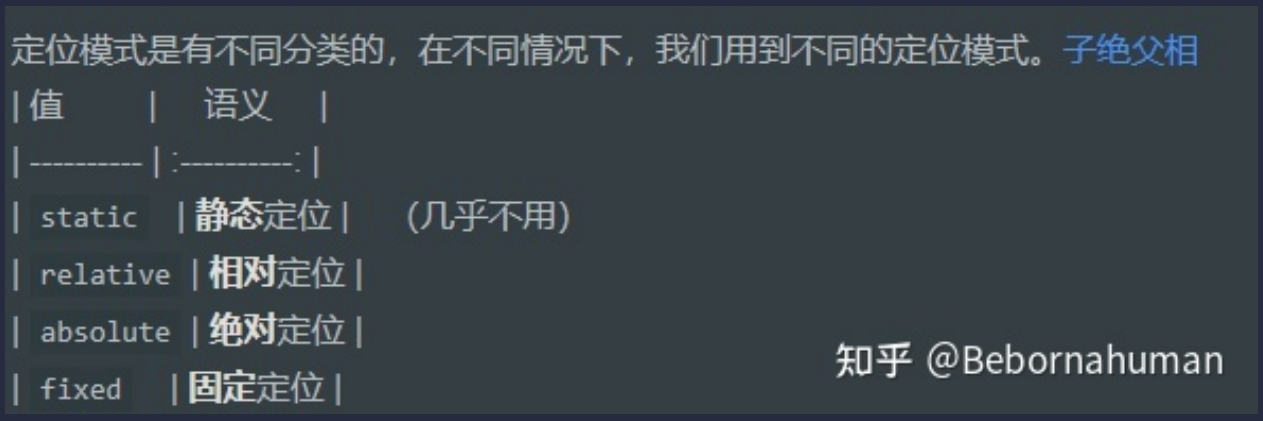
2、行内元素（span,a,img,input,strong）

可以和其他元素处于一行上，元素的高度宽度顶部和底部边距不可设置

12、position的属性

此处，问过相对定位和绝对定位的区别

相对定位：相对于当前元素的位子来移动；绝对定位如果不把父元素设置为相对定位，则相对与页面的左上角定位



13. 谈谈做好seo需要考虑什么？

- 语义化html标签
- 合理的title, description, keywords;

- 重要的html代码放前面
- 少用iframe, 搜索引擎不会抓取iframe中的内容
- 图片加上alt

JavaScript部分

0、ES6新增了哪些方法

1、includes（）**用于判断数组是否包含给定的值** 返回一个布尔值

2、find（）**用于找出第一个符合条件的数组成员**

3、findindex（）**返回第一个符合条件的数组成员的位置，如果所有成员都不符合条件，则返回-1**

4、set数据结构，类似于数组，但是成员的值都是唯一的，没有重复的值

5、、let声明变量、const声明常量（这里就要问你var、let、const的区别了）

6、解构赋值 ...

set 和map 的区别！！！以前被问没看过，懵逼过，所以要记住

1.Map是键值对，Set是值的集合，键和值可以是任何的值；

2.Map可以通过get方法获取值，而set不能因为它只有值，set只能用has来判断，返回一个布尔值；

4.Set的值是唯一的可以做数组去重，Map由于没有格式限制，可以做数据存储

1、promiseApi

`Promise` 构建出来的实例存在以下方法：

- then() 是实例状态发生改变时的回调函数，第一个参数是 `resolved` 状态的回调函数，第二个参数是 `rejected` 状态的回调函数
- catch() 用于指定发生错误时的回调函数
- finally() 用于指定不管 Promise 对象最后状态如何，都会执行的操作

`Promise` 构造函数存在以下方法：

- all() 用于将多个 `Promise` 实例，包装成一个新的 `Promise` 实例
- race() 同样是将多个 Promise 实例，包装成一个新的 Promise 实例
- allSettled()
- resolve()
- reject()
- try()

2、Var、 let 、const 区别?（常问、必记）

`var` 、 `let` 、 `const` 三者区别可以围绕下面五点展开：

- 变量提升

  var `声明的变量存在变量提升，即变量可以在声明之前调用，值为` undefined

  `let` 和 `const` 不存在变量提升，即它们所声明的变量一定要在声明后使用，

否则**报错**

- 暂时性死区

  `var`不存在暂时性死区

  `let`和`const`存在暂时性死区，只有等到声明变量的那一行代码出现，才可以获取和使用该变量

- 块级作用域

  `var`不存在块级作用域

  `let`和`const`存在块级作用域

- 重复声明

  `var`允许重复声明变量

  `let`和`const`在同一作用域不允许重复声明变量

- 修改声明的变量

  `var`和`let`可以

  `const`声明一个只读的常量。一旦声明，常量的值就不能改变

- 使用

  能用`const`的情况尽量使用`const`，其他情况下大多数使用`let`，避免使用`var`

3、== 和 ===区别

相等操作符（**）会做类型转换，再进行值的比较，全等运算符(**=)不会做类型转换

```
let result1 = ("55" === 55); // false，不相等，因为数据类型不同
let result2 = (55 === 55); // true，相等，因为数据类型相同值也相同
null` 和 `undefined` 比较，相等操作符（==）为`true`，全等为`false
let result1 = (null == undefined ); // true
let result2 = (null  === undefined); // false
```

4、数组常用方法

增

下面前三种是对原数组产生影响的增添方法，第四种则不会对原数组产生影响

- push() 接收任意数量的参数，并将它们添加到数组末尾，返回数组的最新长度
- unshift() 开头添加
- concat() 首先会创建一个当前数组的副本，然后再把它的参数添加到副本末尾，最后返回这个新构建的数组，不会影响原始数组

删

下面三种都会影响原数组，最后一项不影响原数组：

- pop() 删除数组的**最后一项**，同时减少数组的`length`值，返回被删除的项
- shift() 删除数组的**第一项**，同时减少数组的`length`值，返回被删除的项
- splice() 传入两个参数，分别是开始位置，删除元素的数量，返回包含删除元素的数组
- slice() 创建一个包含原有数组中一个或多个元素的新数组，不会影响原始数组

改

即修改原来数组的内容，常用`splice`

传入三个参数，分别是开始位置，要删除元素的数量，要插入的任意多个元素，返回删除元素的数组，对原数组产生影响

查

即查找元素，返回元素坐标或者元素值

- indexOf() 返回要查找的元素在数组中的位置，如果没找到则返回 -1
- includes() 返回要查找的元素在数组中的位置，找到返回`true`，否则`false`
- find() 返回第一个匹配的元素

排序方法

数组有两个方法可以用来对元素重新排序：

- reverse() 将数组元素方向反转

下面这个被问过，所以重点展开

- **sort(首元素地址(必填), 尾元素地址的下一个地址(必填), 比较函数(非必填));**

如果直接sort（数组名），则从小到大排序（即升序），以下为倒叙

```
var arr4 = [30,10,111,35,1899,50,45];
arr4.sort(function(a,b){
  return b - a;
})
console.log(arr4);//输出 [1899, 111, 50, 45, 35, 30, 10]
```

转换方法

常见的转换方法有：

join()

join() 方法接收一个参数，即字符串分隔符，返回包含所有项的字符串

迭代方法

常用来迭代数组的方法（都不改变原数组）有如下：

- some() 对数组每一项都运行传入的函数，如果**有一项函数返回 true**，则这个方法返回 true
- every() 对数组每一项都运行传入的函数，如果对**每一项函数都返回 true**，则这个方法返回 true
- forEach() 对数组每一项都运行传入的函数，没有返回值
- filter() 对数组每一项都运行传入的函数，函数返回 `true` 的项会组成数组之后返回
- map() 对数组每一项都运行传入的函数，返回由每次函数调用的结果构成的数组.

去重方法

- 1、利用ES6 Set去重（ES6中最常用）

```
function unique (arr) {
  return Array.from(new Set(arr))
}
var arr = [1,1,'true','true',true,true,15,15,false,false,
undefined,undefined, null,null, NaN, NaN,'NaN', 0, 0, 'a', 'a',{},
{}];
console.log(unique(arr))
 //[1, "true", true, 15, false, undefined, null, NaN, "NaN", 0,
"a", {}, {}]
```

- 2、利用for嵌套for，然后splice去重（ES5中最常用）

```
function unique(arr){
        for(var i=0; i<arr.length; i++){
            for(var j=i+1; j<arr.length; j++){
                if(arr[i]==arr[j]){         //第一个等同于第二个,
splice方法删除第二个
                    arr.splice(j,1);
                    j--;
                }
            }
        }
return arr;
}
```

```
var arr = [1,1,'true','true',true,true,15,15,false,false,
undefined,undefined, null,null, NaN, NaN,'NaN', 0, 0, 'a', 'a',{},
{}];
    console.log(unique(arr))
    //[1, "true", 15, false, undefined, NaN, NaN, "NaN", "a", {…},
{…}]     //NaN和{}没有去重，两个null直接消失了
```

3、利用indexOf去重

```
function unique(arr) {
    if (!Array.isArray(arr)) {
        console.log('type error!')
        return
    }
    var array = [];
    for (var i = 0; i < arr.length; i++) {
        if (array .indexOf(arr[i]) === -1) {
            array .push(arr[i])
        }
    }
    return array;
}
var arr = [1,1,'true','true',true,true,15,15,false,false,
undefined,undefined, null,null, NaN, NaN,'NaN', 0, 0, 'a', 'a',{},
{}];
console.log(unique(arr))
   // [1, "true", true, 15, false, undefined, null, NaN, NaN,
"NaN", 0, "a", {…}, {…}]  //NaN、{}没有去重
```

- **4、利用includes**

```
function unique(arr) {
    if (!Array.isArray(arr)) {
        console.log('type error!')
        return
    }
    var array =[];
    for(var i = 0; i < arr.length; i++) {
            if( !array.includes( arr[i]) ) {//includes 检测数组是否有某个值
                    array.push(arr[i]);
              }
    }
    return array
}
var arr = [1,1,'true','true',true,true,15,15,false,false, undefined,undefined, null,null, NaN, NaN,'NaN', 0, 0, 'a', 'a',{}, {}];
    console.log(unique(arr))
    //[1, "true", true, 15, false, undefined, null, NaN, "NaN", 0, "a", {…}, {…}]     //{}没有去重
```

5、bind、call、apply 区别

- 1.call和apply会调用函数，且会改变函数内部的this指向

  2.call和apply传递的参数不一样，call传递参数aru1，aru2.形式 而apply必须是数组形式[arg]

  3.bind 不会调用函数，可以改变函数内部指向
- 应用场景：

  1.call经常做继承

  2.apply经常和数组有关系，比如借助于数学对象实现数组的max、min

  3.bind不调用函数，但改变this指向，比如改变定时器内部的this指向
- apply：调用一个对象的一个方法，用另一个对象替换当前对象。例如：B.apply(A, arguments);即 A 对象应用 B 对象的方法。 call：调用一个对象的一个方法，用另一个对象替换当前对象。例如：B.call(A, args1,args2); 即 A 对象调用 B 对象的方法。 bind 除了返回是函数以外，它的参数和 call 一样。

6、本地存储的方式有哪些？区别及应用场景？

`javaScript`本地缓存的方法我们主要讲述以下四种：

- cookie
- sessionStorage
- localStorage
- indexedDB

区别

关于`cookie`、`sessionStorage`、`localStorage`三者的区别主要如下：

- 存储大小：`cookie`数据大小不能超过`4k`，`sessionStorage`和`localStorage`虽然也有存储大小的限制，但比`cookie`大得多，可以达到5M或更大
- 有效时间：`localStorage`存储持久数据，浏览器关闭后数据不丢失除非主动删除数据；`sessionStorage`数据在当前浏览器窗口关闭后自动删除；`cookie`设置的`cookie`过期时间之前一直有效，即使窗口或浏览器关闭
- 数据与服务器之间的交互方式，`cookie`的数据会自动的传递到服务器，服务器端也可以写`cookie`到客户端；`sessionStorage`和`localStorage`不会自动把数据发给服务器，仅在本地保存

应用场景

在了解了上述的前端的缓存方式后，我们可以看看针对不对场景的使用选择：

- 标记用户与跟踪用户行为的情况，推荐使用`cookie`
- 适合长期保存在本地的数据（令牌），推荐使用`localStorage`
- 敏感账号一次性登录，推荐使用`sessionStorage`
- 存储大量数据的情况、在线文档（富文本编辑器）保存编辑历史的情况，推荐使用`indexedDB`

7、说说你对闭包的理解？闭包使用场景

闭包就是函数中包含另一个函数，可以让你在函数外部读取到内部的变量（就是在函数内部再定义一个函数），让这些变量的值始终保持在内存中，可以达到延长变量生命周期的效果，过多使用会导致内存泄漏的问题

（在创建私有变量和想延长变量的生命周期时会用到闭包）

8、深拷贝浅拷贝的区别?

浅拷贝，指的是创建新的数据，这个数据有着原始数据属性值的一份精确拷贝

如果属性是基本类型，拷贝的就是基本类型的值。如果属性是引用类型，拷贝的就是内存地址

在`JavaScript`中，存在浅拷贝的现象有：

- Object.assign
- Array.prototype.slice()
- Array.prototype.concat()
- 使用拓展运算符实现的复制

深拷贝开辟一个新的栈，两个对象属完成相同，但是对应两个不同的地址，修改一个对象的属性，不会改变另一个对象的属性

常见的深拷贝方式有：

- _.cloneDeep()
- jQuery.extend()
- JSON.stringify()
- 手写循环递归

9、JavaScript中的数据类型?

string、number、Boolean、undefined、null、object

10、什么是防抖和节流?

定义

- 防抖: n 秒后在执行该事件，若在 n 秒内被重复触发，则重新计时
- 节流: n 秒内只运行一次，若在 n 秒内重复触发，只有一次生效

电梯第一个人进来后，等待15秒。如果过程中又有人进来，15秒等待重新计时，直到15秒后开始运送，这是**防抖**

电梯第一个人进来后，15秒后准时运送一次，这是**节流**

11、如何解决数字精度丢失的问题?

理论上用有限的空间来存储无限的小数是不可能保证精确的，但我们可以处理一下得到我们期望的结果

当你拿到 `1.4000000000000001` 这样的数据要展示时，建议使用 `toPrecision` 凑整并 `parseFloat` 转成数字后再显示，如下:

```
parseFloat(1.4000000000000001.toPrecision(12)) === 1.4  // True
```

封装成方法就是:

```
function strip(num, precision = 12) {
  return +parseFloat(num.toPrecision(precision));
}
```

最后还可以使用第三方库，如Math.js、BigDecimal.js

12、 JavaScript 中内存泄漏的几种情况?

使用闭包

13、原型，原型链？有什么特点?

`JavaScript` 常被描述为一种基于原型的语言——每个对象拥有一个原型对象

当试图访问一个对象的属性时，它不仅仅在该对象上搜寻，还会搜寻该对象的原型，以及该对象的原型的原型，依次层层向上搜索，直到找到一个名字匹配的属性或到达原型链的末尾

原型对象也可能拥有原型，并从中继承方法和属性，一层一层、以此类推。这种关系常被称为原型链 (prototype chain)，它解释了为何一个对象会拥有定义在其他对象中的属性和方法

14、如何实现上拉加载，下拉刷新?

开源社区有很多优秀的解决方案，如 `iscroll` 、 `better-scroll` 、 `pulltorefresh.js` 库等等

15、说说你对作用域链的理解

1、作用域就是变量与函数的可访问范围

2、一般情况下，变量取值到创建这个变量的函数的作用域中取值。 但是如果在当前作用域中没有查到值，就会向上级作用域去查，直到查到全局作用域，这么一个查找过程形成的链条就叫做作用域链

16、typeof 与 instanceof 区别

`typeof` 与 `instanceof` 都是判断数据类型的方法，区别如下：

- `typeof` 会返回一个变量的基本类型， `instanceof` 返回的是一个布尔值
- `instanceof` 可以准确地判断复杂引用数据类型，但是不能正确判断基础数据类型
- 而 `typeof` 也存在弊端，它虽然可以判断基础数据类型（`null` 除外），但是引用数据类型中，除了 `function` 类型以外，其他的也无法判断

17、js基本数据类型

string、number、null、defined、boolean、object、symbol、bigint

18、ajax、axios、jsonp的理解

1、jsonp是一种可以解决跨域问题的方式，就是通过动态创建script标签用src引入外部文件实现跨域，script加载实际上就是一个get请求，并不能实现post请求。(其他实现跨域的方法有：iframe,window.name,postMessage,CORS...)

2、ajax是一种技术，ajax技术包含了get和post请求的，但是它仅仅是一种获取数据的技术，不能直接实现跨域，只有后台服务器配置好Access-Control-Allow-Origin，才可以实现请求的跨域。

4、axios是通过promise实现对ajax技术的一种封装，axios是ajax，ajax不止axios。

总结：

juery的$.ajax实现get请求能跨域是因为jsonp或者因为原生ajax和服务器的配合，post请求能跨域就只能是因为原生ajax和服务器的配合。

19、ajax的请求过程

```
// ajax 提交 post 请求的数据
// 1. 创建核心对象
var xhr = new XMLHttpRequest();
// 2. 准备建立连接
xhr.open("POST", "register.php", true);
// 3. 发送请求
// 如果要POST提交数据，则需要设置请求头
// 有的面试官会问为什么要设置请求头？ 知道请求正文是以什么格式
// Content-Type: application/x-www-form-urlencoded，请求正文是类似 get 请求 url 的请求参数
// Content-Type: application/json，请求正文是一个 json 格式的字符串
xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
// 发送数据
```

```
xhr.send(querystring);
// 4. 处理响应
xhr.onreadystatechange = function () {
     if (xhr.readyState == 4) { // 请求处理完毕，响应就绪
          if (xhr.status == 200) { // 请求成功
               var data = xhr.responseText;
               console.log(data);
          }
     }
}
```

20、ajax请求的时候get 和post方式的区别

1、get请求不安全，post安全；

2、get请求数据有大小限制，post无限制；

3、get请求参数会在url中显示，容易被他人窃取，post在请求体中，不会被窃取；

4、post需要设置请求头。

21、什么是事件委托以及优缺点

js事件委托就是利用冒泡的原理，把本应该添加到某个元素上的事件委托给他的父级，从而减少DOM交互达到网页优化。

优点：

1.可以大量节省内存占用，减少事件注册。比如ul上代理所有li的click事件就很不错。 2.可以实现当新增子对象时，无需再对其进行事件绑定，对于动态内容部分尤为合适

缺点：

事件代理的常用应用应该仅限于上述需求，如果把所有事件都用事件代理，可能会出现事件误判。即本不该被触发的事件被绑定上了事件。

Vue部分

1、为什么使用虚拟DOM(常问)

- 创建真实DOM的代价高：真实的 DOM 节点 node 实现的属性很多，而 vnode 仅仅实现一些必要的属性，相比起来，创建一个 vnode 的成本比较低。
- 触发多次浏览器重绘及回流：使用 vnode ，相当于加了一个缓冲，让一次数据变动所带来的所有 node 变化，先在 vnode 中进行修改，然后 diff 之后对所有产生差异的节点集中一次对 DOM tree 进行修改，以减少浏览器的重绘及回流。
- 虚拟dom由于本质是一个js对象，因此天生具备跨平台的能力，可以实现在不同平台的准确显示。
- Virtual DOM 在性能上的收益并不是最主要的，更重要的是它使得 Vue 具备了现代框架应有的高级特性。

2、Vue组件通信

父组件向子组件传值

- 父组件发送的形式是以属性的形式绑定值到子组件身上。
- 然后子组件用属性props接收
- 在props中使用驼峰形式，模板中需要使用短横线的形式字符串形式的模板中没有这个限制

子组件向父组件传值

- 子组件用`$emit()`触发事件
- `$emit()` 第一个参数为 自定义的事件名称 第二个参数为需要传递的数据 $(event)来接收
- 父组件用v-on 缩写为@ 监听子组件的事件

兄弟之间的传递

- 兄弟之间传递数据需要借助于事件中心，通过事件中心传递数据
    - 提供事件中心 var hub = new Vue()

- 传递数据方，通过一个事件触发hub.$emit(方法名，传递的数据)
- 接收数据方，通过mounted(){} 钩子中 触发hub.$on()方法名

- 销毁事件 通过hub.$off()方法名销毁之后无法进行传递数据

3、Vue中key是用来做什么的？为什么不推介使用index作为key?

1、key的作用主要是为了高效的更新虚拟DOM（使用key，它会基于key的变化重新排列元素顺序，并且会移除key不存在的元素）

2、当以数组的下标index作为index值时，其中一个元素（如增删改查）发生了变化就有可能导致所有元素的key值发生变化

4、生命周期：

从Vue实例创建、运行、到销毁期间，伴随着的各种事件，这些事件统称为生命周期

生命周期函数分类：

- 创建期间的生命周期函数：
  - **beforeCreate**：实例刚在内存中被创建出来，此时还没有初始化好data和methods属性
  - **created**：实例已经在内存中创建出来，此时的data和methods以及创建完成，但是还没有开始编译模板
  - **beforeMount**：此时已经完成了模板的编译，但是还没有挂载到页面上

- **mounted**：已经将编译好的模板，挂载到了页面指定的容器中显示

- 运行期间的生命周期函数：
  - **beforeUpdate**：状态更新之前执行此函数，此时data中的状态值是最新的，但是界面上显示的数据还是旧的，因为此时还没有开始重新渲染DOM节点
  - **updated**：实例更新完毕之后调用此函数，此时data中的状态值和界面上显示的数据，都已经完成了更新，界面已经被重新渲染好了

- 销毁期间的生命周期函数：
- 注意！！！vue3，则是beforeunmount和unmount
  - **beforeDestory**：实例销毁之前调用，在这一步，实例仍然完全可用
  - **destroyed**：Vue实例销毁之后调用。调用后，Vue实例指示的所有东西都会解绑定，所有的事件监听器会被移除，所有的子实例也会被销毁

5、v-show和v-if的区别

v-show原理是修改元素的css属性display:none来决定是显示还是隐藏

v-if则是通过操作DOM来进行切换显示

6、双向数据绑定

实现mvvm的双向绑定，是采用数据劫持结合**发布者-订阅者模式**的方式，通过Object.defineProperty()来劫持各个属性的setter，getter，在数据变动时发布消息给订阅者，触发相应的监听回调。

7、Vue导航守卫的钩子函数有哪些?

全局守卫

- router.beforeEach：全局前置守卫，进入路由之前
- router.beforeResolve：全局解析守卫，在beforeRouteEnter调用之后调用
- router.afterEach：全局后置钩子，进入路由之后

路由组件内的守卫

- beforeRouteEnter()：进入路由前
- beforeRouteUpdate()：路由复用同一个组件时
- beforeRouteLeave()：离开当前路由时

8、vue编程式的导航跳转传参的方式有哪些?

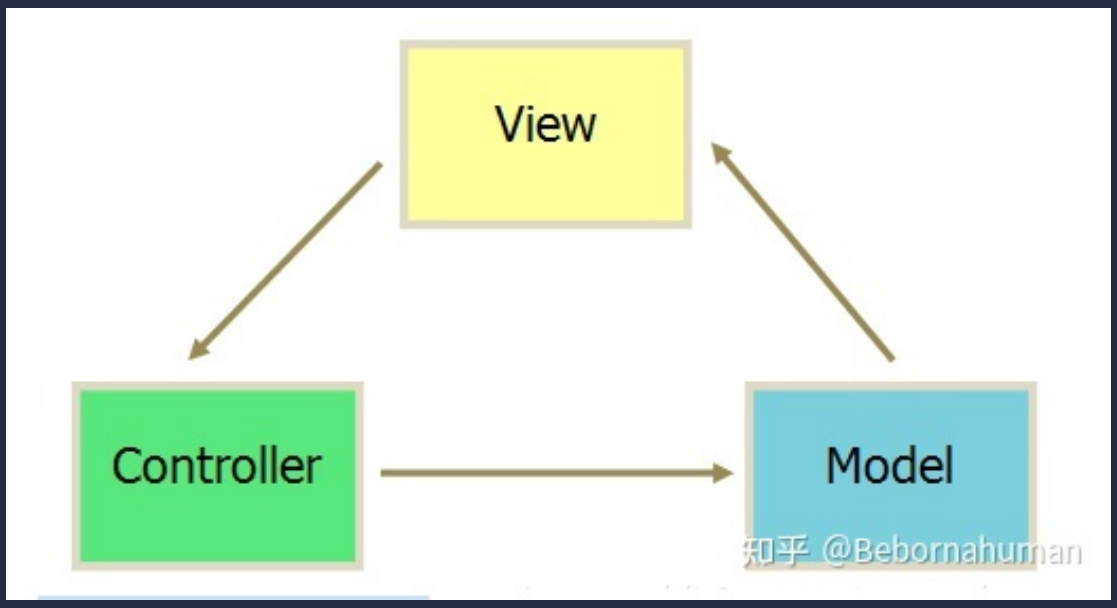
```
// 命名的路由
router.push({ name: 'user', params: { userId: '123' }})

// 带查询参数，变成 /register?plan=private
router.push({ path: 'register', query: { plan: 'private' }})
```

9、vuex是什么？怎么使用？哪种功能场景使用它?

在main.js引入store，注入。新建了一个store目录，然后..... export 。 场景：单页应用中，组件之间的共享状态和方法 **state** Vuex 使用单一状态树,即每个应用将仅仅包含一个store 实例，但单一状态树和模块化并不冲突。存放的数据状态，不可以直接修改里面的数据。 **mutations**mutations定义的方法动态修改Vuex 的 store 中的状态或数据。 **getters** 类似vue的计算属性，主要用来过滤一些数据。 **action** actions可以理解为通过将mutations里面处里数据的方法变成可异步的处理数据的方法，简单的说就是异步操作数据。view 层通过 store.dispath 来分发 action。 **modules** 项目特别复杂的时候，可以让每一个模块拥有自己的state、mutation、action、getters,使得结构非常清晰，方便管理。

10、mvc和mvvc的区别

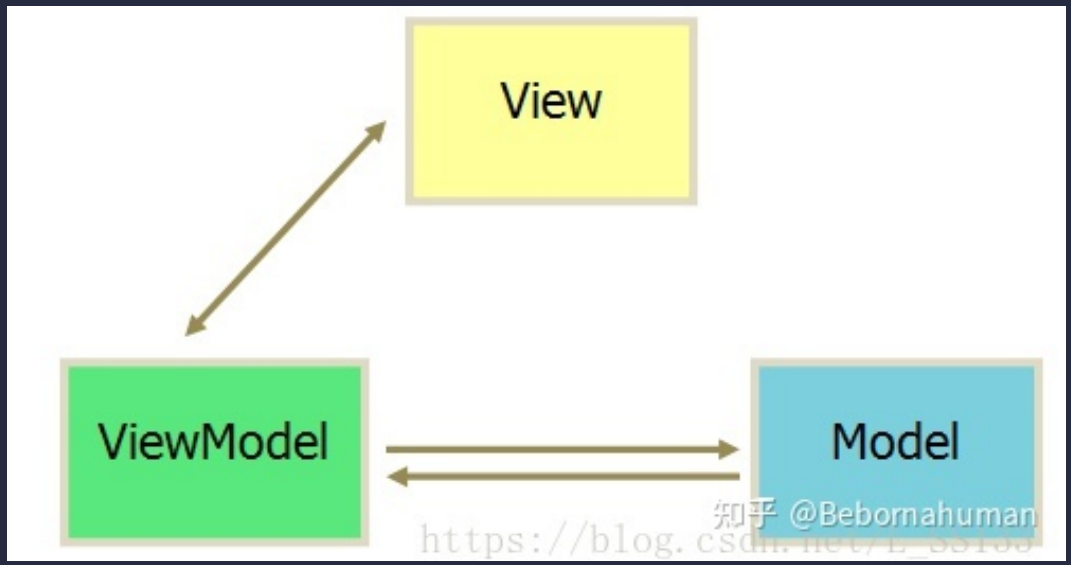
• MVC



MVC包括view视图层、controller控制层、model数据层。各部分之间的通信都是单向的。

View 传送指令到 Controller Controller 完成业务逻辑后，要求 Model 改变状态 Model 将新的数据发送到 View，用户得到反馈

• MVVM



MVVM包括view视图层、model数据层、viewmodel层。各部分通信都是双向的。

采用双向数据绑定，View的变动，自动反映在 ViewModel，反之亦然。 mvvm代表框架：Angularjs、React、Vue mvvm主要解决了mvc中大量 dom操作使得页面渲染性能降低，加载速度变慢，影响用户体验

11、说出至少vue 3个常用事件修饰符?

.stop 阻止点击事件冒泡

.prevent 阻止默认事件

.once 只执行一次

.self 只在元素本身触发

12、vuex有哪几种属性

有五种,分别是State，Getter，Mutation，Action，Module (就是mapAction)

1. state：vuex的基本数据，用来存储变量
2. geeter：从基本数据(state)派生的数据，相当于state的计算属性
3. mutation：提交更新数据的方法，必须是同步的(如果需要异步使用action)。每个mutation 都有一个字符串的 事件类型 (type) 和 一个 回调函数 (handler)。回调函数就是我们实际进行状态更改的地方，并且它会接受 state 作为第一个参数，提交载荷作为第二个参数。

4. action：和mutation的功能大致相同，不同之处在于 ==》1. Action 提交的是 mutation，而不是直接变更状态。 2. Action 可以包含任意异步操作。
5. modules：模块化vuex，可以让每一个模块拥有自己的state、mutation、action、getters,使得结构非常清晰，方便管理。

HTTP

1、说一下http和https

http：超文本传输协议，是一个客户端和服务器端请求和应答的标准（TCP）

https：是以安全为目标的HTTP通道，即HPPT下加入SSL层，比http更安全

区别：

http传输的数据都是未加密的（明文），https协议是由https和SSL协议构建的可进行加密传输和身份认证的网络协议，需要ca证书，费用较高

2、tcp三次握手，一句话概括

客户端和服务端都需要直到各自可收发，因此需要三次握手

3、HTTP状态码

- **1xx Informational（信息性状态码） 接受的请求正在处理**

- **2xx Success（成功状态码） 请求正常处理完毕**

（1）、**200 OK**：表示从客户端发送给服务器的请求被正常处理并返回；

（2）、204 No Content：表示客户端发送给客户端的请求得到了成功处理，但在返回的响应报文中不含实体的主体部分（没有资源可以返回）；

（3）、206 Patial Content：表示客户端进行了范围请求，并且服务器成功执行了这部分的GET请求，响应报文中包含由Content-Range指定范围的实体内容。

- **3xx Redirection（重定向） 需要进行附加操作以完成请求**

（1）、301 Moved Permanently：永久性重定向，表示请求的资源被分配了新的URL，之后应使用更改的URL；

（2）、302 Found：临时性重定向，表示请求的资源被分配了新的URL，希望本次访问使用新的URL；

（3）、301与302的区别：前者是永久移动，后者是临时移动（之后可能还会更改URL）

（4）、303 See Other：表示请求的资源被分配了新的URL，应使用GET方法定向获取请求的资源；

（5）、302与303的区别：后者明确表示客户端应当采用GET方式获取资源

（6）、304 Not Modified：表示客户端发送附带条件（是指采用GET方法的请求报文中包含if-Match、If-Modified-Since、If-None-Match、If-Range、If-Unmodified-Since中任一首部）的请求时，服务器端允许访问资源，但是请求为满足条件的情况下返回改状态码；

（7）、307 Temporary Redirect：临时重定向，与303有着相同的含义，307会遵照浏览器标准不会从POST变成GET；（不同浏览器可能会出现不同的情况）；

- **4xx Client error（客户端错误） 客户端请求出错，服务器无法处理请求**

400 Bad Request：表示请求报文中存在语法错误；

401 Unauthorized：未经许可，需要通过HTTP认证；

403 Forbidden：服务器拒绝该次访问（访问权限出现问题）

404 Not Found：表示服务器上无法找到请求的资源，除此之外，也可以在服务器拒绝请求但不想给拒绝原因时使用；

- **5xx Server Error（服务器错误） 服务器处理请求出错**

500 Inter Server Error：表示服务器在执行请求时发生了错误，也有可能是web应用存在的bug或某些临时的错误时；

503 Server Unavailable：表示服务器暂时处于超负载或正在进行停机维护，无法处理请求；

问你项目描述：

1.你这个项目的整体描述，需求，功能；

2.你这个项目所用到的技术栈，以及选择该技术栈的原因；

3.你这个项目中所遇到的问题，以及解决方法。

问你前端优化：

1、降低请求量：合并资源，减少HTTP请求数，minify/gzip压缩

2、加快请求速度：预解析DNS，减少域名数，并行加载

3、缓存：HTTP协议缓存请求，离线缓存manifest，离线数据缓存localStorage

4、渲染：JS/CSS优化，加载顺序，服务端渲染

一、HTML 篇

• 1. 简述一下你对 HTML 语义化的理解?

> “
> 用正确的标签做正确的事情。
> html 语义化让页面的内容结构化，结构更清晰，便于对浏览器、搜索引擎解析；即使在没有样式 CSS 情况下也以一种文档格式显示，并且是容易阅读的;
> 搜索引擎的爬虫也依赖于 HTML 标记来确定上下文和各个关键字的权重，利于 SEO;
> 使阅读源代码的人对网站更容易将网站分块，便于阅读维护理解。

2. 标签上 title 与 alt 属性的区别是什么?

> alt 是给搜索引擎识别，在图像无法显示时的替代文本；
> title 是关于元素的注释信息，主要是给用户解读。
> 当鼠标放到文字或是图片上时有 title 文字显示。（因为 IE 不标准）在 IE 浏览器中 alt 起到了 title 的作用，变成文字提示。
> 在定义 img 对象时，将 alt 和 title 属性写全，可以保证在各种浏览器中都能正常使用。

3. iframe的优缺点?

> 优点：
>
> - 解决加载缓慢的第三方内容如图标和广告等的加载问题
> - Security sandbox
> - 并行加载脚本
>
> ---
>
> 缺点：
>
> - iframe会阻塞主页面的Onload事件
> - 即时内容为空，加载也需要时间
> - 没有语意

4. href 与 src?

> “
>
> - href (Hypertext Reference)指定网络资源的位置，从而在当前元素或者当前文档和由当前属性定义的需要的锚点或资源之间定义一个链接或者关系。（目的不是为了引用资源，而是为了建立联系，让当前标签能够链接到目标地址。）
> - src source（缩写），指向外部资源的位置，指向的内容将会应用到文档中当前标签所在位置。
> - href与src的区别
>   - 1、请求资源类型不同：href 指向网络资源所在位置，建立和当前元素（锚点）或当前文档（链接）之间的联系。在请求 src 资源时会将其指向的资源下载并应用到文档中，比如 JavaScript 脚本，img 图片；
>   - 2、作用结果不同：href 用于在当前文档和引用资源之间确立联系；src 用于替换当前内容；
>   - 3、浏览器解析方式不同：当浏览器解析到src ，会暂停其他资源的下载和处理，直到将该资源加载、编译、执行完毕，图片和框架等也如此，类似于将所指向资源应用到当前内容。这也是为什么建议把 js 脚本放在底部而不是头部的原因。

二、CSS 篇

1. 介绍一下 CSS 的盒子模型?

> 有两种， IE 盒子模型、W3C 盒子模型；
> 盒模型： 内容(content)、填充(padding)、边界(margin)、 边框(border)；
> 区 别： IE 的 content 部分把 border 和 padding 计算了进去;

2. css 选择器优先级?

> !important > 行内样式（比重1000） > ID 选择器（比重100） > 类选择器（比重10） > 标签（比重1） > 通配符 > 继承 > 浏览器默认属性

3. 垂直居中几种方式?

> 单行文本: line-height = height
> 图片: vertical-align: middle;
> absolute 定位: top: 50%;left: 50%;transform: translate(-50%, -50%);
> flex: display:flex;margin:auto

4. 简明说一下 CSS link 与 @import 的区别和用法?

> link 是 XHTML 标签，除了加载CSS外，还可以定义 RSS 等其他事务；
> @import 属于 CSS 范畴，只能加载 CSS。
> link 引用 CSS 时，在页面载入时同时加载；@import 需要页面网页完全载入以后加载。
> link 是 XHTML 标签，无兼容问题；@import 是在 CSS2.1 提出的，低版本的浏览器不支持。

> link 支持使用 Javascript 控制 DOM 去改变样式；而@import不支持。

• 5. rgba和opacity的透明效果有什么不同?

> “
> opacity 会继承父元素的 opacity 属性，而 RGBA 设置的元素的后代元素不会继承不透明属性。

• 6. display:none和visibility:hidden的区别?

> “
> display:none 隐藏对应的元素，在文档布局中不再给它分配空间，它各边的元素会合拢，就当他从来不存在。
> visibility:hidden 隐藏对应的元素，但是在文档布局中仍保留原来的空间。

• 7. position的值， relative和absolute分别是相对于谁进行定位的?

> “
> relative:相对定位，相对于自己本身在正常文档流中的位置进行定位。
> absolute:生成绝对定位，相对于最近一级定位不为static的父元素进行定位。
> fixed: （老版本IE不支持）生成绝对定位，相对于浏览器窗口或者frame进行定位。
> static:默认值，没有定位，元素出现在正常的文档流中。
> sticky:生成粘性定位的元素，容器的位置根据正常文档流计算得出。

三、HTML / CSS 混合篇

1. HTML5、CSS3 里面都新增了那些新特性?

> **HTML5**
>
> - 新的语义标签
>   - article 独立的内容。
>   - aside 侧边栏。
>   - header 头部。
>   - nav 导航。
>   - section 文档中的节。
>   - footer 页脚。
> - 画布(Canvas) API
> - 地理(Geolocation) API
> - 本地离线存储 localStorage 长期存储数据，浏览器关闭后数据不丢失；sessionStorage 的数据在浏览器关闭后自动删除
> - 新的技术webworker, websocket, Geolocation
> - 拖拽释放(Drag and drop) API
> - 音频、视频API(audio,video)
> - 表单控件，calendar、date、time、email、url、searc

CSS3

- 2d，3d变换
- Transition, animation
- 媒体查询
- 新的单位（rem, vw，vh 等）
- 圆角（border-radius），阴影（box-shadow），对文字加特效（text-shadow），线性渐变（gradient），旋转（transform）

  transform:rotate(9deg) scale(0.85,0.90) translate(0px,-30px) skew(-9deg,0deg);//旋转,缩放,定位,倾斜
- rgba

2. BFC 是什么?

> BFC 即 Block Formatting Contexts (块级格式化上下文)，它属于普通流，
> 即：元素按照其在 HTML 中的先后位置至上而下布局，在这个过程中，行内元素水平排列，直到当行被占满然后换行，块级元素则会被渲染为完整的一个新行，除非另外指定，否则所有元素默认都是普通流定位，也可以说，普通流中元素的位置由该元素在 HTML 文档中的位置决定。
> 可以把 BFC 理解为一个封闭的大箱子，箱子内部的元素无论如何翻江倒海，都不会影响到外部。
> 只要元素满足下面任一条件即可触发 BFC 特性
>
> - body 根元素
> - 浮动元素：float 除 none 以外的值
> - 绝对定位元素：position (absolute、fixed)
> - display 为 inline-block、table-cells、flex
> - overflow 除了 visible 以外的值 (hidden、auto、scroll)

3. 常见兼容性问题?

> - 浏览器默认的margin和padding不同。解决方案是加一个全局的*{margin:0;padding:0;}来统一。
> - Chrome 中文界面下默认会将小于 12px 的文本强制按照 12px 显示,可通过加入 CSS 属性 -webkit-text-size-adjust: none; 解决.

四、JS 篇

1. JS 数据类型 ?

> 数据类型主要包括两部分:
>
> - 基本数据类型: Undefined、Null、Boolean、Number 和 String
> - 引用数据类型: Object (包括 Object 、Array 、Function)
> - ECMAScript 2015 新增:Symbol(创建后独一无二且不可变的数据类型)

2. 判断一个值是什么类型有哪些方法?

> - typeof 运算符
> - instanceof 运算符

> - Object.prototype.toString 方法

3. null 和 undefined 的区别?

> “
> null 表示一个对象被定义了，值为“空值”；
> undefined 表示不存在这个值。
> （1）变量被声明了，但没有赋值时，就等于undefined。 （2) 调用函数时，应该提供的参数没有提供，该参数等于undefined。 （3）对象没有赋值的属性，该属性的值为undefined。 （4）函数没有返回值时，默认返回undefined。

4. 怎么判断一个变量arr的话是否为数组（此题用 typeof 不行）？

> “
> arr instanceof Array
> arr.constructor == Array
> Object.protype.toString.call(arr) == '[Object Array]'

5. “ ===”、“ ==”的区别?

> “
> **，当且仅当两个运算数相等时，它返回 true，即不检查数据类型**
> **=**，只有在无需类型转换运算数就相等的情况下，才返回 true，需要检查数据类型

6. “eval是做什么的?

> “
> 它的功能是把对应的字符串解析成 JS 代码并运行；
> 应该避免使用 eval，不安全，非常耗性能（2次，一次解析成 js 语句，一次执行）。

7. 箭头函数有哪些特点?

> “
> 不需要function关键字来创建函数
> 省略return关键字
> 改变this指向

8. var、let、const 区别?

> “
> var 存在变量提升。
> let 只能在块级作用域内访问。
> const 用来定义常量，必须初始化，不能修改（对象特殊）

9. new操作符具体干了什么呢?

> “
> 1、创建一个空对象，并且 this 变量引用该对象，同时还继承了该函数的原型。
> 2、属性和方法被加入到 this 引用的对象中。
> 3、新创建的对象由 this 所引用，并且最后隐式的返回 this 。

10. JSON 的了解?

> JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式。
> 它是基于JavaScript的一个子集。数据格式简单, 易于读写, 占用带宽小
> {'age':'12', 'name':'back'}

11. documen.write 和 innerHTML 的区别?

> document.write 只能重绘整个页面
> innerHTML 可以重绘页面的一部分

12. ajax过程?

> (1)创建XMLHttpRequest对象,也就是创建一个异步调用对象.
> (2)创建一个新的HTTP请求,并指定该HTTP请求的方法、URL及验证信息.
> (3)设置响应HTTP请求状态变化的函数.
> (4)发送HTTP请求.
> (5)获取异步调用返回的数据.
> (6)使用JavaScript和DOM实现局部刷新.

13. 请解释一下 JavaScript 的同源策略?

> 概念:同源策略是客户端脚本（尤其是Netscape Navigator2.0，其目的是防止某个文档或脚本从多个不同源装载。
> 这里的同源策略指的是：协议，域名，端口相同，同源策略是一种安全协议。
> 指一段脚本只能读取来自同一来源的窗口和文档的属性。

14. 介绍一下闭包和闭包常用场景?

> “
>
> - 闭包是指有权访问另一个函数作用域中的变量的函数，创建闭包常见方式，就是在一个函数的内部创建另一个函数
> - 使用闭包主要为了设计私有的方法和变量，闭包的优点是可以避免变量的污染，缺点是闭包会常驻内存，会增大内存使用量，使用不当很容易造成内存泄露。在js中，函数即闭包，只有函数才会产生作用域的概念。
> - 闭包有三个特性：
>   - 函数嵌套函数
>   - 函数内部可以引用外部的参数和变量
>   - 参数和变量不会被垃圾回收机制回收
> - 应用场景，设置私有变量的方法
> - 不适用场景：返回闭包的函数是个非常大的函数
> - 闭包的缺点就是常驻内存，会增大内存使用量，使用不当会造成内存泄漏

15. javascript的内存(垃圾)回收机制?

> “
>
> - 垃圾回收器会每隔一段时间找出那些不再使用的内存，然后为其释放内存
> - 一般使用**标记清除方法(mark and sweep)**，当变量进入环境标记为进入环境，离开环境标记为离开环境
>
>   垃圾回收器会在运行的时候给存储在内存中的所有变量加上标记，然后去掉环境中的变量以及被环境中变量所引用的变量（闭包），在这些完成之后仍存在标记的就是要删除的变量了

- 还有**引用计数方法(reference counting)**，在低版本IE中经常会出现内存泄露，很多时候就是因为其采用引用计数方式进行垃圾回收。引用计数的策略是跟踪记录每个值被使用的次数，当声明了一个 变量并将一个引用类型赋值给该变量的时候这个值的引用次数就加1，如果该变量的值变成了另外一个，则这个值得引用次数减1，当这个值的引用次数变为0的时 候，说明没有变量在使用，这个值没法被访问了，因此可以将其占用的空间回收，这样垃圾回收器会在运行的时候清理掉引用次数为0的值占用的空间。
- 在IE中虽然JavaScript对象通过标记清除的方式进行垃圾回收，但BOM与DOM对象却是通过引用计数回收垃圾的， 也就是说只要涉及BOM及DOM就会出现循环引用问题。

16. JavaScript原型，原型链？有什么特点?

“

- 每个对象都会在其内部初始化一个属性，就是prototype(原型)，当我们访问一个对象的属性时，
  如果这个对象内部不存在这个属性，那么他就会去prototype里找这个属性，这个prototype又会有自己的prototype，
  于是就这样一直找下去，也就是我们平时所说的原型链的概念。
- 关系：instance.constructor.prototype = instance.**proto**
- 特点：
  JavaScript对象是通过引用来传递的，我们创建的每个新对象实体中并没有一份属于自己的原型副本。当我们修改原型时，与之相关的对象也会继承这一改变。

五、Vue 篇

1. 谈谈你对MVVM开发模式的理解?

> MVVM分为Model、View、ViewModel三者。
> Model 代表数据模型，数据和业务逻辑都在Model层中定义；
> View 代表UI视图，负责数据的展示；
> ViewModel 负责监听 Model 中数据的改变并且控制视图的更新，处理用户交互操作；
> Model 和 View 并无直接关联，而是通过 ViewModel 来进行联系的，Model 和 ViewModel 之间有着双向数据绑定的联系。因此当 Model 中的数据改变时会触发 View 层的刷新，View 中由于用户交互操作而改变的数据也会在 Model 中同步。
> 这种模式实现了 Model 和 View 的数据自动同步，因此开发者只需要专注对数据的维护操作即可，而不需要自己操作 dom。

2. v-if 和 v-show 有什么区别?

> - v-if 是真正的条件渲染，会控制这个 DOM 节点的存在与否。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；也是惰性的：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 就简单得多——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 的 “display” 属性进行切换。
> - 当我们需要经常切换某个元素的显示/隐藏时，使用v-show会更加节省性能上的开销；当只需要一次显示或隐藏时，使用v-if更加合理。

3. 你使用过 Vuex 吗?

> “
> Vuex 是一个专为 Vue.js 应用程序开发的状态管理模式。每一个 Vuex 应用的核心就是 store（仓库）。“store” 基本上就是一个容器，它包含着你的应用中大部分的状态 (state)。
>
> - （1）Vuex 的状态存储是响应式的。当 Vue 组件从 store 中读取状态的时候，若 store 中的状态发生变化，那么相应的组件也会相应地得到高效更新。
> - （2）改变 store 中的状态的唯一途径就是显式地提交 (commit) mutation。这样使得我们可以方便地跟踪每一个状态的变化。
>
> ---
>
> **主要包括以下几个模块：**
>
> - State => 基本数据，定义了应用状态的数据结构，可以在这里设置默认的初始状态。
> - Getter => 从基本数据派生的数据，允许组件从 Store 中获取数据，mapGetters 辅助函数仅仅是将 store 中的 getter 映射到局部计算属性。
> - Mutation => 是唯一更改 store 中状态的方法，且必须是同步函数。
> - Action => 像一个装饰器，包裹mutations，使之可以异步。用于提交 mutation，而不是直接变更状态，可以包含任意异步操作。
> - Module => 模块化Vuex，允许将单一的 Store 拆分为多个 store 且同时保存在单一的状态树中。

4. 说说你对 SPA 单页面的理解，它的优缺点分别是什么?

> SPA（ single-page application ）仅在 Web 页面初始化时加载相应的 HTML、JavaScript 和 CSS。一旦页> 面加载完成，SPA 不会因为用户的操作而进行页面的重新加载或跳转；取而代之的是利用路由机制实现 > HTML 内容的变换，UI 与用户的交互，避免页面的重新加载。

- 优点：

  用户体验好、快，内容的改变不需要重新加载整个页面，避免了不必要的跳转和重复渲染；

  基于上面一点，SPA 相对对服务器压力小；

  前后端职责分离，架构清晰，前端进行交互逻辑，后端负责数据处理；

- 缺点：

  初次加载耗时多：为实现单页 Web 应用功能及显示效果，需要在加载页面的时候将 JavaScript、CSS 统一> 加载，部分页面按需加载；

  前进后退路由管理：由于单页应用在一个页面中显示所有的内容，所以不能使用浏览器的前进后退功能，所> 有的页面切换需要自己建立堆栈管理；

  SEO 难度较大：由于所有的内容都在一个页面中动态替换显示，所以在 SEO 上其有着天然的弱势。

5. Class 与 Style 如何动态绑定?

> Class 可以通过对象语法和数组语法进行动态绑定：

- 对象语法：

```
<div v-bind:class="{ active: isActive, 'text-danger': hasError }">
</div>
data: {
  isActive: true,
    hasError: false
}
```

- 数组语法:

```
<div v-bind:class="[isActive ? activeClass : '', errorClass]">
</div>
data: {
  activeClass: 'active',
  errorClass: 'text-danger'
}
```

> “
> Style 也可以通过对象语法和数组语法进行动态绑定:

- 对象语法:

```
<div v-bind:style="{ color: activeColor, fontSize: fontSize + 'px'
}"></div>
data: {
  activeColor: 'red',
  fontSize: 30
}
```

- 数组语法：

```
<div v-bind:style="[styleColor, styleSize]"></div>
data: {
  styleColor: {
     color: 'red'
   },
  styleSize:{
     fontSize:'23px'
  }
}
```

6. 怎样理解 Vue 的单向数据流?

> 所有的 prop 都使得其父子 prop 之间形成了一个单向下行绑定：父级 prop 的更新会向下流动到子组件中，但是反过来则不行。
> 这样会防止从子组件意外改变父级组件的状态，从而导致你的应用的数据流向难以理解。
> 额外的，每次父级组件发生更新时，子组件中所有的 prop 都将会刷新为最新的值。
> 这意味着你不应该在一个子组件内部改变 prop。如果你这样做了，Vue 会在浏览器的控制台中发出警告。
> 子组件想修改时，只能通过 $emit 派发一个自定义事件，父组件接收到后，由父组件修改。

7. computed 和 watch 的区别和运用的场景?

> - computed： 是计算属性，依赖其它属性值，并且 computed 的值有缓存，只有它依赖的属性值发生改变，下一次获取 computed 的值时才会重新计算 computed 的值；
> - watch： 更多的是「观察」的作用，类似于某些数据的监听回调 ，每当监听的数据变化时都会执行回调进行后续操作；
> - 运用场景：
>   - 当我们需要进行数值计算，并且依赖于其它数据时，应该使用 computed，因为可以利用 computed 的缓存特性，避免每次获取值时，都要重新计算；
>   - 当我们需要在数据变化时执行异步或开销较大的操作时，应该使用 watch，使用 watch 选项允许我们执行异步操作 (访问一个 API)，限制我们执行该操作的频率，并在我们得到最终结果前，设置中间状态。这些都是计算属性无法做到的。

8. 直接给一个数组项赋值，Vue 能检测到变化吗?

> 由于 JavaScript 的限制，Vue 不能检测到以下数组的变动：
>
> - 当你利用索引直接设置一个数组项时，例如：`vm.items[indexOfItem] = newValue`
> - 当你修改数组的长度时，例如：`vm.items.length = newLength`

- 为了解决第一个问题，Vue 提供了以下操作方法：

```
// Vue.set
Vue.set(vm.items, indexOfItem, newValue)
// vm.$set，Vue.set的一个别名
vm.$set(vm.items, indexOfItem, newValue)
// Array.prototype.splice
vm.items.splice(indexOfItem, 1, newValue)
```

- 为了解决第二个问题，Vue 提供了以下操作方法：

```
// Array.prototype.splice
vm.items.splice(newLength)
```

9. 谈谈你对 Vue 生命周期的理解？

- **生命周期是什么？**

> Vue 实例有一个完整的生命周期，也就是从开始创建、初始化数据、编译模版、挂载 Dom -> 渲染、更新 -> 渲染、卸载等一系列过程，我们称这是 Vue 的生命周期。

- **各个生命周期的作用**

“

生命周期	描述
beforeCreate	组件实例被创建之初，组件的属性生效之前
created	组件实例已经完全创建，属性也绑定，但真实 dom 还没有生成，$el 还不可用
beforeMount	在挂载开始之前被调用：相关的 render 函数首次被调用
mounted	el 被新创建的 vm.$el 替换，并挂载到实例上去之后调用该钩子
beforeUpdate	组件数据更新之前调用，发生在虚拟 DOM 打补丁之前
updated	组件数据更新之后
activited	keep-alive 专属，组件被激活时调用
deadctivated	keep-alive 专属，组件被销毁时调用
beforeDestory	组件销毁前调用
destoryed	组件销毁后调用

10. Vue 的父组件和子组件生命周期钩子函数执行顺序?

“

Vue 的父组件和子组件生命周期钩子函数执行顺序可以归类为以下 4 部分：

- 加载渲染过程：

  父 beforeCreate -> 父 created -> 父 beforeMount -> 子 beforeCreate -> 子 created -> 子 beforeMount -> 子 mounted -> 父 mounted

- 子组件更新过程：

  父 beforeUpdate -> 子 beforeUpdate -> 子 updated -> 父 updated

- 父组件更新过程：

父 beforeUpdate -> 父 updated

- 销毁过程：

  父 beforeDestroy -> 子 beforeDestroy -> 子 destroyed -> 父 destroyed

11. 父组件可以监听到子组件的生命周期吗?

> 比如有父组件 Parent 和子组件 Child，如果父组件监听到子组件挂载 mounted 就做一些逻辑处理，可以通过以下写法实现：

```
// Parent.vue
<Child @mounted="doSomething"/>

// Child.vue
mounted() {
  this.$emit("mounted");
}
```

> 以上需要手动通过 $emit 触发父组件的事件，更简单的方式可以在父组件引用子组件时通过 @hook 来监听即可，如下所示：

```
//  Parent.vue
<Child @hook:mounted="doSomething" ></Child>

doSomething() {
   console.log('父组件监听到 mounted 钩子函数 ... ');
},
```

```
//  Child.vue
mounted(){
   console.log('子组件触发 mounted 钩子函数 ...');
},

// 以上输出顺序为:
// 子组件触发 mounted 钩子函数 ...
// 父组件监听到 mounted 钩子函数 ...
```

> 当然 @hook 方法不仅仅是可以监听 mounted，其它的生命周期事件，例如：created，updated 等都可以监听。

12. 谈谈你对 keep-alive 的了解?

> keep-alive 是 Vue 内置的一个组件，可以使被包含的组件保留状态，避免重新渲染 ，其有以下特性：
>
> - 一般结合路由和动态组件一起使用，用于缓存组件；
> - 提供 include 和 exclude 属性，两者都支持字符串或正则表达式， include 表示只有名称匹配的组件会被缓存，exclude 表示任何名称匹配的组件都不会被缓存 ，其中 exclude 的优先级比 include 高；
> - 对应两个钩子函数 activated 和 deactivated ，当组件被激活时，触发钩子函数 activated，当组件被移除时，触发钩子函数 deactivated。

13. 组件中 data 为什么是一个函数?

- **为什么组件中的 data 必须是一个函数，然后 return 一个对象，而 new Vue 实例里，data 可以直接是一个对象?**

> “
>
> - 因为组件是用来复用的，且 JS 里对象是引用关系，如果组件中 data 是一个对象，那么这样作用域没有隔离，子组件中的 data 属性值会相互影响，
> - 如果组件中 data 选项是一个函数，那么每个实例可以维护一份被返回对象的独立的拷贝，组件实例之间的 data 属性值不会互相影响；而 new Vue 的实例，是不会被复用的，因此不存在引用对象的问题。

14. v-model 的原理?

> “
>
> 我们在 vue 项目中主要使用 v-model 指令在表单 input、textarea、select 等元素上创建双向数据绑定，我们知道 v-model 本质上不过是语法糖，v-model 在内部为不同的输入元素使用不同的属性并抛出不同的事件：
>
> - text 和 textarea 元素使用 value 属性和 input 事件；
> - checkbox 和 radio 使用 checked 属性和 change 事件；
> - select 字段将 value 作为 prop 并将 change 作为事件。

- 以 input 表单元素为例：

```
<input v-model='something'>
```

相当于

```
<input v-bind:value="something" v-on:input="something =
$event.target.value">
```

如果在自定义组件中，v-model 默认会利用名为 value 的 prop 和名为 input 的事件，如下所示：

```
父组件：
<ModelChild v-model="message"></ModelChild>

子组件：
<div>{{value}}</div>

props:{
    value: String
},
methods: {
  test1(){
     this.$emit('input', '小红')
  },
},
```

• 15. Vue 组件间通信有哪几种方式?

Vue 组件间通信是面试常考的知识点之一，这题有点类似于开放题，你回答出越多方法当然越加分，表明你对 Vue 掌握的越熟练。

> Vue 组件间通信只要指以下 3 类通信：父子组件通信、隔代组件通信、兄弟组件通信，下面我们分别介绍每种通信方式且会说明此种方法可适用于哪类组件间通信。

(1) `props / $emit` 适用 父子组件通信

- 这种方法是 Vue 组件的基础，相信大部分同学耳闻能详，所以此处就不举例展开介绍。

(2) `ref` 与 `$parent / $children` 适用 父子组件通信

- `ref`：如果在普通的 DOM 元素上使用，引用指向的就是 DOM 元素；如果用在子组件上，引用就指向组件实例
- `$parent / $children`：访问父 / 子实例

(3) `EventBus （$emit / $on）` 适用于 父子、隔代、兄弟组件通信

- 这种方法通过一个空的 Vue 实例作为中央事件总线（事件中心），用它来触发事件和监听事件，从而实现任何组件间的通信，包括父子、隔代、兄弟组件。

(4) `$attrs/$listeners` 适用于 隔代组件通信

- `$attrs`：包含了父作用域中不被 prop 所识别 (且获取) 的特性绑定 (class 和 style 除外)。当一个组件没有声明任何 prop 时，这里会包含所有父作用域的绑定 (class 和 style 除外)，并且可以通过 `v-bind="$attrs"` 传入内部组件。通常配合 inheritAttrs 选项一起使用。
- `$listeners`：包含了父作用域中的 (不含 .native 修饰器的) v-on 事件监听器。它可以通过 `v-on="$listeners"` 传入内部组件

(5) `provide / inject` 适用于 隔代组件通信

- 祖先组件中通过 provider 来提供变量，然后在子孙组件中通过 inject 来注

入变量。`provide / inject API`主要解决了跨级组件间的通信问题，不过它的使用场景，主要是子组件获取上级组件的状态，跨级组件间建立了一种主动提供与依赖注入的关系。

(6) `Vuex`适用于 父子、隔代、兄弟组件通信

- Vuex 是一个专为 Vue.js 应用程序开发的状态管理模式。每一个 Vuex 应用的核心就是 store（仓库）。“store” 基本上就是一个容器，它包含着你的应用中大部分的状态 (state)。
- Vuex 的状态存储是响应式的。当 Vue 组件从 store 中读取状态的时候，若 store 中的状态发生变化，那么相应的组件也会相应地得到高效更新。
- 改变 store 中的状态的唯一途径就是显式地提交 (commit) mutation。这样使得我们可以方便地跟踪每一个状态的变化。

16. 使用过 Vue SSR 吗？说说 SSR?

“

- Vue.js 是构建客户端应用程序的框架。默认情况下，可以在浏览器中输出 Vue 组件，进行生成 DOM 和操作 DOM。然而，也可以将同一个组件渲染为服务端的 HTML 字符串，将它们直接发送到浏览器，最后将这些静态标记"激活"为客户端上完全可交互的应用程序。
- 即：SSR大致的意思就是vue在客户端将标签渲染成的整个 html 片段的工作在服务端完成，服务端形成的html 片段直接返回给客户端这个过程就叫做服务端渲染。

服务端渲染 SSR 的优缺点如下：

- （1）服务端渲染的优点：
  - 更好的 SEO：因为 SPA 页面的内容是通过 Ajax 获取，而搜索引擎爬取工具并不会等待 Ajax 异步完成后再抓取页面内容，所以在 SPA 中是抓取不到页面通过 Ajax 获取到的内容；而 SSR 是直接由服务端返回已经渲染好的页面（数据已经包含在页面中），所以搜索引擎爬取工具可以抓取渲染好的页面；
  - 更快的内容到达时间（首屏加载更快）：SPA 会等待所有 Vue 编译后的 js 文件都下载完成后，才开始进行页面的渲染，文件下载等需要一定的时间等，所以首屏渲染需要一定的时间；SSR 直接由服务端渲染好页面直接返回显示，无需等待下载 js 文件及再去渲染等，所以 SSR 有更快的内容到达时间；
- (2) 服务端渲染的缺点：
  - 更多的开发条件限制：例如服务端渲染只支持 beforCreate 和 created 两个钩子函数，这会导致一些外部扩展库需要特殊处理，才能在服务端渲染应用程序中运行；并且与可以部署在任何静态文件服务器上的完全静态单页面应用程序 SPA 不同，服务端渲染应用程序，需要处于 Node.js server 运行环境；
  - 更多的服务器负载：在 Node.js 中渲染完整的应用程序，显然会比仅仅提供静态文件的 server 更加大量占用CPU 资源 (CPU-intensive - CPU 密集)，因此如果你预料在高流量环境 (high traffic) 下使用，请准备相应的服务器负载，并明智地采用缓存策略。

17. vue-router 路由模式有几种?

> vue-router 有 3 种路由模式：hash、history、abstract，对应的源码如下所示：

```
switch (mode) {
  case 'history':
    this.history = new HTML5History(this, options.base)
    break
  case 'hash':
    this.history = new HashHistory(this, options.base,
this.fallback)
    break
  case 'abstract':
    this.history = new AbstractHistory(this, options.base)
    break
  default:
    if (process.env.NODE_ENV !== 'production') {
      assert(false, `invalid mode: ${mode}`)
    }
}
```

> 其中，3 种路由模式的说明如下：
>
> - hash: 使用 URL hash 值来作路由。支持所有浏览器，包括不支持 HTML5 History Api 的浏览器；
> - history : 依赖 HTML5 History API 和服务器配置。具体可以查看 HTML5 History 模式；
> - abstract : 支持所有 JavaScript 运行环境，如 Node.js 服务器端。如果发现

> 没有浏览器的 API，路由会自动强制进入这个模式.

• 18. 能说下 vue-router 中常用的 hash 和 history 路由模式实现原理吗?

> “
>
> （1）hash 模式的实现原理
> 早期的前端路由的实现就是基于 location.hash 来实现的。其实现原理很简单，location.hash 的值就是 URL 中 # 后面的内容。比如下面这个网站，它的 location.hash 的值为 '#search'：
> `https://www.word.com#search`
> hash 路由模式的实现主要是基于下面几个特性：
>
> - URL 中 hash 值只是客户端的一种状态，也就是说当向服务器端发出请求时，hash 部分不会被发送；
> - hash 值的改变，都会在浏览器的访问历史中增加一个记录。因此我们能通过浏览器的回退、前进按钮控制hash 的切换；
> - 可以通过 a 标签，并设置 href 属性，当用户点击这个标签后，URL 的 hash 值会发生改变；或者使用 JavaScript 来对 loaction.hash 进行赋值，改变 URL 的 hash 值；
> - 我们可以使用 hashchange 事件来监听 hash 值的变化，从而对页面进行跳转（渲染）。
>
> ---
>
> （2）history 模式的实现原理
> HTML5 提供了 History API 来实现 URL 的变化。其中做最主要的 API 有以下两个：history.pushState() 和 history.repalceState()。这两个 API 可以在不进行刷新的情况下，操作浏览器的历史纪录。

唯一不同的是，前者是新增一个历史记录，后者是直接替换当前的历史记录，如下所示：

```
window.history.pushState(null, null, path);
window.history.replaceState(null, null, path);
```

history 路由模式的实现主要基于存在下面几个特性：

- pushState 和 repalceState 两个 API 来操作实现 URL 的变化；
- 我们可以使用 popstate 事件来监听 url 的变化，从而对页面进行跳转（渲染）；
- history.pushState() 或 history.replaceState() 不会触发 popstate 事件，这时我们需要手动触发页面跳转（渲染）。

19. Vue 框架怎么实现对象和数组的监听?

“

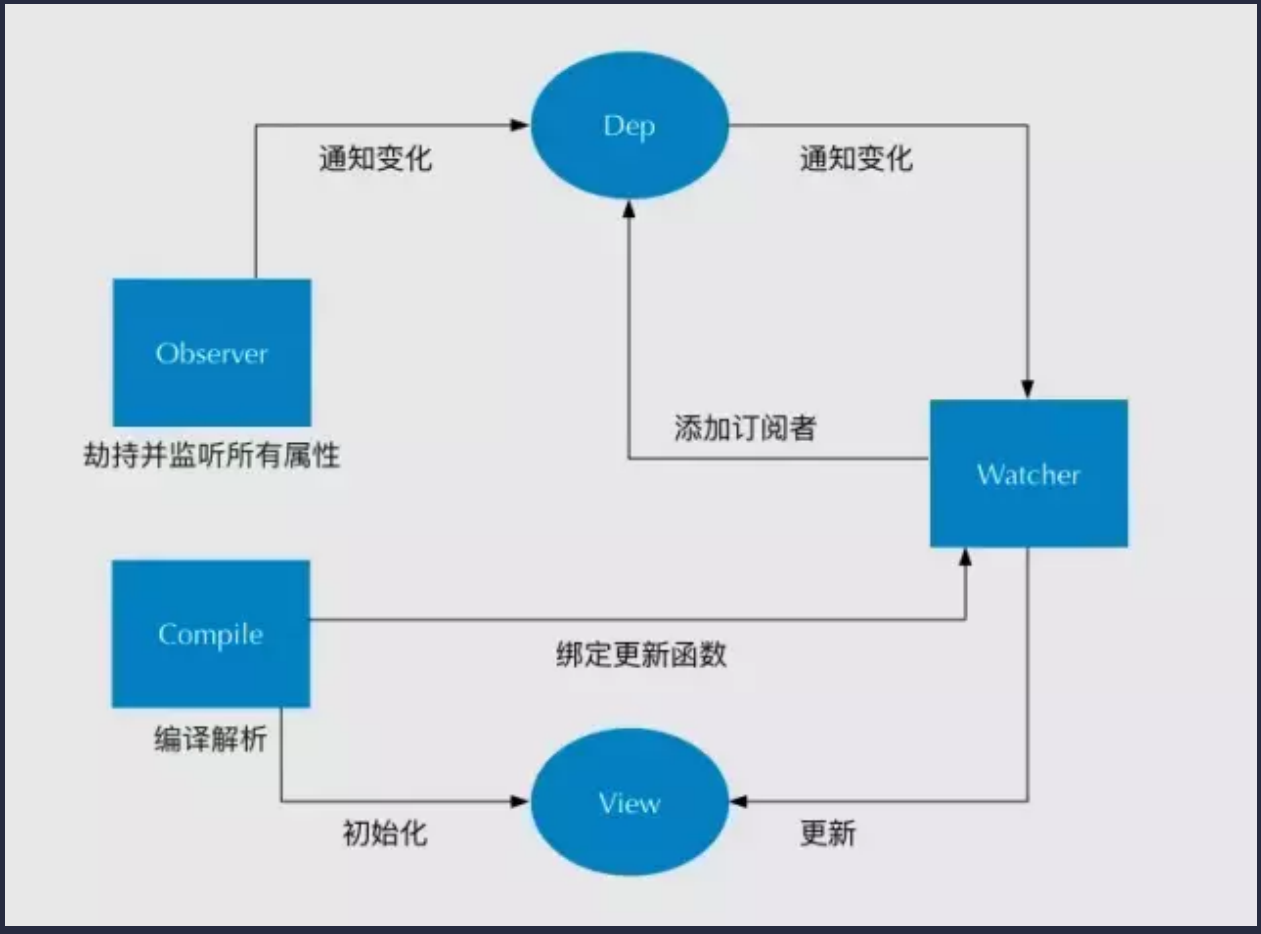
Vue 数据双向绑定主要是指：数据变化更新视图，视图变化更新数据。
即：

- 输入框内容变化时，Data 中的数据同步变化。即 View => Data 的变化。
- Data 中的数据变化时，文本节点的内容同步变化。即 Data => View 的变化。

其中，**View 变化更新 Data ，可以通过事件监听的方式来实现**，所以 Vue 的数据双向绑定的工作主要是**如何根据 Data 变化更新 View**。
Vue 主要通过以下 4 个步骤来实现数据双向绑定的：

- 实现一个监听器 Observer：对数据对象进行遍历，包括子属性对象的属性，利用 Object.defineProperty() 对属性都加上 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。

- 实现一个解析器 Compile：解析 Vue 模板指令，将模板中的变量都替换成数据，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，调用更新函数进行数据更新。
- 实现一个订阅者 Watcher：Watcher 订阅者是 Observer 和 Compile 之间通信的桥梁 ，主要的任务是订阅 Observer 中的属性值变化的消息，当收到属性值变化的消息时，触发解析器 Compile 中对应的更新函数。
- 实现一个订阅器 Dep：订阅器采用 发布-订阅 设计模式，用来收集订阅者 Watcher，对监听器 Observer 和 订阅者 Watcher 进行统一管理。



20. Vue 是如何实现数据双向绑定的?

> 如果被问到 Vue 怎么实现数据双向绑定，大家肯定都会回答 通过 Object.defineProperty() 对数据进行劫持，但是 Object.defineProperty() 只能对属性进行数据劫持，不能对整个对象进行劫持。
> 同理无法对数组进行劫持，但是我们在使用 Vue 框架中都知道，Vue 能检测到对象和数组（部分方法的操作）的变化，那它是怎么实现的呢？我们查看相关代码如下：

```
  /**
   * Observe a list of Array items.
   */
  observeArray (items: Array<any>) {
    for (let i = 0, l = items.length; i < l; i++) {
      observe(items[i])  // observe 功能为监测数据的变化
    }
  }

  /**
   * 对属性进行递归遍历
   */
  let childOb = !shallow && observe(val) // observe 功能为监测数据的
变化
```

> 通过以上 Vue 源码部分查看，我们就能知道 Vue 框架是通过遍历数组 和递归遍历对象，从而达到利用 Object.defineProperty() 也能对对象和数组（部分方法的操作）进行监听。

21. Vue 怎么用 vm.$set() 解决对象新增属性不能响应的问题？

> 受现代 JavaScript 的限制，Vue 无法检测到对象属性的添加或删除。
> 由于 Vue 会在初始化实例时对属性执行 getter/setter 转化，所以属性必须在 data 对象上存在才能让 Vue 将它转换为响应式的。
> 但是 Vue 提供了 `Vue.set（object，propertyName，value）/ vm.$set（object，propertyName，value）` 来实现为对象添加响应式属性，那框架本身是如何实现的呢?
>
> - 我们查看对应的 Vue 源码：`vue/src/core/instance/index.js`

```
export function set (target: Array<any> | Object, key: any, val: any): any {
  // target 为数组
  if (Array.isArray(target) && isValidArrayIndex(key)) {
    // 修改数组的长度，避免索引>数组长度导致splcie()执行有误
    target.length = Math.max(target.length, key)
    // 利用数组的splice变异方法触发响应式
    target.splice(key, 1, val)
    return val
  }
  // key 已经存在，直接修改属性值
  if (key in target && !(key in Object.prototype)) {
    target[key] = val
    return val
  }
  const ob = (target: any).__ob__
  // target 本身就不是响应式数据，直接赋值
  if (!ob) {
    target[key] = val
    return val
```

```
  }
  // 对属性进行响应式处理
  defineReactive(ob.value, key, val)
  ob.dep.notify()
  return val
}
```

> 我们阅读以上源码可知，vm.$set 的实现原理是：
>
> - 如果目标是数组，直接使用数组的 splice 方法触发相应式；
> - 如果目标是对象，会先判读属性是否存在、对象是否是响应式，最终如果要对属性进行响应式处理，则是通过调用 defineReactive 方法进行响应式处理（ defineReactive 方法就是 Vue 在初始化对象时，给对象属性采用 Object.defineProperty 动态添加 getter 和 setter 的功能所调用的方法）

22. 虚拟 DOM 的优缺点?

> 优点：
>
> - 保证性能下限： 框架的虚拟 DOM 需要适配任何上层 API 可能产生的操作，它的一些 DOM 操作的实现必须是普适的，所以它的性能并不是最优的；但是比起粗暴的 DOM 操作性能要好很多，因此框架的虚拟 DOM 至少可以保证在你不需要手动优化的情况下，依然可以提供还不错的性能，即保证性能的下限；
> - 无需手动操作 DOM： 我们不再需要手动去操作 DOM，只需要写好 View-Model 的代码逻辑，框架会根据虚拟 DOM 和 数据双向绑定，帮我们以可预期的方式更新视图，极大提高我们的开发效率；

- 跨平台： 虚拟 DOM 本质上是 JavaScript 对象,而 DOM 与平台强相关，相比之下虚拟 DOM 可以进行更方便地跨平台操作，例如服务器渲染、weex 开发等等。

缺点:

- 无法进行极致优化： 虽然虚拟 DOM + 合理的优化，足以应对绝大部分应用的性能需求，但在一些性能要求极高的应用中虚拟 DOM 无法进行针对性的极致优化。

23. 虚拟 DOM 实现原理?

> 虚拟 DOM 的实现原理主要包括以下 3 部分：
>
> - 用 JavaScript 对象模拟真实 DOM 树，对真实 DOM 进行抽象；
> - diff 算法 — 比较两棵虚拟 DOM 树的差异；
> - pach 算法 — 将两个虚拟 DOM 对象的差异应用到真正的 DOM 树。

24. Vue 中的 key 有什么作用?

> **key 是为 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。**
> Vue 的 diff 过程可以概括为：oldCh 和 newCh 各有两个头尾的变量 oldStartIndex、oldEndIndex 和 newStartIndex、newEndIndex，它们会新节点和旧节点会进行两两对比，即一共有4种比较方式：newStartIndex 和 oldStartIndex 、newEndIndex 和 oldEndIndex 、newStartIndex 和 oldEndIndex 、newEndIndex 和 oldStartIndex，如果以上 4 种比较都没匹配，如果设置了key，就会用 key 再进行比较，在比较的过程中，遍历会往中

> 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。
> **所以 Vue 中 key 的作用是：key 是为 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!**
>
> - 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，源码如下：

```
function createKeyToOldIdx (children, beginIdx, endIdx) {
  let i, key
  const map = {}
  for (i = beginIdx; i <= endIdx; ++i) {
    key = children[i].key
    if (isDef(key)) map[key] = i
  }
  return map
}
```

25. 你有对 Vue 项目进行哪些优化?

> “
> （1）代码层面的优化
>
> - v-if 和 v-show 区分使用场景
> - computed 和 watch 区分使用场景
> - v-for 遍历必须为 item 添加 key，且避免同时使用 v-if
> - 长列表性能优化
> - 事件的销毁

- 图片资源懒加载
- 路由懒加载
- 第三方插件的按需引入
- 优化无限列表性能
- 服务端渲染 SSR or 预渲染

（2）Webpack 层面的优化

- Webpack 对图片进行压缩
- 减少 ES6 转为 ES5 的冗余代码
- 提取公共代码
- 模板预编译
- 提取组件的 CSS
- 优化 SourceMap
- 构建结果输出分析
- Vue 项目的编译优化

（3）基础的 Web 技术的优化

- 开启 gzip 压缩
- 浏览器缓存
- CDN 的使用
- 使用 Chrome Performance 查找性能瓶颈

26. 对于 vue3.0 特性你有什么了解的吗?

> Vue 3.0 的目标是让 Vue 核心变得更小、更快、更强大，因此 Vue 3.0 增加以下这些新特性：
>
> > （1）监测机制的改变
> > 3.0 将带来基于代理 Proxy 的 observer 实现，提供全语言覆盖的反应性跟踪。这消除了 Vue 2 当中基于 Object.defineProperty 的实现所存在的很多限制：
> >
> > - 只能监测属性，不能监测对象
> > - 检测属性的添加和删除；
> > - 检测数组索引和长度的变更；
> > - 支持 Map、Set、WeakMap 和 WeakSet。
> >
> > 新的 observer 还提供了以下特性：
> >
> > - 用于创建 observable 的公开 API。这为中小规模场景提供了简单轻量级的跨组件状态管理解决方案。
> > - 默认采用惰性观察。在 2.x 中，不管反应式数据有多大，都会在启动时被观察到。如果你的数据集很大，这可能会在应用启动时带来明显的开销。在 3.x 中，只观察用于渲染应用程序最初可见部分的数据。
> > - 更精确的变更通知。在 2.x 中，通过 Vue.set 强制添加新属性将导致依赖于该对象的 watcher 收到变更通知。在 3.x 中，只有依赖于特定属性的 watcher 才会收到通知。
> > - 不可变的 observable：我们可以创建值的“不可变”版本（即使是嵌套属性），除非系统在内部暂时将其“解禁”。这个机制可用于冻结 prop 传递或 Vuex 状态树以外的变化。

- 更好的调试功能：我们可以使用新的 renderTracked 和 renderTriggered 钩子精确地跟踪组件在什么时候以及为什么重新渲染。

> **（2）模板**
> 模板方面没有大的变更，只改了作用域插槽，2.x 的机制导致作用域插槽变了，父组件会重新渲染，而 3.0 把作用域插槽改成了函数的方式，这样只会影响子组件的重新渲染，提升了渲染的性能。
> 同时，对于 render 函数的方面，vue3.0 也会进行一系列更改来方便习惯直接使用 api 来生成 vdom 。

> **（3）对象式的组件声明方式**
> vue2.x 中的组件是通过声明的方式传入一系列 option，和 TypeScript 的结合需要通过一些装饰器的方式来做，虽然能实现功能，但是比较麻烦。
> 3.0 修改了组件的声明方式，改成了类式的写法，这样使得和 TypeScript 的结合变得很容易。
> 此外，vue 的源码也改用了 TypeScript 来写。其实当代码的功能复杂之后，必须有一个静态类型系统来做一些辅助管理。
> 现在 vue3.0 也全面改用 TypeScript 来重写了，更是使得对外暴露的 api 更容易结合 TypeScript。静态类型系统对于复杂代码的维护确实很有必要。

> **（4）其它方面的更改**
> vue3.0 的改变是全面的，上面只涉及到主要的 3 个方面，还有一些其他的更改：
>
> - 支持自定义渲染器，从而使得 weex 可以通过自定义渲染器的方式来扩展，而不是直接 fork 源码来改的方式。
> - 支持 Fragment（多个根节点）和 Protal（在 dom 其他部分渲染组建内

> 容）组件，针对一些特殊的场景做了处理。
> - 基于 treeshaking 优化，提供了更多的内置功能。

六、其他杂项篇

1. 常见的浏览器内核有哪些?

> “
> - 主要分成两部分：渲染引擎(layout engineer或Rendering Engine)和JS引擎。
>   - 渲染引擎：负责取得网页的内容（HTML、XML、图像等等）、整理讯息（例如加入CSS等），以及计算网页的显示方式，然后会输出至显示器或打印机。浏览器的内核的不同对于网页的语法解释会有不同，所以渲染的效果也不相同。所有网页浏览器、电子邮件客户端以及其它需要编辑、显示网络内容的应用程序都需要内核。
>   - JS引擎则：解析和执行javascript来实现网页的动态效果。
> - 最开始渲染引擎和JS引擎并没有区分的很明确，后来JS引擎越来越独立，内核就倾向于只指渲染引擎。
> - 常见内核
>   - Trident 内核：IE, MaxThon, TT, The World, 360, 搜狗浏览器等。[又称MSHTML]

- Gecko 内核：Netscape6 及以上版本，FF, MozillaSuite / SeaMonkey 等
- Presto 内核：Opera7 及以上。 [Opera内核原为：Presto，现为：Blink;]
- Webkit 内核：Safari, Chrome等。 [Chrome的：Blink（WebKit 的分支）]

2. 网页前端性能优化的方式有哪些?

> 1.压缩 css, js, 图片
> 2.减少 http 请求次数， 合并 css、js 、合并图片（雪碧图）
> 3.使用 CDN
> 4.减少 dom 元素数量
> 5.图片懒加载
> 6.静态资源另外用无 cookie 的域名
> 7.减少 dom 的访问（缓存 dom）
> 8.巧用事件委托
> 9.样式表置顶、脚本置低

3. 网页从输入网址到渲染完成经历了哪些过程?

> 大致可以分为如下7步：
>
> - 输入网址；
> - 发送到DNS服务器，并获取域名对应的web服务器对应的ip地址；
> - 与web服务器建立TCP连接；
> - 浏览器向web服务器发送http请求；

- web服务器响应请求，并返回指定url的数据（或错误信息，或重定向的新的url地址）；
- 浏览器下载web服务器返回的数据及解析html源文件；
- 生成DOM树，解析css和js，渲染页面，直至显示完成；

4. 线程与进程的区别?

“

- 一个程序至少有一个进程,一个进程至少有一个线程.
- 线程的划分尺度小于进程，使得多线程程序的并发性高。
- 另外，进程在执行过程中拥有独立的内存单元，而多个线程共享内存，从而极大地提高了程序的运行效率。
- 线程在执行过程中与进程还是有区别的。每个独立的线程有一个程序运行的入口、顺序执行序列和程序的出口。但是线程不能够独立执行，必须依存在应用程序中，由应用程序提供多个线程执行控制。
- 从逻辑角度来看，多线程的意义在于一个应用程序中，有多个执行部分可以同时执行。但操作系统并没有将多个线程看做多个独立的应用，来实现进程的调度和管理以及资源分配。这就是进程和线程的重要区别。

5. HTTP常见的状态码?

“

100 Continue 继续，一般在发送post请求时，已发送了http header之后服务端将返回此信息，表示确认，之后发送具体参数信息
200 OK 正常返回信息
201 Created 请求成功并且服务器创建了新的资源
202 Accepted 服务器已接受请求，但尚未处理

301 Moved Permanently 请求的网页已永久移动到新位置。
302 Found 临时性重定向。
303 See Other 临时性重定向，且总是使用 GET 请求新的 URI。
304 Not Modified 自从上次请求后，请求的网页未修改过。
400 Bad Request 服务器无法理解请求的格式，客户端不应当尝试再次使用相同的内容发起请求。
401 Unauthorized 请求未授权。
403 Forbidden 禁止访问。
404 Not Found 找不到如何与 URI 相匹配的资源。
500 Internal Server Error 最常见的服务器端错误。
503 Service Unavailable 服务器端暂时无法处理请求（可能是过载或维护）。

6. 图片懒加载?

> 当页面滚动的时间被触发 -> 执行加载图片操作 -> 判断图片是否在可视区域内 -> 在，则动态将data-src的值赋予该图片

7. 移动端性能优化?

> - 尽量使用css3动画，开启硬件加速
> - 适当使用touch时间代替click时间
> - 避免使用css3渐变阴影效果
> - 可以用transform: translateZ(0) 来开启硬件加速
> - 不滥用float。float在渲染时计算量比较大，尽量减少使用
> - 不滥用web字体。web字体需要下载，解析，重绘当前页面
> - 合理使用requestAnimationFrame动画代替setTimeout
> - css中的属性（css3 transitions、css3 3D transforms、opacity、webGL、

video）会触发GUP渲染，耗电

8. TCP 传输的三次握手、四次挥手策略

“

- 三次握手：

  为了准确无误地吧数据送达目标处，TCP协议采用了三次握手策略。用TCP协议把数据包送出去后，TCP不会对传送后的情况置之不理，他一定会向对方确认是否送达，握手过程中使用TCP的标志：SYN和ACK

  - 发送端首先发送一个带SYN的标志的数据包给对方
  - 接收端收到后，回传一个带有SYN/ACK标志的数据包以示传达确认信息
  - 最后，发送端再回传一个带ACK的标志的数据包，代表“握手”结束
- 如在握手过程中某个阶段莫明中断，TCP协议会再次以相同的顺序发送相同的数据包

- 断开一个TCP连接需要“四次挥手”
  - 第一次挥手：主动关闭方发送一个FIN，用来关注主动方到被动关闭方的数据传送，也即是主动关闭方告诫被动关闭方：我已经不会再给你发数据了（在FIN包之前发送的数据，如果没有收到对应的ACK确认报文，主动关闭方依然会重发这些数据）。但是，此时主动关闭方还可以接受数据
  - 第二次挥手：被动关闭方收到FIN包后，发送一个ACK给对方，确认序

号收到序号 +1（与SYN相同，一个 FIN占用一个序号）

- 第三次挥手：被动关闭方发送一个 FIN。用来关闭被动关闭方到主动关闭方的数据传送，也就是告诉主动关闭方，我的数据也发送完了，不会给你发送数据了
- 第四次挥手：主动关闭方收到FIN后，发送一个ACK给被动关闭方，确认序号为收到序号+1，至此，完成四次挥手

9. HTTP 和 HTTPS，为什么HTTPS安全?

“

- HTTP协议通常承载与 TCP协议之上，在HTTP和TCP之间添加一个安全协议层（SSL或TSL），这个时候，就成了我们常说的HTTPS
- 默认HTTP的端口号为80，HTTPS的端口号为443
- 因为网络请求需要中间有很多的服务器路由的转发，中间的节点都可能篡改信息，而如果使用HTTPS，密钥在你和终点站才有，https之所有说比http安全，是因为他利用ssl/tls协议传输。包含证书，流量转发，负载均衡，页面适配，浏览器适配，refer传递等，保障了传输过程的安全性

七、主观题篇

- **1. 你都做过什么项目呢？具体聊某一个项目中运用的技术.**

> “
> 注意：用心找自己做的项目中自己感觉最拿出来手的（复杂度最高，用的技术最多的项目），描述的时候尽可能往里面添加一些技术名词
> 布局我们用html5+css3
> 我们会用reset.css重置浏览器的默认样式
> JS框架的话我们选用的是jQuery(也可能是Zepto)
> 我们用版本控制工具git来协同开发
> 我们会基于gulp搭建的前端自动化工程来开发（里面包含有我们的项目结构、我们需要引用的第三方库等一些信息，我们还实现了sass编译、CSS3加前缀等的自动化）
> 我们的项目中还用到了表单验证validate插件、图片懒加载Lazyload插件

- **2. 你遇到过比较难的技术问题是？你是如何解决的？**

- **3. 常使用的库有哪些？常用的前端开发工具？开发过什么应用或组件？**

- **4. 除了前端以外还了解什么其它技术么？你最最厉害的技能是什么？**

- **5. 对前端开发工程师这个职位是怎么样理解的？它的前景会怎么样？**

> “
> 前端是最贴近用户的程序员，比后端、数据库、产品经理、运营、安全都近。
> 1、实现界面交互
> 2、提升用户体验
> 3、有了Node.js，前端可以实现服务端的一些事情

> 前端是最贴近用户的程序员，前端的能力就是能让产品从 90分进化到 100分，甚至更好，
> 参与项目，快速高质量完成实现效果图，精确到1px；
> 与团队成员，UI设计，产品经理的沟通；
> 做好的页面结构，页面重构和用户体验；
> 处理hack，兼容、写出优美的代码格式；
> 针对服务器的优化、拥抱最新前端技术。

- **6. 你的优点是什么？缺点是什么？**

觉得有帮助的小伙伴点个赞~