

# 前端基础

## HTML(5)

### 1. Html5 新增那些标签?

布局标签:

header, section, footer, article, aside

表单标签: datalist,

input:type=' week|date|time|datetime|number|search|url|tel|color|email|range'

多媒体标签: audio, video

其他标签: progress(进度条), meter

### 2. 行内元素和块级元素的具体区别是什么?

块级元素独占一行页面空间, 不会和其他元素共享一行页面空间;

行内元素可以和其他非块级元素(行内, 行内块)共享一行页面空间.

### 3. 列举几个块级标签和行内标签?

块级标签: div, p, h1~h6, section, header, footer

行内标签:span, em(i), strong(b), u, em(i), a

#### 4. 行内元素的 padding 和 margin 可设置吗?

行内元素设置水平方向的 padding 和 margin 是可以生效,但是设置垂直方向的 padding 和 margin 虽然看起来对标签起作用,但实际并没有对周围元素产生任何影响,所以行内元素设置垂直方向的 padding 和 margin 是无效的.

#### 5. 简述 readonly 与 disabled 的区别

readonly 是设置表单元素为只读状态;

disabled 是设置表单元素为禁用状态.

#### 6. 哪些标签都存在伪元素?

大部分容器标签(大部分双标签)都有伪元素,iframe 没有伪元素;

大部分单标签都没有伪元素,但是 img 有伪元素

#### 7. 伪元素可以使用 js 来操作吗?

js 不可以操作伪元素

#### 8. Html5 的网页为什么只需要写<!DOCTYPE HTML>?

HTML 4.01 中的 doctype 需要对 DTD 进行引用,因为 HTML 4.01 基于 SGML。而 HTML 5 不基于 SGML,因此不需要对 DTD 进行引用,

但是需要 doctype 来规范浏览器的行为。其中，SGML 是标准通用标记语言，简单的说，就是比 HTML, XML 更老的标准，这两者都是由 SGML 发展而来的，而 HTML5 不是的。

## CSS(3)

9. px em rem 这三中长度单位的区别？

px 是一个绝对单位；em 和 rem 是一个相对单位，em 参考的是当前元素的字体 (font-size) 大小，参考的是页面根元素 html 的字体 (font-size) 大小。

10. CSS3 新增伪类有那些？

p:first-of-type 选择属于其父元素的首个<p>元素。

p:last-of-type 选择属于其父元素的最后一个<p>元素。

p:nth-child(2) 选择属于其父元素的第二个子元素。

p:nth-type-of(2) 选择属于其父元素的第二个子元素 p。

:enabled、:disabled 控制表单控件的禁用状态。

:checked, 单选框或复选框被选中。

11. 谈谈 css 选择器优先级顺序以及判定标准？

优先级从低到高：

通配符选择器<标签选择器<类选择器(属性选择器)<ID 选择器；

行内样式<使用!important 修饰的属性优先级最高；

如果两个选择器(属性完全相同)同时命中一个元素, 并且权重一样, 则书写顺序会影响优先级, 后一个选择器的属性会覆盖前一个选择器中相同的属性.

## 12. position 几个属性的作用?

position 的常见四个属性值: relative, absolute, fixed, static. 一般都要配合"left"、"top"、"right"以及"bottom" 属性使用。

1) static: 默认位置, (static 元素会忽略任何 top、 bottom、 left 或 right 声明)。一般不常用。

2) relative: 位置被设置为 relative 的元素, 偏移的 top, right, bottom, left 的值都以它原来的位置为基准偏移。注意 relative 移动后的元素在原来的位置仍占据空间。

3) absolute: 位置设置为 absolute 的元素, 可定位于相对于包含它的元素的指定坐标。意思就是如果它的父容器设置了 position 属性, 并且 position 的属性值为 absolute 或者 relative, 那么就会依据父容器进行偏移。如果其父容器没有设置 position 属性, 那么偏移是以 body 为依据。注意设置 absolute 属性的元素在标准流中不占位置。

4) fixed: 位置被设置为 fixed 的元素, 可定位于相对于浏览器窗口的指定坐标。不论窗口滚动与否, 元素都会留在那个位置。它始终是以 body 为依据的。注意设置 fixed 属性的元素在标准流中不占位置。

13. position 设置为 absolute 和 fixed 有什么区别?

absolute 是绝对定位, 绝对定位参考的是有明确定位的父元素, 如果直接父元素没有明确定位会一直向上查找, 如果父元素都没有明确定位, 则参考 body 标签;

fixed 是固定定位, 参考对象是浏览器.

14. 在一个页面中给多个元素设置相同的 id, 会导致什么问题?

会导致通过 js 获取 dom 元素的时候, 只能获取到第一个元素, 后面的元素都无法正常获取.

15. 用伪类实现一个上三角?

```
<div class=' triangle_border_up' ></div>
```

```
.triangle_border_up{
  border:20px solid red;
  border-top:0;
  border-left:20px solid transparent;
  border-right:20px solid transparent;
  width:0px;
}
```

16. 怎么让一个不定宽高的 div, 垂直水平居中?

方案一: transform

```
.parent{
  background: #DDD;
```

```
width: 400px;
height: 400px;
}
.son{
  position: relative;
  background: pink;
  width: 200px;
  height: 200px;
  top: 50%;
  left: 50%;
  transform: translate(-50%,-50%);
}
```

### 方案二: flex 弹性布局

```
.parent{
  display: flex;
  justify-content: center;
  align-items: center;
  background: #DDD;
  width: 400px;
  height: 400px;
}
.son{
  background: pink;
  width: 200px;
  height: 200px;
}
```

### 方案三: 绝对定位

```
.parent{
  position: relative;
  background: #DDD;
  width: 400px;
  height: 400px;
}
.son{
  position: absolute;
  top: 0;
  bottom: 0;
  left: 0;
```

```
right:0;
background: pink;
width: 200px;
height: 200px;
margin: auto;
}
```

## 17. 清除浮动有哪些常用的方式?

额外标签法：在浮动元素的最后添加一个块级标签，给其设置一个 `clear:both` 的属性（缺点：会在页面上产生很多空白标签）；

给浮动元素的父元素设置高度：（缺点：不太灵活）；

给浮动元素的父元素设置 `overflow:hidden`；

使用伪元素法：（推荐使用）

```
.clear:after{
  content:'';
  display:block;
  overflow:hidden;
  visibility:hidden;
  clear:both;
}
```

## 18. 让两个块级元素在一行显示有哪些做法?

设置显示模式：`display:inline|inline-block`；

flex 布局：给父元素设置 `display:flex`；

使用浮动

## 19. 如何设置一个元素在垂直方向居中?

首先不考虑代码的灵活性，可以使用 `margin` 外边距或者 `padding` 内边距来实现元素在垂直方向居中显示。具体可以给父元素设置一个垂直方向的 `padding` 内边距；也可以给需要垂直居中的子元素设置垂直方向的外边距。其次如果这个需要垂直居中的元素是一个单行文本，则可以使用行高等于标签高度的方式来实现。也可以使用 `css3` 中的 `flex` 布局，使用 `align-items:center` 设置元素在侧轴(垂直方向)居中对齐。也可以使用绝对定位的方式，设置元素在相对定位的父元素中垂直对齐。

20. 说说图片懒加载的原理?实际开发中用过哪些图片懒加载的插件?  
`img` 标签在加载图片的时候，是通过请求 `src` 属性所指向的文件来加载图片的，那如果 `img` 标签本身没有 `src` 属性的话，那么 `img` 标签在渲染的时候，就不会加载图片。所以图片懒加载的原理就是将 `img` 标签的 `src` 属性给暂时先改成一个自定义的属性，这样页面已加载就会不去加载图片，当 `img` 标签所在区域进入屏幕可视区域后，从存放图片路径的自定义属性中获取图片地址，并动态的设置给对应 `img` 标签的 `src` 属性，这样浏览器就会自动帮助我们去请求对应的图片资源，也就实现了所谓的图片懒加载。图片懒加载的插件有很多，大部分是基于 `jquery` 的，比如 `jquery.lazyload`。当然 `vue` 的中也有实现了图片懒加载的插件，比如 `vue-lazyload`，`vue` 的组件库中也有图片懒加载的组件。



## 21. css3 新增了那些新特性?

媒体查询(@media);

transfrom 系列:translate 平移, scale 缩放, rotate 旋转

动画(animate);

过渡效果(transition);

flex 弹性(伸缩)布局;

盒模型计算方式 box-sizing:border-box;

线性渐变(linear-gradient), 径向渐变;

伪元素, 文字阴影 (text-shadow), 边框阴影 (box-shadow), 圆角 (border-radius)

## 22. display:none 和 visibility:hidden 的区别?

display:none 隐藏元素后, 不占位;

visibility:hidden 隐藏元素后占位.

## 23. Less 是什么?

Less 是一种 css 预处理语言, 在 less 中可以定义一些变量和表达式以及使用嵌套语法; less 中使用@定义变量(@baseColor:pink); 后期可以通过一些编译工具(less)将 less 编译成浏览器能直接识别的 css 样式. 所以 less 只是在开发阶段使用的一种中间语言, 使用 less 的目的是提高开发效率以及提高代码的可维护性.

## 24. Scss 是什么?(sass)

scss 是一种 css 预处理语言，在 less 中可以定义一些变量和表达式以及使用嵌套语法；scss 中使用\$定义变量(\$baseColor:pink)；后期可以通过一些编译工具(node-sass)将 less 编译成浏览器能直接识别的 css 样式。所以 scss 只是在开发阶段使用的一种中间语言，使用 scss 的目的是提高开发效率以及提高代码的可维护性。

## 25. Stylus 是什么?(styl)

stylus 是一种 css 预处理语言，在 stylus 中可以定义一些变量和表达式以及使用嵌套语法(stylus 中是使用缩进的语法表示嵌套关系)；后期可以通过一些编译工具(stylus)将 stylus 编译成浏览器能直接识别的 css 样式。所以 stylus 只是在开发阶段使用的一种中间语言，使用 stylus 的目的是提高开发效率以及提高代码的可维护性。

# JavaScript

## JavaScript 基础

### 26. js 中有哪些数据类型

int(数值), string(字符串), boolean(布尔), null(空), undefined(未定义), object(对象)

### 27. typeof(typeof()) 和 instanceof 的区别?

typeof 可以判断变量的数据类型, 返回值是字符串;

a instanceof b 是判断 b 是不是在 a 的原型链上, 也可以实现判断数据类型, 返回值为布尔.

28. 怎么判断两个对象相等?

先判断俩者是不是对象;

再判断俩个对象的所有 key 值是否相等相同;

最后判断俩个对象的相应的 key 对应的值是否相同

29. js 中函数有哪些定义方式

函数声明: `function fn() {}`

函数表达式: `var fn=function() {}`

构造函数: `var fn=new Function( '参数 1' , '参数 2' , '函数体' )`

30. js 中函数有哪些调用形式?

普通函数, 对象的方法, 事件处理函数, 构造函数, 回调函数

31. "==" 和 "===" 的区别?

== 只会对值进行比较, === 不仅会对值进行比较, 还会对数据类型进行比较.

32. js 中的常用内置对象有哪些？并列举该对象的常用方法？

Math(数学相关);Date(日期相关);Array;Object

33. 列举和数组操作相关的方法

push:将元素添加到数组的末尾, 返回值是数组长度

pop:将数组最后一个元素弹出, 返回值是被弹出的元素

unshift:在数组的开头插入一个元素, 返回值是数组的长度

shift:将数组第一个元素弹出, 返回值是被弹出的元素

splice(index, len):删除数组中指定元素

concat:连接数组

reverse: 翻转数组

34. 列举和字符串操相关的方法

substr(start, len)/substring(start, end): 截取字符串

slice:从数组会字符串中截取一段

indexOf/lastIndexOf:查找某一个字符是否存在于另外一个字符串中, 存在则返回索引, 不存在则返回-1;indexOf 是从前向后顺序查找;

lastIndexOf:是从后向前查找

replace:替换字符串特定的字符

toUpperCase:将字符串转成大写

toLowerCase:将字符串转成小写

charAt: 获取字符串中指定索引的字符

35. document.write 和 innerHTML 的区别?

document.write 是指定在整个页面区域的内容, innerHTML 是指定某一个元素的内容.

36. 分别阐述 split(), slice(), splice(), join() ?

split 可以使用一个字符串切割另外一个字符串, 返回值是数组;

slice 可以从数组中截取一部分(字符串对象也有 slice 方法);

splice(index, len) 可以删除指定的数组元素;

join 可以将数组元素使用特定的连接符拼接成字符串

37. 例举 3 中强制类型转换和 2 中隐式类型转换?

强制转换:

转化成字符串 toString() String() 转换成数字 Number()、  
parseInt()、parseFloat();

隐式转换:

转换成布尔类型 Boolean() 隐式拼接字符串

例子 var str = "" + - / % ===

38. 如何判断一个变量 foo 是数组?

```
foo instanceof Array;
```

```
foo.constructor == Array;  
Array.isArray(foo)  
Object.prototype.toString.call(foo)=="[object Array]"
```

## Javascript 高级

### 39. 什么是原型对象？

每一个构造函数都有一个 prototype 的属性，这个属性的值是一个对象，这个对象就叫做构造函数的原型对象；一般建议将构造函数的成员属性绑定在原型对象 prototype 上，因为原型对象 prototype 身上的属性默认可以通过实例对象访问到；这样做可以保证在每次通过 new 关键字创建实例对象的时候，这些方法不会被重复在内存中创建。

### 40. 什么是原型链？

每个构造函数都有一个 prototype 属性，即原型对象，通过实例对象的\_\_proto\_\_属性也可访问原型对象；而原型对象本质也是一个对象，是对象就有自己的原型对象，最终形成的链状的结构称为原型链。

### 41. 什么是构造函数？

构造函数本质也是一个函数，只不过这个函数在定义的时候首字母一般需要大写；构造函数调用的时候，必须通过一个 new 关键字来调

用；我们一般不直接使用构造函数，而是使用构造函数创建出来的实例对象。构造函数是 js 面向对象的一个重要组成部分。

#### 42. js 中实现继承的方式？

ES6 之前官方并没有提供一种实现继承的语法，所以大部分继承方式都是程序员通过代码在模拟。常见的继承方式有以下几种：

原型继承；

借用构造函数继承；

组合继承；

```
function Person(name,age,gender){
    this.name=name||'';
    this.age=age||'';
    this.gender=gender||'';
}
Person.prototype.sayHi=function(){
    console.log('I am '+this.name);
}
var p1=new Person('zs',30,'男');
function Student(name,age,gender,score){
    // 通过构造继承属性
    Person.call(this,name,age,gender);
}

// 通过原型继承,继承方法
Student.prototype=new Person();
// 修改 constructor 的指向
Student.prototype.constructor=Student;
// 动态添加成员方法
Student.prototype.printScore=function(){
    console.log('my score is '+this.score);
}
// 创建 Student 实例对象
var s1=new Student('zs',30,'男',90);
s1.sayHi();
```

```
s1.printScore();
```

ES6 之后使用 extends 关键字实现继承 (class Student extends Person {})

43. 什么是闭包，有什么作用，使用的时候需要注意什么？

闭包是一个跟函数相关的概念，表现形式是一个父函数内部，嵌套了一个子函数，子函数直接或间接的被返回给外部作用域，并且子函数中会使用到父函数局部作用域中的变量。当我们在外部调用这个子函数的时候，就会发生闭包现象。

闭包的作用：闭包可以延展一个函数的作用域

注意事项：不能滥用闭包，会导致内存泄漏

```
function fn(){  
  var a=100;  
  return function(){  
    return a;  
  }  
}  
var fn1=fn();  
fn1();
```

44. 什么是内存泄漏，那些操作会引起内存泄漏？

内存泄漏是指本应该被垃圾回收机制回收的内存空间由于某种特殊原因没有及时被回收，称之为内存泄漏。滥用全局变量和滥用闭包都会导致内存泄漏。



#### 45. 什么是预解析?

JS 代码在执行之前, 解析引擎会对代码进行一个预先的检查, 主要会对变量和函数的声明进行提升, 将变量和函数的声明提到代码的最前面. 变量只提升声明, 不提升赋值.

#### 46. 说说你对 this 关键字的理解

this 在不同的场景下指向不太一样, 主要分为以下几种情况:

普通函数中指向全局 window;

对象的成员方法中指向该方法的宿主对象;

构造函数中指向 new 出来的实例对象;

事件处理函数中指向事件源;

回调函数中指向全局 window

#### 47. call/apply/bind 的区别

这三个方法都是函数这个特殊对象的方法, 通过这三个方法都可以改变函数内部 this 的指向.

不同点:

call 和 apply 会调用一次函数, 而 bind 不会调用函数, 只会在内存中创建一个函数的副本(修改过 this 指向的函数).

call 从第二个参数开始需要一个参数列表,

apply 第二个参数需要是一个数组

48. new 操作符具体干了什么呢?

第一步创建一个空对象;

第二步将 this 指向空对象;

第三步动态给刚创建的对象添加成员属性;

第四步隐式返回 this

## 代码分析

49. 下面代码的执行结果是什么?

```
var hellword=(function(){  
  console.log('hello one');  
  setTimeout(function(){  
    console.log('hello two');  
  },100);  
  setTimeout(function(){  
    console.log('hello three');  
  },0);  
  console.log('hello four');  
})();
```

依次输出: hello one,hello four,hello three,hello two

50. 下面代码执行结果是什么?

```
var a={  
  id:10  
}  
b=a;  
b.id=1;  
b.name='test';  
console.log(a);
```

执行结果:

输出 {id: 1, name: "test"}

分析过程:

对象是一种引用数据类型, 简单的  $b=a$  只是把  $a$  在内存中的地址赋值给了  $b$ , 所以修改  $b$  会影响  $a$ .

51. 下面代码执行结果是什么?

```
var length=10;
function fn(){
  console.log(this.length);
}
var obj={
  length:5,
  method:function(fn){
    fn();
    arguments[0]()();
  }
}
obj.method(fn,1);
```

执行结果:

在控制台输出 10,2

分析过程:

$fn()$ ; 此时  $this$  指向  $window$ , 所以  $this.length=10$ ;

$arguments[0]()$  中的  $this$  永远指向  $arguments$ , 而  $arguments$  本身有一个  $length$  属性, 就是参数的个数.

52. 下面代码执行完毕, 浏览器依次弹出什么?

```
(function test(){
```

```
var a=b=5;
alert(typeof a);
alert(typeof b);
})();
alert(typeof a);
alert(typeof b);
```

执行结果:

依次弹出: number; number,undefined,number

分析过程:

自调用函数会开辟一个局部作用域, var a=b=5 这句代码 var 只会修饰 a, 所以 a 是一个局部变量, b 是全局变量

53. 下面代码输出结果是什么?

[1,2,3].map(parseInt);

输出结果:[1,NaN,NaN];

分析过程:

```
[1,2,3].map(function(item,index){
  // console.log(item,index);
  //parseInt(数值,进制)
  parseInt(1,0);
  parseInt(2,1);
  parseInt(3,2);
});
```

54. 下面代码执行结果是什么?

```
console.log(square(5));
var square=function(n){
  return n*n;
```

```
}
```

执行结果:

报错(Uncaught TypeError: square is not a function)

分析过程:

函数表达式方式声明的函数只提升声明, 不提升赋值, 所以不能再声明之前调用.

55. 下面代码执行结果是什么?

```
console.log(2.0=='2'==new Boolean(true)=='1');
```

执行结果: 输出 true

分析过程: 2.0=='2' 返回 true; true==new Boolean(true) 返回 true; true=='1'返回 true; 所以最终结果是 true.

56. 下面的代码会输出什么?怎么改动下面代码, 使其依次输出

1,2,3,4,5

```
for(var i=1;i<=5;i++){
  setTimeout(function(){
    console.log(i);
  },1000);
}
```

执行结果:

在控制台输出:6,6,6,6,6

改造后的代码:

```
for (var i = 1; i <= 5; i++) {
  (function (i) {
    setTimeout(function () {
      console.log(i);
    }, 1000*i)
  })(i)
}
```

57. 下面代码执行结果是什么?

```
var a=10;
function Foo(){
  if(true){
    let a=4;
  }
  alert(a);
}
Foo();
```

执行结果: 弹出 10

分析过程: let 声明的变量有块级作用域, 所以 let 声明的 a 只在 if 条件的花括号中生效, 所以会向上级作用域查找.

## 编码题

58. 使用 js 封装一个冒泡排序

```
// 冒泡排序
function sortBubble(arr){
  for(var i=0;i<arr.length;i++){
    for(var j=0;j<arr.length-i;j++){
      if(arr[j]>arr[j+1]){
        var temp=arr[j];
        arr[j]=arr[j+1];
        arr[j+1]=temp;
      }
    }
  }
}
```

```

    }
    return arr;
}

```

## 59. 封装一个方法实现去除数组中的重复元素

方案一:

```

// 数组去重
function unique(arr){
    var newArr=[];
    for(var i=0;i<arr.length;i++){
        if(newArr.indexOf(arr[i])===-1){
            newArr.push(arr[i]);
        }
    }
    return newArr;
}

```

方案二:

```

var arr=[1,2,2,3,3,4]
Array.from(new Set(arr))

```

分析过程:

Set 是 es6 中新增的一种数据类型, 和数组很类似, 但是元素不能重复; Array.from 也是 es6 新增的方法, 可以将类数组对象(伪数组, set), 转换成数组.

## 60. 已知数组 var arr=['This', 'is', 'Woqu', 'Company'], alert 出 "This is Woqu Company".

```

alert(arr.join(' '));

```

61. 编写一个 js 函数 `parseQueryString`, 它的用途是把 url 中的参数解析为一个对象, 如

`var url="http://www.demo.cn/index.html?key1=val1&key2=val2"`

```
function parseQueryString(argu) {
    var str = argu.split('?')[1];
    var result = {};
    var temp = str.split('&');
    for (var i = 0; i < temp.length; i++) {
        var temp2 = temp[i].split('=');
        result[temp2[0]] = temp2[1];
    }
    return result;
}
```

62. 统计 `str="jhadfgskjfajhdewqe"` 字符串中出现最多的字母?

```
function countStr(str){
    var json = {};
    // 循环完后会得到一个对象,如{a:0,b:1,c:2,d:3,e:4}
    for (var i = 0; i < str.length; i++) {
        if (!json[str.charAt(i)]) {
            json[str.charAt(i)] = 1;
        } else {
            json[str.charAt(i)]++;
        }
    }
};
var iMax = 0;
var iIndex = '';
// 查找出现次数最多的字符,和出现次数
for (var i in json) {
    if (json[i] > iMax) {
        iMax = json[i];
        iIndex = i;
    }
}
return {
```



```

    count:iMax, //出现的最多次数
    char:iIndex // 出现次数做多的字符
  }
}

```

### 63. 编码实现对象深拷贝

```

function deepClone(obj) {
  if (obj instanceof Object) {
    let isArray = Array.isArray(obj)
    let cloneObj = isArray ? [] : {}
    for (let key in obj) {
      cloneObj[key] = isObject(obj[key]) ? deepClone(obj[key]) : obj[key]
    }
  } else {
    throw new Error('obj 不是一个对象! ')
  }
  return cloneObj
}

```

### 64. 有 Student 和 Person 两个类, Person 类有 name 属性和 sayName 方法, Student 类继承自 Person 类. 分别使用 ES5 和 ES6 的语法实现.

ES6 实现:

```

class Person{
  constructor(props){
    this.name=props.name;
  }
  sayName(){
    console.log(`My name is ${this.name}`);
  }
}

class Student extends Person{
  constructor(props){
    super(props);
    this.name=props.name;
  }
}

```

```

    }
}

```

## ES5 实现

```

function Person(name='') {
    this.name = name;
}
Person.prototype.sayName = function() {
    console.log(`My name is ${this.name}`);
}

function Student(name) {
    Person.call(this, name)
}
Student.prototype = new Person();
Student.prototype.constructor = Student;

```

65. 写一个左中右布局占满屏幕, 其中左右两块固定宽度 200, 中间自适应, 要求先加载中间块, 请写出结构和样式

## Css 样式

```

*{
    padding: 0;
    margin: 0;
}
html, body {
    height: 100%;
}
.center {
    height: 100%;
    background: #1FA363;
    margin: 0 200px;
}
.left {
    position: absolute;
    width: 200px;
    height: 100%;
    left: 0;
}

```

```

    top: 0;
    background: #DC4C3F;
}
.right{
    position: absolute;
    width: 200px;
    height: 100%;
    right: 0;
    top: 0;
    background: #FFCE44;
}

```

html 结构:

```

<div class="center">center</div>
<div class="left">left</div>
<div class="right">right</div>

```

思路分析: html 标签的加载顺序是自上而下, 所以要想让中间部分先加载, 只需要把中间部分的标签写在最前面即可.

66. 如何扩展 jquery 的静态方法, 如\$.getName();

```

$.extend({
    getName: function () {
        // do something
    }
});

```

67. 使用 js 求 10000 以内的所有质数的和.

```

function getZs(num) {
    var sum=0;
    for (var i = 2; i <= num; i++) {
        //假设所有的数都是质数
        var flag = true;
        //通过嵌套循环找到 i 除了 1 和本身以外所有可能出现的因子
    }
}

```

```

    for (var j = 2; j < i; j++) {
        //判断 i 是否为质数
        if (i % j == 0) { //能进到当前的分支 说明不是质数
            flag = false;
        }
    }
    if (flag == true) {
        console.log(i);
        sum+=i;
    }
}
return sum;
}

```

68. 使用 js 打印出 1-10000 之间的所有对称数(如 121, 1331)

```

function isSymNum(start, end) {
    start = (start <= 11 ? 11 : start);
    for (var i = start; i <= end; i++) {
        var strI = +(i.toString().split('').reverse().join(''));
        if (strI == i) {
            console.log(i);
        }
    }
}
}

```

69. 二维数组根据 num 的值进行升序排序:

```

var list = [
    {
        id: 32, num: 5
    },
    {
        id: 28, num: 12
    },
    {
        id: 23, num: 9
    }
]

```

实现过程:

```
list.sort(function(a,b){
    return a.num-b.num;
})
```

70. Js 中 eval 的功能是什么？缺点是什么？

eval 函数的作用：可以将一个字符串当做 js 代码执行。

缺点：执行效率比较低，不安全。

71. 有一个数列(0,1,1,2,3,5,8,13,21...),定义函数求数列第 n 项

```
function getFibo(n){
    if(n==1) return 0;
    if(n==2) return 1;
    return getFibo(n-1)+getFibo(n-2);
}
```

72. 使用什么办法能让如下条件判断成立？

```
if(a==1&&a==2&&a==3){
    console.log('ok')
}
```

方案一：

```
var a={
    value:1,
    toString:function(){
        return this.value++;
    }
}
```

方案二：

```
var a = [1,2,3];
a.join = a.shift;
```

思路分析:

数组本身有一个 join 方法, 在把数组当做简单数据类型调用的时候, 会自动调用 join; 而 shift 也是一个数组的方法, shift 是将数组的开头元素删除, 返回值就是删除的元素, a.join=a.shift 相当于在每一次调用 a 的时候都会调用 shift 方法.

方案三:

```
var init=1;
Object.defineProperty(window, 'a', {
  get: function () {
    return init++;
  }
});
```

73. 下面代码输出结果是什么?

```
function changeObjectProperty(o){
  // 输出的是这个结果
  o.siteUrl = "http://www.csser.com/";
  o = new Object();
  o.siteUrl = "http://www.popcg.com/";
}
var CSSer = new Object();
changeObjectProperty( CSSer );

console.log( CSSer.siteUrl );
```

输出结果: http://www.csser.com/

## WebAPI

### 74. 列举 DOM 元素增删改查的 API

创建 DOM: `document.createElement()`;

查找 DOM:

`document.getElementById()`;

`document.getElementsByTagName()`;

`document.getElementsByName()`;

`document.querySelectorAll()`;

`document.querySelector()`;

追加 DOM: `parentDom.appendChild()`;

移除 DOM: `parentDom.removeChild()`

### 75. BOM 中有哪些常用的对象?

location:

`location.href`; 页面 url 地址

`location.hash`; url 中#后的部分

`location.search`; url 中?后的部分(查询字符串)

`location.reload()`; 刷新页面;

navigator:

`navigator.userAgent`: 浏览器的 userAgent 信息

history:

`history.go(1)`; 前进 1 步

history.go(-1);后退 1 步;

history.forward();前进

history.back(); 后退

screen:

screen.availWidth: 屏幕有效宽度

screen.availHeight: 屏幕有效高度

76. 列举几个常见的浏览器兼容问题?

主流浏览器发送 ajax 使用 XMLHttpRequest 创建异步对象,

IE 浏览器时候用 XActive 创建异步对象;

主流浏览器注册事件

addEventListener("eventType", "handler", "true|false");

removeEventListener("eventType", "handler", "true|false");

IE 浏览器:

注册事件:attachEvent( "eventType", "handler")

移除事件:detachEvent("eventType", "handler" )

阻止事件冒泡:

主流浏览器:event.stopPropagation()

IE 浏览器:event.cancleBubble=true;

获取事件源:

主流浏览器: event.target

IE 浏览器:event.srcElement



## 77. 什么是事件委托?

本应该注册给子元素的事件，注册给父元素

## 78. 事件委托的原理是什么?

事件冒泡，因为有事件冒泡的存在，所以子元素的事件会向外冒泡，触发父元素的相同事件，根据事件对象可以找到真正触发事件的事件源。

## 79. Javascript 中有几种定时器，有什么区别?

setInterval：间歇定时器，间隔一定的事件就执行，执行多次；

setTimeout：延时定时器，只执行一次

## 80. 如何实现多个标签页的通信?

localStorage 可以实现同一浏览器多个标签页之间通信的原理；

localStorage 是 Storage 对象的实例。对 Storage 对象进行任何修改，都会在文档上触发 storage 事件。当通过属性或者 setItem() 方法保存数据，使用 delete 操作符或 removeItem() 删除数据，或者调用 clear() 方法时，都会发生该事件。

A. html

```
<input type="text">  
<button id="btn">Click</button>  
<script>
```

```

window.onload=function(){
    var oBtn=document.getElementById("btn");
    var oInput=document.getElementsByTagName("input")[0];
    oBtn.onclick=function(){
        var val=oInput.value;
        localStorage.setItem("value",val);
    }
}
</script>

```

B. html

```

<script>
    window.addEventListener("storage",function(event){
        console.log("value is"+localStorage.getItem("value"));
        console.log("key is"+event.newValue);
    },false);
</script>

```

JQuery



81. jquery 中的\$.each 和\$(selector).each()有什么不同?

\$.each 可以循环任何数组, 包括普通数组和 jquery 对象组成的伪数组;

\$(selector).each()只能循环遍历 jquery 对象组成的伪数组.

82. JQuery 中\$.each 和原生 js 中的 forEach 方法有什么区别?

Jquer 中的\$.each 不仅可以循环遍历普通数组, 还可以循环遍历 jquery 对象的伪数组, 原生 js 中的 forEach 只能循环遍历数组; 其次第二个实参函数的参数顺序不一样, \$.each(arr,function(索引,循环单项,数组本身){}), arr.forEach(function(循环单项,索引,数组本身){})

83. 原生 JS 的 window.onload 与 Jquery 的\$(document).ready(function()

{}), \$(function () {}))有什么不同?

执行时机不一样, window.onload 会等待页面元素渲染完毕并且资源文件加载完毕后会才会执行;\$(document).ready(function() {})是当页面元素渲染完毕后会就会执行, 所以执行时机先于 window.onload

84. Jquery 实现连式编程的原理是什么?

jquery 的方法中最后都会 return 一个 this, 这个 this 就是当前元素的 jquery 对象.

```
function $(parma) {
  //如果调用者传入的是一个函数, 则当做入口函数使用
  if (typeof (parma) == 'function') {
    window.onload = parma
  } else { //如果调用者传入的是一个选择器, 则返回一个对象
    var dom = document.querySelector(parma)
    return {
      0: dom,
      //{color:'red',border:'1px solid red'}
      css: function (obj) {
        if (typeof (obj) == 'object') {
          for (var key in obj) {
            dom.style[key] = obj[key];
          }
        }
      },
      return this;
    },
    click: function (fn) {
      // 注册点击事件
      dom.onclick = fn;
      return this;
    },
    hide: function () {
      dom.style.display = 'none';
      return this;
    }
  }
}
```

```

    },
    show: function () {
        dom.style.display = 'block';
        return this;
    },
    get: function () {
        return dom;
    }
}
}
}
}

```

85. Jquery 如何多次给同一个标签绑定同一个事件?

使用 `addEventListener('事件名',function(){}))` 注册的事件, 不会出现事件覆盖, jquery 中也是这样做的.

86. 如何开发 jquery 插件?

Jquery 提供了两种开发插件的方式:

`$.fn`: 可以通过任意 jquery 对象来调用

```

$.fn.green=function(){
    console.log(this,$(this));
    $(this).css({background:'green'});
}
// 调用以后, div 的背景色会被设置成 green
$("div").green();

```

`$.extend`: 开发的插件只能通过 `$` 顶级对象来调用

```

// 定义
$.extend({
    alert: function (msg) {

```

```

    alert(msg);
  }
});
// 调用
$.alert('这是提示信息')

```

87. Jquery 中那些方法不支持链式操作?

`$.trim()`; `$.each()`; `$(selector).html()`, `$(selector).text()`

## H5 新特性

88. H5 都新增了那些新特性?

语义化的标签(header,nav,footer,aside,article,section)

本地存储 sessionStorage,localStorage;

拖拽释放(Drag and drop) API 音频、视频 API(audio,video)

画布(Canvas) API

地理(Geolocation) API

表单控件, calendar、date、time、email、url、search

新的技术 websocket

89. sessionStorage,localStorage 和 cookie 三者有什么区别?

共同点:它们三者都是浏览器端的存储介质, 可以存储一些数据.

不同点:

sessionStorage 是将数据存储在页面的内存中, 所以数据会跟随页面的关闭而销毁, 存储数据相对较少(5M 左右), 只能存储字符串;

localStorage 是将数据存储在电脑的磁盘上, 存储数据量大(20M 左右), 需要手动删除, 只能存储字符串;

cookie 是 http 协议的重要组成部分, 存储数据量相对比较少(4K 左右), 存储 cookie 的时候可以设置过期时间, 到达过期时间后, 会自动销毁, 如果没有设置, 则跟随浏览器的关闭而销毁. cookie 中存储的数据会伴随每一次 http 请求被发送到服务端, 所以不建议在 cookie 中存储大量数据.

## 数据交互(ajax)

90. 使用 jquery 写出一个简单的\$.ajax 的请求

```
$.ajax({  
  url: '/api',  
  type: 'post',  
  data: {},  
  dataType: 'json',  
  success: function(res){  
    console.log(res);  
  }  
});
```

91. 常见 HTTP 状态码都有哪些?

100 => 正在初始化 (一般是看不到的)

101 => 正在切换协议 (websocket 浏览器提供的)

200 或者以 2 开头的两位数 => 都是代表响应主体的内容已经成功返回了

202 => 表示接受

301 => 永久重定向/永久转移

302 => 临时重定向/临时转移（一般用来做服务器负载均衡）

304 => 本次获取的内容是读取缓存中的数据，会每次去服务器校验

400 => 参数出现错误（客户端传递给服务器端的参数出现错误）

401 => 未认证，没有登录网站

403 => 禁止访问，没有权限

404 => 客户端访问的地址不存在

500 => 未知的服务器错误

503 => 服务器超负荷（假设一台服务器只能承受 10000 人，当第 10001 人访问的时候，如果服务器没有做负载均衡，那么这个人的网络状态码就是 503）

92. 你知道的 HTTP 请求方式有几种？

1. GET          请求指定的页面信息，并返回实体主体。
2. HEAD        类似于 get 请求，只不过返回的响应中没有具体的内容，用于获取报头
3. POST        向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求体中。POST 请求可能会导致新的资源的建立和/或已有资源的修改。
4. PUT          从客户端向服务器传送的数据取代指定的文档的内容。
5. DELETE      请求服务器删除指定的页面。

- 6. CONNECT HTTP/1.1 协议中预留给能够将连接改为管道方式的代理服务器。
- 7. OPTIONS 允许客户端查看服务器的性能。
- 8. TRACE 回显服务器收到的请求，主要用于测试或诊断。
- 9. PATCH 实体中包含一个表，表中说明与该 URI 所表示的原内容的区别。
- 10. MOVE 请求服务器将指定的页面移至另一个网络地址。
- 11. COPY 请求服务器将指定的页面拷贝至另一个网络地址。
- 12. LINK 请求服务器建立链接关系。
- 13. UNLINK 断开链接关系。
- 14. WRAPPED 允许客户端发送经过封装的请求。
- 15. LOCK 允许用户锁定资源，比如可以再编辑某个资源时将其锁定，以防别人同时对其进行编辑。
- 16. MKCOL 允许用户创建资源
- 17. Extension-mothed 在不改动协议的前提下，可增加另外的方法。

### 93. 请尽可能详尽的解释 ajax 的工作原理

第一步:创建一部对象 `var xhr=new XMLHttpRequest()`

第二步:设置请求行 `xhr.open('请求方式',请求地址);`

第三步:发送请求 Get 方式 `xhr.send(null),`

如果是 post 请求还要设置请求头



```

xhr.setRequestHeader('Content-Type','application/x-www-form-urlencoded');
xhr.send("name=zs&age=18");
//第四步:监听服务端的响应
xhr.onreadystatechange=function(){
    if(xhr.status==200&&xhr.readyState==4){
        // 获取 json
        var json=xhr.responseText&&JSON.parse(xhr.responseText)
        // 获取 xml
        var xml=xhr.responseXML;
        console.log(json,xml)
    }
}
}

```

94. 页面编码和被请求的资源编码如果不一致如何处理？

a.html 的编码是 gbk 或 gb2312 的。而引入的 js 编码为 utf-8 的，那就在引入的时候

`<script src="http://www.xxx.com/test.js" charset="utf-8"></script>`

同理，如果你的页面是 utf-8 的，引入的 js 是 gbk 的，那么就需要加上 `charset="gbk"`。

95. 如何解决跨域问题？

jsonp, 服务器代理, cors

96. jsonp 跨域的原理是什么？

动态在页面中创建一个 script 标签，使其 src 属性指向后端数据接口，后端数据接口必须返回一个 js 函数的调用字符串（如 `cb({'name':"zs","age":18})`），将要返回给前端的 json 数据作为函数的

实参, 当 `script` 标签加载完毕后会会在浏览器中执行后端返回的函数调用, 所以前端必须事先对调用的函数进行声明. 因为函数是在 `js` 中声明的, 所以可以在函数内部拿到服务端调用的时候传入的实参, 所以就间接实现了跨域请求数据.

### 97. 什么是同步和异步, 那种执行方式更好?

同步是指一个程序执行完了接着去执行另外一个程序, 异步是指多个程序同时执行. 所以异步效率更高, 因为异步不会出现阻塞现象, 前一个程序的执行不会影响后一个程序的执行.

### 98. GET 和 POST 的区别, 何时使用 POST?

`get` 是将要传递的参数拼在 `url` 中进行传递, 传递数据量少, 不安全  
`post` 是将传递的参数放在请求体里传递, 携带数据量大, 相对安全.  
要提交一些敏感数据 (比如登录), 上传文件时, 必须使用 `post` 请求.

### 99. 请解释一下 JavaScript 的同源策略

同源策略是浏览器的一项安全策略, 浏览器只允许 `js` 代码请求和当前所在服务器域名, 端口, 协议相同的数据接口上的数据, 这就是同源策略.

### 100. 一个页面从输入 URL 到页面加载显示完成, 这个过程中都发生了什么?

首先根据域名查询 DNS 服务器获取服务器 IP, 然后拿着服务器 IP 和域名请求对应的服务器, 请求成功后 web 服务器会根据一系列运算, 将客户端需要的数据通过网络传输到客户端浏览器, 最终由浏览器解析后呈现给终端用户.

101. 网站从 http 协议切换到 https 协议需要对代码做哪些处理?  
不需要对代码做任何处理, 只需要在 web 服务器中加入一个 ssl 的安全认证模块即可.

102. 什么是 RESTful API?

RESTful 的核心思想就是, 客户端发出的数据操作指令都是"动词 + 宾语"的结构。比如, GET /articles 这个命令, GET 是动词, /articles 是宾语。

补充说明:

动词通常就是五种 HTTP 方法, 对应 CRUD 操作:

GET: 读取 (Read)

POST: 新建 (Create)

PUT: 更新 (Update)

PATCH: 更新 (Update), 通常是部分更新

DELETE: 删除 (Delete)

# 性能优化

## 103. 什么是页面的回流和重绘?

回流是指当页面的结构或者标签的尺寸发生变化的时候, 浏览器需要对页面进行重排, 并重新渲染;

重绘是指当页面上的标签的外观(比如字体颜色, 或背景颜色)发生改变的时候, 浏览器需要重新对页面进行渲染.

所以回流一定会引起页面的重绘, 重绘不一定会引起回流.

要提高页面性能, 就要尽可能的减少页面的回流和重绘.

## 104. 针对页面性能优化,你有哪些优化方案?

资源加载方面:

减少 http 请求次数, 具体方案, 代码合并(合并 css,js), 使用精灵图;

减少 http 请求数据量, 代码压缩(css,js,html), 合理设置缓存;

启用 CDN 加速服务;

代码层面:

避免滥用全局变量, 减少作用域查找(能用局部变量就不要声明全局变量), 不要滥用闭包;

减少 DOM 操作, 操作 DOM 的时候对已经查找到的 DOM 对象进行缓存, 避免重复查找;

使用图片懒加载, 避免单次加载图片数量过多导致页面卡顿;

将 script 标签写在页面底部, 因为 js 的加载会阻塞页面的渲染;

不要在本地书写大量 cookie, 因为 cookie 会伴随每一次 http 请求;

#### 105. 什么是 CDN 加速?

CDN(Content Delivery Network)全称内容分发网络, 是运营商所提供的一项增值服务, 花钱就可以拥有这项加速服务. CDN 主要是对网站的静态资源进行加速, CDN 在全国会有很多节点服务器(每个城市都有), 当你购买了一个 CDN 服务以后, CDN 服务器会对你的网站的静态资源文件进行缓存处理, 当第二次有人访问的时候, 那么服务器就会从就近的 CDN 节点服务器上获取网站所需的静态资源, 由于 CDN 服务器的性能比较高, 并且距离客户端的物理距离比较近, 所以就可以实现加速. 启用 CDN 服务只需要在运营提供商提供的后台进行配置(配置要对那个域名启用 CDN 服务), 不需要对代码做任何修改.

#### 106. 什么是 SEO?

SEO(Search Engine Optimizing)搜索引擎优化, 就是让搜索引擎去抓取我们的网页. 为了让搜索引擎抓取我们的网页, 我们可以在书写代码的时候做一些工作, 比如合理设置网页 title(标题), keywords(关键字), description(描述); 因为搜索引擎在抓取到网页以后首先回去分析这几个关键信息.

#### 107. 为什么利用多个域名来存储网站静态资源会更有效?

因为浏览器对请求静态资源文件有一个并发数量限制, 每次只能请

求同一个域名下的若干个资源文件(根据浏览器的不同会有差异), 如果把资源文件存放在多个不同的域名下面就会突破浏览器的限制; 其次, 启用多个静态资源服务器, 可以减轻主服务器的压力.

108. 移动端点击事件会有多少秒的延时? 什么原因造成的? 如何解决?

移动端的点击事件会有 300ms 的延时;

是因为浏览器为了保留双击缩放的功能所造成的, 早期浏览器都有一个双击缩放的功能, 在用户点击一次以后, 浏览器会等待第二次点击, 如果用户在 300ms 内进行了第二次点击, 那么浏览器就会执行缩放的功能, 如果 300ms 内没有再次点击, 则会当做单击事件处理;

解决方案:

使用 touch 触摸事件来模拟点击事件; 使用 fastclick 插件来解决; 静止页面缩放功能

109. 你了解到的网站攻击方式有哪些?

常见的网站攻击方式有 xss(跨站脚本攻击), csrf(跨站请求伪造)

110. 谈谈 js 中的垃圾回收机制

主要有以下两种方式:

(1) 标记清除 (mark and sweep):

大部分浏览器以此方式进行垃圾回收, 当变量进入执行环境 (函数中

声明变量)的时候,垃圾回收器将其标记为“进入环境”,当变量离开环境的时候(函数执行结束)将其标记为“离开环境”,在离开环境之后还有的变量则是需要被删除的变量。标记方式不定,可以是某个特殊位的反转或维护一个列表等.垃圾收集器给内存中的所有变量都加上标记,然后去掉环境中的变量以及被环境中的变量引用的变量的标记。在此之后再被加上的标记的变量即为需要回收的变量,因为环境中的变量已经无法访问到这些变量。

## (2)引用计数(reference counting):

这种方式常常会引起内存泄漏,低版本的IE使用这种方式。机制就是跟踪一个值的引用次数,当声明一个变量并将一个引用类型赋值给该变量时该值引用次数加1,当这个变量指向其他一个时该值的引用次数便减一。当该值引用次数为0时就会被回收。

### 111. Js 是单线程还是多线程?

单线程,单位时间内只能处理一个进程

### 112. Js 是如何实现异步操作的?

Js 虽然是单线程的,但是浏览器是多线程的,js 中的异步操作基本都是由浏览器提供的子线程来完成的.

### 113. 分别介绍下 MVC, MVVM, MVP 这三种设计模式?

MVC 是后端语言的一种设计模式,主要是实现对代码分层, M(model)

数据模型层, 主要负责操作数据库; V(view)视图层, 主要负责进行界面展示, 可以认为前端的 html,css,js 充当的就是视图层; C(controller)业务控制层, 主要负责控制具体的业务逻辑, 负责将 model 数据层的数据交给 view 视图层进行展示.

MVVM 是前端的一种设计模式, vue 就是基于这种模式来设计的, 是从 MVC 演变过来的. M(model)数据层, 主要负责数据和方法的初始化; V(view)视图层, 可以认为 html,css 充当的就是视图层的角色; VM(view model)视图模型层, 负责连接数据层和视图层, 将数据层的数据交给视图层展示, 将视图层的行为传递给数据层.

MVP 也是从后端的 MVC 设置模式中演化过来的, 主要应用于安卓开发中. M(model) 数据层, V(view) UI 逻辑; P(Presenter)业务逻辑

## ES6/7/8 新特性

### 114. Es6 中新增了那些数据类型?

Symbol 类型 (基本)

Set 类型 (复杂)

Map 类型 (复杂)

WeakSet 类型 (复杂)

WeakMap 类型 (复杂)

TypedArray 类型 (复杂)

### 115. ES6 新增了那些特性?



`const`(声明常量), `let`(声明变量)关键字;

`map` 和 `set` 数据类型;

模板字符串;

对象数组解构赋值;

函数剩余参数;`(...arg)`

延展运算符;`(...)`

函数默认参数;`fn(name='zs')`

对象字面量的增强(属性名和属性值相同, 可缺省);

`Promise` 异步对象;

`class` 类的支持

116. 使用 `let` 声明的变量和 `var` 声明的变量有什么区别?

使用 `let` 声明的变量有块级作用域, 并且没有变量的声明提升( 使用 `let` 声明的变量在声明之前调用会报语法错误); 使用 `var` 声明的变量有声明提升(在声明之前调用会报 `undefined`), 没有块级作用域.

117. 谈谈 `async/await` 的使用方式和场景

`async` 是用来修饰函数的声明, 使用 `async` 修饰的函数会变成一个异步函数. `await` 用来修饰函数的调用, 被 `await` 修饰的函数必须返回一个 `promise` 异步对象, 使用 `await` 修饰后, 就会将 `promise` 异步对象转换成一个同步操作.

### 118. 箭头函数有什么作用及实际应用场景?

箭头函数可以使函数内部的 `this` 指向和函数外部保持一致; 箭头函数之所以可以让函数内部的 `this` 指向和外部保持一致是因为箭头函数内部没有 `this` 指向. 可以在 `ajax` 的回调函数中使用箭头函数让回调函数中的 `this` 指向事件源; 可以在定时器的第二个参数中使用箭头函数, 避免函数内部的 `this` 指向全局 `window`.

### 119. class 类的如何实现继承

使用 `extends` 关键字实现继承

```
class Person{
  constructor(props){
    this.name=props.name;
    this.age=props.age;
  }
}
// Student 继承 Person
class Student extends Person{
  constructor(props){
    super(props);
    this.score=props.score;
  }
}
```

### 120. 谈谈对 Promise 的理解

`Promise` 本身并没有提供任何的代码逻辑, 它只是帮助我们来改造代码结构, 最显著的一个特点就是通过 `Promise` 可以解决传统的回调地狱. 代码层面 `Promise` 提供了一个构造函数, 在使用的时候必须通过 `new` 创建一个实例对象, 在创建实例对象的时候需要传递一个匿名函数,

这个匿名函数需要两个参数(resolve,reject), resolve 成功处理函数, reject 失败处理函数. 什么时候触发成功处理函数和失败处理函数, 由具体的业务逻辑来决定. resolve 和 reject 需要通过 Promise 实例对象提供的 then 方法来传递.Promise 提供了两个静态方法 all,race,all 可以一次执行多个 Promise 实例, 返回值是数组; race 也可以一次执行多个 Promise 实例, 哪个实例最先执行完, 就返回哪个的执行结果.

## 前端框架

### Vue

121. vue 中如何封装一个组件

首先定义一个后缀名为.vue 的文件.文件内部还是三部分组成, template 模板部分, script 逻辑部分, style 样式部分. 这三部分是组件的核心部分, 组件需要哪些结构, 在模板部分书写, 需要什么样的外观样式, 通过 style 部分书写, 有哪些行为在 script 部分书写.一定要在 script 部分使用 es6 模块化的导出语法(`export default {}`), 进行导出. 然后在需要调用组件的地方使用 es6 模块化导入语法导入即可, 组件需要哪些参数, 直接在调用的部分进行传递即可.主要逻辑还是在组件中完成.

122. Vue 中 computed 和 watch 的区别?

computed 是计算属性, 可以根据 data 中的数据成员,动态计算出一个

新的数据成员(这个数据成员在 `data` 中并不存在), 计算属性的函数必须有返回值; `watch` 是监视器, 可以监视 `data` 中某一个数据成员的改变或路由中的某些属性的改变, 可以根据这个改变, 做一些其他操作 (不仅仅局限于更新其他相关数据).

Watch 监听器:

```
var vm = new Vue({
  el: '#demo',
  data: {
    firstName: 'Foo',
    lastName: 'Bar',
    fullName: 'Foo Bar'
  },
  watch: {
    firstName: function (val) {
      this.fullName = val + ' ' + this.lastName
    },
    lastName: function (val) {
      this.fullName = this.firstName + ' ' + val
    }
  }
})
```

计算属性:

```
var vm = new Vue({
  el: '#demo',
  data: {
    firstName: 'Foo',
    lastName: 'Bar'
  },
  computed: {
    fullName: function () {
      return this.firstName + ' ' + this.lastName
    }
  }
})
```

123. 谈谈对 vue 中插槽的理解?

Vue 中的插槽分为三种, 匿名插槽, 具名插槽, 作用域插槽.

通过插槽可以动态指定某一个组件模板部分的渲染, 我们在调用组件的时候, 在组件的调用标签中间传递了什么样的标签结构, 那么该组件就会把我们传递的标签结构放在他的模板部分进行渲染.

124. `v-show` 和 `v-if` 在隐藏一个元素的时候有什么不同, 应该如何来选择?

`v-show` 是通过 `css` 的方式来隐藏元素, 而 `v-if` 是根据条件是否成立决定是否要创建元素. 如果某个元素需要频繁切换显示状态的话, 建议使用 `v-show`, 因为频繁创建销毁 `DOM` 需要性能开销.

125. 什么是 `Vuex`, 在那种场景下使用?

`Vuex` 是针对 `vue` 的一个状态管理工具. 有几个核心的部分:

`state` 存储状态数据;

`mutations`: 更新数据的方法,

`actions`: 调用 `mutations` 中的方法, 更新 `state` 数据;

`getters`: 对 `state` 中的数据进行预处理

当组件的关系比较复杂的时候, 可以使用 `vuex` 简化组件间的传值.

126. 说说 `Vue` 路由的使用步骤?

第一步: 下载路由模块 `vue-router`;

第二步: 创建路由对象;

第三步:配置路由规则;

第四步:将路由对象注册为 vue 实例对象的成员属性

127. 你所了解到的常见 Vue 组件库有哪些?

PC 端组件库: element-ui, ant-design, iview

移动端: mint-ui, vant, vux

128. 谈谈对于 MVVM 的理解?

MVVM 由三部分组成 M(model 数据层), V(view 视图层), VM(view-model)

视图模型层. 是一种框架的设计思想, vue 就是基于 mvvm 来设计的.

其中 M(model) 层是负责初始化数据, V(view) 只负责页面展示, VM(view-model) 用来连接 view 层和 model 层, 将数据层的数据传递一个视图层进行展示, 将视图层的操作传递到数据层进行持久化.

129. Vue 的生命周期?

创建阶段: 只执行一次

beforeCreate(开始进行一些数据和方法的初始化的操作, data 中的数据和 methods 中的方法还不能用),

created(已经完成数据和方法的初始化, data 中的数据和 methods 中的方法可以使用了),

beforeMount(开始渲染虚拟 DOM),

mounted(已经完成了虚拟 DOM 的渲染, 可以操作 DOM 了, 只执行

一次)

运行阶段: 执行多次

beforeUpdate(data 中的数据即将被更新, 会执行多次)

updated(data 中的数据已经更新完毕, 会执行多次)

销毁阶段: 只执行一次

beforeDestroy(vue 实例即将销毁, 此时 data 中的数据和 methods 中的方法依然处于可用状态)

destroyed(vue 实例已经销毁, 此时 data 中的数据和 methods 中的方法已经不可用)

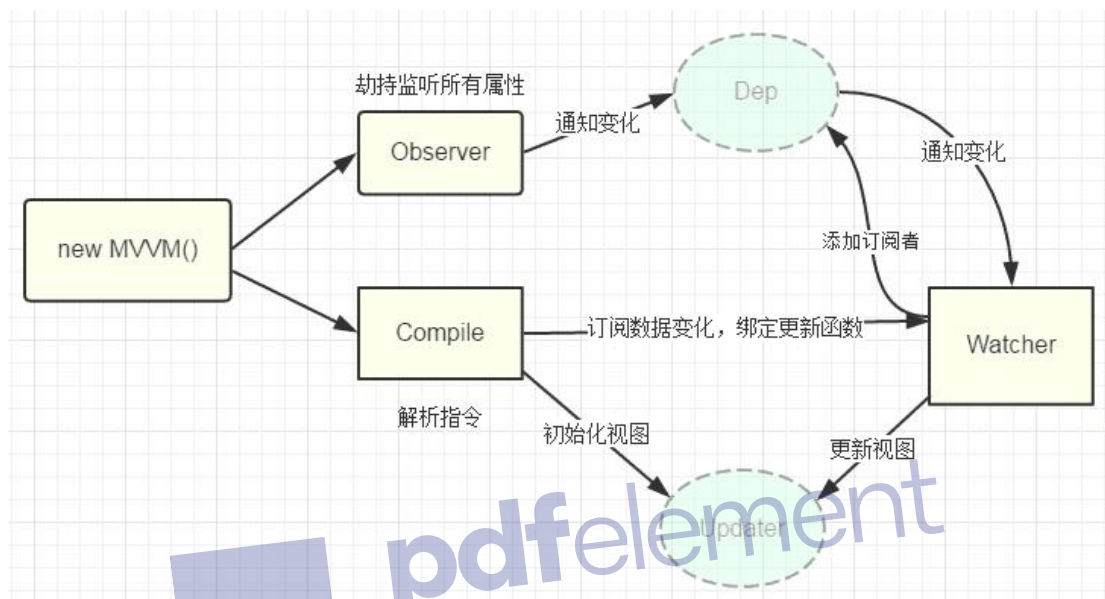
130. Vue 实现数据双向绑定的原理?

Vue 是使用数据劫持, 结合发布者订阅者模式实现双向数据绑定的

```
// 初始值
var data = {
  msg: 'hello',
  title: '标题'
}
for (var key in data) {
  Object.defineProperty(window, key, {
    get: function () {
      // console.log('get 被触发了');
      return data[key];
    },
    set: function (input) {
      // do something: 如通知界面层更新
      console.log(input, '假装在通知界面');
    }
  });
}
title = 'title 初始值';
console.log(title);
```

执行过程分析:

读取 title 或者 msg 的时候 get 方法会自动触发; 重新给 title 或 msg 赋值的时候,set 方法会被自动触发(可以在此处通知界面层更新)



131. Vue 创建组件的时候,data 为什么要使用匿名函数 return 一个对象?

因为对象是一种引用数据类型,在内存中只有一份. 如果 data 的值直接是一个对象的话, 那么后期组件在不同的地方多次调用的时候, 会相互产生影响, 因为每一次调用操作的 data 对象是一样的. 使用函数的方式返回对象, 可以保证组件的每一次调用都会创建一个新对象, 这样组件的每一次调用不会相互产生影响.

132. Vue 父组件向子组件传递参数?

首先在子组件的调用标签上声明一个自定义属性, 属性值来自父组件的 data; 然后在子组件的定义部分声明一个属性 props, 值是一个



数组, 将自定义属性的名字在 `props` 中进行声明; 在子组件的模板部分可以使用 `props` 中声明过的数据.

```
// 父组件
Vue.component('parent',{
  template:`
    <div>
      <child :msgFromParent="msg"></child>
    </div>`,
  data:function(){
    return {
      title:'父组件的标题',
      msg:'这是传给子组件的值'
    }
  },
// 子组件
  components:{
    child:{
      // 模板
      template:`
        <div>
          <h1>{{msgFromParent}}</h1>
        </div>`,
      // 数据模型
      data:function(){
        return {
          title:'子组件标题'
        }
      },
      //获取父组件中传过来的值
      props:['msgFromParent']
    }
  }
});
```

### 133. Vue 兄弟组件间的参数传递?

```
// 公共组件
```

```
var comm=new Vue();
// 兄弟组件 1

Vue.component('coma',{
  template:`
    <div>
      <h1>{{title}}</h1>
      <button @click="sendMsg">发送</button>
    </div>
  `,
  data:function(){
    return {
      title:'组件 A',
      msg:'传给兄弟组件的值'
    }
  },
  methods:{
    sendMsg:function(){
      comm.$emit('myevent',this.msg)
    }
  },
  created:function(){
    // 注册自定义事件
    comm.$emit('myevent',this.msg);
    console.log('注册事件 OK');
  }
});
// 兄弟组件 2
Vue.component('comb',{
  template:`
    <div>
      <h1>{{title}}</h1>
      <h1>{{msg}}</h1>
    </div>
  `,
  data:function(){
    return {
      title:'组件 B',
      msg:''
    }
  },
  mounted:function(){
    comm.$on('myevent',(data)=>{
      this.msg=data
    })
  }
});
```

```

        console.log(data);
    });
}
});

```

#### 134. Vue 中有几种路由模式?

Vue 中的路由模式有两种:hash,history; 默认是 hash 模式; 可以在创建路由对象的时候, 使用 **mode** 属性来切换路由模式.

```
const router=new Router({mode:'history'})
```

#### 135. Vue 路由导航守卫是什么, 以及应用场景

路由守卫是在页面进行路由跳转的时候做一些处理, 比如拦截.

vue-router 中提供了下面几种路由导航守卫:

全局前置守卫

```

const router = new VueRouter({ ... })

router.beforeEach((to, from, next) => {
  // from 从里来
  // to 到哪里去
  // next 是否要放行
})

```

全局后置钩子

```

router.afterEach((to, from) => {
  // ...
})

```

路由独享守卫: 在声明路由的时候, 针对特定路由的构造函数

```
const router = new VueRouter({
  routes: [
    {
      path: '/foo',
      component: Foo,
      beforeEnter: (to, from, next) => {
        // ...
      }
    }
  ]
})
```

136. Vue 如何自定义一个过滤器？

定义全局过滤器：

`Vue.filter('过滤器名称',function(input){ return input });`

定义局部过滤器：

```
new Vue({
  filters:{
    dateFmt:function(input){
      return 'yyyy-mm-dd';
    }
  }
})
```

137. Vue 如何自定义一个 vue 指令？

定义全局指令：

`Vue.directive('指令名',function(el,binding){});`

定义私有指令：

```
new Vue({
  directives:{
```

```

        focus:function(el, binding){
        }
    }
});

```

138. 怎么定义 vue-router 的动态路由? 怎么获取通过路由传过来的参数?

```

const router=new VueRouter({
  routes:[
    {path:'/product/:id',component:Product}
  ]
});
// 获取参数:
this.$route.params.id

```

139. Vue 路由模块中\$route 和\$router 的区别?

\$route 中存储的是跟路由相关的属性(如\$route.params,\$route.query);  
\$router 中存储的是和路由相关的方法(如\$router.push(),\$router.go()),

140. vue 中 v-for 指令循环遍历中 key 属性的作用?

Key 属性的作用是在数据层和视图层之间建立一一对应关系, 方便后期对页面进行局部更新. 如果某一条数据发生改变, 只更新当前数据对应的 DOM 元素.

141. Vue 和 react 有哪些不同的地方?

Vue 实现了双向数据绑定(数据<=>界面);react 仅仅实现了单项数据流(数据层=>界面层); vue 中提供了指令, react 中没有指令的概念. vue 中

使用插值表达式在进行数据渲染, react 中使用 jsx 进行数据的渲染.

142. Vue 有哪些常用的事件修饰符?

.prevent: 阻止默认事件;

.stop: 阻止冒泡;

.once: 事件执行一次;

.self: 只当在 `event.target` 是当前元素自身时触发处理函数

143. 列举 Vue 中常用的指令

v-model: 实现双向数据绑定;

v-bind: 绑定属性;

v-on: 注册事件;

v-class: 动态绑定类名

v-style: 设置行内样式

v-html: 设置标签内容(允许内容 html)

v-text: 设置标签的内容(不允许包含 html)

v-clack: 解决插值表达式闪烁问题

144. Vue 中如何解决插值表达式闪烁问题?

使用 v-html 或 v-text 替代插值表达式;

使用 v-clack 解决插值表达式闪烁,

第一步: 声明属性选择器[v-clack]{display:none}

第二步:在插值表达式所在标签添加属性 v-clack

145. Vue 路由中如何实现通过锚点值的改变切换组件?

通过监听 hashchange 事件, 具体如下:

```
window.addEventListener('hashchange',function(){
  console.log('hash change');
});
```

146. vue 中如何实现给样式添加作用域?说明其实现原理

vue 中要给样式添加作用域, 只需要给 style 标签添加 scoped 属性即可.

实现原理:

添加了 scoped 属性的 style 标签内的样式会被改写成一个交集选择器, 会在原来类名的基础上添加一个随机属性(如.container[v-abcde]), 同时引用该类名的标签也会添加一个相同的属性(如 <div class="container" v-abcde></div>), 这样的话, 这个类名就可以对引用它的标签生效, 同时不会影响其他同类名的标签.

147. Vue 中如何动态添加一个路由规则?

使用 router.addRoutes([{'path:',component:''}])

148. Vue 中有何优化页面的加载效率?

使用路由懒加载; 不要打包一些公共的依赖(vue, 组件库), 使用 CDN

加载这些依赖文件

149. 什么是路由懒加载？路由懒加载有什么好处？如何实现路由懒加载？

路由懒加载是指通过异步的方式来加载对应的路由组件(默认情况是将所有的组件全部加载并打包).

路由懒加载的好处: 可以提高也的加载速度, 尤其是首页的加载速度 (因为使用了懒加载后, 加载首页的时候, 就不需要加载其他页面对应的组件, 在需要的时候在加载)

具体实现:

```
import Vue from 'vue';
import Router from 'vue-router';
// 异步导入组件
const List = resolve => require(['@/components/list'], resolve);
const Detail = resolve => require(['@/components/detail'], resolve);
const Message = resolve => require(['@/components/message'], resolve);
Vue.use(Router);
export default new Router({
  // 路由规则
  routes:[
    {path:'/list',component:List},
    {path:'/detail',component:Detail},
    {path:'/message',component:Message},
  ],
  // 路由模式: 默认 hash, 可选值 hash(如#/index)|history(/index)
  mode:'history'
});
```

150. Vue 中如何触发一个自定义事件？

通过 this.\$emit(event, '数据') 可以触发自定义事件的执行.



151. Vue 中如何监听自定义事件的执行?

通过 `this.$on(event,callback)`可以监听自定义事件的执行

152. Vue 中如何移除自定义事件?

通过 `this.$off(event,callback)`可以移除一个自定义事件; 如果某些特殊场景下,一个事件被触发一次后就需要将其移除, 可以使用 `this.$once(event,callback)`进行事件注册

153. `vm.$mount(selector)`方法的作用是什么?

手动将一个 vue 实例挂载到页面上

```
var MyComponent = Vue.extend({
  template: '<div>Hello!</div>'
})
// 创建并挂载到 #app (会替换 #app)
new MyComponent().$mount('#app')
// 同上
new MyComponent({ el: '#app' })
```

154. Vue 中如何手动销毁一个 vue 实例?

调用 `vm.$destroy()`可销毁一个 vue 实例(清理它与其它实例的连接, 解绑它的全部指令及事件监听器)

155. Vue 中有哪些内置的组件?

component, slot, transition, transition-group, keep-alive

```

<!-- 动态组件由 vm 实例的属性值 `componentId` 控制 -->
<component :is="componentId"></component>

<!--transition 动画组件的使用-->
<transition>
  <div v-if="ok">toggled content</div>
</transition>

```

## 156. vue 实例中有哪些属性?

vm.\$data 可以获取 vm 实例对象 data 中的数据;

vm.\$props 可以获取 vm 组件接收到的 props 对象数据;

vm.\$el 可以获取 vm 实例对象的根 dom 元素;

```

var MyComponent = Vue.extend({
  template: '<div>Hello!</div>'
})
// 在文档之外渲染并且随后挂载
var vm = new MyComponent().$mount()
document.getElementById('app').appendChild(vm.$el)

```

vm.\$refs 可以获取 vm 实例中注册过 ref 特性的所有 dom 元素和组件实例.

## 157. Vue.use(plugin)的作用是什么, 使用的时候需要注意什么问题?

Vue.use 的作用是安装一个 Vue 插件, 该方法需要在调用 new Vue() 之前被调用

## 158. vm.\$nextTick(fn)的作用是什么?

延迟某个操作的执行, 直到 dom 更新以后在执行.

```

<body>

```

```
<div id="app"><h1 ref="h1">{{title}}</h1></div>
</body>
<script>
var vm = new Vue({
  el: '#app',
  data: {
    title: '这是一个标题'
  }
  created: function () {
    this.$nextTick(() => {
      // 在 created 里直接操作 ref 会报错
      this.$refs.h1.innerHTML = '这是更新以后的标题';
    });
  }
});
</script>
```

159. 什么是 ssr? 如何实现 ssr?

ssr 是全拼(server side rendering), 中文意思, 服务端渲染, 让页面的渲染在服务端完成, 生产环境必须部署 nodeJS 的环境, 因为服务端渲染必须借助 nodeJS 来完成. vue 中可以使用 nuxt 框架实现服务端渲染.

160. 什么是 SPA?

SPA(Single Page Application), 单页面应用程序, 使用 vue, react, angular 创建的项目都属于 SPA. 因为整个项目只有一个页面, 其他页面都是在该页面的基础上局部刷新而来的.

传统方式创建的项目都是 MPA(Multiple Page Application)多页面应用程序.

161. 使用 vue,react,angular 开发的 SPA 单页面应用有什么优缺点?

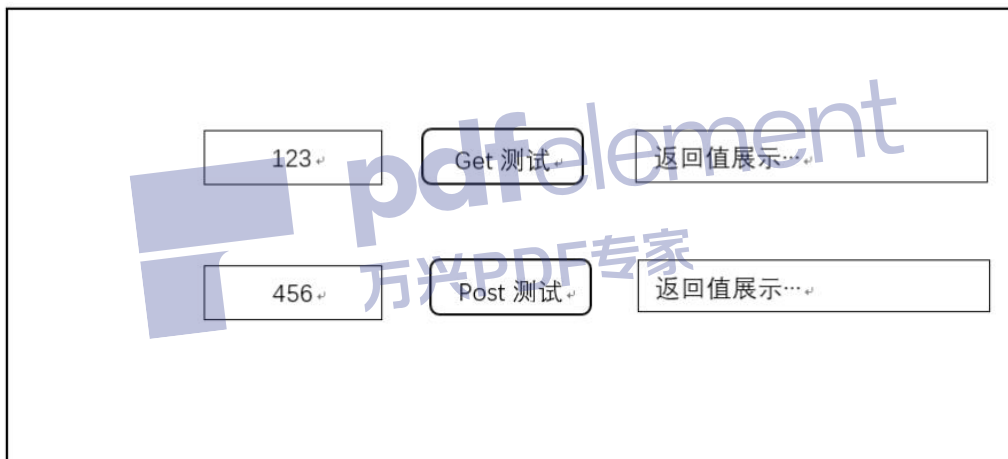
单页面应用虽然性能方面得到了提升,但是有一个致命的缺点就是不利于 seo, 搜索引擎几乎不会抓取单页面应用.

162. 某某公司 vue 机试题

实现下面两个接口请求

1. <http://47.96.26.207:8099/api/Users/GetTest>  
参数: phone, 可测试值 123
2. <http://47.96.26.207:8099/api/Users/PostTest>  
参数: phone, 可测试值 456

界面



提示: 1.下载 postman 先测试接口  
2.用 Vue 实现

Html 部分

```
<div id="app">
  <div>
    <input type="text" v-model="phone1">
    <button @click="getHandle">get 测试</button>
    <span>{{res1}}</span>
  </div>
  <div>
    <input type="text" v-model="phone2">
    <button @click="postHandle">post 测试</button>
    <span>{{res2}}</span>
  </div>
</div>
```

```
</div>
</div>
```

## Js 部分

```
new Vue({
  el: '#app',
  data: {
    phone1: 123,
    phone2: 456,
    res1: '',
    res2: '',
    baseUrl: 'http://47.96.26.207:8099/api/Users'
  },
  methods: {
    getHandle() {
      axios.get(`${this.baseUrl}/GetTest?phone=${this.phone1}`).then
      ((data)=>{
        this.res1=data.result;
      });
    },
    postHandle() {
      axios.post(`${this.baseUrl}/PostTest`,{phone:this.phone2}).then
      (({data})=>{
        this.res2=data.result;
      });
    }
  }
});
```

## React

163. 什么是虚拟 DOM, 使用虚拟 DOM 有什么优势?

虚拟 dom 相当于在 js 和真实 dom 中间加了一个缓存, 利用 dom diff 算法避免了没有必要的 dom 操作, 从而提高性能。

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。

#### 164. 简单介绍下 react 中的 diff 算法?

把树形结构按照层级分解，只比较同级元素。

给列表结构的每个单元添加唯一的 key 属性，方便比较。

React 只会匹配相同 class 的 component（这里面的 class 指的是组件的名字）

合并操作，调用 component 的 setState 方法的时候，React 将其标记为 dirty。到每一个事件循环结束，React 检查所有标记 dirty 的 component 重新绘制。

选择性子树渲染。开发人员可以重写 shouldComponentUpdate 提高 diff 的性能。

```
// 创建对比函数
function updateChildren(vnode, newVnode) {
  var children = vnode.children || []
  var newChildren = newVnode.children || []

  children.forEach(function (childrenVnode, index) {
    // 首先拿到对应新的节点
    var newChildVnode = newChildren[index]
    // 判断节点是否相同
    if (childrenVnode.tag === newChildVnode.tag) {
      // 如果相同执行递归，深度对比节点
      updateChildren(childrenVnode, newChildVnode)
    } else {
      // 如果不同则将旧的节点替换成新的节点
      replaceNode(childrenVnode, newChildVnode)
    }
  })
}
```

```
    }  
  })  
}  
  
// 节点替换函数  
function replaceNode(vnode, newVnode) {  
  var elem = vnode.elem  
  var newEle = createElement(newVnode)  
}
```

### 165. 什么是 Redux?

Redux 是一个状态管理工具, 不仅可以在 react 中使用, 也可在其他框架中使用, 甚至可以脱离框架本身使用. redux 中有几个核心的组成部分:

store 存储数据的对象, 必须通过 createStore 方法来创建;

action 更新数据的规则, 必须有一个属性 state, 值必须是字符串;

reducer 更新数据的函数, 需要传入 state 状态数据和 action 更新规则.

在 react 中使用 redux 的时候, 一般会使用 react-redux 来简化使用步骤.

### 166. React 有哪些常用的组件库?

PC 端组件库:

element-ui(饿了么推出的前端组件库), ant-design(阿里巴巴的前端组件库, 几乎覆盖了三大框架); zent(有赞推出的 PC 端组件库)

### 167. React 中如何操作 DOM?

在需要进行 dom 操作的标签上设置一个 `ref` 属性, 保证值不要重复, 后期在 js 部分可以通过 `this.refs.属性名` 来获取标签的虚拟 dom 对象.

### 168. 什么是高阶组件?

函数的返回值是一个函数, 我们称之为高阶函数. 同理一个组件返回值如果还是一个组件, 那么就称之为高阶组件. `redux` 中提供的 `connect` 就是一个高阶组件.

### 169. React 中调用 `setState` 后发生了什么?

在代码中调用 `setState` 函数之后, `React` 会将传入的参数对象与组件当前的状态合并, 然后触发所谓的调和过程 (Reconciliation)。经过调和过程, `React` 会以相对高效的方式根据新的状态构建 `React` 元素树并且着手重新渲染整个 UI 界面。在 `React` 得到元素树之后, `React` 会自动计算出新的树与老树的节点差异, 然后根据差异对界面进行最小化重渲染。在差异计算算法中, `React` 能够相对精确地知道哪些位置发生了改变以及应该如何改变, 这就保证了按需更新, 而不是全部重新渲染。

### 170. React 中状态 `state` 和属性 `props` 有什么不同?

`state` 是组件的私有数据, 可读可写, `props` 是只读属性, 一般来自外部 (比如父组件)



### 171. React 中有几种创建组件的方式?

通过函数的方式创建组件, 此种方式创建的组件为无状态组件(不常用);

`React.createClass();`

通过 `class` 类的方式创建组件(须继承 `React.Component`), 实际开发中使用此种方式.

### 172. React 中的组件按照职责不同, 可以分为几种类型?

根据组件的职责通常把组件分为 UI 组件和容器组件;

UI 组件负责 UI 的呈现, 容器组件负责管理数据和逻辑;

两者通过 `React-Redux` 提供 `connect` 方法联系起来.

### 173. 类组件(Class component)和函数式组件(Functional component)之间有何不同?

类组件不仅允许你使用更多额外的功能, 如组件自身的状态和生命周期钩子, 也能使组件直接访问 `store` 并维持状态; 当组件仅是接收 `props`, 并将组件自身渲染到页面时, 该组件就是一个 '无状态组件 (stateless component)', 可以使用一个纯函数来创建这样的组件。这种组件也被称为哑组件(dumb components)或展示组件.

### 174. 说说 react 的生命周期函数?

初始化阶段:

`getDefaultProps`:获取实例的默认属性

`getInitialState`:获取每个实例的初始化状态

`componentWillMount`: 组件即将被装载、渲染到页面上

`render`:组件在这里生成虚拟的 DOM 节点

`componentDidMount`:组件真正在被装载之后

运行中状态:

`componentWillReceiveProps`:组件将要接收到属性的时候调用

`shouldComponentUpdate`:组件接受到新属性或者新状态的时候（可以返回 `false`，接收数据后不更新，阻止 `render` 调用，后面的函数不会被继续执行了）

`componentWillUpdate`:组件即将更新不能修改属性和状态

`render`:组件重新描绘

`componentDidUpdate`:组件已经更新

销毁阶段:

`componentWillUnmount`:组件即将销毁

175. react 性能优化可以使用哪个生命周期函数？

`shouldComponentUpdate` 这个方法用来判断是否需要调用 `render` 方法重新描绘 dom。因为 dom 的描绘非常消耗性能，如果我们能在 `shouldComponentUpdate` 方法中能够写出更优化的 dom diff 算法，可以极大的提高性能。

176. 应该在 React 组件的何处发起 Ajax 请求?

在 React 组件中, 应该在 `componentDidMount` 中发起网络请求。这个方法会在组件第一次“挂载”(被添加到 DOM)时执行, 在组件的生命周期中仅会执行一次。更重要的是, 你不能保证在组件挂载之前 Ajax 请求已经完成, 如果是这样, 也就意味着你将尝试在一个未挂载的组件上调用 `setState`, 这将不起作用。在 `componentDidMount` 中发起网络请求将保证这有一个组件可以更新了。

177. React 如何实现服务端渲染?

Next.js 是一个轻量级的 React 服务端渲染应用框架。

## 前端构建工具

178. 你了解到前端有哪些项目构建工具?

webpack, gulp, grunt

179. webpack 和 gulp 在进行代码合并的时候还有什么不同?

webpack 是基于 commonjs 模块化规范进行代码合并, 而 gulp 只是简单的代码合并。

180. 列举几个 gulp 中常用的插件?

合并文件: `gulp-concat`;

压缩 js: `gulp-uglify`;

文件重命名: `gulp-rename`;

压缩 css: `gulp-minify-css`

压缩 html: `gulp-hmltmin`

启动开发服务器: `gulp-connect`

拷贝文件: `gulp-copy`

## 微信小程序

181. 微信小程序目前有几类主流的开发模式?

微信官方:

原生方式;

`wepy`(微信官方推出的开发框架, 为了迎合目前主流前端框架的语法和提高开发效率);

第三方公司:

`mpvue`(使用 `vue` 的语法开发小程序, 是一个阉割版的 `vue`)

`uni-app`(使用 `vue` 的语法进行开发, 是一个阉割版的 `vue`, 据说一套代码可以编译出运行在多个平台的应用)

182. 简单介绍微信小程序的开发过程

首先得注册以为微信小程序, 因为小程序开发过程中需要一个 `appid`;

其次下载腾讯官方的开发者工具(开发者工具必须使用个人微信登录),

小程序只能运行在开发这工具或者微信应用内部;

创建应用, 填入申请的 **appid**, 即可快速生成小程序的项目结构.

### 183. 微信小程序中的 **tabbar** 导航如何制作

小程序中的 **tabbar** 底部导航是配置出来的, 只需要在应用配置文件中添加一个 **tabbar** 配置阶段, 按照官方文档配置即可, **tabbar** 数量至少 2 个, 最多 5 个.

### 184. 微信小程序中如何实现页面跳转?

小程序页面跳转:

使用组件 `<navigator url="../home/home">目标页面</navigator>`

使用 `api: wx.navigateTo({url:'../home/home'})`

**Tabbar** 页面跳转:

`wx.switchTabbar({url:'../index/index'})`

使用组件 `<navigator url="../home/home" open-type="switchTab">目标页面</navigator>`

### 185. 简单描述微信小程序的生命周期?

小程序的生命周期分为应用生命周期和页面生命周期.

应用生命周期:

**onLaunch**: 应用启动, 只执行一次;

**onShow**: 应用切换到前台;

onHide: 应用切换到后台模式;

noError: 运行阶段出现错误;

onPageNotFound: 找不到页面

页面生命周期:

onLoad: 页面开始加载;

onReady: 页面加载完毕;

onShow: 页面进入焦点状态;

onHide: 页面进入后台状态

186. 微信小程序中如何请求数据接口?

通过 wx.request

```
wx.request({
  url: 'test.php', //仅为示例，并非真实的接口地址
  data: {
    x: '',
    y: ''
  },
  header: {
    'content-type': 'application/json' // 默认值
  },
  success(res) {
    console.log(res.data)
  }
})
```

187. 如何优化小程序代码包的体积?

分包加载,使用分包加载可以让小程序代码包体积达到 8M(最多四个

分包,每个分包最大 2M);将资源文件尽量放在远程服务器端.

188. 你了解到微信小程序有哪些组件库?

Vant Weapp, iview Weapp, minui, WeUI, iview-mpvue

## TypeScript

189. Typescript 中有哪些数据类型?

number(数值), string(字符型), boolean(布尔), object(对象),  
undefined(未定义), null(空性), any(任意类型)

190. Typescript 和 javascript 的区别?

Typescript 是 javascript 的一个超级, 是 es6 的实现, 支持所有 es6 的语法, Typescript 只是在开发过程中编写的一种中间语言, 浏览器并不能直接解析 Typescript, 上线以后需要将 Typescript 转换成 javascript.

## 代码版本控制工具

191. git 中有哪些常用的命令?

初始化仓库: `git init`

添加暂存区: `git add 文件名`

提交到本地仓库: `git commit -m '注释'`

推送到远程仓库: `git push 仓库地址 分支名称`

拉取远程仓库代码: `git pull`

克隆仓库: `git clone` 仓库地址

创建分支: `git branch` 分支名称

切换分支: `git checkout` 分支名称

查看分支: `git branch`

合并分支: `git merge` 分支名称

查看日志: `git log` (`git log --oneline`)

查看所有日志: `git reflog`

版本回退: `git reset --hard` 版本号 (commit-id 可以通过 `git log --oneline` 获取)

192. `git` 和 `svn` 有什么不同?

`git` 是一个分布式仓库管理系统, 每个人本地都有一个本地仓库, `svn` 是一个集中式仓库管理系统, 仓库只有一个. `svn` 一般需要服务端给每个人分配账号和密码, `git` 是使用 `ssh` 公钥/私钥来区分不同程序员的.

## 服务器编程

### NodeJS

193. Nodejs 中有哪些常用内置模块?

`http`; `fs`(文件系统); `path`(路径); `querystring`(查询字符串); `url`(解析 url)



#### 194. 常用第三方模块

express; mysql(操作 mysql); cookie-parser(解析 cookie 的中间件);  
express-session(处理 session); crypt(加密模块)

#### 195. Express 框架如何快速创建一个服务器?

```
// 引入 express
const express=require('express')
// 创建服务器
const app=express();
// 监听路由
app.get('/',(req,res)=>{
  res.send('server running');
})
// 启动服务器
app.listen('3000',()=>{
  console.log('server running at http://localhost:3000')
});
```

#### 196. Express 中路由模块的使用步骤

```
var express = require('express')
// 1. 创建一个路由容器
var router = express.Router()
router.get('/get', function (req, res) {
  res.send('请求了/get')
})
router.get('/list', function (req, res) {
  res.send('请求了 /list')
})
router.post('/add', function (req, res) {
  res.send('请求了 /add')
})
// 创建服务器
const app=express();
// 注册路由中间件
```

```
app.use(router)
// 启动服务器
app.listen('3000',()=>{
  console.log('server running at http://localhost:3000')
});
```

## 197. 使用 http 模块快速开启一个 web 服务器

```
var http = require('http');
http.createServer(function (request, response) {
  // 发送 HTTP 头部
  // HTTP 状态值: 200 : OK
  // 内容类型: text/plain
  response.writeHead(200, { 'Content-Type': 'text/plain' });
  // 发送响应数据 "Hello World"
  response.end('Hello World\n');
}).listen(8888);
```

## Hybird(混合) APP

### 198. 目前流行的混合 App 开发方式

Appcan, Hbuilder, ApiCloud, 这几中方式大同小异, 都是通过网页的方式开发 app, 然后通过开发工具将代码上传到远程服务器进行在线打包, 开发工具本身会提供一些基础的 api 来帮助我们操作移动终端.

### 199. 什么是 react-native?

React-native 是 facebook 公司基于 react 的语法设计的一个开发原生移动 app 的解决方案, 可以实现维护一套代码, 即可打包出可以运行在

安卓和苹果上的移动应用. 性能层面几乎和原生方式开发的应用接近.

## 200. 什么是 flutter?

Flutter 是谷歌的移动 UI 框架,可以快速在 iOS 和 Android 上构建高质量的原生用户界面。Flutter 可以与现有的代码一起工作。在全世界,Flutter 正在被越来越多的开发者和组织使用,并且 Flutter 是完全免费、开源的。

## 201. 什么是 weex?

Weex 是使用 vue 的语法来开发原生 app,可以使用一套代码,打包出运行在安卓和苹果两个操作系统上的移动应用,需要在本地配置打包环境,和 react-native 的工作原理类似. 性能层面几乎和原生方式开发的应用接近.

## 202. 你所了解到的产品原型和设计图的共享平台有哪些?

(产品原型和设计图的共享平台,可以帮助互联网团队更好地管理文档和设计图。可以实现在线展示网页原型,自动生成设计图标注,与团队共享设计图,展示页面之间的跳转关系,前端程序员可以从该平台上直接下载设计师设计好的设计图的切片,并且可以在线查看设计图标注信息)

常用产品原型共享平台有: 蓝湖, 墨刀, xiaopiu, Mockplus

## 实际开发相关

### 203. 商城的购物车是怎么实现的?

商城购物车一般会写在本地存储,比如 `cookie` 或者 `localStorage` 中,会采用数组格式的字符串来存储,主要会存储商品 `id`, 商品名称, 商品价格, 商品数量等信息(当然商品价格等敏感信息后期还是以后端为准, 此处存储只是为了方便在页面展示), 如果过要考虑兼容问题, 建议存储在 `cookie` 里(因为 `localStorage` 低版本的浏览器不支持).如果不考兼容问题, 使用 `localStorage` 性能会更好. 购物车可以在用户未登录的状态就可以添加, 也可以在用户登录以后再添加, 这个完全取决于具体业务场景.

### 204. 前后端分离的项目如何实现登录状态的保持?

前后端分离的项目一般会使用 `token` 实现登录状态保持.

`token` 其实就是一个随机字符串, 当用户在登录页面输入账号和密码后, 前端将账号密码发送给后端, 后端校验完账号和密码后, 会生成一个随机的不重复的字符串(即 `token`), 并将其响应给前端, 前端拿到该 `token` 后, 需要在客户端进行持久化(一般会写在 `localStorage` 或者 `sessionStorage` 中, 如果是 SPA 会存储在 `sessionStorage` 中, 如果是 MPA 则储存在 `localStorage` 中), 那么下次在向后端数据接口发送请求的时候, 一般需要将 `token` 一并发送给后端数据接口, 后端数据接口会对 `token` 进行校验, 如果合法则正常响应请求, 如果不合法, 则提

示未登录. 前端则根据本地存储中是否存在 token 判断用户是否处于登录状态.

## 205. 前后端分离开发的后台管理系统一般如何实现权限管理?

首先某个角色有哪些权限肯定是事先配置好的, 并且存储在了数据库里. 当某个账号登录后, 后端数据接口则需要返回该账号对应角色的权限列表, 前端则需要根据该权限列表, 动态渲染管理系统的导航, 当然后端数据接口也会做二次权限校验(当某一个角色请求一个自己没有的授权的数据接口或页面的时候, 后端数据接口也不会正常响应).

## 206. 如何通过最原始的方式实现一个多语言版的网站?

简单来说, 一般是通过配置语言包来实现的, 对于一个多语言的版的网页, 并不一定是网页上所有的地方都需要实现多语言, 可能只是某些关键的导航部分会实现多语言. 所以可以准备一个 json 配置文件, 结构大体如下(只是简单举例):

```
const lang={
  zh:{
    mainNavi:[
      "首页",
      "成功案例",
      "关于我们",
      "联系我们",
      "注册",
      "登录"
    ]
  },
  en:{
```

```

mainNavi:[
  "home",
  "succss cases",
  "about me",
  "concat us",
  "register",
  "login"
]
}
}

```

当用户选择语言后, 可以将用户选择的语言类型存储在本地存储中, 接着可以根据语言类型, 从该语言配置文件中获取对应的配置节点, 动态在页面上输出.

207. 如何实现从列表页跳入详情页, 并从详情页面返回列表页时, 滚动条还在当时的位置, 并且数据不刷新?

为了保证返回时, 我们还能回到刚才的位置, 那么就需要做本地数据存储。[考虑到保证数据和上次用户看到的一致]本地缓存的字段有页码、点击的位置（滚动条的位置），列表数据（具体字段，请自行结合实际）进行本地缓存。合理的使用本地缓存数据有 `cookie`、`sessionStorage`，至于为啥要用两个，可以自行查阅，着重看一下存储大小的限制问题当从详情返回列表时，首先从缓存中取页码、点击的位置（滚动条的位置），列表数据，如果有位置、列表数据，则直接将列表数据渲染到页面，并滚动到存储的位置点.

208. 微信小程序中如何实现页面后退时重新刷新页面?

在后退页面的 `onShow` 方法中手动调用一下 `onLoad` 方法, 代码如下

```
Page({
  onShow(){
    this.onLoad();
  }
});
```

209. Vue 开发中如果由于某种特殊原因数据更新之后, 页面层没有同步更新, 该如何处理?

调用 `vm.$forceUpdate()` 方法强制刷新页面, 该方法可以迫使 Vue 实例重新渲染.

210. Vue 中如何动态设置页面的 title 标题?

```
const router = new Router({
  routes: [
    { /* (首页) 默认路由地址 */
      path: '/',
      component: login,
      meta: {
        title: '首页入口'
      }
    },
    {
      path: '/apply',
      component: apply,
      meta: {
        title: '申请'
      }
    },
    { /* Not Found 路由, 必须是最后一个路由 */
      path: '*',
      component: NotFound,
      meta: {
        title: '找不到页面'
      }
    }
  ]
});
```

```

]
});
router.beforeEach((to, from, next) => {
  /* 路由发生变化修改页面 title */
  if (to.meta.title) {
    document.title = to.meta.title
  }
  next()
})

```

211. 在 vue 中,当切换到新路由时, 想要页面滚到顶部, 或者页面后退时,想保持原先的滚动位置, 该如何实现?

```

const router = new VueRouter({
  routes: [...],
  scrollBehavior (to, from, savedPosition) {
    // return 期望滚动到哪个的位置
    // 返回原来的位置
    // return savePosition
    // 返回页面顶部
    return { x: 0, y: 0 }
  }
})

```

scrollBehavior 方法接收 to 和 from 路由对象。第三个参数 savedPosition 当且仅当通过浏览器的前进/后退按钮触发时才可用。这个方法返回滚动位置的对象信息, 长这样: { x: number, y: number }

212. vue 项目在开发阶段如何解决跨域请求数据?

使用 vue-cli-3.0 版本创建的项目解决跨域请求数据:

第一步: 在项目根目录下创建 vue.config.js

第二步: 在 vue.config.js 中添加如下配置节点



```
module.exports = {  
  // 开发服务器配置节点  
  devServer: {  
    open: true,  
    host: 'localhost',  
    port: 8081,  
    https: false,  
    hotOnly: false,  
    proxy: { // 配置跨域  
      '/api': {  
        // 数据接口服务器地址  
        target: 'http://localhost:5001/api/',  
        ws: true,  
        changOrigin: true,  
        pathRewrite: {  
          '^/api': ''  
        }  
      }  
    }  
  }  
}
```

pdfelement  
万兴PDF专家