前端开发

整理一些有关前端的基础知识点 -- by Qzx 参考网址：前端开发面试题

HTML部分

- Doctype作用？标准模式与兼容模式各有什么区别?

```
1. 告知浏览器的解析器用什么文档标准来解析这个文档，若不存在或者错误，文档将以兼容模式呈现。
2. 标准模式以浏览器支持的最高标准运行，兼容模式中，页面以宽松的向后兼容的方式显示，防止老式浏览器无法工作。
```

- HTML5 为什么只需要写 <!DOCTYPE HTML>?

```
HTML5 不基于 SGML，不需要对DTD进行引用，只需要doctype来规范浏览器的行为。
```

- 行内元素有哪些？块级元素有哪些？ 空(void)元素有那些?

```
1. 行内元素有：a span img input select 等
2. 块级元素有：div ul ol li dl dt dd h1~h6 p 等
3. 常见的空元素有：<br> <hr> <img> <input> <link> <meta> <area>等
```

- 页面导入样式时，使用link和@import有什么区别?

```
1. link 属于XHTML标签，页面加载时，link同时被加载，除加载css,还用于定义RSS,定义rel连接属性。
2. @import是CSS提供，只用于加载css,且会等到页面被加载完再加载。
```

- html5有哪些新特性、移除了那些元素？如何处理HTML5新标签的浏览器兼容问题？如何区分 HTML 和 HTML5?

```
1. 新特性:
    (1) 语义化标签:header nav section article footer 等;
    (2) 表单控件:date time url email search 等;
    (3) 绘画 canvas 音频 video 视频 audio;
    (4) 本地离线存储:localStorage(长期存储数据,浏览器关闭后不丢失) 和 sessionStorage
(数据在浏览器关闭后自动删除);
    (5) 新的技术:websocker webworker 地图

2. 移除的元素:
    纯表现(可用css属性替代):big center font s tt u basefont strike;
    产生负面作用:frame frameset noframes;
3. 处理新标签兼容问题:document.createElement方法生成或用成熟的框架;
4. 区分:DOCTYPE声明  新增的结构元素 功能元素
```

- 简述一下你对HTML语义化的理解?

```
1. 用正确的标签做正确的事;
2. 让页面内容结构化,更清晰,便于浏览器、搜索引擎解析;
3. 即使在没有css样式情况下,也以一种文档格式显示,容易阅读;
4. 有利于搜索引擎的搜索;
5. 源代码更容易阅读分块,便于以后的人维护。
```

- 请描述一下 cookies,sessionStorage 和 localStorage 的区别?

```
1. 传输:
    Cookie是网站为标识用户身份而存储在本地终端上的数据,会在同源的http请求中携带(即使不需要
);
    localStorage和sessionStorage不会自动把数据发给服务器,仅在本地保存;
2. 存储大小:
    cookie数据大小不能超过 4k;
    sessionStorage 和 localStorage 也有存储大小限制,但可以达到5M或者更多;
3. 有效时间:
    cookie    设置cookie过期时间之前一直有效,即使窗口或浏览器关闭
    sessionStorage  数据在当前浏览器窗口关闭后自动删除
    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
```

- HTML5的离线储存怎么使用,工作原理能不能解释一下?

```
离线存储:在设备没有与因特网连接的时候,可以正常访问站点,连接后,更新设备的缓存文件;
原理:基于一个新建的 .appcache文件的缓存机制,通过解析文件上的离线存储清单,可以将资源像coo
kie一样被存储下来,当网络离线状态时,浏览器会通过读取离线存储的资源进行页面展示。
使用:
 1. 在页面<html>头部添加 manifest属性;
 2. 在cache.manifest文件编写离线存储的资源;
 3. 在离线状态时,操作 window.applicationCache 进行需求实现。
```

- 浏览器是怎么对HTML5的离线储存资源进行管理和加载的呢?

```
在线情况下：浏览器发现html头部有manifest属性，会请求 manifest文件，若是第一次访问app，浏览器会根据manifest文件的内容下载相应的资源并且进行离线存储。若已访问过app且资源已经离线存储，则浏览器会使用离线资源加载页面，然后浏览器会对比manifest的新旧文件的差异，若发生改变，则会重新下载文件中的资源并进行离线存储。
离线情况下：浏览器就直接使用离线存储的资源。
```

- iframe有那些缺点?

```
1. iframe会阻塞主页面的onload事件;
2. 搜索引擎无法解读这种页面，不利于SEO;
3. 与主页面共享连接池，影响页面的并行加载;
解决办法：通过JavaScript动态给iframe添加src属性值。
```

- label标签的作用是什么？是怎么用的?

```
作用：来定义表单控制间的关系，当用户选择该标签时，浏览器会自动将焦点转到和标签相关的表单控件上;
 <label for="Name">用户名:</label>
<input type=“text“name="Name" id="Name"/>
 <label>密码:<input type="text" name="password"/></label>
```

- 页面可见性（Page Visibility API） 可以有哪些用途?

```
通过 visibilityState 的值检测页面当前是否可见，以及打开网页的时间等;
在页面被切换到其他后台进程的时候，自动暂停音乐或视频的播放。
document.addEventListener("visibilitychange",function(){});
```

- 实现不使用 border 画出1px高的线，在不同浏览器的标准模式与怪异模式下都能保持一致的效果。

```
<div style="height:1px;overflow:hidden;background:red"></div>
```

- 网页验证码是干嘛的，是为了解决什么安全问题。

```
区分用户是计算机还是人的公共全自动程序，可以防止恶意破解密码、刷票、论坛灌水;
有效防止黑客对某一个特定注册用户用特定程序暴力破解方式进行不断的登录尝试。
```

- 外链的CSS文件和JS文件在html文件的位置？为什么这样放?

```
CSS文件在 <head> 中引入，JS文件在 <body> 底部引入;
原因：script标签的出现都会让页面等待脚本的解析和执行，在解析和执行js的这个过程中，页面渲染和用户交互完全被阻塞了。在body闭合标签之前，将所有的script标签放到页面底部，能够确保在脚本执行前页面已经完成了渲染，用户不会有很长时间的一段白屏页面。
```

CSS部分

- 介绍一下标准的CSS的盒子模型？低版本IE的盒子模型有什么不同的?

```
1. 常规的盒模型包括：内容(content)、填充(padding)、边界(margin)、边框(border);
2. IE的盒模型：content部分把 border 和 padding 计算了进去。
```

- CSS选择符有哪些？哪些属性可以继承?

```
选择符：id选择器、类选择器、标签选择器、相邻选择器、子选择器、后台选择器、通配符选择器、属性选择器、伪类选择器，共9类；
可继承样式：font-size font-family color 等;
不可继承样式：border padding margin width height 等。
```

- CSS优先级算法如何计算?

```
* 优先级就近原则，同权重情况下样式定义最近者为准；
* 载入样式以最后载入的定位为准；
优先级为：
    同权重：内联样式表(标签内部) > 嵌入样式表(当期文件) > 外部样式表(外部文件)。
    !important > id > class > tag
    important 比 内联优先级高
```

- CSS3新增伪类有那些?

```
::befor ::after :disabled :checked :enabled :nth-child(2) :first-child :last-child 等
```

- 如何让一个元素 水平垂直居中显示

```
// 情况一 元素 宽度和高度已知
.item {
    position: absolute;
    width: 500px;
    height: 300px;
    top: 50%;
    left: 50%;
    margin: -150px 0 0 -250px;  /* 外边距为自身宽高的一半 */
}
.item {
    position: absolute;     /* 相对定位或绝对定位均可 */
    width:500px;
    height:300px;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
}
// 情况二 元素 宽度和高度未知
// 1. 利用 Flex布局
.container {
    display: flex;
    align-items: center;        /* 垂直居中 */
    justify-content: center;    /* 水平居中 */
}
// 2. 利用display: table-cell,子元素设置为 inline-block
.container {
     display: table-cell;
    vertical-align: middle;
    text-align: center;
    width: 240px;
    height: 180px;
}
```

- position的值relative和absolute定位原点是?

```
relative 生成相对定位的元素，相对于其正常位置进行定位；
absolute 生成绝对定位的元素，相对于值不为 static的第一个父元素进行定位
fixed    生成固定定位的元素，相对于浏览器窗口进行定位
static   默认值，没有定位，元素出现在正常流
inherit  规定从父元素继承 position 属性的值
```

- CSS3有哪些新特性?

```
圆角、阴影、文字特效、线性渐变、过渡、缩放、动画、多背景、各种伪类选择器等
```

- 请解释一下CSS3的Flexbox（弹性盒布局模型）,以及适用场景?

Flexbox意为"弹性布局"，用来为盒状模型提供最大的灵活性。采用Flex布局的元素，称为Flex容器（flex container），简称"容器"。它的所有子元素自动成为容器成员，称为Flex项目（flex item），简称"项目"。常规布局是基于块和内联流方向，而Flex布局是基于flex-flow流，可以很方便的用来做局中，能对不同屏幕大小自适应。在布局上有了比以前更加灵活的空间

- 为什么要初始化CSS样式。

浏览器的兼容问题，不同浏览器对有些标签的默认值是不同的，表现上有差异；

- css定义的权重

元素标签的权重为 1, class 的权重为 10， id的权重为 100,多选择器的时候权重相加，权重相同时，则最后定义的样式会起作用。

- 请解释一下为什么需要清除浮动？清除浮动的方式

原因：清除浮动是为了清除浮动元素产生的影响，浮动的元素，高度会坍塌，而导致我们后面的布局不能正常的显示。
方式：常规的是通过伪类class清除：

```
.clearfix::before, .clearfix::after {
    content: " ";
    display: table;
}
.clearfix::after {
    clear: both;
}
.clearfix {
    *zoom: 1;
}
```

- 什么是外边距合并?

当两个垂直外边距相遇时，会形成一个外边距，高度等于高度较大者。

- CSS优化、提高性能的方法有哪些?

减少选择器层级、提取项目的通用公有样式、按模块编写组件、增强项目的协同开发性、可维护性和可扩展性；使用预处理工具或构建工具(对css进行语法检查、自动补前缀、打包压缩、自动优雅降级)；

- 什么是响应式设计？响应式设计的基本原理是什么？如何兼容低版本的IE?

响应式设计：页面的设计和开发应当根据用户行为以及设备环境进行相应的相应和调整，具体包括弹性网格和布局、图片、媒体查询的使用，让一个网站能够兼容多个不同的终端。
基本原理：通过媒体查询检测不同的设备屏幕尺寸做处理，页面头部必须有meta声明viewport。
兼容：通过js辅助调整

- 让页面里的字体变清晰，变细用CSS怎么做?

```
-webkit-font-smoothing: antialiased;
```

- 什么是CSS 预处理器 / 后处理器?

```
1. 预处理器：LESS、Sass、Stylus,增强代码的复用性，还有层级、mixin、变量、循环、函数等，具有很方便的UI组件模块化开发能力，极大的提高工作效率。
2. 后处理器：PostCSS，通常视为在完成的样式表根据CSS规范处理CSS，目前常用的是给CSS属性添加浏览器私有前缀。
```

- rem布局的优缺点

```
rem自适应原理：rem是根据 根元素<html> 的font-size大小来变化，通过js根据设备的宽度动态的去调整 html字号，从而实现自适应布局。
实现方式：1. 通过JS动态计算根元素的font-size,淘宝目前使用这种方式；2. 事先统计网站要兼容的屏幕设备，通过css3的媒体查询去实现font-size的调整。
缺点：rem只能在移动端进行布局，然后等比例缩放，pc端还是px实现更好，rem不能实现响应式布局。
```

- 描述一下css的渐进增强和优雅降级之间的不同

```
渐进增强：针对低版本浏览器进行构建页面，保证最基本的功能，然后再针对高级浏览器进行效果、交互等改进和追加功能达到更好的用户体验。
优雅降级：一开始就构建完整的功能，再针对低版本浏览器进行兼容。
区别：优雅降级是从复杂的现状开始，并试图减少用户体验的供给，而渐进增强则是从一个非常基础的、能够起作用的版本开始，并不断扩充，以适应未来环境的需要。
```

JavaScript 部分

- 介绍js的基本数据类型。

```
基本数据类型：String、Number、Boolean、Null、Undefined;
引用类型：Object、Array、Function;
ES6新增：Symbol (独一无二不可变的值)
```

- js原始类型和引用类型的区别?

```
存储位置不同：
  原始数据类型：直接存储在栈(stack)中的简单数据段，占据空间小、大小固定，属于被频繁使用数据，所以放入栈中存储；
  引用数据类型：存储在堆(deap)中的对象，占据空间大、大小不固定，若存储在栈中，将会影响程序运行的性能，引用数据类型在栈中存储了指针，该指针指向该实体的起始位置，当解释器寻找引用值时，会首先检索其在栈中的地址。
```

- 介绍js有哪些内置对象?

```
Object 是 JavaScript 中所有对象的父对象
数据封装类对象：Object、Array、Boolean、Number、String
其他对象：Function、Arguments、Math、Date、RegExp、Error
```

- 说几条写JavaScript的基本规范?

```
1. 尽量不要在同一行声明多个变量;
2. 尽量使用 ===/!== 来比较 true/false 或者数值
3. 尽量使用对象字面量( []、{} ) 替代 new的构造形式( new Array() );
4. 尽量少的使用全局函数;
5. For循环和If语句尽量都使用大括号包裹。
```

- 如何将浮点数点左边的数每三位添加一个逗号，如12000000.11转化为『12,000,000.11』?

```
function commafy(num) {
    return num&&num.toString()
    .replace/(\d)(?=(\d{3})+\.)/g, function($1, $2) {
        return $2 + ',';
    })
}
```

- 如何实现数组的随机排序?

```
var arr = [1,2,3,4,5,6,7,8,9];
arr.sort(function() {
    return Math.random() - 0.5;
})
console.log(arr);
```

- 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，这就是原型链的概念。
关系：instance.constructor.prototype = instance.__proto__ ;
特点：JS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。
```

- Javascript如何实现继承?

  原型链继承、构造继承、实例继承、拷贝继承、组合继承、寄生组合继承

```
// 定义一个父类 -- 动物类
function Animal(name) {
    // 属性
    this.name = name || 'Animal';
    // 实例方法
    this.sleep = function() {
        console.log(this.name + '正在睡觉! ');
    }
}
// 原型方法
Animal.prototype.eat = function(food) {
    console.log(this.name + '正在吃: ' + food);
};
```

1. 原型链继承：将父类的实例作为子类的原型：Cat.prototype = new Animal();
   优点：很纯粹的继承，父类新增原型方法和原型属性，子类都能访问到，简单，易实现；
   缺点：(1)要为子类新增原型属性和方法，必须放在 new Animal() 语句后，不能放构造器中；
   (2)无法实现多继承；
   (3)来自原型对象的引用属性是所有实例共享的；
   (4)创建实例时，无法向父类构造函数传参。

2. 构造继承：使用父类的构造函数来增强子类实例，等于复制父类的实例属性给子类(没用原型)
   代码：

   ```
   function Cat(name) {
       Animal.call(this);
       this.name = name || 'Tom';
    }
   ```

   优点：(1)解决了子类实例共享父类引用属性的问题；
   (2) 创建子类实例时，可以向父类传递参数；
   (3) 可以实现多继承(call多个父类对象)。
   缺点：(1) 实例并不是父类的实例，只是子类的实例；
   (2) 只能继承父类的实例属性和方法，不能继承原型属性/方法；
   (3) 无法实现函数复用，每个子类都有父类实例函数的副本，影响性能。

3. 实例继承：为父类实例添加新特性，作为子类实例返回
   代码：

```
function Cat() {
        var instance = new Animal();
        instance.name = name || 'Tom';
        return instance;
    }
```

优点：不限制调用方式，不管是 new子类，还是子类，返回的对象具有相同的效果
缺点：实例是父类的实例，不是子类的实例；不支持多继承。

4. 拷贝继承： 代码：

```
function Cat(name) {
        var animal = new Animal();
        for(var p in animal) {
            Cat.prototype[p] = animal[p];
        }
        Cat.prototype.name = name || 'Tom';
    }
```

优点：支持多继承
缺点：效率较低，内存占用高(要拷贝父类的方法)；无法获取父类不可枚举的方法。

5. 组合继承：调用父类构造，继承父类的属性并保留传参的优点，然后将父类实例作为子类原型，实现函数复用
代码：

```
function Cat(name) {
    Animal.call(this);
    this.name = name || 'Tom';
}
Cat.prototype = new Animal();
```

优点：(1) 弥补了构造继承的缺陷，可以继承实例属性/方法，也可以继承原型属性/方法；
(2) 既是子类的实例，也是父类的实例；
(3) 不存在引用属性共享问题；
(4) 可以传参，函数也可以复用；
缺点：调用了两次父类构造函数，生成了两份实例(子类实例将子类原型上的那份屏蔽了)。

6. 寄生组合继承：通过寄生方式，砍掉父类的实例属性，这样在调用两次父类的构造的时候，就不会初始化两次实例方法/属性，避免组合继承的缺点。
代码：

```
function Cat(name) {
        Animal.call(this);
        this.name = name || 'Tom';
      }
      (function() {
        // 创建一个没有实例方法的类
        var Super = function() {};
        Super.prototype = Animal.prototype;
        // 将实例作为子类的原型
        Cat.prototype = new Super();
      })();
```

优点：堪称完美
缺点：实现较为复杂

- JavaScript创建对象的几种方式?

  1. 对象字面量的方式

```
person = {
    firstname: "Mark",
    lastname: "Yun",
    age: 25
};
```

  2. 用工厂方式来创建(内置对象)

```
var wcDog = new Object();
wcDog.name = "旺财";
wcDog.age = 3;
wcDog.work = function() {
    alert("我是" + wcDog.name);
}
```

  3. 用function来模拟无参的构造函数

  4. 用function来模拟参构造函数来实现（用this关键字定义构造的上下文属性）

  5. 用混合方式来创建

```
function Car(name, price) {
    this.name = name;
    this.price = price;
}
Car.prototype.sell = function() {
    alert("我是"+this.name+"，我现在卖"+this.price+"万元");
}
var camry = new Car("凯美瑞", 27);
camry.sell();
```

- javascript作用链域?

全局函数无法查看局部函数的内部细节，但局部函数可以查看其上层的函数细节，直至全局细节。
当需要从局部函数查找某一属性或方法时，若当前作用域没有找到，就会上溯到上层作用域查找，直至全局函数，这种组织形式就是作用域链。

- 谈谈This对象的理解。

this总是指向函数的直接调用者(而非间接调用者)；
如果有new关键字，this指向new出来的那个对象；
在事件中，this指向触发这个事件的对象。

- 什么是window对象，什么是document对象?

window对象是指浏览器打开的窗口。
document对象是指Documentd对象(HTML文档对象)的一个只读引用，window对象的一个属性。

- null, undefined的区别?

null 表示一个对象是"没有值"的值，也就是值为"空"；
undefined 表示一个变量声明了没有初始化(赋值)。

- ["1", "2", "3"].map(parseInt) 答案是多少?

答案是：[1, NaN, NaN]
parseInt() 函数能解析一个字符串，并返回一个整数，需要两个参数 (val, radix)

- 事件是？IE与火狐的事件机制有什么区别？ 如何阻止冒泡?

事件：我们在网页中的某个操作(有的操作对应多个事件)，可以被JavaScript侦查到。
事件处理机制：IE是事件冒泡、Firefox同时支持两种事件模型：捕获型事件和冒泡型事件。
阻止冒泡：event.stopPropagation();

- 什么是闭包（closure），为什么要用它?

闭包是指有权访问另一个函数作用域中变量的函数，创建闭包的最常见的方式就是在一个函数内创建另一个函数，通过另一个函数访问这个函数的局部变量，利用闭包可以突破作用域链，将函数内部的变量和方法传递到外部。
闭包的特性：

```
1. 函数内再嵌套函数；
2. 内部函数可以引用外层的参数和变量；
3. 参数和变量不会被垃圾回收机制回收。
for(var i=0; i<5; i++) {
    (function(i){
        setTimeout(function timer() {
            console.log(i);
        }, 1000);
    })(i);
}
输出结果为：0 1 2 3 4
```

- javascript 代码中的"use strict";是什么意思 ? 使用它区别是什么?

是ECMAscript5 添加的严格运行模式，是JavaScript在更严格的条件下运行；
区别：使JS编码更加规范化，消除语法上的一些不合理，不严谨，减少一些怪异行为；
消除代码运行的一些不安全之处，保证代码运行的安全。提高编译器效率，增加运行速度；

- 如何判断一个对象是否属于某个类?

```
使用 instanceof : if( a instanceof Person ){ alert('yes'); }
```

- 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，并最后隐式的返回 this。
```

- 用原生JavaScript的实现过什么功能吗?
- Javascript中，有一个函数，执行时对象查找时，永远不会去查找原型，这个函数是?

```
hasOwnProperty : 返回一个布尔值，指出一个对象是否具有指定名称的属性，不会检查该对象的原型。
使用方法：object.hasOwnProperty(proName);
```

- 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。
```

- js延迟加载的方式有哪些?

```
defer 和 async 、动态创建DOM方式(用的最多)、按需异步载入js
```

- Ajax 是什么? 如何创建一个Ajax?

```
ajax的全称：Asynchronous Javascript And XML (异步传输 + js + xml);
异步：向服务器发送请求，我们不必等待结果，可以同时做其他事情，等有了结果它自己会根据设定进行后续操作，与此同时，页面是不会发生整页刷新，提高了用户体验。
创建过程：
    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以及验证信息；
    3. 设置相应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。
```

- 同步和异步的区别?

```
同步：只有当一个请求结束后，才会去执行下一个请求，强调的顺序性，等待时间长，交互效果差；
异步：一个请求发出后不必去等待它的返回结果，可以继续做下面的事情。
```

- 如何解决跨域问题?

```
jsonp 、iframe、window.name、window.postMessage、服务器上设置代理页面
```

- 模块化开发怎么做?

  使用立即执行函数，不暴露私有成员

```
var module1 = (function() {
    var _count = 0;
    var m1 = function() {
      // ...
    };
    var m2 = function() {
        // ...
    };
    return {
        m1: m1,
        m2: m2
    };
})();
```

- AMD（Modules/Asynchronous-Definition）、CMD（Common Module Definition）规范区别?

异步模块(Asynchronous Module Definition)：所有的模块将被异步加载，模块加载不影响后面语句运行。所有依赖某些模块的语句均放置在回调函数中。
区别：
    1. 对于依赖的模块，AMD 是提前执行，CMD是延迟执行。不过RequireJS从2.0开始，也改为延迟执行。
    2. CMD推崇依赖就近，AMD 推崇依赖前置。

```
// CMD 语法代码
define(function(require, exports, module) {
    var a = require('./a')
    a.doSomething()
    // 此处省略 100 行
    var b = require('./b') // 依赖可以就近书写
    b.doSomething()
    // ...
})
// AMD 默认推荐
define(['./a', './b'], function(a, b) { // 依赖必须一开始就写好
    a.doSomething()
    // 此处省略 100 行
    b.doSomething()
    // ...
})
```

- requireJS的核心原理是什么？（如何动态加载的？如何避免多次加载的？如何 缓存的？）

参考：http://annn.me/how-to-realize-cmd-loader/

- 谈一谈你对ECMAScript6的了解?

ECMAScript 6.0(简称 ES6)：是JavaScript语言的下一代标准，已经于 2015年6月正式发不了。目标是使得JavaScript语言可以用来编写复杂的大型应用程序，成为企业级开发语言。
ES6是一个泛指，含义是5.1版以后的JavaScript的下一代标准，涵盖了 ES2015、ES2016、ES2017等等。
ES6一些新增的特性：块级作用域(let和const)、默认参数、模板表达式、多行字符串、变量解构赋值、箭头函数、Promise对象、Class类、Module(模块化)实现等

- ECMAScript6 怎么写class类，为什么会出现class这种东西?

在JavaScript语言中，生成实例对象的传统方法是通过构造函数，这与传统的面向对象语言差异很大，故在 ES6中引入了 Class(类) 的概念，作为对象的模板，通过 `class`关键字，可以定义类。
基本上，ES6的class可以看做只是一个语法糖，它的绝大部分功能，ES5都可以做到，新的class类写法只是让对象原型的写法更加清晰、更像面向对象编程的语法而已。

```
// ES6 定义类代码
class Point {
    constructor(x, y) {
        this.x = x;
        this.y = y;
    }

    toString() {
        return '(' + this.x + ', ' + this.y + ')';
    }
}
// ES5 的构造函数方式代码
function Point(x, y) {
    this.x = x;
    this.y = y;
}
Point.prototype.toString = function () {
    return '(' + this.x + ', ' + this.y + ')';
};

var p = new Point(1, 2);
```

- ES6中 Class类继承的实现 -- extends

Class通过 extends关键字实现继承，可继承父类的所有属性和方法，其中子类必须在 constructor方法中调用 super 方法，在这里表示父类的构造函数，用来新建父类的 this对象，若没有，则新建实例时会报错。
ES5的继承，实质是先创造子类的实例对象 this，然后再将父类的方法添加到 this上面(Parent.apply(this))。ES6 的继承机制则完全不同，实质是先创造父类的实例对象 this(必须先调用 super方法)，再用子类的构造函数修改 this。
原生构造函数继承(ES5不可继承，ES6可以)：Boolean()、Number()、String()、Array()、Date()、Function()、RegExp()、Error()、Object().

- documen.write和 innerHTML的区别

```
document.write 只能重绘整个页面
innerHTML可以重绘页面的某一部分
```

- DOM操作 -- 怎样添加、移除、移动、复制、创建和查找节点?

```
(1) 创建新节点
    document.createDocumnetFragment()  // 创建一个DOM片段
    document.createElement()   // 创建一个具体的元素
    document.createTextNode()  // 创建一个文本节点
(2) 节点的添加、移除、替换、插入
    element.appendChild()  // 向节点添加最后一个子节点
    element.removeChild()  // 从元素中移除子节点
    element.replaceChild()  // 替换元素中的子节点
    element.insertBefore()  // 在已有的子节点前插入一个新的子节点
(3) 查找
    document.getElementsByTagName()  // 通过标签名称查找元素
    document.getElementsByClassName()  // 通过元素的class类名来查找元素
    document.getElementById()         // 通过元素Id，唯一性
```

- .call() 和 .apply() 的区别?

```
相同：都用来改变当前函数调用的对象
不同：call的第二个参数要依次传入，apply的第二个参数传入一个参数数组，
```

- 数组和对象有哪些原生方法，列举一下？ ES6 新添加的方法又有哪些?

```
数组(Array)的对象属性：
  1. concat() 连接两个或更多的数组，并返回结果；
  2. join()   把数组中的元素放入一个字符串；
  3. push()   向数组末尾添加一个或更多元素；
  4. pop()    删除数组的最后一个元素并返回删除的元素；
  5. unshift() 向数组的开头添加一个或更多元素，并返回新的长度；
  6. shift()  删除并返回数组的第一个元素；
  7. sort()   对数组的元素进行排序；
  8. reverse() 反转数组的元素顺序；
  9. slice()   选取数组的一部分，并返回一个新数组；
  10. splice() 从数组中添加或者删除元素；
  11. indexOf()  搜索数组中的元素，并返回它所在的位置
  12. map()      通过指定函数处理数组的每个元素，并返回处理后的数组
ES6 扩展的数组方法：
    1. 扩展运算符(...) ： 将一个数组，变为参数序列；
    2. from() ： 将两类对象转为真正的数组(类数组对象和可遍历对象)
    3. of() ：  将一组值，转换为数组
    4. copyWithin()：将指定位置的成员复制到其他位置(会覆盖原有成员)，返回当前数组
    5. find() 和 findIndex()：找出第一个符合条件的数组成员 和 返回第一个符合条件成员的位
置
    6. fill()：使用给定值，填充一个数组
    7. keys()：对数组键名的遍历
    8. values()：对数组键值的遍历
    9. entries()：对数组键值对的遍历
    10. includes() 判断一个数组是否包含一个指定的值

ES6 对象的扩展：
    1. 属性的简介表示：const baz = {foo} 等同于 {foo：foo}
    2. 属性名表达式：(1) obj.foo = true; (2) obj['a'+'bc'] = 123;
    3. 方法的 name属性;
    4. Object.is()：比较两个值是否严格相等，与(===)行为基本一致；
    5. Object.assign()：用于对象的合并，将源对象的所有可枚举属性，复制到目标对象;
    6. super关键字：指向当前对象的原型对象；
    7. Object.keys()：返回一个数组，成员是参数对象自身(不含继承的)所有可遍历属性的键名；
    8. Object.values()：返回一个数组，成员是对象自身(不含继承的)所有可遍历属性的键值；
    9. Object.entries()：返回一个数组，成员是对象自身(不含继承)所有可遍历属性的键值对数组
；
    10. 解构赋值
```

- 移动端的点击事件的有延迟，时间是多久，为什么会有？ 怎么解决这个延时?

```
click 有 300ms延迟，为了实现safari的双击事件的设计，浏览器要知道你是不是要双击操作，常规的
解决方式是 引入 fastclick.js
```

前端框架

- React使用场景?

```
逻辑复杂单页应用，偏中后台管理系统，纯展示型的UI页面不合适
```

- 描述一下React 生命周期

```
渲染过程 - 首次实例化
* getDefaultProps      // 只调用一次，获取默认的props
* getInitialState      // 实例创建时调用一次，初始化实例的state,可访问this.props
* componentWillMount   // 完成首次渲染前调用，此时仍可修改组件的state
* render               // 创建虚拟 DOM，渲染到真实的DOM
* componentDidMount    // 真实的DOM被渲染出来后调用，可通过this.getDOMNode()访问到真实的DOM元素。

实例化完全后的更新
* getInitialState
* componentWillMount
* render
* componentDidMount

存在期的状态改变
* componentWillReceiveProps  // 组件接收到新的props时调用，将其作为nextProps使用
* shouldComponentUpdate      // 组件是否应当渲染新的props或state，true或false
* componentWillUpdate        // 接收到新的props或者state后，进行渲染前调用，不允许更新props或者state,防止进入死循环
* render
* componentDidUpdate    // 完成渲染新的props或者state后调用，可访问到新的DOM元素

销毁&清理期
* componentWillUnmount  // 组件被移除之前调用，主要做一些清理工作
```

- 实现组件的几种方式?

```
1. React.createClass 使用API来定义组件
2. React ES6 class component 用 ES6 的 class 来定义组件
3. Functional stateless component   通过函数定义无状态组件
```

- 当组件的setState函数被调用之后，发生了什么?

```
将你传递给 setState的参数对象合并到组件原先的state,
```

- React-router 路由的实现原理?
- 受控组件(Controlled Component)与非受控组件(Uncontrolled Component)的区别
- refs 是什么?

Refs是能访问DOM元素或组件实例的一个函数；

- 并不是父子关系的组件，如何实现相互的数据通信?

使用父组件，通过props将变量传入子组件

- React的批量更新机制 BatchUpdates?

- React与Vue，各自的组件更新进行对比，它们有哪些区别?

一、相同部分：
1．数据驱动视图，提供响应式的视图组件
2．都有virtual DOM，组件化开发，通过props参数进行父子组件数据的传递，都实现了webComponents规范
3．都支持服务端渲染
4．都有native 解决方案，ReactNative(Facebook团队) 和 Weex(阿里团队)
5．都是模块化思想的严格践行者，以及通过component拆分完整系统

二、不同部分
1．社区：react相对来讲要大于vue,第三方组件库和插件更加的丰富,因为背后是一个团队在开发和维护，但是目前vue的增长速度是要高于react增长速度，未来的发展趋势都是未可知的；
2．开发模式：React本身，是严格的view层，MVC模式；Vue则是MVVM模式的一种实现方式；
3．数据绑定：Vue借鉴了angular,采用双向数据绑定的方式(value的单向绑定+onChange事件侦听的一个语法糖，组件间的数据传递，默认是单向的)；React,则采取单向数据流的方式；
4．数据渲染：对于大规模数据渲染，React要高于Vue一些；但对于小轻量的，Vue更高效；
React的渲染建立在Virtual DOM上-- 一种在内存中描述DOM树状态的数据结构。当状态发生变化时，React重新渲染Virtual DOM，比较计算之后给真实DOM打补丁。Virtual DOM提供了一个函数式的方法描述视图，不使用数据观察机制，每次都会重新

5．数据更新：Vue采取依赖追踪，默认是优化状态，动了多少数据，就触发多少更新，按需更新

6．使用场景：React配合严格的Flux架构，适合超大规模多人协作的复杂项目；在小快灵的项目上，Vue具有更大的优势，学习成本低，开发快速，在需要对DOM进行很多自定义操作上灵活性强。
7．开发风格偏好：React 推荐的写法是 JSX + inline style，也就是把HTML和CSS全部整合进JavaScript。Vue的默认API是以简单易上手为目标，进阶后推荐使用webpack+vue-loader的单文件模板文件。JSX在逻辑表达能力上面完爆模板，但容易写出凌乱的render函数，不如模板看起来一目了然。

- Vue 和 React 项目选型的探讨

```
Vue
 * mvvm架构的核心及价值是数据的双向绑定，引入mvvm框架主要看业务上是否有这类需求，目前看后台业务比较适合。
 * vue不适合持续工程迭代，是因为数据是可以在view和model进行双向流动的，但当工程规模比较大时，如果处理不当，前端的行为将变得难以预测，进而就难以调试和维护了。
 * 对于简单页面，小型应用，vue开发则非常高效，只要处理好view，然后传入 data就好了
 * vue模板更接近于web component的实现
 * vue源码相对简单，业务上的调整也更加顺手
Vue 和 React 的选择，可理解为：
 * 双向绑定及单向数据流的选择
 * web component 与 virtual dom的选择
 * 技术上来说，没有过不去的坎，两种库的选择，实际上是哪个更符合未来趋势及标准
 * 而最终趋势和标准在表现在 使用人群、框架影响力及社区成熟度
```

-

其他问题

- 原来公司工作流程是怎么样的，如何与其他人协作的？如何跨部门合作的?
- 你遇到过比较难的技术问题是？你是如何解决的?
- 设计模式 知道什么是singleton, factory, strategy, decrator么?
- 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?
- 页面重构怎么操作?

```
网站重构：在不改变外部行为的前提下，简化结构、添加可读性，而在网站前端保持一致的行为。
也就是说在不改变UI的情况下，对网站进行优化，在扩展的同时保持一致的UI。
 1. 减少代码间的耦合
 2. 严格按照规范编写代码
 3. 设计可扩展的API
 4. 代替旧有的框架、语言
 5. 增强用户体验
 6. 优化网站的速度
 7. 压缩JS、CSS、Image等前端资源
 8. 程序的性能优化
 9. 采用CDN来加载静态资源
 10. 加强 JS 对 DOM操作的优化
 11. HTTP服务器的文件缓存
```

- 是否了解公钥加密和私钥加密。

```
一般情况下是指 私钥用于对数据进行签名，公钥用于对签名进行验证；
HTTP网站在浏览器端用公钥加密敏感数据，然后在服务器端再用私钥解密。
```

- WEB应用从服务器主动推送Data到客户端有那些方式?

```
1. html5提供的Websocket技术
2. 不可见的iframe
3. XHR长时间连接-长轮询技术
4. <script>标签的长时间连接(可跨域)
```

- 对Node的优点和缺点提出了自己的看法?

```
优点：因为Node是基于事件驱动和无阻塞的，所有非常适合处理并发请求，因此构建在Node上的代理服务器相比其他技术实现的服务器表现好很多。此外，与Node代理服务器交互的客户端代码是由JavaScript语言编写，故客户端和服务器端都用同一种语言编写，很方便。
缺点：Node是一个相对新的开源项目，所以不太稳定，它总是一直在变，而且缺少足够多的第三方库支持，还需要时间的沉淀。
```

- 你有用过哪些前端性能优化的方法?

```
1. 减少http请求次数：css精灵图、JS/CSS源码压缩、图片大小的控制、CDN托管静态资源、data缓存、图片服务器；
2. 前端模板 JS+数据，减少由于HTML标签导致的带宽浪费，前端用变量保存AJAX请求结果；
3. 用innerHTML代替DOM操作，减少DOM操作次数，优化JavaScript性能；
4. 当需要设置的样式很多时，设置className 而不是去直接操作style;
5. 少用全局变量、缓存DOM节点查找的结果，减少IO读取操作；
6. 图片预加载，将样式表放在顶部，将脚本放在底部，加上时间戳。
```

- http状态码有那些？分别代表是什么意思?

```
100  Continue  继续，正常发送http请求后服务器返回此信息，表示确认，之后发送具体参数信息
200  OK        正常返回请求的信息
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(可能过载或维护)
```

- 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？（流程说的越详细越好）

```
1. 浏览器会开启一个线程来处理这个请求，对URL分析判断如果是 http协议就按照Web方式来处理;
2. 调用浏览器内核中的对应方法，比如WebView 中的 loadUrl 方法;
3. 通过DNS解析获取网址的IP地址，设置UA等信息发出第二个GET请求;
4. 进行HTTP协议会话，客户端发送请求报头;
5. 进入到Web服务器上的 Web Server，如Apache、NodeJS等服务器;
6. 进入部署好的后端应用，如PHP、Java、Python等，找到对应的请求处理;
7. 处理结束，回馈报头，若浏览器访问过，缓存上有对应资源，会与服务器最后修改时间对比，一致则返回304;
8. 浏览器开始下载html文档(响应报头，状态码200),同时使用缓存;
9. 文档树建立，根据标记请求所需指定MIME类型的文件(比如css、js),同时设置了 cookie;
10. 页面开始渲染DOM，JS根据DOM API操作DOM,执行事件绑定等，页面显示完成。
```

- 你用的得心应手用的熟练地编辑器&开发环境是什么样子?

```
Sublime Text 3 + ATOM + 插件
Google chrome 查看页面UI、动画效果和交互功能，Firebug 测试兼容
NodeJS + webpack + Yarn
Git 版本控制
```

- 对前端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、运营、安全都近
1、实现界面交互 2、提示用户体验 3、有了Node.js，前端可以实现服务器端的一些事情
4、参与项目的开发，快速高质量完成实际效果图，精确到 1px;
5、与团队成员，UI设计、产品经理、后端人员的沟通;
6、做好页面结构、页面重构和用户体验;
7、处理hack、兼容、写出优美的代码格式;
8、针对服务器的优化，拥抱最新前端技术。
```

- 平时如何管理你的项目?

```
先期，团队必须确定好全局样式，编码模式等;
编写习惯必须一致(空格、换行、模块化封装);
标注模块编写人的时间，各模块及时标注，添加注释;
对页面进行分块标注(例如 页面、模块、开始和结束);
CSS、JS 和 HTML 分文件夹并行存放，命名统一;
后台API请求接口统一处理和存放位置;
图片采用base64、Svg等格式，减少http请求。
```

- 简单叙述下你项目开发的前期准备和流程?

前期准备：

1．仔细阅读产品的需求和原型，分析整个项目可能会用到的功能和技术难点，确定要使用的技术是使用第三方插件还是自己写，列出一个项目页面整体表和技术点(页面、功能、技术)；

2．根据需求确定技术选型，确定代码对于浏览器的版本兼容，决定用什么编辑器开发，使用什么框架，使用那些插件和技术。

3．和产品讨论项目需求中，不明确的地方和功能交互效果的展示，确定页面逻辑

4．和UI讨论页面的组件化开发，确定一致的开发风格和主题；

5．和后端人员确定api接口的风格和测试的服务器地址；

6．着手准备项目库的搭建和初始环境的结构目录；

7．确定好代码开发规范、项目结构目录和命名规范；

7．进行第三方插件的查找和学习，具体功能模块的开发、全局调用模块的封装；

8．拿到UI效果图后，根据UI的标注，进行样式和效果的沟通核实；

9．拿到后台API文档后，仔细核实，查看返回数据格式是否符合要求，及时与后台沟通。

10．项目开发结束后，进行UI样式还原和功能交互测试。

-