(2) 前端基础 (3) JS(2)1

1 JS判断类型的方法【typeof、
instanceof、
Object.prototype.toString.call()
（其 不等价于某实例、 如 obj.toString() ， 因为 toString方法可能被重写了）。
它们之间的各种区别】
1.1 typeof 也可以写成 typeof() ，但是 其不是函数、而是一个操作符！！ 基本数据类型可以用 typeof 去判断、但是 typeof null ==> 出来时 "object" —— 这是历史遗留问题。（低位的 000 开头的！！）
其实有一个骚操作， typeof可以判断具体的复杂数据类型、如 typeof obj 出来是 "object"；在判断一下 obj 有没有 数组的方法、有的话就是 数组对象！！！

1.2 instanceof 也是一个操作符，但是它需要 2个“运算值”、typeof只需要一个 “运算值”！！！
一层层往上找其“父类、基类”等。只要 在继承链上 就是 true 。

1.3 Object.prototype.toString.call()
一般这是用来区分 复杂数据类型的，如

```
Object.prototype.toString.call()
"[object Undefined]"
Object.prototype.toString.call([])
"[object Array]"
Object.prototype.toString.call({})
"[object Object]"
Object.prototype.toString.call(new Function())
"[object Function]"
```

2 数组的常用方法【注意：区分有、无副作用？！！】
push、pop、shift、unshift、splice、sort、reverse、map 等

3 闭包的定义以及使用场景？？

（1）什么是闭包：

闭包是指有权访问另外一个函数作用域中的变量的函数。

闭包就是函数的局部变量集合，只是这些局部变量在函数返回后会继续存在。闭包就是就是函数的“堆栈”在函数返回后并不释放，我们也可以理解为这些函数堆栈并不在栈上分配而是在堆上分配。当在一个函数内定义另外一个函数就会产生闭包。

（2）为什么要用：

匿名自执行函数：我们知道所有的变量，如果不加上var关键字，则默认的会添加到全局对象的属性上去，这样的临时变量加入全局对象有很多坏处，比如：别的函数可能误用这些变量；造成全局对象过于庞大，影响访问速度(因为变量的取值是需要从原型链上遍历的)。除了每次使用变量都是用var关键字外，我们在实际情况下经常遇到这样一种情况，即有的函数只需要执行一次，其内部变量无需维护，可以用闭包。

结果缓存：我们开发中会碰到很多情况，设想我们有一个处理过程很耗时的函数对象，每次调用都会花费很长时间，那么我们就需要将计算出来的值存储起来，当调用这个函数的时候，首先在缓存中查找，如果找不到，则进行计算，然后更新缓存并返回值，如果找到了，直接返回查找到的值即可。闭包正是可以做到这一点，因为它不会释放外部的引用，从而函数内部的值可以得以保留。

封装：实现类和继承等。

4 事件代理（委托）在捕获阶段的实际应用
可以在父元素层面 阻止事件 向子元素传播【对应哪个函数API？？ 阻止冒泡是e.stopPropagation。 阻止默认行为 e.preventDefault。 return false 可以阻止事件冒泡和默认行为】

5 去除字符串首尾空格
使用正则(^\s*)|(\s*$)即可

6 性能优化！！

减少HTTP请求

使用内容发布网络（CDN）

添加本地缓存

压缩资源文件

将CSS样式表放在顶部，把javascript放在底部（浏览器的运行机制决定）

避免使用CSS表达式

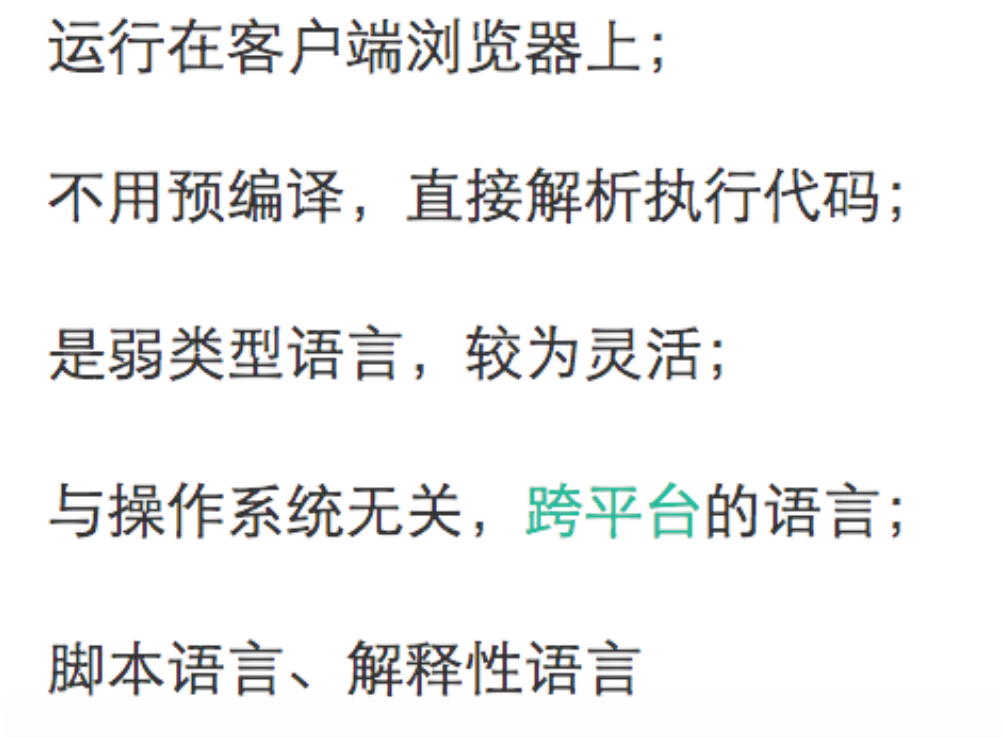
减少DNS查询

使用外部javascript和CSS

避免重定向

图片lazyLoad

7 JS的语言特性【运行在客户端上（其实服务器也可以吧？？ Node.js？？）；脚本语言、解释性语言；弱类型语言、较为灵活；不用编译、直接解释执行代码；与 OS操作系统无关、跨平台的语言】



8 JS实现跨域的方法【可以达到 9种？！！
JSONP；
CORS；
postMessage；
websockect；
node中间件；
nginx代理；
iframe + 其他3种组合。
】

JSONP：通过动态创建script，再请求一个带参网址实现跨域通信。document.domain + iframe跨域：两个页面都通过js强制设置document.domain为基础主域，就实现了同域。

location.hash + iframe跨域：a欲与b跨域相互通信，通过中间页c来实现。 三个页面，不同域之间利用iframe的location.hash传值，相同域之间直接js访问来通信。

window.name + iframe跨域：通过iframe的src属性由外域转向本地域，跨域数据即由iframe的window.name从外域传递到本地域。

postMessage跨域：可以跨域操作的window属性之一。

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

代理跨域：启一个代理服务器，实现数据的转发

9 JS深度拷贝一个 元素的具体实现

```
var deepCopy = function(obj) {
if (typeof obj !== 'object') return;
var newObj = obj instanceof Array ? [] : {};
for (var key in obj) {
if (obj.hasOwnProperty(key)) {
newObj[key] = typeof obj[key] === 'object' ? deepCopy(obj[key]) : obj[key];
}
}
return newObj;
}
```

10 重排【亦称为 回流】 和 重绘。

重绘（repaint或redraw）：当盒子的位置、大小以及其他属性，例如颜色、字体大小等都确定下来之后，浏览器便把这些原色都按照各自的特性绘制一遍，将内容呈现在页面上。重绘是指一个元素外观的改变所触发的浏览器行为，浏览器会根据元素的新属性重新绘制，使元素呈现新的外观。

触发重绘的条件：改变元素外观属性。如：color，background-color等。

注意：table及其内部元素可能需要多次计算才能确定好其在渲染树中节点的属性值，比同等元素要多花两倍时间，这就是我们尽量避免使用table布局页面的原因之一。

重排（重构/回流/reflow）：当渲染树中的一部分(或全部)因为元素的规模尺寸，布局，隐藏等改变而需要重新构建, 这就称为回流(reflow)。每个页面至少需要一次回流，就是在页面第一次加载的时候。

重绘和重排的关系：在回流的时候，浏览器会使渲染树中受到影响的部分失效，并重新构造这部分渲染树，完成回流后，浏览器会重新绘制受影响的部分到屏幕中，该过程称为重绘。所以，重排必定会引发重绘，但重绘不一定会引发重排。

11 JS的全排列【全排列的定义？？？！ 自己手撕一下？？】

```
function permutate(str) {
var result = [];
if(str.length > 1) {
var left = str[0];
var rest = str.slice(1, str.length);
var preResult = permutate(rest);
for(var i=0; i<preResult.length; i++) {
for(var j=0; j<preResult[i].length; j++) {
var tmp = preResult[i],slice(0, j) + left + preResult[i].slice(j, preResult[i].length);
result.push(tmp);
}
}
} else if (str.length == 1) {
return [str];
}
return result;
}
```

12 this的指向有哪几种？？

默认绑定：全局环境中，this默认绑定到window。

隐式绑定：一般地，被直接对象所包含的函数调用时，也称为方法调用，this隐式绑定到该直接对象。

隐式丢失：隐式丢失是指被隐式绑定的函数丢失绑定对象，从而默认绑定到window。显式绑定：通过call()、apply()、bind()方法把对象绑定到this上，叫做显式绑定。

new绑定：如果函数或者方法调用之前带有关键字new，它就构成构造函数调用。对于this绑定来说，称为new绑定。

【1】构造函数通常不使用return关键字，它们通常初始化新对象，当构造函数的函数体执行完毕时，它会显式返回。在这种情况下，构造函数调用表达式的计算结果就是这个新对象的值。

【2】如果构造函数使用return语句但没有指定返回值，或者返回一个原始值，那么这时将忽略返回值，同时使用这个新对象作为调用结果。

【3】如果构造函数显式地使用return语句返回一个对象，那么调用表达式的值就是这个对象。

13 AngularJS的双向绑定

Angular将双向绑定转换为一堆watch表达式，然后递归这些表达式检查是否发生过变化，如果变了则执行相应的watcher函数（指view上的指令，如ng-bind，ng-show等或是{{}}）。等到model中的值不再发生变化，也就不会再有watcher被触发，一个完整的digest循环就完成了。

Angular中在view上声明的事件指令，如：ng-click、ng-change等，会将浏览器的事件转发给$scope上相应的model的响应函数。等待相应函数改变model，紧接着触发脏检查机制刷新view。

watch表达式：可以是一个函数、可以是$scope上的一个属性名，也可以是一个字符串形式的表达式。$watch函数所监听的对象叫做watch表达式。watcher函数：指在view上的指令（ngBind，ngShow、ngHide等）以及{{}}表达式，他们所注册的函数。每一个watcher对象都包括：监听函数，上次变化的值，获取监听表达式的方法以及监听表达式，最后还包括是否需要使用深度对比（angular.equals()）

14 怎么使用 requestAnimationFrame？？

该方法告诉浏览器，您希望执行动画并请求浏览器在 下一次重绘之前 调用指定的函数 来更新动画。

该方法是用一个回调函数作为参数，这个回调函数会在浏览器重绘之前调用。

requestAnimationFrame() 方法告诉浏览器您希望执行动画并请求浏览器在下一次重绘之前调用指定的函数来更新动画。该方法使用一个回调函数作为参数，这个回调函数会在浏览器重绘之前调用。

15 什么是 按需加载？？

当用户触发了动作时才加载对应的功能。触发的动作，是要看具体的业务场景而言，包括但不限于以下几个情况：鼠标点击、输入文字、拉动滚动条，鼠标移动、窗口大小更改等。加载的文件，可以是JS、图片、CSS、HTML等。

16 什么是 虚拟DOM(virtual dom)？？

virtual DOM 的本质就是在 JS 和 DOM 之间做了一个缓存。

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所构建的真正的DOM树上，视图就更新了。Virtual DOM 本质上就是在 JS 和 DOM 之间做了一个缓存。

17 webpack 可以做什么？？

webpack是一个现代的JS应用程序的静态模块打包器（module bundler）。
当 webpack 处理应用程序时，它会递归的构建一个依赖关系图，其中包含应用程序需要的每个模块，然后将这些模块打包成一个或多个bundle。

webpack 是一个现代 JavaScript 应用程序的静态模块打包器(module bundler)。当 webpack 处理应用程序时，它会递归地构建一个依赖关系图(dependency graph)，其中包含应用程序需要的每个模块，然后将所有这些模块打包成一个或多个bundle。

18 ant-design 的优缺点？？

优点：组件非常全面，样式效果也都比较不错。

缺点：框架自定义程度低，默认UI风格修改困难。

19 JS中继承实现的几种方式【多多理解、实践！！！ 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】

1、原型链继承，将父类的实例作为子类的原型，他的特点是实例是子类的实例也是父类的实例，父类新增的原型方法/属性，子类都能够访问，并且原型链继承简单易于实现，缺点是来自原型对象的所有属性被所有实例共享，无法实现多继承，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，即复制父类的实例属性给子类，

构造继承可以向父类传递参数，可以实现多继承，通过call多个父类对象。但是构造继承只能继承父类的实例属性和方法，不能继承原型属性和方法，无法实现函数服用，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，实例继承的特点是不限制调用方法，不管是new 子类（）还是子类（）返回的对象具有相同的效果，缺点是实例是父类的实例，不是子类的实例，不支持多继承

4、拷贝继承：特点：支持多继承，缺点：效率较低，内存占用高（因为要拷贝父类的属性）无法获取父类不可枚举的方法（不可枚举方法，不能使用for in 访问到）

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

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，避免的组合继承的缺点

20 写一个函数 —— 第一秒打印 1， 第二秒 打印2。【let 块级作用域 | 闭包 | setInterval() ？？】

两个方法，第一个是用let块级作用域

```
for(let i=0;i<5;i++){
setTimeout(function(){
console.log(i)
},1000*i)
}
```

第二个方法闭包

```
for(var i=0;i<5;i++){
(function(i){
setTimeout(function(){
console.log(i)
},1000*i)
})(i)
}
```

第3个方法就是结合 IIFE 和 setTimeout传入的第3个参数。

第4个方法就是 fn(i) ， 核心：利用了 函数传值的特点。
完