**[前端面试题2017（篇幅长，附答案）](https://segmentfault.com/a/1190000010384519)**

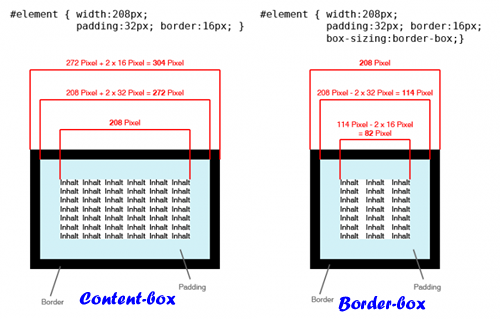
* [javascript](https://segmentfault.com/t/javascript/blogs)
* [css](https://segmentfault.com/t/css/blogs)
* [html](https://segmentfault.com/t/html/blogs)
* [[https://sfault-avatar.b0.upaiyun.com/102/500/1025005915-54cb538538eea_small](https://segmentfault.com/t/html5/blogs)html5](https://segmentfault.com/t/html5/blogs)

**说明：**简答题没有固定答案，以下给出的答案是从别处摘录或自己总结，有错之处欢迎指出。

**html 篇**

1. **标签上title和alt属性的区别是什么？**
   * alt是html标签的属性，而title既是html标签，又是html属性。
   * title作为属性时，用来为元素提供额外说明信息。例如，给超链接标签a添加了title属性，把鼠标移动到该链接上面，就会显示title的内容，以达到补充说明或者提示的效果。
   * alt属性则是用来指定替换文字，只能用在img、area和input元素中，用于网页中图片无法正常显示时给用户提供文字说明使其了解图像信息。注意，alt是替代图像作用而不是提供额外说明文字的。
   * 使用alt属性还具有搜索引擎优化效果，因为搜素引擎是无法直接读取图像的信息的，alt可以为其提供文字信息所以对搜索引擎比较友好。

**css 篇**

1. **relative、absolute、static、fixed的作用，相对谁定位？**
   * static（静态定位）：**occurs where it normally would in the document.**  
     默认值。元素使用正常的布局行为，相对于document，即元素在文档流中当前的布局位置。此时top,bottom, left, right 或者 z-index无效。
   * relative（相对定位）：**you can use top,bottom... to move the element relative to where it would normally occur in the document.**  
     生成相对定位的元素，相对于本该在文档中的位置。通过top,bottom,left,right的设置相对于其正常（原先本身）位置进行定位。可通过z-index进行层次分级。
   * absolute (绝对定位）：**the element will removed from the document and placed exactly where you tell it to go.**  
     生成绝对定位的元素，相对于 static 定位以外的第一个父元素进行定位。元素的位置通过 "left", "top", "right" 以及 "bottom" 属性进行规定。可通过z-index进行层次分级。绝对定位可以设置margin，且不会与其他元素合并。
   * fixed（固定定位）：生成绝对定位的元素，相对于屏幕饰扣进行定位。元素的位置通过 "left", "top", "right" 以及 "bottom" 属性进行规定。可通过z-index进行层次分级。打印时，元素会出现在每页的固定位置。
   * sticky（粘性定位）：是相对定位和固定定位结合，在跨越阈值之前相对定位，之后固定定位。
2. **画出css盒模型**
   * W3C标准盒模型：**width = contentWidth+padding+border+margin。**  
     盒子在页面占据的大小包括了margin，border，padding以及content。而盒子的实际大小包括border，padding以及内容区域content，但是不包括margin。另一点需要说明的是，我们通过JavaScript代码获取某一个元素的大小时，所得到的width和height其实是盒子模型中的content的大小。
   * IE盒模型：**width = contentWidth(包含border+padding)+margin。**  
     IE盒模型也包含margin，border，padding以及content这几部分。IE盒模型的content部分包含了border和padding。
3. **border-box和content-box的区别和使用场景？**  
   为了更能形像看出box-sizing中content-box和border-box两者的区别，我们来个示例图，如下所示，  
     
   这个box-sizing属性，我们都有哪些使用场景呢？
   * 特殊场景的布局：假设我们有这样的一个场景，设置子类元素的margin或者border时，可能会撑破父层元素的尺寸，这时我就需要使用box-sizing: border-box来将border包含进元素的尺寸中，这样就不会存在撑破父层元素的情况了。
   * 统一风格的表单元素：表单中有一些input元素其实还是展现的是传统IE盒模型，带有一些默认的样式，而且在不同平台或者浏览器下的表现不一，造成了表单展现的差异。此时我们可以通过box-sizing属性来构建一个风格统一的表单元素。
4. **css清除浮动的方法**
   * 设置好一个类用来clearfix，再把该类名加至父元素上：
   * .clearfix:before, .clearfix:after {
   * content: " ";
   * display: table;
   * }
   * .clearfix:after {clear: both;}

.clearfix: {zoom: 1}

clearfix技巧是基于在父元素上使用":before"和":after"两个伪类。我们可以在浮动元素的父容器前面和后面创建隐藏元素，":before"用来防止元素顶部的外边距塌陷，用display:table来创建一个匿名的table-cell元素，也确保IE6/7下一致":after"用来防止子元素底部的外边距塌陷，以及用来清除浮动元素。缺点是代码多一点，但还好，用的人也多，兼容性也好。

* + overflow方法：在父元素上设置overflow: hidde/auto。使用auto在IE中有一个滚动条，最好还是用hidden。缺点：如果有元素需要移动到父容器外面，就回隐藏，或者给子元素在没有padding下做个阴影，也会被父元素切断
  + 在父容器结束标签前加一个空标签，并设置clear:both。但如果有很多元素都包含浮动元素，就麻烦，语义化也不好。

1. **谈谈你是怎么适配移动端各种尺寸手机的，以设计稿宽度为375px为例?**
2. **根据如下代码，判断p元素中文字的颜色?**
3. <style>
4. .blue{color: blue;}
5. .red{color: red;}
6. </style>

<p class="blue red">内容</p> //red

**javascript 篇**

1. **代码题**
2. function fun(n, o) {
3. console.log(o);
4. return {
5. fun: function(m) {
6. return fun(m, n)
7. }
8. }
9. }

var a = fun(1).fun(2).fun(4).fun(8) //undefined,1,2,4

var User = {

count: 1,

getCount: function() {

return this.count

}

}

var fn = User.getCount

console.log(fn()); //undefined

function test() {

for(var i=0; i<5; i++) {

setTimeout(function() {

console.log(i)

}, 0)

}

}

test(); //5,5,5,5,5

var o = {

fn: function() {

console.log(fn)

}

}

console.log(o.fn()); //ReferenceError

function Animal(name) {

this.name = name;

}

Animal.prototype.sayName = function() {

console.log(this.name)

}

function Cat(name) {

Animal.call(this, name)

}

var kat = new Cat('Jim')

kat.sayName();

//TypeError: kat.sayName is not a function

//除非加上 Cat.prototype = new Animal()

1. **写一个数组去重的方法**
2. /\*\* 方法一：
3. \* 1.构建一个新的数组存放结果
4. \* 2.for循环中每次从原数组中取出一个元素，用这个元素循环与结果数组对比
5. \* 3.若结果数组中没有该元素，则存到结果数组中
6. \* 缺陷：不能去重数组中得引用类型的值和NaN
7. \*/
8. function unique(array){
9. var result = [];
10. for(var i = 0;i < array.length; i++){
11. if(result.indexOf(array[i]) == -1) {
12. result.push(array[i]);
13. }
14. }
15. return result;
16. }
17. // [1,2,1,2,'1','2',0,'1','你好','1','你好',NaN,NaN] => [1, 2, "1", "2", 0, "你好",NaN,NaN]

// [{id: '1'}, {id: '1'}] => [{id: '1'}, {id: '1'}]

//方法二：ES6

Array.from(new Set(array))

// [1,2,1,2,'1','2',0,'1','你好','1','你好',NaN,NaN] => [1, 2, "1", "2", 0, "你好", NaN]

1. **写一个方法，将URL中的查询字符串解析成一个字典对象**
2. var path = 'www.u.com/home?id=2&type=0&dtype=-1';
3. function parseUrl(url) {
4. var result = [];
5. var tmp = (url.split('?'))[1];
6. var tmpArr = tmp.split('&');
7. tmpArr.map(function(v, k) {
8. var value, key;
9. var obj = {};
10. key = (v.split('='))[0];
11. value = (v.split('='))[1];
12. obj[key] = value;
13. result.push(obj)
14. })
16. return result;
17. }

console.log(parseUrl(path)); //[{id: '2'},{type: '0'},{dtype: '-1'}]

//功能：获取url中的参数的值

function UrlSearch(url) {

var name,value;

var str = url; //取得整个地址栏

var num = str.indexOf("?");

str = str.substr(num+1); //取得所有参数 stringvar.substr(start [, length ]

var arr=str.split("&"); //各个参数放到数组里

for(var i=0;i < arr.length;i++){

num = arr[i].indexOf("=");

if(num>0){

name=arr[i].substring(0,num);

value=arr[i].substr(num+1);

this[name]=value;

}

}

}

// 测试

var path = 'www.u.com/home?id=2&type=0&dtype=-1';

var Request= new UrlSearch(path); //实例化

console.log(Request); //[{dtype: '-1'},{id: '2'},{type: '0'}]

var nickname = encodeURIComponent(Request.dtype);

console.log(nickname); //-1

//功能：修改url参数值

function changeURLArg(url,arg,arg\_val){

var pattern=arg+'=([^&]\*)';

var replaceText=arg+'='+arg\_val;

if(url.match(pattern)){

var tmp='/('+ arg+'=)([^&]\*)/gi';

tmp=url.replace(eval(tmp),replaceText);

return tmp;

}else{

if(url.match('[\?]')){

return url+'&'+replaceText;

}else{

return url+'?'+replaceText;

}

}

return url+'\n'+arg+'\n'+arg\_val;

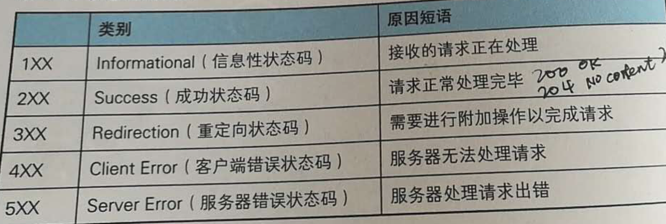
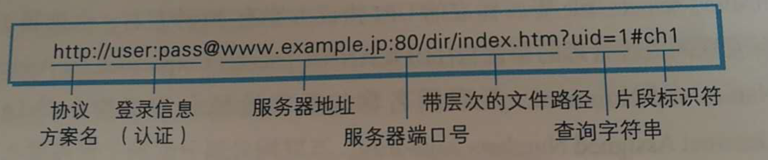
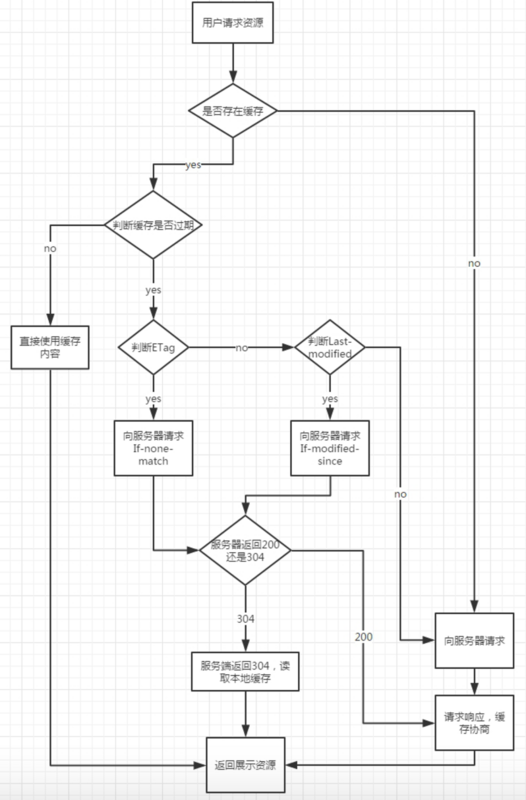
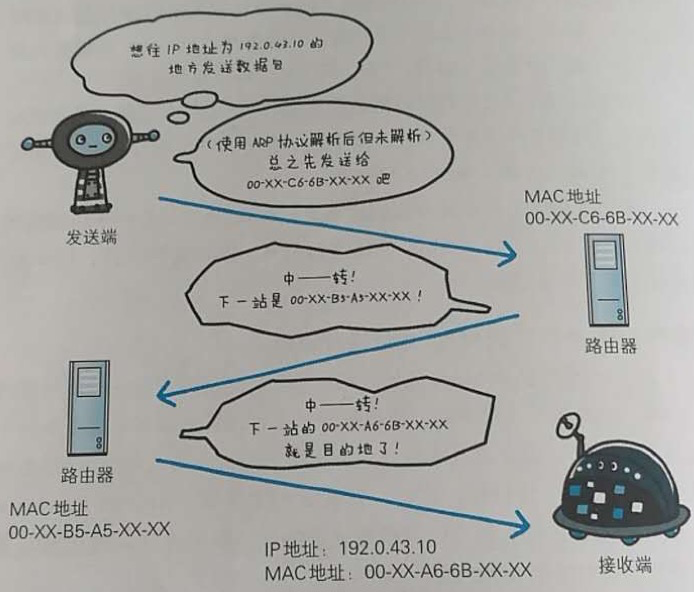
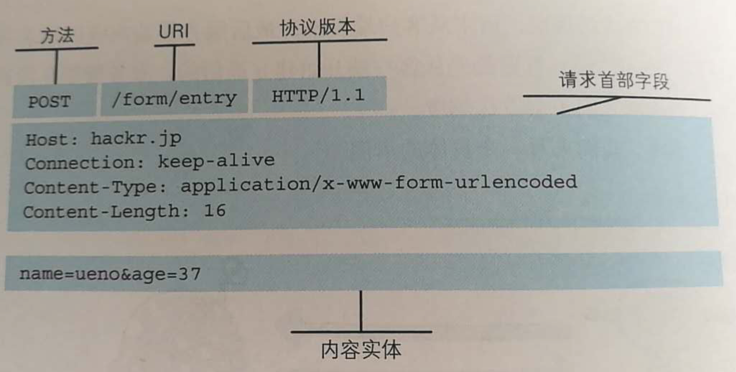
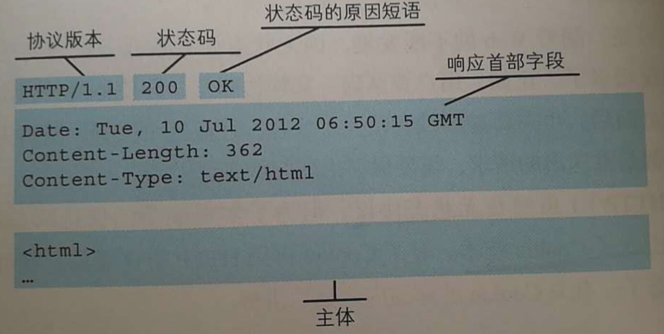
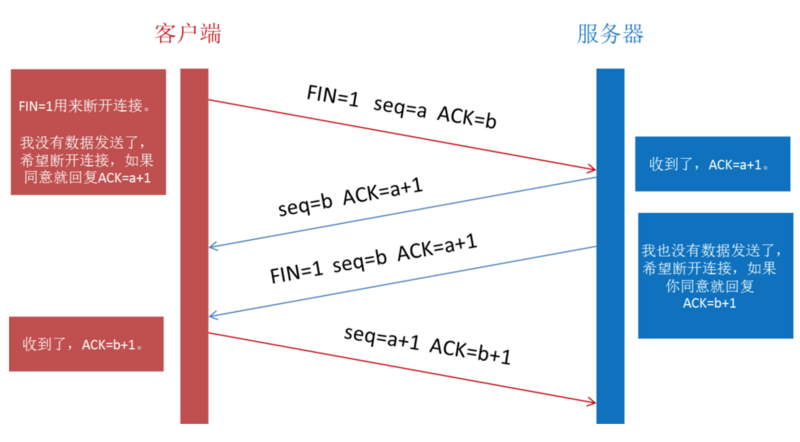
}

var path = 'www.u.com/home?id=2&type=0&dtype=-1';

console.log(changeURLArg(path,'type',4)); //"www.u.com/home?id=2&type=4&dtype=4"

1. **Ajax请求的时候get和post的区别**
2. **谈谈cookie，sessionStorage, localStorage的区别和优缺点**
3. **你知道的性能优化的方法**

**http 篇**

1. **http状态码有哪些？分别表示什么意思？**  
   状态码告知从服务器返回的请求结果。  
   **2XX表明请求被正常处理了**。200OK、204No Content（服务器接收的请求已经处理成功，但在返回的响应报文中不包含实体的主体部分）、206Partial Content(客户端进行了范围请求，对资源的某一部分请求)；**3XX重定向**，301moved permanently(资源已分配新的uri)、302 found(本次使用新uri访问)、303 see other(以get定向到另一个uri)、304 not modified、307 temporary redirect(不会从post改为get)；**4XX客户端错误**，400 bad request（请求报文中存在语法错误），402 unauthorized（发送的请求需要通过http认证）、403 forbidden（服务器拒绝了对资源的访问）、404 not found（服务器上没有请求的资源）；**500XX服务器错误** 500internal server error（服务器在执行是发生了错误）、503 service unavailable（服务器正忙或停机维护）。
2. **一个页面从输入 URL 到页面加载完的过程中都发生了什么事情？**  
   总的来说，根据web浏览器地址栏中指定的URL，web浏览器从web服务器端获取资源文件等信息，从而显示出web页面。具体而言，有如下几个步骤：
   * 1.浏览器接收URL(包含的信息：协议方案名、服务器地址:服务器端口号、带层次的文件路径、查询字符串？、片段标识符)  
     
   * 2.将URL与缓存进行比对如果请求的页面在缓存中且未过期，则直接进行第8步。  
     
   * 3.负责域名解析DNS服务。系统进行DNS查找，并将IP地址返回给浏览器。  
     DNS服务适合HTTP协议一样位于应用层的协议。它提供域名到IP地址之间的解析服务。可以通过域名查找IP地址，也可以逆向从IP地址反查找域名。
   * 4.浏览器与服务器建立TCP连接（在传输层）  
     这一步很复杂因为用到了HTTPS。三次握手(SYN,SYN/ACK,ACK)，若有一次任何一方收不到信息那么将会重新发起 TCP 连接。
   * 5.浏览器向服务器通过TCP协议连接发送HTTP请求。  
     数据经过应用层、传输层、网络层、链路层，传输到下一个目的地。**应用层**决定了向用户提供应用服务时的通信（http协议，DNS服务）。**传输层**对上层应用层，提供处于网络连接中的两台计算机之间的数据传输（TCP协议和UDP协议）。**网络层**用来处理网络上流动的数据包，数据包是网络传输的最小数据单位，该层规定通过怎样的路径到达对方计算机，并把数据传输给对方。**链路层**处理连接网络的硬件部分（网卡，光纤）。这一步骤还会涉及发送给服务器的请求报文（请求方法、请求URI、协议版本、首部字段、内容实体）。
   * 6.服务器收到请求，从它的文档空间中查找资源并返回HTTP响应。  
     服务器返回的响应报文包括协议版本、状态码、解释状态码的原因短语、响应首部字段、实体主体。
   * 7.浏览器接受 HTTP 响应，检查 HTTP header 里的状态码，并做出不同的处理方式。关于返回的状态码的具体说明看上个问题。
   * 8.如果是可以缓存的，这个响应则会被存储起来。  
     根据首部字段判断是否进行缓存。例如，Cache-Control, no-cache(每次使用缓存前和服务器确认)，no-store 绝对禁止缓存。
   * 9.浏览器解码响应  
     浏览器拿到index.html文件后，就开始解析其中的html代码，遇js/css/image等静态资源时，就向服务器端去请求下载；解析成对应的树形数据结构DOM树、CSS规则树，Javascript脚本通过DOM API和CSSOM API来操作DOM树、CSS规则树。
   * 10.渲染页面
   * 11.关闭TCP连接或继续保持连接  
     通过四次挥手关闭连接(FIN ACK, ACK, FIN ACK, ACK)。
3. **页面渲染的过程**  
   页面渲染过程总的来说是，浏览器拿到html文档，自上而下构建dom树，再根据dom树构建渲染树（和dom树有区别，浏览器解析html文档和解析css样式形成dom树和cssdom树，结合这两种才会构建渲染树）。这里有一种情况，如果在head中写入link某个css文件，而后插入script标签，引入一个巨大的js文件，由于浏览器下载css文件需耗时间，这个时候浏览器还在自上而下地去解析dom树，但是遇到script标签，并且由于样式表没有下载下来，阻塞了构建渲染树。一旦css文件加载完成，会结合刚在的dom树构建渲染树，显示到页面，接续加载js文件。但是恰巧js文件巨大，阻塞了浏览器向下解析，等待这个巨大的js文件下载完成后才继续向下解析。如此反复， 最后将渲染树渲染到页面。
4. **jsonp原理**

**react 篇**

1. **react生命周期**
2. **组件间的通信**

**算法篇**

1. **写一个大整数乘法计算的方法**  
   假如一个数很大很大，计算的时候会导致精度丢失的问题。网上有各种解法，大家可以自行google。这里我给出的思路是把整数转换成字符串类型的数据，再把转换后的数据拆分成单个数据，分别计算单个数据相乘，最后相加并把结果转换回number类型。不知可不可行。。。
2. **写出一种排序算法**  
   之前整理过一篇blog，适合算法小白入门——[几种基本排序算法](https://segmentfault.com/a/1190000008778205)

输入url到页面展示参考：[http://www.jianshu.com/p/71cf...](http://www.jianshu.com/p/71cf7f69eca8)