**知识点汇总**

1. **从输入网址到最后浏览器呈现页面内容，中间发生了什么？**

1.浏览器需要获取服务器的IP地址，具体的做法就是发送一个UDP的包给DNS服务器，DNS服务器会返回coder.com的IP, 这时候浏览器通常会把IP地址给缓存起来，这样下次访问就会加快。

2.浏览器与服务器建立TCP链接，建立好后发送HTTP请求。（HTTP服务端口默认为80）。

3.请求到达WEB服务器（如ngnix），WEB服务器需要判断是静态请求（如获取css、js、图片）还是动态请求（需要处理业务逻辑），是静态的话读取相关文件直接返回，是动态的话需要转发给（负载均衡）后端服务器（tomcat）处理。

4.后端服务器处理传过来的数据，分发给下面的控制器来进行处理（访问数据库），处理完后把响应Response发送给Ngnix，Ngnix又转发给浏览器。（TCP链接在HTTP1.1下默认为不关闭，HTTP1.0看是否有参数Connection：keep-alive）。

5.浏览器获取到Response后开始准备页面，下载相关的资源（建立多个TCP链接（最多6个）并行的下载），将资源缓存到本地，下次打开更快。

6.之后浏览器将HTML变为DOM Tree，将CSS变为CSS Rule Tree，JS修改DOM Tree，渲染出相关内容。

参考文章：https://mp.weixin.qq.com/s/3\_DZKSP492uq9RfQ3eW4\_A

**其他答案：**

(1)操作系统(window/linux/macos)访问网络上DNS服务器，把域名转换IP地址

(2)浏览器通过ip向淘宝服务器发送HTTP请求

(3)服务器接收请求，解析请求，查找资源,

可能访数据库,构建响应消息，发送浏览器中

(4)浏览器接收并解析响应消息

(5)浏览器缓存接收响应主体，解析和渲染响应内容

1. **CSS中使元素水平垂直居中**
2. 在 CSS 中对元素进行水平居中是非常简单的：如果它是一个行内元素， 就对它的父元素应用 text-align: center; 如果它是一个块级元素，就对它自身应用 margin: auto。
3. 对于垂直居中目前有以下几种方案：

HTML演示代码及方法如下（我们想要.box相对于.container居中）：

<div class="container">

<div class="box">

<p class=”text”>center me!</p>

</div>

</div>

.container{

position: relative;

background-color: #333;

height: 450px;

}

* 1. 基于绝对定位的居中方法（position: absolute）

**局限性：它要求居中元素具有固定的宽度和高度。**

**1.先把这个元素的左上角放置在视口(或最近的、具有定位属性的祖先元素)的正中心**

**2.然后再利用负外边距把它向左、向上移动(移动距离相当于它自身宽高的一半)，从而把元素的正中心放置在视口的正中心**

.box{

positon: absolute;

left: 50%;

top: 50%;

margin-left: -100px;

margin-top: -100px;

background-color: #ffff00;

width: 200px;

height: 200px;

}

* 1. 一个不需要居中元素固定宽高的方案（transform: translate(-50%, -50%)）。

**局限性：必须要使用绝对定位，有时候绝对定位对布局影响太大。**

**在某些浏览器中，这个方法可能会导致元素的显示有一些模糊，因为元素可能被放置在半个像素上。这个问题可以用 transform- style: preserve-3d 来修复。**

.box{

position: absolute;

left: 50%;

top: 50%;

transform: translate(-50%, -50%);

background-color: #ffff00;

}

* 1. 基于视口viewport。

**局限性：这个技巧的实用性是相当有限的，因为它只适用于在视口中居中的场景（感觉比绝对定位的局限性还大）。**

.box{

width: 18em;

padding: 1em 1.5em;

margin: 50vh auto 0;

transform: translateY(-50%);

background-color: #ffff00;

}

* 1. 基于flexbox弹性布局（display: flex）

.container{

display: flex;

min-height: 450px;

background-color: #333;

}

.box{

margin: auto;

background-color: #ffff00;

}

**Flexbox 的另一个好处在于，它还可以将匿名容器(即没有被标签包裹的文本节点)垂直居中。如下：**

.text{

display: flex;

align-items: center; justify-content: center; width: 18em;

height: 10em;

}

参考文章：<https://segmentfault.com/a/1190000017539123#articleHeader0>

<http://csscoke.com/2018/08/21/css-vertical-align/>

<https://segmentfault.com/a/1190000016389031>

1. **for...in...和for...of...和forEach**

1.推荐在循环对象属性的时候，使用for...in,在遍历数组的时候的时候使用for...of。

2.for...in循环出的是key，for...of循环出的是value

3.注意，for...of是ES6新引入的特性。修复了ES5引入的for...in的不足

4.for...of不能循环普通的对象，需要通过和Object.keys()搭配使用

5.forEach() 方法对数组的每个元素执行一次提供的函数

var arrayTest = ['a', 12, {a: "uoy", 12: 158}];

arrayTest["name"] = "lee";

for(var key in arrayTest){

console.log(arrayTest[key]);

}

for(var val of arrayTest){

console.log(val);

}

Array.prototype.method=function(){

console.log(this.length);

}

var myArray=[1,2,4,5,6,7]

myArray.name="数组"

for (var index in myArray) {

console.log(myArray[index]);

}

输出结果：

a

12

{12: 158, a: "uoy"}

lee

a

12

{12: 158, a: "uoy"}

1

2

4

5

6

7

数组

ƒ (){

console.log(this.length);

}

使用for in会遍历数组所有的可枚举属性，包括原型。例如上栗的原型方法method和name属性。

所以for in更适合遍历对象，不要使用for in遍历数组。

forEach用法：

myArray.forEach(function(val, index, arraySelf){

console.log(index + "-" + val);

})

注意：没有办法中止或者跳出 forEach() 循环，除了抛出一个异常。break和return都不会生效。

参考文章：<https://segmentfault.com/q/1010000006658882>

<https://www.jianshu.com/p/c43f418d6bf0>

1. **JS中数组和字符串常用函数总结**

**详见思维导图数组和字符串部分。**

1. **事件捕获、事件冒泡和事件委托（代理）**

事件流描述的是从页面中接受事件的顺序，分为冒泡流和捕获流；

事件冒泡是指事件从最具体的元素接收，然后逐级向上传播，直到不具体的节点（通常指文档节点）；而事件捕获相反，它是从不具体的节点开始，逐步到最具体的节点；

IE的事件流是冒泡流，而火狐和Chrome同时支持冒泡流和捕获流；

**阻止事件冒泡：e.stopPropagation()，IE则是使用e.cancelBubble = true**

**阻止默认行为：e.preventDefault(),IE则是event.returnValue = false**

1.事件冒泡和事件捕获

注：绑定事件方法（addEventListener）的第三个参数，就是控制事件触发顺序是否为事件捕获。true,事件捕获；false,事件冒泡。默认false,即事件冒泡。所以当你对一个dom节点触发事件时，若外层的节点也设置了相应的事件函数，就会使外层的节点也触发自己的事件函数。

**1.1 设置为false**

<div id="parent">

　　<div id="child" class="child"></div>

</div>

<script type="text/javascript">

document.getElementById("parent").addEventListener("click",function(e){

console.log("parent事件被触发，"+this.id);

},false);

document.getElementById("child").addEventListener("click",function(e){

console.log("child事件被触发，"+this.id);

},false);

</script>

上述代码给child和parent元素都设置了点击事件，并且都是采用的事件冒泡机制。

1.当点击child元素时，先触发child的点击事件，再触发parent的点击事件，打印相应的内容；

2.当点击parent元素时，只触发parent的点击事件，因为parent元素冒泡上去没有dom元素设置了click事件的函数；

**1.2 设置为true**

<div id="parent">

　　<div id="child" class="child"></div>

</div>

<script type="text/javascript">

document.getElementById("parent").addEventListener("click",function(e){

console.log("parent事件被触发，"+this.id);

},true);

document.getElementById("child").addEventListener("click",function(e){

console.log("child事件被触发，"+this.id);

},true);

</script>

上述代码采用事件捕捉

1.当点击child元素时，先触发parent的点击事件，再触发child的点击事件，打印相应的内容；

2.当点击parent元素时，只触发parent的点击事件，因为parent元素从上面捕捉下来一直到parent元素没有dom元素设置了click事件的函数；

当有多个类似的元素需要绑定事件时，一个一个去绑定即浪费时间，又不利于性能，这时候就可以用到事件委托，给他们的一个共同父级元素添加一个事件函数去处理他们所有的事件情况，根据下面代码分析;

**1.3 事件委托演示**

<ul id="1ist">

<li>111</li>

<li>222</li>

<li>333</li>

<li>444</li>

</ul>

<script type="text/javascript">

document.getElementById('1ist').addEventListener('click',function(e){

e.target.innerHTML = "被点击";

});

</script>

e.target表示在事件冒泡中触发事件的源元素，在IE中是e.srcElement

并且e.target有很多属性可以操作

event.target.nodeName 　　//获取事件触发元素标签name（li,p…）

event.target.id　　　　　　//获取事件触发元素id

event.target.className　　//获取事件触发元素classname

event.target.innerHTML　　//获取事件触发元素的内容（li）

参考文章：<https://blog.csdn.net/juzipidemimi/article/details/73388282>

<https://segmentfault.com/a/1190000012729080>

<https://www.cnblogs.com/zhangmingze/p/4864367.html>

1. **Date对象（计算时间差等）**

常用内容巩固：

//获取当前时间

var time1 = new Date();

console.log(time1); //Tue Feb 12 2019 16:22:43 GMT+0800 (中国标准时间)

//生成指定时间

var time2 = new Date("2019/2/12 16:19:25");

console.log(time2); //Tue Feb 12 2019 16:19:25 GMT+0800 (中国标准时间)

//生成指定时间

var time3 = new Date(2019,2-1,12,16,19,25); //月要减一

console.log(time3); //Tue Feb 12 2019 16:19:25 GMT+0800 (中国标准时间)

//生成指定时间（时间戳）

var time4 = new Date(1549959565000);

console.log(time4); //Tue Feb 12 2019 16:19:25 GMT+0800 (中国标准时间)

//获取当前时间的时间戳的方法（其他时间的时间戳只需要在Date()中指定时间即可）

console.log(Date.now()); //1549960354801

console.log(new Date().getTime()); //1549960354801

console.log(new Date().valueOf()); //1549960354801

//时间格式转换

console.log(new Date().toString()); //Tue Feb 12 2019 16:34:24 GMT+0800 (中国标准时间)

console.log(new Date().toLocaleString()); //2019/2/12 下午4:35:16

console.log(new Date().toLocaleDateString()); //2019/2/12

console.log(new Date().toLocaleTimeString()); //下午4:36:22

console.log(new Date().toGMTString()); //Tue, 12 Feb 2019 08:37:13 GMT

//获取时间的各个部分，

console.log(new Date().getFullYear()); //2019 注：getYear()不返回千禧年2000，已废弃

console.log(new Date().getMonth()); //1 注：JS中月份要加一，他是从0开始计算的

console.log(new Date().getDate()); //12 注：表示12号

console.log(new Date().getDay()); //2 注：表示星期二，0表示星期天

console.log(new Date().getHours()); //16 注：24小时制

console.log(new Date().getMinutes()); //19

console.log(new Date().getSeconds()); //47

console.log(new Date().getTime()); //1549961419912

//两日期对象可以相减得到毫秒数，用来计算倒计时

console.log(new Date() - time2);

**七、bind、apply和call方法**

call、apply和bind是Function对象自带的三个方法，都是为了改变函数体内部 this 的指向。

apply 、 call 、bind 三者第一个参数都是 this 要指向的对象，也就是想指定的上下文；

apply 、 call 、bind 三者都可以利用后续参数传参；

bind 是返回对应函数，便于稍后调用，你必须调用它才会被执行；apply 、call 则是立即调用 。

对于 apply、call 二者而言，作用完全一样，只是接受参数的方式不太一样。call是把参数按顺序传递进去，而apply则是把参数放在数组里，bind的参数格式与call类似。

演示代码如下：

var name = "coco", age = 18;

var obj1 = {

name: "lala",

objAge: this.age,

say: function(){

console.log("I am " + this.name + ", age: " + this.age);

},

sayMore: function(loc, dir){

console.log("I am " + this.name + ", age: " + this.age + " From " + loc + " To " + dir);

}

}

console.log(obj1.objAge); //18

console.log(obj1.say()); //I am lala, age: undefined

var other = {

name: "juju",

age: 25

}

console.log(obj1.say.apply(other)); //I am juju, age: 25

console.log(obj1.say.call(other)); //I am juju, age: 25

console.log(obj1.say.bind(other)()); //I am juju, age: 25

console.log(obj1.sayMore.apply(other, ['重庆', '成都'])); //I am juju, age: 25 From 重庆 To 成都

console.log(obj1.sayMore.call(other, '重庆', '成都')); //I am juju, age: 25 From 重庆 To 成都

console.log(obj1.sayMore.bind(other, '重庆', '成都')()); //I am juju, age: 25 From 重庆 To 成都console.log(obj1.sayMore.bind(other, ['重庆', '成都'])()) //I am juju, age: 25 From 重庆,成都 To undefined

其他补充要点：

**1.apply与call**

var array1 = [12,'foo',{name:'Joe'},-2458];

var array2 = ['Doe' , 555 , 100];

Array.prototype.push.call(array1, array2);

// 这里用 call 第二个参数不会把 array2 当成一个数组,而是一个元素

//等价于array1.push(‘‘'Doe' , 555 , 100’’);

//array1.length=5;

Array.prototype.push.apply(array1, array2); // 这里用 apply 第二个参数是一个数组

// 等价于: array1.push('Doe' , 555 , 100);

//array1.length=7;

**2.类数组使用数组方法（变成数组）**

var elems = document.getElementsByTagName("span");

console.log(elems); //有三个span

var sliEle = Array.prototype.slice.apply(elems, [0, 2]); //截取前两个span

console.log(sliEle);

**3.在 Javascript 中，多次 bind() 是无效的。更深层次的原因， bind() 的实现，相当于使用函数在内部包了一个 call / apply ，第二次 bind() 相当于再包住第一次 bind() ,故第二次以后的 bind 是无法生效的。**

var bar = function(){

console.log(this.x);

}

var foo = {

x:3

}

var sed = {

x:4

}

var func = bar.bind(foo).bind(sed);

func(); //3

var fiv = {

x:5

}

var func = bar.bind(foo).bind(sed).bind(fiv);

func(); //3

参考资料：<https://www.cnblogs.com/Shd-Study/archive/2017/03/16/6560808.html>

<https://www.cnblogs.com/zhg277245485/p/6559475.html>

<http://www.admin10000.com/document/6711.html>

**八、写一个兼容全浏览器的去掉字符串前后空格的代码。**

if(!String.prototype.trim){

String.prototype.trim = function(){

return this.replace(/^[\s\uFEFF\xA0]+|[\s\uFEFF\xA0]$/g, '');

}

}

var str = “ asdfg ghj “;

console.log(str.trim()); //asdfg ghj

详细解释：

\s：空格

\uFEFF：字节次序标记字符（Byte Order Mark），也就是BOM,它是es5新增的空白符

\xA0：禁止自动换行空白符，相当于html中的&nbsp;

参考文档：<https://imququ.com/post/bom-and-javascript-trim.html>

**中途穿插（#滑稽）**

分别使用正则和非正则实现1234567890.12格式化为1,234,567,890.12（也就是给数字加千分符）

参考文章：<https://blog.csdn.net/itpinpai/article/details/48002697>

<https://www.jb51.net/article/99675.htm>

var str = '2359844564654';

function test3(str) {

var iNum = str.length % 3; //余数

var prev = '';

var arr = [];

var iNow = 0;

var tmp = '';

if(iNum != 0) {

prev = str.substring(0, iNum); //将余数截取出来

arr.push(prev);

}

str = str.substring(iNum);

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

iNow++;

tmp += str[i];

if(iNow == 3 && tmp) {

arr.push(tmp);

tmp = '';

iNow = 0;

}

}

return arr.join(',');

}

alert(test3(str));

1. **HTTP协议讲解及常见状态码**

1\*\* 信息，服务器收到请求，需要请求者继续执行操作

2\*\* 成功，操作被成功接收并处理

3\*\* 重定向，需要进一步的操作以完成请求

4\*\* 客户端错误，请求包含语法错误或无法完成请求

5\*\* 服务器错误，服务器在处理请求的过程中发生了错误

**常见状态码解释：**

1XX系列：指定客户端应相应的某些动作，代表请求已被接受，需要继续处理。由于 HTTP/1.0 协议中没有定义任何 1xx 状态码，所以除非在某些试验条件下，服务器禁止向此类客户端发送 1xx 响应。

　　2XX系列：代表请求已成功被服务器接收、理解、并接受。这系列中最常见的有200、201状态码。

　　200状态码：表示请求已成功，请求所希望的响应头或数据体将随此响应返回

　　201状态码：表示请求成功并且服务器创建了新的资源，且其 URI 已经随Location 头信息返回。假如需要的资源无法及时建立的话，应当返回 '202 Accepted'

　　202状态码：服务器已接受请求，但尚未处理

　　3XX系列：代表需要客户端采取进一步的操作才能完成请求，这些状态码用来重定向，后续的请求地址（重定向目标）在本次响应的 Location 域中指明。这系列中最常见的有301、302状态码。

　　301状态码：被请求的资源已永久移动到新位置。服务器返回此响应（对 GET 或 HEAD 请求的响应）时，会自动将请求者转到新位置。

　　302状态码：请求的资源临时从不同的URI响应请求，但请求者应继续使用原有位置来进行以后的请求

304自从上次请求后，请求的网页未修改过。服务器返回此响应时，不会返回网页内容。 如果网页自请求者上次请求后再也没有更改过，您应将服务器配置为返回此响应(称为 If-Modified-Since HTTP 标头)。

　　4XX系列：表示请求错误。代表了客户端看起来可能发生了错误，妨碍了服务器的处理。常见有：401、404状态码。

　　401状态码：请求要求身份验证。 对于需要登录的网页，服务器可能返回此响应。

　　403状态码：服务器已经理解请求，但是拒绝执行它。与401响应不同的是，身份验证并不能提供任何帮助，而且这个请求也不应该被重复提交。

　　404状态码：请求失败，请求所希望得到的资源未被在服务器上发现。没有信息能够告诉用户这个状况到底是暂时的还是永久的。假如服务器知道情况的话，应当使用410状态码来告知旧资源因为某些内部的配置机制问题，已经永久的不可用，而且没有任何可以跳转的地址。404这个状态码被广泛应用于当服务器不想揭示到底为何请求被拒绝或者没有其他适合的响应可用的情况下。

　　5xx系列：代表了服务器在处理请求的过程中有错误或者异常状态发生，也有可能是服务器意识到以当前的软硬件资源无法完成对请求的处理。常见有500、503状态码。

　　500状态码：服务器遇到了一个未曾预料的状况，导致了它无法完成对请求的处理。一般来说，这个问题都会在服务器的程序码出错时出现。

　　503状态码：由于临时的服务器维护或者过载，服务器当前无法处理请求。通常，这个是暂时状态，一段时间会恢复

参考文章：<https://www.jianshu.com/p/27862635c077>

<https://blog.csdn.net/q1056843325/article/details/53147180>

**十、移动端 Retina屏 各大主流网站1px的解决方案**

**参考文章：<http://www.cnblogs.com/surfaces/p/5158582.html>**

**<https://www.w3cplus.com/mobile/vw-layout-in-vue.html>**

**十一、BFC是啥子？如何创建BFC？？？**

在一个Web页面的CSS渲染中，块级格式化上下文 (Block Fromatting Context)是按照块级盒子布局的。W3C对BFC的定义如下：

浮动元素和绝对定位元素，非块级盒子的块级容器（例如 inline-blocks, table-cells, 和 table-captions），以及overflow值不为“visiable”的块级盒子，都会为他们的内容创建新的BFC（块级格式上下文）。

为了便于理解，我们换一种方式来重新定义BFC。一个HTML元素要创建BFC，则满足下列的任意一个或多个条件即可：

1、float的值不是none。

2、position的值不是static或者relative。

3、display的值是inline-block、table-cell、flex、table-caption或者inline-flex

4、overflow的值不是visible

BFC是一个独立的布局环境，其中的元素布局是不受外界的影响，并且在一个BFC中，块盒与行盒（行盒由一行中所有的内联元素所组成）都会垂直的沿着其父元素的边框排列。

参考文章：<https://www.cnblogs.com/libin-1/p/7098468.html>

<https://www.cnblogs.com/lhb25/p/inside-block-formatting-ontext.html>

**十二、跨域相关知识点**

简单总结一下：

1. jsonp（使用script、link和img标签跨域，但局限性是只适用于get方法）。
2. CORS（在服务器上设置允许跨域，兼容性较差）。
3. WebSocket。
4. postMessage（跨域DOM操作）。

参考文章：<https://segmentfault.com/a/1190000015597029>

<https://mp.weixin.qq.com/s/Hwwl1V3RcKNWB6oiKnYgSw>

**十三、原生AJAX写法**

var xhr = null;

if(window.XMLHttpRequest){

xhr = new XMLHttpRequest();

}else{

xhr = new ActiveXObject("Microsoft.XMLHttp");

}

xhr.onreadystatechange=function(){

if(xhr.readyState==4 && xhr.status==200){

var str=xhr.responseText;

$("uname-show").innerHTML=str;

}

}

//获取用户名输入框的值

var uname=$("uname").value;

//get请求

xhr.open("get","checkUname.php?uname="+uname,true);

//post请求

xhr.open("post", "day20\_check.php", true);

xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

var name = $("name").value;

xhr.send("uname=" + name);

xhr.send(null);

**十四、前端性能优化**

简略总结一下：

1. 压缩和合并CSS、JS、图片。（gulp、webpack）
2. 使用CDN内容分发网络。
3. 使用缓存（减少HTTP请求）。
4. 使AJAX可缓存。
5. 减少DOM个数。
6. 。。。。。。。（参见雅虎军规）。

参考文章：<https://segmentfault.com/a/1190000015052545>

**十五、浏览器兼容性**

**十六、数组去重**

参考资料：<https://segmentfault.com/a/1190000016418021>