**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

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>