**一个HTML由三部分组成：**

1.HTML：超文本标记语言；2.CSS：层叠样式表；3.JavaScript：动态脚本语言。

**将CSS引用到页面中有四种方式：**

1.行内样式：写在html元素上面；

2.内嵌样式：在head中写一个style标签，把样式写在style中；

3.外链式：在项目中建一个CSS文件，页面中用link把它引入进来；

4.导入式：（@import 路径，不常用）。

**将JavaScript引用到页面中的几种方式：**

1.行内引用：不推荐，安全性能比较低；

2.内嵌式：将JavaScript代码写在script脚本块中间；

3.外链式：将JavaScript代码写在外面文件中，通过src找到引入。

**细节1：**在外链式中，script脚本块中间不可以写JavaScript脚本代码，写了也不会执行

**细节2：**CSS和JavaScript脚本都可以放在HTML页面的任何位置，一般将CSS放在head标签内，将JavaScript脚本放在body标签内的最后面。原因：HTML页面是从上到下依次加载的，JavaScript通常是获取HTML页面中的内容，并且进行相关的操作的，所以需要先加载HTML标签才能再加载我们的JavaScript代码，保证HTML元素加载完成后再加载JavaScript代码

**我们通常将我们的js文件放在body中最后面的位置?**因为JavaScript通常是用来操作我们html元素的，我们需要先加载元素标签在加载我们的JavaScript (这也是我们推荐的写法)，但是，有些时候,我们想把JavaScript代码写在body的前面，也想获取到我们的元素，那需要如何的处理：  
window.onload=function(){  
 //把JavaScript代码写在这里  
}  
当页面中的html标签、图片、文字等都加载完成后，在执行我们的JavaScript代码(这个方法只能在同一个页面中使用1次，使用两次下面的会把上面的覆盖掉)

**JavaScript的组成部分：**

1.ECMAScript：定义了JS的命名规范、变量、数据类型、操作语句等基本的知识（JS的核心）；

2.DOM：document object model 文档对象模型，提供了操作页面中元素的属性和方法；

3.BOM：browser object model 浏览器对象模型，提供了浏览器中常用的属性和方法。

**JavaScript中常用的几种输出方式：**

1.在浏览器的控制台输出  
console.log("要输出的内容!");  
console.dir(); 在控制台输出一个对象的详细信息  
console.table(); 把一个数据(json)按照表格的形式输出，我们在查看结构的时候更加的清晰一些

2.向指定HTML元素中动态添加内容innerText/innerHTML  
innerText和innerHTML的区别：  
1）firefox浏览器中不支持innerText，而innerHTML不存在浏览器兼容问题  
2）innerHTML不仅可以添加文字，还可以添加html标签，而innerText不能识别有效的html标签的，他会把html也当做文字添加进去  
项目中我们统一使用innerHTML，基本不使用innerText

3.在页面中弹出对应的框显示我们的内容  
alert("要输出的内容"); 在浏览器中弹出一个框，<把我们要输出的内容先转化为字符串>，然后在输出  
confirm(); 是我们浏览器中确认选择弹出层，当用户选择的是确定，方法会返回true，如果点击的是取消，返回的是false，我们通过方法的返回值进行对应的相关操作即可！

4.在页面中输出内容 document.write()  
在页面中输出显示我们的内容，经常我们如果要在页面中添加一个广告的话，我们用这个来添加

**浏览器F12打开我们的控制台，例如谷歌浏览器(每一个前端开发必备的浏览器)：**

**Elements:**我们页面中的元素都在这个里面，我们经常用这个来调试我们的样式(在firefox浏览器中我们安装一个fireBug插件也可以实现)

**NetWork:**包含了我们的所有的资源文件(css、js外链文件、html页面、所有的图片、我们向和后台请求的数据接口等)，并且告诉我们每一个文件加载完成的时间 ,对于时间过长的我们进行相关的优化

**Sources:**包含了我们项目中的html、js、css所有的源代码，扒一个网站在这里可以找到所有的源码

**Resources:**当前网站本地存储的数据都在这个里面(cookie、storage)

**Console:**我们页面中在控制台输出的内容就在这里面

有一个小手机的图片，单击后进入我们的手机端模拟器，我们以后的移动端开发就是在模拟器里面完成的

我们还可以用控制台加断点进行页面bug调试

**JS中的命名规范：**

**1、JS中是严格区分大小写的**

test Test 是不同的东西

**2、遵循驼峰命名法**

第一个单词首字母小写，其余后面每一个有意义的单词的首字母大写

命名可以使用数字、字母、\_、$进行命名(数组不能开头)

**3、不能使用关键字和保留字**

关键字：在JS中有特殊意义的 保留字：未来可能会成为关键字的

**变量：**可以改变的量，在JS中的作用就是用来存储值和代表值的

JS中定义一个变量是需要有语法的: **var 变量名=值;** 把值赋给变量名，那么这个变量名就代表了这个值

在JS中：

**=:**代表的是赋值，把值赋给变量 =左边是变量 =右边是值

**==:**判断是否相等 value1==value2 判断value1是否和value2相等，如果相等返回true，不相等返回的是false

**===:**先判断数据类型两边是否相同，若相同，再进行比较，若不同，返回false

**JS中包含的数据类型：**

**基本数据类型(值类型)：**number(数字)、string(字符串)、boolean(布尔类型)、null 空、undefined 空

**引用数据类型:**他的数据结构相对来说复杂一些

对象数据类型 object

{}:Object []:Array /^$/:RegExp(正则表达式)

函数数据类型 function

**number类型：**

正数 负数 0 小数

**NaN:**not a number不是一个数，但是属于number数据类型的

**NaN!=NaN** 因为NaN代表的不是一个数，具体的是啥谁都不知道，没有办法比较是不是相等，**NaN和NaN永远不相等**

**什么情况下出现NaN:**

1、数字+undefined

2、用Number()等将其他的数据类型转换为number类型的时候，如果不能转化为数字，返回NaN

console.log(0==0);//true

console.log(0==1);//false

console.log(NaN==NaN);//false

**Number()：**将其他的数据类型转化为number类型**(强制转化)**。

如果转化的是字符串，必须都是有效的数字字符才可以转换，只有有一个不是返回的就是NaN

Number(true);-->1 Number(false);-->0 Number("");-->0 Number("13");-->13

console.log(Number("13")); //输出的是数值13

将其他的数据类型转化为number类型:**parseInt/parseFloat(非强制转化)**:从左到右，把找到的有效的数字返回即可

**parseInt();** 从左到右依次取整数遇到非数字停止

**parseFloat();** 从左到右依次取小数（浮点数）遇到非数字停止

Number("14.3px") -->NaN parseInt("14.3px") -->14 parseFloat("14.3px") -->14.3

**isNaN:**检测一个值是否为有效的数字(检测一个值不是有效数字的观点是否正确)，如果值不是一个有效数字返回的true，是有效的数字返回false

isNaN()在判断一个完全是"number"例如（"888"）返回fasle，否则返回true；如果被判断字符串是例如("888px")会发生强制类型转换，将字符串强制转化为数字类型Number("888px");

isNaN("13")-->false：首先调用Number()将"13"-->13 然后isNaN(13)-->false

**string类型：**

1+"13" ==>"113" 在JS中+不仅仅是加法运算符，如果遇到的是数字+字符串 属于字符串拼接

- \* / %:这些都是数学运算符 1-"13"=-12

**Boolean类型：**

只有true/false

Boolean() 将其他的数据类型转换为布尔类型

只有**0 NaN "" null undefined** 这五个会变为false，其他的都是真

**!：**首先将值转换为我们的布尔类型，然后在对布尔值取反

**!!：**和Boolean()作用相同，将值转换为布尔类型

**0和""**也带表的是没有

**null 和 undefined**都代表空(没有)

**0 ""和null**的区别：""挖坑了但是没种树 null是懒到连坑都不挖了

**null 和 undefined：**null是指现在没有，以后可能会有值；undefined是指现在没有，将来也不会有。

两个对象类型的值做比较返回值永远为假

对象==字符串 把对象强制转化为字符串然后在比较

对象 == 布尔 把对象和布尔值全部转化为数字然后再进行比较

对象 == 数字 把对象转化为数字然后在比较

数字 == 布尔 把布尔转化为数字

数字 == 字符串 把字符串转化为数字

布尔 == 字符串 都转化为数字

null == undefined -->ture真

null & undefined 和任何比较都是false

三个===比较是先比较数据类型。若数据类型相同，再比较值是否相等，相等反回true，否则返回false；若数据类型不同，直接返回false.

**对象数据类型：**

每一个对象数据类型都是由:属性名和属性值组成的，用来描述这个对象的特征的，我们把属性专业的叫法是 键值对(key:value)（一组属性名和属性值）

JS规定一个对象的属性名不能重复

对象的属性名是可以为数字的：我们只能用对象名[数字属性名]来操作，不能用对象.数字属性名 例如：personInfo[1] personInfo["1"] 不能personInfo.1

**给对象增加属性名和属性值的方法：**

可以直接在｛｝中书写

对象名.属性名=属性值 personInfo.age=17;

对象名["属性名"]=属性值 (千万不要忘记给属性名加"")

personInfo["age"]=17; personInfo.age=17;

如果之前没有这个属性名是新增，有的话相当于修改值

**查找：**

console.log(personInfo.name);

console.log(personInfo["name"]);

如果对象并没有该属性，默认的属性值是undefined

console.log(personInfo["height"]);//undefined

**删除：**

personInfo.age=null;//假删除

delete personInfo.age;//真正删除 返回的值是true

**JS中浏览器提供给我们四种检测数据类型的方式**

**1、typeof** 检测数据类型的运算符 typeof 要检查的值==>返回的结果是数据类型对应的字符串，例如："number"、"string"、"boolean"、"undefined"、"object"、"function" 返回的结果首先是一个字符串，其次里面才包含了对应的数据类型

特殊记忆：

1)**typeof null的结果是"object"**而不是"null"，但是**typeof undefined的结果就是"undefined"**

2)typeof是存在局限性的：不能细分object下的数组、正则等详细的数据类型，typeof []==>"object"

**2、instanceof**

**3、constructor**

**4、Object.prototype.toString.call()**

**基本数据类型和引用数据类型的本质区别：**

**基本数据类型是直接的操作值的，**例如：var num1=12; 是直接的把12这个值给num1这个变量

**引用数据类型不直接操作值，操作的是内存地址。**首先开辟一个新的存储的空间(有一个地址，我们假设为xxxfff000)，然后把对象中的属性名和属性值存到这个空间中，然后把地址给变量，所以引用数据类型存储的只是一个虚拟的地址，接下来的所有的操作都要通过地址找到空间，对空间里面的内容进行操作，例如：var obj={name:"珠峰"};

1、首先开一个空间(地址：xxxfff000)

2、把属性名和属性值 name:"珠峰" 存储到空间中

3、把xxxfff000这个地址给我们的obj变量

**数据类型转换：**

=:赋值

=:比较是否相等，在比较的时候，遇到左右数据类型不一样了，默认先转化成一样,然后再比较

===:绝对相等，只能数据类型一样才能比较，遇到数据类型不一样不能自动转换

console.log(1=="1");//true

console.log(1==="1");//false

==在比较的时候value1==value2，数据类型间的转换规则:

1、对象和对象永远不相等

2、对象和字符串：对象-->字符串 调用toString方法

3、对象和布尔：对象-->字符串-->数字 布尔-->数字 字符串和布尔转化为数字调用Number() ""-->0 true-->1 false-->0

4、对象和数字: 对象-->字符串-->数字 []-->""-->0

5、字符串和布尔：都转化为数字

6、字符串和数字：字符串-->数字

7、布尔和数字：布尔-->数字

8、null和undefined相等 null==undefined;//true null===undefined;//false

9、null和undefinend跟其他任何的数据类型都不相等

10、NaN和任何东西都不相等，和自己也不相等

**检测JS中数据类型的几种方法：**

**一、typeof**

用typeof检测数据类型返回的首先是一个字符串，然后包含我们的数据类型信息，例如："number"、"string"、"boolean"、"undefined"、"object"、"function"

typeof null; -->”object”

typeof undefined; -->”undefined”

一个变量如果没有定义，不能直接读操作，但是可以赋值;

一个变量没有定义，是可以用typeof去运算的

局限性：typeof [] ==> "object" 不能具体检测对象数据类型细分的那几个(数组、正则...)

**二、instanceof 检测某一个实例是否属于这个类**

var ary = [1, 2, 3];

console.log(ary instanceof Array);//如果是true，说明ary是Array的一个实例，也就是ary是一个数组，反之，返回false则不是一个数组

console.log(ary.constructor == Array);//通过**constructor**也可以

局限性:在关于类之间通过原型链的方式实现继承的时候，instanceof检测出来的结果是不准确的

**三：toString检测数据类型**

Array --> toString 将数组转换为字符串

RegExp --> toString 将正则转换为字符串

Date --> toString 将时间格式的数据转换为字符串

......

toString检测数据类型的原理：执行Object原型上的toString方法，让这个方法中的this变为要检测的那个值，就是实现了数据类型检测

Object --> toString 不是将对象转换为字符串，而是返回当前检测数据所对应的类的信息

"[object 所属的类]"，例如："[object Number]"、"[object String]"、"[object Boolean]"、"[object Null]"、"[object Undefined]"、"[object Object]"、"[object Array]"、"[object RegExp]"、"[object Date]"、"[object Function]".....

var obj = {name: ""};

console.log(obj.toString());//不是"{name:''}" 而是"[object Object]"

null和undefined其实是一个特殊的Object数据类型，没有任何的方法，因为它们俩在JS中，只起标识的作用.把这两个值上的所有的方法和属性给屏蔽了。

**null：**

1、手动赋值,后期会有值，初期用一个null占着位置 var num=null;

2、获取HTML元素的时候，如果元素不存在，得到的结果就是null。例如：getElementById、previousSibling、offsetParent...

3、用正则捕获的时候，如果没有捕获到内容结果就是null

**undefined：**

1、预解释的时候，只声明未定义，默认值是undefined

2、void 0; 就是undefined

3、方法没有返回值，默认的返回值是undefined

4、定义形参没有传值，默认值也是undefined

5、获取一个对象不存在的属性名的属性值，默认是undefined