es12新特性

# 1.新增逻辑操作符

|  |
| --- |
|  |

## 小案例

|  |
| --- |
| <script>          //逻辑赋值操作符 ??=,&&= ,||=          // let a=0,b=5          // a &&=b      //    console.log("a的新值:",a);//0          // a ||=b          // console.log("a的新值:",a) //5          // a ??=5          // console.log("a的新值:",a);//0          let c=null,d=10          //  c &&=d          //  console.log("c的新值:",c);//null          //   c ||=d          //   console.log("c的新值:",c) //10               c??=d               console.log("c的新值:",c) //10      </script> |
|  |

# 2.数字分隔符

|  |
| --- |
|  |

## 小案例

|  |
| --- |
| <body>      <script>          let num = 1\_000\_000.00 //1000000 其实是普通数字，只不过他允许你用一个分隔符来书写          console.log(num);          let num2 = num +1000          console.log(num2);//1001000      </script>  </body> |

# 3.字符串的replaceAll方法

|  |
| --- |
|  |

## 小案例

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>es12新特性3-str.replaceAll</title>  </head>  <body>      <script>          let str = `I don't fucking care about what the fuck you tell me!just get the fuck out of here,y fucking ass hole!!`          let res = str.replaceAll('fuck','\*')          console.log(res);      </script>  </body>  </html> |
|  |

## 注意：上面的功能如果使用str.replace方法来做，需要传递一个正则表达式

|  |
| --- |
| <script>          let str = `I don't fucking care about what the fuck you tell me!just get the fuck out of here,y fucking ass hole!!`          // let res = str.replaceAll('fuck','\*')          // console.log(res);          let res2 = str.replace(/fuck/g,'\*')          console.log(res2);      </script> |
|  |

# 4.Promise.any，特点是只要有一个promise返回成功，最终结果就是成功，只有当所以的promise都返回rejected，最终结果才是rejected

|  |
| --- |
|  |

## 小案例：模拟ajax请求访问本地文件，有些实际路径，有效不存在会引发错误，我们来看看效果

|  |
| --- |
|  |
| <body>      <script>          function myAjax(url) {          return new Promise(function(resolve,reject) {              let xhr = new XMLHttpRequest()              xhr.open("get",url,true)              xhr.send()              xhr.onreadystatechange = function () {                  if(xhr.readyState === 4){                      if(xhr.status>=200 && xhr.status<=300){                         resolve(JSON.parse(xhr.responseText))                      } else {                          reject(xhr.responseText)                      }                  }                }          })        }      let arr = [myAjax("1.json"),myAjax("2.json"),myAjax("3.json")]      // let arr = [myAjax("3.json"),myAjax("1.json"),myAjax("2.json")]      // let arr = [myAjax("3.json"),myAjax("2.json"),myAjax("1.json")]      // let arr = [myAjax("ok.json"),myAjax("2.json"),myAjax("1.json")]      let ret = Promise.any(arr)      // console.log(ret);      ret.then(data=>console.log(data)) //{data: 1000} 如果你的任务列表中有一个是成功的它可以拿到，如果有多个成功，它也只能拿到任意成功的      </script>  </body> |
|  |
|  |

### 你会发现这个函数是随机获取结果的，运行多次，得到的结果是不一样的

### 注意：

### 1.这个方法只要你有一个Promise执行成功，它就一定能够拿到能够成功的结果，当使用promise都失败，才会被catch捕获，返回rejected结果

### 2.即使你有多个Promise执行成功，它也只能够拿到一个最快返回的成功的结果，这个过程是随机的

### 3.也就是说它的结果只是一个promise而不是一个promise数组

### 注意Promise.all,Promise.race,Promise.allSettled和Promise.any的区别和使用场景

# 5.弱引用WeakRefs

## 在学习弱引用之前，我们先来看看weakset和weakmap

### weakSet和weakMap一样，都是用来防止内存泄漏的，在weakset里面只要你把里面的一个对象赋值为空，该对象就会被weakset删除，就是这个特性可以防止内存泄漏的，weakset不会使用引用计数。weakSet，weakMap

|  |
| --- |
| <script>          let obj1 = {              name:"Jack"          }          let obj2 = {              name:"Jackline"          }          let ws = new WeakSet([obj1,obj2])          console.log(ws);        </script> |
|  |
| <script>          let obj1 = {              name:"Jack"          }          let obj2 = {              name:"Jackline"          }          let ws = new WeakSet([obj1,obj2])          console.log(ws);           obj2 = null           console.log(ws);      </script> |
|  |

### 看看weakmap

|  |
| --- |
| <script>          let obj1 = {              name:"Jack"          }          let obj2 = {              name:"Jackline"          }          let wm = new WeakMap()          wm.set(obj1,18)          wm.set(obj2,28) //weakmap的键只能是对象类型的          console.log(wm);        </script> |
|  |
| <script>          let obj1 = {              name:"Jack"          }          let obj2 = {              name:"Jackline"          }            let wm = new WeakMap()          wm.set(obj1,18)          wm.set(obj2,28) //weakmap的键只能是对象类型的          console.log(wm);          obj1 = null //当weakmap的一个键为null，weakmap就会把它删除          console.log(wm);      </script> |
|  |

### 注意：weakSet和weakmap是不能使用for循环遍历的，有没有size属性

### 课堂小案例

|  |
| --- |
|  |
|  |

#### 上面的方法是删除不了weakmap里面的数据的，因为这里需要于有两个引用，你删除了按钮能够引用，没有影响到weakmap里面的引用，如果需要从weakmap里面删除这个数据需要把like赋值为null或者直接把节点添加到weakmap或者使用WeakRefs

|  |
| --- |
|  |

### 然后移除的时候也得这么写

|  |
| --- |
|  |
|  |

## 这种写法非常麻烦，所以就有了我们下面的弱引用

|  |
| --- |
|  |

### 小案例

|  |
| --- |
| <script>          let obj1 = {              name:"Jack"          }            //弱引用          let wobj = new WeakRef(obj1)          console.log(wobj);        </script> |
|  |

### 然后我们将obj1赋值为null

|  |
| --- |
| <script>          let obj1 = {              name:"Jack"          }            //弱引用          let wobj = new WeakRef(obj1)          console.log(wobj);          console.log("======将obj1赋值为空后=======");          obj1=null          console.log(wobj);      </script> |
|  |

### 注意：弱引用的对象需要用deref()来获取

### 我们来改造上面weakmap案例，把like改为弱引用

|  |
| --- |
|  |
|  |

# 6.FinalizationRegistry，建议少用

|  |
| --- |
|  |
|  |

## 小案例，结合上面的案例

|  |
| --- |
|  |
|  |

### 注意：需要在按钮消失之前点击，完整案例

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>es12新特性5-FinalizationRegistry</title>  </head>  <body>      <button id="like">like</button>      <script>          //1.创建一个registry对象         let  registry = new FinalizationRegistry(data=>{              console.log(`按钮不在了，被点击了${data}下`);         })         let like = new WeakRef(document.getElementById("like"))         let wmap = new WeakMap()         wmap.set(like.deref(),{click:0})         like.deref().onclick = function () {            wmap.get(like.deref()).click++            console.log(wmap.get(like.deref()).click);         }         setTimeout(() => {          //注册监视对象          registry.register(like.deref(),wmap.get(like.deref()).click)          document.body.removeChild(like.deref())         }, 3000);      </script>  </body>  </html> |
|  |