es13新特性

# 1.class新增特性

## 1）私有属性和方法

|  |
| --- |
|  |
|  |
|  |

### 可以这么写

|  |
| --- |
| class Person{          #privacy ={} //私有属性             get(key){              return this.#privacy[key]             }              set(key,value){                  this.#privacy[key] = value              }          }  let p = new Person()          p.like = "fish"          console.log(p.like);          p.like = "watermelon"          console.log(p.like); |
|  |

### 也可以这么写

|  |
| --- |
| <script>          class Person{             #privacy = ""  //私有属性             get like(){              return this.#privacy             }              set like(value){                  this.#privacy = value              }          }          let p = new Person()          // p.set("food","fish")          // console.log(p);          // console.log(p.get());          p.like = "fish"          console.log(p.like);          p.like = "watermelon"          console.log(p.like); |
|  |

### 有多个私有属性的类需要使用下面的写法

|  |
| --- |
| class Student{              #sex = ""              #stuNum = ""              get gender(){                  return this.#sex              }              set gender(sex){                  this.#sex = sex              }              get id(){                  return this.#stuNum              }              set id(num){                  this.#stuNum = num              }          }          let s = new Student()          s.gender="female"          s.id="#506-ik301-66xxx"          console.log(s.gender,s.id); |
|  |

### 私有方法也是一样的，在前面加#

|  |
| --- |
|  |

### 静态属性也能私有

|  |
| --- |
|  |

## 2）构造函数外面定义属性

|  |
| --- |
|  |
|  |

## 3）静态代码块

|  |
| --- |
|  |
|  |

### 可以有多个静态代码块

|  |
| --- |
|  |

### 静态代码块主要用来初始化类的静态属性的

## 4）用in操作符来判断一个属性是否是一个类的私有属性

|  |
| --- |
|  |

# 2.最外层的await

|  |
| --- |
|  |

### 注意：需要在es6模块中在nodejs后置commonJs中是不能这样子写的

### 如果你有一个异步获取数据的模块

|  |
| --- |
|  |

### 你需要搭配动态导入来使用

|  |
| --- |
|  |

### 注意：必须是es模块化开发才能这么写

# 3.at函数

|  |
| --- |
|  |

### 小案例

|  |
| --- |
| <body>      <script>          //at          let arr =["Java","python",'golang',"kotlin"]          console.log(arr.at(1));          console.log(arr.at(-1));          console.log(arr.at(-2));      </script>  </body> |
|  |

### 注意：String也有这个实例方法，如str.at(3)

# 4.正则匹配的开始和结束索引

|  |
| --- |
|  |

# 5.findlast和findLastIndex函数

|  |
| --- |
|  |

### 小案例

|  |
| --- |
| <script>          let arr = [100,200,300,400,500,600]          let res1=arr.find(item=>item > 300)          console.log(res1);//400          let res2=arr.findIndex(item=>item > 300)          console.log(res2);//3          //从后往前找，索引的计数还是从左往右          let res3=arr.findLast(item=>item > 300)          console.log(res3); //600          let res4=arr.findLastIndex(item=>item > 300)          console.log(res4);//5      </script> |
|  |

# 6. Error 实例增加 cause 属性，可携带更多错误信息。

|  |
| --- |
|  |

### 以前的处理

|  |
| --- |
|  |

### 现在的处理

|  |
| --- |
|  |
|  |

### cause的值可以是一个对象

|  |
| --- |
|  |

### 小案例：

|  |
| --- |
| <script>          function getData() {              throw new Error("get data error",{cause:"invalid url..."})          }          function test() {              try{                 getData()              }catch(e){                  console.log("错误原因：",e.cause);              }          }          test()      </script> |
|  |

#### 这个特性对应后端返回数据给前端的时候很有用

# 7.Object.hasOwn 代替 Object.prototype.hasOwnProperty，判断对象是否含有属性

|  |
| --- |
| <body>      <script>          let obj = {              name:"kenny",              age:100          }          //以前的写法        console.log(Object.prototype.hasOwnProperty.call(obj, 'name')); //true        //现在的写法        console.log(Object.hasOwn(obj,"age"));//true      </script>  </body> |
|  |