什么是ES6

|  |
| --- |
| ES6， 全称 ECMAScript 6.0 ，是 JavaScript 的下一个版本标准，2015.06 发版 |
| github连接 <https://github.com/tc39/ecma262> |
| ES6 主要是为了解决 ES5 的先天不足，比如 JavaScript 里并没有类的概念，但是目前浏览器的 JavaScript 是 ES5 版本，大多数高版本的浏览器也支持 ES6，不过只实现了 ES6 的部分特性和功能。 |

IE7,IE8,IE9

|  |  |
| --- | --- |
| 在线转换 |  |
| 提前编译 |  |

变量

|  |  |  |
| --- | --- | --- |
| const(常量声明) | 不能重复声明，不可以修改，块级作用域 | const PI=3.1415926 |
| let(变量声明) | 不能重复声明，可以修改，块级作用域 |  |
| var | 可重复，可修改，函数级 |  |
|  |  |  |

函数

|  |  |
| --- | --- |
| 箭头函数 | window.onload=function(){}=window.onload=(a,b)=>{console.log(a+b}  arr.sort((n1,n2)=>{return n1-n2;})  只有一个参数可以省略括号，只有一个return，可以省略大括号 |
| 解构赋值 | es6按照一定的模式，从数组中和对象中取值，对变量进行赋值，这种按照一定的模式为解构 |

剩余参数

|  |  |  |
| --- | --- | --- |
| ...args | 得放在形参的最后，三个点,收集剩余的参数，或展开数组 | function show=(a,b,...args)=>{console.log(...args)}  let arr2=[1,2,2]; let=[4.5.6];  let arr=[...arr2,...arry3] |
| arguments | 所有形参的集合 |  |

数组

|  |  |  |
| --- | --- | --- |
| map() | 数组，一个对一个 | let arr=[1,2,80,4,5]  let result=arr.map((item)=>item>=60?”及格”:”不及格”);  不及格，不及格，及格，不及格，不及格 |
| reduce() | 汇总，一堆出来一个 | let arr[=[11,39,78,812]  let reuslt=arr.reduce((one,two,other)=>) |
| filter() | 过滤，过滤一堆 |  |
| forEach() | 循环 |  |

字符串

|  |  |  |
| --- | --- | --- |
| startsWith() | 以书面开头 | if(str.startsWith(“http://”)){  console.log(“普通网址”);  }else if(str.startsWith(“https://”)){  console.log(“加密网址”);  } |
| endsWith() | 以书面结尾 | if(str.endsWith(“.jpg”)){  console.log(“图片”);  }else if(str.endsWith(“.txt”)){  console.log(“文本”);  } |
| ${} | 拼接 | str=<h1>${title}</h1> |

面向对象 新引入的类概念

1. class关键字，构造器和类分开了
2. class里面直接加方法

json简写

|  |  |
| --- | --- |
| key与value一样时，可以保留一个同名 | let json={“name”:””,”age”:18}  let a=”小星”; let b=48;  let json={a,b}; |
| 方法省去了:function | let json={show:function(){console.log(“小明”)};  let json={show(){console.log(“小明”)}; |

Promise 对象 状态对象

|  |  |  |
| --- | --- | --- |
| 异步 | 操作之间 | |
| 同步 | 操作之间有关系，同时只能做一件事 | |
| Promise.all()  执行所有 | Promise.all([  createPromise(“t1.json”),  createPromise(“t2.json”)  ]).then(function(result){  let [arr,json]=result;  alert(“全都成功了”);  },function(){  alert(“至少有一个失败”)  }) | Promise.all([  $.ajax({}),  $.ajax({})  ]).then(result=>{  let [arr,json]=result;  alert(“全都成功了”);  },err=>{  alert(“至少有一个失败”)  }) |
| Promise.race()  竞速 | Promise.race([  $.ajax({url:””,dataType:”json”}),  $.ajax({url:””,dataType:”json”}),  $.ajax({url:””,dataType:”json”}),  $.ajax({url:””,dataType:”json”})  ]) |  |

generator 函数（生成器函数） 也是解决异步的一个方式

|  |  |
| --- | --- |
| yield(队列)  能传参 | runner(function \*(){  let date1=yield $.ajax(url:xxx,dataType=”json”);  let date2=yield $.ajax(url:xxx,dataType=”json”);  let date3=yield $.ajax(url:xxx,dataType=”json”);  }); |
| async await | async function show(){  let date1=await $.ajax(url:xxx,dataType=”json”);  let date2=await $.ajax(url:xxx,dataType=”json”);  let date3=await $.ajax(url:xxx,dataType=”json”);  }); |
| async await 不在依赖于外面的runner, 统一性能 | |

异步加载

|  |  |
| --- | --- |
| ajax | 一次读一个比较好用 |
| Promise | 一次读一堆比较好用 |
| generator | 有逻辑判断时，比较好用，node里面使用koa模块方便  let server=new koa;  server.use(function \*(){ let date1=yield mysql.query(sql) } ) |
| async await |  |
|  | 同时请求有结果时，Promise,generator都比较好用 |

es7，es8相对es6改进的

数组

let arr=[12,25,4,8]

arr.includes(8); //包含

3\*\*8=Math.pow(3,8)