正课:

1. ES6

let

参数增强

箭头函数

模板字符串

解构

for of

class

\*\*\*\*\*Promise

封装ajax函数

集合

1. let: 代替var

为什么: js中广受诟病的缺点: 声明提前! 破坏程序的正常执行顺序

特点:

1. 防止声明提前

2. 不允许重复声明同名变量

3. 添加块级作用域:

什么是块级作用域(block): 一个{}程序结构内，也是一个作用域。

比如: for while do...while if...else if...else

问题: js中没有块级作用域，块内的变量会影响外部！

为什么: 将块内的变量限制在内部，不要影响外部

如何: 只要用let声明的变量，仅在当前块内有效

原理: 在块内添加匿名函数自调，来划分临时作用域

4. 和for一起使用，可形成闭包!

原理: (function(i){...})(i)

2. 参数增强:

default: 在定义函数时，为函数参数添加默认值

有默认值的参数，在调用函数时，不必赋值

强调: 有默认值的参数必须在参数列表的结尾！

如何: fun(参数1,......,参数n=默认值)

rest: 代替arguments

为什么: arguments的问题:

1. 不是数组类型，无法使用数组的API!

2. 只能获得全部参数

何时: 只要不确定参数个数时，都用rest代替arguments

如何: 在定义函数时，固定参数之后，的剩余参数位置: ...数组名

spread: 代替apply, 将数组打散为单个元素，分别传入

如何: fn(..., ...数组)

3. 箭头函数: 省略function

何时: 所有匿名函数和回调函数，都可用箭头函数简写

如何: 去function，改=>

其实可以更简化:

如果只有一个参数，可省略()

如果函数体只有一句话，可省略{}

如果仅有的一句话还是return，可继续省略return

特点: 箭头函数内外，共用同一个this!

4. 模板字符串: 简化复杂字符串拼接:

今后，只要字符串拼接，都用模板字符串代替+

如何:

整个字符串用``包裹

``中支持换行，支持"", ''

``中要想执行表达式，只要放在${...}即可

5. 解构: 简化批量赋值:

何时: 只要从一个大的对象中，挑选部分属性值使用时

如何: 3种:

1. 数组解构: 下标对下标

var [变量1,变量2,...]=[值1,值2,...]

同时声明多个变量

从原数组中对应位置取出值，赋值给相同位置的变量

2. 对象解构: 属性对属性

var {属性1:变量1, 属性2:变量2,...}={属性1:值1, 属性2:值2,...}

3. 参数解构:

定义时: function ajax({type,url, data, dataType}){

type, url, data, dataType

}

调用时:

ajax({

dataType:"json"

url:"xxx.php",

})

何时: 今后，只要参数是否有值，或顺序都不确定

6. for of: 简化普通for循环

何时: 只要依次获取数组/类数组对象中每个元素时

如何: for(var 变量 of 数组/类数组对象){

变量会自动获得数组/类数组对象中每个元素

}

局限:

1. 只能获得元素内容，无法获得位置！

2. 只能完整遍历，无法选择

3. 只能遍历数字下标的数组或类数组对象, 无法遍历关联数组和对象（用for in遍历）

7. class:

封装:

class类型名{

constructor(){

this.xx=xx;

this.xx=xx;

}

方法名(){

}

方法名(){

}

}

强调: class中构造函数和各个方法之间不用逗号分隔

继承:

class Father{

constructor(){...}

}

class Child extends Father{

constructor(){

super(属性参数...)

}

}

静态: static 不需要创建子对象，就可用构造函数直接调用的方法。

如何:

class 类型名{

static 方法名(){...}

}

调用:

类型名.方法名()

访问器属性:

class 类型名{

constructor(参数){

this.访问器属性=参数

}

//prototype

get 访问器属性(){ return this.数据属性}

set 访问器属性(val){ if(判断条件) this.数据属性=val}

}

8. Promise: 解决: 回调地狱

什么是callback hell: 由于使用参数传递回调函数，导致步骤多时，参数的嵌套层级很深。

何时: 只要异步调用，可能发生延迟时，都要用Promise代替传统参数callback

如何: 定义时

function 第一件事(){

return new Promise(fn=>{

第一件事的内容

fn()

})

}

function 第二件事(){

return new Promise(fn=>{

第二件事的内容

fn()

})

}

function 第三件事(){

第三件事的内容

}

调用时:

第一件事()//return Promise(fn)

.then(第二件事)//return Promise(fn)

.then(第三件事)