Javascript是由ECMAScript规格的具体实现。每一年都会有新的ECMAScript规范，以及新的特性。

2015年为ECMA2015,也就是ES2015，也可以称为ES6（比年代+1）.

2016年ECMA2016,也就是ES2016,也可以称为ES7.

2017年ECMA2017,也就是ES2017,也可以称为ES8.

2018年ECMA2018,也就是ES2018,也可以称为ES9.

2019年ECMA2019,也就是ES2019,也可以称为ES10.

1.

箭头函数=>的语法，实现在ES6标准中。箭头函数=>的语法相当于创建了一个this不会改变的函数。

箭头函数和一般的函数不同，**箭头函数不会绑定this。 或者说箭头函数不会改变this本来的绑定. this还是指向原来的对象。**

函数需要参数：

(参数1，参数2) => {函数体}

函数不需要参数:

() => {函数体}

另外箭头函数还可以更加的简洁：

函数只需要一个参数， 那么参数地方不需要括号：

参数1=> {函数体}

箭头函数（fat arrow function）, 如果有花括号则**默认没有**return,花括号中只是方法语句。需要return的时候需要写上return（…）。

如果没有花括号则默认return所有语句，没有花括号的时候，也可以不使用()而直接跨行使用多条表达式。

2.

#### Javascript的三元运算符

语法：条件?代码块1：代码块2

如果条件为真则执行代码块1，反之执行代码块2.

例如：let test = 1>3?1:3;

则test为3

3.

**Javascript的foreach方法：**

**语法**

array.forEach(function(currentValue, index, arr), thisValue)

**参数**

|  |  |
| --- | --- |
| **参数** | **描述** |
| *function (currentValue, index, arr)* | 必需。 数组中每个元素需要调用的函数。 函数参数:   |  |  | | --- | --- | | **参数** | **描述** | | *currentValue* | 必需。当前元素 | | *index* | 可选。当前元素的索引值。可以在定义的回调函数中不定义 | | *arr* | 可选。当前元素所属的数组对象。可以在定义的回调函数中不定义 | |
| *thisValue* | 可选。传递给函数的值一般用 "this" 值。 如果这个参数为空， "undefined" 会传递给 "this" 值 |

**Filter函数**

在Javascript中，我们可以用数组的filter函数从数组中取出某些对象，例如在list of dict中取出某个dict元素。

注意的是，filter方法将提供一个回掉函数作为参数，该回调函数包括三个参数，元素自身，元素的index以及原数组。如果在回调函数中为true则将元素将会保留，如果为false该元素将会被去除。结果返回到一个新的数组中，而不影响原数组。

例如：

var array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0];

var filtered = array.filter(function(value, index, arr){

return value > 5;

});

//filtered => [6, 7, 8, 9]

//array => [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]

**Map函数**

在Javascript中，我们可以用数组的map函数对**数组中**的每个元素执行一次函数，**并且将函数的返回值放入新的数组**当中。

例如：

var numbers = [1, 2, 3, 4];  
newArray= numbers.map(x => {

return x\*x

});

// newArray = [1, 4, 6, 9]

**Assign函数**

在Javascript中，我们可以用assign函数将源目标中的属性拷贝到目标对象当中。需要注意的是assign执行的是浅拷贝，也就是如果源目标的属性对应的是引用类型时，那么拷贝到目标中的是同一个对象，而不是一个新的对象。

例如：

const target = {a: 1, b: 2};

const source = {b: 4, c: 5};

const returnedTarget = Object.assign(target, source1， source2….);

// returnedTarget为{a:1, b:4, c:5}

**Find函数**

find() 方法返回通过测试（函数内判断）的数组的第一个元素的值。

find() 方法为数组中的每个元素都调用一次函数执行：

* 当数组中的元素在测试条件时返回 true 时, find() 返回符合条件的元素，之后的值不会再调用执行函数。
* 如果没有符合条件的元素返回 undefined

注意: find() 对于空数组，函数是不会执行的。

注意: find() 并没有改变数组的原始值。

测试函数的语法为：

用法：

array.find(function(currentValue, index, arr),thisValue)  
参数：  
  
*currentValue 必需。当前元素*

*index 可选。当前元素的索引值*  
*arr 可选。当前元素所属的数组对象*  
*thisValue 可选。 传递给函数的值一般用 "this" 值。*  
*如果这个参数为空， "undefined" 会传递给 "this" 值*

例子:

*var ages = [4, 12, 16, 20];  
 function checkAdult(age) {   
　　return age >= document.getElementById("ageToCheck").value;   
}   
function myFunction() {   
　　document.getElementById("demo").innerHTML = ages.find(checkAdult);   
}*

这里checkAdult函数便是测试函数， 他有一个参数也就是必须参数age, 表示每一个ages中的元素， 如果大于id=”ageToCheck”的数字则返回true, **那么find函数返回第一个符合条件的元素的值**。

4.

Javascript向数组头插入元素使用方法unshift(元素)， 向数组尾部插入元素使用方法push(元素)

5.

Javascript的解构引入语法：

import React, {Component} from ‘react’🡪意思是从react中引入React, 以及**React中的**Component.以后便可以直接在使用Component.

6.

Javascript模块化引入工具requireJs

随着网站功能逐渐丰富，网页中的js也变得越来越复杂和臃肿，原有通过script标签来导入一个个的js文件这种方式已经不能满足现在互联网开发模式，我们需要团队协作、模块复用、单元测试等等一系列复杂的需求。

RequireJS是一个非常小巧的JavaScript模块载入框架，是AMD规范最好的实现者之一。最新版本的RequireJS压缩后只有14K，堪称非常轻量。它还同时可以和其他的框架协同工作，使用RequireJS必将使您的前端代码质量得以提升。

在引入requireJs后，我们便可以在使用require关键字引入javascript模块。

8.

ECMAScript 和 JavaScript 的关系是，**前者是后者的规格，后者是前者的一种实现**。

javascript是netscape创造的并交给了国际标准化组织ECMA来制定标准化过程。

ES6是ECMA的为JavaScript制定的第6个版本的标准，标准委员会最终决定，标准在每年的 6 月份正式发布一次，作为当年的正式版本。**ECMAscript 2015 是在2015年6月份发布的ES6的第一个版本**。依次类推ECMAscript 2016 是ES6的第二个版本、 ECMAscript 2017 是ES6的第三个版本。

ES6增加的例子有：

|  |
| --- |
| 新增关键字Let：Let只在代码块内有效，var在全局范围内有效。Let只能声明一次，var可以声明多次。  **let关键字不存在变量提升，var关键字存在变量提升**。  例如：  console.log(a); //ReferenceError: a is not defined  let a = "apple";  console.log(b); //undefined  var b = "banana";  变量 b 用 var 声明存在变量提升，所以当脚本开始运行的时候，b 已经存在了，但是还没有赋值，所以会输出 undefined。  变量 a 用 let 声明不存在变量提升，在声明变量 a 之前，a 不存在，所以会报错。  Undefined和ReferenceError: x is not defined不一样，undefined是申明了但没有赋值具体的数值，值就是undefined,而没有申明过直接调用则是not defined, 是错误。 |
| const 声明一个只读变量，声明之后不允许改变，不允许赋值或者重新声明。意味着，一旦声明必须初始化，否则会报错。 |
| 使用语法糖class来定义类 |
| 提供箭头函数=》 |
| 模板字符串语法，在**反单引号**中，以${varname}的形式插入变量  例如：  let str=” world”;  console.log(`Hellow ${str}`) // 输出Hellow World |
| 扩展语法  扩展操作符 …  使用扩展操作符…, 用于将数组或者字典打开并且并且复制  例子1:  var mid = [3, 4];  var arr = [1, 2, ...mid, 5, 6]; // 通过扩展操作符，mid数组将被打开成为3,4元素插入到新的数组当中  例子2:  var arr = ['a', 'b', 'c'];  var arr2 = [...arr]; // **通过扩展操作符，复制出arr2和arr有同样内容。 这样就算arr2改变，arr也不会改变。如果通过arr2 = arr,那样则会因为是同一个引用地址而同时改变。**  arr2.push('d');  例子3:  var str = "hello";  var chars = [...str]; // 通过扩展操作符， 将字符串hello拆分为数组 |

9.

Javascript定义函数：

Javascript为动态类型语言，申明函数的时候不需要指定函数的类型

function myFunc (name, weather) {

…

}

Javascript不像Java,C#那样支持多态，Javascript不支持多态(polymorphism)。函数名一样， 不同函数参数不会因此产生多态效果，只会让后定义的函数覆盖先定义的函数。

调用Javascript函数的时候，不需要和定义函数时需要的参数个数相同。

如果调用时传入的函数少于定义，则剩下的函数为undefined.

**function** myFunc(name, weather) {  
 console.log("Hello " + name + ".");  
 console.log("It is " + weather + " today.");  
}  
myFunc("Adam"); #执行正常。只是未被传值的变量为undefined.

如果调用时传入的函数多与定义，则多余的函数会被忽略，或者通过定义...extraArgs放入数组当中。

**function** myFunc(name, weather) {  
 console.log("Hello " + name + ".");  
 console.log("It is " + weather + " today.");  
}  
myFunc("Adam", "sunny", "GGG", "AAA"); #执行正常，没有错误

多余的参数可以用...extraArgs数组来存放。

**function** myFunc(name, weather, ...extraArgs) {  
 console.log("Hello " + name + ".");  
 console.log("It is " + weather + " today.");  
 **for** (**let** i=0; i<extraArgs.length; i++){  
 console.log(i);  
 }  
}  
myFunc("Adam", "sunny", "one","two","three"); #执行正常，多余参数在extraArgs中

Javascript支持参数的默认值。

**function** myFunc(name, weather="Sunny") {  
 console.log("Hello " + name + ".");  
 console.log("It is " + weather + " today.");  
}  
myFunc("Adam"); #执行正常。 It is Sunny today.

Javascript将函数看作是对象，所以可以将函数作为参数进行传递，调用。

**function** myFunc(nameFunction){  
 **return** ("Hello " + nameFunction() + ".")  
}  
  
**function** printName(nameFunction, printFunction){  
 printFunction(myFunc(nameFunction))  
}  
  
printName(**function**() {**return** "Adam"}, console.log); #输出为Hello Adam.

Javascript函数中的变量闭合。意思是Javascript在函数中定义的另一个函数，可以获取外层函数的本地变量。需要注意的是，**内层函数如果需要使用counter, index这种计数变量的时候，不要和外层的计数变量混在一起。**

**function** myFunc(name) {  
 **let** myLocalVar = "sunny";  
 **let** innerFunction = **function**() {  
 **return** ("Hello " + name +". Today is " + myLocalVar + ".");  
 };  
 **return** innerFunction();  
}  
console.log(myFunc("Adam")); # 内层函数可以获取外层函数的本地变量, name, myLocalVar

Javascript函数的定义方法有两种, 一种是比较熟悉的function myFunc(parameter){…}形式，另外一种是const myFunc2 = function(parameter){…}的方式。

**function** myFunc(name){  
 console.log("Hello " + name)  
}  
**const** myFunc1 = **function**(name){  
 console.log("Hello " + name)  
};  
  
myFunc("A");  
myFunc1("B");

10.

Javascript的六种基本数据类型：

String: 字符串类型

Number: 数字类型

Boolean: 布尔类型

Undefined: 数据已经定义但是没有赋值

Null: 数据定义但是值为空

Symbol

字符串类型常用自带函数：

|  |  |
| --- | --- |
| .length | 返回字符串的长度 |
| .charAt(index) | 返回在index位置的字母 |
| .concat(string) | 将字符串与传入的字符串拼接 |
| .indexOf(term, start) | 返回查找字符串的首先出现的位置，如果没有则为-1 |
| ,replace(term, newTerm) | 将字符串中的term换为newTerm |
| .slice(start, end) | 将字符串从指定开始位置截取到结束位置 |
| .split(term) | 将字符串根据term字符进行分割 |
| toUpperCase() | 将字符串全部变为大写 |
| toLowerCase() | 将字符串全部变为小写 |
| trim() | 将字符串中的开头前，结尾后的空白字符移除 |
| `${变量名` | ES6的特性，字符串模板，将变量插入字符串的内容中 |

11.

== 与 === 的区别。当javascript进行==比较的时候，Javascript会将变量的类型变为一致，然后比较他们的值是否相同。而如果使用===进行比较，Javascript不会进行任何转换，而是直接比较。如果相同，则需要类型，和值都相同。

12.

Javascript在运算过程中会自动转换数据类型。有时候并不是我们需要的。

例如, Javascript中 “+” 的字符串拼接比运算加号的优先级高所以

let myData2 = 5 + “5”#将会变为55， 而不是数字类型10.这是Javascript自动转换的结果。如果我们需要变为数字10，则需要手动转换为let myData2 = 5 + parseInt(“5”)

数字转为字符串(Number to String)

|  |  |
| --- | --- |
| toString() | 返回数字在十进制下的字符串 |
| toString(2) | 返回数字在二进制下的字符串 |
| toString(8) | 返回数字在八进制下的字符串 |
| toString(16) | 返回数字在十六进制下的字符串 |
| toFixed(n) | 返回数字字符串表示，带有n位小数点 |
| toExponential(n) | 返回数字的指数型表示，带有n位小数点 |

字符串转为数字(String to Number)

|  |  |
| --- | --- |
| Number(str) | 将字符串变为整形数字或者浮点数 |
| parseInt(str) | 将字符串变为整形数字 |
| parseFloat(str) | 将字符串变为整数变为浮点数 |

13.

Javascript创建数组的两种方式：

let myArray = new Array(); #以构建函数的方式创建数组

let myArray = [100, “Adam”, true] # 以普通变量的方式构建数组

Javascript的数组可以自动扩展数组长度，并且可以存放任何数据类型到数组当中。

Javascript数组Built-in方法：

|  |  |
| --- | --- |
| .concat(otherArray) | 将数组与参数提供的数组相拼接，然后返回。 |
| .join(separator) | 将数组中的元素使用separator连接起来成为字符串并且返回 |
| .pop() | 移除数组中的最后一个元素，并且返回 |
| .shift() | 移除数组中的第一个元素，并且返回 |
| .push(item) | 将元素放入数组的结尾 |
| .unshift(item) | 将元素放入数组的开头 |
| .reverse() | 将数组倒序形成新的数组并返回 |
| .slice(start, end) | 将数组进行切割，从start位置到end位置 |
| .sort() | 将数组进行排序，可以传入自定义的比较方法 |
| .splice(index, count) | 将数组从index位置开始，移除count个元素，并且将移除的元素返回 |
| .every(test) | 对数组中每个元素运行test方法，如果数组中所有元素能让test函数都为true, 则返回true,否则为false |
| .some(test) | 对数组中每个元素运行test方法，如果数组中任一元素能让test函数都为true, 则返回true,否则为false |
| .filter(test) | 对数组中每个元素运行test方法，保留能让test函数为true的元素并且以新数组的形式返回。 |
| .find(test) | 返回第一个让test函数为true的元素 |
| .findIndex(test) | 返回第一个让test函数为true的元素下标 |
| .foreach(callback) | 对数组中的每个元素调用callback函数 |
| .map(callback) | 对数组中的每个元素调用callback函数，并且将结果放入新的数组中返回 |
| .reduce(callback) | 对数组中的每个元素调用callback函数，获取累计的总计数值.  Callback要求两个参数，第一个参数是在轮训中保持的数值，例如总数。第二个参数是每一个数组中的元素。 |

14.

创建Object的两种方式：

let myObj = new Object() #以构建函数的方式创建对象

let myObj = { #以普通变量的方式构建对象

“xx”: xxx,

xx: function () {

}

}

15.Javascript的Class中，constructor方法用于构建类的实例，它将会在每个类的实例化时自动调用。