1. **var, let, const声明变量**

use const by default; only use let if rebinding is needed; var shouldn’t be used in ES6

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | scope | defined | rebinding | property |
| var | function | Y | Y | Y |
| let | block | N | Y | Y |
| const | block | N | N | Y |

***Ex1:***

Immediately-Invoked Function Expression(IIFE)

(function () {

     const name = 'cc';

})();

***Ex2:***

for (var i=0; i<10; i++)

{

     console.log(i);

     setTimeout (function(){ console.log(`i:${i}`) }, 1000)

}

***Ex3:*** 暂时性死区

console.log(color); console.log(color); console.log(color);

color = ‘yellow’; let color = ‘yellow’; const color = ‘yellow’;

undefined (变量提升) reference error reference error

1. **Arrow function**

简明的语法; 隐式返回; 不绑定this(父级的作用值)

An arrow function does not have its own this; the this value of the enclosing execution context is used.

Arrow functions do not have a prototype property.

(1) Arrow functions cannot be used as constructors and will throw an error when used with new.

(2) Since arrow functions do not have their own this, the methods call() or apply() can only pass in parameters. This Arg is ignored.

(3) Arrow functions do not have their own arguments object. Thus, arguments is simply a reference to the arguments of the enclosing scope.

(4) arrow functions cannot be used as generators.

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions>

***Ex1:***

const double = numbers.map(number => number \* 2);

***Ex2:***

const jelly ={

name: 'Jelly',

hobbies: {['Coding', 'Sleeping', 'Reading']},

printHobbies: function(){

this.hobbies.map(hobby => { console.log(`${this.name} loves ${hobby}`);})

}

}

1. **ES6参数默认值**

function (a = 5, b = 3)

{

return a \* b;

}

multiply (undefined,5)

1. **Template Strings/literals**

const person ='cc'

const age =5

console.log(`${person} is ${age \* 5} years old. `)

1. **Tagged Template Strings**

highlight`${user} has command on your topic ${topic}.`

function highlight(strings, ...values){}

1. **Tagged Template Sanitize**

function sanitize(strings,...values)

{

const dirty = strings.reduce((prev,curr,i)=>`${prev}${curr}${values[i]||''}`);

return DOMPurify.sanitize(dirty);

}

sanitize`<div class="comment-header">${user}</div><div class="comment-body">

${textarea.value}</div>`.trim();

1. **ES6中新增的字符串函数**

startsWith(), endsWith(), includes(), repeat()

1. **ES6 对象解构**

let name ='';

({ name, age:a , sister='no sister'} = Tom);

let {mother, father, brother} = Tom.family;

1. **数组解构**

const numbers = ['one', 'two', 'three', 'four', 'five', 'six'];

const [one,,three,...others] = numbers;

let number1 = 10, number2= 20;

[number1,number2] = [number2,number1];

1. **ES6 for of**

For of循环可以用于数组、字符串、map、NodeList等，目前还不支持对象。

.entries() 取到索引和值

console.log(fruit[1]); 当前元素的值

const fruits = ['Apple', 'Banana', 'Orange', 'Mango']

fruits.describe ='My favorite fruits';

for(let i=0;i<fruits.length;i++){

console.log(fruits[i]);

}

fruits.forEach(element => {

//if (element == 'Orange') {

//continue; //error

//}

});

for(let index in fruits){

console.log(fruits[index]); // all property

}

用结构的语法[index,fruit]

const fruits = ['Apple','Banana','Orange','Mango'];

for(let fruit of fruits.entries()){

console.log(fruit);

}

用于数组，求和

function sum(){

let total = 0;

for(let num of arguments){

total +=num;

}

return total;

}

sum(10,12,78,789,52,63);

用于字符串循环

let name ='Veg.kim';

for(let char of name){

console.log(char);

}

用于Node List循环

const lis = document.querySelectorAll('li');

for(let li of lis)

{

li.addEventListener('click', function(){

this.classList.toggle('completed');

})

}

1. **Array for & of**

//Node List

const todos = document.querySelectorAll('li');

const names = Array.from(todos,todo => todo.textContent);

//Array.from(todos).map(todo => todo.textContent);

//Arguments Object

function sum(){

return Array.from(arguments).reduce((prev,curr) => prev +curr,0);

}

//Strings

const website = 'MTR';

const sss = Array.from(website);

// Array.of()弥补Array()不足

Array.of(7)

Array(7)

1. **Array方法使用**

.find() .findIndex() .some() .every()

let res1 = inventory.find(e=>e.name=="bananas"); //true

let res2 = inventory.findIndex(e=>e.name=="bananas"); //1

let res3 = inventory.some(e=>e.quantity>2); //true

let res4 = inventory.every(e=>e.quantity>2); //false

1. **剩余参数**

//对函数参数处理

function sum(...numbers)

{

return numbers.reduce((prev,curr) => prev+curr,0);

}

console.log(sum(1,2,3,4));

function convertCurrency(rate, ...amounts)

{

return amounts.map(amount=>amount\*rate);

}

console.log(convertCurrency(0.89,12,23,655,23));

//数组解构

const players = ['Jelly', 123,5.4,6.7,3.4,8.3,7.9];

const [name,id,...scores] = players;

1. **ES6扩展运算符**

const younger = [' George', 'John', 'Thomas'];

const olders = [' James', 'Adrew', 'Martin'];

const members = [...younger, 'Mary', ...olders];

const currentMembers = [...members]; //[].concat(members);

younger.push(...olders);

**Ex1**

const heading = document.querySelector(".heading");

heading.innerHTML = wrapWithSpan(heading.textContent);

function wrapWithSpan(word)

{

return [...word].map(letter=>`<span>${letter}</span>`).join('');

}

**Ex2**

//NodeList

const lis = document.querySelectorAll('li');

for(let li of [...lis]){

console.log(li.textContent);

}

//数组

const favorites ={

color: ['yellow', 'blue'],

fruits: ['banana', 'mongo']

};

const shoppingList = ['milk', 'sweet', ...favorites.fruits]

**Ex3**

//函数

let fruit = ['apple','banana','pear'];

const newFruit = ['orange', 'mongo'];

fruit.push(...newFruit);

const dateFields = [2018,9,28];

const date = new Date(...dateFields);

1. **对象字面量的扩展**

//set value for object

const ll = {

name,

age,

birthday,

greet(){ alert(`Hello ${name}!`); }

}

//identity

let id=0;

const userIds = {

[`user-${++id}`]: id,

[`user-${++id}`]: id,

[`user-${++id}`]: id,

[`user-${++id}`]: id,

[`user-${++id}`]: id

}

// key & value

const keys = ['name', 'age', 'birthday'];

const values = ['Laravist', 2, '2018-2'];

const Lara = {

[keys.shift()]: values.shift(),

[keys.shift()]: values.shift(),

[keys.shift()]: values.shift(),

}

1. **ES6 Promise**

**Ex1:** axios

let user;

const usersPromise = axios.get("https://api.github.com/users");

usersPromise.then(response=>{

user = response.data[0].login;

console.log(user);

return axios.get(`https://api.github.com/users/${user}/repos`);

})

.then(res=>{

console.log(res.data);

})

.catch(err=>{

console.log(err);

})

**Ex2:**

const p = new Promise((resolve,reject)=>{

setTimeout(()=>{

resolve('Laravist is awesome');

},1000)

});

p.then(data=>console.log(data))

.catch(err=>console.log(err));

Ex3:

function getRepoById(id)

{

return new Promise((resolve,reject)=>{

let repo = repos.find(p=>p.id==id);

if(repo){

resolve(repo);

}

else{

reject(Error('Repo is not found.'));

}

})

}

function getOwnerByRepo(repo)

{

return new Promise((resolve,reject)=>{

let owner = owners.find(p=>p.name==repo.owner)

if(owner){

resolve(owner);

}

else{

reject(Error('Owner is not found.'))

}

})

}

getRepoById(1)

.then(repo=>{ return getOwnerByRepo(repo) })

.then(owner=>console.log(owner))

.catch(err=>console.log(err));

Ex3: 处理多个 Promise

Promise.all([userPromise,moviePromise]) //.race

.then(response=>{

const [user,movie]=response;

console.log(user);

console.log(movie);

}).catch(err=>{console.log(err)});

1. **Symbol理解**

const perter = Symbol('peter'); //生成唯一标识符，解决对象命名冲突问题。

const classRoom ={

[Symbol('lily')]:{grade:60,gender:'female'},

[Symbol('nina')]:{grade:90,gender:'female'},

[Symbol('nina')]:{grade:90,gender:'female'},

}

const syms = Object.getOwnPropertySymbols(classRoom).map(sym=>classRoom[sym]);

console.log(syms);

1. **ESLint**

npm install -g install eslint

eslint –init

eslint bad-code.js

.eslintrc.json

{

"env": {

"browser": true,

"commonjs": true,

"es6": true

},

"extends": "airbnb",

"plugins": ["html","markdown"],

"rules": {

    "no-console":0

}

}

/\* globals Vue\*/

/\* eslint-disable no-new\*/

/\* eslint-enable no-new\*/

1. **123**