ES6语法简单介绍

ES6语法

ES 全称 ECMAScript，是由欧洲计算机协会(ECMA)制定的一种脚本语言的标准化规范。

实际上ES6就是给 JavaScript 制定的一种语法规范，写 js 的时候如果按照 ES6 中的规范去写，写的代码不仅简洁而且效率很高。

ES6 发行于 2015 年 6 月，这个版本的语法规范极大地提高了前端开发人员的效率。目标是使得 JavaScript 语言可以用来编写复杂的大型应用程序，成为企业级开发语言。

es6介绍：https://es6.ruanyifeng.com/#docs/intro

ECMAScript历史

- 1997 年 ECMAScript 1.0 诞生。
- 1998 年 6 月 ECMAScript 2.0 诞生，包含一些小的更改，用于同步独立的 ISO 国际标准。
- 1999 年 12 月 ECMAScript 3.0诞生，它是一个巨大的成功，在业界得到了广泛的支持，它奠定了 JS 的基本语法，被其后版本完全继承。直到今天，我们一开始学习 JS ，其实就是在学 3.0 版的语法。
- 2000 年的 ECMAScript 4.0 是当下 ES6 的前身，但由于这个版本太过激烈，对 ES3 做了彻底升级，导致标准委员会的一些成员不愿意接受。这个版本最后没有通过，但是它的大部分内容被 ES6 继承了。因此，ES6 制定的起点其实算是 2000 年。
- 2009 年 12 月，ECMAScript 5.0 版正式发布。ECMA 专家组预计 ECMAScript 的第五个版本会在 2013 年中期到 2018 年作为主流的开发标准。2011年6月，ES 5.1 版发布，并且成为 ISO 国际标准。
- 2013 年，ES6 草案冻结，不再添加新的功能，新的功能将被放到 ES7 中；2015年6月， ES6 正式通过，成为国际标准。

01 let 和 const

let 是 ES6 中新增加的用于声明变量的关键字，具有如下特点：

1、不存在变量提升（不能先使用再声明）

```
console.log(x)
var x = 3
console.log(y)
let y = 7
```

```
PS C:\web-projects\www-site\nodejs-app\ES6> node .\index01.js
undefined
C:\web-projects\www-site\nodejs-app\ES6\index01.js:3
console.log(y)
            ^

ReferenceError: Cannot access 'y' before initialization
```

2、只在块级作用域有效

```
> {let num=20;}
< undefined
> console.log(num);
✖ ▶Uncaught ReferenceError: num is not defined
      at <anonymous>:1:13
```

01 let 和 const

let 定义的变量只能在所在的代码块 { } 中使用。

```
var num = 20
{
  let num = 30
}
console.log(num) // 输出20
```

01 let 和 const

const 也是 ES6 中新增加的用于声明变量的关键字，主要用来声明常量。

1、声明常量时必须赋值。

```
const name;
console.log(name); //SyntaxError: Missing initializer in const declaration
```

2、只在块级作用域有效

```
var message = 'Hello';
{
    const  messge='World';
}
console.log(message);   // 输出Hello
```

01 let 和 const

3、赋值后，值不能修改。

```
const message = 'East';
message='North';
console.log(message);   //TypeError: Assignment to constant variable.


const user = { id: 123, name: '张三' }
user = { id: 234, name: '李四' }
console.log(user) //TypeError: Assignment to constant variable.


const user = { id: 123, name: '张三' }
user.name = '李四'
console.log(user) //{ id: 123, name: '李四' }
```

由此可见const 赋值的常量如果是基本数据类型，不能重新赋值；如果是对象等复杂数据类型，不能更改地址，但是可以更改对象中属性的值。

01 let 和 const

var、let、const 的区别

1. var 声明的变量作用域在所处的函数内，let 和 const 声明的变量作用域在所处的大括号内。
2. var 声明的变量存在变量提升现象，let 和 const 声明的变量不存在变量提升现象。
3. const 声明变量时必须要赋值，赋值之后不能再重新赋值。

02 箭头函数

1. ( ) 代表函数
2. { } 代表函数体
3. const ft = ()=>{} 代表把一个函数赋值给 ft
4. ft() 调用该函数

02 箭头函数

1、无参数，函数体只有一行代码

```
// 常规写法
function print() {
  console.log("Hello");
}

// 箭头函数
const ft = ()=> console.log("Hello");
// 调用函数
ft();
```

02 箭头函数

2、有参数，函数体只有一行代码

```
// 常规写法
function print(name, content) {
    return name + content
}

// 箭头函数
const ft = (name, content) => name + content
// 调用函数
console.log(ft('梧桐树', '阳光穿过梧桐树的叶子，在树上留下了斑光'))
```

02 箭头函数

3、只有一个参数，可以去掉大括号

```
// 常规写法
function print(name) {
  return name + '真好看'
}

// 箭头函数
const ft = name => name + '真好看'
// 调用函数
console.log(ft('你'))
```

02 箭头函数

4、多个参数，函数体有多行

```
// 箭头函数：获取年龄最大值
const ft = (userArray, sex) => {
  let ageArray = userArray.filter(user => user.sex == sex).map(item => item.age);
  return Math.max(...ageArray);
}

let userArray = [{ name: '张三', sex: '男', age: 18 },
{ name: '李四', sex: '女', age: 19 },
{ name: '王五', sex: '男', age: 21 }]
// 调用函数
console.log(ft(userArray, '男')); // 21
```

03 解构

解构就是把数据结构进行分解，然后为定义的变量赋值。

1、数组解构

传统方式

```
const num =[0,1,2,3];
const a= num[0];
const b= num[1]
console.log(a+b);
```

解构

```
let [a, b] = [0, 1, 2, 3]
console.log(a + b)
```

从数组中提取值，按照对应位置，对变量赋值

03 解构

2、对象解构

传统方式

```
let user = { name: '张三', age: 19 }
let name = user.name
let age = user.age
console.log('姓名:' + name + ',年龄:' + age)
```

解构

```
let { name, age } = { name: '张三', age: 19 }
console.log('姓名:' + name + ',年龄:' + age)
```

对象的解构与数组有一个重要的不同。数组的元素是按次序排列的，变量的取值由它的位置决定；而对象的属性没有次序，变量必须与属性同名，才能取到正确的值。

04 剩余参数

剩余参数允许将一个未知数量的参数表示为一个数组。

1、语法表示为：...参数名

```
> const print = (num, ...args) => {
      console.log(num);
      console.log(args);
  }
  print(0, 1, 2)
  0
  ▶ (2) [1, 2]
```

参数 args 是一个数组

04 剩余参数

2、和解构连用

```
let users = ['张三', '李四', '王五'];
let [u1, ...u2] = users;
console.log(u1);
console.log(u2);
```

```
张三
▶ (2) ['李四', '王五']
```

04 剩余参数

3、合并数组

```
let u1 = ['张三', '李四', '王五'];
let u2 = ['张3', '李4', '王5'];
let u3 = [...u1,...u2];
console.log(u3);
```

```
▶ (6) ['张三', '李四', '王五', '张3', '李4', '王5']
```

05 可选链

可选链 ?. 是一种 访问嵌套对象属性的防错误方法。

即使中间的属性不存在，也不会出现错误。

如果可选链 ?. 前面部分是 undefined 或者 null，它会停止运算并返回 undefined。

05 可选链

可选链的三种形式：

1. obj ?.prop. --------如果obj 存在则返回 obj.prop ,否则返回undefined。
2. obj ?.[prop] -------如果存在则返回obj[prop]， 否则返回undefined。
3. obj.method?.()------如果obj.method 存在则调用obj.method(),否则返回undefined。

05 可选链

如果想获取一个嵌套对象的属性，一般写法：

```
let res = {
  data: {
    data: {
      success: true,
      id: '20220425'
    }
  }
}
if (res && res.data && res.data.data.success) {
  let id = res.data.data.id
  console.log(id) //20220425
}
```

05 可选链

使用可选链：

```
let res = {
  data: {
    data: {
      success: true,
      id: '20220425'
    }
  }
}
if (res?.data?.data?.success) {
  let id = res?.data?.data?.id
  console.log(id) //20220425
}
```

可选链的使用前提是 ?. 前的变量必须已声明，如果没有就会发生错误

06 Set

Set 是 ES6 提供的一种数据结构，和数组很像，但是它里面的数据不可重复。

```
> const set1 = new Set([2, 2, 3, 4, 5, 5]);
  const set2 = new Set(['Apple','Orange','Apple']);
  console.log(set1);
  console.log(set2);
```

```
▶ Set(4) {2, 3, 4, 5}
▶ Set(2) {'Apple', 'Orange'}
```

06 Set

1、添加数据

```
> const set1 = new Set([1, 2, 3, 4, 5, 5]);
  set1.add(6);
< ▼Set(6) {1, 2, 3, 4, 5, …} i
    ▼[[Entries]]
      ▶0: 1
      ▶1: 2
      ▶2: 3
      ▶3: 4
      ▶4: 5
      ▶5: 6
       size: 6
    ▶[[Prototype]]: Set
```

06 Set

2、删除数据

```
> const set2 = new Set([1, 2, 3, 4, 5, 5]);
  set2.delete(2);
< true
```

3、包含数据

```
> const set3 = new Set([1, 2, 3, 4, 5, 5]);
  const rs = set3.has(1);
  console.log(rs);
  true
```

06 Set

4、清除数据

```
> const set4 = new Set([1, 2, 3, 4, 5, 5]);
  set4.clear();
< undefined

> set4
< ▼Set(0) {size: 0}
    ▼[[Entries]]
        No properties
      size: 0
    ▶[[Prototype]]: Set
```

07 数组操作

1、合并数组

```
let u1 = ['张三', '李四', '王五'];
let u2 = ['张3', '李4', '王5'];
let u3 = [...u1,...u2];
console.log(u3);
```

```
▶ (6) ['张三', '李四', '王五', '张3', '李4', '王5']
```

2、includes()用来判断该数组是否包含某个值，返回值是布尔值。

07 数组操作

3、find()找第一个符合条件的成员，没有找到返回 undefined

```
> let users = [{ name: '张三', age: 18 }, { name: '李四', age: 20 }];
  let user = users.find((item, index) =>
      item.age > 18
  )
  console.log(user);
```

```
▶ {name: '李四', age: 20}                                    VM2448:5
```

4、findIndex()找第一个符合条件的成员的索引，没有的话返回 -1

```
> let users = [{ name: '张三', age: 18 }, { name: '李四', age: 20 }];
  let index = users.findIndex((item, index) =>
      item.age > 18
  )
  console.log(index)
```

```
1                                                            VM2558:5
```

07 数组操作

5、filter()用来返回一个满足条件的新数组，不满足条件返回空数组

```
let users = [{ name: '张三', age: 18 }, { name: '李四', age: 20 }];
let array = users.filter((item, index) =>
    item.age > 21
)
console.log(array);
```

```
▼[] i                                                    VM2509:5
    length: 0
  ▶[[Prototype]]: Array(0)
```

07 数组操作

6、map()用来返回一个对成员进行加工之后的新数组

```
> let users = [{ name: '张三', age: 18 }, { name: '李四', age: 20 }];
  let array = users.map((item, index) => {
      item.name += "123";
      item.age += 12;
      return item;
  })
  console.log(array);
```

```
▼(2) [{…}, {…}]                                          VM2724:7
  ▶0: {name: '张三123', age: 30}
  ▶1: {name: '李四123', age: 32}
    length: 2
  ▶[[Prototype]]: Array(0)
```

08 字符串扩展方法

1、startsWith() 和 endsWith()

分别表示该字符串参数是否在某个字符串头部和尾部。

2、模板字符串

模板字符串是 ES6 新增加的创建字符串的方式。定义方式：反引号

```
> let c = `China`;
  let msg = `I Love you, ${c}`;
  console.log(msg);
  I Love you, China
```

08 字符串扩展方法

3、调用函数

```
> const print = message=>message+", World";
  let message = `${print('Hello')}`;
  console.log(message);
```

```
Hello, World
```