```
ES6:
  主要把常用的记住，不常用的（鸡肋）了解下，通过查文档来加深记忆。

  js:
     ECMAScript
       字符串数组的方法、数据类型、if for(;;)
     DOM
     BOM

  ES6:ES2015,JavaScript 语言的下一代标准

  现在的一些浏览还不兼容，需要使用babel去编译，使所有浏览器都兼容
    */
/*
        解构赋值:
            有对象的和数组的解构赋值
    */
//  var a = 10;
//  var b = 5;
//
//  var [b,a] = [a,b];
//  console.log(a,b);

//  let c = a;
//
//  a = b;
//  b = c;


//  var arr = [1, [[2], 3]];
//
//  let [a,[[b],c]]  = arr;

//  var arr = [12,[6,[5,[4,[3]]]],7]
//
//  let [f,[d,[c,[b,[a]]]],e] = arr;
//
//  console.log(a,b,c,d,e,f);


//  let [x,, y] = [1, 2, 3];
//
//  console.log(x,y)

    //报错
//  let [foo] = 1; //数字不能解构
//  let [foo] = false;  //布尔值不能解构
//  let [foo] = NaN; //NaN也不能解构
```

```
//  let [foo] = undefined;//
//  let [foo] = null;
//  let [foo] = {};
//  let [fn] = function fn2(){}

    //小总结，如果使用数组的方式去解构，那么除了数组别的数据解构都报错

    //console.log(fn)



//  let [x, y='b'] = ['a'];

    //let [x, y=x] = ['a']; //a,a

    //let [x, y=x] = [,'a']; //有配置走配置，没配置走默认

    //console.log(x,y); //undefined 'a'

    //let [x, y=x] = []; //如果没有数组中没有传，那么默认就是undefined


    /*

        如果一个数组中有个变量且这个变量没有赋值，那么会报错

        当使用解构赋值之后（前提条件用var来声明解构），数组不会因为a而报错

        这个时候发生什么事呢？

            解构之后，如果使用了var，那么会先给左边的变量预解析，
            然后右边赋值给左边，左边再去读取刚才右边的每个值。


        ***使用let的时候一定要注意没有预解析的问题。
    */

//  var [a,b] = [1,a];

    //var [a,b] = [1,a];

    //var [a,b] = [,a]; // a = 10;

    //console.log(c,b);



//  function f() {
//    console.log('aaa');
```

```
//  }
//
//  let [x = f()] = [1];
//
//  console.log(x); //1



//  let x;
//  if ([1][0] === 1) {
//    x = f();
//  } else {
//    x = [1][0];
//  }
//  console.log(x);

//  let [x = 1, y = x] = [];//1,1

//  let [x = 1, y = x] = [2]

//  let [x = 5, y = x] = [1, 2];

//  var [x = y, y = 1] = [];

    console.log(x,y);
```

```
    //  var [a,b] = [];

//  var [a=5,b=a] = [1];

//  var [a=b,b] = [1,3];

//  let [a=b,b] = [,3];

//  let [a] = 12;

//  let [a,[b]] = [12,[3]]

//  console.log(a,b);


//  let { foo, bar } = { foo: "aaa", bar: "bbb" };

    //当左边解构赋值之后还取了别名，key值就失效，正确找法，找冒号右边的变量
    //let { foo:a, bar:b } = { foo: "aaa", bar: "bbb" };


//  let obj = { first: 'hello', last: 'world' };
//  let { first: f, last: l } = obj;
//  console.log(f,l);


    //不管数组还是对象的解构，最左边都是以逗号分隔
```

```
    var div1 = document.getElementById('div1');
    //let {width:w,height:h,fontSize:f} = getComputedStyle(div1);
    //console.log(w,h,f);
    //console.log(width,height,fontSize);


    //解构的值并不是克隆，只是读取的一种方式,如果操作，其实还是操作对象本身的属性。
//  let obj = {
//      arr:[1,2,3,4],
//      age:20,
//      name:'周庆林'
//  }
//
//  let {arr:a,age:b} = obj;
//
//  //a.push(5);
//
////    var arr = obj.arr;
////    var age = obj.age;
//  console.log(a,b);


//  let foo;
//  let {foo} = {foo: 1};  //报错


//  let obj = {
//    p: [
//      'Hello',
//      { y: 'World' }
//    ]
//  };
//
//  let {p} = obj;
//
//  let [a,{y}] = p;
//
//  console.log(a,y)


//  var node = {
//    loc: {
//      start: {
//        line: 1,
//        column: 5
//      }
//    }
//  };
//
//  let {loc:{start:{line:abc,column}}} = node;
```

```
//
//  console.log(column,b)


//  //有配置走配置，没配置走默认
//  var {x:y} = {};
//  console.log(y)



//  let str = 'miaov';
////    let {length:l} = str;
//  let [a,b,c,d] = str;
//
//  console.log(a);

    //console.log(l);


    //查看下面数据是不是数组
//  let arr = [1,2,3,4];
//  let {push} = arr;
//  console.log(push == Array.prototype.push);

//  let arr = [1,2,3,4];
//  let {hehe} = arr;
//  console.log(hehe)
```

变量的解构赋值用途很多。

（1）交换变量的值

```
let x = 1;
let y = 2;

[x, y] = [y, x];
```

上面代码交换变量x和y的值，这样的写法不仅简洁，而且易读，语义非常清晰。

（2）从函数返回多个值

函数只能返回一个值，如果要返回多个值，只能将它们放在数组或对象里返回。有了解构赋值，取出这些值就非常方便。

```
// 返回一个数组

function example() {
  return [1, 2, 3];
}
let [a, b, c] = example();

// 返回一个对象

function example() {
  return {
    foo: 1,
    bar: 2
  };
}
let { foo, bar } = example();
```

（3）函数参数的定义

解构赋值可以方便地将一组参数与变量名对应起来。

```
// 参数是一组有次序的值
function f([x, y, z]) { ... }
f([1, 2, 3]);

// 参数是一组无次序的值
function f({x, y, z}) { ... }
f({z: 3, y: 2, x: 1});
```

（4）提取JSON数据

解构赋值对提取JSON对象中的数据，尤其有用。

```
let jsonData = {
  id: 42,
  status: "OK",
  data: [867, 5309]
};

let { id, status, data: number } = jsonData;

console.log(id, status, number);
// 42, "OK", [867, 5309]
```

上面代码可以快速提取 JSON 数据的值。

（5）函数参数的默认值

```
jQuery.ajax = function (url, {
  async = true,
  beforeSend = function () {},
  cache = true,
  complete = function () {},
  crossDomain = false,
  global = true,
  // ... more config
}) {
  // ... do stuff
};
```

指定参数的默认值，就避免了在函数体内部再写`var foo = config.foo || 'default foo';`这样的语句。

（6）遍历Map结构

任何部署了Iterator接口的对象，都可以用`for...of`循环遍历。Map结构原生支持Iterator接口，配合变量的解构赋值，获取键名和键值就非常方便。

```
var map = new Map();
map.set('first', 'hello');
map.set('second', 'world');

for (let [key, value] of map) {
  console.log(key + " is " + value);
}
// first is hello
// second is world
```

如果只想获取键名，或者只想获取键值，可以写成下面这样。

```
// 获取键名
for (let [key] of map) {
  // ...
}

// 获取键值
for (let [,value] of map) {
  // ...
}
```

（7）输入模块的指定方法

加载模块时，往往需要指定输入哪些方法。解构赋值使得输入语句非常清晰。

```
const { SourceMapConsumer, SourceNode } = require("source-map");
```