ES6新特性

1 let与const、块级作用域

- 在ES6之前JS是没有块级作用域的，const与let填补了这方面的空白，分别使用let、const声明变量和常量，const与let都是块级作用域。
- 使用var定义的变量为函数级作用域：
- 使用let与const定义的变量为块级作用域：

```
{
    var i = 0;
    var i = 1;
    let j = 0;
    const k = 1;
}
console.log("i:" + i); // 正常输出 i = 1;
//console.log("j:" + j); // 报错,Uncaught ReferenceError: j is not defined
//console.log("k:" + k); // 报错,Uncaught ReferenceError: K is not defined
```

2 函数参数默认值

- ES6支持在定义函数的时候为其设置默认值：传值覆盖，空值默认

  格式：function 函数名称(参数名称1=值1，参数名称2=值2，.....){}

```
// ES5
function method(name,age){
    name = name || 'XXX';
    age = age || 1;
    console.log(name + ":" +  age);
}
method();// XXX:1
method('小明',16); // 小明:16
method('小明',0);  // 小明:1,参数0表示为false
// ES6
function method1(name = 'YYY',age = 1){
    console.log(name + ":" +  age);
}
method1();// YYY:1
method1('小张',16); // 小明:16
method1('小张',0);  // 小明:0
```

3 箭头函数

- 箭头函数=>不只是关键字function的简写，类似于部分强类型语言中的lambda表达式
- 箭头函数用 => 符号来定义。
- 箭头函数相当于匿名函数，所以采用函数表达式的写法。
- 基本结构

匿名函数：function(参数){}

箭头函数：(参数)=>{}

- 箭头函数的箭头=>之前是一个空括号、单个的参数名、或用括号括起的多个参数名，而箭头之后可以是一个表达式（作为函数的返回值），或者是用花括号括起的函数体（需要自行通过return来返回值，否则返回的是undefined）。

```
//常规函数表达式书写方式
let sum0 = function(x,y){
    return x + y;
}
// 箭头函数 ()=>{} 完整写法
let sum1 = (x,y) => {
    return x + y;
}
```

- 箭头函数例子:某些情况进一步简写

（1）当要执行的代码块只有一条return语句时，可省略大括号和return关键字：

（2）当传入的参数只有一个时，可以省略小括号：

（3）当不需要参数时，使用空的圆括号：

```
//某些情况简写
// （1）当要执行的代码块只有一条return语句时，可省略大括号和return关键字：
let sum2 = (x,y) => x + y;

// （2）当传入的参数只有一个时，可以省略小括号：
let sum3 = function(x){
    return x + x;
}
let sum4 = x => x + x;

// （3）当不需要参数时，使用空的圆括号：
let sum5 = function(){
    return 100 + 100;
}
let sum6 = ()=> 100 + 100;
```

4 对象属性缩写

- 在ES6中允许我们在设置一个对象的属性(函数)的时候不指定属性名(函数名)。
- 当对象属性（函数）名称和外部变量（函数）名称，同名的情况下，可以省略属性名的重复书写以及冒号。

```
// 数据值
const name = "小明";
const age = 16;
const city = "上海";
// 构建一个对象
const student = {
    // ES5语法
    // 属性名:属性值,
    name: name,
```

```
    age: age,
    city: city,
    show: function() {
        console.log("show~~");
    }
}
// 输出对象信息
console.log(student); // {name: "小明", age: 16, city: "上海"}
student.show();
// 使用ES6，当属性名和外部变量名同名情况，可以直接写入变量名,更加简写
// 构建一个对象
const student1 = {
    // ES6语法,此时属性名就是变量名，属性的值就是变量的值。
    name,
    age,
    city,
    show() {
        console.log("show~~");
    }
}
// 输出对象信息
console.log(student1); // {name: "小明", age: 16, city: "上海"}
student1.show();
```

5 模板字符串

- ES6支持模板字符串，使得字符串的拼接更加的简洁、直观。
- 不使用模板字符串：

  在ES5中字符串拼接通过【+】实现

```
let first = "张";
let last = "四";
let name = 'Your name is ' + first + ' ' + last + '.'
```

- 使用模板字符串：

  ES6中使用反引号【``】来拼接字符串，使用【${}】来包含变量

```
let first = "张";
let last = "四";
let name = `Your name is ${first} ${last}.`;
```

6 模块化(Module)

- 导出(export)ES6允许在一个模块中使用export来导出多个变量或函数。
  - 导出变量

    ES6将一个文件视为一个模块，上面的模块通过 export 向外输出了一个或多个变量。

    test.js

```
// 导出一个变量
// export var name = 'Jane'
// 同时往外面输出多个变量。
var name = 'Jane';
var age = '24';
export {name, age};
```

- 导出函数

```
test2.js
// 导出一个函数
export function show(arg) {
    return arg;
}
```

- 导入(import)

  定义好模块的输出以后就可以在另外一个模块通过import引用。

```
test3.js
// 导入
import {name,age} from 'test';// test.js
import {show} from 'test2';// test2.js

```

- ES5不支持原生的模块化，在ES6中模块作为重要的组成部分被添加进来。模块的功能主要由 export 和 import 组成。每一个模块都有自己单独的作用域，模块之间的相互调用关系是通过 export 来规定模块对外暴露的接口，通过import来引用其它模块提供的接口。同时还为模块创造了命名空间，防止函数的命名冲突。
- 在js文件中通过import引入另外的js文件，提示Uncaught SyntaxError: Unexpected string，或者Uncaught SyntaxError: Unexpected identifier错误，原因是import属于es6的语法，但是浏览器不支持es6语法，所以需要转换。后期的Vue中会接解决这个问题。