ECMAScript6 - Advice

github地址 (https://github.com/lixinliang/blog/tree/master/notebook/2016.04.28-es6-advice)

前言

相信大家都对es6已经有初步的认识与了解，这里就不重复介绍了。每个人都有自己的学习方式与方法，大家可以对感兴趣的内容自行搜索了解。

以前，我们还没有把es6加入到工作流之中，谈论使用，还为时过早。但现在，我们有了 `generator-gw` ，我们随时随地都可以作出新的尝试了。

然而，大家可能心中还有一丝疑虑，使用es6就意味着不兼容。在ie7，ie8等浏览器，连es5的api都没有达到完全支持，又怎么使用es6呢?

所以讨论一下：关于es6的使用建议。

顺便把一些容易掉坑的地方都踩一遍。

语法	描述	IE7
let		√
const		√
destructuring	解构赋值	√
template-string	模板字符串	√
default	解构赋值与函数参数的默认值	√
rest		√
spread	扩展运算符	√
arrow-function	箭头函数	√

object-literals	属性的简洁表示法	√
concise-methods	方法的简洁表示法	√
computed-property-names	属性名表达式	√

destructuring

康熙谁来了

```
"uid": 200[illegible]7163,
"title": "蓝波【舞】",
"desc": "台湾明星舞蹈老师",
"time": "4月15日 20:00",
"poster": "http://img.dwstatic.com/webme/1604/323868140264/1459912945835.jpg",
"video": {
    "mp4": "http://yycloudvod1346469499.bs2dl.yy.com/dmU5MGMyYmVlNmZkZDVhODI4NW
    "m3u8": "http://yycloudvod1346469499.bs2dl.yy.com/dmU5MGMyYmVlNmZkZDVhODI4N
},
"isLive": 0,
"isSecret": 0,
```

```
var { uid, title, desc, time, poster, video, isLive, isSecret } = data;
var item = mediaItem.clone();
item.find('.ui-media__img').prop('src', poster);
item.find('.ui-media__tag').html('预告');
item.find('.ui-media__tit').html(title);
item.find('.ui-media__desc').html(desc);
item.find('.ui-media__time').html(time);
```

春日优惠礼包

```
var { actId, propId, productId, totalCount, saledCount, displaySaledCount, status, srcamount = 30 } = result.data[0];
body.find('.ui-tips').data('num', (+totalCount) - saledCount).displaySaledCount();
var btn = body.find('.ui-btn').data('propId', propId).data('productId', productId).data('srcamount', srcamount);
```

```
▼ data: [{actId: "5711e744bde481aeb05f5690", name: "豪华套装", propId: 210015,…}]
  ▼ 0: {actId: "5711e744bde481aeb05f5690", name: "豪华套装", propId: 210015,…}
      actId: "5711e744bde481aeb05f5690"
      displaySaledCount: 3
      name: "豪华套装"
      position: 1
      productId: "com.yy.ourtimes.propspacket30"
      propId: 210015
      saledCount: 1
      status: 0
      totalCount: 20
```

遇到过比较坑的一次是，使用了 `var [...]` 来解构对象，导致全部变量都解构失败。所以使用解构赋值时，先确认解构对象的 类型 ，如果是解构对象，还要确认，变量名是否为解构对象的 键名 ，如果是解构数组，则确认变量的顺序是 一一对应 。

object-literals

康熙谁来了

```
var { mp4, m3u8 } = video;
var id = mobileVideoList.length;
mobileVideoList.push({ id, poster, mp4, m3u8 });
```

```
// 定义视频播放列表信息，其中`id`与上面step1说的videoID相对应
let mobileVideoList = [
    {
        id: 'video1',
        poster: '',
        mp4: '',
        m3u8: ''
    },
    {
        id: 'video2',
        poster: '',
        mp4: '',
        m3u8: ''
    }
];
```

当对象的属性与赋值的变量一致时，我们可以使用简洁表示法。

arrow-function

`function` 这个关键字不仅词频高，字符还长。而箭头函数就是为了解决这个问题，用更简洁的表示方式，表达我们的编程思路。

使用箭头函数时，我们可以把 `=>` 理解为一个操作符，比如下面这个例子 `var a = a => a;` 。当出现 `=>` 时，那么它的运算结果就是一个函数。而在 `=>` 的左边是这个函数的形参，而在 `=>` 的右边是这个函数的返回值。

是不是这样理解就一目了然了呢?

那么，我们只需要关注 `=>` 的两侧。

在箭头的左侧，有以下表达方式:

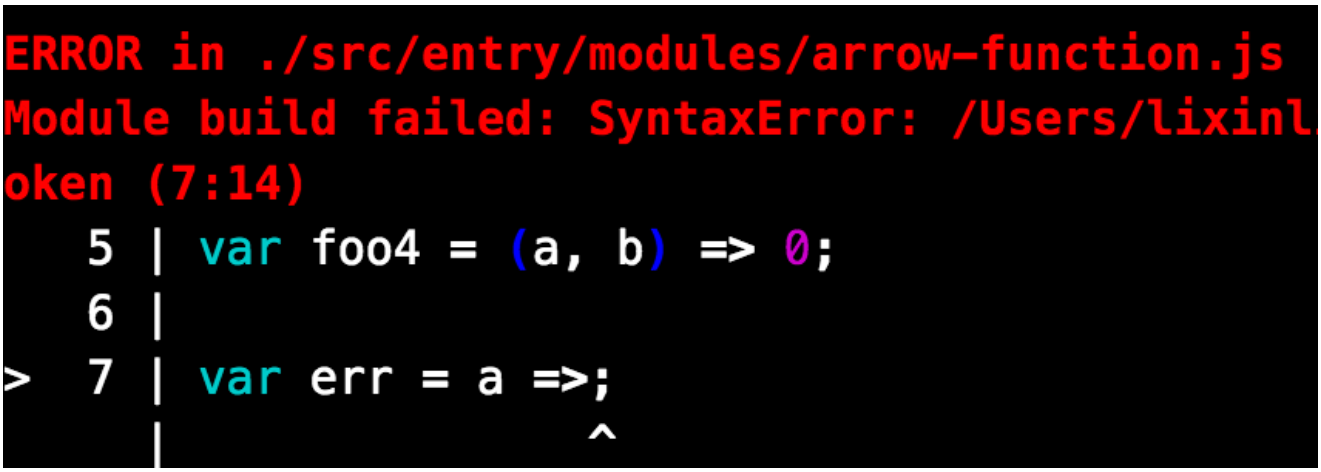
```
1
2   var foo1 = a => 0;
3   var foo2 = (a) => 0;
4   var foo3 = () => 0;
5   var foo4 = (a, b) => 0;
6
```

JS index.js ×

```
66        var foo1 = function foo1(a) {
67          return 0;
68        };
69        var foo2 = function foo2(a) {
70          return 0;
71        };
72        var foo3 = function foo3() {
73          return 0;
74        };
75        var foo4 = function foo4(a, b) {
76          return 0;
77        };
```

为了避免混淆，箭头函数的形参部分，必须使用()。

无论是箭头的左侧还是右侧，缺失的话，都会抛出错误。



而在箭头的右侧，情况就更多了，我们可以归类为三种情况，第一种是返回一个值，第二种是返回一个表达式，第三种是代码块。

我们先看只返回一个值的时候。这个值可以是任意类型。

```
10  var foo1 = (a) => 0;
11  var foo2 = (a) => '0';
12  var foo3 = (a) => true;
13  var foo4 = (a) => [];
14  var foo5 = (a) => [0];
15  var foo6 = (a) => {};
16  var foo7 = (a) => { a : 0 };
17  var foo8 = (a) => function(){};
18  var foo9 = (a) => undefined;
19  var foo10 = (a) => null;
20  var foo11 = (a) => /\d/;
21  var foo12 = (a) => a;
```

JS index.js ×

```
81      var foo1 = function foo1(a) {
82        return 0;
83      };
84      var foo2 = function foo2(a) {
85        return '0';
86      };
87      var foo3 = function foo3(a) {
88        return true;
89      };
90      var foo4 = function foo4(a) {
91        return [];
92      };
93      var foo5 = function foo5(a) {
94        return [0];
95      };
96      var foo6 = function foo6(a) {};
97      var foo7 = function foo7(a) {
98        a: 0;
99      };
```

```
100      var foo8 = function foo8(a) {
101        return function () {};
102      };
103      var foo9 = function foo9(a) {
104        return undefined;
105      };
106      var foo10 = function foo10(a) {
107        return null;
108      };
109      var foo11 = function foo11(a) {
110        return (/\d/
111        );
112      };
113      var foo12 = function foo12(a) {
114        return a;
115      };
```

这里一切都与我们的预期一样，除了有一点，就是 对象 。因为js里面，解析引擎无法识别 {} 是代码块还是对象。在箭头函数这里，只要遇到 {} 一律解析为代码块。

而且比较坑的是，当我们返回对象时，无论编译还是运行都会报错。但返回只有一个键对值的对象时，却不会提示错误。

```
ERROR in ./src/entry/modules/arrow-function.js
Module build failed: SyntaxError: /Users/lixinliang/
token (25:29)
  23 |
  24 | var err1 = (a) => { a : 0 };
> 25 | var err2 = (a) => { a : 0, b : 1, c : 2 };
     |                             ^
```

```
> typeof a
< "undefined"
> a:0
< 0
```

那是因为， `a:0` 这是一句符合规范的代码。@see label syntax (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/label)

那么，正确的返回对象的箭头函数的写法应该如下：

```
var foo1 = (a) => ({});
var foo2 = (a) => ({ a : 0 });
var foo3 = (a) => ({ a : 0, b : 1, c : 2 });
```

JS index.js ×

```
var foo1 = function foo1(a) {
  return {};
};
var foo2 = function foo2(a) {
  return { a: 0 };
};
var foo3 = function foo3(a) {
  return { a: 0, b: 1, c: 2 };
};
```

使用（）包起来。

还有返回箭头函数，用es6的新语法定义的对象等：

```
31  var foo1 = (a) => () => a;
32  var foo2 = (a) => ({ a });
33  var foo3 = (a = 3) => ({ a });
34
```

index.js

```
130     var foo1 = function foo1(a) {
131       return function () {
132         return a;
133       };
134     };
135     var foo2 = function foo2(a) {
136       return { a: a };
137     };
138     var foo3 = function foo3() {
139       var a = arguments.length <= 0 || arguments[0] === undefined ? 3 : arguments[0];
140       return { a: a };
141     };
```

然后第二种情况是表达式：

优先级	运算类型	关联性	运算符
19	圆括号	n/a	(…)
18	成员访问	从左到右	… . …
	需计算的成员访问	从左到右	… […]
	new (带参数列表)	n/a	new … (…)
17	函数调用	从左到右	… (…)
	new (无参数列表)	从右到左	new …
16	后置递增(运算符在后)	n/a	… ++
	后置递减(运算符在后)	n/a	… --
15	逻辑非	从右到左	! …
	按位非	从右到左	~ …
	一元加法	从右到左	+ …
	一元减法	从右到左	- …
	前置递增	从右到左	++ …
	前置递减	从右到左	-- …
	typeof	从右到左	typeof …
	void	从右到左	void …
	delete	从右到左	delete …
14	乘法	从左到右	… * …
	除法	从左到右	… / …
	取模	从左到右	… % …

13	加法	从左到右	… + …
	减法	从左到右	… - …
12	按位左移	从左到右	… << …
	按位右移	从左到右	… >> …
	无符号右移	从左到右	… >>> …
11	小于	从左到右	… < …
	小于等于	从左到右	… <= …
	大于	从左到右	… > …
	大于等于	从左到右	… >= …
	in	从左到右	… in …
	instanceof	从左到右	… instanceof …
10	等号	从左到右	… == …
	非等号	从左到右	… != …
	全等号	从左到右	… === …
	非全等号	从左到右	… !== …
9	按位与	从左到右	… & …
8	按位异或	从左到右	… ^ …
7	按位或	从左到右	… | …
6	逻辑与	从左到右	… && …
5	逻辑或	从左到右	… || …
4	条件运算符	从右到左	… ? … : …
3	赋值	从右到左	… = …
			… += …
			… -= …
			… *= …
			… /= …
			… %= …
			… <<= …
			… >>= …
			… >>>= …
			… &= …
			… ^= …
			… |= …

2	yield	从右到左	`yield …`
	yield*	从右到左	yield* …
1	Spread	n/a	`...` …
0	逗号	从左到右	… , …

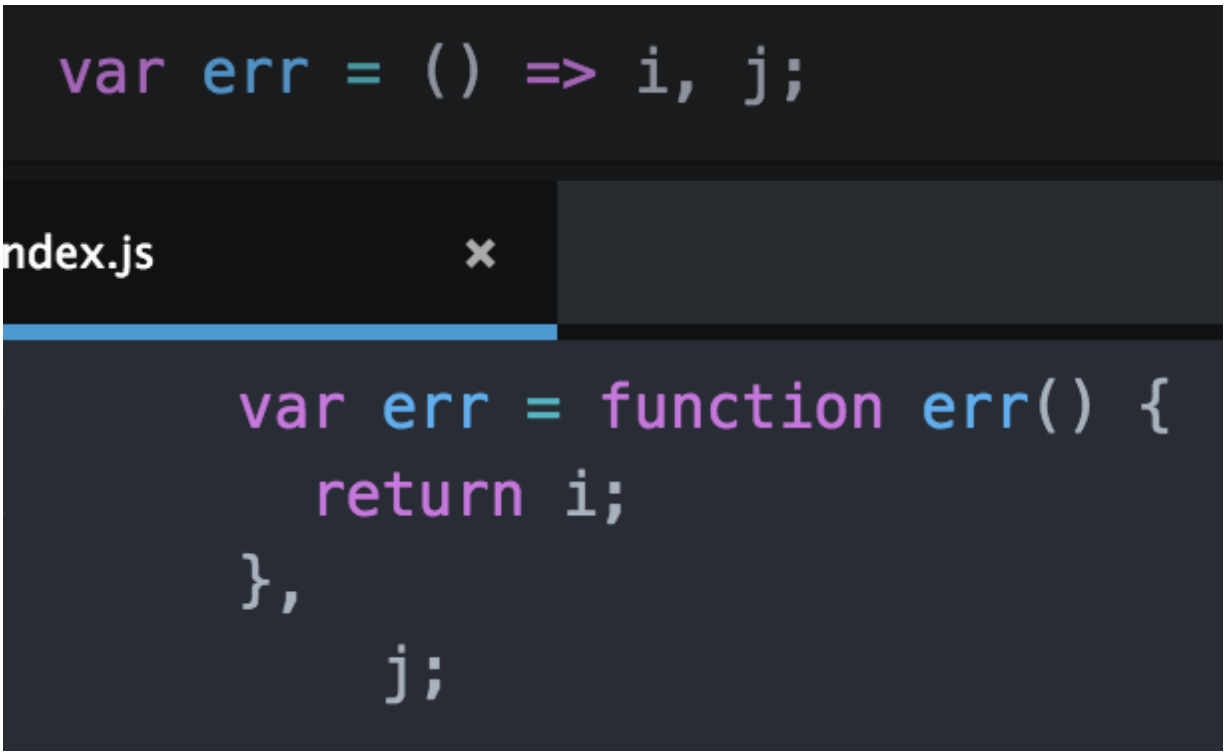
```
var foo1 = () => document.body;
var foo2 = () => document['body'];
var foo3 = () => new Image;
var foo4 = () => typeof window;
var foo5 = () => void 0;
var foo6 = () => 1 + 1;
var foo7 = () => 2 > 1;
var foo8 = () => i += j;
```

index.js

```
var foo1 = function foo1() {
  return document.body;
};
var foo2 = function foo2() {
  return document['body'];
};
var foo3 = function foo3() {
  return new Image();
};
var foo4 = function foo4() {
  return typeof window === 'undefined' ? 'undefined' : _typeof(window);
};
var foo5 = function foo5() {
  return void 0;
};
var foo6 = function foo6() {
  return 1 + 1;
};
var foo7 = function foo7() {
  return 2 > 1;
};
var foo8 = function foo8() {
  return i += j;
};
```

箭头函数会将表达式的结果作为返回值。

但是遇到逗号操作符则不会。



所以优先级大概是0~3之间。

第三种情况是代码块。

```
var foo = () => {
    // some code
    return 0
};
```

以上是使用箭头函数，要注意的写书格式问题。
除此以外，还有要注意的关于用法上的地方：
1. this
2. arguments
3. new

在箭头函数内，没有 `this` 也没有 `arguments` 。

```
function outer () {
    var foo = () => {
        console.log(this, arguments);
    };
    return foo
}
```

index.js

```
function outer() {
    var _this = this,
        _arguments = arguments;

    var foo = function foo() {
        console.log(_this, _arguments);
    };
    return foo;
}
```

```
> var foo = () => this;
< undefined
> foo();
< ▶ Window {external: Object,
> foo.call(document);
< ▶ Window {external: Object,
>
```

所以，什么情况下使用 function 什么情况下使用 => 由开发者自行判断。

```
var foo = () => 0;

console.log(typeof foo); // function

new foo(); // Uncaught TypeError: foo is not a constructor
```

箭头函数的类型是函数但不能使用new关键字。

立即执行函数IIFE(Immediately-Invoked Function Expression)

```
((win) => {
    // some code
})(window);
```