网上接触到的，把实际项目中感觉可以用上的整理出来，方便查看

#### 一些风格

##### 1.对象尽量静态化，一旦定义，就不得随意添加新的属性。如果添加属性不可避免，要使用Object.assign方法。

```bash

// bad

const a = {};

a.x = 3;

// if reshape unavoidable

const a = {};

Object.assign(a, { x: 3 });

// good

const a = { x: null };

a.x = 3;

```

##### 2.多用箭头函数

```bash

(() => { //立即执行函数

console.log('Welcome to the Internet.');

})();

[1, 2, 3].map(x => x \* x);

```

<!--More-->

#### 详细区别

##### 1.let声明的变量,只在代码块内有效

```bash

var a = [];

for (let i = 0; i < 10; i++) { //i只在循环体内部有效 用var循环外部也可以调用i

a[i] = function () {

console.log(i);

};

let i = 'abc'; //函数内部的变量i和外部的变量i是分离的(循环语句部分是一个父作用域，而循环体内部是一个单独的子作用域)

console.log(i); //输出10次abc

}

a[6](); // 6 当前的i只在本轮循环有效，所以每一次循环的i其实都是一个新的变量，如果是var新的i值都会覆盖旧值，导致最后输出的是最后一轮的i的值，也就是10

```

##### 2.let没有变量提升

```bash

var tmp = new Date();

function f() {

console.log(tmp); // undefined 内存变量提升，覆盖外部变量

if (false) {

var tmp = "hello world";

}

}

function f1() {

console.log(n);// 报错ReferenceError

let n = 5;

if (true) {

let n = 10; //只在作用域内有效

}

console.log(n); // 5 ,不受10的影响

}

有了块作用域之后，立即执行函数就没必要写了

// IIFE 写法

(function () {

var tmp = ...;

...

}());

// 块级作用域写法

{

let tmp = ...;

...

}

```

##### 3.暂时性死区

```bash

var tmp = 123;

if (true) {

tmp = 'abc'; // ReferenceError let所在的区域不受外部影响，无法赋值

let tmp; // 定义之前的块作用域为暂时性死区

console.log(tmp); // undefined

}

```

##### 4.const变量

```bash

const PI = 3.1415;

PI = 3; // TypeError: Assignment to constant variable. const变量不能赋值

const foo; // SyntaxError: Missing initializer in const declaration .必须立刻赋值

const foo = {}; //存储了一个地址，地址指向一个对象

foo.prop = 123; //对象本身是可变的

foo.prop // 输出123

foo = {}; // TypeError: "foo" is read-only //不能更换地址

const a = [];

a.push('Hello'); // 可执行

a.length = 0; // 可执行

a = ['Dave']; // 报错 常量a是一个数组，这个数组本身是可写的，但是如果将另一个数组赋值给a就不对

```

##### 5.变量的解构赋值

```bash

let [head, ...tail] = [1, 2, 3, 4]; // head//1 tail//[2, 3, 4]

let [x, y, ...z] = ['a']; // x//"a" y//undefined z//[]

// 报错 转为对象以后不具备Iterator 接口（前五个表达式），本身就不具备Iterator接口（最后一个表达式）。

let [foo] = 1;

let [foo] = false;

let [foo] = NaN;

let [foo] = undefined;

let [foo] = null;

let [foo] = {};

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

let [x = 1] = [null]; x // null 如果一个数组成员不严格等于undefined，默认值是不会生效的。

let [x = f()] = [1]; // x能取到值，所以函数f根本不会执行。

[[1, 2], [3, 4]].map(([a, b]) => a + b); // [ 3, 7 ]

[1, undefined, 3].map((x = 'yes') => x); // [ 1, 'yes', 3 ] undefined会触发函数参数的默认值。

map结构

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

}

for (let [key] of map) {// 获取键名

// ...

}

for (let [,value] of map) {// 获取键值

// ...

}

```

##### 6.字符串新增方法

```bash

var s = 'Hello world!';

s.startsWith('world', 6) // true 六之后的字符

s.includes('Hello', 6) // false 六之后的字符

s.endsWith('Hello', 5) // true 前五个个字符

'x'.repeat(3) // "xxx"

'na'.repeat(0) // ""

'na'.repeat(-0.9) // "" 会先进行取整运算。0到-1之间的小数，取整以后等于-0，repeat视同为0。

'na'.repeat('na') // "" 参数为字符串会先转换为数字

'na'.repeat('3') // "nanana"

'na'.repeat(NaN) // "" NAN undefined null 都会当做0处理

padStart()，padEnd() 字符串补全长度

```

##### 7.模板字符串

```bash

$('#result').append(

'There are <b>' + basket.count + '</b> ' +

'items in your basket, ' +

'<em>' + basket.onSale +

'</em> are on sale!'

);

$('#result').append(`

There are <b>${basket.count}</b> items

in your basket, <em>${basket.onSale}</em>

are on sale!

`);

`In JavaScript '\n' is a line-feed.`

`In JavaScript this is

not legal.`

大括号内部可以放入任意的JavaScript表达式(函数也可)，可以进行运算，以及引用对象属性。

var x = 1;

var y = 2;

`${x} + ${y \* 2} = ${x + y \* 2}` "1 + 4 = 5"

var obj = {x: 1, y: 2};

`${obj.x + obj.y}` // 3

```

##### 8.数值的新增方法

```bash

// ES5的写法

parseInt('12.34') // 12

parseFloat('123.45#') // 123.45

// ES6的写法(逐步减少全局性方法，使得语言逐步模块化。)

Number.parseInt('12.34') // 12

Number.parseFloat('123.45#') // 123.45

Math.trunc方法用于去除一个数的小数部分，返回整数部分。

Math.sign方法用来判断一个数到底是正数、负数、还是零。(+1,0,-0,-1)

```

##### 9.数组的新增方法

```bash

Array.from() 将类数组对象和可遍历对象，转为真正的数组

let arrayLike = {

'0': 'a',

'1': 'b',

'2': 'c',

length: 3

};

// ES5的写法

var arr1 = [].slice.call(arrayLike); // ['a', 'b', 'c']

// ES6的写法

let arr2 = Array.from(arrayLike); // ['a', 'b', 'c']

Array.of() 方法用于将一组值，转换为数组。

Array.of(3, 11, 8) // [3,11,8]

Array(3) // [, , ,] 以前的只写一个参数，会以为是数组长度，有解析差异

```

##### 10.函数的新增方法

```bash

var x = 1;

function foo(x, y = function() { x = 2; }) { //foo函数的作用域为A

var x = 3; //作用域为B ，如果去掉var x的作用域和foo一样为A，那么foo()的结果就为2了

y();

console.log(x);

}

foo() // 3

x // 1

```

##### 11.对象的新增方法

```bash

var birth = '2000/01/01';

var Person = {

name: '张山',

birth, //等同于birth: birth

hello() { console.log('我的名字是', this.name); } // 等同于hello: function ()...

};

同值相等算法

+0 === -0 //true

NaN === NaN // false

Object.is(+0, -0) // false

Object.is(NaN, NaN) // true

```

##### 12.set结构

```bash

var set = new Set([1, 2, 3, 4, 4]);

[...set] // [1, 2, 3, 4]

// 去除数组的重复成员

[...new Set(array)] //直接去重

function dedupe(array) {

return Array.from(new Set(array)); //Array.from方法可以将Set结构转为数组。

}

dedupe([1, 1, 2, 3]) // [1, 2, 3]

add(value)：添加某个值，返回Set结构本身。

delete(value)：删除某个值，返回一个布尔值，表示删除是否成功。

has(value)：返回一个布尔值，表示该值是否为Set的成员。

clear()：清除所有成员，没有返回值。

keys()：返回键名的遍历器

values()：返回键值的遍历器

entries()：返回键值对的遍历器

forEach()：使用回调函数遍历每个成员

```

##### 13. class类

```bash

class Point {

constructor(){

// ...

}

}

Object.assign(Point.prototype, {

toString(){},

toValue(){}

});

```

主要参考：http://es6.ruanyifeng.com/

未完待续~