**export\_1516687482071**

exports与 module.exports属于 commonJS规范

export与export default 属于 ES6规范

都属于导出语法

分别对应的导入语法： require（） import

exports 与 module.exports的区别

commonJS将每个文件视为一个模块，每个模块内部，module变量代表当前模块，这个变量是一个对象，它的exports属性（module.exports）是对外的接口。加载一个模块，就是加载这个module.exports属性。

而exports指向 module.exports，可以改变exports对象的属性，但是不能直接将exports指向另一个值。

export 与 export default的区别和联系

是ES6中导出的接口，一个是导出一个一个的单独接口，一个是导出整体的接口。

**absolute\_1516945381838**

absolute定位于scroll元素的表现

定位元素会固定在一开始的位置，并不会随滚动而滚动。（滚动的部分为超出部分）

#### [anonymous function](C:/Users/Administrator/AppData/Local/youdao/dict/Application/7.5.2.0/resultui/dict/javascript:;)\_1519290561525

实际上的匿名函数也包括表达式函数（但是）

recursion\_1519611483438

一、函数调用

function isPalindrome( text ) {

if ( text.length <=1 ) return true

if( text.charAt(0) !== text.charAt(text.length -1) ) return false

return isPalindrome( text.substr( 1, text.length - 2 ) )

}

二、方法调用

var ninja = {

chirp: function(n) {

return n > 1 ? ninja.chirp(n - 1) + "-chirp" : "chirp";

}

};

var samurai = { chirp: ninja.chirp };

**优化**

return n > 1 ? this.chirp(n - 1) + "-chirp" : "chirp";

1. **内联命名函数**

上面的问题在于，其他对象引用时必须给方法命名为chirp，而用内联命令函数则不会有这个问题

var ninja = {

chirp: function signal(n) { //#1

return n > 1 ? signal(n - 1) + "-chirp" : "chirp";

}

};

内联函数的名称在函数外是不可用的。只在函数内部是可用的。如：

var a = function b() {

}

b -- 》 undefined

这也是为什么全局属性会被作为window的方法进行创建的原因，不使用window的属性，无法引用这些函数。

注意： arguments.callee() 也可以调用当前的函数，但是，es5规范中禁止使用这个属性。

函数存储

问题：如何存储一组独立的函数？

关键： 可以将函数视为对象，在其上添加属性

函数属性存储优势

将状态和缓存信息存储在独立位置

利于代码组织

外部存储或缓存对象无需污染作用域

var store = {

nextId: 1, //#1

cache: {}, //#2

add: function(fn) { //#3

if (!fn.id) { //#3

fn.id = store.nextId++; //#3

return !!(store.cache[fn.id] = fn); //#3

} //#3

}

};

**自记忆函数**

用于缓存记忆昂贵的计算结果

function isPrime(value) {

if (!isPrime.answers) isPrime.answers = {}; //#1

if (isPrime.answers[value] != null) { //#2

return isPrime.answers[value]; //#2

} //#2

var prime = value != 1; // 1 can never be prime

for (var i = 2; i < value; i++) {

if (value % i == 0) {

prime = false;

break;

}

}

return isPrime.answers[value] = prime; //#3

}

但是也要权衡一下缺点

节省了计算性能，但是任何类型的缓存肯定会牺牲内存。

缓存是否应该与业务逻辑放在一起？（函数或方法应该只做一件事？）

缓存记忆DOM元素

function getElements( name ) {

if( !getElements.cache ) getElements.cache = {};

return getElements.cache[name] ||

document.getElementsByTagName(name)

}

类数组缓存

var elems = {

length: 0,

add: function(elem) {

Array.prototype.push.call(this, elem)

},

gather: function(id) {

this.add(document.getElementById(id))

}

}