**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。

fetch\_1519795453831

ajax的取代品 ：fetch

fetch是基于Promise设计

一、用await

    try {

      const response = await fetch(url, requestConfig);

      const responseJson = await response.json();

      return responseJson

    } catch (error) {

      throw new Error(error)

    }

**二、promise形式**

fetch(url).then(response => response.json())

.then(data => console.log(data))

.catch(e => console.log("Oops, error", e))

**问题：**

1. Fetch 请求默认是不带 cookie 的，需要设置 fetch(url, {credentials: 'include'})

2.服务器返回 400，500 错误码时并不会 reject，只有网络错误这些导致请求不能完成时，fetch 才会被 reject。

3. IE不支持原生Fetch。

cross-origin\_1520389506693

1. 通过jsonp跨域

**能实现不同二级域之间的通信**

根本原理在于，服务端监听到传过来的回调函数fn

返回一个全局脚本

fn（{...}）

缺点：

必须有一个fn全局函数。

2、 document.domain + iframe跨域

**能实现同一二级域下的子域之间的iframe交互**

缺点：

必须在两个页面都设置同一个domain，且只能设置为自身或者上一级domain，而且都要设置

document.domain = ‘dongyuewu.me’

3、 location.hash + iframe

**能实现不同二级域之间的iframe交互**

原理：通过中间页面 来沟通交互

4、 window.name + iframe跨域  
5、 postMessage跨域  
6、 跨域资源共享（CORS）  
7、 nginx代理跨域  
8、 nodejs中间件代理跨域  
9、 WebSocket协议跨域

**iframe\_1520413438349**

1.无法再iframe加载前获取iframe

也就是说 document.getElementById(‘iframe’)必须写在iframe创建之后

1. 无法给iframe设定onload时间句柄

因为iframe的加载事件是在iframe的文档对象模型中激活的，而不是父页面。