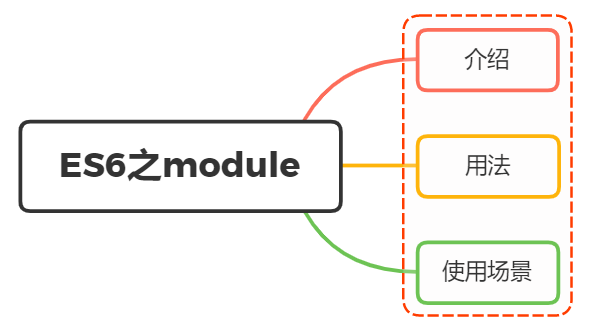
# 面试官：你是怎么理解ES6中Module的？使用场景？



## 一、介绍

模块，（Module），是能够单独命名并独立地完成一定功能的程序语句的**集合（即程序代码和数据结构的集合体）**。

两个基本的特征：外部特征和内部特征

* 外部特征是指模块跟外部环境联系的接口（即其他模块或程序调用该模块的方式，包括有输入输出参数、引用的全局变量）和模块的功能
* 内部特征是指模块的内部环境具有的特点（即该模块的局部数据和程序代码）

### 为什么需要模块化

* 代码抽象
* 代码封装
* 代码复用
* 依赖管理

如果没有模块化，我们代码会怎样？

* 变量和方法不容易维护，容易污染全局作用域
* 加载资源的方式通过script标签从上到下。
* 依赖的环境主观逻辑偏重，代码较多就会比较复杂。
* 大型项目资源难以维护，特别是多人合作的情况下，资源的引入会让人奔溃

因此，需要一种将JavaScript程序模块化的机制，如

* CommonJs (典型代表：node.js早期)
* AMD (典型代表：require.js)
* CMD (典型代表：sea.js)

### AMD

Asynchronous ModuleDefinition（AMD），异步模块定义，采用异步方式加载模块。所有依赖模块的语句，都定义在一个回调函数中，等到模块加载完成之后，这个回调函数才会运行

代表库为require.js

/\*\* main.js 入口文件/主模块 \*\*/  
// 首先用config()指定各模块路径和引用名  
require.config({  
 baseUrl: "js/lib",  
 paths: {  
 "jquery": "jquery.min", //实际路径为js/lib/jquery.min.js  
 "underscore": "underscore.min",  
 }  
});  
// 执行基本操作  
require(["jquery","underscore"],function($,\_){  
 // some code here  
});

### CommonJs

CommonJS 是一套 Javascript 模块规范，用于服务端

// a.js  
module.exports={ foo , bar}  
  
// b.js  
const { foo,bar } = require('./a.js')

其有如下特点：

* 所有代码都运行在模块作用域，不会污染全局作用域
* 模块是同步加载的，即只有加载完成，才能执行后面的操作
* 模块在首次执行后就会缓存，再次加载只返回缓存结果，如果想要再次执行，可清除缓存
* require返回的值是被输出的值的拷贝，模块内部的变化也不会影响这个值

既然存在了AMD以及CommonJs机制，ES6的Module又有什么不一样？

ES6 在语言标准的层面上，实现了Module，即模块功能，完全可以取代 CommonJS和 AMD规范，成为浏览器和服务器通用的模块解决方案

CommonJS 和AMD 模块，都只能在运行时确定这些东西。比如，CommonJS模块就是对象，输入时必须查找对象属性

// CommonJS模块  
let { stat, exists, readfile } = require('fs');  
  
// 等同于  
let \_fs = require('fs');  
let stat = \_fs.stat;  
let exists = \_fs.exists;  
let readfile = \_fs.readfile;

ES6设计思想是尽量的静态化，使得编译时就能确定模块的依赖关系，以及输入和输出的变量

// ES6模块  
import { stat, exists, readFile } from 'fs';

上述代码，只加载3个方法，其他方法不加载，即 ES6 可以在编译时就完成模块加载

由于编译加载，使得静态分析成为可能。包括现在流行的typeScript也是依靠静态分析实现功能

## 二、使用

ES6模块内部自动采用了严格模式，这里就不展开严格模式的限制，毕竟这是ES5之前就已经规定好

模块功能主要由两个命令构成：

* export：用于规定模块的对外接口
* import：用于输入其他模块提供的功能

### export

一个模块就是一个独立的文件，该文件内部的所有变量，外部无法获取。如果你希望外部能够读取模块内部的某个变量，就必须使用export关键字输出该变量

// profile.js  
export var firstName = 'Michael';  
export var lastName = 'Jackson';  
export var year = 1958;  
  
或   
// 建议使用下面写法，这样能瞬间确定输出了哪些变量  
var firstName = 'Michael';  
var lastName = 'Jackson';  
var year = 1958;  
  
export { firstName, lastName, year };

输出函数或类

export function multiply(x, y) {  
 return x \* y;  
};

通过as可以进行输出变量的重命名

function v1() { ... }  
function v2() { ... }  
  
export {  
 v1 as streamV1,  
 v2 as streamV2,  
 v2 as streamLatestVersion  
};

### import

使用export命令定义了模块的对外接口以后，其他 JS 文件就可以通过import命令加载这个模块

// main.js  
import { firstName, lastName, year } from './profile.js';  
  
function setName(element) {  
 element.textContent = firstName + ' ' + lastName;  
}

同样如果想要输入变量起别名，通过as关键字

import { lastName as surname } from './profile.js';

当加载整个模块的时候，需要用到星号\*

// circle.js  
export function area(radius) {  
 return Math.PI \* radius \* radius;  
}  
  
export function circumference(radius) {  
 return 2 \* Math.PI \* radius;  
}  
  
// main.js  
import \* as circle from './circle';  
console.log(circle) // {area:area,circumference:circumference}

输入的变量都是只读的，不允许修改，但是如果是对象，允许修改属性

import {a} from './xxx.js'  
  
a.foo = 'hello'; // 合法操作  
a = {}; // Syntax Error : 'a' is read-only;

不过建议即使能修改，但我们不建议。因为修改之后，我们很难差错

import后面我们常接着from关键字，from指定模块文件的位置，可以是相对路径，也可以是绝对路径

import { a } from './a';

如果只有一个模块名，需要有配置文件，告诉引擎模块的位置

import { myMethod } from 'util';

在编译阶段，import会提升到整个模块的头部，首先执行

foo();  
  
import { foo } from 'my\_module';

多次重复执行同样的导入，只会执行一次

import 'lodash';  
import 'lodash';

上面的情况，大家都能看到用户在导入模块的时候，需要知道加载的变量名和函数，否则无法加载

如果不需要知道变量名或函数就完成加载，就要用到export default命令，为模块指定默认输出

// export-default.js  
export default function () {  
 console.log('foo');  
}

加载该模块的时候，import命令可以为该函数指定任意名字

// import-default.js  
import customName from './export-default';  
customName(); // 'foo'

### 动态加载

允许您仅在需要时动态加载模块，而不必预先加载所有模块，这存在明显的性能优势

这个新功能允许您将import()作为函数调用，将其作为参数传递给模块的路径。 它返回一个 promise，它用一个模块对象来实现，让你可以访问该对象的导出

import('/modules/myModule.mjs')  
 .then((module) => {  
 // Do something with the module.  
 });

### 复合写法

如果在一个模块之中，先输入后输出同一个模块，import语句可以与export语句写在一起

export { foo, bar } from 'my\_module';  
  
// 可以简单理解为  
import { foo, bar } from 'my\_module';  
export { foo, bar };

同理能够搭配as、\*搭配使用

## 三、使用场景

如今，ES6模块化已经深入我们日常项目开发中，像vue、react项目搭建项目，组件化开发处处可见，其也是依赖模块化实现

vue组件

<template>  
 <div class="App">  
 组件化开发 ---- 模块化  
 </div>  
</template>  
  
<script>  
export default {  
 name: 'HelloWorld',  
 props: {  
 msg: String  
 }  
}  
</script>

react组件

function App() {  
 return (  
 <div className="App">  
 组件化开发 ---- 模块化  
 </div>  
 );  
}  
  
export default App;

包括完成一些复杂应用的时候，我们也可以拆分成各个模块

## 参考文献

* https://macsalvation.net/the-history-of-js-module/
* https://es6.ruanyifeng.com/#docs/module