

ECMAScript6详解

---- 老孙

课程目标：

- 1、es6简介
- 2、搭建es6环境
- 3、ES6 的语法使用

1.ECMAScript6 简介

ECMAScript 6.0 (以下简称 ES6) 是 JavaScript 语言的下一代标准, 已经在 2015 年 6 月正式发布了。

它的目标, 是使得 JavaScript 语言可以用来编写复杂的大型应用程序, 成为企业级开发语言

1.1 ECMAScript 和 JavaScript 的关系

- 要讲清楚这个问题, 需要回顾历史。
- 1996 年 11 月, JavaScript 的创造者 Netscape 公司, 决定将 JavaScript 提交给标准化组织 ECMA, 希望这种语言能够成为国际标准
- ECMA 发布 262 号标准文件 (ECMA-262) 的第一版, 规定了浏览器脚本语言的标准, 并将这种语言称为 ECMAScript, 这个版本就是 1.0 版。
- 因此, ECMAScript (宪法) 和 JavaScript (律师) 的关系是, 前者是后者的规格, 后者是前者的一种实现

1.2 ES6 与 ECMAScript 2015 的关系

- 2011 年, ECMAScript 5.1 版发布后, 就开始制定 6.0 版了。因此, ES6 这个词的原意, 就是指 JavaScript 语言的下一个版本。
- ES6 既是一个历史名词, 也是一个泛指, 含义是 5.1 版以后的 JavaScript 的下一代标准, 涵盖了 ES2015、ES2016、ES2017 等等

2.搭建前端环境

2.1 Node 环境

2.1.1 什么是Node.js

简单的说 Node.js 就是运行在服务端的 JavaScript。

JavaScript程序, 必须要依赖浏览器才能运行! 没有浏览器怎么办? OK, nodejs帮你解决

Node.js是脱离浏览器环境运行的JavaScript程序, 基于Google的V8引擎, V8引擎执行JavaScript的速度非常快, 性能非常好。

2.1.2 Node.js有什么用

如果你是前端程序员，你不懂得像PHP、Python或Ruby等动态编程语言，然后你想创建自己的服务，那Node.js是一个非常好的选择。

Node.js 是运行在服务端的 JavaScript，如果你熟悉Javascript，那么你将会很容易的学会Node.js。

当然，如果你是后端程序员，想部署一些高性能的服务，那么学习Node.js也是一个非常好的选择。

2.1.3 安装

2.1.3.1 下载

官网：<https://nodejs.org/en/>

中文网：<http://nodejs.cn/>

LTS：长期支持版本

Current：最新版

安装：Windows下双击点击安装——>Next——>finish

注意：

- node-v14.5.0-x64.msi 最新版本，如果是win7系统的话，可能安装不了。
- 如果是win7系统，安装node-v10.14.2-x64.msi这个版本

2.1.3.2 查看版本

在dos窗口中执行命令查看版本号

```
node -v
```

2.1.4.1 创建文件夹 lagou-node

用vscode打开目录，其目录下创建 hello.js

```
console.log("hello,nodejs");
```

打开命令行终端：Ctrl + Shift + y

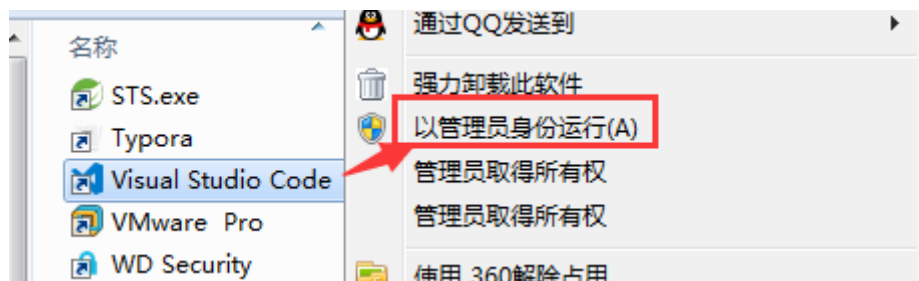
输入命令

```
node hello.js
```

第一次运行，可能会报错！

```
F:\workspace\VScode\node>node v
'node' 不是内部或外部命令，也不是可运行的程序
或批处理文件。
```

兼容性的问题，以管理员身份运行即可



这样，没有使用浏览器，我们也可以运行js程序了

```
F:\workspace\VScode\node>node -v
v10.14.2

F:\workspace\VScode\node>node hello.js
hello,nodejs
```

2.1.4 服务器端应用开发（了解）

- 创建 node-server.js

```
const http = require("http"); // node中自带的require引入方法，http也是node中自带的服务对象
http.createServer( function(request,response){

    // 发出http请求的头部信息
    // http的状态码: 200: OK
    // 请求的内容类型: text/plain
    response.writeHead(200,{"Content-Type": "text/plain"});

    //响应的数据 "hello,welcome! ",此时，并不支持中文（以后一定会解决！）
    response.end("hello,welcome!");
} ).listen(8888); // 监听端口

console.log("服务器已启动，请访问 http://127.0.0.1:8888");
```

- 服务器启动成功后，在浏览器中输入：<http://localhost:8888/> 查看webserver成功运行，并输出html页面
- 停止服务：ctrl + c

2.2 NPM环境

2.2.1 什么是NPM

NPM全称Node Package Manager，是Node.js包管理工具

是全球最大的模块生态系统，里面所有的模块都是开源免费的，也是Node.js的包管理工具，相当于前端的Maven

如果一个项目需要引用很多第三方的js文件，比如地图，报表等，文件杂而乱，自己去网上下载，到处是广告和病毒

那么，我们就想办法，把这些js文件统一放在一个仓库里，大家谁需要，谁就去仓库中拿过来，方便多了

npm就是这个仓库系统，如果你需要某个js文件，那就去远程仓库中下载，放在本地磁盘中，进而引用到我们的项目中

管理前端工程 so easy !

2.2.2 NPM工具的安装位置

node的环境在安装的过程中，npm工具就已经安装好了。

Node.js默认安装的npm包和工具的位置：Node.js目录\node_modules

在这个目录下你可以看见 npm目录，npm本身就是被NPM包管理器管理的一个工具，说明 Node.js已经集成了npm工具

```
#在命令提示符输入 npm -v 可查看当前npm版本  
npm -v
```

2.2.3 使用npm管理项目

2.2.3.1 项目初始化

全新创建一个目录，作为项目目录，使用dos命令进入此目录，输入命令

```
npm init  
  
# 接下来是一堆项目信息等待着你输入，如果使用默认值或你不知道怎么填写，则直接回车即可。  
  
# package name: 你的项目名字叫啥  
# version: 版本号  
# description: 对项目的描述  
# entry point: 项目的入口文件（一般你要用那个js文件作为node服务，就填写那个文件）  
# test command: 项目启动的时候要用什么命令来执行脚本文件（默认为node app.js）  
# git repository: 如果你要将项目上传到git中的话，那么就需要填写git的仓库地址（这里就不写地址了）  
# keywords: 项目关键字（我也不知道有啥用，所以我就不写了）  
# author: 作者的名字（也就是你叫啥名字）  
# license: 发行项目需要的证书（这里也就自己玩玩，就不写了）
```

最后会生成package.json文件，这个是包的配置文件，相当于maven的pom.xml

我们之后也可以根据需要进行修改。

上述初始化一个项目也太麻烦了，要那么多输入和回车。想简单点，一切都按照默认值初始化即可，ok，安排

```
npm init -y
```

2.2.4 修改npm镜像 和 存储地址

NPM官方的管理的包都是从 <http://npmjs.com> 下载的，但是这个网站在国内速度很慢。

这里推荐使用淘宝 NPM 镜像 <http://npm.taobao.org/>，淘宝 NPM 镜像是一个完整npmjs.com 镜像，同步频率目前为 10分钟一次，以保证尽量与官方服务同步。

设置镜像和存储地址：

```
#经过下面的配置，以后所有的 npm install 都会经过淘宝的镜像地址下载
npm config set registry https://registry.npm.taobao.org

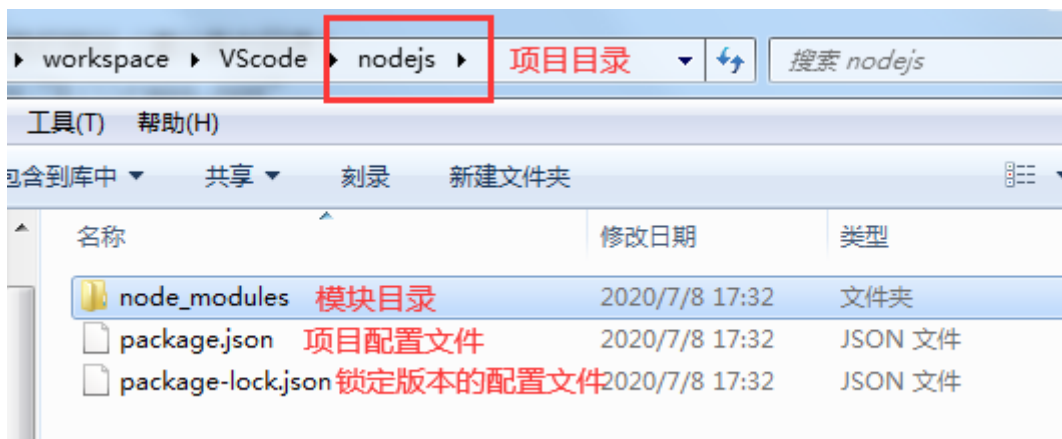
#设置npm下载包时保存在本地的地址（建议英文目录）
npm config set prefix "E:\\repo_npm"

#查看npm配置信息
npm config list
```

2.2.5 npm install命令的使用

```
npm install jquery
```

- 使用 npm install 安装依赖包的最新版
- 模块安装的位置：项目目录\node_modules
- 安装会自动在项目目录下添加 package-lock.json文件，这个文件帮助锁定安装包的版本
- 同时package.json 文件中，依赖包会被添加到dependencies节点下，类似maven中的<dependencies>



- jQuery版本有很多，上述命令下载的什么版本的？最新版
- 如果我的项目使用1.9.1版本进行开发的，通过npm安装的3.5.1版本太新，会导致项目失效，如何安装指定版本库？

```
npm install jquery@1.9.1
```

3. ES6基本语法

- ES标准中不包含 DOM 和 BOM的定义，只涵盖基本数据类型、关键字、语句、运算符、内建对象、内建函数等通用语法。
- 本部分只学习前端开发中ES6的必要知识，方便后面项目开发中对代码的理解。

3.1 let声明变量

- 与我们的JavaScript中var声明变量有什么区别？

1、作用域不同

```
{
    var a = 0; // var声明的变量是全局变量
    let b = 0; // let声明的变量是局部变量
}

console.log(a);
console.log(b); //b is not defined: b没有定义
```

2、声明次数不同

```
// var可以声明多次
// let只能声明一次
var m = 1;
var m = 2;
let n = 3;
let n = 4; //SyntaxError: Identifier 'n' has already been declared (语法错误: n已经声明过了)

console.log(m);
console.log(n);
```

3、声明与使用顺序不同

```
// var 声明的变量会全局存储
// let 声明的变量只能在执行后才存储

console.log( x ); //没有报错, 输出: undefined
var x = "苹果";

console.log(y); //y is not defined (y没有定义)
let y = "香蕉";
```

3.2 const声明常量

const 声明常量,为**只读变量**

1. 一旦声明之后, 其值是不允许改变的
2. 一但声明必须初始化, 否则会报错 `SyntaxError: Missing initializer in const declaration` (语法错误, 声明常量丢失了初始化)

```
const PI = 3.14;
PI = 3.1415; //Assignment to constant variable. (声明的是常量)

console.log( PI );
```

3.3 解构赋值

- 解构赋值是对赋值运算符的扩展
- 它是一种针对**数组**或者**对象**进行模式匹配, 然后对其中的变量进行赋值。
- 解构, 顾名思义, 就是将集合型数据进行分解, 拆分, 把里面的值逐一遍历获取
- 在代码书写上简洁且易读, 语义更加清晰明了; 也方便了复杂对象中数据字段获取。

3.3.1 数组解构

```
var arr = [1,2,3];

// 传统的js
let a = arr[0];
let b = arr[1];
let c = arr[2];
console.log(a,b,c);

//es6的解构
var [x,y,z] = arr;
console.log(x,y,z);
```

3.3.2 对象解构

```
var user = {
  username : "吕布",
  weapon:"方天画戟",
  horse:"赤兔马"
};

// 传统的js
let mingzi = user.username;
let wuqi = user.weapon;
let zuoji = user.horse;
console.log("姓名:"+mingzi+",武器:"+wuqi+",坐骑:"+zuoji);

//es6的解构
let {username,weapon,horse} = user; // 注意：解构的变量名必须是对象中的属性
console.log("姓名:"+username+",武器:"+weapon+",坐骑:"+horse);
```

3.4 模板字符串

- 模板字符串相当于加强版的字符串
- 用反引号 ```，除了作为普通字符串，还可以用来定义多行字符串
- 还可以在字符串中加入变量和表达式。

3.4.1 定义多行字符串

再见了，\n

```
let str = `hello,
你俩在哪呢？
心情不好，出来喝点啊！`;

console.log(str);
```

3.4.2 字符串插入变量和表达式

再见了，字符串的拼接用 +

```
let name = `吕布`;
let age = 24;

// 传统的拼接字符串
var info1 = "我叫:" + name + ",我今年" + age + "岁! ";
console.log(info1);

// es6的拼接字符串
var info2 = `我叫:${name},我明年${age+1}岁!`;
console.log(info2);
```

3.4.3 字符串中调用函数

还能这么玩

```
function test(){
    return "吃喝玩乐";
}

let str = `悲催的人生，从${test()}开始`;
console.log( str );
```

3.5 声明对象简写

定义对象的时候，可以用变量名作为属性名

```
let name = `吕布`;
let age = 28;

//传统
let user1 = {
    name : name,
    age : age
};
console.log(user1);

//es6新语法中的简写
let user2 = {name,age};
console.log(user2);
```

3.6 定义方法简写

```
// 传统
let user1 = {
    say : function(){
        console.log("大家好! ");
    }
};

user1.say();

//es6
let user2 = {
```



```
    say(){
      console.log("大家好啊！");
    }
  };

  user2.say();
```

3.7 对象拓展运算符

拓展运算符 {...} 将参数对象中所有可以遍历的属性拿出来，然后拷贝给新对象

3.7.1 拷贝对象(深拷贝)

科幻电影中的一滴血，就可以制作出一个完全一模一样的克隆人

```
let user1 = {
  name: "项羽",
  age: 34
};

let user2 = {...user1}; // 深拷贝（克隆）

console.log(user1);
console.log(user2);
```

3.7.2 合并对象

吞噬合并（两个对象合并成一个对象）

```
let user1 = {
  name: "项羽",
  age: 34
};

let user2 = {head: "诸葛亮"};

let user = {...user1, ...user2};

console.log( user );    { name: '项羽', age: 34, head: '诸葛亮' }
```

3.8 函数的默认参数

形参处已声明，但不传入实参会怎样？

那一个未定义的项为undefined，但是可以设置默认值

```
function test(name , age 默认参数 = 18){
  console.log(`我叫${name}, 我今年${age}岁`);
}

test("吕布", 33); //我叫吕布, 我今年33岁
test("貂蝉"); //我叫貂蝉, 我今年18岁
test("关羽", null); //我叫关羽, 我今年null岁
test("马超", ""); //我叫马超, 我今年岁
test("张飞", undefined); //我叫张飞, 我今年18岁
```

3.9 函数的不定参数

定义方法时，不确定有几个参数？

```
function test( ...arg ){
  console.log(`传入了${arg.length}个参数`);
  for(var i = 0 ; i < arg.length; i++){
    console.log(arg[i]);
  }
}

test(1);
test(1,2);
test(1,2,3,4,5,6);
test();
test("郭", "嘉", 28);
```

3.10 箭头函数

箭头函数提供了一种更加简洁的函数书写方式。基本语法是：参数 => 函数体

```
// 传统
var f1 = function(a){
  return a*10;
}

console.log( f1(10) );

// es6
var f2 = a=>a*10;
console.log( f2(20) );

// 当箭头函数一个参数时，（）可以省略
// 当箭头函数没有参数或者有多个参数，要用（）括起来
// 当箭头函数的函数体有多行语句，用{}括起来，表示代码块
// 当只有一条语句，并且需要返回时，可以省略{}，结果会自动返回

var f3 = (a,b) => {
  let sum = a+b;
  return sum;
}
```

```
console.log( f3(3,7) );

// 可以将f3进行简化
var f4 = (a,b) => a + b;
console.log( f3(11,22) );
```

3.11 Promise (了解)

- 用来解决回调函数的嵌套噩梦（后面学习ajax才能更好的理解）
- 我们回顾一下嵌套噩梦：
 - 你和我打架，我打不过你，我说：你在这等着，我找我哥来揍你！
 - “哥，我被老王打了，你帮我揍他！”，“行，我先把饭做完，你先去找刘备，把刘备找来，我和刘备再一起帮你报仇！”
 - “刘备，我被老王打了，我哥叫我来找你，一起帮我报仇！”，“没问题，兄弟，我去交电话费，你去找我二弟关羽，他打架厉害，把他叫来，我们一起更有把握！”
 - “关羽，我被人打了，我哥叫我找刘备，刘备让我来找你，一起帮我报仇！”，“没问题，兄弟，我有点事，你先去找我三弟张飞，把张飞找来，我这边就办完事了，我们一起帮你报仇去！”
 - “张飞，我让老王打了，谁谁谁，哎，反正关羽让我找你，你跟我走，一起帮我报仇去！”，“行啊，老铁，我一会从老丈人回来就去找你，你去把我二哥的儿子关平找来，正好打一架让这小子练练胆。找到了我们一起帮你报仇！”
 - ...
 - 满嘴的兄弟情深，一个拖一个，就是不想帮我报仇
- 使用promise解决掐架找人的噩梦
 - “哥，老王打我！”，“走，但老王厉害，我们去找刘备一起帮你报仇！”
 - “刘备，走着，我们一起揍老王去”，“没问题！”，“二弟，过来，有人欺负我们，去揍他！”
 - ... 找人过程中，都是立刻跟着走，没人拖！
 - 最终，我们108人一起找到了老王！！！！
- 找到一个人成功后，再继续找下一个人。逐渐形成了“队伍”
- 组建队伍的过程中，如果找某个人失败了，则“队伍”失败
- 其实有点类似“击鼓传花”的游戏，一个成功拿到花之后，才能传递给下一个人。依次类推！

```
setTimeout(()=>{
  console.log(1);
  setTimeout(()=>{
    console.log(2);
    setTimeout(()=>{
      console.log(3);
      setTimeout(()=>{
        console.log(4);
      },1000);
    },1000);
  },1000);
},1000);
```

使用promise

```
next = n =>
//Promise的构造函数接收一个参数，是函数，
```

```

    //并且传入两个参数：resolve(异步操作执行成功后的回调函数)，reject(异步操作执行失败后的回调函数)
    new Promise(function(resolve, reject) {
      setTimeout(function() {
        resolve(n);
      }, 1000);
    });

    next(1)
    .then(res => { // 成功
      console.log(res);
      return next(2); //在then方法中调用的next方法，一定要用return，否则不会通过resolve
    })
    .then(res => {
      console.log(res);
      return next(3);
    })
    .then(res => {
      console.log(res);
    })
    .catch(() => { //处理失败：catch方法的第二个参数是失败的回调
      console.log("出错啦！");
    });

```

3.12 模块化

- 如果在a.js文件中定义了5个方法，现在b.js文件中想使用a中的5个方法，怎么办？
- java语言的做法是import引入之后，就能使用了。es6的模块化，就是这个过程
- 将一个js文件声明成一个模块导出之后，另一个js文件才能引入这个模块
- 每一个模块只加载一次（是单例的），若再去加载同目录下同文件，直接从内存中读取。

3.12.1 传统的模块化

创建user.js文件

```

function addUser(name){
  return `保存${name}成功!`;
}

function removeUser(id){
  return `删除${id}号用户!`;
}

// 声明模块并导出
// module.exports={
//   save:addUser,
//   delete:removeUser
// }

// 声明模块导出的简写
module.exports={
  addUser,
  removeUser
}

```

test.js

```
let user = require("./user.js"); //引入user模块

console.log( user );

let result1 = user.addUser("吕布");
let result2 = user.removeUser(101);

console.log(result1);
console.log(result2);
```

3.12.2 ES6的模块化

user.js

```
let name = "老孙";
let age = 66;
let fn = function(){
    return `我是${name}!我今年${age}岁了!`;
}

// 声明模块并导出
export{
    name,
    age,
    fn
}
```

test.js

```
import {name,age,fn} from "./user.js"

console.log(name);
console.log(age);
console.log(fn);
```

运行test.js，报错：`SyntaxError: Unexpected token {`（语法错误，在标记{的位置）

原因是node.js并不支持es6的import语法，我们需要将es6转换降级为es5！

3.13 babel环境

babel是一个广泛使用的转码器，可以将ES6代码转为ES5代码，从而在现有的环境中执行。

这意味着，你可以现在就用 ES6 编写程序，而不用担心现有环境是否支持

3.13.1 安装babel客户端环境

创建新目录 lagou-babel，在终端中打开，运行命令：

```
npm install --global babel-cli
```

查看版本

```
babel --version
```

如果报错1：（win7系统中）

'babel' 不是内部或外部命令，也不是可运行的程序或批处理文件。（babel命令在当前系统中不被认可）

- 由于babel是通过npm安装，所以babel会默认安装到 E:\repo_npm
- 在 E:\repo_npm 这个目录下，进入dos命令，执行 babel --version，如果是成功的，说明babel已经安装成功
- 但是在vscode的终端中打不开，那么只可能是两个原因：
 - 环境变量没有配置
 - 系统环境变量中 path中加入 ;E:\repo_npm;
 - 在任意位置进入dos窗口，babel --version可以了，说明环境变量配置成功
 - vscode关掉，重新以“管理员身份运行”

如果报错2：（win10系统中）

```
PS G:\workspace-vscode\lagou-babel> babel --version
babel : 无法加载文件 G:\repo_npm\babel.ps1，因为在此系统上禁止运行脚本。有关详细信息，请参阅 http://go.microsoft.com/fwlink/?LinkID=135170 中的 about_Execution_Policies。
所在位置 行:1 字符: 1
+ babel --version
+ ~~~~~
+ CategoryInfo          : SecurityError: (:) [], PSSecurityException
+ FullyQualifiedErrorId : UnauthorizedAccess
```

windows10默认禁止运行有危险的脚本，修改一下系统策略就好了

开始菜单-> Windows PowerShell (切记要以管理员身份运行)，输入代码

```
set-ExecutionPolicy RemoteSigned
```

```
PS C:\WINDOWS\system32> set-ExecutionPolicy RemoteSigned

执行策略更改
执行策略可帮助你防止执行不信任的脚本。更改执行策略可能会产生安全风险，如 http://go.microsoft.com/fwlink/?LinkID=135170
中的 about_Execution_Policies 帮助主题所述。是否要更改执行策略？
[Y] 是(Y) [A] 全是(A) [N] 否(N) [L] 全否(L) [S] 暂停(S) [?] 帮助 (默认值为“N”) y
PS C:\WINDOWS\system32>
```

现在，就可以看版本号了。

3.13.2 安装转码器

1. 创建lagou-babel专属目录，在其中初始化项目

```
npm init -y
```

2. 创建babel配置文件 .babelrc，并输入代码配置：

```
{
  "presets": ["es2015"],
  "plugins": []
}
```

3. 安装转码器

```
npm install --save-dev babel-preset-es2015
```

4. 转码

创建dist目录，用来存放转码后的文件

```
babel user.js --out-file .\dist\user.js  
或  
babel user.js -o .\dist\user.js
```

5. 运行转码后的文件

```
node .\dist\test.js
```

3.14 ES6模块化的另一种写法

3.14.1 as的用法

user.js：如果你不想暴露模块当中的变量名字，可以通过as来进行操作：

```
let name = "老孙";  
let age = 66;  
let fn = function(){  
  return `我是${name}!我今年${age}岁了!`;  
}  
  
// 声明模块并导出  
export{  
  name as a,  
  age as b,  
  fn as c  
}
```

test.js

```
import {a,b,c} from "./user.js";  
  
console.log(a);  
console.log(b);  
console.log( c() );
```

也可以接收整个模块

test.js

```
import * as info from "./user.js"; // 通过*来批量接收，as来指定接收的名字  
  
console.log(info.a);  
console.log(into.b);  
console.log( into.c() );
```

3.14.2 默认导出

可以将所有需要导出的变量放入一个对象中，然后通过default export进行导出

```

/*****导出*****/
export default{
  name:"老孙",
  eat(){
    return "吃点啥! ";
  }
}

/*****导入*****/
import p from "./person.js";
console.log( p.name, p.eat() );

```

3.14.3 重命名export和import

如果导入的多个文件中，变量名字相同，即会产生命名冲突的问题，为了解决该问题，ES6为提供了重命名的方法，当你在导入名称时可以这样做：

```

/*****student1.js*****/
export let name = "我是来自student1.js";

/*****student2.js*****/
export let name = "我是来自student2.js";

/*****test_student.js*****/
import {name as name1} from './student1.js';
import {name as name2} from './student2.js';

console.log( name1 ); // 我是来自student1.js
console.log( name2 ); // 我是来自student2.js

```