引入： require / import

导出：export / module.exports / exports

Nodejs 不支持 import 和 export，因为在我们一贯的node模块中，我们采用的是CommonJS规范，使用require引入模块，使用module.exports导出接口。

es6 兼容以上所有语法，当然需要 webpack + babel 来支撑

在ES6出现之前，js是没有统一的模块体系。

服务器端使用CommonJS规范（node.js），而浏览器端又有AMD(require.js)和CMD两种规范

webpack的思想就是一切皆模块，官方推荐使用commonJS规范，这使得我们浏览器端也可以使用commonJS的模块化写法。如webpack的配资文件即用commonJS的模块化写法

commonJS的模块化写法：

a文件通过require引入：

```
var say = require('./util');
say();
```

b 文件通过module.exports导出

```
module.exports = function say() {
    console.log('hello world');
}
```

webpack默认不支持转码es6，但是`import` `export`这两个语法却单独支持。所以我们可以改写前面的模块化写法

a 文件引入：

```
import say from './util';
say();
```

b文件导出：

```
export default function say() {
    console.log('hello world ');
}
```

或者：

a 文件引入：

```
import say from './util';
say();
```

b文件导出：

```
 function say() {
    console.log('hello world ');
}
export default say;
```

****

下面比较一下export default和export 输出。

```
// 第一组
export default function car() { // 输出
  // ...
}
import car from 'car'; // 输入

// 第二组
export function car2() { // 输出
```

```
  // ...
};
import {car2} from 'car2'; // 输入
```

可以看到第一组是使用export default，import语句不需要使用大括号；第二组使用export，对应的import语句需要使用大括号，一个模块只能有一个默认输出，所以export default只能使用一次。

****

下面比较一下module.exports和exports 输出。

根据CommonJS规范，每个文件就是一个模块，有自己的作用域。在一个文件里面定义的变量、函数、类，都是私有的，对其他文件不可见。

CommonJS规范规定，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（即module.exports）是对外的接口。

- module.exports 初始值为一个空对象 {}
- exports 是指向的 module.exports 的引用
- require() 返回的是 module.exports 而不是 exports

如果执行 others = module.exports之后，即 module.exports 指向新的对象时，exports 断开了与 module.exports 的引用，那么通过 exports = module.exports 让 exports 重新指向 module.exports 即可。