export、import与ECMAScript 6的模块机制

2016年03月21日 3785 声明

在 ES6 之前，JavaScript中并没有在语言标准中提供模块定义规范，这对开发规模较大、较复杂的应用造成一定的影响。而在非语言层面，一些开源社区制定了模块定义规范，主要有 CommonJS (http://itbilu.com/other/relate/Nyc6AnDV.html)和 AMD 两种。在 ES6 中，定义了 import 和 export 两种语法声明，从而在语言面实现了模块功能。

1. ECMAScript 6 的模块化

1.1 ECMAScript 6 之前的模块化

JavaScript最初被设计时并不是用来大型应用的，所以在其设计中也并没有模块化标准。随着其应用越来越广泛，一些开源社区和开发者提出了一些模块标准，如：CommoneJS 模块化、异步模块定义（AMD）等。这些规范在提出后得到了广泛关注，并逐步被应用到了一些企业级的大型应用。

CommoneJS模块化：该标准最成功的应用是 Node.js（ Node.js 在 CommoneJS 的基础上就进行了一定的扩展）。其主要特点是语法简洁，模块使用同步加载机制，主要使用在服务器端。

异步模**块定义（AMD）**：该标准的典型应该是 RequireJS 。其主要特点是模块使用异步加载，主要使用在浏览器端。

1.2 ES6 模块机制

ECMAScript 6 基于 export 和 import，定义了模块的导出和导入规范，在语言标准层面实现了模块机制。该标准的目标是创建一种能够兼容 CommoneJS 和 AMD 两标准的规范，即可以像 CommoneJS 一样语法简洁、使用单一的接口且支持循环依赖，又可以像 AMD 支持异步加载和可配置的模块加载。

ES6 定义的模块标准由两部分组成：

- 声明语法（定义引入与导出）
- 编程式加载接口（API）：用于配置如何加载模块和按条件加载模块

ES6 的模块机制具有以下特点：

- 简洁的语法。语法将比 CommoneJS 更简单，只使用 export 和 import 实现模块的导出和导入
  - 使用 export 关键字定义导出对象，这个关键字可以无限次使用
  - 使用 import 关键字引入导入对象，这个关键字可导入任意数量的模块
- 模块结构可以做静态分析。这使得在编译时就能确定模块的依赖关系，以及输入和输出的变量
- 模块支持异步加载
- 为加载模块提供编程支持，可以按需加载
- 比 CommonJS 更优秀的循环依赖处理

1.3 支持状况

ES6 为JavaScript带来了模块机制，但 ES6 的模块机制在当前所有的浏览器及Node.js中均不受支持。但我们可以通过一些编译器来对 ES6 语法进行转换，从而利用这些新特性给我们项目带来便利：

- Babel (http://itbilu.com/nodejs/npm/41K9OSwpe.html)－ Babel 是一个 ES6 语法转换为 ES5 语法的转换器，其支持对 ES6 模块语法的转换，包括：异步加载、状态隔离、顶级命名空间隔离等
- es6-module-transpiler (https://github.com/umdjs/es6-module-transpiler) －将 ES6 模块编译为 AMD 规范或者 CommonJS 规范的模块
- ES6 module loader (https://github.com/ModuleLoader/es6-module-loader)－能支持动态加载 ES6 风格的模块
- Traceur (https://github.com/google/traceur-compiler)－Google开发的JS转换编译器，目的在于支持更多的JavaScript特性包括 ES6 模块

2. export 与模块导出

2.1 export 概述

export 语法声明用于导出函数、对象、指定文件（或模块）的原始值。
export 有两种模块导出方式：命名式导出（名称导出）和定义式导出（默认导出），命名式导出每个模块可以多个，而默认导出每个模块仅一个。

export 可能会有以下几种形式的导出语法：

```
export { name1, name2, …, nameN };
export { variable1 as name1, variable2 as name2, …, nameN
export let name1, name2, …, nameN; // 也可以是 var
export let name1 = …, name2 = …, …, nameN; // 也可以是 var,

export default expression;
export default function (…) { … } // 也可以是 class, functi
export default function name1(…) { … } // 也可以是 class, f
export { name1 as default, … };

export * from …;
export { name1, name2, …, nameN } from …;
export { import1 as name1, import2 as name2, …, nameN } fr
```

- name1… nameN －导出的“标识符”。导出后，可以通过这个“标识符”在另一个模块中使用 import 引用
- default －设置模块的默认导出。设置后 import 不通过“标识符”而直接引用默认导入
- * －继承模块并导出继承模块所有的方法和属性
- as －重命名导出“标识符”
- from －从已经存在的模块、脚本文件…导出

2.2 模块导出相关介绍

命名式**导**出

模块可以通过 export 前缀关键词声明导出对象，导出对象可以是多个。这些导出对象用名称进行区分，称之为 命名式导出 。

```
export { myFunction }; // 导出一个已定义的函数
export const foo = Math.sqrt(2); // 导出一个常量
```

我们可以使用 * 和 from 关键字来实现的模块的继承：

```
export * from 'article';
```

模块导出时，可以指定模块的导出成员。导出成员可以认为是类中的公有对象，而非导出成员可以认为是类中的私有对象：

```
var name = 'IT笔录';
var domain = 'http://itbilu.com';

export {name, domain};
```

模块导出时，我们可以使用 as 关键字对导出成员进行重命名：

```
var name = 'IT笔录';
var domain = 'http://itbilu.com';

export {name as siteName, domain};
```

默**认导**出

默认导出 也被称做 定义式导出 。命名式导出可以导出多个值，但在在 import 引用时，也要使用相同的名称来引用相应的值。而默认导出每个导出只有一个单一值，这个输出可以是一个函数、类或其它类型的值，这样在模块 import 导入时也会很容易引用。

```
export default function() {}; // 可以导出一个函数
export default class(){}; // 也可以出一个类
```

命名式**导**出与默**认导**出

默认导出可以理解为另一种形式的命名导出，默认导出可以认为是使用了 default 名称的命名导出。

下面两种导出方式是等价的：

```
const D = 123;

export default D;
export { D as default };
```

2.3 export 使用示例

使用名称导出一个模块时:

```
// "my-module.js" 模块
export function cube(x) {
  return x * x * x;
}
const foo = Math.PI + Math.SQRT2;
export { foo };
```

在另一个模块(脚本文件)中,我们可以像下面这样引用:

```
import { cube, foo } from 'my-module';
console.log(cube(3)); // 27
console.log(foo);    // 4.555806215962888
```

使用默认导出一个模块时:

```
// "my-module.js"模块
export default function (x) {
  return x * x * x;
}
```

在另一个模块(脚本文件)中,我们可以像下面这样引用,相对名称导出来说使用更为简单:

```
// 引用 "my-module.js"模块
import cube from 'my-module';
console.log(cube(3)); // 27
```

3. import

3.1 import 概述

import 语法声明用于从已导出的模块、脚本中导入函数、对象、指定文件(或模块)的原始值。import 模块导入与 export 模块导出功能相对应,也存在两种模块导入方式:命名式导入(名称导入)和定义式导入(默认导入)。

import 可能会有以下几种形式的导入语法:

```
import defaultMember from "module-name";
import * as name from "module-name";
import { member } from "module-name";
import { member as alias } from "module-name";
import { member1 , member2 } from "module-name";
import { member1 , member2 as alias2 , [...] } from "modu
import defaultMember, { member [ , [...] ] } from "module
import defaultMember, * as name from "module-name";
import "module-name";
```

- `name` －从将要导入模块中收到的导出值的名称
- `member, memberN` －从导出模块，导入指定名称的多个成员
- `defaultMember` －从导出模块，导入默认导出成员
- `alias, aliasN` －别名，对指定导入成员进行的重命名
- `module-name` －要导入的模块。是一个文件名
- `as` －重命名导入成员名称（“标识符”）
- `from` －从已经存在的模块、脚本文件等导入

3.2 模块导入相关介绍

命名式**导**入

我们可以通过指定名称，就是将这些成员插入到当作用域中。导出时，可以导入单个成员或多个成员：

```
import {myMember} from "my-module";
import {foo, bar} from "my-module";
```

通过 * 符号，我们可以导入模块中的全部属性和方法。当导入模块全部导出内容时，就是将导出模块（'my-module.js'）所有的导出绑定内容，插入到当前模块（'myModule'）的作用域中：

```
import * as myModule from "my-module";
```

导入模块对象时，也可以使用 as 对导入成员重命名，以方便在当前模块内使用：

```
import {reallyReallyLongModuleMemberName as shortName} fr
```

导入多个成员时，同样可以使用别名：

```
import {reallyReallyLongModuleMemberName as shortName, an
```

导入一个模块，但不进行任何绑定：

```
import "my-module";
```

默**认导**入

在模块导出时，可能会存在默认导出。同样的，在导入时可以使用 `import` 指令导出这些默认值。

直接导入默认值：

```
import myDefault from "my-module";
```

也可以在命名空间导入和名称导入中，同时使用默认导入：

```
import myDefault, * as myModule from "my-module"; // myMo
```

或

```
import myDefault, {foo, bar} from "my-module"; // 指定成员导
```

3.3 import 使用示例

导入一个二级文件，用于在当前模块中进行AJAX JSON (http://itbilu.com/javascript/js/EkIzWrt7.html)请求：

```
// --file.js--
function getJSON(url, callback) {
  let xhr = new XMLHttpRequest();
  xhr.onload = function () {
    callback(this.responseText)
  };
  xhr.open("GET", url, true);
  xhr.send();
}

export function getUsefulContents(url, callback) {
  getJSON(url, data => callback(JSON.parse(data)));
}

// --main.js--
import { getUsefulContents } from "file";
getUsefulContents("http://itbilu.com", data => {
  doSomethingUseful(data);
});
```

下一篇：JavaScript BOM对象－BOM介绍 (/javascript/js/4k9JcnZRl.html)

上一篇：JavaScript HTML DOM节点类型之Attr类型 (/javascript/js/Ekt3KKGog.html)
