《前端高手进阶》

朱德龙 前中兴软创主任工程师

— 拉勾教育出品 —

第10讲：怎么复用你的代码？

前言

作为前端工程师的你

或许早已习惯了在编写浏览器组件时使用 import 和 from 来管理代码模块

在编写 Node.js 服务时通过 require 和 module.exports 来复用代码

但 JavaScript 模块化之路充满了坎坷

这一课时就带你由近及远

看看 JavaScript 模块发展史上那些著名的**模块规范与实现**

ES6 模块

定义和引用

- ES6 模块**强制自动采用严格模式**

  所以说不管有没有“user strict”声明都是一样的

  换言之，编写代码的时候不必再刻意声明了
- 虽然大部分主流浏览器支持 ES6 模块

  但是和引入普通 JS 的方式略有不同

  需要在对应 script 标签中将属性 **type 值设置为“module”**才能被正确地解析为 ES6 模块
- 在 Node.js 下使用 ES6 模块则需要将文件名**后缀改为“.mjs”**

  用来和 Node.js 默认使用的 CommonJS 规范模块作区分

ES6 模块

特性

- **值引用**

  值引用是指 export 语句输出的接口

  与其对应的值是动态绑定关系

  即通过该接口，可以取到模块内部实时的值

  可以简单地理解为**变量浅拷贝**

- **静态声明**

ES6 模块


```
// a.js
export var a = '';
setTimeout(() => a = 'a', 500);
// b.js
import { a } from './a.js'
console.log(a) // ''
setTimeout(() => console.log(a), 1000) // 'a'
```

ES6 模块


```
// 必须首部声明
leta a = 1
import { app } from './app';
//不允许使用变量或表达式
 import { 'a' + 'p' + 'p' } from './app';
//不允许被嵌入语句逻辑
if (moduleName === 'app') {
 import { init } from './app';
} else {
 import { init } from './bpp';
}
```

延申 1：import 的动态模块提案

在 ES2020 规范提案中，希望通过 import() 函数来支持动态引入模块

```
import(`./section-modules/${link.dataset.entryModule}.js`)
.then(module => {
  module.loadPageInto(main);
})
.catch(err => {
  main.textContent = err.message;
});
```

延申 1：import 的动态模块提案

- **违反首部声明要求**

  意味着可以在代码运行时按需加载模块

  这个特性就可以用于首屏优化，根据路由和组件只加载依赖的模块

- **违反变量或表达式要求**

  意味着可以根据参数动态加载模块

- **违反嵌入语句逻辑规则**

  可想象空间更大

  比如可以通过 Promise.race 方式同时加载多个模块

  选择加载速度最优模块来使用，从而提升性能

CommonJS

定义和引用

CommonJS 规定每个文件就是一个**模块**，有**独立**的作用域

每个模块内部，都有一个 module 对象，代表当前模块

通过它来导出 API

CommonJS

- id 模块的识别符，通常是带有绝对路径的模块文件名
- filename 模块的文件名，带有绝对路径
- loaded 返回一个布尔值，表示模块是否已经完成加载
- parent 返回一个对象，表示调用该模块的模块
- children 返回一个数组，表示该模块要用到的其他模块
- exports 表示模块对外输出的值

CommonJS

特性

- **值拷贝**

  一旦输出一个值

  模块内部的变化就影响不到这个值

  可以简单地理解为**变量深拷贝**

- **动态声明**

CommonJS


```
// a.js
var a = '';
setTimeout(() => a = 'a', 500);
module.exports = a
// b.js
var a = require('./a.js')
console.log(a) // ''
setTimeout(() => console.log(a), 1000) // ''
```

AMD

定义和引用

AMD 规范只定义了一个全局函数 **define**

```
define(id?, dependencies?, factory);
```

AMD

定义和引用

```
define("alpha", ["require", "exports", "beta"], function (require, exports, beta) {
    exports.verb = function() {
        return beta.verb();
    }
});
```

AMD

特性

- **异步加载**　指同时并发加载所依赖的模块
  当所有依赖模块都加载完成之后
  再执行当前模块的回调函数

AMD

```
var requirejs, require, define;
(function (global, setTimeout) {
 ...
 define = function (name, deps, callback) {
  ...
  if (context) {
   context.defQueue.push([name, deps, callback]);
   context.defQueueMap[name] = true;
  } else {
   globalDefQueue.push([name, deps, callback]);
  }
 };
```

AMD

```
...
req.load = function (context, moduleName, url) {
 ...
 if (isBrowser) {
  node = req.createNode(config, moduleName, url);
  ...
  if (baseElement) {
   head.insertBefore(node, baseElement)
  } else {
   head.appendChild(node)
  }
  currentlyAddingScript = null;
```

AMD

```
    if (baseElement) {
      head.insertBefore(node, baseElement);
    } else {
      head.appendChild(node)
    }
    currentlyAddingScript = null;
    return node
  }
 };
 ...
}(this, (typeof setTimeout === 'undefined' ? undefined :
setTimeout)));
```

CMD

定义和引用

```
define(factory);
```

CMD

定义和引用

```
define(function(require, exports, module) {
    //...
});
```

CMD

定义和引用

```
define(function(require, exports, module) {
  var add = require('math').add;
  exports.increment = function(val) {
    return add(val, 1);
  };
  module.id == "increment";
});
```

CMD

特性

- **懒加载**

不需要在定义模块的时候声明依赖

可以在模块执行时动态加载依赖

CMD 同时支持同步加载模块和异步加载模块

它整合了 CommonJS 和 AMD 规范的特点

遵循 CMD 规范的代表开源项目是 sea.js

它的实现和 requirejs 没有本质差别

UMD

UMD（Universal Module Definition，统一模块定义）其实并不是模块管理规范

而是带有前后端同构思想的**模块封装工具**

通过 UMD 可以在合适的环境选择对应的模块规范

UMD

- 先判断是否支持 Node.js 模块格式（exports 是否存在）

  存在则使用 Node.js 模块格式
- 再判断是否支持 AMD（define 是否存在）

  存在则使用 AMD 方式加载模块
- 若前两个都不存在，则将模块公开到全局（Window 或 Global）

UMD

```
(function (root, factory) {
  if (typeof define === 'function' && define.amd) {
    define([], factory);
  } else if (typeof exports === 'object') {
module.exports,
    module.exports = factory();
  } else {
    root.returnExports = factory();
  }
}(this, function () {
  //。。。
  return {};
}));
```

延申2：ES5 标准下如何编写模块

模块的核心就是创建独立的作用域

```
var mod = {
  a: '',
  f: function() {
    ...
  },
}
```

延申2：ES5 标准下如何编写模块


```
var mod = (function(w){
  function f() {
    ...
  }
  var a = ''
  ...
  return {
        f,
        a
  };
})(window);
```

延申2：ES5 标准下如何编写模块


```
// index.js
import { text, write } from './m'
write(`<h1>${text} ${text2}</h1>`)
// m.js
const write = content => document.write(content)
var text = 'hello'
export { text, write }
```

延申2：ES5 标准下如何编写模块


```
// bundle.js
(function(modules) {
  ...
})({
  "./index.js": (function(module,
__webpack_exports__, __webpack_require__) {
    ...
  },
  "./m.js": (function(module, __webpack_exports__,
__webpack_require__) {
    ...
  }
});
```

延申2：ES5 标准下如何编写模块


```
function (modules) {
 var installedModules = {};
 function __webpack_require__(moduleId) {
  if (installedModules[moduleId]) {
   return installedModules[moduleId].exports;
  }
  var module = installedModules[moduleId] = {
   i: moduleId,
   l: false,
   exports: {}
```

延申2：ES5 标准下如何编写模块


```
    exports: {}
  };
  modules[moduleId].call(module.exports, module, module.exports,
__webpack_require__);
  module.l = true;
  return module.exports;
}
...
return __webpack_require__(__webpack_require__.s = "./index.js");
}
```

延申2：ES5 标准下如何编写模块


```
// index.js 中引入 m.js 模块
var _m__WEBPACK_IMPORTED_MODULE_0__ = __webpack_require__( /*! ./m */ "./m.js");

// m.js 中导出字符串 text 和函数 write
__webpack_require__.d(__webpack_exports__, "text", function () {
  return text;
});
__webpack_require__.d(__webpack_exports__, "write", function () {
  return write;
});
const write = content => document.write(content)
var text = 'hello'
```

总结

介绍了 JavaScript **模块化规范**

包括原生规范 ES6 模块、Node.js 采用的 CommonJS

以及开源社区早期为浏览器提供的规范 AMD

具有 CommonJS 特性和 AMD 特性的 CMD

让 CommonJS 和 AMD 模块跨端运行的 UMD

总结

如果要实现一个支持动态加载的 import() 函数该怎么做呢？

Next：第11讲《为什么说 JavaScript 不适合大型项目？》

拉勾教育

—互联网人实战大学—

下载「拉勾教育App」
获取更多内容

L / A / G / O / U