目前，ESM 有三种方式的实现：

- 浏览器
- webpack 以及类似的构建工具
- Node（未完成，但可能在年底作为一个实验功能）

浏览器

截至到 2017 年 5 月，所有主流浏览器都开始做了 ESM 的实现工作。不过，大部分仍处于在实验性质，如果你试图在 script 标签使用 import 或者 export 语句，会抛出一个 SyntaxError。 可以通过如下资源解决。

```
<script src="https://google.github.io/traceur-
compiler/bin/traceur.js"></script>
<script src="https://google.github.io/traceur-
compiler/bin/BrowserSystem.js"></script>
<script src="https://google.github.io/traceur-
compiler/src/bootstrap.js"></script>
<script type="module" src="index.js"></script>
```

目前在实验的写法为： 需要在 script 标签添加 type="module" 属性，如下所示：

```
<script type="module" src="main.js"></script>
```

在一个模块中，当前只能使用有效的 URL 作为模块标识符。模块标识符是用于 require 或 import 其他模块的字符串。为了确保未来兼容 CJS 模块标识符，“bare”导入标志符（如 import "lodash"）现在还不支持。模块标识符必须是绝对 URL 或者是以 /， ./, ../ 开头：

webpack

最终会生成一段 script，通常是在一定程度上模拟 CJS 和 ESM 行为的模块运行时。 例

```
// a.js
export let number = 42;
export function incr() {
    number++;
```

```
}
// b.js
import { number } from "./a";

console.log(number);
```

webpack 使用函数包装器封装模块范围和对象引用来模拟 ESM 实时绑定,每次编译，将包括一个模块运行时区，负责引导和缓存模块。此外，将模块标识转换为数字模块ID。这样可以减少打包的大小和引导时间。 看看编译输出

```
(function(modules) {
// This is the module runtime.
// It's only included once per compilation.
// Other chunks share the same runtime.
var installedModules = {};
// The require function
function __webpack_require__(moduleId) {
        ...
    }
    ...
// Load entry module and return exports
return __webpack_require__(__webpack_require__.s = 1);
})
([ // An array that maps module ids to functions
// a.js as module id 0
function (module, __webpack_exports__, __webpack_require__) {
        "use strict";
Object.defineProperty(__webpack_exports__, "a", {
configurable: false,
enumerable: true,
get: () => number
        });

let number = 42;

function incr() {
            number++;
        }
    },
// test.js as module id 1
function (module, __webpack_exports__, __webpack_require__) {
        "use strict";
var __WEBPACK_IMPORTED_MODULE_0__a__ = __webpack_require__(0);

// Object reference as "live binding"
console.log(__WEBPACK_IMPORTED_MODULE_0__a__["a" /* number */]);
```

```
    }
]);
```

webpack 在 exports 对象上将所有 export 语句替换成 Object.defineProperty，并使用属性访问器替换对引入值的所有引用。还要注意每个 ESM 开始时的 "use strict" 指令，这是由 webpack 自动添加，在 ESM 中必须是严格模式。

Node

node 在执行esm时出现了问题，因为还需要支持cjs，一个动态一个静态，具体权衡和解决方法正在探讨，等进一步结果