Webpack Bundle.js

Zz

Webpack 哲学三问：

- 这是 Webpack 的问题吗?
- 我要怎么解决这个问题?
- 咦我是怎么解决的?

Webpack打包出来的文件

配置文件

```
// webpack.config.js
const path = require('path');
const webpack = require('webpack');

module.exports = {
    entry: {
        bundle1: path.resolve(__dirname, 'src/index1.js'),
        bundle2: path.resolve(__dirname, 'src/index2.js')
    },
    output: {
        path: path.resolve(__dirname, 'dist'),
        filename: '[name].js'
    },
    plugins: [
        new webpack.optimize.CommonsChunkPlugin({
            name: 'manifest'
        })
    ]
};
```

待打包文件（module）

同步加载

```js
// index1.js
const test1 = require('./test1');
const test3 = require('./test3')
console.log(test1);
console.log(test3);

// test1.js
const str = 'test1 is loaded';
module.exports = str;

// test3.js
const str = 'test3 is loaded';
module.exports = str;
```

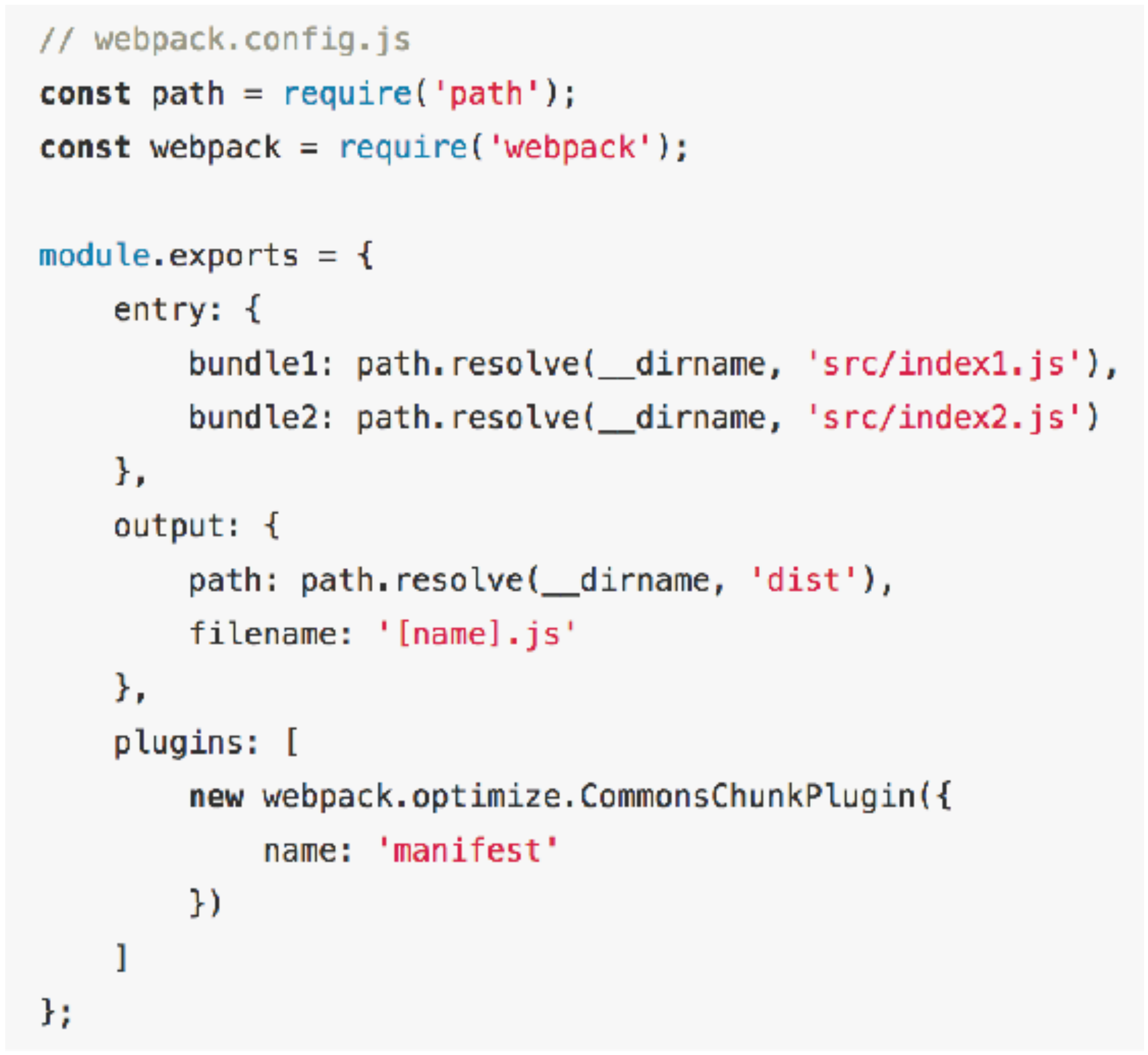
异步加载

```js
// index2.js
setTimeout(function() {
    require.ensure([], function() {
        const test2 = require('./test2');
        console.log(test2);
    });
}, 5000);

// test2.js

const str = 'test2 is async loaded';
module.exports = str;
```

Module 和 Chunk

- **Module** 其实就是打包前，import 或者 require 的js文件。
- **Chunk** 是打包后的文件，需要注意 **一个 chunk 可能包含若干 module**。

打包的结果文件



```
// webpack.config.js
const path = require('path');
const webpack = require('webpack');

module.exports = {
    entry: {
        bundle1: path.resolve(__dirname, 'src/index1.js'),
        bundle2: path.resolve(__dirname, 'src/index2.js')
    },
    output: {
        path: path.resolve(__dirname, 'dist'),
        filename: '[name].js'
    },
    plugins: [
        new webpack.optimize.CommonsChunkPlugin({
            name: 'manifest'
        })
    ]
};
```

bundle1.js、bundle2.js、0.js和 **manifest.js**

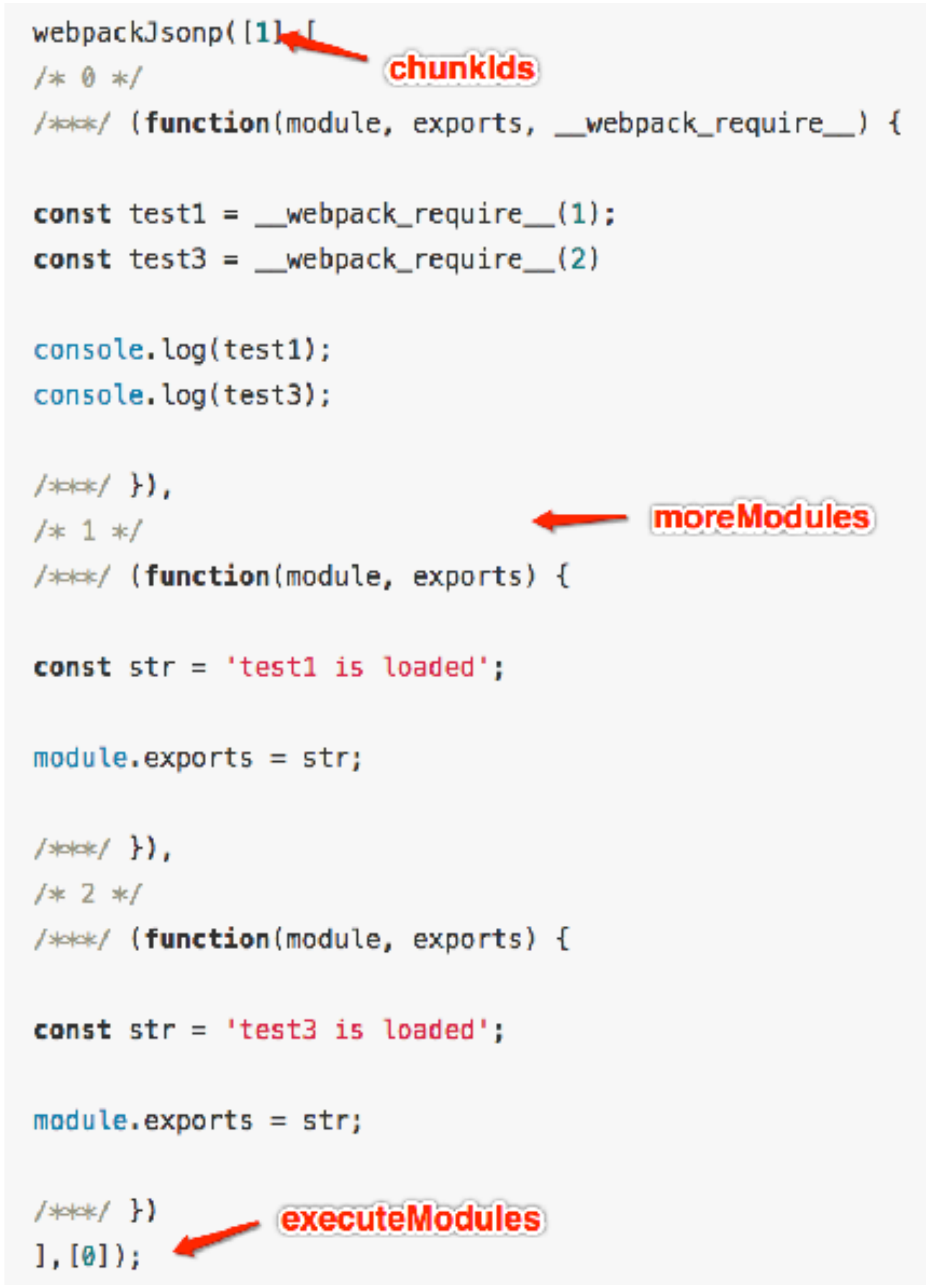
Manifest.js

Manifest.js 先运行注入了一些方法

下面三个是最核心的方法：

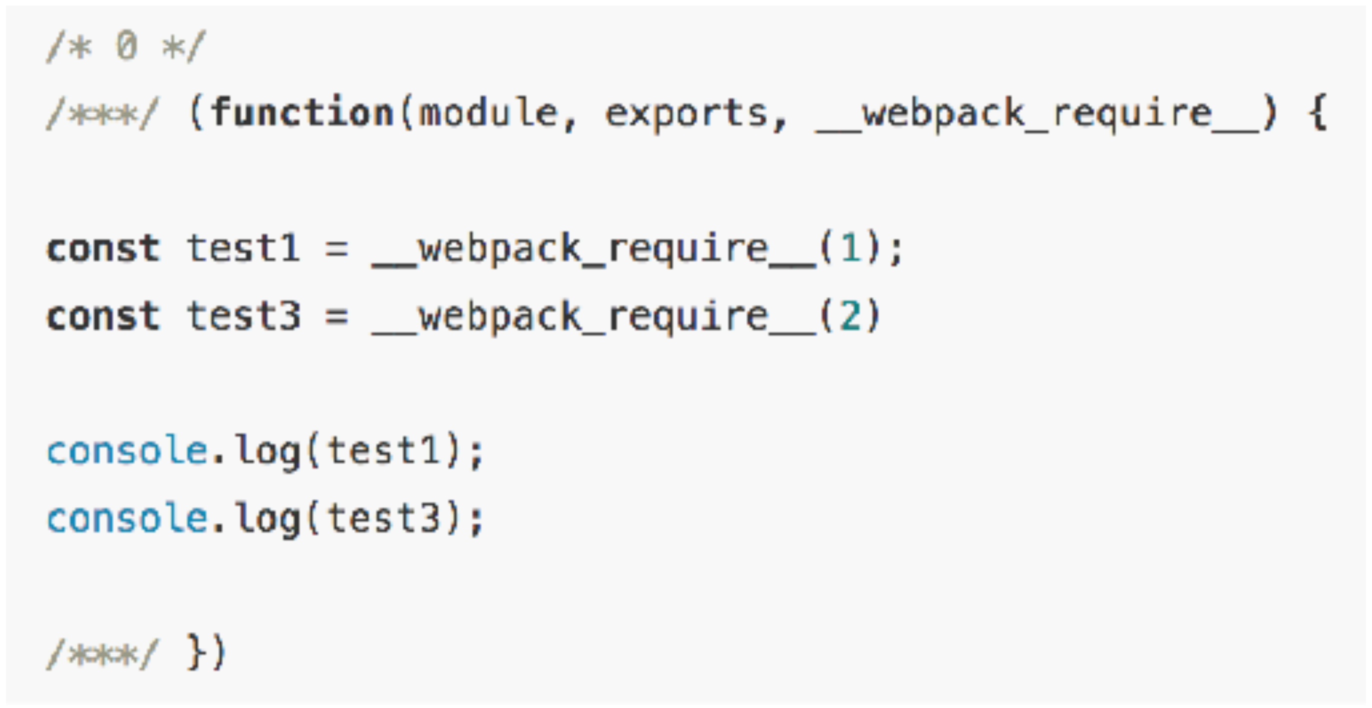
- webpackJsonp（chunkIds, moreModules, executeModules）
- webpack_require（加载module）
- webpack_require.e（加载chunk & jsonp -> webpackJsonp）

Bundle.js

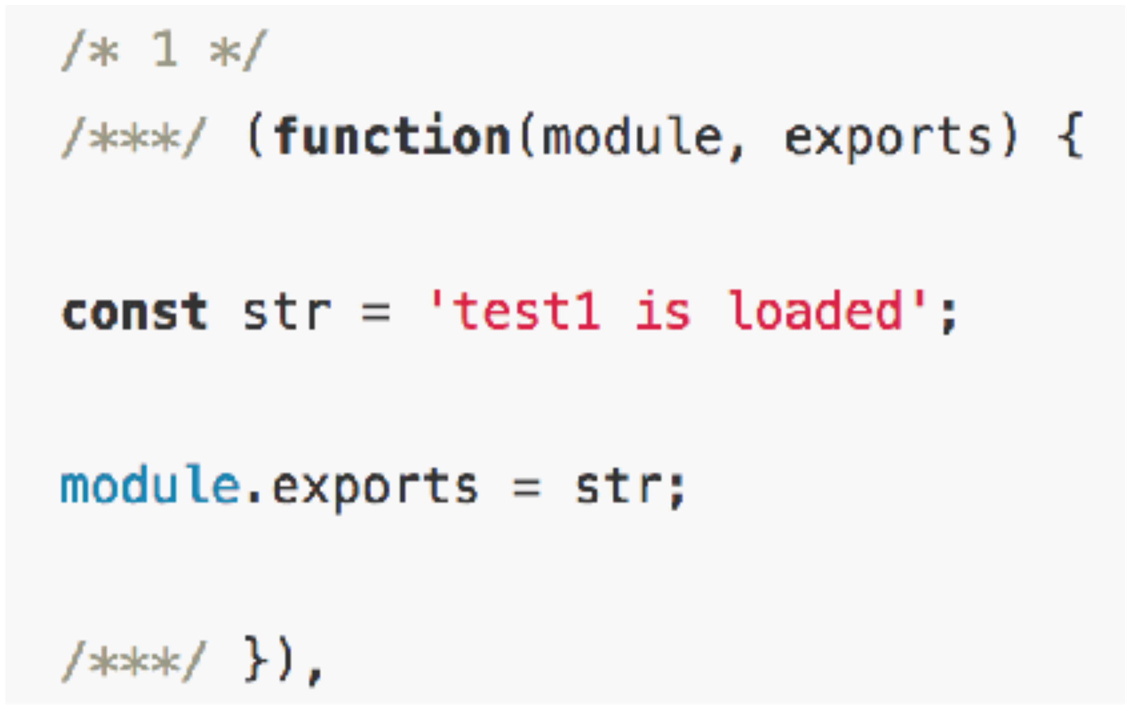


```
webpackJsonp([1],[
/* 0 */
/***/ (function(module, exports, __webpack_require__) {

const test1 = __webpack_require__(1);
const test3 = __webpack_require__(2)

console.log(test1);
console.log(test3);

/***/ }),
/* 1 */
/***/ (function(module, exports) {

const str = 'test1 is loaded';

module.exports = str;

/***/ }),
/* 2 */
/***/ (function(module, exports) {

const str = 'test3 is loaded';

module.exports = str;

/***/ })
],[0]);
```

moreModules



```
/* 0 */
/***/ (function(module, exports, __webpack_require__) {

const test1 = __webpack_require__(1);
const test3 = __webpack_require__(2)

console.log(test1);
console.log(test3);

/***/ })
```

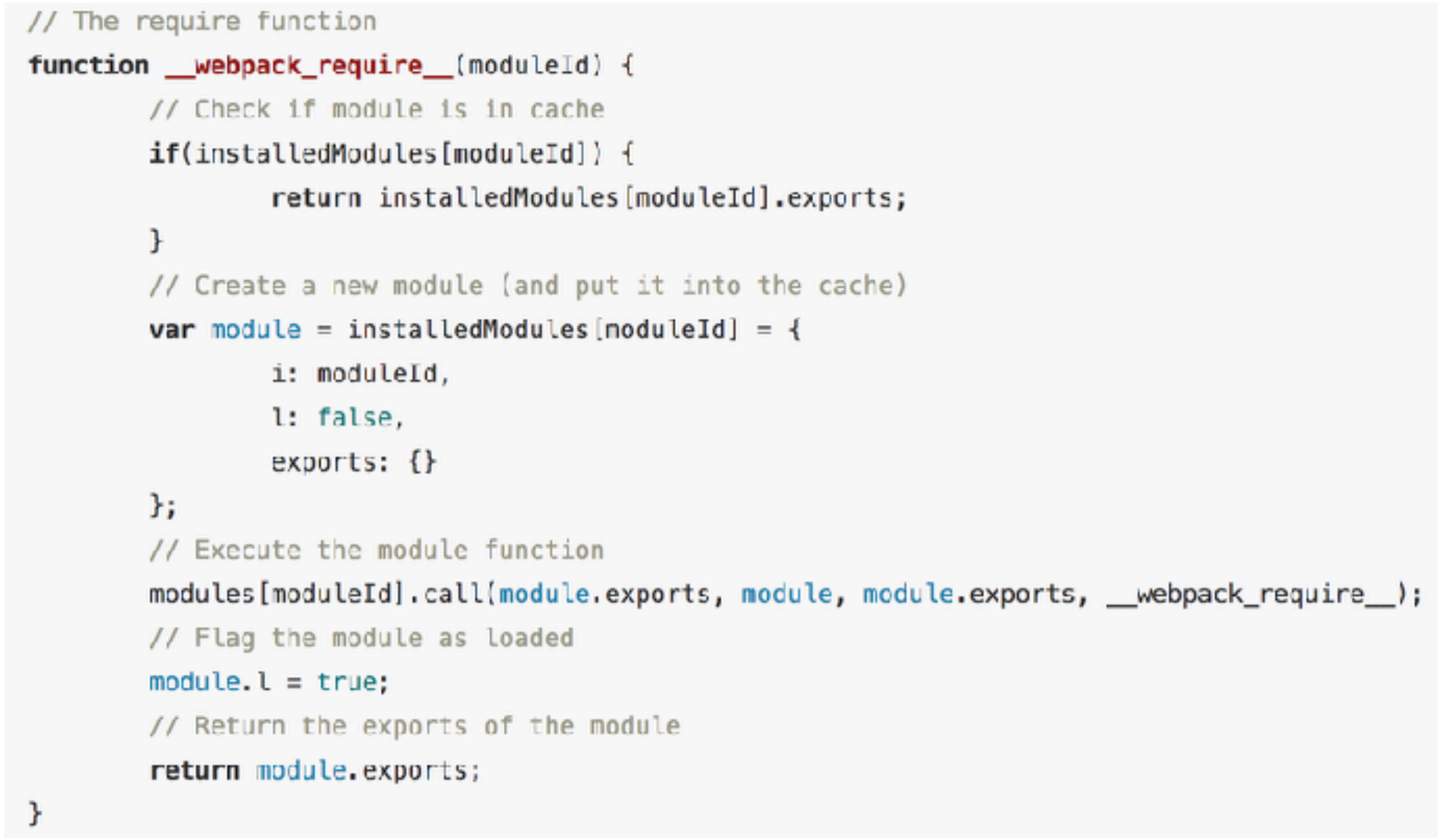


```
/* 1 */
/***/ (function(module, exports) {

const str = 'test1 is loaded';

module.exports = str;

/***/ }),
```

Common.js

执行moreModules数组中对应的元素的函数，就能够变相的将这个module想要export的内容挂载到输入到函数的参数对象 -> module和export上

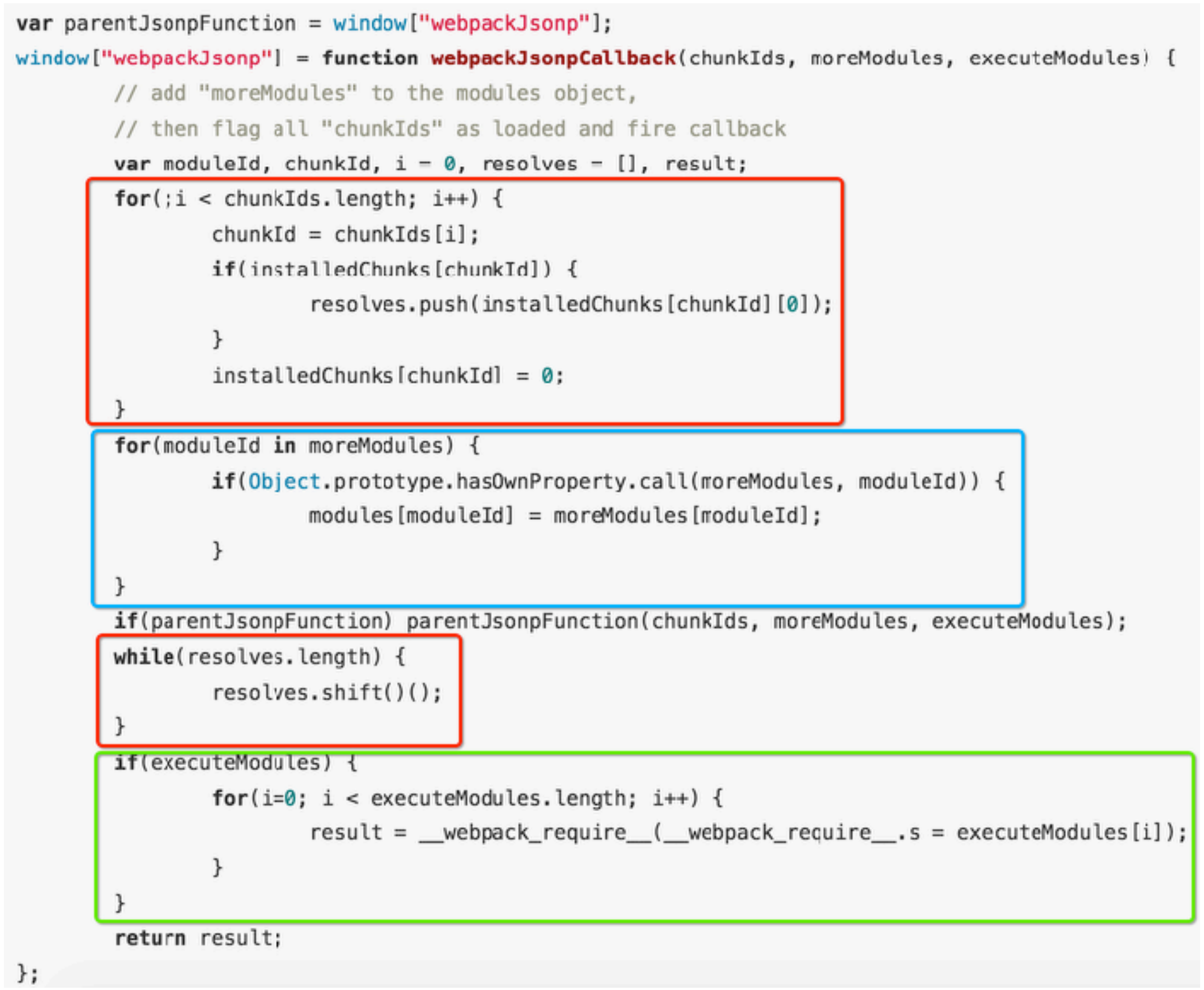
__webpack_require__



```
// The require function
function __webpack_require__(moduleId) {
        // Check if module is in cache
        if(installedModules[moduleId]) {
                return installedModules[moduleId].exports;
        }
        // Create a new module (and put it into the cache)
        var module = installedModules[moduleId] = {
                i: moduleId,
                l: false,
                exports: {}
        };
        // Execute the module function
        modules[moduleId].call(module.exports, module, module.exports, __webpack_require__);
        // Flag the module as loaded
        module.l = true;
        // Return the exports of the module
        return module.exports;
}
```

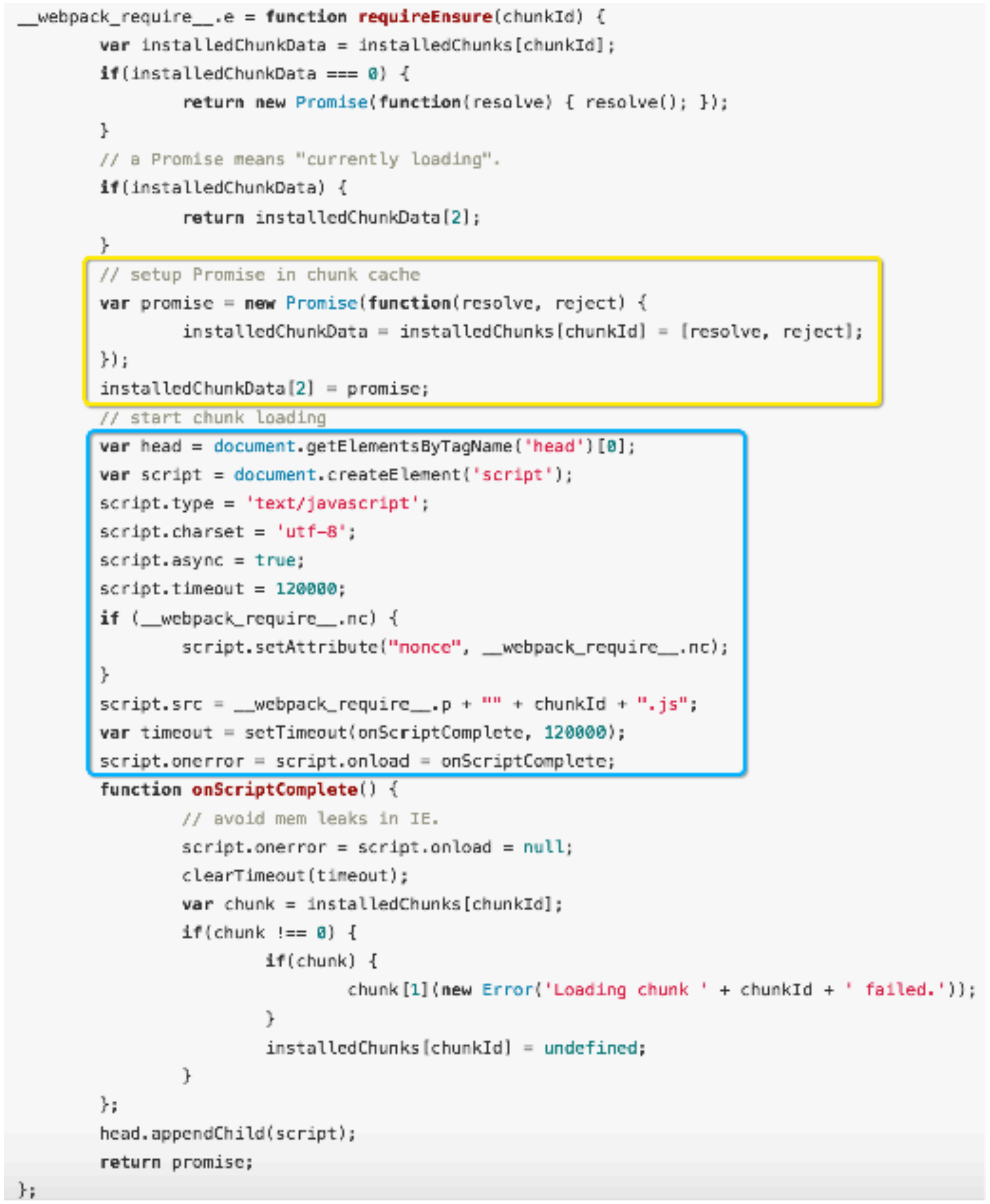
每个module只会在最开始依赖到的时候加载一次，之后会从installedModules直接获取不在加载。如果module依赖的module继续依赖其他module的话，上述的过程会递归的执行下去，但是加载过的依赖只会加载一次。

webpackJsonp



```
var parentJsonpFunction = window["webpackJsonp"];
window["webpackJsonp"] = function webpackJsonpCallback(chunkIds, moreModules, executeModules) {
        // add "moreModules" to the modules object,
        // then flag all "chunkIds" as loaded and fire callback
        var moduleId, chunkId, i = 0, resolves = [], result;
        for(;i < chunkIds.length; i++) {
                chunkId = chunkIds[i];
                if(installedChunks[chunkId]) {
                        resolves.push(installedChunks[chunkId][0]);
                }
                installedChunks[chunkId] = 0;
        }
        for(moduleId in moreModules) {
                if(Object.prototype.hasOwnProperty.call(moreModules, moduleId)) {
                        modules[moduleId] = moreModules[moduleId];
                }
        }
        if(parentJsonpFunction) parentJsonpFunction(chunkIds, moreModules, executeModules);
        while(resolves.length) {
                resolves.shift()();
        }
        if(executeModules) {
                for(i=0; i < executeModules.length; i++) {
                        result = __webpack_require__(__webpack_require__.s = executeModules[i]);
                }
        }
        return result;
};
```

__webpack_require__.e



```
__webpack_require__.e = function requireEnsure(chunkId) {
        var installedChunkData = installedChunks[chunkId];
        if(installedChunkData === 0) {
                return new Promise(function(resolve) { resolve(); });
        }
        // a Promise means "currently loading".
        if(installedChunkData) {
                return installedChunkData[2];
        }
        // setup Promise in chunk cache
        var promise = new Promise(function(resolve, reject) {
                installedChunkData = installedChunks[chunkId] = [resolve, reject];
        });
        installedChunkData[2] = promise;
        // start chunk loading
        var head = document.getElementsByTagName('head')[0];
        var script = document.createElement('script');
        script.type = 'text/javascript';
        script.charset = 'utf-8';
        script.async = true;
        script.timeout = 120000;
        if (__webpack_require__.nc) {
                script.setAttribute("nonce", __webpack_require__.nc);
        }
        script.src = __webpack_require__.p + "" + chunkId + ".js";
        var timeout = setTimeout(onScriptComplete, 120000);
        script.onerror = script.onload = onScriptComplete;
        function onScriptComplete() {
                // avoid mem leaks in IE.
                script.onerror = script.onload = null;
                clearTimeout(timeout);
                var chunk = installedChunks[chunkId];
                if(chunk !== 0) {
                        if(chunk) {
                                chunk[1](new Error('Loading chunk ' + chunkId + ' failed.'));
                        }
                        installedChunks[chunkId] = undefined;
                }
        };
        head.appendChild(script);
        return promise;
};
```

__webpack_require__.e

```
// index2.js
setTimeout(function() {
    require.ensure([], function() {
        const test2 = require('./test2');
        console.log(test2);
    });
}, 5000);

// test2.js

const str = 'test2 is async loaded';
module.exports = str;
```

```
// test2.js
const str = 'test2 is async loaded';

module.exports = str;
```

```
// bundle2.js
webpackJsonp([2],{

/***/ 3:
/***/ (function(module, exports, __webpack_require__) {

setTimeout(function() {
    __webpack_require__.e/* require.ensure */(0).then((function() {
        const test2 = __webpack_require__(4);
        console.log(test2);
    }).bind(null, __webpack_require__)).catch(__webpack_require__.oe);
}, 5000);

/***/ })

},[3]);
```

```
// 0.js
webpackJsonp([0],{

/***/ 4:
/***/ (function(module, exports) {

const str = 'test2 is async loaded';

module.exports = str;

/***/ })

});
```

Summary

Webpack打包后的文件，自己实现了一套模块加载的机制，
这样方便实现比如代码分割等功能和优化。

END