链滴

Webpack JS 模块化原理

作者：zhangzhengyi12

原文链接：https://ld246.com/article/1567739921797

来源网站：链滴

最近一直在看一些和JS模块化发展历程的东西，正好呢，也想了解一下Webpack在我们背后帮我们了那些事情，所以就有了今天的这篇文章。

首先我们自然需要搭建一个最简单的Webpack环境，我这里也上传到了github

我们这里只建立一个最简单的模块依赖，只有两个文件

```
// index.js
import sayHello from './mod'

sayHello()

// mod.js
function sayHello() {
  console.log('Hello')
}

export default sayHello
```

接下来我们需要用dev模式来进行打包

```
webpack --mode development --config webpack.config.js
```

输出

接下来我们直接观察最终的输出来查看最终结果

由于代码量不小，我就讲一些解释写在注释之中。

```
// 这个一个非常巨大的IIFE 参数在这个函数的后面，由Webpack直接导出生成
;(function(modules) {
  var installedModules = {} // 该对象为模块缓存
  /**
   * @param {*} moduleId
   * 模块的key实际上在开发环境和生产环境是不同的
   * 在生成环境中是一个数字id 而开发环境是一个字符串的key
   */
  function __webpack_require__(moduleId) {
    // 如果该模块已经导出过了 直接返回缓存
    if (installedModules[moduleId]) {
      return installedModules[moduleId].exports
    }
    // 建立一个基本模块 准备进行安装
    var module = (installedModules[moduleId] = {
      i: moduleId,
      l: false,
      exports: {}
    })

    // 每个模块都被封装成了一个函数
    // 这里就执行该函数 获取导出值
    // 请注意参数顺序
    // 这里可以思考一下 为什么当我们使用CommonJS模块导出的时候
    // module.exports = xx 有效 但直接 exports = xx无效
```

```
  // 原因其实就是因为 exports = xx只是给函数的实参重新赋值而已
  // 但module.exports是一个真正的外部应用 改变是有效的
  modules[moduleId].call(
    module.exports,
    module,
    module.exports,
    __webpack_require__
  )

  module.l = true
  // 一旦完成调用 该模块的导出值就会注射到module.exports上
  // 无论你在默认的exports上增加属性 还是直接改变module.exports的指向
  return module.exports
}

// 这里面都是一些工具函数 应该是提供给每个模块内部去使用

__webpack_require__.m = modules

__webpack_require__.c = installedModules

// 这个工具函数用于ES Module导出多个模块的情况 实质上就是挂载为exports对象的属性
__webpack_require__.d = function(exports, name, getter) {
  if (!__webpack_require__.o(exports, name)) {
    Object.defineProperty(exports, name, {
      enumerable: true,
      get: getter
    })
  }
}
__webpack_require__.r = function(exports) {
  if (typeof Symbol !== 'undefined' && Symbol.toStringTag) {
    Object.defineProperty(exports, Symbol.toStringTag, {
      value: 'Module'
    })
  }
  Object.defineProperty(exports, '__esModule', { value: true })
}
__webpack_require__.t = function(value, mode) {
  if (mode & 1) value = __webpack_require__(value)
  if (mode & 8) return value
  if (mode & 4 && typeof value === 'object' && value && value.__esModule)
    return value
  var ns = Object.create(null)
  __webpack_require__.r(ns)
  Object.defineProperty(ns, 'default', {
    enumerable: true,
    value: value
  })
  if (mode & 2 && typeof value != 'string')
    for (var key in value)
      __webpack_require__.d(
        ns,
        key,
```

```
        function(key) {
          return value[key]
        }.bind(null, key)
      )
    return ns
  }

  __webpack_require__.n = function(module) {
    var getter =
      module && module.__esModule
        ? function getDefault() {
            return module['default']
          }
        : function getModuleExports() {
            return module
          }
    __webpack_require__.d(getter, 'a', getter)
    return getter
  }

  __webpack_require__.o = function(object, property) {
    return Object.prototype.hasOwnProperty.call(object, property)
  }

  __webpack_require__.p = '' // Load entry module and return exports

  return __webpack_require__((__webpack_require__.s = './index.js'))
})({
  './index.js': function(module, __webpack_exports__, __webpack_require__) {
    'use strict'
    // 同样的 在开发模式和生产模式下 IIFE传入到参数也不同 开发模式为一个对象 而生成模式是一个
组
    // 可以看到 第一步其实就是使用工具函数，设定该模块是一个ESModule
    // 如果是CommonJS模块的话，其实就不会帮你去改变参数的名字了,第二个参数就会是exports
    // ESModule 改参数名应该是为了避免冲突
    eval(
      '__webpack_require__.r(__webpack_exports__);\n/* harmony import */ var _mod_js__WEBP
CK_IMPORTED_MODULE_0__ = __webpack_require__(/*! ./mod.js */ "./mod.js");\n\n\nObject(_
od_js__WEBPACK_IMPORTED_MODULE_0__["default"])()\n\n\n//# sourceURL=webpack:///./in
ex.js?'
    )
  },

  './mod.js': function(module, __webpack_exports__, __webpack_require__) {
    'use strict'
    eval(
      '__webpack_require__.r(__webpack_exports__);\nfunction sayHello() {\n  console.log(\'Hello
')\n}\n\n/* harmony default export */ __webpack_exports__["default"] = (sayHello);\n\n\n//# s
urceURL=webpack:///./mod.js?'
    )
  }
})
```

原文链接：Webpack JS 模块化原理

总结

1. 整个JS的运行过程是一个大型的IIFE,并在这个函数的尾部运行根JS模块（入口JS文件）
2. 模块将会被封装进一个函数，并被传入该模块的上下文
3. 一但需求某个模块 将会优先检查模块缓存，如果未装载，则触发模块函数，获取最终导出
4. 对于ES Module,他会转换成类似CommonJS的模块化规范
5. webpack的打包在不同环境下有很大的差异，如在开发环境下会利用eval来进行sourceMap的映（当然这只是一种sourceMap的映射方式）

原文链接：Webpack JS 模块化原理