# 深入学习webpack

## 0. 个人介绍

大圣老师， 8年前端开发经验，三年授课经验，前百度资深前端

擅长Vue/React、源码架构、小程序、移动端和Nodejs整个前端技术栈，对前端萌新如何快速进阶有丰富的经验

今天webpack和工程化是主题，大家对别的话题感兴趣，也可以提出来，一起探讨，比如前端成长，面试，框架学习等等

## 1. 课前准备

1. webpack功能
2. webpack源码 https://github.com/webpack/webpack
3. loader https://github.com/webpack-contrib/style-loader
4. plugin

## 2. 主题



## 3. 课堂目标

1. 手写webpack原理
2. 定制自己的lodaer和plugin

# 4. 知识点

## webpack

1. `entry:` 定义整个编译过程的起点
2. `output:` 定义整个编译过程的终点
3. `module:` 定义模块 `module` 的处理方式
4. `plugin` 对编译完成后的内容进行二度加工
5. `resolve.alias` 定义模块的别名

```
module.exports =  {
        entry: './index.js',
        output: {
            path: path.resolve(process.cwd(),'dist/'),
            filename: '[name].js'
        },
        resolve: {
            alias:{ jquery: 'src/lib/jquery.js', }
        },
        plugins: [
            new WebpackNotifierPlugin()
        ],
        module: {
            loaders: [{
                test: /\.js[x]?$/,
                exclude: /node_modules/,
                loader: 'babel-loader'
            },  {
                test: /\.less$/,
                loaders:['style-loader', 'css-loader','less-loader']
            },{
                test: /\.html/,
                loader: "html-loader?" + JSON.stringify({minimize: false })
            } ]
        }
    };
```

## kkbpack

新建目录kkbpack 和02

02中 `npm install webpack webpack-cli -D` 新建src/index.js 和src/a.js

```
const sayHi = require('./a.js')

sayHi('开课吧')

// a.js

module.exports = (name)=>{
    console.log('hello '+name)
}
```

为了方便阅读代码，打开调试模式，新建webpack.config.js

```
module.exports = {
    mode:'development',
    entry:"./src/index.js",
    output:{
        filename:'pack.js'
    }
}
```

## webpack编译后代码

执行Npx webpack 查看打包后的代码，删除一些无用的代码后，大概是这个样子q

```
(function(modules) { // webpackBootstrap
    // The module cache
    var installedModules = {};
    // 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;q
        // Return the exports of the module
        return module.exports;
```

```
    }

    // Load entry module and return exports
    return __webpack_require__(__webpack_require__.s = "./src/index.js");
})({
 "./src/a.js":(function(module, exports) {

eval("module.exports = (name)=>{\n    console.log('hello '+name)\n}\n\n//#
sourceURL=webpack:///./src/a.js?");

}),

"./src/index.js":(function(module, exports, __webpack_require__) {

eval("\nconst sayHi = __webpack_require__(/*! ./a.js */ \"./src/a.js\")\n\n\nsayHi('开课
吧')\n\n//# sourceURL=webpack:///./src/index.js?");

/***/ })

});
```

大概的意思就是，我们实现了一个**webpack_require** 来实现自己的模块化，把代码都缓存在installedModules里，代码文件以对象传递进来，key是路径，value是包裹的代码字符串，并且代码内部的require，都被替换成了**webpack_require**

## npm link

环境oK，开始搞起

## 读取配置文件

咱们的配置文件，就叫kkbpack.config.js把

```
module.exports = {
    output:{
        filename:'kkb.js'
    }
}
```

```
#! /usr/bin/env node
const path = require('path')

const defaultConfig = {
    entry:"./src/index.js",
    output:{
        filename:'bundle.js'
    }
}
const config =  {...defaultConfig, ...require(path.resolve('./kkbpack.config.js'))}
console.log(config)
```

```
class KkbPack {
    constructor(config){
        this.config = config
    }
    start(){
        console.log('开始啦')
    }
}

const kkb = new KkbPack(config)
kkb.start()
```

## 读取入口文件

```
class KkbPack {
    constructor(config){
        this.config = config
        // 入口
        this.entry =config.entry
        // 工作路径
        this.root = process.cwd()
        // 依赖关系
        this.modules = {}
    }
    // 创建模块
    createModule(modulePath, name){
        // modulePath是绝对路径，获取文件
        // name是相对路径，作为key
        let code = fs.readFileSync(modulePath, 'utf-8')
        console.log(name, code)
    }
    start(){
        // 创建模块依赖关系
        console.log('开始啦1')
        const entryPath = path.resolve(this.root,  this.entry)
        this.createModule(entryPath, this.entry)
    }

}

const kkb = new KkbPack(config)
kkb.start()
```

## 解析源码

```
"./src/index.js":(function(module, exports, __webpack_require__) {

eval("\nconst sayHi = __webpack_require__(/*! ./a.js */ \"./src/a.js\")\n\n\nsayHi('开课吧')\n\n//# sourceURL=webpack:///./src/index.js?");

})
```

1. 替换require为webpack_require,
2. ./a.js 替换为 ./src/a.js

```
parse(code, parent){
    console.log(code,parent)
    // 识别 require('xx')
    var r = /require\((.*)\)/g;
    let match
    code = code.replace(r, function(match, arg){
        // console.log(1,match, arg.replace(/'|"/g,''))
        const retPath = path.join(parent,arg.replace(/'|"/g,''))
        return `__kkbpack_require_("./${retPath}")`
    })
    console.log(code,parent)
}
```

```
const sayHi = __kkbpack_require_("./src/a.js")

sayHi('开课吧') ./src
```

## 依赖列表

依赖的文件需要继续解析，而且文件可能还依赖于其他文件

比如新建B.js

```
module.exports = function sayBye(name){
    console.log('byebye '+ name)
}
```

所以我们parse的时候 要记录下所有的依赖，进行继续递归解析

```
deps.push(retPath)

deps.forEach(dep=>{
    // console.log('xx',dep)
    this.createModule(path.join(this.root,dep), dep)
})
```

```
class KkbPack {
    constructor(config){
        this.config = config
        // 入口
        this.entry =config.entry
        // 工作路径
        this.root = process.cwd()
        // 依赖关系
        this.modules = {}
    }
    // 创建模块
    createModule(modulePath, name){
        // modulePath是绝对路径，获取文件
        // name是相对路径，作为key
        let fileContent = fs.readFileSync(modulePath, 'utf-8')
        // ./src/index.js 文件的父目录，其实就是src
        // 解析soruce源码
        const {code, deps} = this.parse(fileContent, path.dirname(name))
        console.log(code, deps)

        this.modules[name] = code

        // 递归获取依赖
        deps.forEach(dep=>{
            // console.log('xx',dep)
            this.createModule(path.join(this.root,dep), dep)
        })
    }
    parse(code, parent){
        let deps = []
        // console.log(code,parent)
        // 识别 require('xx')
        var r = /require\((.*)\)/g;
        let match
        code = code.replace(r, function(match, arg){
            // console.log(1,match, arg.replace(/'|"/g,''))
            const retPath = path.join(parent,arg.replace(/'|"/g,''))
            deps.push(retPath)
            return `__kkbpack_require_("./${retPath}")`
        })
        return {code, deps}
    }
    start(){
        // 创建模块依赖关系
        console.log('开始啦1')
        const entryPath = path.resolve(this.root,  this.entry)
        this.createModule(entryPath, this.entry)
        console.log(this.modules)
    }
```

打印结果

```
{ './src/index.js':
   '\nconst sayHi = __kkbpack_require_("./src/a.js")\n\n\nsayHi(\'开课吧\')',
  'src/a.js':
   '\nconst sayBye = __kkbpack_require_("./src/test/b.js")\nmodule.exports = (name)=>
{\n    console.log(\'hello \'+name)\n    sayBye(name)\n}',
  'src/test/b.js':
   '\n\nmodule.exports = function sayBye(name){\n    console.log(\'byebye \'+ name)\n}'
}
```

有点webpack的意思了

## 生成文件

webpack编译后的文件

```
({
 "./src/a.js":(function(module, exports) {

eval("module.exports = (name)=>{\n    console.log('hello '+name)\n}\n\n//#
sourceURL=webpack:///./src/a.js?");

}),

"./src/index.js":(function(module, exports, __webpack_require__) {

eval("\nconst sayHi = __webpack_require__(/*! ./a.js */ \"./src/a.js\")\n\n\nsayHi('开课
吧')\n\n//# sourceURL=webpack:///./src/index.js?");

/***/ })

})
```

存储的并不只是文件的内容，而是用模块化包装起来的函数，传递了Module,exports，**webpack_require** 内部使用eval把字符串转化为函数，内部的Module,exports，__webpack_require就可以运行起来

Template.js

```
// 模板

!function start(modules) {
    // 缓存
    var installedModules = {};
    // The require function
    function __kkbpack_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] = {
            exports: {}
        };
        // Execute the module function
        modules[moduleId].call(module.exports, module, module.exports,
__kkbpack_require__);
        // Flag the module as loaded
        module.l = true;
        // Return the exports of the module
        return module.exports;
    }

    // Load entry module and return exports
    return __kkbpack_require__(__kkbpack_require__.s = "__entry__");
}({__content__})
```

index.js

```
    generateModuleStr(){
        // 生成model字符串
        let fnTemp = ""
        Object.keys(this.modules).forEach(name=>{
            fnTemp += `"${name}":${this.modules[name]},`
        })
        return fnTemp
    }
    generateFile(){
        // console.log(this.modules, this.entry)
        // console.log(123,this.root)
        let  template = fs.readFileSync(path.resolve(__dirname,'./template.js'), 'utf-
8')
        template = template.replace('__entry__',this.entry)
                           .replace('__content__', this.generateModuleStr())
        fs.writeFileSync('./dist/'+this.config.output.filename, template)
    }
    start(){
        // 创建模块依赖关系
        console.log('开始啦1')
        const entryPath = path.resolve(this.root,  this.entry)
        this.createModule(entryPath, this.entry)
        this.generateFile()

    }
```

## build

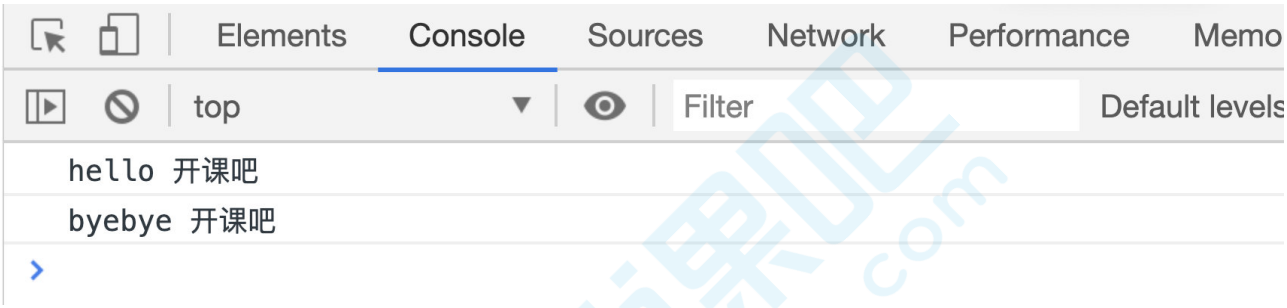
打开dist/kkb.js

```
// 模板

!function start(modules) {
    // 缓存
    var installedModules = {};
    // The require function
    function __kkbpack_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] = {
            exports: {}
        };
        // Execute the module function
        modules[moduleId].call(module.exports, module, module.exports,
__kkbpack_require__);
        // Flag the module as loaded
        module.l = true;
        // Return the exports of the module
        return module.exports;
    }

    // Load entry module and return exports
    return __kkbpack_require__(__kkbpack_require__.s = "./src/index.js");
}({"./src/index.js":function(module,exports, __kkbpack_require__){
            eval(`
const sayHi = __kkbpack_require__("./src/a.js")


sayHi('开课吧')`)
        },"./src/a.js":function(module,exports, __kkbpack_require__){
            eval(`
const sayBye = __kkbpack_require__("./src/test/b.js")
module.exports = (name)=>{
    console.log('hello '+name)
    sayBye(name)
}`)
        },"./src/test/b.js":function(module,exports, __kkbpack_require__){
            eval(`

module.exports = function sayBye(name){
    console.log('byebye '+ name)
}`)
        },})
```

新建index.html

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
</head>
<body>


    <script src="dist/kkb.js"></script>
</body>
</html>
```




## 5. 扩展

1. AST
2. 模板语言 比如ejs，jade
3. tapable
4. loader
5. plugin

## 6. 总结